bcc	title: RUNNING A PROCESS		prefix/class-number.revision: PRUN/W-8	
checked: [signature]	authors: Butler W. Lampson [signature]		approval date: 6/20/69	revision date
checked			classification: Working Paper	
approved: [signature]			distribution: Company Private	pages: 13

ABSTRACT and CONTENTS

Describes, first in outline, then in detail, the mechanisms involved in running a process. All the interfaces between the various modules are specified.

TABLE OF CONTENTS

Introduction

This document describes the logical structure of all the mechanisms in the system which are concerned with the running of a process. It also specifies all the interfaces involved. The system modules which take part are:

- the scheduler, a routine in the monitor which runs whenever a process blocks or the interval timer runs out
- the swapper, a microprogram in the AMC which is responsible for transferring processes between core and secondary memory
- the microscheduler, a microprogram which handles wakeups and decides which processes should be given processors at each instant
- the block and time-out routines in the CPU
- the CPU microprogram which is responsible for saving the state of the current process and loading the state of the new process

Process States

The position of a process from the point of view of the scheduling and swapping operations is recorded in a collection of bits in the PRT. Certain functions of these bits determine states which are of concern to us. These are

active	the process has a PRT entry
ready	the process is not awaiting a wakeup. This is NOT BLK
blocked	a process which is not ready is blocked.

loaded	more or less, the working set of the process is in core. This is not a precise definition, since the process may have modified the working set. The meaning of this state should be clarified by the description of how states are changed. Loaded is LDD
swapping-in	the swapper is bringing the process in and will pass it on to the microscheduler when it has come in SWQ OR PQ OR PDK OR CBC
μready	the process is loaded and ready and on a micro-scheduler queue. It will run if a processor becomes available MSQ
running	the process has a processor RUN the number of the processor is given by CPU
scheduled	ready and not on a scheduler queue. Either the process is μready or it is swapping in MSQ OR PQ OR PDK OR CBC OR SWQ An alternative description is (NOT BLK) AND (NOT (SCQ OR WAQ))

Figure 1 shows how the various states are related.

Transitions

The operation of the entire system for running processes is determined by the states which processes can be in from the viewpoint of that system (described above) and the allowed transitions between states (described below). For each transition we give the event which causes it, the modules and calls between modules which implement it, and any other action which is taken or conditions which are relevant.

It seems desireable to start with an overview of the life history of a process, which is diagrammed in Figure 2. For a normal non-resident process receiving a series of quanta the sequence is

on scheduler queue
swapping in
on μscheduler queue
running for one quantum
swapping out

repeated for each quantum. The figure shows the variations on this theme in some detail. The following list of allowed transitions describes all the possibilities.

Blocked -► Ready — Happens because of a WAKEUP directed to the μscheduler from some other module (CPU, CHIO, disk driver, etc). Clear BLK. Then there are two cases:

1) If the process is loaded, it is put on a μscheduler queue

and becomes μready. Set MSQ.

2) Otherwise, it is put on the wakeup queue. Set WAQ. The next time the scheduler runs, it will be removed from the wakeup queue and put onto a scheduler queue. At this time clear WAQ and set SCQ.

Ready -► Scheduled and swapping-in — Happens because the scheduler decides (using algorithms described elsewhere) that the process should run. The scheduler makes a SWAPIN call on the swapper to bring the process's working set into core. Clear SCQ and set SWQ. When the swapper starts to load the process, clear SWQ and set CBC. When the context block comes in successfully, clear CBC and set PQ.

Swapping-in -► μReady — Happens because the swapper completes the reading of the CWS for the process. It clears PQ and sets LDD. Then it sends a WAKE-UP to the μscheduler. This time case (1) will hold.

