SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

DISTRIBUTED INTERACTIVE COMPUTING NOTE 747

PERQ UNIX IMPLEMENTATION NOTE # 44
Accent/Unix: TermDriver

Issued by
J.C.Malone

26 January 1983

DISTRIBUTION:
R W Witty
K Robinson
C Prosser
A S Williams
L O Ford
T Watson
E V C Fielding
J C Malone
P J Smith
A J Kinroy
J M Loveluck
C P Wadsworth
Martin Ritchie (ICL Dalkeith)
P Palmer (ICL Dalkeith)
RL Support/PERQ/Unix Implementation Notes file

The terminal driver (TermDriver) handles all input via keyboard and output to screen.

The TermDriver process is started up by Init [5], together with any other drivers.

It fathers login sessions, starting a new session whenever the previous one terminates.

1. MAINTAINING CHILD PROCESSES

The terminal driver forks a child process, which execs Getty (for boot sequence, see [5]).

The child calls SetPGrp to become the head of a new process group [2], so that on ^C it is possible for the Terminal Driver to kill the child's process group without killing itself.

The Registrar [3] Wait function would inform the Terminal Driver when a child has died, returning its process id. However, it is not desirable for the driver to hang on a Wait system call [8] and so the Wait request message is sent to Registrar without waiting for the reply [4]. The Wait reply will be received in the main control loop and dealt with in ServiceMsg, the StartWait routine being called to reissue the Wait.

procedure ExecGetty
: Uses the StartChild procedure in the MidWife module to fork and exec the login session.

procedure StartWait
: The Wait request message is sent to Registrar. Called at the beginning of TermDriver and upon receipt of a reply. Also used in shutting down, when waiting for all processes in child's process group to die.

procedure CheckChildren
: The reply to the Wait message has been received. If the dead child was the login session, then a new session is begun.

function GetPId
: The Wait message reply is decoded to get the process id of the dead child.

2. KEYED INPUT

The Canvas server [1] is used to receive keyed input in a similar manner to the use of Registrar for Wait requests (see above): a GetKey message is sent to Canvas whenever TermDriver is ready for the next key and the reply is received in the control loop.

procedure StartKey

The GetKey request is sent to Canvas. Called at the beginning of TermDriver and whenever a reply is received.

function WaitingKey

If the TermDriver has been requested to write a large amount of data, it is desirable to check for keyed input during the write, particularly when the user has typed ^C or ^S (see TermWrite).

This function returns a true value if the Accent supplied boolean array shows that the Canvas reply port has queued waiting messages.

procedure ReceiveKey

Receives a message from Canvas and requests another key. This is used by TermWrite when a key is known to be waiting, or when a key is needed to restart the display.

procedure DecodeKey

Given a GetKey reply message, obtains the key typed. Those control keys which are interpreted by Canvas as command keys must be translated back into their ASCII values.

3. SCREEN DISPLAY

Keyed Input is kept in a circular buffer (Buff), awaiting read requests. CRCntlZCount always holds the number of CR and EOF keys currently unprocessed in the buffer, so that it is always possible to tell whether there is a full line of input ready.

When a key is received, it is normally echoed, using the xTypeScript module, and placed in the buffer. However, certain control characters have special meanings and are neither echoed nor passed on to the user.

Note that when the current line is to be redisplayed (i.e. after ^H or ^U) the xTypeScript module is given the number of keys to ignore and takes care itself of erasing control characters (two characters) and tabs (up to eight spaces).

Having calculated the window size, TermDriver provides a paging mode.

procedure FlushBuffer
Empties Buff and resets CRCntlZCount.

function LastKey
Returns the position in the buffer of the last key typed.

procedure GetLine
Finds the beginning of the current line in the buffer, and its length. Used when erasing and redisplaying line.

procedure ActOnKey
Given the ASCII character just typed, decides on the action to be taken.

Currently:

```
          ^C - kill child's process group, and FlushBuffer
          ^H - backspace
          ^I - tab
          ^P - switch paging mode
          ^Q - ignored (used to restart printing after ^S
                        without being passed on to user)
          ^R - retype line
          ^S - halt printing until next key is typed
          ^U - erase current line
          ^Z - EOF
        null - null or unrecognised keys are ignored
   otherwise - echo and add to buffer
```

4. QUEUE WAITING READ REQUESTS

Since the TermDriver must wait for a CR before supplying a line to a reader, and since the method of input is relatively slow, a read may take any length of time to be satisfied. It is not feasible for the TermDriver to wait until the read request has been answered before receiving any other requests, so read requests must be queued until satisfied.

When a user process receives an interrupt [9] and is in OBOE waiting for a reply from a server [4], the interrupt is usually ignored until the reply has been received. However, since read requests are queued by the TermDriver, the read call can no longer be thought of as an indivisible remote procedure call. In order to avoid the delay in handling the interrupt, the read reply must be forgotten. To prevent TermDriver from sending an extraneous message back to OBOE (and thus losing type), and also to clear obsolete requests from TermDriver's queue, OBOE sends a CancelReq message to TermDriver, and waits for a reply. If the request is still in the queue, then TermDriver dequeues it and replies immediately. Otherwise, the request has been satisfied and the reply already sent. Either way, OBOE receives a prompt reply and may handle the interrupt.

procedure QRead
    Add message to ReaderQ as soon as it is received.

procedure DQRead
    Remove request from head of ReaderQ and dispose message.

procedure AttemptReads
    Called by ServiceMsg if there is a line of input ready, to satisfy as many read requests as possible.

procedure CancelRead
    A CancelReq message has been received, remove the request from the queue and send acknowledgement, if the read is still outstanding.

procedure KillRequest
    Search for message from given port in ReaderQ (there will only be one request from any one process) and, if found, squeeze the entry from the queue by moving other entries down and dispose of message.

