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 compatibility 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 service 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 services 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 repetition 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-specific, priority oriented ordered list of packets called I/O Queue. Each I/O queue list head 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 successfully.

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 device-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 other 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 Auxiliary 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 overalayable and also let it compete for other system resources equitably with other tasks in the system. It
allows ACP to get services from 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 passes 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 queuing 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

```c
struct channel {
    Uchar ch_type;           /* channel control block */
    Uchar ch_flag;           /* type of the Channel */
    Ushort ch_tcb;           /* protection flags */
    Ushort rwndn_cnt;        /* owner task's TCB address */
    union {
        Ushort ch_soid;    /* I/O rundown count on this channel */
        struct {
            Ushort base;    /* socket id returned by EXOS */
            Ushort off;     /* EXOS memory pointer */
        }
    }
} 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. When 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, interrupt 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, will 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/maintenance 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 DRIVER/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 decisions 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 operations 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 transferred 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 (rmmsg 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 course.

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 for 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 immediately 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 read 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 scheduled 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 corrupt 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 but 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 callable macro routine "dealc_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 similar 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 ACS1SA 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 structureSock 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 straightforward 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 vector 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 physical addresses (which are calculated by the routine "reloc()"). The field in the configuration 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 seconds 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 configuration 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 buffering. 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 CDB 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 specifying 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 ef n 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 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 putting 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 particular 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 both the 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/MAINTANANCE 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 transferring 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 transferred 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 $RELOC. This routine is called with the virtual address as the input in R0 and it returns the relocated address in two registers. R1 contains the relocation bias and R2 contains displacement bias in the block plus 140000 (PAR6 bias). Actually the relocation bias is the higher 16-bits of the physical address and the lower 6 bits of the displacement bias are the lower 6-bits of the physical address because the relocation bias is to be loaded into the PAR6 and the displacement bias contains the virtual address to be actually addressed. The displacement bias's higher 3 bits are 6 which select the APR 6 and hence the required physical memory will be addressed. But we donot need to address the physical memory but to calculate it and this is simply done by manipulating (by shifting and masking) these two registers to get the higher 6-bits of the physical address in one word and the lower 16-bits in another. The routine "$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 reassigned 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 restoring 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_in()" 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.URM::"), 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 address. 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 configuation 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 SREXXS 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:

```c
#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 (unibuf) 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 
(unibuf). These variables are long words.

### 2.7.2 DATA TRANSFER 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 relibuf 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 transferred 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 transferred. 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 INEFFECTIVE 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 simplified 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
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 educated guess is that it should 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 BUFFERING:

As described in the limitations this feature is not implemented but it can be done by using the routines $STSTBF, $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.
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, passwd.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. passwd.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. Of course, 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_tsrcnt 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_sioiid. 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 possible. 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) TEOPT_BINARY - a QIOW #SF.SMC is sent to the ZTDRV to set this option with the bit TC.BIN set.
ii) TEOPT_ECHO - same as TEOPT_BINARY but here the bit is TS.NEC that is cleared to set the echo option.
iii) TEOPT_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_svr) (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_svr) 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 TTDVR 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 TTDVR code plus the changes in it to suit the new pseudo controller is named ZTDRV - the new pseudo terminal driver for telnet.

4.1 DECISION FOR KEEPING ZTDRV AS A SEPARATE TERMINAL DRIVER:

This decision was taken for the following reasons:

1. It would be a lot easier to debug a separate driver rather than the TTDVR 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 decision 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 dependent 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 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 simultaneously. 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 controller 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 TTDVR 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 'Y...T' as our controller name is DT'-11. An assembly time label called D$DT11 has to be defined to inform the ZTDRV software of the existence 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 route.

4.3.3 THE RELEVANT DATABASES:

Besides the data structures 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 controller 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 control 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 maintenance 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 of 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 capable 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 describing the DT-11 controller 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 interrupts.
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 controller - 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 IOPIN.

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 logs 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 pseudo function code (the one not described in the DBC) 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 at all 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 list 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 buffering, 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 transferred 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 #10.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 transferred 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 some .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 : 
   
2. #453 : 
   "TTPHI == T.V2XX"
3. #456 : 
   
4. #458 : 
   "TTPHI == T.BMPI" (added).
5. #460 : 
   
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"
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=.I.F D$ M$PRO
LCKDF$ SPIN
.IFTF
.IIF NDF $ZTUQ $ZTUQ==.
.WORD 0,,,-2
;
.IF T
.WORD X1
X1=X1*2
.ENDC
.IIF NDF $ZTFB $ZTFB==.
.WORD 0,0,0,0,0
ZT$UQL==.-ZT$UQL
.ENDR
;
;
;
;
;
.
.IF DF M$PRO
X1=$ZTUQ+2
$ZTUQT::
.REPT M$PRO
```

INDEX TABLE TO ZT DRIVER
UCB QUEUE HEADS AND FORK BLOCKS
6. Topic: Table of pointers to UCB tables.

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

Changes to existing code: Addition of one more entry in the Table of Pointers to the UCB tables for DT-ll 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-llM systems and not for RSX-llM-PLUS systems as they donot require these tables.

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

   DTUCB::
   N=0
   .REPT D$S$T11
   .WORD DTUCO+N
   N=N+4
   .ENDR
   DTUCO::
   .BLKW D$S$T11*2 ; DT UCB/CSR TABLE
   .ENDC ; DF D$S$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 #$DEVH,D,R0
   35$:
   MOV @R0,R0
   BNE 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 ZTDBL.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 mnemonic 'ZT'.

Line numbers: 61 and 141.

Changes in the existing code: label names changed.
1. #61: instead of "$ZTDBL::" it is now "$ZTDBL::"
2. #141 instead of "$ZTDBE::" it is now "$ZTDBE::"

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  $ZTOUP
   .WORD  0
   ZTCTBP::WORD  0

6.4 ZTDB2.MAC: .IDENT /02/

This file contains the data base for the ZTDRV and is coded in such a
way that it will automatically assemble 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 - non existant.
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 --> multiuser protection/CLI address used by the main driver code.
U.LUIC --> login uic - initialized to zero - used by the main code.
U.ONW --> owning terminal's UCB address if device alocated.
U.DCB --> initialized to zero here.
U.RED --> back pointer to the DCB.
U.CTL --> 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 power failure (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.RECDV.CCLIDV.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.TCPV --> 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 --> interrupt 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 - YINDEX.
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 improvement this field should be initialized to the label ZECOR 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.OIN  -->  Owning UCB address. Initialized as the corresponding UCB address.
K.CQ  -->  Controller request queue list head 0 and address of the SCB which is -2
K.FRK/9.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$ST11 = 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 mainainance 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 TTDVR. Since the code for ZTDRV is extracted from the TTDVR 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 TTDVR since it is a link list of
the UCB addresses and this link list was being shared by both the TTDVR 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 TTDVR 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 quitting. 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 ZTDRV, a QIOW #IO.HNC 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 QPINF 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
.NLST 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/

.ENABLE LC

.MCALL HWDDFS$, UCBDFS$, DCBDFS$, SCBDFS$, TCBDFS$, PKTDFS$
HWDDFS$
UCBDFS$
DCBDFS$
SCBDFS$
TCBDFS$
PKTDFS$

.PSEC'T 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

.IPF ;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
$SETBL:

; WORD ZEINI ; initiator entry point
; WORD ZECAN ; cancel entry point
; WORD ZEOUT ; time-out entry point
; WORD ZEPWF ; power fail entry point

; 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 administrative 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.GPC = 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 parameters in the I/O
; packet. Last three parameters (I.PRM+6 I.PRM+12) are shifted 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 builds 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:

; following 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$: ;
; shift parameter 4, 5 & 6 by two words in I/O packet
;
MOV R1,R3
ADD #I.PRM+12,R3
MOV #3,R4
60$: MOV (R3),4(R3)
TST -(R3)
SOB R4,60$

; 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
BEQ 70$ ; is it socket accept request?
CMP I.FCN(R1),#IO.ACS!SA.CON
BEQ 70$ ; is it socket connect request?
CMP I.FCN(R1),#IO.ACS!SA.OPN
BEQ 70$ ; is it socket open request?
CMP I.FCN(R1),#IO.ACS!SA.SAD
BEQ 70$ ; is it obtain socket address request?
CMPB I.FCN+1(R1),#IO.SOC/400
BEQ 70$ ; is it socket control request?
CMP I.FCN(R1),#IO.XFR!IX.RCV
BEQ 70$ ; is it a receive message request?
CMP I.FCN(R1),#IO.XFR!IX.SND
BEQ 70$ ; is it a send message request?
BNE 100$ ; if NE no, so process other requests
70$:
TST I.PRM+4(R1)
BNE 80$ ; is Soioctl structure address there?
MOV #IE.SPC&377,R0
MOV R1,R3
JMP 500$

; address check and relocate parameter #3, if any, which contains the
; address to the socket related parameters buffer
;
80$:
MOV (R3),R0
MOV R1,R3

.IS DF A$CHK!M$MGE

MOV #SOICTL,R1
CALL $ACHKB
BCC 90$
MOV #IE.SPC&377,R0
JMP 500$

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

; address check and relocate user buffer if necessary

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.EXCEX.CNF ; is it read config msg fn
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.STS ; is it read EXOS stat. fn
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.RST ; is it read & reset EXOS stat fn
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.SAR ; is it set ARP function
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.GAR ; is it get ARP function
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.DAR ; is it delete ARP function
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.ART ; is it add an RT entry fn
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.DRT ; is it delete an RT entry fn
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.SRT ; is it fetch an RT entry fn
BEQ 120$ ; if EQ yes
CMP I.FCN(R1),#IO.EXCEX.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$SCHK1M$S$MGE

MOV I.PRM+2(R1),R1 ; get length of buffer
CALL $ACHKB ; address check buffer byte align
BCC 140$ ; 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  #IRE.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 parameters
;
; 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:
       MOV  R5,UCBCAN  ; CANCEL IO ENTRY POINT
       MOV  R1,TCBCAN  ; save UCB address
       MOV  #I.LGTH,R1  ; save TCB address of current task
       CALL  $ALOCCB
       MOV  #IO.KIL,I.FCN(R0)  ; Allocate an I/O packet
       MOV  TCBCAN,R5  ; move function code
       MOV  R5,I.TCB(R0)  ; get TCB address of current task
       MOV  UCBCAN,R5  ; set TCB address
       MOV  R5,I.UBC(R0)  ; 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 RO with ACP address
JMP    $EXRQP ; Q pkt & wakeup ACP

; $ZEOA/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
; calculated and stored in the UCB. This setup is only required for
; software running on the UNIBUS machines.
;
.IF DF  R$$MPL

$ZEPWF:
$ZEUNL::
    RETURN

$ZEOA::

.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

$ZEKR8: ; 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 handle device interrupts. As the EXOS device processes all the requests in a pure asynchronous 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 unstoping it if it is sleeping.

sema:: .word 177776 ; initial value of semaphore

$ZEINT::
INTSV$ ZE,PR4,Z$E11
sec
ror sema
bcc 10$
return

10$:
MOV UCBR5,R5 ; get address of UCB before 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
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::
 .BLKW 1 ; start of local buffer pool

.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$

.PSEC $$$

.GLOBL $ZEVEC
.ENABLE LC

$ZEDAT::

; start of the ZEDRV device table

.IF DF R$MPL

;---------
; ZE CTB 
;---------

.WORD 0

; L.ICB

$CTB0:

.WORD $CTBl
.ASCII /ZE/
.WORD .ZCO
.BYTE 1
.BYTE 0

$ECTB::

.WORD $ZEA

; L.KRB

.IFTF R$MPL

$ZETBL = 0
$ZEDCB::

.ZCO:

.WORD .ZCl
.WORD .ZEO
.ASCII /ZE/
.BYTE 0,0

; D.LNK link to next DCB
; D.UCB pointer to first UCB
; D.NAM device name
; D.UNIT lowest and highest unit number
.WORD ZEND-ZEST ; D.UCBL length 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 0 ; D.PCB PCB address of driver partition

ZEST = .

.WORD 0 ; U.OWN owning terminal's UCB address
.ZEO::
.WORD .ZCO ; 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$$MPL
.BYTE US.RED!US.PUB!US.OFL ; U.STS 2nd status flag - unit cannot be
.IFF ;R$$MPL
.BYTE US.RED!US.PUB
.ENDC ;R$$MPL
.WORD DV.EXT ; redirected
 ; U.CWL characteristic word 1 --> device
 ; is connected to 22-bit direct
 ; addressing controller
.WORD 0 ; U.CW2 char word 2
.WORD 0              ; U.CW3 char word 3
.WORD 0              ; U.CW4 char word 4, no buffer required
.WORD $ZEO         ; U.SCB pointer to SCB
.WORD 0              ; U.ATT attached task UCB
.BLKW 3              ; U.BUF, U.BUF+2 & U.CNT
.IFT  ;R$$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 starting 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 $ZEVGC/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::
.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 ;
;---------------------'

$ZEO:
.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.CON1S2.LOC ; 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-existent)

;---------------------;
; 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.UFVH - 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

.ZCl = 0 ; end of DCB list for ZE:
$CTBl = 0 ; end of CTB list for ZE:

.END ; end of file ZETAB.MAC
          .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 $DEL .GOTO 10
PIP ZEDRV.MAC;*,ZETAB.MAC;*/DE
.10:
;
; Now build the ZE (EXOS) driver.
;
;
; Create the task builder input file. Ask for the interrupt vector
; location use default if the installer does not want to change it.
;
;
; Create the input command file for the linker
;
.OPEN ZETKBD.CMD
.DATA LB:[1,54]ZEDRV/-HD/-MM,,ZEDRV=
.DATA ZEDRV,ZETAB
.DATA LB:[1,54]RSX11M.STB/SS
.DATA LB:[1,1]EXELIB/LB
.DATA /
.DATA STACK=0
.DATA PAR=DRVPAR:120000:14000
.DATA GBLDEF=$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>'ZETKBD
;
; delete indirect command file
;
PIP '<UIC>'ZETKBD.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/WOR:FO
.ENABLE QUIET
.ENABLE SUBSTITUTION
.DISABLE DISPLAY
.IFND $VRBS .ASK $VRBS Verbose ? [Y/N]
.IFT $VRBS .DISABLE QUIET
.IFND $DEL .ASK $DEL Delete source file from current UPD? [Y/N]
.IFND $NORE .ASK $NORE 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 interrupt 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 $NORE PIP LB:[1,54]ZEDRV.TSK; */DE
; .IFT $NORE PIP LB:[1,54]ZEDRV.STB; */DE
TKB '@<UIC>'ZETKB
;
; delete indirect command file
;
.PIP '<UIC>'ZETKB.CMD; */DE
.PIP '<UIC>'ZEDRV.OBJ; */DE
.PIP '<UIC>'ZETAB.OBJ; */DE
;
; set protection for the driver
;
*ENABLE DISPLAY
zedrv/-hd/-mm,zedrv/-sp,zedrv=
zedrv,zetab,lb:[1,54]rsx1lm.stb/ss
lb:[1,1]exelib/1b
/
stack=0
par=drvpar:120000:14000
gbldef=$zevec:400
//
$ ! skeleton for bld.com
$ !
$ if "''pl''" .nes. "?" then goto doit
$ typ sys$input

command file to build the task image

required command files: None

required logical names: None

required parameters:
pl - 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 "''pl''" .eqs. "" then $ pl = "''f$logical("sys$disk")''f$directory()"
$ set def 'pl'
$ 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/1b"
$ 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:[unilm.] 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/1b"
$ 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]rxmlm.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 ""plT"" .eqs. "" then $ pl = ""f$logical("sys$disk")"f$directory()"
$ set def 'pl'
$ show def
$ show logical 1b
$ 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:[unil1m.] lb:
$ show logical 1b
$ 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:[unil1mp.] lb:
$ show logical 1b
$ 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 ""pl"" .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 device */
#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 {
    Uchar ch_type;    /* channel control block */
    Uchar ch_flag;    /* type of channel free, socket & etc */
    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_msg 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: EXIOCMND.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 SOSTADDR (55)
#define SOCLOSE (56)
#define SOVERIFY (57)
#define SOIOCTL (58)
#define SOSELECT (59)

#define NET_DLOAD 0 /* net download */
#define NET_UPLOAD 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 BRDCARP /* 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 therefore 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 associated constants definition for the EXOS-203 
* ethernet controller, compatible 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_sal;
    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_ioctlcmd; /* 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_pdata[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 */
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_tsdatal[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_telnets;
    } 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 {
    Ushort ma_rlink;
    struct msg ma_rmsg[NET_RBUFS];
    struct msg *ma_lastr;
};

struct wmsg_area {
    Ushort mamlink;
    struct msg ma_wmsg[NET_WBUFS];
    struct msg *ma_lastw;
};
```
/*
 * These are the DIC and DPB lengths of the Executive directives
 */
#define QIO 06001
#define QIOW 06003
#define ALUN 02007
#define WTSE 01051
#define CTIM 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 buffer */
#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_CNFIN 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_KAR 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 BRDELRRT /* 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 BRDAADDR /* 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_UGC 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 /* shutdown r/w on socket */
#define SO_SKP SIOCKEEP /* set keep alive */
#define SO_GKP SIOCKEEP /* inspect keep alive */
#define SO_SLG SIOCSLINGER /* set linger time */
#define SO_GLC SIOCSLINGER /* get linger time */
#define SO_OSB SIOCSENDOBB /* send out of band */
#define SO_RSB SIOCSRCVOBB /* receive out of bound */
#define SO_AMK SIOCATMARK /* at oob mark ? */
#define SO_SPG SIOCSPCRP /* set process group */
#define SO_GPG SIOCSPCRP /* get process group */
#define SO_NRD SIONREAD /* SIONREAD */
#define SO_NBO SIONBIO /* SIONBIO */
#define SOASY FIOASYNC /* FIOASYNC */

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

/*
 * All the Socket related parameters in the QIO call are passed
 * through the structure "SOict1" defined below.
 */

struct SOict1 {
    short   hassa;       /* non-zero if sa specified */
    struct sockaddr *sa; /* socket address (optional) */
    short   hassp;       /* non-zero if sp specified */
    struct sockaddr *sp; /* socket protocol (optional) */
    int     type;        /* socket type */
    int     options;     /* options */
    /* these are for select () */
    int     nfds;
    long    *wp;
}
long    *rp;
long    timo;
};
/*  * finename:   EXREG.H  */

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

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

struct excrl {
    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 040000 /* 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: WARNING - 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 IPPROTO_ICMP 1 /* control message protocol */
#define IPPROTO_GGP 2 /* gateway^2 (deprecated) */
#define IPPROTO_TCP 6 /* tcp */
#define IPPROTO_PUP 12 /* pup */
#define IPPROTO_UDP 17 /* user datagram protocol */
#define IPPROTO_RAW 255 /* raw IP packet */
#define IPPROTO_MAX 256

/*
 * Port/socket numbers: network standard functions
 */
#define IPPROTO_ECHO 7
#define IPPROTO_DISCARD 9
#define IPPROTO_SYSTAT 11
#define IPPROTO_DAYTIME 13
#define IPPROTO_NETSTAT 15
#define IPPROTO_TFTP 21
#define IPPROTO_TELNET 23
#define IPPROTO_SMTP 25
#define IPPROTO_TIMESERVER 37
#define IPPROTO_NAMESERVER 42
#define IPPROTO_WHOIS 43
#define IPPROTO_MTP 57

/*
 * Port/socket numbers: host specific functions
 */
#define IPPROTO_TFTP 69
#define IPPROTO_RJE 77
#define IPPROTO_FINGER 79
#define IPPROTO_TTYLINK 87
#define IPPROTO_SUPDUP 95

/*
 * UNIX TCP sockets
 */
#define IPPROTO_EXECSERVER 512
#define IPPROTO_LOGINSERVER 513
#define IPPROTO_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
 * 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 ntohs().
 */

/* 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 0x00f00000L  /* 8 bits of net */
#define INCA_LNA 0xff000000L
#define INCB 0x00400000L
#define INCB_NET 0x0ff00000L  /* 16 bits of net */
#define INCB_LNA 0x0000ff00L
#define INCC_NET 0x00ff0000L  /* 24 bits of net */
#define INCC_LNA 0x0000ff00L
#endif

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

#ifdef VAX
    #define IN_CLASSA 0x00000080
#define INCA_NET 0x000000ff  /* 8 bits of net */
#define INCA_LNA 0xfffffff00
#define INCB 0x00000040
#define INCB_NET 0x0000ffff  /* 16 bits of net */
#define INCB_LNA 0xfffff000
#define INCC_NET 0x0000ffff  /* 24 bits of net */
#define INCC_LNA 0x00000000
#endif

#ifdef PDP11
    #define IN_CLASSA 0x00800000L
#define INCA_NET 0x00f00000L  /* 8 bits of net */
#define INCA_LNA 0xff000000L
#define INCB 0x00400000L
#define INCB_NET 0x0ff00000L  /* 16 bits of net */
#define INCB_LNA 0x0000ff00L
#define INCC_NET 0x00ff0000L  /* 24 bits of net */
#define INCC_LNA 0x0000ff00L
#endif

#ifdef I8086
    #define IN_CLASSA 0x00000080
#define INCA_NET 0x000000ff  /* 8 bits of net */
#define INCA_LNA 0xfffffff00
#define INCB 0x00000040
#define INCB_NET 0x0000ffff  /* 16 bits of net */
#define INCB_LNA 0xfffff000
#define INCC_NET 0x0000ffff  /* 24 bits of net */
#define INCC_LNA 0x00000000
#endif

#ifdef M68000
    #define IN_CLASSA 0x80000000L
#define INCA_NET 0xff000000L  /* 8 bits of net */
#define INCA_LNA 0x000000ffL
#define INCB 0x40000000L
#define INCB_NET 0x0ff00000L  /* 16 bits of net */
#define INCB_LNA 0x0000ff00L
#define INCC_NET 0x00ff0000L  /* 24 bits of net */
#define INCC_LNA 0x00000000L
#endif

#ifdef Z8000
    #define IN_CLASSA 0x80000000L
#define INCA_NET 0xff000000L  /* 8 bits of net */
#define INCA_LNA 0x000000ffL
#define INCB 0x40000000L
#define INCB_NET 0x0ff00000L  /* 16 bits of net */
#define INCB_LNA 0x0000ff00L
#define INCC_NET 0x00ff0000L  /* 24 bits of net */
#define INCC_LNA 0x00000000L
#endif
#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 0x00000000L
#endif

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

#define IN_NETOF(in) \
  ((in).s_addr&INCLASSA) == 0 ? (in).s_addr&INCA_NET : \
    ((in).s_addr&INCB) == 0 ? (in).s_addr&INCB_NET : \
    (in).s_addr&INCC_NET)
#define IN_LNTOF(in) \
  ((in).s_addr&INCLASSA) == 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_lnaf();
#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 mode (0) */
    char im_hdf0[2]; /* host data format option */
    char im_junk[3]; /* host address mode */
    char im_addrmode;
    char im_dummy2;
    char im_mnsize;
    char im_byteln[4]; /* data order byte pattern */
    Ushort im_wordln[2]; /* data order word pattern */
    long im_longplnt;
    char im_mmap[20]; /* (rest of) memory map (returned) */
    short im_10loff;
    short im_101seg;
    char im_nproc;
    char im_nmb;
    char im_nslots;
    char im_nhosts;
    /* movable block offset */
    /* movable block segment */
    /* number of exos 101 processes */
    /* number of exos 101 mailboxes */
    /* number of address slots */
    /* 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
struct rel_addr {
    Ushort rel_bias;
    Ushort dis_bias;
};

struct iopkt {
    struct iopkt *i_lnk;
    Uchar i_pri;
    Uchar i_efn;
    Ushort i_tcb;
    Ushort i_ln2;
    Ushort i_tcb;
    Ushort i_fcn;
    struct {
        Ushort v_iosb;
        struct rel_addr r_iosb;
    } i_iosb;
    Ushort i_ast;
    struct {
        struct rel_addr i_buf;
        Ushort i_cnt;
        struct rel_addr i_soictrl;
        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 RLGRON 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 TSDOOP 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_fn; /* 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.
*/

#ifndef 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 swap ex_swap
#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_SOMAXCONN 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 AFETHER 11 /* Ethernet Address */
#define AF_COUNT 12 /* A count */
#define AFETYPEFILTER 13 /* Ethernet filter */

#define AF_MAX 14

/*
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 s1_family;
    u_short s1_types[6];
    short s1_zero;
    #ifdef ONBOARD
    struct enreq *s1_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) /* on pty: set/clear packet mode */
#define FIONBIO      (126)  /* data packet */
#define FIOASYNC     (125)  /* flush packet */
#define TIOCPKT      (112)  /* stop output */
#define TIOCPKT_DATA 0x00   /* start output */
#define TIOCPKT_FLUSHREAD 0x01 /* no more ^S, ^Q */
#define TIOCPKT_FLUSHWRITE 0x02
#define TIOCPKT_STOP 0x04   /* now do ^S, ^Q */
#define TIOCPKT_NOSTOP 0x10
#define TIOCPKT_DOSTOP 0x20
#define SIOCDTUNE   (0)     /* shutdown read/write on socket */
#define SIOCSKEEP   (1)     /* set keep alive */
#define SIOCGKEEP   (2)     /* inspect keep alive */
#define SIOCSSLINGER (3)    /* set linger time */
#define SIOCGLINGER (4)     /* get linger time */
#define SIOCSENDIOB (5)     /* send out of band */
#define SIOCRCVOOB  (6)     /* get out of band */
#define SIOCATMARK  (7)     /* at out of band mark? */
#define SIOCSPCRP   (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 MAXSO IOCTL 5 /* max no of SOioctl structure */

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

#define NOSOBUF -10
#define NOSO IOSB -11
#define NOSO IOCTL -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 ZEO: UCB */

long  phy_buf = 0;  /* physical address of pool */
long  unibuf = 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_init();   /* 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_INIT:
                        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:
                        break;
                }
        }
    }
}
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);
answer()
{
    register int i;
    register struct msg *current;
    register struct iopkt *pending;

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

    current = rmser_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 SOSESELECT:
            case SOSEWAKEUP:
                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 solicited */
        reply();
        if (pending){ /* if it was solicited */
            i = pending->iprm.i_prm6; /* get channel # */
            if (inform) { /* only if boards processing is */
                ackuser(pending); /* over then acknowledge 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 */
            }
        }
    }

    #ifdef DEBUG
    qio_write("unsolicited reply",20,040);
    #endif
    #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 issuing 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*/
            { /* other code */
            }
        }
    }
}
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))
            dealloc_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))
            dealloc_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 delay of
 * (tmag * 20 msec) is obtained.
 * If tunit='S', then a time delay 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);
int drive()
{
    FOREVER { /* fall into an eternal loop */

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

    }

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

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

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

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

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

        default: /* no channel */
            break;

        }
    }
}
case IO_EXC|EX_POS: /* position the memory relocator */
iobs.cc = 1; iobs.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 iobs.cc = IE_PRI;
ack = 1;
break;

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

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

case IO_ACS|SA_USL:
iobs.cc = 1; iobs.lc = 0;
if ( inrange(chn) && sametask(chn) )
    fin_pen(SA_USL);
else
    iobs.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_sid;
/* 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 SOict1 SOict1 = {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 *mrkclks = {0};
struct msg *free_slot = {0};
struct msg *mp = {0};
Uchar *nxtrst = {0};
Uchar *nxtwst = {0};
struct SOict1 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) /* find 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
 * until 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 beginning */
        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);
}
/* 
* 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;

#define DEBUG
qio write("remque", 7,040);
#else
prev = 0;
current = *head_ptr;
while ( current )
{
    next = current -> i_link;
    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 dealllocate it */
        else {
            iosb.cc = IE_ABO;
            /* return abort status to user */
            current->i_ast= 0;
            /* make sure ast routine is not entered */
            ackuser( current );
    }
    /* dequeue 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;

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


```c
{
    if (ch_des[i].ch_type == CH_EXOS) /* ch is Admin type */
#endif 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 */
#endif 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.
    */
#endif DEBUG
    qio_write("iokill",7,040);
#endif
    if (ch_no = srchn( io_pkt->i_tcb )){
#ifif DEBUG
    qio_write("close ch",9,040);
#endif
    PKT.i_prm6 = ch_no;
    ex_mg.mh_length = sizeof ( struct messages ) - factor;
    ex_mg.mh_request = SOCLOSE;
    return (i); /* send request to board */
}
else /* no more channel remains opened for this task */
{
#endif DEBUG
```
qio_write("kill all pending I/O",20,040);
#endif

remque(&int_que); /* remove all pending requests */
/* donot remove outstanding 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_ABORT ); /* 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 fills 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 requesteer*/
            ch_des[i].runrn_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:
            break;
        case SOSELEWAKEUP:
            io_pkt = (struct iopkt *)ex_sel.nm_proc;
            break;
        default:
            io_pkt = (struct iopkt *)ex_mg.nm_userid;
            break;
    }

    chn = PKT.i_prm6;
    iosb.sc = ex_mg.nm_reply; /* board reply status */
    iosb.cc = 1; /* QIO success */
    iosb.nread = 0;
    inform = 1; /* acknowledge the user immedielt */

    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_d1.nm_xmbyte, (char *)&param, sizeof (char));
            ucopy((char *)&param, &PKT.i_buf.rel_bias, sizeof ( char ));

        case NET_DLOAD:
            iosb.nread = ex_d1.nm_length; /* no of bytes read */
            ch_des[chn].ch_u.ch_addr.of += iosb.nread;

#ifndef 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.lc == 0 ){
        ch_des[chn].ch_u.ch_soild = ex_cmd.nm_soid;
        iosb.nread = chn; /* return channel # */
    }
    break;

case SOACCEPT:
case SOCONNECT:
case SOKTADDR:
    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 SOSELUakeup:
    /* 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_in = iopend;  /* 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_in = int_que; /* put it in internal Q again */
        int_que = io_pkt;
    } else
        clist();            /* put the close packet in the close list */
    inform = 0;                        /* donot inform user right now */
    ch_des[chn].rundown_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 */
    close(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_soctl.rel_bias, sizeof(char));
        break;

    case SIOCGKEEP:
    case SIOCGLINGER:
    case SIOCATMARK:
    case SIOCMPGRP:
        param.hassa = *(short*) ex_ctl.nm_iocdata;
        ucopy((char*) &param, &PKT.i_soctl.rel_bias,
sizeof ( short ));
break;

case FIONREAD:
  bcopy(ex_ctl.nm_iocdata,&param.hassa,sizeof(long));
  ucopy((char *) &param, &io_pkt.iprm.i_socket.rel_bias,
        sizeof ( long ));
  break;
default:
  break;
}
break;

case SOHASOOB:
  fin_pen(SA_URG);
  /* give a signal oob 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;((wpf != '\n') && (*pf != '\0'));cnt++,pf++);
  qio_write(&mp->nm_u.nm_printf.nm_prdata,cnt,0);
  if(wpf == '\n')
    qio_write("\r\n",2,0);
  break;

