my Lee

SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

DISTRIBUTED INTERACTIVE COMPUTING NOTE 745

PERQ UNIX IMPLEMENTATION NOTE # 42
Accent/Unix: Registrar

Issued by
L.O. Ford

23 December 1982

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

Registrar is the process manager for Unix processes. It keeps track of the births, deaths and ancestry of Unix processes and makes them visible to the user as procman[1] makes visible all processes running on top of Accent, including Unix processes. See also[2].

Registrar is also the Signal manager for Accent/Unix. It accepts signal requests from user or system processes and sends then, using Accent emergency messages, to the appropriate process or group of processes[6]. Alarm signals are generated by Registrar.

Wait requests are handled by registrar and when it is notified that a child has died satisfies the wait request of the parent.

1. Important data Structures

1.1 ProcTable

```
Const
    MAXNOPROCESSES = 64;
```

```
Type
    Alarmtype     = record
                        Flag            : boolean;
                        Clock           : long;
                    end;

    Deadtype      = record
                        Defunct         : boolean;
                        Deadstatus      : long;
                    end;

    Etimestype    = record
                        Elapsed         : long;
                        EDeadChildren   : long;
                    end;

    Protecttype   = record
                        UID             : long;
                        EUID            : long;
                        GID             : long;
                        EGID            : long;
                    end;

    Rtimestype    = record
                        RunTime         : long;
                        RDeadChildren   : long;
                    end;

    Signaltype    = long;

    Waittype      = record
                        State         : boolean;
                        Status        : long;
                        PID           : long;
                    end;

    Signals       = 1 .. NSIG;

    ProcEntry = record
                        ProcessID     : long;
                        DataPort      : Port;
                        KnPort        : Port;
                        ParentID      : long;
                        ProcessGrp    : long;
                        Protect       : Protecttype;
                        Alarm         : Alarmtype;
                        Dead          : Deadtype;
                        Waitx         : Waittype;
                        Signal        : Signaltype;
                        Etimes        : Etimestype;
                        Rtimes        : Rtimestype;
                        Lastcom       : String;
                        IgnoredSigs   : set of Signals
                end;
```

```
PTableType = array [1..MAXNOPROCESSES] of ProcEntry;
```

The unique identifier used by Registrar to identify a process is its DataPort. This is for historical reasons as at one time it was not thought necessary to store the processes kernelport and the dataport was needed to send signals (Accent emergency messages)[6].
However, it was found necessary to store the kernelport to enable Registrar to satisfy PS requests and to terminate the process on an Exit request.

1.2 WaitTable

```
MAXNOWAIT      = MAXNOPROCESSES;

MessArray = array [1..MAXWMESS] of integer;

Messbuff  = record
                    Hdr  : Msg;
                    Data : MessArray;
            end;

ptrMess   = ^Messbuff;

WaitEntry = record
                    Ptr   : ptrMess;
                    DataP : Port;
            end;

WaitTable      : array [1..MAXNOWAIT] of WaitEntry;
```

2. Messages Processed by Registrar

```
Alarm
Exec          - Registering new command line
Exit
Forkr         - Registering portion of Fork system call
Getug         - Getuid, Geteuid, Getgid, Getegid
Getpid
Kill
Newprgrp      - Generate new unique process group
Ps            - Return info for Ps program
Setsignal     - Toggle state of ignored signal flag
Setug         - Setuid, Seteuid, Setgid, Setegid
Times
Wait
```

3. Procedures

The following are the major procedures and functions within Registrar. The routines generated by Matchmaker[5], are not documented here

and reference should be made to[3].

Function PMREGISTERPROCESS

Registers new Unix process with Accent/Pos process manager Procman[1].

function IOGetTime

Gets current elapsed time since Accent booted in milli-seconds.

procedure GetPrFree

Returns an index to a free entry in ProcTable. Free entries in the table have field ProcessID set to NOENTRY.

procedure RemoveProc

Removes an entry in the ProcTable. Sets ProcessID to NOENTRY and clears other fields appropriately.

procedure InitRegistrar

Initialises the ProcTable and WaitTable, establishes Init process by entering it in its tables and informing Procman and sets the counter for generating PID's[2].

Function Ignored

Checks to see if the given signal refers to a signal which is in the set being ignored by the process.

Function CheckChild

Returns NOTFOUND if the process has no children otherwise returns the index within ProcTable of its first child.

procedure DumpProcTable

Dumps the contents of the ProcTable in the case of inconsistencies being detected.

procedure GenNewPID

Generates a new unique Process Id[2] which lies in the range 1 to 2147483647. Process id 1 is the Init process[4].

procedure GenNewPrGrp

Generates a new unique Process Group[2] which lies in the range 1 to 2147483647.

Function GetWaitEntry

Scans the WaitTable and returns the index of the first free entry indicated by field Ptr being NIL.

