Published: 03/20/67

## Identification

Overview of Process Switching
R. L. Rappaport

## Purpose

This document describes the overall strategy used in the switching of processes on the various processors in Multics.

## Introduction

A running process eventually comes to the point at which, for the time being, it is unable to continue running. There are three reasons why a process might discontinue running.

1. The running process may have no further need for a processor until some other event has taken place.

2. The running process may be forced to relinquish control of the processor because it has run out of time or because a process of higher priority has need of the processor.

3. The running process may wish to give temporary control of the processor to another process so that the other process can schedule itself. In this case the original process regains the processor almost instantaneously.

Process switching motivated by either of the first two reasons is inherently different from process switching motivated by the third reason. In the first two cases, the process switched to gains unconditional control of the processor. In the third case the process switched to can only use the processor to schedule itself whereupon it must return the processor to the original calling process. The Process Switching Module has two entry points, swap-dbr (see Section BJ.5.01) and ready-him (see Section BJ.5.02), which correspond to the two types of switches that occur

in Multics. Entry point swap-dbr is used to give away unconditional control of the processor. Ready-him is used when one process wishes to allow another process to schedule itself.

Description

The basic hardware mechanism by which a processor switches from one process to another is the "load descriptor segment base register" instruction.

Conceptually swap-dbr is nothing more than a ldbr instruction which loads the descriptor segment base register with the address of the base of the descriptor segment of the process to which control is being given. Ready-him conceptually is nothing more than an ldbr followed by a call to scheduler (see Section BJ.4.00) followed by a return ldbr. The first ldbr in ready-him loads the descriptor segment base register with the address of the descriptor segment of the process due to schedule itself. The second ldbr reloads the register with the address of the descriptor segment of the process which originally called ready-him.

In actuality swap-dbr and ready-him are slightly more complex than as described above. The additions to the conceptual outlines perform three distinct tasks.

1. Simple book-keeping and housekeeping tasks are performed in the Process Switching Module. For example, swap-dbr is involved in accounting for processor usage.

2. Both swap-dbr and ready-him are partially involved in the tasks of loading and unloading of processes in Multics.

3. Because the Process Switching Module uses shared data bases it must involve itself with interlocking schemes to protect the validity of the data with which it deals.

The first and third tasks above are straight-forward and nothing more need be said about them in this overview. They are described in great detail in BJ.5.01 and BJ.5.02. The second task is more involved and a brief description of the steps taken in ready-him and swap-dbr is presented here.

An active process (see Section BJ.1.00) in Multics is a process that has an Active Process Table entry. In addition, an active process has several entries in file system tables that need not concern us here. A loaded process is an active process that has a hardcore ring descriptor segment and also has its Process Data Segment (which contains the Process Concealed Stack) in core. In order to switch control of a processor to an unloaded, active process, one must first make the unloaded process appear to be loaded. That is, the process trying to switch control to the unloaded process must create a descriptor segment for the unloaded processs and it must provide the unloaded process with a stack.

Ready-him when faced with the problem of switching to an unloaded process merely makes a copy of a hardcore ring descriptor segment template for the unloaded process prior to the switch. Since ready-him is called on the Processor Stack and since it uses this stack to call scheduler no special stack must be created for the unloaded process. After the return from scheduler and after the return switch, the descriptor segment created above is destroyed.

The descriptor segment is created in Ready-him by simply copying from the template descriptor segment (see Section BJ.5.06) and therefore destroying it destroys no useful information. The descriptor segment was created only as a tool to allow the normal scheduling mechanism to work. Figure 1 is a simplified flow diagram of ready-him.

Swap-dbr must handle the preparation of unloaded processes a little differently because the process once switched to will need to do more than the restricted operation of scheduling itself. Swap-dbr prepares both a hardcore ring descriptor segment and an Interim Process Data Segment (which contains an empty Interim Process Concealed Stack) for the unloaded process. It then switches control to the unloaded process. The unloaded process now has a standard descriptor segment and an empty Interim Process Concealed Stack. This process, still executing in swap-dbr, calls the Process Bootstrap Module (see Section BJ.5.03) using the Interim Stack. The Process Bootstrap Module, among other things, restores this process' Process Data Segment to core storage. Upon return to swap-dbr the process has all the information in core that is necessary

for a loaded process. However, its descriptor segment is still the standard one. In order to "personalize" this segment swap-dbr switches stacks to the Process Concealed Stack (from the Interim Process Concealed Stack) and calls subroutine overlay (see Section BJ.5.05). This procedure restores the process' hardcore ring descriptor segment to its exact condition before it was unloaded. Upon return from overlay, the Interim Process Data Segment is destroyed (it is now useless) and swap-dbr returns to its caller. Figure 2 is a simplified flow diagram of swap-dbr.

