Published: 03/24/67

Identification

Ready-him
R. L. Rappaport

Purpose

Entry point ready-him in the Process Switching Module, provides the facility by which a process, the calling process, can give temporary control of a processor to another process, the target process, so that the target process can schedule itself.

Preface

The description of ready-him that follows is divided into two sections. The first section presents the basic outline of the subroutine. This would be an adequate description if it could be assumed that processes in the system are never unloaded. The second section presents the necessary additions to the basic outline that enable the unloading of processes to be accomplished.

Basic Outline

A process wishing to signal another process calls subroutine wakeup (see Section BJ.3.02) on behalf of this second process. The signaling is accomplished in wakeup by explicitly turning on a switch belonging to the signaled process and by insuring that the signaled process is scheduled to run in the future. If the signaled process is currently in the blocked state, this means the signaled process is not scheduled to run in the future and wakeup calls entry point ready-him in order to schedule the signaled process. Ready-him accomplishes the scheduling of this target process by first switching the processor to the address space of the target and by then calling the scheduler that exists in that address space. Upon return from this scheduler, ready-him switches the processor back to the address space of the calling process and returns to wakeup. (One should note that the above few sentences imply that ready-him exists in the respective address spaces of both the calling and target processes and in the same relative location in each address space.)

Ready-him is called with one argument, the Active Process Table index of the process due to schedule itself. The calling sequence is:

```
call ready-him (apt_index);
```

where apt_index is the index mentioned above. The stack used in this call is the Processor Stack (see Section BK.1.03) of the processor executing ready-him.

Conceptually ready-him is simple. It consists of an ldbr instruction followed by a call to the scheduler (see Section BJ.4.00) followed by a second ldbr instruction. The first ldbr switches the processor from the address space of the calling process to that of the target process. The second ldbr switches the processor back to the address space of the calling process. Two additional steps must be added to ready-him to complete the description but we first present a little background information to clarify the issues.

The stack used in ready-him is the Processor Stack. In order for both the calling process and the target process to use the same stack segment, this segment must appear in the address space of each process. That is, each process must have a segment descriptor word, for this segment, in its own descriptor segment. In order to guarantee that the target process has such a segment descriptor word in its descriptor segment, ready-him passes this word to the target process. When an ldbr instruction is executed the address space of the processor changes although the machine registers remain fixed (except of course the descriptor base register). Therefore, immediately before the ldbr instruction, the calling process loads the A-register with the segment descriptor word for the Processor Stack. It obtains the word from its own descriptor segment. Immediately after the ldbr instruction, the target process stores the A-register directly into its own descriptor segment and in this way has access to the segment. This step implies ready-him knows the segment number of the Processor Stack in each process. Actually hardware and software constraints already require that all processes must know this segment by the same segment number so therefore ready-him must only know or be able to obtain one number. The second ldbr need not be preceeded and followed by the above steps since this time we are switching to a process that is guaranteed to have a segment descriptor word for the Processor Stack.

We can now formally specify the basic outline of ready-him. The steps are tabulated below and are illustrated in figure 1.

1. Load the A-register with the segment descriptor word for the processor stack.

2. Execute the ldbr instruction to switch to the target process.

3. Store the A-register into the current (target's) descriptor segment.

4. Call scheduler.

5. Execute the ldbr instruction to return the processor to the calling process.

6. Return.

<u>Additions to Enable Unloading of Processes</u>

The loaded state for processes can be defined explicitly (see Section BJ.1.00) however a more intuitive definition is called for here. Certain modules in the harcore supervisor perform functions whose execution cannot be interrupted by page faults. For example, all modules engaged in servicing page faults would be included in this category. Basically, a loaded process is one which is capable of executing in these hardcore modules without generating a page fault.

If the target process in the call to ready-him is not loaded, ready-him must prepare the target so that it will not get any page faults from the time it receives control until it relinquishes it. Since the target process will only be executing in a certain well defined subset of the hardcore supervisor it need not be completely loaded. In fact, all that needs be done in preparation is to create a descriptor segment for the target if it is unloaded. This descriptor segment is created in two steps. First entry point createseg in Segment Control (see Section BG.3.00) is called to obtain an empty wired-down segment. Then the contents of the template descriptor segment (see Section BJ.5.06) is copied into the newly created segment. This descriptor segment will serve the target until the return. When the calling process regains control, it destroys the descriptor segment created above by an explicit call to entry point killseg in Segment Control.

The reason for destroying the descriptor segment has to do with the nature of entry point createseg. One of the arguments specified in this call is the urgency with which this segment to be created is needed. This urgency argument determines whether or not this request for wired-down core can be met with the present use of core space. That is, it determines whether there is enough free core space available to service the request. If there is not enough space available createseg performs an error return to its caller. Since ready-him may be called in response to a system interrupt, it cannot tolerate a refusal from createseg. In order to justify the high priority that it needs, ready-him guarantees to return the core space in a short space of time. Therefore the segment is destroyed as soon as it has served its purpose. In this way ready-him is allowed to call createseg with an urgency argument that createseg guarantees to service. In particular, if there are N processors with K levels of interrupt each, there will never be more than N times K segments created with this urgency at once.