default:
  break;
}

ex_hd.mb_length = sizeof( union exos_u ) - sizeof ( struct headers);
ex_hd.mb_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.nlm_request = NET_DLOAD;
            }
            ex_dl.nlm_source = getpool(io_pkt,1);
#ifdef UNIBUS
            ex_dl.nlm_source = getpool(io_pkt,1);
#endif
        }
        else{
#ifdef UNIBUS
            ex_dl.nlm_source = getpool(io_pkt,0);
#endif
            ex_dl.nlm_request = NET_ULOAD;
        }
        ex_dl.nlm_length = PKT.i_cnt;
    }
```c
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_sal = 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 */

        else if ( (subcmd != SOCLOSE) && (subcmd != SOSELECT) ) /* no soioctl 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 SOKTADDR:
            ex_hd.mh_length = sizeof ( struct Sock_cmd ) - factor;
            if ( ex_cmd.nm_isaddr = param.hasa )
                bcopy( &param.sa, &ex_cmd.nm_saddr, sizeof (struct sockaddr));
                if ( ex_cmd.nm_isproto = param.hasp )
                    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.iprm4 + 1; /* read = 1 and write = 2 */
    ex_sel.nm_proc = ((Ushort)io_pkt >> 1) & 0x7FFF;
    /* pass the pkt address with msb 0 */
    PKT.iprm5 |= NOREPLY; /* indicate no reply initially */
    break;

default:

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

/*
 * 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;
#endif
    ex_pkt.nm_bufaddr = absadr(&PKT.i_buf);
#endif

    if ( (subcmd == SOSEND) || (subcmd == SORECEIVE) )
    {
        scopy( &PKT.i_soiictl.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;
    }
#endif
    ex_pkt.nm_bufaddr = getpool(io_pkt, 1);
#endif
} else {
    ex_pkt.nm_request = SORECEIVE;
#endif
    ex_pkt.nm_bufaddr = getpool(io_pkt, 0);
#else
    return(1);
#endif
    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_soioctl.rel_bias, sizeof ( struct sockaddr ));

    switch( subcmd ){

        case SIOCGESKEEP:
        case SIOCGLINGER:
        case SIOCRCVOOB:
        case SIOCATMARK:
        case SIOCGPCRP:
        case FIONREAD:
            break;

        case SIOCSEENDOOB:
            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_in;
    io_pkt->i_in = 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 SOict1));
    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-initiaialize ch # as IO_CAN does*/
    }    /* not have any in iR */
    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 acknowledge user */
if(ex_send > 0){
ex_msg.nm_userid = (long) io_pkt;
ex_msg.nm_reply = 0;
ex_hd.mh_status |= MQ_EXOS;
if(io_pkt){ /* if io_pkt == 0 do not append */
append();
chn_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);
wmmsg_area.ma_lastw = mp; /* release unused slot */
nxtwst = &wmmsg_area.ma_lastw->nm_u.msg_hd.mh_status;
}
else /* if not action */
{
wmmsg_area.ma_lastw = mp;
nxtwst = &wmmsg_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 == PTYN0)
    return(0);

void
	/*
	* 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 = TWRITE;
                    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);
}
deloc_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_targst = p->request;
    ex_tel.nm_tsdlen = p->byte_cnt;

    bcopy(p->w_data,ex_tel.nm_tsddata,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');
#endif 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);
        }
#endif DEBUG
    qio_write("out caron",9,040);


#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 {
#endif DEBUG
{
    int i;
    i = 0;
    i = '0' + p->nm_soid;

qio_write(&i, 2, 040);
}
#endif
qio_zt(p->nm_soid, p->nm_tsdatal, p->nm_tsdalen);

#ifdef DEBUG
qio_write("out zt_read", 11, 040);
#endif
}

/*
 * WRITE_REPLY
 */
wr_reply(p) /* TWRITE (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
    /*
     * do something
     */
    st->reply_pending = 1;
    /*
     * do something
     */
    out_int(p->nm_soid);
if(!st->carrier_on)
    return(0);
else  {
    switch (p->nm_tdata[0])  {
    case AO-MAXBYTEVAL :
        ch = ctrl('O');
        break;
    case EC-MAXBYTEVAL :
        ch = BS;
        break;
    case EL-MAXBYTEVAL :
        ch = ctrl('U');
        break;
    case IP-MAXBYTEVAL :
        ch = ctrl('C');
        break;
    case AYT-MAXBYTEVAL:
        default:
            return;
        } /* end of switch */
    qio_zt(p->nm_soid,&ch,1);
    } /* end of else */

#ifdef DEBUG
    qio_write("out nvtfunc ",l2,040);
#endif

}  /* end of nvtfunc() */

/*
 * 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;

#endif DEBUG
    qio_write("in do_option ",l2,040);
#endif

if(!st->carrier_on)
    return(0);
else  {
    switch (p->nm_tdata[0]) {
    case TEOPT_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 TEOPT_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 TEOPT_SGA:  
    default: 
        return;         /* end of switch */
    qio_smc(p->nm_soid,stadd);      /* end of else */
#endif  
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
by 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;
    dealloc_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_link = (Ushort)( reloc(rmsgarea->ma_rmsg) - r_base);
/* exos link to read queue */

current = (struct msg *) (&rmsgarea->ma_rmsg[NET_RBUFS-1]);
rmsgarea->ma_lastn = rmsgarea->ma_rmsg;
nxtstrt = &rmsgarea->ma_lastn->nm_u.msg_hd.mh_status;
for (i=0;i<NET_RBUFS;i++) {
    next = *(struct msg *) &rmsgarea->ma_rmsg[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 );

wmsgarea->ma_lastn = wmsgarea->ma_wmsgs;
nxtxst = &wmsgarea->ma_lastn->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_img;

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_hfo[0]=im->im_hfo[1] = 1;           /* do auto-byte/word swapping*/
im->im_addrmode = 3;                       /* absolute address mode */

/* data order test patterns */

im->im_bytectn[0] = 1;
im->im_bytectn[1] = 3;
im->im_bytectn[2] = 7;
im->im_bytectn[3] = 0XF;
im->im_wordctn[0] = 0X103;
im->im_wordctn[1] = 0X70F;
im->im_longctn = 0X103070F;
im->im_l01off = im->im_l01seg = 0xFFFF;
im->im_nhosts = 1;
im->im_result = im->im_nmb = im->im_nproc = im->im_nslots=0XF;
im -> im_h2exqaddr =
#ifndef 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 =
#ifndef 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_addr[0..3] is init as 0xFFFF0000 */
#ifndef 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 */

#ifndef 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));
}

#ifndef DEBUG
    qio_write ("over",5,040);
#endif
delay(2,'s');
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;
    }
}

#ifndef 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){  
            if(pkt->i_prm.i_prm4;  
                c = ex_oob.nm_sid;
            }
        }
        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 & NOREPPLY){
                    pkt->i_prm.i_prm5 |= UNSELECT;  /* set it unselect */
                    prev = pkt;
                    pkt = pkt->i_link;
                    continue;
                }
            if ( prev )
                prev->i_link = pkt->i_link;
            else
                io_pend = pkt->i_link;

            if((x == SA_USL) || (x == SA_ROO) )  /* only for SA USL and SA ROO */
                /* see that ast is not entered */
            ch_no = pkt->i_prm.i_prm6;  /* get the channel number */
        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_link;
}
/* filename: UNIACP.C */
/*
This file contains the 'C' code for incorporating ACP on a UNIBUS M/C
*/
/*
UNI_INIT
This routine is called for initializing the unibus related stuff. It calls a macro routine to assign the UMR's.
*/

uni_init() {
    srex();  /* specify exit ast for cleanup of UMR's */
clear(pool_im,sizeof pool_im);
rel_pool();  /* initialize relocated address of pool */
    if(fass_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 lst 4KW ? */
                    tmp_addr.rel_bias = relbuf.rel_bias;
                } /* is it within 256KW ? */
                ...} /* is it within 8KW */
            } /* is it within 128KW */
            else { /* is it within 4KW */


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(unilbuf + (POOLBUFSIZE * i));
else
    return(unil2buf + (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->i_lnk = tmp;
        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 = rel1buf.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(&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.
;
; RO returns the completion code 0 -- unsucces 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       $SWSTK,RET      ; switch to system state and return to user
CLR         SUCC           ; state at RET after execution of RETURN
MOV         #$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
MOV         D.UNIT+1(R2),R3  ; get number of UCBs (units)
MOV         D.UCB(L(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,泽UCB                  ;; 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

.ENDC

.SUCC:  .BLKW    1

.PSECT  C$TEXT,I,RO
.EVEN
.END
This routine dequeues a packet from the listhead of the ACP task. It first switches to system state before dequeuing. 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 #.RCVL,R0 ;; get receive queue listhead
CALL $QRMVF ;; attempt to dequeue packet
BCS 20$ ;; if CS no packet
MOV R1,IOPKT ;; return address of I/O packet
BR 60$ ;; return

20$: MOV NXTRST,R2 ;; get ptrntr to status field of reply Q
BEQ 40$ ;; 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 availability 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, proceeds 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 compatible 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.UBR(R3),R5 ; move address of UCB in R5
   ADD #I.PRMR,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 $IOPIN ; complete io process

   RETURN ; return to task state
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::

.JR 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

.JR 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::

IFDEF 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)

IFDEF 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 */
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 UCBDFS,PKTDFS,DCBDFS,SCBDFS,TCBDFS
UCBDFS ,,TTDEF
PKTDFS
DCBDFS
SCBDFS
TCBDFS

IO.INP = 5400
IO.OUT = 6000

; OUT.INT --> This routine gives an O/P interrupt to ZTDRV

.psect c$text,i,ro
.MCALL ALUNS$,#QIO$S,#QIOW$S

OUT.INT::
    jsr R5,c$sav
    MOV 4(R5),R2 ; get pty_no_first parameter
    ALUNS$ #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$sav
    MOV 4(R5),R0 ; pty_no
    MOV 6(R5),R1 ; buffer ptr to be o/p
    MOV 10(R5),R2 ; length of buffer
    ALUNS$ #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 teviewer

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 5F.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,#"2T,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)                      ; pty number
       
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.DSBIUS.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$set
       
.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

MOV B 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
MOV B 6(R5),R0 ; move a byte value in R0

.ENDC

MOV B 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 soict1 buffer into a global
; buffer of acp. this routine is "c" callable as
;
; copy (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:: ; copy 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,R0 ; load R0 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,R0 ; get pointer to source relocated addr
MOV (R0)+,R1 ; move source relocation bias
MOV (R0),R2 ; move source disp bias (in terms of APR6)
ADD #120000-140000,R2 ; make it APR5 bias
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
.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 registers
MOV R5,-(SP) ; save frame pointer
MOV 4(R5),FROM ; get source addr pointer
MOV 6(R5),TO ; get dest relocated 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,R4 ; 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
.PSEC T 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$S$MPL
S.UNI = S.EMB + 2
.IFF ;R$S$MPL
S.UNI = S.FRK + 14
.ENDC ;R$S$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?
BBQ   FAILS ; if EQ then no
MOV   M.UMVL(R0),UN1BUF+2 ; save lower order unibus address
MOV   M.UMVH(R0),UN1BUF ; save higher order word
MOV   UN1BUF,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,UN1BUF ; restore the high order address
MOV   UN1BUF+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 number
MOV   R3,R4 ; save R3 in R4
MOV   UN1BUF+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,UN1BUF ; higher order address 2 bits
MOV   R3,UN1BUF+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  $ALOCCB ; 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
TST   (SP)+ ; which returns the physical address
MOV   (SP)+,R2 ; pop stack
MOVB  R0,M.BFVH(R2) ; unsave pointer to UMR ass. block
MOV   R1,M.BFVL(R2) ; higher order physical address
MOV   R2,R0 ; lower order physical address
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
MOV 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
MOV 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:
; RO -> 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::

.IMF DF C$SPRT

.jsr R5,c$sav

.ENDOR

.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

.IMF DF C$SPRT

.JMP C$RET

.IF RET

.RETURN

.ENDOR

; 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$text,i,ro
.IF DF C$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    #$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::

.IP DF C$SPRT
 Jou        R5,c$sav

.ENDC
EXIT$S

.IP 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 ] :
.
.L.
.IFDF $VEC .GOTO 5
.SETS $VEC "400"
.ASKS [::$VEC] $VEC Interrupt vector location ? [ D : 400 ]
.
.
;
; 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
.
;
.IFF $DEL .GOTO 10
PIP ACPUCB.MAC;*/DE
PIP RTHMAC.MAC;*/DE
PIP DQPKT.MAC;*/DE
PIP RWPORT.MAC;*/DE
PIP SCOPY.MAC;*/DE
PIP UCOPY.MAC;*/DE
.
.L.
;
; task builds the acp and creates the image file in [1,54]
;
;
; Create the task builder input definition file
;
.
.OPEN ACPTKB.CMD
.DATA LB:[1,54]RTHACP/AC:5/-CP=
.DATA RTH/LB:CMDTAB,ACPUCB,DQPKT,RWPORT,RTHMAC,UCOPY,SCOPY
.DATA SY:'<UIC>"PROLOGUE/LB:CHDR
.DATA SY:'<UIC>"PROLOGUE/LB,LB:[1,1]EXELIB/LB
.DATA LB:[1,54]RSX11M.STB
.DATA /
.DATA UNITS=7
.DATA TASK=...RTH
.DATA GBLPAT=CMDB:ZEPORT:'$PORT'
.DATA GBLPAT=CMDB:ZSINT:'$VEC'
.DATA ASG=COO: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 DPKT.OBJ;*/DE
PIP RWRT.TSK;*/DE
PIP RTHMAC.OBJ;*/DE
PIP UCOPY.OBJ;*,SCOPY.OBJ;*/DE
PIP ACPTKB.CMD;*/DE
;
; set appropriate protection for the ACP
;
 .ENABLE DISPLAY
.ENABLE QUIET
.DISABLE DISPLAY
.IFNDIF $VRBS .ASK $VRBS Verbose ? [Y/N]
.IFT $VRBS .DISABLE QUIET
.IFNDIF $DEL .ASK $DEL Delete source file from current UFD? [Y/N]
.IFNDIF $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]RSMC,SY:'<UIC>'RWPORT
 MAC UCOPY=LB:[1,1]EXEMC/ML,[11,10]RSMC,SY:'<UIC>'UCOPY
 MAC SCOOPY=LB:[1,1]EXEMC/ML,[11,10]RSMC,SY:'<UIC>'SCOOPY
 MAC ACPIB=LB:[1,1]EXEMC/ML,[11,10]RSMC,SY:'<UIC>'UNIBUS,ACPUCB
 MAC RTHMAC=LB:[1,1]EXEMC/ML,[11,10]RSMC,SY:'<UIC>'RTHMAC
 MAC DQPKT=LB:[1,1]EXEMC/ML,[11,10]RSMC,SY:'<UIC>'DQPKT
 MAC UNIMAC=LB:[1,1]EXEMC/ML,[11,10]RSMC,SY:'<UIC>'UNIMAC
 .
 .; Delete temporary files
 .
 .IFF $DEL .GOTO 10
 PIP ACPIB.MAC;*/DE
 PIP RTHMAC.MAC;*/DE
 PIP DQPKT.MAC;*/DE
 PIP RWPORT.MAC;*/DE
 PIP SCOOPY.MAC;*/DE
 PIP UCOPY.MAC;*/DE
 PIP UNIBUS.MAC;*/DE
 PIP UNIMAC.MAC;*/DE
 .10:
 ;
 ; task builds the acp and creates the image file in [1,54]
 ;
 .
 .; Create the task builder input definition file
 .
 .OPEN ACPTKB.CMD
 .DATA LB:[1,54]RTHACP/AC:5/-CP=
.DATA RTH/LB:CMDTAB,ACPUCB,DQPKT,RWPORT,RTHMAC,UCOPY,UNIMAC,SCOPY
.DATA SY:'<UIC>'PROLOGUE/LB:CHDR
.DATA SY:'<UIC>'PROLOGUE/LB,LB:[1,1]EXELIB/LB
.DATA LB:[1,54]RSX11M.STB
.DATA /
.DATA UNITS=7
.DATA TASK=...RTH
.DATA GBLPAT=CMDTAB:ZEPORT:'$PORT'
.DATA GBLPAT=CMDTAB:ZEINT:'$VEC'
.DATA ASG=COO:5
.DATA //
.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
;
 .ENABLE DISPLAY
RTHACP/AC:5/-CP,RTHACP/-sp/CR=
RTH/LB:CMDTAB,ACPCUB,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
//
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 = $verify(1)
$ on error then $ goto abnormal_exit
$ assign nowhere sys$print
$ !
$ !    now make assignment for RSX11M UNIBUS version
$ !
$ assign dra0:[unilms.] lb:
$ assign dra0:[unilms.] lb0:
$ if "'pl'" .eqs. "" then $ pl = "'$logical("sys$disk")'"$directory()
$ set def 'pl'
$ show def
$ show logical lb
$ !
$ !    now set up environment for C compiler
$ !
$ cpp = "mcr cpp"
$ cpl = "mcr cpl1"
$ cp2 = "mcr cp2"
$ assign dra0:[albert.cutil]cpp.exe ccpp
$ assign dra0:[albert.cutil]cpl.exe ccpp
$ assign dra0:[albert.cutil]cp2.exe ccpp
$ !
$ !    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 ucopuyuni,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,acpucbu
$ mac unimac,unimac/~sp=lb:[1,1]exemc/ml,lb:[11,10]rsxmc,sy:[1,3]unimac
$ mac rthmacuni,rthmacuni/-sp=1b:[1,1]exemc/ml,1b:[11,10]rsxmc,sy:[1,3]rthmac
$ mac dqpktuni,dqpktuni/-sp=1b:[1,1]exemc/ml,1b:[11,10]rsxmc,sy:[1,3]dqpkt
$ !
$ !  C program
$ !
$ cpp -x -i 1b:[1,1]|sy:[10,10]|sy:[1,3] -o sy:[1,3]c1.tmp sy:[1,3]u.h [1,3]body.c
$ cpl -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 cl.tmp;*,c2.tmp;*,c3.tmp;*
$ exit 1
$ abnormal_exit:
$ exit 2
skeleton for bld.com

command file to build the task image

required command files: None

required logical names: None

required parameters:
  pl - 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 "'pl'".eqs."" then $ pl = "'f$logical("sys$disk")'"'f$directory()'"
set def 'pl'
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:[unillm.] 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 "RTHACUNI,UCOPYUNI,SCOPYUNI"
$ write lnkdrv "UNIMAC"
$ write lnkdrv "$y:[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:[unilmp.] lb:
$ open/write lnkdrv tkb.cmd
$ write lnkdrv "RTHACPUF/AC:5/-CP,RTHACPUF/-sp/CR="
$ write lnkdrv "RTHUP/LB:CMDTAB,ACPUCBU,DQPKTUP,RWPORTUP"
$ write lnkdrv "RTHMACUP,UCOPYUP,SCOPYUP"
$ write lnkdrv "UPMAC"
$ write lnkdrv "$y:[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 "'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:
$assign _dra0:[qbus11m.] lb0:
$!
   if "'pl" .eqs. "" then $ pl = "'f$logical("sys$disk")"'f$directory()"
$set def 'pl'
$show def
$show logical lb
$!
$cpp == "mcr cpp"
$cpl == "mcr cpl"
$cp2 == "mcr cp2"
$assign dra0:[albert.cutil]cpp.exe cpp
$assign dra0:[albert.cutil]cpl.exe cpl
$assign dra0:[albert.cutil]cp2.exe cp2
$!
$! go compile all the files
$!
$lbr rth/cr
$mac rwport,rwport/-sp=1b:[1,1]exemc/ml,1b:[11,10]rsxmc,sys:[1,3]rwport
$mac ucopy,ucopy/-sp=1b:[1,1]exemc/ml,1b:[11,10]rsxmc,sys:[1,3]ucopy
$mac scopy,scopy/-sp=1b:[1,1]exemc/ml,1b:[11,10]rsxmc,sys:[1,3]scopy
$mac acpucb,acpucb/-sp=1b:[1,1]exemc/ml,1b:[11,10]rsxmc,sys:[1,3]acpucb
$mac rthmac,rthmac/-sp=1b:[1,1]exemc/ml,1b:[11,10]rsxmc,sys:[1,3]rthmac
$ mac dpkt,dpkt/-sp=lb:[1,1]exemc/ml;lb:[11,10]rsxmc,sy:[1,3]dpkt
$ !
$ !  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
$ cpl -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     rwpport.mac   exos$mfg:[rsx]
$ copy/log     ucopy.mac     exos$mfg:[rsx]
$ copy/log     scopy.mac     exos$mfg:[rsx]
$ copy/log     acpuch.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 "'-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-Plus UNIBUS version
!
assign _dra0:[unillmp.] lb:
assign _dra0:[unillmp.] lb0:
if "'-pl'" .eqs. "'" then $ pl = "'"f$logical("sys$disk")'"f$directory()"'
set def 'pl'
show def
show logical lb
!
now set up environment for C compiler
!
cpp == "mcr cpp"
cpl == "mcr cpl"
cp2 == "mcr cp2"
assign dra0:[albert.cutil]cpp.exe cpp
assign dra0:[albert.cutil]cpl.exe cpl
assign dra0:[albert.cutil]cp2.exe cp2
!
go compile all the files
!
lbr rthup/cr
mac rwpportup, rwpportup/-sp=lb:[1,1]exemc/ml, lb:[11,10]rsxmc, sy:[1,3]rwpport
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 1b:[1,1]|sy:[10,10]|sy:[1,3] -o sy:[1,3]c1.tmp sy:[1,3]u.h 1,3]body.c
$ cpl -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

.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 $SESSION 1
.ASKN [:::$SESSION] $SESSION 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 <STRLN> 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]RWPRT.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]zttyt.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]ztos.MAC/NM
PIP /NV/CD='DEV'[1,1]ztddw.MAC/NM
PIP /NV/CD='DEV'[1,1]ztbl1.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]ztp.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
.
.
; 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 $NOMP .GOTO 25
.PIP 'DESTUI'ARP.TSK;*/DE
.PIP 'DESTUI'BSTAT.TSK;*/DE
.PIP 'DESTUI'NETLOAD.TSK;*/DE
.PIP 'DESTUI'NETSTAT.TSK;*/DE
.PIP 'DESTUI'TCP.TSK;*/DE
.PIP 'DESTUI'XROUTE.TSK;*/DE
.PIP 'DESTUI'FTP.TSK;*/DE
.PIP 'DESTUI'FTPDEMON.TSK;*/DE
.PIP 'DESTUI'TELNET.TSK;*/DE
.PIP 'DESTUI'LOGIN.TSK;*/DE
.PIP 'DESTUI'FTPD.TSK;*/DE
.25:
.DMO '$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]PASSWORD.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 $DEL PIP BLDLN.CMD;/DE
.PIP 'DESTUI'/FO/CO/NV/CD=SY:'<UIC>'LOGIN.TSK/NM
.PIP LOGIN.TSK;/DE/NM
.PIP /FO/NV/CD='$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]FTP.D.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 INITINO Do you want to initialize the network addresses file (HOSTS.NET)
.IFF INITINO .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 $NOPRE PIP LB:[1,1]EXOSLOAD.CMD;*/DE
.OPEN LB:[1,1]EXOSLOAD.CMD
.DATA .ENABLE SUBSTITUTION
.DATA .IFACT DEMTO ABO DEMTO
.DATA .IFACT LGNTO ABO LGNTO
.DATA .IFACT ...DEM ABO ...DEM
.DATA .IFACT ...LGN ABO ...LGN
.SETN LCOUNT 0
.
.80$:
 .IF  LCOUNT  >= 'S$ESS' .GOTO 89$
 .DATA .IFACT FTD00'LCOUNT' ABO FTD00'LCOUNT'
 .DATA .IFINS FTD00'LCOUNT' REM FTD00'LCOUNT'
 .DATA .IFINS XDRO0'LCOUNT' REM XDRO0'LCOUNT'
 .INC LCOUNT
 .GOTO 80$
 .89$:
 .DATA .IFINS ...DEM REM ...DEM
 .DATA .IFINS ...ARP REM ...ARP
 .DATA .IFINS ...BST REM ...BST
 .DATA .IFINS ...FTP REM ...FTP
 .DATA .IFINS ...NET REM ...NET
 .DATA .IFINS ...TEL REM ...TEL
 .DATA .IFINS ...TTC REM ...TTC
 .DATA .IFINS ...ROU REM ...ROU
 .DATA .IFINS ...NST REM ...NST
 .DATA .IFINS ...LGN REM ...LGN
 .DATA .IFACT ...RTH ABO ...RTH
 .DATA .IFACT RTHTO ABO RTHTO
 .DATA .IFINS ...RTH REM ...RTH
 .DATA .IF <SYSTEM> => 6 .GOTO 10$
 .DATA .IFLOA ZE: CON OFFLINE ZEA
 .DATA .IFLOA ZE: CON OFFLINE ZEO:
 .DATA .IFNOA 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 ZTO:
 .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$:
 .DATA .IFLOA ZE: UNL ZE:
 .DATA .IFLOA ZT: UNL ZT:
 .DATA LOA ZE:/PAR=GEN/HIGH/SIZE=20000
 .DATA .IF <SYSTEM> => 6 LOA ZT:
 .DATA .IF <SYSTEM> => 6 UNL ZT:
 .DATA ; You can ignore the error message: "Loadable driver larger than 4KW"
 .DATA LOA ZT:/HIGH/SIZE=20000
.DATA 
.DEF <SYSTEM> <> 6 .GOTO 20$
 .DATA ; configure the devices online
 .DATA CON ONLINE ZEA
 .DATA CON ONLINE ZEO:
 .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 ZTI VEC=0
 .DATA CON ONLINE ALL
 .DATA .20$:
 .DATA INS SRTACP/PRI=150.
 .DATA .XQT RTH
 .DATA INS 'DESTUI'ARP.TSK
 .DATA INS 'DESTUI'BSTAT.TSK
 .DATA INS 'DESTUI'FTPC.TSK
 .DATA INS 'DESTUI'FTPD.TSK
 .DATA INS 'DESTUI'FTPDEMON.TSK
 .DATA INS 'DESTUI'NETLOAD.TSK
 .DATA INS 'DESTUI'TELNET.TSK
 .DATA INS 'DESTUI'TCPC.TSK
 .DATA INS 'DESTUI'XROUTE.TSK
 .DATA INS 'DESTUI'NETSTAT.TSK
 .DATA INS 'DESTUI'LOGIN.TSK
 .SETN LCOUNT 0
 .DATA .SETS FTDOPT ""'
 .DATA .IF <SYSTEM> = 6 .SETS FTDOPT "/XHR=NO"
 .90$:
 IF LCOUNT >= '$SESS' .GOTO 99$
 .DATA INS 'DESTUI'FTPD.TSK/ TASK=FTD00/ LCOUNT' 'FTDOPT'
 .DATA INS $SPIP/TASK=XDR00/ LCOUNT'
 .INC LCOUNT
 .GOTO 90$
 .99$:
 .DATA .ASK DWN Do you want to initialize the EXOS front end processor
 .DATA .IPT DWN net
 .DATA .ASK DMN Do you want to start the FTP server
 .DATA .IPT DMN .XQT dem
 .DATA .IPT 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

.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]ACPCUCB.MAC/NM
PIP /NV/CD='DEV'[1,1]DQPKT.MAC/NM
PIP /NV/CD='DEV'[1,1]RWPRT.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]blgst.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]zttyt.MAC/NM
PIP /NV/CD='DEV'[1,1]ztini.MAC/NM
PIP /NV/CD='DEV'[1,1]zttrw.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]ztosis.MAC/NM
PIP /NV/CD='DEV'[1,1]ztddat.MAC/NM
PIP /NV/CD='DEV'[1,1]ztttbl.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
;
;
.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 DEMTO ABO DEMTO
.DATA .IFACT LGNTO ABO LGNTO
.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 .IPINS FTD00'LCOUNT' REM FTD00'LCOUNT'
.DATA .IPINS XDR00'LCOUNT' REM XDR00'LCOUNT'
.INC LCOUNT
.GOTO 80$

.89$:
.DATA .IPINS ...DEM REM ...DEM
.DATA .IPINS ...ARP REM ...ARP
.DATA .IPINS ...BST REM ...BST
.DATA .IPINS ...FTP REM ...FTP
.DATA .IPINS ...NET REM ...NET
.DATA .IPINS ...TEL REM ...TEL
.DATA .IPINS ...TTC REM ...TTC
.DATA .IPINS ...ROU REM ...ROU
.DATA .IPINS ...NST REM ...NST
.DATA .IPINS ...LGN REM ...LGN
.DATA .IFACT ...RTH ABO ...RTH
.DATA .IFACT RHTO ABO RHTO
.DATA .IPINS ...RTH REM ...RTH
.DATA .IF <SYSTEM> <= 6 .GOTO 10$
.DATA .IFLOA ZE: CON OFFLINE ZEA
.DATA .IFLOA ZE: CON OFFLINE ZEO:
.DATA .IFNOA 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 ZTO:
.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 ZEO:
.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 >= 'S$ESS' .GOTO 99$
.DATA INS 'DESTUI'FTP.TSK/TASK=FTDOO'LCOUNT'''FTDOPT"
.DATA INS $PIP/TASK=XRDOO'LCOUNT'
.INC LCOUNT
.GOTO 90$
.99$:
.DATA .ASK DWN Do you want to initialize the EXOS front end processor
.DATA .IFT DWN net
.DATA .ASK DWN Do you want to start the FTP server
.DATA .IFT DWN .XQT dem
.DATA .IFT DWN .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 BLDLN.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'READR.C;*/DE
.IFT $NOPRE PIP 'DESTUI'CONNECT.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 'DESTUI8030.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 $NOPRE PIP LB:[1,1]EXOSLOAD.CMD;*/DE
.OPEN LB:[1,1]EXOSLOAD.CMD
.DATA .ENABLE SUBSTITUTION
.DATA .IFACT DEMTO ABO DEMTO
.DATA .IFACT LGNTO ABO LGNTO
.DATA .IFACT ...DEM ABO ...DEM
.DATA .IFACT ...LGN ABO ...LGN
.SETN LCOUNT 0
.80$:
.IF LCOUNT >= '$SESS'.GOTO 89$
.DATA .IFACT FTD00'LCOUNT' ABO FTD00'LCOUNT'
.DATA .IFACT FTD00'LCOUNT' REM FTD00'LCOUNT'
.DATA .IFACT XRD00'LCOUNT' REM XRD00'LCOUNT'
.INC LCOUNT
.GOTO 80$
.89$:
.DATA .IFACT ...DEM REM ...DEM
.DATA .IFACT ...ARP REM ...ARP
.DATA .IFACT ...BST REM ...BST
.DATA .IFACT ...FTP REM ...FTP
.DATA .IFACT ...NET REM ...NET
.DATA .IFACT ...TEL REM ...TEL
.DATA .IFACT ...TTC REM ...TTC
.DATA .IFACT ...ROU REM ...ROU
.DATA .IFACT ...NST REM ...NST
.DATA .IFACT ...LGN REM ...LGN
.DATA .IFACT ...RTH ABO ...RTH
.DATA .IFACT RTHTO ABO RTHTO
.DATA .IFACT ...RTH REM ...RTH
.DATA .IF <SYSTEM> <> 6 .GOTO 10$
.DATA .IFLOA ZE: CON OFFLINE ZEA
.DATA .IFLOA ZE: CON OFFLINE ZEO:
.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 ZEO:
.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 $RTAC/PRI=150.
.DATA .XQT RTH
.DATA INS 'DESTUI'ARP.TSK
.DATA INS 'DESTUI'BSAT.TSK
.DATA INS 'DESTUI'FTP.TSK
.DATA INS 'DESTUI'FTPD.TSK
.DATA INS 'DESTUI'FTPDEMOS.TSK
.DATA INS 'DESTUI'NETLOAD.TSK
.DATA INS 'DESTUI'TELNET.TSK
.DATA INS 'DESTUI'TTCP.TSK
.DATA INS 'DESTUI'XROUTE.TSK
.DATA INS 'DESTUI'NETSTAT.TSK
.DATA INS 'DESTUI'LOGIN.TSK
.SETN LCOUNT 0
.DATA .SETS FTDPT ""
.DATA .IF <SYSTEM> = 6 .SETS FTDPT "/XHR=NO"
.90$
. IF LCOUNT = 'SSESS' .GOTO 99$
.DATA INS 'DESTUI'FTP.TSK/TASK=FTD00'LCOUNT''FTDOPT'
.DATA INS $PIP/TASK=XDROO'LCOUNT'
.INC LCOUNT
.GOTO 90$

.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 every time 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.
Oct 17 16:18 1985 tapeuni.cmd Page 1

; 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> .go 5
.disable quiet
; Error: You must be privileged in order to install EXOS 8030 software.
.exit

.5:
.ift <uc> = "[1,100]" .go 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 ]
.setw $sess l
.askn [::$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
@bldztr
.ift $del pip bldztr.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>'TTCPS.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
.PTP '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'RHOSUS.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:1,2NET;/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>'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$:
.DATA .IFINS ...DEM REM ...DEM
.DATA .IFINS ...ARP REM ...ARP
.DATA .IFINS ...BST REM ...BST
.DATA .IFINS ...FTP REM ...FTP
.DATA .IFINS ...NET REM ...NET
.DATA .IFINS ...TEL REM ...TEL
.DATA .IFINS ...TTC REM ...TTC
.DATA .IFINS ...ROU REM ...ROU
.DATA .IFINS ...NST REM ...NST
.DATA .IFINS ...LGN REM ...LGN
.DATA .IFACT ...RTH ABO ...RTH
.DATA .IFACT RHT0 ABO RHT0
.DATA .IFINS ...RTH REM ...RTH
.DATA .IF <SYSTEM> <> 6 .GOTO 10$
.DATA .IFLOA ZE: CON OFFLINE ZEA
.DATA .IFLOA ZE: CON OFFLINE ZEO:
.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 ZEO:
.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=FTDOO'LCOUNT''FTDOPT''
.DATA INS $PIP/TASK=XDR00'LCOUNT'
   .INC LCOUNT
   .GOTO 90$
.data
 DATA .ASK DWN Do you want to initialize the EXOS front end processor
 DATA .IPT DWN net
 DATA .ASK DMN Do you want to start the FTP server
 DATA .IPT DMN .XQT dem
 DATA .IPT 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 every time 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.
/*
 * filename: FTPDEMON.H
 */

#include <rsxos.h>
#define EXEFN 010001 /* max. no of connections */
#define MAXCONN 4 /* common event flag no. 50 */
#define ACC_EFN 50 /* common event flag no. 51 */
#define SLEEP_EFN 51 /* length of task name */
#define TASKNAMLEN 6 /* length of task name */
#define SDRA 01153

struct task_block {
    struct task_block *link; /* link to next task block */
    char task_name[TASKNAMLEN]; /* task name */
    int esb[8]; /* exit status block */
} tskblk[MAXCONN] = {0};

/* GLOBAL variables */

struct task_block *rdy2run = tskblk; /* ptr to rdy 2 run task */
struct task_block *accept_on = 0; /* pointer to task in accept */

char cmdlin[] = "INS LB:[1,2]FTPD/TASK=FTD00 ";
char line[] = "REM FTD00 ";
long cli = 0; /* CLI name in RAD50 */
int cmdlen = 0;
int len = 0;
int flgbuf[4] = {0}; /* event flag buffer */
int connect = 1; /* total no of connections */
char *ftpcmd = (char *) 0;
int ftplen = 0;
int tcblist[MAXCONN] = {0}; /* pointer to task control block */
/*
 * filename: FTPDEMON.C
 */

/* This file contains the code for the master ftp task which monitors the
 * generation of different ftp daemons for different connections.
 */

#include "ftpDemon.h"
extern int ast();
extern long radix();
int connect = 1;
extern int ast_recv();

int main()
{
    priv_user();    /* check user is priv &
                    task is not active    */
    gmcrc();
    emt(SDRA,ast_recv);    /* specify receive data ast */
    initialize();

    FOREVER {
        if(!read_efn(flgbuf)) {    /* is efn 50 clear?    */
            if(ry2run) {    /* any rdy2run task present */
                emt(SETF,ACC_EFN);    /* set common efn 50 to
                                          indicate accept is on */
                ins_spawn();    /* install and spawn one */
                update();        /* update rdy2run pointer */
            }
        }
        emt(ENAR);    /* enable ast recognition */
        emt(STSE,SLEEP_EFN);    /* sleep */
        emt(CLEF,SLEEP_EFN);    /* clear sleep efn */
        emt(DSAR);    /* disable ast recognition so that it does */
        /* not interfere with main task's execution */
    }    /* end of FOREVER */

    /* INITIALIZE
     *
     * Initialize the world of MASTER
     */
    initialize()
    {
        register struct task_block *t = tskblk;    /* start of task block */
        int i,j;
        cli = radix("MCR...");
        cmdlen = strlen(cmdlin);
        len = strlen(line);
for(i=0;i<connect;i++) {
    tcblist[i] = 0;
    for(j=0;j<5;j++)
        t->task_name[j] = cmdlin[cmlen - TASKNAMLEN +j];
    t->task_name[5] = '0' + i;
    t->link = t + 1;
    t++;
}

(--t)->link = 0;
emt(CLEF,ACC_EFN);
emt(CLEF,SLEEP_EFN);
emt(DSAR);  /* disable ast recognition so that ast's do not */
            /* bother the masin task */
}

/*
* UPDATE
*
*       Update rdy2run pointer
*
*/

update()
{
    accept_on = rdy2run;
    rdy2run = rdy2run->link;
    accept_on->link = 0;
}

/* FROM_AST
*
*       This routine is called from the AST routine when a task
*       exits
*
*/

from_ast(p)
int *p;  /* pointer to esb of exit task */
{
    register struct task_block *exit_task;
    int index;

    exit_task = (struct task_block *) (p-4); /* point to start of str. */
    exit_task->link = rdy2run;
    rdy2run = exit_task;  /* make the exit task the next available */
    /* rdy2run task */

    line[len - 1] = exit_task->task_name[TASKNAMLEN -1];  /* the task no. */
    /*emt(SWPN,cli,0,0,0,0,EXEFN,0,0,line,len,0,CO);*/  /* rem task */
    /*emt(WTSE,1);*/  /* wait for task to get removed */
    index = exit_task - tskblk;
    if(tcblist[index])
        mkpriv(tcblist[index]);  /* make sure task becomes priv. */
    if(accept_on == exit_task)
        emt(CLEF,ACC_EFN);  /* then task has exit before accept */
    emt(SETF, SLEEP_EFN);  /* unstop the master task */
113 /*
114  * INS_SPAWN
115  *
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       /* now install the task */
128 
129       /* st = emt(SPWN, cli, 0, 0, 0, 0, EXEFN, 0, 0, cmdlin, cmdlen, 0, CO); */
130       /* emt(WTSE, 1); */ /* wait for task to get installed */
131       /* now spawn the task */
132       st = emt(SPWN, t_name, 0, 0, 0, 0, 010000, ast, rdy2run->esb, ftpcmd, ftplen, 0, 0); 
133       if(st == IE_ACT) 
134           /* yet to decide what to do if task is active */
135           /* such a condition should never arise but if it does */
136           /* then what? */
137 
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 }
; FILENAME: DEMON.MAC

This file includes AST service routine for demon. It also has a routine to read EFN 50.

.psect c$text,i,ro
.mcalls RDAF$

READ.EFN:

.jsr R5,c$sav
.mov 4(R5),r0 ; pointer to a 4 word buffer
.rdafs r0 ; read all event flags
.bit #1,6(r0) ; check if event no. 50 is set or clear
.bne 10$ ; if NE then it is set
.clr r0 ; clear return value also

10$:

.jmp c$ret ; if efn is set then return value is > 0

; GMCR

.psect c$data,d,rw
.mcalls DIR$,GMCR$

GMCR:

.cmcr$

.psect c$text,i,ro

GMCR:

.jsr R5,c$SAV ; save registers
.dir$ #GMCRD ; get MCR command line
.cmp #IE,AST,$DSW ; check return status
.beq NMC$ ; if EQ No MCR Command
.mov #GMCRD+G.MCRB,FTP CMD ; get mcr buffer address
.mov $DSW,FTPLE N ; buffer size

NMC$:

.jmp C$RET ; unsave register's and return

; tsk:

.psect c$data,d,rw

.if define R$$MPL
.rad50 /DEMT0/
.iff ;R$$MPL
.rad50 /...DEM/
.endc ;R$$MPL

rcode:

.word 0

; R1: .asciz /*FATAL*/ USER MUST BE PRIVILEGED/
.even

; R2:
.ASCIZ /***FATAL**-- TASK ALREADY ACTIVE/
.EVEN

.MCALL TCBDF$UCBDF$,QIW$S,EXIT$S,DCBDF$
DCBDF$
TCBDF$
UCBDF$

.psect c$text,i,ro
.ENABLE LSB

priv.user:

JSR R5,C$SAV
CALL $SWSTK,RET ; switch to system state
MOV $TKTCB,R0 ; get current TCB address
MOV T.UCB(R0),R1 ; get TI: UCB address
BIT #U2.PRIV,U.CW2(R1);; check user is priv.
BEQ ERR1 ; If EQ user is not priv.
CMP tsk,T.NAM(R0) ;; compare first word of task name
BNE ERR2 ;; if NE task already active.
CMP tsk+2,T.NAM+2(R0);; compare second word of task name
BNE ERR2 ;; if NE task already active
MOV #$DEVHD,R1 ;; get gevice header
10$:
MOV (R1),R1 ;; get next DCB address
BEQ 20$ ;; if EQ none
CMP "$CO,D.NAM(R1) ;; is it console
BNE 10$ ;; if NE no
MOV D.UCB(R1),T.UCB(R0) ;; get CO UCB address
20$:
BR RTN ;;
ERR1:
MOV #-1,RTNCODE ;; user must be priv.
BR RTN
ERR2:
MOV #-2,RTNCODE ;; task already active
RTN:
RETURN ;; return to task state
RET:
MOV RTNCODE,R0 ;; return value
BEQ SUCC
CMP #-1,R0 ; check error code
BEQ E1 ;
BR E2
EL: MOV #ERL,R1 ; address of error message
BR ERMSC
E2: MOV #ERR2,R1 ; address of error message
BR ERMSC
SUCCE:
JMP C$RET
ERMSCE:
QIW$S #IO.WVB,#5,#1,“,,<R1,#38.,#40>
EXIT$S

.psect c$text,i,ro
mkpriv::
JSR R5,C$SAV
MOV 4(R5),R0 ; get tcb address
CALL $SWSTK,RET1 ; switch to system state
BIS #T3.PRV,T.ST3(R0) ; make server as priv.
RETURN
RET1:
JMP C$RET
.DSABL LSB
.psect c$data,d, rw
.even
.psect c$text,i,ro
.mcalls ASTX$S
AST::
MOV R0,-(SP) ; save R0
MOV R1,-(SP) ; save R1
MOV R2,-(SP) ; save R2
MOV R3,-(SP) ; save R3
MOV R4,-(SP) ; save R4
MOV R5,-(SP) ; save R5
MOV 14(SP),-(SP) ; 1st param is the esb address on the stack
JSR PC,FROM.AST ; call C - routine to do the job
TST (SP)+ ; pop off param passed
MOV (SP)+,R5 ; pop off R5
MOV (SP)+,R4 ; pop off R4
MOV (SP)+,R3 ; pop off R3
MOV (SP)+,R2 ; pop off R2
MOV (SP)+,R1 ; pop off R1
MOV (SP)+,R0 ; pop off R0
TST (SP)+ ; pop off stack for ast
ASTX$S ; exit from AST routine
.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$,RCVD$,SDAT$,ASTX$
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$ #PKT ; receive 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 $ACTHD,RO ; get active task header pointer
52 20$:
53 CMP T.NAM(RO),PKT ; compare first word of task
54 BNE NXT ; If NE not match , next tcb
55 CMP T.NAM+2(RO),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 MOV   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$:
73 MOV   T.PCB(R0),R0  ;; get PCB address of task
74 MOV   P,HDR(R0),R0  ;; Get header control block
75 MOV   PKT+4,H.CUIC(R0)  ;; set current task uic as remote user's
76 MOV   PKT+4,H.DUIC(R0)  ;; set default task uic
78 30$:
80 RETURN  ;; switch to task state
81 RET:
82 SDAT$  #PKT,#PKT+4  ;; send dummy pkt to child task
83 BR    AGAIN  ;; go for next pkt.
84 ERR:
85 EXT:
86 UNSAVE  ;; unsave all registered
87 ASTX$  ;; exit from AST routine
89 90 .END

```c
#include <rsxos.h>
#define EFN 1
extern valacnt();
extern char *entry;
char *msg = " ";

main()
{
    register int  i,r;
    char  *p;
    for(;;)
    {
        /*
        if(emt(RCVX,(long) 0, msg) < 0)
            emt(EXST,-2);
        */
        if(emt(RCST,(long) 0, msg) == IS_SET)
            continue;
        for(i=4;msg[i] != '*';i++);
        i++;
        r = valacnt(msg+4,msg+i);
        *((int *)(msg + 4)) = r;
        if( r == 0)
        {
            p = msg+6;
            *p++ = '[';
            for(i=0;i<3;i++)
                *p++ = *(entry + A_GRP + i);
            *p++ = ':';
            for(i=0;i<3;i++)
                *p++ = *(entry + A_MBR + i);
            *p++ = ']';
            *p++ = '0';
            /* now fill in the login default device name starting at 16th */
            for(i=0;i<4;i++)
                *p++ = *(entry + A_SYDV + i);
            *p++ = '\0';
        }
        emt(SDAT,*((long*)msg,msg+4,0);
    }
}

extern int namflg;
extern char *puic;

acct(ac)
char  *ac;

{ int  hasbracket = 0;
int  charcount;
int  leadzero;  /* count of leading zeroes needed */
char  *chptr;
char  *delimiter;  /* delimiter */
while (*ac == ' ')
    ac++;  /* skip blank */
if (((*ac >= 'A') && (*ac <= 'Z'))) {
    namflg = 1;
    return(1);
}
    } else if (*ac == '[') {
        hasbracket = 1;
        ac++;  
    } else if (((*ac < '0') && (*ac > '7')) {
        return(2);
    }

    /* now must start with a numeric number */
    chp.ptr = ac;
    charcount = 0;
    while (((*chp.ptr != ' ') && (*chp.ptr != ',')) {
        if (**charcount > 3)
            return(2);   /* group number too long */
        chp.ptr++;
    }
    delimiter = chp.ptr;

    for (leadzero = 3 - charcount; leadzero > 0; leadzero--)
        *puic++ = '0';
    for (chp.ptr = ac; charcount > 0; charcount--) {
        if ((*chp.ptr < '0') || (*chp.ptr > '7'))
            return(2);   /* syntax error */
        *puic++ = *chp.ptr++;
    }

    while (*chp.ptr == ' ')
        chp.ptr++;
    /* skip blank */
    if (*chp.ptr == ',') {
        chp.ptr++;
    } else
        return(2);
    while (*chp.ptr == ' ')
        chp.ptr++;
    /* skip blank */

    /* now handle the member part */
    delimiter = chp.ptr;
    charcount = 0;
    while (((*chp.ptr != ' ') && (*chp.ptr != ']') && (*chp.ptr != ',*')) {
        if (**charcount > 3)
            return(2);   /* member number too long */
        chp.ptr++;
    }

    if (((*chp.ptr == ']') && (!hasbracket))
        return(2);

    for (leadzero = 3 - charcount; leadzero > 0; leadzero--)
        *puic++ = '0';
    for (chp.ptr = delimiter; charcount > 0; charcount--) {
        if ((**chp.ptr < '0') || (**chp.ptr > '7'))
            return(2);   /* syntax error */
        *puic++ = *chp.ptr++;
    }
    if (hasbracket) {

    }
while (*chptr != ']') {
    if (*chptr == '*')
        return(2);
    chptr++;
}
return(0);
1 ;       .TITLE ACTFIL - ACCOUNT FILE CONTROL BLOCKS
2 .NLIST
3     .IDENT /4.0/
4 ;
5 ;       COPYRIGHT (C) 1981 BY
6 ;       DIGITAL EQUIPMENT CORPORATION, MAYNARD
7 ;       MASSACHUSETTS. ALL RIGHTS RESERVED.
8 ;
9 ;       THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
10 ; AND Copied only in accordance with the terms of such license
11 ; AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS
12 ; SOFTWARE OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR
13 ; OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON. NO TITLE TO AND
14 ; OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERED.
15 ;
16 ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
17 ; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
18 ; EQUIPMENT CORPORATION.
19 ;
20 ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
21 ; ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
22 ;
23 ;-------------------------------------------------------------------------
24 ;
25 ; COPYRIGHT (C) 1981 BY DIGITAL EQUIPMENT CORPORATION.
26 ; ALL RIGHTS RESERVED.
27 ;
28 ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
29 ; OR COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE.
30 ;
31 ;
32 ; VERSION 04
33 ; BY: H. LEV
34 ; DATE: 7/15/75
35 ;
36 ; MODIFIED:
37 ;
38 ; EB051 21-MAY-77 LOOK FOR ACNT FILE ON LB: RATHER THAN SY:
39 ;
40 ; DG002 LOOK FOR LATEST VERSION OF RSXi1.SYS
41 ;
42 ; MLG007 03-NOV-78 FIND PHYSICAL LB:
43 ;
44 ; MLC044 30-JAN-79 SPOOL LISTING FILE (ACNT)
45 ;
46 ; MLC081 10-APR-79 DO NOT LEAVE ACCOUNT FILE LOCKED
47 ;
48 ;
49 ; SA213 ADD FIELDS FOR SLAVE BIT, DEFAULT CLI NAME
50 ; AND CHANGE OPENING OF ACNT FILE
51 ;
52 ;
53 .LIST
54 .MCALL FDBDF$,FDOP$A,FSRSZ$
55 .IF DF R$$MPL
.MCALL ACTDF$  
.IFF ;R$MPL
.MACRO ACTDF$,L,B
.ASECTION
.=0
A.GRP:'L' .BLKB 3 ; GROUP CODE (ASCII)
A.MBR:'L' .BLKB 3 ; MEMBER CODE
A.PSWD:'L' .BLKB 6 ; PASSWORD
A.LNM:'L' .BLKB 14. ; LAST NAME
A.FNM:'L' .BLKB 12. ; FIRST NAME
A.LDAT:'L' .BLKB 6 ; DATE OF LAST LOG ON (DD/MM/YY HH:MM:SS
A.NLOG:'L' .BLKB 2 ; TOTAL NUMBER OF LOGONS
A.SYDV:'L' .BLKB 4 ; DEFAULT SYSTEM DEVICE
.ACLI:'L' .BLKW 1 ; UNUSED
.A.LPRV:'L' .BLKW 2 ; RAD50 DEFAULT CLI NAME
.A.LPRV:'L' .BLKW 2 ; UNUSED (FOR COMPATIBILITY W/ MPLUS)
.A.LPRV:'L' .BLKW 1 ; LOGIN PRIVILEGE WORD
.A.LPRV:'L' .BLKW 1 ; UNUSED
A.LEN = 'B' 128. ; LENGTH OF CONTROL BLOCK

; BIT DEFINITION ON A.LPRV - LOGIN PRIVILEGES
AL.SLV = 'B' 1 ; SLAVE TERMINAL ON LOGIN
.PSELECT
.ENSEM
.ENDEC ;R$MPL

; CONSTANTS
LUN2 == 2 ; ACCOUNT FILE LUN
$BFLEN == 2048. ; LENGTH OF ACCOUNT FILE BUFFER
ACTDF$ <:,<= ; DEFINE OFFSETS INTO ACCOUNT FILE

$ACTFL:: FDBDF$ ; DEFINE ACCOUNT FILE FDB
.IF DF R$MPL
FDOP$A LUN2,DSPT,,,FA.ENB!FA.DLK!FA.EXC
.IFF ;R$MPL
FDOP$A LUN2,DSPT,,,FA.ENB!FA.DLK ; SETUP LUN, DSD, AND P.ACTL
.ENDC ;R$MPL
.DSPT: .WORD 0 ; DATA SET DESCRIPTOR
.WORD 0 ; DEVICE NAME (ALUN USED)
.WORD 5
.WORD DIRNAM
.WORD 9.
.WORD FILNAM
115 DIRNAM: .ASCII /[0,0]/
116 FILNAM: .ASCII /RSX11.SYS/
117 .EVEN
118
119 FSRSZ$ 1 ; SET UP FOR A FILE IN GET PUT MODE
120
121 $ACTBF$: .BLKB $BFLEN ; CREATE ACCOUNT FILE BUFFER
122 .EVEN
123 .END
filename: PASWORD.MAC

; This routine is callable from 'C' as well as from a Macro program.
; If C$SPRT is defined then it becomes callable from 'C'.

.MCALL DCBDF$
DCBDF$
C$SPRT = 1

; DATABASE

.MCALL QIO$,MRKT$,WTSE$,QIOW$,ALUN$,CLOSE$
.MCALL OPEN$,R$,FINIT$,GET$
.MCALL NBOF$L

.IF DF R$MPL
.MCALL OPNS$U
.IFF ;R$MPL
.MCALL OPEN$U
. ENDC ;R$MPL

.psect c$data,d,rw
.enabl gbl

ENCRIPT = 0 ; ENCRYPTION SUBROUTINE NOT PRESENT
LUN4 = 4 ; LUN FOR SYSTEM DEVICE
EFN1 = 1 ; EVENT FLAG FOR ALL I/O
PSWDBF:.WORD 0 ; ADDRESS OF PASSWORD BUFFER
UIC:.ASCII /000000/; UIC
PUIC:.WORD UIC ; POINTER TO UIC
NAME:.ASCII / ; LAST NAME AREA IF NAME USED
.EVEN
NBOF$L ; DEFINE BLOCK OFFSETS

FDBP:.QIO$ IO,RVB,LUN2,EFN1,IOSB,<$ACTBF,$BFLEN,>,1>
IOSB:.BLKW 2 ; I/O STATUS BLOCK
OPNERR:.WORD 0 ; A/C FILE OPEN ERROR FLAG
FILOPN:.WORD 0 ; FILE OPEN IF = 1.
NAMFLG:.WORD 0 ; NAME FLAG, 0 = A/C, 1 = NAME
ENTRY:.WORD 0 ; ADDRESS OF A/C ENTRY
MKT:.MRKT$ 1,60,1 ; WAIT FOR 1 SEC
FRMPTR:.WORD 0 ; C - FRAME POINTER STORAGE
ER1:.ASCIZ <15>/**FATAL**-----CANNOT FIND PHYSICAL LB:/
ER2:.ASCIZ <15>/**FATAL**-----ACCOUNT FILE OPEN ERROR/
ER3:.ASCIZ <15>/**FATAL**-----INVALID ACCOUNT/
.EVEN

; INPUTS TO MAC CALLABLE ROUTINE
; R3 --> POINTER TO ACCOUNT
57 ;        R4  --  POINTER TO PASSWORD BUFFER
58 ;
59 .psect  c$text,i,ro
60
61 VALACNT::
62
63 .IF    DF    C$SPRT
64 jsr    R5,c$sav
65 MOV    R5,FRMPTR ; SAVE FRAME POINTER
66 MOV    4(R5),R3 ; GET POINTER TO ACCOUNT OR NAME
67 MOV    6(R5),R4 ; GET POINTER TO PASSWORD
68 .ENDC
70
71 ;
72 ; NOW FILL UP UIC AND PASSWORD IN THEIR RESPECTIVE PLACES
73 ;
74 MOV    #UIC,PUIC ; set up pointer to UIC
75 MOV    R3,(SP) ; PARAM  --  POINTER TO ACCOUNT OR NAME
76 CALL    ACNT ; CHECK FOR ACCOUNT OR USER NAME
77 TST    R0 ; RETURN CODE
78 BEQ    15$ ; IF EQ THEN ACCNT SPECIFIED CORRECTLY
79 ; AND XFERED UIC TO CORRECT PLACE
80 CMP    R0,#1 ; SEE IF NAME SPECIFIED OR NOT
81 BEQ    10$ ; IF EQ THEN IT IS SPECIFIED
82 JMP    ERR3 ; SYNTAX ERROR
83 10$:
84 MOV    #NAME,R2 ; ADDRESS OF NAME
85 MOV    #14.,R1 ; LENGTH OF NAME
86 12$:
87 MOVB   (R3)+,(R2)+ ; XFER NAME
88 SOB    R1,12$ ; LOOP
89 15$:
90 MOV    R4,PSWDBF ; ADDRESS OF PASSWORD
91 20$:
92 SWSTK$ 50$ ; ; SWITCH TO SYSTEM STATE
93 MOV    $DEVHD,R2 ; ; START AT BEGINNING OF DEVICE TABLE
94 30$:
95 CMP    D.NAM(R2),"LB" ; AND LOOK FOR LB:
96 BEQ    40$ ; ; IF EQ FOUND
97 MOV    D.LNK(R2),R2 ; ; NEXT DEVICE
98 BNE    30$ ; ; TRY IT!
99 CLR    4(SP) ; ; INDICATE ERROR BY SETTING USER R1 = 0
100 RETURN ; ; RETURN TO USER STATE
101 40$:
102 MOV    D.UCB(R2),R0 ; ; GET UCB ADDRESS
103 MOV    U.RED(R0),R0 ; ; FIND PHYSICAL LB:(I.E. FIRST REDIRECT)
104 MOV    U.DCB(R0),R2 ; ; FIND DCB OF PHYSICAL DEVICE
105 MOV    D.NAM(R2),4(SP) ; ; PUT LB DEVICE INTO USER STATE R1
106 SUB    D.UCB(R2),R0 ; ; CALCULATE UNIT NO.
107 MOV    D.UCBL(R2),R1 ; ;
108 CALL    $DIV ; ;
109 ADD    D.UNIT(R2),R0 ; ;
110 BIC    #177400,R0 ; ; CLEAR UNWANTED BITS
111 MOV    R0,6(SP) ; ; PUT UNIT NO. INTO USER STATE R2
113  RETURN     ;; RETURN TO TASK STATE
114  50$:     ; 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$:     
120  CLR  N.FID+F.PNB+$ACTPL     ; ASSUME NOT OPEN BY FILE ID
121  ALUNS$  #LUN2,R1,R2     ; ASSIGN LUN TO DEVICE.
122  MOV  $TKPS,MKT+M.HTME     ; USE TICKS/SEC TO MARK TIME FOR 1SEC.
123  70$:     
124  CALL  OPEN     ; OPEN ACCOUNT FILE
125  BCC  100$     ; IF CC - OPEN SUCCESSFUL.
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  #10.WVB,#5,#1,,,<R1,#80,,#40>
168  ; CLR  RO     ; SET UNSUCCESSFUL
169    JMP  RET       ; RETURN TO CALLER
170    ;
171    ; *** - SEARCH - SEARCH FILE FOR ACCOUNT NUMBER
172    ;
173    ; OUTPUT:
174    ; RO - 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  NAMPLC       ; IS NAME SPECIFIED?
187    BEQ  15$       ; NO
188    MOV  RO,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$: CMPB (RO)+,(R1)+       ; NAMES THE SAME?
195    BEQ  14$       ; 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  RO,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  #$BPLEN/512,.Q.IOPL+10(R4); NO, POINT TO NEXT VBN
224    ADC  Q.IOPL+6(R4)       ;
BR 5$ ; READ IN NEXT BUFFER
SEC ; ERROR, ACCOUNT NOT FOUND
RETURN

; *** - TPSWD - TEST PASSWORD
; CARRY SET - INVALID PASSWORD
; CARRY CLEAR - GOOD PASSWORD
; NOTE: THIS CODE ALLOWS PSW/TIME. IF THERE IS A/, IT DISREGARDS
; WHAT FOLLOWS BECAUSE, BATCH (ON M+ ONLY) SENDS TIME LIMIT TO BE
; DISREGARDED BY HELLO
;
TPSWD: MOV PSWDBF, R1 ; LOCATION OF PASSWORD FIELD
MOV 4(R1), -(SP) ; PUT PASSWORD ON STACK
MOV 5(R1), l(SP)
MOV 2(R1), -(SP)
MOV 3(R1), l(SP)
MOV 0(R1), -(SP)
MOV 1(R1), l(SP)
MOV SP, R1 ; POINT TO PASSWORD
MOV R0, -(SP) ; SAVE R0
MOV #6, R0 ; LENGTH OF PASSWORD FIELD

101$: CMPB (R1), #40 ; VALID CHAR?
BLO 105$ ; LO-NO.
CMPB (R1), #'/
BEQ 105$ ; IS IT SLASH (TIME-LIMIT COMING)?
CMPB (R1), #140 ; EQ- YES,,TREAT AS END-OF-PASSWORD
BLOS 102$ ; LOWER CASE?
CMPB (R1), #172 ; NO
BHI 102$ ; MAYBE
BICB #40, (R1) ; CONVERT TO UPPER CASE

102$: INC R1 ; LOOK AT NEXT BYTE
DEC R0 ; DECRM CHAR COUNT
BGT 101$ ; GT- MORE TO DO.
BR 108$ ; NO NEED TO SPACE FILL.

105$: DEC R0 ; ANY MORE TO FILL?
BMI 108$ ; MI- NO.
MOVB #40, (R1)+ ; SPACE-IT-OUT!
BR 105$ ; TRY AGAIN.
TST #ENCRIPT ; PASSWORD ENCRYPTION SUBR PRESENT?
BEQ 109$ ; EQ- NO.
MOV SP, R0 ; SHOW WHERE PASSWORD IS
ADD #2, R0
CALL ENCRYPT ; ENCRYPT THE PASSWORD
MOV (SP)+, R0 ; RESTORE R0
ADD #A.PSWD, R0 ; POINT TO PASSWORD IN FILE
MOV SP, R1 ; POINT TO (FILLED) ENTERED PASSWORD
MOV #6., R2 ; SET SIZE OF PASSWORD
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$SMPL
304
305    OPNS$U #$ACTFL,,,#FD.RWM ; OPEN FILE
306
307    .IFF ;R$SMPL
308
309    OPEN$U #$ACTFL,,,#FD.RWM ; OPEN FILE
310
311    .ENDC ;R$SMPL
312
313    BCS 10$ ; IF CC ERROR
314    INC FILOPN ; SET FILE IS OPEN
315 10$:  RETURN
316
317
318 ;
319 ; *** - CLOSE - CLOSE FILE
320 ;
321 CLOSE:
322    TST FILOPN ; IS FILE OPEN?
323    BEQ 10$ ; NO
324    CLR FILOPN ; FILE IS NOW CLOSING
325    CLOSE$ #$ACTFL ; YES - CLOSE FILE
326 10$:  RETURN
327
328 ;
329 ; *** - QIO - ISSUE QIO
330 ;
331 ; INPUT:
332 ; R4 - DPB ADDRESS
333 ;
334 QIO:
335    DIR$ R4 ; ISSUE QIO
336    BCS 10$ ; ERROR
337       MOVB     Q.IOEP(R4),R5          ; GET EVENT FLAG TO WAIT ON
338       WTSE$$    R5               ; AND WAIT
339 10$:    RETURN
340
341
342 .psect  c$text,i,ro
343 .even
344 .psect  c$data,d,rw
345 .even
346 .END
/* "@(#)compat.h  1.9 4/15/85" */
/* added by billn */
/* #include <exos/misc.h> */
#define index /* system 3 or 5 */
#include <fcntl.h>
#define dup2(f,n) { close(n); fcntl(f, F_DUPFD, n);} 
#endif
#endif void
#define void int
#endif
#define VOID (void)
#endif SIGCHLD
#define SIGCHLD SIGCLD
#endif /* end billn */
#endif MAXPATHLEN
#define MAXPATHLEN 33
#endif
#define receive_data rec_data
#define wait3 wait2
#define initgroups(a,b)
#define inappropriate_request inapreq
#endif BSD4dot2
#else
#endif V7
#include <sys/timeb.h>
struct timeval { long tv_sec; long tv_usec; };
struct timeb ftimeb;
#define gettimeofday(a,b) ( ftime (&ftimeb), 
(a)->tv_sec = ftimeb.time, (a)->tv_usec = ftimeb.millitm)
#else
struct timeval { long tv_sec; long tv_usec; };
extern long xtime();
#define gettimeofday(a,b) ((a)->tv_sec = time(0), (a)->tv_usec = 0)
#endif V7
#endif BSD4dot2
#endif CTRL
#define CTRL(x) 037&'x'
#endif
#define SOL_SOCKET 0
#define SO_REUSEADDR 0
/ * filename: LIBSOCK.H 
  * 
  * this file contains all the system dependent definitions 
  * used in the socket library 
  */ 

extern char *strstr(), *strstrchr(); 

#define HOSTS "LB:[1,1]HOSTS.NET" 
#define HOSTSLOCAL "LB:[1,1]HOSTLOCAL.NET"
#define connected conned
#define connecthelp connhelp
#define mdeletehelp mdelhelp
#define receivehelp recehel
#define verbosehelp verbhelp
/*
 * filename: ACCEPT.C
 */

#include <stdio.h>
#include <errno.h>
#include "libhdr.c"

xACcept(s, from)
    int s;
    struct sockaddr *from;
    {
        register XFILE *file;
        struct SOictl SOictl;
        struct iosb iosb;
        int ret;

        if( s < 0 || s >= _XNFILE )
            return( XEBADF );
        file = & xiob[s];
        if( !(file->flag & _XUsed))
            return( XEBADF );
        SOictl.hassa = from ? 1 : 0;
        ret = libemt(IO_ACS|SA_ACC, &iosb, 0, 0, &SOictl, 0, 0, (int) file->_sys_id);
        libcopy(&SOictl.sa,from,sizeof(struct sockaddr));
        return(ret);
    }

/* Objective of this function is to process different type of error resulting
 * from a call to the driver via QIO ( or emt call in 'C' ) call. A QIO
 * executive directive call reports error in two different ways through the
 * DSW ( directive status word ) and also in the IO statusblock. Again in the
 * IOSB it is divided into two parts one device specific and the other generic.
 * The generic and the dsw are returned to the caller after shifting it by -512
 * and the device specific code is just sign changed. If all is fine then an
 * non zero value is returned.
 */
/*
 *    FILENAME       ALLOC.C
 */

#include <rsxos.h>
#include <stdio.h>
typedef int ALIGN; /* forces alignment on PDP-11 */

union header { /* free block header */
    struct {
        union header *ptr; /* next free block */
        unsigned size; /* size of this free block */
    } s;
    ALIGN x; /* force alignment of blocks */
};

typedef union header HEADER;

static HEADER _base = {0}; /* empty list to get started */
static HEADER *allocp = XNULL; /* last allocated block */
char *xmalloc(nbytes) /* general-purpose storage allocator */
unsigned nbytes;
{
    static HEADER *morecore();
    register HEADER *p, *q;
    register int nunits;

    nunits = 1+(nbytes+sizeof(HEADER)-1)/sizeof(HEADER);
    if( (q = allocp) == XNULL) { /* no free list yet */
        _base.s.ptr = allocp = q = &_base;
        _base.s.size = 0;
    }
    for( p=q->s.ptr; ; q=p, p=p->s.ptr ) {
        if( p->s.size >= nunits) { /* big enough */
            if( p->s.size == nunits) /* exactly */
                q->s.ptr = p->s.ptr;
            else { /* allocate tail end */
                p->s.size -= nunits;
                p += p->s.size;
                p->s.size = nunits;
            }
        }
        allocp = q;
        return ((char *)(p+1));
    }
    if( p == allocp ) /* wrapped around free list */
        if(( p = morecore(nunits)) == XNULL)
            return(XNULL); /* none left */
}
#define NALLOC 16 /* #units to allocate for memory */
HEADER *morecore(nu) /* ask system for memory */
unsigned nu;
{
    register char *cp;
    register HEADER *up;
    register int rnu;

    rnu = NALLOC * ((nu+NALLOC-1) / NALLOC);
    cp = sbreak(rnu * sizeof(HEADER));
    if( (int)cp == -1) /* no space at all */
        return (XNULL);
    up = (HEADER *)cp;
    up->s.size = rnu;
    xfree((char *)(up+1));
    return(allocp);
}

xfree(ap) /* put block ap in free list */
char *ap;
{
    register HEADER *p, *q;

    p = (HEADER *)ap -1; /* point to the header */
    for( q=allocp; (!p > q && p < q->s.ptr); q=q->s.ptr )
        if( q >= q->s.ptr && (p > q || p < q->s.ptr) )
            break; /* at one end or other */
    if( p+p->s.size == q->s.ptr ) { /* join to upper nbr */
        p->s.size += q->s.ptr->s.size;
        p->s.ptr = q->s.ptr->s.ptr;
    } else
        p->s.ptr = q->s.ptr;
    if( q+q->s.size == p ) { /* join to lower nbr */
        q->s.size += p->s.size;
        q->s.ptr = p->s.ptr;
    } else
        q->s.ptr = p;
    allocp = q;
}

