SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

D I S T R I B U T E D C O M P U T I N G N O T E 5 8 5

CAMBRIDGE RING
Protocol Implementation for Unix V7
on DEC and PERQ - Version 1

Issued by
W P Sharpe

1 March 1982

DISTRIBUTION:
R W Witty
D A Duce
C P Wadsworth
W P Sharpe
A S Williams
J M Loveluck
L O Ford
C Prosser
RL Support/C Ring/Working Papers file
RL Support/Unix/General file

1. OVERVIEW

This note describes the proposed implementation of the Cambridge Ring protocols for Unix V7.

1.1 Objectives

The main objectives are:

(1) To provide transport service over the ring.

(2) To give access to the Basic Block protocol directly; this is required for implementing simple connectionless protocols for servers such as name, date & time etc.

(3) To have an implementation in which as much code as possible can be common on Perq Unix and large and small kernel PDP11 Unix.

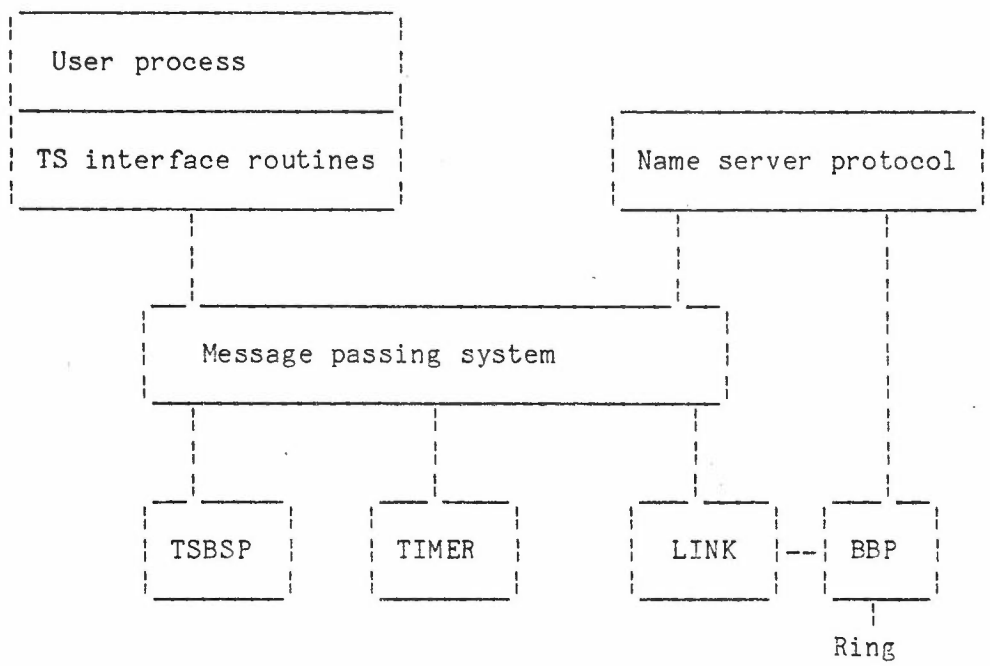
(4) To aim at future integration of the Ring transport service with the X25 transport service so that all routing is performed transparently.

(5) In addition to the transport service interface to the ring there is a good case for providing a 'data stream' interface ie when a process performs the usual Unix open, close, read and write calls on a special file it is connected transparently to a remote process across the ring, there being no interface to the more elaborate facilities of the transport service. This facility is useful for such things as line printer spoolers. It is not expected to be

implemented in the first release.

1.2 Modules and Interfaces

The following diagram illustrates the software modules of the implementation:



The functions of these modules are summarised here and given in more detail below.

1.2.1 Message Passing System

With the intention of keeping the design as modular as possible and easing portability the various functions and layers of the protocols have been assigned to separate modules which communicate by messages and do not rely on specific features of the PDP11 Unix kernel. The specific assumptions made of the message system are:

(1) Messages up to some reasonable size (256 octets) may be submitted 'instantly' up to some maximum capacity of the message system ie the message is copied by the system and the user can then continue executing and use his message buffer before the recipient may have received the message.

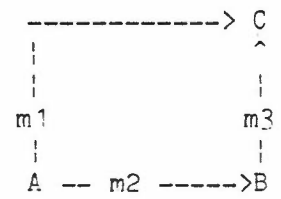
(2) More than one message at a time can be posted from one process to another, and they will arrive in the order they are sent.

