Draft 6.5.68

Identification

Overview of Quit

Robert L. Rappaport, Michael J. Spier

Purpose

Sometimes a process may wish to stop another process' execution. A typical example is that of a system operator who wants to stop all the processes in the system prior to a general system-shutdown. Another example is that of the Overseer process which stops the execution of a working process. We name this operation the 'quitting' of a process; as a result of it, the target process is put into the 'quit' state.

Technically, a quit process is similiar to a blocked process, the difference being that a process is forced into the quit state, and that it enters this state without the assurance that it might ever come out of it again.

Discussion

As mentioned above, a quit process is not guaranteed to ever run again; indeed, a quit process is more often than not on its way to destruction. Consequently, putting a process into the quit state must include safeguards to assure that the sudden disappearance of this process from the midst of an interacting multiplexed computer system will not cause any damage to the system.

By definition, a process is the execution of a virtual processor within the boundaries of a private memory space; consequently, a process can cause system-damage only in places where its memory-space overlaps the memory-space of one or more other processes. By convention, all systemwide supervisor data bases are located in the hardcore ring. Rather than try and keep track of a process' execution in order to be able to find out whether or not it is currentl manipulating such a data-base, it has been agreed never to quit a process while it executes in the hardcore ring. Thus it is guaranteed that a quit process always leaves systemwide data-bases in a predictable

state.

Another delicate subject is that of restarting a quit process, for example following a system shutdown, or after a user has been subject to an automatic logout. If a quit process is "stopped dead in its tracks" then in order to be able to restart is we would have to conserve its stack history symbolically (machine addresses are no good because by the time the process is restarted, one or more of the procedures which it was executing might have changed).

Rather than go into all that expense, a scheme is used which guarantees a standard (and known) ring 0 history for all quit processes; thus in restarting a process one can reconstruct the process' history without actually having had to remember its stacks.

Quitting strategy

The Traffic Controller entry point 'quit' is called whenever a process wishes to quit another process (or possibly itself.)

```
call quit(B)
```

where 'B' is the target process' ID,sets in that process' Active Process Table (APT) entry a flag known as the 'quit_pending' flag.

Whenever a process is chosen to run, that flag is placed in the processor's interrupt cell which is assigned to the quit interrupt. A process is masked against that interrupt for as long as it executes in the hardcore ring. Whenever a process abandons its processor, the quit interrupt cell is remembered in the process' quit-pending flag. This strategy insures that a process does not lose the received quit signal, and that it will not be affected by that signal as long as it is in the hardcore ring.

As soon as the process executes outside of the hardcore ring (as soon as the quit interrupt is unmasked) it gets 'hit' by the quit interrupt; it diverts its execution into the quit-interrupt handler which in turn calls the Traffic Controller

call i_quit

Subroutine i_quit puts the process APT entry on the list of blocked processes and gives its processor away. Thus the process is made to quit itself and therefore leaves its ring 0 history in a known state.

There is only one case in which a process must be quit within the hardcore ring. It is the case of a blocked process which happens to be in ring 0 because it called block in behalf of the user. In order to allow the quitting of such a process, subroutine quit always sends a wakeup signal to the target process, and whenever a process returns from subroutine block it gets momentarily unmasked against quit interrupts. In this way, if a process is in ring 0 in behalf of the user, it does get 'hit' by the quit interrupt which makes it call i_quit.