THE DESIGN DOCUMENTS

1. ACP OVERVIEW by Dipu Bose.
2. ACP DETAILED DESIGN by Asim Mehta.
3. UNIBUS DESIGN by Asim Mehta.
4. RSX-11M to RSX-11M-PLUS DESIGN NOTES by Asim Mehta.
5. TELNET SERVER DESIGN by Asim Mehta.

DESIGN OVERVIEW OF THE EXOS DRIVER/ACP FOR RSX-11M SYSTEMS

by

Dipu Bose

1. I/O PHILOSOPHY OF RSX-11M

Memory constraints and compatability between different versions of RSX-11M dominated the design philosophy and the strategy used in creating the RSX-11M Operating System. To meet its performance and space goals, the RSX-11M I/O system attempts to centralize the common functions,thus eliminating the inclusion of repeatative code in each and every driver in the system. To achieve this, tabular data structures are designed in the system, which are used to drive the centralized routines. The effect is to reduce substantially the size of individual I/O drivers.

2. THE STRUCTURE OF THE I/O DRIVER

The next few sections describe the I/O driver structure of RSX-11M. Refer to the "Guide to writing an I/O driver" Manual for more detailed and thorough treatment.

The Executive processes I/O requests using the following :

- Ancillary Control Processor (ACP)
- A collection of Executive components consisting of :

  a. QIO directive processing
  b. I/O related subroutines
  c. The I/O drivers

2.1 An ACP is responsible for maintaining the structures and integrity of the device (or a collection of devices) related data structures. It is an asynchronous privileged task which implements a protocol (or set of services) for a class of a device. It functions as an extension of the Executive and frequently operates with Executive privilege. Since an ACP is a task, it has all the attributes of a task together with the ability to receive I/O packets from other tasks, as a process. The latter attribute permits it to act as an I/O handler, which can compete with user tasks for system resources more equitably than an I/O driver could. This is because an I/O driver has no task identity of its own. Also unlike the I/O driver, an ACP can perform I/O to other devices during the processing of an I/O request.

2.2 The QIO directive is the lowest level of task I/O. Any task

can issue a QIO directive which allows direct control over devices that are connected to a system and have an I/O driver. The QIO directive forces all I/O requests from user tasks to go through the Executive. The Executive works to prevent tasks from destructively interfering with each other and with the Executive itself.

2.3 An I/O driver is an asynchronous process (not a task) that calls and is called by the Executive to serivce an external I/O device or devices. The role of an I/O driver in the RSX-11M I/O structure is specific and limited. A driver performs the following functions :

2.3.1 Receives and serivces interrupts from its I/O devices.
2.3.2 Initiates I/O operations when requested to do so by the Executive.
2.3.3 Cancels in-progress I/O operations.
2.3.4 Performs other (device-specific) functions upon power failures and device time-out.

As an integral part of the Executive, a driver possesses its own context, allows or disallows interrupts, and synchronizes its access to shared data bases with that of other Executive processes. A driver can handle several device controllers, all operating in parallel.

Every I/O driver in the RSX-11M system has the following entry points:

a. Device interrupts
b. I/O initiator
c. Device time-out
d. Cancel I/O
e. Power failure

Apart from the first entry point, which is entered by hardware device interrupts, all are entered by calls from the Executive.

2.4 I/O Related Subroutines

RSX-11M provides a set of centralized subroutines which operate on the centralized data bases and give the user a significant amount of flexibility while maintaining the integrity and uniformity in coding. Also a significant amount of code repeatation can be avoided with their proficient use.

3. I/O RELATED DATA STRUCTURE

An I/O driver interacts with the following data structures :

a. Device Control Block (DCB)

b. Unit Control Block (UCB)
c. Status Control Block (SCB)
d. The I/O Packet
e. The I/O Queue
f. The Fork List
e. Device Interrupt Vectors

The first four of these data structures are especially important to the driver because it is by means of these data structures that all I/O operations are effected. They also serve as communication and co-ordination vehicles between the Executive and individual drivers. Entry to a driver following a device interrupt is accomplished through the appropriate hardware device interrupt vector/s.

a. The Device Control Block (DCB)

At least one DCB exists for each type of device appearing in a system. The function of the DCB is to describe the static characteristics of both the device controller and the units attached to the controller. All the DCB's in a system forms a forward link list, with the last DCB having a link to zero.

b. The Unit Control Block (UCB)

One UCB exists for each device unit attached to a system. Much of the information in the UCB is static, though a few dynamic parameters exist. From the UCB, however, it is possible to access most of the other structures in the I/O data base. Few of its contents are used and modified by both Executive and the driver.

c. The Status Control Block (SCB)

One SCB exists for each device controller in the system. This is true even if the controller handles more than one device unit. Most of the information in the SCB is dynamic. Both the Executive and the driver use the SCB.

d. The I/O Packet

The I/O Packet is built dynamically during the QIO directive processing and is subsequently delivered to the driver by a call to the system Executive. No static fields exist with respect to a driver and is generated mostly from the information passed in the directive parameter block.

e. I/O Queue

The QIO directive after successfully generating an I/O packet inserts it into a device-specefic, priority oriented ordered list of packets called I/O Queue. Each I/O queue listhead is located in the SCB to which the I/O request apply. When a device needs work, it requests the Executive to dequeue the next I/O packet and delivers it to the requesting driver.

Normally the driver does not directly manipulate the I/O queue.

f. Fork List

Fork List is a mechanism by which RSX-11M splits off a process that requires access to shared data bases, or that require more CPU time to process an interrupt. A process that calls $FORK(an executive routine), requests the Executive to transform it into a 'fork process' and place it in a fork list. A call to $FORK saves a "snapshot" of the process (R4,R5 and PC) in a fork block. This fork block is queued on the fork list in first-in-first-out order.

g. The Device Interrupt Vector

The device interrupt vector consists of two consecutive words giving the address of the interrupt service routine and the priority at which it is to run. The low four bits of the second word of the interrupt vector must contain the number of the controller that interrupts through this vector. This requirement enables a driver to service several controllers with a few code changes.

4. EXECUTIVE SERVICES :

The Executive provides services related to I/O drivers that can be categorized as pre- and post- driver initiation. The pre initiation services are those performed by the Executive during its processing of a QIO directive. Its goal is to extract from the QIO directive all I/O support functions not directly related to the actual issuance of a function request to a device.

The post initiation services are made available to the driver after it has been given control, either by the Executive or as the result of an interrupt. They are available as needed by means of Executive calls.

5. ASYNCHRONOUS SYSTEM TRAPS (AST)

The primary purpose of an AST is to inform the task that a certain event has occurred. For example, the completion of an I/O operation. As soon as the task has serviced the event, it can return to the interrupted code. When an AST occurs, the Executive pushes the task's Wait For Mask Word, the DSW, the PSW and the PC into the task's stack. This information saves the state of the task so that the AST service routine has access to all the available Executive services. Most of the Executive directive calls has an optional AST entry point, such that AST occurs upon a certain condition, e.g. an I/O completion, so that some user specified operation could now be done at that entry point.

6. FLOW OF AN I/O REQUEST

The flow of an I/O request, issued by the user by issuing a QIO directive, is as follows :

6.1 Task issues QIO directive.

6.2 QIO processing.

6.2.1 First level validity checks.

The QIO directive processor validates the Logical Unit Number (LUN) and UCB pointer.

6.2.2 Redirect Algorithm .

Because the Unit Control Block (UCB) may have been dynamically redirected by an MCR redirect command, the QIO directive traces the redirect linkage until the target UCB is found.

6.2.3 Additional Validity Checks.

The Event Flag Number (EFN) and the address of the I/O status block (IOSB) are validated. The event flag is reset and the I/O status block is cleared.

6.3 Executive obtains storage for and creates an I/O packet.

The QIO directive processor now requires an 18-word block of dynamic storage for use as an I/O packet. It inserts into the packet, data items that are used subsequently by both the Executive and the driver in fulfilling the I/O request. Most items originate in the requesting task's directive parameter block (DPB).

6.4 Executive validates the function requested.

The function is one of the four possible types :

- Control
- No-op
- ACP
- Transfer

Control functions are queued to the driver. If the function is IO.KIL, the driver is called at its cancel I/O entry point. The IO.KIL request is then completed sucessfully.

No-op functions do not result in data transfers. The Executive

"performs" them without calling the driver. No-ops return a status of IS.SUC in the I/O status block.

ACP functions are those functions which are to be processed by the ACP. The Executive queues the I/O packet to the ACP and issues a run request of the ACP, if it is stopped.

Transfer functions are address checked and queued to the proper driver. Then the driver is called at its initiator entry point.

6.5 Driver Processing

6.5.1 Request work

To obtain work, the driver calls the $GTPKT routine. $GTPKT either provides work, if it exists, or informs the driver that no work is available, or that the SCB is busy. If no work exists, the driver returns to its caller. If work is available, $GTPKT sets the device controller and unit to "busy", dequeues an I/O request packet and returns to the driver.

If UC.QUE is set, the packet is passed to the driver at its initiator entry point. The driver is entered at its entry point with some registers set to specific values like address of I/O packet, address of the UCB and etc. If the request is to be processed by an ACP, the packet is queued to the ACP.

6.5.2 Issue I/O

From the available data structures, the driver initiates the required I/O operation and returns to its caller. A subsequent interrupt may inform the driver that the initiated function is complete, assuming the device is interrupt driven.

6.6 Interrupt Processing.

When a previously issued I/O operation interrupts the driver, the interrupt causes a direct entry into the driver, which processes the interrupt according to the programming protocol. According to the protocol, the driver may process the interrupt at priority 7, at the priority of the interrupting device, or at fork level. If the processing of the I/O request associated with the interrupt is still incomplete, the driver initiates further I/O to the device. When the processing of an I/O request is complete, the driver calls $IODON.

6.7 I/O Done Processing

$IODON removes the "busy" status from the device unit and controller, queues an AST, if required, and determines if a checkpoint request pending for the issuing task can now be effected. The IOSB and event flag, if specified, are updated and $IODON returns to the driver. The driver branches to its initiator entry point and looks for more work. This procedure is followed until the driver finds the queue empty, whereupon the driver returns to its caller.

Eventually, the processor is granted to another ready-to-run task that issues a QIO directive, starting the I/O flow anew.

8. DESIGN PHILOSOPHY FOR THE EXOS DRIVER

The EXOS front-end Ethernet Controller board is modelled as a controller of a single devive-unit which supports multiple paths of communication with the network and the board itself. These paths are called channels and are designated by a channel descriptor number, called the channel number. The channels grossly correspond to a socket (an end-point in the network communication) or a path for obtaining services from the front-end (e.g. initializing and configuring the board , downloading protocol software to the board's memory, etc).

The user program should create a channel either for administrative operations or to obtain services from the network. In either case the user should use the channel number, which the driver software returns to him in response to an open channel call, for subsequent operations. The channel provides the user task a protection mechanism from destructively interfering with each others path of communication. For example, a socket created by one task cannot be accessed by another task.

One of the major decisions in the design of the EXOS driver was to attach an Auxilliary Control Processor with the driver. A substantial amount of drivers work is done by the ACP. In fact, ACP is the central routine which does all the work and the driver just acts as an traffic controller, routing all the requests from the users to the ACP. The reason behind taking this decision are:

- to overcome the 16 KB of driver space restriction: As the driver accesses the 8 KB of the I/O page and 20 KB of the system executive space (executive routines and data), it has only 16 KB left to itself.

- to minimise processing time at the interrupt level at the drivers interrupt entry point by waking up the ACP from this point and letting it do the work at task level.

- to exploit the task feature of the ACP, which makes it overlayable and also let it compete for other system resources equitaibly with other tasks in the system. It

allows ACP to get services fron other devices as well (via QIO'S).

- to have overall design simplicity for easy maintainance and also portability to other variations of RSX-11M operating systems like RSX-11M-PLUS and Micro RSX.

9. IMPLEMENTATION DETAILS

9.1 General Information

The driver's role in the EXOS I/O handler package is very small and limited only to that of an I/O request traffic controller. ACP is the major module in this package which does most of the work. The management and processing of the EXOS-HOST Message queue (refer to chapter 4 of EXOS 203 Manual) is done by the ACP. This message queue forms a part of the ACP's local data area which is physically shared (better say accessed) by the EXOS front-end. All transactions with the EXOS is done via the ACP. Interrupts from the board are received by the driver at its interrupt entry point. The driver passess on this information to the ACP by just waking it up.

I/O requests received by the driver are queued to the ACP by the driver after a minimal processing. The driver address checks the user buffers (if specified) and relocates their virtual addresses in terms of kernel APR 6. It also rearranges the function dependent parameters in the I/O packet and then queues the packet to the ACP requesting for work.

9.2 Driver Data Structures:

A Device Control block (DCB), an Unit Control Block (UCB) and a Status Control Block were defined for the EXOS device driver. The logical name for the EXOS device was given 'ZE' and is defined in the DCB. Most of the functions are defined as Control functions so that the driver receives the request first and then queues the same request to the ACP after some processing. The IO.ATT & IO.DET are made No-op functions.

The UC.QUE bit is set in the U.CTL byte of the UCB. This tells the QIO executive routine to call the Driver at its initiator entry point and pass the I/O packet without queueing it in the driver's I/O queue. Also the UC.KIL bit is set so that the driver is called on a cancel I/O request, even if the unit is not busy.

9.3 ACP Data Structure

There is a special data structure in the ACP, called Channel Descriptor, which keeps all channel related information. The structure of the Channel descriptor is

```
struct channel {            /* channel control block      */
  Uchar  ch_type;           /* type of the Channel        */
  Uchar  ch_flag;           /* protection flags           */
  Ushort ch_tcb;            /* owner task's TCB address   */
  Ushort rundn_cnt;         /* I/O rundown count on this channel */
  union {
    Ushort ch_soid;         /* socket id returned by EXOS */
    struct {                /* EXOS memory pointer        */
      Ushort base;
      Ushort off;
    }
  }
} ch_des[ MAXCHANNEL ];
```

This control block keeps sufficient information for channel managements.

The Message Queue forms a major data structure of the ACP task. The format and fields of the Message Queue are defined in the EXOS 203 Manual.

9.4 QIO Processing

Once the Executive receives a QIO request, it does a first level of validity check as described in section 6.2. It then creates an I/O packet and fills up the appropriate fields from the Directive Parameter Block specified by the user. Since all the I/O functions are control functions and the UC.QUE bit is set, the executive calls the driver at its initiator entry point and passes the address of I/O packet to it. This prevents user context switching so that the driver can execute and relocate the user specified buffers while the user context is intact.

9.5 Driver Processing

The Driver, upon receiving an I/O packet, does some processing. It first address checks the user buffer and then relocates the buffer in terms of kernel APR 6. It places the relocated address in the I/O packet itself by slightly rearranging the parameters. After this it simply queues the packet to the ACP. Queueing of I/O packet is not priority oriented but in first-in-first-out order. So the ACP receives the request in the same order as they have been issued by the user.

9.6 Interrupt Processing

The processing time at the interrupt level is minimised by letting the ACP do the work. Whenever the driver is entered at its interrupt entry point it immediately goes to the fork level and then unstops the ACP and returns. No processing of the EXOS Reply Message Queue is done by the driver. The ACP systematically processes the Reply Message Queue whenever it is unstopped.

9.7 ACP Processing

The ACP iterates an eternal loop. Wnen there is no work pending for the ACP it stops itself and goes to sleep. It is woken up by the driver either from the initiator routine, interrrupt service routine or cancel I/O routine. It first dequeues a packet from its external queue and if it is successful it calls the routines that process the request by filling up the appropriate fields in the appropriate message area and then passing the control of the message queue to the EXOS. The EXOS, to give a reply to a request, willl interrupt the host and the ACP get control as it is unstopped by the driver and calls the routines that process the replies. The actions taken in the request and reply processing are dependent on the function codes in the I/O packet (for requests) and the request codes of the message area (for the replies).

The detailed descriptions of the ACP processing is given in the additional design/maintainance document for the ACP/driver.

DESIGN/MAINTAINANCE DOCUMENTATION FOR THE EXOS DRIVER/ACP FOR THE RSX-11M/RSX-11M-PLUS O.S.

by

Asim K. Mehta

1. INTRODUCTION:

The preliminary design overview of the driver/ACP is given in the DESIGN OVERVIEW OF THE EXOS DRIVEV/ACP document by Dipu Bose. That document describes the basic I/O philosophy of the RSX-11M systems, I/O related data structures, I/O related system protocols to be followed by the device drivers, etc, and the reasons for the important decesions like having the ACP as a separate entity which does all the I/O related operations and that the driver is just a traffic controller for the I/O requests for the EXOS board, etc.

This document will describe the implementation and minor design issues related to the ACP only. A separate document describes the changes made to the RSX-11M driver/ACP to make it work on the RSX-11M-PLUS system. Another document describes the design issues for the driver on the UNIBUS machine.

2. THE MAIN ACP FLOW:

The file acproot.c contains the main ACP routine "_main()". First the local initialization of some data structures is done in the routine "init()". Then the TCB address of the ACP is stored in the ZE UCB in the C - callable macro routine "acpucb()". For the UNIBUS machine this routine also fetches the physical 22-bit address of the start of the local pool into a local data structure. Then unibus initialization is done for the UNIBUS machines. (refer to the unibus doc for more info on this). After these initialization routines, the main loop of the acp code starts. First a packet is dequeued from the acp's external queue and if no packet is available or if no work is pending then the ACP goes off to sleep (all this is done in the C - callable macro routine "dqpkt()"). On waking up the ACP first looks for a packet and if none is available it sees if any work is pending. If so then it returns and does the required processing of the pending work or the processing of the packet, if one was dequeued.

Unless the main do-while loop gets a configuration request for the board and the board gets successfully initialized, the routine "drive()" is not entered. When the board gets ready, this routine is entered and it goes into an eternal loop constantly looking for work. If none is found then it stops (sleeps). On getting work it first checks if the request is for board initialization or not. If so then it serves that request and if not then it puts the I/O packet into an internal queue which is serviced later on in the routine "request()". Then it enters the routine "answer()" where it first checks if any replies have come from the board or not and if so then it serves those replies in the routine "reply()". After the reply processing is over it goes and serves the requests if there are any free slots available in the message queue through which the ACP will communicate with the board. The routine "request()" is called where the I/O packets are served until they get exhausted or they can no-longer be served due to lack of some resources. In this case they are again put in the internal queue so that in the next iteration of the eternal loop it might get a chance to be served if that resource has been freed.

3. THE REQUEST PROCESSING:

The requests are of two major kinds. One is the kind which does not require any participation from the board in honouring the request and the other kind is which requires it. The former are serviced immediately by the main ACP routines - "_main()" and "drive()". The latter kinds of requests are put into an internal queue which is serviced by the routine "request()" whenever a slot in the message area is available for communicating with the board.

Inside the routine "request()", first, the I/O packet is dequeued from the internal queue. Then the control is passed to the appropriate routine according to the function code. The types of requests here are the kinds which just require the board's local statistics and perform operations local to the board and are not involved with the network ("admin()"), the kinds which require the access opearations for the socket ("access()"), the kinds that indulge in data transfer operations to the network ("transfer()"), and the kinds involved in the socket control operations ("excontrol()").

The kinds of requests that are served directly by the main ACP routines are the ones that involve opening/closing of sockets ("opench()" & "closech()"), board setup and initialization procedures ("exsetup()"), the seek operation on the board's memory, retrieval of the configuration message, unselect request ("fin_pen()") and preparing the urgent requests.

The requests that require the participation of the board are first transfered to the board via the message queue and the I/O packet address is put in a pending I/O list which is to be processed by the reply processing. The requests not requiring the participation of the board are finished immediately, after they are serviced, by calling the routine "ackuser()" which calls $IOFIN.

For the requests put in the pending list, the I/O rundown count for that channel is incremented showing it as busy.

4. THE REPLY PROCESSING:

The routine "answer()" is called whenever the acp is woken up. Here the message area (rmsg_area) is scanned to see if any slot has a reply for the host from the board. If it does then it retrieves the I/O packet address from the nm_userid field of the message area and calls the routine "reply()" which does the actual replying for the board to the user, according to the kind of function code, of cource.

The reply routine just fills in the return status nm_reply into the I/O status block and then finishes the I/O by calling the routine "ackuser()". This C - callable macro routine calls $IOFIN fot the purpose. The I/O rundown count for this channel is decremented indicating an I/O was complete. This is done for almost all the kinds of requests (unless otherwise dictated by the request! - as in the case of the reply for the select request in which if the socket is not yet ready the packet is put back into the pending list and it is considered that the I/O has not yet finished since one more reply is expected from the board to indicate that the socket is ready for read/write and it is then that the I/O is considered finished and the I/O rundown count is decremented).

5. DESCRIPTION OF THE DIFFERENT MECHANISMS:

5.1 OPENING/CLOSING OF A CHANNEL:

These operations are essential for the user to request if any kind of communication with the board (involving the network or not) is desired. There

exists an array of 40 channel descriptors which means 40 concurrent channels or paths for communicating with the board can be opened simultaneously. These descriptors are similar to file descriptors and contain information like the type of the channel: {can be administrative or can correspond to a socket for communicating with the network or can be free - not assigned}; they contain flags indicating the status of the channel at run-time: {opened in read/write mode, whether privileged or not, whether marked for close or not}; They contain the owner of this channel: {the TCB address of the issuing task - used as the ownership ID of the user}; the rundown count: {contains the number of concurrent I/O's active on the channel (mechanism described later)}; and it contains the socket ID: {returned by the board if opened for networking operations or it may contain the memory locator of the EXOS memory if the channel is opened for administrative operations}.

These operations are immediate ones. They are serviced immediatly in the main ACP loops and the result is returned to the user.

5.1.1 OPENING A CHANNEL:

The routine "opench()" in file opench.c is called for the purpose of opening a channel. The channel which is marked CH_FREE is searched sequentially and the channel number (ch_no) of the first available free channel is returned to the user. The privilege of the user is checked in the routine "getpriv()" by checking the task and the terminal privilege of the user. If both are privileged then the flag CH_PRIV is set in the ch_flag field of the ch_des[ch_no]. If the channel is requested to be opened in the write mode, the flag CH_WRITE is set.

5.1.2 CLOSING A CHANNEL:

The routine "closech()" in the file opench.c is called for the purpose. First it is checked if the channel number specified is in range (<=40) and if the ID (TCB address) is correct. If so then it checks whether the rundown count is not zero. If it isn't zero that means some I/O is already pending on the channel and hence the channel cannot be really closed. This is so because if, for example, a read is pending and the socket is closed and the user task exits, then if some other task is shceduled to reside in the same memory area as the task which had the read pending, then the DMA from the board may still be on and that may curropt the new task in the memory and cause problems. Hence, the task is blocked until the reply for that read comes. That's the main reason for having this rundown count mechanism. If some I/O is pending then it is marked for close - CH_MCLOSE and the I/O packet for the close request is put in another queue called the mrkcls (marked for close) queue. This means no further requests will be entertained on this channel and as soon as the replies for the pending requests arrive the channel is closed whenever the rundown count becomes zero. While the channel is in the process of being closed, all the fields are reset, the channel is marked CH_FREE. If there are any replies pending on this channel number that are requests to the board for closing the socket, then these are no longer useful as the socket has already been closed and all the I/O is finished on this channel. This packet is dequeued from the mrkcls queue and that I/O is finished by calling "ackuser()" routine. There may be more than one request for closing the channel (in some cases where two SOCLOSEs are issued for the same channel!) and so all are finished.

5.2 I/O RUNDOWN:

The file cancel.c contains the routine "io_rundown()" which finishes all the pending and outstanding I/O's when the board has to be re-initialized.

All the open channels excepting the one opened for re-initialization are closed. The internal queue contains the requests for all the outstanding I/O's, the pending I/O's are in the io_pend queue and the marked for close packets are in the mrkcls queue. These queues are emptied off by finishing all the I/O's in them by calling the routine "ackuser()" for all the I/O's except the IO_KIL and IO_TEL packets which are not the regular I/O packets buts are the ones allocated in the ZE and the ZT drivers respectively for the purpose of IO_KIL and TELNET. These packets are deallocated back to the system pool by calling the C - callable macro routine "dealoc_b()".

NOTE: Now that the pseudo function code TS_HNG has been added for the purpose of hanging up a telnet connection when a bye is given, a hangup packet might be caught up in the internal queue when the request for re-initialization comes. Hence this packet must also be deallocated back into the system pool. (this is not being done now)

Also, if the local pool is allocated by the requests (in the case of UNIBUS machines), then it is deallocated.

5.3 IOKILL MECHANISM:

This is the mechanism to finish all the I/O's of a particular task either when it is aborted or when it itself issues a QIO IO.KIL to finish off all the I/O's before exiting. After this the control comes to the cancel entry point of the ZE driver. Here a dummy IO_KIL packet is allocated from the system pool and sent to the ACP via a $EXRQP.

Here, in the ACP, when the IO_KIL request is received, the control comes to the routine "iokill()". It first checks if any channel is open for that task (done in routine "srchn()"). If so then it issues an SOCLOSE request to the board, increments the rundown count and returns. When the reply for this SOCLOSE arrives, the IO_KIL packet is put into the internal queue so that again this routine is called in the next request cycle and any other open socket for this task is also closed in a simialr way. The rundown count is decremented and the channel is closed (if the rundown count is zero). When control again comes to this routine and if no open channel is found, then all the packets belonging to this task are finished off in the routine "remque()" and the current IO_KIL packet is deallocated.

5.4 SELECT AND UNSELECT PROCESSING:

Select is a mechanism for the user tasks to know whether a socket is ready for read or write so that he can issue a read or a write which would be sure to succeed and take lesser time. The board immediately gives a reply and indicates in the reply field whether the socket is ready or not. If it is not ready then the I/O request packet is put into the pending list and the rundown count is not decremented (described above). If it is ready then the user is informed of this by calling "ackuser()". If the socket is not ready then the board is expected to give a reply some time later indicating a selected socket. This reply is supposed to be unsolicited and the request code is not SOSELECT but SOSELWAKEUP. The user may not want to wait for that long. Or even if he waits the I/O rundown count remains non-zero and the task cannot be aborted. In that case the user can issue a QIO IO_ACS!SA_USL (UNSELECT) request which informs the ACP to finish off the pending select request regardless of the socket being ready or not.

This request (SOSELECT) is unlike other requests in the sense that all the other requests use the nm_userid field for filling the io_pkt address to

recognize the owner of that request but this request uses the nm_proc field of the structure Sock_select so this has to be handled differently by the reply routines. The nm_proc field's MSB has to be a zero for correct operation. This means that the I/O packet address higher than 0x8000 will cause all sorts of problems in the board code. Since the I/O packet address is always on an even boundary, it is shifted one bit to the right and then stored in the nm_proc field. After it is retrieved from the reply message, it is again shifted one bit to the left and then compared to the actual address in the pending list. This causes the MSB to remain reset. The mechanism used for the purpose of fulfilling the protocol of select and unselect is as follows:

When the request for select is made the field i_prm5 of the io_pkt is used as a status word which is initially set to NOREPLY which indicates that no reply has yet come. When the first reply comes it is set to ~NOREPLY which indicates a reply has indeed come. This is done because if the first reply has not yet come and if a request for unselect comes then the routine "fin_pen()" is called with the parameter SA_USL and in this routine this bit is tested for the first reply and if it hasn't yet come then the status word is set as UNSELECT(ed) and nothing else is done. Now when the first reply comes it is tested for UNSELECT and if it is true then a normal reply is given back to the user and the packet is not put in the pending list as would be done after the first reply. This would unselect the select request. Now, if the first reply has come when "fin_pen()" is called, then that packet is finished off by the "ackuser()" routine and also the AST field of the I/O packet is reset so that control does not come to the ast service routine for the select request in the user Task after a request has been given for unselecting the socket.

5.5 OUT OF BAND PROCESSING:

The out of band mechanism is one in which a user can either send out-of-band packets to the remote systems or receive them from the remote systems. The sending of out-of-band packets is a very straight forward mechanism but receiving the packets can become a pain if none is received and the user wants to exit from his task. An I/O will remain pending and the task will remain marked for abort and will hang. When a socket is closed, and if an out-of-band request is pending, which will be a very common case because while the out of band request is pending and if the user task exits or if he aborts the task, then the control comes to the "iokill()" routine which issues an SOCLOSE to the board for that socket. This will still not force a reply for the OOB request because there are no OOB packets available. Hence, when the reply for the SOCLOSE comes the routine "fin_pen()", with the parameter as SA_ROO i.e. remove out of band request, is called. This routine removes the OOB packet from the pending list and finishes the I/O on it by calling "ackuser()" and it also decrements the rundown count. This will cause the channel to close which in turn will cause the task to abort peacefully.

5.6 SETUP PROCESS:

The routine "exsetup()" is called when the request is made to the ACP for initializing/configuring the board - IO_EXC|EX_INI. This routine is called with a parameter called the setup mode. If it is 0x80 then infinite timeout is specified for debugging purposes with the ON-BOARD debugger.

In the setup process, the important data structure is the configuration message which contains information like the interrupt vectot address, the start of the message area, the types of longwords used by the host (byte swapped or not), status bytes, the reply status bytes, etc. The start physical address of this configuration message is passed to the board software by writing it byte by

byte into the PORTB and reading the status from the PORTA.

First the host message area is setup in a way specified in the EXOS 203 manual. The offsets in the message area are calculated by finding the differences between the physcal addresses (which are calculated by the routine "reloc()"). The field in the configuraiton message for the start of the physical address is a longword and is an 18-bit value for the UNIBUS and a 22-bit value for the Q-BUS. After preparing the configuration message, the board is reset by writing a 0 into PORTA. After a 2 second delay the PORTB is read to find out whether the board has been initialized or not. If mode is 0x80 then infinite timeout is given for the board to get reset else only 2 sconds are given for resetting the board. The value of the PORTB is stored and later initialized to the im_dummy2 field of the configuration message. The netload program uses this field to indicate whether the loopback test failed or not (that is if the Xceiver cable is in or not). Then the start physical address of the configuration message is passed to the board by writing it into the PORTB. This address is calculated by the "reloc()" routine for the Q-BUS software but this 22-bit physical address is loaded into the UMR address and the 18-bit address is passed as the physical start address of the configuration message. (described in detail in the UNIBUS doc)

After the board is reset and it gets the configuraiton message, it prepares its local copy of the configuration message and sets up its message queues. After this the board is ready to take on any requests from the host and the host is prepared to take any unsolicited replies or solicited ones.

6. RESOURCE USAGE BY THE ACP/DRIVER:

This section describes some of the important system resource usage by the driver and the ACP.

6.1 MEMORY:

The driver size for the Q-BUS systems is only about 1KB. But for the UNIBUS systems it is the full 8KW as 7KW out of the 8KW are taken up by the local pool for intermediate bufferring. The ACP's size is almost 6KW for the UNIBUS systems and about 5.5KW for the Q-BUS systems.

6.2 SYSTEM POOL USAGE:

The driver uses the system pool only when the control comes to the cancel entry point. Here it allocates a packet from the system pool to queue to the ACP. It is deallocated in the ACP.

Depending on the number of requests made to the ACP at one time if the network is slow or if the board is slow in responding to the requests then all the I/O packets are hung up in the pending list of the ACP and this causes a depletion of the system pool. the size of the packet for RSX-11M systems is 36 bytes and for RSX-11M-PLUS is 40 bytes. Telnet also uses up a lot of system pool as described in it's respective document.

The driver data base is located in the the system pool. There is only one DCB, one UCB and one SCB for the RSX-11M systems and an additional CTB and a KRB/SCB combination for the RSX-11M-PLUS systems and together they take up about 110 bytes for the RSX-11M systems and about 140 bytes for the RSX-11M-PLUS systems.

6.3 EVENT FLAGS:

The event flag number 8 is used by the dealy routine after it marks the

time by spaecifying the event flag 8 and then waiting for event flag 8 to set. If an AST routine is added in the ACP (some reason knwn only to the person who will add it!!) it must not use this event flag or any other used in the QIO calls in the ACP. The telnet requests also use the efn 1 for QIO's to the ZTDRV for input and output interrupts.

6.4 CPU TIME:

The driver hardly uses the CPU time since as soon as it gets a request, after a bit of processing, it queues the request to the ACP and returns back to the system. The ACP is in a forever loop and it seems as though it might take up a lot of CPU time but most of the time it is stopped and waiting for a request to come and it would then wake up. If the traffic is more then the CPU time usage will be more.

6.5 LUN'S:

The ZEACP task does not use any file system so it really does not need to specify more LUN's than are assigned to it by default by the task builder. But the routines for telnet require to issue QIO's to the the ZTDRV and they assign the LUN 7 dynamically to one of the 8 units whichever is required to communicate with.

7. ENHANCEMENTS AND IMPROVEMENTS:

7.1 CHANGE IN THE DATA BASE:

There is one change that is suggested to be made in the data base for the ZE driver. The CSR address is to be stored in the KRB of the data base and it has to be a valid one because the CON task, while putiing the device controller online, probes at this address in the I/O page to see if the device is actually present or not. Hence, this field will have to be initialized to a global symbol which will be initialized during task build time and its value will correspond to the actual CSR on the particulare host system which the end user will supply during the build time.

At present the interrupt address is initialized in a similar way. This is only required for the RSX-11M-PLUS systems and not for the RSX-11M but since the data base is generic for both, the change will affect bothe distributions(?)

7.2 ADDING MORE CALLS TO THE DRIVER:

If, in the future, another QIO call is to be added for the driver, then it can be done very simply. It's mask will have to be added into the D.MSK field of the DCB and a case statement is to be added in the request routine's switch statement and the serving routine can be called here. If it requires to give an immediate reply to the user then the inform bit should be set and so on. All the protocols used by other requests should be followed - if the message slot is not used then the action bit is not set and in that case the slot is returned unused. Similarly the case statement is to be added in the reply routine for the reply processing.

THE DESIGN/MAINTAINANCE DOCUMENTATION FOR THE RSX-11M/RSX-11M-PLUS UNIBUS SOFTWARE

by

Asim K. Mehta

Note: Adequate knowledge about the design of the EXOS driver and ACP for the RSX-11M/RSX-11M-PLUS (Q-BUS) systems is required to thoroughly understand the design of the UNIBUS software for the same Operating systems (it is described in the relevant design/maintainance document).

1. INTRODUCTION:

The whole driver/ACP software is written in such a way, that, for the respective type of the bus, Q-BUS or UNIBUS, the build procedure will conditionally compile and task-build the software to suit the type of the system.

The main difference in the Q-BUS and the UNIBUS software is the use of the UNIBUS mapping registers for transfering data to/from the board to the host memory. This document will describe in detail how these are allocated, how they are used in data transfers, etc.

2. DESIGN DETAILS:

2.1 UMR REQUIREMENTS:

With one UMR, a transfer of a maximum of 4KW of data can take place. The ACP requires about 1KB of memory for the message area and this piece of memory is shared by both the host and the board. Both of them require to utilize this space almost simultaneously and hence this area has to be mapped by one UMR all the time. This UMR is allocated during initialization time of the ACP and is loaded with the 22-bit physical address of the start of the message area.

For data transfers from user tasks to the board memory and vice-versa, ideally, for each request one UMR (per 4KW) would be assigned for the transfer and would be loaded with the physical address of the start of the user buffer. For a write request this sounds quite O.K. but if a read is requested then it may hang forever thus tieing up the system resources (UMR's) and degrading system performance because there are only 32 UMR's available for the whole system including for the disc I/O and other peripheral I/O.

To solve this problem, a fixed local pool of about 14KB is allocated in the ZE (EXOS) driver virtual space which uses only less than 1KB of virtual memory for its code. This is further subdivided into fixed parts of 1KB each so that each can be allocated for a request (which will NOT specify more than 1KB as the buffer size) and then deallocated when the request is over. If all the buffers get allocated then the requesting task would be blocked until a buffer is freed when the request is made again for a buffer in the pool.

For this pool area only two UMR's would be required for as long as the ACP is running. The contents of the user buffer would first have to be transfered to the allocated buffer for a write request and the board is to be informed about the 18-bit physical address corresponding to the start of the allocated buffer which is the UMR originally allocated plus the buffer no. times the size of the buffer. The buffers starting at the address greater then 4KW

from the start of the pool are assigned the second UMR's 18-bit address plus their no. times their size. The first UMR is loaded with the 22-bit physical start address of the pool area and the next UMR is loaded with the start of the pool 22-bit physical address plus 4KW.

Hence, the total consumption of the UMR's is three for almost all of the time.

2.2 VIRTUAL TO PHYSICAL ADDRESS CALCULATION:

The virtual address is converted into the 22-bit physical address with the use of the system routine $RELOC. This routine is called with the virtual address as the input in R0 and it returns the relocated address in two registers. R1 contains the relocation bias and R2 contains displacement bias in the block plus 140000 (PAR6 bias). Actually the relocation bias is the higher 16-bits of the physical address and the lower 6 bits of the displacement bias are the lower 6-bits of the physical address because the relocation bias is to be loaded into the PAR6 and the displacement bias contains the the virtaul address to be actually addressed. The displacement bias's higher 3 bits are 6 which select the APR 6 and hence the required physical memory will be addressed. But we donot need to address the physical memory but to calculate it and this is simply done by manipulating (by shifting and masking) these two registers to get the higher 6-bits of the physical address in one word and the lower 16-bits in another. The routine "RELOC::" actually does this in the ACP and also the power up for RSX-11M and load for RSX-11M-PLUS entry points do the same.

2.3 LOCAL POOL ALLOCATION:

The local pool for intermediate bufferring is allocated in the driver virtual space beginning exactly after 1KW from the start of the driver code area. This is done while the driver is being loaded. The driver is called at its power fail entry point while it is being loaded for RSX-11M systems and at the loadable driver entry point for the RSX-11M-PLUS systems. Here the driver calculates the physical address of the start of the pool area and stores it in two words in the UCB - at U.ACP+2 and U.ACP+4 with the lower 16 bits in the higher word and the higher 6 bits in the lower word. Now that the start of the local pool is in the system pool (UCB), the ACP can easily access it.

2.4 UMR ASSIGNMENT:

The three required UMR's are assigned at initialization time of the ACP. The routine $ASUMR is called for the purpose and not $STMAP or $STMP1 as these calls are for assigning UMR's for the duration of the data transfer and are deassigned as soon as the I/O is finished by the Executive and the ACP does not keep much of the control of the UMR's. $ASUMR just assigns the UMR's and it is the the ACP's responsibility to deassign them (which is done when the ACP is aborted for restaring the network or shutting it down by the call to the routine $DEUMR).

The routine "ass_umr()" is called at initialization time by the routine "uni_ini()" which actually does this assignment and initialization of the UMR's. There exists a 6-word Unibus Mapping Register Assignment Block in the SCB of the driver data base. The start address to this block and the no. of UMR's to be assigned in one of it's fields is passed as the input to $ASUMR. This routine, called at system state (done in Macro routine ".AS.UMR::"), returns the UMR address and the 18-bit physical address mapped by this UMR (giving the UMR number from the higher 5-bits) in the different fields of the UMR Assignment

Block. The no. of UMR's specified will map 4KW of physical memory each, and these 4KW of memory mapped by each UMR's will have to be contiguous in the physical memory. For this reason, two UMR's are assigned first for the pool area and one assigned later for the message area and it's UMR Assignment Block is allocated from the system pool.

2.4 DETAILS OF FORMING THE 18-BIT UNIBUS ADDRESS AND LOADING OF THE UMR'S:

The UMR Assignment Block contains 6-Words as described in the section 7.4.2 of the Guide for writing I/O drivers manual for RSX-11M-PLUS. After the call to $ASUMR, the field M.UMRA is initialized with the address of the UMR (in the I/O page). The field M.UMVL is initialized with the lower 16-bits of the 18-bit address mapped by the first assignd UMR. The bits 4 and 5 (counting from 0,1,... onwards) of the field M.UMVH are initialized with the two higher order bits of the 18-bit unibus addresss. The higher 5-bits of the 18-bit unibus address determine the number of the UMR that will map the physical memory. This UMR is to be loaded with the 22-bit physical address of the buffer the peripheral device has to communicate with. To access the next contiguous UMR, the UMR number is calculated by fetching the high 5-bits of the 18-bit physical address and then adding one to this to get the higher order 5-bits of the new 18-bit unibus address of which the lower order 13 bits are same as the previous UMR.

The Unibus Mapping Registers are actually a set of 32 two word pairs in the I/O page starting at the location called UBMPR. The two words hold the 22-bit physical address to be mapped by that particular UMR. The address of the UMR is in the field M.UMRA in the UMR Assignment Block and the lower order 16-bits are loaded into the lower word and the higher order 6-bits in the higher order word (This is done by simple move instructions).

2.5 LOADING THE UMR'S:

The first UMR is loaded with the start physical address of the pool area and the next one with the start address plus 4KW. The third UMR is loaded with the start physical address of the message area. All the fields in the configuration message related to the message area are just offsets relative to this start address. Under normal circumstances these UMR initializations would remain permanent.But during the time when the board is being setup, the board needs to read the configuration message directly from the host memory. This requires a UMR assigned and loaded with the start of the configuration message for a short duration of time. This is temporarily done in the routine "exsetup()" and the UMR is reloaded with the start of the pool area when the board has finished reading the cofiguration message and has initialized the board and its message queues.

2.6 DEASSIGNING THE UMR'S:

At initialization time (in the routine "uni_ini()") a system call SREX$S is made (from the routine "srex()", which specifies the routine "DE.UMR" so that control comes to this macro routine whenever the ACP is aborted or it exits. This routine calls the system routine $DEUMR to deallocate all the three UMR's with the input as the start of the UMR Assignment Blocks and then exits the ACP peacefully.

2.7 LOCAL POOL MANAGEMENT:

An image of the local pool (struct pool_im in file unidata.h) is kept

in the ACP which holds information about the allocation of the buffers and the owners of the allocated buffers. The pool_im structure is as follows:

```
#define POOL_BUFS 14
struct pool_im {
        Ushort state;
        struct iopkt *owner;
        } pool_im[POOL_BUFS] = {0};
```

The state field indicates whether the particular buffer is allocated or not. The owner field contains the address of the I/O packet which corresponds to the I/O request from the user task.

2.7.1 POOL ALLOCATION:

The pool allocation is done (in routine "getpool()") by first finding a free buffer and in the process also finding the buffer number from the pool image. The I/O packet address, which is passed as the first parameter to this routine, is stored in the owner field of the pool image. The 18-bit physical address is calculated by adding the buffer size times the number of the buffer to the start 18-bit physical address of the start of the local pool. This 18-bit start address is calculated during the UMR assignment time after the UMRs are assigned from the information present in the UMR Assignment Block and stored in a global variable (unilbuf) for the pool management routines to use. If the buffer number turns out to be greater than 8, then the 18-bit address of the next 4KW of the local pool is taken which is stored in another global variable (uni2buf). These variables are long words.

2.7.2 DATA TRANFER TO/FROM USER/POOL ADDRESS SPACE:

The second parameter to the "getpool()" routine indicates whether the requested buffer is for a read or a write request. If it is for a read request then the parameter is 0 and 1 if it is a write request. For a write request, the routine copies the contents of the user's buffer into the buffer in the local pool allocated for the purpose. For this copying, the Macro routine "acopy()" is called which calls the system routine $BLXIO to do the transfer of the data from the user's area to the driver's area where the local pool is situated. This routine need the relocated addresses of the source and the destination buffers. The relocated address for the user's buffer is already present in the I/O packet but the relocated address of the pool area is calculated at initialization time by the Macro routine "REL.POOL::" and stored in global variables rel1buf and rel2buf for the 1st and 2nd 4KW of the local pool respectively.

2.7.3 POOL DEALLOCATION:

The routine to free the buffer, when the request is over and the reply has arrived, is "freepool()". The first parameter is the I/O packet address for which the request was made and the second one indicates whether the request was for read or write (0 or a 1 resp.). The pool image is searched for an entry corresponding to the I/O packet address passed as the 1st parameter and if a match is found then that entry's status field is initialized as DEALLOCTED. If the request had been for a read then the data from the pool is transfered to the user's area by the same routine "acopy()". For a write request nothing is done.

There are requests that require both read/write kind of interaction with the board like the requests for ARP, ROUTE etc. The "getpool()" and the

"freepool()" routines are both called with the second parameter as 1 so that the user's read/write requests are both honoured.

3. CHANGES IN THE XOSLIB TO PASS ONLY 1KB OF DATA TO THE ACP FOR UNIBUS M/C's:

The routine which finally does the QIO to the board - "libemt()" - is modified for the purpose. A global integer called unibus is initialized to 0 at compile time and this indicates a Q-BUS machine. If it is 0 then libemt does not check the buffer size and directly passes the buffer and the buffer size to the board (ACP). But if it is set to 1, then libemt breaks up the buffer into 1KB blocks and issues QIO's in a sequence with each having no more than 1KB of data to be transfered. The value of unibus is zapped to 1 for UNIBUS M/C's.

4. LIMITATIONS:

4.1 SPEED:

The main limitation with respect to the Q-BUS driver/ACP is the speed of data transfer. Since intermediate buffering is inevitable in the UNIBUS design, as described above, the time taken to first transfer the data from user buffer to pool area or vice versa is an extra burden and slows down the data transfer by about 40%.

4.2 EXHAUSTION OF POOL SPACE:

If there are many tasks requesting for the pool space for data transfer the pool area might get exhausted and in such a happening the ACP will put the requesting task's I/O packet in a secondary queue which is again put into the internal queue after all requests are honoured and so again they become eligible for requesting the pool and again, if no pool has become free then the process is repeated until a buffer gets free and then this request is honoured. During all this time the buffers are not free, the task will keep waiting and hence will eat up that memory space as it cannot be checkpointed during the time the buffered I/O is in process. This is because task checkpointing during the buffered I/O is not implemented because the same code is being used for the Q-BUS machines which donot indulge in buffered I/O. To implement this the code size would increase and would further complicate the already complicated logic of the ACP making it difficult to maintain.

4.3 BUFFER SIZE:

The buffer size specified by the user should not be greater than 1KB and if it is then an error status is returned and the request is not honoured. The user is advised to do a series of QIO's to transfer more than 1KB of data This might further slow down the process of data transfer.

4.4 INEFFICIENT USE OF THE POOL AREA:

The pool is divided into 14 buffers of exactly 1KB size. This means that for a data transfer of less than a hundred bytes would use up 1KB of pool space and a task requesting more than 1KB would then have to wait. This limitation is due to the simpplified approach used in managing the pool and thus keeping the size of the ACP to the minimum and the code simple. This problem would arise only when the traffic is very high and all the pool space gets exhausted but normal circumstances when one FTP client and one FTP server plus a telnet client are running there wont be any problem depending on how fast the network

is.

4.5 UNIBUS FOR PDP-11/70

It is not certain that the current software would run properly for the PDP-11/70 processor since that processor uses the MASSBUS. Unless this software is tested on such a machine nothing can be said about its performance on that machine but the best edducated guess is that it shiould work!

5. ENHANCEMENTS AND IMPROVEMENTS:

5.1 POOL MANAGEMENT:

This could be made more complex by making it to allocate any given numbers of bytes in a way similar to the "malloc()" and "free()" routines in a high level language run time support. But an upper limit of 2 or 4KW would anyway will have to be put because if, for example, the "TTCP" program does a read for 4KW in loopback mode then the other TTCP will have to do a write of 4KW and hence they would both be hung up for ever. Hence, the complexity is the main thing that will increase for better pool management.

5.2 TASK CHECKPOINTING DURING THE INTERMEDIATE BUFFERRING:

As described in the limitations this feature is not implemented but it can be done by using the routines $TSTBF, $INIBF and $QUEBF as described in the Guide to writing I/O drivers for RSX-11M-PLUS, section 1.4.8. This feature would definitely improve system performance as the memory would not be tied up by the issuing task as it would be checkpointed. This could be done for both read and write requests.

RSX TO MPLUS ---> MAJOR CHANGES

The following changes were necessary to be made in the EXOS driver ZEDRV/RTHACP for the RSX-11M to make it possible to run on the RSX-11M-PLUS operating system.

The RSX-11M-PLUS O.S. has some added features incorporated to support different kinds of controllers and the system has taken more control over the handling of different types of controllers. There are two major data structures added for this purpose - The CTB (controller table) and the KRB (controller request block). The CTB defines the type of controller and the KRB describes individual controllers and their characteristics.

In the existing data structures for the RSX-11M driver the only ones that have almost remained the same are the DCB (device control block) and the UCB (unit control block). The SCB (status control block) has changed.

The other major change in the driver code is the Driver Dispatch Table (DDT). There are some new entry points that have been added which are helpful in initializing the driver, getting the controller and units on/off line etc.

1. THE DETAILED DESCRIPTION OF THE CHANGES:

1.1. DCB:

no changes.

1.2. UCB:

U.UCBX is an added field. Also initializing the units as offline. (they will be made online by the CON task.)

1.3. SCB/KRB:

The SCB and the KRB are to be made contiguous which means no more than one unit can operate at a time on one controller. Since the EXOS controller does not use this strategy of physical units attached to the controller, but has the concept of logical units (channels), this minimal strategy is maintained. There are some new fields added to the SCB concerning error logging, I/O page registers, KRB address, status fields etc. The KRB has information about the status of each controller, the interrupt vector address (which was first in the SCB), CSR address, priority, UCB table, I/O count, active unit's UCB address etc.

1.4. CTB:

This describes the characteristics of the EXOS ethernet controller like the name, status, pointer to DCB etc.

1.5. DDT:

The driver dispatch table is now just a Macro call which initializes the dispatch table. This contains information regarding the various entry points to the driver - the four conventional ones; initiator, cancel, powerfail and timeout plus the new ones specially for the

RSX-11M-PLUS system - the loadable driver entry point, unload entry point (these are called while loading and unloading the driver), the controller and unit online/offline entry points (to perform certain functions while bringing the controller and units on/off line).

There has been no change in the logical flow of the driver code but the powerfail entry point for the RSX-11M is now the load entry point for the RSX-11M-PLUS system.

1.6. ACP:

The ACP, being a task, has not suffered many changes. The only place where the problem arises is in the file UNIMAC.MAC where the offsets refering to the SCB are not altogether symbolic and hence the offsets get changed. Some conditional coding has been added here such that both the systems would get their respective offsets.
(the conditional coding for UNIBUS and Q-BUS M/C's would remain as such)

Most of the code that has been changed has been condionalized at the assembly level such that it will also run on the RSX-11M Q-BUS or UNIBUS systems. Digital only allows user written device driver names staring with 'Z' for RSX-11M systems and the ones starting with 'J' or 'Q' for RSX-11M-PLUS systems. But to maintain the simplicity in maintaining the code ,i.e. having one piece of code conditionally written such that it will run on all the four types of systems - RSX-11M (UNIBUS and Q-BUS) and RSX-11M-PLUS (UNIBUS and Q-BUS), the driver on the M-PLUS system was also given the name 'ZE'. This was not according to the conventions of DEC but, well, our convenience is first preference!

2. CHANGES FOR THE UTILITIES AND XOSLIB IN CHANGING FROM RSX-1M TO RSX-11M-PLUS:

The main changes made were in the files radix.mac, pasword.mac, xinitenv.c. These changes were such that these files could also be used for the RSX-11M systems. The changes were as follows:

1. radix.mac: It did not support the blanks in the input ascii name and now it does.
2. pasword.mac: There were some potential bugs in the RSX-11M version which came to light in the M+ software and were fixed. The account file was not being closed by the login task because it was first exiting after validating the account. But when the strategy to keep the login task running all the time, letting it dequeue packets for validating the account, was made, the login task never closed the account file and no other user could login. Earlier, when it was exiting, the file was being closed.
3. xinitenv.c: The task name of the login and master tasks in the M+ are different from that in the M software. To take care of these differences the executive call get task info is called and it is checked which system this task is running on and then the correct task name is issued in the send data requests.

DESIGN/MAINTAINANCE DOCUMENTATION FOR THE TELNET SERVER ON RSX-11M/RSX-11M-PLUS

by
Asim K. Mehta

1. INTRODUCTION:

The Telnet server comprises of Three distinct parts:

i) The ON-BOARD Telnet Server (which is downloaded onto the board),
ii) The routines in the ACP which handle the Telnet Server requests and
iii) The Pseudo Terminal Driver which actually serves the remote terminals.

The first part, the ON-BOARD Telnet Server is not host dependent and will not be discussed here. The second part is the interface between the first and the third. These other two parts reside on the host and need a thorough investigation as to how the design was done and how to maintain them.

2. OVERVIEW:

The Board/Host interface regarding telnet is described in the "ON-BOARD Telnet Server To Host Interface" by George Powers.

The ACP receives the requests from the remote terminal via the EXOS-to-HOST message queue and gives back replies to the remote terminal via the HOST-to-EXOS message queue (The method of the ACP receiving messages and giving back messages from/to the board is described in the relevant design document). On receipt of any request/reply for telnet, the ACP dispatches it to the relevant routine which does the job of interfacing with the Pseudo Terminal Driver/EXOS board.

The interface with the Pseudo Terminal Driver (called ZTDRV) is similar to that of a normal modem multiplexer used with the TTDRV (like the DLV11-E asynchronous line interface with full modem control). Except for the concept of ringing, everything else is almost similarly modelled. Ofcourse, there are no CSR's in our case as it is modelled as a pseudo multiplexer and the input and output interrupts are simulated from the ACP by QIO calls to ZTDRV.

3. DESIGN DETAILS OF THE BOARD TO HOST (AND VICE-VERSA) INTERFACE FOR TELNET:

The Host and Board communicate via the message queue mechanism and the Telnet Server requests are distinguished from other requests by the nm_request field of the message structure called Telnet_srvr which is initialized as TSCOMMAND for telnet requests/replies. As soon as the "request()/reply()" routines recognise the request to be that for telnet, they pass control to the routines which handle telnet requests/replies.

If the request is from the board then it is an unsolicited reply for the ACP and the routine "reply()" recognises it as one for telnet and calls the routine "dispatch()" (in file RTH.C) which dispatches to the correct routine depending on the telnet command specified in the nm_tsrqst field. The following commands could be expected from the board and the appropriate action is taken as described below:
(the routines to which the dispatcher dispatches are all in the file RTH.C)

3.1 TSCARON/RLCARON:

This command tells the host that the carrier is ON for a remote terminal whose pty no. is in the field nm_sioid. The dispatcher calls the routine

"caron()" which establishes the carrier ON and enables the unit (US.CRW clear and US.DSB clear) in the ZTDRV database. It also sends a CNTRL'C' to the ZTDRV as an unsolicited input so that an MCR> prompt is sent to the remote to indicate a successfully established connection.

3.2 TSCAROFF:

This is sent to the host when the remote terminal wants to break the connection. The routine "bye()" is called for the purpose. It gives a CNTRL'C' followed by a 'BYE\r' to the ZTDRV as an unsolicited input which logs off the user from the system. The ^C is given because, for example, just in case text edition is in progress then the line "BYE\r" will be written as new text instead of a logout request. ^C will put the process in the background and then logout the user.(Won't work for EDT, though!)

3.3 TSREAD:

The remote terminal sends unsolicited input to the ZTDRV via the read data stream in the array tsdata[] field of the Telenet_srvr structure. (It may be just be read data for a process running on the remote terminal and not unsolicited input!) The routine "zt_read()" is called by the dispatcher ("dispatch()") which passes the data to the ZTDRV by a simple QIOW #IO.INP call which is accepted by the ZTDRV as an input interrupt and the data is input into the driver and processed normally (described later in this doc as to how).

3.4 TSNVTFUNCT:

These are requests for the standard Network Virtual Terminal Functions which are described below:
(They are serviced by the routine "nvtfunct()" called by the dispatcher.)

i) AO - abort O/P - ^O is sent to ZTDRV as an unsolicited input.
ii) AYT - are you there? - ignored as the board takes care.
iii) EC - erase character - BS is sent as an unsolicited input.
iv) EL - erase line - ^U is sent as an unsolicited input.
v) IP - interrupt process - a ^C is sent as an unsolicited input.

3.5 TSDOOPT:

The board sends certain telnet options which the client requests and the host is supposed to fulfil these options as far as possile. The routine "do_option()" is called to set the options. The following are the possible options that would be asked to be set by the telnet client:

i) TELOPT_BINARY - a QIOW #SF.SMC is sent to the ZTDRV to set this option with the bit TC.BIN set.
ii) TELOPT_ECHO - same as TELOPT_BINARY but here the bit is TS.NEC that is cleared to set the echo option.
iii) TELOPT_SGA - suppress go ahead - no action is taken.

3.6 TSDONTOPT:

The function "dont_option()" is called which calls "do_option()" with the second parameter non-zero indicating it to reset the options instead of setting them.

3.7 TSWRITE (h2x):

When the System has to send some data to the remote terminal, then the ZTDRV sends the write data in an I/O packet queued to the ACP via $EXRQF system call. The function code is a pseudo fn code IO_TEL with which the ACP (routine "request()") recognises the request as one for telnet to be sent to the board. The routine "telnet()" is called which prepares the message queue (Telnet_srvr) (by calling "wr_to_exos()") from the information present in the packet, and thus the write data is sent to the remote terminal. Then this packet is deallocated back to the system pool (as it was allocated in ZTDRV from the system pool and this is not a regular I/O packet but one to serve our purpose of sending data to the board).

3.8 TSWRITE (x2h):

This is a reply from the ON-BOARD telnet server to the last TSWRITE (h2x) request and this is considered as an output interrupt to the ZTDRV to signal the completion of an output to the board. The output interrupt is given as a QIO #IO.OUT in the routine "write_reply()" dispatched to by the routine "dispatch()". This is a simulated output interrupt and the ZTDRV takes this as a normal QIO request but the controller dependent routine considers it as an O/P interrupt.

3.9 TSHANGUP (h2x):

This is a request which the host has to make to the ON-BOARD telnet server when a remote terminal logs out of the system. When the user types in 'bye' or 'logout' as an unsolicited input, the BYE task is invoked which first logs off the user and then calls the ZTDRV with a QIOW #IO.HNG which gives control to the time out entry point of the controller dependent routines and here a packet with a pseudo fn code TS.HNG is created and queued to the ACP via the $EXRQF system call. The ACP, after getting this packet, gives control to the routine "hangup()". This routine prepares the message area (Telnet_srvr) and sends the TSHANGUP request to the ON-BOARD telnet server which severs the connection fot that pseudo tty.

4. ZTDRV - THE TELNET DRIVER:

The ZTDRV is a pseudo terminal driver for the remote terminals and actually does the character processing. Most of the ZTDRV code stems from the standard TTDRV code for the RSX-11M/RSX-11M-PLUS systems. The module which actually does the interfacing with the standard terminal driver code is the controller dependent routine for the new pseudo controller added into the existing terminal driver. This pseudo controller is called the DT-11 and the controller dependent routine is called ZTYT. The reasons for the pseudo controller not being added to the existing terminal driver are described in the next section. The code for this pseudo controller dependent routine and the rest of the TTDRV code plus the changes in it to suit the new pseudo controller is named ZTDRV - the new pseudo terminal driver for telnet.

4.1 DECESION FOR KEEPING ZTDRV AS A SEPARATE TERMINAL DRIVER:

This decesion was taken for the following reasons:

1. It would be a lot easier to debug a separate driver rather than the TTDRV which would be already resident and to make some change in the

driver, Sysgen would have to be performed all over again to rebuild it.

2. To add another controller to the existing TTDRV would mean that the source files of the standard TT driver would have to be modified and this would mean a re-Sysgen to incorporate the new TT driver with the pseudo controller. The main aim of the present EXOS software is to try to incorporate networking on existing systems and it would be ridiculous to ask the customer to do a SYSGEN to incorporate the pseudo terminal driver.
3. There are certain terminal characteristics which are necessary for the pseudo controller like modem support which might not be supported on the user system. To add that support a re-Sysgen is necessary.

The main drawback of this decesion is the utilization of a lot of resident memory space - 8KW - as the ZT driver is always resident in the memory while it is loaded and its data base is always resident while it is unloaded. And it also utilizes a lot of space from the system pool as will be discussed in the section for System resource consumption.

4.2 CONTROLLER HANDLING IN A TERMINAL DRIVER:

The TTDRV handles different kinds of controllers especially made by DEC (e.g. DL,DJ,DZ,DH,DM,etc.) and each is of a different kind and has to be handled separately by the driver. Most of the code for the TTDRV is common to all the controllers. But, for their specific functions there are controller dependent routines which are called upon to do the required specific functions.

A typical flow of a normal controller action would be as follows:

4.2.1 A TYPICAL CONTROLLER ROUTINE FLOW:

When a character is typed from the terminal, an interrupt is raised which brings control to the input interrupt entry point of the controller dep. routine. This causes the routine to pass the character to the input character processing routine common to all the controllers and then if echoing is required then it is output via the output interrupt routine - the character is first put in the proper XBUF and the output interrupt is enabled. The controller raises the output interrupt which means the character has been successfully output and the control comes to the output interrupt routine. If there are more characters to be output then the same procedure is followed. When a task has to output any buffer onto the terminal, then it calls this output interrupt routine and the same procedure takes place.

When the TT driver wants to stop the output say, when a ^S arrives, then the controller dependent routine is called at its stop output entry point. Here the output interrupts are disabled by setting the appropriate bit in the CSR. Similarly there are other entry points for other purposes like the resume O/P entry point, the modem timeout entry point, the power-up entry point, etc which are called when the appropriate action is required.

4.2.2 DATA BASE RELEVANT TO THE CONTROLLER DEPENDENT ROUTINES:

For the RSX-11M systems the following data structures are relevant for for the controller dependent routines:

1. The controller type. It is a number given to different controller types by DEC and the different controller types are accessed by this number.

2. The controller index. For a particular type of a controller, there may be more than one controllers existing simultaneosly. These are given numbers called the controller index.
3. The controller table CTBL. This is a dispatch table containing the addresses of controller dependent routines which are to be called whenever required by the driver. Each routine has its particular number and this allows proper dispatch for any controller.
4. The UCB table. This is a table of UCB and the CSR addresses for a particular type of a controller by which, when it is interrupted, it can get the UCB and the CSR address of the correct unit by indexing the table with the controller index which is passed in the PS word (bits 0-3) when an interrupt arrives.
5. The UCB and the SCB are also extensively used by these routines.

For the RSX-11M-PLUS systems the following data structures are relevant on top of the ones discussed above for the RSX-11M except the UCB table which is not used here:

1. The Controller table CTB. This is a data structure in the pool area and has information like the controller name, addresses of contoller request blocks, some status information, link to the next controller table, etc. Each controller type is defined by such a block.
2. The Controller request block KRB. This contains all the information like the CSR address the controller type, the vector address etc. Every controller has to have one such block by which its run-time status, its controller index, etc. can be determined.
3. The SCB and the KRB may be contiguous for controllers having only one unit and allowing full duplex operation.

Please see the guide to writing I/O drivers for RSX-11M-PLUS for further information on these data structures.

4.3 THE PSEUDO CONTROLLER FOR TELNET:

4.3.1 OVERVIEW:

To interface the telnet protocol to the system, there was a need to communicate between the terminal driver and the ACP, since it was the ACP that got all the telnet protocols from the board. The best way was to model a pseudo controller in the ZTDRV which would do this job. Hence, the main function of this module would be to somehow take in characters received from the remote terminal and input them to the input character processing routines of the terminal driver and to somehow get to output characters to the ACP which could transfer them to the board and finally to the remote terminal.

4.3.2 NAMING CONVENTIONS AND GENERAL DESCRIPTION:

This controller is called DT-11 and the module which handles this is called ZTYT. The controller number given to this pseudo controller is not fixed but is so coded that at assembly time it would get the last controller number after the ones defined by DEC. This is done to take into account the fact that DEC might upgrade the TTDRV by increasing the number of controllers supported by the terminal driver and that would conflict with our design. All the controller dependent routines start with the letter 'Y' and so our controller dependent routines are called 'YT...' as our controller name is D'T'-11. An assembly time label called D$$T11 has to be defined to inform the ZTDRV software of the existance of such a controller and its value indicates the number of

units of these controllers existing (8, in our case, at present).
The controller dependent routines for this controller are added to the controller table CTBL and hence they would be called whenever there is a request for this controller. The controller type is stored in the UCB for RSX-11M (U.CTYP) systems and in the KRB for RSX-11M-PLUS systems (K.PRM). It is from here that the driver accesses the controller type and then dispatches to the required routine.

4.3.3 THE RELEVANT DATABASES:

Besides the data strucures required by the System viz. DCB, UCB and the SCB for RSX-11M and on top of these the CTB and KRB for RSX-11M-PLUS there are a few used by the controller dependent routines for the pseudo controller DT-11. These are added separately and are described below:

UCBADD --> local storage for UCB address for use by the conroller dep. routines for the pseudo controller.
LOCBUF --> stores upto 32 input characters temporarily.
COUNT --> byte count for the I/P characters.
ADLBUF --> address of pointer to I/P characters.

Also added are the input and output interrupt entry points for the controller which correspond to the I/O function codes added - IO.INP and IO.OUT in the dispatch table for the entry points for different function codes - QPDSP These are called QPINP and QPOUT. The initiator entry point for the ZTDRV dispatches to the required routines according to the function codes specified and hence for IO.INP and IO.OUT the contol comes to QPINP and QPOUT. These function codes are also added in the DCB for the pre-driver processor to recognise these I/O codes.
The UCB table is added just for consistency requirements in the terminal driver code but here there is no functional use for the UCB table.
All the detailed description of these added data structures are given in the section on maintainance of the ZTDRV with filenames and line numbers.
In the RSX-11M system there is a DCB describing the device type for the ZTDRV which has fields describing the legal function codes allowed on this driver and also types o function codes allowed. There is one DCB for the ZTDRV. There is one UCB for each unit which has some static and some run-time status information of the individual units. At present there are only 8 units supported as more would eat up a lot of system pool. Since each unit is capapble of being active simultaneously, there exists an SCB for each unit which keeps run-time information.
For the RSX-11M-PLUS systems in addition to the DCB and UCB's there exists a CTB, the controller table describing the type of controller supported by the driver. There is one CTB descrinbing the DT-11 contoller whose name is 'ZT'. There exists a contiguous SCB and KRB combination since each controller has only one unit attached and also each units is capable of full duplex operation. The KRB describes each individual controller.
The important fields worth a mention in these data structures are as as follows:
DCB:
D.NAM --> device name 'ZT' by which the system will recognise the device.
UCB:
U.CTL --> control flag UC.QUE which calls driver before queueing the packet.
U.STS --> US.CRW says unit waiting for carrier.

US.DSB says unit disabled.
U.CW2 --> U2.RMT says unit is a remote one.

SCB:

S.VEC --> vector address initialized as 0 since no real interrrupts.
S.CSR --> CSR address also initialized as 0 since no real device.

KRB:

K.VEC --> vector address initialized as 0 since no real interrupts.
K.CSR --> initialized to the CSR for ZE device - ZECSR - since the 'CON' task requires to probe into the CSR before putting the devive or controller ON-LINE. This constant is defines during task building of the ZTDRV depending on what the actual CSR is. This is a suggested improvement but presently it is initialized to 164000.

CTB:

L.NAM --> controller name for the pseudo controler - initialized as 'ZT' since it does not take a separate name from the device name.
L.KRB --> table of KRB addresses for all the 8 controllers.

4.3.4 CONTROL FLOW OF TYPICAL TELNET REQUESTS:

The flow of the controller dependent routines is as follows:

When there is a request for making the carrier on from ther board for a particular pseudo tty then the routine in the ACP sets the unit as "not waiting for carrier" and enables the unit. This allows the request to come to the ZTDRV whenever there is a QIO #IO.INP for unsolicited input. The control first comes to the initiator entry point ZTINI. This routine dispatches to the proper function servicing routine using the table QPDSP. The control then comes to the routine QPINI for the function code IO.INP and to the routine QPOUT for the function code IO.OUT.

4.3.4.1 QPINP:

In the routine QPINP the input data is transfered into the local data structure LOCBUF and then one by one each character is input to the input character processing routine ICHAR1. The control flow is modelled similar to the DLV11-E with modem control. Then, for echoing the character, the start output routine YTSTAX is called which calls a routine OUTBUF which prepares a packet of 48 bytes from the system pool and queues it to the ACP via a $EXRQF. The TCB address of the ACP is found from the ZE data base U.ACP in its UCB. After the input characters are processed, the routines are called which process any other packet that would have arrived and also any other type of procesing like start unsolicited input processing, post fork processing etc.

4.3.4.2 QPOUT, OUTBUF:

For doing an output to the remote terminal, a QIO/QIOW #IO.WLB or IO.WBT is done which brings control to the controller dependent routine YTSTAX and this routine calls the routine OUTBUF which creates a packet in which the output data is stored and queues it to the ACP. After any data is queued to the ACP i.e. after data is output to the board, there has to be an output interrupt to acknowledge the completeion of output. The board gives a write reply after every write to the board and this is considered as the output interrupt and sent as a QIO #IO.OUT to the ZTDRV which brings control to the routine QPOUT in the ZTYT module through the initiator entry point ZTINI. Here the routine OUTBUF is

called where the output buffer is first checked for any bytes left to be output and if so then another packet is created and queued to the ACP which again sends an output interrupt. If there is no data left for output then the routine ODONE is called which finishes the I/O by an IOFIN.

4.3.4.3 YTRESX:

The resume output entry point is called whenever there is a ^Q in the unsolicited input data stream. For a typical controller this routine is supposed to enable the output interrupts which will resume the output. But here there is no way of enabling the output interrupt but to simulate one that will cause the output to resume as the main driver code resets the bit S1.CTS which was set by a ^S. The output interrupt is simulated by sending a dummy packet to the ACP with byte count as 0 and it recognises this packet and sends a QIO #IO.OUT and this starts the output in the usual way.

4.3.4.4 YTMTIM:

The modem time out entry point is called by the main ZT driver code whenever an I/O is cancelled by an IO.KIL (by doing an ABO to a running task on this terminal) and when a user loggs out and the 'BYE' routine gives a QIO #IO.HNG to the ZTDRV which calls the controller dependent routine at this entry point if the unit is a remote one. Here it is first checked if the user is logged in or not. If logged in then control has come due to an IO.KIl and this call is discarded and directly returned to the caller. If not logged in and if the carrier is still on (i.e. not waiting for carrier) then control has again for an IO.KIL as user is not logged in but could still run the HELP facility. If the unit is waiting for a carrier then the control has come from PPHNG, the routine that services the function code IO.HNG. In this case a packet with a pesudo function code (the one not described in the DCB) of TS.HNG is created and sent to the ACP via a $EXRQF (similar to that in YTRESX) and the ACP calls the routine hangup() to send a TSHANGUP request to the board. Here the unit is also disabled (US.DSB) and the routine PPHNG sets the unit as waiting for carrier.

4.3.4.5 YTUOFF:

For RSX-11M-PLUS systems control comes to this entry point whenever the unit is brought offline. Here the typeahead buffer is deallocated since it is allocated in the online entry point for the driver and not deallocated atall so if a driver is unloaded and loaded again, the previous address of the typeahead buffer remains in the UCB (which remains resident) and while loading the driver again the typeahead buffer is not allocated as some garbage address is present in that filed in the UCB. This causes the system to crash. If the typeahead buffer is deallocated when the driver is brought offline then that field is cleared and reloading the driver causes no problems.

4.3.4.6 UNITNO:

This routine calculates the unit number of the unit in question and stores it into the pty_no field of the packet queued to the ACP.

4.3.4.7 GETACP:

This routine gets the TCB address of the ACP from the ZE data base U.ACP of its UCB and returns it in R0.

4.3.4.8 ZTSET:

This is the setup routine for the input interrupt entry point similar to the TTSET routine in the TTDRV which is common to all the controllers. This routine's structure is similar to the TTSET's but since TTSET is called at interrupt level there are some extra things it does over there (calling $FORK etc.) which are not required here as control comes here via a QIO. This routine is called as a coroutine from QPINP and when input processing is over control comes back to ZTSET and here it checks if any other processing is required or not.

4.3.4.9 YTCOFF::

This is the controller offline entry point for the RSX-11M-PLUS systems and control comes here while taking the controller offline. Here the Clock Block that was allocated from the system pool is deallocated back to the system pool. First the clock block is removed from the clock queue by finding the entry in the link lisk for clock blocks called $CLKHD and then it is deallocated to the system pool.

5. RESOURCE USAGE BY THE TELNET DRIVER:

The telnet server, as a whole uses the following system resources:

5.1 SYSTEM POOL:

The main carrier for communication between the different of the Telnet Server is the I/O packet. This is allocated from the System Pool which is one of the most critical system resource and the whole performance of the system depends on this.

The ZTDRV's code size is around 4KW and the rest of the available 4KW are used up in forming the local pool which is used for allocating all sorts of buffers for internal use of the driver like the UCB extension, the type-ahead buffer, the buffers for intermediate bufferring, etc. If for some reason this local pool gets exhausted due to extensive load, then the system pool is used. This cannot be estimated but it depends on the load on the ZTDRV (no. of remote users, no. of tasks running on the remote terminals, etc.).

The data to be output to the board from the ZTDRV is transfered to the ACP via a packet allocated from the system pool. The size of this packet is 48 bytes. The ACP deallocates this packet only when the request from the ZTDRV is honoured otherwise it is kept in an internal ACP queue till it is serviced. The amount of such packets depends on the size of the buffer to be output and if the rate at which the packets are allocated is higher than the rate at which they are deallocated, then the system pool might get exhausted. This, again, depends on the amount of traffic in the ACP. Normally these rates are almost same.

When the ACP gives an O/P interrupt to the ZTDRV via a QIO #IO.OUT, a packet is used up for the QIO (18 words for the RSX-11M and 20 words for the RSX-11M-PLUS). But this packet is given back to the pool as soon as the control comes to the ZTDRV. Also for the unsolicited input a QIOW is done which uses up one packet. This is also almost immediately returned to the system pool as soon as the input data is transfered to the local buffer.

5.2 CPU TIME:

Most of the processing takes place at priority 0 and hence it does not hog the CPU at any time. Since there are no interrupts, the ZTDRV never operates at interrupt level and this causes no grief for other peripherals.

5.3 UMR'S:

The ZTDRV as such uses no UMR'S as it does not use the UNIBUS but the ACP does transfer the data to the board via the message area which contantly uses one UMR for the purpose.

5.4 EVENT FLAGS:

Only the event flag number 1 is used by the ACP for QIO's to the ZTDRV. So in adding any directive to the ACP this should be taken care of though it will not cause any trouble as it is used in blocked I/O's.

6. MAINTAINANCE GUIDE FOR THE ZTDRV:

The following is a line-by-line description of the changes done from the standard TTDRV to the make the ZTDRV. The reasons for the changes are also given and also their effects on the performance of the telnet operation:

6.1 In all the files of the TTDRV, the .TITLE TT... is changes to ZT... as these are the module names for the new ZT driver.

6.2 ZTDAT.MAC: .IDENT /04.03/

This file contains all the local data structures for the ZTDRV. These include the dispatch tables for different function code handlers, for the controller dependent routines, for the terminal characteristics routines, character processing routines, etc. Also these contain the definitions for the different controller types, terminal types, controller tables, etc.

1. Topic: Support for certain terminal characteristics is not there in certain versions of the RSX-11M. To take care of this sime .IF's have been added.

Line numbers: 451-460 After ".ENDM ETERM..."
500-504 After "TERM T.BMPI..."
522-526 After "ETERM T.V132..."
Changes to existing code:

Previously: "TTPHI == T.V2XX"

(The following are the line numbers after the changes)
(The .IF's and their corresponding .ENDC's are added but the rest already exists)

1. #451 : ".IF DF T.V2XX"
2. #453 : "TTPHI == T.V2XX"
3. #456 : ".IFF"
4. #458 : "TTPHI == T.BMP1" (added).
5. #460 : ".ENDC ;T.V2XX"
6. #500 : ".IF DF T.V2XX"
7. #502 : "TERM T.V2XX WID=80.,LEN=24.,HHT=1,SCP=1,CUP=3"
8. #504 : ".ENDC ;T.V2XX"

```
9. #522 :       ".IF DF T.V2XX"
10.#524 :       "ETERM  T.V2XX  ANI=1,DEC=1,AVO=1,EDT=1,SFC=1
11.#526 :       ".ENDC  ;T.V2XX"
12.#985 :       ".IF DF TC.SFC"
13.#987 :       "MCGEN  TC.SFC,U.TSTA+6,S4.SFC  ;SOFT CHARACTERS
14.#989 :       ".ENDC  ;TC.SFC"
```

2. Topic: Table of pointers to dispatch tables in controller dependent routines.

   line numbers: 530-644 After "ETERM   T.V2XX..."

   Changes to existing code: Addition of an entry into the dispatch table but DEC's future releases and addition of new controllers will not affect our code.

   Added code/data structures:

   1. #553: "I = 0"
      Constant symbol 'I' added for the purpose of calculating the controller type (index for these dispatch tables).
   2. #558: "I = I + 2"
      Iterate this expression the number of times as there are DEC's standard controllers so that I gets the value of the last controller plus 2.
   3. #610: "YTINDX == I"
      A global symbol defined as the controller type (I) and is used in the ZT data base SCB and the UCB.
   4. #614-644: "$YTTBL..."
      The dispatch table for the DT-11 controller with routine names starting with 'YT'

3. Topic: Verification of the value of the function codes and the dispatch table for processing different function codes before entering a packet in the I/O queue.

   Line numbers: 709-710 After "ASSUME IO.RTT/400,12..."
   766-767 and after ".WORD QPRLB..."

   Changes to existing code: Addition of entries into the dispatch table which will affect the future releases if DEC adds new function codes. There will be a conflict with our function codes (IO.INP and IO.OUT) and these have to have values such that they can index the last entries in the dispatch table which are contiguous entries.

   Added code/data structures:

   1. #709: "ASSUME IO.INP/400,13"
      #710: "ASSUME IO.OUT/400,14"
      These function codes are initialized with the values one more than the highest existing function code value i.e. IO.RTT/400 is 12 and the next higher value is 13 which is for IO.INP/400 and 14 for IO.OUT/400.
   2. #766: ".WORD QPINP"
      #767: ".WORD QPOUT"
      The entries in the dispatch table which are the input and output interrupt entry points for the DT-11 controller.

4. Topic: Local data structures added for the YT controller dependent routines.

   Line numbers: 1065-1068 After "OPTIMR::.WORD OPTIME..."

   Changes to existing code: none.

   Additions:

   1. #1065: "UCBADD:: .WORD 0"
      Storage for the UCB address.
   2. #1066: "ADLBUF:: .WORD 0"
      Address of the local buffer.
   3. #1067: "LOCBUF:: .BLKB 32."
      Local buffer for input characters.
   4. #1068: "COUNT:: .WORD 0"
      Byte count for the input characters.

5. Topic: Data structures are added to include ztdrv's own Clock BLock, Fork Block and UCB Queue.

   Line numbers: 1073-1110 After "COUNT:: .WORD 0..."

   Changes to existing code: none.

   Additions:

   In all from lines 1073 to 1110:

```
          X1=1
        X1=1
        .IIF NDF M$$PRO X2=1
        .IIF DF M$$PRO X2=M$$PRO
        .REPT   X2
        ZT$UQL=.
        .IF DF  M$$PRO
        LCKDF$  SPIN
        .IFTF
        .IIF NDF $ZTUQ $ZTUQ==.
        .WORD   0,.-2
;
        .IFT
        .WORD   X1
        X1=X1*2
        .ENDC
        .IIF NDF $ZTFB $ZTFB==.
        .WORD   0,0,0,0,0
        ZT$UQL==.-ZT$UQL
        .ENDR
;
;
;                                       INDEX TABLE TO ZT DRIVER
;                                       UCB QUEUE HEADS AND FORK BLOCKS
;
        .IF DF  M$$PRO
        X1=$ZTUQ+2
$ZTUQT::
        .REPT   M$$PRO
```

```
        .WORD   X1
        X1=X1+ZT$UQL
        .ENDR
        .ENDC
;
;
;                                       ZT DRIVER CLOCK BLOCK
;
$ZTCB:: .WORD   0       ; ADDRESS OF THE CLOCK BLOCK
          .EVEN
```

6. Topic: Table of pointers to UCB tables.

   Line numbers: 1245-1249 After "TTUCB::..."
   1274-1284 After "DL-11 Data bases..."

   Changes to existing code: Addition of one more entry in the Table of Pointers to the UCB tables for DT-11 controller. This change does not affect the existing code even if DEC upgrades or introduces support for more controllers as this entry will always be the last one and will be indexed by the controller type which is the highest always.
   This change will only be valid for RSX-11M systems and not for RSX-11M-PLUS systems as they donot require these tables.

   Additions:
   1. #1245: ".IF DF D$$T11"
   2. #1247: ".WORD DTUCB"
   3. #1249: ".ENDC ;D$$T11"
   4. from #1274-1284:

```
                .IF DF  D$$T11

        DTUCB::                         ; DT UCB POINTER TABLE
        N=0
                .REPT   D$$T11
                .WORD   DTUC0+N
        N=N+4
                .ENDR
        DTUC0:  .BLKW   D$$T11*2        ; DT UCB/CSR TABLE

                .ENDC   ; DF D$$T11
```

7. Topic: Instructions added and modified to allocate the Clock Block from the system pool. And the Fork Block is made ZTDRV's and not TTDRV's.

   Line numbers: 1661, 1673 and 1877-1886

   Changes to existing code:
   1. #1661: "MOV #$ZTFB+10,R1 ;GET POINTER TO KINAR6 WORD IN"
   2. #1673: "MOV R0,@#$ZTFB+10 ..." commented out.
   3. #1877-1886:

```
                "MOV    #$DEVHD,R0
                35$:
                 MOV    @R0,R0
                 BEQ    50$
                 CMP    D.NAM(R0),#"ZT
                 BNE    35$
```

```
                        CALL    $ALCLK
                        MOV     R0,$ZTCB
                        MOV     #TTICK,C.SUB(R0)"
```

Additions:

1. #1943: "50$:"
   A label where control comes when the ZT data base is not found.
3. #1944: "RETURN"
   When control comes to 50$ it just returns and no further action is taken.

6.3 ZTTBL.MAC: .IDENT /V4.00/

This file contains the driver dispatch table and some routines which are called when the driver is either loaded or put online etc.

1. Topic: Naming conventions. The start of the dispatch table should start with the device's nmemonic 'ZT'.

Line numbers: 61 and 141.

Changes in the existing code: label names changed.

1. #61: instead of "$TTTBL::" it is now "$ZTTBL::"
2. #141 instead of "$TTTBE::" it is now "$ZTTBE::"

Additions: none.

2. Topic: Addition of the interrupt entry points in the dispatch table.

Line numbers: 135-139 After "Y'X'CTBP:: ...."

Changes in existing code: just added the interrupt entry points for the new controller and in the end so it will not affect the software if new controllers are added.

Additions:

1. from #135-139 the following is added:

```
                .ASCII  /ZT/
                .WORD   $ZTINP
                .WORD   $ZTOUT
                .WORD   0
        ZTCTBP::.WORD   0
```

6.4 ZTTAB.MAC: .IDENT /02/

This file contains the data base for the ZTDRV and is coded in such a way that it will automatically asswemble for RSX-11M or RSX-11M-PLUS systems and generate the required database for that particular system.

This section describes the type of data base selected for the pseudo terminal driver and gives the appropriate reasons and also describes the fields of the data base and their static initialized values.

1. CTB (for RSX-11M-PLUS only):

One CTB describes the type of the controller used - the DT-11 - whose

name is 'ZT' (same as the device name).
It's different fields are:

L.ICB --> interrupt control block - nonexistant.
L.LNK --> link to next is 0 as only one controller.
L.NAM --> .ASCII /ZT/
L.DCB --> pointer to the DCB
L.NUM --> number of controllers = 8
L.STS --> status = 0
L.KRB --> table of all the 8 KRB address.

2. DCB (for both M and M+):

One DCB exists to describe the type of the device attached to the controller. The fields are as follows:

D.LNK --> link field is 0 as driver only supports one device type.
D.UCB --> pointer to the first UCB.
D.NAM --> .ASCII /ZT/
D.UNIT --> lowest and highest unit nos.
D.UCBL --> length og the UCB's
D.DSP --> pointer to the driver dispatch table now null but later initialized by the LOA task.
D.MSK --> function masks - has all the function codes supported by the TTDRV plus two function codes IO.INP and IO.OUT whose mask bits are 13 and 14 respectively.
D.PCB --> PCB address of the partition in which the driver will be loaded - filled by the LOA task.

3. UCB (for both M and M+):

One UCB exists for each unit attached to each controller. Here we have one unit per controller. The fields are initialized as follows:

U.UAB --> (for M+ only) User account block address - not used.
U.MUP/U.CLI --> mutliuser protection/CLI address used by the main driver code.
U.LUIC --> login uic - initialized to zero - used by the main code.
U.OWN --> owning terminal's UCB address if device alocated. initialized to zero here.
U.DCB --> back pointer to the DCB.
U.RED --> redirect UCB address - here redirected to itself.
U.CTL --> control flags:
UC.ATT!UC.PWF!UC.KIL!UC.QUE
Control comes to the driver whenever there is a request for attaching the terminal(UC.ATT), on powerfailure (UC.PWF), for an IO KILL requests(UC.KIL), and during a normal request the packet is not queued to the driver's internal queue as task context is required to relocate user specified buffers(UC.QUE).
U.STS --> US.OIU - initialized as output interrupt unexpected.
U.UNIT --> Physical unit no. i.e. the number of the unit w.r.t. the ones connected to one controller - here it is 0.
U.ST2 --> 0 for M and US.OFL for M+. For M+ unit is initialized as being offline and the CON task makes it online.
U.CW1 --> DV.REC!DV.CCL!DV.TTY
This device is a record oriented device(DV.REC), also it is a carriage control device(DV.CCL) and it is a terminal device(DV.TTY).

U.CW2	-->	U2.LOG!U2.CRT!U2.LWC!U2.RMT The unit is not loggid in(U2.LOG), the unit is a CRT terminal(U2.CRT), it is set to lower case(U2.LWC) and it is a remote terminal(U2.RMT) so that modem facilities can be availed of.
U.CW3	-->	0
U.CW4	-->	80. The default buffer size of the terminal before wrap around takes place.
U.SCB	-->	SCB address
U.ATT	-->	0 - attached task's TCB address - run time parameter.
U.TUX	-->	pointer to the UCB extension - 0 - initialized at initialization time.
U.TSTA	-->	unit status - 0.
U.TSTA+2	-->	S2.ACR!S2.FLF Automatic carriage return and forced line feed.
U.TSTA+4	-->	S3.TAB need for type-ahead buffer.
U.TSTA+6	-->	0
U.UIC	-->	(for M+ only) 0.
U.TLPP	-->	lines per page = 24.
U.TFRQ	-->	0
U.TFLK	-->	0
U.TCHP	-->	0
U.TCVP	-->	0
U.UIC	-->	(for M only) 0.
U.TTYP	-->	terminal type 0 - unknown.
U.TMTI	-->	modem timer 0.
U.TTAB	-->	address of the type-ahead buffer - 0 - initialized at the initialization time.
U.CTYP	-->	(for M only) = YTINDX - the controller type.

4. SCB (for RSX-11M only):

There is one SCB for one unit since each unit operate independantly and have different contexts at the same time. This requires separate SCB's to store thier run time contexts. The different fields are:

S.LHD	-->	0 and start of the SCB in the two words resp. This is the I/O queue list head which is so initialized but later used by the system and the driver.
S.PRI	-->	Priority of this device - PR5
S.VCT	-->	interupt vector address by 4. Here 0.
S.ITM	-->	initial time out count - 5.
S.CTM	-->	current time out count - 0.
S.CON	-->	controller index - the number of the controller of the same kind.
S.STS	-->	0
S.CSR	-->	CSR address - 0.
S.PKT	-->	address of the I/O packet of the currently active I/O.
S.FRK	-->	Fork link word - 0.

5. Contiguous KRB/SCB (for M+ only):

The ZTDRV requires a contiguous SCB and KRB because only one unit is supposed to be connected to a controller and in this case context would have to be saved for only one unit at a time which requires only one SCB and one KRB for the controller. In the M+ I/O philosophy, in such a case pool space is saved by avoiding two separate KRB's and SCB's by

making them contiguous and in this case some fields become common to the SCB and the KRB both. The fields are as follows:

K.PRM --> device dependent but here the controller type - YTINDX.
K.PRI --> priority - PR5.
K.VCT --> vector address - 0.
K.CON --> controller index - for unit n it is n * 2.
K.IOC --> I/O count for the controller - 0.
K.STS --> status - KS.OFL - controller is offline, initially, till the CON task makes it online.
K.CSR --> CSR address = 164000 the CSR for the EXOS board. This is initialized even though it isn't required because the CON task probes into the CSR to see if controller is present or not. The EXOS device has to be present if the ZTDRV has to become online - hence the initialization. As an improvemtent this field should be initialized to the label ZECSR which will be defined during task building time of the ZTDRV and its value will depend on the actual CSR of the target system.

K.OFF --> offset to the UCB table - 0.
K.HPU --> 0
K.OWN --> Owning UCB address. Initialized as the corresponding UCB address.
K.CRQ --> Controller request queue listhead 0 and address of the SCB which is .-2
K.FRK/S.FRK --> Fork block - 0's.
S.KS5 --> APR5 of the driver when it calls $FORK
S.PKT --> 0
S.CTM --> 0
S.ITM --> 5 initial time out count.
S.STS --> 0
S.ST3 --> 0
S.ST2 --> S2.CON - indicates that the SCB and KRB are contiguous.
S.KRB --> address of the corresponding KRB.

6.5 ZTMAC.MAC:

This is the assembly prefix file for the ZTDRV.

1. Topic: Initialization of some constants used during the assembly time.

Line numbers: 40-45 After The ".MCALL UCBDF$...."

Changes from the existing code/data structures: none.

Additions:

1. #40 : "D$$T11 = 10"
   The controller DT-11 is recognized throughout the ZTDRV by this symbol and its value indicates the number of such controllers existant.
2. #41 : "IO.INP = 5400"
   The input interrupt I/O function code.
3. #43 : "IO.OUT = 6000"
   The output interrupt I/O function code.
4. #44 : "IO.TEL = 177000"
   The pseudo function code for telnet requests to the board from the

ZTDRV. ('pseudo' because it is not within the allowed 32 legal function codes but it's purpose is not for the system but local to the communication between the ZTDRV and the ACP. Since the system is not comming into the picture (DRQIO) it can be initialized as it is.

5. #45 : "TS.HNG = 176000"
The pseudo function code for the Hangup request to the board. Since this request is to be handled differently by the ACP (different from the normal output data TSWRITE requests), it is made into a separate pseudo function code.

2. Topic: Modem support

Line number: 82 After ".IIF DF P$$GEN,..."

Changes from the existing code:

1. #82 : ".IIF NDF D$$LMD D$$LMD = 0"
D$$LMD, which indicates the modem support for the DLV11-E controller is forcefully defined to include the modem support routines in the ZTDRV code at assembly time.
It is suggested that instead of forcefully defining D$$LMD, to inturn define T$$MOD, T$$MOD should be defined forcefully as follows:
after the line where T$$MOD might get defined in current line number 84,
".IIF NDF T$$MOD T$$MOD = 0"

7. IMPROVEMENTS AND ENHANCEMENTS:

The areas under which some improvement can be made in the ZTDRV are:

7.1 CALLING THE ZTDRV DIRECTLY AND NOT VIA QIO'S:

Some code changes could be made to somehow get the control into the input and output interrupt entry points directly and not via QIO's from the ACP. A lot of investigation into the interrupt handling of the executive would be required for the purpose. If a method to do so is found then it will speed up the telnet driver manifold and also reduce the size of the ACP.

The best way to do this would be to find the input/output interrupt entry point addresses and then load the APR 5 with the APR 5 value stored in the PCB for the ZTDRV and then call those routines directly. This calling cannot be done inside the ACP or the ZEDRV since they are mapped by the APR 5. It will have to be done from inside the executive by first calling a routine in the executive which does this dispatching to the input/output interrupt entry points. Hence, the problem is to smuggle in a routine into the executive!! How to do this???

7.2 SOME DEBUGGING:

The commented instructions in the routine INIT:: in the file ZTDAT.MAC cause problems while loading the driver. One has something to do with the fork block and the other with the clock block (refer to the maintainance guide). After commenting them there weren't any problems faced so investigation is required as to why the problems were caused. The problems of the clock block and the fork block have been solved but the one for the UCB queue is still not. The ZTDRV uses the TTDRV's UCB queue and some method must be applied to allocate the UCB queue for ZTDRV from the system pool and deallocate it when the driver is unloaded. For the clock block, which is allocated from the system pool when

the driver is loaded (for RSX-11M systems) or when it's first controller is put online (for RSX-11M-PLUS systems), it is never deallocated for the RSX-11M systems because control never comes to the driver while it is being unloaded. But for the RSX-11M-PLUS systems it is deallocated when the driver receives control while putting the controller offline. This means that the ZTDRV for the RSX-11M systems can never be unloaded (only if the system is re-booted) but for the RSX-11M-PLUS systems it can be unloaded.

7.3 LOADING THE DRIVER TWICE FOR THE RSX-11M SYSTEMS:

This problem is faced because the driver is called at the power fail entry point while loading it before the data base is made resident into the system pool. The INIT routine checks in the device tables if ZT is present or not. Since it does not find ZT data base during the virgin initialization of the driver, the local pool is not allocated and initialized and due to this all the system pool is eaten up. If the driver is loaded once and then again, the second time around it does find the data base and initializes the local pool. This double loading and unloading could be avoided by just loading the driver once and then as soon as the first QIO comes, it would also initialize the local pool, if it wasn't already done so. This could mean a lot of changes in the ZTINI.MAC file and hence the maintainance would become more difficult. Hence, the present scheme is good enough unless there is some way out in the initialization time only (???).

For RSX-11M-PLUS systems this is not a problem because when INIT:: is called the data base is already resident as it is called while making the controller online AFTER the driver is fully loaded.

During the starting time for the network software the ZTDRV can be loaded twice and unloaded once to initialize the local pool. The first time it is unloaded the data base is not yet put in the system pool so the clock block is also not allocated at that time. It is only allocated when the data base is found in the system pool.

7.4 ABOUT THE UCB QUEUE:

The $TTUQ data structure, as defined in the file SYSTB.MAC, is for the purpose of the TTDRV. Since the code for ZTDRV is extracted from the TTDRV code, this data structure has remained in the ZTDRV code. The UCB queue didn't cause any problems even though it is meant for the TTDRV since it is a link list of the UCB addresses and this link list was being shared by both the TTDRV and the ZTDRV which turns out to be O.K. This is definitely not advisable and the ZTDRV's own data structure - $ZTUQ should be defined in the ZTDAT.MAC file exactly as the ones for the TTDRV are done. But the problem is that since the systems might just be refering to this data structure, it is advisable to allocate it from the system pool and store the address of the UCB queue in a data structure called $ZTUQ and deallocate this back to the system pool when the driver is unloaded (for RSX-11M-PLUS systems).

7.6 IF TERMINAL IS ATTACHED BY A TASK THEN telnet>q FAILS TO LOGOUT USER:

This problem most generally occurs with terminals running EDT and then typing the escape character and quiting. This causes the ACP to call the routine "bye()" and this sends a ^C followed by a "BYE\r" as an unsolicited input to the ZTDRV. If EDT is running then it traps this ^C and also the BYE command line. The terminal remains logged in and EDT keeps running.

The solution to this problem is that instead of giving a ^C "BYE\r" as an unsolicited input to the ZTRDV, a QIOW #IO.HNG should be given to that unit

so that control will come to the modem timeout entry point of the controller dependent routines YTMTIM. Here the routine MHUP should be called which queues a BYE to the MCR for that terminal and this would cause the terminal to be logged out. If the user has privilege then the task might be even aborted by the BYE task.

7.7 CHANGES TO BE MADE IF DEC ADDS NEW QIO CALLS:

If DEC happens to increase the number of QIO calls to the TTDRV then it will affect our design if we were to upgrade the ZTDRV software. The following changes will have to be made to live up to this change:

1. The value of the function codes has to be just above the last highest function code supported by DEC but the overall numbers of function codes should not exceed 32. This change will have to be made in the file ZTMAC.MAC.
2. The entries of the input and output interrupt entry points in the dispatch table for the function codes service routines has to be the last ones i.e. the QPINP and QPOUT should always be the last entries in the dispatch table QPDSP. Changes will confine to the file ZTDAT.MAC.
3. The definitions of the function codes IO_INP and IO_OUT will have to be changed in the file exqio.h.