5. IDENTIFY MESSAGE

procedure ServiceMsg
Called by the control loop to handle message received:

When a key is supplied by Canvas (see KEYED INPUT), another key is requested; the message is decoded to produce the character, which is passed to ActOnKey;

When Registrar replies to the Wait request (see MAINTAINING CHILD PROCESSES), CheckChildren is called to see if the child must be restarted and another Wait is issued;

On CancelReq, CancelRead is called to remove unwanted read requests from ReaderQ.

Read requests are queued;

Otherwise, TermServer is called to satisfy the user request on this server.

After processing message, AttemptReads is called to satisfy queued reads, if there is a line of input waiting.

6. TERMINAL SYSTEM ROUTINES

Function TERMServer
Decodes server request message, calls relevant service routine and encodes reply message [4].

procedure TermFStat
Calls on SwitchBoard to perform Stat on /dev/tty.

procedure TermLSeek
Checks the arguments are valid, and returns offset value (as on 11/70).

procedure TermIOCtl
Returns OK, but No-Op.

procedure TermClose
Returns OK, but No-Op.

procedure TermOpen
Called indirectly, via SwitchBoard, and returns user Port to be passed back by SwitchBoard. All future requests from the user will be made to this port.

procedure TermFork
No-Op.

procedure TermRead
Returns characters from Buff, up to the number requested, giving the virtual address of the buffer plus the offset of the beginning of the data. If the line has overflowed around the end of the buffer, then the buffer must be rotated so that the data is returned contiguously.

It is here that CRs are translated into LFs before being returned.

When all the characters up to an EOF character have been processed, the EOFFlag is set. The purpose of this is to clear all waiting read requests: AttemptReads will continue to loop through ReaderQ and the readers will receive EOF.

procedure TermWrite
Displays characters on screen, using xTypeScript module.

Accept keyed input during large writes, allow for paging mode and frozen display (after ^S).

In paging mode, after each window of output, CR will produce one more line, while any other key advances to the next page. These keys are all thrown away, apart from ^C, which is obeyed.

7. SHUT DOWN

procedure CatchSigHup

Nominates ShutDown as the signal handler for SIGHUP [9]. This is to shut the device driver down as cleanly as possible [5].

function ShutDown

Sends SIGKILL to all processes in the child's process group.

Then services all requests (ensuring that EOF is returned to any read requests, by calling FlushBuffer and ignoring any further keys) until there are no more children left.

Checks that there are no more children left by waiting for a -1 return from the split wait call, StartWait.

Then exits.

8. INITIALISATION

procedure TerminalRegister

Registers with SwitchBoard [6], by sending the tty device filename and open request port [7].

procedure TermInit

Initialises modules used and TermDriver's variables and ports.

Calculates size of the window currently running in, for paging.

9. MAIN PROGRAM

After initialisation, sets up the child process (using ExecGetty) and signal handler (using CatchSigHup).

Then issues the first requests to Registrar (StartWait) and Canvas (StartKey).

Remains in a loop, receiving messages and calling ServiceMsg to process them.

REFERENCES

1. J. EUGENE BALL, Canvas: the Spice graphics package, Dept of Computer Science, Carnegie-Mellon University, October 81.

2. L.O. FORD, "Perq Unix Implementation Note # 53 - Accent/Unix: Process ids and process groups", DIC Note # 758, Rutherford Appleton Laboratory (December 82). [to be published].

3. L.O. FORD, "Perq Unix Implementation Note # 42 - Accent/Unix: Registrar", DIC Note # 745, Rutherford Appleton Laboratory (December 82). [to be published].

4. L.O. FORD, "Perq Unix Implementation Note # 45 - Accent/Unix: Overview of Serverloops and remote procedure call interfaces", DIC

Note # 748, Rutherford Appleton Laboratory (January 83). [to be published].

5. J.C. MALONE, "Perq Unix Implementation Note # 43 - Accent/Unix: Boot Sequence", DIC Note # 746, Rutherford Appleton Laboratory (January 83).

6. J.C. MALONE, "Perq Unix Implementation Note # 41 - Accent/Unix: Switchboard", DIC Note # 744, Rutherford Appleton Laboratory (January 83).

7. J.C. MALONE, "Perq Unix Implementation Note # 51 - Accent/Unix: Special Files, Mountable File Systems", DIC Note # 756, Rutherford Appleton Laboratory (January 83).

8. A.S. WILLIAMS, "Perq Unix Implementation Note # 31 - UNIX System Call Specification", DCS Note # 727, Rutherford Appleton Laboratory (November 1982). [In Preparation].

9. A.S. WILLIAMS, "Perq Unix Implementation Note # 50 - Accent/Unix: Signals and deadports", DIC Note # 755, Rutherford Appleton Laboratory (December 82). [to be published].