(3) The system is not expected to provide to a process demultiplexing of incoming messages according to source address.

(4) A process is able to wait for a message to come from any one of a number of sources.

(5) Recipients are informed of the length of messages.

(6)



If A sends m1 then m2, and B sends m3 only on receiving m2, then it is assumed that m1 will arrive at C before m3.

Such a system can be implemented quite simply within the PDP11 Unix kernel (see below), comments are requested on whether the same is true on ACCENT.

1.2.2 UUP / TS interface routines

The interface provided to a Unix User Process (UUP) to the transport service will be that defined by Keith Ruttle for the York X25 Transport Service [1]. This provides access to all the TS facilities, with the following characteristics:

(1) Each UUP is restricted to only one connection active at a time.

(2) There is a fixed maximum size (256 octets) for transport service control messages received during the data phase of a connection. This is only of concern if you wish to receive TS addresses longer than this - a most unlikely occurence since most X25 implementations will not accept anything longer than 128 octets. This restriction is a feature of the interface, not the implementation, neither the X25 nor the Ring implementations will be restricted.

(3) Expedite messages are restricted to one octet. TSBSP can carry longer expedited messages but it is not proposed to alter the interface.

(4) A feature of the X25 implementation, rather than the interface, is that a user who excutes a 'listen' has not the ability to reject a call from a calling address he does not like - the call is accepted automatically. The same will be true of the ring implementation, which is justified because of timing constraints in replying to the incoming OPEN. This is not serious since the user can immediately close the connection if he wishes.

These routines perform a certain amount of blocking and unblocking of user data and TS parameters and handle all the communication with the other modules. It will also be the responsibility of these routines to handle table lookup or name server interactions for title/address translation.

1.2.3 TSBSP

This module will be a single process maintaining all the active connections. The maximum number of simultaneous connections will be a compile time parameter. It will be event driven, with all events being signalled by messages from the other modules. It will be written in C. Its interface to the message system will be by sets of routines that provide a consistent interface on PDP11 and Perq Unix. With the exception of these routines the remainder of the module will be completely portable between the two systems. In the first implementation this module will reside in user space on PDP11 Unix, but the design is intended to allow it to operate within the Unix kernel also, thus enhancing performance considerably.

Specific implementation decisions are as follows:

(i) The "high throughput" technique of TSBSP will not be implemented. This is an experimental facility required by Universe to overcome the restrictions encountered using the window-of-one over a satellite link.

(ii) Packet sizes of less than 64 octets will not be supported.

1.2.4 Link

This module provides the link level interface for TSBSP. The TSBSP module has no requirement to process the user data and so this module 'knits' together the data from the user with TSBSP protocol commands and control messages from TSBSP into complete packets (Basic Blocks), and correspondingly splits up incoming packets. This module is responsible for posting the ring i/o requests with the Basic Block Protocol driver. This requires a fairly intimate connection and on PDP11 Unix necessitates this module being within the kernel. Thus its portability will be restricted and it has not been constrained to communicate with BBP solely by the message system.

For receiving packets the requirements LINK has of BBP are as follows:

(i) A list of requests may be set up, requiring only one entry per basic block port. Each request is of the form (remote station, local basic block port, buffer address, maximum packet size). BBP fills the buffer with up to the maximum data octets and signals LINK. If "remote station" is 255 then BBP will accept a packet from any remote station and will inform LINK of the source station number.

(ii) A request to receive may be cancelled.

(iii) LINK can request allocation of a currently exclusive local basic block port number, and can relinquish usage of the number some time later.

For transmitting packets the requirements LINK has of BBP are as follows:

(i) Requests may be added to a queue which is serviced in strict FIFO order. Each request is of the form (remote station, remote basic block port, buffer address, packet length). BBP informs LINK of the completion of a transmission attempt, but a specific result (busy, absent, etc) is not required.

1.2.5 BBP

This is the low level Basic Block Protocol driver; it will have two interfaces:

(1) An intimate interface to the LINK module.