THE 'ZE' DRIVER

or

THE EXOS DRIVER

```
.NLIST  CND
.NLIST  SYM
;
; filename:     ZEDRV.MAC
;
;       ZEDRV:  Driver code of the EXCELAN ethernet controller for
;               RSX-11M on a Q-BUS/UNIBUS system.
;
        .TITLE  ZEDRV
        .IDENT  /01/

        .ENABL  LC

        .MCALL  HWDDF$,UCBDF$,DCBDF$,SCBDF$,TCBDF$,PKTDF$
        HWDDF$
        UCBDF$
        DCBDF$
        SCBDF$
        TCBDF$
        PKTDF$

        .PSECT  ABC

        ZESTART = .
;
; LOCAL DATA
;
;       UCBR5 is a local storage to remember UCB address
;
UCBR5:  .BLKW   1
UCBCAN: .BLKW   1
TCBCAN: .BLKW   1

        .IF DF  R$$MPL

        .IFF    ;R$$MPL

CNTBL:  .WORD   0

        .IFTF   ;R$$MPL

;
; LD$ZE  -->    Driver is loadable
; Z$$E11 -->    No controller
;

LD$ZE  = 0
Z$$E11 = 1
;
;       Driver dispatch table
;

        .IFT    ;R$$MPL

        DDT$    ZE,Z$$E11,NEW=Y         ; generate dispatch table
```

```
        .IFF    ;R$$MPL

$ZETBL::
        .WORD   ZEINI                   ; initiator entry point
        .WORD   ZECAN                   ; cancel entry point
        .WORD   ZEOUT                   ; time-out entry point
        .WORD   ZEPWF                   ; power fail entry point

        .ENDC   ;R$$MPL

;
; This section contains all the I/O functions and their corresponding I/O
; codes with their value, for the ZE ethernet controller device
;

IO.EXC = 002400                 ; EXOS device administratve operation
   EX.INI = 0000                ; Reset and configure EXOS
   EX.STR = 0001                ; Execute EXOS process
   EX.STS = 0005                ; Read board's statistics
   EX.RST = 0006                ; Read and reset board's statistics
   EX.CNF = 0007                ; get configuration message
   EX.OPN = 0020                ; Open an administrative channel
   EX.CLS = 0021                ; Close administrative channel
   EX.POS = 0022                ; seek into EXOS's memory
   EX.SAR = 0024                ; set up an ARP table entry
   EX.GAR = 0025                ; get an ARP table entry
   EX.DAR = 0026                ; delete an ARP table entry
   EX.ART = 0027                ; add an Routing table entry
   EX.DRT = 0030                ; delete an Routing table entry
   EX.SRT = 0031                ; fetch an Routing table entry
   EX.NRT = 0032                ; fetch next Routing table entry

IO.ACS = 003000                 ; Socket related operations
   SA.OPN = 0062                ; Open a socket for communication
   SA.ACC = 0063                ; Accept connection on a remote socket
   SA.CON = 0064                ; Connect to a remote socket
   SA.SAD = 0067                ; get socket information
   SA.CLS = 0070                ; close a socket connection
   SA.SEL = 0073                ; check possibility of I/O on socket
   SA.USL = 0210                ; kill a pending select call
   SA.URG = 0200                ; prepare for an urgent message
   SA.ROO = 0220                ; remove oob pkt from the pending list

IO.XFR = 003400                 ; data transfer operations on a socket
   IX.RDS = 0000                ; read from TCP stream
   IX.WRS = 0001                ; write to TCP stream
   IX.SND = 0065                ; send datagram to socket
   IX.RCV = 0066                ; receive socket datagram

IO.SOC = 004000                 ; real socket control operations
   SO.DON = 0000                ; shutdown read/write operation
   SO.SKP = 0001                ; set keep alive
   SO.GKP = 0002                ; inspect keep alive
   SO.SLG = 0003                ; set linger time
   SO.GLG = 0004                ; get linger time
   SO.SOB = 0005                ; send out of band
```

```
   SO.ROB = 0006                ; receive out of band
   SO.AMK = 0007                ; at out of band mark?
   SO.SPG = 0010                ; set process group id
   SO.GPG = 0011                ; get process group id
   SO.NRD = 0157                ; FIONREAD
   SO.NBO = 0156                ; FIONBIO
   SO.ASY = 0155                ; FIOASYNC

IO.LOG = 004400                 ; read error log from EXOS



SOICTL = 56                     ; size of SOictl structure
CH.WRITE = 1                    ; open channel in WRITE mode

;
; ZEINI -->     EXOS driver initiator entry point.
;               All functions are made control functions. As the UC.QUE bit is
;               set, the QIO directive will pass the I/O packet, instead of
;               queueing it, to the driver so that the user's context is not
;               lost. The driver, on receiving a packet, does some address
;               checking depending on the function, and relocates it. It
;               also rearranges the driver dependent parametersin the I/O
;               packet. Last three parameters (I.PRM+6  I.PRM+12) are shifted to
;               to (I.PRM+12 to I.PRM+20).
;               After rearranging and relocating the parameters, the driver
;               inserts the packet into the ACP's queue and wakes it up. Hence,
;               the actual queue buils up at the ACP.
;
;
; INPUTS:       When the QIO directive passes the packet to the driver, it
;               passes the following:
;
;               R1 --> Address of the I/O packet.
;               R4 --> Address of the status control block.
;               R5 --> Address of the UCB of the device unit.
;

        .PSECT  ABC

ZEINI:

; folowing four statements are coded temporarily to keep the
; address of any UCB stored in the local variable UCBR5 so
; that on entry at the interrupt entry point the TCB address
; of the ACP can be found;
;

        TST     UCBR5           ; test whether it is already initialised
        BNE     10$             ; already initialised
        MOV     R5,UCBR5        ; move UCB address in UCBR5

10$:


;
```

```
; shift parameter 4, 5 & 6 by two words in I/O packet
;
        MOV     R1,R3                   ; move I/O pkt address in R3
        ADD     #I.PRM+12,R3            ; make R3 points to param 6
        MOV     #3,R4                   ; loop 3 times
60$:    MOV     (R3),4(R3)              ; shift by two words
        TST     -(R3)                   ; decrement R3 by two
        SOB     R4,60$                  ; loop

;
; check the following function codes whether they have the Soioctl structure
; address specified or not. If not then abort that request because that
; parameter is essential for these requests to succeed.
;
        CMP     I.FCN(R1),#IO.ACS!SA.ACC ; is it socket accept request?
        BEQ     70$                     ; if EQ yes
        CMP     I.FCN(R1),#IO.ACS!SA.CON ; is it socket connect request?
        BEQ     70$                     ; if EQ yes
        CMP     I.FCN(R1),#IO.ACS!SA.OPN ; is it socket open request?
        BEQ     70$                     ; if EQ yes
        CMP     I.FCN(R1),#IO.ACS!SA.SAD ; is it obtain socket address request?
        BEQ     70$                     ; if EQ yes
        CMPB    I.FCN+1(R1),#IO.SOC/400  ; is it socket control request?
        BEQ     70$                     ; if EQ yes
        CMP     I.FCN(R1),#IO.XFR!IX.RCV ; is it a receive message request?
        BEQ     70$                     ; if EQ yes
        CMP     I.FCN(R1),#IO.XFR!IX.SND ; is it a sebd message request?
        BNE     100$                    ; if NE no, so process other requests
70$:
        TST     I.PRM+4(R1)             ; is Soioctl structure address there?
        BNE     80$                     ; if NE then yes so address check and
                                        ; relocate it.
        MOV     #IE.SPC&377,R0          ; set illegal or no buffer error
        MOV     R1,R3                   ; retrieve iopkt address
        JMP     500$                    ; abort request

;
; address check and relocate parameter #3, if any, which contains the
; address to the socket related parameters buffer
;

80$:
        MOV     (R3),R0                 ; move soictl buf address in R0
        MOV     R1,R3                   ; save I/O packet address in R3

        .IF DF A$$CHK!M$$MGE

        MOV     #SOICTL,R1              ; get size of SOICTL buffer
        CALL    $ACHKB                  ; address check buffer byte algn
        BCC     90$                     ; if CC ok
        MOV     #IE.SPC&377,R0          ; set illegal buffer error
        JMP     500$                    ; abort request

        .ENDC
```

```
90$:    CALL    $RELOC                  ; relocate SOICTL buffer
        MOV     R1,I.PRM+6(R3)          ; move relocation bias
        MOV     R2,I.PRM+10(R3)         ; move displacement bias
        MOV     R3,R1                   ; restore I/O packet address in R1

;
; address check and relocate user buffer if neccessary
;

100$:
        CMPB    I.FCN+1(R1),#IO.XFR/400 ; is it a data transfer request
        BEQ     120$                    ; if EQ yes
        CMPB    I.FCN+1(R1),#IO.ACS/400 ; is it socket access request
        BEQ     160$                    ; EQ yes
        CMPB    I.FCN+1(R1),#IO.SOC/400 ; is it socket control fn
        BEQ     160$                    ; if EQ yes
        CMPB    I.FCN+1(R1),#IO.WLB/400 ; is it EXOS memory write fn
        BEQ     120$                    ; if EQ yes
        CMPB    I.FCN+1(R1),#IO.RLB/400 ; is it EXOS memory read fn
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.CNF; is it read config msg fn
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.STS; is it read EXOS stat. fn
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.RST; is it read & reset EXOS stat fn
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.SAR; is it set ARP function
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.GAR; is it get ARP function
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.DAR; is it delete ARP function
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.ART; is it add an RT entry fn
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.DRT; is it delete an RT entry fn
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.SRT; is it fetch an RT entry fn
        BEQ     120$                    ; if EQ yes
        CMP     I.FCN(R1),#IO.EXC!EX.NRT; is it fetch next RT entry fn
        BEQ     120$                    ; if EQ yes
        CMPB    I.FCN+1(R1),#IO.LOG/400 ; is it read error log fn
        BNE     160$                    ; if NE no, then fn have no buf

120$:   MOV     I.PRM(R1),R0            ; move user buf addr in R0
        MOV     R1,R3                   ; save I/O packet address

        .IF DF A$$CHK!M$$MGE

        MOV     I.PRM+2(R1),R1          ; get length of buffer
        CALL    $ACHKB                  ; address check buffer byte algn
        BCC     140$                    ; if CC ok
        MOV     #IE.SPC&377,R0          ; set illegal buffer code
        JMP     500$                    ; and abort request

        .ENDC
```

```
140$:   CALL    $RELOC
        MOV     I.PRM+2(R3),I.PRM+4(R3) ; shift byte count by a word
        MOV     R1,I.PRM(R3)            ; move relocation bias
        MOV     R2,I.PRM+2(R3)          ; move displacement bias
        MOV     R3,R1                   ; restore address of I/O packet

;
; now queue the iopacket to acp and unstop it
;

160$:   MOV     U.ACP(R5),R0            ; get TCB address of ACP task
        BNE     200$                    ; if NE acp task is active
        MOV     #IE.DNR&377,R0          ; else acp not active
        MOV     R1,R3                   ; move I/O pkt address in R3
        JMP     500$                    ; abort request

200$:   JMP     $EXRQP                  ; que I/O pkt to acp and wake it

        RETURN


500$:
        CLR     I.PRM+16(R3)            ; clear the diagnostic field
        JMP     $IOFIN                  ; finish I/O operation and inform


;
; ZECAN: The cancel I/O entry point. The driver is called at this entry
;        point by the executive with the following parametrs
;
;        R5 -> UCB address
;        R4 -> SCB address
;        R3 -> Controller index
;        R1 -> Address of TCB of current task
;        R0 -> Address of active ( if any ) I/O packet
;
;        Out of all these parameters we are only interested in the TCB
;        address. In our case the I/O packet address will be zero as
;        we do not remember anything in the SCB
;
;        At this point we will create an I/O packet and fill up its
;        function code , TCB and UCB fields and then queue the packet
;        to ACP, which will do the rest of the work.
;


ZECAN:                                  ; CANCEL IO ENTRY POINT
        MOV     R5,UCBCAN               ; save UCB address
        MOV     R1,TCBCAN               ; save TCB address of current task
        MOV     #I.LGTH,R1              ; Allocate an I/O packet
        CALL    $ALOCB                  ;
        MOV     #IO.KIL,I.FCN(R0)       ; move function code
        MOV     TCBCAN,R5               ; get TCB address of current task
        MOV     R5,I.TCB(R0)            ; set TCB address
        MOV     UCBCAN,R5               ; get UCB address
        MOV     R5,I.UCB(R0)            ; move UCB address to packet
```

```
        MOV     R0,R1                   ; set R1 with packet address
        MOV     U.ACP(R5),R0            ; set R0 with ACP address
        JMP     $EXRQP                  ; Q pkt & wakeup ACP

;
; $ZELOA/ZEPWF
;
;       The loadable driver/power fail entry point is entered upon by the
;       Executive. The 22-bit physical start address of the local pool is
;       calulated and stored in the UCB. This setup is only required for
;       software running on the UNIBUS machines.
;

        .IF DF  R$$MPL

ZEPWF:
$ZEUNL::
        RETURN

$ZELOA::

        .IFF    ;R$$MPL

ZEPWF:

        .ENDC   ;R$$MPL

        .IF DF  UNIBUS

        NOP                             ; breakpoint for XDT

        MOV     @#KISAR5,R0             ; get start of driver code

        OFFSET = LOCPOOL - ZESTART

        MOV     R0,R1                   ; copy start of driver code
        ASH     #-12,R0                 ; get lower 6-bits of hi-order addr
        BIC     #177700,R0              ; mask out remaining high bits
        ASH     #6,R1                   ; get upper 10 bits of lo-order address
        BIC     #000077,R1              ; mask out remaining bits
        ADD     #OFFSET,R1              ; get the start of driver's local pool

        MOV     R0,U.ACP+2(R5)          ; save hi-order physical address and
        MOV     R1,U.ACP+4(R5)          ; lo-order physical address in UCB

        RETURN

        .ENDC   ; UNIBUS

ZEOUT:                                  ; time-out entry point

        .IF DF  R$$MPL

ZEKRB:                                  ; controller on/off line entry point
ZEUCB:                                  ; unit on/off line entry point
```

```
        .ENDC

        RETURN

;
; ZEINT:        ZE device driver entry point.
;
;               This is a very uncommon way to to handle device interrupts. As
;               the EXOS device processes all the requests in a pure
;               asyncchronous way, it is very handy to process the interrupt
;               service in the ACP which actually has all the necessary
;               information. Hence, the driver's job is to deflect the interrupt
;               to the ACP by just unstopping it if it is sleeping.
;

sema::  .word   177776                  ; initial value of semaphore

$ZEINT::                                ; interrupt entry point of ZE device
        INTSV$  ZE,PR4,Z$$E11           ; generate interrupt save code
        sec                             ;;; set carry bit
        ror     sema                    ;;; shift to set semaphore
        bcc     10$                     ;;; semaphore OK
        return                          ;;; return and dismiss interrupt
10$:
        MOV     UCBR5,R5                ; get address of UCB befor calling FORK
        CALL    $FORK                   ; create system process

        MOV     UCBR5,R5                ; unsave UCB address into R5
        MOV     U.ACP(R5),R0            ; move address of the TCB of the ACP
        call    $EXRQU                  ; request ACP execution after inserting
                                        ; the I/O packet
        mov     #177776,sema            ; release semaphore
        RETURN


        ZESIZE = . - ZESTART            ; size of zedrv code area

        .IF DF  UNIBUS

LEAVE:  .BLKW   1024. - ZESIZE          ; leave total of 1kw before start of
                                        ; driver's local pool

LOCPOOL::                               ; start of local buffer pool
        .BLKW   1

        .ENDC

        .END
```

```
        .NLIST  CND
        .NLIST  SYM

;
; filename:     ZETAB.MAC
;
;       ZETAB:  The database of the ZE driver is defined as follows.
;
;
        .TITLE  ZETAB
        .IDENT  /01/

;       System Macro Calls

        .MCALL  UCBDF$,HWDDF$,SCBDF$,UCBDF$
        UCBDF$
        HWDDF$
        SCBDF$
        UCBDF$

        .PSECT  $$$

        .GLOBL  $ZEVEC
        .ENABL  LC

$ZEDAT::                                ; start of the ZEDRV device table

        .IF DF  R$$MPL

                ;--------,
                ; ZE CTB ;
                ;--------'

        .WORD   0                       ; L.ICB
$CTB0:
        .WORD   $CTB1                   ; L.LNK end of CTB list for ZE
        .ASCII  /ZE/                    ; L.NAM controller's name
        .WORD   .ZC0                    ; L.DCB
        .BYTE   1                       ; L.NUM no. of controllers
        .BYTE   0                       ; L.STS status
$ZECTB::
        .WORD   $ZEA                    ; L.KRB

                ;------------,
                ; END OF CTB ;
                ;------------'

        .IFTF   ;R$$MPL

$ZETBL = 0                              ; loadable ZEDRV
$ZEDCB::
.ZC0:
        .WORD   .ZC1                    ; D.LNK link to next DCB
        .WORD   .ZE0                    ; D.UCB pointer to first UCB
        .ASCII  /ZE/                    ; D.NAM device name
        .BYTE   0,0                     ; D.UNIT lowest and highast unit number
```

```
        .WORD   ZEND-ZEST               ; D.UCBL lenght of UCB
        .WORD   $ZETBL                  ; D.DSP pointer to device dispatch table
;
; The following tables define all the legal functions and their subdivisions
; in terms of NO-OP's, ACP, CONTROL, TRANSFER functions. Apart from IO.KIL,
; IO.ATT and IO.DET, all other functions are made control functions. With
; the UC.QUE bit in the U.CTL of the UCB set, the QIO directive will pass the
; I/O packet to the driver without queueing, such that user's context is saved.
; The IO_ATT & IO_DET functions are made NO-OPS.
;
        .WORD   001777                  ; D.MSK legal functions  0 - 15.
        .WORD   001747                  ;       control functions 0 - 15.
        .WORD   000030                  ;       NO-OP functions  0 - 15.
        .WORD   000000                  ;       ACP functions    0 - 15.
        .WORD   000000                  ;       legal functions       16. - 31.
        .WORD   000000                  ;       control functions     16. - 31.
        .WORD   000000                  ;       NO-OP functions       16. - 31.
        .WORD   000000                  ;       ACP functions         16. - 31.
        .WORD   0                       ; D.PCB PCB address of driver partition

ZEST = .                                ; start of UCB

        .WORD   0                       ; U.OWN owning terminal's UCB address
.ZE0::
        .WORD   .ZC0                    ; U.DCB back pointer to DCB
        .WORD   . - 2                   ; U.RED redirect pointer

        .IF DF  UNIBUS

        .BYTE   UC.KIL!UC.QUE!UC.PWF!UC.NPR ; device is an NPR device
                                        ; control flag byte, call on IO.KILL
                                        ; and pass packet to driver
        .IFF    ;UNIBUS

        .BYTE   UC.KIL!UC.QUE!UC.PWF    ; control flag byte, call on IO.KILL
                                        ; and pass packet to driver
        .ENDC   ;UNIBUS

        .BYTE   0                       ; U.STS status flag U.STS
        .BYTE   0                       ; U.UNIT --  does not apply

        .IF DF  R$$MPL

        .BYTE   US.RED!US.PUB!US.OFL    ; U.ST2 2nd status flag - unit cannot be

        .IFF    ;R$$MPL

        .BYTE   US.RED!US.PUB

        .ENDC   ;R$$MPL
                                        ; redirected
        .WORD   DV.EXT                  ; U.CW1 characteristic word 1 --> device
                                        ; is connected to 22-bit direct
                                        ; addressing controller
        .WORD   0                       ; U.CW2 char word 2
```

```
        .WORD   0                       ; U.CW3 char word 3
        .WORD   0                       ; U.CW4 char word 4, no buffer required
        .WORD   $ZE0                    ; U.SCB pointer to SCB
        .WORD   0                       ; U.ATT attached task UCB
        .BLKW   3                       ; U.BUF, U.BUF+2 & U.CNT

        .IFT    ;R$$MPL

        .WORD   0                       ; U.UCBX UCB extension

        .IFTF   ;R$$MPL

        .WORD   0                       ; U.ACP TCB address of ZEACP

        .IF DF  UNIBUS

        .BLKW   2                       ; storage for the staring 22-bit
                                        ; physical address of the local pool
        .ENDC

ZEND=.                                  ; end of UCB

                ;------------,
                ; END OF UCB ;
                ;------------'

        .IFT    ;R$$MPL

                ;----------------------------,
                ; ZE KRB AND SCB - CONTIGUOUS ;
                ;----------------------------'

        .BYTE   PR4                     ; K.PRI device priority
        .BYTE   $ZEVEC/4                ; K.VCT interrupt vector by 4
        .BYTE   0 * 2                   ; K.CON controller number times 2
        .BYTE   0                       ; K.IOC I/O count
        .WORD   KS.OFL                  ; K.STS controller specific status
$ZEA::                                  ; start address of KRB
        .WORD   164000                  ; K.CSR CSR address (default)
        .WORD   ZEA - $ZEA              ; K.OFF offset to start of UCB table
        .BYTE   0,0                     ; K.HPU highest physical unit number
        .WORD   0                       ; K.OWN UCB of currently active unit

                ;---------------------------,
                ; CONTIGUOUS SCB HERE FOR ZE ;
                ;---------------------------'
$ZE0:
        .WORD   0,.-2                   ; S.LHD & K.CRQ
        .WORD   0,0,0,0                 ; S.FRK fork block
        .WORD   0                       ; S.KS5 - KISAR5 saved here
        .WORD   0                       ; S.PKT address of I/O packet
        .BYTE   0,0                     ; S.CTM, S.ITM crnt & init. timeout cnts
        .BYTE   0,0                     ; S.STS, S.ST3 status bytes
        .WORD   S2.CON!S2.LOG           ; S.ST2
        .WORD   $ZEA                    ; S.KRB currently assigned KRB(the only)
        .WORD   2                       ; S.RCNT no. of words in I/O page
```

```
        .WORD   0                       ; S.ROFF offset from S.CSR to start of
                                        ;       device registers
        .WORD   0                       ; S.EMB for error logging
        .BLKW   6                       ; MAPPING ASSIGNMENT BLOCK
        .WORD   0                       ; KE.RHB start of UNIBUS mapping
                                        ; register work area
ZEA::                                   ; start of UCB table (non-existant)

                ;----------------,
                ; END OF KRB/SCB ;
                ;----------------'

        .IFF    ;R$$MPL

$ZE0:
        .WORD   0,.-2                   ; device I/O queue listhead
        .BYTE   PR4,$ZEVEC/4            ; device priority and vector
        .BYTE   0,0                     ; current and initial timeout
        .BYTE   0,0                     ; controller index and device status
        .WORD   0                       ; CSR address
        .WORD   0                       ; address of I/O packet
        .BLKW   5                       ; FORK BLOCK

        .IF DF  UNIBUS

        .WORD   0                       ; S.MPR not used here but $IODONE
                                        ; checks it so we keep it 0
;
; MAPPING ASSIGNMENT BLOCK (FOR UNIBUS MAPPING REGISTER ASSIGNMENT)
;
        .BLKW   1                       ; M.LNK - link word
        .BLKW   1                       ; M.UMRA - address of 1st ass. UMR
        .BLKW   1                       ; M.UMRN - no. of UMR's * 4
        .BLKW   1                       ; M.UMVL - lo 16-bits mapped by 1st UMR
        .BLKB   1                       ; M.UMVH - hi 2-bits mapped by 1st UMR
        .BLKB   1                       ; M.BFVH - hi 6-bits of phy buffer addr
        .BLKW   1                       ; M.BFVL - lo 16-bits of phy buf addr

        .ENDC   ;UNIBUS

        .ENDC   ;R$$MPL

$ZEEND::                                ; end of ZE data base

.ZC1 = 0                                ; end of DCB list for ZE:
$CTB1 = 0                               ; end of CTB list for ZE:

        .END                            ; end of file ZETAB.MAC
```

```
PIP LB:[1,54]ZEDRV.STB/PR/OW:RWED/SY:RWED/GR:RWED/WO:R/FO
        .ENABLE DISPLAY
```

```
        .ENABLE QUIET
        .ENABLE SUBSTITUTION
        .DISABLE DISPLAY
        .IFNDF $VRBS .ASK $VRBS Verbose ? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .IFNDF $DEL .ASK $DEL Delete source file from current UFD? [Y/N]
        .IFNDF $NOPRE .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
        .IFDF $VEC .GOTO 5
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .5:
;
; Assemble the driver code
;
MAC ZEDRV=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,SY:'<UIC>'ZEDRV
MAC ZETAB=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,SY:'<UIC>'ZETAB

        .IFF $DEL .GOTO 10
PIP ZEDRV.MAC;*,ZETAB.MAC;*/DE
.10:
;
; Now build the ZE (EXOS) driver.
;
.;
.; Create the task builder input file. Ask for the interrup vector
.; location use default if the installer does not want to change it.
.;
;
; Create the input command file for the linker
;
        .OPEN ZETKB.CMD
        .DATA LB:[1,54]ZEDRV/-HD/-MM,,ZEDRV=
        .DATA ZEDRV,ZETAB
        .DATA LB:[1,54]RSX11M.STB/SS
        .DATA LB:[1,1]EXELIB/LB
        .DATA /
        .DATA STACK=0
        .DATA PAR=DRVPAR:120000:14000
        .DATA GBLDEF=$ZEVEC:'$VEC'
        .CLOSE
;
; Task build driver
;
.IFT $NOPRE PIP LB:[1,54]ZEDRV.TSK;*/DE
.IFT $NOPRE PIP LB:[1,54]ZEDRV.STB;*/DE
TKB @'<UIC>'ZETKB
;
; delete indirect command file
;
PIP '<UIC>'ZETKB.CMD;*/DE
PIP '<UIC>'ZEDRV.OBJ;*/DE
PIP '<UIC>'ZETAB.OBJ;*/DE
;
; set protection for the driver
;
PIP LB:[1,54]ZEDRV.TSK/PR/OW:RWED/SY:RWED/GR:RWED/WO:R/FO
```

```
        .ENABLE QUIET
        .ENABLE SUBSTITUTION
        .DISABLE DISPLAY
        .IFNDF $VRBS .ASK $VRBS Verbose ? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .IFNDF $DEL .ASK $DEL Delete source file from current UFD? [Y/N]
        .IFNDF $NOPRE .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
        .IFDF $VEC .GOTO 5
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .5:
;
; Assemble the driver code
;
MAC ZEDRV=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,SY:'<UIC>'UNIBUS,ZEDRV
MAC ZETAB=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,SY:'<UIC>'UNIBUS,ZETAB

        .IFF $DEL .GOTO 10
PIP ZEDRV.MAC;*,ZETAB.MAC;*/DE
.10:
;
; Now build the ZE (EXOS) driver.
;
.;
.; Create the task builder input file. Ask for the interrup vector
.; location use default if the installer does not want to change it.
.;
;
; Create the input command file for the linker
;
        .OPEN ZETKB.CMD
        .DATA LB:[1,54]ZEDRV/-HD/-MM,,ZEDRV=
        .DATA ZEDRV,ZETAB
        .DATA LB:[1,54]RSX11M.STB/SS
        .DATA LB:[1,1]EXELIB/LB
        .DATA /
        .DATA STACK=0
        .DATA PAR=DRVPAR:120000:14000
        .DATA GBLDEF=$ZEVEC:'$VEC'
        .CLOSE
;
; Task build driver
;
.IFT $NOPRE PIP LB:[1,54]ZEDRV.TSK;*/DE
.IFT $NOPRE PIP LB:[1,54]ZEDRV.STB;*/DE
TKB @'<UIC>'ZETKB
;
; delete indirect command file
;
PIP '<UIC>'ZETKB.CMD;*/DE
PIP '<UIC>'ZEDRV.OBJ;*/DE
PIP '<UIC>'ZETAB.OBJ;*/DE
;
; set protection for the driver
;
PIP LB:[1,54]ZEDRV.TSK/PR/OW:RWED/SY:RWED/GR:RWED/WO:R/FO
```

```
PIP LB:[1,54]ZEDRV.STB/PR/OW:RWED/SY:RWED/GR:RWED/WO:R/FO
        .ENABLE DISPLAY
```

```
zedrv/-hd/-mm,zedrv/-sp,zedrv=
zedrv,zetab,lb:[1,54]rsx11m.stb/ss
lb:[1,1]exelib/lb
/
stack=0
par=drvpar:120000:14000
gbldef=$zevec:400
//
```

```
$ !
$ !     skeleton for bld.com
$ !
$ if "'''p1'" .nes. "?" then goto doit
$ typ sys$input

 command file to build the task image

 required command files:    None

 required logical names:    None

 required parameters:
        p1      - default directory (default - current directory)

 required files:        None

 required symbols:      None

$ exit
$ doit:
$ sv = f$verify(1)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ if "'''p1'" .eqs. "" then $ p1 = "'''f$logical("sys$disk")'''f$directory()'"
$ set def 'p1'
$ show def
$ !
$ !     Put your own commands here
$ !
$ !     Make assignment for QBUS RSX11M
$ !
$ assign __dra0:[qbus11m.] lb:
$ open/write lnkdrv tkb.cmd
$ write lnkdrv "zedrv/-hd/-mm,zedrv/-sp,zedrv="
$ write lnkdrv "zedrv,zetab,lb:[1,54]rsx11m.stb/ss"
$ write lnkdrv "lb:[1,1]exelib/lb"
$ write lnkdrv "/"
$ write lnkdrv "stack=0"
$ write lnkdrv "par=drvpar:120000:14000"
$ write lnkdrv "gbldef=$zevec:400"
$ write lnkdrv "//"
$ close lnkdrv
$ tkb @tkb.cmd
$ delete tkb.cmd;
$ !
$ !     Unibus M
$ deassign lb
$ assign __dra0:[unill m.] lb:
$ open/write lnkdrv tkb.cmd
$ write lnkdrv "zedrvuni/-hd/-mm,zedrvuni/-sp,zedrvuni="
$ write lnkdrv "zedrvuni,zetabuni,lb:[1,54]rsx11m.stb/ss"
$ write lnkdrv "lb:[1,1]exelib/lb"
$ write lnkdrv "/"
$ write lnkdrv "stack=0"
$ write lnkdrv "par=drvpar:120000:14000"
```

```
$ write lnkdrv "gbldef=$zevec:400"
$ write lnkdrv "//"
$ close lnkdrv
$ tkb @tkb.cmd
$ delete tkb.cmd;
$ deassign lb
$!
$!      Unibus MPlus
$!
$ assign __dra0:[unillmp.] lb:
$ open/write lnkdrv tkb.cmd
$ write lnkdrv "zedrvup/-hd/-mm,zedrvup/-sp,zedrvup="
$ write lnkdrv "zedrvup,zetabup,lb:[1,54]rsx11m.stb/ss"
$ write lnkdrv "lb:[1,1]exelib/lb"
$ write lnkdrv "/"
$ write lnkdrv "stack=0"
$ write lnkdrv "par=drvpar:120000:14000"
$ write lnkdrv "gbldef=$zevec:400"
$ write lnkdrv "//"
$ close lnkdrv
$ tkb @tkb.cmd
$ delete tkb.cmd;
$ deassign lb
$ exit 1
$ abnormal_exit:
$ deassign lb
$ exit 2
```

```
$ !
$ !     skeleton for cmplbr.com
$ !
$ if "''pl'" .nes. "?" then goto doit
$ typ sys$input

 command file to compile and link the library

 required command files:    None

 required logical names:    None

 required parameters:
        pl      - default directory (default - current directory)

 required files:
        none

 required symbols:
        none

 Note:
    You need to edit this file to setup the symbols objlib and inclib as the
    file specifications for the the object and include libraries
$ exit
$ doit:
$ sv = f$verify(1)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ !
$ !     now make assignment for RSX11M Q-bus version
$ !
$ assign __dra0:[qbus11m.] lb:
$ if "''pl'" .eqs. "" then $ pl = "''f$logical("sys$disk")''f$directory()'"
$ set def 'pl'
$ show def
$ show logical lb
$ mac zedrv,zedrv/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,2]zedrv
$ mac zetab,zetab/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,2]zetab
$ !
$ !     now for unibus
$ !
$ assign __dra0:[unill m.] lb:
$ show logical lb
$ mac zedrvuni,zedrvuni/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,2]unibus,zedrv
$ mac zetabuni,zetabuni/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,2]unibus,zetab
$ !
$ !     now for unibus, M-Plus
$ !
$ assign __dra0:[unillmp.] lb:
$ show logical lb
$ mac zedrvup,zedrvup/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,2]unibus,zedrv
$ mac zetabup,zetabup/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,2]unibus,zetab
$ exit 1
$ abnormal_exit:
$ exit 2
```

```
$ !
$ !     skeleton for deliver.com
$ !
$ if "''p1'" .nes. "?" then goto doit
$ typ sys$input

 command file to copy the deliver files to manufacturing area
 You should modify this file to copy the deliverables to
    exos$mfg:[target_directory]

 required command files:    None

 required logical names:    None
        exos$mfg         - pseudo disk for deliverables

 required parameters:   Noe

 required files:        None

 required symbols:      None

$ exit
$ doit:
$ sv = f$verify(0)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ show def
$ !
$ !     Put your own commands here
$ !
$ copy/log      zedrv.mac       exos$mfg:[rsx]
$ copy/log      zetab.mac       exos$mfg:[rsx]
$ copy/log      blddrv.cmd      exos$mfg:[rsx]
$ copy/log      install.cmd     exos$mfg:[rsx]
$ copy/log      net.            exos$mfg:[rsx]net.
$ copy/log      hosts.net       exos$mfg:[rsx]
$ copy/log      hostlocal.net   exos$mfg:[rsx]
$ copy/log      tapeins.cmd     exos$mfg:[rsx]
$ copy/log      8030.hlp        exos$mfg:[rsx]
$ copy/log      blduni.cmd      exos$mfg:[rsxunibus]blddrv.cmd
$ copy/log      instuni.cmd     exos$mfg:[rsxunibus]install.cmd
$ copy/log      tapeuni.cmd     exos$mfg:[rsxunibus]tapeins.cmd
$ exit 1
$ abnormal_exit:
$ exit 2
```