µReady - Running	Happens because the µscheduler decides (using algorithms described elsewhere) that the process should run on CPU i. It removes the process from its queue, puts the absolute address of the PRT entry for the process into a cell called CPUi and sends a SWITCH call to CPU i. It also clears MSQ and sets RUN and CPU in the PRT. The CPU does the switch as soon as it finds itself out of monitor mode.
Running - µReady	This always happens as a counterpart to the previous transition. The µscheduler tells a CPU to switch away from the process. The CPU sends the µscheduler a RETURN for the process when it completes the switch. When the µscheduler processes a RETURN it clears RUN.
Running - Blocked	This results from the monitor's decision to block. To do so, the CPU stores its state and sends a BLOCK call to the µscheduler. It then waits for a SWITCH call,

	upon which it loads a new state from the context block found in CPUi. The µscheduler clears RUN and sets BLK.
Running - Blocked and Unloaded	This is the same as the previous transition, except that the monitor has also decided that the process should be thrown out. It does a BLOCKOUT call on the µscheduler, which proceeds as before. However, the µscheduler also sends a SWAPOUT call to the swapper, puts the process on the request list and clears LDD. When the swapper processes the request it puts the pages of the process on the write list.
Running - Ready and Unloaded	This is the same as the previous transition except that the process is not blocked. It normally happens because of a timer trap. The monitor does an UNLOAD call on the µscheduler, which proceeds as before except that it does not set BLK and it also puts the process on the wakeup queue for the scheduler and sets WAQ.

Running - Swapping-in: This happens when a page-fault occurs and the monitor decides that the process should not be thrown out. It does a PAGEWAIT call on the µscheduler, which clears LDD and sets PQ. The CPU behaves as on a BLOCK.

The following is a list of the modules which can set or clear and which need to test each bit in PRT mentioned so far in this document.

Bit	Set	Clear	Test
SCQ	CPU (scheduler)	CPU (scheduler)	
SWQ	CPU (scheduler), µscheduler	swapper	
MSQ	µscheduler	µscheduler	
WAQ	µscheduler	scheduler	
BLK	µscheduler	µscheduler	
RUN, CPU	µscheduler	µscheduler	
CBC, PQ, PDK	swapper	swapper	swapper (to suppress unneeded reads)
LDD	swapper	µscheduler	µscheduler

Needless to say, setting of PRT bits must be done under a protect.

Calls between modules

In this section all the calls required for the various modules which implement the IWS are described. With each one is a detailed description or a reference to another document where such a description can be found.

SWAPIN: CPU (scheduler) or µscheduler to swapper

This call requests the swapper to bring in a process. To make it, the CPU obtains a swapper request node and puts the request into the node. It then chains the node onto the swapper request queue and sets SWQ. The swapper interrogates the queue periodically. Details are to be found in MMI/W-1.

SWAPOUT: µscheduler to swapper

This call requests the swapper to write out a process. It is made very much like a SWAPIN. Again, details are to be found in MMI/W-1. SWQ is not set.

GIVEUP: swapper to µscheduler

This parameterless call is made by the swapper when it wants a process to write out. If the microscheduler can find a suitable one on a low priority queue, it will return it to the swapper with a SWAPOUT call, clear LDD, set WAQ and put the process on the wakeup queue. This operation will not be implemented initially.

All calls on the µscheduler are done through an input buffer (USIB) which is a stack in core. All requests to it are put into two-word entries in this buffer, and each is accompanied by an attention signal directed to the µscheduler. The strategy of the μscheduler is very simple: whenever the attention signal is received, reset it and empty the buffer.

The NSIB is $<2^n$ words long, starts at USIBASE, and ends at a word (USIEND) whose address is Ø mod 2^n. Associated with it is a pointer (USIBTOP) to the top.

Signalling to the μscheduler is done under a protect and proceeds as follows:

Protect

Fetch USIBTOP to TOP

TOP ← TOP +2

If TOP ≡ Ø mod 2^n the buffer is full. Unprotect and start over

Store the message in the double word addressed by TOP

Store TOP in USIBTOP

Unprotect

Send ATTN to μscheduler

The μscheduler proceeds as follows to read the buffer:

Protect

Fetch USIBTOP to TOP

If TOP = USIBASE, the buffer is empty. Unprotect and wait for the attention signal to reappear

Fetch the message from the double word addressed by TOP

Store TOP-2 in USIBTOP

The buffer is initialized by setting USIBTOP to USIBASE.

