MULTICS TECHNICAL BULLETIN

To: MTB Distribution

From: Charles Hornig

Date: 26 June 1979

Subject: Pseudo-terminal support for Multics

## Introduction

This MTB describes a proposal to provide support for `pseudo-terminals' under Multics. A `pseudo-terminal', or PTY, is a software object which appears to one process as a MCS terminal channel and to another, its `owner', as an I/O device. This channel behaves in the standard way and can be used in accordance with its service type just as a FNP terminal. The PTY's owner process plays the part of the physical device. It signals dialups, hangups, and quits; provides data for input; and accepts data written over the terminal channel. The owner process in effect simulates a top level demultiplexor. PTY's are used with success on many other operating systems.

## Why have them?

PTY's provide a means for a user process to simulate a ring zero demultiplexed channel. This means that support for new communication protocols which would normally be implemented as ring zero demultiplexed channels can instead be done in a user process. This makes support for experimental and infrequently used protocols much simpler. While a process using PTY's is clearly much less efficient than support in ring zero, user ring code is also far easier to develop and debug. PTY's can be a `quick and dirty' way to support new protocols. This is especially appropriate for site-specific protocols. Sites often do not have the experience or the time to develop a ring zero demultiplexor and would prefer to pay the user ring performance cost.

PTY's are also appropriate for some network protocols which are difficult to implement as demultiplexors. PTY's could be used to allow remote login for a network without putting all the network support into ring zero.

PTY's also can provide the `remote login' function for a Multics site which is not on a computer network. It is often desirable to be able start up a new process `on the side'. Users at current Multics ARPAnet sites often do this now by opening an

ARPAnet connection back to the local host. Using PTY's for this function will allow all Multics sites to do this in an efficient manner. In addition, the user could have the intermediate process perform conversions on the data it handles. An example of this is the EMACS TELNET window capability, which could easily be extended to work with PTY's.

The remote login capability could also be used to develop more sophisticated benchmarking tools. A driver program could log in many remote jobs and simulate an interactive workload more accurately than present tools.

## Implementation

PTY's will be implemented as a MCS top level ring zero demultiplexor. PTY channels will be defined in the CMF with names like "pty.xxx". All CMF parameters can be specified for them as for any other channel. "pty" is defined as a multiplexor channel of type "pty". On the MCS side PTY's will support only those control orders processed by the TTY DIM. They will support all TTY DIM modes and those modes defined by the PTY owner at attach time. There will be a gate, "pty_", into ring zero to control the owner side of PTY channels. Only users with access to this gate can control PTY's. The functions provided are:

attach: Simulate dialup of a listening PTY channel. The caller can specify a star name to specify to what channels he wishes to be connected and a structure defining which modes are permitted for this channel. The system will search the PTY's whose names match the given star name for one on which a "listen" order has been given. If one is found the MCS channel is given a dialup interrupt and the caller is givan a handle by which he can refer to the channel.

write: Write data into PTY input buffer. Moves characters into the PTY input buffer. If the PTY was waiting for input a wakeup is sent. Users will not be permitted to queue more than a certain amount of data at a time in ring zero. If this amount is exceeded ring zero will not take all the supplied characters and will send a wakeup to the owner when more space is available.

read: Read from PTY output buffer. If there are not enough characters in ther output buffer to fill the callers buffer a wakeup will be sent when more become available.

interrupt: Send a `quit' signal. Causes "quit" to be signalled in the PTY's process if it has enabled quits.

detach: Simulate hangup of PTY channel. The PTY channel is hung up and the channel is released.

Name: pty_

The pty_ subroutine gives the user access to the Multics pseudo-terminal facility.

Entry: pty_$attach

This entry attaches a pseudo-terminal and simulates a dialup on the associated MCS channel.

Usage:

```
declare pty_$attach entry (char (*), ptr, fixed bin (71),
     fixed bin, fixed bin(35));

call pty_$attach (name, pty_data_ptr, ev_chn, subchan,
     code);
```

where:

1. name (Input)
   is a star name specifying the class of PTY channels desired.

2. data_ptr (Input)
   points to the following structure, declared in ???.incl.pl1:

   (This structure will contain information about which modes are legal for the channel.)

3. ev_chn (Input)
   is the name of an event channel over which wakeups will be sent then output data is available or there is room for more input data.

4. subchan (Output)
   is the PTY subchannel assigned. This number is used by the other entries to specify a particulat channel.

5. code (Output)
   is a standard system status code.

Entry: pty_$read

This entry reads characters out of the PTY's output buffer into the caller's buffer. If the buffer is not filled a wakeup will be sent when more data is available.

Usage:

```
declare pty_$read entry (fixed bin, pointer,  fixed bin  (21),
     fixed bin (21), fixed bin (35));

call pty_$read (subchan, buff_ptr, nelem, nelem_tr, code);
```

where:

1. subchan (Input)
   is the subchannel number returned by pty_$attach.

2. buff_ptr (Input)
   is a pointer to the caller's buffer.

3. nelem (Input)
   is the size of the caller's buffer in characters.

4. nelem_tr (Output)
   is the number of characters actually transerred. If this is less than nelem a wakeup will be sent when more data are available.

5. code (Output)
   is a standard system status code.

Entry: pty_$write

This entry moves data from a caller buffer into the PTY's input buffer.

Usage:

```
declare pty_$write entry (fixed bin, pointer, fixed bin
     (21), fixed bin (21), fixed bin (35));

call pty_$write (subchan, buff_ptr, nelem, nelem_tr, code);
```

where:

1. subchan (Input)
   is the subchannel number returned by pty_$attach.

2. buff_ptr (Input)
   is a pointer to the caller's buffer.

3. nelem (Input)
   is the size of the caller's buffer in characters.

4. nelem_tr (Output)
is the number of characters actually transferred. If this is less than nelem a wakeup will be sent when more space is available.

5. code (Output)
is a standard system status code.

Entry: pty_$interrupt

This entry causes the "quit" condition to be signalled by the PTY if it has enabled quits.

Usage:

```
declare pty_$interrupt entry (fixed bin, fixed bin (35));

call pty_$interrupt (subchan, code);
```

where:

1. subchan (Input)
is the subchannel number returned by pty_$attach.

2. code (Output)
is a standard system status code.

Entry: pty_$detach

This entry detaches the PTY channel and simulates a hangup on the MCS channel.

Usage:

```
declare pty_$detach entry (fixed bin, fixed bin (35));

call pty_$detach (subchan, code);
```

where:

1. subchan (Input)
is the subchannel number returned by pty_$attach.

2. code (Output)
is a standard system status code.