#define EXTK 01531
#define BLK 64

extern int _brk;
sbreak(nbytes)
register int nbytes;
{
    register int ret = _brk;

    if( emt(EXTK, 1+(nbytes-1)/BLK, 0) >= 0 ) {
        _brk += nbytes;
        return ret;
    }
    else {
        /*
        xprintf(" Task extention failed \n", rval);
        */
    }
}
113  */
114  return -1;
115  }  /* No memory  */
116  }
/*
 * filename: BOARD.C
 */

#define u_long long
#include <stdio.h>
#include <xspecial.h>
#include <xerrno.h>
#include <libhdr.c>
#include <brdioctl.h>
#include <init.h>
#include <route.h>

int brdopen( brd_no, mode) /* open an administrative channel */
int brd_no;
int mode;
{
    int ret;
    struct iosb iosb;

    if ( mode == 1 ) /* mode is readonly */
        mode = 0;
    else /* else mode is read write */
        mode = 1;
    ret = libemt(IO_EXC|EX_OPN, &iosb, 0, 0, 0, mode, 0, 0);
    if ( ret == 0 )
        ret = iosb.nread; /* return channel # */
    return ( ret );
}

int xbrdclose( fd ) /* close an administrative channel */
int fd;
{
    int ret;
    struct iosb iosb;
    ret = libemt(IO_EXC|EX_CLS,&iosb,0,0,0,0,0,fd);
    return ( ret );
}

int xbrdwrite( sys_id, buf, len)
int sys_id; /* must have been char *sys_id */
char *buf;
int len;
{
    int fd, ret;
    struct iosb iosb;
    register XFILE *file;
    ret = libemt(IO_WLB,&iosb,buf,len,0,0,0,sys_id);
    if ( ret == 0 )
        ret = iosb.nread;
    return ( ret );
int xbrdread( sys_id, buf, len ) /* read boards memory */
int sys_id;
char *buf;
int len;
{
    int fd, ret;
    struct iosb iosb;
    register XFILER *file;
    ret = libemt(IO_RLB,&iosb,buf,len,0,0,0,sys_id);
    if ( ret == 0 )
        ret = iosb.nread;
    return ( ret );
}

int xbrdioct1( sys_id, cmd, arg )
int sys_id, cmd;
char *arg;
{
    int i, fd, len = 0, ret;
    long along = 0;
    Ushort base = 0, off = 0;
    char *buf = 0;
    int qio_fn ;
    struct iosb iosb;
    register XFILER *file;

    switch ( cmd ){
    case BRDINIT:
        /* translate the mode */
        base = *( int * ) arg; /* mode of configuration */
        switch ( base ){
        case 0: base = 1; /* host down load */
        break;
        case 1: base = 2; /* net down load */
        break;
        case 2: base = 0; /* link level mode */
        break;
        case 0x80: /* infinite timeout */
        base |= 1; /* include with download mode */
        break;
        default:
        base = 1; /* forced to download mode */
        }
    qio_fn = IO_EXC|EX_INIT;
    break;
    case BRDADDR:
    case BRDSTART:
        along = *( long * ) arg;
        base = (Ushort)(( ( along >> 16 ) & 0x0000ffff )
        off = (Ushort)(( along & 0x0000ffff ));
        if ( cmd == BRDADDR )
            qio_fn = IO_EXC|EX_POS;
else
    qio_fn = IO_EXC|EX_STR;
break;

case BRDGSTAT:
case BRDRSSTAT:
    buf = (char *) arg;
    len = sizeof ( struct EXbdstats );
    if ( cmd == BRDGSTAT )
        qio_fn = IO_EXC|EX_STS;
    else
        qio_fn = IO_EXC|EX_RST;
    break;

case BRDCONF:
    buf = (char *) arg;
    len = sizeof ( struct init_msg);
    qio_fn = IO_EXC|EX_CNF;
break;

case BRDSARP:
case BRDGARP:
case BRDDARP:
    buf = (char *) arg;
    len = sizeof( struct EXarp_ioctl);
    if ( cmd == BRDSARP )
        qio_fn = IO_EXC|EX_SAR;
    else if ( cmd == BRDGARP )
        qio_fn = IO_EXC|EX_GAR;
    else
        qio_fn = IO_EXC|EX_DAR;
    break;

case BRDADDR:
case BRDDELRT:
case BRDSHOWRT:
case BRDDISPRRT:
    buf = (char *) arg;
    len = sizeof ( struct rtentry );
    if ( cmd == BRDADDR )
        qio_fn = IO_EXC|EX_ART;
    else if ( cmd == BRDDELRT )
        qio_fn = IO_EXC|EX_DRT;
    else if ( cmd == BRDSHOWRT )
        qio_fn = IO_EXC|EX_SRT;
else qio_fn = IO_EXC|EX_NRT;
break;

default:
    break;
}
return ( libemt(qio_fn, &iosb, buf, len, 0, base, off, sys_id ));
xbrdopen(brdno, mode)

int brdno; /* ignore for now */
int mode;
{
    int retval;
    int exosfd;
    int ioflag;
    int uflag;
    register XFILE *file;

    uflag = xtranmode(mode, &ioflag);
    if (uflag < 0)
        return(uflag);
    retval = brdopen(1, mode);
    if(retval < 0)
        return(retval);
    exosfd = xnewod(); /* get a free file descriptor */
    if(exosfd < 0)
    {
        xbrdclose(retval);
        return(exosfd);
    }
    file = &xiob[exosfd];
    file->fflag |= ioflag;
    file->sys_id = (char*)retval;
    file->read = xbrdread;
    file->ioctl = xbrdioctl;
    file->write = xbrdwrite;
    file->close = xbrdclose;
    return(exosfd);
}
static char sccsId[] = "@(#)bzero.c 1.4 3/26/85";

/*
code to make 4.2 style code, sort of, happy.
*/
bzero( pt, len )
/*
clear a block
*/
char *pt;
int len;
{
for( ; len > 0 ; --len )
{
    *pt++ = 0;
}
}
/* filename: CATCHOOG.C */

#include <xgenlib.h>
define MAXCHN 40#include "libhdr.c"

struct _astst _stast[MAXCHN] = { 0 Note: This is the ast service routine written in macro */

int xcatchoog( s, handler)
    int s;
    int (*handler());
{
    register struct iosb *iosb;
    int ch_no;

    if ( iosb = giosb()){
        ch_no = (int ) xio[s].sys_id; /* get channel number */
        if (_stast[ch_no].stast == FREE){
            _stast[ch_no].stast = USED;
            _stast[ch_no].xio[no] = s; /* store xio number */
            _stast[s].userast = handler;
            emt(QIO,IO_ACS|SA_URG,SOLUN,0,iost, _astcatch,0,0,0,0,0,0, ch_no);
        }
    else return (-1);
}
else return (NOSOIOSB);
}

libas( iosb)
struct iosb *iosb;
{
    Ushort ch_no;
    Ushort s;

    if( iosb ) /* if a iosb was specified-- which is in this case */{
        ch_no = iosb->nread; /* this is set in the ACP */
        fiosb(iosb);
        _stast[ch_no].stast = FREE; /* mark it free for use */
        s = _stast[ch_no].xio[no]; /* get file no. */
        if (_stast[ch_no].userast)
            (*_stast[ch_no].userast)(s);
    }
}

struct iosb *
giosb()
{
    return(xmalloc( sizeof (struct iosb )));
}
struct iosb *iosb;
{
    xfree(iosb);
}
#include <xgenlib.h>
#include <fcs.h>

char *inprm[MAXPRM] = {0};        /* array of pointers to input string */

extern char _xctype[];
extern long radix();

char *pcli;                         /* pointer to command line */

int count = 0;
char *p = pcli;
int i = 3;

while(*p) { *p = tolower(*p); ++p; }
while(pcli && *pcli) {
    switch(*pcli) {
    case '<':
        inprm[0] = pcli + 1;
        break;
    case '>':
        inprm[1] = pcli + 1;
        break;
    case '~':
        inprm[2] = pcli + 1;
        break;
    default:
        inprm[i++] = pcli;
        count++;
    }
    pcli = firstwhite(pcli, ' ');
    *pcli++ = 0;        /* make argument as string */
    pcli = skipwhite(pcli, ' ');
}
return main(count, &inprm[3]);
1 /*
2 * filename: CONNECT.C
3 */
4
5 #include <stdio.h>
6 #include <xerrno.h>
7 #include "libhdr.c"
8
9 xconnect(s, addr)
10 int s;
11 struct sockaddr *addr;
12 {
13     register XFILE *file;
14     struct SOictl SOictl;
15     struct iosb iosb;
16
17     if( s < 0 || s >= XNFILE )
18         return( XEBADF );
19     file = & xio[b][s];
20     if( !(file->flag & XUsed ) )
21         return( XEBADF );
22     if ( addr){
23         SOictl . hassa = 1;
24         libcopy(addr,&SOictl.sa,sizeof (struct sockaddr));
25     }
26     else SOictl.hassa = 0;
27     return(libemt(IO_ACS|SA_CON, &iosb,
28                     0, 0, &SOictl, 0, 0, (int) file->sys_id));
29 }
```c
#include <rsxos.h>
#include <xstdio.h>
#include <fcs.h>

extern struct _rcb _rcb[];
struct dbbuf hbuf={0}, nbuf={0};
 /*
 extern int disk_efn;
 */
 extern char luntbl[];
 extern struct dbbuf hbuf,nbuf;

#define CNTRLZ 0366

dio(sysid, call, ast, wait)
    register struct _rcb *sysid;
    int    (*call)();
    int    (*ast)();
    int    (*wait)();
{
    static int iosb[2] = {0};
    int    rval;
    int    ret;

    if ( sysid->flags & DBLBUF ) {
        /*
         disk_efn += d_efn();
       */
        emt(WTSE,DISKEFN);    /* stop for any pending i/o */
        /* efn is set at ast    */
        hbuf.stat[hbuf.active] = 0;
        hbuf.active = !hbuf.active;
        rval = hbuf.stat[hbuf.active];
        if ( rval > 0 ) {
            emt(CLEF,DISKEFN);
            ret = ( *call)(sysid->fdb,sysid->bptr,0,iosb,ast);
            if(ret <= 0) {
                hbuf.stat[!hbuf.active] = ret;
                emt(SETF,DISKEFN);
            }
        }
        sysid->bptr = hbuf.buffer[!hbuf.active];
    }
    else {
        rval = ( *call)(sysid->fdb,sysid->bptr,DISKEFN,iosb,0);
        if(rval > 0) {
            (* wait)(sysid->fdb,iosb);
            rval = iosb[1];
        }
        sysid->bnptr = sysid->bptr;
    }
    return rval;
}

static char mask[8] = {1, 2, 4, 8, 16, 32, 64, 128};
#define BYTE 8
```

#define MAXLUN 255

assign(lun)
int lun;
{
    *(luntbl + lun/BYTE) |= mask[ lun % BYTE];
}

dassign(lun)
int lun;
{
    *(luntbl + lun/BYTE) &= ~mask[ lun % BYTE];
}

glun()
{
    register int bit = 0;
    int i;
    for (i = 1; i <= MAXLUN; ++i) {
        if( !(*(luntbl + i / BYTE) & mask[ i % BYTE ])) {
            *( luntbl + i/BYTE ) |= mask[ i % BYTE ];
            return i;
        }
    }
    return -1;
}

nstat(iosb)
register struct iosb *iosb;
{
    register int *p;
    p = &nbuf.stat[nbuf.active];
    if((iosb->cc >= (unsigned char)0) && (iosb->lc == (unsigned char)0))
        *p = iosb->nread;
    else if(iosb->cc < (unsigned char)0)
        *p = iosb->cc - 512;
    else
        *p = ( -(iosb->lc & 0xFF));
    enet(SETF, SOEFN);    /* socket i/o is completed */
}

dstat(iosb)
register struct iosb *iosb;
{
    register int *p;
    p = &hbuf.stat[!hbuf.active];
    if( iosb->cc == CNTRLZ )
        *p = 0;
}
else if(iosb->cc > 0)
    *p = iosb->nread;
else
    *p = iosb->cc - 512;
emt(SETF, DISKEF); /* disk i/o is completed */
}
getclient( type, pf, sin, options, typical_serv )

int type;
struct sockproto *pf;
/*
struct sockaddr *sin;
*/
struct sckaddr_in *sin;
int options;
int (*typical_serv)();
{
int s;
int errno;
int status;
struct sockaddr from;

start:
s = xsocket( type, pf, sin, options );
if ( s < 0 )
{
xerror( s, "getclient socket" );
xsleep( 5 );
goto start;
}

/*
wait for service request
*/
if ( ( errno = xaccept( s, &from ) ) < 0 )
{
xerror( errno, "getclient accept" );
xclose( s );
xsleep( 5 );
goto start;
}

/*
RSX specific process management
*/
xspawn();
(*typical_serv)( s, &from );
#include <rsxos.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <special.h>
#include <libsock.h>

extern char *strstrchr(), *strrchr();
extern char *firstwhite();
extern char *skipwhite();
extern char *lastwhite();

char *
xgname(name, nchars)
char *name;
int nchars;
{
    int od;
    FILE *op;
    char hbuf[XBUFSIZE], *cp, *ahost;
    int rc;

    od = xodopen(HOSTS, XFREAD | XFASCII, FILE_NAME);
    if ( ( od < 0 ) || !( op = xodopen(od, "r") ) ){
        x perror(XEBADF, "getname!");
        rc = 1;
        goto egress;
    }

    while (XNULL != xogets(hbuf, sizeof(hbuf), op)) {
        *strstrchr(hbuf, '\n') = 0;
        if (hbuf[0] == '\#' )
            continue;
        for (;;) {
            cp = lastwhite(hbuf, ' ');
            if (cp == XNULL)
                break;
            if (!strstrcmp(cp+1, "localhost")) {
                ahost = firstwhite(hbuf, '\') + 1;
                ahost = skipwhite(ahost);
                cp = firstwhite(ahost, '\');
                if (cp)
                    *cp = 0;
                if (xstrlen(ahost) + 1 > nchars) {
                    rc = 1;
                    goto egress;
                }
                xstrcpy(name, ahost);
                rc = 0;
                goto egress;
            }
            *cp = 0;
        }
    }
    rc = 1;
egress:
xclose(od);
return (rc);
/*
 * filename: HTONS.C
 *

unsigned short
xhtons(x)
{
    unsigned short x;
    return(((unsigned short)((x<<8)|(x>>8)&0xff)));
}

long
xhtonl(x)
{
    long x;
    union {
        long l;
        struct {
            unsigned short s_high, s_low;
        } sl;
    } h;
    h.l = x;
    h.sl.s_high = xhtons(h.sl.s_high);
    h.sl.s_low = xhtons(h.sl.s_low);
    return(h.l);
}

unsigned short
xntohs(x)
{
    unsigned short x;
    return(xhtons(x));
}

long
xntohl(x)
{
    long x;
    return(xhtons(x));
}
#define PDPI

#define USED 1
#define FREE 0

struct __asts{
    short stast;
    short xiobno;
    int (*userast)();
};

struct seg_addr {
    Ushort base; /* segment base address */
    Ushort off;  /* segment offset */
};
/*
 * filename: LIBRTS.C
 */

#include <std.h>
#include <rsx.h>
#include <extypes.h>
#include <solibdef.h>

unsigned short ex_libinit = 0;
unsigned short unibus = 0; /* if on a UNIBUS m/c */

/* below is a definition of a structure for handling user specified
   AST function calls in the catchoob() library function call */

struct __asts{
  short stast;
  int (*userast)();
};

struct seg_addr
{
  Ushort base;     /* segment base address */
  Ushort off;      /* segment offset     */
};

int libinit()
{
  ex_libinit = 1;
}

int libcopy(from,to,size)
Uchar *from, *to;
int size;
{
  while (size--)
    *to++ = *from++;
}

/*
 * Objective of this function is to process different type of error resulting
 * from a call to the driver via QIO ( or emt call in 'C' ) call. A QIO
 * executive directive call reports error in two different ways through the
 * DSW ( directive status word ) and also in the IO statusblock. Again in the
 * IOSB it is divided into two parts one device specific and the other generic.
 * The generic and the dsw are returned to the caller after shifting it by -512
 * and the device specific code is just sign changed. If all is fine then an
 * non zero value is returned.
 */

libemt(cmd,iosb,p1,p2,p3,p4,p5,p6)
Ushort cmd;
struct iosb *iosb;
Ushort p1, p2, p3, p4, p5, p6;
{  
  int j = 0,dsw;
  register int cnt,i;
  register int count = 1024; /* 1 KB */

  if(p2 <= 0){
    cnt = 1;
    count = 0;
  }
  else
    cnt = p2;

  for(i = 0; cnt > 0; i++) {
    if((cnt < count) || (!unibus))
      count = cnt;
    dsw =
      emt(QIOW, cmd, SOLUN, SOEFN, iosb, 0, (p1 + j),count, p3, p4, p5, p6);
    if((dsw >= 0) && (iosb->cc >= 0) && (iosb->lc == 0)) {
      if(p2 <= 0)
        return 0;
      j += iosb->nread;
      continue; /* continue on success */
    }
  }
  else
    if(dsw < 0)
      return(dsw - 512); /* directive error */
  else
    if(iosb->cc < 0)
      return(iosb->cc - 512); /* generic I/O error */
  else
    return(-(iosb->lc & 0xff)); /* device specific error */

  iosb->nread = j; /* total # of bytes transacted */
  return 0; /* return success */
}
/*
System entry point for client programs running under RSX.
Note: terminal => unbuffered io.
*/
#include <xgenlib.h>
#include <xspecial.h>
#include <xpwd.h>
#include <fcs.h>

#define SY 054523

extern xttysread();
extern xttyswrite();
extern xttysclose();
extern xnofunc();
extern xdread();
extern xfwrite();
extern xdclose();

struct _xiobuf _xiob[ XNFILE ] = {0};
struct passwd xpassword = {0};
struct passwd *pw = &xpassword;
struct ttybuf ttybuf = {0};
int _ttyinput = 0;       /* 0 -- interactive . 1 -- non-interactive */
struct _rcb _rcb[ XNFILE ] = {0};
char _luntbl[32] = {0};  /* array of 256 bits used to maintain LUN */
int _brk = 0;            /* USED by C_RTS ALLOC & FREE */
extern char _xctype[];
extern char *_inprm[];

main( argc, argv )
{int argc;
 char **argv;
{int i;
 register XFILe *file;
 char *p;
 int rval;
 int ioflag;
 int mod;
 int buf[16];
 int maxlun;

/* initialize _xiob structure */
for(p=(char *)_xiob; p < ((char *)_xiob + sizeof _xiob); )
  *p++ = '\0';
/* initialize _rcb structure */
for( i=0; i < XNFILE ; ++i )
  _rcb[i].flags = RFREE;
/*
 * initialize terminal I/O buffer
 */
57  ttybuf.cur_pos = ttybuf.linetty;
58  ttybuf.tsize = 0;
59
60  for(i = 1; i < 5; i++)
61    emt(ALUN, i, SY, 0);
62    emt(GTSK,buf);
63    _brk = buf[13];  /* task size */
64    maxlun = buf[8];  /* # of LUN used */
65    ppass(pw->cur_uic, buf[7]);
66    ppass(pw->login_uic,buf[15]);
67    emt(GLUN,1,buf);
68    /* get phy. device name */
69    xstrncpy(pw->log_dev,buf,2);
70    /* copy device name */
71    pw->log_dev[2] = (*((char *) buf + 2)) + 060;  /* get unit # */
72    /* make it string */
73    xstrcpy(pw->cur_dev,pw->log_dev);
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      xttysopen(XFREAD|XWRITE);
83    }
84    else {
85      if(i == 0)
86        mod = XFASCII | XFREAD;
87      else
88        mod = XFASCII | XPCREAT | XWRITE;
89      xdopen(inprm[i], mod, FILE_NAME);
90    }
91    if( i == 0 )
92      xddopen( i, "r" );
93    else {
94      file->flag |= _XIOLBF;
95      file->cnt = 0;
96      xddopen( i, "w" );
97    }
98  }
99  xputchar('\n');
100  clientinit();
101  xmain(argc, argv);
102  xexit(0);
103  }
104
105  /*
106   * ISATTY:  check object descriptor directs to terminal or not.
107   * if it is terminal returns 1 else 0.
108   *
109   */
110
111  */
isatty(od)
int od;
{
    if( !inprm[od] )
    {
        if( od == 0)
            ttyinput = 1;
        return(1);
    }
    else
        return(0);
}
/*
* @(#)xmkarglist.c 1.3 3/29/85
*
* include <rsxos.h>
*
* define ARGPOINTERSP 200 /* bytes for storing argument pointers */
* define ARGSPACE 400 /* bytes for storing arguments */
*
* static char *argbase = {0};
* static char *stringbase = {0};
*
* char **
* xmkarglist( line, count )
*
* char *line; /* IN */
* int *count; /* OUT */
*
* char **argp;
* char *slurpstring();
* char *argvsp;
* int margc;
*
* margc = 0;
* /*
* Allocate space for argv and tokens in line
* */
* if( xstrlen( line ) > ARGSPACE )
* {
* return( (char **)0 );
* }
* argvsp = xmalloc( ARGPOINTERSP + ARGSPACE );
* if( argvsp == (char *)0 )
* {
* return( (char **)0 );
* }
* argbase = &argvsp[ARGPOINTERSP]; /* store from first of buffer */
* stringbase = line; /* scan from first of buffer */
* argvp = (char **)argvsp;
* while (*argvp++ = slurpstring())
* margc++;
* *count = margc;
* return( (char **)argvsp );
* }
*
* /*
* Parse string into argbuf;
* * implemented with FSM to
* * handle quoting and strings
* */
* char *
* slurpstring()
* {
* int got_one = 0;
* register char *sb = stringbase;
register char *ap = argbase;
char *tmp = argbase;  /* will return this if token found */

/*
Used to return '!' for shell event processing...
Ignore significance of '!'..
*/

S0:
switch (*sb) {
  case '\0':
    goto OUT;
    
    case ' ':  
    case '\t':
      sb++; goto S0;
  
    default:
      goto S1;
  }

S1:
switch (*sb) {
  case ' ':  
  case '\t':
  case '\0':
    goto OUT;  /* end of token */

    case '\\':
      sb++; goto S2; /* slurp next character */

    case '"':
      sb++; goto S3; /* slurp quoted string */

    default:
      *ap++ = *sb++; /* add character to token */
      got_one = 1;
      goto S1;

  }

S2:
switch (*sb) {
  case '\0':
    goto OUT;

  default:
    *ap++ = *sb++;  
    got_one = 1;
    goto S1;

  }

S3:
switch (*sb) {

case '\0':
    goto OUT;

case '"':
    sb++; goto S1;

default:
    *ap++ = *sb++;
    got_one = 1;
    goto S3;

OUT:
    if (got_one)
    *ap++ = '\0'; /* update storage pointer */
    argbase = ap;      /* update scan pointer */
    stringbase = sb;
    if (got_one)
        return(tmp);
    return((char *)0);
  
  xdealglob( pt )
  /*
  Free space allocated by either xglob or xmkarglist
  */

char **pt;
{ 
  xfree( (char *)pt );
}
# include <xgenlib.h>
/
/*
 * filename: MKCMD.C

 mkcmd creates a MCR command line. It takes a pointer to the commandline and multiple pointers to string.
 */

char *
mkcmd(line, str)
char *line;
char *str;
{
    char **argp = &str;

    *line = '\0'; /* clear command line */
    while( *argp ) /* till a null argument */
        xstrcat( line, *argp++ );

    return(0);
}
/* FILENAME: MKNAME.C */

/* This routine updates the name according to default dev & dir. */
/* it must be invoked after parse. It takes the input from CSI */
/* control block, which is created by parse routine. */

/* OUTPUT: */
/* If more file_spec it returns size of current file_spec */
/* else 0 -- no more in-spec */

#include <xpwd.h>
#include <xgenlib.h>

extern struct passwd *pw;
extern char *csiblk;

mkname(name)
char *name;
{
int rval;
char dev[6];
char uic[10];
char nam[15];
int filelen = 0;

dev[0] = '0';
uic[0] = '0';
nam[0] = '0';

xstrncpy(dev,pw->cur_dev); /* initialize default dev. */
xstrncpy(uic,pw->cur_uic);  /* initialize default dir. */
if(csiblk[C_STAT] & CS_DVF){
  xcbcopy(*(int *)(csiblk+C_DEVD+2),dev,*(int *)(csiblk + C_DEVD));
  dev[*(int *)(csiblk + C_DEVD ) ] = '\0';
  filelen += *(int *)(csiblk + C_DEVD ) + 1;
}
if(csiblk[C_STAT] & CS_DIF){
  xcbcopy(*(int *)(csiblk+C_DIRD+2),uic,*(int *)(csiblk + C_DIRD));
  uic[*(int *)(csiblk + C_DIRD ) ] = '\0';
  filelen += *(int *)(csiblk + C_DIRD );
}
if(csiblk[C_STAT] & CS_NMF){
  xcbcopy(*(int *)(csiblk+C_FILD+2),nam,*(int *)(csiblk + C_FILD));
  nam[*(int *)(csiblk + C_FILD ) ] = '\0';
  filelen += *(int *)(csiblk + C_FILD );
}
mkcmd(name,dev,":",uic,nam,0);
if(csiblk[C_STAT] & CS_MOR)
  rval = filelen;
else
  rval = 0;
return(rval);
}
/*
 * filename: MUXIO.C
 */

#include <rsxos.h>
#include <stdio.h>
#include <xspecial.h>
#include <solibdef.h>

#define IO XFR 003400
#define IX_RDS 0000
#define TTYEPFN 2
#define SLEEP_EFN 3
#define strip(x) ((x) & 0177)

short ready1 = 1; /* rod1 is readable */
short ready2 = 0; /* rod2 is initially not readable */
struct iosb {
    char cc;
    char lc;
    int nread;
};
struct iosb isbl = {0}; /* IO status block for netread */
struct iosb isb2 = {0}; /* IO status block for netwrite */
int _rod2 = 0;

static char _sibuf[XBUFSIZ] = {0};
static char _tibuf[XBUFSIZ] = {0};
extern int ttyraw; /* 1 == raw 0 == line_edit */
extern char escape;
static int scc = 0; /* byte count for net_read */
static int tcc = 0; /* byte count for tty_read */

char *_tbufp = _tibuf;
extern mynetread();
extern myttyread();
extern _xsioioctl();
extern _xtttyread();
extern astrd2();
extern int wrap;
extern int ttyln;

xmux_io( serv_id, io_procl, rod1, wod1, io_proc2, rod2, wod2 )

char *serv_id; /* service identifier, see getclient(3X) */
int (*io_procl)(); /* Network to terminal process */
int rod1; /* descriptor for first process to read */
int wod1; /* descriptor for first process to write */
int (*io_proc2)(); /* Terminal to network process */
int rod2; /* descriptor for second process to read */
int wod2; /* descriptor for second process to write */
{
    short last_read = 1; /* last descriptor read */
    short netrfin = 1; /* initialize - net read has finished */
int pid1;  /* dummy process id - not used */
int pid2;  /* dummy process id - not used */

_rod2 = rod2;
_xiob[rod1].read = mynetread;
_xiob[rod1].ioctl = _xsioioctl;

emt(CLEF,SLEEP_EFN);
emt(QIOW,IO ATA,ttylun,TTYEFN,0,0,astrd2,0,0,0,0,0);
for( ;; ) {

_xiob[rod2].read = myttyread;

if(ready1)
    if(netrfin){
        rdl(rod1);  /* do a net read */
        netrfin = 0;  /* netread is pending */
    }

    if((ready1 && ready2 && (last_read == 2)) ||
       (ready1 && !ready2 ))
    {
        (*io_proc1)(pid2,rod1,wod1);
        last_read = 1;
        netrfin = 1;  /* net read has finished */
    }
    else if(ready2)
    {
        emt(DSAR);
        (*io_proc2)(pid1,rod2,wod2);
        last_read = 2;
        ready2 = 0;
        _tbufp = _tibuf;
        tcc = 0;
        emt(ENAR);
    }
    else {
        emt(STSE, SLEEP_EFN);
        emt(CLEF, SLEEP_EFN);
    }
}

/*
 *    RD1
 */

rdl(rod1)
int rod1;
{
    ready1 = 0;  /* make rod1 non-readable */
    _xsoread(rod1,_sibuf,sizeof _sibuf);
}

/*
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 = _tobuf[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         xiof_[rod2]._read = xtttyread;  
142     if( (cnt=tcc) > 0)  
143         xbcopy(_tobuf,buf,tcc);  
144
145     /*
146         _tbufp = _tobuf;
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 )xiof[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
return status of the read issued to the network.

nrstat(iosb)
struct iosb *iosb;
{
    if((iosb->cc >= (unsigned char )0) && (iosb->lc == (unsigned char )0))
        scc = iosb->nread;
    else if(iosb->cc < (unsigned char )0)
        scc = iosb->cc - 512; /* generic I/O error */
    else
        scc = (~(iosb->lc & 0xFF)); /* device specific error*/

    ready1 = 1; /* rd1 is now ready to read */
    emt(ENCF); /* enable checkpointing */
    emt(SEF, SLEEP_EFN);
}

/*
XKILL -- waits for any outstanding I/O on the network
*/

xkill(pid)
int pid;
{
    char stadd[2];
    emt(QIOW, IO_DET, ttyln, TTYEFN, 0, 0, 0, 0, 0, 0, 0);
    emt(QIOW, IO_KIL, ttyln, TTYEFN, 0, 0, 0, 0, 0, 0, 0);
    if(wrap) /* previously in wrap mode so set it accordingly */
        stadd[0] = TC_ACR;
    stadd[1] = 1;
    emt(QIOW, SF_SMC, ttyln, TTYEFN, 0, 0, stadd, 2, 0, 0, 0, 0);

    xexit(0);
}

/*
xsoioctl -- kill any outstanding I/O on the network and
then call actual xsoioctl function.
*/

xsoioctl(net, cmd, arg)
int net;
int cmd;
int arg;
{
    emt(ENAR); /* enable ast recognition */
    /*
    emt(QIOW, IO_KIL, SOLUN, SOEFN, 0, 0, 0, 0, 0, 0, 0, 0);
    */
    xsoioctl(net, cmd, arg);
}

/*
TRSTAT -- get a character from ast stack and put it into the tibuf
*/
* /

testat(c)
char c;
{
    *_tbufp++ = c;
tcc++;
    ready2 = 1;
    emt(SETF, SLEEP_EFN);
}
/*
 @(#)passthru.c 1.3 3/29/85

 Xpasstnet(3X) and xpassfnet(3X) for Rsx.
 */

#include <rsxos.h>
#include <xstdio.h>
#include <xerrno.h>
#include <ftp.h>
#include <extypes.h>
#include <fcs.h>

#define HASHSIZE 1024

extern int type;
extern int hash;
extern struct _rcb _rcb[];
extern long _xpass();
extern int _dread();
extern int _fwrite();
extern int _sread();
extern int _fwrite();

#define CNTRLZ 0366

extern struct dbdbuf hbuf ;
extern struct dbdbuf nbuf ;
long
xpasstnet( inod, outod )
{
    long bytes;
    int rval;
    register struct _rcb *insys = ( struct _rcb * )inod->_sys_id;
    /* make od's as double buffer */
    hbuf.buffer[1] = inod->base;
    inod->base = outod->base;
    nbuf.buffer[0] = outod->base;
    if( ( rval = xmalloc(XBUFSIZE)) == XNULL ){
        /*
         xprintf(xstdout,"passtnet buffer pointer = %d\n",rval);
         */
        return (long) XENOMEM;
    }
    nbuf.buffer[1] = (char *) rval;
    outod->write = _fwrite;
    nbuf.stat[1] = 1; /* initialize write status */
    hbuf.active = 1;
    nbuf.active = 0;
    insys->bptr = insys->bnptr = hbuf.buffer[1];
    insys->bleft = 0;
    insys->flags |= DBLBUF;
    hbuf.fd = inod;
    nbuf.fd = outod;
    bytes = _xpass( inod, outod );
inod->base = hbuf.buffer[!hbuf.active];
outod->base = nbuf.buffer[!nbuf.active];
insys->flags &= ~DBLBUF;
emt(WTSE,SOEFN);
/*
xprintf(" No. of socket i/o wait %d\n",socket_efn);
xprintf(" No. of disk i/o wait %d\n",disk_efn);
*/
xfree(nbuf.buffer[nbuf.active]);
return bytes;
}

long
xpassfsnet( inod, outod )
{
    register FILE *inode;  /* input EXOS io object */
    register FILE *outode;  /* output EXOS io object */
    {
        long  bytes;
        int   rval;
        register struct _rcb *outsys = ( struct _rcb * ) outod->sys_id;
        /* make od's as double buffer */
        hbuf.buffer[1] = outod->base;
        outod->base = inod->base;
        nbuf.buffer[0] = inod->base;
        if( (rval = xmalloc(XBUFSIZE)) == XNULL ) {
            /*
            xprintf(xstdout,"passnet buffer pointer = %d\n",rval);
            */
            return (long) XENOMEM; /* No memory */
        }
        nbuf.buffer[1] = (char *) rval;
        inod->read = _sread;
        nbuf.stat[0] = xsoread(inod->sys_id,inod->base, XBUFSIZE);
        hbuf.stat[1] = 1;  /* initialize write status */
        inod->base = nbuf.buffer[1];
        nbuf.active = 1;
        hbuf.active = 0;
        outsys->flags |= DBLBUF;
        hbuf.fd = outod;
        nbuf.fd = inod;
        bytes = xpass( inod, outod );
        outod->base = hbuf.buffer[!hbuf.active];
inod->base = nbuf.buffer[!nbuf.active];
        outsys->flags &= ~DBLBUF;
        emt(WTSE,DISKEFN);
        /*
        xprintf(" No. of socket i/o wait %d\n",socket_efn);
        xprintf(" No. of disk i/o wait %d\n",disk_efn);
        */
        xfree(nbuf.buffer[nbuf.active]);
        return bytes;
    }
}
long _xpass( inod, outod )
register XFILE *inod; /* input EXOS io object */
register XFILE *outod; /* output EXOS io object */
{
    int c;
    int d = 0;
    long bytes = (long)0;
    long hashbytes = XBUFSIZ;
    
    emt(SETF, SOEFN); /* No pending i/o on socket */
    emt(SETF, DISKEFN); /* No pending i/o on disk */
    while ((c = xread(xfileno(inod), inod->_base, XBUFSIZ)) > 0) {
        if ((d = xwrite(xfileno(outod), outod->_base, c)) < 0)
            break;
        bytes += c;
        if (hash) {
            xputchar('#');
            xfflush(xstdout);
        }
    }
    if (hash) {
        xputchar('n');
        xfflush(xstdout);
    }
    if (c < 0) {
        xerror( c, "on input");
        return( (long)c );
    }
    if (d < 0) {
        /*
         * Throw any data remaining in pipe
         */
        while ((c=xread(xfileno(inod), inod->_base, XBUFSIZ)) > 0)
            ;
        xerror( d, "on output");
        return( (long)d );
    }
    return bytes;
}

dread(sysid,buf,size)
_register struct _rcb *sysid;
char *buf;
int size;
{
    if( size < 0 )
        return -1; /* error */
    if( sysid->flags & REOF )
        return 0; /* eof */
    if( type == TYPE_A )
        return (getnet(sysid,buf,size));
else
return ( bread(sysid,buf,size) );

getnet(sysid,buf,size)
register struct _rcb *sysid;
char *buf;
register int size;
{
    register int count = 0;
    int rval;
    while(size--){
        if( !(sysid->flags & REOLN ) ) {
            if( !sysid->bleft && ((rval = getblk(sysid)) < 0) )
                return count ? count : rval;
            if( sysid->rec.rleft <= 0 && ((rval = endrec(sysid)) < 0) )
                return count ? count : rval;
            if( sysid->flags & REOF ) {
                *buf++ = '\r';
                *buf = '\n';
                return count+2;
            }
        }
        if(sysid->flags & REOLN) {
            if(sysid->flags & RCRFLAG) {
                *buf++ = '\r';
                sysid->flags &= ~RCRFLAG;
            }
            else {
                *buf++ = '\n';
                sysid->flags &= ~REOLN;
            }
        }
        else if( sysid->rec.rleft ) {
            *buf++ = *sysid->bnptr++;
            --sysid->bleft;
            --sysid->rec.rleft;
        }
        else {
            /* case of zero records */
            ++size;
            continue;
        }
        ++count;
    } /* end of while */
return count;

bread(sysid)
register struct _rcb *sysid;
{
int *p;
int *q;
int rval;
rval = getblk(sysid);
  p = hbuf.buffer[hbuf.active];
  q = nbuf.buffer[nbuf.active];
  hbuf.buffer[hbuf.active] = q;
  sysid->bptr = sysid->bnptr = q;
  nbuf.buffer[nbuf.active] = p;
  hbuf.fd->base = nbuf.fd->_base = p;
  return rval;
}

__writet(sysid,buf,size)
register struct _rcb *sysid;
char *buf;
int size;
{
  if( size < 0 )
    return -1;    /* error */
  if( type == TYPE_A )
    return __put(sysid,buf,size));
  else
    return ( bwrite(sysid,buf,size));
}

bwrite(sysid,buf,size)
register struct _rcb *sysid;
char *buf;
int size;
{
  int *p;
  int *q;
  p = hbuf.buffer[hbuf.active];
  q = nbuf.buffer[nbuf.active];
  hbuf.buffer[hbuf.active] = q;
  sysid->bptr = sysid->bnptr = q;
  sysid->bleft = BLKSIZE - size;
  nbuf.buffer[nbuf.active] = p;
  nbuf.fd->_base = hbuf.fd->_base = p;
  if ( size < BLKSIZE )
    return 1;
  return putblk(sysid);
}
soread(s,buf,len)
int s;
char *buf;
int len;
{
  return ( sio(s,buf,len,IO_XFR|IX_RDS));
}

tsowrite(s,buf,len)
int s;
char *buf;
int len;
{
    return ( sio(s,buf,len,IO_XFR|IX_WRS) );
}
}

extern int astsio();

sio(s,buf,len,iocode)
int s;
char *buf;
int len;
int iocode;
{
    static struct iosb ios = {0};
    int rval;
    char *pbuf;
    int ret;

    /*
     * socket_efn += s_efn();
     */
    emt(WTSE, SOEFN);    /* stop for i/o completion */
    rval = nbuf.stat[!nbuf.active]; /* # of bytes read */
    if( (rval > 0) && (rval < len) && ( type != TYPE_A) &&
        (iocode == ( IO_XFR | IX_RDS ))
    )
        { /* Previous buffer is not yet completely read. 
            Since in the binary mode, buffer's are flipped 
            instead of data transfer. We need to read buffer 
            fully ( disk block = 512 bytes ). 
            */
            pbuf = nbuf.buffer[!nbuf.active];
            while (rval < len) {
                if(libemt(iocode,&ios,pbuf+rval,len-rval,0,0,0,s))
                    break;    /* I/O error */
                if(!(ret = ios.nread))
                    break;    /* EOF */
                rval += ret;
            } /* repeat loop, buffer is not yet read fully */
            nbuf.stat[!nbuf.active] = rval; /* # of bytes read */
    }
    /*
     * flip the buffer
     */
    nbuf.stat[nbuf.active] = 0;
    nbuf.active = !nbuf.active;
    nbuf.fd-> base = nbuf.buffer[nbuf.active];
    hbuf.fd-> base = nbuf.fd-> base;
    if( rval > 0 ) {
        emt(CLEF,SOEFN);
        emt(QIO,iocode,SOLUN,0,&ios,astsio,buf,len,0,0,0,s);
    }
}
337           return rval;
338
339         }
340
#include <rsxos.h>
#include <stdio.h>
#include <xspecial.h>
#include <xerrno.h>
#include <libsock.h>

extern char *xstrchr();
extern char *firstwhite();
extern char *lastwhite();
extern char *skipwhite();

char *
xraddr(desaddr)
   long desaddr;
{
    int od;
    XFILE *op ;
    char hbuf[XBUFSIZ], *cp, *host;
    int first = 1;

    od = xdopen( HOSTS, XREAD|XFASCII, FILE_NAME );
    if ( ( od < 0 ) || !( op = xdopen ( od, "r") ) )
        xerror(XEBADF, "xraddr: ");
    xexit(1);
    }

    top:
    while ( xogets(hbuf, sizeof (hbuf), op) & xstrchr(hbuf, '\n') ) {
        long addr,rnumber();

        *xstrchr(hbuf, '\n') = 0;
        if (hbuf[0] == '#')
            continue;
        if ((addr = rnumber(hbuf)) == -1)
            continue;
        if (addr != desaddr)
            continue;
        host = firstwhite(hbuf, ' ') + 1;
        host = skipwhite(host);
        cp = firstwhite(host, ' ');
        if (cp)
           *cp = 0;
        cp = xmalloc(xstrlen(host)+1);
        xstrcpy(cp, host);
        xclose( od );
        return (cp);
    }

    if (first == 1) {
        first = 0;
        xclose(od);
        if ( ((od = xdopen(HOSTSLOCAL,XREAD|XFASCII,FILE_NAME))>= 0)
          && ( op = xdopen( od, "r") ) )
            goto top;
        else{
            xerror( XEBADF ,"xraddr: ");
        }
57     xexit( 1 );
58     }
59     return (0);
60 }
61 bad:
62     xclose(od);
63     return (0);
64 }
/*
 * filename: RECEIVE.C
 */

#include <stdio.h>
#include <xeromo.h>
#include "libhdr.c"

static
int receive (s,from,buf,len)
int s;
struct sockaddr *from;
char *buf;
int len;
{
    int ret,i;
    struct SOictl SOictl;
    struct iosb iosb;

    if (from) {
        SOictl . hassa = 1;
        libcopy(from,&SOictl.sa,sizeof(struct sockaddr));
    } else SOictl . hassa = 0;
    ret = libemt(IO_XFRIX_RCV, &iosb, buf, len, &SOictl, 0, 0, s);
    if ( ret == 0)
        ret = iosb . nread;
    return ( ret );
}

xreceive(s, from, msg, len)
int s;
struct sockaddr *from;
char *msg;
int len;
{
    register XFILE *file;

    if( s < 0 || s >= XNFILE )
        return( XEBADF );
    file = & xiob[s];
    if( !( file->flag & XUsed ) )
        return( XEBADF );
    return( receive( (int)file->_sys_id, from, msg, len ));
}
#include <rsxos.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.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 xhost(ahost)
{
        char **ahost;

        int od;
        FILE *op;
        char hbuf[XBUFSIZE], *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 = xdopen(od, "r"))){
                xerror(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;
                                        strncpy(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 = xdopen (od, "r")))
    66            goto top;
    67        else{
    68            x perror(XEBADF,"rhost:");
    69            xexit(1);
    70        }
    71        return (-1);
    72    }
    73    bad:
    74        xclose(od);
    75        return (-1);
    76    }

77    long
78    rnumber(cp)
79    {
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')
    base = 16, cp++;
    while (c = *cp) {
        if (isdigit(c)) {
            val = (val * base) + (c - '0');
            cp++;
            continue;
        }
        if (base == 16 && isxdigit(c)) {
            val = (val << 4) + (c + 10 - (islower(c) ? 'a' : 'A'));
            cp++;
            continue;
        }
        break;
    }
    if (*cp == '.') {
/*
 * Internet format:
 *    net.host.lh.imp
 */
    if (pp >= pplim)
        return (-1);
    *pp++ = val, cp++;
    goto again;
    }
    if (*cp) {
/*
* if (*cp == 'n') return (xhtonl(val));
 */
    if (*cp == 'n') return (val);
    if ((*cp != ' ') && (*cp != '	')) return (-1);
    }
    if (pp >= pplim)
        return (-1);
    *pp++ = val;
/*
* return xhtonl(parts);
*/
    return (parts);
  }
  }

char *
skipwhite( cpt )
{
    char *cpt;
    {
        while( cpt && ( *cpt == ' ' || *cpt == '	' ) )
          ++cpt;
        return ( cpt );
    }
}

char *
firstwhite( cpt, ch )
/*
find first white space
169 */
170
171 char *cpt;
172 char ch;
173 {
174     while( cpt && *cpt && *cpt != ' ' && *cpt != '\' )
175         ++cpt;
176     if ( cpt && *cpt ) {
177         return( cpt );
178     }
179     return ( XNULL );
180 }
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     while( ( cpt = firstwhite( cpt, ' ' ) ) != XNULL ) {
194         ocpt = cpt;
195         cpt++;
196     }
197     return( ocpt );
198 }
199 }
int send (s, to, msg, len)
    int s;
    struct sockaddr *to;
    char *msg;
    int len;
{
    int ret,i;
    struct iosb iosb;
    struct SOictl SOictl;

    if ( to ) {
        SOictl . hassa = 1;
        libcopy(to,&SOictl.sa,sizeof (struct sockaddr));
    }
    else SOictl.hassa = 0;
    ret = libext(IO_XFR|IX SND, &iosb, msg, len, &SOictl, 0, 0, s);
    if ( ret == 0 )
        ret = iosb . nread;
    return (ret);
}

xsend(s, to, msg, len)

    int s;
    struct sockaddr *to;
    char *msg;
    int len;
{
    register XFILE *file;

    if( s < 0 || s >= _XNFILE )
        return( XEBADF );
    file = & xiob[s];
    if( !(file->flag & _XUsed )
        return( XEBADF );
    return( send((int)file->_sys_id, to, msg, len ));
}
int xsktaddr(s, addr)
    int s;
    struct sockaddr *addr;
{
    register XFILE *file;
    int ret;
    struct SOioctl SOioctl;
    struct iosb iosb;
    if( s < 0 || s >= XNFILE )
        return( XEBADF );
    file = &xiob[s];
    if( !(file->flag & _XUsed ) )
        return( XEBADF );
    if ( addr ){
        SOioctl . hassa = 1;
        libcopy(addr,&SOioctl.sa,sizeof(struct sockaddr));
    } else SOioctl . hassa = 0;
    ret = libmt(IO_ACS|SA_SAD,&iosb,0,0,&SOioctl,0,0, (int) file->_sys_id);
    if ( ret >= 0 )
        libcopy(&SOioctl.sa, addr, sizeof ( struct sockaddr ));
    return ( ret );
/*
 * filename: SOCKET.C
 */

#include <stdio.h>
#include <libhdr.c>

extern xsoread();
extern xswrite();
extern xsoioctl();

static
int socket( type, pf, addr, options)
{
    int type;
    struct sockproto *pf;
    struct sockaddr *addr;
    int options;

    int ret;
    struct iosb iosb;
    struct SOioctl SOioctl;

    if ( addr ){
        SOioctl. hassa = 1;
        libcopy( addr, &SOioctl.sa, sizeof ( struct sockaddr ));
    }
    else SOioctl. hassa = 0;

    if ( pf ){
        SOioctl. hassp = 1;
        libcopy( pf, &SOioctl.sp, sizeof ( struct sockproto ));
    }
    else SOioctl.hassp = 0;

    SOioctl . type = type;
    SOioctl . options = options;
    ret= libemt(IO ACS|SA_OPN,&iosb,0,0,&SOioctl,0,0,0);
    if ( ret == 0 )
        ret = iosb.nread;
    return ( ret );
}

int xsoclose( s )
{
    int s;
    {
        int ret;
        struct iosb iosb;

        ret = libemt(IO ACS|SA_CLS, &iosb, 0, 0, 0, 0, 0, 0, 0, 0, s);

        if (( iosb.cc ) && ( iosb.lc == 0 ))
            ret = 1;
        else ret = -1;
        return ( ret );
    }
xsocket( type, pf, addr, options )
int type;
struct sockproto *pf;
struct sockaddr *addr;
int options;
{
int rval;
int exosfd;
register XFILE *file;
rval = socket( type, pf, addr, options );
if( rval < 0 )
    return( rval );
exosfd = xnewod(); /* get a free file descriptor */
if( exosfd < 0 )
    return( exosfd );
file = & xioib[exosfd];
file->_flag |= _XIORW | _XPrimary ;
file->_sys_id = (char *)rval;
file->_read = xsoread;
file->_write = xsowrite;
file->_ioctl = xsoioctl;
file->_close = xsoclose;
return( exosfd );
}
int xsoioctl( dev, cmd, addr)
{
  int dev, cmd;
  char *addr;
  
  int ret;
  struct iosb iosb;
  struct SOioctl SOioctl;
  Uchar CMD = (Uchar) cmd;
  
  libcopy( addr, (char *)&SOioctl.hassa, sizeof ( struct SOioctl));
  ret = libmem(IO_SOC|CMD, &iosb, 0, 0, &SOioctl, 0, 0, dev);
  switch ( cmd ){
    case SIOCRCVOOB :
      *addr = *(char *)&SOioctl.hassa;
      break;
    case SIOCGLINGER :
    case SIOCGKEEP :
    case SIOCATMARK :
    case SIOCGPGRP :
      *(short *)addr = SOioctl.hassa;
      break;
    default :
      break;
  }
  
  return ( ret );
}
#include <stdio.h>
#include <xerror.h>
#include "libhdr.c"

int xsread(s, buf, len)
       int s;
       char *buf;
       int len;
       int ret, i;
       struct iosb iosb;
       ret = libemt(IO_XFR|IX_RDS, &iosb, buf, len, 0, 0, 0, s);
       if (ret==0)
           ret = iosb.nread;
       return(ret);
int xsowrite ( s, msg, len )
    int s;
    char *msg;
    int len;
    {
        int ret, i;
        struct iosb iosb;
        ret = libent(IO XPARIX WRS, &iosb, msg, len, 0, 0, 0, s);
        if ( ret==0 )
            ret = iosb . nread;
        return(ret);
/*
@(#)xaccess.c 1.4 3/29/85

RSX version of routine to check access rights.
*/

#include <xspecial.h>
#include <xerrno.h>

xaccess( name, special, mode )
char *name;
int special;
int mode;
{
    register int rval;
    char buf[ MXNAMELEN + 1 ];

    /* modify name (if necessary) for special meanings */

    rval = xmodname( &name, special, buf, sizeof( buf ) );
    if( rval < 0 )
        return( rval );
    if( mode == 0x0 )
        {
            /* check file exist or not */
            rval = xdopen( name, XFREAD, FILE_NAME );
            if(rval >= 0) /* check for success */
                xclose(rval);
            rval = 0;
        }
    else
        /* fail to open , ie file does not exist */
        rval = rval - 512;

    else
        /* rval = access(name,mode);
        ... to be implement
        ... Not used by FTP */
        rval = XSYSERR;
}

return( rval );
/*
 */

FILE_NAME: XCHDIR.C */

#include <xgenlib.h>
#include <xspecial.h>
#include <xpwd.h>
#include <xerrno.h>

#define EXEFN 010001

extern long radix();
extern struct passwd *pw;
extern char *csiblk;

xchdir(name, special)
char *name; /* name of uic to be modified */
int special; /* flag for special files */
{

char *uic;
int rval = 0;

switch ( special ) {

    case FILE_NAME:
        uic = name;
        break;

    case CURRENT_DIR:
        return(0);

    case HOME_DIR:
        xstrcpy(pw->cur_dev,pw->log_dev);
        xstrcpy(pw->cur_uic,pw->login_uic);
        return(0);

    default:

        return(XEINVAL);

}

rval = parse(uic,strlen(uic));
if(rval<0)
    return(XENOTDIR);
if(csiblk[C_STAT] & CS_NMF)
    return(XENOTDIR);
if(csiblk[C_STAT] & CS_DVF){
    xbcopy(*(int *)(csiblk+C_DEVD+2),pw->cur_dev,*(int *)(csiblk+C_DEVD));
    pw->cur_dev[*(int *)(csiblk + C_DEVD)] = '\0';
}
if(csiblk[C_STAT] & CS_DIF){
    uic = (char *)(*(int *)(csiblk + C_DIRD + 2 ));
    rval = val_uic(uic);
    if(rval >= 0)
        xstrcpy(pw->cur_uic, uic);
}
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);
}
/*
 * @(#)xchown.c 1.4 3/29/85
 * Xchown for RSX.
 */
#include <xspecial.h>
#include <rsxos.h>
#include <xpwd.h>

extern struct passwd *pw;

xchown( name, special )
char *name;
int special;
{
char buf[MXNAMELEN +1];
int rval;
struct pr {
char sy[5];
char ow[5];
char gr[5];
char wo[5];
} pr;

/*
char cmdlin[CMDSIZE];

rval = xmodname( &name, special, buf, sizeof(buf) );
if( rval < 0 )
return( rval );
mkcmd(cmdlin,"PIP ",name,"/NM/PR",
"/SY:",getown(pr.sy,pw->lgn_prv&017),
"/OW:",getown(pr.ow,pw->lgn_prv&0360),
"/GR:",getown(pr.gr,pw->lgn_prv&07400),
"/WO:",getown(pr.wo,pw->lgn_prv&0170000),
"/FO",0);
mkcmd(cmdlin,"PIP ",name,"/PR/NM/FO",0);
return(cmdcall(cmdlin));
*/
return(0);
}

/*@*
char *
getown(s,v)
char *s;
int v;
{
if(v & 01)
    *s++ = 'R';
if(v & 02)
    *s++ = 'W';
if(v & 04)
    *s++ = 'E';
if(v & 010)
    *s++ = 'D';
*s = '\0';
57     return(s);
58 }
59 */
/*
 */

Function to use for all RSX low level close routines.