THE SOURCE CODE FOR THE ACP

1. The Include *.h files.
2. The Source *.c files.
3. The Assembly routine *.mac files.
4. The Indirect command *.cmd files.

```
/*
 * filename: BRDIOCTL.H
 */

/*
 * This file defines all the equate symbol for the administrative
 * device's ioctl commands. Some of them are passed as it is to the
 * board, hence should not be modified.
 */


#define BRDINIT         (0)             /* Reset EXOS  devive   */
#define BRDSTART        (1)             /* start exos running   */
#define BRDGSTAT        (5)             /* get board statistics */
#define BRDRSSTAT       (6)             /* get/reset board statistics*/
#define BRDGCONF        (7)             /* get configuration msg   */
#define BRDADDR         (10)            /* set exos memory locator */

#define BRDSARP         (20)            /* set an ARP table entry  */
#define BRDGARP         (21)            /* get an ARP table entry  */
#define BRDDARP         (22)            /* delete an ARP tbl entry */

#define BRDADDRT        (23)            /* add routing table entry */
#define BRDDELRT        (24)            /* delete RT entry      */
#define BRDSHOWRT       (25)            /* show   RT entry      */
#define BRDDISPRT       (26)            /* display RT entry     */


/* Data structure used to send board statistics to host */

struct EXbdstats {
        long    xmt;            /* frames transmitted successfully */
        long    excess_coll;    /* xmits aborted due to excess coll */
        long    late_coll;      /* xmits aborted due to late coll */
        long    tdr;            /* time domain reflectometer */
        long    rcv;            /* error free frames received */
        long    align_err;      /* frames rcvd with alignment err */
        long    crc_err;        /* frames rcvd with crc errors */
        long    lost_err;       /* frames lost due to no buffers */

        /* other bits of info about the board */

        short   fw_release;     /* firmware release */
        short   sw_release;     /* software release */
        short   hw_release;     /* hardware release */
};

/*
 * Ioctl structure for manipulation of the ARP codes
 */

struct EXarp_ioctl {
  struct sockaddr  arp_pa;      /* protocol address     */
  struct sockaddr  arp_ha;      /* hardware address     */
  long             arp_flags;   /* flags                */
  };
```

```
#define ATF_COM         2       /* completed entry      */
#define ATF_PERM        4       /* permanant entry      */
#define ATF_PUBL        8       /* respond for another host     */
```

```
#define MAXCHANNEL 40

#define CH_FREE    0
#define CH_EXOS    1
#define CH_SOCKET  2

#define CH_WRITE        0x01
#define CH_PRIV         0x02
#define CH_READ         0x00
#define CH_MCLOSE       0x80

struct channel {                /* channel control block         */
  Uchar ch_type;                /* type of channel free, socket & etc  */
  Uchar ch_flag;                /* protection flags              */
  Ushort ch_tcb;                /* tcb address of the associated task */
  Ushort rundn_cnt;             /* I/O rundown count on this channel */
  union {
    Ushort ch_soid;             /* socket id returned by the board */
    struct exos_paddr ch_addr;  /* memory locator of the Exos board */
    } ch_u;
  };
```

```
/*
 * filename:    DEFINES.H
 */

#define PKT      io_pkt->i_prm

#define ex_hd    mp->nm_u.msg_hd
#define ex_mg    mp->nm_u.msg_msg
#define ex_dl    mp->nm_u.nm_dload
#define ex_str   mp->nm_u.nm_start
#define ex_cmd   mp->nm_u.nm_cmd
#define ex_pkt   mp->nm_u.nm_packet
#define ex_ctl   mp->nm_u.nm_ioctl
#define ex_sel   mp->nm_u.nm_select
#define ex_oob   mp->nm_u.nm_hasoob
#define ex_tel   mp->nm_u.nm_telnet

/*
 * following are some functions defined as macros
 */

#define sametask(chn)   ((ch_des[chn].ch_tcb==io_pkt->i_tcb) ? 1 : 0)
#define inrange(chn)    (((chn > 0) && (chn < MAXCHANNEL)) ? 1 : 0)
#define writeprv(x) ((ch_des[x].ch_flag&(CH_PRIV|CH_WRITE))==(CH_PRIV|CH_WRITE))
#define ch_mfor_close(chn) ((ch_des[chn].ch_flag & CH_MCLOSE) ? 1 : 0)

/* dalpkt is defined to be dealoc_b after RTH merger */

#define dalpkt(p)       dealoc_b(p, sizeof( struct iopkt))

/* following is just a dummy structure to be replaced by the actual one */

struct rtentry{
      char rt[40];
   };

#define NOREPLY         0x1
#define UNSELECT        0x2

/*
 * the following definitions are included from the actual soioctl.h file
 *  used by the board code and other systems. As the SOIOCTL definitions
 *  formed by these macros cannot be passed as io subfunction codes, the
 *  final code for the board is made in the acp using these macros.
 */

#define _IOXFIO(y)      (('f' << 8) | y)
#define _IOXSIO(y)      (('s' << 8) | y)
```

```
/*
 * filename: EXIOCMD.H
 */

/*
 * following are the requests send to the board
 * - host to board request must be less than 64 ;
 *   flags takes up upper two bits.
 */

#define SOSOCKET        (50)
#define SOACCEPT        (51)
#define SOCONNECT       (52)
#define SOSEND          (53)
#define SORECEIVE       (54)
#define SOSKTADDR       (55)
#define SOCLOSE         (56)
#define SOVERIFY        (57)
#define SOIOCTL         (58)
#define SOSELECT        (59)


#define NET_DLOAD       0               /* net download         */
#define NET_ULOAD       1               /* net upload           */
#define NET_START       2               /* start downloaded stuff*/

#define NET_GSTAT       BRDGSTAT        /* read net statistics  */
#define NET_RSTAT       BRDRSSTAT       /* read & reset stats   */

#define NET_GCONF       BRDGCONF        /* get configuration msg*/

#define NET_SARP        BRDSARP         /* set ARP              */
#define NET_GARP        BRDGARP         /* get ARP              */
#define NET_DARP        BRDDARP         /* delete ARP           */

#define NET_ADDRT       BRDADDRT        /* add RT entry         */
#define NET_DELRT       BRDDELRT        /* delete RT entry      */
#define NET_SHOWRT      BRDSHOWRT       /* show RT              */
#define NET_DISPRT      BRDDISPRT       /* display RT           */


/*  unsolicited messages from board */

#define SOSELWAKEUP      (80)
#define SOHASOOB         (81)
#define NET_PRINTF      100             /* print out msg        */
#define NET_PANIC       101             /* oh-my-gosh           */
#define IM_ALIVE        102             /* I think therfore I am*/

#define TSCOMMAND       40              /* telnet request code  */

#define REPLY_OK        0x00            /* all is well          */


#define NM_MAGIC_DATA   0x80
```

```
#define MQ_EXOS          0x01          /* exos own Q element   */
#define MQ_DONE          0x02          /* exos done with Q elmnt*/
#define MQ_OVERFLOW      0x04          /* data are too big     */
```

```
/*
 * filename: EXOS.H
 *
 *
 *   Data structures and associted constants definition for the EXOS-203
 *   ethernet controller , compatable with 3.1 version of the net module.
 *
 */


 struct exos_paddr{
        Ushort  base;           /* segment value        */
        Ushort  off;            /* offset value         */
        };

 /*
  * General headers
  *
  */

 struct headers {
        /* Q or mailbox header  */

        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        };

 struct messages{               /* Q or mailbox header  */
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */

        };

/*
 * NET_DLOAD structure
 *
 */

 struct net_dload{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/
```

```
        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */

        /* semantic of this structure */

        Ushort  nm_length;      /* length of data       */
        long    nm_source;      /* source address       */
        struct  exos_paddr nm_dest;     /* destination address  */
        Uchar   nm_xmbyte;
        };

/* NET_START structure  */

 struct net_start{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */
        short   nm_sa1;
        short   nm_sa2;
        };

/*
 * the following messages all pertain to the tcp/ip/socket
 * software which runs on the board;
 */

 /*  SOCK_PKT structure: send/receive data to/from a socket  */


 struct Sock_pkt{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */
        short   nm_isaddr;      /* non-zero iff nm_sadr */
        struct  sockaddr nm_saddr;      /* socket address       */
```

```
        long    nm_bufaddr;     /* host buffer addr     */
        Ushort  nm_count;       /* byte count           */
        char    nm_data;        /* place for data       */
        };

/* Sock_cmd structure: send/receive command to/from exos        */

 struct Sock_cmd{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */

        /* semantics of this structure  */

        short   nm_isaddr;      /* non-zero iff nm_saddr*/
        struct  sockaddr nm_saddr;      /* socket address       */
        struct  sockproto nm_sproto;    /* protocol structure   */
        short   nm_isproto;     /* non-zero iff sproto  */
        short   nm_type;        /* family with protocol */
        short   nm_options;     /* flags                */
        short   nm_iamroot;     /* is this priv user    */
        };

/* Sock_ioctl structure: socket ioctl command           */

 struct Sock_ioctl{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */
        /* semantics of this structure  */

        short   nm_ioccmd;      /* ioctl command        */
        char    nm_iocdata[40]; /* holder for stuff     */
        };

/* Sock_printf structure: printf/panic from exos        */

struct Sock_printf{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
```

```
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */

        /* semantics of this structure  */

        short   nm_dummy;       /* align to long word   */
        char    nm_prdata[48];  /* printf data          */
        };

/* Sock_select structure: select on socket             */

 struct Sock_select{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */

        /* semantic of this structure   */

        short   nm_rw;          /* how to select (read=0/write=1  */
        short   nm_proc;        /* host proc which is selecting  */
        short   nm_selcoll;     /* number of select collision for host */
        };

/* Sock_hasoob for when get out-of-band data    */

 struct Sock_hasoob{
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */

        /* semantic of this structure   */

        short   nm_sogrp;       /* proc group   */
```

```
        };
/* Telnet_srvr structure to hold telnet data            */

struct Telnet_srvr {
        Ushort  mh_link;        /* exos link address    */
        Uchar   mh_reserved;    /* not used must be 0   */
        Uchar   mh_status;      /* status of Q element  */
        Ushort  mh_length;      /* length of data packet*/

        /* header in message proper */

        short   nm_soid;        /* socket id            */
        long    nm_userid;      /* seq # attached to msg*/
        Uchar   nm_request;     /* command to exos      */
        Uchar   nm_reply;       /* reply from exos      */

        /* semantics of the structure   */

        Uchar   nm_tsrqst;      /* telnet server command*/
        Uchar   nm_tsdlen;      /* data length          */
        char    nm_tsdata[32];  /* data buffer          */
};


/*
 * Format of a standard "exos-to-host" or "host-to-exos" message:
 *   - this is what is linked together in a Q which both the host
 *     and exos manipulates while talking to each other.
 *   - a message contains:
 *       - a header describing the state of the message and its
 *         size
 *       - an actual network message
 *       - ( For the host:
 *       - a link for the host to use to maintain and follow the
 *         message queue with
 *
 */

struct msg{
  union exos_u {
        struct headers msg_hd;
        struct messages msg_msg;
        struct net_dload nm_dload;
        struct net_start nm_start;
        struct Sock_pkt nm_packet;
        struct Sock_cmd nm_cmd;
        struct Sock_ioctl nm_ioctl;
        struct Sock_printf nm_printf;
        struct Sock_select nm_select;
        struct Sock_hasoob nm_hasoob;
        struct Telnet_srvr nm_telnet;
   } nm_u;
  struct msg *msg_link;                 /* host link to next msg */
  };

/*
```

```
 *  To run this board, a static data area is kept in the ACP task
 *  which will contain the linked list of this messages acting as
 *  ring buffer.
 *  The [rw]msg_area structures is used to contain the working
 *  queues which both the host and exos manipulates
 *
 */

#define NET_RBUFS       7
#define NET_WBUFS       7

 struct rmsg_area {                         /* read message queue   */
   Ushort ma_rlink;                         /* exos link to next msg*/
   struct msg ma_rmsgs[NET_RBUFS];          /* exos to host msgs    */
   struct msg *ma_lastr;                    /* last examined msg    */
   };

 struct wmsg_area {
   Ushort ma_wlink;                         /* exos link to queue   */
   struct msg ma_wmsgs[NET_WBUFS];          /* host to exos msg     */
   struct msg *ma_lastw;                    /* last examined msg    */
   };
```

```
/*
 * These are the DIC and DPB lengths of the Executive directives
 */
# define QIO  06001
# define QIOW 06003
# define ALUN 02007
# define WTSE 01051
# define GTIM 01075
# define SPWN 06413
# define SDRC 03615
# define SDAT 02507
# define STOP 0603
# define RCVD 02113
# define MRKT 02427

/*  Executive return status */

# define IE_BAD  -01                    /* bad parameters       */
# define IE_IFC  -02                    /* illegal function     */
# define IE_DNR  -03                    /* device not ready     */
# define IE_SPC  -06                    /* illegal bufferr      */
# define IE_ABO  -15                    /* request aborted      */
# define IE_PRI  -16                    /* priv or channel error*/
# define IE_DFU  -24                    /* no free channel      */
# define IE_FHE  -59                    /* fatal hardware error */
# define IE_OFL  -65                    /* device offline       */

/*
 * These are the function codes related to the QIO call to the ZE device
 */

/*
 * following five codes are already defined in standard rsx header file
 * rsx.h and are not defined here only shown under comment for clarity

 define IO_KIL    000012              # kill all outstanding request #
 define IO_WLB    000400              # write to the EXOS memory     #
 define IO_RLB    001000              # read from the EXOS memory    #
 define IO_ATT    001400              # attach fn: made no-op        #
 define IO_DET    002000              # detach fn: made no-op        #

 */

#define IO_EXC    002400              /* EXOS board admn. operation */
#define    EX_INI       BRDINIT          /* Reset and configure EXOS   */
#define    EX_CNF       BRDGCONF                 /* get configuration msg      */
#define    EX_STR       BRDSTART         /* Execute EXOS procedure     */
#define    EX_STS       BRDGSTAT         /* Read network statistics    */
#define    EX_SAR       BRDSARP          /* set up an ARP table entry  */
#define    EX_GAR       BRDGARP          /* Retrive an ARP table entry */
#define    EX_DAR       BRDDARP          /* Delete an ARP table entry  */
#define    EX_ART       BRDADDRT         /* Add an Routing table entry */
#define    EX_DRT       BRDDELRT         /* Delete an RT entry         */
#define    EX_SRT       BRDSHOWRT        /* Fetch an RT entry          */
#define    EX_NRT       BRDDISPRT        /* Fetch next RT entry        */
```

```
#define    EX_RST        BRDRSSTAT       /* Read & Reset network stats */
#define    EX_OPN        0020            /* Open an admin channel      */
#define    EX_CLS        0021            /* Close an admin channel     */
#define    EX_POS        BRDADDR         /* Seek EXOS's memory         */

#define IO_ACS    003000              /* Socket access operations   */
#define    SA_OPN        50              /* Open a socket              */
#define    SA_ACC        51              /* Accept a remote socket     */
#define    SA_CON        52              /* Connect to a remote socket */
#define    SA_SAD        55              /* get socket informations    */
#define    SA_CLS        56              /* close an opened socket     */
#define    SA_SEL        59              /* perform select op on socket*/
#define    SA_USL        0210            /* kill the outstanding select call */
#define    SA_URG        0200            /* prepare for urgent msg     */
#define    SA_ROO        0220            /* remove oob pkt from pending list */

#define IO_XFR    003400              /* data transfer operation    */
#define    IX_RDS        0000            /* read from TCP stream       */
#define    IX_WRS        0001            /* write to TCP stream        */
#define    IX_SND        53              /* send datagram to a socket  */
#define    IX_RCV        54              /* receive socket datagram    */

#define IO_SOC    004000              /* socket control operations  */
#define    SO_DON        SIOCDONE        /* shutdowm r/w on socket     */
#define    SO_SKP        SIOCSKEEP       /* set keep alive             */
#define    SO_GKP        SIOCGKEEP       /* inspect keep alive         */
#define    SO_SLG        SIOCSLINGER     /* set linger time            */
#define    SO_GLG        SIOCGLINGER     /* get linger time            */
#define    SO_SOB        SIOCSENDOOB     /* send out of band           */
#define    SO_ROB        SIOCRCVOOB      /* receive out of bound       */
#define    SO_AMK        SIOCATMARK      /* at oob mark ?              */
#define    SO_SPG        SIOCSPGRP       /* set process group          */
#define    SO_GPG        SIOCGPGRP       /* get process group          */
#define    SO_NRD        FIONREAD        /* FIONREAD                   */
#define    SO_NBO        FIONBIO         /* FIONBIO                    */
#define    SO_ASY        FIOASYNC        /* FIOASYNC                   */

#define IO_LOG          004400          /* read error msg from EXOS */
#define IO_TEL          0177000         /* telnet server pseudo fn code */
#define TS_HNG          0176000         /* hangup carrier pseudo fn code*/

/*
 * All the Socket related parameters in the QIO call are passed
 * throgh the structure "SOictl" defined below.
 */

struct  SOictl  {
        short   hassa;                  /* non-zero if sa specified   */
        struct  sockaddr sa;            /* socket address (optional)   */
        short   hassp;                  /* non-zero if sp specified       */
        struct  sockproto sp;           /* socket protocol (optional)   */
        int     type;                   /* socket type             */
        int     options;                /* options                 */
        /* these are for select ()              */
        int     nfd;
        long    *wp;
```

```
        long    *rp;
        long    timo;
        };
```

```
/*
 * finename:    EXREG.H
 */

/*
 * data structures for the Excelan exos/203 ethernet controller
 */

/*
 * The exctrl structure is used to maintain the software device during
 * its use.
 */

struct  exctrl {
        Ushort  ex_port;                /* our port address 164000   */
        struct  init_msg *ex_imsg;      /* virtual pointer to init msg   */
        Ushort  ex_state;               /* state of the controller   */
        Uchar   ex_init;                /* device has been initialized    */
        };

/*
 * ex_state values
 */

# define  ST_INIT       0x01            /* device has been setup   */
# define  ST_WAITING    0x02            /* waiting for setup       */

/*
 * port address word
 */

# define  EX_PORT       04000           /* port address offset in I/O page*/
# define  EX_PORTA      0               /* offset for PORTA        */
# define  EX_PORTB      2               /* offset for PORTB        */

/*
 * macros for ease of use
 */

# define  PORTA         (ex_db.ex_port + EX_PORTA)
# define  PORTB         (ex_db.ex_port + EX_PORTB)

/*
 * bits in port B
 */

# define  PB_ERROR      001             /* fatal error when 0      */
# define  PB_INT        002             /* exos has interrupted when 1    */
# define  PB_READY      008             /* exos is ready when 0    */
```

```
/* unsigned data types (shorthand) */
typedef unsigned int    Uint;
typedef long            Ulong;
typedef unsigned short  Ushort;
typedef char            Uchar;
```

```
/* @(#)in.h     1.3 4/12/85 */

/*
 * GAP 1/11/85:  W A R N I N G  - This file is included by both host
 * and board code.  Make changes with extreme caution, and test
 * effects on both the host and board sides.
 */

/*
 * Constants and structures defined by the internet system,
 * Per RFC 790, September 1981.
 */

/*
 * Protocols
 */
#define IPRO_ICMP               1               /* control message protocol */
#define IPPROTO_GGP             2               /* gateway^2 (deprecated) */
#define IPRO_TCP                6               /* tcp */
#define IPRO_PUP                12              /* pup */
#define IPRO_UDP                17              /* user datagram protocol */

#define IPRO_RAW                255             /* raw IP packet */
#define IPRO_MAX                256

/*
 * Port/socket numbers: network standard functions
 */
#define IPPORT_ECHO             7
#define IPRT_DISCARD            9
#define IPRT_SYSTAT             11
#define IPPORT_DAYTIME          13
#define IPRT_NETSTAT            15
#define IPRT_FTP                21
#define IPPORT_TELNET           23
#define IPPORT_SMTP             25
#define IPRT_TIMESERVER 37
#define IPPORT_NAMESERVER       42
#define IPPORT_WHOIS            43
#define IPPORT_MTP              57

/*
 * Port/socket numbers: host specific functions
 */
#define IPRT_TFTP               69
#define IPRT_RJE                77
#define IPPORT_FINGER           79
#define IPRT_TTYLINK            87
#define IPRT_SUPDUP             95

/*
 * UNIX TCP sockets
 */
#define IPRT_EXECSERVER 512
#define IPPORT_LOGINSERVER      513
#define IPPORT_CMDSERVER        514
```

```
/*
 * UNIX UDP sockets
 */
#define IPPORT_BIFFUDP          512
#define IPRT_WHOSERVER  513

/*
 * Ports < IPPORT_RESERVED are reserved for
 * privileged processes (e.g. root).
 */
#define IPPORT_RESERVED         1024

/*
 * Link numbers
 */
#define IMPLK_IP                155
#define IMPLK_LOWEXPER  156
#define IMPLINK_HIGHEXPER       158

/*
 * Internet address (old style... should be updated)
 */
struct in_addr {
        union {
                struct { char s_b1,s_b2,s_b3,s_b4; } S_un_b;
                struct { unsigned short s_w1,s_w2; } S_un_w;
                long S_addr;
        } S_un;
#define s_addr  S_un.S_addr     /* can be used for most tcp & ip code */
#define s_host  S_un.S_un_b.s_b2        /* host on imp */
#define s_net   S_un.S_un_b.s_b1        /* network */
#define s_imp   S_un.S_un_w.s_w2        /* imp */
#define s_impno S_un.S_un_b.s_b4        /* imp # */
#define s_lh    S_un.S_un_b.s_b3        /* logical host */
#define S_baddr S_un.S_un_b
};

/*
 * Macros for dealing with Class A/B/C network
 * numbers.  High 3 bits of uppermost byte indicates
 * how to interpret the remainder of the 32-bit
 * Internet address.  The macros may be used in time
 * time critical sections of code, while subroutine
 * versions also exist use in other places.
 */
/*
 * GAP 1/10/85:  Apparently these are designed to work on internet
 * addresses which reside in network order in RAM, if regarded as
 * a byte string.  Be careful, because 4.2BSD defines just one
 * version of these macros, which works on internet addresses only
 * after they are swapped into proper order (in a CPU register)
 * by ntohl().
 */

/* GAP 1/10/85: Note fancy footwork below to share header with board code */
```

```
#ifdef ONBOARD          /* board make does not define MACHINE type */
#define IN_CLASSA       0x00800000L
#define INCA_NET    0x00ff0000L     /* 8 bits of net # */
#define INCA_LNA    0xff00ffffL
#define INCB        0x00400000L
#define INCB_NET    0xffff0000L     /* 16 bits of net # */
#define INCB_LNA    0x0000ffffL
#define INCC_NET    0xffff00ffL     /* 24 bits of net # */
#define INCC_LNA    0x0000ff00L
#endif

#ifndef ONBOARD         /* board make does not define MACHINE type */

#ifdef VAX
#define IN_CLASSA       0x00000080
#define INCA_NET        0x000000ff      /* 8 bits of net # */
#define INCA_LNA        0xffffff00
#define INCB    0x00000040
#define INCB_NET        0x0000ffff      /* 16 bits of net # */
#define INCB_LNA        0xffff0000
#define INCC_NET        0x00ffffff      /* 24 bits of net # */
#define INCC_LNA        0xff000000
#endif
#ifdef PDP11                    /* Also 8086 XENIX V7 C */
#define IN_CLASSA       0x00800000L
#define INCA_NET    0x00ff0000L     /* 8 bits of net # */
#define INCA_LNA    0xff00ffffL
#define INCB        0x00400000L
#define INCB_NET    0xffff0000L     /* 16 bits of net # */
#define INCB_LNA    0x0000ffffL
#define INCC_NET    0xffff00ffL     /* 24 bits of net # */
#define INCC_LNA    0x0000ff00L
#endif
#ifdef I8086                    /* XENIX 3.0, Lattice C */
#define IN_CLASSA       0x00000080
#define INCA_NET        0x000000ff      /* 8 bits of net # */
#define INCA_LNA        0xffffff00
#define INCB    0x00000040
#define INCB_NET        0x0000ffff      /* 16 bits of net # */
#define INCB_LNA        0xffff0000
#define INCC_NET        0x00ffffff      /* 24 bits of net # */
#define INCC_LNA        0xff000000
#endif
#ifdef M68000
#define IN_CLASSA       0x80000000L
#define INCA_NET    0xff000000L     /* 8 bits of net # */
#define INCA_LNA    0x00ffffffL
#define INCB        0x40000000L
#define INCB_NET    0xffff0000L     /* 16 bits of net # */
#define INCB_LNA    0x0000ffffL
#define INCC_NET    0xffffff00L     /* 24 bits of net # */
#define INCC_LNA    0x000000ffL
#endif
#ifdef Z8000
#define IN_CLASSA       0x80000000L
#define INCA_NET    0xff000000L     /* 8 bits of net # */
```

```
#define INCA_LNA    0x00ffffffL
#define INCB        0x40000000L
#define INCB_NET    0xffff0000L     /* 16 bits of net # */
#define INCB_LNA    0x0000ffffL
#define INCC_NET    0xffffff00L     /* 24 bits of net # */
#define INCC_LNA    0x000000ffL
#endif

#endif ONBOARD          /* board make does not define MACHINE type */

#define IN_NETOF(in) \
        (((in).s_addr&IN_CLASSA) == 0 ? (in).s_addr&INCA_NET : \
                ((in).s_addr&INCB) == 0 ? (in).s_addr&INCB_NET : \
                        (in).s_addr&INCC_NET)
#define IN_LNAOF(in) \
        (((in).s_addr&IN_CLASSA) == 0 ? (in).s_addr&INCA_LNA : \
                ((in).s_addr&INCB) == 0 ? (in).s_addr&INCB_LNA : \
                        (in).s_addr&INCC_LNA)

#define INADDR_ANY      0x00000000

/*
 * Socket address, internet style.
 */
struct sckadr_in {
        short   sin_family;
        unsigned short  sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};

#ifdef KERNEL
long in_netof(),in_lnaof();
#endif
```

```
/*
 * filename:    INIT.H
 */

/*
 * Structure used for initialization only.
 */

/* some of the dummy entries are due to byte swapping */

struct  init_msg {
        short   im_newstyle;            /* new style init msg?      */
        char    im_version[4];          /* version to the hardware   */
        char    im_result;              /* completion code          */
        char    im_mode;                /* set to link moce (0)     */
        char    im_hdfo[2];             /* host data format option   */
        char    im_junk[3];
        char    im_addrmode;            /* host address mode        */
        char    im_dummy2;
        char    im_mmsize;              /* memory map size (returned)   */
        char    im_byteptn[4];          /* data order byte pattern   */
        Ushort  im_wordptn[2];          /* data order word pattern   */
        long    im_longptn;             /* data order long pattern   */
        char    im_mmap[20];            /* (rest of) memory map (returned)*/

        short   im_101off;              /* movable block offset     */
        short   im_101seg;              /* movable block segment    */
        char    im_nproc;               /* number of exos 101 processes   */
        char    im_nmb;                 /* number of exos 101 mailboxes   */
        char    im_nslots;              /* number of address slots   */
        char    im_nhosts;              /* number of hosts == 1     */

/* "host to exos" stuff */

        long    im_h2exqaddr;           /* host to exos msg a address   */
        short   im_h2exoff;             /* offset from base of actual q   */
        char    im_h2extype;            /* interrupt type for h2ex msg q  */
        char    im_h2exvalue;           /* interrupt value          */
        long    im_h2exaddr;            /* interrupt address        */

/* "exos to host" stuff */

        long    im_ex2hqaddr;           /* exos to host msg q address   */
        short   im_ex2hoff;             /* offset from base of actual q   */
        char    im_ex2htype;            /* interrupt type for ex2h msg q  */
        char    im_ex2value;            /* interrupt value          */
        long    im_ex2haddr;            /* interrupt address        */
};

/* im_mode */

# define  EXOS_LINKMODE 0
# define  EXOS_HOSTLOAD 1
# define  EXOS_NETLOAD  2
```

```
/*
 * filename:    IOPKT.H
 */

struct  rel_addr {                          /* struct relocated address   */
        Ushort  rel_bias;                   /* relocation bias            */
        Ushort  dis_bias;                   /* displacement bias          */
        };

struct  iopkt {                             /* I/O pakcet field definition    */
        struct  iopkt *i_lnk;               /* link to next I/O packet   */
        Uchar   i_pri ;                     /* priority of the requesting task*/
        Uchar   i_efn ;                     /* event flag number          */
        Ushort  i_tcb ;                     /* TCB address of requester   */
        Ushort  i_ln2 ;                     /* address of second LUT word   */
        Ushort  i_ucb ;                     /* address of UCB             */
        Ushort  i_fcn ;                     /* function code + modifier       */
        struct  {
           Ushort       v_iosb;             /* virtual address of IOSB   */
           struct       rel_addr r_iosb;  /* relocated address of IOSB   */
             } i_iosb;
        Ushort  i_ast ;                     /* virtual address of AST routine */
        struct  {
          struct  rel_addr  i_buf;
          Ushort  i_cnt;
          struct  rel_addr  i_soictl;
          Ushort  i_prm4;
          Ushort  i_prm5;
          Ushort  i_prm6;
          } i_prm;
        };
```

```
/*
 * filename:   RTHDATA.H
 */

        /* DATA STRUCTURES FOR THE TELNET SERVER */

        # define        MAXCNT  1
        # define        ctrl(x)   ((x)&037)
        # define        strip(x)  ((x)&0177)
        # define        PTYNO   8
        # define        BS      010     /* character back space's ascii value*/
        # define        TC_BIN  065
        # define        TC_NEC  047
        # define        SF_SMC  02440
        # define        MAXBYTVAL       256

        /* EXOS-to-host requests are : */

        # define        TSCARON     0   /* x2h: carrier on (open connection) */
        # define        RLCARON     1   /* x2h: carrier on (for rlogin) */
        # define        TSCAROFF    2   /* x2h:carrier off(closed connection)*/
        # define        TSREAD      3   /* x2h: read data (net-to-host) */
        # define        TSNVTFUNCT  4   /* x2h: IP, AYT, EC, EL, AO */
        # define        TSDOOPT     5   /* x2h: do BINARY, ECHO, etc */
        # define        TSDONTOPT   6   /* x2h: don't BINARY, ECHO, etc */

        /* Host-to-EXOS request codes ae as follows : */


        # define   TSWRITE      32      /* h2x: write data */
        # define   TSHANGUP     33      /* h2x: close connection */

        /*
         * In reply message from the EXOS to the host, nm_reply may contain
         * the following values, for any request:
         */

        # define        TSERRBADSOID    32
        # define        TSERRPENDING    33
        # define        TSERRCLOSING    34
        # define        TSERRBADREQ     35
        # define        TSERRTOOBIG     36

        /* The NVTFUNCT 's */

        # define        IP      244
        # define        AO      245
        # define        AYT     246
        # define        EC      247
        # define        EL      248

        /* The terminal options */

        # define        TELOPT_BINARY   0
        # define        TELOPT_ECHO     1
        # define        TELOPT_SGA      3
```

```
/* Command table structure */

struct cmd {
        TEXT    tsrqst;         /* telnet server command */
        int     (*handler)();
       } cmdtab[] = {
                { TSCARON, caron},
                { RLCARON, caron},
                { TSCAROFF, bye},
                { TSREAD, zt_read },
                { TSNVTFUNCT, nvtfunct},
                { TSDOOPT, do_option},
                { TSDONTOPT, dont_option },
                { TSWRITE, wr_reply },
                0
        };

/* The following is the status structure for all the pty's */

struct status {
        short pty_number;       /* pty_device no. */
        short carrier_on;       /* if 1, then logged on */
        short rlogin;           /* if 1 then it is a remote login */
        int   reply_pending;    /* a counter whose int indicates no.*/
                                   /* of pkts sent to EXOS, */
                                   /* MAX value = MAXCNT */
        short echo_opt;         /* If 1, then echo set */
        short binary_opt;       /* If 1, then binary option set */
        short sga_opt;          /* If 1, then sqa option set */
} pty_status[] = {
        { 0,0,0,0,0,0,0,0 },
        { 1,0,0,0,0,0,0,0 },
        { 2,0,0,0,0,0,0,0 },
        { 3,0,0,0,0,0,0,0 },
        { 4,0,0,0,0,0,0,0 },
        { 5,0,0,0,0,0,0,0 },
        { 6,0,0,0,0,0,0,0 },
        { 7,0,0,0,0,0,0,0 },
         0
};

struct  packet {
        struct  packet *link;   /* link word */
        Ushort  moreto_op;      /* if 1 then more O/P to come  */
        Ushort  tcb_dummy;      /* always zero */
        Ushort  pty_no;         /* unit number */
        Ushort  ucb_dummy;      /* UCB address */
        Ushort  i_fcn;          /* always IO_TEL = 0177000  */
        Ushort  request;        /* telnet request        */
        Ushort  byte_cnt;       /* byte count            */
        char    w_data[32];     /* write-data */
    };
```

```
/* @(#)socket.h 1.8 7/29/85 */
/*      socket.h        4.16    82/06/08        */

/*
 * GAP 1/11/85:  W A R N I N G  - This file is included by both host
 * and board code.  Make changes with extreme caution, and test
 * effects on both the host and board sides.
 */

#ifdef  BSD4dot2
#define accept          ex_accept
#define connect         ex_connect
#define gethostname     ex_gethostname
#define receive         ex_receive
#define select          ex_select
#define send            ex_send
#define socket          ex_socket
#define socketaddr      ex_socketaddr
#define shutdown        ex_shutdown

#define htonl           ex_htonl
#define htons           ex_htons
#define ntohl           ex_ntohl
#define ntohs           ex_ntohs
#define swab            ex_swab
#endif  BSD4dot2

/*
 * Externally visible attributes of sockets.
 */

/*
 * Socket types.
 *
 * The kernel implement these abstract (session-layer) socket
 * services, with extra protocol on top of network services
 * if necessary.
 */
#define SOCK_STREAM     1               /* stream socket */
#define SOCK_DGRAM      2               /* datagram socket */
#define SOCK_RAW        3               /* raw-protocol interface */
#define SOCK_RDM        4               /* reliably-delivered message */
#define SOCK_ETH        5               /* link-mode access to e-net packets */
#define SOCK_ICMP       6               /* access to ICMP */

/*
 * Option flags per-socket.
 */
#define SO_DEBUG        0x01            /* turn on debugging info recording */
#define SO_ACCEPTCONN   0x02            /* willing to accept connections */
#define SO_DONTLINGER   0x04            /* don't linger on close */
#define SO_KEEPALIVE    0x08            /* keep connections alive */
#define SO_DONTROUTE    0x10            /* just use interface addresses */
#define SO_SMALL        0x20            /* use smaller (1/2K) buffer quota */
#define SO_REUSEADDR    0x40            /* permit local port ID duplication */
```

```
/*
 * Generic socket protocol format.
 *
 * Each process is normally operating in a protocol family,
 * whose protocols are used unless the process specifies otherwise.
 * Most families supply protocols to the basic socket types.  When
 * protocols are not present in the family, the higher level (roughly
 * ISO session layer) code in the system layers on the protocols
 * to support the socket types.
 */
struct sockproto {
        short   sp_family;              /* protocol family */
        short   sp_protocol;            /* protocol within family */
};

#define PF_UNSPEC       0               /* unspecified */
#define PF_UNIX         1               /* UNIX internal protocol */
#define PF_INET         2               /* internetwork: UDP, TCP, etc. */
#define PF_IMPLINK      3               /* imp link protocols */
#define PF_PUP          4               /* pup protocols: e.g. BSP */
#define PF_CHAOS        5               /* mit CHAOS protocols */
#define PF_OISCP        6               /* ois communication protocols */
#define PF_NBS          7               /* nbs protocols */
#define PF_ECMA         8               /* european computer manufacturers */
#define PF_DATAKIT      9               /* datakit protocols */
#define PF_CCITT        10              /* CCITT protocols, X.25 etc */

/*
 * Generic socket address format.
 *
 * Each process is also operating in an address family, whose
 * addresses are assigned unless otherwise requested.  The address
 * family used affects address properties: whether addresses are
 * externalized or internalized, location dependent or independent, etc.
 * The address can be defined directly if it fits in 14 bytes, or
 * a pointer and length can be given to variable length data.
 * We give these as two different structures to allow initialization.
 */
struct sockaddr {
        short   sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address */
};

/*
 * The first few address families correspond to protocol
 * families.  Address families unrelated to protocol families
 * are also possible.
 */
#define AF_UNSPEC       0               /* unspecified */
#define AF_UNIX         1               /* local to host (pipes, portals) */
#define AF_INET         2               /* internetwork: UDP, TCP, etc. */
#define AF_IMPLINK      3               /* arpanet imp addresses */
#define AF_PUP          4               /* pup protocols: e.g. BSP */
#define AF_CHAOS        5               /* mit CHAOS protocols */
#define AF_OISCP        6               /* ois communication protocols */
#define AF_NBS          7               /* nbs protocols */
```

```
#define AF_ECMA         8               /* european computer manufacturers */
#define AF_DATAKIT      9               /* datakit protocols */
#define AF_CCITT        10              /* CCITT protocols, X.25 etc */
#define AF_ETHER        11              /* Ethernet Address */
#define AF_COUNT        12              /* A count */
#define AF_ETYPEFILTER  13              /* Ethernet filter */

#define AF_MAX          14

/*
MWP:
Sockaddr structure for link mode access to EXOS board.
*/

#ifndef u_short
#define u_short unsigned short
#endif

#define sockaddr_link sad_link  /* for compiler */
struct sockaddr_link {
        short           sl_family;
        u_short         sl_types[6];
        short           sl_zero;
#ifdef ONBOARD
        struct enreq    *sl_pndpkt;     /* a part-empty pkt on this socket */
#endif
};

/* a handy macro */
#define saptr(x) ((struct sockaddr_link *)(((struct socket *)(x))->so_pcb))
```

```
/*
 * filename: SOIOCTL.H
 */


/*
 * This file defines all the equate symbols for socket ioctl
 * commands. These values are actually passed onto to the board,
 * hence should not be altered.
 */



#define FIONREAD        (127)
#define FIONBIO         (126)
#define FIOASYNC        (125)
#define TIOCPKT         (112)           /* on pty: set/clear packet mode */
#define         TIOCPKT_DATA            0x00    /* data packet */
#define         TIOCPKT_FLUSHREAD       0x01    /* flush packet */
#define         TIOCPKT_FLUSHWRITE      0x02    /* flush packet */
#define         TIOCPKT_STOP            0x04    /* stop output */
#define         TIOCPKT_START           0x08    /* start output */
#define         TIOCPKT_NOSTOP          0x10    /* no more ^S, ^Q */
#define         TIOCPKT_DOSTOP          0x20    /* now do ^S ^Q */

#define SIOCDONE        (0)     /* shutdown read/write on socket */
#define SIOCSKEEP       (1)     /* set keep alive */
#define SIOCGKEEP       (2)     /* inspect keep alive */
#define SIOCSLINGER     (3)     /* set linger time */
#define SIOCGLINGER     (4)     /* get linger time */
#define SIOCSENDOOB     (5)     /* send out of band */
#define SIOCRCVOOB      (6)     /* get out of band */
#define SIOCATMARK      (7)     /* at out of band mark? */
#define SIOCSPGRP       (8)     /* set process group */
#define SIOCGPGRP       (9)     /* get process group */
#define SIOCADDRT       (10)    /* add a routing table entry */
#define SIOCDELRT       (11)    /* delete a routing table entry */
#define SIOCCHGRT       (12)    /* change a routing table entry */
```

```
#define ELMNTBUSY       1       /* the element is busy */
#define ELMNTFREE       0       /* the element is free */
#define NULLPOINTER     0       /* it is pointing to null element */


#define MAXBUF          2       /* max no of transfer buffer  */
#define BUFSIZE         1024    /* size of each such buffer   */
#define MAXIOSB         10      /* max no of IO status block  */
#define MAXSOICTL       5       /* max no of SOictl structure */

#define SOLUN   20              /* EXOSO LUN    */
#define SOEFN   1               /* efn          */

#define NOSOBUF         -10
#define NOSOIOSB        -11
#define NOSOICTL        -12
#define NOFREESOCKET    -13
```

```
/*
 * filename:    UNIDATA.H
 */
/*
 *      This file contains the data structures required for the ACP to
 *      run on a UNIBUS machine PDP-11/24
 */


#define POOL_BUFS       14      /* 14Kb buffers each of size = BUFSIZE */
#define ALLOCATED       0x1
#define DEALLOCATED     0x0
#define POOLBUFSIZE     1024

struct  pool_im {
        Ushort  state;
        struct  iopkt   *owner;
        };

struct  pool_im pool_im[POOL_BUFS] = {0};       /* pool's image */

struct  rel_addr rel1buf = {0}; /* relocated address of 1st 4kw of pool */
struct  rel_addr rel2buf = {0}; /* relocated address of 2nd 4kw of pool */

struct  iopkt   *sec_que = {0}; /* sec_que for pkts not getting pool space */

unsigned int *umraddr = {0};    /* umr addr of umr of 1st 4kw of pool */
unsigned int zeucb = 0;         /* storage for ZE0: UCB */

long    phy_buf = 0;            /* physical address of pool      */
long    uni1buf = 0;            /* 18-bit unibus address of 1st 4kw of pool */
long    uni2buf = 0;            /* 18-bit unibus address of 2nd 4kw of pool */
long    uni_msg = 0;            /* 18-bit unibus address of message area    */
```

```
/*
 * filename:    ACPROOT.C
 */

/*
 * This is the main root of the acp task. It calls init() to make some local
 * initializations.
 */

main()
{
        init();                         /* local initializations */

#ifdef DEBUG
        qio_write("OUT INIT",9,040);
#endif

        if (acpucb()) {

#ifdef  UNIBUS
         uni_ini();     /* initialize unibus related stuff      */
#endif

         do
          {
          io_pkt = dqpkt();             /* deque an user request */
          action = 1;
          if ( io_pkt ){                /* if it's an request    */
            chn = PKT.i_prm6; /* get ch # if any       */
            switch ( io_pkt->i_fcn ){   /* check the request     */

              case IO_KIL:
                io_kill();
                break;

              case IO_EXC|EX_OPN:
                iosb.cc = 1; iosb.lc = 0;
                iosb.nread = opench(CH_EXOS, PKT.i_prm4);
                if ( iosb.nread < 1 )
                  iosb.cc = IE_DFU;     /* no free channel */
                break;

              case IO_EXC|EX_CLS:
                iosb.cc = 1; iosb.lc = 0;
                if ( inrange(chn) && sametask(chn) )
                  iosb.cc = closech(chn);
                break;
              case IO_EXC|EX_INI:
                iosb.lc = 0;
                if (inrange(chn) && sametask(chn) && writeprv(chn) )
                  iosb.cc = exsetup(PKT.i_prm4);
                else
                  iosb.cc = IE_PRI;     /* priv or channel error */
                break;

              default:
```

```
                qio_write("error: EXOS not configured",27,040);
                iosb.cc = IE_DNR;       /* device not ready     */

            }
            if ( action )
              ackuser(io_pkt);
          }
        }while (!ex_db.ex_init);
        drive();
    }
    else qio_write("error: EXOS dev not ready",25,040);
}
```

```
/*
 * filename:    ANSWER.C
 */

/*
 * This function scans the entire reply message buffer starting from the next
 * to the last message buffer. For each buffer, it checks it's status field.
 * If it is owned by the host then it calls a function rprocess, to process
 * the reply and updates the status field.
 */

answer()
{
        register int i;
        register struct msg     *current;
        register struct  iopkt   *pending;

#ifdef DEBUG
        qio_write("answer",7,040);
#endif

        current = rmsg_area.ma_lastr;           /* start where we left */
        while (( current->nm_u.msg_hd.mh_status & 0x03 ) == 0 ) /* reply for host */
         {
           mp = current;
           switch(ex_mg.nm_request & 0x7F) {
                case SOSELECT:
                case SOSELWAKEUP:
                        ex_sel.nm_proc <<= 1;
                        pending = getpend((struct iopkt *)ex_sel.nm_proc);
                        break;
                default:
                        pending = getpend((struct iopkt *)ex_mg.nm_userid);
                        break;
                  }
           /* check whether the reply was solicitated */

           reply();

           if ( pending ){                  /* if it was solicitated       */
                i = pending->i_prm.i_prm6; /* get channel #               */
                if ( inform ) {             /* only if boards processing is   */
                   ackuser(pending);        /* over then acknowlege user      */
                   ch_des[i].rundn_cnt--;   /* decrement I/O rundown count    */
                  }
                if(ch_mfor_close(i)){       /* is it marked for close?if so...*/
                   closech(i);              /* ...try to close the channel    */
                  }
             }

           else

#ifdef DEBUG
          qio_write("unsolicitated reply", 20,040);
#else
          ;   /* null statement */
```

```
#endif

            rmsg_area.ma_lastr = current->msg_link;
            current = current->msg_link;
           }
        nxtrst = &rmsg_area.ma_lastr->nm_u.msg_hd.mh_status;
}
```

```
/*
 * filename: append.c
 */

/*
 * append() : this routine appends the requested io_pkt to the
 *            I/O pending list just before sending it to EXOS
 *            so that on return it can be double checked for
 *            issueing IOFIN and differentiate between solicited
 *            and unsolicited reply from EXOS.
 */


int append()
 {
   register struct iopkt *next;

   if (!io_pend)
     io_pend = io_pkt;
   else
     {
      next = io_pend;
      while ( next->i_lnk)              /* reach till end of list */
        next = next->i_lnk;
      next->i_lnk = io_pkt;             /* append it to the end   */
     }
   io_pkt->i_lnk = 0;                   /* terminate the list     */
 }


 /*
  * getpend() : this routine is called to find a match in the list of
  *             pending I/O request . If a match is found it returns
  *             the I/O packet address.
  */


 struct iopkt *getpend(pkt)
   struct iopkt *pkt;
    {
      register struct iopkt *prev, *current;

      if (io_pend)                  /* if at all any request is pending in EXOS */
       {
        prev = 0;
        current = io_pend;          /* start searching from the begining        */
        while ((current != pkt) && (current->i_lnk != 0))
                                    /* search for a match or end of list        */
          {
            prev = current;
            current = current->i_lnk;
          }
        if (current==pkt)           /* if match */
         {
           if(prev)                 /* if it is not the first element in the list*/
```

```
            prev->i_lnk = current->i_lnk;
           else
            io_pend = current->i_lnk;
           return(current);
         }
        else return(0);
     }
     else return(0);
  }
/*
 * pend_list(pkt)       ---> This routine checks if the specified packet is in
 *                      the pending list and waiting for a reply from the board
 */
/*
 * commenting out this whole routine
pend_list(pkt)
struct iopkt *pkt;
{
        register struct iopkt *current;

        if(io_pend) {
        current = io_pend;      ** start of pending list        **
        do {
           if(current == pkt)   ** match?       **
                return(1);      ** yes          **
           current = current->i_lnk;    ** no - see next**
          } while(current);
        }
        return(0);
}
*/
```

```
/*
 * FILENAME:     body.c
 */
# include       <header.c>
# include       <acproot.c>
# include       <drive.c>
# include       <setup.c>
# include       <init.c>
# include       <request.c>
# include       <append.c>
# include       <answer.c>
# include       <signaloob.c>
# include       <reply.c>
# include       <insert.c>
# include       <findslot.c>
# include       <iokill.c>
# include       <cancel.c>
# include       <delay.c>
# include       <opench.c>
# include       <rth.c>

#ifdef  UNIBUS
        #include        <uniacp.c>
#endif
```

```
/*
 * filename: CANCEL.C
 */



io_rundown(ch_no)               /* cancel all outstanding request */
 int ch_no;
 {
   register int i;
   register struct iopkt *pkt;

   /* close all channels except this one */
   for ( i=0; i<MAXCHANNEL; i++)
     if ((i != ch_no) && (ch_des[i]. ch_type != CH_FREE )) {
       ch_des[i].rundn_cnt = 0; /* force rundown count to 0 so that channel
                                   may be closed */
       closech(i);
     }

   /* kill all outstanding requests from the user */

   while(mrkcls) {      /* kill all SOCLOSE packets */
          pkt = mrkcls;
          mrkcls = mrkcls->i_lnk;
          iosb.cc = IE_ABO;
          ackuser(pkt);
        }

   while(int_que)
     {
          pkt = int_que;
          int_que = int_que->i_lnk;
          iosb.cc = IE_ABO;
          if((pkt->i_fcn == IO_KIL) || (pkt->i_fcn == IO_TEL))
            dealoc_b(pkt,sizeof(pkt));
          else
            ackuser(pkt);
     }
   while(io_pend)
     {
      pkt = io_pend;
      io_pend = io_pend->i_lnk;
      iosb.cc = IE_ABO;
      if((pkt->i_fcn == IO_KIL) || (pkt->i_fcn == IO_TEL))
         dealoc_b(pkt,sizeof(pkt));
      else {

#ifdef  UNIBUS
        freepool(pkt,0);        /* must free the pool if allocated */
#endif

        ackuser(pkt);
      }
     }
 }
```

```
/*
 * filename:    DELAY.C
 */

/*
 * The delay routine gives a time delay specified by the arguments passed:
 * tmag and tunit. If tunit = character 'T' (ticks) then a time dealy of
 * (tmag * 20 msec) is obtained.
 * If tunit='S', then a time dealy of tmag seconds is obtained.
 */

delay(tmag,tunit)
     int  tmag;
     char tunit;
{
        register int a;

        if (tunit == 'T' || tunit == 't')
                a = 1;
        else
                a = 2;                  /* default unit is seconds */
        emt(MRKT,8,tmag,a,0);
        emt(WTSE,8);
}
```

```
/*
 * filename: DRIVE.C
 */

/*
 * This is the main control flow routine of the ACP task.Its an
 * forever loop. While in the loop it first tries to dequeue a
 * packet from its external queue. These packets are nothing but
 * user's request queued by driver in packet form to the ACP task.
 * It returns from the DQPKT procedure iff some work is pending
 * for it in the form request from the user or reply from EXOS
 * or already pending requests in its internal queue. If it gets
 * a request from the user it first checks whether it needs EXOS's
 * participation or not. If not so, then it immedietly processes
 * it, otherwise queues it to its own internal FIFO queue. After
 * that it responds to all the pending reply from EXOS and then
 * processes the pending user request from its internal queue
 * subjected to the availability of free slot in the Host-to-EXOS
 * ring buffer queue. When it can not proceed any further it tries
 * to deque another packet thus completing a cycle.
 *
 */
 int drive()
   {
     FOREVER {         /* fall into an eternal loop */

       io_pkt = dqpkt();         /* deque an I/O packet */

#ifdef DEBUG
        qio_write("waked up", 8,040);
#endif

       if ( io_pkt )             /* if any request      */
        {
         int ack = 0;            /* do not acknowlege user immedietly */

         chn = PKT.i_prm6;
         switch ( io_pkt->i_fcn ) {

           case IO_EXC|EX_OPN:   /* open an admin channel */
             iosb.cc=1; iosb.1c=0;
             iosb.nread = opench( CH_EXOS, PKT.i_prm4);
             if (iosb.nread < 1)
               iosb.cc = IE_DFU;        /* channel open error   */
             ack = 1;
             break;

           case IO_EXC|EX_INI:  /* reinitialise EXOS             */
             iosb.cc=1; iosb.1c = 0;
             if ( inrange(chn) && sametask(chn) && writeprv(chn) ){
                io_rundown(chn);   /* abort all outstanding I/O     */
                iosb.cc = exsetup(PKT.i_prm4);
                }
             else iosb.cc = IE_PRI;     /* priv or channel error  */
             ack = 1;
             break;
```

```
    case IO_EXC|EX_POS:  /* position the memory relocator */
      iosb.cc= 1; iosb.lc = 0;
      if ( inrange(chn) && sametask(chn) ){
        ch_des[chn].ch_u.ch_addr.base = PKT.i_prm4;
        ch_des[chn].ch_u.ch_addr.off  = PKT.i_prm5;
        }
      else iosb.cc = IE_PRI;
      ack = 1;
      break;

    case IO_EXC|EX_CNF:  /* get configuration message    */
      iosb.cc = 1; iosb.lc = 0;
      if ( inrange(chn) && sametask(chn) )
         ucopy( (char *) &init_msg, &PKT.i_buf.rel_bias,
                 sizeof( struct init_msg ));
      else
         iosb.cc = IE_PRI;
      ack = 1;
      break;

    case IO_EXC|EX_CLS:  /* close admin channel */
      iosb.cc= 1; iosb.lc = 0;
      if ( inrange(chn) && sametask(chn) )
        iosb.cc = closech(chn);
      ack=1;
      break;

    case IO_ACS|SA_USL:
      iosb.cc = 1; iosb.lc = 0;
      if ( inrange(chn) && sametask(chn) )
         fin_pen(SA_USL);
      else
         iosb.cc = IE_PRI;
      ack = 1;
      break;

    case IO_ACS|SA_URG:  /* prepare for urgent msg */
      if ( inrange(chn) && sametask(chn) )
        PKT.i_prm4 = ch_des[chn].ch_u.ch_soid;
      /* remember the socket id in the pending packet
         for future match on receive of urgent signal */
      ch_des[chn].rundn_cnt++;   /* increment I/O rundown count */
      append();
      break;

    default:
      insert();      /* put the request in internal queue  */
    }
  if ( ack )       /* processed request, inform requester */
    ackuser(io_pkt);
}
answer();          /* process reply msg queue             */

/* loop to process pending request on availability of free slots */
```

```
        while ( int_que && ( free_slot = findslot() ) )
          request();

#ifdef  UNIBUS
        put_sec_que();  /* put the secondary que onto the top of int_que */
#endif

      }
  }
```

```
/*
 * filename:    EXVAR.C
 */

/*
 * This file defines all global variables for ACP task.
 */

struct  rmsg_area       rmsg_area = {0};

#ifdef  UNIBUS
char align[(((sizeof(rmsg_area)/020) * 020) + 020) - sizeof(rmsg_area)] = {0};
                /*
                 * align is defined to make sure the unibus address
                 * corresponding to wmsg_area is so aligned that its
                 * lower 4-bits are always zero - this is for the
                 * convenience of the board to make the unibus address
                 * 16-byte aligned.
                 */
#endif

struct  wmsg_area       wmsg_area = {0};
struct  SOictl          SOictl    = {0};
struct  iosb            iosb      = {0};
struct  exctrl          ex_db     = {0};
struct  init_msg        init_msg  = {0};
struct  iopkt           *io_pkt   = {0};
struct  iopkt           *int_que  = {0};
struct  iopkt           *io_pend  = {0};
struct  iopkt           *mrkcls   = {0};
struct  msg             *free_slot= {0};
struct  msg             *mp = {0};
Uchar                   *nxtrst = {0};
Uchar                   *nxtwst = {0};
struct SOictl           param = {0};
Ushort                  inform = 1;
Ushort                  action = 1;
Ushort                  cmd = 0, subcmd = 0;
int                     chn = 0;
struct channel ch_des[MAXCHANNEL] = {0};
int                     exopnfrwrite = 0;
int                     factor = sizeof( struct headers );
int                     zeint = 0;      /* interrupt vector address */
int                     zeport = 0;     /* port offset */
```

```
/*
 * filename: FINDSLOT.C
 */

/*
  this function checks the status of the next available buffer
  in the queue and returns it if it belongs to host otherwise
  simply returns null pointer;
 */

struct msg *findslot()
  {
    register struct msg *current;

    current = wmsg_area.ma_lastw;        /* set to currently available buffer */
    if ((current->nm_u.msg_hd.mh_status & 03) == 0 ) /* check the ownership  */
      {
       wmsg_area.ma_lastw = current->msg_link;   /* set it to the next buffer */
       nxtwst = &wmsg_area.ma_lastw->nm_u.msg_hd.mh_status;
       return( current );
      }
    else
      return( 0 );                       /* return a null pointer */
  }
```

```
/*
 * filename : HEADER.C
 */

/*
 *  this file includes entire environment files
 */

/*  define the machine type as RSX */

#define RSX 11

# include <std.h>
# include <rsx.h>
# include <socket.h>
# include <soioctl.h>
# include <brdioctl.h>
# include <in.h>
# include <extypes.h>
# include <defines.h>
# include <exqio.h>
# include <exos.h>
# include <exiocmd.h>
# include <iopkt.h>
# include <channel.h>
# include <init.h>
# include <rthdata.h>
# include <exreg.h>
# include <exvar.c>

#ifdef UNIBUS
        # include <unidata.h>
#endif
```

```
/*
 * filename:    INIT.C
 *
 *
 * This function initializes the global variables
 */

init()
{
        clear(&rmsg_area, sizeof rmsg_area );
        clear(&wmsg_area, sizeof wmsg_area );
        clear(ch_des,MAXCHANNEL*sizeof( struct channel ));
        clear(&SOictl, sizeof SOictl );
        clear(&iosb,    sizeof iosb);
        clear(&ex_db, sizeof (ex_db));
        clear(&init_msg, sizeof init_msg );
        ex_db.ex_imsg = &init_msg;
        ex_db.ex_port = zeport;       /* zeport = ex_port address  */
}

/*
 * This function clears a buffer p of length size
 */

clear(p,size)
     register char *p;
     unsigned int size;
{
        int i;

        for(i=0;i<size;i++)
           *p++=0;
}
```

```
/*
 * filename:    INSERT.C
 */

/* This routine enters a currently dequeued I/O packet into
 * the ACP's internal FIFO queue
 */

insert()
  {
        register struct iopkt *next;

#ifdef DEBUG
        qio_write("insert ",8,040);
#endif
        if (!int_que)                   /* if the queue is empty */
          int_que = io_pkt;             /* make it first element */
        else                            /* else enter it at the end */
          {
           next = int_que;
           while(next->i_lnk)           /* fnd the last element  */
             next = next->i_lnk;
           next->i_lnk = io_pkt;        /* insert at the end     */
          }
        io_pkt->i_lnk = 0;              /* move null to the last link */
    }

/*
 * CL_LIST
 *
 *              This routine puts a pending IO_KIL or an SOCLOSE packet
 *              into the close list which is used to hold these packets
 *              untill all I/O on their corresponding channels is finished.
 */
cl_list()
{
        register struct iopkt *next;

        if(!mrkcls)
           mrkcls = io_pkt;
        else {
           next = mrkcls;
           while(next->i_lnk)
                next = next->i_lnk;
           next->i_lnk = io_pkt;
         }
        io_pkt->i_lnk = 0;
}

/*
 *      GET_CLS
 *
 *              This routine gets the SOCLOSE and the IO_KIL packets from
 *              the close list mrkcls and returns their address if a match
 *              is found corresponding to the channel number passed.
 */
```

```
struct iopkt *get_cls(ch_no)
       int ch_no;
   {
     register struct iopkt *prev, *current;

     if (mrkcls)               /* if at all any request is pending in EXOS */
      {
       prev = 0;
       current = mrkcls;       /* start searching from the begining        */
       while ((current->i_prm.i_prm6 != ch_no) && (current->i_lnk != 0))
                               /* search for a match or end of list        */
         {
           prev = current;
           current = current->i_lnk;
         }
       if (current->i_prm.i_prm6 == ch_no)     /* if match */
        {
          if(prev)             /* if it is not the first element in the list*/
           prev->i_lnk = current->i_lnk;
          else
           mrkcls = current->i_lnk;
          return(current);
        }
       else return(0);
     }
     else return(0);
}
```

```
/*
 * filename: IOKILL.C
 */

/*
 * this routine closes all opened channel together with any opened
 * socket, after which it issues io-done for all the pending I/O
 * request in ACP.
 */

remque( head_ptr)   /* remove all request from this que */
 struct iopkt **head_ptr;
 {
  register struct iopkt *prev, *current, *next;

#ifdef DEBUG
  qio_write("remque", 7,040);
#endif
  prev = 0;
  current = *head_ptr;
  while ( current )
    {
     next = current -> i_lnk;
     if (current->i_tcb == io_pkt->i_tcb) /* I/O request by same task */
       {
        if(current->i_fcn == IO_KIL) /* if it is an IO_KIL packet */
                dalpkt(current);        /* then deallocate it */
        else {
           iosb.cc = IE_ABO;          /* return abort status to user */
           current->i_ast= 0;         /* make sure ast routine is not entered */
           ackuser( current );
          }

        /* deque the packet from the list */

        if ( prev )
          prev -> i_lnk = next;
        else
          *head_ptr = next;
       }
     else prev = current;
     current = next;                        /* check next                */
    }
 }

int srchn ( tcb )   /* return channel number having same tcb         */
  Ushort tcb;
  {
   register int i;

#ifdef DEBUG
   qio_write("srchn",6,040);
#endif
   for ( i=0; i<MAXCHANNEL ; i++)   /* search all channels */
     if ( ( ch_des[i]. ch_tcb == tcb ) /* channel ownned by this task */
                        && !ch_mfor_close(i) ) /* ch not marked for close */
```

```
        {
         if (ch_des[i].ch_type == CH_EXOS){  /* ch is Admin type   */
#ifdef DEBUG
             qio_write("close admin ch",15,040);
#endif
             closech(i);                     /* just close the ch  */
             continue;                       /* search for next ch */
          }
         else {
             ch_des[i].ch_flag |= CH_MCLOSE;  /* mark it for close */
#ifdef DEBUG
             qio_write("return ch",10,040);
#endif
             return ( i );                    /* return this channel */
         }
        }
   return (0);         /* no more opened channel for this task */
  }

extern int cl_list();

int io_kill()
 {
  register int ch_no;

  /*
   * check if there is any opened channel for this task. If so then
   * get channel # and issue SOCLOSE request and exit. ( in the
   * reply routine if it is a reply to SOCLOSE then it checks
   * whether the I/O function code in the io packet is IO_KIL, and
   * if so instead of issuing IODONE it again insert the packet to
   * the internal I/O request queue pointed by int_que thus allowing
   * the ACP to close the second socket, if any).
   * Else if there is no opened channel for this task then it goes to
   * kill all outstanding I/O ( whether the request has been issued
   * to the board or not). Then it issues an IODONE for the IO_KIL
   * request packet.
   */

#ifdef DEBUG
   qio_write("iokill",7,040);
#endif
  if ( ch_no = srchn( io_pkt->i_tcb ) ){
#ifdef DEBUG
    qio_write("close ch",9,040);
#endif
    PKT.i_prm6 = ch_no;
    ex_mg.nm_soid = ch_des[ch_no].ch_u.ch_soid;
    ex_mg.mh_length = sizeof ( struct messages ) - factor;
    ex_mg.nm_request = SOCLOSE;
    return (1);      /* send request to board */
    }
  else               /* no more channel remains opened for this task */
    {

#ifdef DEBUG
```

```
        qio_write("kill all pending I/O",20,040);
#endif

    remque(&int_que); /* remove all pending requests */
/* donot remove outstandig requests as their replies will come from the board */
    dalpkt(io_pkt);   /* deallocate the dummy I/O packet */
    action = 0;       /* do not take any action after this */
    return( IE_ABO ); /* reply user with termination status */
    }
}
```

```
/*
 * filename: OPENCH.C
 */

/*
 * this routine first check the privilege of the task, if neccessary
 * and then finds a free channel and fils up few fields such as
 * channel type, ch_flag ( mode and protection ) and the tcb field.
 * If either there is privilege violation or no channel free it is
 * immedietly informed to the caller by returning a negative value.
 * If everything is fine it returns a channel number to the caller.
 */

int opench( dev, mode)
  int dev, mode;
  {
   register int i, priv_flag = 0;

     priv_flag = getpriv(io_pkt->i_tcb);      /* get privlege info */

    /* Now get a free channel omitting the zeroth one so that
       channel # cannot be zero                                  */

    for ( i = 1; i < MAXCHANNEL; i++ )
      if ( ch_des[i].ch_type == CH_FREE ){
        ch_des[i].ch_type = dev;        /* either CH_EXOS or CH_SOCKET */
        ch_des[i].ch_tcb  = io_pkt->i_tcb;  /* tcb address of the requester*/
        ch_des[i].rundn_cnt = 0;        /* set initial rundown count as 0  */
        if ( mode == CH_WRITE )
          ch_des[i].ch_flag |= CH_WRITE;
        if ( priv_flag )
          ch_des[i].ch_flag |= CH_PRIV;
        return (i);                     /* return channel # */
        }
    return(IE_DFU);                     /* return no free channel */

    }

/*
 * function closech(ch_no) frees an open channel unconditionally
 * by clearing all its field;
 */

extern struct iopkt *get_cls();

int closech( ch_no)
  int ch_no;
    {
    register struct iopkt *p;

    if ( inrange(ch_no) && ( sametask(ch_no) ||
                        (io_pkt->i_fcn == IO_KIL) || ch_mfor_close(ch_no) ) )
     {
       if(ch_des[ ch_no ]. rundn_cnt > 0) {  /* I/O is pending on this channel*/
          ch_des[ ch_no ].ch_flag |= CH_MCLOSE; /* then mark it for close */
          return(1);
```

```
        }
      else {
         ch_des[ ch_no ]. ch_type = CH_FREE;
         ch_des[ ch_no ]. ch_flag = 0;
         ch_des[ ch_no ]. ch_tcb  = 0;
         ch_des[ ch_no ]. ch_u. ch_addr. base = 0;
         ch_des[ ch_no ]. ch_u. ch_addr. off  = 0;
         /* now get the packets from the close list,if any,and iodone them */
         while(p = get_cls(ch_no))
             ackuser(p);
         return (1);
        }
    }
  else return (IE_PRI);        /* privilege or ownership error */
}
```

```
/*
 * reply() -> this routine post process the request to the board
 */

int reply()
  {
   register int cmd = 0;
   register int cnt;
   register char *pf;

#ifdef DEBUG
        qio_write("REPLY",6,040);
#endif

   switch(cmd = (int) ex_mg.nm_request & 0x7F) {        /* the request code */
        case SOSELECT:
        case SOSELWAKEUP:
                io_pkt = (struct iopkt *)ex_sel.nm_proc;
                break;
        default:
                io_pkt = (struct iopkt *)ex_mg.nm_userid;
                break;
           }
   chn = PKT.i_prm6;
   iosb.lc = ex_mg.nm_reply;                /* board reply status */
   iosb.cc = 1;                             /* QIO success        */
   iosb.nread = 0;
   inform = 1;          /* acknowledge the user immedietly    */

   switch ( cmd ){

     case NET_ULOAD:
       /* copy the content of nm_xmbyte first into a local buffer and
          then stick this byte to the first byte of the user buffer
          and then fall through the code of NET_DLOAD              */

       bcopy((char *)&ex_dl.nm_xmbyte, (char *)&param, sizeof (char));
       ucopy((char *) &param, &PKT.i_buf.rel_bias,
            sizeof ( char ));

     case NET_DLOAD:
       iosb.nread = ex_dl.nm_length;        /* no of bytes read */
       ch_des[chn].ch_u.ch_addr.off += iosb.nread;

#ifdef  UNIBUS
       freepool(io_pkt,((cmd == NET_ULOAD) ? 1 : 0));
#endif

       break;

     case NET_START:


     case NET_GSTAT:
     case NET_RSTAT:
```

```
      case NET_SARP:
      case NET_GARP:
      case NET_DARP:

      case NET_ADDRT:
      case NET_DELRT:
      case NET_SHOWRT:
      case NET_DISPRT:

#ifdef  UNIBUS
     freepool(io_pkt,1);        /* consider all as read requests */
#endif

       break;

      case SOSOCKET:
        if ( iosb.1c == 0 ){
          ch_des[chn].ch_u.ch_soid = ex_cmd.nm_soid;
          iosb.nread = chn;              /* return channel # */
          }
        break;
      case SOACCEPT:
      case SOCONNECT:
      case SOSKTADDR:
         if ( ex_cmd.nm_isaddr ){
          bcopy((char *)&ex_cmd.nm_saddr, (char *)&param.sa,
                 sizeof( struct sockaddr ));
          ucopy((char *) &param, &PKT.i_soictl.rel_bias,
                sizeof ( struct sockaddr));
          }
         break;

      case SOSEND:
        iosb.nread = ex_pkt.nm_count;

#ifdef  UNIBUS
       freepool(io_pkt,0);      /* write request so no Xfer involved here */
#endif

       break;

      case SORECEIVE:
        iosb.nread = ex_pkt.nm_count;
        if ( ex_pkt.nm_isaddr ){
          bcopy((char *)&ex_pkt.nm_saddr, (char *)&param.sa,
                 sizeof( struct sockaddr));
          if ( PKT.i_soictl.rel_bias )
            ucopy((char *)&param, &PKT.i_soictl.rel_bias,
                   sizeof ( struct sockaddr ));
          }

#ifdef  UNIBUS
       freepool(io_pkt,1);
#endif

       break;
```

```
case SOSELWAKEUP:       /* socket ready for I/O */
     /*
       In this case the I/O packet address is returned in the
       nm_proc field of Sock_select structure in the SELECT
       request to the board. nm_userid field is not used here.
       */
      iosb.nread = chn;
      break;

 case SOSELECT:
    PKT.i_prm5 &= ~NOREPLY; /* reply has indeed come ! */
    if(PKT.i_prm5 & UNSELECT) {     /* if unselect is requested */
       iosb.nread = chn;    /* acknowledge the user normally */
       break;
      }
    if(!ex_sel.nm_reply) {  /* not ready yet */
       inform = 0;  /* donot inform user    */
       io_pkt->i_lnk = io_pend; /* put back the packet in the */
       io_pend = io_pkt;    /* pending list */
     }
    else
      iosb.nread = chn;     /* return channel # in 2nd IOSB word */
   break;

 case SOCLOSE:
    if((io_pkt->i_fcn == IO_KIL)) {         /* issued by io_kill */
      io_pkt->i_lnk = int_que;  /* put it in internal Q again   */
      int_que = io_pkt;
      }
    else
       cl_list();   /* put the close packet in the close list */
    inform = 0;     /* donot inform user right now           */
    ch_des[chn].rundn_cnt--;/* decrement I/O rundown count as this I/O */
                            /* is to be considered done     */
    fin_pen(SA_USL);/* remove select pkts from the pending list */
    fin_pen(SA_ROO);/* remove oob pkts from the pending list */
    closech(chn);   /* close shop in ACP                     */
    break;


 case SOIOCTL:
   switch ( ex_ctl.nm_ioccmd ){

     case SIOCRCVOOB:
       bcopy(ex_ctl.nm_iocdata, &param.hassa, sizeof (char));
       ucopy((char *) &param, &PKT.i_soictl.rel_bias,
             sizeof ( char ));
       break;

     case SIOCGKEEP:
     case SIOCGLINGER:
     case SIOCATMARK:
     case SIOCGPGRP:
       param.hassa = *(short *) ex_ctl.nm_iocdata;
       ucopy((char *) &param, &PKT.i_soictl.rel_bias,
```

```
                sizeof ( short ));
            break;

          case FIONREAD:
            bcopy(ex_ctl.nm_iocdata,&param.hassa,sizeof(long));
            ucopy((char *) &param, &io_pkt.i_prm.i_soictl.rel_bias,
                  sizeof ( long ));
            break;

          default:
            break;
         }
         break;

        case SOHASOOB:
          fin_pen(SA_URG);          /* give a signaloob to the user */
          inform = 0;
          break;


         case TSCOMMAND:            /* telnet server command */
           dispatch(&ex_tel);
           inform = 0;              /* donot do any IODONE on this packet */
           break;

         case NET_PRINTF:
         case NET_PANIC:
           pf = &mp->nm_u.nm_printf.nm_prdata;
           for(cnt=0;((*pf != '\n') && (*pf != '\0'));cnt++,pf++);
           qio_write(&mp->nm_u.nm_printf.nm_prdata,cnt,0);
           if(*pf == '\n')
              qio_write("\r\n",2,0);
           break;

         defaut:
           break;
      }

      ex_hd.mh_length = sizeof( union exos_u ) - sizeof ( struct headers);
      ex_hd.mh_status |= MQ_EXOS;       /* change ownership */
      write_port(PORTB, 0);             /* inform EXOS      */
    }
```

```
/*
 * filename: REQUEST.C
 */


#ifdef  UNIBUS
        extern long getpool();
#endif

extern long absadr();

/*
 *      int admin()
 */


int admin()
  {

#ifdef DEBUG
        qio_write("ADMIN",6,040);
#endif

   if ( inrange(chn) && sametask(chn) && !ch_mfor_close(chn) )
     {

#ifdef  UNIBUS
        if(PKT.i_cnt > POOLBUFSIZE)
           return(IE_SPC);      /* return illegal buffer */
#endif

      switch ( cmd ){

        case IO_RLB:            /* Time being this is equated with IO_WLB */
        case IO_WLB:
          ex_dl.mh_length  = sizeof( struct net_dload ) - factor;
          if ( cmd == IO_WLB ){
            if ( !writeprv(chn) ) return (IE_PRI);
            ex_dl.nm_request = NET_DLOAD;

#ifdef  UNIBUS
            ex_dl.nm_source = getpool(io_pkt,1);
#endif
            }
          else {

#ifdef  UNIBUS
            ex_dl.nm_source = getpool(io_pkt,0);
#endif
            ex_dl.nm_request = NET_ULOAD;
           }
          ex_dl.nm_length  = PKT.i_cnt;

#ifndef UNIBUS
```

```
            ex_dl.nm_source  = absadr( & PKT.i_buf );
#endif

            ex_dl.nm_dest.base = ch_des[chn].ch_u.ch_addr.base;
            ex_dl.nm_dest.off  = ch_des[chn].ch_u.ch_addr.off;
            break;

          case IO_EXC:
            switch ( subcmd ){

              case BRDSTART:
                if ( writeprv(chn) ){
                  ex_hd.mh_length = sizeof(struct net_start) - factor;
                  ex_str.nm_request = NET_START;
                  ex_str.nm_sa1   = PKT.i_prm4;
                  ex_str.nm_sa2   = PKT.i_prm5;
                }
                else return (IE_PRI);
                break;


              case NET_GSTAT:
              case NET_RSTAT:

              case NET_SARP:
              case NET_GARP:
              case NET_DARP:

              case NET_ADDRT:
              case NET_DELRT:
              case NET_SHOWRT:
              case NET_DISPRT:
                ex_hd.mh_length  = sizeof( struct Sock_pkt ) - factor;
                ex_pkt.nm_soid   = 0;
                ex_pkt.nm_request= subcmd;

#ifdef  UNIBUS
                ex_pkt.nm_bufaddr = getpool(io_pkt,1);
#else
                ex_pkt.nm_bufaddr= absadr(&PKT.i_buf);
#endif

                ex_pkt.nm_count = PKT.i_cnt;
                ex_pkt.nm_isaddr = 0;
                switch ( subcmd ){       /* check for write protection */

                  case NET_RSTAT:

                  case NET_SARP:
                  case NET_DARP:

                  case NET_ADDRT:
                  case NET_DELRT:
                          examine();
                    if (!writeprv(chn))
                      return (IE_PRI);
```

```
                default:;
                }
              break;

            default:
              return  IE_IFC;           /* illegal function     */
          }
          break;

        default:
          break;

      }
      return(1);
    }
    else return (IE_PRI);
   }

examine()
{
/* a dummy routine to set a breakpoint */
}

int access()
  {
    if ( subcmd == SOSOCKET )
      if ( chn = opench( CH_SOCKET, CH_WRITE ))
        PKT.i_prm6 = chn;   /* store the channel # in I/O packet*/
      else return (IE_DFU);                    /* channel open error   */
    else
      if ( inrange(chn) && sametask(chn) && !ch_mfor_close(chn) )
        ex_mg.nm_soid = ch_des[chn].ch_u.ch_soid;  /* get socket id    */
      else  return (IE_PRI);   /* error condition */

    if ( (subcmd != SOCLOSE) && (subcmd != SOSELECT)) /* no soictl struct */
      if ( PKT.i_soictl.rel_bias)
        scopy(&PKT.i_soictl.rel_bias, sizeof (struct SOictl));
        /* copy SOictl buffer from user space to my space in var param */
      else  return (IE_BAD);    /* invalid param  */

    switch( subcmd ){

      case SOSOCKET:
      case SOACCEPT:
      case SOCONNECT:
      case SOSKTADDR:
        ex_hd.mh_length = sizeof ( struct Sock_cmd ) - factor;
        if ( ex_cmd.nm_isaddr = param.hassa )
          bcopy( &param.sa, &ex_cmd.nm_saddr, sizeof (struct sockaddr));
        if ( ex_cmd.nm_isproto = param.hassp )
          bcopy(&param.sp,&ex_cmd.nm_sproto,sizeof( struct sockproto));
        ex_cmd.nm_type = param.type;
        ex_cmd.nm_options = param.options;
        ex_cmd.nm_iamroot = ((ch_des[chn].ch_flag & CH_PRIV) ? 1 : 0 );
        break;
```

```
      case SOCLOSE:
        ex_mg.mh_length = sizeof ( struct messages ) - factor;
        break;

      case SOSELECT:

        ex_sel.mh_length = sizeof( struct Sock_select) - factor;
        ex_sel.nm_rw = PKT.i_prm4 + 1;  /* read = 1 and write = 2 */
        ex_sel.nm_proc = ((Ushort)io_pkt >> 1) & 0x7FFF;
                              /* pass the pkt address with msb 0 */
        PKT.i_prm5 |= NOREPLY ;          /* indicate no reply initially */
        break;

      default:

        return (IE_IFC);        /* unknown command */
       }
      ex_mg.nm_request = subcmd;
      return (1);
    }

/*
 * int transfer()
 */

int transfer()
  {
   if ( inrange(chn) && sametask(chn) && !ch_mfor_close(chn) )
     {

#ifdef  UNIBUS
        if(PKT.i_cnt > POOLBUFSIZE)
           return(IE_SPC);      /* return illegal buffer */
#endif

      ex_pkt.mh_length = sizeof( struct Sock_pkt ) - factor;
      ex_pkt.nm_soid   = ch_des[chn].ch_u.ch_soid;
      ex_pkt.nm_count  = PKT.i_cnt;

#ifndef UNIBUS
      ex_pkt.nm_bufaddr= absadr(&PKT.i_buf);
#endif

      if ( (subcmd == SOSEND) || (subcmd == SORECEIVE) )
        {
         scopy( &PKT.i_soictl.rel_bias, sizeof(struct SOictl));
         if ( ex_pkt.nm_isaddr = param.hassa )
           bcopy(&param.sa, &ex_pkt.nm_saddr,sizeof( struct sockaddr ));
        }
      if ( (subcmd == SOSEND) || (subcmd == IX_WRS) ) {
        ex_pkt.nm_request = SOSEND;

#ifdef  UNIBUS
        ex_pkt.nm_bufaddr = getpool(io_pkt,1);
#endif
```

```
      }
      else {
        ex_pkt.nm_request = SORECEIVE;

#ifdef  UNIBUS
        ex_pkt.nm_bufaddr = getpool(io_pkt,0);
#endif

        }
       return(1);
      }
   else return (IE_PRI);
  }

/*
 * int excontrol()
 */

int excontrol()
  {
   char achar;
   short anint;
   struct rtentry route;

   if ( inrange(chn) && sametask(chn) && !ch_mfor_close(chn) )
     {
      ex_ctl.mh_length = sizeof( struct Sock_pkt ) - factor;
      ex_ctl.nm_request= SOIOCTL;
      ex_ctl.nm_soid   = ch_des[chn].ch_u.ch_soid;

      switch (subcmd) {
        case FIONREAD:
        case FIONBIO:
        case FIOASYNC:
                ex_ctl.nm_ioccmd = _IOXFIO(subcmd);
                break;
        default:
                ex_ctl.nm_ioccmd = _IOXSIO(subcmd);
                break;
           }

      scopy( &PKT.i_soictl.rel_bias, sizeof ( struct sockaddr ));

      switch( subcmd ){

        case SIOCGKEEP:
        case SIOCGLINGER:
        case SIOCRCVOOB:
        case SIOCATMARK:
        case SIOCGPGRP:
        case FIONREAD:
          break;

        case SIOCSENDOOB:
          bcopy(&param.hassa, &achar, sizeof ( achar ));
```

```
          ex_ctl.nm_iocdata[0] = achar;
          break;

        case SIOCSLINGER:
        case SIOCSKEEP:
        case SIOCSPGRP:
        case SIOCDONE:
        case FIONBIO:
        case FIOASYNC:
          bcopy(&param.hassa, &anint, sizeof ( anint ));
          *(short *)ex_ctl.nm_iocdata = anint;
          break;

        default:
          return(IE_IFC);         /* unknown comand */
      }
    }
   else return (IE_PRI);         /* if not inrange or sametask   */
   return(1);                    /* else return success          */
  }

/*
 * int request()
 */

request()
{
   register int ex_send = 1;

#ifdef DEBUG
        qio_write("request",8,040);
#endif

   io_pkt= int_que;       /* deque an packet from internal queue */
   int_que = int_que->i_lnk;
   io_pkt->i_lnk = 0;
   cmd = io_pkt->i_fcn & 0xff00;  /* mask lower 8 bits */
   subcmd = io_pkt->i_fcn & 0x00ff;/* mask off upper 8 bits */
   mp = free_slot;
   chn = PKT.i_prm6;      /* channel # if any          */
   clear(&param.hassa, sizeof ( struct SOictl));
   action = 1;    /* take action always unless not restricted by any routine */

   if(io_pkt->i_fcn == IO_KIL) {
        ex_send = io_kill();
        chn = PKT.i_prm6;        /* re-initaialize ch # as IO_CAN does*/
      }                                  /* not have any in it */
   else
        switch ( cmd ){

          case IO_WLB:                    /* write into EXOS's memory */
          case IO_RLB:
          case IO_EXC:
               ex_send = admin();
               break;
```

```
            case IO_ACS:                        /* socket access operation */
                 ex_send = access();
                 break;

            case IO_XFR:                         /* data transfer with the socket */
                 ex_send = transfer();
                 break;

            case IO_SOC:                        /* real socket control operations */
                 ex_send = excontrol();
                 break;

            case IO_TEL:
                 ex_send = telnet();
                 break;

            case TS_HNG:
                 ex_send = hangup();
                 break;

            default:
                 ex_send = IE_PRI;              /* error no such command   */

       }

     if(action)      /* send request or acknowlege user */
         if(ex_send > 0){
            ex_mg.nm_userid = ( long ) io_pkt;
            ex_mg.nm_reply  = 0;
            ex_hd.mh_status  |= MQ_EXOS;
            if(io_pkt){  /* if io_pkt == 0 do not append */
                 append();
                 ch_des[chn].rundn_cnt++;        /* increment rundown count */
               }
            write_port( PORTB, 0);          /* interrupt EXOS */
            return (1);                     /* success */
          }
         else {          /* if ex_send < 0 */
            iosb.cc = ex_send;              /* return errorcode */
            ackuser(io_pkt);
            wmsg_area.ma_lastw = mp;        /* release unused slot */
            nxtwst = &wmsg_area.ma_lastw->nm_u.msg_hd.mh_status;
           }
   else         /* if not action */
     {
      wmsg_area.ma_lastw = mp;
      nxtwst = &wmsg_area.ma_lastw->nm_u.msg_hd.mh_status;
     }
}

/* bcopy() : copy two buffers by count */

 int bcopy( from, to, count)
   char *from, *to;
   int count;
   {
```

```
    for ( ; count > 0; count-- )
      *to++ = *from++;
  }
```

```
/*
 * filename:   RTH.C
 */

/*
 * Code for RTH -> the telnet server on RSX-11M - The different routines
 */

/*
 * DISPATCH --> this routine calls the relevant routine according to the
 *              received telnet command
 */

struct cmd *getcmd();

dispatch(ser)
struct  Telnet_srvr *ser;
{
        register struct cmd *c;

#ifdef  DEBUG
        qio_write("in dispatch",11,040);
#endif

        if(c = getcmd(ser->nm_tsrqst))
           (*c->handler)(ser,0); /* the 2nd param is 0 for do-option routine  */

#ifdef  DEBUG
        qio_write("out dispatch ",12,040);
#endif

}

 /*
  * GETCMD --> this routine searches for the relevant routine according to
  *           the given telnet command
  */

struct cmd *
getcmd(req)
TEXT req;
{
        register int    i;
        register struct cmd *tab = cmdtab;

#ifdef  DEBUG
        qio_write("in getcmd",9,040);
#endif

        for(i=0;i<PTYNO;i++,tab++) {
                if(tab->tsrqst == req) {

#ifdef  DEBUG
                    qio_write("out getcmd ",10,040);
#endif
```

```
                        return(tab);
                    }
                }
            if(i == PTYNO)
                    return(0);
}


 /*
  * TELNET--> this routine sends a message to the EXOS for telnet.
  */

telnet()
{
        register struct packet *p = (struct packet *)io_pkt;
        register struct status *st = pty_status + p->pty_no;

#ifdef  DEBUG
        qio_write("in telnet",9,040);
#endif

        action = 0;     /* assuming we are not sending any request to EXOS */
        if(p->byte_cnt)
                if(st->carrier_on) {
                        if(!st->reply_pending) {
                                p->request = TSWRITE;
                                wr_to_exos(p);  /* write into the wmsg_area */
                                st->reply_pending = 1;/* reply is now pending*/
                                io_pkt = 0; /* so that it is'nt put in the */
                                            /* pending queue of the ACP    */
                           }
                        else {

#ifdef  DEBUG
        qio_write("** SEVERE ERROR ** - pkt from ZT before reply",45,040);
#endif

                                    io_pkt = 0;
                                }
                            }
                else {
        /*
         * If not logged on then packet cannot go to
         * EXOS and hence we give an O/P interrupt and
         * also deallocate the packet from ZT.
         */
                        if(p->moreto_op);
                           out_int(p->pty_no);

#ifdef  DEBUG
        qio_write("pkt from ZT lost as not logged in",33,040);
#endif

                     }
        else {
           /* then it is a dummy packet */
```

```
                if(!st->reply_pending)
               /* then we won't get a write_reply from EXOS so give an O/P int. */
                    out_int(p->pty_no);
                }
        dealoc_b(p,sizeof(struct packet));  /* deallocate packet from ZT */

#ifdef  DEBUG
        qio_write("out telnet ",10,040);
#endif

        return(1);              /* ex_send should always be 1 for telnet */
}       /* end of wr_to_exos */

/*
 * WR_TO_EXOS -- This routine fills up the wmsg_area for telnet
 */

wr_to_exos(p)
struct packet *p;
{

        action = 1;     /* we are sending a request to EXOS */
        ex_tel.mh_length  = sizeof(struct Telnet_srvr) - factor;
        ex_tel.nm_soid    = p->pty_no;
        ex_tel.nm_request = TSCOMMAND;
        ex_tel.nm_tsrqst  = p->request;
        ex_tel.nm_tsdlen  = p->byte_cnt;

        bcopy(p->w_data,ex_tel.nm_tsdata,ex_tel.nm_tsdlen);
}


/*
 * CARON
 */

caron(p)        /* TSCARON/RLCARON */
struct  Telnet_srvr *p;
{
        register struct status  *st = pty_status + p->nm_soid;
        char c = ctrl('C');

#ifdef  DEBUG
        qio_write("in caron ",8,040);
#endif

        if(st->carrier_on)
           return(0);
        else
           if(set_car_on(st->pty_number)){  /* enable unit and set got carrer */
                st->carrier_on = 1;     /* say carrier on */
                qio_zt(p->nm_soid,&c,1);
             }

#ifdef  DEBUG
        qio_write("out caron",9,040);
```

```
#endif

}

 /*
  * BYE
  */

static char *bye_msg = "BYE\r ";

bye(p)          /* TSCAROFF */
struct  Telnet_srvr *p;
{
        register struct status *st = pty_status + p->nm_soid;
        char c = ctrl('C');

#ifdef  DEBUG
        qio_write("in bye ",6,040);
#endif

        if(!st->carrier_on)
           return(0);
        else    {
           st->carrier_on = 0;          /* indicate carrier off */
           qio_zt(p->nm_soid,&c,1);     /* send a ^C first */
           qio_zt(p->nm_soid,bye_msg,4);
        }

#ifdef  DEBUG
        qio_write("out bye",7,040);
#endif

}

 /*
  * ZT_READ
  */

zt_read(p)      /* TSREAD */
struct  Telnet_srvr *p;
{
        register struct status *st = pty_status + p->nm_soid;

#ifdef  DEBUG
        qio_write("in zt_read ",10,040);
#endif

        if(!st->carrier_on)
           return(0);
        else    {

#ifdef  DEBUG
        {
        int i;
        i = 0;
        i = '0' + p->nm_soid;
```

```
        qio_write(&i,2,040);
        }
#endif

           qio_zt(p->nm_soid,p->nm_tsdata,p->nm_tsdlen);
        }

#ifdef  DEBUG
        qio_write("out zt_read",11,040);
#endif

}

 /*
  * WRITE_REPLY
  */

wr_reply(p)             /* TSWRITE (x2h) */
struct  Telnet_srvr *p;
{
        register struct status *st = pty_status + p->nm_soid;

#ifdef  DEBUG
        qio_write("in wr_reply",11,040);
#endif

        if(!st->carrier_on)
          return(0);
        else    {
           if(p->nm_reply == TSERRPENDING)
                return(0);
           else {
                st->reply_pending = 0;
                out_int(p->nm_soid);
              }
          }

#ifdef  DEBUG
        qio_write("out wr_reply ",12,040);
#endif

}

 /*
  * NVTFUNCT
  */

nvtfunct(p)             /* TSNVTFUNCT */
struct  Telnet_srvr *p;
{
        char ch;
        register struct status *st = pty_status + p->nm_soid;

#ifdef  DEBUG
        qio_write("in nvtfunct",11,040);
#endif
```

```
        if(!st->carrier_on)
           return(0);
        else    {
           switch (p->nm_tsdata[0])     {
                case AO-MAXBYTVAL :
                        ch = ctrl('O');
                        break;
                case EC-MAXBYTVAL :
                        ch = BS;
                        break;
                case EL-MAXBYTVAL :
                        ch = ctrl('U');
                        break;
                case IP-MAXBYTVAL :
                        ch = ctrl('C');
                        break;
                case AYT-MAXBYTVAL:
                default:
                        return;
                }       /* end of switch */
           qio_zt(p->nm_soid,&ch,1);
        }       /* end of else */

#ifdef  DEBUG
        qio_write("out nvtfunct ",12,040);
#endif

}       /* end of nvtfunct() */

 /*
  * DO_OPTION
  */

do_option(p,t)          /* TSDOOPTION */
struct  Telnet_srvr *p;
int     t;
{
        static char stadd[2];
        register int i=0;
        register struct status *st = pty_status + p->nm_soid;

#ifdef  DEBUG
        qio_write("in do_option ",12,040);
#endif

        if(!st->carrier_on)
           return(0);
        else    {
           switch (p->nm_tsdata[0]) {
                case TELOPT_BINARY:  {
                        stadd[0] = TC_BIN;
                        if(t) {         /* if t = 1 then it is a dont_option */
                           st->binary_opt = 0;
                           stadd[1] = 0;
                           break;
```

```
                        }
                    else if(!t) {
                            st->binary_opt = 1;
                            stadd[1] = 1;
                            break;
                        }
                    }
                case TELOPT_ECHO:  {
                        stadd[0] = TC_NEC;
                        if(t)  {        /* if t = 1, it is a dont option */
                           st->echo_opt = 0;
                           stadd[1] = 1;
                           break;
                          }
                        else if(!t) {
                                st->echo_opt = 1;
                                stadd[1] = 0;
                                break;
                            }
                        }
                    case TELOPT_SGA:
                    default:
                            return;
                }           /* end of switch */
            qio_smc(p->nm_soid,stadd);
        }       /* end of else */

#ifdef  DEBUG
        qio_write("out do_option",13,040);
#endif

}               /* end of function */

 /*
  * DONT_OPTION
  */

dont_option(p)          /* TSDONTOPTION */
struct  Telnet_srvr *p;
{

#ifdef  DEBUG
        qio_write("in dont_option ",14,040);
#endif

        do_option(p,1);

#ifdef  DEBUG
        qio_write("out dont_option",15,040);
#endif

}

/*
 * HANGUP
 *
```

```
 *              This routine is called from 'request' when a 'BYE' is given
 *      bye the remote user and the 'BYE' task gives a QIO IO.HNG to the ZT
 *      driver which in turn gives a packet to ACP with a func code TS.HNG
 *      and this routine actually sends the request to the board to hangup
 *      the line.
 */

hangup()
{
        register struct packet *p = (struct packet * )io_pkt;
        register struct status *st = pty_status + p->pty_no;

        if(st->carrier_on){
                p->request = TSHANGUP;
                wr_to_exos(p);
                st->carrier_on = 0;     /* drop carrier */
              }
        else
           action = 0;
        io_pkt = 0;
        dealoc_b(p,sizeof(struct packet));  /* deallocate packet from ZT */
        return(1);
}

/*
 * This ends the code for RTH
 */
```

```
/*
 * filename:    SETUP.C
 */

/*
 * exsetup:
 *      - setup message queue
 *      - send init message to exos
 *      - analyse board response
 */

extern int zeint;
extern long reloc();

#ifdef  UNIBUS
        extern int *umradd;
        extern long unilbuf;    /* 18-bit unibus address for local pool */
        extern long uni_msg;    /* 18-bit unibus address for msg area */
        extern long phy_buf;
#endif

int exsetup(mode)
    int mode;
{
        struct  rmsg_area       *rmsgarea;
        struct  wmsg_area       *wmsgarea;
        register struct msg     *current,       *next;
        long    addr;
        long    r_base, w_base;
        Uchar   *ap,    init_addr[8];
        int     err,    timeout;
        register struct init_msg        *im;
        int     i;
        Uint    Xceiver;

        rmsgarea = &rmsg_area;
        wmsgarea = &wmsg_area;

        r_base = reloc(rmsgarea)        /* rmsgarea base segment addr */
#ifdef  UNIBUS
        ;               /* for UNIBUS the 18-bit addr is 16-byte aligned */
#else
        & 0x3FFFF0;     /* in Q-bus make phy-addr 16-byte aligned */
#endif

        w_base = reloc(wmsgarea)        /* wmsgarea base segment addr */
#ifdef  UNIBUS
        ;               /* for phy-addr need not be 16-byte aligned */
#else
        & 0x3FFFF0;     /* for Q-bus it must be 16-byte aligned  */
#endif

                /* link together the read "exos to host" message queue */

        rmsgarea->ma_rlink = (Ushort)( reloc(rmsgarea->ma_rmsgs) - r_base);
```

```
            /* exos link to read queue */

        current = (struct msg *) (&rmsgarea->ma_rmsgs[NET_RBUFS-1]);
        rmsgarea->ma_lastr = rmsgarea->ma_rmsgs;
        nxtrst = &rmsgarea->ma_lastr->nm_u.msg_hd.mh_status;
        for(i=0;i<NET_RBUFS;i++) {
          next =(struct msg *)( &rmsgarea->ma_rmsgs[i]);
          current->nm_u.msg_hd.mh_link = (Ushort)(reloc(next) - r_base);
          current->nm_u.msg_hd.mh_length = sizeof(union exos_u)
                                           - sizeof( struct headers);
          current->nm_u.msg_hd.mh_status =3;
          current->msg_link = next;
          current = next;
        }

        /* link together the write "host to exos" message queue */

        wmsgarea->ma_wlink = (Ushort)( reloc(wmsgarea->ma_wmsgs) - w_base );

        current = (struct msg *) (&wmsgarea -> ma_wmsgs[NET_WBUFS-1]);
        wmsgarea->ma_lastw = wmsgarea -> ma_wmsgs;
        nxtwst = &wmsgarea->ma_lastw->nm_u.msg_hd.mh_status;
        for (i=0;i < NET_WBUFS;i++) {
          next = (struct msg *) (&wmsgarea-> ma_wmsgs[i]);
          current->nm_u.msg_hd.mh_link = (Ushort)(reloc(next) - w_base);
          current->nm_u.msg_hd.mh_length = sizeof (union exos_u) -
                                           sizeof( struct headers );
          current->nm_u.msg_hd.mh_status = 0;
          current -> msg_link = next;
          current = next;
        }

/* setup initialization message */

        im =  ex_db.ex_imsg;
        clear(im,sizeof(struct init_msg));      /* clear the init_msg area */
        im -> im_newstyle = 1;                  /* use new style message   */
        im -> im_result = 0xFF;                 /* reserved                */
        im -> im_mode = mode & 0x07F;           /* setup mode              */
        im->im_hdfo[0]=im->im_hdfo[1] = 1;      /* do auto-byte/word swapping*/
        im -> im_addrmode = 3;                  /* absolute address mode   */

        /* data order test patterns */

        im -> im_byteptn[0] = 1;
        im -> im_byteptn[1] = 3;
        im -> im_byteptn[2] = 7;
        im -> im_byteptn[3] = 0XF;
        im -> im_wordptn[0] = 0X103;
        im -> im_wordptn[1] = 0X70F;
        im -> im_longptn    = 0X103070F;

        im -> im_101off = im -> im_101seg = 0XFFFF;
        im -> im_nhosts = 1;
        im -> im_result = im -> im_nmb = im -> im_nproc = im -> im_nslots=0XFF;
```

```
        im -> im_h2exqaddr =
#ifdef  UNIBUS
        uni_msg + (w_base - r_base);
#else
        w_base;
#endif

        /* 22 bit physical base address */

        im->im_h2exoff = (Ushort)(reloc(&wmsgarea->ma_wlink) - w_base);

        /* 16 bit physical address */

        im -> im_h2extype = 0;          /* polled by EXOS */
        im -> im_h2exaddr = 0;

        im -> im_ex2hqaddr =
#ifdef  UNIBUS
        uni_msg;
#else
        r_base;
#endif

        /* 22 bit physical base address */
        im->im_ex2hoff = (Ushort)(reloc(&rmsgarea->ma_rlink) - r_base);

        im -> im_ex2htype = 4;          /* bus vectored interrupt */
        im -> im_ex2haddr = ((long) zeint << 16);  /* interrupt address */
        /* the address is shifted 16 bit so that lower word remains zero */

        /* init message initialization is complete */

        /* reset exos by writing onto port A; then after 2 secs
           check the status and report an error         */

        write_port(PORTA,0);
        delay(2,'s');   /* wait for 2 secs for successful initialization */
        for(;;){
           if(((Xceiver = read_port(PORTB)) & PB_ERROR) == 0){
                                          /* check if success bit is clear */
              if(mode & 0x80)   /* if infinite timeout is requested */
                continue;
              else
                return( PB_ERROR);
            }
           else
                break;
          }

        init_addr[0] = init_addr[1] = -1;       /* move FF */
        init_addr[2] = init_addr[3] = 0 ;       /* move  0 */
        addr = reloc(ex_db.ex_imsg);            /* int_addrs[0..3] is init
                                                        as 0XFFFF0000 */
#ifdef  UNIBUS
        {
           unsigned int *p = (int )&addr;
```

```
            *umradd++ = *++p;     /* use the first UMR of the pool and load it */
            *umradd-- = *--p;
            addr = unilbuf;       /* 18-bit address */
         }
#endif

         for(i = 0; i<4; i++) {
            init_addrs[i+4] = addr;
            addr >>= 8;
         }

         /* write the init_addrs to port B preceded by 0XFFFF0000 */

#ifdef DEBUG
        qio_write("init",5,040);
#endif
         for ( i = 0; i < 8; i++ ){
            timeout = 100000;
            while((read_port(PORTB) & PB_READY)&& timeout--)
              if(timeout == 0){
                 if(mode & 0x80){         /* is infinite timeout requested */
                    timeout = 100000;
                    continue;
                   }
                 return(read_port(PORTB));
               }
            write_port(PORTB,((init_addr[i])&0XFF));
         }

#ifdef DEBUG
        qio_write ("over",5,040);
#endif
         delay(2,'s');
         for(;;){
            if(im->im_result){
                 if(mode & 0x80){
                    delay(2,'s');
                    continue;     /* infinite timeout */
                   }
                 ex_db.ex_init = 0;
                 break;
               }
            else {
                 ex_db.ex_init = 1;
                 break;
               }
           }

#ifdef  UNIBUS
         {
            unsigned int *p = (int )&phy_buf;

            *umradd++ = *++p;     /* restore 1st UMR */
            *umradd-- = *--p;
         }
```

```
#endif

        im->im_dummy2 = Xceiver;        /* error status of Xceiver cable */
        return(im->im_result);
}
```

```
/*
 * filename: SIGNALOOB.C
 */

int fin_pen(x)
    int x;
  {
    register struct iopkt *pkt, *prev;
    int fn_code,b,c,ch_no;

    prev = 0;
    pkt = io_pend;

    if(x == SA_USL)
        fn_code = IO_ACS|SA_SEL;
    else
        fn_code = IO_ACS|SA_URG;
    c = chn;

    while ( pkt ){
        if(x == SA_URG){
            b = pkt->i_prm.i_prm4;
            c = ex_oob.nm_soid;
          }
        else
           b = pkt->i_prm.i_prm6;

        if((pkt->i_fcn == fn_code) && (b == c)){
           if(x == SA_USL)
              if(pkt->i_prm.i_prm5 & NOREPLY){
                pkt->i_prm.i_prm5 |= UNSELECT;  /* set it unselect */
                prev = pkt;
                pkt = pkt->i_lnk;
                continue;
              }
           if ( prev )
                prev->i_lnk = pkt->i_lnk;
           else
                io_pend = pkt->i_lnk;
           if((x == SA_USL) || (x == SA_ROO)) /* only for SA_USL and SA_ROO */
                pkt->i_ast = 0;         /* see that ast is not entered */
           ch_no = pkt->i_prm.i_prm6;   /* get the channel nmumber */
           if(x == SA_URG)
                iosb.nread = ch_no;             /* return channel number in iosb*/
           ch_des[ch_no].rundn_cnt--;   /* rundown the I/O       */
           ackuser( pkt );
         }
        else
           prev = pkt;
        pkt = pkt->i_lnk;
     }
 }
```

```
/*
 *      filename:       UNIACP.C
 */
/*
 *      This file contains the 'C' code for incorporating ACP on a UNIBUS M/C
 */
/*
 *      UNI_INI
 *
 *              This routine is called for initializing the unibus related
 *              stuff. It calls a macro routine to assign the UMR's.
 */

uni_ini()
{
        srex();         /* specify exit ast for cleanup of UMR's */
        clear(pool_im,sizeof pool_im);
        rel_pool();     /* initialize relocated address of pool */
        if(!ass_umr()) {
           qio_write("** FATAL ** - NO UMR'S AVAILABLE",32,040);
           exit_();
         }
                        /* call a macro routine to assign 3 UMR's for pool
                         * area and the message area and also load them and
                         * save the physical UNIBUS address (18-bit) in a
                         * global area.
                         */
}

/*
 *      GETPOOL
 *
 *              This routine gets a free buffer from the pool and allocates
 *              it for the requester. This returns the 18-bit UNIBUS address
 *              of the allocated slot. If allocation fails then the packet is
 *              put in a secondary queue and action is set to '0' so that the
 *              board does not get any message for the time being.
 */

long
getpool(pkt,st)
struct iopkt *pkt;
Ushort st;
{
        register struct pool_im *pl = pool_im;
        struct rel_addr tmp_addr;
        int i;

        for(i=0;i < POOL_BUFS;i++,pl++)
           if(pl->state != ALLOCATED) {
              pl->owner = pkt;
              pl->state = ALLOCATED;
              if(st) {          /* if it is a write request then do Xfering */
                if(i <= 7) {    /* is it within 1st 4KW ?       */
                  tmp_addr.rel_bias = rellbuf.rel_bias;
```

```
                tmp_addr.dis_bias = rel1buf.dis_bias + (POOLBUFSIZE * i);
               }
              else {
               tmp_addr.rel_bias = rel2buf.rel_bias;
               tmp_addr.dis_bias = rel2buf.dis_bias + (POOLBUFSIZE * (i-8));
               }
              acopy(&pkt->i_prm.i_buf,&tmp_addr.rel_bias,pkt->i_prm.i_cnt);
             }
            break;
           }
        if(i == POOL_BUFS) {     /* if no pool available */
           action = 0;           /* donot send anything to the board */
           pkt->i_lnk = sec_que; /* put the pkt on top of the sec que */
           sec_que = pkt;
           return(0);
         }
        if(i <= 7)
                return(uni1buf + (POOLBUFSIZE * i));
        else
                return(uni2buf + (POOLBUFSIZE * (i - 8)));
}

/*
 * PUT_SEC_QUE
 */

/*
 * Puts the secondary que on the top of the internal queue in the reverse
 * order i.e. the last element of the sec queue will finally be on top of
 * the internal queue.
 */

put_sec_que()
{
        register struct iopkt *tmp;

        while(sec_que) {
                tmp = sec_que->i_lnk;
                sec_que->i_lnk = int_que;
                int_que = sec_que;
                sec_que = tmp;
             }
}

/*
 *      FREEPOOL
 *
 *              This routine frees the allocated pool and also Xfers the data
 *              which has arrived from the board to the user area.
 */

freepool(pkt,st)
struct iopkt *pkt;
Ushort st;
{
        register struct pool_im *pl =  pool_im;
```

```
        struct rel_addr tmp_addr;
        register int i;

        for(i=0;i < POOL_BUFS;i++,pl++)
           if(pl->owner == pkt) {
              pl->state = DEALLOCATED;
              pl->owner = 0;
              break;
            }
        if(st) {
           if(i <= 7) {
              tmp_addr.rel_bias = rellbuf.rel_bias;
              tmp_addr.dis_bias = rellbuf.dis_bias + (POOLBUFSIZE * i);
             }
           else {
             tmp_addr.rel_bias = rel2buf.rel_bias;
             tmp_addr.dis_bias = rel2buf.dis_bias + (POOLBUFSIZE * (i - 8));
            }
           acopy(&tmp_addr.rel_bias,&pkt->i_prm.i_buf,pkt->i_prm.i_cnt);
                        /* Xfer read data from pool to the user buffer */
         }
}
```

```
; FILEMANE:     ACPUCB.MAC
;
; ACPUCB: -->   This routine searches the DCB list and picks up the ZE device
;               DCB. It then moves the TCB address of the ACP(current) task's
;               TCB address to the U.ACP field of each UCB of the device.
;               As it manipulates the system database it first switch itself
;               to system state such that all other processes are lock, by
;               calling to $SWSTK routine.
;
;               R0 returns the completion code 0 --> unsucess 1 --> success
;

;
;
        .TITLE  ACPUCB
        .IDENT  /01/


;
;
;
;       SYSTEM MACRO CALLS
;

        .MCALL  UCBDF$,DCBDF$
        UCBDF$
        DCBDF$


C$SPRT=0

        .PSECT  C$TEXT,I,RO

ACPUCB::                        ; global reference label

        .IF DF C$SPRT

        JSR     R5,C$SAV        ; make it 'C' callable
        MOV     R5,-(SP)        ; save C frame pointer

        .ENDC

        CALL    $SWSTK,RET      ;; switch to system state and return to user
                                ;; state at RET after execution of RETURN
        CLR     SUCC            ;; indicate unsuccessful
        MOV     #$DEVHD,R2      ;; set pointer to the first DCB
20$:    MOV     (R2),R2         ;; get next DCB address
        BEQ     60$             ;; no more DCB exit, it is unsuccessful exit
        CMP     #"ZE,D.NAM(R2)  ;; is it ZE device ?
        BNE     20$             ;; if NE no;; go for next DCB
        INC     SUCC            ;; indicate success
        MOVB    D.UNIT+1(R2),R3 ;; get number of UCBs (units)
        MOV     D.UCBL(R2),R4   ;; get size of the UCB
        MOV     D.UCB(R2),R2    ;; get first UCB address in R2
40$:    MOV     $TKTCB,U.ACP(R2);; get ACP(current) task TCB address
        CLR     U.CW2(R2)       ;; clear user characteristics word
```

```
        .IF DF  UNIBUS

        MOV     U.ACP+2(R2),PHY.BUF     ;; higher order address
        MOV     U.ACP+4(R2),PHY.BUF+2   ;; lower order address
        MOV     R2,ZEUCB                ;; save UCB address

        .ENDC

        ADD     R4,R2           ;; get next UCB address
        DEC     R3              ;; decrement UCB count
        BPL     40$             ;; if PL(us) more UCB
60$:    RETURN                  ;; switch to user state

RET:    MOV     SUCC,R0         ;return result in R0

        .IF DF  C$SPRT

        MOV     (SP)+,R5        ; adjust frame pointer
        JMP     C$RET           ; return to caller

        .IFF

        RETURN

        .ENDC


SUCC:   .BLKW   1

        .PSECT  C$TEXT,I,RO
        .EVEN
        .END
```

```
;
;  filename:    DQPKT.MAC
;
;
;
;  This routine dequeues a pakcet from the listhead of the ACP task. It first
;  switches to system state before dequeueing. The address of the dequeued
;  packet is returned in R0 making it callable from C.
;

C$SPRT=0                ; this routine becomes callable from a C routine

        .MCALL  TCBDF$, UCBDF$
        TCBDF$
        UCBDF$

        .TITLE  DQPKT
        .IDENT  /01/
        .PSECT  c$text,i,ro

IOPKT:  .BLKW   1               ; local variable to hold I/O packet address

DQPKT::

        .IF DF  C$SPRT

        JSR     R5,c$sav        ;  save register R2-R5 and adjust stack
        MOV     R5,-(SP)        ;  save R5 i.e frame pointer of C routine

        .ENDC


        CLR     IOPKT           ;  clear I/O packet address

        SWSTK$  USR             ;; switch to system state to lockout other
                                ;; processes
        MOV     $TKTCB,R0       ;; get ACP(our) TCB address
        ADD     #T.RCVL,R0      ;; get receive queue listhead
        CALL    $QRMVF          ;; attempt to dequeue packet
        BCS     20$             ;; if CS no pakcet
        MOV     R1,IOPKT        ;; return address of I/O packet
        BR      60$             ;; return
20$:    MOV     NXTRST,R2       ;; get pntr to status field of reply Q
        BEQ     40$             ;; initially the ptr is null and since
                                ;; there is no job for acp - sleep
        BITB    #3,(R2)         ;; check ownership
        BEQ     60$             ;; if EQ owner=host, process reply
        TST     INT.QUE         ;; check if anything pending in internal Q
        BEQ     40$             ;; if EQ nothing, then sleep
        MOV     NXTWST,R2       ;; check availibility of free slot
        BEQ     40$             ;; initially ptr is null so sleep since no job
        BITB    #3,(R2)         ;; check ownership
        BEQ     60$             ;; if EQ slot available, procees request
40$:    JMP     $STPCT          ;; go to sleep
60$:
        RETURN                  ;; return to user state
```

```
USR:    MOV     IOPKT,R0        ; return I/O packet address in R0

        .IF DF C$SPRT

        MOV     (SP)+,R5        ;  restore frame pointer of the C routine
        JMP     c$ret           ;; unsave register and adjust stack & return

        .IFF

        RETURN

        .ENDC

;
;
; ACCKUSER : this is a C callable routine, which will issue a $IOFIN
;            to inform the requesting task of IO completion. This is
;            only compatable with C function call.
;
; C function:
;
;   ackuser(io_pkt)
;     struct iopkt *io_pkt;
;
;   IOSB is the address of the IOSB
;
;

ACKUSER::

        .IF  DF C$SPRT

        JSR     R5,c$sav        ; save register and adjust stack
        MOV     R5,-(SP)        ; save frame pointer
        MOV     4(R5),R3        ; move address of I/O packet

        .ENDC

        MOV     R3,R0           ; move address of I/O pkt in R0
        MOV     R3,IOPKT        ; save I/O pkt addr
        MOV     I.UCB(R3),R5    ; move address of UCB in R5
        ADD     #I.PRM,R0       ; R0 now points to parameter block
        MOV     #10,R1          ; clear 8 words in param block
10$:    CLR     (R0)+           ; clear parameter word
        DEC     R1              ; decrement loop count
        BNE     10$

        CALL    $SWSTK,RET      ; switch to system state
        MOV     IOSB,R0         ; move first word of IOSB
        MOV     IOSB+2,R1       ; move second word of IOSB
        MOV     IOPKT,R3        ; get I/O pkt addr
        CALL    $IOFIN          ;; complete io process

        RETURN                  ; return to task state
RET:
```

```
        .IF DF  C$SPRT

        MOV     (SP)+,R5        ; restore frame pointer
        JMP     C$RET           ; return to the caller

        .IFF

        RETURN

        .ENDC

;
;
; This is a 'C' callable routine, which returns the absolute
; physical address of an input virtual address.
;
; long reloc(v_addr)
;   Ushort v_addr;
;
; This routine is also callable from macro, input outputs are
;
; INPUT:  R0 -> virtual address
;
; OUTPUT: R0 -> higher order address word
;         R1 -> lower order address word
;
;

RELOC::
        .IF DF C$SPRT

        JSR     R5,C$SAV        ; save all register
        MOV     R5,-(SP)        ; save frame pointer
        MOV     4(R5),R0        ; get address parameter

        .ENDC

        CALL    $RELOC          ; relocate virtual address
        BIC     #160000,R2      ; mask out APR index and get displacement
        MOV     R1,R0           ; get relocation bias in R0
        ASH     #-12,R0         ; get upper 6 (out of 22) bits in R0
        BIC     #177700,R0      ; mask other 10 bits
        ASH     #6,R1           ; get upper 10 bits of lower 16 bits in R1
        BIS     R2,R1           ; append lower 6 bit offset

        .IF DF C$SPRT

        MOV     (SP)+,R5        ; restore frame pointer
        JMP     C$RET           ; restore all register and return

        .IFF

        RETURN

        .ENDC
```

```
;
; this is a "C" callable routine which returns the absolute physical
; address of an input pointer to a relocated address.
;
; long  absadr( reladr )
;   struct rel_addr *reladr;
;
; this routine is also callable from macro with input & output as
;
; INPUT:  R0 -> pointer to the relocated address
; OUTPUT: R0 -> higher order physical address
;         R1 -> lower order physical address
;
;

ABSADR::
        .IF DF C$SPRT

        JSR     R5,C$SAV                ; save all registers
        MOV     R5,-(SP)                ; save frame pointer
        MOV     4(R5),R0                ; get the input parameter

        .ENDC

        MOV     (R0),R1                 ; get relocation bias in R1
        MOV     2(R0),R2                ; get displacement bias in R1
        BIC     #160000,R2              ; mask out the APR index
        MOV     R1,R0                   ; get relocation bias in R0
        ASH     #-12,R0                 ; get lower 6 bits of higher order adr
        BIC     #177700,R0              ; mask out remaining bits
        ASH     #6,R1                   ; get upper 10 bits of lower address
        BIS     R2,R1                   ; append lower 6 bit offset( displa)

        .IF DF C$SPRT

        MOV     (SP)+,R5                ; retore frame pointer
        JMP     C$RET                   ; restore all register and return

        .IFF

        RETURN

        .ENDC


;
; this is a 'C' callable routine to get the privilege info of a task
;
; int getpriv( tcb)
;   int tcb;            /* tcb address of the task */
;
```

```
; if called from macro , input & outputs are
;   INPUT  : R3 -> tcb address of the task
;   OUTPUT : R0 -> = 1 if priv else clear
;

GETPRIV::
        .IF DF C$SPRT

        JSR     R5,C$SAV                ; save all register
        MOV     R5,-(SP)                ; save frame pointer
        MOV     4(R5),R3                ; get tcb address

        .ENDC

        CLR     R0                      ; assume non-privilege
        BIT     T.ST3(R3),#T3.PRV       ; test privilege bit
        BEQ     20$                     ; if EQ then task is non-privileged
        MOV     T.UCB(R3),R2            ; get the ucb of 'ti:'
        BIT     U.CW2(R2),#U2.PRV       ; test privilege bit
        BNE     10$                     ; if NE then privileged
        MOV     U.DCB(R2),R2            ; get 'TI:' DCB
        CMP     #"CO,D.NAM(R2)          ; is it the console?
        BNE     20$                     ; if NE then no, so non-privileged
10$:
        INC     R0                      ; output privilege
20$:
        .IF DF C$SPRT

        MOV     (SP)+,R5                ; restore frame pointer
        JMP     C$RET                   ; restore register and return

        .IFF

        RETURN

        .ENDC


        .PSECT  C$TEXT,I,RO

        .EVEN
        .END
```

```
;
; filename:     RTHMAC.MAC
;
; This file contains all the C - callable routines written in MACRO-11 assembly
; language
;

        .TITLE  RTHMAC
        .IDENT  /01/

        .MCALL  UCBDF$,PKTDF$,DCBDF$,SCBDF$,TCBDF$
        UCBDF$  ,,TTDEF
        PKTDF$
        DCBDF$
        SCBDF$
        TCBDF$

IO.INP = 5400
IO.OUT = 6000

;
; OUT.INT --> This routine gives an O/P interrupt to ZTDRV
;

        .psect  c$text,i,ro
        .MCALL  ALUN$S,QIO$S,QIOW$S

OUT.INT::
        jsr     R5,c$sav

        MOV     4(R5),R2        ; get pty_no_first parameter
        ALUN$S  #7,#"ZT,R2
        QIO$S   #IO.OUT,#7,,,,,

        jmp     c$ret

;
; QIO.ZT --> This routine does a QIO IO.INP to ZTDRV which simulates an
;            I/P interrupt.
;

QIO.ZT::
        jsr     R5,c$sav

        MOV     4(R5),R0        ; pty_no
        MOV     6(R5),R1        ; buffer ptr to be o/p
        MOV     10(R5),R2       ; length of buffer
        ALUN$S  #7,#"ZT,R0
        QIOW$S  #IO.INP,#7,#1,,,,<R1,R2>

        jmp     c$ret           ; return to caller

;
; DEALOC.B --> This routine deallocates a packet back to the system pool
;
```

```
DEALOC.B::
        jsr     R5,c$sav
        MOV     R5,-(SP)

        MOV     6(R5),R1        ; size of pkt to be deallocated
        MOV     4(R5),R0        ; address of that pkt
        CALL    $DEACB          ; deallocate pkt back to the system pool
                                ; also return to task state
        MOV     (SP)+,R5
        jmp     c$ret           ; return to caller


;
; QIO.WRITE --> This routine writes to the tewrminal
;

QIO.WRITE::
        jsr     R5,c$sav
        MOV     R5,-(SP)

        MOV     4(R5),R0        ; buffer pointer
        MOV     6(R5),R1        ; buffer length
        MOV     10(R5),R2       ; vertical format character
        QIOW$S  #IO.WLB,#5,#1,,,,<R0,R1,R2>

        MOV     (SP)+,R5
        jmp     c$ret

;
; QIO.SMC --> This routine does a QIO SF.SMC to ZTDRV to set and reset terminal
;             options.
;

QIO.SMC::
        jsr     R5,c$sav
        MOV     R5,-(SP)

        MOV     4(R5),R1        ; pty number
        MOV     6(R5),R2        ; address of buffer

        ALUN$S  #7,#"ZT,R1
        QIOW$S  #SF.SMC,#7,#1,,,,<R2,#2>

        MOV     (SP)+,R5
        jmp     c$ret

        .psect  c$data,d,rw

RTVAL:
        .WORD   0

        .psect  c$text,i,ro

SET.CAR.ON::

        jsr     R5,c$sav
```

```
        MOV     R5,-(SP)

        MOV     4(R5),R0        ; pty number

        SWSTK$  30$             ;; switch to system state
        MOV     #$DEVHD,R2      ;; start of device tables
10$:
        MOV     (R2),R2         ;; get next DCB
        BEQ     20$             ;; if EQ device not in system
        CMP     #"ZT,D.NAM(R2)  ;; is it the 'ZT' device?
        BNE     10$             ;; if NE no, keep searching
        MOV     D.UCBL(R2),R1   ;; get length of UCB
        MOV     D.UCB(R2),R2    ;; get address of first UCB
        MUL     R0,R1           ;; get offset to the correct UCB in R1
        ADD     R1,R2           ;; get UCB address in R2
        BICB    #US.DSB!US.CRW,U.STS(R2) ;; enable unit and not waiting for car.
        MOV     #1,RTVAL        ;; return sucess
        RETURN                  ;; return to user state at 30$
20$:
        MOV     #0,RTVAL        ;; indicate failure as ZT device not found
        RETURN                  ;; return to user state
30$:
        MOV     RTVAL,R0        ; return value

        MOV     (SP)+,R5        ; restore frame pointer
        jmp     c$ret

        .psect  c$text,i,ro
        .even
        .psect  c$data,d,rw
        .even
        .END                    ; end of file RTHMAC.MAC
```

```
;
; filename:     RWPORT.MAC
;
; NAME:
;       read.port, write.port -- read and write from the port
;
; SYNOPSIS:
;       int read_port(PORT)
;           int PORT;
;
;       int write_port(PORT,value)
;           int PORT;
;
;           value;
;
; FUNCTION:
;       read_port reads the specified port and returns the value
;
;       write_port writes the given value into the specified address.
;
;

        .TITLE  RWPORT
        .IDENT  /01/

IOPAGE = 160000
C$SPRT = 0

        .PSECT  EX$RWI,RO

READ.P::                        ; read port entry point

        .IF DF C$SPRT

        JSR     R5,C$SAV        ; save registers if C interface
        MOV     4(R5),R1        ; get port address in I/O page in R1

        .ENDC

        MOVB    IOPAGE(R1),R0   ; read a byte from port in R0

        .IF DF C$SPRT

        JMP     C$RET           ; restore register

        .IFF

        RETURN                  ; return to caller

        .ENDC


WRITE.::                        ; write port entry point

        .IF DF C$SPRT
```

```
        JSR     R5,C$SAV        ; save all register in C environment
        MOV     4(R5),R1        ; get port address in I/O page in R1
        MOVB    6(R5),R0        ; move a byte value in R0

        .ENDC

        MOVB    R0,IOPAGE(R1)   ; write a byte into port

        .IF DF C$SPRT

        JMP     C$RET           ; restore register and return

        .IFF

        RETURN                  ; return to caller

        .ENDC

        .PSECT  RWPORT,I,RO
        .EVEN
        .END
```

```
        .TITLE  SCOPY
        .IDENT  /01/

        .PSECT  C$TEXT,I,RO


C$SPRT=0
;
; SCOPY: this routine copies user soictl buffer into a global
;        buffer of acp. this routine is "C" callable as
;
;        scopy( from, count)
;          struct rel_addr *from;       /* pointer to source relocated addr */
;          int count;                   /* byte count          */
;


FROM:   .BLKW   1
TO:     .BLKW   1
COUNT:  .BLKW   1


SCOPY::                                 ; scopy entry point

        .IF DF C$SPRT

        JSR     R5,C$SAV        ; save all register
        MOV     R5,-(SP)        ; save frame pointer
        MOV     4(R5),FROM      ; get source relocated addr pointer
        MOV     6(R5),COUNT     ; get byte count

        .ENDC

        CALL    $SWSTK,RET      ;; switch to system state
        MOV     #PARAM,RO       ;; load RO with the acp buffer
        CALL    $RELOC          ;; relocate the destination address
        MOV     R1,R3           ;; move dest relocation bias to R3
        MOV     R2,R4           ;; move dest displacement bias to R4
        MOV     FROM,RO         ;; get pointer to source relocated addr
        MOV     (RO)+,R1        ;; move source relocation bias
        MOV     (RO),R2         ;; move source disp bias ( in terms of APR6 )
        ADD     #120000-140000,R2       ;; make it APR5 bias
        MOV     COUNT,RO        ;; move byte count
        CALL    $BLXIO          ;; move data
        RETURN                  ;; return to task state

RET:
        .IF DF C$SPRT

        MOV     (SP)+,R5        ; restore frame pointer
        JMP     C$RET           ; restore register and return

        .ENDC
```

```
        .PSECT  C$TEXT,I,RO
        .EVEN
        .END
```

```
        .TITLE  UCOPY
        .IDENT  /01/

        .PSECT  C$TEXT,I,RO


C$SPRT=0
;
; UCOPY: this routine copies user soictl buffer from the global
;        buffer of acp. this routine is "C" callable as
;
;        ucopy( from, to, count)
;          char *from;                   /* pointer to source buffer */
;          struct rel_addr *to;          /* pointer to dest relocated addr */
;          int count;                    /* byte count          */
;


FROM:   .BLKW   1
TO:     .BLKW   1
COUNT:  .BLKW   1


UCOPY::                                 ; scopy entry point

        .IF DF C$SPRT

        JSR     R5,C$SAV        ; save all register
        MOV     R5,-(SP)        ; save frame pointer
        MOV     4(R5),FROM      ; get source addr pointer
        MOV     6(R5),TO        ; get dest relocted addr pointer
        MOV     10(R5),COUNT    ; get byte count

        .ENDC

        CALL    $SWSTK,RET      ;; switch to system state
        MOV     FROM,R0         ;; load R0 with the source buf
        CALL    $RELOC          ;; relocate the source address
        ADD     #120000-140000,R2       ;; make it APR5 bias
        MOV     TO,R0           ;; get pointer to dest relocated addr
        MOV     (R0)+,R3        ;; move destination relocation bias
        MOV     (R0),R4         ;; move dest disp bias ( in terms of APR6 )
        MOV     COUNT,R0        ;; move byte count
        CALL    $BLXIO          ;; move data
        RETURN                          ;; return to task state

RET:
        .IF DF C$SPRT

        MOV     (SP)+,R5        ; restore frame pointer
        JMP     C$RET           ; restore register and return

        .ENDC
```

```
        .PSECT  C$TEXT,I,RO
        .EVEN
        .END
```

```
UNIBUS = 1
```

```
.NLIST  SYM
.NLIST CND
;
; filename:     UNIACP.MAC
;
;               This file contains all the macro routines for incorporating
;               the ACP on a UNIBUS machine.

        .TITLE  UNIMAC
        .IDENT  /01/

;
; ASS.UMR
;               this routine assigns 3 UMR's for the pool and the message area
;               and also loads them and also saves the unibus addresses in
;               some global area so that they can be accessed by other routines.
;

        .MCALL  SCBDF$,UCBDF$
        SCBDF$  ,,SYSDEF
        UCBDF$


        C$SPRT  = 1

        .IF DF  R$$MPL

        S.UNI = S.EMB + 2

        .IFF    ;R$$MPL

        S.UNI = S.FRK + 14

        .ENDC   ;R$$MPL

        SCBDF$

        .psect  c$text,i,ro

ASS.UMR::

        .IF DF  C$SPRT

        jsr     R5,c$sav
        MOV     R5,-(SP)

        .ENDC

        MOV     ZEUCB,R4                ; get UCB address
        MOV     U.SCB(R4),R4            ; get SCB address
        MOV     #10,S.UNI+M.UMRN(R4)    ; no. of UMR's to be allocated
        MOVB    PHY.BUF,S.UNI+M.BFVH(R4); higher order physical address
        MOV     PHY.BUF+2,S.UNI+M.BFVL(R4) ; lower order address
        MOV     #S.UNI,R0               ;
        ADD     R4,R0                   ; point to UMR mapping table
        CALL    .AS.UMR                 ; assign the two UMR's
```

```
TST     SUCC                    ; was it successful?
BEQ     FAILS                   ; if EQ then no
MOV     M.UMVL(R0),UNI1BUF+2    ; save lower order unibus address
MOVB    M.UMVH(R0),UNI1BUF      ; save higher order word
MOV     UNI1BUF,R3              ; get higher order address
MOV     R3,R4                   ; copy higher order address
ASH     #-4,R4                  ; shift bits 4 and 5 to 0 and 1
MOV     R4,UNI1BUF              ; restore the high order address
MOV     UNI1BUF+2,R4            ; lower order address
BIC     #177717,R3              ; mask all but bits 4 & 5 in high order
ASR     R3                      ; get bits 4 & 5 into 3 & 4
ASH     #-13.,R4                ; high 3 bits in low 3 bits of low order
BIC     #177770,R4              ; mask remaining bits
BIS     R4,R3                   ; append bits 0,1 & 2 of LO to 3 & 4- HO
INC     R3                      ; get next UMR nnumber
MOV     R3,R4                   ; save R3 in R4
MOV     UNI1BUF+2,R2            ; get lower order address
ASH     #13.,R3                 ; get lower 3 bits in upper 3
BIC     #017777,R3              ; mask out rest of the bits
BIC     #160000,R2              ; mask high 3 bits in lower order addr
BIS     R2,R3                   ; final lower order address in R3
ASH     #-3,R4                  ; get bits 3 & 4 in 0 and 1
MOV     R4,UNI2BUF              ; higher order address 2 bits
MOV     R3,UNI2BUF+2            ; lower order address 16 bits

MOV     M.UMRA(R0),R1           ; get address of 1st UMR
MOV     R1,UMRADD               ; save this address for further use
MOV     PHY.BUF+2,R3            ; save lower order address
MOV     PHY.BUF,R2              ; higher order address
MOV     R3,(R1)+                ; load lower order address
MOV     R2,(R1)+                ; load higher order address
ADD     #20000,R3               ; add an equ. of 4KW
ADC     R2                      ;
MOV     R3,(R1)+                ; load lower order address of next 4kw
MOV     R2,(R1)+                ; higher order address of next 4kw

MOV     #12.,R1                 ; size of UMR ass. block
CALL    $ALOCB                  ; allocate it from the system pool
BCS     FAILS                   ; if CS then no system pool available

MOV     R0,UMRMSG               ; save ptr to ass. block
MOV     #4,M.UMRN(R0)           ; No. of UMR's to assign * 4
MOV     R0,-(SP)                ; save R0

MOV     #RMSG.A,-(SP)           ; 1st parameter
CALL    RELOC                   ; call a 'C' callable macro routine
                                ; which returns the physical address
TST     (SP)+                   ; pop stack
MOV     (SP)+,R2                ; unsave pointer to UMR ass. block
MOVB    R0,M.BFVH(R2)           ; higher order physical address
MOV     R1,M.BFVL(R2)           ; lower order physical address
MOV     R2,R0                   ; restore R0

CALL    .AS.UMR                 ; assign the UMR
TST     SUCC                    ; was it successful ?
BEQ     FAILS                   ; if EQ no
```

```
        MOV     M.UMVL(R0),UNI.MSG+2    ; lower order unibus address
        MOVB    M.UMVH(R0),R4           ; higher order unibus address
        ASH     #-4,R4                  ; shift bits 4 & 5 to 0 & 1
        MOV     R4,UNI.MSG              ; store higher order address
        MOV     M.UMRA(R0),R1           ; get UMR address
        MOV     M.BFVL(R0),(R1)+        ; load lower order address
        MOVB    M.BFVH(R0),(R1)         ; load higher order address

        MOV     #1,R0                   ; return success
        BR      RTN
FAILS:
        MOV     #0,R0                   ; unsuccessful
RTN:

        .IF DF  C$SPRT

        MOV     (SP)+,R5
        jmp     c$ret

        .IFF

        RETURN

        .ENDC

;
; .AS.UMR
;
;               This 'mac' callable routine actually goes int system state
;               to assign the UMR's
;
;       inputs:
;               R0 -> address of UMR assignment block with no. of UMR's * 4 to
;                     assign in M.UMRN
;

        .psect  c$data,d,rw

SUCC:   .WORD   0               ; return status

        .psect  c$text,i,ro

.AS.UMR::

        SWSTK$  20$             ;; switch to system state
        CALL    $ASUMR          ;; assign UMR's
        BCS     10$             ;; if CS then it fails
        MOV     #1,SUCC         ;; indicate success
        RETURN                  ;; return to task state at 20$
10$:
        CLR     SUCC            ;; indicate failure
        RETURN                  ;; return to task state
20$:
        RETURN
```

```
;
; REL.POOL
;
;               This 'C' callable routine fills up the relocated address of
;               the pool in the global data structures.
;

        .psect  c$text,i,ro

REL.POOL::

        .IF DF  C$SPRT

        jsr     R5,c$sav

        .ENDC

        MOV     PHY.BUF,R0              ; higher order address
        MOV     PHY.BUF+2,R1            ; lower order address

        ASHC    #10.,R0                 ; calculate rel bias and the disp.
        ASHC    #-10.,R1                ;

        MOV     R0,REL1BUF              ; relocation bias
        ADD     #140000,R1              ; set displacement
        MOV     R1,REL1BUF+2            ; store it

        ADD     #200,R0                 ; add an eq. of 4KW
        MOV     R0,REL2BUF              ; rel bias for next 4KW
        MOV     R1,REL2BUF+2            ; displ bias is same

        .IF DF  C$SPRT

        JMP     C$RET

        .IFF

        RETURN

        .ENDC

;
; ACOPY
;
;       This 'C' callable routine is used to Xfer data from one part of the
;       physical memory to the other using inputs as the relocated addresses
;       of both source and destination.
;
;       INPUTS:
;               R0 --> source rel addr pointer
;               R1 --> destination rel addr pointer
;               R2 --> byte count
;

        .psect  c$text,i,ro
```

```
ACOPY::

        .IF DF  C$SPRT

        jsr     R5,c$sav
        MOV     R5,-(SP)

        MOV     4(R5),R0                ; source relocated addr pointer
        MOV     6(R5),R1                ; destination rel addr pointer
        MOV     10(R5),R2               ; byte count

        .ENDC

        MOV     R2,R5                   ; save count in R5
        SWSTK$  RET                     ;; switch to system state
        MOV     (R1)+,R3                ;; dest. rel. bias
        MOV     (R1)+,R4                ;; dest. displ. bias
        MOV     (R0)+,R1                ;; src rel. bias
        MOV     (R0),R2                 ;; src displ. bias
        ADD     #120000-140000,R2       ;; convert src to APR5 bias
        MOV     R5,R0                   ;; get byte count
        CALLR   $BLXIO                  ;; move data and return to task state
RET:

        .IF DF  C$SPRT

        MOV     (SP)+,R5
        jmp     c$ret

        .IFF

        RETURN

        .ENDC

        .psect  c$text,i,ro

        .MCALL  SREX$S,EXIT$S

SREX::
        .IF DF  C$SPRT

        jsr     R5,c$sav

        .ENDC

        SREX$S  #DE.UMR

        .IF DF  C$SPRT

        jmp     c$ret

        .IFF

        RETURN
```

```
        .ENDC

        .psect  c$data,d,rw

UMRMSG: .WORD   0               ; address of UMR ass. block for msg area

        .psect  c$text,i,ro

DE.UMR::
        ADD     (SP),SP         ; cleanup stack
        MOV     ZEUCB,R2        ; get UCB address
        MOV     U.SCB(R2),R2    ; get SCB address
        ADD     #S.UNI,R2       ; point to UMR ass block for pool area

        CALL    $DEUMR          ; deallocate the UMR's

        MOV     UMRMSG,R2       ; get ptr of UMR ass block for msg area
        MOV     R2,R0           ; save it

        CALL    $DEUMR          ; deallocate the UMR

        MOV     #12.,R1         ; size of this allocated block
        CALL    $DEACB          ; deallocate this blopck back to the sys. pool

        EXIT$S                  ; exit properly

        .psect  c$text,i,ro

EXIT.::
        .IF DF  C$SPRT

        jsr     R5,c$sav

        .ENDC

        EXIT$S

        .IF DF  C$SPRT

        jmp     c$ret

        .IFF

        RETURN

        .ENDC

        .psect  c$data,d,rw
        .even
        .psect  c$text,i,ro
        .even

        .END
```

```
        .ENABLE QUIET
        .DISABLE DISPLAY
        .IFNDF $VRBS .ASK $VRBS Verbose ? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .IFNDF $DEL .ASK $DEL Delete source file from current UFD? [Y/N]
        .IFNDF $NOPRE .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
;
; Assemble and build the ACP code.
;
        .ENABLE SUBSTITUTION
.;
.; Prepare the indirect input file for the tkb and ask for the EXOS's
.; port A address offset in the I/O page. ( the virtual address of the
.; port A is expressed as an offset in the I/O page ).
.;
        .IFDF $PORT .GOTO 1
        .SETS $PORT "4000"
        .ASKS [::$PORT] $PORT OFFSET ADDRESS OF PORTA ? [ D : 4000 ] :
.1:
        .IFDF $VEC .GOTO 5
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .5:

;
; Assemble the Macro source code of the ACP.
;
MAC RWPORT=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'RWPORT
MAC UCOPY=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'UCOPY
MAC SCOPY=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'SCOPY
MAC ACPUCB=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'ACPUCB
MAC RTHMAC=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'RTHMAC
MAC DQPKT=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'DQPKT
.;
.; Delete temporary files
.;
        .IFF $DEL .GOTO 10
PIP ACPUCB.MAC;*/DE
PIP RTHMAC.MAC;*/DE
PIP DQPKT.MAC;*/DE
PIP RWPORT.MAC;*/DE
PIP SCOPY.MAC;*/DE
PIP UCOPY.MAC;*/DE

.10:
;
; task builds the acp and creates the image file in [1,54]
;
.;
.; Create the task builder input definition file
.;
        .OPEN ACPTKB.CMD
        .DATA LB:[1,54]RTHACP/AC:5/-CP=
        .DATA RTH/LB:CMDTAB,ACPUCB,DQPKT,RWPORT,RTHMAC,UCOPY,SCOPY
        .DATA SY:'<UIC>'PROLOGUE/LB:CHDR
        .DATA SY:'<UIC>'PROLOGUE/LB,LB:[1,1]EXELIB/LB
```

```
        .DATA LB:[1,54]RSX11M.STB
        .DATA /
        .DATA UNITS=7
        .DATA TASK=...RTH
        .DATA GBLPAT=CMDTAB:ZEPORT:'$PORT'
        .DATA GBLPAT=CMDTAB:ZEINT:'$VEC'
        .DATA ASG=CO0:5
        .DATA //
        .CLOSE
;
; Task build ACP
;
.IFT $NOPRE PIP LB:[1,54]RTHACP.TSK;*/DE
TKB @ACPTKB
;
; Delete object files
;
PIP ACPUCB.OBJ;*/DE
PIP DQPKT.OBJ;*/DE
PIP RWPORT.OBJ;*/DE
PIP RTHMAC.OBJ;*/DE
PIP UCOPY.OBJ;*,SCOPY.OBJ;*/DE
PIP ACPTKB.CMD;*/DE
;
;       set appropriate protection for the ACP
;
PIP LB:[1,54]RTHACP.TSK/PR/SY:RWED/OW:RWED/GR:RWED/WO:R/FO
        .ENABLE DISPLAY
```

```
        .ENABLE QUIET
        .DISABLE DISPLAY
        .IFNDF $VRBS .ASK $VRBS Verbose ? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .IFNDF $DEL .ASK $DEL Delete source file from current UFD? [Y/N]
        .IFNDF $NOPRE .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
;
; Assemble and build the ACP code.
;
        .ENABLE SUBSTITUTION
.;
.; Prepare the indirect input file for the tkb and ask for the EXOS's
.; port A address offset in the I/O page. ( the virtual address of the
.; port A is expressed as an offset in the I/O page ).
.;
        .IFDF $PORT .GOTO 1
        .SETS $PORT "4000"
        .ASKS [::$PORT] $PORT OFFSET ADDRESS OF PORTA ? [ D : 4000 ] :
.1:
        .IFDF $VEC .GOTO 5
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .5:

;
; Assemble the Macro source code of the ACP.
;
MAC RWPORT=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'RWPORT
MAC UCOPY=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'UCOPY
MAC SCOPY=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'SCOPY
MAC ACPUCB=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'UNIBUS,ACPUCB
MAC RTHMAC=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'RTHMAC
MAC DQPKT=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'DQPKT
MAC UNIMAC=LB:[1,1]EXEMC/ML,[11,10]RSXMC,SY:'<UIC>'UNIMAC
.;
.; Delete temporary files
.;
        .IFF $DEL .GOTO 10
PIP ACPUCB.MAC;*/DE
PIP RTHMAC.MAC;*/DE
PIP DQPKT.MAC;*/DE
PIP RWPORT.MAC;*/DE
PIP SCOPY.MAC;*/DE
PIP UCOPY.MAC;*/DE
PIP UNIBUS.MAC;*/DE
PIP UNIMAC.MAC;*/DE

.10:
;
; task builds the acp and creates the image file in [1,54]
;
.;
.; Create the task builder input definition file
.;
        .OPEN ACPTKB.CMD
        .DATA LB:[1,54]RTHACP/AC:5/-CP=
```

```
        .DATA RTH/LB:CMDTAB,ACPUCB,DQPKT,RWPORT,RTHMAC,UCOPY,UNIMAC,SCOPY
        .DATA SY:'<UIC>'PROLOGUE/LB:CHDR
        .DATA SY:'<UIC>'PROLOGUE/LB,LB:[1,1]EXELIB/LB
        .DATA LB:[1,54]RSX11M.STB
        .DATA /
        .DATA UNITS=7
        .DATA TASK=...RTH
        .DATA GBLPAT=CMDTAB:ZEPORT:'$PORT'
        .DATA GBLPAT=CMDTAB:ZEINT:'$VEC'
        .DATA ASG=CO0:5
        .DATA //
        .CLOSE
;
; Task build ACP
;
.IFT $NOPRE PIP LB:[1,54]RTHACP.TSK;*/DE
TKB @ACPTKB
;
; Delete object files
;
PIP ACPUCB.OBJ;*/DE
PIP DQPKT.OBJ;*/DE
PIP RWPORT.OBJ;*/DE
PIP RTHMAC.OBJ;*/DE
PIP UCOPY.OBJ;*,SCOPY.OBJ;*/DE
PIP ACPTKB.CMD;*/DE
PIP UNIMAC.OBJ;*/DE
;
;       set appropriate protection for the ACP
;
PIP LB:[1,54]RTHACP.TSK/PR/SY:RWED/OW:RWED/GR:RWED/WO:R/FO
        .ENABLE DISPLAY
```

```
RTHACP/AC:5/-CP,RTHACP/-sp/CR=
RTH/LB:CMDTAB,ACPUCB,DQPKT,RWPORT,RTHMAC,UCOPY,SCOPY
sy:[1,3]PROLOGUE/LB,LB:[1,1]EXELIB/LB
LB:[1,54]RSX11M.STB
/
UNITS=7
TASK=...RTH
GBLPAT=CMDTAB:ZEPORT:4000
GBLPAT=CMDTAB:ZEINT:400
//
```

```
$ !
$ !     skeleton for cmplbr.com
$ !
$ if "''p1'" .nes. "?" then goto doit
$ typ sys$input

 command file to compile and link the library

 required command files:    None

 required logical names:    None

 required parameters:
        p1      - default directory (default - current directory)

 required files:
        none

 required symbols:
        none

 Note:
    You need to edit this file to setup the symbols objlib and inclib as the
    file specifications for the the object and include libraries
$ exit
$ doit:
$ sv = f$verify(1)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ !
$ !     now make assignment for RSX11M UNIBUS version
$ !
$ assign __dra0:[unil1m.] lb:
$ assign __dra0:[unil1m.] lb0:
$ if "''p1'" .eqs. "" then $ p1 = "''f$logical("sys$disk")''f$directory()'"
$ set def 'p1'
$ show def
$ show logical lb
$ !
$ !     now set up environment for C compiler
$ !
$ cpp == "mcr cpp"
$ cp1 == "mcr cp1"
$ cp2 == "mcr cp2"
$ assign dra0:[albert.cutil]cpp.exe cpp
$ assign dra0:[albert.cutil]cp1.exe cp1
$ assign dra0:[albert.cutil]cp2.exe cp2
$ !
$ !     go compile all the files
$ !
$ lbr rthuni/cr
$ mac rwportuni,rwportuni/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]rwport
$ mac ucopyuni,ucopyuni/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]ucopy
$ mac scopyuni,scopyuni/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]scopy
$ mac acpucbu,acpucbu/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]unibus,acpucb
$ mac unimac,unimac/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]unimac
```

```
$ mac rthmacuni,rthmacuni/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]rthmac
$ mac dqpktuni,dqpktuni/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]dqpkt
$ !
$ !     C program
$ !
$ cpp -x -i lb:[1,1]|sy:[10,10]|sy:[1,3] -o sy:[1,3]c1.tmp sy:[1,3]u.h [1,3]body.c
$ cp1 -o sy:[1,3]c2.tmp sy:[1,3]c1.tmp
$ cp2 -o sy:[1,3]c3.tmp sy:[1,3]c2.tmp
$ mac body=c3.tmp
$ lbr rthuni/rp=body
$ delete/log c1.tmp;*,c2.tmp;*,c3.tmp;*
$ exit 1
$ abnormal_exit:
$ exit 2
```

```
$ !
$ !     skeleton for bld.com
$ !
$ if "'''p1'" .nes. "?" then goto doit
$ typ sys$input

 command file to build the task image

 required command files:    None

 required logical names:    None

 required parameters:
        p1      - default directory (default - current directory)

 required files:        None

 required symbols:      None

$ exit
$ doit:
$ sv = f$verify(1)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ if "'''p1'" .eqs. "" then $ p1 = "'''f$logical("sys$disk")'''f$directory()'"
$ set def 'p1'
$ show def
$ !
$ !     Put your own commands here
$ !
$ !     Make assignment for QBUS RSX11M
$ !
$ assign __dra0:[qbus11m.] lb:
$ copy/log prologue.sav prologue.olb
$ open/write lnkdrv tkb.cmd
$ write lnkdrv "RTHACP/AC:5/-CP,RTHACP/-sp/CR="
$ write lnkdrv "RTH/LB:CMDTAB,ACPUCB,DQPKT,RWPORT,RTHMAC,UCOPY,SCOPY"
$ write lnkdrv "sy:[1,3]PROLOGUE/LB,LB:[1,1]EXELIB/LB"
$ write lnkdrv "LB:[1,54]RSX11M.STB"
$ write lnkdrv "/"
$ write lnkdrv "UNITS=7"
$ write lnkdrv "TASK=...RTH"
$ write lnkdrv "GBLPAT=CMDTAB:ZEPORT:4000"
$ write lnkdrv "GBLPAT=CMDTAB:ZEINT:400"
$ write lnkdrv "//"
$ close lnkdrv
$ tkb @tkb.cmd
$ delete tkb.cmd;
$ deassign lb
$ !
$ !     Make assignment for UNIBUS RSX11M
$ !
$ assign __dra0:[unill1m.] lb:
$ open/write lnkdrv tkb.cmd
$ write lnkdrv "RTHACPUNI/AC:5/-CP,RTHACPUNI/-sp/CR="
$ write lnkdrv "RTHUNI/LB:CMDTAB,ACPUCBU,DQPKTUNI,RWPORTUNI"
```

```
$ write lnkdrv "RTHMACUNI,UCOPYUNI,SCOPYUNI"
$ write lnkdrv "UNIMAC"
$ write lnkdrv "sy:[1,3]PROLOGUE/LB,LB:[1,1]EXELIB/LB"
$ write lnkdrv "LB:[1,54]RSX11M.STB"
$ write lnkdrv "/"
$ write lnkdrv "UNITS=7"
$ write lnkdrv "TASK=...RTH"
$ write lnkdrv "GBLPAT=CMDTAB:ZEPORT:4000"
$ write lnkdrv "GBLPAT=CMDTAB:ZEINT:400"
$ write lnkdrv "//"
$ close lnkdrv
$ tkb @tkb.cmd
$ delete tkb.cmd;
$ deassign lb
$ !
$ !     Make assignment for UNIBUS RSX11M-Plus
$ !
$ assign __dra0:[unillmp.] lb:
$ open/write lnkdrv tkb.cmd
$ write lnkdrv "RTHACPUP/AC:5/-CP,RTHACPUP/-sp/CR="
$ write lnkdrv "RTHUP/LB:CMDTAB,ACPUCBU,DQPKTUP,RWPORTUP"
$ write lnkdrv "RTHMACUP,UCOPYUP,SCOPYUP"
$ write lnkdrv "UPMAC"
$ write lnkdrv "sy:[1,3]PROLOGUE/LB,LB:[1,1]EXELIB/LB"
$ write lnkdrv "LB:[1,54]RSX11M.STB"
$ write lnkdrv "/"
$ write lnkdrv "UNITS=7"
$ write lnkdrv "TASK=...RTH"
$ write lnkdrv "GBLPAT=CMDTAB:ZEPORT:4000"
$ write lnkdrv "GBLPAT=CMDTAB:ZEINT:400"
$ write lnkdrv "//"
$ close lnkdrv
$ tkb @tkb.cmd
$ delete tkb.cmd;
$ deassign lb
$ exit 1
$ abnormal_exit:
$ deassign lb
$ exit 2
```

```
$ !
$ !     skeleton for cmplbr.com
$ !
$ if "'''p1'" .nes. "?" then goto doit
$ typ sys$input

 command file to compile and link the library

 required command files:    None

 required logical names:    None

 required parameters:
        p1      - default directory (default - current directory)

 required files:
        none

 required symbols:
        none

 Note:
    You need to edit this file to setup the symbols objlib and inclib as the
    file specifications for the the object and include libraries
$ exit
$ doit:
$ sv = f$verify(1)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ !
$ !     now make assignment for RSX11M Q-bus version
$ !
$ assign __dra0:[qbus11m.] lb:
$ assign __dra0:[qbus11m.] lb0:
$ if "'''p1'" .eqs. "" then $ p1 = "'''f$logical("sys$disk")'''f$directory()'"
$ set def 'p1'
$ show def
$ show logical lb
$ !
$ !     now set up environment for C compiler
$ !
$ cpp == "mcr cpp"
$ cp1 == "mcr cp1"
$ cp2 == "mcr cp2"
$ assign dra0:[albert.cutil]cpp.exe cpp
$ assign dra0:[albert.cutil]cp1.exe cp1
$ assign dra0:[albert.cutil]cp2.exe cp2
$ !
$ !     go compile all the files
$ !
$ lbr rth/cr
$ mac rwport,rwport/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]rwport
$ mac ucopy,ucopy/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]ucopy
$ mac scopy,scopy/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]scopy
$ mac acpucb,acpucb/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]acpucb
$ mac rthmac,rthmac/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]rthmac
```

```
$ mac dqpkt,dqpkt/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]dqpkt
$ !
$ !     C program
$ !
$ cpp -x -i lb:[1,1]|sy:[10,10]|sy:[1,3] -o sy:[1,3]c1.tmp sy:[1,3]body.c
$ cp1 -o sy:[1,3]c2.tmp sy:[1,3]c1.tmp
$ cp2 -o sy:[1,3]c3.tmp sy:[1,3]c2.tmp
$ mac body=c3.tmp
$ lbr rth/rp=body
$ delete/log c1.tmp;*,c2.tmp;*,c3.tmp;*
$ @altcmplbr
$ @umpcmplbr
$ exit 1
$ abnormal_exit:
$ exit 2
```

```
$ !
$ !     skeleton for deliver.com
$ !
$ if "''p1'" .nes. "?" then goto doit
$ typ sys$input

 command file to copy the deliver files to manufacturing area
 You should modify this file to copy the deliverables to
    exos$mfg:[target_directory]

 required command files:    None

 required logical names:    None
        exos$mfg         - pseudo disk for deliverables

 required parameters:   Noe

 required files:        None

 required symbols:      None

$ exit
$ doit:
$ sv = f$verify(0)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ show def
$ !
$ !     Put your own commands here
$ !
$ copy/log      bldacp.cmd      exos$mfg:[rsx]
$ copy/log      rth.olb         exos$mfg:[rsx]
$ copy/log      rwport.mac      exos$mfg:[rsx]
$ copy/log      ucopy.mac       exos$mfg:[rsx]
$ copy/log      scopy.mac       exos$mfg:[rsx]
$ copy/log      acpucb.mac      exos$mfg:[rsx]
$ copy/log      rthmac.mac      exos$mfg:[rsx]
$ copy/log      dqpkt.mac       exos$mfg:[rsx]
$ copy/log      prologue.olb    exos$mfg:[rsx]
$ copy/log      unibus.mac      exos$mfg:[rsxunibus]
$ copy/log      rthuni.olb      exos$mfg:[rsxunibus]rth.olb
$ copy/log      unimac.mac      exos$mfg:[rsxunibus]
$ copy/log      blduni.cmd      exos$mfg:[rsxunibus]bldacp.cmd
$ exit 1
$ abnormal_exit:
$ exit 2
```

```
$ !
$ !     skeleton for cmplbr.com
$ !
$ if "''p1'" .nes. "?" then goto doit
$ typ sys$input

 command file to compile and link the library

 required command files:    None

 required logical names:    None

 required parameters:
        p1      - default directory (default - current directory)

 required files:
        none

 required symbols:
        none

 Note:
    You need to edit this file to setup the symbols objlib and inclib as the
    file specifications for the the object and include libraries
$ exit
$ doit:
$ sv = f$verify(1)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ !
$ !     now make assignment for RSX11M-Plus UNIBUS version
$ !
$ assign __dra0:[unillmp.] lb:
$ assign __dra0:[unillmp.] lb0:
$ if "''p1'" .eqs. "" then $ p1 = "''f$logical("sys$disk")''f$directory()'"
$ set def 'p1'
$ show def
$ show logical lb
$ !
$ !     now set up environment for C compiler
$ !
$ cpp == "mcr cpp"
$ cp1 == "mcr cp1"
$ cp2 == "mcr cp2"
$ assign dra0:[albert.cutil]cpp.exe cpp
$ assign dra0:[albert.cutil]cp1.exe cp1
$ assign dra0:[albert.cutil]cp2.exe cp2
$ !
$ !     go compile all the files
$ !
$ lbr rthup/cr
$ mac rwportup,rwportup/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]rwport
$ mac ucopyup,ucopyup/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]ucopy
$ mac scopyup,scopyup/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]scopy
$ mac acpucbu,acpucbu/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]unibus,acpucb
$ mac upmac,upmac/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]unimac
```

```
$ mac rthmacup,rthmacup/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]rthmac
$ mac dqpktup,dqpktup/-sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]dqpkt
$ !
$ !     C program
$ !
$ cpp -x -i lb:[1,1]|sy:[10,10]|sy:[1,3] -o sy:[1,3]c1.tmp sy:[1,3]u.h [1,3]body.c
$ cp1 -o sy:[1,3]c2.tmp sy:[1,3]c1.tmp
$ cp2 -o sy:[1,3]c3.tmp sy:[1,3]c2.tmp
$ mac body=c3.tmp
$ lbr rthup/rp=body
$ delete/log c1.tmp;*,c2.tmp;*,c3.tmp;*
$ exit 1
$ abnormal_exit:
$ exit 2
```

```
;
;       COPYRIGHT (c) 1985 BY EXCELAN, INC.
;       SAN JOSE, CALIFORNIA.  ALL RIGHTS RESERVED.
;
.; THIS  SOFTWARE  IS FURNISHED UNDER A LICENSE AND MAY
.; BE USED AND  COPIED  ONLY  IN  ACCORDANCE  WITH  THE
.; TERMS  OF SUCH LICENSE AND WITH THE INCLUSION OF THE
.; ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY  OTHER
.; COPIES  THEREOF  MAY  NOT  BE  PROVIDED OR OTHERWISE
.; MADE AVAILABLE TO ANY OTHER  PERSON.   NO  TITLE  TO
.; AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
.;
.; THE  INFORMATION  IN  THIS  SOFTWARE  IS  SUBJECT TO
.; CHANGE WITHOUT NOTICE AND SHOULD  NOT  BE  CONSTRUED
.; AS  A  COMMITMENT  BY EXCELAN, INC.
.;
.; EXCELAN, INC. ASSUMES NO RESPONSIBILITY  FOR  THE  USE
.; OR RELIABILITY  OF  ITS  SOFTWARE  ON EQUIPMENT THAT IS
.; NOT SUPPLIED BY EXCELAN, INC.
        .ENABLE QUIET
        .ENABLE LOWERCASE
        .ENABLE GLOBAL
        .ENABLE SUBSTITUTION
.IFT <PRIVIL> .GOTO 5
; Error: You must be privileged in order to install EXOS 8030 software.
.EXIT
.5:
        .DISABLE DISPLAY
        .ASK $VRBS Verbose? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
        .ASK $DEL Delete source file from current UFD in target disk? [Y/N]
        .ASK $DRV Build driver and ACP only? [Y/N]
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .SETS $PORT "4000"
        .ASKS [::$PORT] $PORT OFFSET ADDRESS OF PORTA ? [ D : 4000 ]
        .SETN $SESS 1
        .ASKN [::$SESS] $SESS Maximum number of concurrent FTP server sessions? [D : 1]

;
;       This command file copies the required files from the distribution
;       floppy
;
.;      Ask for source device name
.;
        .ASKS   $DEV Copy from device [ddnn:]:
.;
.;      check if the device is mounted and mount if necessary
.;
.TESTDEVICE '$DEV'
.TEST <EXSTRI> "MTD"
.IF <STRLEN> NE 0 .GOTO 10
.;
.;      device not mounted
.;
```

```
MOU '$DEV'EXOS1
.;
.;      start copy
.;
.10:
PIP /NV/CD='$DEV'[1,1]BLDDRV.CMD/NM
PIP /NV/CD='$DEV'[1,1]ZEDRV.MAC/NM
PIP /NV/CD='$DEV'[1,1]ZETAB.MAC/NM
PIP /NV/CD='$DEV'[1,1]RTH.OLB/NM
PIP /NV/CD='$DEV'[1,1]ACPUCB.MAC/NM
PIP /NV/CD='$DEV'[1,1]DQPKT.MAC/NM
PIP /NV/CD='$DEV'[1,1]RWPORT.MAC/NM
PIP /NV/CD='$DEV'[1,1]SCOPY.MAC/NM
PIP /NV/CD='$DEV'[1,1]UCOPY.MAC/NM
PIP /NV/CD='$DEV'[1,1]RTHMAC.MAC/NM
PIP /NV/CD='$DEV'[1,1]BLDACP.CMD/NM
PIP /NV/CD='$DEV'[1,1]PROLOGUE.OLB/NM
PIP /NV/CD='$DEV'[1,1]bldzt.cmd/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS2 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS2
PIP /NV/CD='$DEV'[1,1]zttab.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztyt.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztini.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztrw.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztich.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztcan.MAC/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS3 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS3
PIP /NV/CD='$DEV'[1,1]ztatt.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztois.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztdat.MAC/NM
PIP /NV/CD='$DEV'[1,1]zttbl.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztsub.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztcis.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztfp.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztodn.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztmis.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztmod.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztmac.MAC/NM
.15:
.;
.;      build the driver
.;
@BLDDRV
.IFT $DEL PIP BLDDRV.CMD;/DE
.;
.;      build the pseudo-terminal driver
@BLDZT
```

```
.IFT $DEL PIP BLDZT.CMD;/DE
.;
.;      build the ACP
.;
@BLDACP
.IFT $DEL .AND .IFT $DRV PIP PROLOGUE.OLB;/DE
.IFT $DEL PIP RTH.OLB;/DE
.IFT $DEL PIP BLDACP.CMD;/DE
.;
.;      Now copy utilities to various destination location
.;
.IFT $DRV DMO '$DEV'
.IFT $DRV .EXIT
.20:
.ASKS DESTUI Please enter the UFD for the EXOS utilities
.IF DESTUI = "" .GOTO 20
.;
.;      Copy task image
.;
.IFF $NOPRE .GOTO 25
PIP 'DESTUI'ARP.TSK;*/DE
PIP 'DESTUI'BSTAT.TSK;*/DE
PIP 'DESTUI'NETLOAD.TSK;*/DE
PIP 'DESTUI'NETSTAT.TSK;*/DE
PIP 'DESTUI'TTCP.TSK;*/DE
PIP 'DESTUI'XROUTE.TSK;*/DE
PIP 'DESTUI'FTPC.TSK;*/DE
PIP 'DESTUI'FTPDEMON.TSK;*/DE
PIP 'DESTUI'TELNET.TSK;*/DE
PIP 'DESTUI'LOGIN.TSK;*/DE
PIP 'DESTUI'FTPD.TSK;*/DE
.25:
DMO '$DEV'
;
;       Please mount floppy labelled EXOS4 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS4
PIP /FO/NV/CD='$DEV'[1,1]LOGIN.OLB/NM
PIP /FO/NV/CD='$DEV'[1,1]PASWORD.MAC/NM
PIP /FO/NV/CD='$DEV'[1,1]ACTFIL.MAC/NM
PIP /FO/NV/CD='$DEV'[1,1]BLDLGN.CMD/NM
@BLDLGN
.IFT $DEL PIP LOGIN.OLB;*/DE
.IFT $DEL PIP BLDLGN.CMD;*/DE
PIP 'DESTUI'/FO/CO/NV/CD=SY:'<UIC>'LOGIN.TSK/NM
PIP LOGIN.TSK;/DE/NM
PIP /FO/NV/CD='$DEV'[1,1]DEMON.OLB/NM
PIP /FO/NV/CD='$DEV'[1,1]RECVAST.MAC/NM
PIP /FO/NV/CD='$DEV'[1,1]BLDDEM.CMD/NM
PIP /FO/NV/CD='$DEV'[1,1]DEMON.MAC/NM
@BLDDEM
.IFT $DEL PIP DEMON.OLB;*/DE
.;.IFT $DEL PIP RECVAST.MAC;*/DE