(2) A public interface to user processes wishing to make direct BBP transactions. This interface will be a simplified version of the interface to the current driver (App A). A number of processes may have direct BBP transactions at a time, with data passing through kernel buffers (on PDP11); a raw device accessible to only one process at a time will cut out the kernel copying to provide higher throughput. This enhancement will be an option, and probably not needed in the first release.

On machines having a UMCZ80 implementing the BBP this module will be small; LSI11 Unix systems will have their problems aggravated by having to implement this protocol in the kernel as well.

1.2.6 Timer

TSBSP has an interface to the timer providing it with two timers per active connection. The functional split between the TSBSP timer routines and the TIMER module will probably be system dependent.

1.2.7 Name Server Protocol

In order to provide future integration of LAN and WAN communication it appears necessary to require the TS interface routines to perform name lookups rather than TSBSP. Currently the X25 interface routines do not include lookup facilities, though directories are defined. In the short term the ring routines will use local tables, but implementation of some form of name server will have high priority.

1.3 Security and Accounting

No accounting of local usage is required, and their is no defined mechanism for retrieving call statistics from a WAN gateway since the parameterisation of statistics in DISCONNECT is considered in the Yellow Book to be "an area for further study".

Users are expected to build security into the applications that use the transport service.

2. MESSAGES

The messages by which the UUP TS interface routines access the other modules are similar, but not identical to those defined for the X25 service [1]. In the remainder of this note messages will be given as a

octets followed by the to and from addresses in parentheses. The addresses are omitted when they are defined by the context.

2.1 Message Identifiers

The first octet of a message is an identifier. The following values are defined:

A_OK	31
ACCEPT	17
ADDRESS	20
CLOSE	18
COMMAND	40
CONTROL	41
DISCARD	42
EXPEDITE	22
FETCHDATA	28
FLUSH	43
ICP	44
INIT	26
LISTEN	25
OPEN	16
PORT	45
RESET	19
RST_ACK	32
SENDCONTROL	46
SENDDATA	27

2.2 Long Messages

Individual messages are limited in length to a compilation parameter MAXMESG (64 < MAXMESG <=256). Where there is a requirement for two processes to exchange a longer message it is broken up into messages of length <= MAXMESG by the sender. The receiver solicits each continuation message explicitly by sending the message A_OK,message_id to the sender. Continuation into a following message is indicated by setting the most significant bit of the message identifier. Each following continuation message will contain the same message identifier, and it is legal for the rest of the message in a continuation to be null. In the case of messages from TS to UUP it may happen that that a transport service event intervenes and the continuation is replaced by a message pertaining to the new event.

2.3 Message Exchanges

This section lists all the messages that may be exchanged between the modules with their meaning and usage.

2.3.1 UUP --> TSBSP

When UUP wishes to send a TS parameter to TSBSP the mechanism for long messages is used but UUP must divide the parameter up into fragments less than 60 octets in length.

INIT,0

First message from UUP to TSBSP when wishing to establish a connection. The reply will be A_OK,0.

INIT, local TS address

The second message from UUP to TSBSP. Defines the transport service address of UUP, to be used as the calling address if the next message is OPEN, or matched against incoming called address if the next message is LISTEN. The reply will be A_OK,0.

LISTEN,0

UUP sends this message and waits for a reply. The reply only comes when an incoming CONNECT specifies the local TS address as the called address. The reply is OPEN, maxdata, calling address.

OPEN, 0, called address

The station/port/function part of the called address when this process wishes to initiate a connection. There is no reply to this message.

OPEN, 1, called address

The remainder of the transport service called address. See section 2.3.5 below for the conventions to be followed for titles and addresses. Note that this component of the address can be of arbitrary length. UUP waits for a reply to this message which will be ACCEPT or CLOSE (see below).

SENDDATA, push flag

UUP wishes to transmit data, with following PUSH if push flag is non-zero. This is preceded by data (LINK, UUP), and the reply is A_OK,0 (UUP, TSBSP) or a TS event message.

FETCHDATA, 0

UUP is ready to receive up to maxdata octets of data. It is preceded by data (LINK, UUP). The reply is either FETCHDATA, push flag (UUP, TSBSP) followed by data (UUP, LINK) or a TS event message.

A_OK, message_id

Fetch continuation of a long message. Reply is the next fragment or a TS event message.

RESET, 0

Send a transport service RESET. Reply is A_OK or a TS event message.

