Published: 12/06/68

Identification

The Hardcore IPC
Michael J. Spier

Purpose

The Hardcore IPC is a collection of 4 procedures in the hardcore ring, all of which are accessible from outer rings (in fact none of them is ever invoked from within the hardcore ring). Only one of them, hcs_$wakeup is intended to be invoked directly by the user procedure. The remaining three are normally invoked by the user ring ipc only (see BJ.10.03). However, all may be called from non-hardcore rings without causing damage (except perhaps to the process itself). The procedures are named wakeup, block, read_events and ipc_init.

Wakeup

Entry point hcs_$wakeup is the user's interface with the Traffic Controller's entry point wakeup. To send a wakeup to some process,

```
call hcs_$wakeup(processid, chname, message, code);

dcl processid bit (36), (ev_chn, message)
    fixed bin (71), code fixed;
```

where <u>processid</u> is the ID of the target process (possibly one's own process), <u>chname</u> is the name of one of the target process' event channels, <u>message</u> is a 72-bit string associated with this particular wakeup (the value of which is specified by the caller), <u>code</u> is a returned error status which can assume one of the following values:

0	no error
1	signalling correctly done, but target process was found to be in the "stopped" state.
2	erroneous call arguments (illegal process ID or event channel name). Call aborted.
3	target process not found (wrong process ID, or process has been destroyed). Call aborted.

hcs_$wakeup checks for error condition 2, then if arguments are valid it calls the Traffic Controller's entry point pxss$wakeup to which it passes its first three arguments.

If the Traffic Controller fails to find the process it returns with error condition 3, otherwise it allocates an entry in the wired-down interprocess Transmission Table (ITT) where it stores the following items:

1. sending process' ID, extracted from PDS.
2. receiving (target) process' ID
3. a flag to indicate that this is a user-event message (to distinguish it from a device-signal message)
4. the caller's ring number, extracted from PDS.
5. the event channel name.
6. the event message.

it appends this ITT message to an event queue belonging to the target process, then sends a wakeup to that process, and returns. If the target process was found in the "stopped" state, pxss$wakeup returns error code 1.

<u>block</u>

Entry point hcs_$block is the user's interface with the Traffic Controller's entry point pxss$block. It has no arguments and is normally invoked by the wait coordinator only (see BJ.10.03). When called, hcs_$block does the following:

1. call pxss$block. This suspends the process' execution until some event signal is received by it (some other process sends it a wakeup). pxss$block returns to its caller the head of the process' ITT event queue. This event queue consists of event messages directed towards specific event channels. The Hardcore IPC knows nothing about event channels per-se, but it knows that every event message must be copied into the ECT in which the addressed event channel resides.

The IPC maintains in segment <process_info> an array of 63 pointer variables corresponding to (potential) rings 1->63. Each one of these pointers is either null or it points to the base of the ECT of the corresponding ring.

The number of the ring in which its ECT resides is part of an event channel name (see BJ.10.03). Our procedure now scans the returned even-queue message by message, finding the event channel name and extracting its ring number. With this ring number it looks up the pointer array. If the pointer is null, then this event message is discarded, else the pointer is assumed to point to the base of a valid ECT and the message is copied into that ECT'S ITT transcription area (see BJ.10.02). When the whole event queue has been processed, a call is made to pxss$free_itt which puts those entries on the ITT's empty list.

hcs_$block assumes that it was called by the wait coordinator in some ring (say, ring n). It knows that the wait coordinator will not return to its caller unless some event of interest has been received by its (or an outer) ring. Consequently, to avoid meedless thrashing between the wait coordinator and the hardcore_ipc, block returns to its caller only if at least one of the ITT messages was directed towards one of the rings n->63. If all received event messages were directed towards rings m, where m<n, it simply loops back and calls pxss$block again.

<u>Read events</u>

Entry point hcs_$read_events is invoked by the wait coordinator entry ipc$read_ev_chn. It is similiar to entry point block except for the two following differences,

1. Instead of calling pxss$block it calls pxss$get_event which returns the current (possibly null) event queue from the ITT but never blocks the process, regardless of whether or not wakeups have actually occurred.

2. It always returns to its caller, regardless of whether or not such messages were directed to the caller's ring.

<u>Ipc init</u>

Entry point hcs_$ipc_init is invoked by ipc$init in order to register in the pointer array of <process_info> a pointer to a new ECT.

call hcs_$ipc_init(ect_pointer)

validity checks ect_pointer, then puts it into the pointer array at the slot pointed to by sb|3 (caller's validation ring number).