.IFT $DEL PIP BLDDEM.CMD;*/DE
.IFT $DEL PIP PROLOGUE.OLB;*/DE
```

```
PIP 'DESTUI'/FO/CO/NV/CD=SY:'<UIC>'FTPDEMON.TSK/NM
PIP FTPDEMON.TSK;/DE/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]ARP.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]BSTAT.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]NETLOAD.TSK/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS5 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS5
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]NETSTAT.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]TTCP.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]XROUTE.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]FTPC.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]TELNET.TSK/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS6 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS6
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]FTPD.TSK/NM
.;
.;      copy specific programs
.;
.IFT $NOPRE PIP 'DESTUI'RHOST.C;*/DE
.IFT $NOPRE PIP 'DESTUI'RADDR.C;*/DE
.IFT $NOPRE PIP 'DESTUI'SOCKET.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.H;*/DE
.IFT $NOPRE PIP LB:[1,2]NET.;*/DE
.IFT $NOPRE PIP 'DESTUI'8030.HLP;*/DE
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]RHOST.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]RADDR.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]SOCKET.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]TTCP.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]TTCP.H/NM
PIP LB:[1,2]/FO/NV/CD='$DEV'[1,1]NET./NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]8030.HLP/NM
.ASK INITHO Do you want to initialize the network addresses file (HOSTS.NET)
.IFF INITHO .GOTO SETLD
.IFT $NOPRE PIP LB:[1,1]HOSTS.NET;*/DE
PIP LB:[1,1]/FO/NV/CD='$DEV'[1,1]HOSTS.NET/NM
.OPENA LB:[1,1]HOSTS.NET
.ASKS HNAME Name of host
.ASKS HADDR Host internet address
.DATA 'HADDR' 'HNAME' localhost
.CLOSE
.IFT $NOPRE PIP LB:[1,1]HOSTLOCAL.NET;*/DE
PIP LB:[1,1]/FO/NV/CD='$DEV'[1,1]HOSTLOCAL.NET/NM
.;
.;      Write out the EXOSLOAD command file
.;
.SETLD:
.IFT $NOPRE PIP LB:[1,1]EXOSLOAD.CMD;*/DE
```

```
    .OPEN LB:[1,1]EXOSLOAD.CMD
    .DATA .ENABLE SUBSTITUTION
    .DATA .IFACT DEMT0 ABO DEMT0
    .DATA .IFACT LGNT0 ABO LGNT0
    .DATA .IFACT ...DEM ABO ...DEM
    .DATA .IFACT ...LGN ABO ...LGN
    .SETN LCOUNT 0
.80$:
    .IF   LCOUNT >= '$SESS' .GOTO 89$
    .DATA .IFACT FTD00'LCOUNT' ABO FTD00'LCOUNT'
    .DATA .IFINS FTD00'LCOUNT' REM FTD00'LCOUNT'
    .DATA .IFINS XDR00'LCOUNT' REM XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 80$
.89$:
    .DATA .IFINS ...DEM REM ...DEM
    .DATA .IFINS ...ARP REM ...ARP
    .DATA .IFINS ...BST REM ...BST
    .DATA .IFINS ...FTP REM ...FTP
    .DATA .IFINS ...NET REM ...NET
    .DATA .IFINS ...TEL REM ...TEL
    .DATA .IFINS ...TTC REM ...TTC
    .DATA .IFINS ...ROU REM ...ROU
    .DATA .IFINS ...NST REM ...NST
    .DATA .IFINS ...LGN REM ...LGN
    .DATA .IFACT ...RTH ABO ...RTH
    .DATA .IFACT RTHT0 ABO RTHT0
    .DATA .IFINS ...RTH REM ...RTH
    .DATA .IF <SYSTEM> <> 6 .GOTO 10$
    .DATA .IFLOA ZE: CON OFFLINE ZEA
    .DATA .IFLOA ZE: CON OFFLINE ZE0:
    .DATA .IFNLOA ZT: .GOTO 10$
    .DATA CON OFFLINE ZTA
    .DATA CON OFFLINE ZTB
    .DATA CON OFFLINE ZTC
    .DATA CON OFFLINE ZTD
    .DATA CON OFFLINE ZTE
    .DATA CON OFFLINE ZTF
    .DATA CON OFFLINE ZTH
    .DATA CON OFFLINE ZTJ
    .DATA CON OFFLINE ZT0:
    .DATA CON OFFLINE ZT1:
    .DATA CON OFFLINE ZT2:
    .DATA CON OFFLINE ZT3:
    .DATA CON OFFLINE ZT4:
    .DATA CON OFFLINE ZT5:
    .DATA CON OFFLINE ZT6:
    .DATA CON OFFLINE ZT7:
    .DATA .10$:
    .DATA .IFLOA ZE: UNL ZE:
    .DATA .IFLOA ZT: UNL ZT:
    .DATA LOA ZE:/PAR=GEN/HIGH/SIZE=20000
    .DATA .IF <SYSTEM> <> 6 LOA ZT:
    .DATA .IF <SYSTEM> <> 6 UNL ZT:
    .DATA ; You can ignore the error message: "Loadable driver larger than 4KW"
    .DATA LOA ZT:/HIGH/SIZE=20000
```

```
    .DATA .IF <SYSTEM> <> 6 .GOTO 20$
    .DATA ;     configure the devices online
    .DATA CON ONLINE ZEA
    .DATA CON ONLINE ZE0:
    .DATA CON SET ZTA VEC=0
    .DATA CON SET ZTB VEC=0
    .DATA CON SET ZTC VEC=0
    .DATA CON SET ZTD VEC=0
    .DATA CON SET ZTE VEC=0
    .DATA CON SET ZTF VEC=0
    .DATA CON SET ZTH VEC=0
    .DATA CON SET ZTJ VEC=0
    .DATA CON ONLINE ALL
    .DATA .20$:
    .DATA INS $RTHACP/PRI=150.
    .DATA .XQT RTH
    .DATA INS 'DESTUI'ARP.TSK
    .DATA INS 'DESTUI'BSTAT.TSK
    .DATA INS 'DESTUI'FTPC.TSK
    .DATA INS 'DESTUI'FTPDEMON.TSK
    .DATA INS 'DESTUI'NETLOAD.TSK
    .DATA INS 'DESTUI'TELNET.TSK
    .DATA INS 'DESTUI'TTCP.TSK
    .DATA INS 'DESTUI'XROUTE.TSK
    .DATA INS 'DESTUI'NETSTAT.TSK
    .DATA INS 'DESTUI'LOGIN.TSK
    .SETN LCOUNT 0
    .DATA .SETS FTDOPT ""
    .DATA .IF <SYSTEM> = 6 .SETS FTDOPT "/XHR=NO"
.90$:
    .IF   LCOUNT >= '$SESS' .GOTO 99$
    .DATA INS 'DESTUI'FTPD.TSK/TASK=FTD00'LCOUNT'''FTDOPT''
    .DATA INS $PIP/TASK=XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 90$
.99$:
    .DATA .ASK DWN Do you want to initialize the EXOS front end processor
    .DATA .IFT DWN net
    .DATA .ASK DMN Do you want to start the FTP server
    .DATA .IFT DMN .XQT dem
    .DATA .IFT DMN .XQT lgn
    .CLOSE
    PIP LB:[1,1]EXOSLOAD.CMD/PR/FO
;
;       Please add the following line to LB:[1,2]STARTUP.CMD so that the
;       network is reloaded everytime the system is rebooted.
;
;       @LB:[1,1]EXOSLOAD
;
;       You may need to edit the file LB:[1,1]EXOSLOAD.CMD to set up the
;       options in loading the network module.
;
.;
.;
.;      dismount device
.;
```

```
DMO '$DEV'
;
;       Installation completed.  Now you can execute
;       @LB:[1,1]EXOSLOAD
;       to start up the network connection.
```

```
;
;       COPYRIGHT (c) 1985 BY EXCELAN, INC.
;       SAN JOSE, CALIFORNIA.  ALL RIGHTS RESERVED.
;
.; THIS  SOFTWARE  IS FURNISHED UNDER A LICENSE AND MAY
.; BE USED AND  COPIED  ONLY  IN  ACCORDANCE  WITH  THE
.; TERMS  OF SUCH LICENSE AND WITH THE INCLUSION OF THE
.; ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY  OTHER
.; COPIES  THEREOF  MAY  NOT  BE  PROVIDED OR OTHERWISE
.; MADE AVAILABLE TO ANY OTHER  PERSON.   NO  TITLE  TO
.; AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
.;
.; THE  INFORMATION  IN  THIS  SOFTWARE  IS  SUBJECT TO
.; CHANGE WITHOUT NOTICE AND SHOULD  NOT  BE  CONSTRUED
.; AS  A  COMMITMENT  BY EXCELAN, INC.
.;
.; EXCELAN, INC. ASSUMES NO RESPONSIBILITY  FOR  THE  USE
.; OR RELIABILITY  OF  ITS  SOFTWARE  ON EQUIPMENT THAT IS
.; NOT SUPPLIED BY EXCELAN, INC.
        .ENABLE QUIET
        .ENABLE LOWERCASE
        .ENABLE GLOBAL
        .ENABLE SUBSTITUTION
.IFT <PRIVIL> .GOTO 5
; Error: You must be privileged in order to install EXOS 8030 software.
.EXIT
.5:
        .DISABLE DISPLAY
        .ASK $VRBS Verbose? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
        .ASK $DEL Delete source file from current UFD in target disk? [Y/N]
        .ASK $DRV Build driver and ACP only? [Y/N]
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .SETS $PORT "4000"
        .ASKS [::$PORT] $PORT OFFSET ADDRESS OF PORTA ? [ D : 4000 ]
        .SETN $SESS 1
        .ASKN [::$SESS] $SESS Maximum number of concurrent FTP server sessions? [D : 1]

;
;       This command file copies the required files from the distribution
;       floppy
;
.;      Ask for source device name
.;
        .ASKS   $DEV Copy from device [ddnn:]:
.;
.;      check if the device is mounted and mount if necessary
.;
.TESTDEVICE '$DEV'
.TEST <EXSTRI> "MTD"
.IF <STRLEN> NE 0 .GOTO 10
.;
.;      device not mounted
.;
```

```
MOU '$DEV'EXOS1
.;
.;      start copy
.;
.10:
PIP /NV/CD='$DEV'[1,1]BLDDRV.CMD/NM
PIP /NV/CD='$DEV'[1,1]UNIBUS.MAC/NM
PIP /NV/CD='$DEV'[1,1]ZEDRV.MAC/NM
PIP /NV/CD='$DEV'[1,1]ZETAB.MAC/NM
PIP /NV/CD='$DEV'[1,1]RTH.OLB/NM
PIP /NV/CD='$DEV'[1,1]ACPUCB.MAC/NM
PIP /NV/CD='$DEV'[1,1]DQPKT.MAC/NM
PIP /NV/CD='$DEV'[1,1]RWPORT.MAC/NM
PIP /NV/CD='$DEV'[1,1]UNIMAC.MAC/NM
PIP /NV/CD='$DEV'[1,1]SCOPY.MAC/NM
PIP /NV/CD='$DEV'[1,1]UCOPY.MAC/NM
PIP /NV/CD='$DEV'[1,1]RTHMAC.MAC/NM
PIP /NV/CD='$DEV'[1,1]BLDACP.CMD/NM
PIP /NV/CD='$DEV'[1,1]PROLOGUE.OLB/NM
PIP /NV/CD='$DEV'[1,1]bldzt.cmd/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS2 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS2
PIP /NV/CD='$DEV'[1,1]zttab.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztyt.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztini.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztrw.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztich.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztcan.MAC/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS3 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS3
PIP /NV/CD='$DEV'[1,1]ztatt.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztois.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztdat.MAC/NM
PIP /NV/CD='$DEV'[1,1]zttbl.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztsub.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztcis.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztfp.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztodn.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztmis.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztmod.MAC/NM
PIP /NV/CD='$DEV'[1,1]ztmac.MAC/NM
.15:
.;
.;      build the driver
.;
@BLDDRV
.IFT $DEL PIP BLDDRV.CMD;/DE
.;
```

```
.;      build the pseudo-terminal driver
@BLDZT
.IFT $DEL PIP BLDZT.CMD;/DE
.;
.;      build the ACP
.;
@BLDACP
.IFT $DEL .AND .IFT $DRV PIP PROLOGUE.OLB;/DE
.IFT $DEL PIP RTH.OLB;/DE
.IFT $DEL PIP BLDACP.CMD;/DE
.;
.;      Now copy utilities to various destination location
.;
.IFT $DRV DMO '$DEV'
.IFT $DRV .EXIT
.20:
.ASKS DESTUI Please enter the UFD for the EXOS utilities
.IF DESTUI = "" .GOTO 20
.;
.;      Copy task image
.;
.IFF $NOPRE .GOTO 25
PIP 'DESTUI'ARP.TSK;*/DE
PIP 'DESTUI'BSTAT.TSK;*/DE
PIP 'DESTUI'NETLOAD.TSK;*/DE
PIP 'DESTUI'NETSTAT.TSK;*/DE
PIP 'DESTUI'TTCP.TSK;*/DE
PIP 'DESTUI'XROUTE.TSK;*/DE
PIP 'DESTUI'FTPC.TSK;*/DE
PIP 'DESTUI'FTPDEMON.TSK;*/DE
PIP 'DESTUI'TELNET.TSK;*/DE
PIP 'DESTUI'LOGIN.TSK;*/DE
PIP 'DESTUI'FTPD.TSK;*/DE
.25:
DMO '$DEV'
;
;       Please mount floppy labelled EXOS4 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS4
PIP /FO/NV/CD='$DEV'[1,1]LOGIN.OLB/NM
PIP /FO/NV/CD='$DEV'[1,1]PASWORD.MAC/NM
PIP /FO/NV/CD='$DEV'[1,1]ACTFIL.MAC/NM
PIP /FO/NV/CD='$DEV'[1,1]BLDLGN.CMD/NM
@BLDLGN
.IFT $DEL PIP LOGIN.OLB;*/DE
.IFT $DEL PIP BLDLGN.CMD;*/DE
PIP 'DESTUI'/FO/CO/NV/CD=SY:'<UIC>'LOGIN.TSK/NM
PIP LOGIN.TSK;/DE/NM
PIP /FO/NV/CD='$DEV'[1,1]DEMON.OLB/NM
PIP /FO/NV/CD='$DEV'[1,1]RECVAST.MAC/NM
PIP /FO/NV/CD='$DEV'[1,1]BLDDEM.CMD/NM
PIP /FO/NV/CD='$DEV'[1,1]DEMON.MAC/NM
@BLDDEM
.IFT $DEL PIP DEMON.OLB;*/DE
.;.IFT $DEL PIP RECVAST.MAC;*/DE
```

```
.IFT $DEL PIP BLDDEM.CMD;*/DE
.IFT $DEL PIP PROLOGUE.OLB;*/DE
PIP 'DESTUI'/FO/CO/NV/CD=SY:'<UIC>'FTPDEMON.TSK/NM
PIP FTPDEMON.TSK;/DE/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]ARP.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]BSTAT.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]NETLOAD.TSK/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS5 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS5
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]NETSTAT.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]TTCP.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]XROUTE.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]FTPC.TSK/NM
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]TELNET.TSK/NM
DMO '$DEV'
;
;       Please mount floppy labelled EXOS6 in '$DEV'
;
.ASK MONT Press return when ready:
MOU '$DEV'EXOS6
PIP 'DESTUI'/FO/CO/NV/CD='$DEV'[1,1]FTPD.TSK/NM
.;
.;      copy specific programs
.;
.IFT $NOPRE PIP 'DESTUI'RHOST.C;*/DE
.IFT $NOPRE PIP 'DESTUI'RADDR.C;*/DE
.IFT $NOPRE PIP 'DESTUI'SOCKET.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.H;*/DE
.IFT $NOPRE PIP LB:[1,2]NET.;*/DE
.IFT $NOPRE PIP 'DESTUI'8030.HLP;*/DE
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]RHOST.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]RADDR.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]SOCKET.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]TTCP.C/NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]TTCP.H/NM
PIP LB:[1,2]/FO/NV/CD='$DEV'[1,1]NET./NM
PIP 'DESTUI'/FO/NV/CD='$DEV'[1,1]8030.HLP/NM
.ASK INITHO Do you want to initialize the network addresses file (HOSTS.NET)
.IFF INITHO .GOTO SETLD
.IFT $NOPRE PIP LB:[1,1]HOSTS.NET;*/DE
PIP LB:[1,1]/FO/NV/CD='$DEV'[1,1]HOSTS.NET/NM
.OPENA LB:[1,1]HOSTS.NET
.ASKS HNAME Name of host
.ASKS HADDR Host internet address
.DATA 'HADDR' 'HNAME' localhost
.CLOSE
.IFT $NOPRE PIP LB:[1,1]HOSTLOCAL.NET;*/DE
PIP LB:[1,1]/FO/NV/CD='$DEV'[1,1]HOSTLOCAL.NET/NM
.;
.;      Write out the EXOSLOAD command file
.;
```

```
.SETLD:
.IFT $NOPRE PIP LB:[1,1]EXOSLOAD.CMD;*/DE
    .OPEN LB:[1,1]EXOSLOAD.CMD
    .DATA .ENABLE SUBSTITUTION
    .DATA .IFACT DEMT0 ABO DEMT0
    .DATA .IFACT LGNT0 ABO LGNT0
    .DATA .IFACT ...DEM ABO ...DEM
    .DATA .IFACT ...LGN ABO ...LGN
    .SETN LCOUNT 0
.80$:
    .IF   LCOUNT >= '$SESS' .GOTO 89$
    .DATA .IFACT FTD00'LCOUNT' ABO FTD00'LCOUNT'
    .DATA .IFINS FTD00'LCOUNT' REM FTD00'LCOUNT'
    .DATA .IFINS XDR00'LCOUNT' REM XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 80$
.89$:
    .DATA .IFINS ...DEM REM ...DEM
    .DATA .IFINS ...ARP REM ...ARP
    .DATA .IFINS ...BST REM ...BST
    .DATA .IFINS ...FTP REM ...FTP
    .DATA .IFINS ...NET REM ...NET
    .DATA .IFINS ...TEL REM ...TEL
    .DATA .IFINS ...TTC REM ...TTC
    .DATA .IFINS ...ROU REM ...ROU
    .DATA .IFINS ...NST REM ...NST
    .DATA .IFINS ...LGN REM ...LGN
    .DATA .IFACT ...RTH ABO ...RTH
    .DATA .IFACT RTHT0 ABO RTHT0
    .DATA .IFINS ...RTH REM ...RTH
    .DATA .IF <SYSTEM> <> 6 .GOTO 10$
    .DATA .IFLOA ZE: CON OFFLINE ZEA
    .DATA .IFLOA ZE: CON OFFLINE ZE0:
    .DATA .IFNLOA ZT: .GOTO 10$
    .DATA CON OFFLINE ZTA
    .DATA CON OFFLINE ZTB
    .DATA CON OFFLINE ZTC
    .DATA CON OFFLINE ZTD
    .DATA CON OFFLINE ZTE
    .DATA CON OFFLINE ZTF
    .DATA CON OFFLINE ZTH
    .DATA CON OFFLINE ZTJ
    .DATA CON OFFLINE ZT0:
    .DATA CON OFFLINE ZT1:
    .DATA CON OFFLINE ZT2:
    .DATA CON OFFLINE ZT3:
    .DATA CON OFFLINE ZT4:
    .DATA CON OFFLINE ZT5:
    .DATA CON OFFLINE ZT6:
    .DATA CON OFFLINE ZT7:
    .DATA .10$:
    .DATA .IFLOA ZE: UNL ZE:
    .DATA .IFLOA ZT: UNL ZT:
    .DATA LOA ZE:/PAR=GEN/HIGH/SIZE=20000
    .DATA .IF <SYSTEM> <> 6 LOA ZT:
    .DATA .IF <SYSTEM> <> 6 UNL ZT:
```

```
    .DATA ; You can ignore the error message: "Loadable driver larger than 4KW"
    .DATA LOA ZT:/HIGH/SIZE=20000
    .DATA .IF <SYSTEM> <> 6 .GOTO 20$
    .DATA ;     configure the devices online
    .DATA CON ONLINE ZEA
    .DATA CON ONLINE ZE0:
    .DATA CON SET ZTA VEC=0
    .DATA CON SET ZTB VEC=0
    .DATA CON SET ZTC VEC=0
    .DATA CON SET ZTD VEC=0
    .DATA CON SET ZTE VEC=0
    .DATA CON SET ZTF VEC=0
    .DATA CON SET ZTH VEC=0
    .DATA CON SET ZTJ VEC=0
    .DATA CON ONLINE ALL
    .DATA .20$:
    .DATA INS $RTHACP/PRI=150.
    .DATA .XQT RTH
    .DATA INS 'DESTUI'ARP.TSK
    .DATA INS 'DESTUI'BSTAT.TSK
    .DATA INS 'DESTUI'FTPC.TSK
    .DATA INS 'DESTUI'FTPDEMON.TSK
    .DATA INS 'DESTUI'NETLOAD.TSK
    .DATA INS 'DESTUI'TELNET.TSK
    .DATA INS 'DESTUI'TTCP.TSK
    .DATA INS 'DESTUI'XROUTE.TSK
    .DATA INS 'DESTUI'NETSTAT.TSK
    .DATA INS 'DESTUI'LOGIN.TSK
    .SETN LCOUNT 0
    .DATA .SETS FTDOPT ""
    .DATA .IF <SYSTEM> = 6 .SETS FTDOPT "/XHR=NO"
.90$:
    .IF   LCOUNT >= '$SESS' .GOTO 99$
    .DATA INS 'DESTUI'FTPD.TSK/TASK=FTD00'LCOUNT'''FTDOPT''
    .DATA INS $PIP/TASK=XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 90$
.99$:
    .DATA .ASK DWN Do you want to initialize the EXOS front end processor
    .DATA .IFT DWN net
    .DATA .ASK DMN Do you want to start the FTP server
    .DATA .IFT DMN .XQT dem
    .DATA .IFT DMN .XQT lgn
    .CLOSE
    PIP LB:[1,1]EXOSLOAD.CMD/PR/FO
;
;       Please add the following line to LB:[1,2]STARTUP.CMD so that the
;       network is reloaded everytime the system is rebooted.
;
;       @LB:[1,1]EXOSLOAD
;
;       You may need to edit the file LB:[1,1]EXOSLOAD.CMD to set up the
;       options in loading the network module.
;
.;
.;
```

```
.;      dismount device
.;
DMO '$DEV'
;
;       Installation completed.  Now you can execute
;       @LB:[1,1]EXOSLOAD
;       to start up the network connection.
```

```
;
;       COPYRIGHT (c) 1985 BY EXCELAN, INC.
;       SAN JOSE, CALIFORNIA.  ALL RIGHTS RESERVED.
;
.; THIS  SOFTWARE  IS FURNISHED UNDER A LICENSE AND MAY
.; BE USED AND  COPIED  ONLY  IN  ACCORDANCE  WITH  THE
.; TERMS  OF SUCH LICENSE AND WITH THE INCLUSION OF THE
.; ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY  OTHER
.; COPIES  THEREOF  MAY  NOT  BE  PROVIDED OR OTHERWISE
.; MADE AVAILABLE TO ANY OTHER  PERSON.   NO  TITLE  TO
.; AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
.;
.; THE  INFORMATION  IN  THIS  SOFTWARE  IS  SUBJECT TO
.; CHANGE WITHOUT NOTICE AND SHOULD  NOT  BE  CONSTRUED
.; AS  A  COMMITMENT  BY EXCELAN, INC.
.;
.; EXCELAN, INC. ASSUMES NO RESPONSIBILITY  FOR  THE  USE
.; OR RELIABILITY  OF  ITS  SOFTWARE  ON EQUIPMENT THAT IS
.; NOT SUPPLIED BY EXCELAN, INC.
        .ENABLE QUIET
        .ENABLE LOWERCASE
        .ENABLE GLOBAL
        .ENABLE SUBSTITUTION
.IFT <PRIVIL> .GOTO 5
.DISABLE QUIET
; Error: You must be privileged in order to install EXOS 8030 software.
.EXIT
.5:
.IF <UIC> = "[1,100]" .GOTO 7
.DISABLE QUIET
; Error: EXOS 8030 must be installed from UIC [1,100]
.EXIT
.7:
        .DISABLE DISPLAY
        .ASK $VRBS Verbose? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
        .ASK $DEL Delete source file from current UFD in target disk? [Y/N]
        .ASK $DRV Build driver and ACP only? [Y/N]
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .SETS $PORT "4000"
        .ASKS [::$PORT] $PORT OFFSET ADDRESS OF PORTA ? [ D : 4000 ]
        .SETN $SESS 1
        .ASKN [::$SESS] $SESS Maximum number of concurrent FTP server sessions? [D : 1]
.;
.;      build the driver
.;
@BLDDRV
.IFT $DEL PIP BLDDRV.CMD;/DE
.;
.;      build the pseudo-terminal driver
@BLDZT
.IFT $DEL PIP BLDZT.CMD;/DE
.;
.;      build the ACP
```

```
.;
@BLDACP
.IFT $DEL .AND .IFT $DRV PIP PROLOGUE.OLB;/DE
.IFT $DEL PIP RTH.OLB;/DE
.IFT $DEL PIP BLDACP.CMD;/DE
.;
.;      Now copy utilities to various destination location
.;
.IFT $DRV .EXIT
.20:
.ASKS DESTUI Please enter the UFD for the EXOS utilities
.IF DESTUI = "" .GOTO 20
.;
.;      Copy task image
.;
.IFF $NOPRE .GOTO 25
PIP 'DESTUI'ARP.TSK;*/DE
PIP 'DESTUI'BSTAT.TSK;*/DE
PIP 'DESTUI'NETLOAD.TSK;*/DE
PIP 'DESTUI'NETSTAT.TSK;*/DE
PIP 'DESTUI'TTCP.TSK;*/DE
PIP 'DESTUI'XROUTE.TSK;*/DE
PIP 'DESTUI'FTPC.TSK;*/DE
PIP 'DESTUI'FTPDEMON.TSK;*/DE
PIP 'DESTUI'TELNET.TSK;*/DE
PIP 'DESTUI'LOGIN.TSK;*/DE
PIP 'DESTUI'FTPD.TSK;*/DE
.25:
@BLDLGN
.IFT $DEL PIP LOGIN.OLB;*/DE
.IFT $DEL PIP BLDLGN.CMD;*/DE
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'LOGIN.TSK/NM
@BLDDEM
.IFT $DEL PIP DEMON.OLB;*/DE
.;.IFT $DEL PIP RECVAST.MAC;*/DE
.IFT $DEL PIP BLDDEM.CMD;*/DE
.IFT $DEL PIP PROLOGUE.OLB;*/DE
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'FTPDEMON.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'ARP.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'BSTAT.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'NETLOAD.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'NETSTAT.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'TTCP.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'XROUTE.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'FTPC.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'TELNET.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'FTPD.TSK/NM
.;
.;      copy specific programs
.;
.IFT $NOPRE PIP 'DESTUI'RHOST.C;*/DE
.IFT $NOPRE PIP 'DESTUI'RADDR.C;*/DE
.IFT $NOPRE PIP 'DESTUI'SOCKET.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.H;*/DE
.IFT $NOPRE PIP LB:[1,2]NET.;*/DE
```

```
.IFT $NOPRE PIP 'DESTUI'8030.HLP;*/DE
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'RHOST.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'RADDR.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'SOCKET.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'TTCP.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'TTCP.H/NM
PIP LB:[1,2]/RE/FO/NV/CD=SY:'<UIC>'NET./NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'8030.HLP/NM
.ASK INITHO Do you want to initialize the network addresses file (HOSTS.NET)
.IFF INITHO .GOTO SETLD
.IFT $NOPRE PIP LB:[1,1]HOSTS.NET;*/DE
PIP LB:[1,1]/RE/FO/NV/CD=SY:'<UIC>'HOSTS.NET/NM
.OPENA LB:[1,1]HOSTS.NET
.ASKS HNAME Name of host
.ASKS HADDR Host internet address
.DATA 'HADDR' 'HNAME' localhost
.CLOSE
.IFT $NOPRE PIP LB:[1,1]HOSTLOCAL.NET;*/DE
PIP LB:[1,1]/RE/FO/NV/CD=SY:'<UIC>'HOSTLOCAL.NET/NM
.;
.;      Write out the EXOSLOAD command file
.;
.SETLD:
.IFT $NOPRE PIP LB:[1,1]EXOSLOAD.CMD;*/DE
    .OPEN LB:[1,1]EXOSLOAD.CMD
    .DATA .ENABLE SUBSTITUTION
    .DATA .IFACT DEMT0 ABO DEMT0
    .DATA .IFACT LGNT0 ABO LGNT0
    .DATA .IFACT ...DEM ABO ...DEM
    .DATA .IFACT ...LGN ABO ...LGN
    .SETN LCOUNT 0
.80$:
    .IF   LCOUNT >= '$SESS' .GOTO 89$
    .DATA .IFACT FTD00'LCOUNT' ABO FTD00'LCOUNT'
    .DATA .IFINS FTD00'LCOUNT' REM FTD00'LCOUNT'
    .DATA .IFINS XDR00'LCOUNT' REM XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 80$
.89$:
    .DATA .IFINS ...DEM REM ...DEM
    .DATA .IFINS ...ARP REM ...ARP
    .DATA .IFINS ...BST REM ...BST
    .DATA .IFINS ...FTP REM ...FTP
    .DATA .IFINS ...NET REM ...NET
    .DATA .IFINS ...TEL REM ...TEL
    .DATA .IFINS ...TTC REM ...TTC
    .DATA .IFINS ...ROU REM ...ROU
    .DATA .IFINS ...NST REM ...NST
    .DATA .IFINS ...LGN REM ...LGN
    .DATA .IFACT ...RTH ABO ...RTH
    .DATA .IFACT RTHT0 ABO RTHT0
    .DATA .IFINS ...RTH REM ...RTH
    .DATA .IF <SYSTEM> <> 6 .GOTO 10$
    .DATA .IFLOA ZE: CON OFFLINE ZEA
    .DATA .IFLOA ZE: CON OFFLINE ZE0:
    .DATA .IFNLOA ZT: .GOTO 10$
```

```
    .DATA CON OFFLINE ZTA
    .DATA CON OFFLINE ZTB
    .DATA CON OFFLINE ZTC
    .DATA CON OFFLINE ZTD
    .DATA CON OFFLINE ZTE
    .DATA CON OFFLINE ZTF
    .DATA CON OFFLINE ZTH
    .DATA CON OFFLINE ZTJ
    .DATA CON OFFLINE ZT0:
    .DATA CON OFFLINE ZT1:
    .DATA CON OFFLINE ZT2:
    .DATA CON OFFLINE ZT3:
    .DATA CON OFFLINE ZT4:
    .DATA CON OFFLINE ZT5:
    .DATA CON OFFLINE ZT6:
    .DATA CON OFFLINE ZT7:
    .DATA .10$:
    .DATA .IFLOA ZE: UNL ZE:
    .DATA .IFLOA ZT: UNL ZT:
    .DATA LOA ZE:/PAR=GEN/HIGH/SIZE=20000
    .DATA .IF <SYSTEM> <> 6 LOA ZT:
    .DATA .IF <SYSTEM> <> 6 UNL ZT:
    .DATA ; You can ignore the error message: "Loadable driver larger than 4KW"
    .DATA LOA ZT:/HIGH/SIZE=20000
    .DATA .IF <SYSTEM> <> 6 .GOTO 20$
    .DATA ;     configure the devices online
    .DATA CON ONLINE ZEA
    .DATA CON ONLINE ZE0:
    .DATA CON SET ZTA VEC=0
    .DATA CON SET ZTB VEC=0
    .DATA CON SET ZTC VEC=0
    .DATA CON SET ZTD VEC=0
    .DATA CON SET ZTE VEC=0
    .DATA CON SET ZTF VEC=0
    .DATA CON SET ZTH VEC=0
    .DATA CON SET ZTJ VEC=0
    .DATA CON ONLINE ALL
    .DATA .20$:
    .DATA INS $RTHACP/PRI=150.
    .DATA .XQT RTH
    .DATA INS 'DESTUI'ARP.TSK
    .DATA INS 'DESTUI'BSTAT.TSK
    .DATA INS 'DESTUI'FTPC.TSK
    .DATA INS 'DESTUI'FTPDEMON.TSK
    .DATA INS 'DESTUI'NETLOAD.TSK
    .DATA INS 'DESTUI'TELNET.TSK
    .DATA INS 'DESTUI'TTCP.TSK
    .DATA INS 'DESTUI'XROUTE.TSK
    .DATA INS 'DESTUI'NETSTAT.TSK
    .DATA INS 'DESTUI'LOGIN.TSK
    .SETN LCOUNT 0
    .DATA .SETS FTDOPT ""
    .DATA .IF <SYSTEM> = 6 .SETS FTDOPT "/XHR=NO"
.90$:
    .IF   LCOUNT >= '$SESS' .GOTO 99$
    .DATA INS 'DESTUI'FTPD.TSK/TASK=FTD00'LCOUNT'''FTDOPT''
```

```
    .DATA INS $PIP/TASK=XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 90$
.99$:
    .DATA .ASK DWN Do you want to initialize the EXOS front end processor
    .DATA .IFT DWN net
    .DATA .ASK DMN Do you want to start the FTP server
    .DATA .IFT DMN .XQT dem
    .DATA .IFT DMN .XQT lgn
    .CLOSE
    PIP LB:[1,1]EXOSLOAD.CMD/PR/FO
;
;       Please add the following line to LB:[1,2]STARTUP.CMD so that the
;       network is reloaded everytime the system is rebooted.
;
;       @LB:[1,1]EXOSLOAD
;
;       You may need to edit the file LB:[1,1]EXOSLOAD.CMD to set up the
;       options in loading the network module.
;
;
;       Installation completed.  Now you can execute
;       @LB:[1,1]EXOSLOAD
;       to start up the network connection.
```

```
;
;       COPYRIGHT (c) 1985 BY EXCELAN, INC.
;       SAN JOSE, CALIFORNIA.  ALL RIGHTS RESERVED.
;
.; THIS  SOFTWARE  IS FURNISHED UNDER A LICENSE AND MAY
.; BE USED AND  COPIED  ONLY  IN  ACCORDANCE  WITH  THE
.; TERMS  OF SUCH LICENSE AND WITH THE INCLUSION OF THE
.; ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY  OTHER
.; COPIES  THEREOF  MAY  NOT  BE  PROVIDED OR OTHERWISE
.; MADE AVAILABLE TO ANY OTHER  PERSON.   NO  TITLE  TO
.; AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
.;
.; THE  INFORMATION  IN  THIS  SOFTWARE  IS  SUBJECT TO
.; CHANGE WITHOUT NOTICE AND SHOULD  NOT  BE  CONSTRUED
.; AS  A  COMMITMENT  BY EXCELAN, INC.
.;
.; EXCELAN, INC. ASSUMES NO RESPONSIBILITY  FOR  THE  USE
.; OR RELIABILITY  OF  ITS  SOFTWARE  ON EQUIPMENT THAT IS
.; NOT SUPPLIED BY EXCELAN, INC.
        .ENABLE QUIET
        .ENABLE LOWERCASE
        .ENABLE GLOBAL
        .ENABLE SUBSTITUTION
.IFT <PRIVIL> .GOTO 5
.DISABLE QUIET
; Error: You must be privileged in order to install EXOS 8030 software.
.EXIT
.5:
.IF <UIC> = "[1,100]" .GOTO 7
.DISABLE QUIET
; Error: EXOS 8030 must be installed from UIC [1,100]
.EXIT
.7:
        .DISABLE DISPLAY
        .ASK $VRBS Verbose? [Y/N]
        .IFT $VRBS .DISABLE QUIET
        .ASK $NOPRE Delete previous version of EXOS software? [Y/N]
        .ASK $DEL Delete source file from current UFD in target disk? [Y/N]
        .ASK $DRV Build driver and ACP only? [Y/N]
        .SETS $VEC "400"
        .ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
        .SETS $PORT "4000"
        .ASKS [::$PORT] $PORT OFFSET ADDRESS OF PORTA ? [ D : 4000 ]
        .SETN $SESS 1
        .ASKN [::$SESS] $SESS Maximum number of concurrent FTP server sessions? [D : 1]
.;
.;      build the driver
.;
@BLDDRV
.IFT $DEL PIP BLDDRV.CMD;/DE
.;
.;      build the pseudo-terminal driver
@BLDZT
.IFT $DEL PIP BLDZT.CMD;/DE
.;
.;      build the ACP
```

```
.;
@BLDACP
.IFT $DEL .AND .IFT $DRV PIP PROLOGUE.OLB;/DE
.IFT $DEL PIP RTH.OLB;/DE
.IFT $DEL PIP BLDACP.CMD;/DE
.;
.;      Now copy utilities to various destination location
.;
.IFT $DRV .EXIT
.20:
.ASKS DESTUI Please enter the UFD for the EXOS utilities
.IF DESTUI = "" .GOTO 20
.;
.;      Copy task image
.;
.IFF $NOPRE .GOTO 25
PIP 'DESTUI'ARP.TSK;*/DE
PIP 'DESTUI'BSTAT.TSK;*/DE
PIP 'DESTUI'NETLOAD.TSK;*/DE
PIP 'DESTUI'NETSTAT.TSK;*/DE
PIP 'DESTUI'TTCP.TSK;*/DE
PIP 'DESTUI'XROUTE.TSK;*/DE
PIP 'DESTUI'FTPC.TSK;*/DE
PIP 'DESTUI'FTPDEMON.TSK;*/DE
PIP 'DESTUI'TELNET.TSK;*/DE
PIP 'DESTUI'LOGIN.TSK;*/DE
PIP 'DESTUI'FTPD.TSK;*/DE
.25:
@BLDLGN
.IFT $DEL PIP LOGIN.OLB;*/DE
.IFT $DEL PIP BLDLGN.CMD;*/DE
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'LOGIN.TSK/NM
@BLDDEM
.IFT $DEL PIP DEMON.OLB;*/DE
.;.IFT $DEL PIP RECVAST.MAC;*/DE
.IFT $DEL PIP BLDDEM.CMD;*/DE
.IFT $DEL PIP PROLOGUE.OLB;*/DE
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'FTPDEMON.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'ARP.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'BSTAT.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'NETLOAD.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'NETSTAT.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'TTCP.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'XROUTE.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'FTPC.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'TELNET.TSK/NM
PIP 'DESTUI'/RE/FO/CO/NV/CD=SY:'<UIC>'FTPD.TSK/NM
.;
.;      copy specific programs
.;
.IFT $NOPRE PIP 'DESTUI'RHOST.C;*/DE
.IFT $NOPRE PIP 'DESTUI'RADDR.C;*/DE
.IFT $NOPRE PIP 'DESTUI'SOCKET.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.C;*/DE
.IFT $NOPRE PIP 'DESTUI'TTCP.H;*/DE
.IFT $NOPRE PIP LB:[1,2]NET.;*/DE
```

```
.IFT $NOPRE PIP 'DESTUI'8030.HLP;*/DE
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'RHOST.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'RADDR.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'SOCKET.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'TTCP.C/NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'TTCP.H/NM
PIP LB:[1,2]/RE/FO/NV/CD=SY:'<UIC>'NET./NM
PIP 'DESTUI'/RE/FO/NV/CD=SY:'<UIC>'8030.HLP/NM
.ASK INITHO Do you want to initialize the network addresses file (HOSTS.NET)
.IFF INITHO .GOTO SETLD
.IFT $NOPRE PIP LB:[1,1]HOSTS.NET;*/DE
PIP LB:[1,1]/RE/FO/NV/CD=SY:'<UIC>'HOSTS.NET/NM
.OPENA LB:[1,1]HOSTS.NET
.ASKS HNAME Name of host
.ASKS HADDR Host internet address
.DATA 'HADDR' 'HNAME' localhost
.CLOSE
.IFT $NOPRE PIP LB:[1,1]HOSTLOCAL.NET;*/DE
PIP LB:[1,1]/RE/FO/NV/CD=SY:'<UIC>'HOSTLOCAL.NET/NM
.;
.;      Write out the EXOSLOAD command file
.;
.SETLD:
.IFT $NOPRE PIP LB:[1,1]EXOSLOAD.CMD;*/DE
    .OPEN LB:[1,1]EXOSLOAD.CMD
    .DATA .ENABLE SUBSTITUTION
    .DATA .IFACT DEMT0 ABO DEMT0
    .DATA .IFACT LGNT0 ABO LGNT0
    .DATA .IFACT ...DEM ABO ...DEM
    .DATA .IFACT ...LGN ABO ...LGN
    .SETN LCOUNT 0
.80$:
    .IF   LCOUNT >= '$SESS' .GOTO 89$
    .DATA .IFACT FTD00'LCOUNT' ABO FTD00'LCOUNT'
    .DATA .IFINS FTD00'LCOUNT' REM FTD00'LCOUNT'
    .DATA .IFINS XDR00'LCOUNT' REM XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 80$
.89$:
    .DATA .IFINS ...DEM REM ...DEM
    .DATA .IFINS ...ARP REM ...ARP
    .DATA .IFINS ...BST REM ...BST
    .DATA .IFINS ...FTP REM ...FTP
    .DATA .IFINS ...NET REM ...NET
    .DATA .IFINS ...TEL REM ...TEL
    .DATA .IFINS ...TTC REM ...TTC
    .DATA .IFINS ...ROU REM ...ROU
    .DATA .IFINS ...NST REM ...NST
    .DATA .IFINS ...LGN REM ...LGN
    .DATA .IFACT ...RTH ABO ...RTH
    .DATA .IFACT RTHT0 ABO RTHT0
    .DATA .IFINS ...RTH REM ...RTH
    .DATA .IF <SYSTEM> <> 6 .GOTO 10$
    .DATA .IFLOA ZE: CON OFFLINE ZEA
    .DATA .IFLOA ZE: CON OFFLINE ZE0:
    .DATA .IFNLOA ZT: .GOTO 10$
```

```
    .DATA CON OFFLINE ZTA
    .DATA CON OFFLINE ZTB
    .DATA CON OFFLINE ZTC
    .DATA CON OFFLINE ZTD
    .DATA CON OFFLINE ZTE
    .DATA CON OFFLINE ZTF
    .DATA CON OFFLINE ZTH
    .DATA CON OFFLINE ZTJ
    .DATA CON OFFLINE ZT0:
    .DATA CON OFFLINE ZT1:
    .DATA CON OFFLINE ZT2:
    .DATA CON OFFLINE ZT3:
    .DATA CON OFFLINE ZT4:
    .DATA CON OFFLINE ZT5:
    .DATA CON OFFLINE ZT6:
    .DATA CON OFFLINE ZT7:
    .DATA .10$:
    .DATA .IFLOA ZE: UNL ZE:
    .DATA .IFLOA ZT: UNL ZT:
    .DATA LOA ZE:/PAR=GEN/HIGH/SIZE=20000
    .DATA .IF <SYSTEM> <> 6 LOA ZT:
    .DATA .IF <SYSTEM> <> 6 UNL ZT:
    .DATA ; You can ignore the error message: "Loadable driver larger than 4KW"
    .DATA LOA ZT:/HIGH/SIZE=20000
    .DATA .IF <SYSTEM> <> 6 .GOTO 20$
    .DATA ;     configure the devices online
    .DATA CON ONLINE ZEA
    .DATA CON ONLINE ZE0:
    .DATA CON SET ZTA VEC=0
    .DATA CON SET ZTB VEC=0
    .DATA CON SET ZTC VEC=0
    .DATA CON SET ZTD VEC=0
    .DATA CON SET ZTE VEC=0
    .DATA CON SET ZTF VEC=0
    .DATA CON SET ZTH VEC=0
    .DATA CON SET ZTJ VEC=0
    .DATA CON ONLINE ALL
    .DATA .20$:
    .DATA INS $RTHACP/PRI=150.
    .DATA .XQT RTH
    .DATA INS 'DESTUI'ARP.TSK
    .DATA INS 'DESTUI'BSTAT.TSK
    .DATA INS 'DESTUI'FTPC.TSK
    .DATA INS 'DESTUI'FTPDEMON.TSK
    .DATA INS 'DESTUI'NETLOAD.TSK
    .DATA INS 'DESTUI'TELNET.TSK
    .DATA INS 'DESTUI'TTCP.TSK
    .DATA INS 'DESTUI'XROUTE.TSK
    .DATA INS 'DESTUI'NETSTAT.TSK
    .DATA INS 'DESTUI'LOGIN.TSK
    .SETN LCOUNT 0
    .DATA .SETS FTDOPT ""
    .DATA .IF <SYSTEM> = 6 .SETS FTDOPT "/XHR=NO"
.90$:
    .IF   LCOUNT >= '$SESS' .GOTO 99$
    .DATA INS 'DESTUI'FTPD.TSK/TASK=FTD00'LCOUNT'''FTDOPT''
```

```
    .DATA INS $PIP/TASK=XDR00'LCOUNT'
    .INC LCOUNT
    .GOTO 90$
.99$:
    .DATA .ASK DWN Do you want to initialize the EXOS front end processor
    .DATA .IFT DWN net
    .DATA .ASK DMN Do you want to start the FTP server
    .DATA .IFT DMN .XQT dem
    .DATA .IFT DMN .XQT lgn
    .CLOSE
    PIP LB:[1,1]EXOSLOAD.CMD/PR/FO
;
;       Please add the following line to LB:[1,2]STARTUP.CMD so that the
;       network is reloaded everytime the system is rebooted.
;
;       @LB:[1,1]EXOSLOAD
;
;       You may need to edit the file LB:[1,1]EXOSLOAD.CMD to set up the
;       options in loading the network module.
;
;
;       Installation completed.  Now you can execute
;       @LB:[1,1]EXOSLOAD
;       to start up the network connection.
```

```
 1  /*
 2   * filename:    FTPDEMON.H
 3   */
 4
 5  #include        <rsxos.h>
 6  #define EXEFN           010001
 7  #define MAXCONN         4               /* max. no of connections */
 8  #define ACC_EFN         50              /* common event flag no. 50 */
 9  #define SLEEP_EFN       51              /* common event flag no. 51 */
10  #define TASKNAMLEN      6               /* length of task name */
11  #define FOREVER         for(;;)
12  #define SDRA    01153
13
14  struct  task_block {
15          struct  task_block *link;       /* link to next task block  */
16          char    task_name[TASKNAMLEN];  /* task name */
17          int     esb[8];                 /* exit status block  */
18        } tskblk[MAXCONN] = {0};
19
20  /* GLOBAL variables */
21
22  struct  task_block *rdy2run   = tskblk; /* ptr to rdy 2 run task */
23  struct  task_block *accept_on = 0;      /* pointer to task in accept */
24
25  char cmdlin[] = "INS LB:[1,2]FTPD/TASK=FTD00 ";
26  char line[] = "REM FTD00 ";
27  long cli = 0;   /* CLI name in RAD50 */
28  int  cmdlen = 0;
29  int  len = 0;
30  int  flgbuf[4] = {0};           /* event flag buffer */
31  /* int  connect = 1;             total no of connections */
32  char *ftpcmd = (char *) 0;
33  int ftplen = 0;
34  int tcblist[MAXCONN] = {0};     /* pointer to task control block  */
```

```
 1  /*
 2   * filename:    FTPDEMON.C
 3   */
 4
 5  /*
 6   * This file contains the code for the master ftp task which monitors the
 7   * generation of different ftp daemons for different connections.
 8   */
 9
10  #include "ftpdemon.h"
11  extern int ast();
12  extern long radix();
13  int connect = 1;
14  extern int ast_recv();
15   main()
16  {
17
18          priv_user();                    /* check user is priv &
19                                             task is not active     */
20          gmcr();
21          emt(SDRA,ast_recv);                     /* specify receive data ast */
22
23          initialize();
24
25          FOREVER {
26                  if(!read_efn(flgbuf)) {         /* is efn 50 clear?      */
27                          if(rdy2run) {           /* any rdy2run task present */
28                                  emt(SETF,ACC_EFN);/* set common efn 50 to
29                                                       indicate accept is on */
30                                  ins_spawn();    /* install and spawn one    */
31                                  update();       /* update rdy2run pointer   */
32                              }
33                          }
34
35                  emt(ENAR);      /* enable ast recognition */
36                  emt(STSE,SLEEP_EFN);            /* sleep */
37                  emt(CLEF,SLEEP_EFN);    /* clear sleep efn */
38                  emt(DSAR);      /* disable ast recognition so that it does */
39                                  /* not interfere with main task's execution */
40              }  /* end of FOREVER */
41  }
42
43  /*
44   * INITIALIZE
45   *
46   *              Initialize the world of MASTER
47   */
48
49   initialize()
50   {
51          register struct task_block *t = tskblk; /* start of task block */
52          int i,j;
53
54          cli = radix("MCR...");
55          cmdlen = strlen(cmdlin);
56          len = strlen(line);
```

```
57          for(i=0;i<connect;i++) {
58                  tcblist[i] = 0;
59                  for(j=0;j<5;j++)
60                      t->task_name[j] = cmdlin[cmdlen - TASKNAMLEN +j];
61                  t->task_name[j] = '0' + i;
62                  t->link = t + 1;
63                  t++;
64                }
65          (--t)->link = 0;
66          emt(CLEF,ACC_EFN);
67          emt(CLEF,SLEEP_EFN);
68          emt(DSAR);      /* disable ast recognition so that ast's do not */
69                          /* bother the masin task */
70  }
71
72  /*
73   * UPDATE
74   *
75   *              Update rdy2run pointer
76   */
77
78   update()
79   {
80
81          accept_on = rdy2run;
82          rdy2run = rdy2run->link;
83          accept_on->link = 0;
84  }
85
86  /*
87   * FROM_AST
88   *
89   *              This routine is called from the AST routine when a task
90   *              exits
91   */
92
93   from_ast(p)
94   int *p;        /* pointer to esb of exit task */
95   {
96          register struct task_block *exit_task;
97          int index;
98
99          exit_task = (struct task_block *)(p-4); /* point to start of str. */
100         exit_task->link = rdy2run;
101         rdy2run = exit_task;    /* make the exit task the next available */
102                                 /* rdy2run task */
103         line[len - 1] = exit_task->task_name[TASKNAMLEN -1];/* the task no. */
104         /*emt(SPWN,cli,0,0,0,0,EXEFN,0,0,line,len,0,CO);*/ /* rem task */
105         /*emt(WTSE,1);*/                /* wait for task to get removed */
106         index = exit_task - tskblk;
107         if(tcblist[index])
108                 mkpriv(tcblist[index]); /* make sure task becomes priv. */
109         if(accept_on == exit_task)
110            emt(CLEF,ACC_EFN);   /* then task has exit before accept */
111         emt(SETF,SLEEP_EFN);    /* unstop the master task */
112 }
```

```
113
114  /*
115   * INS_SPAWN
116   *
117   *              Install and spawn the next rdy2run task
118   */
119
120   ins_spawn()
121   {
122           char msg[26];
123           long t_name = radix(rdy2run->task_name);
124           int st;
125
126           cmdlin[cmdlen - 1] = rdy2run->task_name[TASKNAMLEN -1];
127
128           /* now install the task */
129
130           /*st = emt(SPWN,cli,0,0,0,0,EXEFN,0,0,cmdlin,cmdlen,0,CO);*/
131           /*emt(WTSE,1);*/        /* wait for task to get installed */
132           /* now spawn the task */
133           st = emt(SPWN,t_name,0,0,0,0,010000,ast,rdy2run->esb,ftpcmd,ftplen,0,0);
134           if(st == IE_ACT)
135                   ;       /* yet to decide what to do if task is active */
136                           /* such a condition should never arise but if it does*/
137                           /* then what? */
138
139  }
140
141  /* return the size of string */
142  strlen( s )
143  char *s;
144  {
145  char *p = s;
146
147   while( *p != '\0' ) p++;
148   return(p - s);
149  }
```

```
 1  ;
 2  ; FILENAME:     DEMON.MAC
 3  ;
 4  ;       This file includes AST service routine for demon. It also has
 5  ;       a routine to read EFN 50.
 6  ;
 7          .psect  c$text,i,ro
 8
 9          .MCALL  RDAF$S
10  READ.EFN::
11          jsr     R5,c$sav
12          MOV     4(R5),R0        ; pointer to a 4 word buffer
13          RDAF$S  R0              ; read all event flags
14          BIT     #2,6(R0)        ; check if event no. 50 is set or clear
15          BNE     10$             ; if NE then it is set
16          CLR     R0              ; clear return value also
17  10$:
18          jmp     c$ret           ; if efn is set then return value is > 0
19
20  ;
21  ; GMCR
22  ;
23          .psect  c$data,d,rw
24          .mcall  DIR$,GMCR$
25  GMCRD:
26          GMCR$
27          .psect  c$text,i,ro
28  GMCR::
29          JSR     R5,C$SAV                ; save registers
30          DIR$    #GMCRD                  ; get MCR command line
31  ;       CMP     #IE.AST,$DSW            ; check return status
32  ;       BEQ     NMCR                    ; if EQ No MCR Command
33          MOV     #GMCRD+G.MCRB,FTPCMD    ; get mcr buffer address
34          MOV     $DSW,FTPLEN             ; buffer size
35  NMCR:
36
37          JMP     C$RET                   ; unsave register's and return
38
39          .psect c$data,d,rw
40  tsk:
41
42          .IF DF  R$$MPL
43
44          .rad50  /DEMT0/
45
46          .IFF    ;R$$MPL
47
48          .rad50  /...DEM/
49
50          .ENDC   ;R$$MPL
51
52  rtncode:
53          .word   0
54  ER1:    .ASCIZ  /**FATAL**---- USER MUST BE PRIVILEGED/
55          .EVEN
56  ER2:
```

```
    57          .ASCIZ  /**FATAL**---- TASK ALREADY ACTIVE/
    58          .EVEN
    59
    60          .MCALL TCBDF$,UCBDF$,QIOW$S,EXIT$S,DCBDF$
    61          DCBDF$
    62          TCBDF$
    63          UCBDF$
    64
    65
    66          .psect c$text,i,ro
    67          .ENABL LSB
    68
    69  priv.user::
    70          JSR     R5,C$SAV
    71          CALL    $SWSTK,RET      ; switch to system state
    72          MOV     $TKTCB,R0       ;; get current TCB address
    73          MOV     T.UCB(R0),R1    ;; get TI: UCB address
    74          BIT     #U2.PRV,U.CW2(R1);; check user is priv.
    75          BEQ     ERR1            ;; If EQ user is not priv.
    76          CMP     tsk,T.NAM(R0)   ;; compare first word of task name
    77          BNE     ERR2            ;; If NE task already active.
    78          CMP     tsk+2,T.NAM+2(R0);; compare second word of task name
    79          BNE     ERR2            ;; if NE task already active
    80          MOV     #$DEVHD,R1      ;; get gevice header
    81  10$:
    82          MOV     (R1),R1         ;; get next DCB address
    83          BEQ     20$             ;; if EQ none
    84          CMP     #"CO,D.NAM(R1)  ;; is it console
    85          BNE     10$             ;; if NE no
    86          MOV     D.UCB(R1),T.UCB(R0)     ;; get CO UCB address
    87  20$:
    88          BR      RTN             ;;
    89  ERR1:
    90          MOV     #-1,RTNCODE     ;; user must be priv.
    91          BR      RTN
    92  ERR2:
    93          MOV     #-2,RTNCODE     ;; task already active
    94  RTN:
    95          RETURN                  ;; return to task state
    96  RET:
    97          MOV     RTNCODE,R0      ;; return value
    98          BEQ     SUCC
    99          CMP     #-1,R0          ; check error code
   100          BEQ     E1              ;
   101          BR      E2
   102  E1:     MOV     #ER1,R1         ; address of error message
   103          BR      ERMSG
   104  E2:     MOV     #ER2,R1         ; address of error message
   105          BR      ERMSG
   106  SUCC:
   107          JMP     C$RET
   108  ERMSG:
   109          QIOW$S  #IO.WVB,#5,#1,,,,<R1,#38.,#40>
   110          EXIT$S
   111
   112          .psect c$text,i,ro
```

```
113
114  mkpriv::
115          JSR     R5,C$SAV
116          MOV     4(R5),R0          ; get tcb address
117          CALL    $SWSTK,RET1       ; switch to system state
118          BIS     #T3.PRV,T.ST3(R0) ;; make server as priv.
119          RETURN
120  RET1:
121          JMP     C$RET
122          .DSABL LSB
123
124          .psect c$data,d,rw
125          .even
126          .psect  c$text,i,ro
127
128          .MCALL  ASTX$S
129  AST::
130          MOV     R0,-(SP)          ; save R0
131          MOV     R1,-(SP)          ; save R1
132          MOV     R2,-(SP)          ; save R2
133          MOV     R3,-(SP)          ; save R3
134          MOV     R4,-(SP)          ; save R4
135          MOV     R5,-(SP)          ; save R5
136          MOV     14(SP),-(SP)      ; 1st param is the esb address on the stack
137          JSR     PC,FROM.AST       ; call C - routine to do the job
138          TST     (SP)+             ; pop off param passed
139          MOV     (SP)+,R5          ; pop off R5
140          MOV     (SP)+,R4          ; pop off R4
141          MOV     (SP)+,R3          ; pop off R3
142          MOV     (SP)+,R2          ; pop off R2
143          MOV     (SP)+,R1          ; pop off R1
144          MOV     (SP)+,R0          ; pop off R0
145          TST     (SP)+             ; pop off stack for ast
146          ASTX$S                    ; exit from AST routine
147
148          .END
```

```
 1  ;
 2  ; filename:     RECVAST.MAC
 3  ;
 4          .title  RECVAST
 5          .MACRO  SAVE
 6
 7          MOV     R0,-(SP)
 8          MOV     R1,-(SP)
 9          MOV     R2,-(SP)
10          MOV     R3,-(SP)
11          MOV     R4,-(SP)
12          MOV     R5,-(SP)
13
14          .ENDM
15
16          .MACRO  UNSAVE
17
18          MOV     (SP)+,R5
19          MOV     (SP)+,R4
20          MOV     (SP)+,R3
21          MOV     (SP)+,R2
22          MOV     (SP)+,R1
23          MOV     (SP)+,R0
24
25          .ENDM
26
27          .MCALL TCBDF$,PCBDF$,HDRDF$,RCVD$S,SDAT$S,ASTX$S
28          TCBDF$
29          PCBDF$
30          HDRDF$
31  PKT:
32          .BLKW   15.
33  GRP:
34          .WORD   0
35  BASE:
36          .RAD50  /000/
37
38          .enabl  lsb
39
40  AST.RE::
41          SAVE                    ; save all registers
42  AGAIN:
43          RCVD$S  ,#PKT           ; recieve pkt from ftd000 task
44          CMP     #IS.SUC,$DSW    ;check for success
45          BEQ     10$             ; If EQ YES
46          CMP     #IE.ITS,$DSW    ; check error code
47          BEQ     EXT             ; no pkt. return
48          BR      ERR             ; error
49  10$:
50          CALL    $SWSTK,RET      ; switch to system state
51          MOV     $ACTHD,R0       ; get active task header pointer
52  20$:
53          CMP     T.NAM(R0),PKT   ;; compare first word of task
54          BNE     NXT             ;; If NE not match , next tcb
55          CMP     T.NAM+2(R0),PKT+2 ;; compare second word of task
56          BEQ     SUCC            ;; If EQ found tcb
```

```
57  NXT:
58          MOV     T.ACTL(R0),R0   ;; get next tcb address
59          CMP     R0,#$HEADR      ;; Check if it is last tcb
60          BEQ     30$             ;; If EQ yes
61          BR      20$             ;; Loop
62  SUCC:
63          MOVB    PKT+5,GRP       ;; get group
64          CMP     GRP,#10         ;; priv uic?
65          BLOS    25$             ;; br if yes
66          BIC     #T3.PRV,T.ST3(R0) ;; make child as non-priv.
67          MOV     PKT+2,R2        ;; get second word of task name
68          SUB     BASE,R2         ;; calculate index(word)
69          ASL     R2              ;; index(byte)
70          ADD     #TCBLIST,R2     ;;
71          MOV     R0,(R2)         ;; save tcb address
72  25$:
73          MOV     T.PCB(R0),R0    ;; get PCB address of task
74          MOV     P.HDR(R0),R0    ;; Get header control block
75          MOV     PKT+4,H.CUIC(R0) ;; set current task uic as remote user's
76                                  ;; login uic
77          MOV     PKT+4,H.DUIC(R0) ;; set default task uic
78
79  30$:
80          RETURN                  ;; switch to task state
81  RET:
82          SDAT$S  #PKT,#PKT+4     ; send dummy pkt to child task
83          BR      AGAIN           ; go for next pkt.
84  ERR:
85  EXT:
86          UNSAVE
87                          ; unsave all registered
88          ASTX$S                  ; exit from AST routine
89
90          .END
```

```
 1  #include <rsxos.h>
 2  #define EFN     1
 3  extern valacnt();
 4  extern char *entry;
 5  char *msg = "                                   ";
 6   main()
 7  {
 8          register int    i,r;
 9          char    *p;
10  for(;;) {
11          /*
12          if(emt(RCVX,(long) 0,msg) < 0)
13                  emt(EXST,-2);
14          */
15          if(emt(RCST,(long) 0,msg) == IS_SET)
16             continue;
17          for(i=4;msg[i] != '*';i++);
18          i++;
19          r = valacnt(msg+4,msg+i);
20          *((int *)(msg + 4)) = r;
21          if( r == 0){
22                  p = msg+6;
23                  *p++ = '[';
24                  for(i=0;i<3;i++)
25                          *p++ = *(entry + A_GRP + i );
26                  *p++ = ',';
27                  for(i=0;i<3;i++)
28                          *p++ = *(entry + A_MBR + i );
29                  *p++ = ']';
30                  *p++ = '\0';
31                  /* now fill in the login default device name starting at 16th */
32                  for(i=0;i<4;i++)
33                          *p++ = *(entry + A_SYDV + i);
34                  *p++ = '\0';
35          }
36          emt(SDAT,*(long*)msg,msg+4,0);
37     }
38  }
39
40  extern int namflg;
41  extern char *puic;
42
43  accnt(ac)
44  char    *ac;
45  {
46  int     hasbracket = 0;
47  int     charcount;
48  int     leadzero;       /* count of leading zeroes needed */
49  char    *chptr;
50  char    *delimiter;     /* delimiter    */
51
52          while (*ac == ' ')
53                  ac++;           /* skip blank   */
54          if ((*ac >= 'A') && (*ac <= 'Z')) {
55                  namflg = 1;
56                  return(1);
```

```
57          } else if (*ac == '[') {
58                  hasbracket = 1;
59                  ac++;
60          } else if ((*ac < '0') && (*ac > '7')) {
61                  return(2);
62          };
63
64          /* now must start with a numeric number */
65
66          chptr = ac;
67          charcount = 0;
68          while ((*chptr != ' ') && (*chptr != ',')) {
69                  if (++charcount > 3)
70                          return(2);      /* group number too long */
71                  chptr++;
72          };
73
74          delimiter = chptr;
75
76          for (leadzero = 3 - charcount; leadzero > 0; leadzero--)
77                  *puic++ = '0';
78          for (chptr = ac; charcount > 0; charcount--) {
79                  if ((*chptr < '0') || (*chptr > '7'))
80                          return(2);      /* syntax error */
81                  *puic++ = *chptr++;
82          };
83
84          while (*chptr == ' ')
85                  chptr++;                /* skip blank   */
86          if (*chptr == ',') {
87                  chptr++;
88          } else
89                  return(2);
90          while (*chptr == ' ')
91                  chptr++;                /* skip blank   */
92
93          /* now handle the member part   */
94          delimiter = chptr;
95          charcount = 0;
96          while ((*chptr != ' ') && (*chptr != ']') && (*chptr != '*')) {
97                  if (++charcount > 3)
98                          return(2);      /* member number too long */
99                  chptr++;
100         };
101
102         if ((*chptr == ']') && (!hasbracket))
103                 return(2);
104
105         for (leadzero = 3 - charcount; leadzero > 0; leadzero--)
106                 *puic++ = '0';
107         for (chptr = delimiter; charcount > 0; charcount--) {
108                 if ((*chptr < '0') || (*chptr > '7'))
109                         return(2);      /* syntax error */
110                 *puic++ = *chptr++;
111         };
112         if (hasbracket) {
```

```
113                     while (*chptr != ']') {
114                             if (*chptr == '*')
115                                     return(2);
116                             chptr++;
117                             };
118                     };
119             return(0);
120     }
```

```
;       .TITLE  ACTFIL - ACCOUNT FILE CONTROL BLOCKS
.NLIST
        .IDENT  /4.0/
;
;               COPYRIGHT (C) 1981 BY
;       DIGITAL EQUIPMENT CORPORATION, MAYNARD
;        MASSACHUSETTS.  ALL RIGHTS RESERVED.
;
; THIS  SOFTWARE  IS  FURNISHED  UNDER  A LICENSE AND MAY BE USED
; AND  COPIED  ONLY IN  ACCORDANCE WITH THE TERMS OF SUCH LICENSE
; AND WITH  THE INCLUSION  OF THE ABOVE  COPYRIGHT  NOTICE.  THIS
; SOFTWARE  OR ANY OTHER  COPIES  THEREOF, MAY NOT BE PROVIDED OR
; OTHERWISE MADE  AVAILABLE TO ANY OTHER PERSON.  NO TITLE TO AND
; OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERED.
;
; THE INFORMATION  IN THIS DOCUMENT IS SUBJECT  TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT  BY  DIGITAL
; EQUIPMENT CORPORATION.
;
; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
;
;
;-----------------------------------------------------------------------
;
; COPYRIGHT (C) 1981 BY DIGITAL EQUIPMENT CORPORATION.
; ALL RIGHTS RESERVED.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
; OR COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE.
;
;
; VERSION       04
; BY:           H. LEV
; DATE:         7/15/75
;
; MODIFIED:
;
;       EB051   21-MAY-77 LOOK FOR ACNT FILE ON LB: RATHER THAN SY:
;
;       DG002   LOOK FOR LATEST VERSION OF RSX11.SYS
;
;       MLG007  03-NOV-78  FIND PHYSICAL LB:
;
;       MLG044  30-JAN-79  SPOOL LISTING FILE (ACNT)
;
;       MLG081  10-APR-79  DO NOT LEAVE ACCOUNT FILE LOCKED
;
;
;       SA213   ADD FIELDS FOR SLAVE BIT, DEFAULT CLI NAME
;               AND CHANGE OPENING OF ACNT FILE
;
.LIST
        .MCALL  FDBDF$,FDOP$A,FSRSZ$

        .IF DF  R$$MPL
```

```
 57
 58          .MCALL  ACTDF$
 59
 60          .IFF    ;R$$MPL
 61
 62          .MACRO  ACTDF$,L,B
 63          .ASECT
 64  .=0
 65  A.GRP:'L'       .BLKB   3               ; GROUP CODE (ASCII)
 66  A.MBR:'L'       .BLKB   3               ; MEMBER CODE
 67  A.PSWD:'L'      .BLKB   6               ; PASSWORD
 68  A.LNM:'L'       .BLKB   14.             ; LAST NAME
 69  A.FNM:'L'       .BLKB   12.             ; FIRST NAME
 70  A.LDAT:'L'      .BLKB   6               ; DATE OF LAST LOG ON (DD/MM/YY HH:MM:SS
 71  A.NLOG:'L'      .BLKB   2               ; TOTAL NUMBER OF LOGONS
 72  A.SYDV:'L'      .BLKB   4               ; DEFAULT SYSTEM DEVICE
 73                  .BLKW   1               ; UNUSED
 74  A.CLI:'L'       .BLKW   2               ; RAD50 DEFAULT CLI NAME
 75                  .BLKW   2               ; UNUSED (FOR COMPATIBILITY W/ MPLUS)
 76  A.LPRV:'L'      .BLKW   1               ; LOGIN PRIVILEGE WORD
 77                  .BLKW   1               ; UNUSED
 78  A.LEN   ='B'    128.                    ; LENGTH OF CONTROL BLOCK
 79  ;
 80  ; BIT DEFINITION ON A.LPRV - LOGIN PRIVILEGES
 81  ;
 82  AL.SLV  ='B'    1               ; SLAVE TERMINAL ON LOGIN
 83          .PSECT
 84          .ENDM
 85
 86          .ENDC   ;R$$MPL
 87
 88  ;
 89  ; CONSTANTS
 90  ;
 91  LUN2    ==      2               ; ACCOUNT FILE LUN
 92  $BFLEN  ==      2048.           ; LENGTH OF ACCOUNT FILE BUFFER
 93
 94          ACTDF$  <:>,<=>         ; DEFINE OFFSETS INTO ACCOUNT FILE
 95
 96  $ACTFL:: FDBDF$         ; DEFINE ACCOUNT FILE FDB
 97
 98          .IF DF  R$$MPL
 99
100          FDOP$A  LUN2,DSPT,,,FA.ENB!FA.DLK!FA.EXC
101
102          .IFF    ;R$$MPL
103
104          FDOP$A  LUN2,DSPT,,,FA.ENB!FA.DLK ; SETUP LUN, DSD, AND F.ACTL
105
106          .ENDC   ;R$$MPL
107
108  DSPT:   .WORD   0       ; DATA SET DESCRIPTOR
109          .WORD   0       ; DEVICE NAME (ALUN USED)
110          .WORD   5               ;
111          .WORD   DIRNAM          ;
112          .WORD   9.
```

```
113          .WORD   FILNAM          ;
114
115  DIRNAM: .ASCII  /[0,0]/         ;
116  FILNAM: .ASCII  /RSX11.SYS/
117          .EVEN
118
119          FSRSZ$  1               ; SET UP FOR A FILE IN GET PUT MODE
120
121  $ACTBF:: .BLKB  $BFLEN          ; CREATE ACCOUNT FILE BUFFER
122          .EVEN
123          .END
```

```
 1  ; filename:     PASWORD.MAC
 2  ;
 3  ;
 4  ; This routine is callable from 'C' as well as from a Macro program.
 5  ; If C$$SPRT is defined then it becomes callable from 'C'.
 6  ;
 7
 8          .MCALL  DCBDF$
 9          DCBDF$
10  C$$SPRT = 1
11
12  ; DATABASE
13
14          .MCALL  QIO$,MRKT$,WTSE$S,QIOW$S,ALUN$S,CLOSE$
15          .MCALL  OPEN$R,FINIT$,GET$
16          .MCALL  NBOF$L
17
18          .IF DF  R$$MPL
19
20          .MCALL  OPNS$U
21
22          .IFF    ;R$$MPL
23
24          .MCALL  OPEN$U
25
26          .ENDC   ;R$$MPL
27
28          .psect  c$data,d,rw
29  .enabl    gbl
30
31          ENCRPT = 0              ; ENCRYPTION SUBROUTINE NOT PRESENT
32          LUN4 = 4                ; LUN FOR SYSTEM DEVICE
33          EFN1 = 1                ; EVENT FLAG FOR ALL I/O
34          PSWDBF: .WORD   0       ; ADDRESS OF PASSWORD BUFFER
35          UIC:    .ASCII  /000000/; UIC
36          PUIC::  .WORD   UIC     ; POINTER TO UIC
37          NAME:   .ASCII  /               / ; LAST NAME AREA IF NAME USED
38          .EVEN
39          NBOF$L          ; DEFINE BLOCK OFFSETS
40
41          FDPB:   QIO$    IO.RVB,LUN2,EFN1,,IOSB,,<$ACTBF,$BFLEN,,,1>
42          IOSB:   .BLKW   2               ; I/O STATUS BLOCK
43          OPNERR: .WORD   0               ; A/C FILE OPEN ERROR FLAG
44          FILOPN: .WORD   0               ; FILE OPEN IF = 1.
45          NAMFLG::        .WORD   0               ; NAME FLAG, 0 = A/C, 1 = NAME
46          ENTRY:: .WORD   0               ; ADDRESS OF A/C ENTRY
47          MKT:    MRKT$   1,60,1          ; WAIT FOR 1 SEC
48          FRMPTR: .WORD   0               ; C - FRAME POINTER STORAGE
49          ER1:    .ASCIZ  <15>/**FATAL**----CANNOT FIND PHYSICAL LB:/
50          ER2:    .ASCIZ  <15>/**FATAL**----ACCOUNT FILE OPEN ERROR/
51          ER3:    .ASCIZ  <15>/**FATAL**----INVALID ACCOUNT/
52          .EVEN
53
54  ;
55  ;       INPUTS TO MAC CALLABLE ROUTINE
56  ;       R3 --> POINTER TO ACCOUNT
```

```
;       R4 --> POINTER TO PASSWORD BUFFER
;

        .psect  c$text,i,ro

VALACNT::

        .IF     DF      C$$SPRT
        jsr     R5,c$sav
        MOV     R5,FRMPTR       ; SAVE FRAME POINTER
        MOV     4(R5),R3        ; GET POINTER TO ACCOUNT OR NAME
        MOV     6(R5),R4        ; GET POINTER TO PASSWORD
        .ENDC

;
; NOW FILL UP UIC AND PASWORD IN THEIR RESPECTIVE PLACES
;
        MOV     #UIC,PUIC       ; set up pointer to UIC
        MOV     R3,(SP)         ; PARAM -> POINTER TO ACCOUNT OR NAME
        CALL    ACCNT           ; CHECK FOR ACCOUNT OR USER NAME
        TST     R0              ; RETURN CODE
        BEQ     15$             ; IF EQ THEN ACCNT SPECIFIED CORRECTLY
                                ; AND XFERED UIC TO CORRECT PLACE
        CMP     R0,#1           ; SEE IF NAME SPECIFIED OR NOT
        BEQ     10$             ; IF EQ THEN IT IS SPECIFIED
        JMP     ERR3            ; SYNTAX ERROR
10$:
        MOV     #NAME,R2        ; ADDRESS OF NAME
        MOV     #14.,R1         ; LENGTH OF NAME
12$:
        MOVB    (R3)+,(R2)+     ; XFER NAME
        SOB     R1,12$          ; LOOP

15$:
        MOV     R4,PSWDBF       ; ADDRESS OF PASSWORD
20$:
        SWSTK$  50$             ;; SWITCH TO SYSTEM STATE
        MOV     $DEVHD,R2       ;; START AT BEGINNING OF DEVICE TABLE
30$:
        CMP     D.NAM(R2),#"LB  ;; AND LOOK FOR LB:
        BEQ     40$             ;; IF EQ FOUND
        MOV     D.LNK(R2),R2    ;; NEXT DEVICE
        BNE     30$             ;; TRY IT!
        CLR     4(SP)           ;; INDICATE ERROR BY SETTING USER R1 = 0
        RETURN                  ;; RETURN TO USER STATE
40$:
        MOV     D.UCB(R2),R0    ;; GET UCB ADDRESS
        MOV     U.RED(R0),R0    ;; FIND PHYSICAL LB:(I.E. FIRST REDIRECT)
        MOV     U.DCB(R0),R2    ;; FIND DCB OF PHYSICAL DEVICE
        MOV     D.NAM(R2),4(SP) ;; PUT LB DEVICE INTO 1USER STATE R1
        SUB     D.UCB(R2),R0    ;; CALCULATE UNIT NO.
        MOV     D.UCBL(R2),R1   ;;
        CALL    $DIV            ;;
        ADD     D.UNIT(R2),R0   ;;
        BIC     #177400,R0      ;; CLEAR UNWANTED BITS
        MOV     R0,6(SP)        ;; PUT UNIT NO. INTO USER STATE R2
```

```
113          RETURN                    ;; RETURN TO TASK STATE
114  50$:                              ; REF LABEL
115          CLR     OPNERR            ; SET TO OPEN ERROR
116          TST     R1                ; DID WE FIND PHYSICAL LB:?
117          BNE     60$               ; IF NE YES
118          JMP     ERR1              ; NO --- ERROR
119  60$:
120          CLR     N.FID+F.FNB+$ACTFL ; ASSUME NOT OPEN BY FILE ID
121          ALUN$S  #LUN2,R1,R2       ; ASSIGN LUN TO DEVICE.
122          MOV     $TKPS,MKT+M.KTMG  ; USE TICKS/SEC TO MARK TIME FOR 1SEC.
123  70$:
124          CALL    OPEN              ; OPEN ACCOUNT FILE
125          BCC     100$              ; IF CC - OPEN SUCCESFUL.
126          CMP     OPNERR,#5         ; FIVE FAILURES?
127          BLT     90$               ; NO
128  80$:
129          JMP     ERR2              ; YES
130  90$:
131          DIR$    #MKT              ; NO,WAIT FOR 1 SEC
132          BCS     80$               ; ERROR
133          WTSE$S  #1                ; WAIT FOR TIME
134          INC     OPNERR            ; INCREMENT TIME TRIED
135          BR      70$               ; TRY AGAIN
136  ;
137  ; SEARCH FOR ACCOUNT IN FILE
138  ;
139  100$:
140          CALL    SEARCH            ; SEARCH FOR ACCOUNT NUMBER
141          BCC     110$              ; IF CC - OKAY
142          CALL    CLOSE             ; CLOSE THE ACNT FILE BEFORE GIVING ERROR
143          JMP     ERR3              ; ACCOUNT OR PASSWORD NOT FOUND
144  110$:
145          CALL    CLOSE             ; CLOSE THE ACNT FILE
146          MOV     #0,R0             ; INDICATE SUCCESSS TO CALLER
147  RET:
148          .IF     DF      C$$SPRT
149          MOV     FRMPTR,R5         ; RESTORE FRAME POINTER
150          jmp     c$ret             ; RETURN TO 'C' CALLER
151          .IFF
152          RETURN                    ; RETURN TO 'MAC' CALLER
153          .ENDC
154
155  ERR1:
156  ;       MOV     #ER1,R1           ; ADDRESS OF ERROR MESSAGE
157          MOV     #-1,R0            ; RETURN ERROR CODE
158          BR      ERMSG             ; DISPLAY IT
159  ERR2:
160  ;       MOV     #ER2,R1           ; ADDRESS OF ERROR MESSAGE
161          MOV     #-2,R0            ; RETURN ERROR CODE
162          BR      ERMSG
163  ERR3:
164  ;       MOV     #ER3,R1           ; THIRD ERROR
165          MOV     #-3,R0            ; RETURN ERROR CODE
166  ERMSG:
167  ;       QIOW$S  #IO.WVB,#5,#1,,,,<R1,#80.,#40>
168  ;       CLR     R0                ; SET UNSUCCESSFUL
```

```
169             JMP     RET             ; RETURN TO CALLER
170     ;+
171     ; *** - SEARCH - SEARCH FILE FOR ACCOUNT NUMBER
172     ;
173     ; OUTPUT:
174     ;       R0 - ADDRESS OF ACCOUNT ENTRY
175     ;       CARRY CLEAR - ACCOUNT FOUND
176     ;       CARRY SET - ACCOUNT NOT FOUND
177     ;-
178     SEARCH: MOV     #FDPB,R4                ; GET FILE DPB ADDRESS
179             CLR     OPNERR                  ; ZERO ATTEMPT COUNT (FOR M+ ONLY)
180             MOV     #1,Q.IOPL+10(R4)        ; SET TO START AT VBN 1
181             CLR     Q.IOPL+6(R4)            ;
182     5$:     CALL    QIO                     ; READ NEXT BLOCK
183             MOV     IOSB+2,R2               ; GET COUNT OF WORDS READ
184             BEQ     25$                     ; ZERO, NO WORDS READ
185             MOV     #$ACTBF,R0              ; GET BUFFER ADDRESS
186     10$:    TST     NAMFLG                  ; IS NAME SPECIFIED?
187             BEQ     15$                     ; NO
188             MOV     R0,ENTRY                ; YES, SAVE ENTRY ADDRESS
189             MOV     R1, -(SP)               ; SAVE BYTES LEFT
190             MOV     R2, -(SP)               ;
191             ADD     #A.LNM,R0               ; GET ADDRESS OF LAST NAME
192             MOV     #NAME,R1                ; GET ADDRESS OF NAME ENTERED
193             MOV     #14.,R2                 ; SET LENGTH OF NAME
194     12$:    CMPB    (R0)+,(R1)+             ; NAMES THE SAME?
195             BEQ     14$                     ; YES
196             SEC                             ; NO
197             BR      18$                     ;
198     14$:    DEC     R2                      ; SO FAR
199             BGT     12$                     ; CONTINUE TILL END
200             MOV     ENTRY,R0                ; RESTORE ENTRY ADDRESS
201             BR      17$                     ; NAME IS THE SAME
202     15$:    CMP     UIC,A.GRP(R0)           ; GROUP CODES MATCH
203             BNE     20$                     ; NO
204             CMP     UIC+2,A.GRP+2(R0)       ; MAYBE
205             BNE     20$                     ; NO
206             CMP     UIC+4,A.MBR+1(R0)       ; YES, MEMBER CODES MATCH?
207             BNE     20$                     ; NO
208             MOV     R0,ENTRY                ; SAVE ENTRY POINTER
209             MOV     R1,-(SP)                ; SAVE R1 AND R2
210             MOV     R2,-(SP)                ;
211     17$:    CALL    TPSWD                   ; CHECK PASSWORD
212     18$:    MOV     (SP)+,R2                ; RESTORE R1 AND R2
213             MOV     (SP)+,R1                ;
214             MOV     ENTRY,R0                ; RESTORE ENTRY POINTER
215             BCC     40$                     ; PASSWORD CHECKS OUT
216     20$:    ADD     #A.LEN,R0               ; POINT TO NEXT ENTRY
217             SUB     #A.LEN,R2               ; COMPUTE WORDS LEFT IN BUFFER
218             BHI     10$                     ; LOOP, MORE LEFT
219     25$:    CMPB    #IE.EOF,IOSB            ; END OF FILE?
220             BEQ     30$                     ; YES
221             TSTB    IOSB                    ; ANY ERRORS?
222             BMI     30$                     ; YES
223             ADD     #$BFLEN/512.,Q.IOPL+10(R4); NO, POINT TO NEXT VBN
224             ADC     Q.IOPL+6(R4)            ;
```

```
225             BR      5$                      ; READ IN NEXT BUFFER
226     30$:    SEC                             ; ERROR, ACCOUNT NOT FOUND
227     40$:    RETURN                          ;
228
229
230
231     ;+
232     ; *** - TPSWD - TEST PASSWORD
233     ;
234     ; CARRY SET - INVALID PASSWORD
235     ; CARRY CLEAR - GOOD PASSWORD
236     ;
237     ; NOTE: THIS CODE ALLOWS PSW/TIME. IF THERE IS A/, IT DISREGARDS
238     ; WHAT FOLLOWS BECAUSE, BATCH (ON M+ ONLY) SENDS TIME LIMIT TO BE
239     ; DISREGARDED BY HELLO
240     ;-
241     TPSWD:  MOV     PSWDBF,R1               ; LOCATION OF PASSWORD FIELD
242             MOVB    4(R1),-(SP)             ; PUT PASSWORD ON STACK
243             MOVB    5(R1),1(SP)             ;
244             MOVB    2(R1),-(SP)             ;
245             MOVB    3(R1),1(SP)             ;
246             MOVB    0(R1),-(SP)             ;
247             MOVB    1(R1),1(SP)             ;
248             MOV     SP,R1                   ; POINT TO PASSWORD
249             MOV     R0,-(SP)                ; SAVE R0
250             MOV     #6,R0                   ; LENGTH OF PASSWORD FIELD
251
252     101$:   CMPB    (R1),#40                ; VALID CHAR?
253             BLO     105$                    ; LO-NO.
254             CMPB    (R1),#'/                ; IS IT SLASH (TIME-LIMIT COMING)?
255             BEQ     105$                    ; EQ- YES,,TREAT AS END-OF-PASSWORD
256             CMPB    (R1),#140               ; LOWER CASE?
257             BLOS    102$                    ; NO
258             CMPB    (R1),#172               ; MAYBE
259             BHI     102$                    ; NO
260             BICB    #40,(R1)                ; CONVERT TO UPPER CASE
261     102$:
262             INC     R1                      ; LOOK AT NEXT BYTE
263             DEC     R0                      ; DECRM CHAR COUNT
264             BGT     101$                    ; GT- MORE TO DO.
265             BR      108$                    ; NO NEED TO SPACE FILL.
266     105$:
267             DEC     R0                      ; ANY MORE TO FILL?
268             BMI     108$                    ; MI- NO.
269             MOVB    #40,(R1)+               ; SPACE-IT-OUT!
270             BR      105$                    ; TRY AGAIN.
271     108$:   TST     #ENCRPT                 ; PASSWORD ENCRYPTION SUBR PRESENT?
272             BEQ     109$                    ; EQ- NO.
273             MOV     SP,R0                   ; SHOW WHERE PASSWORD IS
274             ADD     #2,R0                   ;
275             CALL    ENCRPT                  ; ENCRYPT THE PASSWORD
276     109$:   MOV     (SP)+,R0                ; RESTORE R0
277             ADD     #A.PSWD,R0              ; POINT TO PASSWORD IN FILE
278             MOV     SP,R1                   ; POINT TO (FILLED) ENTERED PASSWORD
279             MOV     #6.,R2                  ; SET SIZE OF PASSWORD
280     2$:
```

```
281             CMPB    (R1)+,(R0)+             ; NO, MATCH?
282             BNE     10$                     ; NO, ERROR
283             DEC     R2                      ; ALL DONE?
284             BGT     2$                      ; NO, LOOP
285             BR      20$                     ; YES
286     4$:     CMPB    (R0)+,#'                ; BLANK FROM HERE ON?
287             BNE     10$                     ; NO, ERROR
288             DEC     R2                      ; DONE?
289             BGT     4$                      ; NO, LOOP
290             BR      20$                     ; YES
291     10$:    ADD     #6,SP                   ; CLEAN STACK
292             SEC                             ; SET ERROR
293             RETURN                          ;
294     20$:    ADD     #6,SP                   ; CLEAN STACK
295             RETURN                          ; RETURN (NO ERROR- ADD CLEARS CARRY)
296     ;
297     ; *** - OPEN - OPEN A FILE
298     ;
299     OPEN:
300     ; NOTE - RECORD LOCKING IS OPTIONAL ON M. THIS IS WHY M IS NOT OPENED
301     ; FOR SHARED ACCESS.
302
303             .IF DF  R$$MPL
304
305             OPNS$U  #$ACTFL,,,#FD.RWM       ; OPEN FILE
306
307             .IFF    ;R$$MPL
308
309             OPEN$U  #$ACTFL,,,#FD.RWM       ; OPEN FILE
310
311             .ENDC   ;R$$MPL
312
313             BCS     10$                     ; IF CC  ERROR
314             INC     FILOPN                  ; SET FILE IS OPEN
315     10$:
316             RETURN
317
318     ;
319     ; *** - CLOSE - CLOSE FILE
320     ;
321     CLOSE:
322             TST     FILOPN                  ; IS FILE OPEN?
323             BEQ     10$                     ; NO
324             CLR     FILOPN                  ; FILE IS NOW CLOSING
325             CLOSE$  #$ACTFL                 ; YES - CLOSE FILE
326     10$:
327             RETURN
328     ;
329     ; *** - QIO - ISSUE QIO
330     ;
331     ; INPUT:
332     ;       R4 - DPB ADDRES
333     ;
334     QIO:
335             DIR$    R4                      ; ISSUE QIO
336             BCS     10$                     ; ERROR
```

```
337             MOVB    Q.IOEF(R4),R5           ; GET EVENT FLAG TO WAIT ON
338             WTSE$S  R5                      ; AND WAIT
339     10$:
340             RETURN
341
342             .psect  c$text,i,ro
343             .even
344             .psect  c$data,d,rw
345             .even
346             .END
```

```
 1
 2  /* "@(#)compat.h        1.9 4/15/85" */
 3
 4  /* added by billn */
 5  /* #include <exos/misc.h> */
 6  #ifdef index /* system 3 or 5 */
 7  #include <fcntl.h>
 8  #define dup2(f,n) { close(n); fcntl(f, F_DUPFD, n);}
 9  #endif
10  #ifndef void
11  #define void int
12  #endif
13
14  #define VOID (void)
15
16  #ifndef SIGCHLD
17  #define SIGCHLD SIGCLD
18  #endif
19  /* end billn */
20
21  #ifndef MAXPATHLEN
22  #define MAXPATHLEN 33
23  #endif
24
25  #define receive_data rec_data
26  #define wait3 wait2
27  #define initgroups(a,b)
28  #define inappropriate_request inapreq
29
30  #ifdef BSD4dot2
31  #else
32  #ifdef V7
33  #include <sys/timeb.h>
34  struct timeval { long tv_sec; long tv_usec; };
35  struct timeb ftimeb;
36  #define gettimeofday(a,b) ( ftime (&ftimeb), \
37  (a)->tv_sec = ftimeb.time, (a)->tv_usec = ftimeb.millitm)
38  #else
39  struct timeval { long tv_sec; long tv_usec; };
40  extern long xtime();
41  #define gettimeofday(a,b) ((a)->tv_sec = time(0), (a)->tv_usec = 0)
42  #endif V7
43  #endif BSD4dot2
44
45  #ifndef CTRL
46  #define CTRL(x) 037&'x'
47  #endif
48
49  #define SOL_SOCKET      0
50  #define SO_REUSEADDR    0
```

```
 1
 2  /*
 3   * filename: LIBSOCK.H
 4   *
 5   *           this file contains all the system dependent definitions
 6   *           used in the socket library .
 7   */
 8
 9
10
11  extern char *xstrchr(), *xstrrchr();
12
13  #define HOSTS "LB:[1,1]HOSTS.NET"
14  #define HOSTSLOCAL "LB:[1,1]HOSTLOCAL.NET"
```

```
1
2  /*@(#)varpat.h  1.8 4/11/85*/
3
4  #define connected       conned
5
6  #define connecthelp     connhelp
7  #define mdeletehelp     mdelhelp
8  #define receivehelp     recehelp
9  #define verbosehelp     verbhelp
```

```
 1
 2  /*
 3   * filename: ACCEPT.C
 4   */
 5
 6  #include <xstdio.h>
 7  #include <xerrno.h>
 8  #include "libhdr.c"
 9
10
11  xaccept(s, from)
12      int s;
13      struct sockaddr *from;
14      {
15      register XFILE *file;
16      struct SOictl SOictl;
17      struct iosb iosb;
18      int ret;
19
20
21      if( s < 0 || s >= _XNFILE )
22              return( XEBADF );
23      file = &_xiob[s];
24      if( !(file->_flag & _XUsed))
25              return( XEBADF );
26      SOictl.hassa = from ? 1 : 0;
27      ret = libemt(IO_ACS|SA_ACC, &iosb,0, 0, &SOictl, 0, 0, (int) file->_sys_id);
28
29      libcopy(&SOictl.sa,from,sizeof(struct sockaddr));
30      return(ret);
31      }
32
33  /*
34   * Objective of this function is to process different type of error resulting
35   * from a call to the driver via QIO ( or emt call in 'C' ) call. A QIO
36   * executive directive call reports error in two different ways through the
37   *  DSW ( directive status word ) and also in the IO statusblock.  Again in the
38   * IOSB it is divided into two parts one device specific and the other generic.
39   * The generic and the dsw are returned to the caller after shifting it by -512
40   * and the device specific code is just sign changed. If all is fine then an
41   * non zero value is returned.
42   */
43
```

```
 1  /*
 2   *      FILENAME        ALLOC.C
 3   *
 4   */
 5
 6  #include <rsxos.h>
 7  #include <xstdio.h>
 8  typedef int ALIGN;              /* forces alignment on PDP-11 */
 9
10  union header {  /* free block header    */
11          struct {
12                  union header *ptr;      /* next free block */
13                  unsigned size;          /* size of this free block */
14          } s;
15          ALIGN   x;                      /* force allignment of blocks */
16  };
17
18  typedef union header HEADER;
19
20
21  static HEADER __base = {0};     /* empty list to get started */
22  static HEADER *allocp = XNULL;  /* last allocated block */
23
24  char *xmalloc(nbytes)   /* genral- purpose storage allocator */
25  unsigned nbytes;
26  {
27  static  HEADER *morecore();
28          register HEADER *p, *q;
29          register int nunits;
30
31          nunits = 1+(nbytes+sizeof(HEADER)-1)/sizeof(HEADER);
32          if( (q = allocp) == XNULL) {    /* no free list yet */
33                  __base.s.ptr = allocp = q = &__base;
34                  __base.s.size = 0;
35          }
36          for( p=q->s.ptr; ; q=p, p=p->s.ptr ) {
37             if( p->s.size >= nunits) {   /* big enough   */
38                  if( p->s.size == nunits) /* exactly     */
39                       q->s.ptr = p->s.ptr;
40                  else {          /* allocate tail end    */
41                       p->s.size -= nunits;
42                       p += p->s.size;
43                       p->s.size = nunits;
44                  }
45                  allocp = q;
46                  return ((char *)(p+1));
47             }
48             if( p == allocp )    /* wrapped around free list */
49                  if(( p = morecore(nunits)) == XNULL)
50                          return(XNULL);  /* none left    */
51          }
52  }
53
54
55  #define  NALLOC 16      /* #units to allocate for memory */
56
```

```
 57  HEADER *morecore(nu)   /* ask system for memory        */
 58  unsigned nu;
 59  {
 60          register char *cp;
 61          register HEADER *up;
 62          register int rnu;
 63
 64          rnu = NALLOC * ((nu+NALLOC-1) / NALLOC);
 65          cp = sbreak(rnu * sizeof(HEADER));
 66          if( (int)cp == -1)      /* no space at all      */
 67                  return ( XNULL );
 68          up = (HEADER *)cp;
 69          up->s.size = rnu;
 70          xfree((char *)(up+1));
 71          return(allocp);
 72  }
 73
 74  xfree(ap)       /* put block ap in free list    */
 75  char *ap;
 76  {
 77          register HEADER *p, *q;
 78
 79          p = (HEADER *)ap -1;    /* point to the header  */
 80          for( q=allocp; !(p > q && p < q->s.ptr); q=q->s.ptr )
 81              if( q >= q->s.ptr && (p > q || p < q->s.ptr) )
 82                  break;          /* at one end or other  */
 83
 84          if( p+p->s.size == q->s.ptr) {  /* join to upper nbr */
 85              p->s.size += q->s.ptr->s.size;
 86              p->s.ptr = q->s.ptr->s.ptr;
 87          } else
 88              p->s.ptr = q->s.ptr;
 89          if( q+q->s.size == p ) {        /* join to lower nbr */
 90              q->s.size += p->s.size;
 91              q->s.ptr = p->s.ptr;
 92          } else
 93              q->s.ptr = p;
 94          allocp = q;
 95  }
 96
 97  #define EXTK    01531
 98  #define BLK     64
 99
100  extern int _brk;
101  sbreak(nbytes)
102  register int    nbytes;
103  {
104          register int ret = _brk;
105
106          if(  emt(EXTK, 1+(nbytes-1)/BLK, 0) >= 0 )  {
107                  _brk += nbytes;
108                  return ret;
109                  }
110          else {
111                  /*
112                  xprintf(" Task extention failed %o\n", rval);
```

```
113                     */
114                     return -1;              /* No memory    */
115                     }
116     }
```

```
 1  /*
 2   * filename: BOARD.C
 3   */
 4
 5
 6  #define u_long long
 7  #include <xstdio.h>
 8  #include <xspecial.h>
 9  #include <xerrno.h>
10  #include <libhdr.c>
11  #include <brdioctl.h>
12  #include <init.h>
13  #include <route.h>
14
15
16  int brdopen( brd_no, mode)  /* open an administrative channel */
17    int brd_no;
18    int mode;
19    {
20      int ret;
21      struct iosb iosb;
22
23      if ( mode == 1 )       /* mode is readonly    */
24        mode = 0;
25      else                   /* else mode is read write */
26        mode = 1;
27      ret = libemt(IO_EXC|EX_OPN, &iosb, 0, 0, 0, mode, 0, 0);
28      if ( ret == 0 )
29        ret = iosb.nread;    /* return channel #    */
30      return ( ret );
31    }
32
33  int xbrdclose( fd )          /* close an administrative channel */
34    int fd;
35    {
36     int ret;
37     struct iosb iosb;
38
39       ret = libemt(IO_EXC|EX_CLS,&iosb,0,0,0,0,0,fd);
40       return ( ret );
41    }
42
43
44  int xbrdwrite( sys_id, buf, len)
45    int sys_id;                  /* must have been char *sys_id */
46    char *buf;
47    int len;
48    {
49     int fd, ret;
50     struct iosb iosb;
51     register XFILE *file;
52
53     ret = libemt(IO_WLB,&iosb,buf,len,0,0,0,sys_id);
54     if ( ret == 0 )
55       ret = iosb.nread;
56     return ( ret );
```

```
 57    }
 58
 59  int xbrdread( sys_id, buf, len )       /* read boards memory */
 60    int sys_id;
 61    char *buf;
 62    int len;
 63    {
 64     int fd, ret;
 65     struct iosb iosb;
 66     register XFILE *file;
 67
 68     ret = libemt(IO_RLB,&iosb,buf,len,0,0,0,sys_id);
 69     if ( ret == 0 )
 70       ret = iosb.nread;
 71     return ( ret );
 72    }
 73
 74  int xbrdioctl( sys_id, cmd, arg )
 75    int sys_id, cmd;
 76    char *arg;
 77    {
 78     int i, fd, len = 0, ret;
 79     long along = 0;
 80     Ushort base = 0 , off = 0;
 81     char *buf = 0;
 82     int qio_fn ;
 83     struct iosb iosb;
 84     register XFILE *file;
 85
 86     switch ( cmd ){
 87       case BRDINIT:
 88                 /* translate the mode */
 89         base = *( int *) arg;    /* mode of configuration */
 90         switch ( base ){
 91           case 0: base = 1;      /* host down load        */
 92                   break;
 93           case 1: base = 2;      /* net down load         */
 94                   break;
 95           case 2: base = 0;      /* link level mode       */
 96                   break;
 97           case 0x80:             /* infinite timeout      */
 98                   base |= 1;     /* include with download mode */
 99                   break;
100           default:
101                   base = 1;      /* forced to download mode */
102           }
103         qio_fn = IO_EXC|EX_INI;
104         break;
105
106       case BRDADDR:
107       case BRDSTART:
108         along = *( long * ) arg;
109         base = (Ushort)( ( along >> 16 ) & 0x0000ffff );
110         off  = (Ushort)( along & 0x0000ffff );
111         if ( cmd == BRDADDR )
112           qio_fn = IO_EXC|EX_POS;
```

```
113         else
114           qio_fn = IO_EXC|EX_STR;
115         break;
116
117       case BRDGSTAT:
118       case BRDRSSTAT:
119         buf = ( char *) arg;
120         len = sizeof ( struct EXbdstats );
121         if ( cmd == BRDGSTAT )
122           qio_fn = IO_EXC|EX_STS;
123         else
124           qio_fn = IO_EXC|EX_RST;
125         break;
126
127       case BRDGCONF:
128         buf = (char *) arg;
129         len = sizeof (struct init_msg);
130         qio_fn = IO_EXC|EX_CNF;
131         break;
132
133       case BRDSARP:
134       case BRDGARP:
135       case BRDDARP:
136         buf = (char *) arg;
137         len = sizeof( struct EXarp_ioctl);
138         if ( cmd == BRDSARP )
139           qio_fn = IO_EXC|EX_SAR;
140         else if ( cmd == BRDGARP )
141           qio_fn = IO_EXC|EX_GAR;
142         else
143           qio_fn = IO_EXC|EX_DAR;
144         break;
145
146       case BRDADDRT:
147       case BRDDELRT:
148       case BRDSHOWRT:
149       case BRDDISPRT:
150         buf = (char *) arg;
151         len = sizeof ( struct rtentry );
152         if ( cmd == BRDADDRT )
153           qio_fn = IO_EXC|EX_ART;
154         else if ( cmd == BRDDELRT )
155           qio_fn = IO_EXC|EX_DRT;
156         else if ( cmd == BRDSHOWRT )
157           qio_fn = IO_EXC|EX_SRT;
158         else qio_fn = IO_EXC|EX_NRT;
159         break;
160
161      default:
162         break;
163     }
164     return ( libemt(qio_fn, &iosb, buf, len, 0, base, off, sys_id ));
165
166  }
167
168
```

```
169
170
171  xbrdopen( brdno, mode )
172
173  int brdno;              /* ignore for now */
174  int mode;
175  {
176  int rval;
177  int exosfd;
178  int ioflag;
179  int uflag;
180  register XFILE *file;
181
182  uflag = xtranmode( mode, &ioflag);
183  if ( uflag < 0 )
184    return( uflag );
185  rval = brdopen(1, mode );
186  if( rval < 0 )
187          return( rval );
188  exosfd = xnewod();              /* get a free file descriptor */
189  if( exosfd < 0 ){
190          xbrdclose( rval );
191          return( exosfd );
192          }
193  file = & _xiob[exosfd];
194  file->_flag |= ioflag;
195  file->_sys_id = (char *)rval;
196  file->_read = xbrdread;
197  file->_ioctl = xbrdioctl;
198  file->_write = xbrdwrite;
199  file->_close = xbrdclose;
200  return( exosfd );
201  }
```

```
 1  static char sccsId[] = "@(#)bzero.c     1.4 3/26/85";
 2
 3  /*
 4  code to make 4.2 style code, sort of, happy.
 5  */
 6  bzero( pt, len )
 7  /*
 8  clear a block
 9  */
10
11  char *pt;
12  int len;
13  {
14
15  for( ; len > 0 ; --len )
16          {
17          *pt++ = 0;
18          }
19  }
```

```
 1  /*
 2   * filename: CATCHOOB.C
 3   */
 4
 5  #include <xgenlib.h>
 6  #define MAXCHN  40
 7  #include "libhdr.c"
 8
 9  struct __asts __stast[MAXCHN] = { 0 };
10  extern int _astcatch();   /* this is the ast service routine written in macro */
11
12  int xcatchoob( s, handler)
13    int s;
14    int (*handler)();
15    {
16      register struct iosb *iosb;
17      int ch_no;
18
19      if ( iosb = giosb()){
20        ch_no = (int )_xiob[s]._sys_id;    /* get channel number */
21        if ( __stast[ch_no].stast == FREE){
22          __stast[ch_no].stast = USED;
23          __stast[ch_no].xiobno = s;       /* store xiob number */
24          __stast[s].userast = handler;
25          emt(QIO,IO_ACS|SA_URG,SOLUN,0,iosb,_astcatch,0,0,0,0,0,ch_no);
26          }
27        else return (-1);
28        }
29      else return (NOSOIOSB);
30    }
31
32   libast( iosb )
33    struct iosb *iosb;
34    {
35      Ushort ch_no;
36      Ushort s;
37
38      if( iosb )      /* if a iosb was specified-- which is in this case */
39        {
40          ch_no = iosb->nread;      /* this is set in the ACP */
41          fiosb(iosb);
42          __stast[ch_no].stast = FREE;     /* mark it free for use */
43          s = __stast[ch_no].xiobno;       /* get file no. */
44          if ( __stast[ch_no].userast )
45            (*__stast[ch_no].userast)(s);
46        }
47    }
48
49  struct iosb *
50  giosb()
51  {
52          return(xmalloc( sizeof (struct iosb) ));
53  }
54
55
56  fiosb(iosb)
```

```
57  struct iosb *iosb;
58  {
59          xfree(iosb);
60  }
```

```
 1  #include <xgenlib.h>
 2  #include <fcs.h>
 3
 4  char *inprm[MAXPRM] = {0};      /* array of pointers to input string */
 5
 6  extern char _xctype[];
 7  extern long radix();
 8
 9
10  cmain(pcli)
11  char *pcli;                              /* poiter to command line  */
12  {
13
14          int     count = 0;
15          char    *p = pcli;
16          int     i = 3;
17
18          while( *p ) { *p = _tolower( *p ); ++p; }
19          while( pcli && *pcli ) {
20            switch ( *pcli ) {
21
22                  case '<':
23                          inprm[0] = pcli + 1;
24                          break;
25                  case '>':
26                          inprm[1] = pcli + 1;
27                          break;
28                  case '~':
29                          inprm[2] = pcli + 1;
30                          break;
31                  default :
32                          inprm[i++] = pcli;
33                          count++;
34                  }
35            pcli = firstwhite( pcli, ' ');
36            *pcli++ = 0;          /* make argumet as string  */
37            pcli = skipwhite( pcli, ' ');
38          }
39          return main(count,  &inprm[3]);
40  }
```

```
 1  /*
 2   * filename: CONNECT.C
 3   */
 4
 5  #include <xstdio.h>
 6  #include <xerrno.h>
 7  #include "libhdr.c"
 8
 9
10  xconnect(s, addr)
11  int s;
12  struct sockaddr *addr;
13  {
14      register XFILE *file;
15      struct SOictl SOictl;
16      struct iosb iosb;
17
18      if( s < 0 || s >= _XNFILE )
19              return( XEBADF );
20      file = &_xiob[s];
21      if( !(file->_flag & _XUsed ))
22              return( XEBADF );
23      if ( addr){
24        SOictl . hassa = 1;
25        libcopy(addr,&SOictl.sa,sizeof (struct sockaddr));
26        }
27      else SOictl.hassa = 0;
28      return(libemt(IO_ACS|SA_CON, &iosb,
29                  0, 0, &SOictl, 0, 0, (int) file->_sys_id));
30  }
```

```
 1  #include <rsxos.h>
 2  #include <xstdio.h>
 3  #include <fcs.h>
 4
 5  extern struct _rcb _rcb[];
 6  struct dblbuf hbuf={0}, nbuf= {0};
 7  /*
 8  extern int disk_efn;
 9  */
10  extern char luntbl[];
11  extern struct dblbuf hbuf,nbuf;
12
13  #define CNTRLZ 0366
14
15  dio(sysid, call,ast,wait)
16  register struct _rcb *sysid;
17  int     ( * call)();
18  int     ( * ast)();
19  int     ( * wait)();
20  {
21          static int iosb[2] = {0};
22          int     rval;
23          int     ret;
24
25          if( sysid->flags & DBLBUF) {
26                  /*
27                  disk_efn += d_efn();
28                  */
29                  emt(WTSE,DISKEFN);      /* stop for any pending i/o */
30                                          /* efn is set at ast      */
31                  hbuf.stat[hbuf.active] = 0;
32                  hbuf.active = !hbuf.active;
33                  rval = hbuf.stat[hbuf.active];
34                  if( rval > 0) {
35                  emt(CLEF,DISKEFN);
36                  ret = ( *call)(sysid->fdb,sysid->bptr,0,iosb,ast);
37                  if(ret <= 0) {
38                          hbuf.stat[!hbuf.active] = ret;
39                          emt(SETF,DISKEFN);
40                          }
41                  }
42                  sysid->bptr = hbuf.buffer[hbuf.active];
43                  }
44          else {
45              rval = ( *call)(sysid->fdb,sysid->bptr,DISKEFN,iosb,0);
46                  if(rval > 0) {
47                          (* wait)(sysid->fdb,iosb);
48                          rval = iosb[1];
49                          }
50                  }
51          sysid->bnptr = sysid->bptr;
52          return rval;
53  }
54
55  static char mask[8] = {1, 2, 4, 8, 16, 32, 64, 128};
56  #define BYTE 8
```

```
 57  #define MAXLUN 255
 58  assign(lun)
 59  int lun;
 60  {
 61          *(luntbl + lun/BYTE) |= mask[ lun % BYTE];
 62
 63  }
 64
 65  dassign(lun)
 66  int     lun;
 67  {
 68
 69          *(luntbl + lun/BYTE) &= ~mask[ lun % BYTE];
 70  }
 71
 72  glun()
 73  {
 74          register int    bit = 0;
 75          int     i;
 76
 77          for (i = 1; i <= MAXLUN; ++i) {
 78                  if( !(*(luntbl + i / BYTE) & mask[ i % BYTE] ) ) {
 79                          *( luntbl + i/BYTE ) |= mask[ i%BYTE];
 80                          return i;
 81                          }
 82          }
 83          return -1;
 84
 85  }
 86
 87
 88  nstat(iosb)
 89  register struct iosb *iosb;
 90  {
 91          register int *p;
 92
 93          p = &nbuf.stat[!nbuf.active];
 94  if((iosb->cc >= (unsigned char)0 ) && (iosb->lc == ( unsigned char)0))
 95          *p = iosb->nread;
 96  else if(iosb->cc < ( unsigned char) 0 )
 97          *p = iosb->cc - 512;
 98  else
 99          *p = ( -(iosb->lc & 0xFF));
100  emt(SETF, SOEFN);               /* socket i/o is completed  */
101
102  }
103
104  dstat(iosb)
105  register struct iosb *iosb;
106  {
107          register int *p;
108
109          p = &hbuf.stat[!hbuf.active];
110
111          if( iosb->cc == CNTRLZ )
112                  *p = 0;
```

```
113             else if(iosb->cc > 0)
114                     *p = iosb->nread;
115             else
116                     *p = iosb->cc - 512;
117             emt(SETF, DISKEFN);     /* disk i/o is completed */
118     }
119
120
```

```
 1  /*
 2  RSX version of getclient.
 3  */
 4  #include <xstdio.h>
 5  #include <socket.h>
 6  #include <rsxos.h>
 7  #include <in.h>
 8
 9  getclient( type, pf, sin, options, typical_serv )
10
11          int type;
12          struct sockproto *pf;
13          /*
14          struct sockaddr *sin;
15          */
16          struct sckadr_in *sin;
17          int options;
18          int (*typical_serv)();
19  {
20          int s;
21          int errno;
22          int status;
23          struct sockaddr from;
24
25  start:
26          s = xsocket( type, pf, sin, options );
27          if ( s < 0 )
28                  {
29                  xperror( s, "getclient socket" );
30                  xsleep( 5 );
31                  goto start;
32                  }
33          /*
34          wait for service request
35          */
36          if ( ( errno = xaccept( s, &from ) ) < 0 )
37                  {
38                  xperror( errno, "getclient accept" );
39                  xclose( s );
40                  xsleep( 5 );
41                  goto start;
42                  }
43          /*
44          RSX specific process management
45          */
46          xspawn();
47          (*typical_serv)( s, &from );
48  }
49
```

```
 1
 2  #include <rsxos.h>
 3  #include <xstdio.h>
 4  #include <xctype.h>
 5  #include <xerrno.h>
 6  #include <xspecial.h>
 7  #include <libsock.h>
 8
 9  extern char *xstrchr(), *xstrrchr();
10  extern char *firstwhite();
11  extern char *skipwhite();
12  extern char *lastwhite();
13
14  char *
15  xghname(name, nchars)
16          char *name;
17          int nchars;
18  {
19          int od;
20          XFILE *op;
21          char hbuf[XBUFSIZ], *cp, *ahost;
22          int rc;
23
24          od = xdopen( HOSTS, XFREAD | XFASCII , FILE_NAME);
25          if( ( od < 0) || !( op = xodopen(od, "r")) ){
26                  xperror( XEBADF, "gethname:");
27                  rc = 1;
28                  goto egress;
29          }
30
31          while (XNULL != xogets(hbuf, sizeof (hbuf), op)) {
32                  *xstrchr(hbuf, '\n') = 0;
33                  if (hbuf[0] == '#')
34                          continue;
35                  for (;;) {
36                          cp = lastwhite(hbuf, ' ');
37                          if (cp == XNULL)
38                                  break;
39                          if (!xstrcmp(cp+1, "localhost")) {
40                                  ahost = firstwhite(hbuf, ' ')+1;
41                                  ahost = skipwhite(ahost);
42                                  cp = firstwhite(ahost, ' ');
43                                  if (cp)
44                                          *cp = 0;
45                                  if (xstrlen(ahost)+1 > nchars) {
46                                          rc = 1;
47                                          goto egress;
48                                  }
49                                  xstrcpy(name, ahost);
50                                  rc = 0;
51                                  goto egress;
52                          }
53                          *cp = 0;
54                  }
55          }
56          rc = 1;
```

```
57  egress:
58          xclose(od);
59          return (rc);
60  }
```

```
 1
 2  /*
 3   * filename: HTONS.C
 4   */
 5
 6
 7
 8  unsigned short
 9    xhtons(x)
10      unsigned short x;
11    {
12          return((unsigned short)((x<<8)|((x>>8)&0xff)));
13    }
14
15  long
16    xhtonl(x)
17      long x;
18    {
19      union {
20          long l;
21          struct {
22            unsigned short s_high, s_low;
23          } sl;
24      } h;
25     h.l = x;
26     h.sl.s_high = xhtons(h.sl.s_high);
27     h.sl.s_low  = xhtons(h.sl.s_low);
28     return ( h.l );
29   }
30  unsigned short
31    xntohs(x)
32      unsigned short x;
33    {
34      return ( xhtons(x));
35    }
36
37  long
38    xntohl(x)
39      long x;
40    {
41      return( xhtons(x) );
42    }
```

```
 1
 2  #define PDP11
 3
 4
 5  # include <std.h>
 6  # include <rsx.h>
 7  # include <extypes.h>
 8  # include <exiocmd.h>
 9  # include <soioctl.h>
10  # include <socket.h>
11  # include <exqio.h>
12  # include <solibdef.h>
13
14
15  extern unsigned short   ex_libinit ;
16
17  extern int libinit();
18  extern int libemt();
19  /* extern int check();*/
20  extern int libcopy();
21
22
23  /* below is a definition of a structure for handling user specified
24     AST function calls in the catchoob() library function call       */
25
26  #define USED    1
27  #define FREE    0
28
29  struct __asts{
30    short stast;
31    short xiobno;
32    int (*userast)();
33    }
34  struct seg_addr
35    {
36      Ushort base;                /* segment base address */
37      Ushort off;                 /* segment offset       */
38    };
39
```

```
 1
 2  /*
 3   *  filename: LIBRTS.C
 4   */
 5
 6
 7  # include <std.h>
 8  # include <rsx.h>
 9  # include <extypes.h>
10  # include <solibdef.h>
11
12
13  unsigned short  ex_libinit = 0;
14  unsigned short  unibus = 0;     /* if on a UNIBUS m/c */
15
16  /* below is a definition of a structure for handling user specified
17     AST function calls in the catchoob() library function call       */
18
19  struct __asts{
20    short stast;
21    int (*userast)();
22    }
23  struct seg_addr
24    {
25      Ushort base;                /* segment base address */
26      Ushort off;                 /* segment offset       */
27    };
28
29     int libinit()
30      {
31        ex_libinit = 1;
32      }
33
34     int libcopy(from,to,size)
35      Uchar *from, *to;
36      int size;
37      {
38        while ( size-- )
39          *to++ = *from++;
40      }
41
42  /*
43   * Objective of this function is to process different type of error resulting
44   * from a call to the driver via QIO ( or emt call in 'C' ) call. A QIO
45   * executive directive call reports error in two different ways through the
46   *  DSW ( directive status word ) and also in the IO statusblock.  Again in the
47   * IOSB it is divided into two parts one device specific and the other generic.
48   * The generic and the dsw are returned to the caller after shifting it by -512
49   * and the device specific code is just sign changed. If all is fine then an
50   * non zero value is returned.
51   */
52
53  libemt(cmd,iosb,p1,p2,p3,p4,p5,p6)
54  Ushort cmd;
55  struct iosb *iosb;
56  Ushort p1, p2, p3, p4, p5, p6;
```

```
57  {
58          int j = 0,dsw;
59          register int cnt,i;
60          register int count = 1024;              /* 1 KB */
61
62          if(p2 <= 0){
63             cnt = 1;
64             count = 0;
65            }
66          else
67             cnt = p2;
68
69          for(i = 0; cnt > 0; i++) {
70           if((cnt < count) || (!unibus))
71              count = cnt;
72           dsw =
73           emt(QIOW, cmd, SOLUN, SOEFN, iosb, 0, (p1 + j),count, p3, p4, p5, p6);
74           if((dsw >= 0) && (iosb->cc >= 0) && (iosb->1c == 0)) {
75             if(p2 <= 0)
76                return 0;
77             cnt -= count;
78             j += iosb->nread;
79             continue;                    /* continue on success      */
80            }
81           else
82             if(dsw < 0)
83                  return(dsw - 512);      /* directive error */
84             else
85                  if(iosb->cc < 0)
86                     return(iosb->cc - 512);          /* generic I/O error */
87             else
88                     return( - (iosb->1c & 0xff)); /* device specific error */
89           }
90          iosb->nread = j;        /* total # of bytes transacted */
91          return 0;               /* return success */
92  }
```

```
 1
 2  /*
 3
 4  System entry point for client programs running under RSX.
 5  Note: terminal => unbuffered io.
 6  */
 7  #include <xgenlib.h>
 8  #include <xspecial.h>
 9  #include <xpwd.h>
10  #include <fcs.h>
11
12  #define SY 054523
13
14  extern xttyread();
15  extern xttywrite();
16  extern xttyclose();
17  extern xnofunc();
18  extern xdread();
19  extern xdwrite();
20  extern xdclose();
21
22  struct _xiobuf _xiob[_XNFILE] = {0};
23  struct passwd xpassword = {0};
24  struct passwd *pw = &xpassword;
25  struct ttybuf ttybuf = {0};
26  int _ttyinput = 0;              /* 0 -- interactive . 1 -- non-interactive */
27  struct _rcb     _rcb[_XNFILE] = {0};
28  char   _luntbl[32] = {0};       /* array of 256 bits used to maintain LUN */
29  int _brk = 0;                   /* USED by C_RTS ALLOC & FREE  */
30
31  extern char _xctype[];
32  extern char *inprm[];
33
34
35  main( argc, argv )
36  int argc;
37  char **argv;
38  {
39  int i;
40  register XFILE *file;
41  char    *p;
42  int rval;
43  int ioflag;
44  int mod;
45  int buf[16];
46  int maxlun;
47
48  /* initialize _xiob structure */
49  for(p=(char *)_xiob; p < ( (char *)_xiob + sizeof _xiob); )
50     *p++ = '\0';
51  /* initialize _rcb structre     */
52  for( i=0; i < _XNFILE ; ++i )
53          _rcb[i].flags = RFREE;
54  /*
55   * initailize terminal I/O buffer
56   */
```

```
 57
 58   ttybuf.cur_pos = ttybuf.linetty;
 59   ttybuf.tsize   = 0;
 60
 61  for(i = 1; i < 5; i++)
 62     emt(ALUN, i, SY, 0);
 63  emt(GTSK,buf);
 64  _brk = buf[13];                 /* task size            */
 65  maxlun = buf[8];                /* # of LUN used        */
 66  ppasc(pw->cur_uic, buf[7]);
 67  ppasc(pw->login_uic,buf[15]);
 68  emt(GLUN,1,buf);                          /* get phy. device name */
 69  xstrncpy(pw->log_dev,buf,2);              /* copy device name */
 70  pw->log_dev[2] = (*((char *) buf + 2)) + 060;   /* get unit #  */
 71  pw->log_dev[3] = '\0';                    /* make it string */
 72  xstrcpy(pw->cur_dev,pw->log_dev);
 73
 74  while(maxlun) {
 75          if(emt(GLUN, maxlun,buf) > 0 )
 76                  assign(maxlun);
 77          --maxlun;
 78  }
 79  for( i= 0, file = xstdin ; i < 3 ; ++i, ++file )
 80          {
 81          if(isatty(i)){
 82                  xttyopen(XFREAD|XFWRITE);
 83                  }
 84          else {
 85                  if(i == 0)
 86                          mod = XFASCII | XFREAD;
 87                  else
 88                          mod = XFASCII | XFCREAT | XFWRITE;
 89                  xdopen(inprm[i], mod, FILE_NAME);
 90              }
 91          if( i == 0 )
 92                  xodopen( i, "r" );
 93          else {
 94                  file->_flag |= _XIOLBF;
 95                  file->_cnt = 0;
 96                  xodopen( i , "w" );
 97              }
 98          }
 99  xputchar('\n');
100
101  clientinit();
102
103  xmain(argc, argv);
104  xexit(0);
105  }
106
107  /*
108   * ISATTY:      check object descriptor directs to terminal or not.
109                  if it is terminal returns 1 else 0.
110   *
111   */
112
```

```
113  isatty(od)
114  int od;
115  {
116          if( !inprm[od] )
117                  {
118                  if( od == 0)
119                          _ttyinput = 1;
120                  return(1);
121                  }
122          else
123                  return(0);
124
125  }
126
127
```

```
 1  /*
 2  @(#)xmkarglist.c        1.3 3/29/85
 3
 4  */
 5  #include <rsxos.h>
 6
 7  #define ARGPOINTERSP    200     /* bytes for storing argument pointers */
 8  #define ARGSPACE        400     /* bytes for storing arguments */
 9
10  static char *argbase = {0};
11  static char *stringbase = {0};
12
13
14  char **
15  xmkarglist( line, count )
16
17  char *line;             /* IN */
18  int *count;             /* OUT */
19  {
20  char **argp;
21  char *slurpstring();
22  char *argvsp;
23  int margc;
24
25  margc = 0;
26  /*
27  Allocate space for argv and tokens in line
28  */
29  if( xstrlen( line ) > ARGSPACE )
30          {
31          return( (char **)0 );
32          }
33  argvsp = xmalloc( ARGPOINTERSP + ARGSPACE );
34  if( argvsp == (char *)0 )
35          {
36          return( (char **)0 );
37          }
38  argbase = &argvsp[ARGPOINTERSP];        /* store from first of buffer */
39  stringbase = line;                      /* scan from first of buffer */
40  argp = (char **)argvsp;
41  while (*argp++ = slurpstring())
42          margc++;
43  *count = margc;
44  return( (char **)argvsp );
45  }
46
47  /*
48   * Parse string into argbuf;
49   * implemented with FSM to
50   * handle quoting and strings
51   */
52  char *
53  slurpstring()
54  {
55          int got_one = 0;
56          register char *sb = stringbase;
```

```
57          register char *ap = argbase;
58          char *tmp = argbase;            /* will return this if token found */
59
60          /*
61          Used to return '!' for shell event processing...
62          Ignore significance of '!'.
63          */
64  S0:
65          switch (*sb) {
66
67          case '\0':
68                  goto OUT;
69
70          case ' ':
71          case '\t':
72                  sb++; goto S0;
73
74          default:
75                  goto S1;
76          }
77
78  S1:
79          switch (*sb) {
80
81          case ' ':
82          case '\t':
83          case '\0':
84                  goto OUT;       /* end of token */
85
86          case '\\':
87                  sb++; goto S2;  /* slurp next character */
88
89          case '"':
90                  sb++; goto S3;  /* slurp quoted string */
91
92          default:
93                  *ap++ = *sb++;  /* add character to token */
94                  got_one = 1;
95                  goto S1;
96          }
97
98  S2:
99          switch (*sb) {
100
101         case '\0':
102                 goto OUT;
103
104         default:
105                 *ap++ = *sb++;
106                 got_one = 1;
107                 goto S1;
108         }
109
110 S3:
111         switch (*sb) {
112
```

```
113          case '\0':
114                  goto OUT;
115
116          case '"':
117                  sb++; goto S1;
118
119          default:
120                  *ap++ = *sb++;
121                  got_one = 1;
122                  goto S3;
123          }
124
125  OUT:
126          if (got_one)
127                  *ap++ = '\0';
128          argbase = ap;                   /* update storage pointer */
129          stringbase = sb;                /* update scan pointer */
130          if (got_one)
131                  return(tmp);
132          return((char *)0);
133  }
134
135  xdealglob( pt )
136  /*
137  Free space allocated by either xglob or xmkarglist
138  */
139
140  char **pt;
141  {
142
143  xfree( (char *)pt );
144  }
```

```
 1  # include <xgenlib.h>
 2  /*
 3   * filename: MKCMD.C
 4   *
 5   *          mkcmd creates a MCR command line . It takes a pointer
 6   *          to the commandline and multiple pointers to string .
 7   */
 8
 9  char *
10  mkcmd(line, str)
11    char  *line;
12    char  *str;
13    {
14      char  **argp = &str;
15
16      *line = '\0';       /* clear command line   */
17      while( *argp)       /* till a null argument */
18        xstrcat( line, *argp++ );
19    return(0);
20    }
```

```
 1  /*
 2   * FILENAME:    MKNAME.C
 3   *
 4   *       This routine updates the name according to default dev & dir.
 5   *       it must be invoked after parse. It takes the input from CSI
 6   *       control block, which is created by parse routine.
 7   *
 8   * OUTPUT:
 9   *       If more file_spec  it returns size of current file_spec
10   *       else    0 -- no more in-spec
11   */
12
13   #include <xpwd.h>
14   #include <xgenlib.h>
15
16   extern struct passwd *pw;
17   extern char *csiblk;
18
19  mkname(name)
20  char *name;
21  {
22   int rval;
23   char dev[6] ;
24   char uic[10] ;
25   char nam[15] ;
26   int filelen = 0;
27
28   dev[0] = '\0';
29   uic[0] = '\0';
30   nam[0] = '\0';
31
32   xstrcpy(dev,pw->cur_dev);                /* initialize default dev. */
33   xstrcpy(uic,pw->cur_uic);                /* initialize default dir. */
34   if(csiblk[C_STAT] & CS_DVF){
35         xbcopy(*(int *)(csiblk+C_DEVD+2),dev,*(int *)(csiblk + C_DEVD));
36         dev[*(int *)(csiblk + C_DEVD )  ] = '\0';
37         filelen += *(int *)(csiblk + C_DEVD) + 1;
38   }
39   if(csiblk[C_STAT] & CS_DIF){
40         xbcopy(*(int *)(csiblk+C_DIRD+2),uic,*(int *)(csiblk + C_DIRD));
41         uic[*(int *)(csiblk + C_DIRD )  ] = '\0';
42         filelen += *(int *)(csiblk + C_DIRD );
43   }
44   if(csiblk[C_STAT] & CS_NMF){
45         xbcopy(*(int *)(csiblk+C_FILD+2),nam,*(int *)(csiblk + C_FILD));
46         nam[*(int *)(csiblk + C_FILD )  ] = '\0';
47         filelen += *(int *)(csiblk + C_FILD );
48   }
49   mkcmd(name,dev,":",uic,nam,0);
50   if(csiblk[C_STAT] & CS_MOR)
51         rval = filelen;
52   else
53         rval = 0;
54   return(rval);
55  }
```

```
 1  /*
 2   * filename:    MUXIO.C
 3   */
 4
 5  #include <rsxos.h>
 6  #include <xstdio.h>
 7  #include <xspecial.h>
 8  #include <solibdef.h>
 9
10  #define IO_XFR  003400
11  #define IX_RDS  0000
12  #define TTYEFN  2
13  #define SLEEP_EFN  3
14  #define strip(x)        ((x) & 0177)
15
16  short ready1 = 1;       /* rod1 is readable             */
17  short ready2 = 0;       /* rod2 is initially not readable */
18  struct iosb {
19          char cc;
20          char lc;
21          int  nread;
22          };
23  struct iosb isb1 = {0}; /* IO status block for netread  */
24  struct iosb isb2 = {0}; /* IO status block for netwrite */
25  int _rod2 = 0;
26
27  static char _sibuf[XBUFSIZ] = {0};
28  static char _tibuf[XBUFSIZ] = {0};
29  extern int ttyraw ;     /* 1 == raw  0 == line_edit     */
30  extern char escape;
31  static int scc = 0;             /* byte count for net_read */
32  static int tcc = 0;             /* byte count for tty_read */
33
34  char *_tbufp = _tibuf;
35  extern mynetread();
36  extern myttyread();
37  extern _xsoioctl();
38  extern int xttyread();
39  extern astrd2();
40  extern int wrap;
41  extern int ttylun;
42
43
44  xmux_io( serv_id, io_proc1, rod1, wod1, io_proc2, rod2, wod2 )
45
46  char *serv_id;          /* service identifier, see getclient(3X) */
47  int (*io_proc1)();      /* Network to terminal process    */
48  int rod1;               /* descriptor for first process to read  */
49  int wod1;               /* descriptor for first process to write  */
50  int (*io_proc2)();      /* Terminal to network process    */
51  int rod2;               /* descriptor for second process to read  */
52  int wod2;               /* descriptor for second process to write  */
53  {
54
55          short last_read = 1;    /* last descriptor read   */
56          short netrfin = 1;      /* initialize - net read has finished  */
```

```
57          int pid1;               /* dummy process id - not used   */
58          int pid2;               /* dummy process id - not used   */
59
60          _rod2 = rod2;
61          _xiob[rod1]._read = mynetread;
62          _xiob[rod1]._ioctl = _xsoioctl;
63
64          emt(CLEF,SLEEP_EFN);
65          emt(QIOW,IO_ATA,ttylun,TTYEFN,0,0,astrd2,0,0,0,0,0);
66          for( ;; ) {
67
68                  _xiob[rod2]._read = myttyread;
69
70                  if(ready1)
71                          if(netrfin){
72                             rd1(rod1);           /* do a net read  */
73                             netrfin = 0;         /* netread is pending  */
74                             }
75
76                  if(( ready1 && ready2 && (last_read == 2)) ||
77                          (ready1 && !ready2 ))
78                          {
79                          (*io_proc1)(pid2,rod1,wod1);
80                          last_read = 1;
81                          netrfin = 1;            /* net read has finished */
82                          }
83                  else if(ready2)
84                          {
85                          emt(DSAR);
86                          (*io_proc2)(pid1,rod2,wod2);
87                          last_read = 2;
88                          ready2 = 0;
89                          _tbufp = _tibuf;
90                          tcc = 0;
91                          emt(ENAR);
92                          }
93                  else {
94                          emt(STSE, SLEEP_EFN);
95                          emt(CLEF, SLEEP_EFN);
96                     }
97                  }
98  }
99
100 /*
101  *      RD1
102  */
103
104 rd1(rod1)
105 int rod1;
106 {
107
108         ready1 = 0;     /* make rod1 non-readable */
109         _xsoread(rod1,_sibuf,sizeof _sibuf);
110 }
111
112 /*
```

```
113   *       MYNETREAD
114   */
115
116   mynetread(s,buf,len)
117   int s;
118   char *buf;
119   int len;
120   {
121           if(scc > 0)
122                   xbcopy(_sibuf,buf,scc);
123           return(scc);
124   }
125
126   /*
127    *      MYTTYREAD
128    */
129
130   myttyread(sys_id,buf,len)
131   char *sys_id;
132   char *buf;
133   int len;
134   {
135           char c = _tibuf[0];
136           int cnt;
137
138           /*emt(DSAR);     disable ast recognition        */
139           /* if first char is an escape then  do normal read */
140           if(strip(c) == escape)
141                   _xiob[_rod2]._read = xttyread;
142           if((cnt=tcc) > 0)
143                   xbcopy(_tibuf,buf,tcc);
144
145           /*
146           _tbufp = _tibuf;
147           tcc = 0;
148           emt(ENAR);
149           */
150           return(cnt);
151   }
152
153   extern astrdl();
154
155   _xsoread( s, buf, len)
156   int s;
157   char *buf;
158   int len;
159   {
160           int     i;
161
162           i = (int )_xiob[s]._sys_id;
163           emt(DSCP);                      /* disable checkpointing */
164           emt(QIO,IO_XFR|IX_RDS,SOLUN,0,&isbl,astrdl,buf,len,0,0,0,i);
165   }
166
167   /*
168    *      NRSTATUS -- called from the ast service routine astrdl to set the
```

```
169   *                 return status of the read issued to the network.
170   */
171
172  nrstat(iosb)
173  struct iosb *iosb;
174  {
175
176          if((iosb->cc >= (unsigned char )0) && (iosb->lc == (unsigned char )0))
177                  scc = iosb->nread;
178          else if(iosb->cc < (unsigned char )0)
179                  scc = iosb->cc - 512;           /* generic I/O error */
180               else
181                  scc = (-(iosb->lc & 0xFF));     /* device specefic error*/
182
183          readyl = 1;     /* rodl is now ready to read    */
184          emt(ENCP);      /* enable checkpointing         */
185          emt(SETF, SLEEP_EFN);
186  }
187
188
189  /*
190   *      XKILL -- waits for any outstaning I/O on the network
191   */
192
193  xkill(pid)
194  int pid;
195  {
196          char stadd[2];
197
198          emt(QIOW,IO_DET,ttylun,TTYEFN,0,0,0,0,0,0,0,0);
199          emt(QIOW,IO_KIL,ttylun,TTYEFN,0,0,0,0,0,0,0,0);
200          if(wrap) {      /* previously in wrap mode so set it accordingly */
201              stadd[0] = TC_ACR;
202              stadd[1] = 1;
203              emt(QIOW,SF_SMC,ttylun,TTYEFN,0,0,stadd,2,0,0,0,0);
204             }
205
206          xexit(0);
207  }
208  /*
209   *      _xsoioctl -- kill  any outstanding I/o on the network and
210   *                   then call actual xsoioctl function.
211   */
212
213  _xsoioctl(net,cmd,arg)
214  int net;
215  int cmd;
216  int arg;
217  {
218          emt(ENAR);              /* enable ast recoginition */
219  /*      emt(QIOW,IO_KIL,SOLUN,SOEFN,0,0,0,0,0,0,0,0);   */
220          xsoioctl(net,cmd,arg);
221  }
222  /*
223   *
224   *      TRSTAT  -- get a character from ast stack and put it into the tibuf
```

```
225   */
226
227  trstat(c)
228  char c;
229  {
230
231          *_tbufp++ = c;
232          tcc++;
233          ready2 = 1;
234          emt(SETF, SLEEP_EFN);
235  }
```

```
 1  /*
 2  @(#)passthru.c  1.3 3/29/85
 3
 4  Xpasstnet(3X) and xpassfnet(3X) for Rsx.
 5  */
 6  #include <rsxos.h>
 7  #include <xstdio.h>
 8  #include <xerrno.h>
 9  #include <ftp.h>
10  #include <extypes.h>
11  #include <fcs.h>
12  #define HASHSIZE        1024
13  extern int type;
14  extern int hash;
15  extern struct _rcb _rcb[];
16  extern long _xpass();
17  extern int _dread();
18  extern int _dwrite();
19  extern int _soread();
20  extern int _sowrite();
21
22  #define CNTRLZ 0366
23
24  extern struct dblbuf hbuf ;
25  extern struct dblbuf nbuf ;
26  long
27  xpasstnet( inod, outod )
28
29  register XFILE *inod;   /* input EXOS io object */
30  register XFILE *outod;  /* output EXOS io object */
31  {
32          long    bytes;
33          int     rval;
34          register struct _rcb *insys = ( struct _rcb * )inod->_sys_id;
35          /* make od's as double buffer   */
36
37          hbuf.buffer[1] = inod->_base;
38          inod->_base   = outod->_base;
39          nbuf.buffer[0] = outod->_base;
40          if( (rval = xmalloc(XBUFSIZ)) == XNULL ) {
41                  /*
42                  xoprintf(xstdout,"passtnet buffer pointer = %d\n",rval);
43                  */
44                  return (long) XENOMEM;
45                  }
46          nbuf.buffer[1] = (char *) rval;
47          outod->_write  = _sowrite;
48          nbuf.stat[1] = 1;               /* initialize write status */
49          hbuf.active = 1;
50          nbuf.active = 0;
51          insys->bptr = insys->bnptr = hbuf.buffer[1];
52          insys->bleft = 0;
53          insys->flags |= DBLBUF;
54          hbuf.fd =  inod;
55          nbuf.fd =  outod;
56          bytes = _xpass(  inod, outod  );
```

```
57          inod->_base = hbuf.buffer[!hbuf.active];
58          outod->_base =nbuf.buffer[!nbuf.active];
59          insys->flags &= ~DBLBUF;
60          emt(WTSE,SOEFN);
61          /*
62          xprintf(" No. of socket i/o wait %d\n",socket_efn);
63          xprintf(" No. of disk   i/o wait %d\n",disk_efn);
64          */
65          xfree(nbuf.buffer[nbuf.active]);
66          return bytes;
67  }
68
69
70  long
71  xpassfnet( inod, outod )
72
73  register XFILE *inod;   /* input EXOS io object */
74  register XFILE *outod;  /* output EXOS io object */
75  {
76          long    bytes;
77          int     rval;
78          register struct  _rcb *outsys = ( struct _rcb * ) outod->_sys_id;
79          /* make od's as double buffer   */
80
81          hbuf.buffer[1] = outod->_base;
82          outod->_base   = inod->_base;
83          nbuf.buffer[0] = inod->_base;
84          if( (rval = xmalloc(XBUFSIZ)) == XNULL ) {
85                  /*
86                  xoprintf(xstdout,"passfnet buffer pointer = %d\n",rval);
87                  */
88                  return (long) XENOMEM;          /* No memory */
89                  }
90          nbuf.buffer[1] = (char *) rval;
91          inod->_read    = _soread;
92          nbuf.stat[0] = xsoread(inod->_sys_id,inod->_base, XBUFSIZ);
93          hbuf.stat[1] = 1;               /* initialize write status */
94          inod->_base = nbuf.buffer[1];
95          nbuf.active = 1;
96          hbuf.active = 0;
97          outsys->flags |= DBLBUF;
98          hbuf.fd =  outod;
99          nbuf.fd =  inod;
100         bytes = _xpass(  inod, outod  );
101         outod->_base = hbuf.buffer[!hbuf.active];
102         inod->_base  = nbuf.buffer[!nbuf.active];
103         outsys->flags &= ~DBLBUF;
104         emt(WTSE,DISKEFN);
105         /*
106         xprintf(" No. of socket i/o wait %d\n",socket_efn);
107         xprintf(" No. of disk   i/o wait %d\n",disk_efn);
108         */
109         xfree(nbuf.buffer[nbuf.active]);
110         return bytes;
111 }
112
```

```
113
114 long
115 _xpass(  inod, outod )
116 register XFILE *inod;   /* input EXOS io object */
117 register XFILE *outod;  /* output EXOS io object */
118 {
119 int c;
120 int d = 0;
121 long bytes = (long )0;
122 long hashbytes = XBUFSIZ;
123
124         emt(SETF, SOEFN);               /* No pending i/o on socket  */
125         emt(SETF, DISKEFN);             /* No pending i/o on disk    */
126         while ((c = xread(xfileno(inod), inod->_base, XBUFSIZ)) > 0) {
127                 if ((d = xwrite(xfileno(outod), outod->_base, c)) < 0)
128                         break;
129                 bytes += c;
130                 if (hash) {
131                         xputchar('#');
132                         xfflush(xstdout);
133                 }
134         }
135         if (hash) {
136                 xputchar('\n');
137                 xfflush(xstdout);
138         }
139         if (c < 0) {
140                 xperror( c, "on input");
141                 return( (long)c );
142         }
143         if (d < 0) {
144                 /*
145                  Throw any data remaining in pipe
146                  */
147                  while
148                  ((c=xread(xfileno(inod),inod->_base,XBUFSIZ)) > 0)
149                         ;
150                 xperror( d, "on output");
151                 return( (long)d );
152         }
153         return bytes;
154
155 }
156
157 _dread(sysid,buf,size)
158 register struct _rcb *sysid;
159 char   *buf;
160 int     size;
161 {
162
163         if( size < 0 )
164                 return -1;              /* error */
165         if( sysid->flags & REOF )
166                 return 0;               /* eof  */
167         if( type == TYPE_A )
168                 return ( getnet(sysid,buf,size) );
```

```
169         else
170                 return ( bread(sysid,buf,size) );
171 }
172
173 getnet(sysid,buf,size)
174 register struct _rcb *sysid;
175 char   *buf;
176 register int    size;
177 {
178         register int    count = 0;
179         int     rval;
180         while(size--) {
181           if( !(sysid->flags & REOLN )) {
182                   if( !sysid->bleft && ((rval = getblk(sysid)) < 0) )
183                         return count ? count : rval;
184                   if( sysid->rec.rleft <= 0 && ((rval = endrec(sysid)) < 0) )
185                         return count ? count : rval;
186                   if( sysid->flags & REOF ) {   /* EOF  */
187                         *buf++ = '\r';
188                         *buf = '\n';
189                         return count+2;
190                         }
191                 }
192           if(sysid->flags & REOLN) {
193                 if(sysid->flags & RCRFLAG) {
194                         *buf++ = '\r';
195                         sysid->flags &= ~RCRFLAG;
196                         }
197                 else {
198                         *buf++ = '\n';
199                         sysid->flags &= ~REOLN;
200                         }
201           }
202           else if( sysid->rec.rleft ) {
203                   *buf++ = *sysid->bnptr++;
204                   --sysid->bleft;
205                   --sysid->rec.rleft;
206                 }
207           else {        /* case of zero records */
208                  ++size;
209                  continue;
210                 }
211           ++count;
212         }       /* end of while */
213
214         return count;
215 }
216
217 bread(sysid)
218 register struct _rcb *sysid;
219 {
220
221         int *p;
222         int *q;
223         int rval;
224
```

```
225         rval = getblk(sysid);
226         p = hbuf.buffer[hbuf.active];
227         q = nbuf.buffer[nbuf.active];
228         hbuf.buffer[hbuf.active] = q;
229         sysid->bptr = sysid->bnptr = q;
230         nbuf.buffer[nbuf.active] = p;
231         hbuf.fd->_base = nbuf.fd->_base = p;
232         return rval;
233 }
234
235 _dwrite(sysid,buf,size)
236 register struct _rcb *sysid;
237 char   *buf;
238 int     size;
239 {
240         if( size < 0)
241                 return -1;              /* error */
242         if( type == TYPE_A )
243                 return ( _put(sysid,buf,size) );
244         else
245                 return ( bwrite(sysid,buf,size) );
246 }
247
248
249 bwrite(sysid,buf,size)
250 register struct _rcb *sysid;
251 char *buf;
252 int  size;
253 {
254
255         int     *p;
256         int     *q;
257
258         p = hbuf.buffer[hbuf.active];
259         q = nbuf.buffer[nbuf.active];
260         hbuf.buffer[hbuf.active] = q;
261         sysid->bptr = sysid->bnptr = q;
262         sysid->bleft = BLKSIZE - size;
263         nbuf.buffer[nbuf.active] = p;
264         nbuf.fd->_base = hbuf.fd->_base = p;
265
266         if ( size < BLKSIZE )
267                 return 1;
268         return putblk(sysid);
269 }
270 _soread(s,buf,len)
271 int     s;
272 char   *buf;
273 int     len;
274 {
275         return ( sio(s,buf,len,IO_XFR|IX_RDS) );
276 }
277
278 _sowrite(s,buf,len)
279 int     s;
280 char   *buf;
```

```
281  int     len;
282  {
283          return ( sio(s,buf,len,IO_XFR|IX_WRS) );
284  }
285
286  extern int astsio();
287
288  sio(s,buf,len,iocode)
289  int     s;
290  char   *buf;
291  int     len;
292  int     iocode;
293  {
294          static struct iosb ios = {0};
295          int     rval;
296          char   *pbuf;
297          int     ret;
298
299
300
301          /*
302          socket_efn += s_efn();
303          */
304          emt(WTSE, SOEFN);       /* stop for i/o completion  */
305          rval = nbuf.stat[!nbuf.active]; /* # of bytes read */
306          if( (rval > 0) && (rval < len) && ( type != TYPE_A) &&
307              (iocode == ( IO_XFR | IX_RDS ))
308                  )
309            {     /* Previous buffer is not yet completly read.
310                     Since in the binary mode, buffer's are flipped
311                     instead of data transfer. We need to read buffer
312                     fully ( disk block = 512 bytes ).
313                  */
314
315                  pbuf = nbuf.buffer[!nbuf.active];
316
317                  while (rval < len) {
318                    if(libemt(iocode,&ios,pbuf+rval,len-rval,0,0,0,s))
319                          break;          /* I/O error    */
320                    if(!(ret = ios.nread))
321                          break;  /* EOF                  */
322                    rval += ret;
323                  }       /* repeat loop, buffer is not yet read fully */
324                  nbuf.stat[!nbuf.active] = rval; /* # of bytes read */
325             }
326          /*
327           * flip the buffer
328           */
329          nbuf.stat[nbuf.active] = 0;
330          nbuf.active = !nbuf.active;
331          nbuf.fd->_base = nbuf.buffer[nbuf.active];
332          hbuf.fd->_base = nbuf.fd->_base;
333          if( rval > 0) {
334                  emt(CLEF,SOEFN);
335                  emt(QIO,iocode,SOLUN,0,&ios,astsio,buf,len,0,0,0,s);
336                  }
```

```
337             return rval;
338
339     }
340
```

```
 1  #include <rsxos.h>
 2  #include <xstdio.h>
 3  #include <xspecial.h>
 4  #include <xerrno.h>
 5  #include <libsock.h>
 6
 7
 8  extern char *xstrchr();
 9  extern char *firstwhite();
10  extern char *lastwhite();
11  extern char *skipwhite();
12
13
14  char *
15  xraddr(desaddr)
16          long desaddr;
17  {
18          int od;
19          XFILE *op ;
20          char hbuf[XBUFSIZ], *cp, *host;
21          int first = 1;
22
23          od = xdopen( HOSTS, XFREAD|XFASCII, FILE_NAME );
24          if ( ( od < 0 ) || !( op = xodopen ( od, "r"))){
25              xperror(XEBADF, "raddr: ");
26              xexit(1);
27              }
28  top:
29          while (xogets(hbuf, sizeof (hbuf), op) && xstrchr(hbuf, '\n')) {
30                  long addr,rnumber();
31
32                  *xstrchr(hbuf, '\n') = 0;
33                  if (hbuf[0] == '#')
34                          continue;
35                  if ((addr = rnumber(hbuf)) == -1)
36                          continue;
37                  if (addr != desaddr)
38                          continue;
39                  host = firstwhite(hbuf, ' ') + 1;
40                  host = skipwhite(host);
41                  cp = firstwhite(host, ' ');
42                  if (cp)
43                          *cp = 0;
44                  cp = xmalloc(xstrlen(host)+1);
45                  xstrcpy(cp, host);
46                  xclose( od );
47                  return (cp);
48          }
49          if (first == 1) {
50                  first = 0;
51                  xclose(od);
52                  if (((od =xdopen(HOSTSLOCAL,XFREAD|XFASCII,FILE_NAME))>= 0)
53                       && ( op = xodopen( od, "r")))
54                          goto top;
55                  else{
56                    xperror( XEBADF ,"raddr:");
```

```
57                      xexit( 1 );
58                      }
59                    return (0);
60          }
61  bad:
62          xclose(od);
63          return (0);
64  }
```

```
 1
 2  /*
 3   * filename: RECEIVE.C
 4   */
 5
 6  #include <xstdio.h>
 7  #include <xerrno.h>
 8  #include "libhdr.c"
 9
10
11  static
12  int receive (s,from,buf,len)
13   int s;
14   struct sockaddr *from;
15   char   *buf;
16   int    len;
17  {
18    int ret,i;
19    struct SOictl SOictl;
20    struct iosb iosb;
21
22    if (from) {
23      SOictl . hassa = 1;
24      libcopy(from,&SOictl.sa,sizeof(struct sockaddr));
25      }
26    else SOictl . hassa = 0;
27    ret = libemt(IO_XFR|IX_RCV, &iosb, buf, len, &SOictl, 0, 0, s);
28
29    if ( ret == 0)
30       ret = iosb . nread;
31    return ( ret );
32   }
33
34
35  xreceive(s, from, msg, len)
36
37  int s;
38  struct sockaddr *from;
39  char *msg;
40  int len;
41  {
42      register XFILE *file;
43
44      if( s < 0 || s >= _XNFILE )
45              return( XEBADF );
46      file = &_xiob[s];
47      if( !( file->_flag & _XUsed ))
48              return( XEBADF );
49      return( receive( (int)file->_sys_id, from, msg, len ));
50  }
```

```
#include <rsxos.h>
#include <xstdio.h>
#include <xctype.h>
#include <xerrno.h>
#include <xspecial.h>
#include <libsock.h>


extern char *xstrchr(), *xstrrchr();
extern char *firstwhite();
extern char *lastwhite();
extern char *skipwhite();
static char host_name[40] = {0};

long    rnumber();

long
xrhost(ahost)
        char **ahost;
{
        int od;
        XFILE *op;
        char hbuf[XBUFSIZ], *cp;
        int first = 1;
        long addr;

        if (isdigit(**ahost) && (addr = rnumber(*ahost)) != -1) {
                xoprintf(xstderr, "addr=%x\n", addr);
                return (addr);
        }
        od = xdopen( HOSTS, XFREAD|XFASCII, FILE_NAME);
        if ( ( od < 0 ) || !( op = xodopen( od, "r"))){
             xperror( XEBADF ,"rhost:");
             xexit( 1 );
             }
top:
        while (xogets(hbuf, sizeof (hbuf), op)) {
                *xstrchr(hbuf, '\n') = 0;
                if (hbuf[0] == '#')
                        continue;
                for (;;) {
                        cp = lastwhite(hbuf, ' ');
                        if (cp == XNULL)
                                break;
                        if (!xstricmp(cp+1, *ahost)) {
                                if ((addr = rnumber(hbuf)) == -1)
                                        goto bad;
                                xclose(od);
                                *ahost = firstwhite(hbuf, ' ') + 1;
                                *ahost = skipwhite( *ahost );
                                cp = firstwhite(*ahost, ' ');
                                if (cp)
                                        *cp = 0;
                                xstrcpy(host_name, *ahost);
                                *ahost = host_name;
                                return (addr);
```

```
57                          }
58                          *cp = 0;
59                  }
60          }
61          if (first == 1) {
62                  first = 0;
63                  xclose(od);
64                  od = xdopen(HOSTSLOCAL, XFREAD|XFASCII, FILE_NAME);
65                  if ( ( od >= 0 ) && ( op = xodopen ( od, "r")) )
66                    goto top;
67                  else{
68                     xperror( XEBADF ,"rhost:");
69                     xexit( 1 );
70                     }
71                  return (-1);
72          }
73  bad:
74          xclose(od);
75          return (-1);
76  }
77
78  long
79  rnumber(cp)
80          register char *cp;
81  {
82          register long val;
83          register int base;
84          register char c;
85          long parts = 0;
86          char *pplow = (char *)&parts;
87          char *pplim = pplow+4;
88          char *pp = pplow;
89          long net, imp, hoi;
90
91          if (xstrchr(cp, '/') == 0)
92                  goto again;
93          hoi = xatoi(cp);
94          if (xstrchr(cp, ',')) {
95                  imp = xatoi(xstrchr(cp, '/') + 1);
96                  net = xatoi(xstrchr(cp, ',') + 1);
97                  hoi = xntohs((short)hoi);
98                  val = (net<<24)|(hoi<<8)|imp;
99          } else {
100                 net = xatoi(xstrchr(cp, '/') + 1);
101                 val = (net<<24)|hoi;
102         }
103         /*
104         val = xhtonl(val);
105         */
106         return (val);
107
108 again:
109         val = 0; base = 10;
110         if (*cp == '0')
111                 base = 8, cp++;
112         if (*cp == 'x' || *cp == 'X')
```

```
113                 base = 16, cp++;
114         while (c = *cp) {
115                 if (isdigit(c)) {
116                         val = (val * base) + (c - '0');
117                         cp++;
118                         continue;
119                 }
120                 if (base == 16 && isxdigit(c)) {
121                         val = (val << 4) + (c + 10 - (islower(c) ? 'a' : 'A'));
122                         cp++;
123                         continue;
124                 }
125                 break;
126         }
127         if (*cp == '.') {
128                 /*
129                  * Internet format:
130                  *      net.host.lh.imp
131                  */
132                 if (pp >= pplim)
133                         return (-1);
134                 *pp++ = val, cp++;
135                 goto again;
136         }
137         if (*cp) {
138                 /*
139                 if (*cp == 'n') return (xhtonl(val));
140                 */
141                 if (*cp == 'n') return (val);
142                 if ( (*cp != ' ') && (*cp != '\t')) return (-1);
143         }
144         if (pp >= pplim)
145                 return (-1);
146         *pp++ = val;
147         /*
148         return xhtonl(parts);
149         */
150         return (parts);
151 }
152
153 char *
154 skipwhite( cpt )
155
156 char *cpt;
157 {
158
159         while( cpt && ( *cpt == ' ' || *cpt == '\t' ))
160                 ++cpt;
161         return ( cpt );
162 }
163
164
165 char *
166 firstwhite( cpt, ch )
167 /*
168 find first white space
```

```
169  */
170
171  char *cpt;
172  char ch;
173  {
174
175          while( cpt && *cpt && *cpt != ' ' && *cpt != '\t' )
176                  ++cpt;
177          if ( cpt && *cpt ) {
178                  return( cpt );
179          }
180          return ( XNULL );
181  }
182
183  char *
184  lastwhite( cpt, ch )
185  /*
186  find last white space
187  */
188
189  char *cpt;
190  char ch;
191  {
192  char *ocpt = XNULL;
193
194          while( ( cpt = firstwhite( cpt, ' ' ) ) != XNULL ) {
195                  ocpt = cpt;
196                  cpt++;
197          }
198          return( ocpt );
199  }
```

```
 1
 2  /*
 3   * filename: SEND.C
 4   */
 5
 6  #include <xstdio.h>
 7  #include <xerrno.h>
 8  #include "libhdr.c"
 9
10  int send (s,to,msg,len)
11      int s;
12      struct  sockaddr *to;
13      char    *msg;
14      int     len;
15  {
16    int ret,i;
17    struct iosb iosb;
18    struct SOictl SOictl;
19
20      if ( to ) {
21        SOictl . hassa = 1;
22        libcopy(to,&SOictl.sa,sizeof (struct sockaddr));
23        }
24      else SOictl.hassa = 0;
25      ret = libemt(IO_XFR|IX_SND, &iosb, msg, len, &SOictl, 0, 0, s);
26
27      if ( ret == 0 )
28        ret = iosb . nread;
29      return (ret);
30  }
31
32
33  xsend(s, to, msg, len)
34
35  int s;
36  struct sockaddr *to;
37  char *msg;
38  int len;
39  {
40      register XFILE *file;
41
42      if( s < 0 || s >= _XNFILE )
43              return( XEBADF );
44      file = &_xiob[s];
45      if( !(file->_flag & _XUsed ))
46              return( XEBADF );
47      return( send( (int)file->_sys_id, to, msg, len ));
48  }
49
```

```
 1  /*
 2   * filename: SOCKADDR.C
 3   */
 4
 5  #include <xstdio.h>
 6  #include <xerrno.h>
 7  #include "libhdr.c"
 8
 9
10
11
12  int xsktaddr(s,addr)
13      int s;
14      struct      sockaddr  *addr;
15
16  {
17      register XFILE *file;
18        int ret;
19        struct SOictl SOictl;
20        struct iosb  iosb;
21
22      if( s < 0 || s >= _XNFILE )
23              return( XEBADF );
24      file = &_xiob[s];
25      if( !(file->_flag & _XUsed ))
26              return( XEBADF );
27          if ( addr ){
28            SOictl . hassa = 1;
29            libcopy(addr,&SOictl.sa,sizeof(struct sockaddr));
30            }
31          else SOictl . hassa = 0;
32          ret = libemt(IO_ACS|SA_SAD,&iosb,0,0,&SOictl,0,0, (int) file->_sys_id);
33
34          if ( ret >= 0)
35             libcopy(&SOictl.sa, addr, sizeof ( struct sockaddr ));
36          return ( ret );
37      }
```

```
 1  /*
 2   * filename: SOCKET.C
 3   */
 4
 5  #include <xstdio.h>
 6  #include <libhdr.c>
 7
 8
 9  extern xsoread();
10  extern xsowrite();
11  extern xsoioctl();
12
13  static
14   int socket( type, pf, addr, options)
15     int type;
16     struct sockproto *pf;
17     struct sockaddr *addr;
18     int options;
19     {
20       int ret;
21       struct iosb iosb;
22       struct SOictl  SOictl;
23
24       if ( addr ){
25         SOictl. hassa = 1;
26         libcopy( addr, &SOictl.sa, sizeof ( struct sockaddr ));
27         }
28       else SOictl. hassa = 0;
29       if ( pf ){
30         SOictl. hassp = 1;
31         libcopy( pf, &SOictl.sp, sizeof ( struct sockproto ));
32         }
33       else SOictl.hassp = 0;
34       SOictl . type = type;
35       SOictl . options = options;
36       ret= libemt(IO_ACS|SA_OPN,&iosb,0,0,&SOictl,0,0,0);
37       if ( ret == 0 )
38         ret = iosb.nread;
39       return ( ret );
40     }
41
42
43
44   int xsoclose( s )
45     int s;
46     {
47      int ret;
48      struct iosb iosb;
49
50        ret = libemt(IO_ACS|SA_CLS, &iosb, 0, 0, 0, 0, 0, s);
51
52        if (( iosb.cc ) && ( iosb.lc == 0 ) )
53          ret = 1;
54        else ret = -1;
55        return ( ret );
56     }
```

```
57
58
59
60
61  xsocket( type, pf, addr, options )
62      int type;
63      struct sockproto *pf;
64      struct sockaddr *addr;
65      int options;
66      {
67          int rval;
68          int exosfd;
69          register XFILE *file;
70
71          rval = socket( type, pf, addr, options );
72          if( rval < 0 )
73                  return( rval );
74          exosfd = xnewod();              /* get a free file descriptor */
75          if( exosfd < 0 )
76            return( exosfd );
77          file = & _xiob[exosfd];
78          file->_flag |= _XIORW | _XPrimary ;
79          file->_sys_id = (char *)rval;
80          file->_read = xsoread;
81          file->_write = xsowrite;
82          file->_ioctl = xsoioctl;
83          file->_close = xsoclose;
84          return( exosfd );
85      }
```

```
 1
 2  /*
 3   * filename: SOCONTROL.C
 4   */
 5
 6  #include "libhdr.c"
 7
 8  int xsoioctl( dev, cmd, addr)
 9    int dev, cmd;
10    char *addr;
11    {
12      int ret;
13      struct iosb iosb;
14      struct SOictl SOictl;
15      Uchar CMD = (Uchar) cmd;
16
17      libcopy( addr, (char *)&SOictl.hassa, sizeof ( struct SOictl));
18      ret = libemt(IO_SOC|CMD, &iosb, 0, 0, &SOictl, 0, 0, dev);
19      switch ( cmd ){
20        case SIOCRCVOOB :
21              *addr = *(char *)&SOictl.hassa;
22              break;
23        case SIOCGLINGER :
24        case SIOCGKEEP   :
25        case SIOCATMARK  :
26        case SIOCGPGRP   :
27              *(short *)addr = SOictl.hassa;
28              break;
29        default :
30              break;
31      }
32        return ( ret );
33    }
34
```

```
 1
 2  /*
 3   * filename: xsoread.c
 4   */
 5
 6  #include <xstdio.h>
 7  #include <xerrno.h>
 8  #include "libhdr.c"
 9
10
11   int xsoread( s, buf, len)
12     int s;
13     char *buf;
14     int len;
15     {
16       int ret, i;
17       struct iosb iosb;
18
19       ret = libemt(IO_XFR|IX_RDS, &iosb, buf, len, 0, 0, 0, s);
20       if (ret==0)
21         ret = iosb . nread;
22       return(ret);
23     }
```

```
 1
 2  /*
 3   * filename: XSOWRITE.C
 4   */
 5
 6  #include <xstdio.h>
 7  #include <xerrno.h>
 8  #include "libhdr.c"
 9
10
11   int xsowrite ( s, msg, len )
12      int s;
13      char *msg;
14      int len;
15      {
16       int ret, i;
17       struct iosb iosb;
18
19       ret = libemt(IO_XFR|IX_WRS, &iosb, msg, len, 0, 0, 0, s);
20       if ( ret==0 )
21         ret = iosb . nread;
22       return(ret);
23      }
```

```
 1  /*
 2  @(#)xaccess.c   1.4 3/29/85
 3
 4  RSX version of routine to check access rights.
 5  */
 6
 7  #include <xspecial.h>
 8  #include <xerrno.h>
 9
10
11
12  xaccess( name, special, mode )
13
14  char *name;
15  int special;
16  int mode;
17  {
18  register int rval;
19  char buf[ MXNAMELEN + 1 ];
20
21  /*
22  modify name (if necessary) for special meanings
23  */
24  rval = xmodname( &name, special, buf, sizeof( buf ) );
25  if ( rval < 0 )
26          return( rval );
27  if( mode ==  0x0 )
28          {
29          /* check file exist or not      */
30          rval = xdopen(name,XFREAD,FILE_NAME);
31          if(rval >= 0){  /* check for success    */
32                  xclose(rval);
33                  rval = 0;
34                  }
35          else {
36                  /* fail to open , ie file does not exist */
37                  rval = rval -512;
38                  }
39          }
40   else {
41          /* rval = access(name,mode);
42          ...
43            to be implement
44          ...
45            Not used by FTP
46          */
47          rval = XSYSERR;
48          }
49  return( rval );
50  }
```

```
 1  /*
 2   *
 3    FILE_NAME:    XCHDIR.C
 4   */
 5
 6  #include  <xgenlib.h>
 7  #include <xspecial.h>
 8  #include  <xpwd.h>
 9  #include <xerrno.h>
10
11  #define  EXEFN  010001
12
13  extern long radix();
14  extern struct passwd *pw;
15  extern char *csiblk;
16  xchdir(name, special)
17  char  *name;            /* name of uic to be modified   */
18  int   special;          /* flag for special files       */
19  {
20
21   char *uic;
22   int rval = 0;
23
24    switch ( special ) {
25
26      case FILE_NAME:
27            uic = name;
28            break;
29      case CURRENT_DIR:
30            return(0);
31
32      case HOME_DIR:
33            xstrcpy(pw->cur_dev,pw->log_dev);
34            xstrcpy(pw->cur_uic,pw->login_uic);
35            return(0);
36
37      default:
38
39            return(XEINVAL);
40
41      }
42    rval = parse(uic,xstrlen(uic));
43    if(rval<0)
44          return(XENOTDIR);
45    if(csiblk[C_STAT] & CS_NMF)
46          return(XENOTDIR);
47    if(csiblk[C_STAT] & CS_DVF){
48          xbcopy(*(int *)(csiblk+C_DEVD+2),pw->cur_dev,*(int *)(csiblk+C_DEVD));
49          pw->cur_dev[*(int *)(csiblk + C_DEVD)] = '\0';
50    }
51    if(csiblk[C_STAT] & CS_DIF){
52          uic = (char *)( *(int *)(csiblk + C_DIRD + 2 ));
53          rval = val_uic(uic);
54          if(rval >=  0)
55                  xstrcpy(pw->cur_uic, uic);
56    }
```

```
    return(rval);

}


val_uic(uic)
char *uic;
{
 int rval;
        if(*uic++ != '[')
                return(XENOTDIR);
        rval = group(&uic);
        if(rval < 0)
                return(rval);
        if(*uic++ != ',')
                return(XENOTDIR);
        rval = group(&uic);
        if(rval < 0)
                return(rval);
        if(*uic != ']')
                return(XENOTDIR);
        return(0);
}
group(s)
char **s;
{
 int i;
 char *p = *s;

        for(i=0;i<3;i++,p++)
                if( isdigit(*p))
                        if(*p > 067)            /* octal digit */
                                {
                                *s = p;
                                return(XENOTDIR);
                                }
                        else
                                continue;
                else
                        break;
        *s = p;
        return(0);
}
```

```
 1  /*
 2  @(#)xchown.c    1.4 3/29/85
 3
 4  Xchown for RSX.
 5  */
 6  #include <xspecial.h>
 7  #include <rsxos.h>
 8  #include <xpwd.h>
 9
10  extern struct passwd *pw;
11
12  xchown( name, special )
13  char *name;
14  int special;
15  {
16  char buf[MXNAMELEN +1];
17  int rval;
18  struct pr {
19          char sy[5];
20          char ow[5];
21          char gr[5];
22          char wo[5];
23          } pr;
24  /*
25  char cmdlin[CMDSIZE];
26
27  rval = xmodname( &name, special, buf, sizeof(buf) );
28  if( rval < 0 )
29          return( rval );
30  mkcmd(cmdlin,"PIP ",name,"/NM/PR",
31                      "/SY:",getown(pr.sy,pw->lgn_prv&017),
32                      "/OW:",getown(pr.ow,pw->lgn_prv&0360),
33                      "/GR:",getown(pr.gr,pw->lgn_prv&07400),
34                      "/WO:",getown(pr.wo,pw->lgn_prv&0170000),
35                      "/FO",0);
36  mkcmd(cmdlin,"PIP ",name,"/PR/NM/FO",0);
37  return(cmdcall(cmdlin));
38  */
39  return(0);
40  }
41
42  /*
43  char *
44  getown(s,v)
45  char *s;
46  int v;
47  {
48          if(v & 01)
49                  *s++ = 'R';
50          if(v & 02)
51                  *s++ = 'W';
52          if(v & 04)
53                  *s++ = 'E';
54          if(v & 010)
55                  *s++ = 'D';
56          *s = '\0';
```

```
57
58          return(s);
59  }
60  */
```

```
 1  /*
 2  %W% %G%
 3
 4  Function to use for all RSX low level close routines.
 5  */
 6
 7  #include <rsxos.h>
 8  #include <xstdio.h>
 9  #include <xspecial.h>
10  #include <fcs.h>
11
12  extern struct _rcb _rcb[];
13  xdclose( sysid )
14  register struct _rcb *sysid;
15  {
16
17          register        int     bytes;
18          if( sysid->flags & RFREE )
19                  return 1;
20          sysid->flags &= ~DBLBUF;
21          if( sysid->mode & XFCREAT || sysid->mode & XFAPPEND ) {
22                  bytes = sysid->bleft;
23                  switch (bytes) {
24                  case 512:
25                          bytes = 0;
26                          break;
27                  case   0:
28                          putblk(sysid);
29                          break;
30                  default:
31                          putblk(sysid);
32                          bytes = 512 - bytes;
33                          bytes & 01 ? ++bytes : bytes;
34                          break;
35                  }
36                  _wmrec(sysid->fdb, sysid->rec.rsize,bytes);
37                                  /* Write max rec & first free byte */
38                  xfree( sysid->rptr );           /* free record  */
39                  }
40          __close(sysid->fdb);                    /* call CLOSE$  */
41          xfree( sysid->bptr );                   /* free block buffer */
42          xfree(sysid->fdb);                      /* free FDB  */
43          dassign( sysid->rlun );                 /* mark LUN as free */
44          sysid->flags = RFREE;                   /* mark _RCB as free */
45
46  }
```

```
 1  /*
 2  @(#)xdir.c      1.4 3/29/85
 3
 4  Xdir(3X) for RSX - make a directory, remove a directory, move a file.
 5  */
 6  #include <xspecial.h>
 7  #include <xgenlib.h>
 8  #include <xpwd.h>
 9  extern mkcmd();
10  extern long radix();
11  extern char *csiblk;
12  extern struct passwd *pw;
13  #define EXEFN   010000
14
15  xmkdir (what, special )
16  char *what;
17  int special;
18  {
19          return(XEOPNOTSUPP);
20  /*
21  char buf[ MXNAMELEN + 1 ];
22  char cmdlin[CMDSIZE];
23  int rval;
24
25  rval = xmodname( &what, special, buf, sizeof( buf ) );
26  if( rval < 0 )
27          return(XENOTDIR );
28  if(!(csiblk[C_STAT] & CS_DVF))
29          return(rval - 512);
30  if(!(csiblk[C_STAT] & CS_DIF))
31          return(XENOTDIR);
32  if(csiblk[C_STAT] & CS_NMF)
33          return(XENOTDIR);
34  if((rval = val_uic( *(int *)(csiblk + C_DIRD +2))) < 0)
35          return(rval);
36  mkcmd(cmdlin,"UFD ", what, 0 );
37  return(cmdcall(cmdlin));
38  */
39  }
40
41  xrmdir (what, special )
42  char *what;
43  int special;
44  {
45  char buf[ MXNAMELEN + 1 ];
46  int rval;
47  char cmdlin[CMDSIZE];
48  char uic[UICSIZE];
49  char *puic= uic;
50  char dev[6];
51  char *cuic;
52
53  rval = xmodname( &what, special, buf, sizeof( buf ) );
54  if( rval < 0 )
55          return(XENOENT);
56  if((csiblk[C_STAT] & CS_NMF) || (!(csiblk[C_STAT] & CS_DIF)) )
```

```
57          return(XENOTDIR);
58  if(csiblk[C_STAT] & CS_DVF){
59          xbcopy(*(int *)(csiblk + C_DEVD +2 ),dev, *(int *)(csiblk + C_DEVD));
60          dev[*(int *)(csiblk + C_DEVD)] = '\0';
61          }
62  else {
63          xstrcpy(dev,pw->cur_dev);
64  }
65  cuic = (char *)( *(int *)(csiblk + C_DIRD + 2));
66   while ( *cuic >= 0)
67          {
68         if( isdigit(*cuic) || ( *cuic == 0) )
69             *puic++ = *cuic;
70          cuic++;
71          }
72   mkcmd(cmdlin,"PIP ",dev,":[0,0]",uic,".DIR;*/DE/NM",0);
73   return(cmdcall(cmdlin));
74  }
75
76  xrename (from, from_special, to, to_special )
77  char *from, *to;
78  int from_special, to_special;
79  {
80  char buf[ MXNAMELEN + 1 ];
81  char buf2[ MXNAMELEN + 1 ];
82  char cmdlin[CMDSIZE];
83  int rval;
84
85  rval = xmodname( &from, from_special, buf, sizeof( buf ) );
86  if( rval < 0 )
87          return(XENOENT);
88  rval = xmodname( &to, to_special, buf2, sizeof( buf2 ) );
89  if( rval < 0 )
90          return(XENOENT);
91  mkcmd(cmdlin,"PIP ", to, "=", from, "/RE/NM", 0);
92  return(cmdcall(cmdlin));
93  }
```

```
 1  /*
 2  %W% %G%
 3  xdopen(3x) for Rsx.
 4  */
 5
 6  #include <rsxos.h>
 7  #include <xstdio.h>
 8  #include <xspecial.h>
 9  #include <xerrno.h>
10  #include <fcs.h>
11
12  extern xdread();
13  extern xdwrite();
14  extern xdclose();
15  extern char *csiblk;
16  extern struct _rcb _rcb[];
17
18  xdopen( name, mod, special )
19
20  char *name;
21  register int mod;
22  int special;
23  {
24  int rval;
25  int exosfd;
26  int rmode;
27  int ioflag;
28  int rsize;      /* file type . if ascii then _rcb[sysid].rec.rsize = 0
29                                    else non-zero.
30                  */
31  register struct _rcb *sysid;
32  register XFILE *file;
33  char buf[MXNAMELEN + 1];
34
35  rval = xmodname( &name, special, buf, sizeof( buf ) );
36  if( rval < 0 )
37          return( rval );
38  /*
39  Translate mode to Rsx mode and type.
40  */
41  sysid = newrcb(mod);
42  if((int )sysid < 0)
43          return (int) sysid;
44  rmode = xtranmode( mod, &ioflag);
45  if ( rmode < 0 )
46          return( rmode );
47
48  exosfd = xnewod();              /* get a free file descriptor */
49  if( exosfd < 0 )
50          return( exosfd );
51  if( sysid->mode & XFCREAT)
52          create( sysid->fdb, sysid->rec.rsize);
53  rval = parse(name, xstrlen(name));
54  if(rval < 0)
55          return rval;
56  rval = __open(sysid->fdb,rmode,sysid->rlun,csiblk+C_DSDS);
```

```
 57  if(rval < 0)
 58          return(rval);
 59  if( mod & XFAPPEND )
 60          getlblk(sysid);                 /* get last block */
 61
 62  file = &_xiob[exosfd];
 63  file->_file = exosfd;
 64  file->_flag |= ioflag;
 65  file->_sys_id = (char *)sysid;
 66  file->_read = xdread;
 67  file->_write = xdwrite;
 68  file->_close = xdclose;
 69  return( exosfd );
 70  }
 71
 72  struct _rcb *
 73  newrcb(mod)
 74  register int mod;
 75  {
 76          register struct _rcb *sysid = _rcb;
 77          char    *p;
 78          int     i;
 79
 80          for(i=0; i < _XNFILE; ++i, ++sysid ) {
 81                  if(sysid->flags & RFREE )
 82                          break;
 83          }
 84          if ( i >= _XNFILE)
 85                  return (struct _rcb *) XEMFILE; /* Too many files open */
 86
 87          sysid->mode = mod;
 88          sysid->flags = RUSED;
 89          sysid->bptr = xmalloc(BLKSIZE);
 90          sysid->fdb = xmalloc(FDBSIZE);
 91          for( p = sysid->fdb; p < sysid->fdb + FDBSIZE ; ++p )
 92                  *p = 0;
 93          if( !sysid->bptr  || !sysid->fdb  )
 94                  return (struct _rcb *) XENOMEM;         /* No memory */
 95          sysid->bnptr = sysid->bptr;
 96          if( mod & XFCREAT || mod & XFAPPEND ) {
 97                  sysid->bleft = BLKSIZE;
 98                  sysid->rptr  = xmalloc(RECSIZE);
 99                  if ( mod & XFASCII )
100                          sysid->rec.rsize = 0;
101                  else
102                          sysid->rec.rsize = 512;
103                  }
104          else  {
105                  sysid->bleft = 0;
106                  sysid->rptr =  0;
107                  sysid->rec.rleft = -1;
108          }
109          sysid->rnptr = sysid->rptr;
110          sysid->rlun = glun();
111          return sysid;
112
```

```
113  }
114
115
116  getlblk(sysid)
117  register struct _rcb *sysid;
118  {
119
120          register char *fdb = sysid->fdb;
121          int      ffby = *((int *) ( fdb + F_FFBY ));
122
123          sysid->rec.rsize = *((int *) ( fdb + F_RSIZ ));
124          if( ffby ) {
125                  getblk(sysid);
126                  sysid->bnptr += ffby;
127                  sysid->bleft = BLKSIZE - ffby;
128                  --*((long *)( fdb + F_BKVB ));
129                  }
130  }
```

```
 1  /*
 2  %W% %G%
 3
 4  Function to use for all RSX low level read routines.
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xspecial.h>
 9  #include <extypes.h>
10  #include <fcs.h>
11
12  extern struct _rcb _rcb[];
13
14
15  #define endblk(i) ((!i->bleft) ? getblk(i) : 1 )
16
17  xdread( sysid, buf, size )
18  register struct _rcb *sysid;
19  char *buf;
20  int size;
21  {
22
23          if( size < 0)
24                  return -1;              /* error        */
25          if( sysid->flags & REOF )
26                  return 0;               /* eof  */
27          if( sysid->mode & XFASCII )
28                  return ( _get(sysid,buf,size));
29          else
30                  return ( read(sysid,buf,size));
31
32  }
33
34  read(sysid,buf,size)
35  register struct _rcb *sysid;
36  char   *buf;
37  register int     size;
38  {
39
40          register int    count = 0;
41          int     rval;
42
43          while(size--) {
44            if(( rval = endblk(sysid)) <= 0 )
45                  return count ? count : rval;
46            *buf++ = *sysid->bnptr++;
47            --sysid->bleft;
48            ++count;
49          }
50
51          return count;
52  }
53
54
55  _get(sysid,buf,size)
56  register struct _rcb *sysid;
```

```
 57  char   *buf;
 58  register int    size;
 59  {
 60          register int    count = 0;
 61          int     rval;
 62
 63          while(size--) {
 64            if( !sysid->bleft && ((rval = getblk(sysid)) < 0) )
 65                  return count ? count : rval;
 66            if( sysid->rec.rleft <= 0 && ((rval = endrec(sysid)) < 0) )
 67                  return count ? count : rval;
 68            if( sysid->flags & REOF ) {           /* EOF */
 69                  *buf = '\n';
 70                  return ++count;
 71                  }
 72            if(sysid->flags & REOLN) {
 73                  *buf++ = '\n';
 74                  sysid->flags &= ~REOLN; /* reset */
 75                  }
 76             else if( sysid->rec.rleft) {
 77                    *buf++ = *sysid->bnptr++;
 78                    --sysid->bleft;
 79                    --sysid->rec.rleft;
 80                  }
 81             else {       /* case of zero records        */
 82                    ++size;
 83                    continue;
 84                  }
 85            ++count;
 86          }
 87          return count;
 88  }
 89
 90
 91  endrec(sysid)
 92  register struct _rcb *sysid;
 93  {
 94
 95          register int    rval;
 96
 97                  if( sysid->rec.rleft == 0 )
 98                          sysid->flags |= (REOLN | RCRFLAG );
 99                  if( (Ushort)sysid->bnptr & 01 ) {
100                          ++sysid->bnptr;
101                          --sysid->bleft;
102                          }
103                  if((rval = endblk(sysid)) <= 0)
104                          return rval;
105                  sysid->rec.rleft = *(int *)sysid->bnptr;
106                  sysid->bnptr += 2;              /* adjust the pointer */
107                  sysid->bleft -= 2;
108                  return endblk(sysid) ;
109  }
110
111
112  extern int __read();
```

```
113  extern int rastdio();
114  extern int __rwait();
115
116  getblk(sysid)
117  register struct _rcb *sysid;
118  {
119          register int    ret;
120
121          ret = dio(sysid,__read,rastdio,__rwait);
122          if(ret == 0)
123                  sysid->flags |= REOF;
124          sysid->bleft = ( ret > 0 ) ? ret : 0;
125          return ret;
126
127  }
```

```
 1  /*
 2
 3  Function to use for all low level write routines.
 4
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xspecial.h>
 9  #include <extypes.h>
10  #include <fcs.h>
11
12
13  extern struct _rcb _rcb[];
14
15  xdwrite(sysid, buf, size)
16  register struct _rcb *sysid;
17  char * buf;
18  int     size;
19  {
20
21          if( size < 0 )
22                  return -1;              /* error        */
23          if( sysid->mode & XFASCII )
24                  return _put(sysid, buf, size);
25          else
26                  return write(sysid, buf, size);
27
28  }
29
30  write(sysid, buf, size)
31  register struct _rcb *sysid;
32  char * buf;
33  register int    size;
34  {
35          register int    count = 0;
36          int     rval;
37
38          while ( size--) {
39            if( !sysid->bleft) {
40                  if((rval = putblk(sysid)) <= 0)
41                          return count ? count : rval;
42                  }
43            *sysid->bnptr++ = *buf++;
44            --sysid->bleft;
45            ++count;
46          }
47          return count;
48  }
49
50
51  _put(sysid, buf, cnt)
52  register struct _rcb *sysid;
53  char   *buf;
54  int     cnt;
55  {
56          char    *nbuf;
```

