Published: 11/30/66

Introduction

Wakeup
R. L. Rappaport

Purpose

This procedure provides the mechanism whereby processes may signal one another.

Preface

The description of wakeup 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 execution of the subroutine will take place while:

1) The processor is completely masked against interrupts.

2) A global interlock is on which denies access to the Process Exchange to all processes except the one in which this subroutine is currently executing.

The second section is a complete specification that describes the steps that must be taken to allow more than one processor to be concurrently executing in the Process Exchange.

Basic Outline

In Multics, a process wishing to signal another process, calls subroutine wakeup, in the Process Exchange, on behalf of the other process. (In this document the process calling wakeup will be referred to as the caller and the process being signaled will be referred to as the target.) The action taken by the target process, in response to the signal, does not concern us here. We are only concerned with how the signal is passed along.

The strategy taken by wakeup in attempting to signal the target process depends on the current execution state of the target process. In order for a process to be "aware" of anything, it must be executing. Hence, wakeup must do two things to insure that the target process is made aware of the signal:

1. It must leave evidence of the signal to the target process.

2. It must make sure that the target process is either scheduled to run in the future, or running now, so that it can inform itself of the signal.

The above steps are accomplished in the following way. If the target process is currently in the blocked state, wakeup calls subroutine ready-him (see section BJ.5.02) in the Process Exchange, in order to schedule the blocked target process. This one step accomplishes both of the above tasks. The scheduling clearly accomplishes task 2. The first task is accomplished by implication in that the scheduling of the target insures its future return to the running state, and upon elevation to the running state, the process will return to the procedure which called block (see Section BJ.3.01). This return from block implies that a signal has been received. If, however, the target process is not currently blocked, no scheduling must take place but in this case explicit signaling must take place. This is done by setting on the target process' wakeup-waiting switch. This switch is one of the data items in the target process' entry in the Active Process Table (see Section BJ.2.01).

The calling sequence for wakeup is:

call wakeup (A, error-return):

where A is the process I.D. of the target process and error-return is an alternate return to which control should be passed in case A is not active. An inactive process has no Active Process Table entry and wakeup is incapable of doing anything (see Section BJ.1.00) for such a process. The stack used on this call is the processor stack.

To recap briefly, wakeup simply ascertains the execution state of the target. If this state is "blocked", ready-him is called on behalf of the target. If this state is not blocked then the wakeup-waiting switch for the target is set on. Figure 1 illustrates this basic outline of wakeup.

Complete Specification of Wakeup

With several processes possibly executing in the Process Exchange simultaneously, steps must be taken to coordinate their actions. In particular, two general steps have been taken. First, certain interlocks and switches have been established in the Active Process Table entry of each process. By observing common rules about the interlocks,

the various modules are able to guarantee the integrity of the data with which they deal. Second, the processor may, at times, have to be masked against all interrupts, or inhibited, while it is altering these interlocks or switches. (For a complete discussion of coordination in the Process Exchange see Section BJ.6.). Wakeup makes several contributions to the coordination effort.

To understand the first extra step taken by wakeup one must first understand the purpose of subroutine wakeup and what receipt of a wakeup signal means. To do this, it must be understood that no information is passed along to the target as to the nature of the reason for the signal, as a result of the call to wakeup. Any explicit data communication is accomplished, by the caller, prior to the call to wakeup, for example by storing information in a common data base. The wakeup signal is only an indication that something of interest has occurred. Hence a process wishing to wakeup another process can save itself the trouble if a third process is already in the midst of waking up the desired target, since the second wakeup, itself, will give the target no new information. We call such a call to wakeup a redundant call. A call to wakeup for target process A is redundant if someone is in the midst of waking A, or if process A is not running and if the time at which A ceased running the last time is prior to the last call to wakeup on A's behalf. That is if A is not running between times T_1 and T_2, all but the first call to wakeup for A after T_1 but before T_2 would be redundant. It should be noted that redundant calls to wakeup are not disastrous, only wasteful. Therefore, if a call is possibly non-redundant, it must be allowed to go through. It must also be noted that redundant calls to ready-him must not be allowed as this would put the process on the ready-list twice.

To summarize the above, it is desirable to prevent redundant wakeups and absolutely essential to prevent redundant calls to ready-him. This is accomplished by the use of an interlock known as the wakeup lock. This per-process data item exists in the Active Process Table entry of each process. Wakeup makes use of it in the following way. Upon entry wakeup attempts to set the wakeup lock of the target process. If the lock is already set, wakeup merely returns to its caller knowing that some other process will succeed in waking the target.

If the wakeup lock is not already set, wakeup sets it. Immediately before returning wakeup resets the lock. The wakeup lock succeeds in preventing redundant calls

to ready-him since it allows only one process to be instantaneously waking up a particular target. After the first call to wakeup is complete, the target is guaranteed not to be blocked and hence subsequent calls to wakeup will not be translated into calls to ready-him.

Besides the wakeup lock of the target process, wakeup makes use of one other interlock and one switch in the coordination effort. These are the process-state lock and the intermediate-state switch of the target process.

The process-state lock controls access to a group of data items in the respective process' Active Process Table entry, which define the process' execution state. The effect of this lock is to guarantee that the state of a process, as defined by these data items, can only be referenced or altered by one process at a time. In wakeup this means that the target process' process-state lock must be locked before determining whether or not the process is blocked. It also means that the process-state lock must be unlocked before the call to ready-him is made, if this call is necessary, since ready-him will attempt to lock the target process' process-state lock also.

