Published: 07/03/68

Identification

shutdown and wired_shutdown
M. Turnquist

Purpose

When system operation is discontinued, core must be cleaned. The shutdown primitive will force all segments in core (excepting only the wired_down hardcore segments) onto either the drum or disks.

(When bringing the system up again, system initialization reloads hardcore segments whether they are already there or not. If they are already present they are written over. Hence we need not concern ourselves with clearing them during shutdown.)

Wired_shutdown is a utility routine for shutdown which does the last bit of cleaning up.

Usage

As ring 0 primitives, available to the initializer process only, status active.

call hcs_$shutdown

call wired_shutdown

Implementation of shutdown:

1. Destroy all processes other than the initializer, the idle process, the loader daemon and the hardcore processes. This includes all user processes excepting the initializer, all system processes and all daemons excepting only the loader daemon.

   Set tc_data$system_shutdown to 1 in order that no more apt (active process table) entries can be added.

   Pick up the apt (active process table) pointer from tc_data. Cycle through the apt, testing the class of each entry. If the class of the entry is system, user, or daemon and if the process_id does not equal that of one of the above exceptions, then destroy the process by means of destroy_proc.

   ```
   call destroy_proc (process_id)
   ```

2. Destroy the loader daemon. (The primitive destroy_proc uses the loader daemon process in order to operate. Hence the loader daemon cannot be destroyed earlier.)

   Unload the process.

   call unload_proc (pstep);

   where pstep is a pointer to the PST (process segment table) entry.

   Delete any branches inferior to the process directory of the loader daemon.

   call del_dir_tree (ldpath, errflg);

   where ldpath is the full path name of the process directory of the loader daemon and errflg is an error indicator.

   Delete the process directory of the loader daemon.

   call delentry (path, name, 0, code);

   where path is process_dir_dir,
   name is the entry name of the process directory of the loader daemon, the courtesy switch is off, and code is an error return.

   The loader daemon process is still in the apt (active process table). However shutdown is the only process now in operation, and it will not reference the apt again. Hence the apt need not be cleaned up further.

3. Delete the ast (active segment table) entries for all non hardcore segments. Cycle through the ast hash table. Test each entry to see if deleting conditions hold. (i.e., if the entry hold count, wired down segment count, and page table hold count are off - and in addition, the inferior count=0 - then delete. These conditions are necessary for delastentry to work.)

   If the entry is not to be deleted at this time continued cycling through the ast hash table. To delete, first lock the ast entry. Then pick up the relative pointer to the entry's parent (i.e., the directory to which the entry belongs) and delete the given entry.

call getastentry$delastentry (astep, code);

code is an errcode.

(Note: delastentry expects the entry to be given to it locked.)

Look at the parent of the deleted entry to see if deleting conditions hold. If so, delete, using the method described above. If not, go back to cycling through the hash table.

4. Grow the current length of the stack big enough so that shutdown can complete it's operation without adding pages to the stack. (Shutdown operates on the ring 0 stack, which is paged.)

5. Insure all pages of active segments are assigned addresses in secondary storage.

Before any more ast entries are actually destroyed, some housekeeping must be done if nothing is to be lost by shutting the system down.

Look at the file map for each active segment to see whether the drum address for any of its pages is null. If so, assign a drum address to that page.

call pc$assign_addresses (astep);

where astep is the ast pointer for each active segment.

call the drum dim to stop any further address assignment.

call device_control$shutdown;

6. Update the active information in the branch for each ast entry. To do this the parent directory for each entry must be known to our process. Cycle through the ast. For any given entry a sdw (segment descriptor word) must be put into the descriptor segment of our process.

In the descriptor segment of every process there exists a dummy word. This word may be changed at will, thus adding a sdw for whatever segment we choose.

Call master_mode_ut$swap_sdw (dp, bitsdw, savsdw);

where dp is a pointer to the place in the descriptor segment to be changed.

bitsdw is the sdw for the entry to be known

savsdw is the word being replaced by bitsdw

The parent directory of the given entry is now known to our process, hence the active information in the branch itself can be updated.

call activinfo$wrbranch (dp, slot, id, actsw, itemsptr, dfmp, code);

where

dp is the pointer through the sdw of the parent directory of the given entry.

slot is the slot number of the given entry in its parent directory (picked up from the ast entry).

id is the unique identifier of the branch (picked up from the ast entry).

actsw is the active switch which we set off.

itemsptr is a pointer to a structure made up of the active information (picked up from the ast entry).

dfmp is a pointer to a copy of the entry's file map.

code is an error code, returned.

7. Finish the cleaning up process with a call to wired_shutdown.

call wired_shutdown

Implementation of wired_shutdown

1. Finish cleaning up the ast. Call pc$clean_up for each remaining entry that is not wired down. (Thus we avoid cleaning up our own process.)

2. Put free_storage on the drum by calling free_store$shutdown.

3. call master_mode_ut$bos in order to get back to the BOS subsystem.