Function PortToWaitIndex

Scans the WaitTable and returns the index of the entry matching the DataPort.

function PIDtoindex

Scans the ProcTable and returns the index of the entry matching the PID.

function PortToPIDindex

Scans the ProcTable and returns the index of the entry matching the DataPort. Ignores defunct processes which may have had the same DataPort as a later process.

Procedure KillProcess

Kills the given process. First it checks to see if the process owns any children and if so they are inherited by Init. It then checks to see if any parent is waiting for the process to die and if so sets the field Waitx.PID of the parent to the dying processes ProcessID; allowing the control loop to notify the waiting parent. The dying process is then marked as defunct in the ProcTable.

Procedure PrintAbnormalMessage

Prints the contents of an abnormal message. An abnormal message is defined as one which Registrar has received but was not expecting.

Procedure CheckAbnormalMessage

Checks abnormal messages for PORT_DELETED ones and if the port matches either the kernelport or the dataport of a known living process kills that process.

Function ShortestAlarm

Scans the ProcTable and returns the Shortest time remaining on any alarm clock in 60ths of a second.

Procedure GetRunTime

Returns for the specified process the cpu time used in 60ths of a second.

Procedure UpdateAlarm

Scans the ProcTable and updates the time remaining on Alarm clocks and the elapsed time of each process. The times are kept as 60th second.

Procedure SendSignal

Sends a signal to the specified process if it is not ignoring that particular signal.

Procedure ALARM

Processes Alarm system call.

Procedure EXITX

Processes Exit system call. Updates processes runtime before terminating the process and calling KillProcess.

Procedure FORKR

FORKR is called by a parent process on a Fork system call. It notifies Registrar that a Fork operation has taken place and supplies the childs information.

Registrar generates a new unique PID[2], enters the new processes information in ProcTable and informs Procman[1]. Registrar also registers the parent if it is not known.

Procedure GETPID

Processes Getpid system call.

Procedure KILL

Processes Kill system call. Sets field Signal in ProcTable for either one process, all processes in senders process group or all processes, unless the signal is being ignored, so that the control loop can send signal messages.

Procedure WAITT

Processes Wait system call. Checks to see if any of the processes children are marked as defunct so that the wait request can be satisfied immediately. If this is so the first one so marked in ProcTable is selected. The entry for the defunct child is removed from ProcTable.

If there are no defunct children Sendmsg is marked as WAITX so that the control loop can queue the wait request until it is notified that a child of the parent has died.

Procedure PS

Passes back process information for the PS command. This information is either supplied from the ProcTable or by sending Status requests to Accent.

Procedure SETUG

Satisfies Setuid, Seteuid, Setgid, and Setegid system calls.

Procedure GETUG

Satisfies Getuid, Geteuid, Getgig and Getegid system calls.

Procedure Exec

Used by exec system call to notify Registrar of the changed command line so it can be supplied on a PS call. Registrar passes on the changed command line to Procman[1].

Procedure Newprgrp

If the relationship is acceptable to Registrar it creates a new unique process group[2].

procedure SetSignal

Switches the state of signal between being ignored and caught.

procedure Times

Updates the process run times and returns them to the caller in 60ths second.

Procedure ServerLoop

This procedure contains the main control loop for registrar. The functions of this main control loop are:
a) To issue receive request to Accent - the time set in the parameter Maxwait is the time to the first alarm call. Therefore when control returns from Accent one either has a message to process or an alarm signal must be sent.
b) Updates elapsed times and alarm clocks and sends any alarm signals which are due.
c) If a normal message has been received it calls RegServer to perform the requested function. On return from RegServer the variable Sendmsg informs the control loop whether the reply may be sent back immediately to the user (YESX), is a wait request which cannot be satisfied now (WAITX) or was an Exit call and the process has been terminated (EXITXX).
d) A check is made to see if any signals need to be sent - field Signal in ProcTable not being NOSIG. If a signal is being sent to a process which is waiting then the process is removed from the wait queue.
e) A check is made to see if any waiting processes may be released - field Waitx.State being true and Waitx.PID being not equal to NOENTRY. RegServer is called with the saved message for the wait request to be satisfied.

REFERENCES

1. L.O. FORD, "Perq Unix Implementation Note # 47 - Accent/Unix: Overall design", DIC Note # 750, Rutherford Appleton Laboratory (December 82).

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 # 45 - Accent/Unix: Overview of Serverloops and remote procedure call interfaces", DIC Note

748, Rutherford Appleton Laboratory (December 82). [to be published].

4. J.C. MALONE, "Perq Unix Implementation Note # 43 - Accent/Unix: Boot Sequence", DIC Note # 746, Rutherford Appleton Laboratory (December 82). [to be published].

5. MARY THOMPSON, MICHAEL B. JONES, KATHIE FERRARO, AND KEITH WRIGHT, Matchmaker: A Remote Procedure Call Generator, Dept of Computer Science, Carnegie-Mellon University, November 82.

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