cnt = min (sysid->bleft, size) ;

(kind of) inefficient

if (sysid->bleft <= size) {

bcopy (buf , sysid->bnptr , sysid->bleft);

if (rval = putblk (sysid) <= 0)

return (sysid->bleft ;

buf += sysid->bleft ;

size -= sysid->bleft ;

```
57          register char *wbuf;
58          int     ncnt, wcnt, rval, count = 0, tot = 0;
59
60          while(cnt) {
61                  wbuf = buf;
62                  wcnt = cnt;
63                  xlocc('\n', wcnt, wbuf, &ncnt, &nbuf);
64                  if(ncnt) {
65                          /* found EOL, backtrack to find a '\r', if any */
66
67                          if((*buf != '\n') && (nbuf[-1] == '\r'))
68                                  count = cnt - ncnt - 1;
69                          else
70                                  count = cnt - ncnt;
71
72                          if((rval = putrec(sysid, buf, count)) <= 0)
73                                  return count ? count : rval;
74
75                          cnt = ncnt - 1;
76                          buf = nbuf + 1;
77                          tot += cnt - ncnt + 1;
78                       } /* end of if(ncnt)...   */
79                  else {  /* if ncnt == 0, i.e. no '\n' found in buffer */
80                          xbcopy(buf, sysid->rptr, cnt);
81                          sysid->rnptr += cnt;
82                          if((sysid->rnptr - sysid->rptr) > BLKSIZE)
83                                  sysid->rnptr = sysid->rptr + BLKSIZE;
84                          sysid->flags |= KEPT_ASIDE;
85                          tot += cnt;     /* the kept aside bytes */
86                          cnt = 0;
87                      }
88                  }       /* end of while(cnt)... */
89          return tot;
90  }
91
92
93  putrec(sysid, buf, size)
94  register struct _rcb *sysid;
95  register char *buf;
96  int size;
97  {
98          register int kept_aside;
99          int     rval;
100
101         if((Ushort )sysid->bnptr & 01) {        /* if on a byte boundary */
102                 *sysid->bnptr++ = 0;
103                 --sysid->bleft;
104               }
105         if(!sysid->bleft && ((rval = putblk(sysid)) <= 0))
106                 return rval;
107         if(sysid->flags & KEPT_ASIDE)
108                 kept_aside = sysid->rnptr - sysid->rptr;
109         else
110                 kept_aside = 0;
111
112         *(int * )sysid->bnptr = size + kept_aside;
```

```
113          sysid->bnptr += 2;
114          sysid->bleft -= 2;
115
116          if(sysid->flags & KEPT_ASIDE) {
117                  sysid->flags &= ~KEPT_ASIDE;
118                  if((rval = _rec(sysid, sysid->rptr, kept_aside)) <= 0)
119                          return rval;
120                  sysid->rnptr = sysid->rptr;
121                }
122          if((rval = _rec(sysid, buf, size)) <= 0)
123                  return rval;
124
125          return 1;
126
127  }
128
129
130  _rec(sysid, buf, cnt)
131  register struct _rcb *sysid;
132  char *buf;
133  register int cnt;
134  {
135          register int leftcnt = 0;
136          int rval;
137
138          if(sysid->bleft < cnt) {
139                  leftcnt = sysid->bleft;
140                  xbcopy(buf, sysid->bnptr, leftcnt);
141                  if((rval = putblk(sysid)) <= 0)
142                          return rval;
143                }
144          xbcopy(buf + leftcnt, sysid->bnptr, cnt - leftcnt);
145          sysid->bnptr += cnt - leftcnt;
146          sysid->bleft -= cnt - leftcnt;
147
148          return 1;
149  }
150
151
152  extern int wastdio();
153  extern int __write();
154  extern int __wwait();
155
156
157  putblk(sysid)
158  register struct _rcb *sysid;
159  {
160
161          sysid->bleft = BLKSIZE;
162          return dio(sysid, __write, wastdio, __wwait);
163
164  }
165
166  xlocc(c, cnt1, buf1, acnt2, abuf2)
167  char    c;
168  register int    cnt1;
```

```
register char *buf1;
int     *acnt2;
char    **abuf2;
{
        register int i;
        int     found = 0;

        for(i = 0; i < cnt1; i++)
                if(*buf1++ == c) {
                        found++;
                        break;
                      }
        if(found) {
                *acnt2 = cnt1 - i;
                *abuf2 = --buf1;
              }
        else
                *acnt2 = 0;     /* char 'c' not found   */
}
```

```
 1  /*
 2  %W% %G%
 3
 4  Unix specific close all EXOS file objects and exit program.
 5  */
 6
 7  #include <xstdio.h>
 8
 9  xexit( status )
10
11  int status;
12  {
13  int i;
14
15  for( i = 0; i < _XNFILE ; ++i )
16          {
17          xclose( i );
18          }
19  exit( status );
20  }
```

```
 1
 2  /*
 3  %W% %G%
 4  xftpopen(3x) for Rsx.
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xspecial.h>
 9  #include <xerrno.h>
10  #include <fcs.h>
11  #include <ftp.h>;
12
13  extern xdread();
14  extern xdwrite();
15  extern xdclose();
16  extern _dread();
17  extern _dwrite();
18
19  extern int type;
20  extern struct dblbuf hbuf;
21
22
23  xftpopen( name, mode, special, ftp_attributes )
24
25  char *name;
26  int mode;
27  int special;
28  register struct ftp_attr *ftp_attributes;
29  {
30
31          int     rval;
32          register XFILE *file;
33
34  /*
35  Check that ftp_attributes are supportted.
36  */
37  if( ftp_attributes )
38          {
39          if( (ftp_attributes->rep_type != RT_ASCII &&
40                  ftp_attributes->rep_type != RT_IMAGE ) ||
41                  ftp_attributes->format != TF_NONPRINT ||
42                  ftp_attributes->structure != IS_FILE ||
43                  ftp_attributes->trans_mode != TM_STREAM
44          )
45                  return( XEOPNOTSUPP );
46          }
47  if( type == TYPE_A)
48          mode |= XFASCII;
49  rval =  xdopen(name, mode, special);
50  if(rval >= 0 ) {
51          file = &_xiob[rval];
52          file->_read = _dread;
53          file->_write= _dwrite;
54          if( mode & XFREAD ){
55                  hbuf.stat[0] = getblk(file->_sys_id);
56                  }
```

```
57          hbuf.buffer[0] = ((struct _rcb *) file->_sys_id)->bptr;
58          }
59  return rval;
60  }
```

```
 1  /*
 2  static char sccsId[] = "@(#)xgethbad.c  1.4 3/26/85";
 3
 4  code to make 4.2 style code, sort of, happy.
 5  */
 6
 7  #include <arp.h>
 8
 9  extern long xrhost();
10
11
12
13  struct hostent *
14  ghbaddr( addr, size, family )
15  /*
16  gethostbyaddr for C compilers with 8 character identifiers
17  WARNING ----
18   a second call to this routine will destroy the previous result.
19  */
20
21  struct in_addr *addr;
22  int size, family;
23  {
24  static struct hostent hent;
25
26  hent.h_name = (char *)xraddr( addr->s_addr );
27  return( ( struct hostent *)&hent );
28  }
29
```

```
 1  /*
 2  static char sccsId[] = "@(#)xgethbnam.c 1.4 3/26/85";
 3
 4  code to make 4.2 style code, sort of, happy.
 5  */
 6
 7  #include <arp.h>
 8
 9  extern long xrhost();
10
11  struct hostent *
12  ghbname( host )
13  /*
14  gethostbyname for C compilers with 8 character identifiers
15  WARNING ----
16   a second call to this routine will destroy the previous result.
17  */
18
19  char *host;
20  {
21  static struct hostent hent;
22  static struct sckadr_in sock;
23
24  sock.sin_addr.S_un.S_addr = (long)xrhost( &host );
25  if (sock.sin_addr.S_un.S_addr == -1 )
26          return( (struct hostent *)XNULL);
27  hent.h_addr = (char *)&sock.sin_addr;
28  hent.h_name = host;
29  return( &hent );
30  }
```

```
 1  /*
 2   * FILE_NAME:       XGLOB.C
 3   *
 4   *      Xglob for RSX. Expand wild word in input line.
 5   *
 6   */
 7
 8   #include <xgenlib.h>
 9   #include <xspecial.h>
10
11   #define MAXINSPEC      20
12
13   extern xgfatal();
14
15   static char **gargv = 0;
16   static short gargc = 0;
17   char *globerr = (char *)0;
18   short gflag = 0;
19   char *in_stat = (char *) 0;
20   char *f_stat = (char *) 0;
21   char *in_ver_stat = (char * )0;
22   char *f_ver_stat = (char * )0;
23   int globbing = 0;
24
25   /*
26    * Main root of xglob.
27    *
28    */
29
30  char **
31  xglob( v )
32   register char  **v;
33  {
34   char **agargv;
35
36   /* initialize return parameter */
37   gargv = xmalloc(2);
38   *gargv = (char *)0;
39   globerr = (char *)0;
40   globbing = 1;
41   in_stat = xmalloc(MAXINSPEC);
42   in_ver_stat = xmalloc(MAXINSPEC);
43   gargc = 0;
44    while (*v) {
45      if(wildchar(*v)){
46         agargv  = glob(*v++);
47      }
48      else {
49         agargv = xmalloc(xstrlen(*v)+1 +4);
50         if(agargv == (char *)0)
51             xgfatal("Out of Memory");
52         *agargv = agargv + 2;
53         agargv[1] = (char *)0;
54         xstrcpy(agargv + 2, *v++);
55      }
56      gargv = copyblk(gargv,agargv);
```

```
 57    }
 58   xfree(in_stat);
 59   xfree(in_ver_stat);
 60   globbing = 0;
 61   return(gargv);
 62
 63  }
 64
 65
 66  char **
 67  glob(name)
 68  char *name;
 69  {
 70   char buf[400];
 71   char *line = 0;
 72   char template[16] ;
 73   char list_name[27];
 74   char *bufp = buf;
 75   int len;
 76   int rval;
 77   int sys_id;
 78   XFILE *file;
 79   int argc;
 80   char **argv = (char **)0;
 81
 82   xstrcpy(template,SCRATCHFILE);
 83  /* xmktemp(template);   */
 84   rval = _ls(template, name, LS_ARG);
 85   if(rval < 0){
 86      globerr = " glob failed";
 87      return(0);
 88   }
 89   sys_id = opentemp(template);
 90   if(sys_id >= 0){
 91      file = xodopen(sys_id,"r");
 92     }
 93   if(sys_id < 0 || file < ( XFILE * ) 0 ) {
 94      globerr = " Can't open file for globbing";
 95      return(0);
 96   }
 97   /*
 98    * initialize buff, which is used to filled with list of names
 99    */
100
101  f_stat = in_stat - 1;                    /* used in nam_list */
102  f_ver_stat = in_ver_stat - 1;            /* used in nam_list */
103
104  for(bufp=(char *)buf; bufp<(char *)buf + sizeof buf;)
105          *bufp++ = '\0';
106   bufp = buf;
107   while(rval = xogets(list_name,sizeof(list_name), file) > 0) {
108          if(!nam_list(list_name))
109                  continue;       /* discarding temp name created by glob */
110          remtrail(list_name);
111          len = xstrlen(list_name) + 1;
112          if((buf + sizeof(buf) - bufp) > len)
```

```
113                 {
114                 xstrcpy(bufp,list_name);
115                 bufp += len;
116                 *(bufp -1) = ' ';               /* name separator  */
117                 }
118         else
119                 {
120                 argv = copyblk(argv, xmkarglist(buf, &argc));
121                  for(bufp=(char *)buf; bufp<(char *)buf + sizeof buf;)
122                         *bufp++ = '\0';
123                 bufp = buf;
124                 }
125         }
126  if( bufp > buf)
127         argv= copyblk(argv, xmkarglist(buf, &argc));
128  xclose(xfileno(file));
129  rval = xunlink(template,FILE_NAME);
130  if(rval < 0) {
131    globerr = " system error -- can't delete file ";
132    return(0);
133         }
134
135
136  return(argv);
137
138 }
139
140 char **
141 copyblk(v1, v2)
142 char **v1;
143 char **v2;
144 {
145  register char **nv ;
146  int i;
147     i = (blklen(v1) + 1) * sizeof(char **) + blkslen(v1)
148       + (blklen(v2) + 1) * sizeof(char **) + blkslen(v2)
149       ;
150  nv = xmalloc(i);
151  if(nv == (char *)0)
152    xgfatal("Out of Memory");
153  return(blkcpy(nv, v1, v2));
154
155 }
156
157 char **
158 blkcpy(v, v1, v2)
159  char **v, **v1, **v2;
160 {
161  register char **av = v;
162  char **ov1 = v1;
163  char **ov2 = v2;
164  char *stringp;
165
166 if(v1){
167  while(*v1++)
168       ++av;
```

```
169  }
170  if(v2){
171   while(*v2++)
172        ++av;
173  }
174   stringp = (char *)++av;
175   av = v;
176   v1 = ov1;
177   v2 = ov2;
178
179  if(v1){
180   while(*v1){
181     *av++ = stringp;
182     xstrcpy(stringp, *v1);
183     stringp += xstrlen(*v1++) + 1;
184   }
185  }
186
187  if(v2){
188   while(*v2){
189     *av++ = stringp;
190     xstrcpy(stringp, *v2);
191     stringp += xstrlen(*v2++) + 1;
192   }
193  }
194
195   *av = (char *)0;
196   if(ov1)
197   xfree(ov1);
198   if(ov2)
199   xfree(ov2);
200   return(v);
201  }
202
203  wildchar(p)
204  char *p;
205  {
206
207   while(*p){
208     if(( *p == '*')|| (*p == '%'))
209        return(1);
210     p++;
211    }
212   return(0);
213  }
214
215
216  xgfatal(string)
217  char *string;
218  {
219   xoprintf(xstderr, "xglob:%s\n", string);
220   xexit(1);
221  }
222
223
224  blklen(av)
```

```
225    register char **av;
226  {
227   register int i = 0;
228
229   if(av != XNULL)
230      while(*av++)
231         i++;
232   return(i);
233
234  }
235
236  static
237  blkslen(argp)
238    register char **argp;
239  {
240   int total = 0;
241
242   if(argp != XNULL)
243   while(*argp)
244       {
245        total += xstrlen(*argp++) + 1;
246       }
247   return(total);
248  }
249
250
251  remtrail(s)
252  char *s;
253  {
254  char *start;
255          start = s;
256          s = s + xstrlen(s) - 1;
257          while ((s >= start) && ((*s == '\r') || (*s == '\n')))
258                  *s-- = '\0';
259  }
```

```
 1  /*
 2
 3  RSX implementation of xinit_env(3X).
 4  */
 5  #include <xpwd.h>
 6  #include <xgenlib.h>
 7  #include <xctype.h>
 8
 9  #define upper(c)        (isupper(c)) ? c : _toupper(c)
10  #define EFN     1
11  #define SRDA    01153
12  #define TASK_EFN        2
13
14  extern long radix();
15  extern ast_recv();
16  char msge[] = "                                ";
17  extern struct passwd *pw;
18  static int buf[16] = {0};
19
20  xinit_env( name, password, account )
21
22  char *name;     /* loggin name */
23  char *password; /* password */
24  char *account;  /* login uic  */
25  {
26  int rval;
27  int dirdes[2];
28  long task;
29
30  if ( !name )
31          return( 0 );
32  emt(SRDA,ast_recv);
33  emt(CLEF,TASK_EFN);
34  emt(GTSK,buf);  /* get task info for type of system this demon is running on */
35
36  /* validata user's login information */
37  rval = login(name,password,account);
38  if( rval < 0 )
39          return(rval - 512);
40  dirdes[0] = xstrlen(pw->login_uic);
41  dirdes[1] = (int ) pw->login_uic;
42  ascpp(dirdes,msge);
43  emt(CLEF,TASK_EFN);
44
45  if(buf[14] == 6)        /* is it an RSX-11M-PLUS system */
46      task = radix("DEMTO ");
47  else                    /* it is an RSX-11M system      */
48      task = radix("...DEM");
49
50  emt(SDAT,task,msge,0);                          /*
51                                                   * send login uic to
52                                                   * ...dem , which updates
53                                                   * this task uic as user's
54                                                   * login uic
55                                                   */
56  emt(WTSE,TASK_EFN);
```

```
 57  emt(SRDA,0);
 58
 59  return( 1 );
 60  }
 61
 62  login(name,password,account)
 63  char *name;
 64  char *password;
 65  char *account;
 66  {
 67   int i= -1, cc = 0;
 68   long tsk;
 69   int asefn = 010001;            /* EFN = 1      */
 70   int rval;
 71   int esb[8];
 72
 73   if(buf[14] == 6)       /* if an RSX-11M-PLUS system */
 74      tsk = radix("LGNTO ");
 75   else
 76      tsk = radix("...LGN");
 77
 78   while( name[++i] && cc < 14 ){
 79          if(name[i] == '/')
 80                 break;
 81      msge[cc++] = upper(name[i]);
 82   }
 83   while(cc < 14)
 84       msge[cc++] = ' ';          /* padded the name whith blanks  */
 85   msge[cc++] = '*';              /* separator between name & account */
 86   if( name[i] == '/' && (name[i+1]) ){
 87          while(name[++i] && cc < 26)
 88                 msge[cc++] = upper(name[i]);
 89          }
 90   else {
 91          i = -1;
 92          while( password[++i] && cc < 26 )
 93                 msge[cc++] = upper(password[i]);
 94        }
 95   while(cc < 26)
 96       msge[cc++] = ' ';
 97  emt(SDAT,tsk,msge,0);                   /* send pkt to ...lgn task     */
 98  emt(USTP,tsk);                          /* unstop ...lgn      */
 99  emt(WTSE,TASK_EFN);                     /* wait for receive pkt from ...lgn */
100  rval = *( (int *)(msge + 4));           /* return status      */
101  if(rval == 0) {
102          xstrcpy(pw->cur_uic, msge+6);
103          xstrcpy(pw->login_uic, msge+6);
104          xstrcpy(pw->log_dev, msge+16);
105          xstrcpy(pw->cur_dev, msge+16);
106        }
107
108   return(rval);
109
110  }
```

```
 1  #include <xgenlib.h>
 2  #include <xspecial.h>
 3  #include <xpwd.h>
 4  extern long radix();
 5  extern xread();
 6  extern xdread();
 7  extern char *csiblk;
 8  extern struct passwd *pw;
 9  extern char *f_stat;
10  extern char *in_stat;
11  extern char *f_ver_stat;
12  extern char *in_ver_stat;
13  extern int globbing;
14
15  #define  EXEFN  010000
16
17  static char brief[] = "/BR";
18  static char full[] = "/FU";
19  char hdir[27] = {0};
20
21  xls(od, name, code)
22   int od;                /* io object for network data connection */
23   char *name;            /* name of uic to list, null == current  */
24   int code;
25  {
26   register XFILE  *file;
27   char template[16];
28   int rval;
29   int sys_id;
30   int d;
31
32    if (od<0 || od >= _XNFILE)
33          return(XEBADF);
34    if(name && *name && ((code == LS) || (code == LS_ARG))){
35          rval = checkname(name);
36          if(rval < 0)
37                  return(rval);
38
39
40          /* check name is dir or simple name */
41          if(rval) {
42                  d = xaccess(name,FILE_NAME,0);  /* check file exists or not */
43                  if(d < 0){
44                          return(XENOENT);
45                  }
46                  d = xoprintf(&_xiob[od],"%s\n",name);
47                  if(d < 0){
48                          xperror(d,"on output");
49                  }
50                  return(d);
51          }
52     }
53    xstrcpy(template,SCRATCHFILE);
54  /*  xmktemp(template);  */
55    rval = _ls(template, name, code);
56    if(rval < 0)
```

```
 57      return(rval);
 58    sys_id =  opentemp(template);
 59    if(sys_id < 0)
 60      return(sys_id);
 61     file = xodopen(sys_id,"r");
 62    if(file < (XFILE *)0)
 63      return( ( int )file);
 64     xpass(file, &_xiob[od]);
 65     xclose(xfileno(file));
 66     rval= xunlink(template, FILE_NAME);
 67     return(rval);
 68
 69  }
 70
 71  xpass( inod, outod )
 72
 73  XFILE *inod;     /* input EXOS io object */
 74  XFILE *outod;    /* output EXOS io object */
 75  {
 76   int c;
 77   int d;
 78   char name[512];
 79
 80          while(c = xogets(name,sizeof (name) ,inod) > 0){
 81                  if(!nam_list(name))
 82                          continue;
 83                  xoprintf(outod,"%s",name);
 84                  xfflush(outod);
 85          }
 86          if(c < 0) {
 87                  xperror(c, "on input");
 88                  return(c);
 89          }
 90  }
 91
 92  int
 93  _ls(template, name, code)
 94  char *template;
 95  char *name;
 96  int code;
 97  {
 98   char *swtch;
 99   char cmdlin[CMDSIZE];
100   long tsk;
101   int rval;
102   char *cur_name;                /* use to replace user's if supplied or
103                                    pointing to null.      */
104   char inspec[CMDSIZE];
105   char *inp = inspec;
106   char *f_stat = in_stat;
107   char *f_ver_stat = in_ver_stat;
108   char _stat = '\0';
109
110    switch (code){
111     case LS:
112     case LS_ARG:
```

```
113             swtch = brief;
114             break;
115         case LSLONG:
116         case LSLONG_ARG:
117             swtch = full;
118             break;
119         default:
120             return(XEINVAL);
121         }
122
123     if( name == ( char *)0)
124             cur_name = ( char *)&name;
125     else
126             cur_name = name;
127
128     /* create command to produce list */
129     for( ;; ){
130         rval = parse(cur_name,xstrlen(cur_name));
131         if(rval < 0)
132             return(rval);
133         if(globbing){
134             if((f_stat > in_stat) &&
135                ( ((_stat & CS_DVF) && !(csiblk[C_STAT] & CS_DVF)) ||
136                  ((_stat & CS_DIF) && !(csiblk[C_STAT] & CS_DIF))
137                ))
138                     return(-1);     /* command syntax error */
139             _stat |= csiblk[C_STAT];
140             *f_stat++ = csiblk[C_STAT];
141
142             *f_ver_stat = 0;        /* assume version is not specified */
143             if(csiblk[C_STAT] & CS_NMF){
144                int i;
145                for(i=0;i < *((int * )(csiblk + C_FILD));++i)
146                     if(*((char * )(*(int *)(csiblk + C_FILD + 2)) + i) == ';') {
147                        *f_ver_stat = 1;     /* version is indeed specified! */
148                        break;
149                      }
150               }
151             f_ver_stat++;
152          }
153         if((rval = mkname(inp)) == 0 )
154             break;                          /* no more in spec */
155         inp = inspec +  xstrlen(inspec);
156         *inp++ = ',';
157         cur_name += rval + 1;
158      }
159
160      mkcmd(cmdlin, "PIP ", pw->log_dev, ":", pw->login_uic, template, "=", inspec,
161                                      swtch, "/NM", 0);
162      return(cmdcall(cmdlin));
163     }
164
165     checkname(name)
166     char *name;     /* IN-OUT */
167     {
168     int rval;
```

```
169
170         rval = parse(name,xstrlen(name));
171         if(rval < 0)
172                 return(rval -512);
173         rval = 0;
174         if(csiblk[C_STAT] & CS_NMF)
175                 rval = 1;
176         return(rval);
177 }
178
179 opentemp(file)
180 char *file;
181 {
182 /*
183 char name[27];
184
185         xstrcpy(name,pw->login_uic);
186         xstrcat(name,file);
187         return(xdopen(name, XFREAD|XFASCII, HOME_DIR));
188 */
189         return(xdopen(file, XFREAD|XFASCII, HM_RELATIVE));
190
191 }
192
193 nam_list(name)
194 char *name;
195 {
196  char buf[27];
197  char *cp1, *cp2;
198
199         if(xstrlen(name) > 1)
200            if((name[1] == 'i') || (name[2] == 'i')){
201                 if(globbing){
202                         f_stat++; f_ver_stat++;
203                         hdir[0] = '\0';
204                         if(!(*f_stat & (CS_DVF | CS_DIF)))
205                                 return(0);
206                         cp1 = xstrrchr(name, ' ') + 1;
207                         cp2 = xstrrchr(name, ':') + 1;
208                         if(*f_stat & CS_DVF)
209                            xstrncat(hdir, cp1, cp2 - cp1);
210                         if(*f_stat & CS_DIF)
211                            xstrcat(hdir, cp2);
212                         remtrail(hdir);
213                 }
214                 return(0);
215         }
216         if(( name[0] == '\14') || (name[0] == '\n')
217                                 || (name[0] == '\r'))
218                 return(0);
219         if( (name[0] == ' ') ||
220             (name[1] == '-') ||
221             (name[2] == '-') ||
222             (xstrncmp(name, "Total of ",9) == 0)
223           )
224                 return(0);
```

```
225             if(xstrncmp(name, ".;",2) == 0)
226                     return(0);
227
228             if(globbing && !(*f_ver_stat))
229                     remver(name);
230
231             if(globbing && xstrlen(hdir)){
232                     xstrcpy(buf,hdir);
233                     xstrcat(buf, name);
234                     xstrcpy(name, buf);
235             }
236             return(1);
237     }
238
239
240     remver(s)
241     char *s;
242     {
243      s = xstrchr(s,';');
244      while(*s)
245        *s++ = '\0';
246     }
```

```
1  #include <xgenlib.h>
2  #include <xpwd.h>
3  extern struct  passwd *pw;
4  char *
5  xmktemp(template)
6  char *template;
7  {
8          return(1);
9  }
```

```
 1  /*
 2
 3  Rsx routine to form file names relative to users logoin uic, current
 4  uic, etc.
 5  This routine belongs in Xoslib, but is here to keep the linker happy.
 6  */
 7  #include <xspecial.h>
 8  #include <xgenlib.h>
 9  #include <xpwd.h>
10
11  /*
12  for now ...
13  */
14  #define xsprintf sprintf
15  extern struct passwd *pw;
16
17  xmodname( name, special, buf, sz_buf )
18
19  char **name;
20  int special;
21  char *buf;
22  int sz_buf;
23  {
24  int rval;
25  char *pt = buf;
26
27  switch( special ) {
28          case FILE_NAME:
29                  rval = parse(*name,xstrlen(*name));
30                  if(rval < 0)
31                          return(rval);           /* error while parsing */
32                  mkname(pt);
33                  break;
34          case CURRENT_DIR:
35                  xstrcpy(buf, pw->cur_uic);
36                  break;
37          case HM_RELATIVE:
38          case HOME_DIR:
39                  xstrcpy(buf, pw->log_dev);
40                  xstrcat(buf, ":");
41                  xstrcat(buf, pw->login_uic);
42                  if (special == HM_RELATIVE )
43                          xstrcat(buf, *name);
44                  break;
45          case CD_RELATIVE:
46                  xstrcpy(buf, pw->cur_uic);
47                  xstrcat(buf, *name);
48                  break;
49          default:
50                  return( XEINVAL );
51          }
52  *name = pt;
53  return( 0 );
54  }
```

```
 1  #include <xgenlib.h>
 2  #include <xspecial.h>
 3  #include <xpwd.h>
 4  extern struct passwd *pw;
 5
 6  xpwd(buf, buflen, func_code)
 7   char *buf;              /* buffer to hold name of current uic */
 8   int buflen;             /* length of buffer                   */
 9   int func_code;          /* consistency check                  */
10
11   {
12
13   if(func_code != PWD)
14     return(XEINVAL);
15
16   xstrncpy(buf,pw->cur_dev,buflen);
17   xstrncat(buf,":",buflen);
18   xstrncat(buf, pw->cur_uic, buflen - xstrlen(pw->cur_dev) - 1 );
19   buf[buflen] = '\0';
20   return(0);
21
22  }
```

```
 1  /*
 2  %W% %G%
 3  convert a RSX file descriptor to an EXOS io object
 4  -       useful in debugging and development.
 5  */
 6
 7  #include <xstdio.h>
 8
 9  extern xdread();
10  extern xdwrite();
11  extern xdclose();
12  extern xnofunc();
13
14  xrxtex( rsxfd )
15  int rsxfd;
16  {
17  int rval;
18  int exosfd;
19  register XFILE *file;
20
21  rval = rsxfd;
22  if( rval < 0 )
23          return( rval - 512);
24  exosfd = xnewod();              /* get a free file descriptor */
25  if( exosfd < 0 )
26          return( exosfd );
27  file = &_xiob[ exosfd ];
28  file->_file = exosfd;
29  file->_flag |= _XIORW | _XPrimary ;
30  file->_sys_id = (char *)rsxfd;
31  file->_read = xdread;
32  file->_write = xdwrite;
33  file->_ioctl = xnofunc;
34  file->_close = xdclose;
35  return( exosfd );
36  }
```

```
 1  /*
 2   *      FILENAME        XSELECT.C
 3   *
 4   */
 5
 6  #include <xgenlib.h>
 7  #include "libhdr.c"
 8
 9  #define SELECT_EFN      4
10  #define READ    0
11  #define WRITE   1
12
13  extern int _astrselect();
14  extern int _astwselect();
15
16  long rmask = (long) 0;
17  long wmask = (long) 0;
18  long *prmask = (long *) 0;
19  long *pwmask = (long *) 0;
20  int nfounds = 0;
21  struct iosb iosb_select = {0};
22  unsigned char rsavxiob[_XNFILE] = {0};
23  unsigned char wsavxiob[_XNFILE] = {0};
24
25  xselect(nods,readods,writeods,timeout)
26  int nods;
27  long *readods;
28  long *writeods;
29  long timeout;
30  {
31          int i,ch_no;
32          int tick = (int )( timeout / 20L);
33
34          if( !readods && !writeods)
35                  return(0);
36          rmask = wmask = (long ) 0;
37          emt(CLEF, SELECT_EFN);  /* make sure efn is clear */
38          emt(DSAR);              /* disable ast recoginition */
39
40          if(readods)
41             for( i = 0; i < nods ; ++i){
42                  if(getod(readods,i)) {
43                          ch_no = (int ) _xiob[i]._sys_id;
44                          rsavxiob[ch_no] = i;
45                          emt(QIO,IO_ACS|SA_SEL,SOLUN,0,&iosb_select,
46                                  _astrselect,0,0,0,READ,0,ch_no);
47                       }
48                  }
49          if(writeods)
50             for( i =0; i < nods; ++i){
51                  if(getod(writeods,i)) {
52                          ch_no = (int ) _xiob[i]._sys_id;
53                          wsavxiob[ch_no] = i;
54                          emt(QIO,IO_ACS|SA_SEL,SOLUN,0,&iosb_select,
55                                  _astwselect,0,0,0,WRITE,0,ch_no);
56                       }
```

```
57                  }
58
59          /* initialize mask and return values    */
60
61          rmask = (readods ? *readods : 0);
62          wmask = (writeods ? *writeods : 0);
63          prmask = readods;
64          pwmask = writeods;
65          *prmask = *pwmask = (long ) 0;
66          nfounds = 0;
67
68          /* specify timeout efn          */
69
70          emt(MRKT,SELECT_EFN,(int )tick,1,0);
71
72          emt(ENAR);                          /* enable ast recoginition */
73          emt(WTSE,SELECT_EFN);   /* wait for eithr timeout or at least one ast*/
74          emt(DSAR);                          /* disable ast recoginition */
75          unselect(nods,readods,writeods);/* unselect unready od's */
76          rmask = wmask = (long )0;           /* now on ast must be ignore */
77          emt(ENAR);                          /* enable ast recoginition */
78          return(nfounds);
79
80
81  }
82
83  /*
84   * Ast service routine.
85   */
86
87  astrselect(iosb)
88  struct iosb *iosb;
89  {
90          int ch_no = iosb->nread;
91
92          astselect(&rmask,prmask,rsavxiob[ch_no]);
93
94  }
95
96  astwselcet(iosb)
97  struct iosb *iosb;
98  {
99          int ch_no = iosb->nread;
100
101         astselect(&wmask,pwmask,wsavxiob[ch_no]);
102
103 }
104
105 astselect(mask,pmask,s)
106 long *mask;
107 long *pmask;
108 int s;          /* xiob number */
109 {
110
111         if(!getod(mask,s))
112                 return;                 /* spurious ast  */
```

```
113             setod(pmask,s);
114             nfounds++;
115             emt(SETF,SELECT_EFN);
116     }
117
118     getod(mask,f)
119     long *mask;
120     int f;
121     {
122             int *p = (int *) mask;
123             long r = (long )( 1 << f);
124             int *q = (int *)(&r);
125
126             if(*p++ & *q++)
127                     return(1);
128             if(*p   & *q  )
129                     return(1);
130             return(0);
131     }
132
133     setod(mask,f)
134     long *mask;
135     int f;
136     {
137             int *p = (int *) mask;
138             long r = (long )(1 << f);
139             int *q = (int *)(&r);
140
141             *p++ |= *q++;
142             *p |= *q;
143     }
144
145     /*
146      * This routine unslects the select requests after the time out expires
147      * and the od's that are ready are not unselected.
148      */
149
150     unselect(nods,readods,writeods)
151     int nods;
152     long *readods;
153     long *writeods;
154     {
155             int ch_no,i;
156
157             if(readods)
158                for( i = 0; i < nods ; ++i)
159                     if(getod(&rmask,i))
160                        if(!getod(readods,i)) {
161                           ch_no = (int )_xiob[i]._sys_id;
162                           emt(QIOW,IO_ACS|SA_USL,SOLUN,SOEFN,0,0,0,0,0,0,0,ch_no);
163                          }
164             if(writeods)
165                for( i = 0; i < nods ; ++i)
166                     if(getod(&wmask,i))
167                        if(!getod(writeods,i)) {
168                           ch_no = (int )_xiob[i]._sys_id;
```

```
169                         emt(QIOW,IO_ACS|SA_USL,SOLUN,SOEFN,0,0,0,0,0,0,0,ch_no);
170                        }
171  }
```

```
 1  /*
 2
 3  Xsleep(3X) for RSX.
 4  */
 5
 6  #include <rsxos.h>
 7  xsleep( time )
 8
 9  int time;
10  {
11
12  emt(MRKT,10,time,2,0);
13  emt(WTSE,10);
14  }
```

```
 1  /*
 2
 3  Xsyserr(3X) for RSX - does nothing.
 4  */
 5
 6  static char xsysmsg[] = "unspecified error";
 7
 8  char *xsyserr()
 9
10  {
11
12  return( xsysmsg );
13  }
```

```
 1  /*
 2   *  filename:  XTERM.C
 3   */
 4
 5  #include <rsxos.h>
 6
 7  #define XECHO           1
 8  #define XLINE_EDIT      2
 9  #define XOFF_STERM      0
10  #define XON_STERM       1
11  #define TC_NEC          047
12  #define TC_BIN          065
13  #define TTYEFN  1
14
15  extern int ttylun;
16  extern int ttyraw;
17  extern int ttyecho;
18
19  int (*_ttyhndlr)() = 0;         /* pointer to user handler routine   */
20  extern int _astty();            /* AST routine to service unsolicated ^C */
21
22  struct term_char{
23    unsigned char name;                   /* option or characteristics */
24    unsigned char value;                  /* setting           */
25    };
26
27  struct iosb {
28    unsigned char cc;
29    unsigned char lc;
30    unsigned short nread;
31    };
32
33  xsetterm(option, on_off)
34    int option;           /* XECHO  or  XLINE_EDIT        */
35    int on_off;           /* 1 == on ; 0 == off           */
36    {
37      int rval = 0;
38      struct iosb iosb;
39      struct term_char t_char;
40
41      t_char.name  = ( option == XECHO ) ? TC_NEC : TC_BIN ;
42      t_char.value = !on_off;
43      if(t_char.name == TC_NEC)
44      ttyemt( SF_SMC, ttylun, &iosb, &t_char, sizeof (t_char));
45     if(option == XLINE_EDIT) {
46          rval = !ttyraw;
47          ttyraw = !on_off;
48          }
49    else {
50          rval = ttyecho;
51          ttyecho = on_off;
52          }
53      return ( rval );
54    }
55
56  xrestore_term()
```

```
57    {
58      xsetterm( XECHO, XON_STERM);         /* set echo      */
59      xsetterm( XLINE_EDIT, XON_STERM);    /* set interactive mode */
60    }
61
62  xint_term(handler)
63    int (*handler)();
64    {
65      struct iosb iosb;
66      int rval;
67
68      _ttyhndlr = handler;
69      /*
70      rval = ttyemt(QIO, IO_ATA, ttylun, 0, &iosb, 0,
71                    0, 0, _astty, 0, 0, 0);
72      */
73      rval = 0;
74      return ( rval );
75    }
76
77
78  xraw_term( handler )
79
80  int (*handler)();
81  {
82  int rval;
83
84  xint_term( handler );
85  rval = xsetterm( XECHO, XOFF_STERM );
86  if ( rval < 0 )
87          {
88          return( rval );
89          }
90  rval = xsetterm( XLINE_EDIT, XOFF_STERM );
91  if ( rval < 0 )
92          {
93          xsetterm( XECHO, XON_STERM );
94          return( rval );
95          }
96  return( 0 );
97  }
```

```
 1
 2  #include <rsxos.h>
 3
 4  long xtime()
 5  {
 6   int buf[8];
 7
 8   emt(GTIM, buf);         /* return parm :
 9                                   word 0 : --     year
10                                   word 1 : --     month
11                                   word 2 : --     day
12                                   word 3 : --     hour
13                                   word 4 : --     min
14                                   word 5 : --     sec
15                           */
16
17   return( ((buf[2]*24 + buf[3])*60 + buf[4])*60 + buf[5] );
18
19  }
```

```
 1  /*
 2
 3  Save code space, at the expense of time by providing a common
 4  routine to translate exos open mode flags to RSX mode and types.
 5  */
 6  #include <xerrno.h>
 7  #include <xstdio.h>
 8  #define FO_RD   01
 9  #define FO_WRT  016
10  #define FO_APD  0106
11  #include <xspecial.h>
12
13  xtranmode( mode, ioflag )
14
15  register int mode;
16  int *ioflag;            /* flag to go into _xiob structure */
17  {
18  int rmode;
19
20  /*
21  Translate mode to RSX open modes.
22  */
23  if( mode & XFWRITE )
24          {
25          if( mode & XFREAD )
26                  *ioflag = _XIORW | _XPrimary;
27          else
28                  *ioflag = _XIOWRT;
29                  if( mode & XFAPPEND )
30                          rmode = FO_APD;
31                  else    rmode = FO_WRT;
32          }
33  else if( mode & XFREAD )
34          {
35          if( mode & (XFAPPEND | XFCREAT | XFTRUNC))
36                  {
37                  xperror( XEINVAL, "read and other flags" );
38                  return( XEINVAL );
39                  }
40          *ioflag = _XIOREAD | _XPrimary;
41          rmode = FO_RD;
42          }
43  else
44          {
45          xperror( XEINVAL, "not read or write" );
46          return( XEINVAL );
47          }
48  return( rmode );
49  }
```

```
 1
 2  /*
 3   * filename: XTTY.C
 4   */
 5
 6  #include <xstdio.h>
 7  #include <xspecial.h>
 8  #include <xerrno.h>
 9  #include <rsxos.h>
10  #include <fcs.h>
11
12  extern xttyread();
13  extern xttywrite();
14  extern xttyclose();
15
16  extern struct _rcb _rcb[];
17  #define TTYEFN 1
18  #define CNTRLZ 0366
19
20
21  int ttylun = 5;                 /* lun associated with the TI: */
22  int ttyraw = 0;                 /* 1 == raw  0 == line_edit */
23  int ttyecho = 0;                /* 1 == echo on  0 == echo off */
24
25  xttyopen( mode)
26  register int mode;
27  {
28  int exosfd;
29  int rmode;
30  int ioflag;
31  register XFILE *file;
32  register struct _rcb *sysid = _rcb;
33  int     i;
34
35  /*
36  Translate mode to Rsx mode and type.
37  */
38  rmode = xtranmode( mode, &ioflag);
39  if ( rmode < 0 )
40          return( rmode );
41  for( i=0; i < _XNFILE; ++i, ++sysid )
42          if( sysid->flags & RFREE )
43                  break;
44  if( i >= _XNFILE )
45          return XEMFILE;         /* Too many files open  */
46  sysid->flags = RUSED;
47  sysid->rlun = ttylun;           /* set LUN OF TI:       */
48  exosfd = xnewod();              /* get a free file descriptor */
49  if( exosfd < 0 )
50          return( exosfd );
51  file = & _xiob[exosfd];
52  file->_file = exosfd;
53  file->_flag |= ioflag;
54  file->_sys_id = sysid;
55  file->_read = xttyread;
56  file->_write = xttywrite;
```

```
 57  file->_close = xttyclose;
 58  return( exosfd );
 59  }
 60
 61  xttyclose( sysid )
 62  register struct _rcb *sysid;
 63  {
 64     return(0);
 65     /* its a null procedure */
 66    }
 67
 68  extern struct ttybuf ttybuf;
 69
 70  xttyread(sysid, buf, len)
 71  register struct _rcb *sysid;
 72  char *buf;
 73  int len;
 74  {
 75      int ret;
 76      struct iosb iosb;
 77      int lun = sysid->rlun;
 78      int io_fun = IO_RVB;
 79
 80      if(ttyraw) {
 81          /* in raw mode read 1 character */
 82          len = 1;
 83          io_fun |= TF_RAL;
 84          ret = ttyemt(io_fun,lun,&iosb,buf,len);
 85          return(ret);
 86          }
 87      else {
 88          if(ttybuf.tsize == 0) {
 89      ret = ttyemt(io_fun,lun,&iosb,ttybuf.linetty,132);
 90          if(ret < 0)
 91                  return(ret);
 92          if ((int) iosb.cc == CNTRLZ) {
 93                  xstdin->_flag |= _XIOEOF;
 94                  return(0);
 95                  }
 96            ttyemt(IO_WVB, lun, &iosb, "\n", 1);
 97                              /* give lf after reading a line */
 98              if ( ret >= 0 ) {
 99                if( sysid->flags & DBLBUF )       /* file is used for network */
100                    ttybuf.linetty[ret++] = '\r';
101                ttybuf.linetty[ret++] = '\n';
102                ttybuf.cur_pos = ttybuf.linetty;
103                ttybuf.tsize = ret;
104                }
105              }
106          ret = (len > ttybuf.tsize) ? ttybuf.tsize : len;
107          xbcopy(ttybuf.cur_pos,buf,ret);
108          if(ttybuf.tsize > ret) {
109                  ttybuf.tsize -= ret;
110                  ttybuf.cur_pos += ret;
111                  }
112          else {
```

```
113                     ttybuf.tsize = 0;
114                     ttybuf.cur_pos = ttybuf.linetty;
115                 }
116             }
117
118         return ( ret );
119     }
120
121
122     /*
123      * Objective of this function is to process different type of error resulting
124      * from a call to the driver via QIO ( or emt call in 'C' ) call. A QIO
125      * executive directive call reports error in two different ways through the
126      *  DSW ( directive status word ) and also in the IO statusblock.  Again in the
127      * IOSB it is divided into two parts one device specific and the other generic.
128      * The generic and the dsw are returned to the caller after shifting it by -512
129      * and the device specific code is just sign changed. If all is fine then an
130      * non zero value is returned.
131      */
132
133     int ttyemt(cmd,lun,iosb,p1,p2)
134       unsigned short cmd, lun;
135       struct iosb *iosb;
136       unsigned short p1, p2;
137       {
138        int dsw;
139
140        dsw = emt(QIOW, cmd, lun, TTYEFN, iosb, 0, p1, p2, 0, 0, 0, 0);
141        if  ( dsw < 0 )
142            return ( dsw - 512 );                   /* directive error */
143        else
144            return ( iosb->nread );
145       }
146
147
```

```
 1  /*
 2
 3  Xunlink for RSX.
 4  */
 5  #include <xspecial.h>
 6  #include <xgenlib.h>
 7
 8
 9  xunlink( name, special )
10
11  char *name;
12  int special;
13  {
14  char buf[MXNAMELEN +1];
15  int rval;
16  int ver = 0;
17  char *p = name;
18  char cmdlin[CMDSIZE];
19
20  rval = xmodname( &name, special, buf, sizeof(buf) );
21  if(rval<0)
22    return(rval);
23  while( *p++ )
24     if( *p == ';'){
25          ver = 1;
26          break;
27          }
28   if(ver)
29      mkcmd(cmdlin, "PIP ", name, "/DE/NM",  0 );
30   else
31      mkcmd(cmdlin, "PIP ", name, ";0", "/DE/NM",  0 );
32  return(cmdcall(cmdlin));
33  }
```

```
;
; FILENAME:     ASCBIN.MAC
;
;
;ASCPP: --> convert ascii dir string to binary uic.
;
;               ascpp(pstr, puic)
;               char *pstr;     /* INPUT */
;               int  *puic;     /*OUTPUT */
;
        .TITLE  ASCBIN
        .IDENT  /01/


C$SPRT=0
        .PSECT  C$TEXT,I,RO
ASCPP::                         ; global reference label
        .IF DF C$SPRT

        JSR     R5,C$SAV        ; make it 'C' callable
        MOV     R5,-(SP)        ; save C frame pointer
        MOV     4(R5),R2        ; address of string to be converted
        MOV     6(R5),R3        ; address of uic

        .ENDC


        CALL    .ASCPP          ; system lib routine to convert string
                                ; to binary uic.
        CLR     R0              ; return status
        BCC     RTN             ;
        MOV     #-1,R0          ;
        JMP     RTN

; PPASC: -->    Convert binary uic to ascii dir string
;
;               ppasc(psrt, puic)
;               char *pstr;     /*OUTPUT */
;               int  *puic;     /* INPUT */
PPASC::                         ; global reference label
        .IF DF C$SPRT

        JSR     R5,C$SAV        ; make it 'C' callable
        MOV     R5,-(SP)        ; save C frame pointer
        MOV     4(R5),R2        ; address of string to be return
        MOV     6(R5),R3        ; address of uic to be converted

        .ENDC

        MOV     #1,R4           ; control code ---
                                ; bit 0 is 1
                                ; bit 1 is 0
        CALL    .PPASC          ; system lib routine to convert
                                ; bin uic to string dir
        CLR     R0              ; return status
        BCC     BIN
```

```
57          MOV     #-1,R0          ;
58  BIN:
59  RTN:
60          .IF DF C$SPRT
61
62          MOV     (SP)+,R5        ; adjust frame pointer
63          JMP     C$RET           ; return to caller
64
65          .IFF
66
67          RETURN
68
69          .ENDC
70
71
72          .PSECT C$TEXT,I,RO
73          .EVEN
74          .END
```

```
 1
 2  ;
 3
 4  ; ASTLCONN: --> This is an ast service routine corresponds to directive
 5  ;               SREX$. It clean up the stack and calls lostconn to perform
 6  ;               abnormal termination of task and then exits.
 7  ;
 8
 9          .TITLE  ASTLCO
10          .IDENT  /01/
11          .MACRO  SAVE
12
13          MOV     R0,-(SP)
14          MOV     R1,-(SP)
15          MOV     R2,-(SP)
16          MOV     R3,-(SP)
17          MOV     R4,-(SP)
18          MOV     R5,-(SP)
19
20          .ENDM
21
22          .MACRO  UNSAVE
23
24          MOV     (SP)+,R5
25          MOV     (SP)+,R4
26          MOV     (SP)+,R3
27          MOV     (SP)+,R2
28          MOV     (SP)+,R1
29          MOV     (SP)+,R0
30
31          .ENDM
32
33          .MCALL  ASTX$S,DSAR$S,SETF$S
34          .PSECT  C$TEXT,I,RO
35  ;
36  ;ASTLCO::
37  ;       .MCALL  ASTX$S
38  ;
39  ;       SAVE                    ; save all registers
40  ;
41  ;       CALL    LOSTPEER        ;
42  ;       CALL    XEXIT           ; close all the files.
43  ;
44  ;       UNSAVE                  ; unsave all registered
45  ;
46  ;       MOV     (SP)+,(SP)+     ; clean up stack
47  ;       ASTX$S                  ; ast service exit.
48  ;;
49
50
51  .ASTRS::
52          SAVE                    ; save all registers
53          MOV     14(SP),-(SP)    ; get iosb address as first parameter
54          JSR     PC,ASTRSELECT   ; call ast-select service routine.
55          TST     (SP)+           ; pop off parameter
56          UNSAVE                  ; unsave all registered
```

```
57          TST     (SP)+           ; pop off stack for ASTX$S
58          ASTX$S                  ; exit from AST routine
59
60  .ASTWS::
61          SAVE                    ; save all registers
62          MOVB    14(SP),-(SP)    ; push char as parameter for trstat
63          JSR     PC,ASTWSELECT   ; call ast-select service routine.
64          TST     (SP)+           ; pop off parameter
65          UNSAVE
66          TST     (SP)+           ; pop off stack for ASTX$S
67          ASTX$S
68
69          .PSECT C$TEXT,I,RO
70          .EVEN
71          .PSECT C$DATA,D,RW
72          .EVEN
73          .END
```

```
 1
 2
 3  ;
 4  ; FILENAME:      ASTTY.MAC
 5  ;
 6  ;                This is an ast service routine corresponds to int_term
 7  ;                routine( QIO IO_ATA ). It calls the user handler to
 8  ;                process the char ^C, not by MCR.
 9  ;
10
11          .psect  c$text,i,ro
12
13          .title  .astty
14          .globl  .TTYHN
15
16          .MCALL  ASTX$S
17  .ASTTY::
18          MOV     R0,-(SP)        ; save R0
19          MOV     R1,-(SP)        ; save R1
20          MOV     R2,-(SP)        ; save R2
21          MOV     R3,-(SP)        ; save R3
22          MOV     R4,-(SP)        ; save R4
23          MOV     R5,-(SP)        ; save R5
24          JSR     PC,@.TTYHN      ; call handler to process interrupt
25          MOV     (SP)+,R5        ; pop off R5
26          MOV     (SP)+,R4        ; pop off R4
27          MOV     (SP)+,R3        ; pop off R3
28          MOV     (SP)+,R2        ; pop off R2
29          MOV     (SP)+,R1        ; pop off R1
30          MOV     (SP)+,R0        ; pop off R0
31          TST     (SP)+           ; pop off stack for ast
32          ASTX$S                  ; exit from AST routine
33
34          .END
```

```
 1  ;
 2  ; FILENAME      CHDR
 3  ;
 4  ;               This file contains start & end entry points
 5  ;
 6
 7          .TITLE  CHDR
 8          .MCALL  FSRSZ$,GMCR$,DIR$ ,gtsk$s,wtse$s,spwn$s
 9          .PSECT C$DATA,D,RW
10  GMCR:
11          GMCR$
12  FSRSZ:
13          FSRSZ$  0
14  ;buf:
15  ;       .blkw   16.             ; used to store task info.
16  ;cmd:
17  ;       .ascii  /REA /
18  ;tsk:
19  ;       .ascii  /      /
20  ;lun:
21  ;       .ascii  /   SY0:/
22  ;cmdl   = .-cmd
23  ;       .even
24  ;cli:
25  ;       .rad50  /MCR.../
26          .MCALL  EXIT$S,ALUN$S
27          .PSECT C$TEXT,I,RO
28  START:
29  ;
30  ;       make sure task's default device is same as user's login device
31  ;
32  ;       gtsk$s  #buf            ; get task name
33  ;       bcs     exit            ; if CS error
34  ;       mov     #tsk,r0         ; address of first three byte of task-name
35  ;       mov     buf,r1          ; first word of task-name
36  ;       call    $c5ta           ; convert it to ascii
37  ;       bcs     exit            ; if cs error
38  ;       mov     #tsk+3,r0       ; next three byte of task-name
39  ;       mov     buf+2,r1        ; second word of task-name
40  ;       call    $c5ta           ; convert rad50 to ascii
41  ;       bcs     exit            ; if cs error
42  ;       mov     #4,r3           ; # times REA to be spawned
43  ;AGN:
44  ;       mov     r3,r0                   ; LUN #
45  ;       add     #60,r0                  ; make it char
46  ;       movb    r0,lun+1                ;
47  ;       spwn$s  #cli,,,,,#1,,,#cmd,#cmdl        ; spawn REA
48  ;       bcs     exit            ; if cs error
49  ;       wtse$s  #1              ; wait for task to complete
50  ;       sob     r3,agn          ; lopp
51  ;
52  ; get mcr command line
53  ;
54          DIR$    #GMCR           ; get command line
55          MOV     $DSW,R1         ; get # of char read or error
56          BLT     ERR             ; error
```

```
57          CLRB    GMCR+2(R1)      ; clear terminator char in command line
58  CONT:
59          MOV     #GMCR+2,-(SP)   ; push address of cli buffer
60          CALL    CMAIN           ; call start entry points
61          TST     (SP)+           ; pop the parameter
62          BR      EXIT
63  ERR:
64          CMP     #IE.AST,$DSW    ; check no command line
65          BEQ     CONT            ; yes
66  EXIT::
67          EXIT$S                  ; exit
68          .END START
```

```
;
;       .title  cmdcall  - spawn MCR command line
C$SPRT = 0
        .mcall  gtsk$s,spwn$s,wtse$s
;pwlog  = 22                    ; offset of login UIC in password structure
        .psect  c$data,d,rw
desc:   .blkw   2.              ; string descriptor
uic:    .word   0               ; UIC
task:   .blkw   2.              ; task name
buf:    .blkw   16.             ; used as task infor block and emit status block
        .psect  c$text,i,ro
cmdcall::
        .IF DF C$SPRT
        jsr     r5,c$sav        ; make it 'C' callable
        mov     r5,-(sp)        ; save frame pointer
        mov     4(r5),r4        ; r4 - pointer to command line
        .ENDC

        mov     pw,r0           ; set r0 to address of login UIC
        mov     r0,desc+2       ; put login UIC address in descriptor also
        mov     r0,-(sp)        ; get string length
        jsr     pc,xstrle       ; get string length
        tst     (sp)+           ; pop the argument
        mov     #desc,r2        ; r2 get string descriptor address
        mov     r0, (r2)        ; store in descriptor

        mov     #uic,r3         ; r3 has address of binary UIC
        call    .ascpp          ; convert UIC
        gtsk$s  #buf            ; get task information
        mov     G.TSTN+buf, task ; save task name
        mov     G.TSTN+buf+2, task+2 ; save second half

;       now check if the command to be spawned is UFD
;
        mov     r4, r0          ; r0 has address of command line string
        mov     #1, r1          ; accept period as valid RAD50 character
        call    $cat5           ; convert to RAD50
        cmp     R1,#^RUFD       ; is it UFD?
        beq     10$             ; yes
        cmp     task,#^RFTD     ; am I a FTP server?
        bne     10$             ; no, must be a client
;
;       I am a FTP server and command to be spawned is not UFD
;
        add     #3, r4          ; command starts at 3 characters away
        mov     #^RXDR, task    ; task name is XDR00n
        br      20$
;
;       This is for FTP client or if command is UFD
;
10$:
        mov     #^RMCR, task
        mov     #^R..., task+2  ; task name is MCR...
20$:
;
;       now spawn the task
;
```

```
57          mov     r4, -(sp)       ; get length of command line
58          jsr     pc, xstrle
59          tst     (sp)+           ; reset stack
60          spwn$s  #task,,,uic+1,uic,#1,,#buf,r4,r0        ; spawn task
61          bcs     90$             ; spawn error
62          wtse$s  #1              ; wait for task complete
63          cmp     buf,#1          ; is task OK?
64          bne     30$             ; no-
65          clr     r0              ; yes, return 0
66          br      99$
67  30$:
68          mov     buf,r0          ; error in task exit
69          sub     #512., r0       ; return (esb[0] - 512)
70          br      99$
71  90$:
72          mov     $dsw,r0         ; return (dsw)
73  99$:
74          .IF     DF C$SPRT
75          mov     (sp)+, r5       ; adjust frame pointer
76          jmp     c$ret
77          .IFF
78          RETURN
79          .ENDC
80          .END
```

```
;
; FILENAME      DBLAST.MAC
;
;               This file contain ast service entry points for i/o
; RASTDIO       AST for read
; WASTDIO       AST for write
; ASTSIO        AST for socket i/o
;

        .TITLE  DBLAST
        .IDENT  /01/
;
; MODULE        ASTDIO
;
;               AST SERVICE FOR READ & WRITE
;

        .MACRO  SAVE
        MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R4,-(SP)
        MOV     R5,-(SP)
        .ENDM
        .MACRO  UNSAVE
        MOV     (SP)+,R5
        MOV     (SP)+,R4
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        .ENDM
        .MCALL ASTX$S
        .PSECT  C$TEXT,I,RO
RASTDIO::
        SAVE                                ; SAVE ALL REGISTERS
        MOV     HBUF+10,R0                  ; get address of xiob
        MOV     20(R0),R0                   ; get address of .rcb
        MOV     20(R0),R0                   ; get address of fdb
        CMP     F.BKVB(R0),F.EFBK(R0)       ; IT IT LAST BLOCK
        BGT     SET                         ; IF GT YES
        CMP     F.BKVB+2(R0),F.EFBK+2(R0); IS IT LAST BLOCK
        BGT     SET                         ; IF GT YES
        BR      NEXT
SET:
        MOV     14(SP),R1                   ; GET ADDRESS OF IOSB
        MOV     F.FFBY(R0),2(R1)            ; # OF BYTES READ
NEXT:
        MOV     14(SP),-(SP)                ; PUSH IOSB ADDRESS
        JSR     PC,DSTAT                    ; FILL UP THE RETURN STATUS
        TST     (SP)+                       ; POP THE PARAMETER
        UNSAVE                              ; UNSAVE ALL THE REGISTERS
        TST     (SP)+                       ; POP THE STACK FOR ASTX$S
        ASTX$S                              ; RETURN
WASTDIO::
```

```
57          SAVE                            ; SAVE ALL REGISTERS
58          MOV     HBUF+10,R0              ; get address of xiob
59          MOV     20(R0),R0               ; get address of .rcb
60          MOV     20(R0),R0               ; get address of fdb
61          JMP     NEXT                    ;
62
63
64
65  ;
66  ; MODULE         ASTSIO
67  ;
68  ;               AST SERVICE FOR SOCKET READ/WRITE
69  ;
70
71          .PSECT  C$TEXT,I,RO
72  ASTSIO::
73          SAVE                            ; SAVE ALL REGISTERS
74          MOV     14(SP),-(SP)            ; PUSH ADDRESS OF IOSB
75          JSR     PC,NSTAT                ; FILL UP THE RETURN STATUS
76          TST     (SP)+                   ; POP THE PARAMETER
77          UNSAVE                          ; UNSAVE ALL THE REGISTERS
78          TST     (SP)+                   ; POP THE STACK FOR ASTX$S
79          ASTX$S
80
81          .END
```

```
 1          .title  dummy
 2  getenv::
 3  gethen::
 4  getnba::
 5  getnbn::
 6  getnen::
 7  getpen::
 8  getsbp::
 9  getsen::
10  gpbnam::
11  gpbnum::
12          rts     pc
13          .end
```

```
 1  ;
 2  ; filename:     ENVAST.MAC
 3  ;
 4          .title  ENVAST
 5          .MACRO  SAVE
 6
 7          MOV     R0,-(SP)
 8          MOV     R1,-(SP)
 9          MOV     R2,-(SP)
10          MOV     R3,-(SP)
11          MOV     R4,-(SP)
12          MOV     R5,-(SP)
13
14          .ENDM
15
16          .MACRO  UNSAVE
17
18          MOV     (SP)+,R5
19          MOV     (SP)+,R4
20          MOV     (SP)+,R3
21          MOV     (SP)+,R2
22          MOV     (SP)+,R1
23          MOV     (SP)+,R0
24
25          .ENDM
26
27          .MCALL RCVD$S,SETF$S,ASTX$S
28
29  AST.RE::
30          SAVE                    ; save all registers
31  AGAIN:
32          RCVD$S  ,#msge          ; recieve pkt from task
33          CMP     #IS.SUC,$DSW    ;check for success
34          BEQ     10$             ; If EQ YES
35          BR      EXT             ; no pkt. return
36  10$:
37          SETF$S  #2
38          BR      AGAIN           ; go for next pkt.
39  EXT:
40          UNSAVE                  ; unsave all registered
41          ASTX$S                  ; exit from AST routine
42
43          .END
```

```
    1  ;
    2  ; FILENAME      FIOMAC.MAC
    3  ;
    4  ;               This file contains i/o related entry points.
    5  ; ..CREATE      CREATE
    6  ; ..OPEN        OPEN
    7  ; ..READ        READ
    8  ; ..WRITE       WRITE
    9  ; ..RWAIT       WAIT for read
   10  ; ..WWAIT       WAIT for write
   11  ; ..CLOSE       CLOSE
   12  ;
   13
   14          .TITLE  FIOMAC
   15          .IDENT  /01/
   16
   17          .MCALL FSRSZ$,OPEN$,READ$,WRITE$,WAIT$,CLOSE$,FDBDF$,FDAT$R
   18  ;
   19  ; MODULE        CREATE
   20  ;
   21  ;       INPUT parameters
   22  ;
   23  ;               p1      fdb
   24  ;               p2      type  0 -- ascii type  otherwise binary
   25  ;
   26  ;
   27
   28          .PSECT C$DATA,D,RW
   29
   30
   31  CNTG    =       -5
   32  ALLOC   =       -5
   33
   34          .PSECT  C$TEXT,I,RO
   35  CREATE::
   36          JSR     R5,C$SAV                ; make it 'C' callable
   37          CMP     6(R5),#0                ; check file-type ascii/binary
   38          BEQ     ASC                     ; if EQ ASCII
   39  BIN:                                    ; BINARY
   40          MOV     #R.FIX,R1               ; RTYPE AS FIXED LENGTH RECORD
   41          CLR     R2                      ; RATT AS NO IMPLIED CR
   42          BR      FDAT
   43  ASC:
   44          MOV     #R.VAR,R1               ; RTYP AS VARIABLE LENGTH RECORD
   45          MOV     #FD.CR,R2               ; RATT AS IMPLIED CR
   46  FDAT:
   47          FDAT$R  4(R5),R1,R2,6(R5),#CNTG,#ALLOC
   48                                          ; INITIALIZE ATTRIBUTE SECTION OF FDB
   49          JMP     C$RET                   ; JUMP TO RETURN
   50
   51
   52
   53  ;
   54  ; MODULE        OPEN
   55  ;
   56  ;       INPUT PARAMETERS
```

```
    57  ;
    58  ;               P1      FDB
    59  ;               P2      MODE OF FILE
    60  ;               P3      LUN
    61  ;               P4      DATA SET POINTER
    62  ;
    63
    64          .PSECT  C$TEXT,I,RO
    65  ..OPEN::
    66          JSR     R5,C$SAV                ; MAKE IT 'C' CALLABLE
    67          OPEN$   4(R5),6(R5),10(R5),12(R5),,#FD.RWM,,,EOPEN
    68          CLR     RO                      ; RETURN VALUE
    69          JMP     C$RET                   ; JUMP TO RETURN
    70  EOPEN:
    71          MOV     4(R5),R1                ; GET ADDRESS OF FDB
    72          MOVB    F.ERR(R1),RO            ; ERROR CODE
    73          JMP     C$RET                   ; JUMP TO RETURN
    74
    75
    76  ;
    77  ; MODULE        READ
    78  ;
    79  ;       INPUT PARAMETERS
    80  ;
    81  ;               P1      FDB
    82  ;               P2      BLOCK BUFFER ADDRESS
    83  ;               P3      BKEF
    84  ;               P4      ADDRESS OF IOSB
    85  ;               P5      ADDRESS OF AST
    86  ;
    87
    88
    89  BLKSIZE=512.
    90          .PSECT  C$TEXT,I,RO
    91  ..READ::
    92          JSR     R5,C$SAV                ; MAKE IT 'C' CALLABLE
    93          READ$   4(R5),6(R5),#BLKSIZE,,10(R5),12(R5),14(R5),CKEOF
    94          BCS     CKEOF                   ; CHECK FOR END OF FILE
    95          MOV     #1,RO                   ; RETURN VALUE
    96          JMP     C$RET                   ; JUMP TO RETURN
    97  ERIO:
    98          MOV     4(R5),R1                ; GET ADDRESS OF FDB
    99          MOVB    F.ERR(R1),RO            ; ERROR CODE
   100          SUB     #512.,RO                ; MAKE ERROR AS RSX
   101          JMP     C$RET                   ; JUMP TO RETURN
   102
   103  CKEOF:
   104          MOV     4(R5),R1                ; GET ADDRESS OF FDB
   105          CMPB    F.ERR(R1),#IE.EOF       ; CHECK EOF
   106          BNE     ERIO                    ; IF NE ERROR
   107          CLR     RO                      ; RETURN VALUE
   108          JMP     C$RET                   ; JUMP TO RETURN
   109
   110
   111  ;
   112  ; MODULE        WRITE
```

```
113  ;
114  ;       INPUT PARAMETERS
115  ;
116  ;               SAME AS 'READ'
117  ;
118
119          .PSECT  C$TEXT,I,RO
120  ..WRITE::
121          JSR     R5,C$SAV                ; MAKE IT 'C' CALLABLE
122          WRITE$  4(R5),6(R5),#BLKSIZE,,10(R5),12(R5),14(R5),ERIO
123          BCS     CKEOF                   ; CHECK FOR EOF
124          MOV     #1,R0                   ; RETURN VALUE
125          JMP     C$RET                   ; JUMP TO RETURN
126
127
128
129
130
131  ;
132  ; MODULE         RWAIT
133  ;
134  ;       INPUT PARAMETERS
135  ;
136  ;
137  ;               P1      fdb
138  ;               P2      address of iosb
139  ;
140          .PSECT  C$TEXT,I,RO
141  ..RWAIT::
142          JSR     R5,C$SAV                ; MAKE IT 'C' CALLABLE
143          WAIT$   4(R5),,,ERW
144          CMPB    @6(R5),#0               ; CHECK ERROR
145          BLT     ERW                     ; IF LT ERROR WHILE READ
146  ; SET NO. OF BYTES READ
147          MOV     4(R5),R1                ; GET FDB ADDRESS
148          CMP     F.BKVB(R1),F.EFBK(R1)   ; IS IT LAST BLOCK
149          BGT     SETIO                   ; IF GT YES
150          CMP     F.BKVB+2(R1),F.EFBK+2(R1); IS IT LAST BLOCK
151          BGT     SETIO                   ; IF GT YES
152          JMP     C$RET                   ; JUMP TO RETURN
153  SETIO:
154          MOV     6(R5),R2                ; get address of iosb
155          MOV     F.FFBY(R1),2(R2)        ; GET FIRST FREE BYTE IN BLOCK
156          JMP     C$RET                   ; JUMP TO RETURN
157  ERW:
158          MOV     6(R5),R1                ; GET ADDRESS OF IOSB
159          CMPB    (R1),#IE.EOF            ; eof
160          BEQ     EOF                     ;
161          MOVB    (R1),2(R1)              ; GET I/O ERROR CODE
162          SUB     #512.,2(R1)             ; MAKE ERROR AS 'RSX'
163          JMP     C$RET                   ; JUMP TO RETURN
164  EOF:
165          CLR     2(R1)                   ; return value
166          JMP     C$RET                   ; jump to return
167
168
```

```
169
170  ;
171  ; MODULE        WWAIT
172  ;
173  ;               WAIT FOR DISK WRITE
174  ;
175  ;       INPUT PARAMETERS
176  ;
177  ;               P1      FDB
178  ;               P2      ADDRESS OF IOSB
179  ;
180
181          .PSECT  C$TEXT,I,RO
182
183  ..WWAIT::
184          JSR     R5,C$SAV                ; MAKE IT 'C' CALLABLE
185          WAIT$   4(R5),,,ERW
186          CMPB    @6(R5),#0               ; CHECK ERROR
187          BLT     ERW                     ; IF LT YES, ERROR WHILE WRITE
188          JMP     C$RET                   ; JUMP TO RETURN
189
190
191
192  ;
193  ;
194  ; MODULE        CLOSE
195  ;
196  ;       INPUT PARAMETERS
197  ;
198  ;               P1      fdb
199  ;
200
201
202          .PSECT  C$TEXT,I,RO
203  ..CLOSE::
204          JSR     R5,C$SAV                ; MAKE IT 'C' CALLABLE
205          CLOSE$  4(R5)
206          JMP     C$RET                   ; JUMP TO RETURN
207
208
209
210
211
212  ;
213  ; MODULE        WMREC
214  ;
215  ;               Adjust FDB
216  ;
217  ;       INPUT PARAMETERS
218  ;
219  ;               P1      fdb
220  ;               P2      max record size
221  ;               P3      first free byte in last block
222  ;
223  .WMREC::
224          JSR     R5,C$SAV                ; MAKE IT 'C' CALLABLE
```

```
225          MOV     4(R5),R1                ; GET fdb
226          MOV     6(R5),F.RSIZ(R1)        ; set max rec size
227          MOV     10(R5),F.FFBY(R1)       ; set first freee byte
228          CMP     10(R5),#0               ; check first free byte is 0
229          BEQ     NEXT1                   ; If EQ yes
230          DEC     F.EFBK+2(R1)            ; end of block number
231  NEXT1:
232          JMP     C$RET                   ; JUMP TO RETURN
233          .END
234
```

```
 1  ;
 2  ;       FILENAME        LIBMAC.MAC
 3  ;
 4  ;
 5          .TITLE  LIBMAC
 6          .MCALL  ASTX$S ,EXIT$S
 7          .PSECT  C$TEXT,I,RO
 8
 9  ;
10  ; this routine is the AST service routine specified in the catchoob()
11  ; library call. After it is invoked it simply passes control to another
12  ; library routine called libast() which selectively calls user specified
13  ; handler. The address of the iosb being on top of the stack is automati-
14  ; cally passed to the libstat routine.
15  ;
16  ;
17
18  .ASTCA::                        ; global referance
19
20          JSR     PC,LIBAST       ; call library routine LIBSTAT
21          TST     (SP)+           ; pop off stack to adjust for the ASTX call
22          ASTX$S                  ; exit from ast routine
23
24  ;
25  ; $EXIT : exit from current task
26  ;
27
28  $EXIT:: EXIT$S                  ; make an task exit
29
30
31
32          .PSECT  C$TEXT,I,RO
33
34          .EVEN
35          .END
36
```

```
 1  ;
 2  ; filename:     MUXAST.MAC
 3  ;
 4          .title  MUXAST
 5  ;               This file contains the two ast service routines that are
 6  ;       called by the system when read is completed on either the network
 7  ;       or the terminal. These in turn call C routines that set the return
 8  ;       status for the net_read and net_write "processes".
 9  ;
10          .MACRO  SAVE
11
12          MOV     R0,-(SP)
13          MOV     R1,-(SP)
14          MOV     R2,-(SP)
15          MOV     R3,-(SP)
16          MOV     R4,-(SP)
17          MOV     R5,-(SP)
18
19          .ENDM
20
21          .MACRO  UNSAVE
22
23          MOV     (SP)+,R5
24          MOV     (SP)+,R4
25          MOV     (SP)+,R3
26          MOV     (SP)+,R2
27          MOV     (SP)+,R1
28          MOV     (SP)+,R0
29
30          .ENDM
31
32          .MCALL  ASTX$S,DSAR$S,SETF$S
33
34  ASTRD1::
35          SAVE                    ; save all registers
36          MOV     14(SP),-(SP)    ; get iosb address as first parameter
37          JSR     PC,NRSTAT       ; fill up the return status
38          TST     (SP)+           ; pop off parameter
39          UNSAVE                  ; unsave all registered
40          TST     (SP)+           ; pop off stack for ASTX$S
41          ASTX$S                  ; exit from AST routine
42
43  ASTRD2::
44          SAVE                    ; save all registers
45          MOVB    14(SP),-(SP)    ; push char as parameter for trstat
46          JSR     PC,TRSTAT       ;
47          TST     (SP)+           ; pop off parameter
48          UNSAVE
49          TST     (SP)+           ; pop off stack for ASTX$S
50          ASTX$S
51
52          .END
```

```
 1
 2  ; FILENAME      PARSE.MAC
 3  ;
 4  ;       This routine parese the command string by using CSI$ specific
 5  ;       riutines. It parses the input command line and stores the
 6  ;       return values in CSI control block, which may be directly used
 7  ;       by File open routines.
 8  ;       Following is format of command string:
 9  ;         dev:[g,m]filespec
10  ;       To parse the command string , it has three major function:
11  ;       i) Allocate CST control block
12  ;       ii) Syntax validation of command
13  ;       iii) Semantic check
14  ;
15  ; INPUT:
16  ;       parse(buf,len)
17  ;       char *buf;
18  ;       int len;
19  ;
20  ;OUTPUT:
21  ;       0 -- successful comletion
22  ;            relevant information in CSI control block
23  ;       1 -- unsuccess .
24  ;
25
26
27
28
29          .TITLE  PARSE
30          .IDENT  /01/
31  C$SPRT=0
32          .PSECT C$DATA,D,RW
33          .MCALL  CSI$,CSI$1,CSI$2
34
35          CSI$    DEF$G           ; define CSI control block offsets
36                                  ; and bit values globaaly.
37          .EVEN
38  CSI.BL::
39          .BLKB   C.SIZE          ; allocate required storage
40
41  CSIBLK::
42          .BLKW                   ; to access the CSI control block in 'C'.
43          .PSECT  C$TEXT,I,RO
44  PARSE::                         ; global refernce label
45          .IF DF C$SPRT
46
47          JSR     R5,C$SAV        ; make it 'C' callable
48          MOV     R5,-(SP)        ; save C frame pointer
49          MOV     4(R5),R2        ; get address of command string
50          MOV     6(R5),R3        ; length of commnad string
51
52          .ENDC
53
54          MOV     #CSI.BL,CSIBLK  ; store the address of CSI contorl block
55                                  ; for accessing in 'C' code.
56          CSI$1   #CSI.BL,R2,R3
```

```
57          BCS     ERR             ; check for success
58          CSI$2   #CSI.BL,OUTPUT
59          BCS     ERR             ; check for success
60          CLR     R0              ; yes , return success.
61          JMP     RTN
62  ERR:
63          MOV     #-1,R0          ; error code.
64  RTN:
65          .IF DF C$SPRT
66
67          MOV     (SP)+,R5        ; adjust frame pointer
68          JMP     C$RET           ; return to caller
69
70          .IFF
71
72          RETURN
73
74          .ENDC
75
76          .PSECT C$DATA,D,RW
77          .EVEN
78          .PSECT  C$TEXT,I,RO
79          .EVEN
80          .END
```

```
;
; FILENAME:     RADIX.MAC
;

; RADIX: -->    Converts array of 6 char into 2 word radix 50 format
;


        .TITLE  RADIX
        .IDENT  /01/
C$SPRT=0
        .PSECT C$DATA,D,RW
TMP:
        .WORD 0
        .PSECT  C$TEXT,I,RO
RADIX::                         ; global refernce label
        .IF DF C$SPRT

        JSR     R5,C$SAV        ; make it 'C' callable
        MOV     R5,-(SP)        ; save C frame pointer
        MOV     4(R5),R0        ; address of first char

        .ENDC

        MOV     #1,R1           ; '.' is a valid ascii char for conversion
        CALL    $CAT5B          ; convert 3 ascii char to radix 50(consider ' ')
        BCS     FAIL            ; check for success
        MOV     R1,TMP          ; save converted value
        MOV     #1,R1           ; '.' is a valid char(consider ' ' too as valid)
        CALL    $CAT5B          ; convert next 3 ascii char to radix 50
        BCS     FAIL            ; check for success
;       MOV     R1,R0           ; return value
        MOV     TMP,R0          ; return value
        JMP     RTN
FAIL:
        CLR     R0
        CLR     R1
RTN:
        .IF DF C$SPRT

        MOV     (SP)+,R5        ; adjust frame pointer
        JMP     C$RET           ; return to caller

        .IFF

        RETURN

        .ENDC

        .PSECT C$DATA,D,RW
        .EVEN
        .PSECT  C$TEXT,I,RO
        .EVEN

; c5TA :
;       Converts 16bit rad50 value to ascii string
```

```
57  ;
58  ;INPUT
59  ;       p1 = address of ascii string to be stored
60  ;       p2 = 16 bit rad50 value
61  ;
62          .psect c$text,i,ro
63
64  C5TA::
65          JSR     R5,C$SAV                ; save registers
66          MOV     4(R5),R0                ; get address of ascii string
67          MOV     6(R5),R1                ; get 16 bit rad50 value
68          CALL    $C5TA                   ; call system lib routine to convert
69                                          ; 16 bit rad50 value to ascii str.
70          JMP     C$RET                   ; unsave registers & return
71
72          .psect c$text,i,ro
73          .even
74
75          .END
```

```
;
; FILENAME:     XSETJMP.MAC
;
;               Setjmp & longjmp are only callabe from 'C'
;               Floating point register's are not saved.
;
; Environment:
;               |-------|
;               |  PC   |
;               |-------|
;               |Old FP |
;               |-------|
;               |  R4   |
;               |-------|
;               |  R3   |
;               |-------|
;               |  R2   |
;               |-------|
;               |       |    Scratch cell used by 'C' compiler
;               |-------|
;               |  FP   |
;               |-------|
;


        .TITLE  XSETJMP
        .IDENT  /01/


        .PSECT  C$TEXT,I,RO
SETJMP::                        ; global reference label

        JSR     R5,C$SAV        ; make it 'C' callable
        MOV     R5,-(SP)        ; save C frame pointer
        MOV     #0,-(SP)        ; PUSH DUMMY PARM.
        MOV     #16.,-(SP)      ; PUSH # OF BYTES TO ALLOCATE
        CALL    XMALLOC         ; 'C' RUN-TIME ALLOC ROUTINE
                                ; ALLOCATE 'n' BYTES
        TST     (SP)+           ;
        TST     (SP)+           ; POP THE PARAMETER

        TST     R0              ; CHECK RETURN VALUE
        BEQ     20$             ; IF EQ ALLOCATION FAILURE

        MOV     R0,@4(R5)       ; STORE ADDRESS OF ENV.
        MOV     #7,R3           ; WORD COUNT FOR LOOP
        MOV     R5,R2           ; GET ADDRESS OF FP
        ADD     #4,R2           ; GET HIGH ADDRESS OF CURRENT ENV. WHICH
                                ; IS TO BE SAVE
10$:
        MOV     -(R2),(R0)+
        SOB     R3,10$          ; LOOP

        MOV     #0,R0           ; SUCCESSFUL VALUE
        BR      30$             ; JUMP TO RTN.
20$:
```

```
57          MOV     #-1,R0          ; UNSUCCESSFUL VALUE
58  30$:
59          MOV     (SP)+,R5        ; adjust frame pointer
60          JMP     C$RET           ; return to caller
61
62
63          .PSECT  C$TEXT,I,RO
64
65  XLONGJMP::
66
67          JSR     R5,C$SAV        ; make it 'C' callable
68          MOV     R5,-(SP)        ; save C frame pointer
69
70          TST     @4(R5)          ; CHECK ADDRESS OF ENV.
71          BEQ     30$             ; IF EQ ADDRESS IS NULL
72          MOV     @4(R5),R1       ; GET ADDRESS OF ENV. IN REG 1
73          MOV     #7,R3           ; WORD COUNT
74  20$:
75          MOV     (R1)+,-(SP)     ; PUSH ENV. FROM HEAP TO STACK
76          SOB     R3,20$          ; LOOP FOR 7, TIMES.
77          MOV     SP,R2           ; REMEMBER THE ADDRESS OF SAVED ENV.
78          MOV     #0,-(SP)        ; DUMMY PARM.
79          MOV     @4(R5),-(SP)    ; PUSH ADDRESS OF ENV.
80          CALL    XFREE           ; 'C' RUN-TIME ROUTINE TO DEALLOCATE
81                                  ; ROUTINE
82          TST     (SP)+           ;
83          TST     (SP)+           ; POP THE PARAMETER
84
85          CLR     @4(R5)          ; CLEAR THE POINTER
86          MOV     (R2),R5         ; LOAD THE FRAME POINTER CORRESSPOND TO
87                                  ; SETJMP.
88          MOV     R5,R1           ;
89          SUB     #12,R1          ; LOAD THE LOW ADDRESS OF ENV.
90          MOV     #7,R3           ; #WORD COUNT
91  25$:
92          MOV     (R2)+,(R1)+     ;
93          SOB     R3,25$          ; LOOP
94
95          MOV     #1,R0           ; RETURN VALUE
96          BR      40$
97  30$:
98          MOV     #-1,R0          ; UNSUCCESSFUL RETURN VALUE
99  40$:
100         MOV     (SP)+,R5
101         JMP     C$RET
102
103         .PSECT  C$TEXT,I,RO
104         .EVEN
105
106         .END
```

```
 1          .title  xspawn
 2          .psect  c$text,i,ro
 3
 4          .MCALL  CLEF$S,SETF$S
 5  XSPAWN::
 6          jsr     R5,c$sav
 7          CLEF$S  #50.            ; clear efn 50
 8          SETF$S  #51.            ; set efn 51 to unstop MST
 9          jmp     c$ret
10
11          .END
```

```
 1          .title  xttywrite
 2  ;
 3  ;       xttywrite(sysid, buf, len)
 4  ;       sysid   - address of RSX - control block
 5  ;       buf     - buffer address 6(r5)
 6  ;       len     - buffer length  10(r5)
 7  ;
 8          .mcall  qiow$,dir$
 9          .psect  c$data,d,rw
10  TTYEFN  = 1
11  C$SPRT  = 0
12  ;
13  ;       define offsets used for variables
14  ;
15  .RCB    = 4
16  RLUN    = 2
17  FLAGS   = 4
18  DBLBUF  = 40
19  BUF     = 6
20  LEN     = 10
21  BUFSIZE = 512.
22  LOCBUF  = -6 - BUFSIZE
23
24  iosb:   .blkw   2.
25  count:  .word   0
26  extra:  .word   0
27  iocal:  qiow$   IO.WVB,0,TTYEFN,,iosb
28          .psect  c$text,i,ro
29  xttywrite::
30
31          .IF DF C$SPRT
32          jsr     r5,c$sav        ; make it 'C' callable
33          add     #LOCBUF, sp     ; allocate local space on stack
34          .ENDC
35          mov     .RCB(r5),r0             ; get address of RCB
36          mov     RLUN(r0), iocal+Q.IOLU  ; save LUN
37          tst     ttyraw          ; Is this write using raw mode
38          beq     10$             ; no-
39  ;
40  ;       raw mode I/O
41  ;
42          mov     BUF(r5), iocal+Q.IOPL   ; buffer address
43          mov     LEN(r5), iocal+Q.IOPL+2 ; buffer length
44          dir$    #iocal          ; directive call
45          jsr     pc, ret         ; handle return value
46          br      999$            ; return
47  ;
48  ;       Non-raw mode I/O
49  ;
50  10$:
51          mov     r5, Q.IOPL+iocal
52          add     #LOCBUF, Q.IOPL+iocal   ; set up output buffer address
53          clr     count           ; clear byte count
54          clr     extra           ; clear extra byte count
55          mov     LEN(r5), r1     ; r1 - number of bytes to output
56          mov     r5, r2          ; r2 - end of local buffer
```

```
    57          add     #-6, r2         ; adjust offset for buffer size
    58          mov     BUF(r5), r3     ; r3 - pointer to input buffer
    59          mov     r5 ,r4          ; r4 - pointer to local buffer
    60          add     #LOCBUF, r4     ; adjust for offset
    61  15$:
    62          tst     r1              ; any more bytes to output
    63          beq     50$             ; no
    64          dec     r1              ; decrement output count
    65          cmp     r4, r2          ; check if end of local buffer reached?
    66          bne     20$             ; not yet
    67  ;
    68  ;       reached end of local buffer
    69  ;
    70          mov     #BUFSIZE, Q.IOPL+2+iocal ; output buffer size
    71          dir$    #iocal          ; output the buffer
    72          jsr     pc,ret
    73          add     r0,count        ; update output count
    74          mov     r5, r4          ; reset pointer to local buffer
    75          add     #LOCBUF, r4
    76  20$:
    77  ;
    78  ;       stuff character into output buffer
    79  ;
    80  ;       mov     .RCB(r5),r0     ; get RCB address
    81  ;       bit     #DBLBUF,FLAGS(r0); is buffer from network
    82  ;       bne     30$             ; if NE yes
    83          cmpb    (r3),#12        ; is input a \n
    84          bne     30$             ; no-
    85          movb    (r3)+, (r4)+    ; put LF CR into output buffer
    86          movb    #15, (r4)+
    87          inc     extra           ; increment extra character count
    88          br      15$             ; try next character
    89  30$:
    90  ;
    91  ;       regular character
    92  ;
    93          movb    (r3)+, (r4)+
    94          br      15$
    95  50$:
    96  ;
    97  ;       finish with all the output processing, flush last buffer
    98  ;
    99          sub     #LOCBUF, r4             ; calculate buffer size
   100          sub     r5, r4
   101          mov     r4, Q.IOPL+2+iocal      ; set up the buffer size
   102          dir$    #iocal
   103          jsr     pc,ret                  ; process return value
   104          add     r0, count
   105          mov     LEN(r5), r0
   106          add     extra, r0
   107          cmp     count, r0               ; are all characters sent out?
   108          bne     60$                     ; no-
   109          mov     LEN(r5), count          ; yes, return original length of buffer
   110  60$:
   111          mov     count, r0               ; return count
   112  999$:
```

```
113          .IF     DF C$SPRT
114          jmp     c$ret
115          .IFF
116          RETURN
117          .ENDC
118
119  ret:
120  ;
121  ;       Handle return value from QIO call
122  ;
123          bcs     100$            ; error
124          mov     iosb+2, r0      ; no, return number of bytes read
125          rts     pc
126  100$:
127          mov     $dsw, r0        ; error, return DSW
128          rts     pc
129
130          .END
```

```
 1  /*
 2  @(#)arp.h       1.1 3/26/85
 3
 4  Include files for the ARP program on RSX.
 5  */
 6
 7  #include <xgenlib.h>
 8  #include <xspecial.h>
 9  #include <in.h>
10  #include <socket.h>
11  #include <brdioctl.h>
12  #include <exiocmd.h>
13  #include <types.h>
14  #include <netdb.h>
15  #include <hostarp.h>
16  #define ether_aton      etr_aton
17  #define ether_print     etr_print
```

```
/*
 * filename: BRDIOCTL.H
 */

/*
 * This file defines all the equate symbol for the administrative
 * device's ioctl commands. Some of them are passed as it is to the
 * board, hence should not be modified.
 */


#define BRDINIT          (0)          /* Reset EXOS  devive   */
#define BRDSTART         (1)          /* start exos running   */
#define BRDGSTAT         (5)          /* get board statistics */
#define BRDRSSTAT        (6)          /* get/reset board statistics*/
#define BRDGCONF         (7)          /* get configuration msg   */
#define BRDADDR          (10)         /* set exos memory locator */

#define BRDSARP          (20)         /* set an ARP table entry  */
#define BRDGARP          (21)         /* get an ARP table entry  */
#define BRDDARP          (22)         /* delete an ARP tbl entry */

#define BRDADDRT         (23)         /* add routing table entry */
#define BRDDELRT         (24)         /* delete RT entry      */
#define BRDSHOWRT        (25)         /* show   RT entry      */
#define BRDDISPRT        (26)         /* display RT entry     */


/* Data structure used to send board statistics to host */

struct EXbdstats {
        long    xmt;            /* frames transmitted successfully */
        long    excess_coll;    /* xmits aborted due to excess coll */
        long    late_coll;      /* xmits aborted due to late coll */
        long    tdr;            /* time domain reflectometer */
        long    rcv;            /* error free frames received */
        long    align_err;      /* frames rcvd with alignment err */
        long    crc_err;        /* frames rcvd with crc errors */
        long    lost_err;       /* frames lost due to no buffers */

        /* other bits of info about the board */

        short   fw_release;     /* firmware release */
        short   sw_release;     /* software release */
        short   hw_release;     /* hardware release */
};

/*
 * Ioctl structure for manipulation of the ARP codes
 */

struct EXarp_ioctl {
  struct sockaddr  arp_pa;      /* protocol address     */
  struct sockaddr  arp_ha;      /* hardware address     */
  long             arp_flags;   /* flags                */
  };
```

```
57
58  #define ATF_COM         2       /* completed entry      */
59  #define ATF_PERM        4       /* permanant entry      */
60  #define ATF_PUBL        8       /* respond for another host */
61
```

```
 1  /*
 2   * filename: EXIOCMD.H
 3   */
 4
 5  /*
 6   * following are the requests send to the board
 7   * - host to board request must be less than 64 ;
 8   *   flags takes up upper two bits.
 9   */
10
11  #define SOSOCKET          (50)
12  #define SOACCEPT          (51)
13  #define SOCONNECT         (52)
14  #define SOSEND            (53)
15  #define SORECEIVE         (54)
16  #define SOSKTADDR         (55)
17  #define SOCLOSE           (56)
18  #define SOVERIFY          (57)
19  #define SOIOCTL           (58)
20  #define SOSELECT          (59)
21
22
23  #define NET_DLOAD         0               /* net download  */
24  #define NET_ULOAD         1               /* net upload          */
25  #define NET_START         2               /* start downloaded stuff*/
26
27  #define NET_GSTAT         BRDGSTAT        /* read net statistics  */
28  #define NET_RSTAT         BRDRSSTAT       /* read & reset stats   */
29
30  #define NET_GCONF         BRDGCONF        /* get configuration msg*/
31
32  #define NET_SARP          BRDSARP         /* set ARP       */
33  #define NET_GARP          BRDGARP         /* get ARP       */
34  #define NET_DARP          BRDDARP         /* delete ARP    */
35
36  #define NET_ADDRT         BRDADDRT        /* add RT entry  */
37  #define NET_DELRT         BRDDELRT        /* delete RT entry */
38  #define NET_SHOWRT        BRDSHOWRT       /* show RT       */
39  #define NET_DISPRT        BRDDISPRT       /* display RT    */
40
41
42  /*  unsolicited messages from board */
43
44  #define SOSELWAKEUP       (80)
45  #define SOHASOOB          (81)
46  #define NET_PRINTF        100             /* print out msg */
47  #define NET_PANIC         101             /* oh-my-gosh    */
48  #define IM_ALIVE          102             /* I think therfore I am*/
49
50
51  #define REPLY_OK          0x00            /* all is well   */
52
53
54  #define NM_MAGIC_DATA     0x80
55
56  #define MQ_EXOS           0x01            /* exos own Q element */
```

```
57  #define MQ_DONE          0x02            /* exos done with Q elmnt*/
58  #define MQ_OVERFLOW      0x04            /* data are too big  */
59
60
```

```
 1
 2  /*
 3   * These are the DIC and DPB lengths of the Executive directives
 4   */
 5  # define QIO  06001
 6  # define QIOW 06003
 7  # define ALUN 02007
 8  # define WTSE 01051
 9  # define GTIM 01075
10  # define SPWN 06413
11  # define SDRC 03615
12  # define SDAT 02507
13  # define STOP 0603
14  # define RCVD 02113
15  # define MRKT 02427
16
17  /*  Executive return status */
18
19  # define IE_BAD  -01                    /* bad parameters    */
20  # define IE_IFC  -02                    /* illegal function  */
21  # define IE_DNR  -03                    /* device not ready     */
22  # define IE_SPC  -06                    /* illegal bufferr      */
23  # define IE_ABO  -15                    /* request aborted      */
24  # define IE_PRI  -16                    /* priv or channel error*/
25  # define IE_DFU  -24                    /* no free channel      */
26  # define IE_FHE  -59                    /* fatal hardware error */
27  # define IE_OFL  -65                    /* device offline       */
28
29  /*
30   * These are the function codes related to the QIO call to the ZE device
31   */
32
33  /*
34   * following five codes are already defined in standard rsx header file
35   * rsx.h and are not defined here only shown under comment for clarity
36
37   define IO_KIL    000012            # kill all outstanding request #
38   define IO_WLB    000400            # write to the EXOS memory     #
39   define IO_RLB    001000            # read from the EXOS memory    #
40   define IO_ATT    001400            # attach fn: made no-op        #
41   define IO_DET    002000            # detach fn: made no-op        #
42
43   */
44
45  #define IO_EXC    002400            /* EXOS board admn. operation */
46  #define    EX_INI       BRDINIT         /* Reset and configure EXOS   */
47  #define    EX_CNF       BRDGCONF                /* get configuration msg      */
48  #define    EX_STR       BRDSTART        /* Execute EXOS procedure     */
49  #define    EX_STS       BRDGSTAT        /* Read network statistics    */
50  #define    EX_SAR       BRDSARP         /* set up an ARP table entry  */
51  #define    EX_GAR       BRDGARP         /* Retrive an ARP table entry */
52  #define    EX_DAR       BRDDARP         /* Delete an ARP table entry  */
53  #define    EX_ART       BRDADDRT        /* Add an Routing table entry */
54  #define    EX_DRT       BRDDELRT        /* Delete an RT entry         */
55  #define    EX_SRT       BRDSHOWRT       /* Fetch an RT entry       */
56  #define    EX_NRT       BRDDISPRT       /* Fetch next RT entry       */
```

```
57  #define     EX_RST        BRDRSSTAT        /* Read & Reset network stats */
58  #define     EX_OPN        0020             /* Open an admin channel      */
59  #define     EX_CLS        0021             /* Close an admin channel     */
60  #define     EX_POS        BRDADDR          /* Seek EXOS's memory         */
61
62  #define IO_ACS    003000              /* Socket access operations   */
63  #define     SA_OPN        50               /* Open a socket              */
64  #define     SA_ACC        51               /* Accept a remote socket     */
65  #define     SA_CON        52               /* Connect to a remote socket */
66  #define     SA_SAD        55               /* get socket informations    */
67  #define     SA_CLS        56               /* close an opened socket     */
68  #define     SA_SEL        59               /* perform select op on socket*/
69  #define     SA_USL        0210             /* kill the outstanding select call */
70  #define     SA_URG        0200             /* prepare for urgent msg     */
71  #define     SA_ROO        0220             /* remove oob pkt from pending list */
72
73  #define IO_XFR    003400              /* data transfer operation    */
74  #define     IX_RDS        0000             /* read from TCP stream       */
75  #define     IX_WRS        0001             /* write to TCP stream        */
76  #define     IX_SND        53               /* send datagram to a socket  */
77  #define     IX_RCV        54               /* receive socket datagram    */
78
79  #define IO_SOC    004000              /* socket control operations  */
80  #define     SO_DON        SIOCDONE         /* shutdowm r/w on socket     */
81  #define     SO_SKP        SIOCSKEEP        /* set keep alive             */
82  #define     SO_GKP        SIOCGKEEP        /* inspect keep alive         */
83  #define     SO_SLG        SIOCSLINGER      /* set linger time            */
84  #define     SO_GLG        SIOCGLINGER      /* get linger time            */
85  #define     SO_SOB        SIOCSENDOOB      /* send out of band           */
86  #define     SO_ROB        SIOCRCVOOB       /* receive out of bound       */
87  #define     SO_AMK        SIOCATMARK       /* at oob mark ?              */
88  #define     SO_SPG        SIOCSPGRP        /* set process group          */
89  #define     SO_GPG        SIOCGPGRP        /* get process group          */
90  #define     SO_NRD        FIONREAD         /* FIONREAD                   */
91  #define     SO_NBO        FIONBIO          /* FIONBIO                    */
92  #define     SO_ASY        FIOASYNC         /* FIOASYNC                   */
93
94  #define IO_LOG          004400           /* read error msg from EXOS */
95  #define IO_TEL          0177000          /* telnet server pseudo fn code */
96  #define TS_HNG          0176000          /* hangup carrier pseudo fn code*/
97
98  /*
99   * All the Socket related parameters in the QIO call are passed
100  * throgh the structure "SOictl" defined below.
101  */
102
103 struct  SOictl  {
104         short   hassa;                   /* non-zero if sa specified   */
105         struct  sockaddr sa;             /* socket address (optional)   */
106         short   hassp;                   /* non-zero if sp specified       */
107         struct  sockproto sp;            /* socket protocol (optional)   */
108         int     type;                    /* socket type      */
109         int     options;                 /* options          */
110         /* these are for select ()              */
111         int     nfd;
112         long    *wp;
```

```
113             long    *rp;
114             long    timo;
115             };
116
117
118
```

```
1  /* unsigned data types (shorthand) */
2  typedef unsigned int    Uint;
3  typedef long            Ulong;
4  typedef unsigned short  Ushort;
5  typedef char            Uchar;
```

```
 1  /* RSX CONTROL - BLOCK  */
 2
 3  struct _rcb {
 4          int     mode;                   /* File type      */
 5          int     rlun;                   /* RSX LUN        */
 6          int     flags;                  /* Flags -- described below    */
 7          char    *bptr;                  /* pointer to block buffer */
 8          char    *bnptr;                 /* next position in block */
 9          int     bleft;                  /* bytes left in block read/write*/
10          char    *rptr;                  /* poiter to record buffer */
11          char    *rnptr;                 /* next position in record */
12          char    *fdb;                   /* pointer to FDB  */
13          union {
14                  int     rleft;          /* char left to be read  */
15                  int     rsize;          /* max. record size  */
16          } rec;
17  };
18
19  struct dblbuf {
20          int     stat[2];                /* status of i/o buffer  */
21                                          /*    = 0 -- EOF  */
22                                          /*    < 0 -- I/O Error  */
23                                          /*    > 0 -- Bytes transferred  */
24          char    *buffer[2];
25          XFILE   *fd;                    /* pointer to file descriptor  */
26          char    active;                 /* buffer used for IO  */
27          };
28
29  struct iosb {
30          unsigned char cc;
31          unsigned char lc;
32          int  nread;
33          };
34
35  #define RFREE   01
36  #define RUSED   02
37  #define REOF    04
38  #define REOLN   010
39  #define DBLBUF  040
40  #define RCRFLAG 020
41  #define SOEFN   11
42  #define DISKEFN  12
43  #define FDBSIZE 0140
44  #define F_FFBY  014
45  #define F_RSIZ  02
46  #define F_BKVB  064
47  #define RECSIZE 256
48  #define BLKSIZE 512
49  #define MAXPRM  20
50  #define C_DSDS  06
51  #define IO_XFR  003400                  /* data transfer stream  */
52  #define IX_RDS    0000                  /* read from TCP stream  */
53  #define IX_WRS    0001                  /* write to TCP stream  */
54  #define SOLUN   20              /* EXOSO LUN    */
```

```
 1  /* @(#)ftp.h    1.2 1/14/85 */
 2  /*
 3   * Definitions for FTP
 4   * See RFC-765
 5   */
 6  #ifndef MAXPATHLEN
 7  #define MAXPATHLEN 1024
 8  #endif
 9  #ifndef CTRL
10  #define CTRL(x) 037&x
11  #endif
12  #ifndef SIGCHLD
13  #define SIGCHLD SIGCLD
14  #endif
15
16  /*
17   * Reply codes.
18   */
19  #define PRELIM          1       /* positive preliminary */
20  #define COMPLETE        2       /* positive completion */
21  #define CONTINUE        3       /* positive intermediate */
22  #define TRANSIENT       4       /* transient negative completion */
23  #define ERROR           5       /* permanent negative completion */
24
25  /*
26   * Type codes
27   */
28  #define TYPE_A          1       /* ASCII */
29  #define TYPE_E          2       /* EBCDIC */
30  #define TYPE_I          3       /* image */
31  #define TYPE_L          4       /* local byte size */
32
33  /*
34   * Form codes
35   */
36  #define FORM_N          1       /* non-print */
37  #define FORM_T          2       /* telnet format effectors */
38  #define FORM_C          3       /* carriage control (ASA) */
39
40  /*
41   * Structure codes
42   */
43  #define STRU_F          1       /* file (no record structure) */
44  #define STRU_R          2       /* record structure */
45  #define STRU_P          3       /* page structure */
46
47  /*
48   * Mode types
49   */
50  #define MODE_S          1       /* stream */
51  #define MODE_B          2       /* block */
52  #define MODE_C          3       /* compressed */
53
54  /*
55   * Record Tokens
56   */
```

```
57  #define REC_ESC         '\377'  /* Record-mode Escape */
58  #define REC_EOR         '\001'  /* Record-mode End-of-Record */
59  #define REC_EOF         '\002'  /* Record-mode End-of-File */
60
61  /*
62   * Block Header
63   */
64  #define BLK_EOR         0x80    /* Block is End-of-Record */
65  #define BLK_EOF         0x40    /* Block is End-of-File */
66  #define BLK_ERRORS      0x20    /* Block is suspected of containing errors */
67  #define BLK_RESTART     0x10    /* Block is Restart Marker */
68
69  #define BLK_BYTECOUNT   2       /* Bytes in this block */
```

```
/*@(#)ftp_var.h 1.12 6/13/85*/

/*
 * FTP global variables.
 */
#include "varpat.h"

/*
 * Options and other state info.
 */
extern int      trace;          /* trace packets exchanged */
extern int      hash;           /* print # for each buffer transferred */
extern int      sendport;       /* use PORT cmd for each data connection */
extern int      verbose;        /* print messages coming back from server */
extern int      connected;      /* connected to server */
extern int      fromatty;       /* input is from a terminal */
extern int      interactive;    /* interactively prompt on m* cmds */
extern int      debug;          /* debugging level */
extern int      bell;           /* ring bell on cmd completion */
extern int      doglob;         /* glob local file names */
extern int      autologin;      /* establish user account on connection */

extern char     typename[];     /* name of file transfer type */
extern int      type;           /* file transfer type */
extern char     structname[];   /* name of file transfer structure */
extern int      stru;           /* file transfer structure */
extern char     formname[];     /* name of file transfer format */
extern int      form;           /* file transfer format */
extern char     modename[];     /* name of file transfer mode */
extern int      mode;           /* file transfer mode */
extern char     bytename[];     /* local byte size in ascii */
extern int      bytesize;       /* local byte size in binary */

extern char     *hostname;      /* name of host connected to */

extern struct   servent *sp;    /* service spec for tcp/ftp */

#ifdef zilog
#include <setret.h>
extern ret_buf  toplevel;       /* non-local goto stuff for cmd scanner */
#else
/* #include <setjmp.h>  */
extern jmp_buf  toplevel;       /* non-local goto stuff for cmd scanner */
#endif

extern char     line[]; /* input line buffer */
extern int      margc;          /* count of arguments on input line */
extern char     **margv;        /* args parsed from input line */

extern int      options;        /* used during socket creation */

/*
 * Format of command table.
 */
struct cmd {
        char    *c_name;        /* name of command */
```

```
57          char    *c_help;        /* help string */
58          char    c_bell;         /* give bell when command completes */
59          char    c_conn;         /* must be connected to use command */
60          int     (*c_handler)(); /* function to call */
61  };
62
63  extern  char *tail();
64  extern  char *remglob();
65  extern  int errno;
```

```
 1  /* @(#)host_arp.h       1.3 5/14/85 */
 2  /*
 3  Address structures for ARP ioctl calls
 4  */
 5
 6  /*
 7  Ethernet Address
 8  */
 9
10  struct etr_addr {
11          short ea_family;                /* to match sockaddr structure */
12          char  ea_addr[6];                       /* interesting part */
13          char  ea_extra[8];              /* to match sockaddr structure */
14  };
15
16  /*
17  Count ( for retrieving entire table )
18  */
19
20  struct next_addr {
21          short nxt_family;               /* to match sockaddr structure */
22          long nxt_count;                 /* interesting part */
23          char  nxt_extra[10];            /* to match sockaddr structure */
24  };
```

```
/* @(#)in.h     1.3 4/12/85 */

/*
 * GAP 1/11/85:  W A R N I N G  - This file is included by both host
 * and board code.  Make changes with extreme caution, and test
 * effects on both the host and board sides.
 */

/*
 * Constants and structures defined by the internet system,
 * Per RFC 790, September 1981.
 */

/*
 * Protocols
 */
#define IPRO_ICMP               1               /* control message protocol */
#define IPPROTO_GGP             2               /* gateway^2 (deprecated) */
#define IPRO_TCP                6               /* tcp */
#define IPRO_PUP                12              /* pup */
#define IPRO_UDP                17              /* user datagram protocol */

#define IPRO_RAW                255             /* raw IP packet */
#define IPRO_MAX                256

/*
 * Port/socket numbers: network standard functions
 */
#define IPPORT_ECHO             7
#define IPRT_DISCARD            9
#define IPRT_SYSTAT             11
#define IPPORT_DAYTIME          13
#define IPRT_NETSTAT            15
#define IPRT_FTP                21
#define IPPORT_TELNET           23
#define IPPORT_SMTP             25
#define IPRT_TIMESERVER 37
#define IPPORT_NAMESERVER       42
#define IPPORT_WHOIS            43
#define IPPORT_MTP              57

/*
 * Port/socket numbers: host specific functions
 */
#define IPRT_TFTP               69
#define IPRT_RJE                77
#define IPPORT_FINGER           79
#define IPRT_TTYLINK            87
#define IPRT_SUPDUP             95

/*
 * UNIX TCP sockets
 */
#define IPRT_EXECSERVER 512
#define IPPORT_LOGINSERVER      513
#define IPPORT_CMDSERVER        514
```

```
 57
 58  /*
 59   * UNIX UDP sockets
 60   */
 61  #define IPPORT_BIFFUDP          512
 62  #define IPRT_WHOSERVER  513
 63
 64  /*
 65   * Ports < IPPORT_RESERVED are reserved for
 66   * privileged processes (e.g. root).
 67   */
 68  #define IPPORT_RESERVED         1024
 69
 70  /*
 71   * Link numbers
 72   */
 73  #define IMPLK_IP                155
 74  #define IMPLK_LOWEXPER  156
 75  #define IMPLINK_HIGHEXPER       158
 76
 77  /*
 78   * Internet address (old style... should be updated)
 79   */
 80  struct in_addr {
 81          union {
 82                  struct { char s_b1,s_b2,s_b3,s_b4; } S_un_b;
 83                  struct { unsigned short s_w1,s_w2; } S_un_w;
 84                  long S_addr;
 85          } S_un;
 86  #define s_addr  S_un.S_addr     /* can be used for most tcp & ip code */
 87  #define s_host  S_un.S_un_b.s_b2        /* host on imp */
 88  #define s_net   S_un.S_un_b.s_b1        /* network */
 89  #define s_imp   S_un.S_un_w.s_w2        /* imp */
 90  #define s_impno S_un.S_un_b.s_b4        /* imp # */
 91  #define s_lh    S_un.S_un_b.s_b3        /* logical host */
 92  #define S_baddr S_un.S_un_b
 93  };
 94
 95  /*
 96   * Macros for dealing with Class A/B/C network
 97   * numbers.  High 3 bits of uppermost byte indicates
 98   * how to interpret the remainder of the 32-bit
 99   * Internet address.  The macros may be used in time
100   * time critical sections of code, while subroutine
101   * versions also exist use in other places.
102   */
103  /*
104   * GAP 1/10/85:  Apparently these are designed to work on internet
105   * addresses which reside in network order in RAM, if regarded as
106   * a byte string.  Be careful, because 4.2BSD defines just one
107   * version of these macros, which works on internet addresses only
108   * after they are swapped into proper order (in a CPU register)
109   * by ntohl().
110   */
111
112  /* GAP 1/10/85: Note fancy footwork below to share header with board code */
```

```
113 #ifdef ONBOARD          /* board make does not define MACHINE type */
114 #define IN_CLASSA       0x00800000L
115 #define INCA_NET   0x00ff0000L     /* 8 bits of net # */
116 #define INCA_LNA   0xff00ffffL
117 #define INCB       0x00400000L
118 #define INCB_NET   0xffff0000L     /* 16 bits of net # */
119 #define INCB_LNA   0x0000ffffL
120 #define INCC_NET   0xffff00ffL     /* 24 bits of net # */
121 #define INCC_LNA   0x0000ff00L
122 #endif
123
124 #ifndef ONBOARD         /* board make does not define MACHINE type */
125
126 #ifdef VAX
127 #define IN_CLASSA       0x00000080
128 #define INCA_NET        0x000000ff      /* 8 bits of net # */
129 #define INCA_LNA        0xffffff00
130 #define INCB    0x00000040
131 #define INCB_NET        0x0000ffff      /* 16 bits of net # */
132 #define INCB_LNA        0xffff0000
133 #define INCC_NET        0x00ffffff      /* 24 bits of net # */
134 #define INCC_LNA        0xff000000
135 #endif
136 #ifdef PDP11                    /* Also 8086 XENIX V7 C */
137 #define IN_CLASSA       0x00800000L
138 #define INCA_NET   0x00ff0000L     /* 8 bits of net # */
139 #define INCA_LNA   0xff00ffffL
140 #define INCB       0x00400000L
141 #define INCB_NET   0xffff0000L     /* 16 bits of net # */
142 #define INCB_LNA   0x0000ffffL
143 #define INCC_NET   0xffff00ffL     /* 24 bits of net # */
144 #define INCC_LNA   0x0000ff00L
145 #endif
146 #ifdef I8086                    /* XENIX 3.0, Lattice C */
147 #define IN_CLASSA       0x00000080
148 #define INCA_NET        0x000000ff      /* 8 bits of net # */
149 #define INCA_LNA        0xffffff00
150 #define INCB    0x00000040
151 #define INCB_NET        0x0000ffff      /* 16 bits of net # */
152 #define INCB_LNA        0xffff0000
153 #define INCC_NET        0x00ffffff      /* 24 bits of net # */
154 #define INCC_LNA        0xff000000
155 #endif
156 #ifdef M68000
157 #define IN_CLASSA       0x80000000L
158 #define INCA_NET   0xff000000L     /* 8 bits of net # */
159 #define INCA_LNA   0x00ffffffL
160 #define INCB       0x40000000L
161 #define INCB_NET   0xffff0000L     /* 16 bits of net # */
162 #define INCB_LNA   0x0000ffffL
163 #define INCC_NET   0xffffff00L     /* 24 bits of net # */
164 #define INCC_LNA   0x000000ffL
165 #endif
166 #ifdef Z8000
167 #define IN_CLASSA       0x80000000L
168 #define INCA_NET   0xff000000L     /* 8 bits of net # */
```

```
169 #define INCA_LNA    0x00ffffffL
170 #define INCB        0x40000000L
171 #define INCB_NET    0xffff0000L     /* 16 bits of net # */
172 #define INCB_LNA    0x0000ffffL
173 #define INCC_NET    0xffffff00L     /* 24 bits of net # */
174 #define INCC_LNA    0x000000ffL
175 #endif
176
177 #endif ONBOARD          /* board make does not define MACHINE type */
178
179 #define IN_NETOF(in) \
180         (((in).s_addr&IN_CLASSA) == 0 ? (in).s_addr&INCA_NET : \
181                 ((in).s_addr&INCB) == 0 ? (in).s_addr&INCB_NET : \
182                         (in).s_addr&INCC_NET)
183 #define IN_LNAOF(in) \
184         (((in).s_addr&IN_CLASSA) == 0 ? (in).s_addr&INCA_LNA : \
185                 ((in).s_addr&INCB) == 0 ? (in).s_addr&INCB_LNA : \
186                         (in).s_addr&INCC_LNA)
187
188 #define INADDR_ANY      0x00000000
189
190 /*
191  * Socket address, internet style.
192  */
193 struct sckadr_in {
194         short   sin_family;
195         unsigned short  sin_port;
196         struct  in_addr sin_addr;
197         char    sin_zero[8];
198 };
199
200 #ifdef KERNEL
201 long in_netof(),in_lnaof();
202 #endif
```

```
 1  /*
 2   * filename:    INIT.H
 3   */
 4
 5  /*
 6   * Structure used for initialization only.
 7   */
 8
 9  /* some of the dummy entries are due to byte swapping */
10
11  struct  init_msg {
12          short   im_newstyle;            /* new style init msg?    */
13          char    im_version[4];          /* version to the hardware   */
14          char    im_result;              /* completion code    */
15          char    im_mode;                /* set to link moce (0)    */
16          char    im_hdfo[2];             /* host data format option   */
17          char    im_junk[3];
18          char    im_addrmode;            /* host address mode    */
19          char    im_dummy2;
20          char    im_mmsize;              /* memory map size (returned)   */
21          char    im_byteptn[4];          /* data order byte pattern   */
22          Ushort  im_wordptn[2];          /* data order word pattern   */
23          long    im_longptn;             /* data order long pattern   */
24          char    im_mmap[20];            /* (rest of) memory map (returned)*/
25
26          short   im_101off;              /* movable block offset    */
27          short   im_101seg;              /* movable block segment   */
28          char    im_nproc;               /* number of exos 101 processes   */
29          char    im_nmb;                 /* number of exos 101 mailboxes   */
30          char    im_nslots;              /* number of address slots   */
31          char    im_nhosts;              /* number of hosts == 1    */
32
33  /* "host to exos" stuff */
34
35          long    im_h2exqaddr;           /* host to exos msg a address   */
36          short   im_h2exoff;             /* offset from base of actual q   */
37          char    im_h2extype;            /* interrupt type for h2ex msg q  */
38          char    im_h2exvalue;           /* interrupt value     */
39          long    im_h2exaddr;            /* interrupt address     */
40
41  /* "exos to host" stuff */
42
43          long    im_ex2hqaddr;           /* exos to host msg q address   */
44          short   im_ex2hoff;             /* offset from base of actual q   */
45          char    im_ex2htype;            /* interrupt type for ex2h msg q  */
46          char    im_ex2value;            /* interrupt value    */
47          long    im_ex2haddr;            /* interrupt address    */
48  };
49
50  /* im_mode */
51
52  # define  EXOS_LINKMODE 0
53  # define  EXOS_HOSTLOAD 1
54  # define  EXOS_NETLOAD  2
```

```
 1  /*  @(#)netdb.h 1.3 3/25/85 */
 2  /*
 3   * Structures returned by network
 4   * data base library.  All addresses
 5   * are supplied in host order, and
 6   * returned in network order (suitable
 7   * for use in system calls).
 8   */
 9  struct  hostent {
10          char    *h_name;        /* official name of host */
11          char    **h_aliases;    /* alias list */
12          int     h_addrtype;     /* host address type */
13          int     h_length;       /* length of address */
14          char    *h_addr;        /* address */
15  };
16
17  /*
18   * Assumption here is that a network number
19   * fits in 32 bits -- probably a poor one.
20   */
21  struct  netent {
22          char    *n_name;        /* official name of net */
23          char    **n_aliases;    /* alias list */
24          int     n_addrtype;     /* net address type */
25          long    n_net;          /* network # */
26  };
27
28  struct  servent {
29          char    *s_name;        /* official service name */
30          char    **s_aliases;    /* alias list */
31          unsigned short  s_port;         /* port # */
32          char    *s_proto;       /* protocol to use */
33  };
34
35  struct  protoent {
36          char    *p_name;        /* official protocol name */
37          char    **p_aliases;    /* alias list */
38          int     p_proto;        /* protocol # */
39  };
40
41
42  #define ghbname gethbname
43  #define ghbaddr gethbaddr
44  #define gethostent      gethent
45  #define sethostent      sethent
46  #define endhostent      endhent
47
48  #define gnbname getnbname
49  #define gnbaddr getnbaddr
50  #define getnetent       getnent
51  #define setnetent       setnent
52  #define endnetent       endnent
53
54  #define gsbname getsbname
55  #define gsbport getsbport
56  #define getservent      getsent
```

```
57  #define setservent      setsent
58  #define endservent      endsent
59
60  #define getpbname gpbname
61  #define getpbnumber gpbnumber
62  #define getprotoent     getpent
63  #define setprotoent     setpent
64  #define endprotoent     endpent
65
66  #define inet_lnaof      inetgln
67  #define inet_netof      inetgnet
68
69
70  struct hostent  *ghbname(), *ghbaddr(), *gethostent();
71  struct netent   *gnbname(), *gnbaddr(), *getnetent();
72  struct servent  *gsbname(), *gsbport(), *getservent();
73  struct protoent *gpbname(), *gpbnumber(), *getprotoent();
```

```
 1  /* @(#)route.h  1.6 5/7/85 */
 2
 3  /*
 4   * GAP 1/11/85:  W A R N I N G  - This file is included by both host
 5   * and board code.  Make changes with extreme caution, and test
 6   * effects on both the host and board sides.
 7   */
 8
 9  /*
10   * Kernel resident routing tables.
11   *
12   * The routing tables are initialized at boot time by
13   * making entries for all directly connected interfaces.
14   * Routing daemons can thereafter update the routing tables.
15   *
16   * TODO:
17   *      keep statistics
18   */
19
20  /*
21   * A route consists of a destination address and a reference
22   * to a routing entry.  These are often held by protocols
23   * in their control blocks, e.g. inpcb.
24   */
25  struct route {
26          struct  rtentry *ro_rt;
27          struct  sockaddr ro_dst;
28  #ifdef notdef
29          caddr_t ro_pcb;                 /* not used yet */
30  #endif
31  };
32  #ifdef KERNEL
33  /*
34   * The route ''routetoif'' is a special atom passed to the output routines
35   * to implement the SO_DONTROUTE option.
36   */
37  struct  route routetoif;
38  #endif
39
40  /*
41   * We distinguish between routes to hosts and routes to networks,
42   * preferring the former if available.  For each route we infer
43   * the interface to use from the gateway address supplied when
44   * the route was entered.  Routes that forward packets through
45   * gateways are marked so that the output routines know to address the
46   * gateway rather than the ultimate destination.
47   *
48   * AA - 4/11/85: The rtentry structure below has been set up
49   *      so that it it compatible with the host, board
50   *      and machines such as VAX that like
51   *      to do long alignments.
52   *      !!! DO NOT FIDDLE WITH THIS STRUCTURE UNLESS YOU
53   *      !!! UNDERSTAND THIS
54   */
55  struct rtentry {
56          struct  sockaddr rt_dst;        /* key */
```

```
57          struct  sockaddr rt_gateway;    /* value */
58          struct  rtentry *rt_next;       /* next pointer */
59  #ifdef PDP11
60          short   dummy;                  /* host ptr=4; board ptr =2 */
61  #endif
62  #ifdef xenix286
63          short   dummy;                  /* host ptr=4; board ptr =2 */
64  #endif
65          u_long  rt_use;                 /* raw # packets forwarded */
66          struct  ifnet *rt_ifp;          /* the answer: interface to use */
67  #ifdef PDP11
68          short   dummyx;                 /* host ptr=4; board ptr =2 */
69  #endif
70  #ifdef xenix286
71          short   dummyx;                 /* host ptr=4; board ptr =2 */
72  #endif
73          char    rt_flags;               /* up/down?, host/net */
74          char    rt_refcnt;              /* # held references */
75          u_short rt_hash;                /* to speed lookups */
76  };
77  #define RTHASHSIZ       7
78  #ifdef  ONBOARD
79  struct  rtentry *rthost[RTHASHSIZ] = 0;
80  struct  rtentry *rtnet[RTHASHSIZ]  = 0;
81  #endif
82
83  #define RTF_UP          0x1             /* route useable */
84  #define RTF_GATEWAY     0x2             /* destination is a gateway */
85  #define RTF_HOST        0x4             /* host entry (net otherwise) */
86
87  #define RTFREE(rt) \
88          if ((rt)->rt_refcnt == 1) \
89                  rtfree(rt); \
90          else \
91                  (rt)->rt_refcnt--;
92
```

```
 1  /*
 2   * These are the DIC and DPB lengths of the Executive directives
 3   */
 4
 5  #define RSX     1
 6
 7  # define QIO  06001
 8  # define QIOW 06003
 9  # define ALUN 02007
10  # define WTSE 01051
11  # define GTIM 01075
12  # define SPWN 06413
13  # define SDRC 03615
14  # define SDAT 02507
15  # define STOP 0603
16  # define STSE 01207
17  # define RCVD 02113
18  # define RCVX 02115
19  # define RCST 02213
20  # define MRKT 02427
21  # define GTSK 01077
22  # define SREX 01647
23  # define EXST 01035
24  # define USTP 01605
25  # define SETF 01041
26  # define CLEF 01037
27  # define ENAR 0545
28  # define DSAR 0543
29  # define DSCP 0537
30  # define ENCP 0541
31  # define GLUN 01405
32  # define RQST 03413
33  /*
34   * QIO function codes
35   *
36   */
37
38  # define IO_RLB 01000
39  # define IO_RVB 010400
40  # define IO_RTT 05001
41  # define IO_WVB 011000
42  # define IO_DET 002000
43  # define IO_KIL 000012
44  # define IO_WLB 00400
45  # define IO_ATA 01410
46  # define SF_SMC 02440
47  # define SF_GMC 02560
48  # define TC_FDX 064
49  # define TC_ACR 024
50  /*  Executive return status */
51
52  # define IS_CLR   00                    /* event was clear */
53  # define IS_SUC   01                    /* operation successful */
54  # define IS_SET   02                    /* event flag was set */
55
56  # define IE_BAD  -01                    /* bad parameters    */
```

```
 57  # define IE_IFC  -02                    /* illegal function  */
 58  # define IE_DNR  -03                    /* device not ready     */
 59  # define IE_SPC  -06                    /* illegal bufferr      */
 60  # define IE_ACT  -07                    /* task not active      */
 61  # define IE_ABO  -15                    /* request aborted      */
 62  # define IE_PRI  -16                    /* priv or channel error*/
 63  # define IE_DFU  -24                    /* no free channel      */
 64  # define IE_FHE  -59                    /* fatal hardware error */
 65  # define IE_OFL  -65                    /* device offline       */
 66
 67  /* CSI CONTROL BLOCK OFFSETS AND BIT VALUES DEFINITIONS
 68   *
 69   */
 70  #define CS_DIF  02
 71  #define CS_DVF  04
 72  #define CS_EQU  040
 73  #define CS_INP  01
 74  #define CS_MOR  020
 75  #define CS_NMF  01
 76  #define CS_OUT  02
 77  #define CS_WLD  010
 78  #define C_CMLD  02
 79  #define C_DEVD  06
 80  #define C_DIRD  012
 81  #define C_DSDS  06
 82  #define C_FILD  016
 83  #define C_MKW1  024
 84  #define C_MKW2  026
 85  #define C_STAT  01
 86  #define C_SWAD  022
 87  #define C_TYPR  00
 88  #define A_GRP   00
 89  #define A_MBR   03
 90  #define A_LPRV  074
 91  #define A_SYDV  056
 92  #define TF_RAL  010
 93  /*
 94   C Portable routines.
 95  */
 96  /*
 97   MCR relative parameters
 98  */
 99  # define CMDSIZE 60
100
101  /*
102   * terminal Input buffer structure
103   */
104
105  struct ttybuf {
106          char linetty[132];
107          char *cur_pos;
108          int tsize;
109          };
110  #define rsx
111  #define SCRATCHFILE     "."
```

```
 1  /* @(#)socket.h 1.8 7/29/85 */
 2  /*      socket.h        4.16    82/06/08        */
 3
 4  /*
 5   * GAP 1/11/85:  W A R N I N G  - This file is included by both host
 6   * and board code.  Make changes with extreme caution, and test
 7   * effects on both the host and board sides.
 8   */
 9
10  #ifdef  BSD4dot2
11  #define accept          ex_accept
12  #define connect         ex_connect
13  #define gethostname     ex_gethostname
14  #define receive         ex_receive
15  #define select          ex_select
16  #define send            ex_send
17  #define socket          ex_socket
18  #define socketaddr      ex_socketaddr
19  #define shutdown        ex_shutdown
20
21  #define htonl           ex_htonl
22  #define htons           ex_htons
23  #define ntohl           ex_ntohl
24  #define ntohs           ex_ntohs
25  #define swab            ex_swab
26  #endif  BSD4dot2
27
28  /*
29   * Externally visible attributes of sockets.
30   */
31
32  /*
33   * Socket types.
34   *
35   * The kernel implement these abstract (session-layer) socket
36   * services, with extra protocol on top of network services
37   * if necessary.
38   */
39  #define SOCK_STREAM     1               /* stream socket */
40  #define SOCK_DGRAM      2               /* datagram socket */
41  #define SOCK_RAW        3               /* raw-protocol interface */
42  #define SOCK_RDM        4               /* reliably-delivered message */
43  #define SOCK_ETH        5               /* link-mode access to e-net packets */
44  #define SOCK_ICMP       6               /* access to ICMP */
45
46  /*
47   * Option flags per-socket.
48   */
49  #define SO_DEBUG        0x01            /* turn on debugging info recording */
50  #define SO_ACCEPTCONN   0x02            /* willing to accept connections */
51  #define SO_DONTLINGER   0x04            /* don't linger on close */
52  #define SO_KEEPALIVE    0x08            /* keep connections alive */
53  #define SO_DONTROUTE    0x10            /* just use interface addresses */
54  #define SO_SMALL        0x20            /* use smaller (1/2K) buffer quota */
55  #define SO_REUSEADDR    0x40            /* permit local port ID duplication */
56
```

```
 57 /*
 58  * Generic socket protocol format.
 59  *
 60  * Each process is normally operating in a protocol family,
 61  * whose protocols are used unless the process specifies otherwise.
 62  * Most families supply protocols to the basic socket types.  When
 63  * protocols are not present in the family, the higher level (roughly
 64  * ISO session layer) code in the system layers on the protocols
 65  * to support the socket types.
 66  */
 67 struct sockproto {
 68         short   sp_family;              /* protocol family */
 69         short   sp_protocol;            /* protocol within family */
 70 };
 71
 72 #define PF_UNSPEC       0               /* unspecified */
 73 #define PF_UNIX         1               /* UNIX internal protocol */
 74 #define PF_INET         2               /* internetwork: UDP, TCP, etc. */
 75 #define PF_IMPLINK      3               /* imp link protocols */
 76 #define PF_PUP          4               /* pup protocols: e.g. BSP */
 77 #define PF_CHAOS        5               /* mit CHAOS protocols */
 78 #define PF_OISCP        6               /* ois communication protocols */
 79 #define PF_NBS          7               /* nbs protocols */
 80 #define PF_ECMA         8               /* european computer manufacturers */
 81 #define PF_DATAKIT      9               /* datakit protocols */
 82 #define PF_CCITT        10              /* CCITT protocols, X.25 etc */
 83
 84 /*
 85  * Generic socket address format.
 86  *
 87  * Each process is also operating in an address family, whose
 88  * addresses are assigned unless otherwise requested.  The address
 89  * family used affects address properties: whether addresses are
 90  * externalized or internalized, location dependent or independent, etc.
 91  * The address can be defined directly if it fits in 14 bytes, or
 92  * a pointer and length can be given to variable length data.
 93  * We give these as two different structures to allow initialization.
 94  */
 95 struct sockaddr {
 96         short   sa_family;              /* address family */
 97         char    sa_data[14];            /* up to 14 bytes of direct address */
 98 };
 99
100 /*
101  * The first few address families correspond to protocol
102  * families.  Address families unrelated to protocol families
103  * are also possible.
104  */
105 #define AF_UNSPEC       0               /* unspecified */
106 #define AF_UNIX         1               /* local to host (pipes, portals) */
107 #define AF_INET         2               /* internetwork: UDP, TCP, etc. */
108 #define AF_IMPLINK      3               /* arpanet imp addresses */
109 #define AF_PUP          4               /* pup protocols: e.g. BSP */
110 #define AF_CHAOS        5               /* mit CHAOS protocols */
111 #define AF_OISCP        6               /* ois communication protocols */
112 #define AF_NBS          7               /* nbs protocols */
```

```
113 #define AF_ECMA         8               /* european computer manufacturers */
114 #define AF_DATAKIT      9               /* datakit protocols */
115 #define AF_CCITT        10              /* CCITT protocols, X.25 etc */
116 #define AF_ETHER        11              /* Ethernet Address */
117 #define AF_COUNT        12              /* A count */
118 #define AF_ETYPEFILTER  13              /* Ethernet filter */
119
120 #define AF_MAX          14
121
122 /*
123 MWP:
124 Sockaddr structure for link mode access to EXOS board.
125 */
126
127 #ifndef u_short
128 #define u_short unsigned short
129 #endif
130
131 #define sockaddr_link sad_link  /* for compiler */
132 struct sockaddr_link {
133         short           sl_family;
134         u_short         sl_types[6];
135         short           sl_zero;
136 #ifdef ONBOARD
137         struct enreq    *sl_pndpkt;     /* a part-empty pkt on this socket */
138 #endif
139 };
140
141 /* a handy macro */
142 #define saptr(x) ((struct sockaddr_link *)(((struct socket *)(x))->so_pcb))
```

```
 1
 2  /*
 3   * filename: SOIOCTL.H
 4   */
 5
 6
 7  /*
 8   * This file defines all the equate symbols for socket ioctl
 9   * commands. These values are actually passed onto to the board,
10   * hence should not be altered.
11   */
12
13
14
15  #define FIONREAD          (127)
16  #define FIONBIO           (126)
17  #define FIOASYNC          (125)
18  #define TIOCPKT           (112)          /* on pty: set/clear packet mode */
19  #define          TIOCPKT_DATA            0x00    /* data packet */
20  #define          TIOCPKT_FLUSHREAD       0x01    /* flush packet */
21  #define          TIOCPKT_FLUSHWRITE      0x02    /* flush packet */
22  #define          TIOCPKT_STOP            0x04    /* stop output */
23  #define          TIOCPKT_START           0x08    /* start output */
24  #define          TIOCPKT_NOSTOP          0x10    /* no more ^S, ^Q */
25  #define          TIOCPKT_DOSTOP          0x20    /* now do ^S ^Q */
26
27  #define SIOCDONE          (0)    /* shutdown read/write on socket */
28  #define SIOCSKEEP         (1)    /* set keep alive */
29  #define SIOCGKEEP         (2)    /* inspect keep alive */
30  #define SIOCSLINGER       (3)    /* set linger time */
31  #define SIOCGLINGER       (4)    /* get linger time */
32  #define SIOCSENDOOB       (5)    /* send out of band */
33  #define SIOCRCVOOB        (6)    /* get out of band */
34  #define SIOCATMARK        (7)    /* at out of band mark? */
35  #define SIOCSPGRP         (8)    /* set process group */
36  #define SIOCGPGRP         (9)    /* get process group */
37  #define SIOCADDRT         (10)   /* add a routing table entry */
38  #define SIOCDELRT         (11)   /* delete a routing table entry */
39  #define SIOCCHGRT         (12)   /* change a routing table entry */
```

```
 1
 2
 3
 4  #define ELMNTBUSY        1        /* the element is busy */
 5  #define ELMNTFREE        0        /* the element is free */
 6  #define NULLPOINTER      0        /* it is pointing to null element */
 7
 8
 9  #define MAXBUF           2        /* max no of transfer buffer  */
10  #define BUFSIZE          1024     /* size of each such buffer   */
11  #define MAXIOSB          10       /* max no of IO status block  */
12  #define MAXSOICTL        5        /* max no of SOictl structure */
13
14  #define SOLUN    20               /* EXOS0 LUN    */
15  #define SOEFN    1                /* efn          */
16
17  #define NOSOBUF          -10
18  #define NOSOIOSB         -11
19  #define NOSOICTL         -12
20  #define NOFREESOCKET     -13
21
```

```
 1  typedef struct { int r[1]; } *  physadr;
 2  typedef long            daddr_t;
 3  typedef char *          caddr_t;
 4  typedef unsigned long   mem_t;
 5  typedef unsigned short  ushort;
 6  typedef unsigned char   uchar_t;
 7  typedef ushort          ino_t;
 8  typedef short           cnt_t;
 9  typedef long            time_t;
10  typedef long            label_t[13];    /* regs d2-d7, a2-a7, pc */
11  typedef short           dev_t;
12  typedef long            off_t;
13  typedef long            paddr_t;
```

```
 1  /*
 2  @(#)xctype.h    1.3 5/31/85
 3
 4  character mappings.
 5  */
 6
 7  #define _U      01
 8  #define _L      02
 9  #define _N      04
10  #define _S      010
11  #define _P      020
12  #define _C      040
13  #define _B      0100
14  #define _X      0200
15
16  extern char _xctype[];
17
18  #define isalpha(c)      ((_xctype+1)[c]&(_U|_L))
19  #define isupper(c)      ((_xctype+1)[c]&_U)
20  #define islower(c)      ((_xctype+1)[c]&_L)
21  #define isdigit(c)      ((_xctype+1)[c]&_N)
22  #define isxdigit(c)     ((_xctype+1)[c]&_X)
23  #define isspace(c)      ((_xctype+1)[c]&_S)
24  #define ispunct(c)      ((_xctype+1)[c]&_P)
25  #define isalnum(c)      ((_xctype+1)[c]&(_U|_L|_N))
26  #define isprint(c)      ((_xctype+1)[c]&(_P|_U|_L|_N|_B))
27  #define isgraph(c)      ((_xctype+1)[c]&(_P|_U|_L|_N))
28  #define iscntrl(c)      ((_xctype+1)[c]&_C)
29  #define isascii(c)      ((unsigned)(c)<=0177)
30  #define _toupper(c)     ((islower(c))? (c)-'a'+'A' : (c))
31  #define _tolower(c)     ((isupper(c))? (c)-'A'+'a' : (c))
32  #define toascii(c)      ((c)&0177)
```

```
 1  /*
 2  @(#)xerrno.h    1.3 3/25/85
 3  Error values for xgenlib and xoslib.
 4  Some of these errors will make little sense on non-Unix systems.
 5  Other error numbers should be added for errors which make little
 6  sense on Unix systems.
 7  */
 8  #define XEPERM          -1              /* Not owner */
 9  #define XENOENT         -2              /* No such file or directory */
10  #define XESRCH          -3              /* No such process */
11  #define XEINTR          -4              /* Interrupted system call */
12  #define XEIO            -5              /* I/O error */
13  #define XENXIO          -6              /* No such device or address */
14  #define XE2BIG          -7              /* Arg list too long */
15  #define XENOEXEC                -8              /* Exec format error */
16  #define XEBADF          -9              /* Bad file number */
17  #define XECHILD         -10             /* No children */
18  #define XEAGAIN         -11             /* No more processes */
19  #define XENOMEM         -12             /* Not enough core */
20  #define XEACCES         -13             /* Permission denied */
21  #define XEFAULT         -14             /* Bad address */
22  #define XENOTBLK                -15             /* Block device required */
23  #define XEBUSY          -16             /* Mount device busy */
24  #define XEEXIST         -17             /* File exists */
25  #define XEXDEV          -18             /* Cross-device link */
26  #define XENODEV         -19             /* No such device */
27  #define XENOTDIR                -20             /* Not a directory*/
28  #define XEISDIR         -21             /* Is a directory */
29  #define XEINVAL         -22             /* Invalid argument */
30  #define XENFILE         -23             /* File table overflow */
31  #define XEMFILE         -24             /* Too many open files */
32  #define XENOTTY         -25             /* Not a typewriter */
33  #define XETXTBSY                -26             /* Text file busy */
34  #define XEFBIG          -27             /* File too large */
35  #define XENOSPC         -28             /* No space left on device */
36  #define XESPIPE         -29             /* Illegal seek */
37  #define XEROFS          -30             /* Read-only file system */
38  #define XEMLINK         -31             /* Too many links */
39  #define XEPIPE          -32             /* Broken pipe */
40
41  /* math software */
42  #define XEDOM           -33             /* Argument too large */
43  #define XERANGE         -34             /* Result too large */
44
45  /* interupt and non-blocking io */
46  #define XEWOULDBLOCK    -35             /* Operation would block */
47  #define XEINPROGRESS    -36             /* Operation now in progress */
48  #define XEALREADY       -37             /* Operation already in progress */
49
50  /* argument errors */
51  #define XENOTSOCK       -38             /* Socket operation on non-socket */
52  #define XEDESTADDRREQ   -39             /* Destination address required */
53  #define XEMSGSIZE       -40             /* Message too long */
54  #define XEPROTOTYPE     -41             /* Protocol wrong type for socket */
55  #define XENOPROTOOPT    -42             /* Protocol not available */
56  #define XEPROTONOSUPPORT        -43             /* Protocol not supported */
```

```
57 #define XESOCKTNOSUPPORT        -44             /* Socket type not supported */
58 #define XEOPNOTSUPP     -45             /* Operation not supported on socket */
59 #define XEPFNOSUPPORT   -46             /* Protocol family not supported */
60 #define XEAFNOSUPPORT   -47             /* Address family not supported by protocol fam
61 #define XEADDRINUSE     -48             /* Address already in use */
62 #define XEADDRNOTAVAIL  -49             /* Can't assign requested address */
63
64 /* operational errors */
65 #define XENETDOWN       -50             /* Network is down */
66 #define XENETUNREACH    -51             /* Network is unreachable */
67 #define XENETRESET      -52             /* Network dropped connection on reset */
68 #define XECONNABORTED   -53             /* Software caused connection abort */
69 #define XECONNRESET     -54             /* Connection reset by peer */
70 #define XENOBUFS                -55             /* No buffer space available */
71 #define XEISCONN                -56             /* Socket is already connected */
72 #define XENOTCONN       -57             /* Socket is not connected */
73 #define XESHUTDOWN      -58             /* Can't send after socket shutdown */
74 #define XETOOMANYREFS   -59             /* Too many references: can't splice */
75 #define XETIMEDOUT      -60             /* Connection timed out */
76 #define XECONNREFUSED   -61             /* Connection refused */
77
78 /* random errors */
79 #define XELOOP          -62             /* Too many levels of symbolic links */
80 #define XENAMETOOLONG   -63             /* File name too long */
81 #define XEHOSTDOWN      -64             /* Host is down */
82 #define XEHOSTUNREACH   -65             /* No route to host */
83 #define XSYSERR         -66             /* unspecified os specific error */
```

```
1  #include <rsxos.h>
2  #include <xstdio.h>
3  #include <xctype.h>
4  #include <xerrno.h>
5  #define xstrncpy  _ncpy
6  #define xstrncmp  _ncmp
7  #define xstrncat  _ncat
8
9  #define PTOLBYTE( cp )  (cp = cp)
```

```
 1  /*
 2   this file declares password structre
 3   */
 4  #define MAXUSERNAME     10
 5  #define MAXPASSWORD     8
 6  #define UICSIZE         10
 7  struct  passwd  {
 8     char login_uic[UICSIZE];
 9     char log_dev[6];
10     char cur_uic[UICSIZE];
11     char cur_dev[6];
12     };
13
```

```
 1  /*
 2  @(#)xspecial.h  1.8 5/7/85
 3  flags for special files
 4  */
 5  #define FILE_NAME       50      /* file name argument is to be used (as is) */
 6  #define CURRENT_DIR     51      /* current directory */
 7  #define HOME_DIR        52      /* user's initial location in file system */
 8  #define CD_RELATIVE     53      /* name is relative to current directory */
 9  #define HM_RELATIVE     54      /* name is relative to home directory */
10  #define UP_DIRECTORY    55      /* parent directory (for xchdir) */
11
12  /*
13  flags for psuedo-file objects
14  */
15  #define LS              101     /* short directory listing */
16  #define LS_ARG          102     /* short listing of named directory */
17  #define LSLONG          103     /* long directory listing */
18  #define LSLONG_ARG      104     /* long listing of named directory */
19  #define PWD             105     /* return name of current directory */
20
21  /*
22  flags for file openning modes.
23  */
24  #define XFREAD          1       /* open for reading */
25  #define XFWRITE         2       /* open for writing */
26  #define XFAPPEND        8       /* add to an existing file (FWRITE also
27                                         must be set) */
28  #define XFCREAT         0x80    /* create file, if it doesn't exist */
29  #define XFTRUNC         0x100   /* truncate file (FWRITE also must be set)*/
30  #define XFASCII         0x200   /* file is ascii (for systems which care) */
31  /*
32  Note: XFCREAT is a separate issue from XFTRUNC and XFAPPEND, which are
33  mutually exclusive.
34  */
35
36  /*
37  Information for FTP style files
38  */
39  #define RT_ASCII        1       /* ascii character set */
40  #define RT_EBCDIC       2       /* ebcdic character set */
41  #define RT_IMAGE        3       /* uninterpretted bit stream */
42  #define RT_LOCALBYTE    4       /* wierd sized bytes */
43
44  #define TF_NONPRINT     1       /* no imbedded carriage control */
45  #define TF_TELNET       2       /* telnet style data */
46  #define TF_FORTRAN      3       /* 1st collumn == carriage control */
47
48  #define IS_FILE         1       /* Unstructured file */
49  #define IS_RECORD       2       /* FTP record internal structure */
50  #define IS_PAGE         3       /* FTP page internal structure */
51
52  #define TM_STREAM       1       /* stream transmission */
53  #define TM_BLOCK        2       /* block transmission */
54  #define TM_COMPRESSED   3       /* data compressed */
55
56  struct ftp_attr {
```

```
57          int rep_type;           /* data repesentation one of:
58                                          RT_ASCII, RT_EBCDIC, RT_IMAGE or
59                                          RT_LOCALBYTE */
60          int format;             /* format for character files one of:
61                                          TF_NONPRINT, TF_TELNET or
62                                          TF_FORTRAN */
63          int structure;          /* internal structure one of: IS_FILE,
64                                          IS_RECORD, IS_PAGE */
65          int trans_mode;         /* transmition mode one of: TM_STREAM,
66                                          TM_BLOCK or TM_COMPRESSED */
67          int byte_sz;            /* byte size if represetation type
68                                          is RT_LOCALBYTE */
69  };
70
71  /*
72  Flags for setting terminal options with xsetterm.
73  */
74  #define XON_STERM       1       /* turn option on */
75  #define XOFF_STERM      0       /* turn option off */
76  #define XECHO           1       /* local echo? */
77  #define XLINE_EDIT      2       /* driver handles line edit? */
78
79
80  #define MXNAMELEN       255     /* maximum length for file names
81                                          ( system dependent ) */
82
83  #ifdef zilog
84  /*
85   * S8000 does setjmp() differently, and calls it setret().
86   * Do NOT call setret() from routine which declares register variables!
87   */
88  #define xsetjmp(x)      setret(x)
89  #else
90  #define xsetjmp(x)      setjmp(x)       /* Unix only */
91  #endif
```

```
 1  /*
 2  @(#)xstdio.h    1.5 6/4/85
 3
 4  Definitions for EXOS standard io objects
 5          (useful for porting code to non-unix systems)
 6  */
 7
 8  /*
 9   save space on systems with limited data segment size.
10   */
11  #ifdef xenix286
12  #define XBUFSIZ 512
13  #else
14  #ifdef rsx
15  #define XBUFSIZ 512
16  #else
17  #define XBUFSIZ 1024
18  #endif
19  #endif
20
21  #define _XNFILE 20
22  extern  struct  _xiobuf {
23          int     _cnt;
24          char    *_ptr;
25          char    *_base;
26          int     _bufsiz;
27          short   _flag;
28          char    _file;
29          struct _xiobuf *_succ;          /* forward link (added) */
30          struct _xiobuf *_pred;          /* backward link (added) */
31          char    *_sys_id;               /* system specific identifier (added) */
32          int     (*_read)();             /* field to be added */
33          int     (*_write)();            /* field to be added */
34          int     (*_ioctl)();            /* field to be added */
35          int     (*_close)();            /* field to be added */
36  } _xiob[_XNFILE];
37
38  #define _XIOREAD        01
39  #define _XIOWRT 02
40  #define _XIONBF 04
41  #define _XIOMYBUF       010
42  #define _XIOEOF 020
43  #define _XIOERR 040
44  #define _XIOSTRG        0100
45  #define _XIOLBF 0200
46  #define _XIORW  0400
47  #define _XPrimary 01000         /* primary copy of object */
48  #define _XUsed    02000         /* on if object is in use */
49  #define XNULL   0
50  #define XFILE   struct _xiobuf
51  #define XEOF    (-1)
52
53  #define xstdin  (&_xiob[0])
54  #define xstdout (&_xiob[1])
55  #define xstderr (&_xiob[2])
56  #define xgetc(p)                (--(p)->_cnt>=0? *(p)->_ptr++&0377:_xfilbuf(p))
```

```
57  #define xgetchar()      xgetc(xstdin)
58  #define xputc(x,p) (--(p)->_cnt>=0? ((int)(*(p)->_ptr++=(unsigned)(x))):_xflsbf((unsign
59  #define xputchar(x)     xputc(x,xstdout)
60  #define xfeof(p)                (((p)->_flag&_XIOEOF)!=0)
61  #define xferror(p)      (((p)->_flag&_XIOERR)!=0)
62  #define xfileno(p)      ((p)->_file)
63
64  extern int xnofunc();
65  XFILE   *xodopen();
66  char    *xogets();
67  char    *xsprintf();            /* too painful to do right */
```

```
 1  /*
 2  @(#)ftpc.h      1.2 4/11/85
 3
 4  Header files for generic client side of FTP
 5  */
 6
 7  #include <rsxos.h>
 8  #include <xstdio.h>
 9  #include <xctype.h>
10  #include <xerrno.h>
11  #include <xspecial.h>
12  #include <socket.h>
13  #include <netdb.h>
14
15  typedef int jmp_buf;
16  #include <ftp.h>
17  #include <in.h>
18  #define SIOCDONE XNULL
19  #define FIONBIO (126)
20  #define appendhelp  happend
21  #define deletehelp  hdelete
22  #define disconhelp  hdiscon
23  #define mdeletehelp hmdelete
24  #define renamehelp  hrename
25  #define statushelp  hstatus
26  #define structhelp  hstruct
27  #define renamecmd   cmdrename
28  extern xclose();
29  extern int  figit;
30  extern int  errno;
31  extern long xpasstnet();
32  extern long xpassfnet();
33  #define VOID  figit = (int)
34
35  /*
36   * FTP global referrences.
37   */
38  #include "varpat.h"
39
40  /*
41   * Options and other state info.
42   */
43  extern int trace;               /* trace packets exchanged */
44  extern int hash;                /* print # for each buffer transferred */
45  extern int sendport;    /* use PORT cmd for each data connection */
46  extern int      verbose;        /* print messages coming back from server */
47  extern int      connected;      /* connected to server */
48  extern int      fromatty;       /* input is from a terminal */
49  extern int      interactive;    /* interactively prompt on m* cmds */
50  extern int      debug;          /* debugging level */
51  extern int      bell;           /* ring bell on cmd completion */
52  extern int      doglob;         /* glob local file names */
53  extern int      autologin;      /* establish user account on connection */
54  extern char     typename[32];   /* name of file transfer type */
55  extern int      type;           /* file transfer type */
56  extern char     structname[32]; /* name of file transfer structure */
```

```
57  extern int      stru;           /* file transfer structure */
58  extern char     formname[32];   /* name of file transfer format */
59  extern int      form;           /* file transfer format */
60  extern char     modename[32];   /* name of file transfer mode */
61  extern int      mode;           /* file transfer mode */
62  extern char     bytename[32];   /* local byte size in ascii */
63  extern int      bytesize;       /* local byte size in binary */
64
65  extern char     *hostname;      /* name of host connected to */
66  extern struct   servent *sp;    /* service spec for tcp/ftp */
67
68  extern jmp_buf toplevel;        /* non-local goto stuff for cmd scanner */
69
70  extern char     line[200];      /* input line buffer */
71  extern int      margc;          /* count of arguments on input line */
72  extern char     **margv;        /* args parsed from input line */
73
74  extern int      options;        /* used during socket creation */
75
76  /*
77   * Format of command table.
78   */
79  extern struct cmd {
80          char    *c_name;        /* name of command */
81          char    *c_help;        /* help string */
82          char    c_bell;         /* give bell when command completes */
83          char    c_conn;         /* must be connected to use command */
84          int     (*c_handler)(); /* function to call */
85  };
86
87  extern  char *tail();
88  extern  char *remglob();
89  extern  int errno;
90
```

```
1
2  /*@(#)varpat.h  1.8 4/11/85*/
3
4  #define connected       conned
5
6  #define connecthelp     connhelp
7  #define mdeletehelp     mdelhelp
8  #define receivehelp     recehelp
9  #define verbosehelp     verbhelp
```

```
 1  #ifndef lint
 2  static char sccsid[] =
 3   " @(#)cmds.c   1.24 8/28/85";
 4  #endif
 5
 6  /*
 7   * FTP User Program -- Command Routines.
 8   */
 9  #include "ftpc.h"
10
11  extern  char *globerr;
12  extern  char **xglob();
13  extern  char **xmkarglist();
14  extern  short gflag;
15  extern  char *remglob();
16  extern  char *getenv();
17  extern  char *xstrchr();
18  extern  char *xstrrchr();
19  static char **glizept = (char **)0;
20
21  #define BUFSIZ 1024
22
23  /*
24   * Connect to peer server and
25   * auto-login, if possible.
26   */
27  setpeer(argc, argv)
28          int argc;
29          char *argv[];
30  {
31          struct hostent *host, *hookup();
32          int port;
33          int madeargs = 0;
34
35          if (connected) {
36                  xoprintf(xstdout,
37                          "Already connected to %s, use close first.\n",
38                          hostname);
39                  return;
40          }
41          if (argc < 2) {
42                  xstrcat(line, " ");
43                  xoprintf(xstdout,"(to) ");
44                  xfflush( xstdout );
45                  xgets(&line[xstrlen(line)]);
46                  argv = xmkarglist( line, &argc );
47                  madeargs = 1;
48          }
49          if (argc < 2 || argc > 3) {
50                  xoprintf(xstdout,"usage: %s host-name [port]\n", argv[0]);
51                  goto endspeer;
52          }
53          port = sp->s_port;
54          if (argc > 2) {
55                  port = xatoi(argv[2]);
56                  if (port <= 0) {
```

```
57                          xoprintf(xstdout,"%s: bad port number-- %s\n",
58                                  argv[1], argv[2]);
59                          xoprintf(xstdout,"usage: %s host-name [port]\n", argv[0]);
60                          goto endspeer;
61                  }
62          }
63          port = xhtons(port);
64          host = hookup(argv[1], port);
65          if (host) {
66                  connected = 1;
67                  if (autologin && fromatty )
68                          login(host);
69          }
70  endspeer:
71          if( madeargs )
72                  xdealglob( argv );
73  }
74
75  struct  types {
76          char    *t_name;
77          char    *t_mode;
78          int     t_type;
79          char    *t_arg;
80  } types[] = {
81          { "ascii",      "A",    TYPE_A, 0 },
82          { "binary",     "I",    TYPE_I, 0 },
83          { "image",      "I",    TYPE_I, 0 },
84          { "ebcdic",     "E",    TYPE_E, 0 },
85          { "tenex",      "L",    TYPE_L, bytename },
86          0
87  };
88
89  /*
90   * Set transfer type.
91   */
92  settype(argc, argv)
93          char *argv[];
94  {
95          register struct types *p;
96          int comret;
97
98          if (argc > 2) {
99                  char *sep;
100
101                 xoprintf(xstdout,"usage: %s [", argv[0]);
102                 sep = " ";
103                 for (p = types; p->t_name; p++) {
104                         xoprintf(xstdout,"%s%s", sep, p->t_name);
105                         if (*sep == ' ')
106                                 sep = " | ";
107                 }
108                 xoprintf(xstdout," ]\n");
109                 return;
110         }
111         if (argc < 2) {
112                 xoprintf(xstdout,"Using %s mode to transfer files.\n", typename);
```

```
113                 return;
114         }
115         for (p = types; p->t_name; p++)
116                 if (xstrcmp(argv[1], p->t_name) == 0)
117                         break;
118         if (p->t_name == 0) {
119                 xoprintf(xstdout,"%s: unknown mode\n", argv[1]);
120                 return;
121         }
122         if ((p->t_arg != XNULL) && (*(p->t_arg) != '\0'))
123                 comret = command ("TYPE %s %s", p->t_mode, p->t_arg);
124         else
125                 comret = command("TYPE %s", p->t_mode);
126         if (comret == COMPLETE) {
127                 xstrcpy(typename, p->t_name);
128                 type = p->t_type;
129         }
130 }
131
132 /*
133  * Set binary transfer type.
134  */
135 /*VARARGS*/
136 setbinary()
137 {
138
139         call(settype, "type", "binary", 0);
140 }
141
142 /*
143  * Set ascii transfer type.
144  */
145 /*VARARGS*/
146 setascii()
147 {
148
149         call(settype, "type", "ascii", 0);
150 }
151
152 /*
153  * Set tenex transfer type.
154  */
155 /*VARARGS*/
156 settenex()
157 {
158
159         call(settype, "type", "tenex", 0);
160 }
161
162 /*
163  * Set ebcdic transfer type.
164  */
165 /*VARARGS*/
166 setebcdic()
167 {
168
```

```
169             call(settype, "type", "ebcdic", 0);
170     }
171
172     /*
173      * Set file transfer mode.
174      */
175     setmode(argc, argv)
176             char *argv[];
177     {
178
179             xoprintf(xstdout,"We only support %s mode, sorry.\n", modename);
180     }
181
182     /*
183      * Set file transfer format.
184      */
185     setform(argc, argv)
186             char *argv[];
187     {
188
189             xoprintf(xstdout,"We only support %s format, sorry.\n", formname);
190     }
191
192     /*
193      * Set file transfer structure.
194      */
195     setstruct(argc, argv)
196             char *argv[];
197     {
198
199             xoprintf(xstdout,"We only support %s structure, sorry.\n", structname);
200     }
201
202     /*
203      * Send a single file.
204      */
205     put(argc, argv)
206             int argc;
207             char *argv[];
208     {
209             char *cmd;
210             char *remote;
211             char *local;
212             int madeargs = 0;
213             int madeglob = 0;
214
215             if (argc == 2)
216                     argc++, remote = argv[1];
217             else if (argc < 2) {
218                     xstrcat(line, " ");
219                     xoprintf(xstdout,"(local-file)");
220                     xfflush( xstdout );
221                     xgets(&line[xstrlen(line)]);
222                     argv = xmkarglist( line, &argc );
223                     madeargs = 1;
224             }
```

```
225         else {
226                 remote = argv[2];
227         }
228         if( argc < 2 ) {
229                 xoprintf(xstdout,"%s local-file [remote-file]\n", argv[0]);
230                 goto endput;
231         }
232         if (argc < 3) {
233                 xstrcat(line, " ");
234                 xoprintf(xstdout,"(remote-file, %s is default) ", argv[1] );
235                 xfflush( xstdout );
236                 xgets(&line[xstrlen(line)]);
237                 if( madeargs )
238                         xdealglob( argv );
239                 argv = xmkarglist( line, &argc );
240                 madeargs = 1;
241                 remote = argv[2];
242         }
243         if (argc < 3) {
244                 remote = argv[1];
245         }
246         local = argv[1];
247         if (!(madeglob = globulize(&local)))
248                 goto endput;
249         cmd = (argv[0][0] == 'a') ? "APPE" : "STOR";
250         sendrequest(cmd, local, remote );
251 endput:
252         if( madeglob && doglob )
253                 xdealglob( glizept );
254         if( madeargs )
255                 xdealglob( argv );
256 }
257
258 /*
259  * Send multiple files.
260  */
261 mput(argc, argv)
262         char *argv[];
263 {
264         char **cpp, **gargs = XNULL;
265         int madeargs = 0;
266         int cfrval;
267         int doall = 0;
268
269         if (argc < 2) {
270                 xstrcat(line, " ");
271                 xoprintf(xstdout,"(local-files) ");
272                 xfflush( xstdout );
273                 xgets(&line[xstrlen(line)]);
274                 argv = xmkarglist( line, &argc );
275                 madeargs = 1;
276         }
277         if (argc < 2) {
278                 xoprintf(xstdout,"%s local-files\n", argv[0]);
279                 goto endmput;
280         }
```

```
281         cpp = argv + 1;
282         if (doglob) {
283                 gargs = xglob(cpp);
284                 if (globerr != XNULL) {
285                         xoprintf(xstdout,"%s\n", globerr);
286                         if (gargs)
287                                 xdealglob(gargs);
288                         goto endmput;
289                 }
290         }
291         if (gargs != XNULL)
292                 cpp = gargs;
293         for (; *cpp != XNULL; cpp++)
294                 {
295                 if( !doall )
296                         cfrval = confirm(argv[0], *cpp);
297                 if( cfrval == 'a' )
298                         doall = 1;
299                 if( cfrval == 'q' )
300                         break;
301                 if ( cfrval )
302                         sendrequest("STOR", *cpp, *cpp);
303                 }
304         if (gargs != XNULL)
305                 xdealglob(gargs);
306 endmput:
307         if( madeargs )
308                 xdealglob( argv );
309 }
310
311 /*
312  * Receive one file.
313  */
314 get(argc, argv)
315         char *argv[];
316 {
317         int madeargs = 0;
318         int madeglob = 0;
319         char *local;
320
321         if (argc == 2)
322                 argc++, local = argv[1];
323         else if (argc < 2) {
324                 xstrcat(line, " ");
325                 xoprintf(xstdout,"(remote-file) ");
326                 xfflush( xstdout );
327                 xgets(&line[xstrlen(line)]);
328                 argv = xmkarglist( line, &argc );
329                 madeargs = 1;
330         }
331         else {
332                 local = argv[2];
333         }
334         if (argc < 2) {
335                 xoprintf(xstdout,"%s remote-file [ local-file ]\n", argv[0]);
336                 goto endget;
```

```
337         }
338         if (argc < 3) {
339                 xstrcat(line, " ");
340                 xoprintf(xstdout,"(local-file, %s is default) ", argv[1] );
341                 xfflush( xstdout );
342                 xgets(&line[xstrlen(line)]);
343                 if( madeargs )
344                         xdealglob( argv );
345                 argv = xmkarglist( line, &argc );
346                 madeargs = 1;
347                 local = argv[2];
348         }
349         if (argc < 3) {
350                 local = argv[1];
351         }
352         if (!(madeglob = globulize(&local)))
353                 goto endget;
354         recvrequest("RETR", local, argv[1], "w");
355 endget:
356         if( madeglob && doglob )
357                 xdealglob( glizept );
358         if( madeargs )
359                 xdealglob( argv );
360 }
361
362 /*
363  * Get multiple files.
364  */
365 mget(argc, argv)
366         char *argv[];
367 {
368         char *cp;
369         int madeargs = 0;
370         int cfrval;
371         int doall = 0;
372
373         if (argc < 2) {
374                 xstrcat(line, " ");
375                 xoprintf(xstdout,"(remote-files) ");
376                 xfflush( xstdout );
377                 xgets(&line[xstrlen(line)]);
378                 argv = xmkarglist( line, &argc );
379                 madeargs = 1;
380         }
381         if (argc < 2) {
382                 xoprintf(xstdout,"%s remote-files\n", argv[0]);
383                 goto endmget;
384         }
385         while ((cp = remglob(argc, argv)) != XNULL)
386                 {
387                 if( !doall )
388                         cfrval = confirm(argv[0], cp );
389                 if( cfrval == 'a' )
390                         doall = 1;
391                 if( cfrval == 'q' ) {
392                         while ((cp = remglob(argc, argv)) != XNULL)
```

```
393                                     ;
394                             break;
395                     }
396                     if ( cfrval )
397                             recvrequest("RETR", cp, cp, "w");
398                     }
399     endmget:
400             if( madeargs )
401                     xdealglob( argv );
402     }
403
404     char temp[16] = { 0 };
405     char *
406     remglob(argc, argv)
407             char *argv[];
408     {
409             /*
410             char temp[16];
411             */
412             static char buf[MAXPATHLEN] = {0};
413             static XFILE *ftemp = XNULL;
414             static char **args;
415             int oldverbose;
416             char *cp, *mode;
417             int ftemi;
418             int oldtype;
419             char oldtname[25];
420
421             if (!doglob) {
422                     if (args == XNULL)
423                             args = argv;
424                     if ((cp = *++args) == XNULL)
425                             args = XNULL;
426                     return (cp);
427             }
428             if (ftemp == XNULL) {
429                     xstrcpy(temp, SCRATCHFILE );
430                     xmktemp(temp);
431                     oldverbose = verbose, verbose = 0;
432                     oldtype = type;
433                     if( oldtype != TYPE_A ) {
434                             /*
435                              * do remote globbing in ascii mode
436                              */
437                             xstrcpy( oldtname, typename );
438                             call( settype, "type", "ascii", 0 );
439                     }
440                     for (mode = "w"; *++argv != XNULL; mode = "a")
441                             recvrequest ("NLST", temp, *argv, mode);
442                     if( oldtype != TYPE_A ) {
443                             /*
444                              * restore original type
445                              */
446                             call( settype, "type", oldtname, 0 );
447                     }
448                     verbose = oldverbose;
```

```
449                     ftemi = xdopen(temp, XFREAD | XFASCII, FILE_NAME );
450                     xunlink(temp, FILE_NAME );
451                     ftemp = xodopen( ftemi, "r" );
452                     if (ftemp == XNULL) {
453                             xoprintf(xstdout,
454                                     "can't find list of remote files, oops\n");
455                             return (XNULL);
456                     }
457             }
458             if (xogets(buf, sizeof (buf), ftemp) == XNULL) {
459                     xclose(xfileno(ftemp)), ftemp = XNULL;
460                     return (XNULL);
461             }
462             if ((cp = xstrchr(buf, '\n')) != XNULL)
463                     *cp = '\0';
464             return (buf);
465     }
466
467     char *
468     onoff(bool)
469             int bool;
470     {
471
472             return (bool ? "on" : "off");
473     }
474
475     /*
476      * Show status.
477      */
478     status(argc, argv)
479             char *argv[];
480     {
481
482             if (connected)
483                     xoprintf(xstdout,"Connected to %s.\n", hostname);
484             else
485                     xoprintf(xstdout,"Not connected.\n");
486             xoprintf(xstdout,"Mode: %s; Type: %s; Form: %s; Structure: %s\n",
487                     modename, typename, formname, structname);
488             xoprintf(xstdout,"Verbose: %s; Bell: %s; Prompting: %s; Globbing: %s\n",
489                     onoff(verbose), onoff(bell), onoff(interactive),
490                     onoff(doglob));
491             xoprintf(xstdout,"Hash mark printing: %s; Use of PORT cmds: %s\n",
492                     onoff(hash), onoff(sendport));
493     }
494
495     /*
496      * Set beep on cmd completed mode.
497      */
498     /*VARARGS*/
499     setbell()
500     {
501
502             bell = !bell;
503             xoprintf(xstdout,"Bell mode %s.\n", onoff(bell));
504     }
```

```
505
506  /*
507   * Turn on packet tracing.
508   */
509  /*VARARGS*/
510  settrace()
511  {
512
513          trace = !trace;
514          xoprintf(xstdout,"Packet tracing %s.\n", onoff(trace));
515  }
516
517  /*
518   * Toggle hash mark printing during transfers.
519   */
520  /*VARARGS*/
521  sethash()
522  {
523
524          hash = !hash;
525          xoprintf(xstdout,"Hash mark printing %s", onoff(hash));
526          if (hash)
527                  xoprintf(xstdout," (%d bytes/hash mark)", BUFSIZ);
528          xoprintf(xstdout,".\n");
529  }
530
531  /*
532   * Turn on printing of server echo's.
533   */
534  /*VARARGS*/
535  setverbose()
536  {
537
538          verbose = !verbose;
539          xoprintf(xstdout,"Verbose mode %s.\n", onoff(verbose));
540  }
541
542  /*
543   * Toggle PORT cmd use before each data connection.
544   */
545  /*VARARGS*/
546  setport()
547  {
548
549          sendport = !sendport;
550          xoprintf(xstdout,"Use of PORT cmds %s.\n", onoff(sendport));
551  }
552
553  /*
554   * Turn on interactive prompting
555   * during mget, mput, and mdelete.
556   */
557  /*VARARGS*/
558  setprompt()
559  {
560
```

```
561         interactive = !interactive;
562         xoprintf(xstdout,"Interactive mode %s.\n", onoff(interactive));
563 }
564
565 /*
566  * Toggle metacharacter interpretation
567  * on local file names.
568  */
569 /*VARARGS*/
570 setglob()
571 {
572
573         doglob = !doglob;
574         xoprintf(xstdout,"Globbing %s.\n", onoff(doglob));
575 }
576
577 /*
578  * Set debugging mode on/off and/or
579  * set level of debugging.
580  */
581 /*VARARGS*/
582 setdebug(argc, argv)
583         char *argv[];
584 {
585         int val;
586
587         if (argc > 1) {
588                 val = xatoi(argv[1]);
589                 if (val < 0) {
590                         xoprintf(xstdout,"%s: bad debugging value.\n", argv[1]);
591                         return;
592                 }
593         } else
594                 val = !debug;
595         debug = val;
596         if (debug)
597                 options |= SO_DEBUG;
598         else
599                 options &= ~SO_DEBUG;
600         xoprintf(xstdout,"Debugging %s (debug=%d).\n", onoff(debug), debug);
601 }
602
603 /*
604  * Set current working directory
605  * on remote machine.
606  */
607 cd(argc, argv)
608         char *argv[];
609 {
610         int madeargs = 0;
611
612         if (argc < 2) {
613                 xstrcat(line, " ");
614                 xoprintf(xstdout,"(remote-directory) ");
615                 xfflush( xstdout );
616                 xgets(&line[xstrlen(line)]);
```

```
617                     argv = xmkarglist( line, &argc );
618                     madeargs = 1;
619             }
620             if (argc < 2) {
621                     xoprintf(xstdout,"%s remote-directory\n", argv[0]);
622                     goto endcd;
623             }
624             VOID command("CWD %s", argv[1]);
625     endcd:
626             if( madeargs )
627                     xdealglob( argv );
628     }
629
630     /*
631      * Set current working directory
632      * on local machine.
633      */
634     lcd(argc, argv)
635             char *argv[];
636     {
637             char buf[MAXPATHLEN];
638             char *dir;
639             int madeglob = 0;
640             int rval;
641             int func_code;
642
643             if (argc < 2)
644                     argc++, dir = (char *)0, func_code = HOME_DIR;
645             else
646                     dir = argv[1], func_code = FILE_NAME;
647             if (argc != 2) {
648                     xoprintf(xstdout,"%s local-directory\n", argv[0]);
649                     goto endlcd;
650             }
651             if (!(madeglob = globulize(&dir)))
652                     goto endlcd;
653             if ((rval = xchdir(dir, func_code)) < 0) {
654                     xperror(rval, dir);
655                     goto endlcd;
656             }
657     endlcd:
658             if( madeglob && doglob )
659                     xdealglob( glizept );
660     }
661
662     /*
663      * Delete a single file.
664      */
665     delete(argc, argv)
666             char *argv[];
667     {
668             int madeargs = 0;
669
670             if (argc < 2) {
671                     xstrcat(line, " ");
672                     xoprintf(xstdout,"(remote-file) ");
```

```
673                     xfflush( xstdout );
674                     xgets(&line[xstrlen(line)]);
675                     argv = xmkarglist( line, &argc );
676                     madeargs = 1;
677             }
678             if (argc < 2) {
679                     xoprintf(xstdout,"%s remote-file\n", argv[0]);
680                     goto enddelete;
681             }
682             VOID command("DELE %s", argv[1]);
683     enddelete:
684             if( madeargs )
685                     xdealglob( argv );
686     }
687
688     /*
689      * Delete multiple files.
690      */
691     mdelete(argc, argv)
692             char *argv[];
693     {
694             char *cp;
695             int madeargs = 0;
696             int cfrval;
697             int doall = 0;
698
699             if (argc < 2) {
700                     xstrcat(line, " ");
701                     xoprintf(xstdout,"(remote-files) ");
702                     xfflush( xstdout );
703                     xgets(&line[xstrlen(line)]);
704                     argv = xmkarglist( line, &argc );
705                     madeargs = 1;
706             }
707             if (argc < 2) {
708                     xoprintf(xstdout,"%s remote-files\n", argv[0]);
709                     goto endmdel;
710             }
711             while ((cp = remglob(argc, argv)) != XNULL)
712                     {
713                     if( !doall )
714                             cfrval = confirm(argv[0], cp);
715                     if( cfrval == 'a' )
716                             doall = 1;
717                     if( cfrval == 'q' ) {
718                             while ((cp = remglob(argc, argv)) != XNULL)
719                                     ;
720                             break;
721                     }
722                     if ( cfrval )
723                             VOID command("DELE %s", cp);
724                     }
725     endmdel:
726             if( madeargs )
727                     xdealglob( argv );
728     }
```

```
729
730  /*
731   * Rename a remote file.
732   */
733  renamefile(argc, argv)
734          char *argv[];
735  {
736          int madeargs = 0;
737
738          if (argc < 2) {
739                  xstrcat(line, " ");
740                  xoprintf(xstdout,"(from-name) ");
741                  xfflush( xstdout );
742                  xgets(&line[xstrlen(line)]);
743                  argv = xmkarglist( line, &argc );
744                  madeargs = 1;
745          }
746          if (argc < 2) {
747  usage:
748                  xoprintf(xstdout,"%s from-name to-name\n", argv[0]);
749                  goto endrname;
750          }
751          if (argc < 3) {
752                  xstrcat(line, " ");
753                  xoprintf(xstdout,"(to-name) ");
754                  xfflush( xstdout );
755                  xgets(&line[xstrlen(line)]);
756                  if( madeargs )
757                          xdealglob( argv );
758                  argv = xmkarglist( line, &argc );
759          }
760          if (argc < 3)
761                  goto usage;
762          if (command("RNFR %s", argv[1]) == CONTINUE)
763                  VOID command("RNTO %s", argv[2]);
764          else
765                  VOID command("RNTO  "); /* keep server happy */
766  endrname:
767          if( madeargs )
768                  xdealglob( argv );
769  }
770
771  /*
772   * Get a directory listing
773   * of remote files.
774   */
775  ls(argc, argv)
776          char *argv[];
777  {
778          char *cmd;
779          char *rdir;
780          char *lfile;
781          int madeglob = 0;
782
783          if (argc < 2)
784                  argc++, rdir = XNULL;
```

```
785             else
786                     rdir = argv[1];
787             if (argc < 3)
788                     argc++, lfile = "-";
