Published: 07/20/67
(Supersedes: BJ.2.01, 11/25/66)

Identification

The Active Process Table
J. H. Saltzer, R. L. Rappaport, A. Evans

Purpose

The Active Process Table is a system-wide data base, accessible to and maintained by the modules of the Traffic Controller. It contains an entry for every process in the system which is active. An active process is one which meets one or more of the following conditions:

1. It is running.
2. It is ready.
3. It is responsible for a system interrupt.
4. It is waiting for a page to come in.
5. It is being loaded.
6. It is blocked but has not yet been completely paged out of core.

The Active Process Table must remain in core memory at all times, since it is the primary source of information for scheduling during system interrupts. A characteristic of an active process is that enough information is stored in the Active Process Table that the process can be scheduled and switched to running status without getting any page faults.

An inactive process must be placed in the Active Process Table ("activated") before it is possible to send it a wakeup signal. Activating a process involves creating an APT entry for it and taking other actions relevant to the file system. See BJ.2.00.

A process is deactivated when the basic file system, operating for some other process and looking for core space, notices that the descriptor segment has fallen into disuse. If the process in question does not meet one of the first five conditions above, its descriptor segment is discarded and its Known Segment Table is paged out and deactivated (see section BG.3). The process is then removed from the Active Process Table. See BJ.2.00 for a description of deactivation.

The APT is in segment tc_data. Also in that segment is a hash table which facilitates finding the index of a process in the APT given its process identification. The Hash Table is documented in BJ.7.02. Also in the same segment is the Traffic Control Data Base, documented in BJ.1.08. Section BJ.1.00 contains a discussion of the strategies used in selecting a home for specific data items.

The ready list, a thread through the APT, is shown here and is documented more fully in BJ.1.02. Multics will use initially a simple round-robin scheduler which will be replaced later by a more sophisticated multi-queue scheduler. The second scheduler will require that the ready list be modified. It is the intent that this document and BJ.1.02 describe the eventual ready list rather than the initial one and that the initial ready list be described in BJ.4.05. However, at the present time the requirements of the second scheduler are not yet known, so the declaration shown at the end of this section is that needed by the initial scheduler.

Discussion

An item associated with a given process may be stored either in the APT entry or in the Process Data Block (PDB) of the process. This latter, described in Section BJ.1.04, is a wired down data base contained in the same segment as the Process Concealed Stack. Since the PDB is in the address space of one process only, any item which must be available to other processes must be in the APT rather than the PDB. For convenience in accessing, some items are in both. (The process identification is one such.) One of the items in the PDB is pds$apt_pointer, a pointer variable which points to the process' APT entry. It is through use of this datum that the process is able to reference rapidly its own APT entry. Another item is pds$apt_index, the index in the APT of the process' entry. Both the index and the pointer are useful. See BJ.1.04 for further discussion.

Form of the APT

The APT is in segment <tc_data> at location [apt]. The complete declaration of the APT is given at the end of this section, but it should be noted that the declaration is of the form

```
dcl 1 tc_data$apt external,

      2
     ... header information
      2

      2 entry(200),

        3
       ... declaration of an entry
        3
```

Most procedures which reference the APT need refer only to that entry which "belongs to" the process in which the procedure is executing, and such procedures should include (in their code) the declaration

```
dcl 1 apt_entry based (pds$apt_pointer),

      3
     ...  declaration of an entry
      3
```

Here the text shown at level 3 is precisely the same text shown above in the declaration of the entire APT. (Use of the PL/I "include" macro is indicated here.) Only procedures which reference the APT entry of processes other than their own need include the entire APT declaration.

In order for the scheme just described to work, it is necessary that the data for each APT entry begin at a word boundary. Since the present declaration is such that all data in the substructure is aligned (because of the presence of the "fixed" items), this condition is met. If the declaration is ever to be changed so that the entry substructure is packed, it will be necessary that care be taken that the data origin of each entry continues to fall on a word boundary. There is an additional requirement that both the header information and the entry contain an even number of words, since a clever compiler may optimize by assuming that the based structure apt_entry was created by allocate and thus is aligned to an even word. Again, it should be noted that the present declaration meets this requirement. For further details on storage of structures, see BN.6.02 and BN.9.01.