EXPEDITE, octet

Send an expedited octet. Reply is A_OK,0 or a TS event message.

ADDRESS, address

Send a TS address. The address may be of arbitrary length, and is sent in fragments as detailed in 2.2.

CLOSE, 0

Close the connection and release all associated resources. No reply.

2.3.2 TSBSP --> UUP

TSBSP will always negotiate maximum packet sizes with the remote process in the range 64 - MAXMESG octets. This means:

(i) TS parameters arriving in a packet can be written immediately to UUP in one message.

(ii) TS parameters provided in fragments by UUP can be sent out to LINK as each fragment is received.

Thus only one buffer of size MAXMESG is required by TSBSP for message processing.

A_OK, 0
Reply on successful completion of requested action.

A_OK, message_id
Solicits next fragment of a long message.

ACCEPT, maxdata [,recall address]
Reply to successful OPEN. Maxdata is the maximum number of data octets to be sent by UUP for each SENDDATA message.

OPEN, maxdata [,calling address]
Reply to LISTEN. Maxdata is the maximum number of data octets to be sent by UUP for each SENDDATA message.

CLOSE, 1 [,address of error]
Reply to an unsuccessful OPEN. The logical connection between UUP and TSBSP is closed.

CLOSE, qualifier [,address of error]
DISCONNECT received as an error on the connection. Qualifier is numeric code. The logical connection between UUP and TSBSP is closed.

FETCHDATA, push flag
Reply to FETCHDATA if data arrives before any other TS event.

ADDRESS, ?, address
Received TS ADDRESS primitive. An alternative reply to FETCHDATA. (The ? will match the X25 use when that is known).

2.3.3 UUP <--> LINK

data
This is the only message between these two processes in either direction and so no message identifier is required. This enables transport service data to pass directly between LINK and the user process without copying by the interface routines. TSBSP will always negotiate maximum packet sizes with the remote end such that maxdata (see message OPEN (UUP, TSBSP)) is less than MAXMESG.

2.3.4 TSBSP <--> LINK

The LINK module is required to maintain two dedicated buffers for each active link, one for transmitting and one for receiving. To understand the usage of these buffers and the messages associated with them we note that there are three categories of information carried in packets:

(i) Four octets of TSBSP commands ie RDY, NOTRDY etc. TSBSP requires to fill in these on every packet transmitted and to see them on every packet received.

(ii) User data; indicated by the Data command with control bit (d3) zero in the third command octet. TSBSP has no requirement to see the data, only the TSBSP commands travelling with it.

(iii) TSBSP control messages; indicated by the Data command with control bit (d3) set in the third command octet. These control messages must pass through TSBSP in each direction to allow TSBSP to interpret them, reformat them, add/remove message type octet etc.

Thus the requirements of TSBSP are as follows:

- To fill the data portion of the transmit buffer with data supplied by the user.
- To fill the transmit buffer with control data from itself.
- To transmit the buffer any number of times with varying TSBSP commands.
- To transmit commands independently of the contents of the buffer.
- To receive all incoming TSBSP commands independently of the contents of the receive buffer.
- To receive the entire contents of packets marked as control data.
- To deliver the data portion of the packet to the user.
- For LINK to submit a receive request with BBP whenever the receive buffer is empty. These requirements are actually complicated by the need for TSBSP to handle the OPEN and OPENACK blocks of the initial connection protocol (ICP); for simplicity this is described separately [2.3.5].
- In all the messages between TSBSP and LINK it is necessary to identify the particular UUP connection to which they refer. In the following sections that identifier is a single octet called "channel". This is adequate with the message system to be used on PDP11.

Strictly speaking it should be possible for TSBSP to cancel a queued transmission request when sending a DISCONNECT but, since it is generally observed that transmission queues onto the ring are very short so that any cancellation would probably be ineffective, for simplicity's sake this facility has been omitted.

We also note that in general TSBSP does not inject any messages or data into the connection that have not been supplied by the user. The exceptions to this are internally generated or response DISCONNECT and RESET. In order to prevent clashing use of the transmit buffer it is required of LINK that if a message arrives to be loaded into a transmit buffer that is on the BBP transmit queue the message is discarded. TSBSP uses a command that simultaneously loads the buffer and places it on the transmit queue (SENDCONTROL) to send these special messages. (N.B. Given the restriction in the previous paragraph that there is no cancel transmit mechanism note that SENDCONTROL must not be issued until any outstanding transmission has completed.)