#include <rsxos.h>
#include <stdio.h>
#include <xspecial.h>
#include <fcs.h>

extern struct _rcb _rcb[];

xdclose( sysid )
register struct _rcb *sysid;
{

  register int bytes;
  if( sysid->flags & RPREE )
    return 1;

  sysid->flags &= ~DBLBUF;
  if( sysid->mode & XFCREAT || sysid->mode & XFAPPEND ) {
    bytes = sysid->bleft;
    switch (bytes) {
    case 512:
      bytes = 0;
      break;
    case 0:
      putblk(sysid);
      break;
    default:
      putblk(sysid);
      bytes = 512 - bytes;
      bytes & 01 ? ++bytes : bytes;
      break;
    }

    _wmrec(sysid->fdb, sysid->rec.rsize,bytes);
    /* Write max rec & first free byte */
    xfree( sysid->rptr ); /* free record */
  }

  close(sysid->fdb); /* call CLOSE$ */
  xfree( sysid->bptr ); /* free block buffer */
  xfree(sysid->fdb); /* free FDB */
  dassign( sysid->rlun ); /* mark LUN as free */
  sysid->flags = RPREE; /* mark _RCB as free */
}
/*
@(#)xdir.c     1.4 3/29/85

Xdir(3X) for RSX - make a directory, remove a directory, move a file.
*/
#include <xspecial.h>
#include <xgenlib.h>
#include <xpwd.h>
extern mkcmd();
extern long radix();
extern char *csiblk;
extern struct passwd *pw;
#define EXEFN     010000

mkdir (what, special )
char *what;
int special;
{
    return(XEOPNOTSUPP);
}

char buf[ MXNAMELEN + 1 ];
cmdlin[CMDSIZE];
int rval;

rval = xmodname( &what, special, buf, sizeof( buf ) );
if( rval < 0 )
    return(XENOTDIR );
if(!!(csiblk[C_STAT] & CS_DVF))
    return(rval - 512);
if(!!(csiblk[C_STAT] & CS_DIF))
    return(XENOTDIR);
if(csiblk[C_STAT] & CS_NMF)
    return(XENOTDIR);
if((rval = val_uic( *((int *) (csiblk + C_DIRD + 2))) ) < 0)
    return(rval);
mkcmd(cmdlin,"UPD ", what, 0 );
return(cmdcall(cmdlin));
*/

rmdir (what, special )
char *what;
int special;
{
    char buf[ MXNAMELEN + 1 ];
    int rval;
    char cmdlin[CMDSIZE];
    char uic[UICSIZE];
    char *puic= uic;
    char dev[6];
    char *cuic;

    rval = xmodname( &what, special, buf, sizeof( buf ) );
    if( rval < 0 )
        return(XENOENT);
    if((csiblk[C_STAT] & CS_NMF) || (!(csiblk[C_STAT] & CS_DIF)) )
return(XENOTDIR);
58   if(csiblk[CS_STAT] & CS_DVF){
59     xbcopy((int*)(csiblk + C_DEVD + 2),dev, *((int*)(csiblk + C_DEVD)) -
60       dev[*(int*)(csiblk + C_DEVD)] = '1';
61   }
62   else {
63     xstrncpy(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[ MNAMELELEN + 1 ];
81 char buf2[ MNAMELELEN + 1 ];
82 char cmdlin[CMDSIZE];
83 int rval;
84 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 }

/*
 * xopen(3x) for Rsx.
 */

#include <rsxos.h>
#include <xstdio.h>
#include <xspecial.h>
#include <xerrno.h>
#include <fcs.h>

extern xdread();
extern xfwrite();
extern xfclose();
extern char *csqlk;
extern struct _rcb _rcb[];

xdopen( name, mod, special )
char *name;
register int mod;
int special;
{
    int rval;
    int exosfd;
    int rmode;
    int ioflag;
    int rsize;    /* file type . if ascii then _rcb[sysid].rec.rsize = 0
                  else non-zero.
               */
    register struct _rcb *sysid;
    register XFILE *file;
    char buf[MXNAMELEN + 1];

    rval = xmodname( &name, special, buf, sizeof( buf ) );
    if( rval < 0 )
        return( rval );

    Translate mode to Rsx mode and type.
    */
sysid = newrcb(mod);
    if((int )sysid < 0)
        return (int )sysid;
    rmode = xtranmode( mod, &ioflag );
    if ( rmode < 0 )
        return( rmode );

    exosfd = xnewod();    /* get a free file descriptor */
    if( exosfd < 0 )
        return( exosfd );
    if( sysid->mode & XFCREAT)
        create( sysid->fdb, sysid->rec.rsize);
    rval = parse(name, xstrlen(name));
    if(rval < 0)
        return rval;
    rval = __open(sysid->fdb,rmode,sysid->rlun,csqlk+C_DSDS);
if(rval < 0)
    return(rval);
if( mod & XFAPPEND )
    getblk(sysid); /* get last block */
file = &_xiof[exosfd];
file->_file = exosfd;
file->_flag |= oflag;
file->_sys_id = (char *)sysid;
file->_read = xdread;
file->_write = xdwrite;
file->_close = xdclose;
return( exosfd );
}

struct _rcb *
newrcb(mod)
register int mod;
{
    register struct _rcb *sysid = _rcb;
    char *p;
    int i;

    for(i=0; i < _XNFILE; ++i, ++sysid )
    {
        if(sysid->flags & RFREE )
            break;
    }
    if ( i >= _XNFILE)
        return (struct _rcb *) XEMFILE; /* Too many files open */

    sysid->mode = mod;
    sysid->flags = RUSED;
    sysid->bptr = xmalloc(BLKSIZE);
    sysid->fdb = xmalloc(FDBSIZE);
    for( p = sysid->fdb; p < sysid->fdb + FDBSIZE ; ++p )
        *p = 0;
    if( !sysid->bptr || !sysid->fdb )
        return (struct _rcb *) XENOMEM; /* No memory */
    sysid->bptr = sysid->bptr;
    if( mod & XFCREAT || mod & XFAPPEND )
    {
        sysid->bleft = BLKSIZE;
        sysid->rptr = xmalloc(RECSIZE);
        if ( mod & XFASCII )
            sysid->rec.rsize = 0;
        else
            sysid->rec.rsize = 512;
    }
    else {
        sysid->bleft = 0;
        sysid->rptr = 0;
        sysid->rec.rleft = -1;
    }
    sysid->rnptr = sysid->rptr;
    sysid->rlun = glun();
    return sysid;
getblk(sysid)
register struct _rcb *sysid;
{
    register char *fdb = sysid->fdb;
    int ffby = *((int *)(fdb + F_FFBY ));
    sysid->rec.rsize = *((int *)(fdb + F_RSIZ ));
    if( ffby ) {
        getblk(sysid);
        sysid->bnptr += ffby;
        sysid->bleft = BLKSIZE - ffby;
        --*((long *)(fdb + F_BKVB ));
    }
}
/*
 * FILE XGZ
 *
 * Function to use for all RSX low level read routines.
 */

#include <stdio.h>
#include <xspecial.h>
#include <extypes.h>
#include <fcs.h>

extern struct _rcb _rcb[];

#define endblk(i) (!!i->bleft) ? getblk(i) : 1

xread( sysid, buf, size )
register struct _rcb *sysid;
char *buf;
int size;
{
    if( size < 0 )
        return -1;  /* error */
    if( sysid->flags & REOF )
        return 0;  /* eof */
    if( sysid->mode & XFASCII )
        return (_get(sysid,buf,size));
    else
        return (read(sysid,buf,size));
}

read(sysid,buf,size)
register struct _rcb *sysid;
char *buf;
register int size;
{
    register int count = 0;
    int rval;

    while(size--)
    {
        if((rval = endblk(sysid)) <= 0 )
            return count ? count : rval;
        *buf++ = *sysid->bptr++;
        --sysid->bleft;
        ++count;
    }

    return count;
}

_get(sysid,buf,size)
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) ) /* read in a block */
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'; sysid->flags &= ~REOLN; /* reset */
74 }
75 else if( sysid->rec.rleft ) {
76 *buf++ = *sysid->bnptr++;
77 --sysid->bleft;
78 --sysid->rec.rleft;
79 }
80 else { /* case of zero records */
81 ++size;
82 continue;
83 }
84 ++count;
85 }
86 return count;
87
89
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) /* initialize the end of a file */
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 extern int _read();
extern int rastdio();
extern int __rwait();

getblk(sysid)
register struct _rcb *sysid;
{
    register int ret;
    ret = dio(sysid, __read, rastdio, __rwait);
    if(ret == 0)
        sysid->flags |= REOF;
    sysid->bleft = (ret > 0) ? ret : 0;
    return ret;
}
/*

Function to use for all low level write routines.
*/

#include <stdio.h>
#include <xspecial.h>
#include <etypes.h>
#include <fcs.h>

extern struct _rcb _rcb[];

xdwrite(sysid, buf, size)
register struct _rcb *sysid;
char * buf;
int size;
{
    if ( size < 0 )
        return -1; /* error */
    if( sysid->mode & XFASCII )
        return _put(sysid, buf, size);
    else
        return write(sysid, buf, size);
}

write(sysid, buf, size)
register struct _rcb *sysid;
char * buf;
register int size;
{
    register int count = 0;
    int rval;

    while ( size-- ) {
        if( !sysid->bleft ) {
            if((rval = putblk(sysid)) <= 0)
                return count ? count : rval;
        }
        *sysid->bnptr++ = *buf++;
        --sysid->bleft;
        ++count;
    }
    return count;
}

_put(sysid, buf, cnt)
register struct _rcb *sysid;
char *buf;
int cnt;
{
    char *dbuf;

register char *wbuf;
int ncnt, wcnt, rval, count = 0, tot = 0;

while(cnt) {
    wbuf = buf;
    wcnt = cnt;
    xlocc('\n', wcnt, wbuf, &ncnt, &nbbuf);
    if(ncnt) {
        /* found EOL, backtrack to find a '\r', if any */
        if((*buf != '\n') && (nbbuf[-1] == '\r'))
            count = cnt - ncnt - 1;
        else
            count = cnt - ncnt;
        if((rval = putrec(sysid, buf, count)) <= 0)
            return count ? count : rval;
    cnt = ncnt - 1;
    buf = nbbuf + 1;
    tot += cnt - ncnt + 1;
    } /* end of if(ncnt)... */
else { /* if ncnt == 0, i.e. no '\n' found in buffer */
    xcbcopy(buf, sysid->rptr, cnt);
    sysid->rnptr += cnt;
    if((sysid->rnptr - sysid->rptr) > BLKSIZE)
        sysid->rnptr = sysid->rptr + BLKSIZE;
    sysid->flags |= KEPT_ASIDE;
    tot += cnt; /* the kept aside bytes */
    cnt = 0;
}
return tot;
}

putrec(sysid, buf, size)
register struct _rcb *sysid;
register char *buf;
int size;
{
    register int kept_aside;
    int rval;
    if((Ushort )sysid->bnptr & 01) { /* if on a byte boundary */
        *sysid->bnptr++ = 0;
        --sysid->bleft;
    }
    if(!sysid->bleft && ((rval = putblk(sysid)) <= 0))
        return rval;
    if(sysid->flags & KEPT_ASIDE)
        kept_aside = sysid->rnptr - sysid->rptr;
    else
        kept_aside = 0;
    *(int *)sysid->bnptr = size + kept_aside;
sysid->bnptr += 2;
sysid->bleft -= 2;

if(sysid->flags & KEPT_ASIDE) {
    sysid->flags &= ~KEPT_ASIDE;
    if((rval = _rec(sysid, sysid->rptr, kept_aside)) <= 0)
        return rval;
    sysid->rnptr = sysid->rptr;
}
if((rval = _rec(sysid, buf, size)) <= 0)
    return rval;

return 1;

_rec(sysid, buf, cnt)
register struct _rcb *sysid;
char *buf;
register int cnt;
{
    register int leftcnt = 0;
    int rval;

    if(sysid->bleft < cnt) {
        leftcnt = sysid->bleft;
        xcbcopy(buf, sysid->bnptr, leftcnt);
        if((rval = putblk(sysid)) <= 0)
            return rval;
    }
    xcbcopy(buf + leftcnt, sysid->bnptr, cnt - leftcnt);
    sysid->bnptr += cnt - leftcnt;
    sysid->bleft -= cnt - leftcnt;

    return 1;

extern int wastdio();
extern int __write();
extern int __wait();

putblk(sysid)
register struct _rcb *sysid;
{
    sysid->bleft = BLKSIZE;
    return dio(sysid, __write, wastdio, __wait);
}

xloc(c, cnt1, buf1, acnt2, abuf2)
char c;
register int cnt1;
169 register char *buf1;
170 int *acnt2;
171 char **abuf2;
172 {
173     register int i;
174     int found = 0;
175
176     for(i = 0; i < cnt1; i++)
177         if(*buf1++ == c) {
178             found++;
179             break;
180         }
181     if(found) {
182         *acnt2 = cnt1 - i;
183         *abuf2 = --buf1;
184     }
185     else
186         *acnt2 = 0;     /* char 'c' not found */
/*
 * %W% %G%
 * Unix specific close all EXOS file objects and exit program.
 */

#include <stdio.h>

xexit( status )

int status;
{
    int i;

    for( i = 0; i < _XNFILE ; ++i )
        xclose( i );

    exit( status );
}
/*
 * xftopen(3x) for Rsx.
 */

#include <stdio.h>
#include <xspecial.h>
#include <xerrno.h>
#include <fcs.h>
#include <ftp.h>;

extern xdrread();
extern xdrwrite();
extern xdclose();
extern _dread();
extern _dwrite();

extern int type;
extern struct dbuf hbuf;

xftopen( name, mode, special, ftp_attributes )
char *name;
int mode;
int special;
register struct ftp_attr *ftp_attributes;
{
    int rval;
    register XFILE *file;

    /*
    Check that ftp_attributes are supported.
    */
    if( ftp_attributes )
{
        if( (ftp_attributes->rep_type != RT_ASCII &&
            ftp_attributes->rep_type != RT_IMAGE ) ||
            ftp_attributes->format != TF_NONPRINT ||
            ftp_attributes->structure != IS_FILE ||
            ftp_attributes->trans_mode != TM_STREAM)
            return( XEOFNOTSUPP );
    }

    if( type == TYPE_A)
    mode = XPASCII;
    rval = xdopen(name, mode, special);
    if(rval >= 0 ){
        file = & _xioib[rval];
        file-> _read = _dread;
        file-> _write = _dwrite;
        if( mode & XPREAD ){
            hbuf.stat[0] = getblk(file-> _sys_id);
        }
    }
hbuf.buffer[0] = ((struct _rcb *) file->_sys_id)->bptr;
}
return rval;
}
#include <arp.h>

extern long xrhost();


struct hostent *
ghbaddr( addr, size, family )
/*
gethostbyaddr for C compilers with 8 character identifiers
WARNING ---- a second call to this routine will destroy the previous result.
*/

struct in_addr *addr;
int size, family;
{
    static struct hostent hent;

    hent.h name = (char *)xraddr( addr->s_addr );
    return( ( struct hostent *)&hent );
}
/*
static char scsId[] = "@(#)xgetbhnam.c 1.4 3/26/85"

code to make 4.2 style code, sort of, happy.
*/

#include <arp.h>

extern long xrhost();

struct hostent *
ghbname( host )
/*
gethostbyname for C compilers with 8 character identifiers
WARNING ----
a second call to this routine will destroy the previous result.
*/

char *host;
{
static struct hostent hent;
static struct sckadr_in sock;

sock.sin_addr.S_un.S_addr = (long)xrhost( &host );
if (sock.sin_addr.S_un.S_addr == -1 )
    return( (struct hostent *)XNULL);

hent.h_addr = (char *)&sock.sin_addr;

hent.h_name = host;
return( &hent );
}
/ * FILE_NAME: XGLOB.C */Xglob for RSX. Expand wild word in input line. */
#include <xgenlib.h>
#include <xspecial.h>
define MAXINSPEC 20
extern xgfatal();
static char **gargv = 0;
static short gargc = 0;
char *globerr = (char *)0;
short gflag = 0;
char *in_stat = (char *)0;
char *f_stat = (char *)0;
char *fver_stat = (char *)0;
char *fver_stat = (char *)0;
int globbing = 0;

/*
 * Main root of xglob.
 */
char **
xglob( v )
register char **v;
{
char **agargv;

/* initialize return parameter */
gargv = xmalloc(2);
*agargv = (char *)0;
globerr = (char *)0;
globbing = 1;
in_stat = xmalloc(MAXINSPEC);
inver_stat = xmalloc(MAXINSPEC);
gargc = 0;
while (*v) {
if(wildchar(*v)){
    agargv = glob(*v++);
}
else {
    agargv = xmalloc(strlen(*v)+1 +4);
    if(agargv == (char *)0)
        xgfatal("Out of Memory");
    *agargv = agargv + 2;
    agargv[1] = (char *)0;
    xstrcpy(agargv + 2, *v++);
}
gargv = copyblk(gargv,agargv);
```c
57     }
58     xfree(in_stat);
59     xfree(in_ver_stat);
60     globbing = 0;
61     return((gargv);
62
63 }
64
65 char **
66 glob(name)
67 char *name;
68 {
69     char buf[400];
70     char *line = 0;
71     char template[16];
72     char list_name[27];
73     char *bufp = buf;
74     int len;
75     int rval;
76     int sys_id;
77     XFILE *file;
78     int argc;
79     char **argv = (char **)0;
80     /* xstrcpy(template,SCRATCHFILE);
81     /* xmktemp(template); */
82     rval = ls(template, name, LS_ARG);
83     if(rval < 0){
84         globerr = " glob failed";
85         return(0);
86     }
87     sys_id = opentemp(template);
88     if(sys_id >= 0){
89         file = xodopen(sys_id,"r");
90     }
91     if(sys_id < 0 || file < ( XFILE *) 0 ) {
92         globerr = " Can't open file for globbing";
93         return(0);
94     }
95     /*
96     * initialize buff, which is used to filled with list of names
97     */
98
99     f_stat = in_stat - 1;      /* used in nam_list */
100    f_ver_stat = in_ver_stat - 1;  /* used in nam_list */
101    for(bufp=(char *)buf; bufp<(char *)buf + sizeof buf;)
102        *bufp++ = '\0';
103    bufp = buf;
104    while(rval = xogets(list_name,sizeof(list_name), file) > 0 ) {
105        if((!nam_list(list_name))
106            continue;        /* discarding temp name created by glob */
107        remtrail(list_name);
108        len = xstrlen(list_name) + 1;
109        if((buf + sizeof(buf) - bufp) > len)
```
{  
    xstrncpy(bufp,list_name);
    bufp += len;
    *(bufp -1) = ' ';  /* name separator */
}

else
{
    argv = copyblk(argv, xmkarglist(buf, &argc));
    for(bufp=(char *)buf; bufp<(char *)buf + sizeof buf;)
        *bufp++ = '\0';
    bufp = buf;
}

if( bufp > buf)
    argv = copyblk(argv, xmkarglist(buf, &argc));
xclose(xfileno(file));
rval = xunlink(template,FILE_NAME);
if(rval < 0) {
    globerr = " system error -- can't delete file ";
    return(0);
}

return(argv);
}

char **
copyblk(v1, v2)
char **v1;
char **v2;
{
    register char **nv ;
    int i;
    i = (blklen(v1) + 1) * sizeof(char **) + blklen(v1)
        + (blklen(v2) + 1) * sizeof(char **) + blklen(v2)
    ;
    nv = xmalloc(i);
    if(nv == (char *)0)
        xgfatal("Out of Memory");
    return(blkcpy(nv, v1, v2));
}

char **
blkcpy(v, v1, v2)
char **v, **v1, **v2;
{
    register char **av = v;
    char **ovl = v1;
    char **ov2 = v2;
    char *stringp;

    if(v1){
        while(*v1++)
            ++av;
if(v2){
    while(*v2++)
        **av;
}
stringp = (char *)++av;
av = v;
v1 = ov1;
v2 = ov2;
if(v1){
    while(*v1){
        *av++ = stringp;
xstrncpy(stringp, *v1);
        stringp += strlen(*v1++) + 1;
    }
}
if(v2){
    while(*v2){
        *av++ = stringp;
xstrncpy(stringp, *v2);
        stringp += strlen(*v2++) + 1;
    }
    *av = (char *)0;
}
if(ov1)
xfree(ov1);
if(ov2)
xfree(ov2);
return(v);
}
wildchar(p)
char *p;
{
    while(*p){
        if(( *p == '*' || (*p == '%'))
            return(1);
        p++;
    }
    return(0);
}
xgfatal(string)
char *string;
{
xprintf(xstderr, "xglob:%s
", string);
exit(1);
}
register char **av;
{
    register int i = 0;
    if (av != XNULL)
        while (*av++)
            i++;
    return (i);
}

static
blkslen(argp)
{
    register char **argp;
    int total = 0;
    if (argp != XNULL)
        while (*argp)
            total += xstrlen(*argp++) + 1;
    return (total);
}

remtrail(s)
char *s;
{
    char *start;
    start = s;
    s = s + xstrlen(s) - 1;
    while ((s >= start) && ((s == '\r') || (s == '\n')))
        *s-- = '\0';
}
/ *
3 RSX implementation of xinit_env(3X).
4 */
5 #include <xpwd.h>
6 #include <xgenlib.h>
7 #include <xcctype.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 extern long radix();
14 extern ast_recv();
15 char msge[] = "
16 extern struct passwd *pw;
17 static int buf[16] = {0};
18
19 xinit_env( name, password, account )
20 {
21 char *name; /* loggin name */
22 char *password; /* password */
23 char *account; /* login uic */
24 {
25 int rval;
26 int dirdes[2];
27 long task;
28
29 if ( !name )
30     return(0);
31 emt(SRDA,ast_recv);
32 emt(CLEF,TASK_EFN);
33 emt(CTSK,buf); /* get task info for type of system this demon is running on */
34 /* validate user's login information */
35 rval = login(name,password,account);
36 if ( rval < 0 )
37     return(rval - 512);
38 dirdes[0] = xstrlen(pw->login_uic);
39 dirdes[1] = (int) pw->login_uic;
40 ascpp(dirdes,msge);
41 emt(CLEF,TASK_EFN);
42
43 if(buf[14] == 6) /* is it an RSX-11M-PLUS system */
44     task = radix("D"EMTO ");
45 else /* it is an RSX-11M system */
46     task = radix("...DEM");
47     emt(SDAT,task,msge,0); /* send login uic to ...
dem , which updates this task uic as user's login uic */
48 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 aseln = 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("LGNT0 ");
75   else
76     tsk = radix("...LGN");
77
78   while( name[++i] && cc < 14 ){
79     if(name[i] == '/')
80       break;
81     msg[e][cc++] = upper(name[i]);
82   }
83   while(cc < 14)
84     msg[e][cc++] = ' ';        /* padded the name with blanks */
85   msg[e][cc++] = '*';         /* separator between name & account */
86   if( name[i] == '/' && (name[i+1]) ){
87     while(name[++i] && cc < 26)
88       msg[e][cc++] = upper(name[i]);
89   }
90   else {
91     i = -1;
92     while( password[++i] && cc < 26 )
93       msg[e][cc++] = upper(password[i]);
94   }
95   while(cc < 26)
96     msg[e][cc++] = ' ';        /* send pkt to ...Ign task    */
97   emt(SDAT,tsk,msg,e,0);      /* unstop ...Ign */
98   emt(USTP,tsk);
99   emt(WTSE, TASK_EFN);        /* wait for receive pkt from ...Ign */
100  rval = * ( (int *)(msg[e] + 4) );  /* return status */
101  if(rval == 0) {
102     xstrcpy(pw->cur_uic, msg[e]+6);
103     xstrcpy(pw->login_uic, msg[e]+6);
104     xstrcpy(pw->log_dev, msg[e]+16);
105     xstrcpy(pw->cur_dev, msg[e]+16);
106  }
107
108  return(rval);
109
110 }
#include <xgenlib.h>
#include <xspecial.h>
#include <xpwd.h>
extern long radix();
extern xread();
extern xdread();
extern char *caiblk;
extern struct passwd *pw;
extern char *f_stat;
extern char *in_stat;
extern char *f_ver_stat;
extern char *in_ver_stat;
extern int globbing;

#define EXEFN 010000

static char brief[] = "/BR";
static char full[] = "/FU";
char hdir[27] = {0};

xls(od, name, code)
        /* io object for network data connection */
        /* name of uic to list, null == current */
        int code;
        {
            register XFILE *file;
            char template[16];
            int rval;
            int sys_id;
            int d;

            if (od<0 || od >= XNFLLE)
                return(XEBADP);
            if(name && *name && ((code == LS) || (code == LS_ARG))) {
                rval = checkname(name);
                if(rval < 0)
                    return(rval);

                /* check name is dir or simple name */
                if(rval) {
                    d = xaccess(name,FILE_NAME,0); /* check file exists or not */
                    if(d < 0) {
                        return(XENOENT);
                    }
                    d = xprintf(&_xiob[od],"%s\n",name);
                    if(d < 0) {
                        xerror(d,"on output");
                    }
                    return(d);
                }
            } else {
                strcpy(template,SCRATCHFILE);
                /* xmktemp(template); */
                rval = ls(template, name, code);
                if(rval < 0)
return(rval);
sys_id = opentemp(template);
if(sys_id < 0)
    return(sys_id);
file = xodopen(sys_id,"r");
if(file < (XFILE *)0)
    return((int*)file);
xpass(file, &_xib[od]);
xclose(xfileno(file));
rval = xunlink(template, FILE_NAME);
return(rval);
}

xpass( inod, outod )

XFILE *inod; /* input EXOS io object */
XFILE *outod; /* output EXOS io object */
{
    int c;
    int d;
    char name[512];
    while(c = xogets(name,sizeof (name) ,inod) > 0){
        if(!nam_list(name))
            continue;
        xoprintf(outod,"%s",name);
        xfflush(outod);
    }
    if(c < 0) {
        x perror(c, "on input");
        return(c);
    }
}

int _ls(template, name, code)
char *template;
char *name;
int code;
{
    char *swtch;
    char cmdlin[CMDSIZE];
    long tsk;
    int rval;
    char *cur_name; /* use to replace user's if supplied or
                     pointing to null. */
    char inspec[CMDSIZE];
    char *inp = inspec;
    char *f_stat = in_stat;
    char *f_ver_stat = in_ver_stat;
    char _stat = '\0';
    switch (code){
    case LS:
    case LS_ARG:
        swtch = brief;
        break;
    case LSLONG:
    case LSLONG_ARG:
        swtch = full;
        break;
    default:
        return(XEINVAL);
    }

    if( name == ( char *)0)
        cur_name = ( char *)&name;
    else
        cur_name = name;

    /* create command to produce list */
    for(; ; ){
        rval = parse(cur_name, xstrlen(cur_name));
        if(rval < 0)
            return(rval);
        if((f_stat > in_stat) &&
            ((f_stat & CS_DVF) & !((csiblk[C_STAT] & CS_DVF)) ||
            ((f_stat & CS_DIF) & !((csiblk[C_STAT] & CS_DIF))
            ))
            return(-1);  /* command syntax error */
        f_stat |= csiblk[C_STAT];
        f_stat++ = csiblk[C_STAT];

        *f_ver_stat = 0;  /* assume version is not specified */
        if(csiblk[C_STAT] & CS_NMF){
            int i;
            for(i=0; i < *((int *)(csiblk + C_FILD)); ++i)
                if(*((char *)(int *)(csiblk + C_FILD + 2) + i) == ';') {
                    *f_ver_stat = 1;  /* version is indeed specified */
                    break;
                }
        }
        f_ver_stat++;

    if((rval = mkname(inp)) == 0)
        break;  /* no more in spec */
    inspec += xstrlen(inspec);
    *inp++ = ';';
    cur_name += rval + 1;
}

    mkcmd(cmdlin, "PIP ", pw->log_dev, ",":$, pw->login_uic, template, ",":", inspec,
           swtch, "/NM", 0);
    return(cmdcall(cmdlin));
}

    checkname(name)
    char *name;  /* IN-OUT */
{
    int rval;
rval = parse(name, xstrlen(name));
if(rval < 0)
    return(rval -512);
if(csiblk[C_STAT] & CS_NMF)
    rval = 1;
return(rval);

opentemp(file)
char *file;
{
    /*
    char name[27];
    
    xstrncpy(name, pw->login_uic);
    xstrcat(name, file);
    return(xdopen(name, XFREAD|XFASCII, HOME_DIR));
    */
    return(xdopen(file, XFREAD|XFASCII, HM_RELATIVE));
}

nam_list(name)
char *name;
{
    char buf[27];
    char *cpl, *cp2;
    if(xstrlen(name) > 1)
        if((name[1] == 'i') || (name[2] == 'i')){
            if(globbing){
                f_stat++; f_ver_stat++;
                hdir[0] = '\0';
                if(!(*f_stat & (CS_DVF | CS_DIF)))
                    return(0);
                cpl = xstrchr(name, '\');
                cp2 = xstrchr(name, ':') + 1;
                if(*f_stat & CS_DVF)
                    xstrncat(hdir, cpl, cp2 - cpl);
                if(*f_stat & CS_DIF)
                    xstrcat(hdir, cp2);
                remtrail(hdir);
            }
            return(0);
        }
    if(( name[0] == '\14' ) || (name[0] == '\n')
        || (name[0] == '\r'))
        return(0);
    if( (name[0] == ' ') ||
        (name[1] == '-') ||
        (name[2] == '-') ||
        (xstrn cmp(name, "Total of ",9) == 0)
    )
        return(0);
if(xstrncmp(name, ":","2) == 0)
    return(0);

if(globbing && !(*f_ver_stat))
    remver(name);

if(globbing && xstrlen(hdir)){
    xstrcpy(buf,hdir);
    xstrcat(buf, name);
    xstrcpy(name, buf);
}

return(1);

remver(s)
char *s;
{
    s = xstrchr(s,';');
    while(*s)
        *s++ = '\0';
}
#include <xgenlib.h>
#include <xpwd.h>
extern struct passwd *pw;
char *
xmkttemp(template)
char *template;
{
    return(1);
}
/*
 * Rsx routine to form file names relative to users login uic, current uic, etc.
 * This routine belongs in Xoslib, but is here to keep the linker happy.
 */
#include <xspecial.h>
#include <xgenlib.h>
#include <xpwd.h>

/*
for now ...
*/
define xsprintf sprintf
extern struct passwd *pw;

xmodname( name, special, buf, sz_buf )
char **name;
int special;
char *buf;
int sz_buf;
{
int rval;
char *pt = buf;

switch( special ) {
    case FILE_NAME:
        rval = parse(*name,xstrlen(*name));
        if(rval < 0)
            return(rval); /* error while parsing */
        mknname(pt);
        break;
    case CURRENT_DIR:
        xstrcpy(buf, pw->cur_uic);
        break;
    case HM_RELATIVE:
    case HOME_DIR:
        xstrcpy(buf, pw->log_dev);
        xstrcat(buf, ":" );
        xstrcat(buf, pw->login_uic);
        if ( special == HM_RELATIVE )
            xstrcat(buf, *name);
        break;
    case CD_RELATIVE:
        xstrcpy(buf, pw->cur_uic);
        xstrcat(buf, *name);
        break;
    default:
        return( XEINVAL );
    }
  *name = pt;
  return( 0 );
}
#include <xgenlib.h>
#include <xspecial.h>
#include <xpwd.h>
extern struct passwd *pw;

xpwd(buf, buflen, func_code)
    char *buf;            /* buffer to hold name of current uic */
    int buflen;           /* length of buffer */
    int func_code;        /* consistency check */
{
    if(func_code != PWD)
        return(XEINVAL);
    xstrncpy(buf,pw->cur_dev,buflen);
    xstrncat(buf,":\",buflen);
    xstrncat(buf, pw->cur_uic, buflen - xstrlen(pw->cur_dev) - 1);
    buf[buflen] = '\0';
    return(0);
}
/*
 * %WZ %CZ
 * convert a RSX file descriptor to an EXOS io object
 * - useful in debugging and development.
 */

#include <stdio.h>

extern xdread();
extern xdwrit e();
extern xdclos e();
extern xnofunc();

xrxtex( rsxf d )
int rsxf d;
{
int rval;
int exosfd;
register XFILE *file;
rval = rsxf d;
if( rval < 0 )
    return( rval - 512 );
exosfd = xnewod(); /* get a free file descriptor */
if( exosfd < 0 )
    return( exosfd );
file = & xio[ exosfd ];
file-> file = exosfd;
file-> flag |= _XIORW | _XPrimary ;
file-> sys_id = (char *)rsxf d;
file-> read = xdread;
file-> write = xdwrit e;
file-> ioctl = xnofunc;
file-> close = xdclos e;
return( exosfd );
}
/*
 *  FILENAME          XSELECT.C
 */

#include <xgenlib.h>
#include "libhdr.c"

#define SELECT_EFN  4
#define READ     0
#define WRITE    1

extern int _astrselect();
extern int _astwselect();

long rmask = (long) 0;
long wmask = (long) 0;
long *prmask = (long *) 0;
long *pwmask = (long *) 0;
int nfound = 0;
struct iosb iosb_select = {0};
unsigned char rsavxio[2] = {0};
unsigned char wsavxio[2] = {0};

xselect(nods, readods, writeods, timeout)
int nods;
long *readods;
long *writeods;
long timeout;
{
    int i, ch_no;
    int tick = (int)(timeout / 20L);

    if(!readods && !writeods)
        return(0);
    rmask = wmask = (long) 0;
    emt(CELF, SELECT_EFN); /* make sure efn is clear */
    emt(DSAR); /* disable ast recognition */

    if(readods)
        for( i = 0; i < nods; ++i)
            if(getod(readods, i))
                ch_no = (int) xioib[i].sys_id;
                rsavxio[ch_no] = i;
                emt(QIO, IO_ACS|SA_SEL,SOLUN,0,&iosb_select,
                         _astrselect,0,0,READ,0,ch_no);
            }

    if(writeods)
        for( i = 0; i < nods; ++i)
            if(getod(writeods, i))
                ch_no = (int) xioib[i].sys_id;
                wsavxio[ch_no] = i;
                emt(QIO, IO_ACS|SA_SEL,SOLUN,0,&iosb_select,
                         _astwselect,0,0,WRITE,0,ch_no);
        }
}  
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 nfound = 0;  
67  
68 /* specify timeout efn */  
69 emt(MRKT, SELECT_EFN, (int)t1ck, 1, 0);  
70  
71 emt(ENAR);  
72 emt(WTST, SELECT_EFN); /* wait for either timeout or at least one ast*/  
73 emt(DSAR); /* disable ast recognition */  
74 unselect(nods, readods, writeods); /* unselect unready od's */  
75 rmask = wmask = (long)0; /* now on ast must be ignore */  
76 emt(ENAR); /* enable ast recognition */  
77  
78 return(nfound);  
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     astselect(&rmask, prmask, rsavxiob[ch_no]);  
92 }  
93  
94  
95 astwselect(iosb)  
96 struct iosb *iosb;  
97 {  
98     int ch_no = iosb->nread;  
99     astselect(&wmask, pwmask, wsavxiob[ch_no]);  
100 }  
101  
102  
103  
104  
105 astselect(mask, pmask, s)  
106 long *mask;  
107 long *pmask;  
108 int s; /* xiob number */  
109 {  
110     if(!getod(mask, s))  
111         return; /* spurious ast */  
112 }
setod(pmask,s);
nfounds++;
emt(SETF, SELECT_EFN);
}

getod(mask,f)
long *mask;
int f;
{
    int *p = (int *) mask;
    long r = (long)(1 << f);
    int *q = (int *)(&r);

    if(*p++ & *q++)
        return(1);
    if(*p & *q )
        return(1);
    return(0);
}

setod(mask,f)
long *mask;
int f;
{
    int *p = (int *) mask;
    long r = (long)(1 << f);
    int *q = (int *)(&r);

    *p++ |= *q++;
    *p  |= *q;

    /*
     * This routine unsels the select requests after the time out expires
     * and the od's that are ready are not unselected.
     */

    unselect(nods,readods,writeods)
    int nods;
    long *readods;
    long *writeods;
    {
        int ch_no,i;

        if(readods)
            for( i = 0; i < nods ; ++i)
                if(getod(&rmask,i))
                    if(!getod(readods,i)) {
                        ch_no = (int )_xiob[i].sys_id;
                        emt(QIW, IO_ACS | SA_USL, SOLUN, SEOFN, 0, 0, 0, 0, 0, 0, ch_no);
                    }

        if(writeods)
            for( i = 0; i < nods ; ++i)
                if(getod(&wmask,i))
                    if(!getod(writeods,i)) {
                        ch_no = (int )_xiob[i].sys_id;
                        emt(QIW, IO_ACS | SA_USL, SOLUN, SEOFN, 0, 0, 0, 0, 0, 0, ch_no);
                    }
169     emt(QIOW, IO_ACS, SA_USL, SOLUN, SOEPN, 0, 0, 0, 0, 0, ch_no);
170 }
171 }
/*

Xsleep(3X) for RSX.
*/

#include <rsxos.h>
xsleep( time )

int time;
{
  emt(MRKT,10,time,2,0);
  emt(WTSE,10);
}
/*
Xsyserr(3X) for RSX - does nothing.
*/

static char xsysmsg[] = "unspecified error";
char *xsyserr()
{
    return( xsysmsg );
}
/*
 * filename: XTERM.C
 * /

#include <rxtos.h>

#define XECHO 1
#define XLINE_EDIT 2
#define XOFF_STERM 0
#define XON_STERM 1
#define TC_NEC 047
#define TC_BIN 065
#define TTYEPN 1

extern int ttyln;
extern int ttyraw;
extern int ttyecho;

int (*_ttyhndlr)() = 0; /* pointer to user handler routine */
extern int _astty(); /* AST routine to service unsolicited ^C */

struct term_char{
  unsigned char name; /* option or characteristics */
  unsigned char value; /* setting */
};

struct iosb {
  unsigned char cc;
  unsigned char lc;
  unsigned short nread;
};

xsetterm(option, on_off)
int option; /* XECHO or XLINE_EDIT */
int on_off; /* 1 == on ; 0 == off */
{
  int rval = 0;
  struct iosb iosb;
  struct term_char t_char;

  t_char.name = ( option == XECHO ) ? TC_NEC : TC_BIN ;
  t_char.value = !on_off;
  if(t_char.name == TC_NEC) ttymt(SF_SMC, ttyln, &iosb, &t_char, sizeof(t_char));
  if(option == XLINE_EDIT) {
    rval = !ttyraw;
    ttyraw = !on_off;
  }
  else {
    rval = ttyecho;
    ttyecho = on_off;
  }
  return ( rval );
}

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        _ttyhdlr = 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    xraw_term( handler )
78    int (*handler)();
79    {
80        int rval;
81        xint_term( handler );
82        rval = xsetterm( XECHO, XOFF_STERM );
83        if ( rval < 0 )
84            {
85                return( rval );
86            }
87        rval = xsetterm( XLINE_EDIT, XOFF_STERM );
88        if ( rval < 0 )
89            {
90                xsetterm( XECHO, XON_STERM );
91                return( rval );
92            }
93        return( 0 );
94    }
#include <rsxos.h>

long xtime()
{
    int buf[8];

    emt(GTIM, buf);  /* return parm:
                     word 0 : --  year
                     word 1 : --  month
                     word 2 : --  day
                     word 3 : --  hour
                     word 4 : --  min
                     word 5 : --  sec
      */

    return( ((buf[2]*24 + buf[3])*60 + buf[4])*60 + buf[5]);
}

Save code space, at the expense of time by providing a common
routine to translate exos open mode flags to RSX mode and types.
*/
#include <xerrno.h>
#include <stdlib.h>
define FO_RD 01
#define FO_WRT 016
#define FO_APD 0106
#include <xspecial.h>

xtranmode( mode, ioflag )
register int mode;
int *ioflag; /* flag to go into _xiob structure */
{
    int rmode;

    /*
    Translate mode to RSX open modes.
    */
    if( mode & XFWRITE )
    {
        if( mode &XFREAD )
            *ioflag = _XIORW | _XPrimary;
        else
            *ioflag = _XIOWRT;
        if( mode & XFAPPEND )
            rmode = FO_APD;
        else
            rmode = FO_WRT;
    }
    else if( mode & XFREAD )
    {
        if( mode & (XFAPPEND | XFCREAT | XFRUNC))
            {
                xerror( XEINVAL, "read and other flags" );
                return( XEINVAL );
            }
        *ioflag = _XIORD | _XPrimary;
        rmode = FO_RD;
    }
    else
    {
        xerror( XEINVAL, "not read or write" );
        return( XEINVAL );
    }
    return( rmode );
}
extern xttyread();
extern xttylabel();
extern xttyclose();
extern struct rcb *rcb[];
#define TTYEFN
#define CNTRLZ 0366

int ttylun = 5; /* lun associated with the TI: */
int ttyraw = 0; /* 1 == raw 0 == line_edit */
int ttyecho = 0; /* 1 == echo on 0 == echo off */

xttyopen( mode )
register int mode;
{
    int exosfd;
    int rmode;
    int ioflag;
    register XFILE *file;
    register struct rcb *sysid = _rcb;
    int i;

    /* Translate mode to Rsx mode and type. */
    rmode = xtranmode( mode, &ioflag);
    if ( rmode < 0 )
        return( rmode );
    for( i=0; i < _XNFILE; ++i, ++sysid )
        if( sysid->flags & RFREE )
            break;
    if( i >= _XNFILE )
        return( XEMFILE ); /* Too many files open */
    sysid->flags = RUSED;
    sysid->r lun = ttylun; /* set LUN OF TI: */
    exosfd = xnewfd(); /* get a free file descriptor */
    if( exosfd < 0 )
        return( exosfd );
    file = & xiob[exosfd];
    file->_file = exosfd;
    file->_flag |= ioflag;
    file->sys_id = sysid;
    file->_read = xttyread;
    file->_write = xttylabel;
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 = ttymct(io_fun, lun, &iosb, buf, len);
85       return(ret);
86    }
87 else {
88      if(ttybuf.tsize == 0) {
89       ret = ttymct(io_fun, lun, &iosb, ttybuf.linetty,132);
90       if(ret < 0)
91          return(ret);
92       if ((int) iosb.cc == CNTRLZ) {
93          stdin-> flag |= XIOEOF;
94          return(0);
95       }
96       ttymct(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
113         ttybuf.tsize = 0;
114     ttybuf.cur_pos = ttybuf.linety;
115 }
116 }
117 return( ret );
118 }
119 }
120 }
121 */
122 /* Objective of this function is to process different type of error resulting
123 * from a call to the driver via QIO ( or emt call in 'C' ) call. A QIO
124 * executive directive call reports error in two different ways through the
125 * DSW ( directive status word ) and also in the IO status block. Again in the
126 * IOSB it is divided into two parts one device specific and the other generic.
127 * The generic and the dsw are returned to the caller after shifting it by -512
128 * and the device specific code is just sign changed. If all is fine then an
129 * non zero value is returned.
130 */
131
132 int ttyemt(cmd, lun, iosb, p1, p2)
133     unsigned short cmd, lun;
134     struct iosb *iosb;
135     unsigned short p1, p2;
136 {
137     int dsw;
138
139     dsw = emt(QIOW, cmd, lun, TTYEFN, iosb, 0, p1, p2, 0, 0, 0, 0);
140     if ( dsw < 0 )
141         return( dsw - 512 ); /* directive error */
142     else
143         return( iosb->nread );
144 }
145 }
146
147
/ *
3  Xunlink for RSX.
4  */
5  #include <xspecial.h>
6  #include <xgenlib.h>
7  
xunlink( name, special )
9  
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  }
1 ;
2 ; FILENAME: ASCBIN.MAC
3 ;
4 ;
5 ; ASCPP: --> convert ascii dir string to binary uic.
6 ;
7 ;  ascpp(pstr, puic)
8 ;  char *pstr; /* INPUT */
9 ;  int  *puic; /*OUTPUT */
10 ;
11 ; .TITLE ASCBIN
12 ; .IDENT /01/
13 ;
14 C$SPRT=0
16 .PSECT C$TEXT,I,RO
17 ASCPP:: ; global reference label
18 .IF DF C$SPRT
19
20 JSR R5,C$SAV ; make it 'C' callable
21 MOV R5,-(SP) ; save C frame pointer
22 MOV 4(R5),R2 ; address of string to be converted
23 MOV 6(R5),R3 ; address of uic
24
25 .ENDC
26
27 CALL .ASCPP ; system lib routine to convert string
28 ; to binary uic.
29 CLR RO ; return status
30 BCC RTN ;
31 MOV #0,R0 ;
32 JMP RTN
33
34 ; PPASC: --> Convert binary uic to ascii dir string
36 ;
37 ;  ppasc(psrt, puic)
38 ;  char *pstr; /* OUTPUT */
39 ;  int  *puic; /* INPUT */
40 PPASC:: ; global reference label
41 .IF DF C$SPRT
42
43 JSR R5,C$SAV ; make it 'C' callable
44 MOV R5,-(SP) ; save C frame pointer
45 MOV 4(R5),R2 ; address of string to be return
46 MOV 6(R5),R3 ; address of uic to be converted
47
48 .ENDC
49
50 MOV #1,R4 ; control code ---
51 ; bit 0 is 1
52 ; bit 1 is 0
53 CALL .PPASC ; system lib routine to convert
54 ; bin uic to string dir
55 CLR RO ; return status
56 BCC BIN
MOV #-1, RO

IF DF C$SPRT

MOV (SP)+, R5 ; adjust frame pointer
JMP C$RET ; return to caller

IFF

RETURN

 ENDC

PSECT C$TEXT, I, RO
.EVEN
.END
; ASTLCONN: --> This is an ast service routine corresponds to directive
; SREX$. It clean up the stack and calls lostconnt to perform
; abnormal termination of task and then exits.

.TITLE ASTLCO
.IDENT /01/
.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$, DSAR$, SETF$
.PSECT C$TEXT, I, RO

;ASTLCO::

.MCALL ASTX$

SAVE ; save all registers

CALL LOSTPEER

CALL XEXIT ; close all the files.