Ready-him can now be completely specified. It is tabulated below and illustrated in figure 2.

1. The calling process determines whether the target is loaded. If it is go to step 4. This determination is made by testing the target's not loaded switch, a data item in the target's Active Process Table entry.

2. Createseg is called to create a descriptor segment for the target.

3. The contents of the template descriptor segment are copied into the segment created in step 2.

4. The A-register is loaded with the segment descriptor word for the Processor Stack (which is contained in the Processor Data Segment).

5. The ldbr is executed to switch to the target.

6. The A-register is stored into the current descriptor segment word appropriate for the Processor Data Segment.

7. The scheduler is called.

8. The return ldbr is executed.

9. The target's not loaded switch is again tested. If it is <u>off</u> (i. e., the process is loaded) go to step 11.

10. Entry point killseg is called to destroy the segment created in step 2.

11. Return.

<u>Wrapup</u>

One last thing should be noted at this point. The ldbr instruction may only be executed in master mode. In order to isolate this instruction from the rest of ready-him, the actual instructions are contained in a distinct master mode segment, the ldbr segment (see Section BJ.5.04). Actually, the instructions associated with steps 4 through 6 are located at entry point ldbr_2 in the ldbr segment while the instruction associated with step 8 is located at entry point ldbr_3 in the ldbr segment. These items are noted in figure 2.