TSBSP --> LINK

OPEN, channel
Allocate buffers for this channel.

INIT, channel, station, localport[2], remoteport[2]
Configure local and remote ring connection addresses. LINK issues a receive request to BBP. See ICP.

COMMAND, channel, commands[4]
Transmit the four command octets only. The command portion of the transmit buffer may be used, overwriting previous contents.

SENDCONTROL, channel, commands[4], control data
Put the commands and control data into the transmit buffer and transmit.

CONTROL, channel, control data
Put the control data into the data portion of the transmit buffer.

SENDDATA, channel, commands[4]
Put the four command octets in the command portion of the transmit buffer and transmit the entire packet. Note that the buffer may have been loaded either by UUP with user data or by TSBSP with a previous CONTROL message.

FETCHDATA, channel
Deliver data portion of receive buffer to UUP. When the data message has been written to UUP then LINK issues a new packet receive request to BBP.

DISCARD, channel
Issue packet receive request to BBP.

FLUSH, channel
Cancel the receive request and then reply FLUSH, channel.

PORT, channel
Requests allocation of a new and unique Basic Block port number for this channel. Releases any previously allocated port number. See ICP.

CLOSE, channel
Cancel receive request and release dedicated buffers. Release allocated port number.

LINK --> TSBSP

COMMAND, channel, command[4], packet size
The commands from a packet received, of length "packet size", in which bit d3 of the third octet is zero. Note that "packet size" is in octets and includes the command octets. After sending this message LINK only issues a new receive request to BBP if packet size <=4 ie there was no user data included.

CONTROL, channel, command[4], control data
The entire contents of a packet received in which bit d3 of the third octet is one. After sending this message LINK issues a new receive request to BBP.

A_OK, channel
Signals completion of packet transmission attempt.

FLUSH, channel
Receive request has been cancelled.

PORT, channel, port number[2]
A new and unique port number. See ICP.

ICP, channel, fromstation, data
See ICP.

Note that there is a distinction between LINK having issued a packet receive request to BBP and TSBSP having sent a Rdy soliciting Data from the remote end of the connection.

Examples of the use of these messages follow. First for transmission of user data:

data (LINK, UUP)
User data is copied into the data portion of the transmit buffer.

SENDDATA, 0 (TSBSP, UUP)
After sending this UUP waits for a reply. When the sending stream is ready TSBSP can send the data...

SENDDATA, channel, commands[4] (LINK, TSBSP)

A_OK, channel (TSBSP, LINK)
acknowledges completion of packet transmission by BBP. Some time later an acknowledgement comes from the remote end eg...

COMMAND, channel, commands[4], packet size (TSBSP, LINK)
so TSBSP can tell user his action has been completed..

A_OK, 0 (UUP, TSBSP)

Alternatively, the last two messages could have been replaced by

SENDCONTROL, channel, commands[4], control data (TSBSP, LINK)
carrying a TSBSP Close command (TS DISCONNECT)

CLOSE, qual, address of error (UUP, TSBSP)

An example of receiving user data:

FETCHDATA, 0 (TSBSP, UUP)
User has requested some data and now waits for a reply.

COMMAND, channel, commands[4] (LINK, TSBSP)
TSBSP sends a Rdy command to the remote process.

COMMAND, channel, commands[4], packet size (TSBSP, LINK)
A packet carrying data arrives.

COMMAND, channel, commands[4] (LINK, TSBSP)
TSBSP sends Notrdy command to the remote process.

A_OK, 0 (UUP, TSBSP)
User is informed that data is available and immediately waits for message from LINK.

FETCHDATA, channel (LINK, TSBSP)
Instructs LINK to deliver data to user.

data (UUP, LINK)

TSBSP would need to make use of DISCARD if the data packet received was erroneous in some respect ie wrong sequence number. In place of data some other TS message might have arrived in which case the last 3 messages would be replaced by eg:

CONTROL, channel, commands[4], control data (TSBSP, LINK)

ADDRESS, address

2.3.5 Titles, Addresses and the Initial Connection Protocol

In this area to try to implement the "standard" is to try and hit a moving target. There is considerable confusion over whether the dual use of basic block port numbers for connection identifiers in TSBSP and subaddressing in the initial connection protocol should be allowed. There is also dispute over whether the function code should be used, and if so what for. It should be noted that in general a transport service address for use over the ring consists of station/port/function plus a further ascii string of indefinite length. In traditional BSP implementations the station/port/function numbers identify the service and there is no need for a further name. Faced with a choice of using either/both/neither of the port and function codes for subaddressing the following scheme is proposed.