UNSAVE ; unsave all registered

MOV (SP)+, (SP)+ ; clean up stack

ASTX$ ; ast service exit.

;ASTRS::

SAVE ; save all registers

MOV 14(SP), -(SP) ; get iosb address as first parameter

JSR PC, ASTRSELECT ; call ast-select service routine.

TST (SP)+ ; pop off parameter

UNSAVE ; unsave all registered
TST (SP)+
ASTX$S

; pop off stack for ASTX$S
; exit from AST routine

.ASTWS::

SAVE
MOV B 14(SP),-(SP)
JSR PC,ASTWSELECT
TST (SP)+
UNSAVE
TST (SP)+
ASTX$S

; save all registers
; push char as parameter for trstat
; call ast-select service routine.
; pop off parameter
; pop off stack for ASTX$S

.PSEC T C$TEXT,I,RO
.EVEN
.PSEC C$DATA,D,RW
.EVEN
.END
; FILENAME: ASTTY.MAC

; This is an ast service routine corresponds to int_term
; routine( QIO IO_ATA ). It calls the user handler to
; process the char ^C, not by MCR.

.psect c$text,i,ro

.title .astty
.globl .TTYHN

.MCALL ASTX$$

.ASTTY:;
    MOV     R0,-(SP)       ; save R0
    MOV     R1,-(SP)       ; save R1
    MOV     R2,-(SP)       ; save R2
    MOV     R3,-(SP)       ; save R3
    MOV     R4,-(SP)       ; save R4
    MOV     R5,-(SP)       ; save R5
    JSR     PC,@.TTYHN     ; call handler to process interrupt
    MOV     (SP)+,R5       ; pop off R5
    MOV     (SP)+,R4       ; pop off R4
    MOV     (SP)+,R3       ; pop off R3
    MOV     (SP)+,R2       ; pop off R2
    MOV     (SP)+,R1       ; pop off R1
    MOV     (SP)+,R0       ; pop off R0
    TST     (SP)+          ; pop off stack for ast
    ASTX$$     ; exit from AST routine

.END
; FILENAME CHDR
; This file contains start & end entry points

.TITLE CHDR
.MCALL FSRSZ$,GMCR$,DIR$ ,gtsk$s,wtse$s,spwn$s
.PSECRET C$DATA,D,RW

GMCR:

FSRSZ:

;buf: FSRSZ$ 0
;blkw 16. ; used to store task info.
;cmd:
;ascii /REA /
;tsk:
;ascii / /
;lun:
;ascii / SY0:/
;cmdl = .~cmd
;even
cm:
;cli:
.rad50 /MCR.../
.MCALL EXIT$,ALUN$
.PSECRET C$TEXT,1,RO

START:

; make sure task's default device is same as user's login device

gtsk$s #buf ; get task name
bc exit ; if CS error
mov #tsk,r0 ; address of first three byte of task-name
mov buf,r1 ; first word of task-name
call $c5ta ; convert it to ascii
bc exit ; if cs error
mov #tsk+3,r0 ; next three byte of task-name
mov buf+2,r1 ; second word of task-name
call $c5ta ; convert rad50 to ascii
bc exit ; if cs error
mov #4,r3 ; # times REA to be spawned

;AGN:
mov r3,r0 ; LUN #
add #60,r0 ; make it char
movb r0,lun+1 ;
spwn$s #cli,,,,,#cmdl ; spawn REA
bc exit ; if cs error
wtse$s #1 ; wait for task to complete
sob r3,agn ; lopp

; get mcr command line

DIR$ #GMCR ; get command line
MOV $DSW,R1 ; get # of char read or error
BLT ERR ; error
57    CLR     GMCR+2(R1) ; clear terminator char in command line
58  CONT:  
59    MOV     #GMCR+2,-(SP) ; push address of cli buffer
60    CALL    CMAIN ; call start entry points
61    TST     (SP)+ ; pop the parameter
62    BR      EXIT
63 ERR:  
64    CMP     #IE.AST,$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

;pwlog = 22 ; offset of login UIC in password structure
.psect c$data,d,rw
.desc: .blkw 2. ; string descriptor
.uic: .word 0 ; UIC
.task: .blkw 2. ; task name
.buf: .blkw 16. ; used as task info block and emit status block
.psect c$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 px,xstrlen ; get string length
.tst (sp)+ ; pop the argument
.mov #desc,r2 ; r2 get descriptor address
.mov r0, (r2) ; store in descriptor

.mov #uic,r3 ; r3 has address of binary UIC
.call .ascpp ; convert UIC
.gtks$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 UF

 movimiento r4, r0 ; r0 has address of command line string
.mov #1, rl ; accept period as valid RAD50 character
.call $cat5 ; convert to RAD50
.cmp R1,#^RUF ; is it UF?
.beq 10$ ; yes
.cmp task,#^RPTD ; am I a FTP server?
.bne 10$ ; no, must be a client

; I am a FTP server and command to be spawned is not UF

.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 UF

10$:
.mov #^RMCR, task
.mov #^R..., task+2 ; task name is MCR...
20$:

; now spawn the task
mov r4, -(sp) ; get length of command line
jsr pc, xstrle ; reset stack

spawn$ #task,,,uic+1,uic,#1,,#buf,r4,r0 ; spawn task

bcsc 00$ ; spawn error

wtses $1 ; wait for task complete

cmp buf,#1 ; is task OK?

bne 30$ ; no-

clear r0 ; yes, return 0

br 99$ ;

mov buf,r0 ; error in task exit

sub #512, r0 ; return (esb[0] - 512)

br 99$ ;

mov $dsw,r0 ; return (dsw)

99$:

. IF DF C$SPRT

mov (sp)+, r5 ; adjust frame pointer

jmp c$ret

. IFF

RETURN

. ENDC

. END
1 ;
2 ; FILENAME       DBLAST.MAC
3 ;
4 ;        This file contain ast service entry points for i/o
5 ; RASTDIO       AST for read
6 ; WASTDIO       AST for write
7 ; ASTSIO        AST for socket i/o
8 ;
9 ;
10 .TITLE        DBLAST
11 .IDENT         /01/
12 ;
13 ; MODULE        ASTDIO
14 ;
15 ;        AST SERVICE FOR READ & WRITE
16 ;
17 ;
18 .MACRO         SAVE
19 MOV            R0, -(SP)
20 MOV            R1, -(SP)
21 MOV            R2, -(SP)
22 MOV            R3, -(SP)
23 MOV            R4, -(SP)
24 MOV            R5, -(SP)
25 .ENDM
26 .MACRO         UNSAVE
27 MOV            (SP)+,R5
28 MOV            (SP)+,R4
29 MOV            (SP)+,R3
30 MOV            (SP)+,R2
31 MOV            (SP)+,R1
32 MOV            (SP)+,R0
33 .ENDM
34 .MCALL         ASTX$S
35 .PSECt         C$TEXT, I, RO
36 RASTDIO::
37 SAVE
38 MOV            HBUF+10, R0
39 MOV            20(R0), R0
40 MOV            20(R0), R0
41 CMP            F.BKVB(R0), F.EFBK(R0); IT IT LAST BLOCK
42 BGT            SET; IF CT YES
43 CMP            F.BKVB+2(R0), F.EFBK+2(R0); IS IT LAST BLOCK
44 BGT            SET; IF CT YES
45 BR              NEXT
46 SET:
47 MOV            14(SP), R1
48 MOV            F.FFBY(R0), 2(R1); # OF BYTES READ
49 NEXT:
50 MOV            14(SP), -(SP); PUSH IOSB ADDRESS
51 JSR             PC, DSTAT; FILL UP THE RETURN STATUS
52 TST             (SP)+; POP THE PARAMETER
53 UNSAVE          ; UNSAVE ALL THE REGISTERS
54 TST             (SP)+; POP THE STACK FOR ASTX$S
55 ASTX$S          ; RETURN
56 WASTDIO::
57 SAVE 
58 MOV HBUF+10,RO ; SAVE ALL REGISTERS
59 MOV 20(R0),R0 ; get address of xiofb
60 MOV 20(R0),R0 ; get address of .rcb
61 JMP NEXT ; get address of fdb

65 ;
66 ; MODULE ASTSIO
67 ;
68 ; AST SERVICE FOR SOCKET READ/WRITE
69 ;
70 .PSECT C$TEXT,I,RO
71 ASTSIO::
72 SAVE ; SAVE ALL REGISTERS
73 MOV 14(SP),-(SP) ; PUSH ADDRESS OF IOSB
74 JSR PC,NSTAT ; FILL UP THE RETURN STATUS
75 TST (SP)+ ; POP THE PARAMETER
76 UNSAVE ; UNSAVE ALL THE REGISTERS
77 TST (SP)+ ; POP THE STACK FOR ASTX$$
78 ASTX$$
80 .END
.title dummy

gtevnn::
gtethen::
gteta::
gtnbnn::
gttenen::
gtpen::
gtsbp::
gtssen::
gtpbnam::
gtpbnum::

tst pc
.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$,SETF$,ASTX$
28
29  AST.RE::
30       SAVE ; save all registers
31 AGAIN:
32   RCVD$ ,#msge ; recieve pkt from task
33   CMP   #IS.SUC,$D$ ; check for success
34   BEQ   10$ ; If EQ YES
35   BR    EXT ; no pkt. return
36  10$:
37   SETF$ #2 ; go for next pkt.
38   BR    AGAIN
39 EXT:
40   UNSAVE ; unsave all registered
41   ASTX$ ; 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 ;       pl   fdb
24 ;       p2   type 0 -- ascii type otherwise binary
25 ;
26 ;
27 .PSEC  C$DATA,D,RW
28 29
30 CNTG  =  -5
31 ALLOC =  -5
32
33 .PSEC  C$TEXT,I,RO
34 CREATE::
35       JSR  R5,C$SAV            ; make it 'C' callable
36       CMP  6(R5),#0            ; check file-type ascii/binary
37       BEQ  ASC                ; if EQ ASCII
38 BIN:   MOVC  #R.FIX,R1         ; RTYPE AS FIXED LENGTH RECORD
39 CLR   R2                       ; RATT AS NO IMPLIED CR
40 BR    FDAT
41 ASC:   MOV  #R.VAR,R1          ; RTYP AS VARIABLE LENGTH RECORD
42 MOV  #FD.CR,R2                 ; RATT AS IMPLIED CR
43 FDAT:  FDAT$R  4(R5),R1,R2,6(R5),#CNTG,#ALLOC
44       JMP  C$RET             ; INITIALIZE ATTRIBUTE SECTION OF FDB
45       JMP  C$RET             ; JUMP TO RETURN
46 47 ; MODULE  OPEN
48 49 ;    INPUT PARAMETERS
57 ;
58 ;
P1   FDB
59 ;
P2   MODE OF FILE
60 ;
P3   LUN
61 ;
P4   DATA SET POINTER
62 ;
63 
64 .PSEC C$TEXT,I,RO
65 .OPEN::
66 JSR R5,C$SAV
67 OPEN$ 4(R5),6(R5),10(R5),12(R5),#PD.RWM,,EOPEN
68 CLR RO
69 JMP C$RET
70 EOPEN:
71 MOV 4(R5),R1
72 MOVB F.ERR(R1),RO
73 JMP C$RET
74 
75 ;
76 ; MODULE READ
77 ;
78 ; INPUT PARAMETERS
79 ;
80 ;
P1   FDB
81 ;
P2   BLOCK BUFFER ADDRESS
82 ;
P3   BKEF
83 ;
P4   ADDRESS OF IOSB
84 ;
P5   ADDRESS OF AST
85 ;
86 ;
87 
88 
89 BLKSIZE=512.
90 .PSEC C$TEXT,I,RO
91 .READ::
92 JSR R5,C$SAV
93 READ$ 4(R5),6(R5),#BLKSIZE,,10(R5),12(R5),14(R5),CKEOF
94 BCS CKEOF
95 MOV #1,RO
96 JMP C$RET
97 ERI O:
98 MOV 4(R5),R1
99 MOVB F.ERR(R1),RO
100 SUB #512.,RO
101 JMP C$RET
102 
103 CKEOF:
104 MOV 4(R5),R1
105 CMPB F.ERR(R1),#IE.EOF
106 BNE ERI O
107 CLR RO
108 JMP C$RET
109 
110 ;
111 ;
112 ; MODULE WRITE
113 ;
114 ; 
115 ;
116 ; 
117 ;
118 ;
119 .PSEC$ 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$RET ; JUMP TO RETURN
126
127
128
129
130
131 ;
132 ; MODULE RWAIT
133 ;
134 ; 
135 ;
136 ;
137 ;
138 ;
139 ;
140 .PSEC$ 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 CT YES
150 CMP F.BKVB+2(R1),F.EFBK+2(R1); IS IT LAST BLOCK
151 BGT SETIO ; IF CT 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  .PSEG  C$TEXT,I,RO
182  
183  ..WWAIT::
184  
185  JSR R5,C$SAV   ; MAKE IT 'C' CALLABLE
186  WAIT$ 4(R5),,,ERW
187  CMPB @6(R5),#0   ; CHECK ERROR
188  BLT ERW       ; IF LT YES, ERROR WHILE WRITE
189  JMP C$RET     ; JUMP TO RETURN
190  
191  
192  ;
193  ;
194  ; MODULE CLOSE
195  
196  ;   INPUT PARAMETERS
197  
198  ;   P1  fdb
199  ;
200  
201  
202  .PSEG  C$TEXT,I,RO
203  ..CLOSE::
204  
205  JSR R5,C$SAV   ; MAKE IT 'C' CALLABLE
206  CLOSE$ 4(R5)    ; 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  
225  JSR R5,C$SAV   ; MAKE IT 'C' CALLABLE
MOV 4(R5),R1 ; GET fdb
MOV 6(R5),F.RSIZ(R1) ; set max rec size
MOV 10(R5),F.FFBY(R1) ; set first free byte
CMP 10(R5),#0 ; check first free byte is 0
BEQ NEXT1 ; If EQ yes
DEC F.EFBK+2(R1) ; end of block number
NEXT1:
JMP C$RET ; JUMP TO RETURN
1 ;
2 ; FILENAME LIBMAC.MAC
3 ;
4 ;
5 .TITLE LIBMAC
6 .MCALL ASTX$S ,EXIT$S
7 .PSECT C$TEXT,I,RO
8 ;
9 ; this routine is the AST service routine specified in the catchoob()
10 ; library call. After it is invoked it simply passes control to another
11 ; library routine called libast() which selectively calls user specified
12 ; handler. The address of the iosb being on top of the stack is automatic-
13 ; cally passed to the libstat routine.
14 ;
15 ;
16 ;
17 ;
18 .ASTCA:: ; global reference
19 
20 JSR PC,LIBAST ; call library routine LIBSTAT
21 TST (SP)+ ; pop off stack to adjust for the ASTX call
22 ASTX$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
filename: MUXAST.MAC

.title MUXAST

This file contains the two ast service routines that are
called by the system when read is completed on either the network
or the terminal. These in turn call C routines that set the return
status for the net_read and net_write "processes".

.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$S,DSAR$S,SETF$S

ASTRD1::
SAVE ; save all registers
MOV 14(SP),-(SP) ; get iosb address as first parameter
JSR PC,NRSTAT ; fill up the return status
TST (SP)+ ; pop off parameter
UNSAVE ; unsave all registered
TST (SP)+ ; pop off stack for ASTX$S
ASTX$S ; exit from AST routine

ASTRD2::
SAVE ; save all registers
MOVB 14(SP),-(SP) ; push char as parameter for trstat
JSR PC,TRSTAT ;
TST (SP)+ ; pop off parameter
UNSAVE
TST (SP)+ ; pop off stack for ASTX$S
ASTX$S

.END
This routine parses the command string by using CSI\$/ specific
rituals. It parses the input command line and stores the
return values in CSI control block, which may be directly used
by File open routines.
Following is format of command string:
dev:[g,m]filespec
To parse the command string, it has three major functions:
i) Allocate CST control block
ii) Syntax validation of command
iii) Semantic check

;INPUT:
parse(buf,len)
char *buf;
int len;

;OUTPUT:
0  -- successful completion
relevant information in CSI control block
1  -- unsuccessful.

.TITLE PARSE
.IDENT /01/
C$SPRT=0
.PSECT C$DATA,D,RW
.MCALL CSI$1,CSI$1,CSI$2

CSI$  DEFG  ; define CSI control block offsets
       ; and bit values globally.
.EVEN
CSIBL.K:
.BLKB  C.SIZE  ; allocate required storage

PARSE::
       .IF DF C$SPRT

JSR  R5,C$SAV  ; make it 'C' callable
MOV  R5,-(SP)  ; save C frame pointer
MOV  4(R5),R2  ; get address of command string
MOV  6(R5),R3  ; length of command string

.ENDC

MOV  #CSI.BL,CSIBL.K  ; store the address of CSI control block
                    ; for accessing in 'C' code.

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$TEXT,I,RO
79  .EVEN
80  .END
1 ;
2 ; FILENAME:    RADIUS MAC
3 ;
4
5 ; RADIUS: --> Converts array of 6 char into 2 word radix 50 format
6 ;
7
8 .TITLE RADIUS
9 .IDENT /01/
10 C$SPRT=0
11 .PSECT C$DATA,D,RW
12 TMP:
13 .WORD 0
14 .PSECT C$TEXT,I,RO
15 RADIUS:: ; global reference label
16 .IF DF C$SPRT
17
18   JSR      R5,C$SAV ; make it 'C' callable
19   MOV      R5,-(SP) ; save C frame pointer
20   MOV      4(R5),R0 ; address of first char
21
22 .ENDC
23
24   MOV      #1,R1                   ; '.' is a valid ascii char for conversion
25   CALL     CAT5B                   ; convert 3 ascii char to radix 50(consider ' ')
26   BCS      FAIL                    ; check for success
27   MOV      R1,TMP                  ; save converted value
28   MOV      #1,R1                   ; '.' is a valid char(consider ' ' too as valid)
29   CALL     CAT5B                   ; convert next 3 ascii char to radix 50
30   BCS      FAIL                    ; check for success
31 ;;
32   MOV      R1,R0                   ; return value
33   MOV      TMP,R0                  ; return value
34   JMP      RTN                    ;
35 FAIL:
36   CLR      R0                     ;
37   CLR      R1                     ;
38 RTN:
39 .IF DF C$SPRT
40
41   MOV      (SP)+,R5                ; adjust frame pointer
42   JMP      C$RET                  ; return to caller
43
44 .IFF
45
46 RETURN
47
48 .ENDC
49
50 .PSECT C$DATA,D,RW
51 .EVEN
52 .PSECT C$TEXT,I,RO
53 .EVEN
54
55 ; C5TA:
56 ; Converts 16bit rad50 value to ascii string
57 ;
58 ;INPUT
59 ;   pl = 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
Setjmp & longjmp are only callabe from 'C'
Floating point register's are not saved.

Environment:

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Old FP</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>R4</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>R3</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>-------</td>
</tr>
</tbody>
</table>

Scratch cell used by 'C' compiler

.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$: 
MOV   #1,R0       ; UNSUCCESSFUL VALUE
MOV   (SP)+,R5    ; adjust frame pointer
JMP   C$RET       ; return to caller

.PSECT   C$TEXT,I,RO

XLONGJMP::

JSR   R5,C$SAV    ; make it 'C' callable
MOV   R5,-(SP)    ; save C frame pointer

TST   @4(R5)      ; CHECK ADDRESS OF ENV.
BEQ   30$         ; IF EQ ADDRESS IS NULL
MOV   @4(R5),R1   ; GET ADDRESS OF ENV. IN REG 1
MOV   #7,R3       ; WORD COUNT

MOV   (R1)+,-(SP) ; PUSH ENV. FROM HEAP TO STACK
SUB   R3,20$      ; LOOP FOR 7, TIMES.
MOV   SP,R2       ; REMEMBER THE ADDRESS OF SAVED ENV.
MOV   #0,-(SP)    ; DUMMY PARM.
MOV   @4(R5),-(SP) ; PUSH ADDRESS OF ENV.
CALL  XFREE       ; 'C' RUN-TIME ROUTINE TO DEALLOCATE
                 ; ROUTINE
TST   (SP)+       ;
TST   (SP)+       ; POP THE PARAMETER
CLR   @4(R5)      ; CLEAR THE POINTER
MOV   (R2),R5     ; LOAD THE FRAME POINTER CORRESPOND TO
                 ; SETJMP.
MOV   R5,R1       ;
SUB   #12,R1      ; LOAD THE LOW ADDRESS OF ENV.
MOV   #7,R3       ; #WORD COUNT

MOV   (R2)+,(R1)+ ;
SUB   R3,25$      ; LOOP

MOV   #1,R0       ; RETURN VALUE
BR    40$

MOV   #1,R0       ; UNSUCCESSFUL RETURN VALUE

MOV   (SP)+,R5    
JMP   C$RET

.PSECT   C$TEXT,I,RO

.EVEN

.END
.title xspawn
.psect c$text,i,ro

.MCALL CLEF$S,SETF$S

XSPAWN::
.jsr R5,c$sav

CLEF$S #50. ; clear efn 50
SETF$S #51. ; set efn 51 to unstop MST

jmp c$ret

.END
.title xttywrite

xttywrite(sysid, buf, len)

sysid - address of RSX - control block
buf - buffer address 6(r5)
len - buffer length 10(r5)

.mcall qiow$,dir$
.psect c$data,d,rw

TTYEFN = 1
CSPRT = 0

.define offsets used for variables

.RCB = 4
RLUN = 2
FLAGS = 4
DBLBUF = 40
BUF = 6
LEN = 10
BUFSIZE = 512.
LOCBUF = -6 - BUFSIZE

iosb: .blkw 2.
count: .word 0
extra: .word 0
iocal: qiow$ IO.WVB,0,TTYEFN,,iosb
.psect c$text,i,ro

xttywrite:

IFDEF CSPRT
.jsr r5,c$sav ; make it 'C' callable
.add #LOCBUF, sp ; allocate local space on stack
.ENDIF

mov .RCB(r5),r0 ; get address of RCB
mov RLUN(r0), iocal+Q.IOLU ; save LUN
.tst ttyraw ; Is this write using raw mode
.beq 10$ ; no-

.raw mode I/O

mov BUF(r5), iocal+Q.IOPL ; buffer address
mov LEN(r5), iocal+Q.IOPL+2 ; buffer length
dir$ #iocal ; directive call
.jsr pc, ret ; handle return value
.br 999$ ; return

Non-raw mode I/O

10$:

mov r5, Q.IOPL+iocal
.add #LOCBUF, Q.IOPL+iocal ; set up output buffer address
.clr count ; clear byte count
.clr extra ; clear extra byte count
.mov LEN(r5), r1 ; r1 - number of bytes to output
.mov r5, r2 ; r2 - end of local buffer
add  #r-6, r2 ; adjust offset for buffer size
mov  BUF(r5), r3 ; r3 - pointer to input buffer
mov  r5, r4 ; r4 - pointer to local buffer
add  #LOCBUF, r4 ; adjust for offset

15$:
tst  r1 ; any more bytes to output
beq  50$ ; no
dec  r1 ; decrement output count
cmp  r4, r2 ; check if end of local buffer reached?
bne  20$ ; not yet
76 ;
20$:
    ; reached end of local buffer
mov  #BUFSIZE, Q.IOPL+2+iocal ; output buffer size
dir$  #iocal ; output the buffer
jsr  pc,ret
add  r0,count ; update output count
mov  r5, r4 ; reset pointer to local buffer
add  #LOCBUF, r4

20$:
    ; stuff character into output buffer
mov  .RCB(r5),r0 ; get RCB address
bit  #DBLBUF,FLAGS(r0); is buffer from network
bne  30$ ; if NE yes
cmpb  (r3),#12 ; is input a \n
bne  30$ ; no-
movb  (r3)+, (r4)+ ; put LF CR into output buffer
movb  #15, (r4)+
inc  extra ; increment extra character count
br  15$ ; try next character
30$:
    ; regular character
movb  (r3)+, (r4)+
br  15$
50$:
    ; finish with all the output processing, flush last buffer
sub  #LOCBUF, r4 ; calculate buffer size
sub  r5, r4
mov  r4, Q.IOPL+2+iocal ; set up the buffer size
dir$  #iocal
jsr  pc,ret ; process return value
add  r0, count
mov  LEN(r5), r0
add  extra, r0
cmp  count, r0 ; are all characters sent out?
bne  60$ ; no-
ymov  LEN(r5), count ; yes, return original length of buffer
60$:
mov  count, r0 ; return count
999$:
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$: mov $dsw, r0 ; error, return DSW
128 rts pc
129
130 .END
/*
@(#)arp.h 1.1 3/26/85

Include files for the ARP program on RSX.
*/

#include <xgenlib.h>
#include <xspecial.h>
#include <in.h>
#include <socket.h>
#include <brdioct1.h>
#include <exiocmd.h>
#include <types.h>
#include <netdb.h>
#include <hostarp.h>
define ether_aton etr_aton
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 device */
#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 BRDEDELR (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
 */

/*
 * 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
/*
 * 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 SOKTADDR (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 BRDSSSTAT /* 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 BREDARP /* 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 SOSELEWAKEUP (80)
#define SOHASOOG (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 REPLY_OK 0x00 /* all is well */

#define NM_MAGIC_DATA 0x80
#define MQ_EXOS 0x01 /* exos own Q element */
57  #define MQ_DONE  0x02  /* exos done with Q elemnt*/
58  #define MQ_OVERFLOW  0x04  /* data are too big */
59
60
/* These are the DIC and DPB lengths of the Executive directives */
#define QIO 06001
#define QIOW 06003
#define ALUN 02007
#define WTSE 01051
#define CTIM 01075
#define SPWN 06413
#define SDRS 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_DRT 002000 # detach fn: made no-op #

/*

#define IO_EXC 002400 /* EXOS board admin. operation */
define _EX_INI BRDINIT /* Reset and configure EXOS */
define _EX_CNFG BRDCONF /* get configuration msg */
define _EX_STR BRDSSTART /* Execute EXOS procedure */
define _EX_STS BRDGSTAT /* Read network statistics */
define _EX_SAR BRDGARP /* set up an ARP table entry */
define _EX_GAR BRDGARP /* Retrive an ARP table entry */
define _EX_DAR BRDDELRT /* Delete an ARP table entry */
define _EX_ART BRDDISPRT /* Add an Routing table entry */
define _EX_DRT BRDDELRT /* Delete an RT entry */
define _EX_SRT BRDSTRT /* 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 /* shutdown r/w on socket */
#define SO_SKP SIOCKEEP /* set keep alive */
#define SO_GKP SIOCKEEP /* inspect keep alive */
#define SO_SLG SIOCSSLINGER /* set linger time */
#define SO_GLG SIOCGLINGER /* get linger time */
#define SO_SOB SIOCSENDTOOB /* send out of band */
#define SO_ROB SIOCRCVOOB /* receive out of band */
#define SO_AMK SIOCATMARK /* at oob mark */
#define SO_SPG SIOCSPCRP /* set process group */
#define SO_GPC SIOCGPCRP /* get process group */
#define SO_NRD FIONREAD /* FIONREAD */
#define SO_BNO FIONBIO /* FIONBIO */
#define SOASY 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 sockaddr sp; /* socket protocol (optional) */
    int type; /* socket type */
    int options; /* options */
    /* these are for select () */
    int nfds;
    long *wp;
long *rp;
long timo;
};
/* unsigned data types (shorthand) */
typedef unsigned int Uint;
typedef long Ulong;
typedef unsigned short Ushort;
typedef char Uchar;
struct _rcb {
    int mode;  /* File type */
    int rlun;  /* RSX LUN */
    int flags; /* Flags -- described below */
    char *bptr; /* pointer to block buffer */
    char *bnptr; /* next position in block */
    int bleft; /* bytes left in block read/write*/
    char *rptr; /* pointer to record buffer */
    char *rnptr; /* next position in record */
    char *fdb; /* pointer to FDB */

    union {
        int rleft;  /* char left to be read */
        int rsize;  /* max. record size */
    } rec;
};

struct dblbuf {
    int stat[2];  /* status of i/o buffer */
    /* 0 -- EOF */
    /* < 0 -- I/O Error */
    /* > 0 -- Bytes transferred */
    char *buffer[2];
    XFFILE *fd;  /* pointer to file descriptor */
    char active; /* buffer used for IO */
};

struct iosb {
    unsigned char cc;
    unsigned char lc;
    int nread;
};

#define RFREE  01
#define RUSED  02
#define REOF   04
#define REOLN  010
#define DBLBUF 040
#define RCRFLAG 020
#define SOEFN   11
#define DISKEFN 12
#define FDBSIZE 0140
#define F_FF8Y  014
#define F_RSIZ  02
#define F_BKV8  064
#define RECSIZE 256
#define BLKSIZE 512
#define MAXPRM  20
#define C_DDS0  06
#define IO XFR 003400  /* data transfer stream */
#define IX RDS 0000  /* read from TCP stream */
#define IX WRDS 00001 /* write to TCP stream */
#define SO LUN 20  /* EXOSO LUN */
/* Definitions for FTP */
/* See RFC-765 */

 ifndef MAXPATHLEN
 define MAXPATHLEN 1024
 endif
 ifndef CTRL
 define CTRL(x) 037&x
 endif
 ifndef SIGCHLD
 define SIGCHLD SIGCLD
 endif

 /* Reply codes. */

 define PRELIM 1 /* positive preliminary */
define COMPLETE 2 /* positive completion */
define CONTINUE 3 /* positive intermediate */
define TRANSIENT 4 /* transient negative completion */
define ERROR 5 /* permanent negative completion */

 /* Type codes */
define TYPE_A 1 /* ASCII */
define TYPE_E 2 /* EBCDIC */
define TYPE_I 3 /* image */
define TYPE_L 4 /* local byte size */

 /* Form codes */
define FORM_N 1 /* non-print */
define FORM_T 2 /* telnet format effectors */
define FORM_C 3 /* carriage control (ASA) */

 /* Structure codes */
define STRU_F 1 /* file (no record structure) */
define STRU_R 2 /* record structure */
define STRU_P 3 /* page structure */

 /* Mode types */
define MODE_S 1 /* stream */
define MODE_B 2 /* block */
define MODE_C 3 /* compressed */

 /* Record Tokens */
57 #define REC_ESCAPE '\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 */
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 */
#endif
#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 argc;      /* count of arguments on input line */
extern char **argv;   /* 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    extern    char    *tail();
63    extern    char    *remglob();
64    extern    int    errno;
/* @(#)host_arp.h 1.3 5/14/85 */

Address structures for ARP ioctl calls

/*

Ethernet Address

*/

struct etr_addr {
    short ea_family;               /* to match sockaddr structure */
    char ea_addr[6];              /* interesting part */
    char ea_extra[8];             /* to match sockaddr structure */
};

/*

Count (for retrieving entire table)

*/

struct next_addr {
    short ntxt_family;            /* to match sockaddr structure */
    long ntxt_count;              /* interesting part */
    char ntxt_extra[10];          /* to match sockaddr structure */
};
/* @(#)in.h       1.3 4/12/85 */

/* GAP 1/11/85: WARNING - 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 IPROTO_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 IPPORT_DISCARD 9
#define IPPORT_SYSTAT 11
#define IPPORT_DAYTIME 13
#define IPPORT_NETSTAT 15
#define IPPORT_FTP 21
#define IPPORT_TELNET 23
#define IPPORT_SMTP 25
#define IPPORT_TIMESERVER 37
#define IPPORT_NAMESERVER 42
#define IPPORT_WHOIS 43
#define IPPORT_NTP 57

/*
 * Port/socket numbers: host specific functions */
#define IPPORT_TFTP 69
#define IPPORT_RJE 77
#define IPPORT_FINGER 79
#define IPPORT_TTYLINK 87
#define IPPORT_SUPDUP 95

/*
 * UNIX TCP sockets */
#define IPPORT_EXECSERVER 512
#define IPPORT_LOGINSERVER 513
#define IPPORT_CMDSERVER 514
/* UNIX UDP sockets */
#define IPPROTO_BIFFUDP 512
#define IPPROTO_WhoServer 513

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

/* Link numbers */
#define IMPLK_IP 155
#define IMPLK_LOWEXPER 156
#define IMPLK_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 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 */
#endif

#define IN_CLASSA 0x00800000L
#define INCA_NET 0x00ff0000L  /* 8 bits of net # */
#define INCA_LNA 0xff00ffffL
#define INCB 0x00400000L
#define INCB_NET 0xff000000L  /* 16 bits of net # */
#define INCB_LNA 0x0000ffffL
#define INCC_NET 0xff00ffffL  /* 24 bits of net # */
#define INCC_LNA 0x0000ff00L
#endif

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

#define VAX
#define IN_CLASSA 0x00000080
#define INCA_NET 0x000000ff  /* 8 bits of net # */
#define INCA_LNA 0xffffffff
#define INCB 0x00000040
#define INCB_NET 0x0000ffff  /* 16 bits of net # */
#define INCB_LNA 0xffffffff
#define INCC_NET 0x00ffff00  /* 24 bits of net # */
#define INCC_LNA 0x00000000
#endif

#define PDP11
#define IN_CLASSA 0x00800000L
#define INCA_NET 0x00ff0000L  /* 8 bits of net # */
#define INCA_LNA 0xff00ffffL
#define INCB 0x00400000L
#define INCB_NET 0xffffffffL  /* 16 bits of net # */
#define INCB_LNA 0x0000ffffL
#define INCC_NET 0xffffffffL  /* 24 bits of net # */
#define INCC_LNA 0x0000ff00L
#endif

#define I8086
#define IN_CLASSA 0x00000080
#define INCA_NET 0x000000ff  /* 8 bits of net # */
#define INCA_LNA 0xffffffff
#define INCB 0x00000040
#define INCB_NET 0xffffffffL  /* 16 bits of net # */
#define INCB_LNA 0xffffffff
#define INCC_NET 0x00ffff00  /* 24 bits of net # */
#define INCC_LNA 0x00000000
#endif

#define M68000
#define IN_CLASSA 0x80000000L
#define INCA_NET 0xff000000L  /* 8 bits of net # */
#define INCA_LNA 0x00fffffL
#define INCB 0x80000000L
#define INCB_NET 0xffffffffL  /* 16 bits of net # */
#define INCB_LNA 0xffffffff
#define INCC_NET 0xffffffffL  /* 24 bits of net # */
#define INCC_LNA 0x00000000L
#endif

#define Z8000
#define IN_CLASSA 0x80000000L
#define INCA_NET 0x00ff0000L  /* 8 bits of net # */
#define INCA_LNA 0xffffffffffL
#define INCB 0x40000000L
#define INCB_NET 0xffffffffL /* 16 bits of net # */
#define INCB_LNA 0x0000ffffffL
#define INCC_NET 0xfffffffff00L /* 24 bits of net # */
#define INCC_LNA 0x00000000fL
#endif
#endif ONBOARD /* board make does not define MACHINE type */
#define IN_NETOF(in) 
 ((in).s_addr&IN_CLAS) == 0 ? (in).s_addr&INCA_NET : 
 unbelievable(INCA_NET) 
 (in).s_addr&INCC_NET) 
#define IN_LNAOF(in) 
 ((in).s_addr&IN_CLAS) == 0 ? (in).s_addr&INCA_LNA : 
 unbelievable(INCA_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];
};

#ifndef 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 mode (0) */
    char im_hdfo[2]; /* host data format option */
    char im_junk[3]; /* host address mode */
    char im_addrmode;
    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_10loff; /* movable block offset */
    short im_10lseg; /* movable block segment */
    char im_nproc; /* number of exos 10l processes */
    char im_nmb; /* number of exos 10l 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
/* @(#)netdb.h 1.3 3/25/85 */

/* Structures returned by network data base library. All addresses are supplied in host order, and returned in network order (suitable for use in system calls). */

struct hostent {
    char *h_name; /* official name of host */
    char **h_aliases; /* alias list */
    int h_addrtype; /* host address type */
    int h_length; /* length of address */
    char *h_addr; /* address */
};

/* Assumption here is that a network number fits in 32 bits -- probably a poor one. */

struct netent {
    char *n_name; /* official name of net */
    char **n_aliases; /* alias list */
    int n_addrtype; /* net address type */
    long n_net; /* network # */
};

struct servent {
    char *s_name; /* official service name */
    char **s_aliases; /* alias list */
    unsigned short s_port; /* port # */
    char *s_proto; /* protocol to use */
};

struct protoent {
    char *p_name; /* official protocol name */
    char **p_aliases; /* alias list */
    int p_proto; /* protocol # */
};

#define gethostname gethhostname
#define g HBOADR gethboaddr
#define gethoven gethent
#define sethostname sethent
#define endhostent endhent

#define getnbyname getnbyname
#define gnbboadr getnbboaddr
#define getnetent getnetent
#define setnetent setnetent
#define endnetent endnetent

#define gbsnename getsbname
#define gssboport getsbsoport
#define getservent getservent
57 #define setservent setsent
58 #define endservent endsent
59
60 #define getpbname gpbnname
61 #define getpbnumber gpbnnumber
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();
/ * @(#)route.h 1.6 5/7/85 */

/*
 * GAP 1/11/85: WARN ING - 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.
 */

/*
 * Kernel resident routing tables.
 *
 * The routing tables are initialized at boot time by
 * making entries for all directly connected interfaces.
 * Routing daemons can thereafter update the routing tables.
 *
 * TODO:
 * keep statistics
 */

/*
 * A route consists of a destination address and a reference
 * to a routing entry. These are often held by protocols
 * in their control blocks, e.g. inpcb.
 */

struct route {
    struct rtentry *ro_rt;
    struct sockaddr ro_dst;
    #ifdef notdef
    caddr_t ro_pcb; /* not used yet */
    #endif
};

#define KERNEL

/*
 * The route "routetoif" is a special atom passed to the output routines
 * to implement the SO_DONTROUTE option.
 */

struct route routetoif;

/*
 * We distinguish between routes to hosts and routes to networks,
 * preferring the former if available. For each route we infer
 * the interface to use from the gateway address supplied when
 * the route was entered. Routes that forward packets through
 * gateways are marked so that the output routines know to address the
 * gateway rather than the ultimate destination.
 *
 * AA - 4/11/85: The rtentry structure below has been set up
 * so that it it compatible with the host, board
 * and machines such as VAX that like
 * to do long alignments.
 * !!! DO NOT FIDDLE WITH THIS STRUCTURE UNLESS YOU
 * !!! UNDERSTAND THIS
 */

struct rtentry {
    struct sockaddr rt_dst; /* key */
struct sockaddr rt_gateway; /* value */
struct rtentry *rt_next; /* next pointer */
#endif
short dummy; /* host ptr=4; board ptr =2 */
#endif xenix286
short dummy; /* host ptr=4; board ptr =2 */
u_long rt_use; /* raw # packets forwarded */
struct ifnet *rt_ifp; /* the answer: interface to use */
#endif PDP11
short dummyx; /* host ptr=4; board ptr =2 */
#endif xenix286
short dummyx; /* host ptr=4; board ptr =2 */
char rt_flags; /* up/down?, host/net */
char rt_refcnt; /* # held references */
#endif xenix286
u_short rt_hash; /* to speed lookups */
#endif
#define RTHASHSIZ 7
#endif ONBOARD
struct rtentry *rthost[RTHASHSIZ] = 0;
struct rtentry *rtnet[RTHASHSIZ] = 0;
#endif
#define RTF_UP 0x1 /* route useable */
#define RTF_GATEWAY 0x2 /* destination is a gateway */
#define RTF_HOST 0x4 /* host entry (net otherwise) */
#define RTFREE(rt) 
    if ((rt)->rt_refcnt == 1) 
        rtfree(rt); 
    else 
        (rt)->rt_refcnt--;
/*
 * These are the DIC and DPB lengths of the Executive directives
 */

#define RSX 1

#define QIO 06001
#define QIOW 06003
#define ALUN 02007
#define WTSE 01051
#define GTIM 01075
#define SPWN 06413
#define SDRF 03615
#define SDAT 02507
#define STOP 0603
#define STSE 01207
#define RCVD 02113
#define RCVX 02115
#define RCSV 02213
#define MRKT 02427
#define GTSK 01077
#define SREX 01647
#define EXST 01035
#define USTP 01605
#define SETF 01041
#define CLEF 01037
#define ENAR 0545
#define DSAR 0543
#define DSCP 0537
#define ENCP 0541
#define GLUN 01405
#define RQST 03413

/*
 * QIO function codes
 */

#define IO_RLB 01000
#define IO_RVB 010400
#define IO_RTT 05001
#define IO_WVB 011000
#define IO_DET 002000
#define IO_KIL 000012
#define IO_WLB 00400
#define IO_KAL 01410
#define SF_SMC 02440
#define SF_GMC 02560
#define TC_FDX 064
#define TC_ACR 024

/* Executive return status */

#define IS_CLR 00 /* event was clear */
#define IS_SUC 01 /* operation successful */
#define IS_SET 02 /* event flag was set */

#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_ACT -07 /* task not active */
# 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_OPL -65 /* device offline */

/* CSI CONTROL BLOCK OFFSETS AND BIT VALUES DEFINITIONS */

#define CS_DIFF 02
#define CS_DVF 04
#define CS_EQUI 040
#define CS_INP 01
#define CS_MOR 020
#define CS_NMF 01
#define CS_OUT 02
#define CS_WLD 010
#define C_CMLD 02
#define C_DEVD 06
#define C_DIRD 012
#define C_DSDS 06
#define C_FILD 016
#define C_MKW1 024
#define C_MKW2 026
#define C_STAT 01
#define C_SWAD 022
#define C_TYPRI 00
#define A_CRIP 00
#define A_MBR 03
#define A_LPRVI 074
#define A_SYDIV 056
#define TF_RAL 010

/*
 C Portable routines.
 */

/*
 MCR relative parameters
 */

#define CMDSIZE 60

/*
 terminal Input buffer structure
 */

struct ttybuf {
    char linetty[132];
    char *cur_pos;
    int tsize;
};

#define rsx
#define SCRATCHFILE  "."
/* @(#)socket.h 1.8 7/29/85 */
/*
socket.h 4.16 82/06/08 */
/
/
* GAP 1/11/85: WARNING - 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.
*/

#define 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 swap  ex_swap

#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_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_REUSEADDR  0x40 /* permit local port ID duplication */
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
68 struct sockproto {
69     short sp_family; /* protocol family */
70     short sp_protocol; /* protocol within family */
71 }
72
73 #define PF_UNSPEC 0 /* unspecified */
74 #define PF_UNIX 1 /* UNIX internal protocol */
75 #define PF_INET 2 /* internetwork: UDP, TCP, etc. */
76 #define PF_IMPLINK 3 /* imp link protocols */
77 #define PF_PUP 4 /* pup protocols: e.g. BSP */
78 #define PF_CHAOS 5 /* mit CHAOS protocols */
79 #define PF_OISCP 6 /* ois communication protocols */
80 #define PF_NBS 7 /* nbs protocols */
81 #define PF_ECMA 8 /* euramerican computer manufacturers */
82 #define PF_DATAKIT 9 /* datakit protocols */
83 #define PF_CCITT 10 /* CCITT protocols, X.25 etc */
84
85 /*
86 * Generic socket address format.
87 *
88 * Each process is also operating in an address family, whose
89 * addresses are assigned unless otherwise requested. The address
90 * family used affects address properties: whether addresses are
91 * externalized or internalized, location dependent or independent, etc.
92 * The address can be defined directly if it fits in 14 bytes, or
93 * a pointer and length can be given to variable length data.
94 * We give these as two different structures to allow initialization.
95 */
96
97 struct sockaddr {
98     short sa_family; /* address family */
99     char sa_data[14]; /* up to 14 bytes of direct address */
100 }
101
102 /*
103 * The first few address families correspond to protocol
104 * families. Address families unrelated to protocol families
105 * are also possible.
106 */
107 #define AF_UNSPEC 0 /* unspecified */
108 #define AF_UNIX 1 /* local to host (pipes, portals) */
109 #define AF_INET 2 /* internetwork: UDP, TCP, etc. */
110 #define AF_IMPLINK 3 /* arpanet imp addresses */
111 #define AF_PUP 4 /* pup protocols: e.g. BSP */
112 #define AF_CHAOS 5 /* mit CHAOS protocols */
113 #define AF_OISCP 6 /* ois communication protocols */
114 #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 AFETHER 11      /* Ethernet Address */
#define AF_COUNT 12     /* A count */
#define AFETYPEFILTER 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 SIOCRCVOb    (6) /* get out of band */
#define SIOCATMARK   (7) /* at out of band mark? */
#define SIOCSPG RP   (8) /* set process group */
#define SIOCGRP       (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 */
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 NOPREESOCKET -13
typedef struct { int r[1]; } * physadr;
typedef long daddr_t;
typedef char * caddr_t;
typedef unsigned long mem_t;
typedef unsigned short ushort;
typedef unsigned char uchar_t;
typedef ushort ino_t;
typedef short cnt_t;
typedef long time_t;
typedef long label_t[13]; /* regs d2-d7, a2-a7, pc */
typedef short dev_t;
typedef long off_t;
typedef long paddr_t;
/*
@(#) xctype.h  1.3 5/31/85
character mappings.
*/

#define _U 01
#define _L 02
#define _N 04
#define _S 010
#define _P 020
#define _C 040
#define _B 0100
#define _X 0200

extern char _xctype[];

#define isalpha(c)  ((_xctype+1)[c] & (_U | _L))
#define isupper(c)  ((_xctype+1)[c] & _U)
#define islower(c)  ((_xctype+1)[c] & _L)
#define isdigit(c)  ((_xctype+1)[c] & _N)
#define isxdigit(c) ((_xctype+1)[c] & _X)
#define isspace(c)  ((_xctype+1)[c] & _S)
#define ispunct(c)  ((_xctype+1)[c] & _P)
#define isalnum(c)  ((_xctype+1)[c] & (_U | _L | _N))
#define isprint(c)  ((_xctype+1)[c] & (_P | _U | _L | _N | _B))
#define isgraph(c)  ((_xctype+1)[c] & (_P | _U | _L | _N))
#define iscntrl(c)  ((_xctype+1)[c] & _C)
#define isascii(c)  ((unsigned)(c) <= 0177)
#define toupper(c)  ((islower(c)) ? (c) - 'a' + 'A' : (c))
#definetolower(c)  ((isupper(c)) ? (c) - 'A' + 'a' : (c))
#define toascii(c)  ((c) & 0177)
/*
@(#)xerrno.h 1.3 3/25/85
Error values for xgenlib and xoslib.
Some of these errors will make little sense on non-Unix systems.
Other error numbers should be added for errors which make little
sense on Unix systems.
*/
#define XPERM -1
#define XENOENT -2
#define XESRCH -3
#define XEINTR -4
#define XEIO -5
#define XENXIO -6
#define XEBIG -7
#define XENOEXEC -8
#define XEBADF -9
#define XECHILD -10
#define XEAGAIN -11
#define XENOMEM -12
#define XEACCES -13
#define XEFAULT -14
#define XENOTBLK -15
#define XEBUSY -16
#define XEXIST -17
#define XEXDEV -18
#define XENODEV -19
#define XNODIR -20
#define XESDIR -21
#define XEINVAL -22
#define XENFILE -23
#define XEMFILE -24
#define XNOTTY -25
#define XETXTBSY -26
#define XEBNG -27
#define XENOSPC -28
#define XESPIPE -29
#define XEROFS -30
#define XEMLINK -31
#define XEPipe -32
#define XEDOM -33
#define XERANGE -34
#define XEWOULDBLOCK -35
#define XEINPROGRESS -36
#define XEALREADY -37
#define XENOTSOCK -38
#define XDESTADDRREQ -39
#define XMSGSIZE -40
#define XEPROTO -41
#define XENOPROTOOPT -42
#define XEPNOTSUB -43
/* math software */
/* Argument too large */
/* Result too large */
/* interrupt and non-blocking io */
/* Operation would block */
/* Operation now in progress */
/* Operation already in progress */
/* argument errors */
/* Socket operation on non-socket */
/* Destination address required */
/* Message too long */
/* Protocol wrong type for socket */
/* Protocol not available */
/* Protocol not supported */
57 #define XESOCKTNOSUPPORT -44
58 #define XEPNOTSUPP -45
59 #define XEPFNSUPPORT -46
60 #define XEAFNOSUPPORT -47
61 #define XEADDRINUSE -48
62 #define XEADDRNOTAVAIL -49
63
64 /* operational errors */
65 #define XENETDOWN -50
66 #define XENETUNREACH -51
67 #define XENETRESET -52
68 #define XECONNABORTED -53
69 #define XECONNRESET -54
70 #define XENOBUFS -55
71 #define XEOOBUS -56
72 #define XENOTCONN -57
73 #define XESHUTDOWN -58
74 #define XETOOMANYREFS -59
75 #define XETIMEOUT -60
76 #define XECONNNREFUSED -61
77
78 /* random errors */
79 #define XELOOP -62
80 #define XENAMETOOLONG -63
81 #define XEHOSTDOWN -64
82 #define XEHOSTUNREACH -65
83 #define XSYSERR -66

/* Socket type not supported */
/* Operation not supported on socket */
/* Protocol family not supported */
/* Address family not supported by protocol fam */
/* Address already in use */
/* Can't assign requested address */

/* Network is down */
/* Network is unreachable */
/* Network dropped connection on reset */
/* Software caused connection abort */
/* Connection reset by peer */
/* No buffer space available */
/* Socket is already connected */
/* Socket is not connected */
/* Can't send after socket shutdown */
/* Too many references: can't splice */
/* Connection timed out */
/* Connection refused */