789             else
790                     lfile = argv[2];
791             if (argc > 3) {
792                     xoprintf(xstdout,"usage: %s remote-directory local-file\n", argv[0]);
793                     return;
794             }
795             cmd = argv[0][0] == 'l' ? "NLST" : "LIST";
796             if (xstrcmp(lfile, "-") && !(madeglob = globulize(&lfile)))
797                     goto endls;
798             recvrequest(cmd, lfile, rdir, "w");
799     endls:
800             if( madeglob && doglob )
801                     xdealglob( glizept );
802     }
803
804     /*
805      * Get a directory listing
806      * of multiple remote files.
807      */
808     mls(argc, argv)
809             char *argv[];
810     {
811             char *cmd, *mode;
812             int i, dest;
813             char *rdir;
814             char *lfile;
815             int madeglob = 0;
816             int cfrval;
817
818             if (argc < 2)
819                     argc++, rdir = XNULL;
820             else
821                     rdir = argv[1];
822             if (argc < 3)
823                     {
824                     argc++, lfile = "-";
825                     dest = argc - 1;
826                     }
827             else
828                     {
829                     dest = argc - 1;
830                     lfile = argv[dest];
831                     }
832             cmd = argv[0][1] == 'l' ? "NLST" : "LIST";
833             if (xstrcmp(lfile, "-") != 0)
834                     if (!(madeglob = globulize(&lfile)) ||
835                             !(cfrval = confirm("local-file", lfile)) ||
836                             cfrval == 'q'
837                             )
838                             goto endmls;
839             for (i = 2, mode = "w"; i < dest + 1 ; i++, mode = "a")
840                     {
```

```
841                     recvrequest(cmd, lfile, rdir, mode);
842                     rdir = argv[i];
843                     }
844     endmls:
845             if( madeglob && doglob )
846                     xdealglob( glizept );
847     }
848
849     #ifndef SHELLESCAPE
850
851     /*
852     * shell escape not implemented
853     */
854     shell( )
855
856     {
857     xoprintf(xstdout, "shell escapes not implemented.\n" );
858     }
859
860     #else
861
862     /*
863      * shell escape for a specifc OS
864      */
865     shell()
866     {
867     xshell();
868     }
869
870     #endif SHELLESCAPE
871
872
873     /*
874      * Send new user information (re-login)
875      */
876     user(argc, argv)
877             int argc;
878             char **argv;
879     {
880             char acct[80], *xgetpass();
881             int n;
882             int madeargs = 0;
883             char *password;
884             char *account;
885
886             if (argc < 2) {
887                     xstrcat(line, " ");
888                     xoprintf(xstdout,"(Remote Username) ");
889                     xfflush( xstdout );
890                     xgets(&line[xstrlen(line)]);
891                     argv = xmkarglist( line, &argc );
892                     madeargs = 1;
893             }
894             if (argc < 2 || argc > 4) {
895                     xoprintf(xstdout,
896                             "usage: %s username [password] [account]\n", argv[0]);
```

```
897                     goto enduser;
898             }
899             n = command("USER %s", argv[1]);
900             if (n == CONTINUE) {
901                     if (argc < 3 )
902                             password = xgetpass("Remote Password: "), argc++;
903                     else
904                             password = argv[2];
905                     n = command("PASS %s", password);
906                     if (n == CONTINUE) {
907                             if (argc < 4) {
908                                     xoprintf(xstdout, "Remote Account:");
909                                     VOID xfflush(xstdout);
910                                     VOID xogets(acct, sizeof(acct) - 1, xstdin);
911                                     acct[xstrlen(acct) - 1] = '\0';
912                                     account = acct;
913                                     argc++;
914                             }
915                             else
916                                     account = argv[3];
917                             n = command("ACCT %s", account);
918                     }
919             }
920             if (n != COMPLETE) {
921                     xoprintf(xstderr, "Login failed.\n");
922                     goto enduser;
923             }
924             if( madeargs )
925                     xdealglob( argv );
926             return (1);
927     enduser:
928             if( madeargs )
929                     xdealglob( argv );
930             return( 0 );
931     }
932
933     /*
934      * Print working directory.
935      */
936     /*VARARGS*/
937     pwd()
938     {
939     int noverbose = 0;
940
941             if( !verbose )
942                     {
943                     noverbose = 1;
944                     verbose = 1;
945                     }
946             VOID command("XPWD");
947             if( noverbose )
948                     verbose = 0;
949     }
950
951     /*
952      * Make a directory.
```

```
 953  */
 954 makedir(argc, argv)
 955         char *argv[];
 956 {
 957         int madeargs = 0;
 958
 959         if (argc < 2) {
 960                 xstrcat(line, " ");
 961                 xoprintf(xstdout,"(directory-name) ");
 962                 xfflush( xstdout );
 963                 xgets(&line[xstrlen(line)]);
 964                 argv = xmkarglist( line, &argc );
 965                 madeargs = 1;
 966         }
 967         if (argc < 2) {
 968                 xoprintf(xstdout,"%s directory-name\n", argv[0]);
 969                 goto endmkdir;
 970         }
 971         VOID command("XMKD %s", argv[1]);
 972 endmkdir:
 973         if( madeargs )
 974                 xdealglob( argv );
 975 }
 976
 977 /*
 978  * Remove a directory.
 979  */
 980 removedir(argc, argv)
 981         char *argv[];
 982 {
 983         int madeargs = 0;
 984
 985         if (argc < 2) {
 986                 xstrcat(line, " ");
 987                 xoprintf(xstdout,"(directory-name) ");
 988                 xfflush( xstdout );
 989                 xgets(&line[xstrlen(line)]);
 990                 argv = xmkarglist( line, &argc );
 991                 madeargs = 1;
 992         }
 993         if (argc < 2) {
 994                 xoprintf(xstdout,"%s directory-name\n", argv[0]);
 995                 goto endrmdir;
 996         }
 997         VOID command("XRMD %s", argv[1]);
 998 endrmdir:
 999         if( madeargs )
1000                 xdealglob( argv );
1001 }
1002
1003 /*
1004  * Send a line, verbatim, to the remote machine.
1005  */
1006 quote(argc, argv)
1007         char *argv[];
1008 {
```

```
1009            int i;
1010            char buf[BUFSIZ];
1011            int madeargs = 0;
1012
1013            if (argc < 2) {
1014                    xstrcat(line, " ");
1015                    xoprintf(xstdout,"(command line to send) ");
1016                    xfflush( xstdout );
1017                    xgets(&line[xstrlen(line)]);
1018                    argv = xmkarglist( line, &argc );
1019                    madeargs = 1;
1020            }
1021            if (argc < 2) {
1022                    xoprintf(xstdout,"usage: %s line-to-send\n", argv[0]);
1023                    goto endquote;
1024            }
1025            xstrcpy(buf, argv[1]);
1026            for (i = 2; i < argc; i++) {
1027                    xstrcat(buf, " ");
1028                    xstrcat(buf, argv[i]);
1029            }
1030            VOID command(buf);
1031    endquote:
1032            if( madeargs )
1033                    xdealglob( argv );
1034    }
1035
1036    /*
1037     * Ask the other side for help.
1038     */
1039    rmthelp(argc, argv)
1040            char *argv[];
1041    {
1042            int oldverbose = verbose;
1043
1044            verbose = 1;
1045            VOID command(argc == 1 ? "HELP" : "HELP %s", argv[1]);
1046            verbose = oldverbose;
1047    }
1048
1049    /*
1050     * Terminate session and exit.
1051     */
1052    /*VARARGS*/
1053    quit()
1054    {
1055
1056            if (connected)
1057                    disconnect();
1058            xexit(0);
1059    }
1060
1061    /*
1062     * Terminate session, but don't exit.
1063     */
1064    disconnect()
```

```
1065  {
1066          extern XFILE *cout;
1067          extern XFILE *cin;
1068          extern int data;
1069
1070          if (!connected)
1071                  return;
1072          VOID command("QUIT");
1073          VOID xclose( xfileno(cout));
1074          VOID xclose( xfileno(cin));
1075          cout = XNULL;
1076          cin = XNULL;
1077          connected = 0;
1078          data = -1;
1079  }
1080
1081  confirm(cmd, file)
1082          char *cmd, *file;
1083  {
1084          char line[BUFSIZ];
1085
1086          if ((!interactive) || (!fromatty))
1087                  return (1);
1088          xoprintf(xstdout,"%s %s ?(n==don't,a==do all,q==do no more,y==do)? ",
1089                  cmd, file);
1090          xfflush(xstdout);
1091          xgets(line);
1092          switch ( *line ) {
1093                  case 'n':
1094                  case 'N':
1095                          return( 0 );
1096                  case 'y':
1097                  case 'Y':
1098                          return( 'y' );
1099                  case 'A':
1100                  case 'a':
1101                          return( 'a' );
1102                  case 'Q':
1103                  case 'q':
1104                          return( 'q' );
1105                  default:
1106                          break;
1107                  }
1108          return(1);
1109  }
1110
1111  fatal(msg)
1112          char *msg;
1113  {
1114
1115          xoprintf(xstderr, "ftp: %s\n");
1116          xexit(1);
1117  }
1118
1119  /*
1120   * Glob a local file name specification with
```

```
1121    * the expectation of a single return value.
1122    * Can't control multiple values being expanded
1123    * from the expression, we return only the first.
1124    */
1125   static
1126   globulize(cpp)
1127           char **cpp;
1128   {
1129           char **globbed;
1130           char *argv[2];
1131
1132           if (!doglob)
1133                   return (1);
1134           argv[0] = *cpp;
1135           argv[1] = (char *)0;
1136           globbed = xglob( argv );
1137           if (globerr != XNULL) {
1138                   xoprintf(xstdout,"%s: %s\n", *cpp, globerr);
1139                   if (globbed)
1140                           xdealglob(globbed);
1141                   return (0);
1142           }
1143           if (globbed) {
1144                   *cpp = *globbed;
1145                   /* don't waste too much memory */
1146                   glizept = globbed;
1147           }
1148           else
1149                   return( 0 );
1150           return (1);
1151   }
1152
1153   lls( argc, argv )
1154
1155   int argc;
1156   char *argv[];
1157   {
1158   if( argc > 1 )
1159           llist( argc, argv, LS_ARG );
1160   else
1161           llist( argc, argv, LS );
1162   }
1163
1164   ldir( argc, argv )
1165
1166   int argc;
1167   char *argv[];
1168   {
1169   if( argc > 1 )
1170           llist( argc, argv, LSLONG_ARG );
1171   else
1172           llist( argc, argv, LSLONG );
1173   }
1174
1175   llist( argc, argv, func_code )
1176
```

```
1177  int argc;
1178  char *argv[];
1179  int func_code;
1180  {
1181  char **argv1;
1182  char **argv2;
1183  char *pt;
1184
1185  if( argc > 1 )
1186          {
1187          if ( doglob ) {
1188                  argv1 = xglob( &argv[1] );
1189                  if( argv1 == XNULL || globerr ) {
1190                          xoprintf( xstdout, "No file name matches." );
1191                          if ( argv1 != XNULL )
1192                                  xdealglob( argv1 );
1193                          return;
1194                  }
1195          } else {
1196                  argv1 = &argv[1];
1197          }
1198          argv2 = argv1;
1199          for ( pt = *argv2++ ; pt ; pt = *argv2++ )
1200                  {
1201                  xls( xfileno(xstdout), pt, func_code );
1202                  }
1203          if( doglob )
1204                  xdealglob( argv1 );
1205          }
1206  else
1207          {
1208          xls( xfileno(xstdout), (char *)0, func_code );
1209          }
1210  }
1211
1212  lpwd( argc, argv )
1213
1214  int argc;
1215  char *argv[];
1216  {
1217  char buf[MAXPATHLEN + 1];
1218  int success;
1219
1220  success = xpwd( buf, sizeof( buf ), PWD );
1221  if( !success )
1222          {
1223          xoprintf( xstdout, "local current directory is: %s\n", buf );
1224          }
1225  else
1226          {
1227          xoprintf( xstdout,
1228                  "current directory unknown %s\n", xrerror( success ) );
1229          }
1230  }
```

```
 1  #ifndef lint
 2  static char sccsid[] =
 3   " @(#)cmdtab.c 1.9 6/3/85";
 4  #endif
 5
 6  #include "ftpc.h"
 7
 8  /*
 9   * User FTP -- Command Tables.
10   */
11  int     setascii(), setbell(), setbinary(), setdebug(), setform();
12  int     setglob(), sethash(), setmode(), setpeer(), setport ();
13  int     setprompt(), setstruct();
14  int     settenex(), settrace(), settype(), setverbose();
15  int     disconnect();
16  int     cd(), lcd(), delete(), mdelete(), user();
17  int     ls(), mls(), get(), mget(), help(), put(), mput();
18  int     quit(), renamefile(), status();
19  int     quote(), rmthelp(), shell();
20  int     pwd(), makedir(), removedir();
21  int     lls(), lpwd(), ldir();
22
23  char    appendhelp[] =  "append to a file";
24  char    asciihelp[] =   "set ascii transfer type";
25  char    beephelp[] =    "beep when command completed";
26  char    binaryhelp[] =  "set binary transfer type";
27  char    cdhelp[] =      "change remote working directory";
28  char    connecthelp[] = "connect to remote tftp";
29  char    deletehelp[] =  "delete remote file";
30  char    debughelp[] =   "toggle/set debugging mode";
31  char    dirhelp[] =     "list contents of remote directory";
32  char    disconhelp[] =  "terminate ftp session";
33  char    formhelp[] =    "set file transfer format";
34  char    globhelp[] =    "toggle metacharacter expansion of local file names";
35  char    hashhelp[] =    "toggle printing '#' for each buffer transferred";
36  char    helphelp[] =    "print local help information";
37  char    lcdhelp[] =     "change local working directory";
38  char    lshelp[] =      "nlist contents of remote directory";
39  char    mdeletehelp[] = "delete multiple files";
40  char    mdirhelp[] =    "list contents of multiple remote directories";
41  char    mgethelp[] =    "get multiple files";
42  char    mkdirhelp[] =   "make directory on the remote machine";
43  char    mlshelp[] =     "nlist contents of multiple remote directories";
44  char    modehelp[] =    "set file transfer mode";
45  char    mputhelp[] =    "send multiple files";
46  char    porthelp[] =    "toggle use of PORT cmd for each data connection";
47  char    prompthelp[] =  "force interactive prompting on multiple commands";
48  char    pwdhelp[] =     "print working directory on remote machine";
49  char    quithelp[] =    "terminate ftp session and exit";
50  char    quotehelp[] =   "send arbitrary ftp command";
51  char    receivehelp[] = "receive file";
52  char    remotehelp[] =  "get help from remote server";
53  char    renamehelp[] =  "rename file";
54  char    rmdirhelp[] =   "remove directory on the remote machine";
55  char    sendhelp[] =    "send one file";
56  char    shellhelp[] =   "escape to the shell";
```

```
57  char    statushelp[] =  "show current status";
58  char    structhelp[] =  "set file transfer structure";
59  char    tenexhelp[] =   "set tenex file transfer type";
60  char    tracehelp[] =   "toggle packet tracing";
61  char    typehelp[] =    "set file transfer type";
62  char    userhelp[] =    "send new user information";
63  char    llshelp[] =     "list directory on local machine";
64  char    lpwdhelp[] =    "print local current directory(s)";
65  char    ldirhelp[] =    "long listing of local directory(s)";
66  char    verbosehelp[] = "toggle verbose mode";
67
68  struct cmd cmdtab[] = {
69          { "!",          shellhelp,      0,      0,      shell },
70          { "append",     appendhelp,     1,      1,      put },
71          { "ascii",      asciihelp,      0,      1,      setascii },
72          { "bell",       beephelp,       0,      0,      setbell },
73          { "binary",     binaryhelp,     0,      1,      setbinary },
74          { "bye",        quithelp,       0,      0,      quit },
75          { "cd",         cdhelp,         0,      1,      cd },
76          { "close",      disconhelp,     0,      1,      disconnect },
77          { "delete",     deletehelp,     0,      1,      delete },
78          { "debug",      debughelp,      0,      0,      setdebug },
79          { "dir",        dirhelp,        1,      1,      ls },
80          { "form",       formhelp,       0,      1,      setform },
81          { "get",        receivehelp,    1,      1,      get },
82          { "glob",       globhelp,       0,      0,      setglob },
83          { "hash",       hashhelp,       0,      0,      sethash },
84          { "help",       helphelp,       0,      0,      help },
85          { "lcd",        lcdhelp,        0,      0,      lcd },
86          { "ls",         lshelp,         1,      1,      ls },
87          { "mdelete",    mdeletehelp,    1,      1,      mdelete },
88          { "mdir",       mdirhelp,       1,      1,      mls },
89          { "mget",       mgethelp,       1,      1,      mget },
90          { "mkdir",      mkdirhelp,      0,      1,      makedir },
91          { "mls",        mlshelp,        1,      1,      mls },
92          { "mode",       modehelp,       0,      1,      setmode },
93          { "mput",       mputhelp,       1,      1,      mput },
94          { "open",       connecthelp,    0,      0,      setpeer },
95          { "prompt",     promptthelp,    0,      0,      setprompt },
96          { "sendport",   porthelp,       0,      0,      setport },
97          { "put",        sendhelp,       1,      1,      put },
98          { "pwd",        pwdhelp,        0,      1,      pwd },
99          { "quit",       quithelp,       0,      0,      quit },
100         { "quote",      quotehelp,      1,      1,      quote },
101         { "recv",       receivehelp,    1,      1,      get },
102         { "remotehelp", remotehelp,     0,      1,      rmthelp },
103         { "rename",     renamehelp,     0,      1,      renamefile },
104         { "rmdir",      rmdirhelp,      0,      1,      removedir },
105         { "send",       sendhelp,       1,      1,      put },
106         { "status",     statushelp,     0,      0,      status },
107         { "struct",     structhelp,     0,      1,      setstruct },
108         { "tenex",      tenexhelp,      0,      1,      settenex },
109         { "trace",      tracehelp,      0,      0,      settrace },
110         { "type",       typehelp,       0,      1,      settype },
111         { "user",       userhelp,       0,      1,      user },
112         { "verbose",    verbosehelp,    0,      0,      setverbose },
```

```
113             { "ldir",       ldirhelp,       0,      0,      ldir },
114             { "lls",        llshelp,        0,      0,      lls },
115             { "lpwd",       lpwdhelp,       0,      0,      lpwd },
116             { "?",          helphelp,       0,      0,      help },
117             { 0 },
118     };
119
120     int     NCMDS = (sizeof (cmdtab) / sizeof (cmdtab[0])) - 1;
```

```
 1  #ifndef lint
 2  static char sccsid[] = " @(#)ftp.c      1.20 6/21/85";
 3  #endif
 4
 5  #include "ftpc.h"
 6  struct  sckadr_in hisctladdr = { AF_INET };
 7  struct  sckadr_in data_addr = { AF_INET };
 8  int     data = -1;
 9  struct  sckadr_in myctladdr = { AF_INET };
10  /*
11   * Options and other state info.
12   */
13  int     trace = 0;              /* trace packets exchanged */
14  int     hash = 0;               /* print # for each buffer transferred */
15  int     sendport = -1;  /* use PORT cmd for each data connection */
16  int     verbose = 0;    /* print messages coming back from server */
17  int     connected = 0;  /* connected to server */
18  int     fromatty = 0;   /* input is from a terminal */
19  int     interactive = 0;        /* interactively prompt on m* cmds */
20  int     debug = 0;              /* debugging level */
21  int     bell = 0;               /* ring bell on cmd completion */
22  int     doglob = 0;             /* glob local file names */
23  int     autologin = 0;  /* establish user account on connection */
24
25  char    typename[32] = {0};     /* name of file transfer type */
26  int     type = 0;               /* file transfer type */
27  char    structname[32] = {0};   /* name of file transfer structure */
28  int     stru = 0;               /* file transfer structure */
29  char    formname[32] = {0};     /* name of file transfer format */
30  int     form = 0;               /* file transfer format */
31  char    modename[32] = {0};     /* name of file transfer mode */
32  int     mode = 0;               /* file transfer mode */
33  char    bytename[32] = {0};     /* local byte size in ascii */
34  int     bytesize = 0;   /* local byte size in binary */
35
36  char    *hostname = (char*)0;   /* name of host connected to */
37
38  struct  servent *sp = 0;        /* service spec for tcp/ftp */
39  char    line[200] = {0};        /* input line buffer */
40  int     margc = 0;              /* count of arguments on input line */
41  char    **margv = (char **)0;   /* args parsed from input line */
42
43  int     options = 0;    /* used during socket creation */
44
45  extern char *xgetpass();
46  extern long xpasstnet(), xpassfnet();
47  extern long xtime();
48  extern XFILE *xodopen();
49
50  #define SWAITMAX        90      /* wait at most 90 seconds */
51  #define SWAITINT        5       /* interval between retries */
52
53  int     swaitmax = SWAITMAX;
54  int     swaitint = SWAITINT;
55
56  XFILE   *cin = XNULL, *cout = XNULL;
```

```
 57  XFILE   *dataconn();
 58
 59  struct hostent *
 60  hookup(host, port)
 61          char *host;
 62          int port;
 63  {
 64          register struct hostent *hp;
 65          int s, len;
 66          int rval;
 67
 68          bzero((char *)&hisctladdr, sizeof (hisctladdr));
 69          hp = ghbname(host);
 70          if (hp == XNULL) {
 71                  static struct hostent def;
 72                  static struct in_addr defaddr;
 73                  static char namebuf[128];
 74                  int inet_addr();
 75
 76                  defaddr.s_addr = inet_addr(host);
 77                  if (defaddr.s_addr == -1) {
 78                          xoprintf(xstderr, "%s: Unknown host.\n", host);
 79                          return (0);
 80                  }
 81                  xstrcpy(namebuf, host);
 82                  def.h_name = namebuf;
 83                  hostname = namebuf;
 84                  def.h_addr = (char *)&defaddr;
 85                  def.h_length = sizeof (struct in_addr);
 86                  def.h_addrtype = AF_INET;
 87                  def.h_aliases = 0;
 88                  hp = &def;
 89          }
 90          hostname = hp->h_name;
 91          hisctladdr.sin_family = hp->h_addrtype;
 92          s = xsocket(SOCK_STREAM, (struct sockproto *)0,
 93                  (struct sockaddr *)0, SO_KEEPALIVE);
 94          if (s < 0) {
 95                  xperror(s,"ftp: socket");
 96                  return (0);
 97          }
 98          bcopy(hp->h_addr, (char *)&hisctladdr.sin_addr, hp->h_length);
 99          hisctladdr.sin_port = port;
100          if (( rval = xconnect(s,(char *)&hisctladdr)) < 0){
101                  xperror(rval, "ftp: connect");
102                  goto bad;
103          }
104          len = sizeof (myctladdr);
105          if ((rval = xsktaddr(s, (char *)&myctladdr)) < 0) {
106                  xperror(rval,"ftp: getsockname");
107                  goto bad;
108          }
109          xdup2( s, ( rval = xnewod() ) );
110          cin = xodopen(s, "r");
111          cout = xodopen(rval, "w");
112          if (cin == XNULL || cout == XNULL) {
```

```
113                 xoprintf(xstderr, "ftp: fdopen failed.\n");
114                 if (cin)
115                         xclose(xfileno(cin));
116                 if (cout)
117                         xclose(xfileno(cout));
118                 goto bad;
119         }
120         if (verbose)
121                 xoprintf(xstdout,"Connected to %s.\n", hp->h_name);
122         VOID getreply(0);               /* read startup message from server */
123         return (hp);
124 bad:
125         xclose(s);
126         return ((struct hostent *)0);
127 }
128
129 /*
130 For now, non-interactive use of ftp requires explicate USER and
131 PASS commands.
132 Later, we can define an autologin procedure that will work for all
133 systems.
134 */
135 login(hp)
136         struct hostent *hp;
137 {
138         char acct[80];
139         char *user, *pass;
140         int n;
141
142         if( !fromatty )
143                 return( 0 );
144         user = acct;
145         xoprintf(xstdout,"Remote User Name:"); VOID xfflush(xstdout);
146         if( xogets(user, sizeof(acct) - 1, xstdin) == XNULL ) {
147                 if( xfeof( xstdin ) ) {
148                         xprintf( "\n" );
149                         quit();
150                         xexit( 0 );
151                 } else {
152                         return( 0 );
153                 }
154         }
155         if( xstrlen(acct) - 1 <= 0 )
156                 return( 0 );
157         user[xstrlen(acct) - 1] = '\0';
158         n = command("USER %s", user);
159         if (n == CONTINUE)
160                 {
161                 pass = xgetpass( "Remote Password:" );
162                 xputchar( '\n' );
163                 n = command("PASS %s", pass);
164                 if (n == CONTINUE)
165                         {
166                         xoprintf(xstdout,"Remote Account: ");
167                         VOID xfflush(xstdout);
168                         VOID xogets(acct, sizeof(acct) - 1, xstdin);
```

```
169                         acct[xstrlen(acct) - 1] = '\0';
170                         n = command("ACCT %s", acct);
171                         }
172                 }
173         if (n != COMPLETE) {
174                 xoprintf(xstderr, "Login failed.\n");
175                 return (0);
176         }
177         return (1);
178 }
179
180 /*VARARGS 1*/
181 #ifdef zilog
182 /*
183  * Pick parameters from registers, and put them in an honest-looking
184  * stack frame.
185  */
186 command(fmt, a1, a2, a3, a4, a5, a6)
187         char *fmt;
188         int a1, a2, a3, a4, a5, a6;
189 {
190         int args=a1, aa2=a2, aa3=a3, aa4=a4, aa5=a5, aa6=a6;
191 #else
192 command(fmt, args)
193         char *fmt;
194 {
195 #endif
196         int how;                /* something for xioctl args to point to */
197
198         if (debug) {
199                 xoprintf(xstdout,"---> ");
200                 _mydoprnt(fmt, &args, xstdout);
201                 xoprintf(xstdout,"\n");
202                 VOID xfflush(xstdout);
203         }
204         if (cout == XNULL) {
205                 xperror (0, "No control connection for command");
206                 return (0);
207         }
208         if( !empty( cin ) ) {
209                 /*
210                  * Since we are sending a new command,  it is expected
211                  * that all replies to previous commands have been
212                  * processed.  Thus, if there is any data in the command
213                  * stream, we are out of sync with the server, and
214                  * the data that is now present should be flushed.
215                  */
216                 how = 1;
217                 xioctl( xfileno( cout ), FIONBIO, &how );
218                 if( verbose )
219                         xoprintf( xstderr, "Old reply in command stream:\n" );
220                 VOID getreply( 0 );
221                 how = 0;
222                 xioctl( xfileno( cout ), FIONBIO, &how );
223         }
224         _mydoprnt(fmt, &args, cout);
```

```
225             xoprintf(cout, "\r\n");
226             VOID xfflush(cout);
227             return (getreply(!xstrcmp(fmt, "QUIT")));
228     }
229
230     empty(f)
231             XFILE *f;
232     {
233             long mask;
234             int rval;
235
236             if( f->_cnt > 0 )
237                     return( 0 );
238     #ifndef NOSELECT
239             mask = ( 1 << (xfileno( f )));
240             rval = xselect( 20, &mask, (long *)0, 0L );
241             return ( mask == 0 );
242     #else
243             return( 1 );
244     #endif NOSELECT
245     }
246
247
248
249     getreply(expecteof)
250             int expecteof;
251     {
252             register int c, n;
253             register int code, dig;
254             int originalcode = 0, continuation = 0;
255
256             for (;;) {
257                     dig = n = code = 0;
258                     while ((c = xgetc(cin)) != '\n') {
259                             dig++;
260                             if (c == XEOF) {
261                                     if( xfeof(cin) ) {
262                                             if (expecteof)
263                                                     return (0);
264                                             xoprintf(xstdout, "lost connection.\n");
265                                             return( 5 );
266                                     } else {
267                                             xoprintf(xstdout, "error on read.\n" );
268                                             return( n - '0' );
269                                     }
270                             }
271                             if (verbose && c != '\r' ||
272                                 (n == '5' && dig > 4))
273                                     xputchar(c);
274                             if (dig < 4 && isdigit(c))
275                                     code = code * 10 + (c - '0');
276                             if (dig == 4 && c == '-')
277                                     continuation++;
278                             if (n == 0)
279                                     n = c;
280                     }
```

```
281                 if (verbose || n == '5') {
282                         xputchar(c);
283                         VOID xfflush (xstdout);
284                 }
285                 if (continuation && code != originalcode) {
286                         if (originalcode == 0)
287                                 originalcode = code;
288                         continue;
289                 }
290                 if ( !continuation || (code == originalcode) )
291                         return (n - '0');
292         }
293 }
294
295
296 sendrequest(cmd, local, remote)
297         char *cmd, *local, *remote;
298 {
299         int (*closefunc)();
300         long bytes = 0, hashbytes = 1024;
301         long start, stop;
302         int read_reply = 0;
303         int inod;
304         XFILE *inopt, *outopt;
305         int omode;
306         struct ftp_attr attributes;
307
308         closefunc = XNULL;
309         if (xstrcmp(local, "-") == 0) {
310                 inopt = xstdin;
311         } else {
312                 omode = XFREAD;
313                 attributes.rep_type = type;
314                 attributes.format = form;
315                 attributes.structure = stru;
316                 attributes.trans_mode = mode;
317                 attributes.byte_sz = bytesize;
318                 inod = xftpopen( local, omode, FILE_NAME, &attributes );
319                 if( inod < 0 )
320                         {
321                         xperror( inod, local );
322                         goto bad;
323                         }
324                 inopt = xodopen( inod, "r" );
325                 if (inopt == XNULL) {
326                         xoprintf(xstderr, "xodopen failed\n" );
327                         xclose( inod );
328                         goto bad;
329                 }
330                 closefunc = xclose;
331         }
332         if (initconn())
333                 goto bad;
334         read_reply = 1;
335         if (remote) {
336                 if (command("%s %s", cmd, remote) != PRELIM) {
```

```
337                             --read_reply;
338                             goto bad;
339                     }
340             } else
341                     if (command("%s", cmd) != PRELIM) {
342                             --read_reply;
343                             goto bad;
344                     }
345             outopt = dataconn("w");
346             if (outopt == XNULL)
347                     goto bad;
348             start = xtime();
349             bytes = xpasstnet( inopt, outopt );
350             stop = xtime();
351             if( closefunc != XNULL )
352                     xclose( inod );
353             xclose( xfileno(outopt) );
354             data = -1;
355             if( bytes < 0 )
356                     {
357                     xperror( (int)bytes, "local" );
358                     }
359             VOID getreply(0);
360     done:
361             if (bytes > 0 && verbose)
362                     ptransfer("sent", bytes, &start, &stop);
363             return;
364     bad:
365             if (bytes > 0 && verbose)
366                     stop = xtime();
367             if (data >= 0)
368                     VOID xclose(data), data = -1;
369             if (closefunc != XNULL && inopt != XNULL)
370                     xclose( inod );
371             if (read_reply == 1)
372                     VOID getreply(0);
373             goto done;
374     }
375
376     recvrequest(cmd, local, remote, append )
377             char *cmd, *local, *remote, *append;
378     {
379             int (*closefunc)();
380             long bytes = 0, hashbytes = 1024;
381             long start, stop;
382             int read_reply = 0;
383             int inod, outod;
384             XFILE *inopt, *outopt;
385             int omode;
386             struct ftp_attr attributes;
387
388             closefunc = XNULL;
389             if (initconn())
390                     goto bad;
391             read_reply = 1;
392             if (remote) {
```

```
393                     int x;
394                     if ((x = command("%s %s", cmd, remote)) != PRELIM) {
395     /* fprintf(stderr, "bad return from command(%s %s) = %d\n", cmd, remote, x); */
396                             --read_reply;
397                             goto bad;
398                     }
399             } else {
400                     int x;
401                     if ((x = command("%s", cmd)) != PRELIM) {
402     /* fprintf(stderr, "bad return from command(%s) = %d\n", cmd, x); */
403                             --read_reply;
404                             goto bad;
405                     }
406             }
407             if (xstrcmp(local, "-") == 0) {
408                     outopt = xstdout;
409             } else {
410                     omode = XFWRITE | XFCREAT |
411                             (( *append == 'a' )? XFAPPEND : XFTRUNC );
412                     attributes.rep_type = type;
413                     attributes.format = form;
414                     attributes.structure = stru;
415                     attributes.trans_mode = mode;
416                     attributes.byte_sz = bytesize;
417                     outod = xftpopen(local, omode, FILE_NAME, &attributes );
418                     if( outod < 0 )
419                             {
420                             xperror( outod, local );
421                             goto bad;
422                             }
423                     outopt = xodopen( outod, "w" );
424                     if( outopt == XNULL )
425                             {
426                             xoprintf(xstderr, "xodopen failed\n" );
427                             xclose( outod );
428                             goto bad;
429                             }
430                     closefunc = xclose;
431             }
432             inopt = dataconn("r");
433             if (inopt == XNULL)
434                     goto bad;
435             start = xtime();
436             bytes = xpassfnet( inopt, outopt );
437             stop = xtime();
438             xclose( xfileno(inopt) );
439             data = -1;
440             if( closefunc != XNULL )
441                     xclose( outod );
442             if( bytes < 0 )
443                     {
444                     xperror( (int)bytes, "local" );
445                     }
446             VOID getreply(0);
447     done:
448             if (bytes > 0 && verbose)
```

```
449                 ptransfer("received", bytes, &start, &stop);
450         return;
451 bad:
452         if (bytes > 0 && verbose)
453                 stop = xtime();
454         if (data >= 0)
455                 VOID xclose(data), data = -1;
456         if (closefunc != XNULL && outopt != XNULL)
457                 xclose( outod );
458         if (read_reply == 1)
459                 VOID getreply(0);
460         goto done;
461 }
462
463 /*
464  * Need to start a listen on the data channel
465  * before we send the command, otherwise the
466  * server's connect may fail.
467  */
468
469 initconn()
470 {
471         register char *p, *a;
472         int result, len;
473         int options = SO_KEEPALIVE | SO_ACCEPTCONN;
474         int retry;
475         int rval;
476
477 noport:
478         /*
479         data_addr = myctladdr;
480         */
481         bcopy(&myctladdr, &data_addr, sizeof (struct sckadr_in));
482         if (sendport)
483                 data_addr.sin_port = 0; /* let system pick one */
484         if (data != -1)
485                 VOID xclose (data);
486         for (retry = 0; retry < swaitmax; xsleep (swaitint), retry += swaitint)
487                 {
488                 data = xsocket(SOCK_STREAM, (struct sockproto *)0,
489                         &data_addr, options);
490                 if (data >= 0 || (data != XEADDRINUSE && data != XENOBUFS))
491                         break;
492                 }
493         if (data < 0) {
494                 xperror(data, "ftp: socket");
495                 return (1);
496         }
497         len = sizeof (data_addr);
498         if ((rval = xsktaddr(data, (char *)&data_addr)) < 0) {
499                 xperror(rval, "ftp: xsktaddr");
500                 goto bad;
501         }
502         if (sendport) {
503                 a = (char *)&data_addr.sin_addr;
504                 p = (char *)&data_addr.sin_port;
```

```
505  #define UC(b)   (((int)b)&0xff)
506                  result =
507                      command("PORT %d,%d,%d,%d,%d,%d",
508                        UC(a[0]), UC(a[1]), UC(a[2]), UC(a[3]),
509                        UC(p[0]), UC(p[1]));
510                  if (result == ERROR && sendport == -1) {
511                          sendport = 0;
512                          goto noport;
513                  }
514                  return (result != COMPLETE);
515          }
516          return (0);
517  bad:
518          VOID xclose(data), data = -1;
519          return (1);
520  }
521
522  XFILE *
523  dataconn(mode)
524          char *mode;
525  {
526          struct sckadr_in from;
527          int s, fromlen = sizeof (from);
528
529          s = xaccept(data, &from);
530          if (s < 0) {
531                  xperror(s, "ftp: accept");
532                  VOID xclose(data), data = -1;
533                  return (XNULL);
534          }
535          return (xodopen(data, mode));
536  }
537
538  ptransfer(direction, bytes, t0, t1)
539          char *direction;
540          long bytes;
541          long *t0, *t1;
542  {
543          long sec;
544
545          sec = *t1 - *t0;
546          if (sec <= 0)
547              sec = 1;
548          xoprintf(xstdout,"%ld bytes %s in %ld seconds (%ld bytes/s)\n",
549                  bytes, direction, sec,  bytes / sec );
550  }
551
552  /*
553  Routines from here on are to use names introduced by 4.2 BSD.
554  */
555
556  shutdown (fd, how)
557  int fd, how;
558  {
559          xioctl (fd, SIOCDONE, &how);
560  }
```

```
561
562  /*
563  mp  -  Even if _doprnt is more wonderful than _mydoprnt for systems which
564         have _doprnt, using _doprnt is an incredible maintainance headache.
565         In any case, we should support the same functionality on all
566         systems.
567         Hence, may _doprnt rest in peace.
568  */
569  _mydoprnt(format, argp, FILEp)
570  char *format;
571  int *argp;
572  XFILE *FILEp;
573  {
574          xoprintf(FILEp, format, *argp, *(argp+1), *(argp+2), *(argp+3),
575                  *(argp+4), *(argp+5));
576  }
577
578  bzero (what, size)
579  register char *what;
580  register int size;
581  {
582          while (size-- > 0)
583                  *what++ = 0;
584  }
585
586  bcopy (from, to, size)
587  register char *from, *to;
588  register int size;
589  {
590          while (size-- > 0)
591                  *to++ = *from++;
592  }
593
594  bcmp (left, right, size)
595  register char *left, *right;
596  register int size;
597  {
598          while (size-- > 0)
599                  if (*left++ != *right++)
600                          if (0xff&(*--left) > 0xff&(*--right))
601                                  return (1);
602                          else
603                                  return (-1);
604          return (0);
605  }
606
607
608  struct servent * gsbname (service, proto)
609   char *service, *proto;
610  {
611          static struct servent servstat;
612
613          servstat.s_name = service;
614          servstat.s_aliases = 0;
615          if (xstrcmp (service, "ftp") == 0)
616                  servstat.s_port = (IPRT_FTP);
```

```
617             else
618             if (xstrcmp (service, "telnet") == 0)
619                     servstat.s_port = (IPPORT_TELNET);
620             else
621                     return (0);
622             servstat.s_proto = proto;
623             return (&servstat);
624     }
625
626     struct hostent *
627     ghbname(host)
628     char *host;
629     {
630             static struct hostent def;
631             static struct in_addr defaddr;
632             static char namebuf[128];
633             extern long xrhost ();
634
635             defaddr.s_addr = xrhost(&host);
636             if (defaddr.s_addr == -1)
637                     return (0);
638             xstrcpy(namebuf, host);
639             def.h_name = namebuf;
640             def.h_addr = (char *)&defaddr;
641             def.h_length = sizeof (struct in_addr);
642             def.h_addrtype = AF_INET;
643             def.h_aliases = 0;
644             return (&def);
645     }
646
647     int inet_addr (host)
648     char *host;
649     {
650             return (-1);
651     }
```

```
 1  /*
 2  %W% %G%
 3
 4  Operating system specific initialization for ftp client.
 5          ...stuff the doesn't seem worth the effort of providing
 6          general mechanisms for.
 7  */
 8
 9  #include <xgenlib.h>
10  #include <xpwd.h>
11  #include <xspecial.h>
12  typedef int jmp_buf;
13  int errno = 0;
14  int figit = 0;
15  jmp_buf  toplevel = 0;
16  #include <ftpvar.h>
17  jmp_buf *envptr = {0};
18
19  extern int fromatty;    /* ftp started from terminal */
20  extern int _ttyinput;   /* set in xoslib                */
21  extern int conned;      /* true if connected to server */
22  /* extern astlconn(); */
23
24  extern struct passwd *pw ;
25
26  clientinit()
27  {
28
29  fromatty = _ttyinput;            /* true when used interactively */
30  /* set up routine to print out message and halt program  */
31
32   /*
33   emt(SREX,astlconn);
34   */
35   toplevel = (int ) & envptr;
36
37    }
38  /*
39  lostpeer()
40  {
41          extern XFILE *cout;
42          extern int data;
43
44          xoprintf( xstdout, "Lost Connection.\n" );
45          if (conned) {
46                  if (cout != XNULL) {
47                          shutdown(xfileno(cout), 1+1);
48                          xclose(cout);
49                          cout = XNULL;
50                  }
51                  if (data >= 0) {
52                          shutdown(data, 1+1);
53                           xclose(data);
54                          data = -1;
55                  }
56                  conned = 0;
```

```
57          }
58  }
59  */
60  gethbaddr()
61  {
62  }
```

```
 1  #ifndef lint
 2  static char sccsid[] = " @(#)main.c     1.14 8/28/85";
 3  #endif
 4
 5  /*
 6   * FTP User Program -- Command Interface.
 7   */
 8  #include "ftpc.h"
 9
10  int     intr();
11  extern  int data;
12  extern char **xmkarglist();
13  extern char *xstrrchr();
14
15
16  xmain(argc, argv)
17          char *argv[];
18  {
19          /*
20           * Don't use register declarations in this procedure -- Zilog
21           * S8000 setret() (alias setjmp()) can't abide by them.
22           */
23          char *cp;
24          int top;
25
26          margv = (char **)0;
27          sp = gsbname("ftp", "tcp");
28          if (sp == 0) {
29                  xoprintf(xstderr, "ftp: ftp/tcp: unknown service\n");
30                  xexit(1);
31          }
32          doglob = 1;
33          interactive = 1;
34          autologin = 1;
35          argc--, argv++;
36          while (argc > 0 && **argv == '-') {
37                  for (cp = *argv + 1; *cp; cp++)
38                          switch (*cp) {
39
40                          case 'd':
41                                  options |= SO_DEBUG;
42                                  debug++;
43                                  break;
44
45                          case 'v':
46                                  verbose++;
47                                  break;
48
49                          case 't':
50                                  trace++;
51                                  break;
52
53                          case 'i':
54                                  interactive = 0;
55                                  break;
56
```

```
57                          case 'n':
58                                  autologin = 0;
59                                  break;
60
61                          case 'g':
62                                  doglob = 0;
63                                  break;
64
65                          default:
66                                  xoprintf(xstderr,
67                                    "ftp: %c: unknown option\n", *cp);
68                                  xexit(1);
69                          }
70                  argc--, argv++;
71          }
72          /*
73           * Set up defaults for FTP.
74           */
75          xstrcpy(typename, "ascii"), type = TYPE_A;
76          xstrcpy(formname, "non-print"), form = FORM_N;
77          xstrcpy(modename, "stream"), mode = MODE_S;
78          xstrcpy(structname, "file"), stru = STRU_F;
79          xstrcpy(bytename, "8"), bytesize = 8;
80          if (fromatty)
81                  verbose++;
82          else
83                  interactive = 0;        /* not interactive, prompt off*/
84          /*
85           * Set up the home directory in case we're globbing.
86           */
87          if (argc > 0) {
88                  if (xsetjmp(toplevel))
89                          xexit(0);
90                  xint_term( intr );
91                  setpeer(argc + 1, argv - 1);
92          }
93          top = xsetjmp(toplevel);
94          if (top == 0 || top == 1 ) {
95                  xint_term( intr );
96                  top = 1;
97          }
98          for (;;) {
99                  cmdscanner(top);
100                 top = 1;
101         }
102 }
103
104 intr()
105 {
106
107         /*
108         Should send telnet IP, but ...
109         for now just close data connection so that ftp will fall back
110         into command mode.
111         We still have to wait for other side to complete.
112         */
```

```
113             xint_term( intr );
114             if( data != -1 )
115                     {
116                     shutdown( data, 2 );
117                     xclose( data );
118                     data = -1;
119                     xoprintf(xstdout, "data connection broken\n" );
120                     }
121     }
122
123     char *
124     tail(filename)
125             char *filename;
126     {
127             register char *s;
128
129             while (*filename) {
130                     s = xstrrchr(filename, '/');
131                     if (s == XNULL)
132                             break;
133                     if (s[1])
134                             return (s + 1);
135                     *s = '\0';
136             }
137             return (filename);
138     }
139
140     extern struct cmd cmdtab[];
141     extern int help();
142     /*
143      * Command parser.
144      */
145     cmdscanner(top)
146             int top;
147     {
148             register struct cmd *c;
149             struct cmd *getcmd();
150
151             if (!top)
152                     xputchar('\n');
153             for (;;) {
154                     if (fromatty) {
155                             xoprintf(xstdout,"ftp> ");
156                             xfflush(xstdout);
157                     }
158                     if (xgets(line) == XNULL) {
159                             if( xfeof( xstdin ) ) {
160                                     /*
161                                     quit on end of input
162                                     */
163                                     xprintf( "\n" );
164                                     quit();
165                             } else {
166                                     break;
167                             }
168                     }
```

```
169                 if (line[0] == 0)
170                         break;
171                 if( margv )
172                         xdealglob( margv );
173                 margv = xmkarglist( line, &margc );
174                 c = getcmd(margv[0]);
175                 if (c == (struct cmd *)-1) {
176                         xoprintf(xstdout,"?Ambiguous command\n");
177                         continue;
178                 }
179                 if (c == 0) {
180                         xoprintf(xstdout,"?Invalid command\n");
181                         continue;
182                 }
183                 if (c->c_conn && !connected) {
184                         xoprintf(xstdout,"Not connected.\n");
185                         continue;
186                 }
187                 (*c->c_handler)(margc, margv);
188                 if (bell && c->c_bell)
189                         xputchar(CTRL('G'));
190                 if (c->c_handler != help)
191                         break;
192         }
193         xlongjmp(toplevel, 0);
194 }
195
196 struct cmd *
197 getcmd(name)
198         register char *name;
199 {
200         register char *p, *q;
201         register struct cmd *c, *found;
202         register int nmatches, longest;
203
204         /*
205         convert command to lower case.
206         */
207         for( q = name ; *q ; ++q ) {
208                 if( isupper( *q ) )
209                         *q = _tolower( *q );
210         }
211         longest = 0;
212         nmatches = 0;
213         found = 0;
214         for (c = cmdtab; p = c->c_name; c++) {
215                 for (q = name; *q == *p++; q++)
216                         if (*q == 0)            /* exact match? */
217                                 return (c);
218                 if (!*q) {                      /* the name was a prefix */
219                         if (q - name > longest) {
220                                 longest = q - name;
221                                 nmatches = 1;
222                                 found = c;
223                         } else if (q - name == longest)
224                                 nmatches++;
```

```
225                     }
226             }
227             if (nmatches > 1)
228                     return ((struct cmd *)-1);
229             return (found);
230     }
231
232
233     #define HELPINDENT (sizeof ("directory"))
234
235     extern int NCMDS;
236     /*
237      * Help command.
238      * Call each command handler with argc == 0 and argv[0] == name.
239      */
240     help(argc, argv)
241             int argc;
242             char *argv[];
243     {
244             register struct cmd *c;
245
246             if (argc == 1) {
247                     register int i, j, w;
248                     int columns, width = 0, lines;
249
250                     xoprintf(xstdout,
251                             "Commands may be abbreviated.  Commands are:\n\n");
252                     for (c = cmdtab; c < &cmdtab[NCMDS]; c++) {
253                             int len = xstrlen(c->c_name);
254
255                             if (len > width)
256                                     width = len;
257                     }
258                     width = (width + 8) &~ 7;
259                     columns = 80 / width;
260                     if (columns == 0)
261                             columns = 1;
262                     lines = (NCMDS + columns - 1) / columns;
263                     for (i = 0; i < lines; i++) {
264                             for (j = 0; j < columns; j++) {
265                                     c = cmdtab + j * lines + i;
266                                     xoprintf(xstdout,"%s", c->c_name);
267                                     if (c + lines >= &cmdtab[NCMDS]) {
268                                             xoprintf(xstdout,"\n");
269                                             break;
270                                     }
271                                     w = xstrlen(c->c_name);
272                                     while (w < width) {
273                                             w = (w + 8) &~ 7;
274                                             xputchar('\t');
275                                     }
276                             }
277                     }
278                     return;
279             }
280             while (--argc > 0) {
```

```
281                 register char *arg;
282                 arg = *++argv;
283                 c = getcmd(arg);
284                 if (c == (struct cmd *)-1)
285                         xoprintf(xstdout,"?Ambiguous help command %s\n", arg);
286                 else if (c == (struct cmd *)0)
287                         xoprintf(xstdout,"?Invalid help command %s\n", arg);
288                 else
289                         xoprintf(xstdout,"%-*s\t%s\n", HELPINDENT,
290                                 c->c_name, c->c_help);
291         }
292 }
293
294 /*
295  * Call routine with argc, argv set from args (terminated by 0).
296  */
297 /* VARARGS2 */
298 call(routine, args)
299         int (*routine)();
300         int args;
301 {
302         register int *argp;
303         register int argc;
304
305         for (argc = 0, argp = &args; *argp++ != 0; argc++)
306                 ;
307         (*routine)(argc, &args);
308 }
```

```
 1          .title  dummy
 2  getenv::
 3  gethen::
 4  getnba::
 5  getnbn::
 6  getnen::
 7  getpen::
 8  getsbp::
 9  getsen::
10  gpbnam::
11  gpbnum::
12          rts     pc
13          .end
```

```
 1
 2  /* "@(#)compat.h        1.9 4/15/85" */
 3
 4  /* added by billn */
 5  /* #include <exos/misc.h> */
 6  #ifdef index /* system 3 or 5 */
 7  #include <fcntl.h>
 8  #define dup2(f,n) { close(n); fcntl(f, F_DUPFD, n);}
 9  #endif
10  #ifndef void
11  #define void int
12  #endif
13
14  #define VOID (void)
15
16  #ifndef SIGCHLD
17  #define SIGCHLD SIGCLD
18  #endif
19  /* end billn */
20
21  #ifndef MAXPATHLEN
22  #define MAXPATHLEN 33
23  #endif
24
25  #define receive_data rec_data
26  #define wait3 wait2
27  #define initgroups(a,b)
28  #define inappropriate_request inapreq
29
30  #ifdef BSD4dot2
31  #else
32  #ifdef V7
33  #include <sys/timeb.h>
34  struct timeval { long tv_sec; long tv_usec; };
35  struct timeb ftimeb;
36  #define gettimeofday(a,b) ( ftime (&ftimeb), \
37  (a)->tv_sec = ftimeb.time, (a)->tv_usec = ftimeb.millitm)
38  #else
39  struct timeval { long tv_sec; long tv_usec; };
40  extern long xtime();
41  #define gettimeofday(a,b) ((a)->tv_sec = time(0), (a)->tv_usec = 0)
42  #endif V7
43  #endif BSD4dot2
44
45  #ifndef CTRL
46  #define CTRL(x) 037&'x'
47  #endif
48
49  #define SOL_SOCKET      0
50  #define SO_REUSEADDR    0
```

```
 1  /*
 2  @(#)ftpd.h      1.2 4/11/85
 3
 4  Header files for generic server code.
 5  */
 6  #include <xgenlib.h>
 7  #include <xpwd.h>
 8  #include <netdb.h>
 9
10  typedef int jmp_buf;
11  #include <ftp.h>
12  #include <in.h>
13  #include <socket.h>
14  #include <xspecial.h>
15  #include "telnet.h"
16  #define off_t long
17  extern int figit ;
18  extern long xpasstnet();
19  extern long xpassfnet();
20  #define VOID figit = (int)
21  #define renamecmd  cmdrename
```

```
 1  /*@(#)telnet.h  1.8 4/11/85*/
 2  /*
 3   * Definitions for the TELNET protocol.
 4   */
 5  #define IAC     255             /* interpret as command: */
 6  #define DONT    254             /* you are not to use option */
 7  #define DO      253             /* please, you use option */
 8  #define WONT    252             /* I won't use option */
 9  #define WILL    251             /* I will use option */
10  #define SB      250             /* interpret as subnegotiation */
11  #define GA      249             /* you may reverse the line */
12  #define EL      248             /* erase the current line */
13  #define EC      247             /* erase the current character */
14  #define AYT     246             /* are you there */
15  #define AO      245             /* abort output--but let prog finish */
16  #define IP      244             /* interrupt process--permanently */
17  #define BREAK   243             /* break */
18  #define DM      242             /* data mark--for connect. cleaning */
19  #define NOP     241             /* nop */
20  #define SE      240             /* end sub negotiation */
21
22  #define SYNCH   242             /* for telfunc calls */
23
24  /* telnet options */
25
26  #define TNPBINARY       0       /* 8-bit data path */
27  #define TNPECHO 1       /* echo */
28  #define TNPRCP  2       /* prepare to reconnect */
29  #define TNPSGA  3       /* suppress go ahead */
30  #define TNPNAMS 4       /* approximate message size */
31  #define TNPSTATUS       5       /* give status */
32  #define TNPTM   6       /* timing mark */
33  #define TNPRCTE 7       /* remote controlled transmission and echo */
34  #define TNPNAOL         8       /* negotiate about output line width */
35  #define TNPNAOP         9       /* negotiate about output page size */
36  #define TNPNAOCRD       10      /* negotiate about CR disposition */
37  #define TNPHTS  11      /* negotiate about horizontal tabstops */
38  #define TNPHTD  12      /* negotiate about horizontal tab disposition */
39  #define TNPNAOFFD       13      /* negotiate about formfeed disposition */
40  #define TNPVTS  14      /* negotiate about vertical tab stops */
41  #define TNPVTD  15      /* negotiate about vertical tab disposition */
42  #define TNPNAOLFD       16      /* negotiate about output LF disposition */
43  #define TNPXASCII       17      /* extended ascic character set */
44  #define TNPLOGOUT       18      /* force logout */
45  #define TNPBM   19      /* byte macro */
46  #define TNPDET  20      /* data entry terminal */
47  #define TNPSUPDUP       21      /* supdup protocol */
48  #define TNPEXOPL        255     /* extended-options-list */
49
50  #ifdef TELCMDS
51  char *telcmds[] = {
52          "SE", "NOP", "DMARK", "BRK", "IP", "AO", "AYT", "EC",
53          "EL", "GA", "SB", "WILL", "WONT", "DO", "DONT", "IAC",
54  };
55  #endif
56
```

```
57  #ifdef TELOPTS
58  char *telopts[] = {
59          "BINARY", "ECHO", "RCP", "SUPPRESS GO AHEAD", "NAME",
60          "STATUS", "TIMING MARK", "RCTE", "NAOL", "NAOP",
61          "NAOCRD", "NAOHTS", "NAOHTD", "NAOFFD", "NAOVTS",
62          "NAOVTD", "NAOLFD", "EXTEND ASCII", "LOGOUT", "BYTE MACRO",
63          "DATA ENTRY TERMINAL", "SUPDUP"
64  };
65  #endif
```

```
 1
 2  # line 7 "ftpcmd.y"
 3
 4  #ifndef lint
 5  static char sccsid[] = "@(#)ftpcmd.y    1.16 8/15/85";
 6  #endif
 7
 8  #define PARSER
 9  #include "ftpd.h"
10
11  /*
12  * MWP: 03/06/85
13  *  Make machines which have different sized ints and pointers happy.
14  *  (at least as far as the parser stack is concerned).
15  ******************************************************************************
16  */
17  typedef char * YYSTDEF;
18  #define YYSTYPE YYSTDEF
19  YYSTYPE copy();
20  /*
21  ******************************************************************************
22  */
23
24  extern  struct sckadr_in data_dest;
25  extern  int logged_in;
26  extern  int guest;
27  extern  int logging;
28  extern  int type;
29  extern  int form;
30  extern  int debug;
31  extern  int timeout;
32  extern  char hostname[];
33  extern  char *globerr;
34  extern  char *xghome;
35  extern  int usedefault;
36  extern  char    **xglob();
37  static char     **globargs = 0;
38  static char     **rnf_glob = 0;
39  static char     *username = 0;
40  static char     *userpass = 0;
41
42  static  int cmd_type = 0;
43  static  int cmd_form = 0;
44  static  int cmd_bytesz = 0;
45
46  char    *xstrchr();
47  # define A 257
48  # define B 258
49  # define C 259
50  # define E 260
51  # define F 261
52  # define I 262
53  # define L 263
54  # define N 264
55  # define P 265
56  # define R 266
```

```
 57  # define S 267
 58  # define T 268
 59  # define SP 269
 60  # define CRLF 270
 61  # define COMMA 271
 62  # define STRING 272
 63  # define NUMBER 273
 64  # define USER 274
 65  # define PASS 275
 66  # define ACCT 276
 67  # define REIN 277
 68  # define QUIT 278
 69  # define PORT 279
 70  # define PASV 280
 71  # define TYPE 281
 72  # define STRU 282
 73  # define MODE 283
 74  # define RETR 284
 75  # define STOR 285
 76  # define APPE 286
 77  # define MLFL 287
 78  # define MAIL 288
 79  # define MSND 289
 80  # define MSOM 290
 81  # define MSAM 291
 82  # define MRSQ 292
 83  # define MRCP 293
 84  # define ALLO 294
 85  # define REST 295
 86  # define RNFR 296
 87  # define RNTO 297
 88  # define ABOR 298
 89  # define DELE 299
 90  # define CWD 300
 91  # define LIST 301
 92  # define NLST 302
 93  # define SITE 303
 94  # define STAT 304
 95  # define HELP 305
 96  # define NOOP 306
 97  # define XMKD 307
 98  # define XRMD 308
 99  # define XPWD 309
100  # define XCUP 310
101  # define LEXERR 311
102  #define yyclearin yychar = -1
103  #define yyerrok yyerrflag = 0
104  extern int yychar;
105  extern short yyerrflag;
106  #ifndef YYMAXDEPTH
107  #define YYMAXDEPTH 150
108  #endif
109  #ifndef YYSTYPE
110  #define YYSTYPE int
111  #endif
112  YYSTYPE yylval = 0, yyval = 0;
```

```
113  # define YYERRCODE 256
114
115  # line 539 "ftpcmd.y"
116
117
118  #ifdef zilog
119  extern ret_buf errcatch;
120  #else
121  extern jmp_buf errcatch;
122  #endif
123
124  #define CMD     0       /* beginning of command */
125  #define ARGS    1       /* expect miscellaneous arguments */
126  #define STR1    2       /* expect SP followed by STRING */
127  #define STR2    3       /* expect STRING */
128  #define OSTR    4       /* optional STRING */
129
130  struct tab {
131          char    *name;
132          short   token;
133          short   state;
134          short   implemented;    /* 1 if command is implemented */
135          char    *help;
136  };
137
138  struct tab cmdtab[] = {         /* In order defined in RFC 765 */
139          { "USER", USER, STR1, 1,        "<sp> username" },
140          { "PASS", PASS, STR1, 1,        "<sp> password" },
141          { "ACCT", ACCT, STR1, 0,        "(specify account)" },
142          { "REIN", REIN, ARGS, 0,        "(reinitialize server state)" },
143          { "QUIT", QUIT, ARGS, 1,        "(terminate service)" },
144          { "PORT", PORT, ARGS, 1,        "<sp> b0, b1, b2, b3, b4" },
145          { "PASV", PASV, ARGS, 0,        "(set server in passive mode)" },
146          { "TYPE", TYPE, ARGS, 1,        "<sp> [ A | E | I | L ]" },
147          { "STRU", STRU, ARGS, 1,        "(specify file structure)" },
148          { "MODE", MODE, ARGS, 1,        "(specify transfer mode)" },
149          { "RETR", RETR, STR1, 1,        "<sp> file-name" },
150          { "STOR", STOR, STR1, 1,        "<sp> file-name" },
151          { "APPE", APPE, STR1, 1,        "<sp> file-name" },
152          { "MLFL", MLFL, OSTR, 0,        "(mail file)" },
153          { "MAIL", MAIL, OSTR, 0,        "(mail to user)" },
154          { "MSND", MSND, OSTR, 0,        "(mail send to terminal)" },
155          { "MSOM", MSOM, OSTR, 0,        "(mail send to terminal or mailbox)" },
156          { "MSAM", MSAM, OSTR, 0,        "(mail send to terminal and mailbox)" },
157          { "MRSQ", MRSQ, OSTR, 0,        "(mail recipient scheme question)" },
158          { "MRCP", MRCP, STR1, 0,        "(mail recipient)" },
159          { "ALLO", ALLO, ARGS, 1,        "allocate storage (vacuously)" },
160          { "REST", REST, STR1, 0,        "(restart command)" },
161          { "RNFR", RNFR, STR1, 1,        "<sp> file-name" },
162          { "RNTO", RNTO, STR1, 1,        "<sp> file-name" },
163          { "ABOR", ABOR, ARGS, 0,        "(abort operation)" },
164          { "DELE", DELE, STR1, 1,        "<sp> file-name" },
165          { "CWD",  CWD,  OSTR, 1,        "[ <sp> directory-name]" },
166          { "XCWD", CWD,  OSTR, 1,        "[ <sp> directory-name ]" },
167          { "LIST", LIST, OSTR, 1,        "[ <sp> path-name ]" },
168          { "NLST", NLST, OSTR, 1,        "[ <sp> path-name ]" },
```

```
169         { "SITE", SITE, STR1, 0,        "(get site parameters)" },
170         { "STAT", STAT, OSTR, 0,        "(get server status)" },
171         { "HELP", HELP, OSTR, 1,        "[ <sp> <string> ]" },
172         { "NOOP", NOOP, ARGS, 1,        "" },
173         { "XMKD", XMKD, STR1, 1,        "<sp> path-name" },
174         { "XRMD", XRMD, STR1, 1,        "<sp> path-name" },
175         { "XPWD", XPWD, ARGS, 1,        "(return current directory)" },
176         { "XCUP", XCUP, ARGS, 1,        "(change to parent directory)" },
177         { XNULL,   0,    0,    0,       0 }
178 };
179
180 struct tab *
181 lookup(cmd)
182         char *cmd;
183 {
184         register struct tab *p;
185
186         for (p = cmdtab; p->name != XNULL; p++)
187                 if (xstrcmp(cmd, p->name) == 0)
188                         return (p);
189         return (0);
190 }
191
192
193 /*
194  * getline - a hacked up version of fgets to ignore TELNET escape codes.
195  */
196 char *
197 getline(s, n, iop)
198         char *s;
199         register XFILE *iop;
200 {
201         register c;
202         register char *cs;
203
204         cs = s;
205         while (--n > 0 && (c = xgetc(iop)) >= 0) {
206                 while (c == IAC) {
207                         c = xgetc(iop); /* skip command */
208                         c = xgetc(iop); /* try next char */
209                 }
210                 *cs++ = c;
211                 if (c=='\n')
212                         break;
213         }
214         if (c < 0 && cs == s)
215                 return (XNULL);
216         *cs++ = '\0';
217         if (debug) {
218                 xoprintf(xstderr, "FTPD: command: %s", s);
219                 if (c != '\n')
220                         xputc('\n', xstderr);
221                 xfflush(xstderr);
222         }
223         return (s);
224 }
```

```
225
226  static int
227  toolong()
228  {
229          long now;
230          extern long xtime();
231
232          reply(421,
233            "Timeout (%d seconds): closing control connection.", timeout);
234          if (logging) {
235                  xoprintf(xstderr,
236                          "FTPD: User %s timed out after %d seconds at %ld",
237                          (username ? username : "unknown"), timeout,xtime());
238                  xfflush(xstderr);
239          }
240          xexit(1);
241  }
242
243  yylex()
244  {
245          /*
246           * Don't use register variables -- Zilog S8000 setret() can't cope.
247           */
248          static char cbuf[512];
249          static int cpos, state;
250          char *cp;
251          struct tab *p;
252          int n;
253          char c;
254
255          for (;;) {
256                  switch (state) {
257
258                  case CMD:
259                          if (getline(cbuf, sizeof(cbuf)-1, xstdin) == XNULL) {
260                                  reply(221, "You could at least say goodbye.");
261                                  xexit(0);
262                          }
263                          if (xstrchr(cbuf, '\r')) {
264                                  cp = xstrchr(cbuf, '\r');
265                                  cp[0] = '\n'; cp[1] = 0;
266                          }
267                          if (xstrchr(cbuf, ' '))
268                                  cpos = xstrchr(cbuf, ' ') - cbuf;
269                          else
270                                  cpos = 4;
271                          c = cbuf[cpos];
272                          cbuf[cpos] = '\0';
273                          upper(cbuf);
274                          p = lookup(cbuf);
275                          cbuf[cpos] = c;
276                          if (p != 0) {
277                                  if (p->implemented == 0) {
278                                          nack(p->name);
279                                          xlongjmp(errcatch);
280                                          /* NOTREACHED */
```

```
281                             }
282                             state = p->state;
283                             yylval = (YYSTYPE) p->name;
284                             return (p->token);
285                     }
286                     break;
287
288             case OSTR:
289                     if (cbuf[cpos] == '\n') {
290                             state = CMD;
291                             return (CRLF);
292                     }
293                     /* FALL THRU */
294
295             case STR1:
296                     if (cbuf[cpos] == ' ') {
297                             cpos++;
298                             state = STR2;
299                             return (SP);
300                     }
301                     break;
302
303             case STR2:
304                     cp = &cbuf[cpos];
305                     n = xstrlen(cp);
306                     cpos += n - 1;
307                     /*
308                      * Make sure the string is nonempty and \n terminated.
309                      */
310                     if (n > 1 && cbuf[cpos] == '\n') {
311                             cbuf[cpos] = '\0';
312                             yylval = (YYSTYPE)copy(cp);
313                             cbuf[cpos] = '\n';
314                             state = ARGS;
315                             return (STRING);
316                     }
317                     break;
318
319             case ARGS:
320                     if (isdigit(cbuf[cpos])) {
321                             cp = &cbuf[cpos];
322                             while (isdigit(cbuf[++cpos]))
323                                     ;
324                             c = cbuf[cpos];
325                             cbuf[cpos] = '\0';
326                             yylval = (YYSTYPE)xatoi(cp);
327                             cbuf[cpos] = c;
328                             return (NUMBER);
329                     }
330                     switch (cbuf[cpos++]) {
331
332                     case '\n':
333                             state = CMD;
334                             return (CRLF);
335
336                     case ' ':
```

```
337                             return (SP);
338
339                     case ',':
340                             return (COMMA);
341
342                     case 'A':
343                     case 'a':
344                             return (A);
345
346                     case 'B':
347                     case 'b':
348                             return (B);
349
350                     case 'C':
351                     case 'c':
352                             return (C);
353
354                     case 'E':
355                     case 'e':
356                             return (E);
357
358                     case 'F':
359                     case 'f':
360                             return (F);
361
362                     case 'I':
363                     case 'i':
364                             return (I);
365
366                     case 'L':
367                     case 'l':
368                             return (L);
369
370                     case 'N':
371                     case 'n':
372                             return (N);
373
374                     case 'P':
375                     case 'p':
376                             return (P);
377
378                     case 'R':
379                     case 'r':
380                             return (R);
381
382                     case 'S':
383                     case 's':
384                             return (S);
385
386                     case 'T':
387                     case 't':
388                             return (T);
389
390                     }
391                     break;
392
```

```
393                     default:
394                             fatal("Unknown state in scanner.");
395                     }
396                     state = CMD;
397                     yyerror( "lexical error" );
398             }
399     }
400
401     upper(s)
402             char *s;
403     {
404             while (*s != '\0') {
405                     if (islower(*s))
406                             *s = _toupper(*s);
407                     s++;
408             }
409     }
410
411     YYSTYPE
412     copy(s)
413             char *s;
414     {
415             char *p;
416     /*      extern char *xmalloc(); */
417
418             p = xmalloc((xstrlen(s) + 1));
419             if (p == XNULL)
420                     fatal("Ran out of memory.");
421             xstrcpy(p, s);
422             return ((YYSTYPE)p);
423     }
424
425     help(s)
426             char *s;
427     {
428             register struct tab *c;
429             register int width, NCMDS;
430
431             width = 0, NCMDS = 0;
432             for (c = cmdtab; c->name != XNULL; c++) {
433                     int len = xstrlen(c->name);
434
435                     if (c->implemented == 0)
436                             len++;
437                     if (len > width)
438                             width = len;
439                     NCMDS++;
440             }
441             width = (width + 8) &~ 7;
442             if (s == 0) {
443                     register int i, j, w;
444                     int columns, lines;
445
446                     lreply(214,
447               "The following commands are recognized (* =>'s unimplemented).");
448                     columns = 76 / width;
```

```
449                 if (columns == 0)
450                         columns = 1;
451                 lines = (NCMDS + columns - 1) / columns;
452                 for (i = 0; i < lines; i++) {
453                         xoprintf(xstdout,"    ");
454                         for (j = 0; j < columns; j++) {
455                                 c = cmdtab + j * lines + i;
456                                 xoprintf(xstdout,"%s%c", c->name,
457                                         c->implemented ? ' ' : '*');
458                                 if (c + lines >= &cmdtab[NCMDS])
459                                         break;
460                                 w = xstrlen(c->name);
461                                 while (w < width) {
462                                         xputchar(' ');
463                                         w++;
464                                 }
465                         }
466                         xoprintf(xstdout,"\r\n");
467                 }
468                 xfflush(xstdout);
469                 reply(214, "Direct comments to ftp-bugs@%s.", hostname);
470                 return;
471         }
472         upper(s);
473         c = lookup(s);
474         if (c == (struct tab *)0) {
475                 reply(504, "Unknown command %s.", s);
476                 return;
477         }
478         if (c->implemented)
479                 reply(214, "Syntax: %s %s", c->name, c->help);
480         else
481                 reply(214, "%-*s\t%s; unimplemented.", width, c->name, c->help);
482 }
483 short yyexca[] ={
484 -1, 1,
485         0, -1,
486         -2, 0,
487         };
488 # define YYNPROD 60
489 # define YYLAST 208
490 short yyact[]={
491
492   26,  54, 103, 149, 147, 145, 105, 143, 105, 124,
493   77,  63, 112,  88,  61,  59, 141,  57,   3,   4,
494    5, 148,  25,   6, 146,   7,   8,   9,  11,  12,
495   13, 144, 142,  99,  87,  86,  84,  83,  10, 140,
496   28,  82,  81,  16,  17,  15,  14,  45,  44,  19,
497   20,  21,  22,  23,  24, 139, 138, 137, 136, 135,
498  134, 133, 132, 131, 128, 119, 108, 107, 106, 126,
499  100,  98,  97, 127,  96, 104,  95,  92,  91,  52,
500   51,  46, 102, 101,  94,  93,  90,  89,  85,  80,
501   79,  78,  75,  76,  36,  35,  34,  33,  32,  31,
502   30,  74,  29,  70, 125, 109,  65,  72,  71,  66,
503   37,  67,  68,  53,  27, 111,  18,  73, 110,  69,
504   64,  62,  60,  38,  39,  40,  41,  42,  43,  58,
```

```
505     56,   2,  47,  48,  49,  50,   1,   0,   0,  55,
506      0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
507      0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
508      0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
509      0,   0,   0,   0,   0,   0,   0,   0,   0, 130,
510      0,   0,   0,   0,   0, 113, 114,   0,   0,   0,
511      0, 117,   0, 118,   0, 120, 121,   0,   0, 122,
512    123, 115,   0, 116,   0,   0,   0, 129 };
513   short yypact[]={
514
515   -1000,-256,-1000,-167,-169,-170,-171,-172,-173,-174,
516   -175,-1000,-1000,-1000,-1000,-1000,-1000,-1000,-1000,-222,
517   -189,-1000,-1000,-1000,-1000,-190,-191,-296,-1000,-255,
518   -257,-258,-262,-151,-158,-166,-263,-178,-179,-180,
519   -228,-233,-181,-235,-1000,-259,-1000,-182,-183,-192,
520   -193,-1000,-1000,-1000,-184,-185,-194,-1000,-196,-1000,
521   -198,-1000,-199,-238,-200,-186,-187,-1000,-267,-202,
522   -1000,-1000,-1000,-203,-1000,-1000,-1000,-204,-260,-260,
523   -260,-1000,-260,-1000,-260,-260,-1000,-260,-205,-260,
524   -260,-1000,-1000,-260,-260,-1000,-1000,-1000,-1000,-264,
525   -1000,-195,-195,-265,-1000,-1000,-1000,-1000,-1000,-207,
526   -1000,-1000,-1000,-208,-209,-210,-211,-212,-213,-1000,
527   -214,-215,-231,-254,-239,-1000,-1000,-1000,-1000,-1000,
528   -1000,-1000,-1000,-1000,-1000,-1000,-1000,-1000,-1000,-1000,
529   -1000,-1000,-266,-240,-268,-247,-269,-250,-270,-1000 };
530   short yypgo[]={
531
532      0, 136, 131, 130, 129, 122, 121, 120, 119, 117,
533    110, 105, 118, 116,  75, 104, 115, 114, 113 };
534   short yyr1[]={
535
536      0,   1,   1,   2,   2,   2,   2,   2,   2,   2,
537      2,   2,   2,   2,   2,   2,   2,   2,   2,   2,
538      2,   2,   2,   2,   2,   2,   2,   2,   2,   2,
539      2,   3,   4,   5,  14,   6,  15,  15,  15,   7,
540      7,   7,   7,   7,   7,   7,   7,   8,   8,   8,
541      9,   9,   9,  11,  12,  16,  13,  17,  18,  10 };
542   short yyr2[]={
543
544      0,   0,   2,   4,   4,   4,   4,   4,   4,   4,
545      4,   5,   5,   5,   3,   5,   3,   5,   5,   3,
546      5,   1,   2,   4,   2,   5,   5,   3,   3,   2,
547      2,   1,   1,   1,   1,  11,   1,   1,   1,   1,
548      3,   1,   3,   1,   1,   3,   2,   1,   1,   1,
549      1,   1,   1,   1,   1,   1,   2,   5,   4,   0 };
550   short yychk[]={
551
552   -1000,  -1,  -2, 274, 275, 276, 279, 281, 282, 283,
553    294, 284, 285, 286, 302, 301, 299, 300, -13, 305,
554    306, 307, 308, 309, 310, 278, 256, -17, 296, 269,
555    269, 269, 269, 269, 269, 269, 269, -10, -10, -10,
556    -10, -10, -10, -10, 270, 269, 270, -10, -10, -10,
557    -10, 270, 270, -18, 297, -10,  -3, 272,  -4, 272,
558     -5, 272,  -6, 273,  -7, 257, 260, 262, 263,  -8,
559    261, 266, 265,  -9, 267, 258, 259, 273, 269, 269,
560    269, 270, 269, 270, 269, 269, 270, 269, 272, 269,
```

```
561   269, 270, 270, 269, 269, 270, 270, 270, 270, 271,
562   270, 269, 269, 269, -14, 273, 270, 270, 270, -11,
563   -12, -16, 272, -11, -11, -12, -12, -11, -11, 270,
564   -11, -11, -11, -11, 273, -15, 264, 268, 259, -15,
565   -14, 270, 270, 270, 270, 270, 270, 270, 270, 270,
566   270, 270, 271, 273, 271, 273, 271, 273, 271, 273 };
567  short yydef[]={
568
569     1,  -2,   2,   0,   0,   0,   0,   0,   0,   0,
570     0,  59,  59,  59,  59,  59,  59,  59,  21,   0,
571     0,  59,  59,  59,  59,   0,   0,   0,  59,   0,
572     0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
573     0,   0,   0,   0,  22,   0,  24,   0,   0,   0,
574     0,  29,  30,  56,   0,   0,   0,  31,   0,  32,
575     0,  33,   0,   0,   0,  39,  41,  43,  44,   0,
576    47,  48,  49,   0,  50,  51,  52,   0,   0,   0,
577     0,  14,   0,  16,   0,   0,  19,   0,   0,   0,
578     0,  27,  28,   0,   0,   3,   4,   5,   6,   0,
579     7,   0,   0,   0,  46,  34,   8,   9,  10,   0,
580    53,  54,  55,   0,   0,   0,   0,   0,   0,  23,
581     0,   0,   0,   0,   0,  40,  36,  37,  38,  42,
582    45,  11,  12,  13,  15,  17,  18,  20,  25,  26,
583    58,  57,   0,   0,   0,   0,   0,   0,   0,  35 };
584  #ifndef lint
585  static char yaccpar_sccsid[] = "@(#)yaccpar     4.1     (Berkeley) 2/11/83";
586  #endif not lint
587
588  #
589  # define YYFLAG -1000
590  # define YYERROR goto yyerrlab
591  # define YYACCEPT return(0)
592  # define YYABORT return(1)
593
594  /*      parser for yacc output  */
595
596  #ifdef YYDEBUG
597  #endif
598  YYSTYPE yyv[YYMAXDEPTH] = { 0 }; /* where the values are stored */
599  int yychar = -1; /* current input token number */
600  int yynerrs = 0;  /* number of errors */
601  short yyerrflag = 0;  /* error recovery flag */
602
603  yyparse() {
604
605          short yys[YYMAXDEPTH];
606          short yyj, yym;
607          register YYSTYPE *yypvt;
608          register short yystate, *yyps, yyn;
609          register YYSTYPE *yypv;
610          register short *yyxi;
611
612          yystate = 0;
613          yychar = -1;
614          yynerrs = 0;
615          yyerrflag = 0;
616          yyps= &yys[-1];
```

```
617          yypv= &yyv[-1];
618
619   yystack:    /* put a state and value onto the stack */
620
621   #ifdef YYDEBUG
622   #endif
623                  if( ++yyps> &yys[YYMAXDEPTH] ) { yyerror( "yacc stack overflow" ); retu
624                  *yyps = yystate;
625                  ++yypv;
626                  *yypv = yyval;
627
628   yynewstate:
629
630          yyn = yypact[yystate];
631
632          if( yyn<= YYFLAG ) goto yydefault; /* simple state */
633
634          if( yychar<0 ) if( (yychar=yylex())<0 ) yychar=0;
635          if( (yyn += yychar)<0 || yyn >= YYLAST ) goto yydefault;
636
637          if( yychk[ yyn=yyact[ yyn ] ] == yychar ){ /* valid shift */
638                  yychar = -1;
639                  yyval = yylval;
640                  yystate = yyn;
641                  if( yyerrflag > 0 ) --yyerrflag;
642                  goto yystack;
643                  }
644
645   yydefault:
646          /* default state action */
647
648          if( (yyn=yydef[yystate]) == -2 ) {
649                  if( yychar<0 ) if( (yychar=yylex())<0 ) yychar = 0;
650                  /* look through exception table */
651
652                  for( yyxi=yyexca; (*yyxi!= (-1)) || (yyxi[1]!=yystate) ; yyxi += 2 ) ;
653
654                  while( *(yyxi+=2) >= 0 ){
655                          if( *yyxi == yychar ) break;
656                          }
657                  if( (yyn = yyxi[1]) < 0 ) return(0);   /* accept */
658                  }
659
660          if( yyn == 0 ){ /* error */
661                  /* error ... attempt to resume parsing */
662
663                  switch( yyerrflag ){
664
665                  case 0:   /* brand new error */
666
667                          yyerror( "syntax error" );
668                  yyerrlab:
669                          ++yynerrs;
670
671                  case 1:
672                  case 2: /* incompletely recovered error ... try again */
```

```
673
674                         yyerrflag = 3;
675
676                         /* find a state where "error" is a legal shift action */
677
678                         while ( yyps >= yys ) {
679                            yyn = yypact[*yyps] + YYERRCODE;
680                            if( yyn>= 0 && yyn < YYLAST && yychk[yyact[yyn]] == YYERRCOD
681                               yystate = yyact[yyn];  /* simulate a shift of "error" */
682                               goto yystack;
683                               }
684                            yyn = yypact[*yyps];
685
686                            /* the current yyps has no shift onn "error", pop stack */
687
688 #ifdef YYDEBUG
689 #endif
690                            --yyps;
691                            --yypv;
692                            }
693
694                         /* there is no state on the stack with an error shift ... abort
695
696         yyabort:
697                         return(1);
698
699
700                 case 3:  /* no shift yet; clobber input char */
701
702 #ifdef YYDEBUG
703 #endif
704
705                         if( yychar == 0 ) goto yyabort; /* don't discard EOF, quit */
706                         yychar = -1;
707                         goto yynewstate;   /* try again in the same state */
708
709                         }
710
711                 }
712
713         /* reduction by production yyn */
714
715 #ifdef YYDEBUG
716 #endif
717                 yyps -= yyr2[yyn];
718                 yypvt = yypv;
719                 yypv -= yyr2[yyn];
720                 yyval = yypv[1];
721                 yym=yyn;
722                         /* consult goto table to find next state */
723                 yyn = yyr1[yyn];
724                 yyj = yypgo[yyn] + *yyps + 1;
725                 if( yyj>=YYLAST || yychk[ yystate = yyact[yyj] ] != -yyn ) yystate = yy
726                 switch(yym){
727
728 case 2:
```

```
729  # line 75 "ftpcmd.y"
730  {
731                  if( globargs )
732                          xdealglob( globargs );
733                  globargs = (char **)0;
734                  } break;
735  case 3:
736  # line 83 "ftpcmd.y"
737   {
738                          int success;
739
740                          if( logged_in )
741                                  {
742                                  reply(531, "Already logged in.");
743                                  xfree( yypvt[-1] );
744                                  }
745                          else if ((success =
746                                  xinit_env(yypvt[-1], (char *)0, (char *)0))
747                                  <= 0
748                          )
749                                  {
750                                  guest = 0;
751                                  reply(331, "Password required for %s.", yypvt[-1]);
752                                  if( username )
753                                          xfree( username );
754                                  username = yypvt[-1];
755                                  }
756                          else if ( success < 0 )
757                                  {
758                                  /*
759                                  Do we want to give out this informantion?
760                                  */
761                                  reply(530, "User %s unknown.", yypvt[-1]);
762                                  xfree(yypvt[-1]);
763                                  if( username )
764                                          xfree( username );
765                                  username = (char *)0;
766                                  }
767                          else
768                                  {
769                                  username = yypvt[-1];
770                                  reply(230, "User %s logged in.", yypvt[-1]);
771                                  logged_in = 1;
772                                  }
773                  } break;
774  case 4:
775  # line 121 "ftpcmd.y"
776   {
777                          int success;
778
779                          if( !username )
780                                  {
781                                  reply(530, "Log in with user first.");
782                                  xfree( yypvt[-1] );
783                                  }
784                          else if( logged_in )
```

```
785                                     {
786                                     reply(531, "Already logged in.");
787                                     xfree( yypvt[-1] );
788                                     }
789                             else if ((success =
790                                     xinit_env(username, yypvt[-1], (char *)0))
791                                     == 0
792                             )
793                                     {
794                                     guest = 0;
795                                     reply(331,"Account required for %s.", username);
796                                     if( userpass )
797                                             xfree( userpass );
798                                     userpass = yypvt[-1];
799                                     }
800                             else if ( success > 0 )
801                                     {
802                                     userpass = yypvt[-1];
803                                     reply(230, "User %s logged in.", username);
804                                     logged_in = 1;
805                                     }
806                             else            /* sucess < 0 (tricotomy) */
807                                     {
808                                     reply( 530, "Login failed." );
809                                     xfree( yypvt[-1] );
810                                     }
811                     } break;
812     case 5:
813     # line 158 "ftpcmd.y"
814      {
815                             int success;
816
817                             if( !username )
818                                     {
819                                     reply(530, "Log in with user first.");
820                                     xfree( yypvt[-1] );
821                                     }
822                             else if( logged_in )
823                                     {
824                                     reply(531, "Already logged in.");
825                                     xfree( yypvt[-1] );
826                                     }
827                             else if ((success =
828                                     xinit_env(username, userpass, yypvt[-1]))
829                                     <= 0
830                             )
831                                     {
832                                     guest = 0;
833                                     reply( 530, "Login incorrect." );
834                                     xfree( yypvt[-1] );
835                                     }
836                             else if ( success > 0 )
837                                     {
838                                     reply(230, "User %s logged in.", username);
839                                     logged_in = 1;
840                                     xfree( yypvt[-1] );
```

```
841                                     }
842                     } break;
843     case 6:
844     # line 188 "ftpcmd.y"
845      {
846                             usedefault = 0;
847                             ack(yypvt[-3]);
848                     } break;
849     case 7:
850     # line 193 "ftpcmd.y"
851      {
852                             switch (cmd_type) {
853
854                             case TYPE_A:
855                                     if (cmd_form == FORM_N) {
856                                             reply(200, "Type set to A.");
857                                             type = cmd_type;
858                                             form = cmd_form;
859                                     } else
860                                             reply(504, "Form must be N.");
861                                     break;
862
863                             case TYPE_E:
864                                     reply(504, "Type E not implemented.");
865                                     break;
866
867                             case TYPE_I:
868                                     reply(200, "Type set to I.");
869                                     type = cmd_type;
870                                     break;
871
872                             case TYPE_L:
873                                     if (cmd_bytesz == 8) {
874                                             reply(200,
875                                                 "Type set to L (byte size 8).");
876                                             type = cmd_type;
877                                     } else
878                                             reply(504, "Byte size must be 8.");
879                             }
880                     } break;
881     case 8:
882     # line 224 "ftpcmd.y"
883      {
884                             switch ((int)yypvt[-1]) {
885
886                             case STRU_F:
887                                     reply(200, "STRU F ok.");
888                                     break;
889
890                             default:
891                                     reply(502, "Unimplemented STRU type.");
892                             }
893                     } break;
894     case 9:
895     # line 236 "ftpcmd.y"
896      {
```

```
897                             switch ((int)yypvt[-1]) {
898
899                             case MODE_S:
900                                     reply(200, "MODE S ok.");
901                                     break;
902
903                             default:
904                                     reply(502, "Unimplemented MODE type.");
905                             }
906                     } break;
907     case 10:
908     # line 248 "ftpcmd.y"
909      {
910                             ack(yypvt[-3]);
911                     } break;
912     case 11:
913     # line 252 "ftpcmd.y"
914      {
915                             if (yypvt[-3] && yypvt[-1] != XNULL)
916                                     retrieve(0, yypvt[-1]);
917                     } break;
918     case 12:
919     # line 257 "ftpcmd.y"
920      {
921                             if (yypvt[-3] && yypvt[-1] != XNULL)
922                                     store(yypvt[-1], "w");
923                     } break;
924     case 13:
925     # line 262 "ftpcmd.y"
926      {
927                             if (yypvt[-3] && yypvt[-1] != XNULL)
928                                     store(yypvt[-1], "a");
929                     } break;
930     case 14:
931     # line 267 "ftpcmd.y"
932      {
933                             if (yypvt[-1])
934                                     retrieve( LS, "");
935                     } break;
936     case 15:
937     # line 272 "ftpcmd.y"
938      {
939                             if (yypvt[-3] && yypvt[-1] != XNULL)
940                                     retrieve( LS_ARG, yypvt[-1]);
941                             if (yypvt[-1] != XNULL)
942                                     xfree(yypvt[-1]);
943                     } break;
944     case 16:
945     # line 279 "ftpcmd.y"
946      {
947                             if (yypvt[-1])
948                                     retrieve( LSLONG, "");
949                     } break;
950     case 17:
951     # line 284 "ftpcmd.y"
952      {
```

```
953                         if (yypvt[-3] && yypvt[-1] != XNULL)
954                                 retrieve( LSLONG_ARG, yypvt[-1]);
955                         if (yypvt[-1] != XNULL)
956                                 xfree(yypvt[-1]);
957                 } break;
958 case 18:
959 # line 291 "ftpcmd.y"
960  {
961                         if (yypvt[-3] && yypvt[-1] != XNULL)
962                                 delete(yypvt[-1]);
963                 } break;
964 case 19:
965 # line 296 "ftpcmd.y"
966  {
967                         if (yypvt[-1])
968                                 xchdir( (char *)0, HOME_DIR);
969                 } break;
970 case 20:
971 # line 301 "ftpcmd.y"
972  {
973                         if (yypvt[-3] && yypvt[-1] != XNULL)
974                                 cwd(yypvt[-1], FILE_NAME );
975                 } break;
976 case 22:
977 # line 307 "ftpcmd.y"
978  {
979                         help(0);
980                 } break;
981 case 23:
982 # line 311 "ftpcmd.y"
983  {
984                         help(yypvt[-1]);
985                 } break;
986 case 24:
987 # line 315 "ftpcmd.y"
988  {
989                         ack(yypvt[-1]);
990                 } break;
991 case 25:
992 # line 319 "ftpcmd.y"
993  {
994                         if (yypvt[-3] && yypvt[-1] != XNULL)
995                                 makedir(yypvt[-1]);
996                 } break;
997 case 26:
998 # line 324 "ftpcmd.y"
999  {
1000                        if (yypvt[-3] && yypvt[-1] != XNULL)
1001                                removedir(yypvt[-1]);
1002                } break;
1003 case 27:
1004 # line 329 "ftpcmd.y"
1005  {
1006                        if (yypvt[-1])
1007                                pwd();
1008                } break;
```

```
1009  case 28:
1010  # line 334 "ftpcmd.y"
1011   {
1012                           if (yypvt[-1] && !inappropriate_request(".."))
1013                                   cwd("..", UP_DIRECTORY );
1014                   } break;
1015  case 29:
1016  # line 339 "ftpcmd.y"
1017   {
1018                           reply(221, "Goodbye.");
1019                           xexit(0);
1020                   } break;
1021  case 30:
1022  # line 344 "ftpcmd.y"
1023   {
1024                           yyerrok;
1025                   } break;
1026  case 35:
1027  # line 363 "ftpcmd.y"
1028   {
1029                           register char *a, *p;
1030
1031                           a = (char *)&data_dest.sin_addr;
1032                           a[0] = (int)yypvt[-10]; a[1] = (int)yypvt[-8];
1033                           a[2] = (int)yypvt[-6]; a[3] = (int)yypvt[-4];
1034                           p = (char *)&data_dest.sin_port;
1035                           p[0] = (int)yypvt[-2]; p[1] = (int)yypvt[-0];
1036                           data_dest.sin_family = AF_INET;
1037                   } break;
1038  case 36:
1039  # line 376 "ftpcmd.y"
1040   {
1041                   yyval = (YYSTYPE)FORM_N;
1042           } break;
1043  case 37:
1044  # line 380 "ftpcmd.y"
1045   {
1046                   yyval = (YYSTYPE)FORM_T;
1047           } break;
1048  case 38:
1049  # line 384 "ftpcmd.y"
1050   {
1051                   yyval = (YYSTYPE)FORM_C;
1052           } break;
1053  case 39:
1054  # line 390 "ftpcmd.y"
1055   {
1056                   cmd_type = TYPE_A;
1057                   cmd_form = FORM_N;
1058           } break;
1059  case 40:
1060  # line 395 "ftpcmd.y"
1061   {
1062                   cmd_type = TYPE_A;
1063                   cmd_form = (int)yypvt[-0];
1064           } break;
```

```
1065  case 41:
1066  # line 400 "ftpcmd.y"
1067   {
1068                   cmd_type = TYPE_E;
1069                   cmd_form = FORM_N;
1070           } break;
1071  case 42:
1072  # line 405 "ftpcmd.y"
1073   {
1074                   cmd_type = TYPE_E;
1075                   cmd_form = (int)yypvt[-0];
1076           } break;
1077  case 43:
1078  # line 410 "ftpcmd.y"
1079   {
1080                   cmd_type = TYPE_I;
1081           } break;
1082  case 44:
1083  # line 414 "ftpcmd.y"
1084   {
1085                   cmd_type = TYPE_L;
1086                   cmd_bytesz = 8;
1087           } break;
1088  case 45:
1089  # line 419 "ftpcmd.y"
1090   {
1091                   cmd_type = TYPE_L;
1092                   cmd_bytesz = (int)yypvt[-0];
1093           } break;
1094  case 46:
1095  # line 425 "ftpcmd.y"
1096   {
1097                   cmd_type = TYPE_L;
1098                   cmd_bytesz = (int)yypvt[-0];
1099           } break;
1100  case 47:
1101  # line 432 "ftpcmd.y"
1102   {
1103                   yyval = (YYSTYPE)STRU_F;
1104           } break;
1105  case 48:
1106  # line 436 "ftpcmd.y"
1107   {
1108                   yyval = (YYSTYPE)STRU_R;
1109           } break;
1110  case 49:
1111  # line 440 "ftpcmd.y"
1112   {
1113                   yyval = (YYSTYPE)STRU_P;
1114           } break;
1115  case 50:
1116  # line 446 "ftpcmd.y"
1117   {
1118                   yyval = (YYSTYPE)MODE_S;
1119           } break;
1120  case 51:
```

```
1121  # line 450 "ftpcmd.y"
1122   {
1123                   yyval = (YYSTYPE)MODE_B;
1124           } break;
1125  case 52:
1126  # line 454 "ftpcmd.y"
1127   {
1128                   yyval = (YYSTYPE)MODE_C;
1129           } break;
1130  case 53:
1131  # line 460 "ftpcmd.y"
1132   {
1133                   char *argv[2];
1134
1135                   argv[0] = (char *)yypvt[-0];
1136                   argv[1] = (char *)0;
1137                   globargs = xglob(argv);
1138                   if (globerr != XNULL) {
1139                           reply(550, globerr);
1140                           yyval = (YYSTYPE)XNULL;
1141                   } else if( globargs == XNULL || *globargs == XNULL ) {
1142                           reply(550, "No file name matches.");
1143                           yyval = (YYSTYPE)XNULL;
1144                   } else {
1145                           yyval = (YYSTYPE)*globargs;
1146                   }
1147                   if (inappropriate_request(yyval))
1148                           yyval = (YYSTYPE)XNULL;
1149                   xfree(yypvt[-0]);
1150           } break;
1151  case 54:
1152  # line 482 "ftpcmd.y"
1153   {
1154                   if (yypvt[-0] && inappropriate_request(yypvt[-0])) {
1155                           yyval = (YYSTYPE)XNULL;
1156                           xfree(yypvt[-0]);
1157                   } else
1158                           yyval = yypvt[-0];
1159           } break;
1160  case 56:
1161  # line 495 "ftpcmd.y"
1162   {
1163                   if (yypvt[-1] && yypvt[-0])
1164                           renamecmd(yypvt[-1], yypvt[-0]);
1165                   else
1166                           reply(503, "Bad sequence of commands.");
1167                   /*
1168                    * Since two path names are involved, we should delalocate
1169                    * space for the first one, as globargs contains the result
1170                    * of the second globbing, and will be dealocated when
1171                    * the reduction to cmd takes place.
1172                    */
1173                   if (rnf_glob)
1174                           xdealglob(rnf_glob);
1175           } break;
1176  case 57:
```

```
1177  # line 512 "ftpcmd.y"
1178   {
1179                   char *from = 0, *renamefrom();
1180
1181                   if (yypvt[-3] && yypvt[-1])
1182                           from = renamefrom(yypvt[-1]);
1183                   rnf_glob = globargs;
1184                   yyval = (YYSTYPE)from;
1185           } break;
1186  case 58:
1187  # line 523 "ftpcmd.y"
1188   {
1189                   yyval = yypvt[-1];
1190           } break;
1191  case 59:
1192  # line 529 "ftpcmd.y"
1193   {
1194                   if (logged_in)
1195                           yyval = (YYSTYPE)1;
1196                   else {
1197                           reply(530, "Please login with USER and PASS.");
1198                           yyval = (YYSTYPE)0;
1199                   }
1200           } break;
1201                   }
1202                   goto yystack;  /* stack new state and value */
1203
1204           }
```

```
 1  #ifndef lint
 2  static char sccsid[] = " @(#)ftpd.c      1.16 7/29/85";
 3  #endif
 4
 5  /*
 6   * FTP server.
 7   */
 8  #include "ftpd.h"
 9
10  extern long xpasstnet(), xpassfnet();
11  extern  char version[];
12  extern  XFILE *xodopen();
13  /*
14  extern  int fclose();
15  */
16  extern  char *xrerror();
17  extern  int xclose();
18  extern  char **xmkarglist();
19  extern  char **xglob();
20  extern  char *globerr;
21
22  struct  sckadr_in ctrl_addr = { AF_INET };
23  struct  sckadr_in data_source = { AF_INET };
24  struct  sckadr_in data_dest = { AF_INET };
25  struct  sckadr_in his_addr = { AF_INET };
26
27  struct  hostent *hp = 0;
28
29  int     data = 0;
30  #ifdef zilog
31  ret_buf errcatch;
32  #else
33  jmp_buf errcatch = { 0 };
34  #endif
35  int     logged_in = 0;
36  int     debug = 0;
37  int     timeout = 0;
38  int     logging = 0;
39  int     guest = 0;
40  int     type = 0;
41  int     form = 0;
42  int     stru = 0;                       /* avoid C keyword */
43  int     mode = 0;
44  int     bytesize = 0;
45  int     usedefault = 1;         /* for data transfers */
46  char    hostname[32] = {0};
47  char    *remotehost = (char *)0;
48  struct  servent *sp = (struct servent *)0;
49
50  /*
51   * Timeout intervals for retrying connections
52   * to hosts that don't accept PORT cmds.  This
53   * is a kludge, but given the problems with TCP...
54   */
55  #define SWAITMAX        90      /* wait at most 90 seconds */
56  #define SWAITINT        5       /* interval between retries */
```

```
 57
 58  int     swaitmax = SWAITMAX;
 59  int     swaitint = SWAITINT;
 60
 61  int     lostconn();
 62  XFILE   *dataconn();
 63  char    *ntoa();
 64
 65  ftpdoit( s, from )
 66  /*
 67  start of generic ftp demon code
 68  */
 69
 70  int s;
 71  struct sckadr_in *from;
 72  {
 73          if (logging)
 74                  dolog(&his_addr);
 75          xdup2(s, 0);
 76          if( s != 0 )
 77                  xclose(s);
 78          xdup2(0, 1);
 79          xodopen( 0, "r" );
 80          xodopen( 1, "w" );
 81          /* do telnet option negotiation here */
 82          /*
 83           * Set up default state
 84           */
 85          data = -1;
 86          type = TYPE_A;
 87          form = FORM_N;
 88          stru = STRU_F;
 89          mode = MODE_S;
 90          bytesize = 8;
 91          sp = gsbname("ftp", "tcp");
 92          if (sp == 0) {
 93                  xoprintf(xstderr, "ftpd: ftp/tcp: unknown service\n");
 94                  xexit(1);
 95          }
 96          xghname(hostname, sizeof (hostname));
 97          ctrl_addr.sin_port = xhtons(sp->s_port);
 98          data_source.sin_port = xhtons(sp->s_port - 1);
 99          reply(220, "%s FTP server (%s) ready.", hostname, version);
100          for (;;) {
101                  xsetjmp(errcatch);
102                  if( logging )
103                          {
104                          xoprintf( xstderr, "calling yyparse\n" );
105                          xfflush( xstderr );
106                          }
107                  yyparse();
108          }
109  }
110
111  lostconn()
112  {
```

```
113
114             fatal("Connection closed.");
115     }
116
117     retrieve(cmd, name)
118             int cmd;
119             char *name;
120     {
121             XFILE *fin, *dout;
122             int inod;
123             int (*closefunc)();
124             int omode;
125             struct ftp_attr attributes;
126             char *argv1[2];
127             char **argv2;
128             char **argv3;
129             char *pt;
130
131             if (cmd == 0) {
132                     /*
133                     simple file
134                     */
135                     omode = XFREAD;
136                     attributes.rep_type = type;
137                     attributes.format = form;
138                     attributes.structure = stru;
139                     attributes.trans_mode = mode;
140                     attributes.byte_sz = bytesize;
141                     inod = xftpopen( name, omode, FILE_NAME, &attributes );
142                     if (inod < 0 ) {
143                             reply(550, "%s: %s.", name, xrerror( inod ));
144                             return;
145                     }
146                     fin = xodopen( inod, "r");
147                     if ( fin == XNULL ) {
148                             reply(550, "xodopen failed.");
149                             return;
150                     }
151                     closefunc = xclose;
152             } else {
153                     /*
154                     we are to generate a psuedo file,
155                     at the moment some form of ls => call xls after opening
156                     data connection.
157                     */
158             }
159             dout = dataconn(name, (off_t)0, "w");
160             if (dout == XNULL)
161                     goto done;
162             if( cmd )
163                     {
164                     /*
165                     a psuedo file object ( ls, ls -lg)
166                     */
167                     if( xstrlen( name ) )
168                             {
```

```
169                         /*
170                         name may require globbing (for remote globbing).
171                         */
172                         argv1[0] = name;
173                         argv1[1] = (char *)0;
174                         argv2 = xglob( argv1 );
175                         if( argv2 == XNULL || globerr )
176                                 {
177                                 xclose( xfileno(dout) );
178                                 data = -1;
179                                 reply( 500, "Remote glob failed. %s", globerr );
180                                 if( argv2 != XNULL )
181                                         xdealglob( argv2 );
182                                 return;
183                                 }
184                         argv3 = argv2;
185                         for( pt = *argv3++ ; pt ; pt = *argv3++ )
186                                 {
187                                 xls( xfileno(dout), pt, cmd );
188                                 }
189                         xdealglob( argv2 );
190                         }
191                 else
192                         {
193                         xls( xfileno(dout), name, cmd );
194                         }
195                 xclose( xfileno(dout) );
196                 data = -1;
197                 reply(226, "Transfer complete.");
198                 return;
199                 }
200         else if (send_data(name, fin, dout) )
201                 {
202                 xclose( xfileno(dout)), data = -1;
203                 }
204         else
205                 {
206                 reply(226, "Transfer complete.");
207                 xclose( xfileno(dout)), data = -1;
208                 }
209 done:
210         xclose( xfileno(fin) );
211 }
212
213 store(name, append)
214         char *name, *append;
215 {
216         XFILE *fout, *din;
217         int outod;
218         int omode;
219         int (*closefunc)(), dochown = 1;
220         struct ftp_attr attributes;
221
222         omode = XFWRITE | XFCREAT | (( *append == 'a' )? XFAPPEND : XFTRUNC );
223         attributes.rep_type = type;
224         attributes.format = form;
```

```
225             attributes.structure = stru;
226             attributes.trans_mode = mode;
227             attributes.byte_sz = bytesize;
228             outod = xftpopen( name, omode, FILE_NAME, &attributes );
229             if( outod < 0 )
230                     {
231                     reply(550, "%s: %s.", name, xrerror( outod ) );
232                     return;
233                     }
234             fout = xodopen( outod, "w" ), closefunc = xclose;
235             if (fout == XNULL) {
236                     reply(550, "xodopen failed.");
237                     return;
238             }
239             din = dataconn(name, (off_t)-1, "r");
240             if (din == XNULL)
241                     goto done;
242             if (receive_data(name, din, fout) )
243                     {
244                     }
245             else
246                     {
247                     reply(226, "Transfer complete.");
248                     }
249             xclose( xfileno(din)), data = -1;
250     done:
251             VOID xchown(name, FILE_NAME );
252             xclose( xfileno(fout) );
253     }
254
255
256     getdatasock(mode)
257             char *mode;
258     {
259             int s;
260             int retry;
261
262             if (data >= 0)
263                     return (data);
264             data_source.sin_family = AF_INET;
265             for (retry = 0; retry < swaitmax; xsleep (swaitint), retry += swaitint)
266             {
267                     s = xsocket(SOCK_STREAM, 0, &data_source,
268                       SO_KEEPALIVE|SO_REUSEADDR);
269                     /* GAP 7/25/85: REUSEADDR fixes simultaneous xfer bug */
270                     if (s >= 0 || ( s != XEADDRINUSE && s != XENOBUFS))
271                             break;
272             }
273             if (s < 0)
274                     xperror( s, "getdatasock" );
275             return ( s );
276     }
277
278
279     XFILE *
280     dataconn(name, size, mode)
```

```
281         char *name;
282         off_t size;             /* no longer used */
283         char *mode;
284 {
285         char sizebuf[32];
286         XFILE *file;
287         int retry = 0;
288         int s;
289         int rval;
290
291         if (data >= 0) {
292                 reply(125, "Using existing data connection for %s.", name);
293                 usedefault = 1;
294                 return (xodopen(data, mode));
295         }
296         if (usedefault)
297                 xbcopy( &his_addr, &data_dest, sizeof(struct sockaddr));
298         usedefault = 1;
299         s = getdatasock(mode);
300         if ( s < 0) {
301                 reply(425, "Can't create data socket (%s,%d): %s.",
302                     ntoa(data_source.sin_addr.s_addr),
303                     xntohs(data_source.sin_port),
304                     xrerror( s ));
305                 return (XNULL);
306         }
307         reply(150, "Opening data connection for %s (%s,%d).",
308             name, ntoa(data_dest.sin_addr.s_addr),
309             xntohs(data_dest.sin_port));
310         data = s;
311         while ((rval = xconnect(data, &data_dest)) < 0) {
312                 if (rval == XEADDRINUSE && retry < swaitmax) {
313                         xsleep(swaitint);
314                         retry += swaitint;
315                         continue;
316                 }
317                 reply(425, "Can't build data connection: %s.",
318                     xrerror( rval ) );
319                 VOID xclose(data);
320                 data = -1;
321                 return (XNULL);
322         }
323         file = xodopen( data, mode );
324         return (file);
325 }
326
327 /*
328  * Tranfer the contents of "instr" to
329  * "outstr" peer using the appropriate
330  * encapulation of the date subject
331  * to Mode, Structure, and Type.
332  *
333  * NB: Form isn't handled.
334  */
335 send_data(name, instr, outstr)
336         char *name;
```

```
337         XFILE *instr, *outstr;
338 {
339         long rval;
340
341         rval = xpasstnet( instr, outstr );
342         if( rval == XEOPNOTSUPP )
343                 {
344                 reply(504,"Unimplemented TYPE %d in send_data", type);
345                 return( 1 );
346                 }
347         else if ( rval < 0 )
348                 {
349                 reply(550, "%s: %s.", name, xrerror( rval ) );
350                 return( 1 );
351                 }
352         return (0);
353 }
354
355 /*
356  * Transfer data from peer to
357  * "outstr" using the appropriate
358  * encapulation of the data subject
359  * to Mode, Structure, and Type.
360  *
361  * N.B.: Form isn't handled.
362  */
363 receive_data(name, instr, outstr)
364         char *name;
365         XFILE *instr, *outstr;
366 {
367         long rval;
368
369         rval = xpassfnet( instr, outstr );
370         if( rval == XEOPNOTSUPP )
371                 {
372                 reply(504, "TYPE E not implemented.");
373                 return (1);
374                 }
375         else if ( rval < 0 )
376                 {
377                 reply(550, "%s: %s.", name, xrerror( rval ) );
378                 return (1);
379                 }
380         return( 0 );
381 }
382
383 fatal(s)
384         char *s;
385 {
386         reply(451, "Error in server: %s\n", s);
387         reply(221, "Closing connection due to server error.");
388         xexit(1);
389 }
390
391 #ifdef zilog
392 reply(n, s, a1, a2, a3, a4, a5, a6)
```

```
393         int n;
394         char *s;
395         int a1, a2, a3, a4, a5, a6;
396 {
397         int args=a1, aa2=a2, aa3=a3, aa4=a4, aa5=a5, aa6=a6;
398 #else
399 reply(n, s, args)
400         int n;
401         char *s;
402 {
403 #endif
404         xoprintf(xstdout,"%d ", n);
405         _mydoprnt(s, &args, xstdout);
406         xoprintf(xstdout,"\r\n");
407         xfflush(xstdout);
408         if (debug) {
409                 xoprintf(xstderr, "<--- %d ", n);
410                 _mydoprnt(s, &args, xstderr);
411                 xoprintf(xstderr, "\n");
412                 xfflush(xstderr);
413         }
414 }
415
416 #ifdef zilog
417 lreply(n, s, a1, a2, a3, a4, a5, a6)
418         int n;
419         char *s;
420         int a1, a2, a3, a4, a5, a6;
421 {
422         int args=a1, aa2=a2, aa3=a3, aa4=a4, aa5=a5, aa6=a6;
423 #else
424 lreply(n, s, args)
425         int n;
426         char *s;
427 {
428 #endif
429         xoprintf(xstdout,"%d-", n);
430         _mydoprnt(s, &args, xstdout);
431         xoprintf(xstdout,"\r\n");
432         xfflush(xstdout);
433         if (debug) {
434                 xoprintf(xstderr, "<--- %d-", n);
435                 _mydoprnt(s, &args, xstderr);
436                 xoprintf(xstderr, "\n");
437         }
438 }
439
440 replystr(s)
441         char *s;
442 {
443         xoprintf(xstdout,"%s\r\n", s);
444         xfflush(xstdout);
445         if (debug)
446                 xoprintf(xstderr, "<--- %s\n", s);
447 }
448
```

```
449  ack(s)
450          char *s;
451  {
452          reply(200, "%s command okay.", s);
453  }
454
455  nack(s)
456          char *s;
457  {
458          reply(502, "%s command not implemented.", s);
459  }
460
461  yyerror( message )
462
463  char *message;
464  {
465          reply(500, "Command not understood: %s.", message );
466          xlongjmp( errcatch, 1 );
467  }
468
469  delete(name)
470          char *name;
471  {
472          int rval;
473
474          if ((rval = xunlink(name, FILE_NAME )) < 0) {
475                  reply(550, "%s: %s.", name, xrerror( rval ));
476                  return;
477          }
478          ack("DELE");
479  }
480
481  cwd(path, special)
482          char *path;
483          int special;
484  {
485          int rval;
486
487          if (( rval = xchdir(path, special ) ) < 0) {
488                  reply(550, "%s: %s.", path, xrerror( rval ) );
489                  return;
490          }
491          ack("CWD");
492  }
493
494  makedir(name)
495          char *name;
496  {
497          int rval;
498
499          if ((rval = xmkdir(name, FILE_NAME)) < 0) {
500                  reply(550, "%s: %s.", name, xrerror( rval  ) );
501                  return;
502          }
503          VOID xchown(name, FILE_NAME );
504          ack("MKDIR");
```

```
505  }
506
507  removedir(name)
508          char *name;
509  {
510          int rval;
511
512          if (( rval = xrmdir(name, FILE_NAME)) < 0) {
513                  reply(550, "%s: %s.", name, xrerror( rval ) );
514                  return;
515          }
516          ack("RMDIR");
517  }
518
519  pwd()
520  {
521          char path[MAXPATHLEN + 1];
522          int success;
523
524          success = xpwd( path, MAXPATHLEN + 1, PWD );
525          if ( success < 0) {
526                  reply(451, "working directory not available.");
527                  return;
528          }
529          reply(251, "\"%s\" is current directory.", path);
530  }
531
532  char *
533  renamefrom(name)
534          char *name;
535  {
536          int rval;
537
538          if ( (rval = xaccess( name, FILE_NAME, 0 ) ) < 0  ) {
539                  reply(550, "%s: %s.", name, xrerror( rval ) );
540                  return ((char *)0);
541          }
542          reply(350, "File exists, ready for destination name");
543          return (name);
544  }
545
546  renamecmd(from, to)
547          char *from, *to;
548  {
549          int rval;
550
551          if ((rval = xrename(from, FILE_NAME, to, FILE_NAME) ) < 0) {
552                  reply(550, "rename: %s.", xrerror( rval ) );
553                  return;
554          }
555          ack("RNTO");
556  }
557
558  /*
559   * Test pathname for guest-user safety.
560   */
```

```
561  inappropriate_request(name)
562          char *name;
563  {
564          int depth = 0, length, size;
565          register char *p, *s;
566
567          length = ( name )? xstrlen( name ) : 0 ;
568          /*
569          This functionality probably belongs in xftpopen,
570          but for now.
571          */
572          return( 0 );
573  }
574
575  /*
576   * Convert network-format internet address
577   * to base 256 d.d.d.d representation.
578   */
579  char *
580  ntoa(in)
581          struct in_addr in;
582  {
583          static char b[18];
584          register char *p;
585
586          p = (char *)&in;
587  #define UC(b)   (((int)b)&0xff)
588          xsprintf(b, "%d.%d.%d.%d", UC(p[0]), UC(p[1]), UC(p[2]), UC(p[3]));
589          return (b);
590  }
591
592  dolog(sin)
593          struct sckadr_in *sin;
594  {
595          char saddr[16], *ntoa();
596          struct hostent *hp = ghbaddr(&sin->sin_addr,
597                  sizeof (struct in_addr), AF_INET);
598          char *remotehost;
599          long t;
600          long xtime();
601
602          if (hp)
603                  remotehost = hp->h_name;
604          else
605                  remotehost = ntoa (sin->sin_addr.s_addr);
606          t = xtime();
607          xoprintf(xstderr,"FTPD: connection from %s at %ld",
608                  remotehost, t );
609          xfflush(xstderr);
610  }
611
612
613  /*
614  mp  -  Even if _doprnt is more wonderful than _mydoprnt for systems which
615          have _doprnt, using _doprnt is an incredible maintainance headache.
616          In any case, we should support the same functionality on all
```

```
617          systems.
618          Hence, may _doprnt rest in peace.
619  */
620  _mydoprnt(format, argp, FILEp)
621  char *format;
622  int *argp;
623  XFILE *FILEp;
624  {
625          xoprintf(FILEp, format, *argp, *(argp+1), *(argp+2), *(argp+3),
626                  *(argp+4), *(argp+5));
627  }
628
629
630
631  struct servent * gsbname (service, proto)
632   char *service, *proto;
633  {
634          static struct servent servstat;
635
636          servstat.s_name = service;
637          servstat.s_aliases = 0;
638          if (xstrcmp (service, "ftp") == 0)
639                  servstat.s_port = (IPRT_FTP);
640          else
641          if (xstrcmp (service, "telnet") == 0)
642                  servstat.s_port = (IPPORT_TELNET);
643          else
644                  return (0);
645          servstat.s_proto = proto;
646          return (&servstat);
647  }
648
649  extern char *xraddr();
650  extern struct hostent *ghbname ();
651
652  struct hostent *
653  ghbaddr(addr)
654  struct in_addr *addr;
655  {
656          char *name;
657
658          if ((name = xraddr (addr -> s_addr)) == 0)
659                  return (0);
660          return (ghbname(name));
661  }
662
663  extern long xrhost ();
664
665  struct hostent *
666  ghbname(host)
667  char *host;
668  {
669          static struct hostent def;
670          static struct in_addr defaddr;
671          static char namebuf[128];
672
```

```
673             defaddr.s_addr = xrhost(&host);
674             if (defaddr.s_addr == -1)
675                     return (0);
676             xstrcpy(namebuf, host);
677             def.h_name = namebuf;
678             def.h_addr = (char *)&defaddr;
679             def.h_length = sizeof (struct in_addr);
680             def.h_addrtype = AF_INET;
681             def.h_aliases = 0;
682             return (&def);
683     }
```

```
 1
 2  /*
 3   * FTP server.
 4   */
 5  /*
 6  */
 7  #include <xgenlib.h>
 8  #include <xspecial.h>
 9  #include <netdb.h>
10  #include <in.h>
11  #include <xpwd.h>
12  #include <fcs.h>
13
14  #include <ftp.h>
15  #include <socket.h>
16
17  extern  char version[];
18  extern  int xsmain();
19  extern  int lostconn();
20
21  extern struct   sckadr_in ctrl_addr;
22  struct  _xiobuf _xiob[_XNFILE] = {0};
23  struct passwd xpassword = {0};
24  struct passwd *pw = & xpassword;
25  struct _rcb _rcb[_XNFILE] = {0};                /* RSX - CONTROL BLOCK */
26  typedef int jmp_buf;
27  extern int errcatch;
28  jmp_buf *envptr = {0};
29  int xmodname();
30  int hash = 0;
31  int errno = 0;
32  int figit = 0;
33  int _brk = 0;                       /* USED BY C_RTS ALLOC & FREE */
34  char luntbl[32] = {0};              /* used to maintain LUN  */
35  extern char *inprm[];
36  char *xghome = (char * )0;
37  extern struct   hostent *hp ;
38
39  extern int      logged_in ;
40  extern int      debug ;
41  extern int      timeout ;
42  extern int      logging ;
43  extern int      guest ;
44  extern struct   servent *sp ;
45
46
47  main(argc, argv)
48          int argc;
49          char *argv[];
50  {
51          int ctrl, s, options = 0;
52          char *cp;
53          int rval;
54          int rval2;
55          int buf[16];                    /* 16 word bufferd used for task
56                                             parameters     */
```

```
57
58          int maxlun = 0;
59
60          emt(GTSK,buf);
61          _brk = buf[13];
62          maxlun = buf[8];
63          ppasc(pw->cur_uic, buf[7]);
64          ppasc(pw->login_uic, buf[15]);
65          emt(GLUN,1,buf);                        /* get lun information */
66          xstrncpy(pw->log_dev,buf,2);            /* get phy. device name */
67          pw->log_dev[2] = (*((char *)buf + 2 )) + 060; /* get unit no.    */
68          pw->log_dev[3] = '\0';                  /* make it string */
69          xstrcpy(pw->cur_dev,pw->log_dev);
70          while( maxlun ) {
71                  if( emt(GLUN, maxlun, buf ) > 0 )
72                      assign(maxlun);
73                  --maxlun;
74                  }
75          for( rval = 0; rval < _XNFILE ; ++ rval )
76                  _rcb[rval].flags = RFREE;
77          sp = gsbname("ftp","tcp");
78          errcatch = (int ) & envptr;
79          if (sp == 0) {
80                  xoprintf(xstderr, "ftpd: ftp/tcp: unknown service\n");
81                  xexit(1);
82          }
83          ctrl_addr.sin_port = xhtons(sp->s_port);
84          ctrl_addr.sin_family = AF_INET;
85          options = ( SO_ACCEPTCONN | SO_KEEPALIVE );
86          debug = 0;
87          argc--, argv++;
88          while (argc > 0 && *argv[0] == '-') {
89                  for (cp = &argv[0][1]; *cp; cp++) switch (*cp) {
90
91                  case 'v':
92                          debug = 1;
93                          break;
94
95                  case 'd':
96                          debug = 1;
97                          options |= SO_DEBUG;
98                          break;
99
100                 case 'l':
101                         logging = 1;
102                         break;
103
104                 case 't':
105                         timeout = xatoi(++cp);
106                         goto nextopt;
107                         break;
108
109                 default:
110                         xoprintf(xstderr, "Unknown flag -%c ignored.\n", *cp);
111                         break;
112                 }
```

```
113  nextopt:
114                  argc--, argv++;
115          }
116          rval = xdopen("TI:",XFREAD|XFWRITE|XFASCII, FILE_NAME);
117          xdup2(rval,1);
118          xdup2(1,2);
119          xodopen(2,"w");
120
121          getclient( SOCK_STREAM, (struct sockproto *)0,
122                  &ctrl_addr, options, xsmain);
123  }
124
125  xsmain( s, from )
126
127  /*
128  RSX specific ftp demon start up actions, calls generic code.
129  Operating systems which require the user's id to be established
130  before the process is started may provide two versions of this
131  module, one where logged_in is set to zero and one where it
132  is set to 1.
133  Otherwise, this is where to determine if the user has been authenticated
134  or not.
135  */
136  int s;
137  struct sckadr_in *from;
138  {
139          logged_in = 0;
140          /*
141          Until xprintf is available...
142          s = (int)_xiob[s]._sys_id;
143          */
144          ftpdoit( s, from );
145  }
```

```
1  char version[] = "Version 4.83 Wed Mar 27 11:34:00 PST 1985";
```

```
/*      convert floating point value to string
 *      return length of string converted
 */

/*
 *     remove the definition of  EXTENDED if you want
 *     the exponent of E format to be 2 character places
 *     instead of 3 (e.g.) 1.0E45 instead of 1.0E045
 */

#define EXTENDED

dtos(d,sbuff,prec,cc)
double d;                       /* floating point to convert */
unsigned char *sbuff;           /* buffer to store result */
short prec;                     /* no. fractional places */
unsigned char cc;               /* conversion code (e,f, or g) */
{
short base;                     /* the number base */
short efmt;                     /* true if E format required */
short len;                      /* length of string */
unsigned char *cp;

#ifdef  EXTENDED
#define PAD 3                   /* number of digits in exponent */
#else
#define PAD 2
#endif


  base = _dscale(&d,0);
  efmt = ((cc=='e')||((cc=='g')&&(base>=5||base<=-5))||(base>=20));
  base += _dscale(&d,efmt?prec+2:prec+base+2);
  if (base>=20) efmt=1;
  cp = sbuff + _dtos(d,sbuff,efmt?1:base+1,prec);
  if(efmt){
    *cp++ = 'E';
    if(base<0){*cp++ = '-'; base= -base;}
    else *cp++ = '+';
    if((len = ltos((long)base,cp,10))<PAD){     /* left pad */
        movmem(cp,cp+PAD-len,len+1);
        setmem(cp,PAD-len,'0');
    }
   }
   else if(cc=='g'){
     while(*--cp == '0')*cp = 0;        /* remove trailing zeroes */
     if(*cp == '.') *cp = 0;            /* remove '.' */
   }
   return xstrlen(sbuff);
}


/*      increased accuracy power of ten table
*/

static unsigned int pgiten[]={
```

```
57          0X0000,0X0000,0X0000,0X4024,            /* 1e1 */
58          0X0000,0X0000,0X0000,0X4059,            /* 1e2 */
59          0X0000,0X0000,0X8800,0X40C3,            /* 1e4 */
60          0X0000,0X0000,0XD784,0X4197,            /* 1e8 */
61          0X8000,0X37E0,0XC379,0X4341,            /* 1e16 */
62          0X6E17,0XB505,0XB8B5,0X4693,            /* 1e32 */
63          0XF9F6,0XE93F,0X4F03,0X4D38,            /* 1e64 */
64          0X1D33,0XF930,0X7748,0X5A82,            /* 1e128 */
65          0XBF3F,0X7F73,0X4FDD,0X7515             /* 1e256 */
66  };
67  static double *pgten=pgiten;
68
69  static unsigned int pliten[]={
70          0X999A,0X9999,0X9999,0X3FB9,            /* 1e-1 */
71          0X147B,0X47AE,0X7AE1,0X3F84,            /* 1e-2 */
72          0X432D,0XEB1C,0X36E2,0X3F1A,            /* 1e-4 */
73          0X8C3A,0XE230,0X798E,0X3E45,            /* 1e-8 */
74          0X89BC,0X97D8,0XD2B2,0X3C9C,            /* 1e-16 */
75          0XA732,0XD5A8,0XF623,0X3949,            /* 1e-32 */
76          0XA73C,0X44F4,0X0FFD,0X32A5,            /* 1e-64 */
77          0X979A,0XCF8C,0XBA08,0X255B,            /* 1e-128 */
78          0X6F40,0X64AC,0X0628,0X0AC8             /* 1e-256 */
79  };
80  static double *plten=pliten;
81
82  static double ZERO=0.0,ONE=1.0,TEN=10.0;
83
84  static _dscale(valp,round)
85  double *valp;                               /* value to scale */
86  int round;
87  {
88    int pow=0,sign,j,*ps,*pd;
89    double val,roundval;
90
91    if((val= *valp)<ZERO){
92          val= -val;
93          sign=1;
94    }
95    else sign=0;
96
97    if(val==ZERO || round<0)return 0;
98    if(round){
99      if(round>16) round = 16;              /* the real limit should be 15 ? */
100     for( roundval=5.0; --round;) roundval *= 1.0e-1;
101     val += roundval;
102   }
103   if(val>=TEN){
104     for(j=9;j--;){
105       pow<<=1;
106       if(val>=pgten[j]){
107         val *= plten[j];
108         ++pow;
109       }
110     }
111   } else if(val<ONE){
112     for(j=9;j--;){
```

```
113         pow<<=1;
114         if(val<plten[j]){
115           val *= pgten[j];
116           --pow;
117         }
118       }
119       if(val<ONE){
120         val*=TEN;
121         --pow;
122       }
123     }
124     roundval=0;
125     pd= &roundval;
126     ps= &val;
127     pd[3]=(ps[3]&0x7ff0)-(52<<4);
128     val+=roundval;
129     if(val>=TEN || val<ONE)pow+=_dscale(&val,0);
130     *valp=sign?-val:val;
131     return pow;
132   }
133
134   static _dtos(val,string,iplace,fplace)
135   double val;                       /* the value to convert */
136   unsigned char *string;
137   int iplace;                       /* number of integer places */
138   int fplace;                       /* the number of fractional places */
139   {
140     unsigned char *cp;
141     int j;
142
143     cp=string;
144     if(val<ZERO){
145       val= -val;
146       *cp++ = '-';
147     }
148     if(iplace<1){
149       *cp++ ='0';
150       *cp++ ='.';
151       fplace+=iplace;
152       if(fplace<0){iplace-=fplace;fplace=0;}
153       while(iplace++<0)*cp++ ='0';
154     } else {
155       do {
156         j=val;
157         *cp++ =j+'0';
158         val=(val-j)*TEN;
159       } while(--iplace);
160       if(fplace)*cp++ ='.';
161     }
162     while(fplace--){
163       j=val;                        /* get the integer part */
164       *cp++ = j+'0';
165       val=(val-j)*TEN;
166     }
167     *cp=0;
168     return cp-string;
```

```
169  }
```

```
 1  /*
 2   * @(#)fmtout.c 1.7 5/31/85
 3   *
 4   * GENERIC LIBRARY
 5   *
 6   * filename: FMTOUT.C
 7   */
 8
 9  /*      format data under control of a format string
10  */
11
12  /*      to remove the floating point code, comment out
13          the definition of FLOATS
14  */
15
16  /*
17    #define FLOATS
18   */
19
20  #include "xgenlib.h"
21
22  static  int *Pp = (int *)0;
23
24  #ifndef zilog
25
26  xpinit(svec)
27  int *svec;
28  {
29          Pp = svec;
30  }
31
32  xpint()
33  {
34          return *Pp++;
35  }
36
37  long
38  xplong()
39  {
40  #ifdef rsx
41          register long *p;
42          p = (long *)Pp;
43          Pp += sizeof ( long) / sizeof (Pp);
44          return( (*p));
45  #else
46          return *((long *)Pp)++;
47  #endif
48  }
49
50  typedef char *p2c;
51  typedef p2c *p2p2c;
52
53  char *
54  xpptr()
55  {
56          /* return *((p2p2c)Pp)++; */
```

```
57          register p2p2c cpp = (p2p2c)Pp;
58          register p2c cp = *cpp;
59
60          cpp++;
61          Pp = (int *)cpp;
62          return (cp);
63  }
64
65  double
66  xpdouble()
67  {
68  #ifdef rsx
69          register double *p;
70          p = (double *)Pp++;
71  #else
72          (double *)Pp++;
73  #endif
74          return (double)0;
75  }
76
77  #else
78
79  static  Rcnt;
80  static  int *Rvec;
81  static  int *Svec;
82
83  xpinit(rvec, rcnt, svec)
84  int *rvec;
85  int *svec;
86  {
87          Rcnt = rcnt;
88          Pp = Rvec = rvec;
89          Svec = svec;
90  }
91
92  xpint()
93  {
94          int itmp;
95
96          if (Rcnt == 6)
97                  Pp = Svec;
98          itmp = *Pp;
99          Rcnt++;
100         Pp++;
101         return itmp;
102 }
103
104 long
105 xpswap(lv)
106 long lv;
107 {
108         return lv<<16 | (lv>>16&0xFFFF);
109 }
110
111 long
112 xplong()
```

```
113  {
114          long ltmp;
115
116          switch (Rcnt) {
117          case 0:
118          case 2:
119                  ltmp = xpswap(*(long *)Pp);
120                  Pp += 2;
121                  Rcnt += 2;
122                  break;
123          case 1:
124                  Pp++;
125                  ltmp = xpswap(*(long *)Pp);
126                  Pp += 2;
127                  Rcnt += 3;
128                  break;
129          case 3:
130                  Pp++;
131                  ltmp = xpswap(*(long *)Pp);
132                  Pp = Svec;
133                  Rcnt += 3;
134                  break;
135          case 4:
136                  ltmp = xpswap(*(long *)Pp);
137                  Pp = Svec;
138                  Rcnt += 2;
139                  break;
140          case 5:
141                  Pp = Svec;
142                  ltmp = *(long *)Pp;
143                  Pp += 2;
144                  Rcnt += 3;
145                  break;
146          default:
147                  if (Rcnt == 6)
148                          Pp = Svec;
149                  ltmp = *(long *)Pp;
150                  Pp += 2;
151                  Rcnt += 2;
152                  break;
153          }
154          return ltmp;
155  }
156
157  char *
158  xpptr()
159  {
160          char *cptmp;
161
162          if (Rcnt == 6)
163                  Pp = Svec;
164          cptmp = (char *)*Pp;
165          Rcnt++;
166          Pp++;
167          return cptmp;
168  }
```

```
169
170  double
171  xpdouble()
172  {
173          return (double)0;
174  }
175
176  #endif
177
178  #ifndef zilog
179   fmtout(func, funarg, string, ip)
180  #else
181   fmtout(func, funarg, string, ip, regp, regcnt)
182  int *regp;
183  #endif
184  int (*func)();
185  char *funarg;
186  char *string;
187  int *ip;
188  {
189          char tbuff[128], *cp, cb;
190          int base;
191          int is_number;
192          unsigned leftadj, padchar, width, precflg, precisn, longflg, length;
193          union {
194                  long tlong;
195                  long tulong;
196          }
197          lw;
198  #ifdef  FLOATS
199          double *dp;
200  #endif
201
202  #ifndef zilog
203          xpinit(ip);
204  #else
205          xpinit(regp, regcnt, ip);
206  #endif
207
208          while(*string){
209                  if( *string !='%'){
210                          for(cp=string;*cp && *cp!='%';)
211                                  (*func)( (*cp++) & 0xff, funarg);
212                          string=cp;
213                  }
214                  else {
215                          is_number = 1;
216                          if (leftadj=(*++string == '-'))
217                                  ++string;
218                          padchar= *string & 0xff;
219                          if(padchar=='0')
220                                  ++string;
221                          else padchar=' ';
222                          if (*string == '*') {
223                                  /* width is an argument */
224                                  width = xpint();
```

```
225                             ++string;
226                     }
227                     else
228                             for (width=0;isdigit(*string);)
229                                     width=width*10+(*string++-'0');
230                     if (precflg=(*string=='.')) {
231                             ++string;
232                             if( *string == '*'){
233                                     /* precision is an argument */
234                                     precisn = xpint();
235                                     ++string;
236                             }
237                             else
238                                     for (precisn=0;isdigit(*string);)
239                                             precisn=precisn*10+(*string++-'0'
240                     }
241                     else precisn=0;
242                     if (longflg=(*string =='1'))
243                             ++string;
244
245                     switch (*string) {
246     #ifdef FLOATS
247                     case 'e':
248                     case 'f':
249                     case 'g':
250                             if(!precflg)precisn=6;
251                             dp= (double *)xpdouble();
252                             length=dtos(*dp++, cp=tbuff, precisn, *string & (
253                             break;
254     #endif
255                     case 'B':
256                     case 'b':
257                             base=2;
258                             goto nosign;
259                     case 'O':
260                     case 'o':
261                             base=8;
262                             goto nosign;
263                     case 'U':
264                     case 'u':
265                             base=10;
266                             goto nosign;
267                     case 'X':
268                     case 'x':
269                             base=16;
270                             goto nosign;
271                     case 'D':
272                     case 'd':
273                             base= -10;
274     nosign:
275                             if (!longflg)
276                                     longflg=(*string>='A'&&*string<='Z');
277                             if(longflg){
278                                     lw.tlong= xplong();
279                             }
280                             else if(base<0)lw.tlong=(long)xpint();
```

```
281                                     else lw.tulong=(unsigned)(xpint());
282                                     ltos(lw.tlong, tbuff, base);
283                                     if(precflg){
284                                             cp = tbuff;
285                                             if (lw.tlong <0) ++cp;
286                                             length= xstrlen(cp);
287                                             if (precisn && length < precisn+1 ) {
288                                                     movmem(cp, cp+precisn+1-length,
289                                                     setmem(cp, precisn+1-length, '0'
290                                                     length = precisn +1;
291                                             }
292                                             movmem( cp+length-precisn, cp+length-pre
293                                             cp[length-precisn] = '.';
294                                     }
295                                     length=xstrlen(cp=tbuff);
296                                     break;
297                             case 's':
298                                     cp = xpptr();
299                                     length = xstrlen(cp);
300                                     if(precflg && precisn<length)length=precisn;
301                                     /* leave minus signs alone */
302                                     is_number=0;
303                                     break;
304                             case 'c':
305                                     cb = xpint() & 0x7F;
306                                     cp = &cb;
307                                     length=1;
308                                     is_number=0;
309                                     break;
310                             default:
311                                     cp=string;
312                                     length=1;
313                                     is_number=0;
314                                     break;
315                             }
316                             if (!leftadj && width>length) {
317                                     if (is_number && *cp == '-' && padchar == '0') {
318                                             (*func)((*cp++) & 0xff, funarg);
319                                             --length;
320                                             --width;
321                                     }
322                                     while (width-- >length)
323                                             (*func)(padchar, funarg);
324                             }
325                             if (width>length)
326                                     width-=length;
327                             else
328                                     width=0;
329                             while (length--)
330                                     (*func)((*cp++) & 0xff, funarg);
331                             if (leftadj && width)
332                                     while(width--)
333                                             (*func)(padchar, funarg);
334                             ++string;
335                     }
336             }
```

```
337  }
```

```
 1
 2  /*
 3   * @(#)ltos.c   1.6 6/4/85
 4   * GENERIC LIBRARY
 5   *
 6   * filename: LTOS.C
 7   */
 8
 9  #include "xgenlib.h"
10
11  /*
12   *      movmem  -> move memory
13   */
14
15  int movmem ( from, to, len)
16    char *from, *to;
17    int len;                    /* no of bytes to copy */
18    {
19     while ( len-- )
20       *to++ = *from++;
21    }
22
23  /*
24   * setmem: - set memory to a desired value
25   */
26
27  int setmem( s, len, c)
28    char *s;              /* start address of the memory */
29    int len;              /* no of char to be set        */
30    char  c;              /* character to be set         */
31    {
32     while ( len-- )
33      *s++ = c;
34    }
35
36
37  /*      long to string
38  */
39
40  ltos(val,cp,base)
41  long val;                         /* the number to convert */
42  char *cp;                 /* the address of the string */
43  int base;                         /* the conversion base */
44  {
45    char tempc[34],*tcp;
46    int n=0;                        /* number of characters in result */
47    long uval;              /* unsigned value (not possible on all compilers) */
48    static char dig[]={"0123456789ABCDEF"};
49
50    *(tcp=tempc+33)=0;
51    if(base<0){                     /* needs signed conversion */
52      if(val<0)n=1;
53      else val= -val;
54      do {
55        register int num = -(val%base);
56        *--tcp=dig[ num ];
```

```
57        } while((val/= -base));
58      } else {
59        uval=val;
60        do {
61          register long num = (uval%base);
62          if( num < 0 ) num -= base;
63          *--tcp=dig[num];
64        } while(uval/= base);
65      }
66      if(n)*--tcp='-';
67      n=((int)tempc + 33) - (int)tcp;
68      movmem(tcp,cp,n+1);
69      return n;
70    }
71
72
```

```
 1  /*
 2   *      format data to a memory string
 3   */
 4
 5  /*
 6   * this function behaves much like putc with only difference
 7   * is that it writes to an user buffer instead of a file.
 8   */
 9  static _store(x, to)
10  unsigned char x;
11  unsigned char **to;
12  {
13    **to = x;
14    (*to)++;              /* point to the next location of the buffer */
15    **to=0;               /* terminate for safety */
16  }
17
18  xsprintf(string,control,args)
19  unsigned char *string;
20  unsigned char *control;
21  unsigned args;
22  {
23    return     _fmtout(_store,&string,control,&args);
24  }
25
```

```
 1  /*
 2   * GENERIC LIBRARY
 3   *
 4   * filename: STRETC.C
 5   */
 6
 7  #include <xctype.h>
 8
 9
10  /*
11   * XSTRLEN: return the size of the string
12   */
13
14  int xstrlen( s )
15    unsigned char *s;
16    {
17     unsigned char *p = s;
18
19     while ( *p != '\0' ) p++;
20     return ( p-s );
21    }
22
23  /*
24   * XSTRCPY: copy string 2 to string 1
25   */
26
27  xstrcpy( s, t)
28    unsigned char *s, *t;
29    {
30      while ( *s++ = *t++ );
31    }
32
33  /*
34   * XSTRNCPY: copy a string into a buffer n characters in length
35   */
36
37  unsigned char *xstrncpy( s, t, n)
38    unsigned char *s, *t;
39    int n;
40    {
41     unsigned char *cp;
42
43     for ( cp = s; n && ( *cp++ = *t++ ); --n );
44     while ( n-- )
45       *cp++ = '\0';
46     return ( s );
47    }
48
49
50  /*
51   * XSTRCMP: compare strings and return -ve, 0 or +ve accordingly
52   */
53
54  int xstrcmp( s, t)  /* return <0 if s<t, 0 if s==t, >0 if s>t */
55    unsigned char *s, *t;
56    {
```

```
57      for ( ; *s == *t; s++, t++ )
58        if ( *s == '\0' )
59          return ( 0 );
60      return ( *s - *t );
61     }
62
63   /*
64    * XSTRICMP: case insensitive string comparision
65    */
66
67   #define conv(x) ( isupper(x) ? _tolower(x) : x )
68
69   int xstricmp( s, t)
70     unsigned char *s, *t;
71     {
72      for ( ; conv(*s) == conv(*t); s++, t++ )
73        if ( *s == '\0' )
74          return ( 0 );
75      return ( conv(*s) - conv(*t) );
76     }
77
78
79   /*
80    * XSTRNCMP: string compare up to n characters
81    */
82
83   int xstrncmp( s, t, n)
84     unsigned char *s, *t;
85     int n;
86     {
87      for ( ; n-- && ( *s == *t ); t++ )
88        if ( !*s++ )
89          return ( 0 );
90      if ( n < 0 )
91        return ( 0 );
92      if ( *s < *t )
93        return ( -1 );
94      return ( 1 );
95     }
96
97   /*
98    * XSTRCAT: concatenates string 2 to the end of string 1
99    */
100
101  int xstrcat( s, t )
102    unsigned char *s, *t;
103    {
104     while ( *s++ != '\0' );
105     for ( --s; (*s++ = *t++) != '\0'; );
106    }
107
108
109  /*
110   * XSTRNCAT: concatenate string 2 to string 1 , max n chars
111   */
112
```

```
113  unsigned char *xstrncat( s, t, n)
114    unsigned char *s, *t;
115    int n;
116    {
117      unsigned char *cp;
118
119      for ( cp = s; *cp++; );
120      for ( --cp; n-- && ( *cp++ = *t++ ); ) ;
121      if ( n < 0 )
122        *cp = '\0';
123      return s;
124     }
125
126
127  /*
128   * XSTRCHR: return a pointer to first occurance of character
129   */
130
131  unsigned char *xstrchr( s, c)
132    unsigned char *s, c;
133    {
134     while ( *s )
135       if ( *s++ == c )
136         return --s;
137     return ( 0 );
138    }
139
140  /*
141   * XSTRRCHR: return a pointer to the last occurance of char
142   */
143
144  unsigned char *xstrrchr( s, c)
145    unsigned char *s, c;
146    {
147     unsigned char *cp;
148
149     for ( cp = s + xstrlen(s); --cp >= s; )
150       if ( *cp == c )
151         return cp;
152     return 0;
153    }
```

```
 1  /*
 2  %W% %G%
 3
 4  Temporary entry points for some x routines.
 5  */
 6  #include <xstdio.h>
 7  extern int errno;
 8
 9
10
11  char *xatoi(a)
12    char *a ;
13  {
14
15  return( atoi( a ) );
16  }
17
18  char *
19  xsprintf( )
20  {
21  /*
22  keep the linker happy at least.
23  */
24
25  }
```

```
 1  /*
 2  %W% %G%
 3
 4  Generic close.
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xerrno.h>
 9  #include <stdio.h>
10
11  extern int xnofunc();
12
13  xclose( fd )
14
15  int fd;
16  {
17  int rval;
18  register XFILE *current;
19
20  if( fd >= 0 && fd < _XNFILE )
21          {
22          current = &_xiob[fd];
23          if( !( current->_flag & _XUsed ))
24                  {
25                  fprintf( stderr, "xclose: bad od\n" );
26                  return ( XEBADF );
27                  }
28          /*
29          file descriptor is OK, check for copies.
30          */
31          if( (current->_flag & _XIOWRT ) && (current->_bufsiz > current->_cnt) )
32                  {
33                  /*
34                  Flush write buffer (if appropriate).
35                  */
36                  fprintf( stderr, "xclose: flushing\n" );
37                  xfflush( current );
38                  }
39          if( current->_base &&
40                  (( current->_flag & _XIOMYBUF == 0 )))
41                  {
42                  /*
43                  free buffer allocated by system.
44                  */
45                  fprintf( stderr, "xclose: freeing\n" );
46                  xfree( current->_base );
47                  }
48          if( current->_pred )
49                  {
50                  /*
51                  at least one copy exists
52                  */
53                  struct _xiobuf *next = current->_succ;
54                  struct _xiobuf *previous = current->_pred;
55                  int primary = current->_flag & _XPrimary;
56
```

```
57                      fprintf( stderr, "xclose: uncopying\n" );
58                      if( next == previous )
59                              {
60                              /*
61                              only one other copy
62                              */
63                              previous->_pred = (struct _xiobuf *)0;
64                              previous->_succ = (struct _xiobuf *)0;
65                              }
66                      else
67                              {
68                              /*
69                              remove from linked list
70                              */
71                              previous->_succ = next;
72                              next->_pred = previous;
73                              }
74                      if( primary )
75                              {
76                              /*
77                              Make new primary copy.
78                              */
79
80                              previous->_flag |= _XPrimary;
81                              }
82                      rval = 0;
83                      }
84              else
85                      {
86                      /*
87                      Only copy, perform real close
88                      */
89                      fprintf( stderr, "xclose: closinging\n" );
90                      rval = (*(current->_close))( current->_sys_id );
91                      }
92              /*
93              Cleanup _xiob structure.
94              */
95              current->_flag = 0;
96              current->_cnt = 0;
97              current->_ptr = (char *)0;
98              current->_base = (char *)0;
99              current->_bufsiz = 0;
100             current->_succ = (struct _xiobuf *)0;
101             current->_pred = (struct _xiobuf *)0;
102             current->_read = xnofunc;
103             current->_write = xnofunc;
104             current->_ioctl = xnofunc;
105             current->_close = xnofunc;
106             return( rval );
107             }
108     return( XEBADF );
109     }
```

```
 1  /*
 2  %W% %G%
 3
 4  Copy an EXOS file object.
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xerrno.h>
 9
10  xdup2( orig_fd, new_fd )
11
12  int orig_fd;
13  int new_fd;
14  {
15
16  if( orig_fd > 0 && orig_fd < _XNFILE && new_fd > 0 && new_fd < _XNFILE )
17          {
18          register struct _xiobuf *new = &_xiob[new_fd];
19          register struct _xiobuf *orig = &_xiob[orig_fd];
20
21          if( !(orig->_flag & _XUsed) )
22                  return( XEBADF );
23          if( orig_fd == new_fd )
24                  return( 0 );
25          xclose( new_fd );       ---->   ??
26          /*
27          seperate buffering for new object,
28          everything else identical.
29          */
30          new->_flag = orig->_flag &
31                  ~(_XPrimary | _XIOMYBUF | _XIOLBF | _XIONBF);
32          new->_cnt = 0;
33          new->_ptr = (char *)0;
34          new->_base = (char *)0;
35          new->_file = new_fd;
36          new->_sys_id = orig->_sys_id;
37          new->_close = orig->_close;
38          new->_read = orig->_read;
39          new->_ioctl = orig->_ioctl;
40          new->_write = orig->_write;
41          /*
42          insert into linked list of copies
43          */
44          if ( !orig->_succ )
45                  {
46                  new->_succ = orig;
47                  new->_pred = orig;
48                  orig->_succ = new;
49                  orig->_pred = new;
50                  }
51          else
52                  {
53                  new->_succ = orig->_succ;
54                  new->_pred = orig;
55                  orig->_succ = new;
56                  }
```

```
57          return( 0 );
58          }
59  return( XEBADF );
60  }
```

```
 1  /*
 2  %W% %G%
 3
 4  Unix specific close all EXOS file objects and exit program.
 5  */
 6
 7  #include <xstdio.h>
 8
 9  xexit( status )
10
11  int status;
12  {
13  int i;
14
15  for( i = 0; i < _XNFILE ; ++i )
16          {
17          xclose( i );
18          }
19  exit( status );
20  }
```

```
 1  /*
 2  %W% %G%
 3
 4  Operating system independent routine for flushing buffers
 5  Associated with pointers to io objects.
 6  */
 7  #include <xstdio.h>
 8  #include <xerrno.h>
 9
10  xfflush( file )
11
12  register XFILE *file;
13  {
14  int rval;
15  int nmtowrite;
16  char *pt;
17
18  if( !(file->_flag & _XUsed ) )
19          return( XEBADF );
20  if( !file->_base || !(file->_flag & _XIOWRT ) )
21          return( XEBADF );
22  if( file->_flag & _XIOLBF )
23          {
24          nmtowrite = (int)( file->_ptr - file->_base );
25          file->_cnt = 0;
26          }
27  else
28          {
29          nmtowrite = file->_bufsiz - file->_cnt;
30          file->_cnt = file->_bufsiz;
31          }
32  file->_ptr = file->_base;
33  rval = 0;
34  pt = file->_base;
35  while( nmtowrite > 0 )
36          {
37          rval = xwrite( xfileno( file ), pt, nmtowrite );
38          if( rval <= 0 )
39                  break;
40          nmtowrite -= rval;
41          pt += rval;
42          }
43  return( rval );
44  }
```

```
 1  /*
 2  @(#)_xfilbuf.c  1.4 5/22/85
 3
 4  System independent routine for filling buffers associated with
 5  a pointer to an io object.  (used to implement xgetc).
 6  */
 7  #include "xgenlib.h"
 8
 9  _xfilbuf( file )
10
11  register XFILE *file;
12  {
13  int rval;
14  char ch;
15
16  if( !( file->_flag & _XUsed ) )
17          return( XEOF );
18  if( !( file->_flag & _XIOREAD ) )
19          return( XEOF );
20  if( file->_flag & _XIOERR )
21          {
22          /*
23          * Allow user to retry after an error.
24          */
25          file->_flag &= ~_XIOERR;
26          }
27  if( file->_base )
28          {
29          rval = xread( xfileno(file), file->_base, file->_bufsiz );
30          }
31  else
32          {
33          rval = xread( xfileno(file), &ch, 1 );
34          }
35  if( rval < 0 )
36          {
37          file->_flag |= _XIOERR;
38          return( XEOF );
39          }
40  else if ( rval == 0 )
41          {
42          file->_flag |= _XIOEOF;
43          return( XEOF );
44          }
45  file->_cnt = rval - 1;
46  if( file->_base )
47          {
48          file->_ptr = &file->_base[1];
49          return( (file->_base[0]) & 0xff );
50          }
51  else
52          {
53          return( ch & 0xff );
54          }
55  }
```

```
 1  /*
 2  @(#)_xflsbuf.c  1.6 5/22/85
 3
 4  System independent routine for filling io buffers.
 5  ( used to implement xputc ).
 6  */
 7  #include "xgenlib.h"
 8
 9  _xflsbf( x, file )
10
11  unsigned int x;
12  register XFILE *file;
13  {
14  int rval;
15  int nmtowrite;
16  int storex;
17  char xch;
18  char *pt;
19
20  if( !(file->_flag & _XUsed) )
21          return( XEBADF );
22  if( !(file->_flag & _XIOWRT ) )
23          return( XEBADF );
24  if( file->_flag & _XIOERR )
25          {
26          /*
27           * Allow user to retry after an error.
28           */
29           file->_flag &= ~_XIOERR;
30           }
31  if( file->_base )
32          {
33          storex = 1;             /* put x in buffer after flush */
34          pt = file->_base;
35          /*
36          Check for line buffering
37          */
38          if( file->_flag & _XIOLBF )
39                  {
40                  nmtowrite = (int)(file->_ptr - file->_base);
41                  if( nmtowrite >= file->_bufsiz )
42                          {
43                          /*
44                          flush buffer, because it is full.
45                          */
46                          }
47                  else if( x == '\n' || x == '\r' )
48                          {
49                          /*
50                          flush buffer, because of end of line
51                          */
52                          storex = 0;
53                          *(file)->_ptr++ = x;
54                          ++nmtowrite;
55                          }
56                  else
```

```
57                              {
58                              *(file)->_ptr++ = x;
59                              file->_cnt = 0;
60                              return( 0 );
61                              }
62                      }
63              else
64                      {
65                      nmtowrite = file->_bufsiz - ( file->_cnt + 1 );
66                      }
67              while ( nmtowrite > 0 )
68                      {
69                      rval = xwrite( xfileno(file), pt, nmtowrite );
70                      if( rval < 0 )
71                              {
72                              file->_flag |= _XIOERR;
73                              return( rval );
74                              }
75                      if( rval == 0 )
76                              {
77                              file->_flag |= _XIOERR;
78                              return( XEIO );
79                              }
80                      nmtowrite -= rval;              /* Assert: rval <= nmtowrite */
81                      pt += rval;
82                      }
83              if( file->_flag & _XIOLBF )
84                      {
85                      if( storex && ( x == '\n' || x == '\r' ) )
86                              {
87                              /*
88                              write out carriage return
89                              */
90                              xch = x;
91                              rval = xwrite( xfileno(file), &xch, 1 );
92                              storex = 0;
93                              if( rval < 0 )
94                                      return( rval );
95                              }
96                      file->_cnt = 0;
97                      }
98              else
99                      file->_cnt = file->_bufsiz - 1; /* _cnt == #chars remaining,
100                                                             -1 for "x" */
101             if ( file->_cnt < 0 )
102                     {
103                     /*
104                     This should not happen.
105                     */
106                     return( XEFAULT );
107                     }
108             file->_ptr = file->_base;
109             if( storex )
110                     *file->_ptr++ = x;
111             }
112     else
```

```
113             {
114             xch = x;
115             rval = xwrite( xfileno(file), &xch, 1 );
116             if( rval < 0 )
117                     {
118                     file->_flag |= _XIOERR;
119                     return( rval );
120                     }
121             if( rval != 1 )
122                     {
123                     file->_flag |= _XIOERR;
124                     return( XEIO );
125                     }
126             file->_cnt = 0;
127             }
128     return( 0 );
129     }
```

```
 1  /*
 2   * @(#)xfprintf.c       1.6 5/31/85
 3
 4   Xfprintf(3X).
 5  */
 6  #include "xgenlib.h"
 7
 8  static int _xputc( c, op)
 9     char c;
10     XFILE *op;
11     {
12      xputc( c & 0xff, op);
13     }
14
15  #ifndef zilog
16
17  xoprintf(op,control,args)
18  XFILE *op;
19  char *control;          /* the format control string */
20  unsigned args;
21  {
22
23    return _fmtout(_xputc,op,control,&args);
24  }
25
26  xprintf( control, args )
27  char *control;
28  unsigned args;
29  {
30
31          return _fmtout(_xputc, xstdout, control, &args );
32  }
33
34  #else
35
36  xoprintf(op, control, a1, a2, a3, a4, args)
37  XFILE *op;
38  char *control;          /* the format control string */
39  int a1, a2, a3, a4;
40  unsigned args;
41  {
42          int s1=a1, s2=a2, s3=a3, s4=a4;
43
44          return _fmtout(_xputc, op, control, &args, &s1, 2);
45  }
46
47  xprintf(control, a1, a2, a3, a4, a5, args)
48  char *control;
49  int a1, a2, a3, a4, a5;
50  unsigned args;
51  {
52          int s1=a1, s2=a2, s3=a3, s4=a4, s5=a5;
53
54          return _fmtout(_xputc, xstdout, control, &args, &s1, 1);
55  }
56
```

```
    57  #endif
```

```
 1  /*
 2  %W% %G%
 3
 4  Print password prompt, turn off echoing, and get password, restore terminal,
 5  Using the facilities of xoslib.
 6
 7  Caveat: This assumes xstdin and xstdout == user's terminal.
 8  */
 9  #include <xstdio.h>
10
11  #define MXPWORD 25
12
13  char *
14  xgetpass( prompt )
15
16  char *prompt;
17  {
18  static char buf[ MXPWORD ];
19  register char *pt = &buf[0];
20  register int rval;
21  register int i = 0;
22
23  /*
24  Should use xprintf, but not available now...
25  */
26  while( *prompt != '\0' )
27          {
28          xputchar( *prompt++ );
29          }
30  xfflush( xstdout );
31  xraw_term( xnofunc );
32  do {
33          rval = xread( 0, pt, 1 );
34          if( rval < 0 )
35                  xperror( rval, "xgetpass" );
36          } while ( *pt != '\r' && *pt != '\n' &&
37                  rval == 1 && ++i < MXPWORD && ++pt  );
38  *pt = '\0';
39  xrestore_term();
40  return( &buf[0] );
41  }
```

```
 1  /*
 2  @(#)xgets.c     1.4 5/22/85
 3
 4  Xgets(3X).
 5  */
 6  #include "xgenlib.h"
 7
 8  char *
 9  xgets( string )
10
11  char *string;
12  {
13  int c;
14  char *p = string;
15
16  while ( (c = xgetchar()) != XEOF && c != '\n' )
17          {
18          *p++ = c;
19          }
20  *p = '\0';
21  if( c == XEOF )
22          return( XNULL );
23  return( string );
24  }
```

```
 1  /*
 2  %W% %G%
 3
 4  Generic io control.
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xerrno.h>
 9
10  xioctl( fd, cmd, param )
11
12  int fd;
13  int cmd;
14  char *param;
15  {
16  int rval;
17  register XFILE *file;
18
19  if( fd > 0 && fd < _XNFILE )
20          {
21          file = &_xiob[fd];
22          if( !( file->_flag & _XUsed ))
23                  return ( XEBADF );
24          /*
25          file descriptor is OK.
26          */
27          rval = (*(file->_ioctl))( file->_sys_id, cmd, param );
28          return( rval );
29          }
30  return( XEBADF );
31  }
```