To rephrase the above paragraph, the object of swap-dbr in the case of switching to unloaded processes is to restore the process to the exact state it was in before it was unloaded. This involves retrieving its Process Data Segment from secondary storage and also recreating its hardcore ring descriptor segment. This is accomplished by creating a standard descriptor segment for the unloaded process which will evolve into an exact duplicate of the unloaded descriptor, and by creating an Interim Process Data Segment which will be used as a tool in retrieving the segment.

Finally it should be noted that the ldbr instructions mentioned are not actually imbedded in swap-dbr and ready-him. Since ldbr is an instruction that can be executed only in master mode, the actual ldbr instructions mentioned above are imbedded in a distinct master mode segment. This segment is known as the ldbr procedure (see Section BJ.5.04).

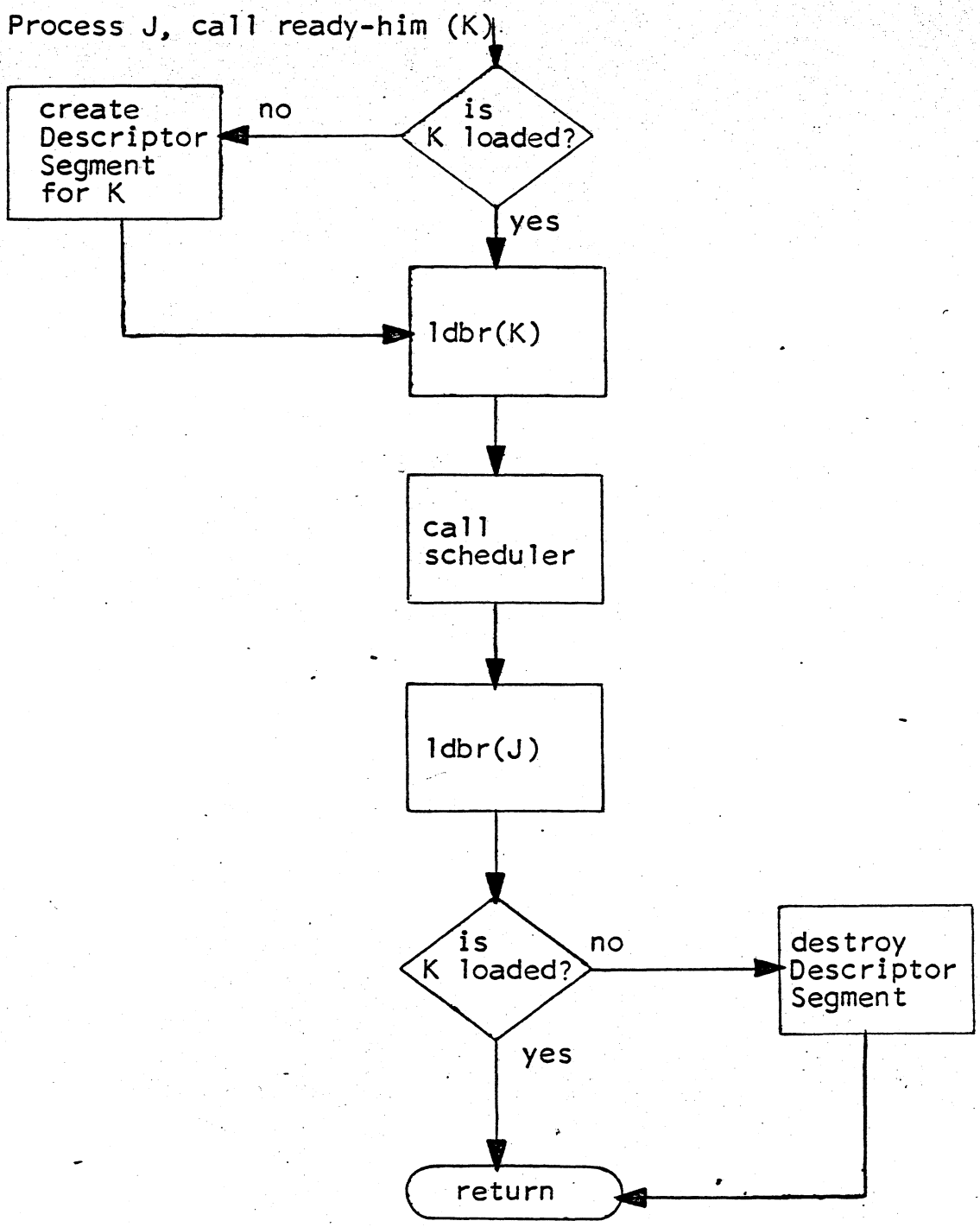


Figure 1. Simplified flow diagram for ready-him.