/* Too many levels of symbolic links */
/* File name too long */
/* Host is down */
/* No route to host */
/* unspecified os specific error */
1 #include <rsxos.h>
2 #include <stdio.h>
3 #include <xcctype.h>
4 #include <xerrno.h>
5 #define xstrncpy _ncpy
6 #define xstrn cmp _ncmp
7 #define xstrnc a t _ncat
8
9 #define PTOLBYTE( cp ) (cp = cp)
/*
this file declares password structre
*/
#define MAXUSERNAME 10
#define MAXPASSWORD 8
#define UICSIZE 10
struct passwd {
    char login_uic[UICSIZE];
    char log_dev[6];
    char cur_uic[UICSIZE];
    char cur_dev[6];
};
/*
@(#)xspecial.h 1.8 5/7/85
flags for special files
*/
#define FILE NAME 50 /* file name argument is to be used (as is) */
#define CURRENT_DIR 51 /* current directory */
#define HOME_DIR 52 /* user's initial location in file system */
#define CD_RELATIVE 53 /* name is relative to current directory */
#define HM_RELATIVE 54 /* name is relative to home directory */
#define UP_DIRECTORY 55 /* parent directory (for xchdir) */

/*
flags for psuedo-file objects
*/
#define LS 101 /* short directory listing */
#define LS_ARG 102 /* short listing of named directory */
#define LS_LONG 103 /* long directory listing */
#define LS_LONG_ARG 104 /* long listing of named directory */
#define PWD 105 /* return name of current directory */

/*
flags for file opening modes.
*/
#define XREAD 1 /* open for reading */
#define XWRITE 2 /* open for writing */
#defineXFAPPEND 8 /* add to an existing file (FWRITE also
must be set) */
#define XCREAT 0x80 /* create file, if it doesn't exist */
#define XFTRUNC 0x100 /* truncate file (FWRITE also must be set) */
#define XFASCII 0x200 /* file is ascii (for systems which care) */

Note: XCREAT is a separate issue from XFTRUNC and XFAPPEND, which are
mutually exclusive.
*/

/*
Information for FTP style files
*/
#define RT_ASCII 1 /* ascii character set */
#define RT_EBCDIC 2 /* ebcdic character set */
#define RT_IMAGE 3 /* uninterpreted bit stream */
#define RT_LOCALBYTE 4 /* weird sized bytes */
#define TF_NONPRINT 1 /* no imbedded carriage control */
#define TF_TELETNET 2 /* telnet style data */
#define TF_FORTRAN 3 /* 1st column == carriage control */
#define IS_FILE 1 /* Unstructured file */
#define IS_RECORD 2 /* FTP record internal structure */
#define IS_PAGE 3 /* FTP page internal structure */
#define IS_STREAM 1 /* stream transmission */
#define IS_BLOCK 2 /* block transmission */
#define IS_COMPRESSED 3 /* data compressed */

struct ftp_attr {

int rep_type; /* data representation one of:
               RT_ASCII, RT_EBCDIC, RT_IMAGE or
               RT_LOCALBYTE */
int format; /* format for character files one of:
            TF_NONPRINT, TF_TELNET or
            TF_FORTAN */
int structure; /* internal structure one of: IS_FILE,
               IS_RECORD, IS_PAGE */
int trans_mode; /* transmission mode one of: TM_STREAM,
                 TM_BLOCK or TM_COMPRESSED */
int byte_sz; /* byte size if representation type
              is RT_LOCALBYTE */
}

/* Flags for setting terminal options with xsetterm. */
#define XON_STERM 1 /* turn option on */
#define XOFF_STERM 0 /* turn option off */
#define XECHO 1 /* local echo? */
#define XLINE_EDIT 2 /* driver handles line edit? */

#define MXNAMELEN 255 /* maximum length for file names
                   ( system dependent ) */

#ifdef zilog
/*
 * S8000 does setjmp() differently, and calls it setret().
 * Do NOT call setret() from routine which declares register variables!
 */
#define xsetjmp(x) setret(x)
#else
#define xsetjmp(x) setjmp(x) /* Unix only */
#endif
/*
@(#)stdio.h  1.5 6/4/85

Definitions for EXOS standard io objects
  (useful for porting code to non-unix systems)
*/

/*
save space on systems with limited data segment size.
*/

 ifndef xenix286
 #define XBUFSIZ 512
 #else
 #define rsx
 #define XBUFSIZ 512
 #else
 #define XBUFSIZ 1024
 #endif
 #endif

#define _XNFILE 20
extern struct _xiobuf {
  int _cnt;
  char *ptr;
  char *base;
  int _bufsz;
  short _flag;
  char _file;

  struct _xiobuf *succ; /* forward link (added) */
  struct _xiobuf *pred; /* backward link (added) */
  char *sys_id; /* system specific identifier (added) */
  int (*_read)(); /* field to be added */
  int (*_write)(); /* field to be added */
  int (*_ioctl)(); /* field to be added */
  int (*_close)(); /* field to be added */
}_xiob[_XNFILE];

#define XIORAD 01
#define XIOWRT 02
#define XIONBF 04
#define XIOMYBUF 010
#define XIOEOF 020
#define XIOERR 040
#define XIOSTRG 0100
#define XIODBF 0200
#define XIORW 0400
#define XPrimary 01000 /* primary copy of object */
#define XUsed 02000 /* on if object is in use */
#define XNULL 0
#define XFILE struct _xiobuf
#define XEOF (-1)

#define xstdin (&_xiob[0])
#define xstdout (&_xiob[1])
#define xstderr (&_xiob[2])
#define xgetc(p) (-(p)->_cnt>=0? *(p)->_ptr++&0377:_xfilbuf(p))
```c
#define xgetchar() xgetc(stdin)
#define xputc(x,p) (--(p)->_cnt++)? ((int)(*(p)->_ptr++)=(unsigned)(x)): xflsbf((unsigned)(x)), xputc(x, stdout)
#define xfeof(p) (((p)->flag & XIOEOF)!=0)
#define xferror(p) (((p)->flag & XIOERR)!=0)
#define xfileno(p) ((p)->_file)

extern int xnofunc();
XFILE *xodopen();
char *xogets();
char *xprintf(); /* too painful to do right */
```
/*
 @((#)ftpc.h 1.2 4/11/85

 Header files for generic client side of FTP
 */

#include <rsxsos.h>
#include <xstdio.h>
#include <xctype.h>
#include <xerrno.h>
#include <xspecial.h>
#include <socket.h>
#include <netinet.h>

typedef int jmp_buf;
#include <ftp.h>
#include <in.h>
define SIOCDONE XNULL
define FIONBIO (126)
define appendhelp happend
define deletemhelp hdelete
define disconnecthelp hdiscon
define mdeletehelp hmdelete
define renamehelp hrename
define statushelp hstatus
define struchelp hstruct
define renamecmd cmdrename
extern xclose();
extern int figit;
extern int errno;
extern long xpasstnet();
extern long xpassfnet();
define VOID figit = (int)

/*
 * FTP global references.
 */
#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 int typename[32]; /* name of file transfer type */
extern int type; /* file transfer type */
extern int strctname[32]; /* name of file transfer structure */
extern int stru; /* file transfer structure */
extern char formname[32]; /* name of file transfer format */
extern int form; /* file transfer format */
extern char modename[32]; /* name of file transfer mode */
extern int mode; /* file transfer mode */
extern char bytename[32]; /* local byte size in ascii */
extern int bytsize; /* local byte size in binary */

extern char *hostname; /* name of host connected to */
extern struct servent *sp; /* service spec for tcp/ftp */

extern jmp_buf toplevel; /* non-local goto stuff for cmd scanner */

extern char line[200]; /* input line buffer */
extern int margc; /* count of arguments on input line */
extern char **argv; /* args parsed from input line */

extern int options; /* used during socket creation */

/*
 * Format of command table.
 */

extern struct cmd {
    char *c_name; /* name of command */
    char *c_help; /* help string */
    char c_bell; /* give bell when command completes */
    char c_conn; /* must be connected to use command */
    int (*c_handler)(); /* function to call */
};

extern char *tail();
extern char *remglob();
extern int errno;
/*@(#)varpat.h 1.8 4/11/85*/

#define connected conned
#define connecthelp connhelp
#define mdeletehelp mdelhelp
#define receivehelp recehelp
#define verbosehelp verbhelp
#ifndef lint
static char scssid[] =
" (@(#)cmds.c 1.24 8/28/85"
#endif

/* FTP User Program -- Command Routines.

#include "ftpc.h"

extern char *globerr;
extern char **xglob();
extern char **xmkarglist();
extern short gflag;
extern char *remglob();
extern char *getenv();
extern char *xstrchr();
extern char *xstrrchr();
static char **glizept = (char **)0;

#define BUFSIZ 1024

/*
 * Connect to peer server and
 * auto-login, if possible.
 */
setpeer(argc, argv)
  int argc;
  char *argv[];
{
  struct hostent *host, *hookup();
  int port;
  int madeargs = 0;

  if (connected) {
    xprintf(xstdout,
      "Already connected to %s, use close first.\n",
      hostname);
    return;
  }

  if (argc < 2) {
    xstrcat(line, " ");
    xprintf(xstdout,"(to ) ");
    xfflush( xstdout );
    xgets(&line[strlen(line)]);
    argv = xmkarglist( line, &argc );
    madeargs = 1;
  }

  if (argc < 2 || argc > 3) {
    xprintf(xstdout,"usage: %s host-name [port]\n", argv[0]);
    goto endspeer;
  }

  port = sp->s_port;
  if (argc > 2) {
    port = xatoi(argv[2]);
    if (port <= 0) {
xprintf(xstdout,"%s: bad port number-- %s\n", argv[1], argv[2]);
xprintf(xstdout,"usage: %s host-name [port]\n", argv[0]);
goto endspere;
}
port = xhtons(port);
host = hookup(argv[1], port);
if (host) {
    connected = 1;
    if (autologin && fromatty )
        login(host);
}
endsperes:
if( madeargs )
xdealglob( argv );
}

struct types {
    char  *t_name;
    char  *t_mode;
    int    t_type;
    char  *t_arg;
} types[] = {
{ "ascii",    "A",    TYPE_A,  0 },
{ "binary",  "B",    TYPE_B,  0 },
{ "image",   "I",    TYPE_I,  0 },
{ "ebcdic",  "E",    TYPE_E,  0 },
{ "tenex",   "L",    TYPE_L,  basename },
0
};

/*
 * Set transfer type.
 */
settype(argc, argv)
    char *argv[];
{
    register struct types *p;
    int comret;
    if (argc > 2) {
        char *sep;
        xprintf(xstdout,"usage: %s [", argv[0]);
        sep = " ";
        for (p = types; p->t_name; p++) {
            xprintf(xstdout,"%s%s", sep, p->t_name);
            if (*sep == ' ')
                sep = " | ";
        }
        xprintf(xstdout," ]\n"");
        return;
    }
    if (argc < 2) {
        xprintf(xstdout,"Using %s mode to transfer files.\n", typename);
    }
        return;
    }
    for (p = types; p->t_name; p++)
        if (xstrcmp(argv[1], p->t_name) == 0)
            break;
    if (p->t_name == 0) {
        xprintf(xstdout,"%s: unknown mode\n", argv[1]);
        return;
    }
    if ((p->t_arg != XNULL) && *(p->t_arg) != '\0'))
        comret = command("TYPE %s %s", p->t_mode, p->t_arg);
    else
        comret = command("TYPE %s", p->t_mode);
    if (comret == COMPLETE) {
        xstrcpy(typename, p->t_name);
        type = p->t_type;
    }
}

/*
 * Set binary transfer type.
 */

/*VARARGS*/

setbinary()
{
    call(settype, "type", "binary", 0);
}

/*
 * Set ascii transfer type.
 */

/*VARARGS*/

setascii()
{
    call(settype, "type", "ascii", 0);
}

/*
 * Set tenex transfer type.
 */

/*VARARGS*/

settenex()
{
    call(settype, "type", "tenex", 0);
}

/*
 * Set ebcDIC transfer type.
 */

/*VARARGS*/

setebcdic()
{
}
169     call(settype, "type", "ebcdic", 0);
170 }
171 /*
172  * Set file transfer mode.
173  */
174 setmode(argc, argv)
175     char *argv[];
176 {
177     xprintf(xstdout,"We only support %s mode, sorry.\n", modename);
178 }
179 /*
180  * Set file transfer format.
181  */
182 setform(argc, argv)
183     char *argv[];
184 {
185     xprintf(xstdout,"We only support %s format, sorry.\n", formname);
186 }
187 /*
188  * Set file transfer structure.
189  */
190 setstruct(argc, argv)
191     char *argv[];
192 {
193     xprintf(xstdout,"We only support %s structure, sorry.\n", structname);
194 }
195 /*
196  * Send a single file.
197  */
198 put(argc, argv)
199     int argc;
200     char *argv[];
201 {
202     char *cmd;
203     char *remote;
204     char *local;
205     int madeargs = 0;
206     int madeglob = 0;
207     if (argc == 2)
208         argc++, remote = argv[1];
209     else if (argc < 2) {
210         strcat(line, " ");
211         xprintf(xstdout,"(local-file)\n");
212         xfflush( xstdout );
213         xgets(&line[xstrlen(line)]);
214         argv = xmkarglist( line, &argc );
215         madeargs = 1;
216     }
else {
    remote = argv[2];
}
} 
if ( argc < 2 ) {
    xoprintf(xstdout,"%s local-file [remote-file]\n", argv[0]);
goto endput;
}
} 
if (argc < 3) {
    xstrcat(line, " ");
xoprintf(xstdout,"(remote-file, %s is default) ", argv[1] );
xfflush( xstdout );
xgets(&line[xstrlen(line)]);
if( madeargs )
    xdealglob( argv );
argv = xmkarglist( line, &argc );
madeargs = 1;
remote = argv[2];
}
if (argc < 3) {
    remote = argv[1];
}
local = argv[1];
if (!madeglob = globulize(&local))
goto endput;
cmd = (argv[0][0] == 'a') ? "APPE" : "STOR";
sendrequest(cmd, local, remote);
endput:
if( madeglob && doglob )
xdealglob( glistept );
if( madeargs )
xdealglob( argv );

/*
 * Send multiple files.
 */
mput(argc, argv)
char *argv[];
{
    char **cpp, **args = XNULL;
    int madeargs = 0;
    int cfrval;
    int doull = 0;
    if (argc < 2) {
        xstrcat(line, " ");
xoprintf(xstdout,"(local-files) ");
xfflush( xstdout );
    xgets(&line[xstrlen(line)]);
    argv = xmkarglist( line, &argc );
    madeargs = 1;
    }
    if (argc < 2) {
        xoprintf(xstdout,"%s local-files\n", argv[0]);
goto endput;
    }
cpp = argv + 1;
if (doglob) {
    gargs = xglob(cpp);
    if (globerr != XNULL) {
        xprintf(xstdout,"%s\n", globerr);
        if (gargs)
            xdealglob(gargs);
        goto endmput;
    }
}
if (gargs != XNULL)
    cpp = gargs;
for (; *cpp != XNULL; cpp++)
    {
        if ( !doall )
            cf rval = confirm(argv[0], *cpp);
        if( cf rval == 'a' )
            doall = 1;
        if( cf rval == 'q' )
            break;
        if ( cf rval )
            sendrequest("STOR", *cpp, *cpp);
    }
if (gargs != XNULL)
endmput:
    xdealglob(gargs);
    if( madeargs )
        xdealglob( argv );
}

/*
 * Receive one file.
 */
get(argc, argv)
    char *argv[];
{
    int madeargs = 0;
    int madeglob = 0;
    char *local;

    if (argc == 2)
        argc++, local = argv[1];
    else if (argc < 2) {
        xstrcat(line, " ");
        xprintf(xstdout,"(remote-file )");
        fflush( xstdout );
        xgets(&line[xstrlen(line)]);
        argv = xmkarglist( line, &argc );
        madeargs = 1;
    }
    else {
        local = argv[2];
    }

    if (argc < 2) {
        xprintf(xstdout,"%s remote-file [ local-file ]\n", argv[0]);
        goto endget;
    }
} 
if (argc < 3) {
    xstrcat(line, " ");
    xoprintf(xstdout,"(local-file, %s is default) ", argv[1] );
    xfflush( xstdout );
    xgets(&line[xstrlen(line)]);
    if( madeargs )
        xdealglob( argv );
    argv = xmkarglist( line, &argc );
    madeargs = 1;
    local = argv[2];
} 
if (argc < 3) {
    local = argv[1];
}
if (!madeglob = globulize(&local))
    goto endget;
recvirequest("RETR", local, argv[1], "A");
endget:
if( madeglob && doglob )
    xdealglob( glizept );
if( madeargs )
    xdealglob( argv );
/*
 * Get multiple files.
 */
mgget(argc, argv)
char *argv[];
{
    char *cp;
    int madeargs = 0;
    int cfrval;
    int doall = 0;
    if (argc < 2) {
        xstrcat(line, " ");
        xoprintf(xstdout,"(remote-files) ");
        xfflush( xstdout );
        xgets(&line[xstrlen(line)]);
        argv = xmkarglist( line, &argc );
        madeargs = 1;
    }
    if (argc < 2) {
        xoprintf(xstdout,%s remote-files\n", argv[0]);
        goto endmgget;
    }
while ((cp = remglobe(argc, argv)) != XNULL)
{
    if( !doall )
        cfrval = confirm(argv[0], cp );
    if( cfrval == 'a' )
        doall = 1;
    if( cfrval == 'q' ) {
        while ((cp = remglobe(argc, argv)) != XNULL)
393             break;
394         }
395         if ( cfrval )
396             recvrequest("RETR", cp, cp, "w");
397     }
398 }
399     endmget:
400     if ( madeargs )
401         xdealglobs( argv );
402 }
403
404     char temp[16] = { 0 };  
405     char *
406     remglobs(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 oldname[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         xstrncpy(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             xstrncpy( oldname, 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", oldname, 0 );
447         }
448         verbose = oldverbose;
fitemi = xopen(temp, XFREAD | XFASCII, FILE_NAME);
xunlink(temp, FILE_NAME);
fitem = xopen( fitemi, "r" );
if (fitem == XNULL) {
xprintf(stdout,
    "can't find list of remote files, oops\n");
    return (XNULL);
}

if (xgets(buf, sizeof (buf), fitem) == XNULL) {
    xclose(xfileno(fitem)), fitem = XNULL;
    return (XNULL);
}
   if ((cp = xstrchr(buf, '\n')) != XNULL)
       *cp = '\0';
    return (buf);
}

char *
onoff(bool)
{
    int bool;
    return (bool ? "on" : "off");
}

/*
 * Show status.
 */
status(argc, argv)
{
    char *argv[];

    if (connected)
xprintf(stdout,"Connected to %s,\n", hostname);
else
    xprintf(stdout,"Not connected,\n");
xprintf(stdout,"Mode: %s; Type: %s; Form: %s; Structure: %s,\n",
    modename, typename, formname, structname);
xprintf(stdout,"Verbose: %s; Bell: %s; Prompting: %s; Globbing: %s,\n",
    onoff(verbose), onoff(bell), onoff(interactive),
    onoff(doglob));
xprintf(stdout,"Hash mark printing: %s; Use of PORT cmds: %s,\n",
    onoff(hash), onoff(sendport));
}

/*
 * Set beep on cmd completed mode.
 */
setbell()
{
    bell = !bell;
xprintf(stdout,"Bell mode %s,\n", onoff(bell));
505 */
506 * Turn on packet tracing.
507 */
508 /*VARARGS*/
509 settrace()
510 {
511    
512        trace = !trace;
513        xprintf(xstdout,"Packet tracing %s.
", onoff(trace));
514    
515 }
516 
517 /*
518 * Toggle hash mark printing during transfers.
519 */
520 /*VARARGS*/
521 sethash()
522 {
523    
524        hash = !hash;
525        xprintf(xstdout,"Hash mark printing %s", onoff(hash));
526        if (hash)
527            xprintf(xstdout," (%d bytes/hash mark)", BUFSIZE);
528        xprintf(xstdout,".
");
529    
530 }
531 */
532 * Turn on printing of server echo's.
533 */
534 /*VARARGS*/
535 setverbose()
536 {
537    
538        verbose = !verbose;
539        xprintf(xstdout,"Verbose mode %s.
", onoff(verbose));
540    
541 */
542 * Toggle PORT cmd use before each data connection.
543 */
544 /*VARARGS*/
545 setport()
546 {
547    
548        sendport = !sendport;
549        xprintf(xstdout,"Use of PORT cmds %s.
", onoff(sendport));
550    
551 }
552 
553 */
554 * Turn on interactive prompting
555 * during mget, mput, and mdelete.
556 */
557 /*VARARGS*/
558 setprompt()
559 {
560
interactive = !interactive;
xprintf(xstdout,"Interactive mode %s
", onoff(interactive));
}

/*
 * Toggle metacharacter interpretation
 * on local file names.
 */
/*VARARGS*/
setglob()
{
doglob = !doglob;
xprintf(xstdout,"Globbing %s
", onoff(doglob));
}

/*
 * Set debugging mode on/off and/or
 * set level of debugging.
 */
/*VARARGS*/
setdebug(argc, argv)
char *argv[];
{
  int val;

  if (argc > 1) {
    val = xatoi(argv[1]);
    if (val < 0) {
      xprintf(xstdout,"%s: bad debugging value.
", argv[1]);
      return;
    }
  } else
    val = !debug;

debug = val;
if (debug)
  options |= SO_DEBUG;
else
  options &= ~SO_DEBUG;

xprintf(xstdout,"Debugging %s (debug=%d).
", onoff(debug), debug);
}

/*
 * Set current working directory
 * on remote machine.
 */
/*
 * cd(argc, argv)
char *argv[];
{
  int madeargs = 0;

  if (argc < 2) {
    xstrcat(line, " ");
    xprintf(xstdout,("remote-directory) ");
    xfflush(xstdout);
    xgets(&line[xstrlen(line)]);
argv = xmkarglist( line, &argc );
madeargs = 1;
}

} if (argc < 2) {
xprintf(xstdout,"%s remote-directory\n", argv[0]);
goingcd;
}

} VOID command("CWD %s", argv[1]);

} if( madeargs )
xdealglob( argv );

} /*
* Set current working directory
* on local machine.
*/
lcd(argc, argv)
char *argv[];
{
char buf[MAXPATHLEN];
char *dir;
int madeglob = 0;
int rval;
int func_code;

if (argc < 2)
    argc++, dir = (char *)0, func_code = HOME_DIR;
else
    dir = argv[1], func_code = FILE_NAME;
if (argc != 2) {
xprintf(xstdout,"%s local-directory\n", argv[0]);
goingcd;
}
if (!madeglob == globulize(&dir))
goingcd;
if ((rval = xchdir(dir, func_code)) < 0) {
xerror(rval, dir);
goingcd;
}
endcd:
if( madeglob && doglob )
xdealglob( glizept );

} /*
* Delete a single file.
*/
delete(argc, argv)
char *argv[];
{
int madeargs = 0;

if (argc < 2) {
xstrcat(line, " ");
xprintf(xstdout,(remote-file ");


xfflush( stdout);

xgets(cline[strlen(line)]);
argv = xmkarglist( line, argc);
madeargs = 1;

} 
if (argc < 2) {

xprintf(stdout,"%s remote-file\n", argv[0]);
goto enddelete;

} 
VOID command("DELE %s", argv[1]);
enddelete:
if( madeargs )

xdealglob( argv );

} 

/*
 * Delete multiple files.
 */
mdelete(argc, argv)

char *argv[];

char *cp;

int madeargs = 0;

int cfrval;

int doall = 0;

if (argc < 2) {

xstrcat(line, " ");
xprintf(stdout,"(remote-files) ");
xfflush( stdout );
xgets(&line[strlen(line)]);
argv = xmkarglist( line, argc );
madeargs = 1;

} 
if (argc < 2) {

xprintf(stdout,"%s remote-files\n", argv[0]);
goto endmdel;

}
while ((cp = remglob(argc, argv)) != XNULL)

{

if( !doall )

cfrval = confirm(argv[0], cp);

if( cfrval == 'a' )
doall = 1;

if( cfrval == 'q' ){

while ((cp = remglob(argc, argv)) != XNULL)

break;

}

if ( cfrval )

VOID command("DELE %s", cp);

}
endmdel:
if( madeargs )

xdealglob( argv );

}
/*
 * Rename a remote file.
 */
renamefile(argc, argv)
    char *argv[];
{
    int madeargs = 0;

    if (argc < 2) {
        xstrcat(line, " ");
xprintf(xstdout,"(from-name) ");
xfflush( xstdout );
xgets(&line[xstrlen(line)]);
argv = xmkarglist( line, &argc );
madeargs = 1;
    }
    if (argc < 2) {
        usage:
        xprintf(xstdout,"%s from-name to-name\n", argv[0]);
goto endrname;
    }
    if (argc < 3) {
        xstrcat(line, " ");
xprintf(xstdout,"(to-name) ");
xfflush( xstdout );
xgets(&line[xstrlen(line)]);
if( madeargs )
    xdealglob( argv );
argv = xmkarglist( line, &argc );
    }
    if (argc < 3)
goto usage;
    if (command("RNF R %s", argv[1]) == CONTINUE)
VOID command("RNTO %s", argv[2]);
    else
VOID command("RNTO "); /* keep server happy */
endrname:
    if( madeargs )
    xdealglob( argv );
}

/*
 * Get a directory listing
 * of remote files.
 */
ls(argc, argv)
    char *argv[];
{
    char *cmd;
    char *rdir;
    char *lfile;
    int madeglob = 0;

    if (argc < 2)
    argv++, rdir = XNULL;}
else
    rdir = argv[1];
if (argc < 3)
    argc++, lfile = "-";
else
    lfile = argv[2];
if (argc > 3) {
   xprintf(xstdout,"usage: %s remote-directory local-file\n", argv[0]);
   return;
}
    cmd = argv[0][0] == 'l' ? "NLST" : "LIST";
    if (xstrcmplfile, "-") && !(madeglob = globulize(lfile)))
        goto endls;
    recvrequest(cmd, lfile, rdir, "w");
}
if (madeglob && doglob )
    xdealglob( glizept );

/*
 * Get a directory listing
 * of multiple remote files.
 */
mls(argc, argv)
char *argv[];
{
    char *cmd, *mode;
    int i, dest;
    char *rdir;
    char *lfile;
    int madeglob = 0;
    int cfrval;

    if (argc < 2)
        argc++, rdir = XNULL;
    else
        rdir = argv[1];
    if (argc < 3)
        {
            argc++, lfile = "-";
            dest = argc - 1;
        }
    else
        {
            dest = argc - 1;
            lfile = argv[dest];
        }
    cmd = argv[0][1] == 'l' ? "NLST" : "LIST";
    if (xstrcmplfile, "-") != 0)
        if (!madeglob = globulize(lfile)) ||
            !(cfrval = confirm("local-file", lfile)) ||
                cfrval == 'q'
         goto endmls;
    for (i = 2, mode = "w"; i < dest + 1 ; i++, mode = "a")
        {
recvrequest(cmd, lfile, rdir, mode);
    rdir = argv[i];
}
#endif
    if( madeglob && doglob )
    xdealglob( glizept );
#endif

/*
 * shell escape not implemented
 */

#ifndef SHELLESCAPE

/*
 * shell escape for a specific OS
 */

#endif SHELLESCAPE

/*
 * Send new user information (re-login)
 */

user(argc, argv)
    int argc;
    char **argv;
{
    char acct[80], *xgetpass();
    int n;
    int madeargs = 0;
    char *password;
    char *account;

    if (argc < 2){
        xstrcat(line, " ");
        xoprintf(xstdout,"(Remote Username) ");
        xfflush( xstdout );
        xgets(&line[xstrlen(line)]);
        argv = xmkarglist( line, &argc );
        madeargs = 1;
    }
    if (argc < 2 || argc > 4) {
        xoprintf(xstdout,
            "usage: %s username [password] [account]\n", argv[0]);
goto enduser;
}

n = command("USER %s", argv[1]);
if (n == CONTINUE) {
    if (argc < 3)
        password = xgetpass("Remote Password: "), argc++;
    else
        password = argv[2];
    n = command("PASS %s", password);
    if (n == CONTINUE) {
        if (argc < 4) {
            xprintf(xstdout, "Remote Account:");
            VOID xfflush(xstdout);
            VOID xogets(acct, sizeof(acct) - 1, xstdin);
            acct[strlen(acct) - 1] = '\0';
            account = acct;
            argc++;
        }
        else
            account = argv[3];
    n = command("ACCT %s", account);
}

if (n != COMPLETE) {
    xprintf(xstderr, "Login failed.\n");
    goto enduser;
}

if( madeargs )
    xdealglob( argv );
return (1);
enduser:
if( madeargs )
    xdealglob( argv );
return( 0 );

/*
 * Print working directory.
 */

/*VARARGS*/
pwd()
{
    int noverbose = 0;

    if( !verbose )
    {
        noverbose = 1;
        verbose = 1;
    }
    VOID command("XPWD");
    if( noverbose )
        verbose = 0;
}

/*
 * Make a directory.
 */
```c
953 */
954 makedir(argc, argv)
955 char *argv[];
956 {
957    int madeargs = 0;
958
959    if (argc < 2) {
960       xstrcat(line, " ");
961       xopprintf(stdout,"(directory-name) ");
962       xfflush( stdout );
963       xgets(&line[xstrlen(line)]);
964       argv = xmkarglist( line, &argc );
965       madeargs = 1;
966    }
967    if (argc < 2) {
968       xopprintf(stdout,"%s directory-name\n", argv[0]);
969       goto endmksd;
970    }
971    VOID command("XMKD %s", argv[1]);
972    endmksd:
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       xopprintf(stdout,"(directory-name) ");
988       xfflush( stdout );
989       xgets(&line[xstrlen(line)]);
990       argv = xmkarglist( line, &argc );
991       madeargs = 1;
992    }
993    if (argc < 2) {
994       xopprintf(stdout,"%s directory-name\n", argv[0]);
995       goto endrmddir;
996    }
997    VOID command("XRMD %s", argv[1]);
998    endrmddir:
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 {
```
int i;
char buf[BUFSIZ];
int madeargs = 0;

if (argc < 2) {
    xstrcat(line, " ");
    xprintf(xstdout,"(command line to send) ");
    xfflush(xstdout);
    xgets(&line[strlen(line)]);
    argv = xmkarglist(line, &argc);
    madeargs = 1;
}
if (argc < 2) {
    xprintf(xstdout,"usage: %s line-to-send\n", argv[0]);
    goto endquote;
}
strcpy(buf, argv[1]);
for (i = 2; i < argc; i++) {
    xstrcat(buf, " ");
    xstrcat(buf, argv[i]);
}
VOID command(buf);
endquote:
if( madeargs )
    xdealglob(argv);
/*
 * Ask the other side for help.
 */
rmthelp(argc, argv);
char *argv[];
int oldverbose = verbose;
verbose = 1;
VOID command(argc == 1 ? "HELP" : "HELP %s", argv[1]);
verbose = oldverbose;
/*
 * Terminate session and exit.
 */
/*VARARGS*/
quit()
{
    if (connected)
        disconnect();
    xexit(0);
}
/*
 * Terminate session, but don't exit.
 */
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      xprintf(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(l);
1109  }
1110
1111  fatal(msg)
1112  char *msg;
1113  {
1114      xprintf(xstderr, "ftp: %s\n");
1115      xexit(l);
1116  }
1117
1118  /*
1119     * Glob a local file name specification with
1120     */
1121
static
globulize(cpp)
{
    char **cpp;
    char **globbed;
    char *argv[2];
    
    if (!doglob)
        return (1);
    argv[0] = *cpp;
    argv[1] = (char *)0;
    globbed = xglob( argv );
    if (globerr != XNULL) {
        xprintf(xstdout,"%s: %s\n", *cpp, globerr);
        if (globbed)
            xdealglob(globbed);
        return (0);
    }
    if (globbed) {
        *cpp = *globbed;
        /* don't waste too much memory */
        glizept = globbed;
    }
    else
        return( 0 );
    return (1);
}

l1s( argc, argv )

int argc;
char *argv[];

{ 
    if( argc > 1 )
        llist( argc, argv, LS_ARG );
    else
        llist( argc, argv, LS );
}

ldir( argc, argv )

int argc;
char *argv[];

{ 
    if( argc > 1 )
        llist( argc, argv, LSLONG_ARG );
    else
        llist( argc, argv, LSLONG );
}

llist( argc, argv, func_code )
int argc;
char *argv[];
int func_code;
{
    char **argv1;
    char **argv2;
    char *pt;

    if ( argc > 1 )
    {
        if ( doglob ) {
            argv1 = xglob( &argv[1] );
            if( argv1 == XNULL || globerr ){
                xprintf( stdout, "No file name matches." );
                if ( argv1 != XNULL )
                    xdealglob( argv1 );
                return;
            }
        } else {
            argv1 = &argv[1];
        }
        argv2 = argv1;
        for ( pt = *argv2++ ; pt ; pt = *argv2++ )
        {
            xls( xfileno(stdout), pt, func_code );
        }
        if( doglob )
            xdealglob( argv1 );
    }
    else
    {
        xls( xfileno(stdout), (char *)0, func_code );
    }
}
lpwd( argc, argv )

int argc;
char *argv[];
{
    char buf[MAXPATHLEN + 1];
    int success;

    success = xpwd( buf, sizeof( buf ), PWD );
    if( !success )
    {
        xprintf( stdout, "local current directory is: %s\n", buf );
    }
    else
    {
        xprintf( stdout,
            "current directory unknown %s\n", xerror( success ) );
    }
}
#ifndef lint
static char scsid[] =
"@(#)cmdtab.c 1.9 6/3/85"
#endif

#include "ftpc.h"

/*
 * User FTP -- Command Tables.
 */

int setascii(), setbell(), setbinary(), setdebug(), setform();
int setglob(), sethash(), setmode(), setpeer(), setport();
int setprompt(), setstruct();
int settenex(), settrace(), settype(), setverbose();
int disconnect();
int cd(), lcd(), delete(), mdelete(), user();
int ls(), mls(), get(), mget(), help(), put(), mput();
int quit(), renamefile(), status();
int quote(), rmthelp(), shell();
int pwd(), makedir(), removedir();
int lls(), lpwd(), ldir();

char appendhelp[] = "append to a file";
char asciihelp[] = "set ascii transfer type";
char beephelp[] = "beep when command completed";
char binaryhelp[] = "set binary transfer type";
char cdhelp[] = "change remote working directory";
char connecthelp[] = "connect to remote tftp";
char deletehelp[] = "delete remote file";
char debughelp[] = "toggle/set debugging mode";
char dirhelp[] = "list contents of remote directory";
char disconhelp[] = "terminate ftp session";
char formhelp[] = "set file transfer format";
char globhelp[] = "toggle metacharacter expansion of local file names";
char hashhelp[] = "toggle printing '#' for each buffer transferred";
char helphelp[] = "print local help information";
char lcdhelp[] = "change local working directory";
char lshelp[] = "nlist contents of remote directory";
char mdeletehelp[] = "delete multiple files";
char mdirhelp[] = "list contents of multiple remote directories";
char mgethelp[] = "get multiple files";
char mkdirc/help[] = "make directory on the remote machine";
char mlshelp[] = "nlist contents of multiple remote directories";
char modehelp[] = "set file transfer mode";
char mputhelp[] = "send multiple files";
char porthelp[] = "toggle use of PORT cmd for each data connection";
char prompthelp[] = "force interactive prompting on multiple commands";
char pwdhelp[] = "print working directory on remote machine";
char quithelp[] = "terminate ftp session and exit";
char quotehelp[] = "send arbitrary ftp command";
char receivehelp[] = "receive file";
char remotehelp[] = "get help from remote server";
char renamehelp[] = "rename file";
char rmdirhelp[] = "remove directory on the remote machine";
char sendhelp[] = "send one file";
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 lshelp[] = "list new user information";
64 char lpwdhelp[] = "print directory on local machine";
65 char ldirhelp[] = "long listing of local directory(s)";
66 char verbosehelp[] = "toggle verbose mode";

68 struct cmd cmdtab[] = {
69    { "!" ,      shellhelp, 0, 0, 0, shell },
70    { "append", appendhelp, 1, 1, 0, put },
71    { "ascii", asciihelp, 0, 0, 0, setascii },
72    { "bell", beephelp, 0, 0, 0, setbell },
73    { "binary", binaryhelp, 0, 1, 0, setbinary },
74    { "bye", quithelp, 0, 0, 0, quit },
75    { "cd", cdhelp, 0, 0, 1, cd },
76    { "close", disconhelp, 0, 1, 0, disconnect },
77    { "delete", delethehelp, 0, 1, 0, delete },
78    { "debug", debughelp, 0, 0, 0, setdebug },
79    { "dir", dirhelp, 1, 0, 1, ls },
80    { "form", formhelp, 0, 0, 1, setform },
81    { "get", receivehelp, 0, 0, 0, setget },
82    { "glob", globhelp, 0, 0, 0, setglob },
83    { "hash", hashhelp, 0, 0, 0, sethash },
84    { "help", helphelp, 0, 0, 0, help },
85    { "lcd", lcdhelp, 0, 0, 0, lcd },
86    { "ls", lshelp, 0, 0, 1, ls },
87    { "mddelete", delethehelp, 0, 0, 0, mdelete },
88    { "mdir", mdirhelp, 0, 0, 0, mls },
89    { "mget", mgethelp, 0, 0, 0, mget },
90    { "mkdir", mkdirhelp, 0, 0, 0, makedir },
91    { "mls", mlshelp, 0, 0, 1, mls },
92    { "mode", modehelp, 0, 0, 1, setmode },
93    { "mput", mputhelp, 0, 0, 0, mput },
94    { "open", connecthelp, 0, 0, 0, setpeer },
95    { "prompt", prompthelp, 0, 0, 0, setprompt },
96    { "sendport", porthelp, 0, 0, 0, setport },
97    { "put", sendhelp, 0, 0, 1, put },
98    { "pwd", pwdhelp, 0, 0, 0, pwd },
99    { "quit", quithelp, 0, 0, 0, quit },
100   { "quote", quotehelp, 0, 0, 1, quote },
101   { "recv", receivehelp, 0, 0, 0, get },
102   { "remotehelp", remotehelp, 0, 0, 0, rmtget },
103   { "rename", renamehelp, 0, 0, 0, renamefile },
104   { "rmddir", rmddirhelp, 0, 0, 0, removedir },
105   { "send", sendhelp, 0, 0, 0, send },
106   { "status", statushelp, 0, 0, 0, status },
107   { "struct", structhelp, 0, 0, 0, setstruct },
108   { "tenex", tenexhelp, 0, 0, 0, settenex },
109   { "trace", tracehelp, 0, 0, 0, settrace },
110   { "type", typehelp, 0, 0, 0, settype },
111   { "user", userhelp, 0, 0, 0, user },
112   { "verbose", verbosehelp, 0, 0, 0, setverbose },
113   { "version", versionhelp, 0, 0, 0, quit },
114   { "who", whohelp, 0, 0, 0, who } };
{ "ldir", ldirhelp, 0, 0, ldir },
{ "lls", llshelp, 0, 0, llsls },
{ "lpwd", lpwdhelp, 0, 0, lpwd },
{ "?", helphelp, 0, 0, help },

int NCMDS = (sizeof (cmdtab) / sizeof (cmdtab[0])) - 1;
#ifndef lint
static char sccsid[] = "@(#)ftp.c  1.20 6/21/85";
#endif

#include "ftp.h"

struct sockaddr_in hisctladdr = { AF_INET };
struct sockaddr_in data_addr = { AF_INET };
int data = -1;
struct sockaddr_in myctladdr = { AF_INET };

/*
 * Options and other state info.
 */
int trace = 0;   /* trace packets exchanged */
int hash = 0;    /* print # for each buffer transferred */
int sendport = -1; /* use PORT cmd for each data connection */
int verbose = 0; /* print messages coming back from server */
int connected = 0; /* connected to server */
int fromatty = 0; /* input is from a terminal */
int interactive = 0; /* interactively prompt on m* cmds */
int debug = 0;    /* debugging level */
int bell = 0;    /* ring bell on cmd completion */
int doglob = 0;  /* glob local file names */
int autologin = 0; /* establish user account on connection */

char typename[32] = {0};   /* name of file transfer type */
int type = 0;              /* file transfer type */
char structname[32] = {0}; /* name of file transfer structure */
int stru = 0;              /* file transfer structure */
char formname[32] = {0};   /* name of file transfer format */
int form = 0;              /* file transfer format */
char modename[32] = {0};   /* name of file transfer mode */
int mode = 0;              /* file transfer mode */
char bytename[32] = {0};   /* local byte size in ascii */
int bytesize = 0;          /* local byte size in binary */

char *hostname = (char*)0; /* name of host connected to */

struct servent *sp = 0;    /* service spec for tcp/ftp */
char line[200] = {0};      /* input line buffer */
int margc = 0;             /* count of arguments on input line */
char ***margv = (char **)0; /* args parsed from input line */
int options = 0;   /* used during socket creation */

extern char *xgetpass();
extern long xpassstnet(), xpassfnet();
extern long xtime();
extern FILE *xodopen();

#define SWAITMAX 90     /* wait at most 90 seconds */
#define SWAITINT 5      /* interval between retries */

int swaitmax = SWAITMAX;
int swaitint = SWAITINT;

XFILE *cin = XNULL, *cout = XNULL;
XFILE *dataconn();

struct hostent *
 hookup(host, port)
     char *host;
     int port;
{
    register struct hostent *hp;
    int s, len;
    int rval;

    bzero((char *)&hisctladdr, sizeof (hisctladdr));
    hp = ghnname(host);
    if (hp == XNULL) {
        static struct hostent def;
        static struct in_addr defaddr;
        static char namebuf[128];
        int inet_addr();

        defaddr.s_addr = inet_addr(host);
        if (defaddr.s_addr == -1) {
            xprintf(xstderr, "%s: Unknown host.\n", host);
            return (0);
        }
        xstrcpy(namebuf, host);
        def.h_name = namebuf;
        hostname = namebuf;
        def.h_addr = (char *)&defaddr;
        def.h_length = sizeof (struct in_addr);
        def.h_addrtype = AF_INET;
        def.h_aliases = 0;
        hp = &def;
    }
    hostname = hp->h_name;
    hisctladdr.sin_family = hp->h_addrtype;
    s = xsocket(SOCK_STREAM, (struct sockaddr *)0,
                (struct sockaddr *)0, SO_KEEPALIVE);
    if (s < 0) {
        x perror(s,"ftp: socket");
        return (0);
    }
    bcopy(hp->h_addr, (char *)&hisctladdr.sin_addr, hp->h_length);
    hisctladdr.sin_port = port;
    if ((rval = xconnect(s,(char *)&hisctladdr)) < 0){
        x perror(rval,"ftp: connect");
        goto bad;
    }
    len = sizeof (myctladdr);
    if ((rval = xsktaddr(s,(char *)&myctladdr)) < 0) {
        x perror(rval,"ftp: getsockname");
        goto bad;
    }
    xdup2(s, (rval = xnewod()));
    cin = xodopen(s, "r");
    cout = xodopen(rval, "w");
    if (cin == XNULL || cout == XNULL) {

xprintf(stderr, "ftp: fopen failed.\n");
if (cin)
  xclose(xfileno(cin));
if (cout)
  xclose(xfileno(cout));
goto bad;
}
if (verbose)
  xprintf(stdout,"Connected to %s.\n", hp->h_name);
VOID getreply(); /* read startup message from server */
return (hp);
bad:
  xclose(s);
return ((struct hostent *)0);
}
*/
/*
For now, non-interactive use of ftp requires explicite USER and
PASS commands.
Later, we can define an autologin procedure that will work for all
systems.
*/
login(hp)
  struct hostent *hp;
{
  char acct[80];
  char *user, *pass;
  int n;

  if( !fromatty )
    return( 0 );
  user = acct;
  xprintf(stdout,"Remote User Name:"); VOID xfflush(stdout);
  if( xogets(user, sizeof(acct) - 1, xstdin) == XNULL ){
    if( xfeof( xstdin ) ){
      xprintf( "\n" );
      quit();
      xexit( 0 );
    } else {
      return( 0 );
    }
  }
  if( xstrlen(acct) - 1 <= 0 )
    return( 0 );
  user[xstrlen(acct) - 1] = '\0';
  n = command("USER %s", user);
  if (n == CONTINUE)
    {
      pass = xgetpass("Remote Password:" );
      xputchar( '
' );
      n = command("PASS %s", pass);
      if (n == CONTINUE)
        {
          xprintf(stdout,"Remote Account: ");
          VOID xfflush(stdout);
          VOID xogets(acct, sizeof(acct) - 1, xstdin);
acct[strlen(acct) - 1] = '\0';
    n = command("ACCT %s", acct);
}

    if (n != COMPLETE) {
        xprintf(stderr, "Login failed.
");
        return (0);
    }

    return (1);

}*/VARARGS 1*/
#else zilog
    /*
    * Pick parameters from registers, and put them in an honest-looking
    * stack frame.
    */
    command(fmt, a1, a2, a3, a4, a5, a6)
    char *fmt;
    int a1, a2, a3, a4, a5, a6;
{
    int args=a1, aa2=a2, aa3=a3, aa4=a4, aa5=a5, aa6=a6;
#else
    command(fmt, args)
    char *fmt;
{
#endif
    int how;          /* something for ioctl args to point to */

    if (debug) {
        xprintf(stdout,"---> ");
        mydoprint(fmt, &args, stdout);
        xprintf(stdout,"\n");
        VOID xfflush(stdout);
}

    if (cout == XNULL) {
        xerror (0, "No control connection for command");
        return (0);
    }

    if(!empty( cin )) {
        /*
        * Since we are sending a new command, it is expected
        * that all replies to previous commands have been
        * processed. Thus, if there is any data in the command
        * stream, we are out of sync with the server, and
        * the data that is now present should be flushed.
        */
        how = 1;
        ioctl( xfileno( cout ), FIONBIO, &how );
        if( verbose )
            xprintf( stderr, "Old reply in command stream:\n" );
        VOID getreply( 0 );
        how = 0;
        ioctl( xfileno( cout ), FIONBIO, &how );
    }

    _mydoprint(fmt, &args, cout);
xprintf(cout, "\r\n");
VOID xfflush(cout);
return (getreply(!(strcmp(fmt, "QUIT"))));
}

empty(f)
XFILE *f;
{
    long mask;
    int rval;
    
    if ( f->_cnt > 0 )
        return( 0 );
    
    #ifndef NOSELECT
    mask = ( 1 << (xfileno(f)));
    rval = xselect( 20, &mask, (long *)0, 0L );
    return ( mask == 0 );
    #else
    return( 1 );
    #endif NOSELECT
}

getreply(expecteof)
int expecteof;
{
    register int c, n;
    register int code, dig;
    int originalcode = 0, continuation = 0;
    
    for (;;) {
        dig = n = code = 0;
        while ((c = xgetc(cin)) != '\n') {
            dig++;
            if (c == XEOF) {
                if (xfeof(cin)) {
                    if (expecteof)
                        return( 0 );
                    xprintf(xstdout, "lost connection.\n");
                    return( 5 );
                } else {
                    xprintf(xstdout, "error on read.\n");
                    return( n - '0' );
                }
            } else {
                if (verbose && c != '\r' ||
                    (n == '5' && dig > 4))
                    xputchar(c);
                if (dig < 4 && isdigit(c))
                    code = code * 10 + (c - '0');
                if (dig == 4 && c == '-')
                    continuation++;
                if (n == 0)
                    n = c;
            }
        }
    }
}
if (verbose || n == '5') {
    xputchar(c);
    VOID xfflush (xstdout);
}

if (continuation && code != originalcode) {
    if (originalcode == 0)
        originalcode = code;
    continue;
}

if (!continuation || (code == originalcode))
    return (n - '0');

sendrequest(cmd, local, remote)
char *cmd, *local, *remote;
{
    int (*closefunc)();
    long bytes = 0, hashbytes = 1024;
    long start, stop;
    int read_reply = 0;
    int inod;
    XFILE *inopt, *outopt;
    int omode;
    struct ftp_attr attributes;

    closefunc = XNULL;
    if (xstrcmp(local, "-") == 0) {
        inopt = xstdin;
    } else {
        omode = XFREAD;
        attributes.rep_type = type;
        attributes.format = form;
        attributes.structure = stru;
        attributes.trans_mode = mode;
        attributes.byte_size = bytesize;
        inod = xftppopen(local, omode, FILE_NAME, &attributes);
        if (inod < 0)
            xerror(inod, local);
        goto bad;
    }
    inopt = xodopen(inod, "r");
    if (inopt == XNULL) {
        xprintf(xstderr, "xodopen failed\n");
        xclose(inod);
        goto bad;
    }
    closefunc = xclose;
}

if (initconn())
    goto bad;
read_reply = 1;
if (remote) {
    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("\n");
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            xerror( (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      closefunc = XNULL;
388      if (initconn())
389        goto bad;
390      read_reply = 1;
391    if (remote) {
392
int x;
    if ((x = command("%s %s", cmd, remote)) != PRELIM) {
        /* fprintf(stderr, "bad return from command(%s %s) = %d\n", cmd, remote, x); */
        --read_reply;
        goto bad;
    }
} else {
    int x;
    if ((x = command("%s", cmd)) != PRELIM) {
        /* fprintf(stderr, "bad return from command(%s) = %d\n", cmd, x); */
        --read_reply;
        goto bad;
    }
    
    if (xstrcmp(local, "-") == 0) {
        outopt = xstdout;
    } else {
        omode = XWRITE | XCREATE |
            (/* append == 'a' */? XAPPEND : XTRUNC );
        attributes.rep_type = type;
        attributes.format = form;
        attributes.structure = stru;
        attributes.trans_mode = mode;
        attributes.byte_sz = bytesize;
        outod = xftopen(local, omode, FILE_NAME, &attributes );
        if( outod < 0 )
            {
            xerror( outod, local );
            goto bad;
            }
        outopt = xodopen( outod, "w" );
        if( outopt == XNULL )
            {
            xprintf(xstderr, "xodopen failed\n" );
            xclose( outod );
            goto bad;
            }
        closefunc = xclose;
    }
    inopt = dataconn("r");
    if (inopt == XNULL)
        goto bad;
    start = xtime();
    bytes = xpassfnet( inopt, outopt );
    stop = xtime();
    xclose( xfileno(inopt) );
    data = -1;
    if( closefunc != XNULL )
        xclose( outod );
    if( bytes < 0 )
        {
        xerror( (int)bytes, "local" );
        }
   _done:
    VOID getreply(0);
    if (bytes > 0 && verbose)
ptransfer("received", bytes, &start, &stop);
\nreturn;
\nbad:
if (bytes > 0 && verbose)
  stop = xtime();
if (data >= 0)
  VOID xclose(data), data = -1;
if (closefunc != XNULL && outopt != XNULL)
  xclose(outod);
if (read_reply == 1)
  VOID getreply(0);
goto done;
}