For outgoing calls the user passes across either a service name in ascii or a station/port/function triple encoded in decimal and then in ascii

and an ascii name. The name component may be null. A service name is not allowed to start with a digit. The TS interface routines translate a service name into a station/port/function triple and a name either by local table lookup or interaction with a name server; the name component may be null. The station/port/function and name are sent to TSBSP in the OPEN messages. This offers complete flexibility in initiating a connection with any other addressing scheme.

For incoming calls neither the port nor function code will be used to subaddress services. Only the service name provided in the INIT message will be used. This is matched against a service name in a CONNECT message on an incoming OPEN. A single basic block port (zero) will be used for all incoming OPEN requests, the value of the function code will be irrelevant. This is a degenerate version of the current options and so might be expected to be compatible with the eventual standard.

The Initial Connection Protocol is implemented by TSBSP. This means that in addition to receiving control packets during the data phase of connections it must have a way of receiving the entire contents of OPEN and OPENACK blocks. This is the use of the ICP message from LINK to TSBSP. LINK compares the first octet on incoming blocks with the bit patterns of the Open and Openack commands of ICP; if a match is found then the ICP message is sent with the entire contents of the block. For the proper management of the protocol it must also be able to acquire new and unique port numbers for use as connection ports. If an OPEN block is sent and no OPENACK is received in the timeout period then it is repeated with a different connection port. In order to ensure that no packets are erroneosly handled by TSBSP on the wrong port number (by incoming packets crossing with outgoing commands in the message system) the FLUSH message is used. When TSBSP wishes to change the port number in use on a channel it sends the FLUSH message before reconfiguring the channel. On receiving FLUSH, LINK cancels the outstanding receive request and echoes back FLUSH. Thus TSBSP can then ignore all messages arriving on that channel until it receives a FLUSH in reply.

The following message exchanges illustrate the use of the messages between TSBSP and LINK - for simplicity the messages exchanged between TSBSP and UUP are omitted.

Connection Initiator:

```
        OPEN, channel (LINK, TSBSP)
loop:   PORT, channel (LINK, TSBSP)
        PORT, channel, port number[2] (TSBSP, LINK)
        FLUSH, channel (LINK, TSBSP)
        INIT etc with allocated local port number
        SENDCONTROL, channel, commands, data (LINK, TSBSP) - send the Open
        .
        .
        FLUSH, channel (TSBSP, LINK)
        if timeout then goto loop
        ICP, channel, station, data (TSBSP, LINK)
        INIT with new remote port number received in the Openack
```

Connection Responder:

OPEN, 0 (LINK, TSBSP)
Channel zero is reserved for handling incoming Opens.

PORT, channel (LINK, TSBSP)

PORT, channel, port number (TSBSP, LINK)
Get a new port number.

INIT, 0, 0 0, 0 0, 255 (LINK, TSBSP)
Configure channel 0 to receive as Basic Block port zero from anywhere.

ICP, channel, station, data (TSBSP, LINK)
An Open (or Openack) block has been received from "station". If it is an Open block specifying the user's service name then...

INIT, channel etc (LINK, TSBSP)
Init the channel with its novel local number and the remote port number communicated in the Open.

SENDCONTROL, 0, commands, data (LINK, TSBSP)
Send the Openack. If there are no other listens outstanding then...

CLOSE, 0 (LINK, TSBSP)

2.3.6 Name Server

This section describes the messages by which processes look up names with the local name server process; it does not the service specification of a ring name server.

--> NS

service name
Serivce name in ascii, beginning with a letter.

<-- NS

station, port, function [,TS address]
Station, port and function are numeric values encoded in decimal and then in ascii. The TS address is an ascii string.

3. TSBSP MESSAGE INTERFACE ROUTINES

The definition of these routines is to be machine independent. Their impelementation will be machine dependent.

To be completed.