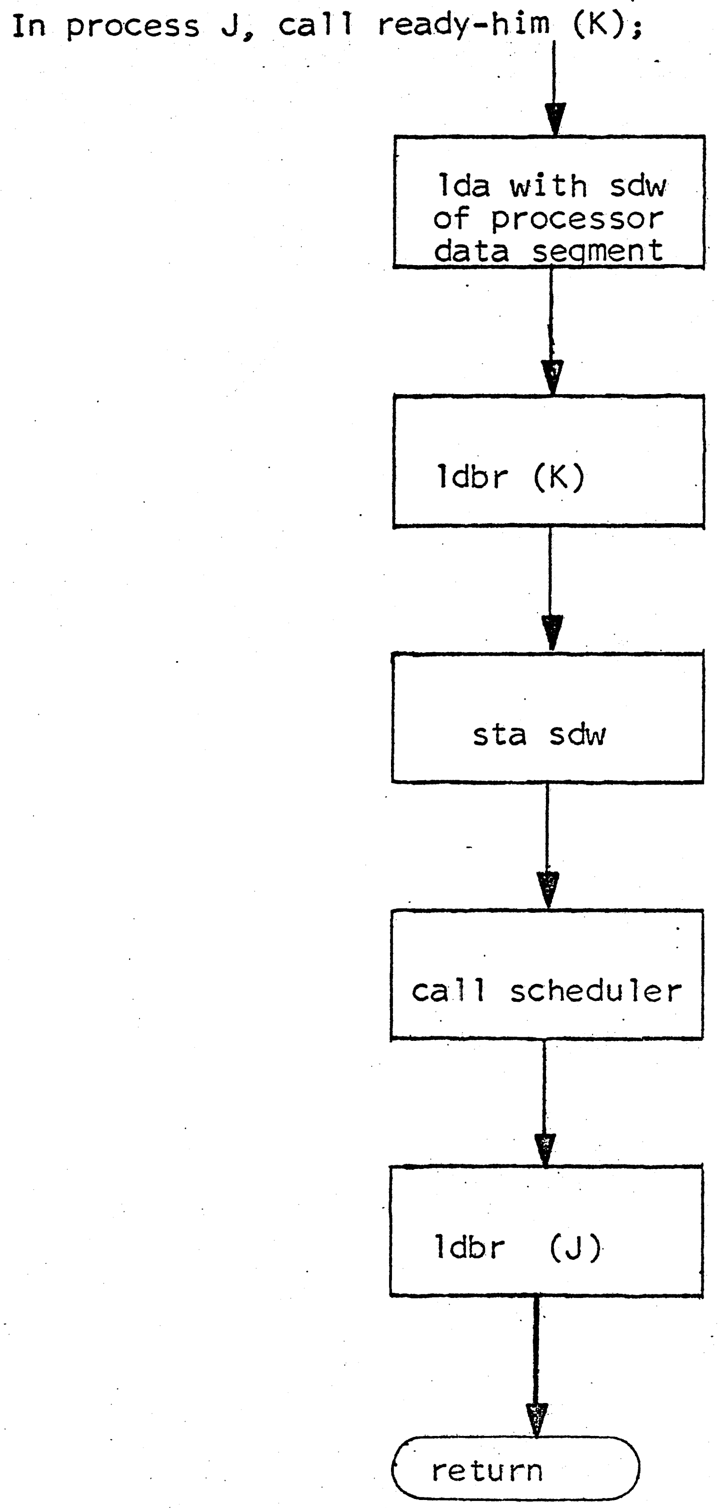
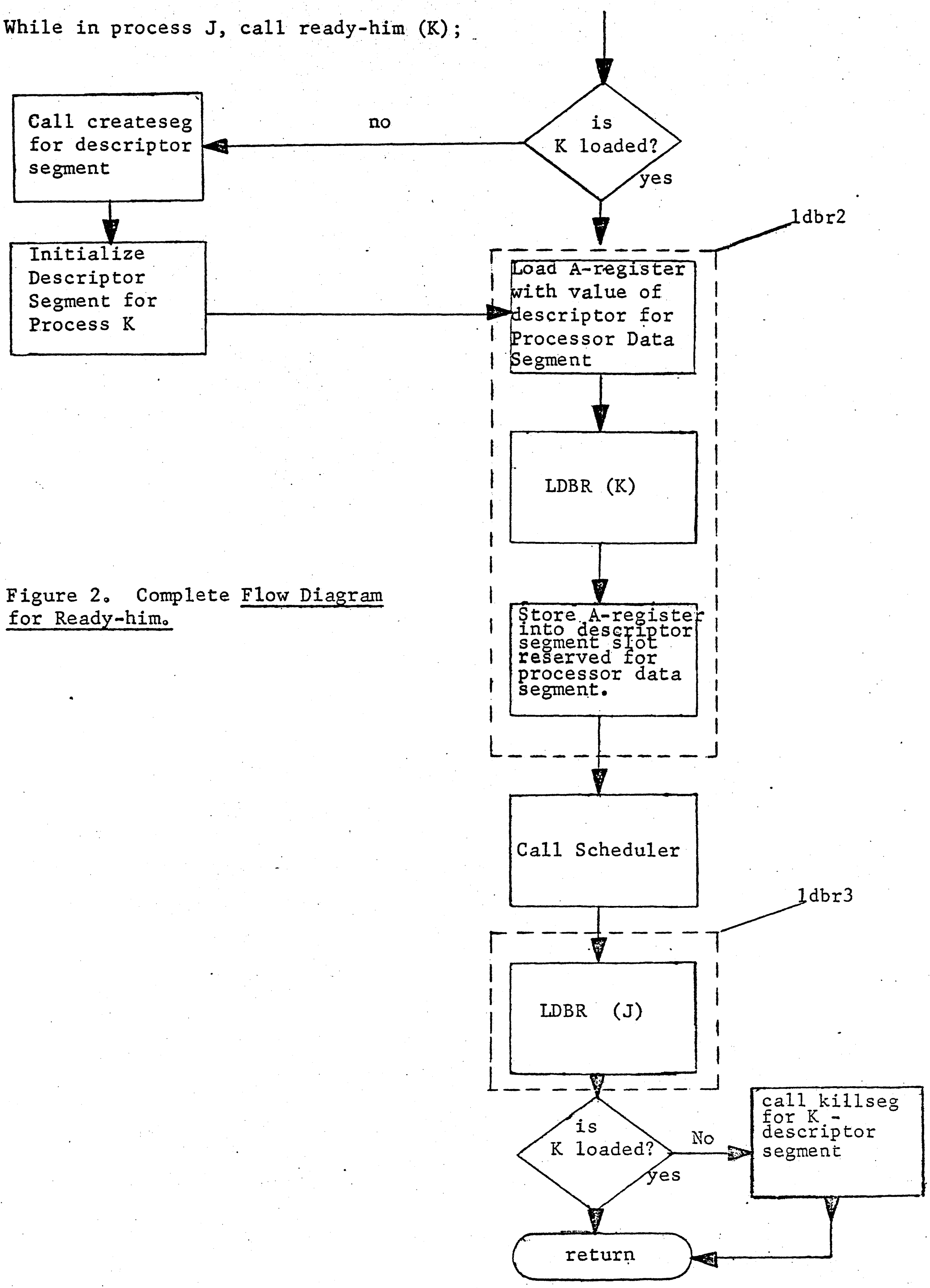

In process J, call ready-him (K);
lda with sdw of processor data segment
ldbr (K)
sta sdw
call scheduler
ldbr (J)
return

Figure 1. Basic outline of Ready-him.

While in process J, call ready-him (K);



no
is K loaded?
yes
Call createseg for descriptor segment
Initialize Descriptor Segment for Process K
ldbr2
Load A-register with value of descriptor for Processor Data Segment
LDBR (K)
Store A-register into descriptor segment slot reserved for processor data segment.
Call Scheduler
ldbr3
LDBR (J)
is K loaded?
No
yes
call killseg for K - descriptor segment
return

Figure 2. Complete Flow Diagram for Ready-him.