/*
 * Need to start a listen on the data channel
 * before we send the command, otherwise the
 * server's connect may fail.
 */

initconn()
{
  register char *p, *a;
  int result, len;
  int options = SO_KEEPALIVE | SO_ACCEPTCONN;
  int retry;
  int rval;

  noport:
  /*
   * data_addr = myctladdr;
   */
  bcopy(&myctladdr, &data_addr, sizeof (struct sckaddr_in));
  if (sendport)
    data_addr.sin_port = 0; /* let system pick one */
  if (data != -1)
    VOID xclose(data);
  for (retry = 0; retry < swaitmax; xsleep(swaitint), retry += swaitint)
    {
      data = xsocket(SOCK_STREAM, (struct sockproto *)0,
                     &data_addr, options);
      if (data >= 0 || (data != XEADDRINUSE && data != XENOBUFS))
        break;
    }
  if (data < 0) {
    x perror(data, "ftp: socket");
    return (1);
  }

  len = sizeof (data_addr);
  if ((rval = xsktaddr(data, (char *)&data_addr)) < 0) {
    x perror(rval, "ftp: xsktaddr");
    goto bad;
  }
  if (sendport) {
    a = (char *)&data_addr.sin_addr;
    p = (char *)&data_addr.sin_port;
#define UC(b) (((int)b)&0xff)

result =
    command("PORT %d,%d,%d,%d,%d,%d", 
    UC(a[0]), UC(a[1]), UC(a[2]), UC(a[3]), 
    UC(p[0]), UC(p[1]));
if (result == ERROR && sendport == -1) {
    sendport = 0;
    goto noport;
}

return (result != COMPLETE);

} return (0);

bad:
    VOID xclose(data), data = -1;
    return (1);
}

XFILE * dataconn(mode)
{
    char *mode;
    struct sckadr_in from;
    int s, fromlen = sizeof (from);
    s = xaccept(data, &from);
    if (s < 0) {
        xerror(s, "ftp: accept");
        VOID xclose(data), data = -1;
        return (XNULL);
    }
    return (xodopen(data, mode));
}

ptransfer(direction, bytes, t0, t1)
char *direction;
long bytes;
long *t0, *t1;
{
    long sec;
    sec = *t1 - *t0;
    if (sec <= 0)
        sec = 1;
    xprintf(xstdout,"%ld bytes %s in %ld seconds (%ld bytes/s)\n", 
       bytes, direction, sec, bytes / sec);
}

/*
Routines from here on are to use names introduced by 4.2 BSD.
*/

shutdown (fd, how)
int fd, how;
{
    xioctl (fd, SIOCDONE, &how);
}
/*
 * mp - Even if _doprnt is more wonderful than _mydoprnt for systems which
 * have _doprnt, using _doprnt is an incredible maintenance headache.
 * In any case, we should support the same functionality on all
 * systems.
 * Hence, may _doprnt rest in peace.
 */

#define _mydoprnt(format, argp, FILEp)
        char *format;
        int *argp;
        FILE *FILEp;

    xprintf(FILEp, format, *(argp+1), *(argp+2), *(argp+3),
            *(argp+4), *(argp+5));

#def bzero (what, size)
        register char *what;
        register int size;

    while (size-- > 0)
        *what++ = 0;

#def bcopy (from, to, size)
        register char *from, *to;
        register int size;

    while (size-- > 0)
        *to++ = *from++;

#def bcmp (left, right, size)
        register char *left, *right;
        register int size;

    while (size-- > 0)
        if (*left++ != *right++)
            if (0xff&(*--left) > 0xff&(*--right))
                return (1);
            else
                return (-1);

        return (0);

#def struct servent *gsbname (service, proto)
        char *service, *proto;

    static struct servent servstat;

    servstat.s_name = service;
    servstat.s_aliases = 0;
    if (xstrcmp (service, "ftp") == 0)
        servstat.s_port = (IPRT_FTP);
else
    if (xstrcmp (service, "telnet") == 0)
        servstat.s_port = (IPPORT_TELNET);
    else
        return (0);
    servstat.s_proto = proto;
    return (&servstat);
}

struct hostent *
ghbyname(host)

char *host;
{
    static struct hostent def;
    static struct in_addr defaddr;
    static char namebuf[128];
    extern long xrhost();

    defaddr.s_addr = xrhost(&host);
    if (defaddr.s_addr == -1)
        return (0);
    xstrcpy(namebuf, host);
    def.h_name = namebuf;
    def.h_addr = (char *)&defaddr;
    def.h_length = sizeof (struct in_addr);
    def.h_addrtype = AF_INET;
    def.h_aliases = 0;
    return (&def);
}

int inet_addr (host)
char *host;
{
    return (-1);
}
Operating system specific initialization for ftp client. 
...stuff the doesn't seem worth the effort of providing 
general mechanisms for.
/*
#include <xgenlib.h>
#include <wpwd.h>
#include <xspecial.h>
typedef int jmp_buf;
int errno = 0;
int figit = 0;
jmp_buf toplevel = 0;
#include <ftpvar.h>
jmp_buf *envptr = {0};

extern int fromatty; /* ftp started from terminal */
extern int _ttyinput; /* set in xoslib */
extern int conned; /* true if connected to server */
/* extern ast1conn(); */

extern struct passwd *pw;

clientinit()
{

fromatty = _ttyinput; /* true when used interactively */
/* set up routine to print out message and halt program */

/*
emt(SREX,ast1conn);
*/
toplevel = (int) & envptr;

}

/*
lostpeer()

{ extern XFILE *cout;
  extern int data;

  xoprintf( xstdout, "Lost Connection.\n"");
  if (conned) {
    if (cout != XNULL) {
      shutdown(xfileno(cout), 1+1);
      xclose(cout);
      cout = XNULL;
    }
    if (data >= 0) {
      shutdown(data, 1+1);
      xclose(data);
      data = -1;
    }
  }
  conned = 0;
}
57       }
58   }
59  
60  getbaddr()
61  {
62  

#ifndef lint
static char scssid[] = "@(#)main.c 1.14 8/28/85";
#endif

/ *
* FTP User Program -- Command Interface.
* /
#include "ftpc.h"

int intr();
extern int data;
extern char **xmkarglist();
extern char *strrchr();

xmain(argc, argv)
char *argv[];
{
    / *
    * Don't use register declarations in this procedure -- Zilog
    * S8000 setret() (alias setjmp()) can't abide by them.
    * /
    char *cp;
    int top;

    argv = (char **)0;
    sp = gsbname("ftp", "tcp");
    if (sp == 0) {
        xoprintf(xstderr, "ftp: ftp/tcp: unknown service\n");
        xexit(1);
    }
    doglob = 1;
    interactive = 1;
    autologin = 1;
    argc--, argv++;
    while (argc > 0 && **argv == '-') {
        for (cp = *argv + 1; *cp; cp++)
            switch (*cp) {
            case 'd':
                options |= SO_DEBUG;
                debug++;
                break;
            case 'v':
                verbose++;
                break;
            case 't':
                trace++;
                break;
            case 'i':
                interactive = 0;
                break;
            }
case 'n':
    autologin = 0;
    break;

case 'g':
    doglob = 0;
    break;

default:
    xoprintf(xstderr, 
        "ftp: %c: unknown option
", *cp); 
    xexit(1);
}

    argc--, argv++;
}

/*
 * Set up defaults for FTP.
*/
xstrcpy(typename, "ascii"), type = TYPE_A;
xstrcpy(formname, "non-print"), form = FORM_N;
xstrcpy(modename, "stream"), mode = MODE_S;
xstrcpy(structname, "file"), stru = STRU_F;
xstrcpy(byname, "8"), bytesize = 8;
if (fromatty)
    verbose++;
else
    interactive = 0;  /* not interactive, prompt off*/

/*
 * Set up the home directory in case we're globbing.
*/
if (argc > 0) {
    if (xsetjmp(toplevel))
        xexit(0);
    xint_term( intr );
    setpeer(argc + 1, argv - 1);
}

top = xsetjmp(toplevel);
if (top == 0 || top == 1) {
    xint_term( intr );
    top = 1;
}
for (;;) {
    cmdscanner(top);
    top = 1;
}

intr()
{
    /*
 Should send telnet IP, but ...
 for now just close data connection so that ftp will fall back
 into command mode.
 We still have to wait for other side to complete.
 */
113         xint_term( intr );
114     if( data != -1 )
115     {
116         shutdown( data, 2 );
117         xclose( data );
118         data = -1;
119         xprintf(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 = xstrchr(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             xprintf(xstdout,"ftp >");
156             xfflush(xstdout);
157         }
158         if (xgets(line) == XNULL) {
159             if( xfeof( xstdin ) ) {
160 /* quit on end of input */
161             quit();
162             xprintf( "\n" );
163             quit();
164         } else {
165             break;
166         }
167     }
168 }
if (line[0] == 0)
  break;
if ( argv )
  xdealglob( argv );
argv = xmklarglist( line, &argc );
c = getcmd(argv[0]);
if (c == (struct cmd *)&-1) {
xprintf(xstdout,"Ambiguous command\n");
  continue;
}
if (c == 0) {
xprintf(xstdout,"Invalid command\n");
  continue;
}
if (c->c_conn && !connected)
  xprintf(xstdout,"Not connected.\n");
  continue;
(*c->c_handler)(margc, argvv);
if (bell && c->c_bell)
xputchar(CTRL('G'));
if (c->c_handler != help)
  break;
xlongjmp(toplevel, 0);
}
struct cmd *
getcmd(name)
{
  register char *name;
  {

  register char *p, *q;
  register struct cmd *c, *found;
  register int nmatches, longest;

  /*
  ** convert command to lower case.
  */
  for( q = name ; *q ; ++q ) {
    if( isupper( *q ) )
      *q = _tolower( *q );
  }

  longest = 0;
  nmatches = 0;
  found = 0;
  for (c = cmdtab; p = c->c_name; c++) {
    for (q = name; *q == *p++; q++)
      if (*q == 0) /* exact match? */
        return (c);
    if (!q) {
      /* the name was a prefix */
      if (q - name > longest) {
        longest = q - name;
        nmatches = 1;
        found = c;
      } else if (q - name == longest)
        nmatches++;
225     }
226 }
227     if (nmatches > 1)
228         return ((struct cmd *)-1);
229     return (found);
230 }
231
232 #define HELPINDENT (sizeof ("directory"))
233
234 extern int NCMDS;
235 /*
236  * Help command.
237  * Call each command handler with argc == 0 and argv[0] == name.
238  */
239 help(argc, argv)
240     int argc;
241     char *argv[];
242 {
243     register struct cmd *c;
244
245     if (argc == 1) {
246         register int i, j, w;
247         int columns, width = 0, lines;
248
249         x fprintf(xstdout,
250             "Commands may be abbreviated. Commands are:\n\n"");
251         for (c = cmdtab; c < &cmdtab[NCMDS]; c++) {
252             int len = x strlen(c->c_name);
253
254             if (len > width)
255                 width = len;
256         }
257         width = (width + 8) &~ 7;
258         columns = 80 / width;
259         if (columns == 0)
260             columns = 1;
261         lines = (NCMDS + columns - 1) / columns;
262         for (i = 0; i < lines; i++) {
263             for (j = 0; j < columns; j++) {
264                 c = cmdtab + j * lines + i;
265                 x fprintf(xstdout,"%s", c->c_name);
266                 if (c + lines >= &cmdtab[NCMDS]) {
267                     x fprintf(xstdout,"\n");
268                     break;
269                 }
270                 w = x strlen(c->c_name);
271                 while (w < width) {
272                     w = (w + 8) &~ 7;
273                     x putchar('\t');
274                 }
275             }
276         }
277         return;
278     }
279     while (argc > 0) {
register char *arg;
arg = *++argv;
c = getcmd(arg);
if (c == (struct cmd *)-1)
    xoprintf(xstdout,"?Ambiguous help command %s\n", arg);
else if (c == (struct cmd *)0)
    xoprintf(xstdout,"?Invalid help command %s\n", arg);
else
    xoprintf(xstdout,"%-*s\t%*s\n", HELPINDENT,
        c->c_name, c->c_help);

*/
* Call routine with argc, argv set from args (terminated by 0).
*/
VARARGS2 */
call(routine, args)

int (*routine)();
int args;
{
    register int *argp;
    register int argc;
    for (argc = 0, argp = &args; *argp++ != 0; argc++)
    ;
    (*routine)(argc, &args);
.title dummy

genv::
gthen::
gtnba::
gtnbn::
gtnen::
gtpen::
gtsbp::
gtsen::
gpbnam::
gpbnunum::

rts    pc

.end
1 / * "(#)compat.h 1.9 4/15/85" */
2
3 /* added by billn */
4 #ifdef index /* system 3 or 5 */
5 #include <fcntl.h>
6 #define dup2(f,n) { close(n); fcntl(f, F_DUPFD, n);} 
7 #endif
8
9 #ifdef void
10 #define void int
11 #endif
12
13 #define VOID (void)
14
15 #ifndef SIGCHLD
16 #define SIGCHLD SIGCLD
17 #endif
18
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 inigroups(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 ftimelb;
36 #define gettimeofday(a,b) ( ftime (&ftimelb), 
37 (a)->tv_sec = ftimelb.time, (a)->tv_usec = ftimelb.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
/*
@(#)ftpd.h   1.2 4/11/85

Header files for generic server code.
*/

#include <xgenlib.h>
#include <xpwd.h>
#include <netdb.h>
typedef int jmp_buf;
#include <ftp.h>
#include <in.h>
#include <socket.h>
#include <xspecial.h>
#include "telnet.h"
define off_t long
extern int figit ;
extern long xpasstnet();
extern long xpassfnet();
define VOID figit = (int)
define renamecmd  cmdrename
/* Definitions for the TELNET protocol. */

#define IAC 255 /* interpret as command */
#define DONT 254 /* you are not to use option */
#define DO 253 /* please, you use option */
#define WONT 252 /* I won't use option */
#define WILL 251 /* I will use option */
#define SB 250 /* interpret as subnegotiation */
#define GA 249 /* you may reverse the line */
#define EL 248 /* erase the current line */
#define EC 247 /* erase the current character */
#define AYT 246 /* are you there */
#define AO 245 /* abort output--but let prog finish */
#define IP 244 /* interrupt process--permanently */
#define BREAK 243 /* break */
#define DM 242 /* data mark--for connect. cleaning */
#define NOP 241 /* nop */
#define SE 240 /* end sub negotiation */
#define SYNCH 242 /* for telfunc calls */

/* telnet options */

#define TNPBINARY 0 /* 8-bit data path */
#define TNECHO 1 /* echo */
#define TNPRCF 2 /* prepare to reconnect */
#define TNPSCA 3 /* suppress go ahead */
#define TNPNAMS 4 /* approximate message size */
#define TNPSTATUS 5 /* give status */
#define TNPMTM 6 /* timing mark */
#define TNPCTE 7 /* remote controlled transmission and echo */
#define TNPNAOL 8 /* negotiate about output line width */
#define TNPNAOP 9 /* negotiate about output page size */
#define TNPNAOCDR 10 /* negotiate about CR disposition */
#define TNPHTS 11 /* negotiate about horizontal tabstops */
#define TNPHTFD 12 /* negotiate about horizontal tab disposition */
#define TNPNOFFD 13 /* negotiate about formfeed disposition */
#define TNPVTS 14 /* negotiate about vertical tab stops */
#define TNPVTD 15 /* negotiate about vertical tab disposition */
#define TNPNAOFD 16 /* negotiate about output LF disposition */
#define TNPXASCII 17 /* extended ascii character set */
#define TNPLOUT 18 /* force logout */
#define TNPBM 19 /* byte macro */
#define TNPDET 20 /* data entry terminal */
#define TNPSPDUP 21 /* supdup protocol */
#define TNPENG 255 /* extended-options-list */

ifdef TELCMDS
char *telcmds[] = {
    "SB", "NOP", "DMARK", "BRK", "IP", "AO", "AYT", "EC",
    "EL", "CA", "SB", "WILL", "WONT", "DO", "DONT", "IAC",
};
endif

#ifndef TEOPTS
char *telopts[] = {
    "BINARY", "ECHO", "RCP", "SUPPRESS GO AHEAD", "NAME",
    "STATUS", "TIMING MARK", "RCTE", "NAOL", "NAOP",
    "NAOCR", "NAOHTS", "NAOHTD", "NAOPFD", "NAOVTS",
    "NAOVD", "NAOLFD", "EXTEND ASCII", "LOGOUT", "BYTE MACRO",
    "DATA ENTRY TERMINAL", "SUPDUP"
};
#endif
/*
 * MWP: 03/06/85
 * Make machines which have different sized ints and pointers happy.
 * (at least as far as the parser stack is concerned).
 * *********************************************************************/

typedef char * YYSTDEF;
#define YYSTYPE YYSTDEF
YYSTYPE copy();

extern struct sockaddr_in data_dest;
extern int logged_in;
extern int guest;
extern int logging;
extern int type;
extern int form;
extern int debug;
extern int timeout;
extern char hostname[];
extern char *globerr;
extern char *xghome;
extern int usedefault;
extern char **xglob();
static char **globargs = 0;
static char **rnf_glob = 0;
static char *username = 0;
static char *userpass = 0;

static int cmd_type = 0;
static int cmd_form = 0;
static int cmd_bytesz = 0;

char *xstrchr(),
# define S 267
# define T 268
# define SP 269
# define CRLF 270
# define COMMA 271
# define STRING 272
# define NUMBER 273
# define USER 274
# define PASS 275
# define ACCT 276
# define REIN 277
# define QUIT 278
# define PORT 279
# define PASV 280
# define TYPE 281
# define STRU 282
# define MODE 283
# define RETR 284
# define STOR 285
# define APPEND 286
# define MLFL 287
# define MAIL 288
# define MSND 289
# define MSOM 290
# define MSAM 291
# define MRSQ 292
# define MRCP 293
# define ALLO 294
# define REST 295
# define RNFR 296
# define RNTO 297
# define ABOR 298
# define DELE 299
# define CWD 300
# define LIST 301
# define NLST 302
# define SITE 303
# define STAT 304
# define HELP 305
# define NOOP 306
# define XMKD 307
# define XRMD 308
# define XPWD 309
# define XCOPY 310
# define LEXERR 311
#define yyclearin yychar = -1
#define yyerror yyerrflag = 0
extern int yychar;
extern short yyerrflag;
#ifndef YYMAXDEPTH
#define YYMAXDEPTH 150
#endif
#ifndef YYSTYPE
#define YYSTYPE int
#endif
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 { "MSRS", MSRS, 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", XWD, OSTR, 1,       "[ <sp> directory-name ]" },
167 { "LIST", LIST, OSTR, 1,       "[ <sp> path-name ]" },
168 { "NLST", NLST, OSTR, 1,       "[ <sp> path-name ]" },
    { "SITE", SITE, STR1, 0, "(get site parameters)" },
    { "STAT", STAT, OSTR, 0, "(get server status)" },
    { "HELP", HELP, OSTR, 1, "<sp> <string> <!-- -->" },
    { "NOOP", NOOP, ARGS, 1, "<!-- -->" },
    { "XMKD", XMKD, STR1, 1, "<sp> path-name" },
    { "XRMD", XRMD, STR1, 1, "<sp> path-name" },
    { "XPWD", XPWD, ARGS, 1, "(return current directory)" },
    { "XCUP", XCUP, ARGS, 1, "(change to parent directory)" },
    { XNULL, 0, 0, 0, 0 }
};

struct tab *
lookup(cmd)
{
    char *cmd;
    register struct tab *p;
    for (p = cmdbuf; p->name != XNULL; p++)
        if (xstrcmp(cmd, p->name) == 0)
            return (p);
    return (0);
}

/*
 * getline - a hacked up version of fgets to ignore TELNET escape codes.
 */
char *
getline(s, n, iop)
char *
getline(s, n, iop)
{ char *s;
  register XFILE *iop;
  
  register c;
  register char *cs;
  
  cs = s;
  while (--n > 0 && (c = xgetc(iop)) >= 0) {
    while (c == IAC) {
        c = xgetc(iop); /* skip command */
        c = xgetc(iop); /* try next char */
    }
    *cs++ = c;
    if (c=='\n')
        break;
  }
  if (c < 0 && cs == s)
      return (XNULL);
  *cs++ = '\0';
  if (debug) {
      xoprintf(xstderr, "FTPD: command: %s", s);
      if (c != '\n')
          xputc('\n', xstderr);
          xfflush(xstderr);
      }
  return (s);
static int
toolong()
{
    long now;
    extern long xtime();
    reply(421,
        "Timeout (%d seconds): closing control connection.", timeout);
    if (logging) {
        xoprintf(xstderr,
            "FTPD: User %s timed out after %d seconds at %ld",
            (username ? username : "unknown"), timeout, xtime());
        xfflush(xstderr);
    }
    xexit(1);
}

yylex()
{
    /*
     * Don't use register variables -- Zilog S8000 setret() can't cope.
     */
    static char cbuf[512];
    static int cpos, state;
    char *cp;
    struct tab *p;
    int n;
    char c;

    for (;;) {
        switch (state) {

            case CMD:
                if (getline(cbuf, sizeof(cbuf)-1, stdin) == XNULL) {
                    reply(221, "You could at least say goodbye.");
                    xexit(0);
                }
                if (xstrchr(cbuf, '\r')) {
                    cp = xstrchr(cbuf, '\r');
                    cp[0] = '\n'; cp[1] = 0;
                }
                if (xstrchr(cbuf, " '))
                    cpos = xstrchr(cbuf, " ') - cbuf;
                else
                    cpos = 4;
                c = cbuf[cpos];
                cbuf[cpos] = '\0';
                upper(cbuf);
                p = lookup(cbuf);
                cbuf[cpos] = c;
                if (p != 0) {
                    if (p->implemented == 0) {
                        nack(p->name);
                        xlongjmp(errcatch);
                        /* NOTREACHED */
state = p->state;
yylval = (YYSTYPE) p->name;
return (p->token);
}
break;

case OSTR:
  if (cbuf[cpos] == '\n') {
    state = CMD;
    return (CRLF);
  }
  /* FALL THRU */

  case STR1:
    if (cbuf[cpos] == ' ') {
      cpos++;
      state = STR2;
      return (SP);
    }
    break;

  case STR2:
    cp = &cbuf[cpos];
    n = xstrlen(cp);
    cpos += n - 1;
    /*
     * Make sure the string is nonempty and \n terminated.
    */
    if (n > 1 && cbuf[cpos] == '\n') {
      cbuf[cpos] = '\0';
      yylval = (YYSTYPE)copy(cp);
      cbuf[cpos] = '\n';
      state = ARGS;
      return (STRING);
    }
    break;

  case ARGS:
    if (isdigit(cbuf[cpos])) {
      cp = &cbuf[cpos];
      while (isdigit(cbuf[++cpos]))
        ;
      c = cbuf[cpos];
      cbuf[cpos] = '\0';
      yylval = (YYSTYPE)xatoi(cp);
      cbuf[cpos] = c;
      return (NUMBER);
    }
  switch (cbuf[cpos++]) {
  case '\n':
    state = CMD;
    return (CRLF);
  case ' ':
    break;
  ...
}
return (SP);

case ',',:
    return (COMMA);

case 'A':
    case 'a':
        return (A);

case 'B':
    case 'b':
        return (B);

case 'C':
    case 'c':
        return (C);

case 'E':
    case 'e':
        return (E);

case 'F':
    case 'f':
        return (F);

case 'I':
    case 'i':
        return (I);

case 'L':
    case 'l':
        return (L);

case 'N':
    case 'n':
        return (N);

case 'P':
    case 'p':
        return (P);

case 'R':
    case 'r':
        return (R);

case 'S':
    case 's':
        return (S);

case 'T':
    case 't':
        return (T);

break;
default:
    fatal("Unknown state in scanner.");
}
    state = CMD;
    yyerror("lexical error");
}
}

upper(s)
char *s;
{
    while (*s != '\0') {
        if (islower(*s))
            *s = _toupper(*s);
        s++;
    }
}

YYSTYPE
抄(s)
char *s;
{
    char *p;
    /* extern char *xmalloc(); */
    p = xmalloc((xstrlen(s) + 1));
    if (p == XNULL)
        fatal("Ran out of memory.");
    xstrcpy(p, s);
    return ((YYSTYPE)p);
}

help(s)
char *s;
{
    register struct tab *c;
    register int width, NCMDS;
    width = 0, NCMDS = 0;
    for (c = cmdtab; c->name != XNULL; c++) {
        int len = xstrlen(c->name);
        if (c->implemented == 0)
            len++;
        if (len > width)
            width = len;
        NCMDS++;
    }
    width = (width + 8) & 7;
    if (s == 0) {
        register int i, j, w;
        int columns, lines;
        lreply(214,
            "The following commands are recognized (* =>'s unimplemented.");
        columns = 76 / width;
if (columns == 0)
    columns = 1;
lines = (NCMDS + columns - 1) / columns;
for (i = 0; i < lines; i++) {
    xprintf(stdout," ");
    for (j = 0; j < columns; j++) {
        c = cmdtab + j * lines + i;
        xprintf(stdout,"%s%c", c->name,
            c->implemented ? ' ' : '*');
        if (c + lines >= &cmdtab[NCMDS])
            break;
        w = xstrlen(c->name);
        while (w < width) {
            xputchar(' ');
            w++;
        }
    }
    xprintf(stdout,"\r\n");
}
xfflush(stdout);
reply(214, "Direct comments to ftp-bugs%@s.", hostname);
return;
upper(s);
c = lookup(s);
if (c == (struct tab *)0) {
    reply(504, "Unknown command %s.", s);
    return;
}
if (c->implemented)
    reply(214, "Syntax: %s %s", c->name, c->help);
else
    reply(214, "%-%s\t%s; unimplemented.", width, c->name, c->help);
short yyexca[] = {
    -1, 1,
    0, -1,
    -2, 0,
};
#define YYNPROD 60
#define YYLAST 208
short yyact[] = {
    26, 54, 103, 149, 147, 145, 145, 105, 143, 105, 124,
    77, 63, 112, 88, 61, 59, 141, 57, 3, 4,
    5, 148, 25, 6, 146, 7, 8, 9, 11, 12,
    13, 144, 142, 99, 87, 86, 84, 83, 10, 140,
    28, 82, 81, 16, 17, 15, 14, 45, 44, 19,
    20, 21, 22, 23, 24, 139, 138, 137, 136, 135,
    134, 133, 132, 131, 128, 119, 108, 107, 106, 126,
    100, 98, 97, 127, 96, 104, 95, 92, 91, 52,
    51, 46, 102, 101, 94, 93, 90, 89, 85, 80,
    79, 78, 75, 76, 36, 35, 34, 33, 32, 31,
    30, 74, 29, 70, 125, 109, 65, 72, 71, 66,
    37, 67, 68, 53, 27, 111, 18, 73, 110, 69,
    64, 62, 60, 38, 39, 40, 41, 42, 43, 58,
short yypact[] = {
-1000, -256, -1000, -167, -169, -170, -171, -172, -173, -174,
-175, -1000, -1000, -1000, -1000, -1000, -1000, -1000, -222,
-189, -1000, -1000, -1000, -190, -191, -296, -1000, -255,
-257, -258, -262, -151, -158, -166, -263, -178, -179, -180,
-228, -233, -181, -235, -1000, -259, -1000, -182, -183, -192,
-193, -1000, -1000, -1000, -184, -185, -194, -1000, -196, -1000,
-198, -1000, -199, -238, -200, -186, -187, -1000, -267, -202,
-1000, -1000, -1000, -203, -1000, -1000, -204, -260, -1000,
-260, -1000, -1000, -260, -260, -1000, -260, -205, -260,
-264, -1000, -1000, -260, -1000, -1000, -1000, -264,
-1000, -195, -195, -265, -1000, -1000, -1000, -207,
-1000, -1000, -1000, -208, -209, -210, -211, -212, -213, -1000,
-214, -215, -231, -254, -239, -1000, -1000, -1000, -1000,
-1000, -1000, -1000, -1000, -1000, -1000, -1000, -1000,
-1000, -1000, -1000, -1000, -1000, -1000, -1000, -1000,
short yypgo[] = {
0, 136, 131, 130, 129, 122, 121, 120, 119, 117,
110, 105, 118, 116, 75, 104, 115, 114, 113};
short yyrl[] = {
0, 1, 1, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 3, 4, 5, 14, 6, 15, 15, 15, 7,
7, 7, 7, 7, 7, 7, 7, 8, 8, 8,
9, 9, 9, 11, 12, 16, 13, 17, 18, 10};
short yyrl2[] = {
0, 0, 2, 4, 4, 4, 4, 4, 4,
4, 5, 5, 5, 3, 9, 3, 5, 5, 3,
5, 1, 2, 2, 2, 2, 2, 2, 2,
2, 1, 1, 1, 1, 1, 1, 1, 1,
3, 1, 3, 1, 1, 3, 2, 1, 1, 1,
1, 1, 1, 1, 1, 1, 2, 5, 4, 0};
short yychk[] = {
-1000, -1, -2, 274, 275, 276, 279, 281, 282, 283,
294, 284, 285, 286, 302, 301, 299, 300, -13, 305,
306, 307, 308, 309, 310, 278, 256, -17, 296, 269,
269, 269, 269, 269, 269, 269, 269, -10, -10, -10,
-10, -10, -10, -10, -10, 270, 269, 270, -10, -10, -10,
-10, 270, 270, -18, 297, -10, -3, 272, -4, 272,
-5, 272, -6, 273, -7, 257, 260, 262, 263, -8,
261, 266, 265, -9, 267, 258, 259, 273, 269, 269,
269, 270, 269, 270, 269, 269, 269, 269, 272, 269,
short yydef[]={
1, -2, 2, 0, 0, 0, 0, 0, 0, 0, 0,
0, 59, 59, 59, 59, 59, 59, 59, 59, 21, 0,
0, 59, 59, 59, 59, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 22, 0, 24, 0, 0, 0, 0,
0, 29, 30, 56, 0, 0, 0, 0, 0, 31, 0, 32,
0, 0, 33, 0, 0, 0, 39, 41, 43, 44, 0,
47, 48, 49, 0, 50, 51, 52, 0, 0, 0, 0,
0, 14, 0, 16, 0, 0, 0, 0, 0, 0, 0,
0, 27, 28, 0, 0, 3, 4, 5, 6, 0, 0,
7, 0, 0, 0, 0, 0, 46, 34, 8, 9, 10, 0,
53, 54, 55, 0, 0, 0, 0, 0, 0, 0, 23,
0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0,
45, 11, 12, 13, 15, 17, 18, 20, 25, 26,
58, 57, 0, 0, 0, 0, 0, 0, 0, 35};
#endif lint

static char yaccpar_sccsid[] = "@(#)yaccpar 4.1 (Berkeley) 2/11/83";
#endif not lint

#define YYFLAG -1000
#define YYERROR goto yyerrlab
#define YYACCEPT return(0)
#define YYABORT return(1)

/* parser for yacc output */

#ifdef YYDEBUG
#endif

YYSTYPE yyv[YYMAXDEPTH] = { 0 }; /* where the values are stored */
int yynchar = -1; /* current input token number */
int yynerrs = 0; /* number of errors */
short yyerrflag = 0; /* error recovery flag */

yparse();

short yys[YYMAXDEPTH];
short yyj, yyn;
register YYSTYPE *yypt;
register short yystate, *yyps, yyn;
register YYSTYPE *yyv;
register short *yyxi;

yystate = 0;
yynchar = -1;
yynerrs = 0;
yyerrflag = 0;
yyps= &yys[-1];
yyv = &yyv[-1];

yystack:    /* put a state and value onto the stack */

#ifdef YYDEBUG
#endif

if( ++yyvps & yyis[YYMAXDEPTH] ) { yyerror("yacc stack overflow"); return
*yyvps = yystate;
++yyv;
*yyv = yyval;
}

yn = yypact[yystate];

if( yn<= YYFLAG ) goto yydefault; /* simple state */

if( yychar<0 ) if( (yychar=yylex())<0 ) yychar=0;
if( (yn == yychar)<0 || yn >= YYLAST ) goto yydefault;

if( yychk[ yn=yyact[ yn ] ] == yychar ){ /* valid shift */
    yychar = -1;
    yyval = yylval;
    yystate = yn;
    if( yyerrflag > 0 ) --yyerrflag;
    goto yystack;
}

yydefault:
    /* default state action */

if( (yn=yydef[yystate]) == -2 ){
    if( yychar<0 ) if( (yychar=yylex())<0 ) yychar = 0;
    /* look through exception table */

    for( yyxi=yyexca; (*yyxi! = (-1)) || (yyxi[1]!=yystate); yyxi += 2 )
    while( *(yyxi+=2) >= 0 ){
        if( *(yyxi == yychar ) break;
    }

    if( (yn = yyxi[1]) < 0 ) return(0); /* accept */

if( yn == 0 ){ /* error */
    /* error ... attempt to resume parsing */

    switch( yyerrflag ){
    case 0:    /* brand new error */
    yyerror("syntax error");
    yyerrlab;
        ++yynerrs;

    case 1:
    case 2: /* incompletely recovered error ... try again */
yyerrflag = 3;

    /* find a state where "error" is a legal shift action */

    while ( (yyps >= yys) ) {
        yyn = yypact[*yyps] + YYERCODE;
        if( yyn >= 0 && yyn < YYLAST && yychk[yyact[yyn]] == YYERRCOD
            yynstate = yyact[yyn]; /* simulate a shift of "error" */
            goto yynstack;
        }
        yyn = yypact[*yyps];

        /* the current yyps has no shift onn "error", pop stack */

    #ifdef YYDEBUG
    #endif

        --yyps;
        --yypv;
    }

    /* there is no state on the stack with an error shift ... abort

    yyabort:
        return(1);

    case 3: /* no shift yet; clobber input char */

    #ifdef YYDEBUG
    #endif

        if( yychar == 0 ) goto yyabort; /* don't discard EOF, quit */
        yynchar = -1;
        yynnewstate; /* try again in the same state */

    }

    /* reduction by production yyn */

    #ifdef YYDEBUG
    #endif

        yyyps = yyr2[yyn];
        yypvt = yypv;
        yypv = yyr2[yyn];
        yyval = yypv[1];
        yyn = yyn;
       /* consult goto table to find next state */

        yyn = yyrl[yyn];
        yyn = yyn + *yyps + 1;
        yyn = yyn;
        yynstate = yyact[yynj] ;
    switch(yym){

        case 2:
# line 75 "ftpcmd.y"
{
    if ( globargs )
        xdealglob( globargs );
    globargs = (char **)0;
} break;

case 3:
# line 83 "ftpcmd.y"
{
    int success;

    if ( logged_in )
        {
            reply(531, "Already logged in.");
            xfree( yypvt[-1] );
        }
    else if ((success = xinit_env(yypvt[-1], (char *)0, (char *)0)) <= 0)
        {
            guest = 0;
            reply(331, "Password required for %s.", yypvt[-1]);
            if ( username )
                xfree( username );
            username = yypvt[-1];
        }
    else if ( success < 0 )
        {
            /*
            Do we want to give out this informantion? */
            reply(530, "User %s unknown.", yypvt[-1]);
            xfree(yypvt[-1]);
            if ( username )
                xfree( username );
            username = (char *)0;
        }
    else
        {
            username = yypvt[-1];
            reply(230, "User %s logged in.", yypvt[-1]);
            logged_in = 1;
        }
} break;

case 4:
# line 121 "ftpcmd.y"
{
    int success;

    if ( !username )
        {
            reply(530, "Log in with user first.");
            xfree( yypvt[-1] );
        }
    else if ( logged_in )

    { reply(531, "Already logged in."); xfree( yypvt[-1] ); }
    } else if (success == xinit_env(username, yypvt[-1], (char *)0) == 0 )
    { guest = 0; reply(331, "Account required for %s.", username); if( userpass )
        xfree( userpass ); userpass = yypvt[-1];
    } else if ( success > 0 )
    { userpass = yypvt[-1]; reply(230, "User %s logged in.", username);
        logged_in = 1;
    } else /* success < 0 (tricotomy) */
    { reply(530, "Login failed."); xfree( yypvt[-1] );
    } break;
  case 5:
  # line 158 "ftpcmd.y"
  { int success;
    if( !username )
    { reply(530, "Log in with user first."); xfree( yypvt[-1] );
    } else if( logged_in )
    { reply(531, "Already logged in."); xfree( yypvt[-1] );
    } else if ( (success == xinit_env(username, userpass, yypvt[-1])) <= 0 )
    { guest = 0; reply(530, "Login incorrect."); xfree( yypvt[-1] );
    } else if ( success > 0 )
    { reply(230, "User %s logged in.", username); logged_in = 1;
        xfree( yypvt[-1] );
    }
case 6:
    switch (cmd_type) {
      case TYPE_A:
        if (cmd_form == FORM N) {
          reply(200, "Type set to A.");
          type = cmd_type;
          form = cmd_form;
        } else
          reply(504, "Form must be N.");
        break;
      case TYPE_E:
        reply(504, "Type E not implemented.");
        break;
      case TYPE_I:
        reply(200, "Type set to I.");
        type = cmd_type;
        break;
      case TYPE_L:
        if (cmd_bytesz == 8) {
          reply(200, "Type set to L (byte size 8).);
          type = cmd_type;
        } else
          reply(504, "Byte size must be 8.");
      break;
    }
    break;
    case 8:
    # line 224 "ftp.c"
    switch ((int)yypvt[-1]) {
      case STRU_F:
        reply(200, "STRU F ok.");
        break;
      default:
        reply(502, "Unimplemented STRU type.");
    }
    break;
    case 9:
    # line 236 "ftp.c"
    {

switch ((int)yypvt[-1]) {
    case MODE_S:
        reply(200, "MODE S ok.");
        break;
    default:
        reply(502, "Unimplemented MODE type.");
}

case 10:
    # line 248 "ftpcmd.y"
    {
        ack(yypvt[-3]);
    } break;

case 11:
    # line 252 "ftpcmd.y"
    {
        if (yypvt[-3] && yypvt[-1] != XNULL)
            retrieve(0, yypvt[-1]);
    } break;

case 12:
    # line 257 "ftpcmd.y"
    {
        if (yypvt[-3] && yypvt[-1] != XNULL)
            store(yypvt[-1], "w");
    } break;

case 13:
    # line 262 "ftpcmd.y"
    {
        if (yypvt[-3] && yypvt[-1] != XNULL)
            store(yypvt[-1], "a");
    } break;

case 14:
    # line 267 "ftpcmd.y"
    {
        if (yypvt[-1])
            retrieve( LS, "".cloudflare.com");
    } break;

case 15:
    # line 272 "ftpcmd.y"
    {
        if (yypvt[-3] && yypvt[-1] != XNULL)
            retrieve( LS ARGS, yypvt[-1]);
        if (yypvt[-1] != XNULL)
            xfree(yypvt[-1]);
    } break;

case 16:
    # line 279 "ftpcmd.y"
    {
        if (yypvt[-1])
            retrieve( LS LONG, "".cloudflare.com");
    } break;

case 17:
    # line 284 "ftpcmd.y"
    {

if (yypvt[-3] && yypvt[-1] != XNULL)
  retrieve( LSNULL_ARG, yypvt[-1]);
if (yypvt[-1] != XNULL)
  xfree(yypvt[-1]);
}
break;
case 18:
  # line 291 "ftpcmd.y"
{
  if (yypvt[-3] && yypvt[-1] != XNULL)
    delete(yypvt[-1]);
}
break;
case 19:
  # line 296 "ftpcmd.y"
{
  if (yypvt[-1])
    xchdir((char *)0, HOME_DIR);
}
break;
case 20:
  # line 301 "ftpcmd.y"
{
  if (yypvt[-3] && yypvt[-1] != XNULL)
    cwd(yypvt[-1], FILE_NAME);
}
break;
case 22:
  # line 307 "ftpcmd.y"
{
  help(0);
}
break;
case 23:
  # line 311 "ftpcmd.y"
{
  help(yypvt[-1]);
}
break;
case 24:
  # line 315 "ftpcmd.y"
{
  ack(yypvt[-1]);
}
break;
case 25:
  # line 319 "ftpcmd.y"
{
  if (yypvt[-3] && yypvt[-1] != XNULL)
    makedir(yypvt[-1]);
}
break;
case 26:
  # line 324 "ftpcmd.y"
{
  if (yypvt[-3] && yypvt[-1] != XNULL)
    removedir(yypvt[-1]);
}
break;
case 27:
  # line 329 "ftpcmd.y"
{
  if (yypvt[-1])
    pwd();
}
break;
case 28:
    # line 334 "ftpcmd.y"
    {
        if (yypvt[-1] &amp; !inappropriate_request(".."))
            cwd("..", UP_DIRECTORY);
    } break;

case 29:
    # line 339 "ftpcmd.y"
    {
        reply(221, "Goodbye.");
        xexit(0);
    } break;

case 30:
    # line 344 "ftpcmd.y"
    {
        yyrerok;
    } break;

case 35:
    # line 363 "ftpcmd.y"
    {
        register char *a, *p;
        a = (char *)&data_dest.sin_addr;
        a[0] = (int)yypvt[-10]; a[1] = (int)yypvt[-8];
        a[2] = (int)yypvt[-6]; a[3] = (int)yypvt[-4];
        p = (char *)&data_dest.sin_port;
        p[0] = (int)yypvt[-2]; p[1] = (int)yypvt[-0];
        data_dest.sin_family = AF_INET;
    } break;

case 36:
    # line 376 "ftpcmd.y"
    {
        yyval = (YYSTYPE)FORM_N;
    } break;

case 37:
    # line 380 "ftpcmd.y"
    {
        yyval = (YYSTYPE)FORM_T;
    } break;

case 38:
    # line 384 "ftpcmd.y"
    {
        yyval = (YYSTYPE)FORM_C;
    } break;

case 39:
    # line 390 "ftpcmd.y"
    {
        cmd_type = TYPE_A;
        cmd_form = FORM_N;
    } break;

case 40:
    # line 395 "ftpcmd.y"
    {
        cmd_type = TYPE_A;
        cmd_form = (int)yypvt[-0];
    } break;
case 41:
# line 400 "ftpcmd.y"
{
    cmd_type = TYPE_E;
    cmd_form = FORM_N;
} break;

case 42:
# line 405 "ftpcmd.y"
{
    cmd_type = TYPE_E;
    cmd_form = (int)yypt[-0];
} break;

case 43:
# line 410 "ftpcmd.y"
{
    cmd_type = TYPE_I;
} break;

case 44:
# line 414 "ftpcmd.y"
{
    cmd_type = TYPE_L;
    cmd_bytesz = 8;
} break;

case 45:
# line 419 "ftpcmd.y"
{
    cmd_type = TYPE_L;
    cmd_bytesz = (int)yypt[-0];
} break;

case 46:
# line 425 "ftpcmd.y"
{
    cmd_type = TYPE_L;
    cmd_bytesz = (int)yypt[-0];
} break;

case 47:
# line 432 "ftpcmd.y"
{
    yyval = (YYSTYPE)STRU_F;
} break;

case 48:
# line 436 "ftpcmd.y"
{
    yyval = (YYSTYPE)STRU_R;
} break;

case 49:
# line 440 "ftpcmd.y"
{
    yyval = (YYSTYPE)STRU_P;
} break;

case 50:
# line 446 "ftpcmd.y"
{
    yyval = (YYSTYPE)MODE_S;
} break;

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:
# line 512 "ftpcmd.y"
{
    char *from = 0, *renamefrom;
    if (yypvt[-3] && yypvt[-1])
        from = renamefrom(yypvt[-1]);
    rnf_glob = globargs;
    yyval = (YSTYPE)from;
    } break;
}
case 58:
# line 523 "ftpcmd.y"
{
    yyval = yypvt[-1];
    } break;
case 59:
# line 529 "ftpcmd.y"
{
    if (logged_in)
        yyval = (YSTYPE)1;
    else {
        reply(530, "Please login with USER and PASS.");
        yyval = (YSTYPE)0;
    }
    } break;
}
goto yystack; /* stack new state and value */
#ifndef lint
static char ssccsid[] = "@(#)/ftp.c 1.16 7/29/85"
#endif

/*
 * FTP server.
 */
#include "ftp.h"

extern long xpassstnet(), xpassfnet();
extern char version[];
extern FILE *xodopen();
/*
extern int fclose();
*/
extern char *xerror();
extern int xclose();
extern char **xmkarglist();
extern char **xglob();
extern char **globerr;

struct sckadr_in ctrl_addr = { AF_INET };
struct sckadr_in data_source = { AF_INET };
struct sckadr_in data_dest = { AF_INET };
struct sckadr_in his_addr = { AF_INET };

struct hostent *hp = 0;
int data = 0;
#endif zilog
ret_buf errcatch;
#else
jmp_buf errcatch = { 0 };
#endif
int logged_in = 0;
int debug = 0;
int timeout = 0;
int logging = 0;
int guest = 0;
int type = 0;
int form = 0;
int stru = 0;        /* avoid C keyword */
int mode = 0;
int bytesize = 0;
int usedefault = 1;   /* for data transfers */
char hostname[64] = {0};
char *remotehost = (char *)0;
struct servent *sp = (struct servent *)0;

/*
 * Timeout intervals for retrying connections
 * to hosts that don't accept PORT cmds. This
 * is a kludge, but given the problems with TCP...
 */
#define SWAITMAX 90  /* wait at most 90 seconds */
#define SWAITINT 5   /* interval between retries */
int swaitmax = SWAITMAX;
int swaitint = SWAITINT;

int lostconn();
FILE *dataconn();
char *ntoa();

ftpdoit( s, from )
/*
start of generic ftp demon code
*/

int s;
struct sckadr_in *from;
{
    if (logging)
        dolog(&his_addr);
    xdup2(s, 0);
    if( s != 0 )
        xclos(e(s);
    xdup2(0, 1);
    xodopen( 0, "r" );
    xodopen( 1, "w" );
    /* do telnet option negotiation here */
    /*
    * Set up default state
    */
    data = -1;
type = TYPE_A;
form = FORM_N;
stru = STRU_F;
mode = MODE_S;
bytezize = 8;
sp = gsbname("ftp", "tcp");
    if (sp == 0) {
        xprintf(xstderr, "ftp: ftp/tcp: unknown service\n");
        xexit(1);
    }
}
xghname(hostname, sizeof (hostname));
ctrl_addr.sin_port = xhtons(sp->s_port);
data_source.sin_port = xhtons(sp->s_port - 1);
reply(220, "%s FTP server (%s) ready.", hostname, version);
for (; ; ) {
    xsetjmp(errcatch);
    if( logging )
        {  
        xprintf( xstderr, "calling yyparse\n");
        xfflush( xstderr );
        }
    yyparse();
}

}
fatal("Connection closed.");

retriever(cmd, name)
int cmd;
char *name;
{
    XFILE *fin, *dout;
    int inod;
    int (*closefunc)();
    int omode;
    struct ftp_attr attributes;
    char *argv[2];
    char **argv2;
    char **argv3;
    char *pt;
    if (cmd == 0) {
        /*
         * simple file
         */
        omode = XFREAD;
        attributes.rep_type = type;
        attributes.format = form;
        attributes.structure = stru;
        attributes.trans_mode = mode;
        attributes.byte_sz = bytesize;
        inod = xftpopen( name, omode, FILE_NAME, &attributes );
        if (inod < 0) {
            reply(550, "%s: %s.", name, xerror( inod ));
            return;
        }
        fin = xodopen( inod, "r");
        if (fin == XNULL) {
            reply(550, "xodopen failed.");
            return;
        }
        closefunc = xclose;
    } else {
        /*
         * we are to generate a pseudo file,
         * at the moment some form of ls => call xls after opening
         * data connection.
         */
    }
    dout = dataconn(name, (off_t)0, "w");
    if (dout == XNULL)
        goto done;
    if (cmd)
    {
        /*
         * a psuedo file object ( ls, ls -lg)
         */
        if (xstrlen(name))
        {
        }
name may require globbing (for remote globbing).