Contents of the APT

Each of the items in the APT will now be described. The order of appearance of the items is the same as their order in the declaration. By convention, an item referred to as a "switch" is a bit string of length one which is

"on" when it is non-zero. An "interlock" is like a process identification and is said to be "set" when it is non-zero. When set, an interlock will contain the process identification of the process doing the setting.

The APT contains, in its header, one each of the following items:

1. Ready list interlock. This interlock is set whenever the ready list is in use.

2. Ready list head. This item contains the index in the APT of the process at the head of the ready list; i.e., the next process to run.

3. Ready list tail. This item contains the index in the APT of the process at the end of the ready list.

4. Empty list interlock. This interlock is set to indicate that the empty list is in use.

5. Empty list head. This item contains the index in the APT of an empty entry. All empty entries are linked together by a thread.

6. Empty list tail. This item contains the index in the APT of the last empty entry.

7. Empty list count. This item contains the number of empty entries on the APT.

8. Dummy. This item appears so that there will be an even number of words in the header, as explained earlier.

The APT contains an instance of each of the following items for each active process.

1. Process identification. This is a bit string used to identify uniquely the process within Multics. Its format is described in BJ.7.03.

2. Wakeup-lock. If this interlock is set, some process is in the midst of trying to wakeup this process; so another process desiring to wakeup this process need take no further action, and getwork will not choose this process for execution.

3. The process-state lock. Processes referring to or changing the settings of any of the next four items must set this lock prior to accessing these data items. Other processes wishing to access these items must wait for the process-state lock to be reset before going ahead.

4. Running switch. This switch is set on whenever the process is running.

5. Ready switch. This switch is set on whenever the process is ready. If neither the running nor ready switches are on, the process is blocked. (They will never both be on.)

6. Wakeup waiting switch. This switch is on when a wakeup is waiting for this process which the process has yet to become aware of.

7. Quit-waiting switch. This switch, if on, indicates another process is trying to "quit" this process.

8. The intermediate-state switch. This switch, if on, indicates the process may be "executing" on a processor regardless of the process state as defined in its ready and running switches.

9. The process-chosen switch. This switch, if on, indicates that this process has been chosen to run by another process operating in getwork, but that it has not yet begun to run. Getwork in choosing subsequent processes will pass over this process.

10. Waiting for page switch. If this switch is on, this process is waiting for a page to come in from secondary storage and cannot be deactivated.

11. Waiting for hardware switch. If this switch is on, this process is responsible for some system interrupt and cannot be deactivated.

12. Process not loaded switch. If this switch is on, the descriptor segment for the process has been paged out of core memory, and special action is needed to switch control to the process. See BJ.5.00.

13. Process loading switch. If this switch is on, this process is attempting to retrieve its process data block, and must not be unloaded. See BJ.5.03.

14. Process Segment Table pointer. For a process which is unloaded but still active, this pointer represents the end of a (long) string which, if pulled hard enough, will retrieve enough of the process to allow it to execute. This quantity is returned by actproc when the process is activated, and it is used in the process bootstrap module to load the process. See BJ.2.00 and BG.3.03.

15. Block lock count. This item records the number of hard-core ring "block type" locks that this process currently has set. When the count is non-zero, the process may not be quit and is to be given special priority by the scheduler.

16. Ready list pointers. The ready list is a thread through the active process table. If this process is ready (switch 5 above), these pointers link the ready list forward and backward. If this entry in the Active Process Table is empty (17, below), the pointers link the empty entry list.

17. Empty entry switch. If on, this APT entry is empty, and the pointers of the previous item link the APT empty list.

18. Process time limit, scheduler imposed. This is the maximum "time" that the process should be allowed to run in a single shot without the scheduler's reconsidering its priority. This limit will generally be re-evaluated each time that the process schedules itself. It is this quantity that is loaded into the hardware "interval" timer when the process starts execution.

19. Time last run. This item contains the calendar time at which this process last left running status. This information is used to help select candidates for unloading and deactivation.

20. Processor currently in use. This item only has meaning if the Running Switch (4 above) is on. It contains the index of the processor being used by this process. This information is needed, for example, to interrupt the process.

21. Processor number required. Certain processes have sufficient authority to demand that they only run on a specific processor. Such a requirement is indicated by this entry being non-zero. This service is used for system initialization, for test and diagnostic, and for the "idle" processes.

