Identification

Quit_proc

Michael J. Spier

Purpose

This section defines the overall strategy applied when quitting a process. A "quit" process does not, basically, differ from a "blocked" process; both processes have given their processors away for an undetermined length of time. However, the process that calls block does so knowing that some other process will wake it up, whereas a quit process is more often than not "doomed", on its way to destruction, and has been made to put itself in the quit state by another process (e.g. the ~~universal~~ non-master overseer process).

In quitting a process, we distinguish between two states of execution for that process:

1. The process is executing in behalf of the user.
2. The process is executing in behalf of the system.

A process must be quit in a way such as to insure that no damage will be inflicted on the system as a result of that quitting.

Discussion

When a process is to be quit, it is in either one of the two above-mentioned states of execution. Without, at this point, going into a precise definition of those states, the quitting policy is as follows:

1. A process that is executing in behalf of the user is interrupted and made to call pxss$i_quit, which puts it in a "quit" state and gives the processor away.

2. A process that is executing in behalf of the system is allowed to "run itself out" of that state; it is forced to quit itself as soon as it executes in behalf of the user.

By convention, a process is said to be executing in behalf of the system whenever it executes in the hardcore ring.

A process might be "unquittable" outside of the hardcore ring; the overseer process sees to it that a process is not quit while it manipulates shared data bases in the administrative ring.

Implementation

When process A wants to quit process B it calls

hcs_$quit_proc(B)

which 1. Sets the target process' wakeup switch in Process B's event channel table to "off", thus inhibiting IPC wakeups, and

2. calls pxss$quit(B) which sets a quit interrupt for process B.

Process B gets interrupted (provided that it executes outside of ring 0), and the interrupt handler calls pxss$i_quit to put the process in a quit state and give the processor away.

This strategy insures that a quit process has always a standard ring 0 stack history.

A quit process can be unquit by calling

hcs_$restart_proc(B)

which sets that process' wakeup switch to "on" and calls for it pxss$unquit(B) to put it in a "ready" state.