argvl[0] = name;
argvl[1] = (char *)0;
argv2 = xglob( argvl );
if( argv2 == XNULL || globerr )
{
    xclose( xfileno(dout) );
data = -1;
    reply( 500, "Remote glob failed. %s", globerr );
    if( argv2 != XNULL )
        xdealglob( argv2 );
    return;
}
argv3 = argv2;
for( pt = *argv3++; pt; pt = *argv3++ )
{
    xls( xfileno(dout), pt, cmd );
}
  xdealglob( argv2 );
else
{
    xls( xfileno(dout), name, cmd );
    xclose( xfileno(dout) );
data = -1;
    reply(226, "Transfer complete.");
    return;
}
else if (send data(name, fin, dout))
{
    xclose( xfileno(dout) ), data = -1;
}
else
{
    reply(226, "Transfer complete.");
    xclose( xfileno(dout) ), data = -1;
}
done:
xclose( xfileno(fin) );

store(name, append)
char *name, *append;
{
    XFILE *fout, *din;
    int outod;
    int omode;
    int (*closefunc)(), dochown = 1;
    struct ftp_attr attributes;
    omode = XWRITE | XFCREAT | ( ( *append == 'a' )? XFAPPEND : XFTRUNC );
    attributes.rep_type = type;
    attributes.format = form;
attributes->structure = stru;
attributes->trans_mode = mode;
attributes->byte_sz = bytesize;
outod = xftppopen( name, omode, FILE_NAME, &attributes );
if( outod < 0 )
{
    reply(550, "%s: %s.", name, xerror( outod ) );
    return;
}
fout = xodopen( outod, "w" ), closefunc = xclose;
if (fout == XNULL) {
    reply(550, "xodopen failed.");
    return;
}

din = dataconn(name, (off_t)-1, "r");
if (din == XNULL)
    goto done;
if (receive_data(name, din, fout))
{
}
else
{
    reply(226, "Transfer complete.");
}
xclose( xfileno(din)), data = -1;
done:
VOID xchown(name, FILE_NAME );
xclose( xfileno(fout) );

getdatasock(mode)
char *mode;
{
    int s;
    int retry;
    if (data >= 0)
        return (data);
    data_source.sin_family = AF_INET;
    for (retry = 0; retry < swaitmax; xsleep (swaitint), retry += swaitint)
    {
        s = xsocket( SOCK_STREAM, 0, &data_source,
            SO_KEEPALIVE|SO_REUSEADDR);
        /* GAP 7/25/85: REUSEADDR fixes simultaneous xfer bug */
        if (s >= 0 || ( s != XEADDRINUSE & s != XENOBUFS))
            break;
    }
    if (s < 0)
        xerror( s, "getdatasock" );
    return ( s );

XFILE *
dataconn(name, size, mode)
char *name;
off_t size; /* no longer used */
char *mode;
{
    char sizebuf[32];
    FILE *file;
    int retry = 0;
    int s;
    int rval;

    if (data >= 0) {
        reply(125, "Using existing data connection for %s.", name);
        usedefault = 1;
        return (xodopen(data, mode));
    }
    if (usedefault)
        xcbcopy(&his_addr, &data_dest, sizeof(struct sockaddr));
    usedefault = 1;
    s = getdatasock(mode);
    if (s < 0) {
        reply(425, "Can't create data socket (%s,%d): %s.",
        ntoa(data_source.sin_addr.s_addr),
        ntohs(data_source.sin_port),
        xerror(s));
        return (XNULL);
    }
    reply(150, "Opening data connection for %s (%s,%d).", 
    name, ntoa(data_dest.sin_addr.s_addr),
    ntohss(data_dest.sin_port));
    data = s;
    while ((rval = xconnect(data, &data_dest)) < 0) {
        if (rval == XEADDRINUSE & retry < swaitmax) {
            xsleep(swaitint);
            retry += swaitint;
            continue;
        }
        reply(425, "Can't build data connection: %s.",
        xerror(rval));
        VOID xclose(data);
        data = -1;
        return (XNULL);
    }
    file = xodopen( data, mode );
    return (file);
}

/*
 * Tranfer the contents of "instr" to
 * "outstr" peer using the appropriate
 * encapsulation of the date subject
 * to Mode, Structure, and Type.
 * *
 * NB: Form isn't handled.
 */
send_data(name, instr, outstr)
    char *name;
XFILE *instr, *outstr;
{
    long rval;
    rval = xpasstnet( instr, outstr );
    if( rval == XEOPNOTSUPP )
    {
        reply(504,"Unimplemented TYPE %d in send_data", type);
        return( 1 );
    }
    else if ( rval < 0 )
    {
        reply(550, "%s: %s.", name, xerror( rval ) );
        return( 1 );
    }
    return( 0 );
}

/*
 * Transfer data from peer to
 * "outstr" using the appropriate
 * encapsulation of the data subject
 * to Mode, Structure, and Type.
 * N.B.: Form isn't handled.
 */
receive_data(name, instr, outstr)
    char *name;
    XFILE *instr, *outstr;
{
    long rval;
    rval = xpasstnet( instr, outstr );
    if( rval == XEOPNOTSUPP )
    {
        reply(504, "TYPE E not implemented.");
        return( 1 );
    }
    else if ( rval < 0 )
    {
        reply(550, "%s: %s.", name, xerror( rval ) );
        return( 1 );
    }
    return( 0 );
}

fatal(s)
    char *s;
{
    reply(451, "Error in server: %s\n", s);
    reply(221, "Closing connection due to server error.");
    xexit(1);
}

#ifdef zilog
reply(n, s, a1, a2, a3, a4, a5, a6)
int n;
char *s;
int a1, a2, a3, a4, a5, a6;
{
    int args=a1, aa2=a2, aa3=a3, aa4=a4, aa5=a5, aa6=a6;
    #else
    reply(n, s, args)
    int n;
    char *s;
    #endif
    xoprintf(xstdout,"%d ", n);
    _mydoprnt(s, &args, xstdout);
    xoprintf(xstdout,"\r\n");
    xfflush(xstdout);
    if (debug) {
        xoprintf(xstderr, "<--- %d ", n);
        _mydoprnt(s, &args, xstderr);
        xoprintf(xstderr, "\n");
        xfflush(xstderr);
    }
}
#endif

#ifdef zilog
reply(n, s, a1, a2, a3, a4, a5, a6)
int n;
char *s;
int a1, a2, a3, a4, a5, a6;
{
    int args=a1, aa2=a2, aa3=a3, aa4=a4, aa5=a5, aa6=a6;
    #else
    reply(n, s, args)
    int n;
    char *s;
    #endif
    xoprintf(xstdout,"%d-", n);
    _mydoprnt(s, &args, xstdout);
    xoprintf(xstdout,"\r\n");
    xfflush(xstdout);
    if (debug) {
        xoprintf(xstderr, "<--- %d-", n);
        _mydoprnt(s, &args, xstderr);
        xoprintf(xstderr, "\n");
    }
}
#endif

replystr(s)
char *s;
{
    xoprintf(xstdout,"%s\r\n", s);
    xfflush(xstdout);
    if (debug)
        xoprintf(xstderr, "<--- %s\n", s);
}
ack(s)
  char *s;
  reply(200, "%s command okay.", s);
}
nack(s)
  char *s;
  reply(502, "%s command not implemented.", s);
}
yyerror( message )
  char *message;
  reply(500, "Command not understood: %s.", message);
  xlongjmp( errcatch, 1 );
delete(name)
  char *name;
  int rval;
  if ((rval = xunlink(name, FILE_NAME )) < 0) {
    reply(550, "%s: %s.", name, xerror( rval ));
    return;
  }
  ack("DELE");
cwd(path, special)
  char *path;
  int special;
  int rval;
  if ((rval = xchdir(path, special )) < 0) {
    reply(550, "%s: %s.", path, xerror( rval ));
    return;
  }
  ack("CWD");
mkdir(name)
  char *name;
  int rval;
  if ((rval = xmkdir(name, FILE_NAME )) < 0) {
    reply(550, "%s: %s.", name, xerror( rval ));
    return;
  }
  VOID xchown(name, FILE_NAME );
  ack("MKDIR");
removedir(name)
char *name;
{
int rval;
if (( rval = xrmdir(name, FILE_NAME)) < 0) {
    reply(550, "%s: %s.", name, xerror( rval ) );
    return;
}
ack("RMDIR");
pwd()
{
    char path[MAXPATHLEN + 1];
    int success;
    success = xpwd( path, MAXPATHLEN + 1, PWD );
    if ( success < 0 ) {
        reply(451, "working directory not available.");
        return;
    }
    reply(251, "\"%s\\" is current directory.", path);
}
char *
renamefrom(name)
char *name;
{
int rval;
if ( (rval = xaccess( name, FILE_NAME, 0 ) ) < 0 ) {
    reply(550, "%s: %s.", name, xerror( rval ) );
    return ((char *)0);
}
    reply(350, "File exists, ready for destination name");
    return (name);
}
renamecmd(from, to)
char *from, *to;
{
int rval;
if ((rval = xrename(from, FILE_NAME, to, FILE_NAME)) < 0) {
    reply(550, "rename: %s.", xerror( rval ) );
    return;
}
ack("RNTO");

/*
 * Test pathname for guest-user safety.
 */
inappropriate_request(name)
{
    int depth = 0, length, size;
    register char *p, *s;
    length = strlen(name) : 0;
    /*
    This functionality probably belongs in xftpopen,
    but for now.
    */
    return(0);
}

/*
* Convert network-format internet address
* to base 256 d.d.d.d representation.
*/
char *
ntoa(in)
{
    struct in_addr in;
    static char b[18];
    register char *p;
    p = (char *)&in;
    #define UC(b) (((int)b)&0xff)
    xprintf(b, "%d.%d.%d.%d", UC(p[0]), UC(p[1]), UC(p[2]), UC(p[3]));
    return(b);
}

dolog(sin)
{
    struct sockaddr_in *sin;
    char saddr[16], *ntoa();
    struct hostent *hp = gbandaddr(&sin->sin_addr,
    sizeof (struct in_addr), AF_INET);
    char *remotehost;
    long t;
    long xtime();
    if (hp)
        remotehost = hp->h_name;
    else
        remotehost = ntoa(sin->sin_addr.s_addr);
    t = xtime();
    xprintf(stderr,"FTPD: connection from %s at %ld",
            remotehost, t);
    xfflush(stderr);
    /*
    * Even if _doprint is more wonderful than _mydoprint for systems which
    * have _doprint, using _doprint is an incredible maintenance headache.
    * In any case, we should support the same functionality on all
    */
systems.
Hence, may _doprnt rest in peace.

*/

mydoprnt(format, argp, FILEp)
char *format;
int *argp;
FILE *FILEp;
{
xprintf(FILEp, format, *argp, *(argp+1), *(argp+2), *(argp+3),
       *(argp+4), *(argp+5));
}

struct servent *gsbname (service, proto)
char *service, *proto;
{
    static struct servent servstat;
    servstat.s_name = service;
    servstat.s_aliases = 0;
    if (xstrcmpl (service, "ftp") == 0)
        servstat.s_port = (IPRT_FTP);
    else
        if (xstrcmp (service, "telnet") == 0)
            servstat.s_port = (IPPORT_TELNET);
        else
            return (0);
    servstat.s_proto = proto;
    return (&servstat);
}

extern char *xraddr();
extern struct hostent *ghbname ();

struct hostent *
ghbaddr(addr)
struct in_addr *addr;
{
    char *name;
    if ((name = xraddr (addr -> s_addr)) == 0)
        return (0);
    return (ghbname(name));
}

extern long xrhost ();

struct hostent *
ghbname(host)
char *host;
{
    static struct hostent def;
    static struct in_addr defaddr;
    static char namebuf[128];
defaddr.s_addr = xrhost(&host);
if (defaddr.s_addr == -1)
    return (0);
xstrncpy(namebuf, host);
def.h_name = namebuf;
def.h_addr = (char *)&defaddr;
def.h_length = sizeof (struct in_addr);
def.h_addrtype = AF_INET;
def.h_aliases = 0;
return (&def);
/*
 * FTP server.
 */

#include <xgenlib.h>
#include <xspecial.h>
#include <netdb.h>
#include <in.h>
#include <xpwd.h>
#include <fcs.h>
#include <ftp.h>
#include <socket.h>

extern char version[];
extern int xsmain();
extern int lostconn();

extern struct sckadr in ctrl_addr;
struct _xiobuf_xiob[_XNFILE] = {0};
struct passwd *password = {0};
struct passwd *pw = &password;
struct _rcb _rcb[_XNFILE] = {0};                        /* RSX - CONTROL BLOCK */
typedef int jmp_buf;
extern int errcatch;
jmp_buf *envptr = {0};
int xmodname();
int hash = 0;
int errno = 0;
int figit = 0;
int _brk = 0;                                          /* USED BY C RTS ALLOC & FREE */
char luntbl[32] = {0};                                 /* used to maintain LUN */
extern char *inprm[];
char *xghome = (char *)0;
extern struct hostent *hp;

extern int logged_in ;
extern int debug ;
extern int timeout ;
extern int logging ;
extern int guest ;
extern struct servent *sp ;

main(argc, argv)
    int argc;
    char *argv[];
{
    int ctrl, s, options = 0;
    char *cp;
    int rval;
    int rval2;
    int buf[16];                                         /* 16 word bufferd used for task parameters */
int maxlun = 0;

emit(GTSK,buf);
_brk = buf[13];
maxlun = buf[8];
ppasc(pw->cur_uic, buf[7]);
ppasc(pw->login_uic, buf[15]);
emt(GLUN,1,buf); /* get lun information */
xstrncpy(pw->log_dev,buf,2); /* get phy. device name */
pw->log_dev[2] = *((char *)buf + 2) + 060; /* get unit no. */
pw->log_dev[3] = '\0'; /* make it string */
xstrncpy(pw->cur_dev,pw->log_dev);
while( maxlun ){
    if( emit(GLUN, maxlun, buf ) > 0 )
        --maxlun;
}
for( rval = 0; rval < _XNFILE ; ++ rval )
    rcb[rval].flags = RFREE;
sp = gsbyname("ftp","tcp");
errcatch = (int ) & envptr;
if (sp == 0) {
xprintf(xstderr, "ftp: ftp/tcp: unknown service\n");
exexit(1);
}
ctrl_addr.sin_port = xhtons(sp->s_port);
ctrl_addr.sin_family = AF_INET;
options = ( SO_ACCEPTCONN | SO_KEEPALIVE );
debug = 0;
argc--, argv++;
while (argc > 0 && *argv[0] == '-') {
    for (cp = &argv[0][1]; *cp; cp++) switch (*cp) {
    case 'v':
        debug = 1;
        break;
    case 'd':
        debug = 1;
        options |= SO_DEBUG;
        break;
    case 'l':
        logging = 1;
        break;
    case 't':
        timeout = xatoi(++cp);
        goto nextopt;
        break;
    default:
        xprintf(xstderr, "Unknown flag -%c ignored.\n", *cp);
        break;
    }
}
nextopt:
    argc--, argv++;
}
    rval = xdopen("TI:",XFREAD|XFWRITE|XFASCII, FILE_NAME);
    xdup2(rval,1);
    xdup2(1,2);
    xdupopen(2,"w");
    getclient( SOCK_STREAM, (struct sockproto *)0,
        &ctrl_addr, options, xsmain);
}

xsmain( s, from )

/*
RSX specific ftp demon start up actions, calls generic code.
Operating systems which require the user's id to be established
before the process is started may provide two versions of this
module, one where logged_in is set to zero and one where it
is set to 1.
Otherwise, this is where to determine if the user has been authenticated
or not.
*/

int s;
struct sckadr_in *from;
{
    logged_in = 0;
    /* Until xprintf is available... */
    s = (int)_xiob[s]._sys_id;
    /* */
    ftpdoit( s, from );
}
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;

#else define PAD 3
#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;
if (efmt){
    *cp++ = 'E';
    if(base<0){*cp++ = '-'; base= -base;}
    else *cp++ = '+';
    if((len = 1tos((long)base,cp,10))<PAD){ /* left pad */
        movmem(cp,cp+PAD-len,len+1);
        setmem(cp,PAD-len,'0');
    }
}
#else
    return xstrlen(sbuff);
#endif

/* increased accuracy power of ten table */
static unsigned int pgiten[]={

57    0X0000,0X0000,0X0000,0X4024, /* le1 */
58    0X0000,0X0000,0X0000,0X4059, /* le2 */
59    0X0000,0X0000,0X8800,0X40C3, /* le4 */
60    0X0000,0X0000,0XD784,0X4197, /* le8 */
61    0X8000,0X37E0,0XC379,0X4341, /* le16 */
62    0X6E17,0X8505,0XB885,0X4693, /* le32 */
63    0XFF9F,0X193F,0X4F03,0X4D38, /* le64 */
64    0XD3D3,0X9F30,0X7748,0X5A82, /* le128 */
65    0XBF3F,0X7F73,0X4FDD,0X7515 /* le256 */
66    
67    static double *pgten=pgten;
68
69    static unsigned int plten[]={
70    0X9999,0X9999,0X9999,0X3FB9, /* le-1 */
71    0X147B,0X47A1,0X7AE1,0X3F84, /* le-2 */
72    0X432D,0XEB1C,0X36E2,0X3F1A, /* le-4 */
73    0X8C3A,0XE230,0X798E,0X3E45, /* le-8 */
74    0X89EC,0X97DB,0XD2B2,0X3C9C, /* le-16 */
75    0X7A32,0XDA58,0XFF63,0X3949, /* le-32 */
76    0X7A3C,0X44D4,0X0FFF,0X32A5, /* le-64 */
77    0X979A,0XCF8C,0XBA08,0X255B, /* le-128 */
78    0X6F40,0X64AC,0X0628,0X0AC8 /* le-256 */
79    
80    static double *plten=plten;
81
82    static double ZERO=0.0,ONE=1.0,TEN=10.0;
83
84    static d_scale(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=ONE){
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+=_scale(&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     cp=string;
143     if(val<ZERO){
144         val= -val;
145         *cp++ = '-';
146     }
147     if(iplace<1){
148         *cp++ = '0';
149         *cp++ = '.';
150         fplace+=iplace;
151         if(fplace<0){iplace-=fplace;fplace=0;}
152         while(iplace++<0)*cp++ = '0';
153     } else {
154         do {
155             j=val;
156             *cp++ = j+'0';
157             val=(val-j)*TEN;
158         } while(--iplace);
159         if(fplace)*cp++ = '.';
160     }
161     while(fplace--){
162         j=val;    /* get the integer part */
163         *cp++ = j+'0';
164         val=(val-j)*TEN;
165     }
166     *cp=0;
167     return cp-string;
}
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 }
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 xpinit()
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()
{
    long ltmp;

    switch (Rcnt) {
        case 0:
            ltmp = xpswap(*(long *)Pp);
            Pp += 2;
            Rcnt += 2;
            break;

        case 1:
            Pp++;
            ltmp = xpswap(*(long *)Pp);
            Pp += 2;
            Rcnt += 3;
            break;

        case 3:
            Pp++;
            ltmp = xpswap(*(long *)Pp);
            Pp = Svec;
            Rcnt += 3;
            break;

        case 4:
            ltmp = xpswap(*(long *)Pp);
            Pp = Svec;
            Rcnt += 2;
            break;

        case 5:
            Pp = Svec;
            ltmp = *(long *)Pp;
            Pp += 2;
            Rcnt += 3;
            break;

        default:
            if (Rcnt == 6)
                Pp = Svec;
            ltmp = *(long *)Pp;
            Pp += 2;
            Rcnt += 2;
            break;
    }
    return ltmp;
}

char *xp.ptr()
{
    char *cptmp;

    if (Rcnt == 6)
        Pp = Svec;
    cptmp = (char *)*Pp;
    Rcnt++;
    Pp++;
    return cptmp;
}
double xpdouble() {
    return (double)0;
}

#ifdef zilog
    fmtout(func, funarg, string, ip)
#else
    fmtout(func, funarg, string, ip, regp, regcnt)
#endif
int *regp;

int (*func)();
char *funarg;
char *string;
int *ip;
{
    char tbuff[128], *cp, cb;
    int base;
    int is_number;
    unsigned leftadj, padchar, width, precflg, precis, longflg, length;
    union {
        long tlong;
        long tulong;
    } lw;
#ifdef FLOATS
    double *dp;
#endif
#endif
zilog(ip);
#else
zilog(regp, regcnt, ip);
#endif

while(*string){
    if( *string != '%' ){
        for(cp=string; *cp & & *cp != '%' ;)
            (*func)( (*cp++) & 0xff, funarg);
        string=cp;
    } else {
        is_number = 1;
        if( (leftadj>(*string == '-' ))
            ++string;
        padchar = *string & 0xff;
        if(padchar=='0')
            ++string;
        else padchar='1';
        if (*string == '*') {
            /* width is an argument */
            width = xpint();
++string;
}
else
    for (width=0;isdigit(*string);
         width=width*10+(*string++-'0');
    if (preclgl=(*string=='.')) {
        ++string;
        if ( *string == '*'){
            /* precision is an argument */
            precsn = xpint();
            ++string;
        } else
            for (precsn=0;isdigit(*string);
                 precsn=precsn*10+(*string++-'0')
            }
    } else precsn=0;
    if (longflg=(*string=='l'))
        ++string;
    switch (*string) {
    case 'e':
        case 'f':
        case 'g':
            if (!preclgl) precsn=6;
            dp = (double *)(doublep());
            length = dtos(*dp++, cp=tbuff, precsn, *string &
                          break;
        #endif
        case 'B':
        case 'b':
            base=2;
            goto nosign;
        case 'O':
        case 'o':
            base=8;
            goto nosign;
        case 'U':
        case 'u':
            base=10;
            goto nosign;
        case 'X':
        case 'x':
            base=16;
            goto nosign;
        case 'D':
        case 'd':
            base = -10;
    nosign:
        if (!longflg)
            longflg=(*string>='A'&&*string<='Z');
    if (longflg){
        lw.tlong = xplong();
    }
    else if (base<0)lw.tlong=(long)xpint();
else lw.tulong=(unsigned)(xprint());
lto(lw.tlong, tbuf, base);
if(precfg){
    cp = tbuf;
    if (lw.tlong <0) ++cp;
    length= xstrlen(cp);
    if (precisn && length < precisn+1 ) {
        movmemm(cp, cp+precisn+1-length, 
        setmem(cp, precisn+1-length, '0'
        length = precisn +1;
    }
    movmemm( cp+length-precisn, cp+length-pre
    cp[length-precisn] = '.';
}
length=xstrlen(cp=tbuf);
break;
case 's':
cp = xpptr();
length = xstrlen(cp);
if(precfg && precisn<length)length=precisn;
/* leave minus signs alone */
is_number=0;
brea;
}
case 'c':
    cb = xprint() & 0x7F;
    cp = &cb;
    length=1;
    is_number=0;
    break;
default:
    cp=string;
    length=1;
    is_number=0;
    break;
}
if (!leftadj & width>length) {
    if (is_number & *cp == '-' & padchar == '0') {
        (*func)((*cp++) & 0xff, funarg);
        --length;
        --width;
    }
    while (width-- >length)
    (*func)(padchar, funarg);
}
if (width>length)
    width-=length;
else
    width=0;
while (length--)
    (*func)((*cp++) & 0xff, funarg);
if (leftadj & width)
    while(width--)
    (*func)(padchar, funarg);
++string;
/*
 * @(#)ltos.c 1.0 6/4/85
 * GENERIC LIBRARY
 * *
 * filename: LTOS.C
 */

#include "xgenlib.h"

/*
 * movmem  ->  move memory
 */

int movmem ( from, to, len)
char *from, *to;
int len;    /* no of bytes to copy */
{
    while ( len-- )
        *to++ = *from++;
}

/*
 * setmem:  ->  set memory to a desired value
 */

int setmem( s, len, c)
char *s;    /* start address of the memory */
int len;    /* no of char to be set */
char c;    /* character to be set */
{
    while ( len-- )
        *s++ = c;
}

/*
 * long to string
 */

ltos(val,cp,base)
long val;    /* the number to convert */
char *cp;    /* the address of the string */
int base;    /* the conversion base */
{
    char tempc[34],*tcp;
    int n=0;    /* number of characters in result */
    long uval;    /* unsigned value (not possible on all compilers) */
    static char dig[]="0123456789ABCDEF";

    *(tcp=tempc+33)=0;

    if(base<0){    /* needs signed conversion */
        if(val<0)n=1;
        else val= -val;
        do {
            register int num = -(val%base);
            *--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 xprintf(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  * XSTRLEN: return the size of the string
11  */
12
13 int xstrlen( s )
14  unsigned char *s;
15  {
16      unsigned char *p = s;
17      while ( (*p != '\0') ) p++;
18      return ( p-s );
19  }
20
21 /*
22  * XSTRCPY: copy string 2 to string 1
23  */
24
25 xstrcp( s, t)
26  unsigned char *s, *t;
27  {
28      while ( (*s++ = *t++ );
29  }
30
31 /*
32  * XSTRNCPY: copy a string into a buffer n characters in length
33  */
34
35 unsigned char *xstrncpy( s, t, n)
36  unsigned char *s, *t;
37  int n;
38  {
39      unsigned char *cp;
40      
41      for ( cp = s; n && ( *cp++ = *t++ ); --n );
42      while ( (n-- )
43          *cp++ = '\0';
44      return ( s );
45  }
46
47
48 /*
49  * XSTRCMP: compare strings and return -ve, 0 or +ve accordingly
50  */
51
52 int xstrcm( s, t ) /* return <0 if s<t, 0 if s==t, >0 if s>t */
53  unsigned char *s, *t;
54  {
55
```c
for ( ; *s == *t; s++, t++ )
    if ( *s == '\0' )
        return ( 0 );
return ( *s - *t );

/*
* XSTRICMP: case insensitive string comparision
*/
#define conv(x) ( isupper(x) ? _tolower(x) : x )

int xstricmp( s, t)
    unsigned char *s, *t;
    {
    for ( ; conv(*s) == conv(*t); s++, t++ )
        if ( *s == '\0' )
            return ( 0 );
    return ( conv(*s) - conv(*t) );
    }

/*
* XSTRNCMP: string compare up to n characters
*/

int xstrncmp( s, t, n)
    unsigned char *s, *t;
    int n;
    {
    for ( ; n-- && ( *s == *t ); t++ )
        if ( !*s++ )
            return ( 0 );
    if ( n < 0 )
        return ( 0 );
    if ( *s < *t )
        return ( -1 );
    return ( 1 );
    }

/*
* XSTRCAT: concatenates string 2 to the end of string 1
*/

int xstrcat( s, t )
    unsigned char *s, *t;
    {
    while ( *s++ != '\0' );
    for ( --s; (*s++ = *t++) != '\0'; );
    }

/*
* XSTRNCAT: concatenate string 2 to string 1 , max n chars
*/
```
unsigned char *xstrncat( s, t, n)
  unsigned char *s, *t;
  int n;
  {
    unsigned char *cp;

    for ( cp = s; *cp++; )
      for ( --cp; n-- && ( *cp++ = *t++ ); )
        if ( n < 0 )
          *cp = '\0';
    return s;
  }

/*
* XSTRCHR: return a pointer to first occurrence of character
*/

unsigned char *xstrchr( s, c)
  unsigned char *s, c;
  {
    while ( *s )
      if ( *s++ == c )
        return --s;
    return ( 0 );
  }

/*
* XSTRRCHR: return a pointer to the last occurrence of char
*/

unsigned char *xstrrchr( s, c)
  unsigned char *s, c;
  {
    unsigned char *cp;

    for ( cp = s + strlen(s); --cp >= s; )
      if ( *cp == c )
        return cp;
    return 0;
  }
Temporary entry points for some x routines.

```
#include <stdio.h>
extern int errno;

char *xatoi(a)
char *a;
{
return( atoi(a) );
}

char *xsprintf()
{
/*
keep the linker happy at least.
*/
}
/*
 * %WZ %GZ
 *
 * Generic close.
 */

#include <stdio.h>
#include <xerrno.h>
#include <stdio.h>

extern int xnofunc();

xclose( fd )
{
    int fd;
    
    int rval;
    register XFILE *current;

    if( fd >= 0 && fd < _XNFILE )
    {
        current = &xiob[fd];
        if( !( current->flag & _XUsed ) )
        {
            fprintf( stderr, "xclose: bad od\n" );
            return ( XEBADF );
        }

        /*
         * file descriptor is OK, check for copies.
         */
        if( (current->flag & _XIOWRT ) && (current->bufsiz > current->cnt) )
        {
            /*
             * Flush write buffer (if appropriate).
             */
            fprintf( stderr, "xclose: flushing\n" );
            xfflush( current );
        }

        if( current->base &&
            (( current->flag & _XIOMYBUF == 0 )))
        {
            /*
             * free buffer allocated by system.
             */
            fprintf( stderr, "xclose: freeing\n" );
            xfree( current->base );
        }

        if( current->pred )
        {
            /*
             * at least one copy exists
             */
            struct_xiobuf *next = current->succ;
            struct_xiobuf *previous = current->pred;
            int primary = current->flag & _XPrimary;
        }
    }
fprintf( stderr, "xclose: uncoping\n" );
if( next == previous )
{
  /*
   only one other copy
   */
  previous->_pred = (struct _xibuf *)0;
  previous->_succ = (struct _xibuf *)0;
}
else
{
  /*
   remove from linked list
   */
  previous->_succ = next;
  next->_pred = previous;
}

if( primary )
{
  /*
   Make new primary copy.
   */
  previous->_flag |= _XPrimary;
  rval = 0;
}
else
{
  /*
   Only copy, perform real close
   */
  fprintf( stderr, "xclose: closing\n" );
  rval = (*(current->_close))( current->_sys_id );
}

/*
Cleanup _xib structure.
*/
current->_flag = 0;
current->_cnt = 0;
current->_ptr = (char *)0;
current->_base = (char *)0;
current->_bufsiz = 0;
current->_succ = (struct _xibuf *)0;
current->_pred = (struct _xibuf *)0;
current->_read = xnofunc;
current->_write = xnofunc;
current->_ioctl = xnofunc;
current->_close = xnofunc;
return( rval );
}
/*
 * Copy an EXOS file object.
 */

#include <stdio.h>
#include <xerrno.h>

xdup2( orig_fd, new_fd )

int orig_fd;
int new_fd;
{

if( orig_fd > 0 && orig_fd < _XNFILE && new_fd > 0 && new_fd < _XNFILE )
{
    register struct _xiobuf *new = &_xio[new_fd];
    register struct _xiobuf *orig = &_xio[orig_fd];

    if( !(orig->flag & XUsed) )
        return( XEBADF );
    if( orig_fd == new_fd )
        return( 0 );

    /* separate buffering for new object,
    * everything else identical.
    */
    new->flag = orig->flag &
        ~(XPrimary | _XIOMYBUF | _XIOLBF | _XIONBF);
    new->cnt = 0;
    new->ptr = (char *)0;
    new->base = (char *)0;
    new->file = new_fd;
    new->sys_id = orig->sys_id;
    new->close = orig->close;
    new->read = orig->read;
    new->ioctl = orig->ioctl;
    new->write = orig->write;

    /* insert into linked list of copies */
    if ( !orig->succ )
    {
        new->succ = orig;
        new->pred = orig;
        orig->succ = new;
        orig->pred = new;
    }
    else
    {
        new->succ = orig->succ;
        new->pred = orig;
        orig->succ = new;
    }
}
57     return( 0 );
58 }
59     return( XEBADP );
60 }
/*
* %W% %C%

Unix specific close all EXOS file objects and exit program.
*/

#include <stdio.h>

xexit( status )

int status;
{
int i;

for( i = 0; i < _XNFILE ; ++i )
{
xclose( i );
}
exit( status );
}
/*
 * Operating system independent routine for flushing buffers
 * Associated with pointers to io objects.
 */

#include <stdio.h>
#include <errno.h>

xfflush( file )

register XFILER *file;
{
    int rval;
    int nmtowrite;
    char *pt;

    if( !(file->flag & XUUsed ) )
        return( XEBADF );
    if( !file->base || !(file->_flag & XIOWRT ) )
        return( XEBADF );
    if( file->_flag & XIOLBF )
    {
        nmtowrite = (int)( file->_ptr - file->_base );
        file->_cnt = 0;
    }
    else
    {
        nmtowrite = file->_bufsz - file->_cnt;
        file->_cnt = file->_bufsz;
    }
    file->_ptr = file->_base;
    rval = 0;
    pt = file->_base;
    while( nmtowrite > 0 )
    {
        rval = xwrite( xfileno( file ), pt, nmtowrite );
        if( rval <= 0 )
            break;
        nmtowrite -= rval;
        pt += rval;
    }
    return( rval );
}
/*
 @((#) _xfilbuf.c  1.4 5/22/85
 System independent routine for filling buffers associated with
 a pointer to an io object. (used to implement xgetc).
 */
#include "xgenlib.h"

_xfilbuf( file )

register XFILE *file;
{
    int rval;
    char ch;

    if( !( file->_flag & XUsed ) )
        return( XEOF );
    if( !( file->_flag & XIOREAD ) )
        return( XEOF );
    if( file->_flag & _XIOERR )
        {
            /*
            * Allow user to retry after an error.
            */
            file->_flag &= ~_XIOERR;
        }
    if( file->_base )
        {
        rval = xread( xfileno(file), file->_base, file->_buFSIZE );
        }
    else
        {
        rval = xread( xfileno(file), &ch, 1 );
        }
    if( rval < 0 )
        {
        file->_flag |= _XIOERR;
        return( XEOF );
        }
    else if( rval == 0 )
        {
        file->_flag |= _XIOEOF;
        return( XEOF );
        }
    file->_cnt = rval - 1;
    if( file->_base )
        {
        file->_ptr = &file->_base[1];
        return( (file->_base[0]) & 0xff );
        }
    else
        {
        return( ch & 0xff );
        }
}
/*
@(#)_xflsbuf.c  1.6 5/22/85

System independent routine for filling io buffers.
( used to implement xputc ).
*/

#include "xgenlib.h"

_xflsbf( x, file )

unsigned int x;
register XFILE *file;
{
  int rval;
  int nmtowrite;
  int storex;
  char xch;
  char *pt;

  if( !(file->_flag & _XUUsed) )
    return( XEBADF );
  if( !(file->_flag & _XIOWRIT) )
    return( XEBADF );
  if( file->_flag & _XIOERR )
    { _
      /*
      * Allow user to retry after an error.
      */
      file->_flag &= ~_XIOERR;
    }
  if( file->_base )
    {
    storex = 1;  /* put x in buffer after flush */
    pt = file->_base;
    /*
    Check for line buffering
    */
    if( file->_flag & _XIOLBF )
      { _
        nmtowrite = (int)(file->_ptr - file->_base);
        if( nmtowrite >= file->_bufsz )
          {
            /*
            flush buffer, because it is full.
            */
            }
        else if( x == '\n' || x == '\r' )
          {
            /*
            flush buffer, because of end of line
            */
            storex = 0;
            *(file)->_ptr++ = x;
            ++nmtowrite;
          }
      }
    else
      { _
    /*
    */
    storex = 0;
    *(file)->_ptr++ = x;
    ++nmtowrite;
    }
    { *(file)->_ptr++ = x; 
     file->cnt = 0; 
     return( 0 ); 
   }
   }
  
  else 
  { 
   nmtowrite = file->_bufsz - ( file->_cnt + 1 );
  }
  
  while ( nmtowrite > 0 )
  {
    rval = xwrite( xfileno(file), pt, nmtowrite );
    if( rval < 0 )
      {
        file->flag |= XIOERR;
        return( rval );
      }
    if( rval == 0 )
      {
        file->flag |= XEIO;
        return( XEIO );
      }
    nmtowrite -= rval;                /* Assert: rval <= nmtowrite */
    pt += rval;
  }

  if( file->flag & _XIOLBF )
  { 
    if( storex && ( x == '\n' || x == '\r' ) )
      {
        /*
         write out carriage return
         */
        xch = x;
        rval = xwrite( xfileno(file), &xch, 1 );
        storex = 0;
        if( rval < 0 )
          return( rval );
      }
    file->_cnt = 0;
  }

  else 
  { 
    file->_cnt = file->_bufsz - 1; /* _cnt == #chars remaining, -1 for "x" */
    if ( file->_cnt < 0 )
      { 
        /*
         This should not happen.
         */
        return( XEFAULT );
      }
    file->_ptr = file->_base;
    if( storex )
      *file->_ptr++ = x;
  }
else
    };
    xch = x;
  rval = xwrite( xfileno(file), &xch, 1 );
  if( rval < 0 )
    {
      file->flag |= _XIOERR;
      return( rval );
    }
  if( rval != 1 )
    {
      file->flag |= _XIOERR;
      return( XEIO );
    }
  file->cnt = 0;
  return( 0 );
}
1  /*
2   * @(#)xfprintf.c    1.6 5/31/85
3   *
4   Xprintf(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  xprintf(op,control,args)
18   XFILE *op;
19   char *control;      /* the format control string */
20   unsigned args;
21   {
22     return _fmtout(_xputc,op,control,&args);
23   }
24
25  xprintf( control, args )
26   char *control;
27   unsigned args;
28   {
29     return _fmtout(_xputc, stdout, control, &args );
30   }
31
32  #else
33
34  xprintf(op, control, a1, a2, a3, a4, args)
35   XFILE *op;
36   char *control;      /* the format control string */
37   int a1, a2, a3, a4;
38   unsigned args;
39   {
40     int s1=a1, s2=a2, s3=a3, s4=a4;
41     return _fmtout(_xputc, op, control, &args, &s1, 2);  
42   }
43
44  xprintf(control, a1, a2, a3, a4, a5, args)
45   char *control;
46   int a1, a2, a3, a4, a5;
47   unsigned args;
48   {
49     int s1=a1, s2=a2, s3=a3, s4=a4, s5=a5;
50     return _fmtout(_xputc, stdout, control, &args, &s1, 1);  
51   }
57 #endif
/*
 %W% %G%

Print password prompt, turn off echoing, and get password, restore terminal,
Using the facilities of xoslib.

Caveat: This assumes xstdin and xstdout == user's terminal.
*/

#include <stdio.h>

#define MXPWORD 25

char *
xgetpass( prompt )

char *prompt;
{
  static char buf[ MXPWORD ];
  register char *pt = &buf[0];
  register int rval;
  register int i = 0;

  /*
  Should use xprintf, but not available now...
  */
  while( *prompt != '\0' )
    {
      putchar( *prompt++ );
    }
  xfflush( xstdout );
  xraw_term( xnofunc );
  do {
    rval = xread( 0, pt, 1 );
    if( rval < 0 )
      xerror( rval, "xgetpass" );
    } while ( *pt != '\r' && *pt != '\n' &&
    rval == 1 && ++i < MXPWORD && ++pt );
  *pt = '\0';
  xrestore_term();
  return( &buf[0] );
}
/*
@(#)xgets.c 1.4 5/22/85
Xgets(3X).
*/
#include "xgenlib.h"
char *
xgets( string )
char *string;
{
int c;
char *p = string;
while ( (c = xgetchar()) != EOF && c != '\n' )
{
    *p++ = c;
}
*p = '\0';
if( c == EOF )
    return( XNULL );
return( string );
}
/*
 * Generic io control.
 */

#include <stdio.h>
#include <xerrno.h>

xioctl(fd, cmd, param)

int fd;
int cmd;
char *param;
{
    int rval;
    register XFILE *file;

    if ( fd > 0 && fd < _XNFILE )
    {
        file = &_xiof[fd];
        if( !( file->flag & _XUsed ) )
            return( XEBADF );

        /*
        file descriptor is OK.
        */
        rval = (file->ioctl)( file->sys_id, cmd, param );
        return( rval );
    }

    return( XEBADF );
}
/**
Xmkarglist from xglob(3X) for Unix.
This file belongs in Xoslib, but is here to keep the linker happy.
*/
define ARGPOINTERSP 200 /* bytes for storing argument pointers */
define ARGSIZE 400 /* bytes for storing arguments */

static char *argbase;
static char *stringbase;

extern char *xmalloc();

#include

char **
xmkarglist( line, count )
char *line; /* IN */
int *count; /* OUT */
{
char **argp;
char *slurpstring();
char *argvsp;
int margc;

margc = 0;
/*
Allocate space for argv and tokens in line */
if( xstrlen( line ) > ARGSIZE )
{
    return( (char **)0 );
}
argvsp = xmalloc( ARGPOINTERSP + ARGSIZE );
if( argvsp <= (char *)-1) < (char *)0 )
{
    return( (char **)0 );
}
argvsp = argvsp + ARGPOINTERSP; /* store from first of buffer */
stringbase = line;
argp = (char **)argvsp;
while (*argp++ = slurpstring())
margc++; /* count = margc;
return( (char **)argvsp );
}
/*
* Parse string into argbuf;
* implemented with FSM to
* handle quoting and strings
*/
char *
slurpstring()
{
    int got_one = 0;
register char *sb = stringbase;
register char *ap = argbase;
char *tmp = argbase;  /* will return this if token found */

/*
Used to return '!' for shell event processing...
Ignore significance of '!' .
*/

S0:
switch (*sb) {
    case '\0':
        goto OUT;
    case ' ': 
    case '\t':
        sb++; goto S0;
    default:
        goto S1;
}

S1:
switch (*sb) {
    case ' ':
    case '\t':
    case '\0':
        goto OUT;  /* end of token */
    case '\\':
        sb++; goto S2; /* slurp next character */
    case '"':
        sb++; goto S3; /* slurp quoted string */
    default:
        *ap++ = *sb++;
        got_one = 1;
        goto S1;
}

S2:
switch (*sb) {
    case '\0':
        goto OUT;
    default:
        *ap++ = *sb++;
        got_one = 1;
        goto S1;
}

S3:
switch (*sb) {
case '0':
goto OUT;

case '':
sb++; goto S1;

default:
    *ap++ = *sb++;
got_one = 1;
goto S3;
}

OUT:  
if (got_one)
    *ap++ = '\0';  /* update storage pointer */
argbase = ap;  /* update scan pointer */
stringbase = sb;
if (got_one)
    return(tmp);
    return((char *)0);

xdealglob( pt )
/*
Free space allocated by either xglob or xmkarglist
*/

char **pt;
{
xfree( (char *)pt );
}
/*
 * Unix routine to form path names relative to the user's home directory,
 * current directory, etc.
 * This routine belongs in Xoslib, but is here to keep the linker happy.
 */
#include <xspecial.h>
#include <xerrno.h>

/*
for now ...
*/
#define xsprintf sprintf

#include <pwd.h>

xmodname( name, special, buf, sz_buf )

char **name;
int special;
char *buf;
int sz_buf;
{
  int rval;
  char *pt = buf;

  switch( special ) {
    case FILE_NAME:
      pt = *name;
      break;
    case CURRENT_DIR:
      buf[0] = '.';
      buf[1] = '0';
      break;
    case UP_DIRECTORY:
      buf[0] = '.';
      buf[1] = '0';
      buf[3] = '0';

    case HM_RELATIVE:
    case HOME_DIR:
      {
        struct passwd *pwent;
        extern struct passwd *getpwuid();
        int uid;

        uid = getuid();
        pwent = getpwuid( uid );
        if( pwent == (struct passwd *)0 )
          return( XEPERM );
        if ( special == HM_RELATIVE )
          {
            /*
             Check for enough room in buffer,
             concatenate home directory and *name.
             */
            if( (2 + xstrlen( pwent->pw_dir ) + xstrlen( *name ))
> 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 }
Given an index to an open io object, 
Associate a buffer with it and return a pointer.
*/

#include <stdio.h>
#include <stdio.h>

extern char *xmalloc();

XFILE *xodopen( od, direction )
int od;    /* object descriptor */
char *direction;      /* 'r' for read, 'w' for write */
{
    XFILE *file;
char *rval;

    if( od < 0 || od >= _XNFILE )
    {
        fprintf( stderr, "bad od\n" );
        return( (XFILE *)XNULL );
    }
    file = &_xio[od];
    /*
    Make sure object has been opened, and in the right direction.
    */
    if( !(file->flag & _XUsed) )
    {
        fprintf( stderr, "not used\n" );
        return( (XFILE *)XNULL );
    }
    switch ( direction[0] )
    {
    case 'r':
        if( file->flag & _XIORD )
        {
            file->flag &= ~_XIOWRT;
            break;
        }
        if( file->flag & _XIOWR )
        {
            file->flag &= ~_XIORD;
            file->flag |= _XIOREAD;
            break;
        }
        break;
    case 'w':
        if( file->flag & _XIOWRT )
        {
            file->flag &= ~_XIOREAD;
            break;
        }
        if( file->flag & _XIOWR )
        {

```
71  if ( rval == (char *)-1 )
72     {
73      fprintf( stderr, "no memory\n" );
74      return( (XFILE *)XNULL );
75    }
76
77 xsetbuf( file, rval, XBUFSIZ );
78 file->flag &= ~_XIOYBUF;            /* for automatic deallocation */
79 return( file );
Xogets(3X).
*/
#include <stdio.h>

char *
xogets( string, n, stream )
{
    char *string;
    int n;
    FILE *stream;
    {
        int c;
        char *p = string;
        while ( (c = xgetc(stream)) != XEOF && c != '\n' && --n > 0 )
        {
            *p++ = c;
        }
        *p = '\0';
        return( string );
    }
}
/*
 @(#)xerror.c   1.4 6/4/85

 Xerror(3X) and xerror.
 */
#include "xgenlib.h"

#define MINEXERR 0

cchar *x_errlist[] = {
   "No Error",
   "Not super-user",
   "No such file or directory",
   "No such process",
   "interrupted system call",
   "I/O error",
   "No such device or address",
   "Arg list too long",
   "Exec format error",
   "Bad file number",
   "No children",
   "No more processes",
   "Not enough core",
   "Permission denied",
   "Bad address",
   "Block device required",
   "Mount device busy",
   "File exists",
   "Cross-device link",
   "No such device",
   "Not a directory",
   "Is a directory",
   "Invalid argument",
   "File table overflow",
   "Too many open files",
   "Not a typewriter",
   "Text file busy",
   "File too large",
   "No space left on device",
   "Illegal seek",
   "Read only file system",
   "Too many links",
   "Broken pipe",
   "Argument too large",
   "Result too large",
   "Operation would block",
   "Operation now in progress",
   "Operation already in progress",
   "Socket operation on non-socket",
   "Destination address required",
   "Message too long",
   "Protocol wrong type for socket",
   "Protocol not available",
   "Protocol not supported",
   "Socket type not supported",
   "Operation not supported on socket",
}
"Protocol family not supported",
"Address family not supported by protocol family",
"Address already in use",
"Can't assign requested address",
"Network is down",
"Network is unreachable",
"Network dropped connection on reset",
"Software caused connection abort",
"Connection reset by peer",
"No buffer space available",
"Socket is already connected",
"Socket is not connected",
"Can't send after socket shutdown",
"Too many references: can't splice",
"Connection timed out",
"Connection refused",
"Too many levels of symbolic links",
"File name too long",
"Host is down",
"No route to host",
};

static char bad_err[] = "UNKNOWN ERROR";
char *xerror();
extern char *xsyserr();

#define MAXEXERR 65

xerror( eval, rname )

int eval;
char *rname;
{
int len;
int olderrno;
char *estring;
olderrno = -eval;
len = xstrlen( rname );
if ( len > 0 )
{
    if ( xwrite( 2, rname, len ) != len )
        return;
    if ( xwrite( 2, ":", 1 ) != 1 )
        return;
}
estring = xerror( eval );
len = xstrlen( estring );
xwrite( 2, estring, len );
xwrite( 2, "\n", 1 );
}

char *
xerror( eval )
int eval;
{
    int olderrno;
    olderrno = -eval;
    if ( eval <= XSYSERR )
        return( xsyserr() );
    }
    else if( olderrno < MINEXERR || olderrno > MAXEXERR )
        { /*
          bad error number
          */
            return( bad_err );
        }
    return( x_errlist[ olderrno - MINEXERR ] );
}
/*
 *
 * Generic read.
 */

#include <stdio.h>
#include <xerrno.h>

xread( fd, buf, len )
int fd;
char *buf;
int len;
{
    register FILE *file;

    if( fd >= 0 && fd < _XNFILE )
    {
        file = &xiob[fd];
        if( !( file->flag & ( _XIOREAD | _XIORW ) ) ||
            !( file->flag & _XUsed ) )
            return( XEBADF );
        /*
         * file descriptor is OK.
         */
        count = (*(file->read))( file->_sys_id, buf, len );
        return( count );
    }
    return( XEBADF );
/*
 * @(#)xsetbuf.c    1.4 5/29/85
 * System independent routine for setting buffer associated with
 * pointers to file objects
 */
#include "xgenlib.h"

xsetbuf( file, flag, len )

FILE *file;
char *flag;
int len;
{

if( !(file->flag & _XUSeD) )
    return( XEBADE );

/*
flush old buffer, if appropriate
*/
xfflush( file );

/*
release old buffer, if appropriate
*/
if( file->base && !(file->flag & _XIOYBUF) )
{
    xfree( file->base );
}
if( !flag )
{
    /*
User specified no buffering
*/
    file->flag &= ~( _XIOYBUF );
    file->flag |= _XIONBF;
    file->base = (char *)0;
    file->cnt = 0;
}
else
{
    /*
User supplied buffer.
*/
    file->bufsz = len;
    file->flag |= _XIOYBUF;
    file->base = flag;
    /*
Assert: !((file->flag & _XIOREAD) && (file->flag & _XIOWRT))
*/
    file->cnt = (file->flag & (~XIOLOBF | _XIOREAD)) ? 0 : len ;
    file->ptr = file->base;
}
return( 0 );
}
/* @(#)xsprintf.c  1.3 5/31/85
   * Xsprintf(3X).
   * format data to a memory string
   */
#include "xgenlib.h"

/*
 * this function behaves much like putc with only difference
 * is that it writes to an user buffer instead of a file.
 */
static _store(x, to)
char x;
char **to;
{
    **to = x;
    (*to)++;
    /* point to the next location of the buffer */
    **to=0;
    /* terminate for safety */
}

#ifdef zilog
char *
xsprintf(string,control,args)
char *string;
char *control;
unsigned args;
{
    return (char *)_fmtout(_store,&string,control,&args);
}
#else
char *
xsprintf(string, control, a1, a2, a3, a4, args)
char *string;
char *control;
int a1, a2, a3, a4;
unsigned args;
{
    int s1=a1, s2=a2, s3=a3, s4=a4;
    return (char *)_fmtout(_store, &string, control, &args, &s1, 2);
}
#endif
Generic write.

*/

#include <stdio.h>
#include <errno.h>

xwrite( fd, buf, len )
{
    int fd;
    char *buf;
    int len;
    {
        int count;
        register XFILE *file;
        if( fd > 0 && fd < _XNFILE )
        {
            file = &_xiof[fd];
            if( !(file->flag & ( XIOWRITE | XIORW )) ||
                !(file->_flag & _XUsed ) )
                return ( XEBADF );
            /*
            file descriptor is OK.
            */
            count = (*(file->write))( file->_sys_id, buf, len );
            return( count );
        }
        return( XEBADF );
    }