22. Descriptor segment base register value. This contains an absolute core address, the base of the hardcore ring descriptor segment for this process. If the process not loaded switch (item 12 above), is on, this entry is meaningless. Swap-DBR loads this quantity into the register DSBR to establish the address space for the process.

23. Active meter table index. This item identifies the metering module meter to be charged for processor usage by this process.

24. Dummy. This item appears so that there will be an even number of words in each entry, as explained earlier.

```
dcl 1 tc_data$apt external,              /* The Active Process Table */

      2 ready_list_lock bit(36),         /* interlock ready list */
      2 ready_list_head fixed,           /* first item in ready list */
      2 ready_list_tail fixed,           /* last item in ready list */

      2 empty_list_lock bit(36),         /* interlock empty list */
      2 empty_list_head fixed,           /* head of the empty list */
      2 empty_list_tail fixed,           /* tail of empty list */
      2 empty_list_count fixed,          /* count entries in empty list */

      2 dummy fixed,                     /* even number of words in header */

      2 entry (200),                     /* entries for active processes */

        3 process_id bit(36),            /* process identification */


        3 wakeup_lock bit(36),           /* someone is trying to awake */

        3 state_lock bit(36),            /* interlock next four switches */
        3 running_sw bit(1),             /* on if running */
        3 ready_sw bit(1),               /* on if ready */
        3 wakeup_waiting_sw bit(1),      /* a wakeup has yet to be seen */
        3 quit_waiting_sw bit(1),        /* a quit has yet to be seen */

        3 intermediate_state_sw bit(1),  /* on if state being changed */
        3 chosen_sw bit(1),              /* process has been chosen to run */


        3 waiting_page_sw bit(1),        /* process awaits page */
        3 waiting_hardware_sw bit(1),    /* process awaits interrupt */
        3 not_loaded_sw bit(1),          /* on if process not loaded */
        3 loading_sw bit(1),             /* on if now trying to load */
        3 pst_pointer bit(18),           /* to fetch unloaded process */

        3 block_lock_count fixed,        /* count block-locks set */


        3 ready_list_frwd fixed,         /* next process in ready list */
        3 ready_list_bkwd fixed,         /* previous process in ready list */
        3 entry_empty_sw bit(1),         /* if on, above link empty entries */


        3 time_limit bit(24),            /* max execution time */
        3 time_last_run bit(72),

        3 processor_in_use fixed,
        3 processor_required fixed,

        3 dsbr_value bit(29),            /* loc of ring 0 desc seg */

        3 amt_index bit(18),             /* active meter table index */

        3 dummy fixed;                   /* even number of words per entry */
```

```
dcl 1 apt_entry based(pds$apt_pointer), /* an entry in the APT */
      3 process_id bit(36),             /* process identification */

      3 wakeup_lock bit(36),            /* someone is trying to awake */

      3 state_lock bit(36),             /* interlock next four switches */
      3 running_sw bit(1),              /* on if running */
      3 ready_sw bit(1),                /* on if ready */
      3 wakeup_waiting_sw bit(1),       /* a wakeup has yet to be seen */
      3 quit_waiting_sw bit(1),         /* a quit has yet to be seen */

      3 intermediate_state_sw bit(1), /* on if state being changed */
      3 chosen_sw bit(1),             /* process has been chosen to run */

      3 waiting_page_sw bit(1),       /* process awaits page */
      3 waiting_hardware_sw bit(1),   /* process awaits interrupt */
      3 not_loaded_sw bit(1),         /* on if process not loaded */
      3 loading_sw bit(1),            /* on if now trying to load */
      3 pst_pointer bit(18),          /* to fetch unloaded process */

      3 block_lock_count fixed,       /* count block-locks set */

      3 ready_list_frwd fixed,        /* next process in ready list */
      3 ready_list_bkwd fixed,        /* previous process in ready list */
      3 entry_empty_sw bit(1),        /* if on, above link empty entries */

      3 time_limit bit(24),           /* max execution time */
      3 time_last_run bit(72),

      3 processor_in_use fixed,
      3 processor_required fixed,

      3 dsbr_value bit(29),           /* loc of ring 0 desc seg */

      3 amt_index bit(18),            /* active meter table index */

      3 dummy fixed;                  /* even number of words per entry */
```