4/11/85

```
 1  /*
 2  %W% %G%
 3
 4  Xmkarglist from xglob(3X) for Unix.
 5  This file belongs in Xoslib, but is here to keep the linker happy.
 6  */
 7  #define ARGPOINTERSP    200     /* bytes for storing argument pointers */
 8  #define ARGSPACE        400     /* bytes for storing arguments */
 9
10  static char *argbase;
11  static char *stringbase;
12
13  extern char *xmalloc(); /* x   #include
14
15  char **
16  xmkarglist( line, count )
17
18  char *line;             /* IN */
19  int *count;             /* OUT */
20  {
21  char **argp;
22  char *slurpstring();
23  char *argvsp;
24  int margc;
25
26  margc = 0;
27  /*
28  Allocate space for argv and tokens in line
29  */
30  if( xstrlen( line ) > ARGSPACE )
31          {
32          return( (char **)0 );
33          }
34  argvsp = xmalloc( ARGPOINTERSP + ARGSPACE );
35  if( argvsp == (char *)-1 )      < (char *)0 )
36          {
37          return( (char **)0 );
38          }
39  argbase = &argvsp[ARGPOINTERSP];        /* store from first of buffer */
40  stringbase = line;                      /* scan from first of buffer */
41  argp = (char **)argvsp;
42  while (*argp++ = slurpstring())
43          margc++;
44  *count = margc;
45  return( (char **)argvsp );
46  }
47
48  /*
49   * Parse string into argbuf;
50   * implemented with FSM to
51   * handle quoting and strings
52   */
53  char *
54  slurpstring()
55  {
56          int got_one = 0;
```

```
57          register char *sb = stringbase;
58          register char *ap = argbase;
59          char *tmp = argbase;            /* will return this if token found */
60
61          /*
62          Used to return '!' for shell event processing...
63          Ignore significance of '!'.
64          */
65  S0:
66          switch (*sb) {
67
68          case '\0':
69                  goto OUT;
70
71          case ' ':
72          case '\t':
73                  sb++; goto S0;
74
75          default:
76                  goto S1;
77          }
78
79  S1:
80          switch (*sb) {
81
82          case ' ':
83          case '\t':
84          case '\0':
85                  goto OUT;       /* end of token */
86
87          case '\\':
88                  sb++; goto S2;  /* slurp next character */
89
90          case '"':
91                  sb++; goto S3;  /* slurp quoted string */
92
93          default:
94                  *ap++ = *sb++;  /* add character to token */
95                  got_one = 1;
96                  goto S1;
97          }
98
99  S2:
100         switch (*sb) {
101
102         case '\0':
103                 goto OUT;
104
105         default:
106                 *ap++ = *sb++;
107                 got_one = 1;
108                 goto S1;
109         }
110
111 S3:
112         switch (*sb) {
```

```
113
114             case '\0':
115                     goto OUT;
116
117             case '"':
118                     sb++; goto S1;
119
120             default:
121                     *ap++ = *sb++;
122                     got_one = 1;
123                     goto S3;
124             }
125
126     OUT:
127             if (got_one)
128                     *ap++ = '\0';
129             argbase = ap;                   /* update storage pointer */
130             stringbase = sb;                /* update scan pointer */
131             if (got_one)
132                     return(tmp);
133             return((char *)0);
134     }
135
136     xdealglob( pt )
137     /*
138     Free space allocated by either xglob or xmkarglist
139     */
140
141     char **pt;
142     {
143
144     xfree( (char *)pt );
145     }
```