4. MACHINE SPECIFIC IMPLEMENTATION

4.1 PDP11 Implementation

In the first implementation TSBSP resides in user space. LINK and BBP reside in the kernel. The message routines are accessed by UUP and TSBSP via the special files /dev/tsbsp?. Minor device zero is used by TSBSP and others are used by UUPs. Each minor device is uncommitted to any connection, and may be used for only one connection at a time. One minor device is used by the name lookup service. Messages are stored in the clist space used by Unix for queues for character devices. The dedicated buffers required by LINK for sending and receiving are taken from the buffer pool. One buffer will be taken for each connection; it is then divided to provide the two buffers. The defined message interactions allow long messages (ie those with user data or TS control messages) to be stored directly in these buffers with pointers to them in the clist message queues.

LINK and BBP are not separate Unix processes but are event driven.

4.2 Perq Implementation

Initially it may be possible to carry over the PDP11 message system as it stands, together with the LINK and BBP modules, and run it as a process on ACCENT. Evaluation of this is an area for further study. Integration of the modules into the message system of ACCENT would be a second phase of development.

REFERENCES

(1) K Ruttle, York Transport Service Interface (Working Draft). Feb 82.

APPENDIX 1A

NAME

bb - Cambridge Ring Basic Block protocol

DESCRIPTION

This section describes an interface to the Cambridge Ring; facilities are provided for users to send and receive blocks of data according to the Cambridge Basic Block protocol. Only type 0 and 3 blocks may be used. There is no provision for sending or receiving single minipackets.

The driver handles blocks of up to 512 bytes. The user is responsible for the blocking and unblocking of data but the driver deals with the encapsulation of the data in header, route and checksum packets for transmission and strips them off on reception.

The files bbp? refer to minor devices which access individual ports in the Basic Block driver. Each port can be used for both sending and receiving blocks and has associated with it one of the following structures, defined in <sys/port.h>:

```
struct portinfo {
        char            pi_type;
        unsigned int    pi_inport;
        unsigned int    pi_station;
        unsigned int    pi_outport;
        unsigned int    pi_accept;
};
```

Basic blocks are routed to ring stations and to a port at that station. The port number is in the range 0-4095. Pi station and pi outport specify the destination station and port respectively of data written by the user on this port. Pi inport is the port number which directs incoming blocks to this port; pi_accept determines the ring stations which are acceptable sources of such blocks. One particular source may be specified otherwise the special symbolic values ANYONE and NOONE may be used (defined in <sys/port.h>). The portinfo structure is loaded using the following ioctl command:

```
#include <sys/port.h>

ioctl(fildes, BBPSET, info)
struct portinfo *info;
```

where fildes is the port file descriptor; and may be read using:

```
ioctl(fildes, BBPGET, info)
```

If the symbolic value DYNAMIC (defined in port.h) is loaded for pi inport then the driver allocates a currently unique value in the range 1000-4095 which is available to subsequent BBPGET commands.

Another ioctl command available on an open port has the form

```
#include <sys/port.h>

ioctl(fildes, BBPENQ, enq)
struct portenq *enq;
```

where portenq is the following structure (defined in <sys/port.h>):

```
struct portenq {
      unsigned int      pn_sender;
      char        pn_xrslt;
};
```

Pn sender contains the source address (ring station) of the last block received for this port, which is only of interest if pi accept has been set to ANYONE. Pn xrslt is the result of the last block transmission. Symbolic values (defined in port.h) are: BB_ACCEPTED, BB_BUSY, BB_DEAF, BB_ABSENT, BB_ERROR.

FILES

/dev/bbp?

DIAGNOSTICS

Some of the error codes defined in <errno.h> have special meanings when they occur as a result of read, write and ioctl commands to the Basic Block driver:

EACCES

The driver has detected failure of the ring (minipacket transmission timeout). This error is also returned by: read or write on a port that has not been configured with BBPSET or that specifies more than 512 bytes; attempting to load a pi inport value with BBPSET that is already in use by an open port.

EIO

As a result of read this means that the received block was larger than the number of bytes requested. As a result of write it means that the block was not accepted and that the reason for non-delivery may be found by BBPENQ.

BUGS

BB type 0 blocks are defined in terms of 16 bit minipackets. If an odd byte count is specified on a write then it is rounded up to an even number. The extra byte is guaranteed to be garbage.