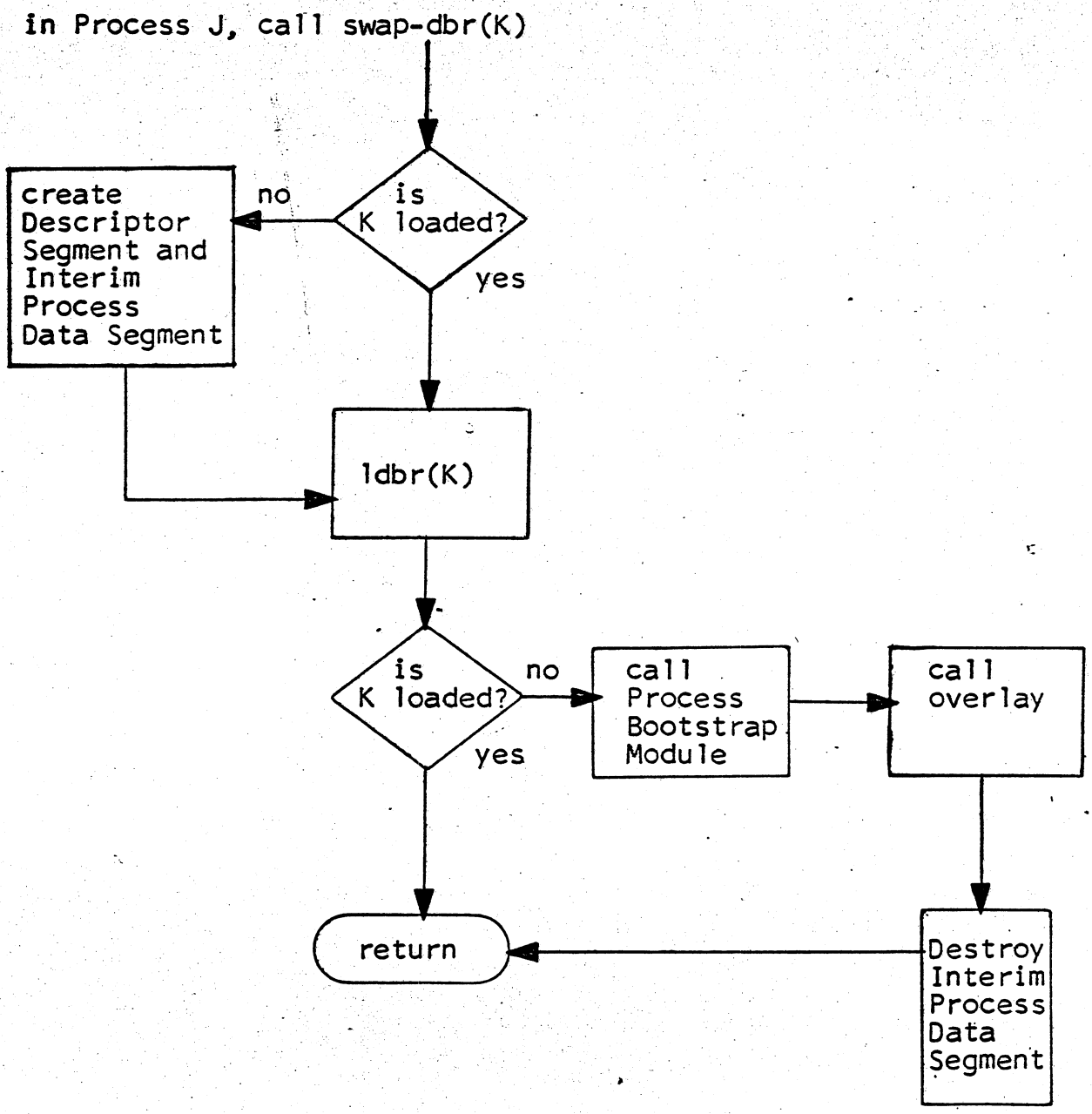


Figure 2. Simplified flow diagram for swap-dbr.