The format of an entry in USIB is as follows:

Word	Bits		Contents
Ø	0-5	OP	Identifies the call
Ø	0-23	PRID	Absolute address of PRT entry for process involved
1	0-23	DATA	Data for call

WAKEUP, IWAKEUP all μprocessors to μscheduler

This call is made by any μprocessor which wants to wakeup a process. The data word specifies the bits of PIW to be set. The μscheduler, when it processes the call, turns off BLK. If LDD is set it then puts the process on its queues at the priority given by PRI and sets MSQ. Otherwise it puts the process on the wakeup queue and sets WAQ. IWAKEUP is identical except that it interprets PRID as the <u>index</u> of a PRT entry.

SWITCH μscheduler to CPU

Each CPU has a core cell called CPUi (i = Ø or 1) which is set to the PRT index of the process which the CPU is supposed to run next. Each CPU also has an activity level (AL) maintained by the μscheduler which can take on one of these values:

I — Idle, if the CPU is not running anything, i.e. the μscheduler has given it the same number of processes via SWITCH as it has given back via BLOCK or RETURN.

R — Running, if the CPU has been given one more process than it has given back. Presumably it is running this process.

P Primed, if the CPU has been given two more processes than it has given back. It enters this state when the μscheduler decides to preempt it, and leaves it when it gives back the preempted process (more or less). This state is therefore considered to be transitory, and the μscheduler is willing to wait for the CPU to leave it.

Finally, each CPU has a priority (PRI) maintained by the μscheduler, which is the priority of the 'running' process. Running in this context means the process on whose behalf the most recent SWITCH call was made.

When the μscheduler is ready to send a SWITCH to CPU i it checks ALi. If ALi is P, it goes into a mode in which it processes calls as usual but does not initiate any switches until ALi drops below P. When ALi is not P, it increases AL by one level, stores the PRT index of the process in CPUi, sets PRIi to the priority of the process and sends an ATTN to CPU i.

The CPU can be in one of three states from the point of view of process switching

- idle - it is running no process
- locked - it is running a process which has the CPU locked, i.e. is in monitor mode
- unlocked - it is running a process but is not locked.

In locked state it ignores an ATTN signal, which is latched and therefore waits. In idle state it clears ATTN, fetches

CPUi, clears it, loads the state of the specified process and starts executing it. In unlocked state it dumps the state of the current process, sends a RETURN call for it to the μscheduler, and goes to idle state.

When the μscheduler gets a BLOCK, BLOCKOUT, UNLOAD, PAGEWAIT or RETURN call from a CPU it it reduces ALi by 1.

As part of storing the state it puts the interval timer, shifted so that the least significant bit counts milliseconds, into the MCT field of the process' PRT entry.

BLOCK, BLOCKOUT CPU to μscheduler

The data word contains the CPU number. This call informs the μscheduler that the CPU is blocking the specified process. The μscheduler clears RUN and turns on BLK for the process and reduces AL for the CPU. In the case of BLOCKOUT it also makes a SWAPOUT call on the swapper for the process and clears LDD.

UNLOAD CPU to μscheduler

The data word contains the CPU number. This call informs the μscheduler that the CPU wants the specified process unloaded and passed to the scheduler. The μscheduler clears RUN and reduces AL for the CPU. It puts the process on the wakeup queue, sets WAQ, and makes a SWAPOUT call on the swapper and clears LDD.

RETURN CPU to µscheduler

The data word contains the CPU number. This call informs the µscheduler that the CPU has stopped running the specified process because it was preempted. The µscheduler clears RUN and sets MSQ for the process, puts the process back on its queues with priority given by its PRI, and reduces AL for the CPU. This call can also be used by the CPU to change the priority of a process.

PAGEWAIT CPU to µscheduler

The data word contains the CPU number. The action is not yet defined.