update Mar 30, 85
not fully -

```
 1  /*
 2  %W% %G%
 3
 4  Unix routine to form path names relative to the user's home directory,
 5  current directory, etc.
 6  This routine belongs in Xoslib, but is here to keep the linker happy.
 7  */
 8  #include <xspecial.h>
 9  #include <xerrno.h>
10  /*
11  for now ...
12  */
13  #define xsprintf sprintf
14
15  #include <pwd.h>
16
17  xmodname( name, special, buf, sz_buf )
18
19  char **name;
20  int special;
21  char *buf;
22  int sz_buf;
23  {
24  int rval;
25  char *pt = buf;
26
27  switch( special ) {
28          case FILE_NAME:
29                  pt = *name;
30                  break;
31          case CURRENT_DIR:
32                  buf[0] = '.';
33                  buf[1] = '\0';
34                  break;
35          case UP_DIRECTORY:
36                  buf[0] = '.';
37                  buf[1] = '.';
38                  buf[3] = '\0';
39          case HM_RELATIVE:
40          case HOME_DIR:
41                  {
42                  struct passwd *pwent;
43                  extern struct passwd *getpwuid();
44                  int uid;
45
46                  uid = getuid();
47                  pwent = getpwuid( uid );
48                  if( pwent == (struct passwd *)0 )
49                          return( XEPERM );
50                  if ( special == HM_RELATIVE )
51                          {
52                          /*
53                          Check for enough room in buffer,
54                          concatenate home directory and *name.
55                          */
56                          if( (2 + xstrlen( pwent->pw_dir ) + xstrlen( *name ))
```

Not required for RSX -

```
57                                      > sz_buf )
58                                      return( XE2BIG );
59                              xsprintf( buf, "%s/%s", pwent->pw_dir, *name );
60                              }
61                      else
62                              {
63                              /*
64                              Use home directory.
65                              */
66                              pt = pwent->pw_dir;
67                              }
68                      }
69                      break;
70              case CD_RELATIVE:
71                      /*
72                      Check for enough room in buffer,
73                      prepend "./" to *name.
74                      */
75                      if( (3 + xstrlen( *name )) > sz_buf )
76                              return( XE2BIG );
77                      xsprintf( buf, "./%s", *name );
78                      break;
79              default:
80                      return( XEINVAL );
81              }
82      *name = pt;
83      return( 0 );
84      }
```

```
 1  /*
 2  %W% %G%
 3
 4  Given an index to an open io object,
 5  Associate a buffer with it and return a pointer.
 6  */
 7  #include <xstdio.h>
 8  #include <stdio.h>
 9
10  extern char *xmalloc();
11
12  XFILE *xodopen( od, direction )
13
14  int od;          /* object descriptor */
15  char *direction;        /* "r" for read, "w" for write */
16  {
17  XFILE *file;
18  char *rval;
19
20  if( od < 0 || od >= _XNFILE )
21          {
22          fprintf( stderr, "bad od\n" );
23          return( (XFILE *)XNULL );
24          }
25  file = &_xiob[od];
26  /*
27  Make sure object has been opened, and in the right direction.
28  */
29  if( !(file->_flag & _XUsed) )
30          {
31          fprintf( stderr, "not used\n" );
32          return( (XFILE *)XNULL );
33          }
34  switch ( direction[0] ) {
35          case 'r':
36                  if( file->_flag & _XIOREAD )
37                          {
38                          file->_flag &= ~_XIOWRT;
39                          break;
40                          }
41                  if( file->_flag & _XIORW )
42                          {
43                          file->_flag &= ~_XIOWRT;
44                          file->_flag |= _XIOREAD;
45                          break;
46                          }
47                  fprintf( stderr, "not readable\n" );
48                  return( (XFILE *)XNULL );
49          case 'w':
50                  if( file->_flag & _XIOWRT )
51                          {
52                          file->_flag &= ~_XIOREAD;
53                          break;
54                          }
55                  if( file->_flag & _XIORW )
56                          {
```

```
57                          file->_flag &= ~_XIOREAD;
58                          file->_flag |= _XIOWRT;
59                          break;
60                          }
61                  fprintf( stderr, "not writeable\n" );
62                  return( (XFILE *)XNULL );
63          default:
64                  fprintf( stderr, "not an option\n" );
65                  return( (XFILE *)XNULL );
66          }
67  rval = xmalloc( XBUFSIZ );
68  if( rval == (char *)-1 )
69          {
70          fprintf( stderr, "no memory\n" );
71          return( (XFILE *)XNULL );
72          }
73  xsetbuf( file, rval, XBUFSIZ );
74  file->_flag &= ~_XIOMYBUF;              /* for automatic deallocation */
75  return( file );
76  }
```

1002

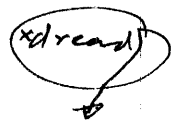
```
 1  /*
 2  %W% %G%
 3
 4  Xogets(3X).
 5  */
 6  #include <xstdio.h>
 7
 8  char *
 9  xogets( string, n, stream )
10
11  char *string;
12  int n;
13  XFILE *stream;
14  {
15  int c;
16  char *p = string;
17
18  while ( (c = xgetc(stream)) != XEOF && c != '\n' && --n > 0 )
19          {
20          *p++ = c;
21          }
22  *p = '\0';
23  return( string );
24  }
```

```
 1  /*
 2  @(#)xperror.c   1.4 6/4/85
 3
 4  Xperror(3X) and xrerror.
 5  */
 6  #include "xgenlib.h"
 7
 8  #define MINEXERR 0
 9
10  char *x_errlist[] = {
11  " No Error ",
12  " Not super-user ",
13  " No such file or directory ",
14  " No such process ",
15  " interrupted system call ",
16  " I/O error ",
17  " No such device or address ",
18  " Arg list too long ",
19  " Exec format error ",
20  " Bad file number ",
21  " No children ",
22  " No more processes ",
23  " Not enough core ",
24  " Permission denied ",
25  " Bad address ",
26  " Block device required ",
27  " Mount device busy ",
28  " File exists ",
29  " Cross-device link ",
30  " No such device ",
31  " Not a directory ",
32  " Is a directory ",
33  " Invalid argument ",
34  " File table overflow ",
35  " Too many open files ",
36  " Not a typewriter ",
37  " Text file busy ",
38  " File too large ",
39  " No space left on device ",
40  " Illegal seek ",
41  " Read only file system ",
42  " Too many links ",
43  " Broken pipe ",
44  " Argument too large ",
45  " Result too large ",
46  " Operation would block",
47  " Operation now in progress",
48  " Operation already in progress",
49  " Socket operation on non-socket",
50  " Destination address required",
51  " Message too long",
52  " Protocol wrong type for socket",
53  " Protocol not available",
54  " Protocol not supported",
55  " Socket type not supported",
56  " Operation not supported on socket",
```

```
 57  " Protocol family not supported",
 58  " Address family not supported by protocol family",
 59  " Address already in use",
 60  " Can't assign requested address",
 61  " Network is down",
 62  " Network is unreachable",
 63  " Network dropped connection on reset",
 64  " Software caused connection abort",
 65  " Connection reset by peer",
 66  " No buffer space available",
 67  " Socket is already connected",
 68  " Socket is not connected",
 69  " Can't send after socket shutdown",
 70  " Too many references: can't splice",
 71  " Connection timed out",
 72  " Connection refused",
 73  " Too many levels of symbolic links",
 74  " File name too long",
 75  " Host is down",
 76  " No route to host",
 77  };
 78
 79  static char bad_err[] = "UNKNOWN ERROR";
 80  char *xrerror();
 81  extern char *xsyserr();
 82
 83  #define MAXEXERR 65
 84
 85  xperror( eval, rname )
 86
 87  int eval;
 88  char *rname;
 89  {
 90  int len;
 91  int olderrno;
 92  char *estring;
 93
 94  olderrno = -eval;
 95  len = xstrlen( rname );
 96  if ( len > 0 )
 97          {
 98          if (xwrite( 2, rname, len ) != len )
 99                  return;
100          if (xwrite( 2, ":", 1 ) != 1 )
101                  return;
102          }
103  estring = xrerror( eval );
104  len = xstrlen( estring  );
105  xwrite( 2, estring , len );
106  xwrite( 2, "\n", 1 );
107  }
108
109
110  char *
111  xrerror( eval )
112
```

```
113  int eval;
114  {
115  int olderrno;
116
117  olderrno = -eval;
118  if ( eval <= XSYSERR )
119          {
120          return( xsyserr() );
121          }
122  else if( olderrno < MINEXERR || olderrno > MAXEXERR )
123          {
124          /*
125          bad error number
126          */
127          return( bad_err );
128          }
129  return( x_errlist[ olderrno - MINEXERR ] );
130  }
```

```
 1  /*
 2  %W% %G%
 3
 4  Generic read.
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xerrno.h>
 9
10  xread( fd, buf, len )
11
12  int fd;
13  char *buf;
14  int len;
15  {
16  int count;
17  register XFILE *file;
18
19  if( fd >= 0 && fd < _XNFILE )
20          {
21          file = & _xiob[fd];
22          if( !( file-> _flag & ( _XIOREAD | _XIORW )) ||
23                  !( file-> _flag & _XUsed )
24          )
25                  return ( XEBADF );
26          /*
27          file descriptor is OK.
28          */
29          count = (*(file-> _read))( file-> _sys_id, buf, len );
30          return( count );
31          }
32  return( XEBADF );
33  }
```



```
 1  /*
 2  @(#)xsetbuf.c   1.4 5/29/85
 3
 4  System independent routine for setting buffer associated with
 5  pointers to file objects
 6  */
 7  #include "xgenlib.h"
 8
 9  xsetbuf( file, flag, len )
10
11  XFILE *file;
12  char *flag;
13  int len;
14  {
15
16  if( !(file->_flag & _XUsed ))
17          return( XEBADF );
18  /*
19  flush old buffer, if appropriate
20  */
21  xfflush( file );
22  /*
23  release old buffer, if appropriate
24  */
25  if( file->_base && !(file->_flag & _XIOMYBUF) )
26          {
27          xfree( file->_base );
28          }
29  if( !flag )
30          {
31          /*
32          User specified no buffering
33          */
34          file->_flag &= ~(_XIOMYBUF );
35          file->_flag |= _XIONBF;
36          file->_base = (char *)0;
37          file->_cnt = 0;
38          }
39  else
40          {
41          /*
42          User supplied buffer.
43          */
44          file->_bufsiz = len;
45          file->_flag |= _XIOMYBUF;
46          file->_base = flag;
47          /*
48          Assert: !( (file->_flag & _XIOREAD) && (file->_flag & _XIOWRT))
49          */
50          file->_cnt = (file->_flag & (_XIOLBF | _XIOREAD))? 0 : len ;
51          file->_ptr = file->_base;
52          }
53  return( 0 );
54  }
```

```
 1  /*
 2   * @(#)xsprintf.c       1.3 5/31/85
 3   *
 4   * Xsprintf(3X).
 5   *      format data to a memory string
 6   */
 7  #include "xgenlib.h"
 8
 9  /*
10   * this function behaves much like putc with only difference
11   * is that it writes to an user buffer instead of a file.
12   */
13  static _store(x, to)
14  char x;
15  char **to;
16  {
17    **to = x;
18    (*to)++;              /* point to the next location of the buffer */
19    **to=0;               /* terminate for safety */
20  }
21
22  #ifndef zilog
23
24  char *
25  xsprintf(string,control,args)
26  char *string;
27  char *control;
28  unsigned args;
29  {
30    return     (char *)_fmtout(_store,&string,control,&args);
31  }
32
33  #else
34
35  char *
36  xsprintf(string, control, a1, a2, a3, a4, args)
37  char *string;
38  char *control;
39  int a1, a2, a3, a4;
40  unsigned args;
41  {
42          int s1=a1, s2=a2, s3=a3, s4=a4;
43
44          return (char *)_fmtout(_store, &string, control, &args, &s1, 2);
45  }
46
47  #endif
```

```
 1  /*
 2  %W% %G%
 3
 4  Generic write.
 5  */
 6
 7  #include <xstdio.h>
 8  #include <xerrno.h>
 9
10  xwrite( fd, buf, len )
11
12  int fd;
13  char *buf;
14  int len;
15  {
16  int count;
17  register XFILE *file;
18
19  if( fd > 0 && fd < _XNFILE )
20          {
21          file = & _xiob[fd];
22          if( !( file-> _flag & ( _XIOWRT | _XIORW )) ||
23                  !( file-> _flag & _XUsed )
24          )
25                  return ( XEBADF );
26          /*
27          file descriptor is OK.
28          */
29          count = (*(file-> _write))( file-> _sys_id, buf, len );
30          return( count );
31          }
32  return( XEBADF );
33  }
```