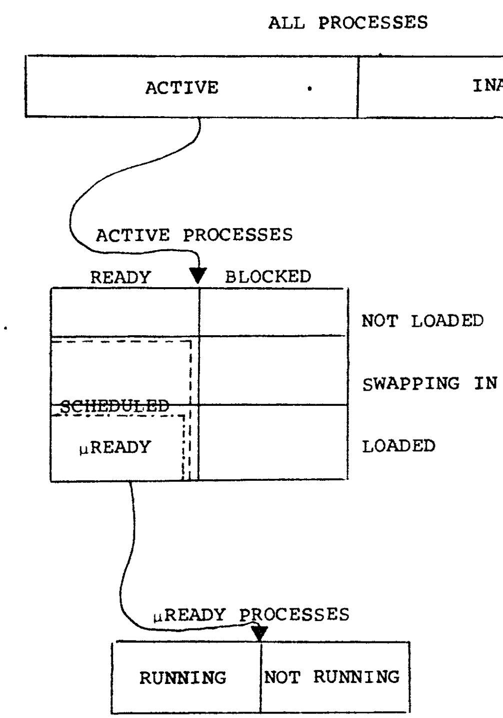

Figure 1

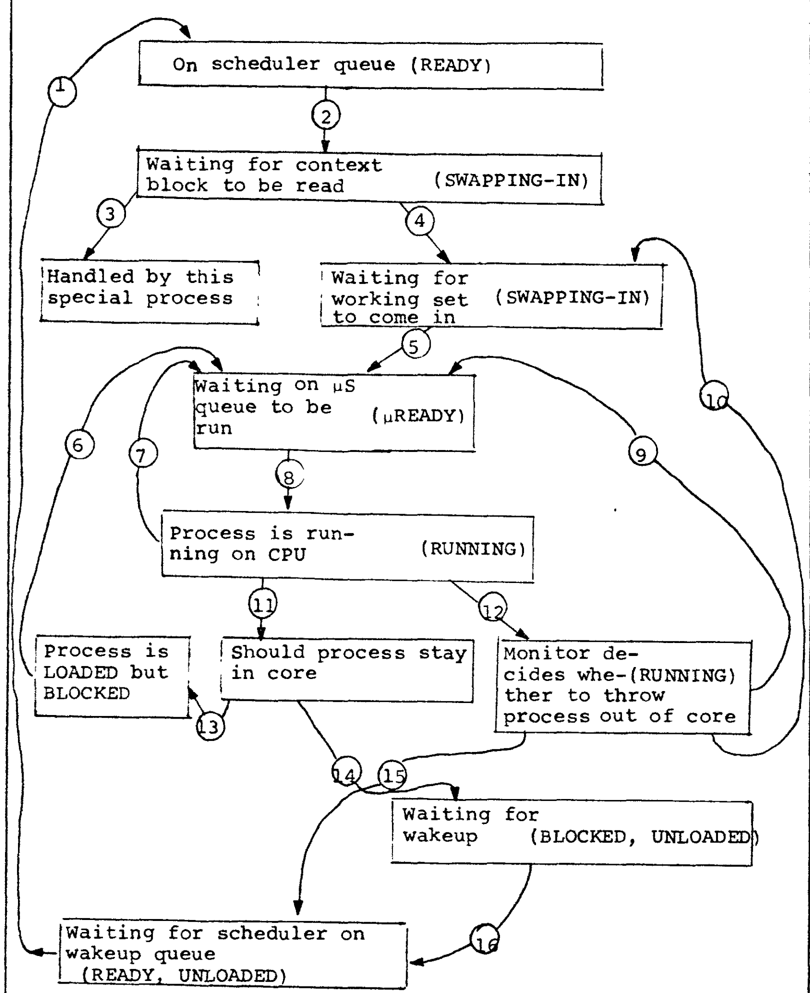

Figure 2
(Continued next page)

1 - Scheduler runs and puts it on scheduler queue

2 - Is scheduled: scheduler passes it to swapper to be read in

3 - CB read fails. Swapper passes it to special process which handles this case

4 - CB read succeeds. Swapper queues reads for working set

5 - Reads are completed. Swapper gives it to µscheduler

6 - Wakeup arrives

7 - Process is pre-empted by higher priority process or lowers its priority

8 - Process becomes highest priority. µS gives it to a CPU

9 - No, and timer ran out. Lowers priority

10 - No, and page fault. Return to swapper

11 - Process blocks

12 - Timer runs out or process page-faults

13 - Yes. It is given to µs

14 - No. Monitor gives process to µs to be blocked and to swapper to be thrown out. Process becomes blocked.

15 - Yes. Monitor gives process to µs to be blocked and to swapper to be thrown out. µS puts process on wakeup queue for scheduler.

16 - Wakeup arrives and µs puts it on wakeup queue for scheduler.

Figure 2 (end)