The intermediate-state switch of a process, if on, indicates that the process should not be considered as a candidate for running at this time, even though the process may be on the ready list. This switch is used in wakeup in the following way. As mentioned above, it is absolutely necessary to insure that all possibly non-redundant calls to wakeup go through. The strategy for accomplishing this goal makes use of the intermediate-state switch of the target if this process is currently blocked. After it is determined that the target process is blocked, its state is unlocked and its intermediate-state switch is set on. The wakeup lock must remain locked to prevent redundant calls to ready-him on behalf of the target, and the intermediate-state switch being on prevents the target from running, until the wakeup lock can be unlocked. If the target process were to start running before its wakeup lock was unlocked, it could generate a need for a wakeup that could be blocked by the wakeup lock. Upon return from ready-him the wakeup lock is unlocked and the intermediate-state switch set off.

In addition to the above use of interlocks and switches, wakeup also makes use of processor masking in its attempt to coordinate with other Process Exchange modules. If the target process is not blocked then the sequence of instructions that unlocks the target's wakeup and state

locks must be executed while the processor is masked against all interrupts. Otherwise, if, in between the time the wakeup lock of the target was reset and the state lock was reset, an interrupt occurred on this same processor which necessitated a call to wakeup for this same target then the processor would be caught in an infinite loop waiting for the target's state to unlock. (This example points out the most important benefit derived from the wakeup lock. It allows subroutine wakeup to be executed while the processor is relatively unmasked (i.e., wakeup executes with the mask with which it was called) and yet this lock still prevents redundant calls to ready-him the same target.) Figure 2 is a complete flow diagram of wakeup.

It might be argued that reversing the two steps described in the immediately preceeding paragraph would make unnecessary the masking of the processor. However, this will not work for the same reason that the intermediate-state switch was a necessity above. If, in between the time the process-state lock and the wakeup lock were reset, an interrupt occurred on the processor executing wakeup, enough time would be spent in servicing the interrupt so that the target may have generated the need for a wakeup which would have been stopped by the wakeup lock. (See Section BJ.6 for a complete justification of the above arguments.)

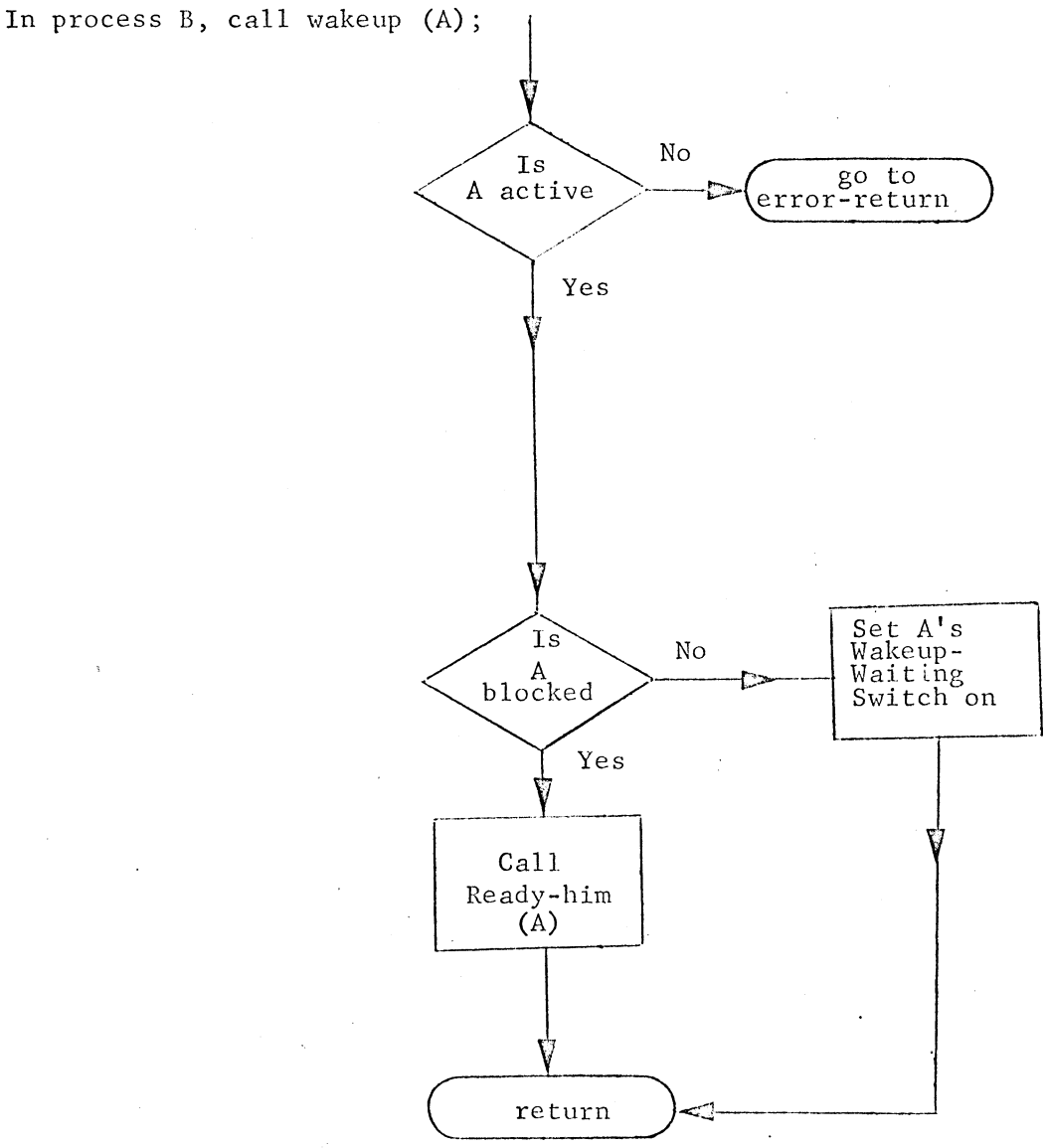
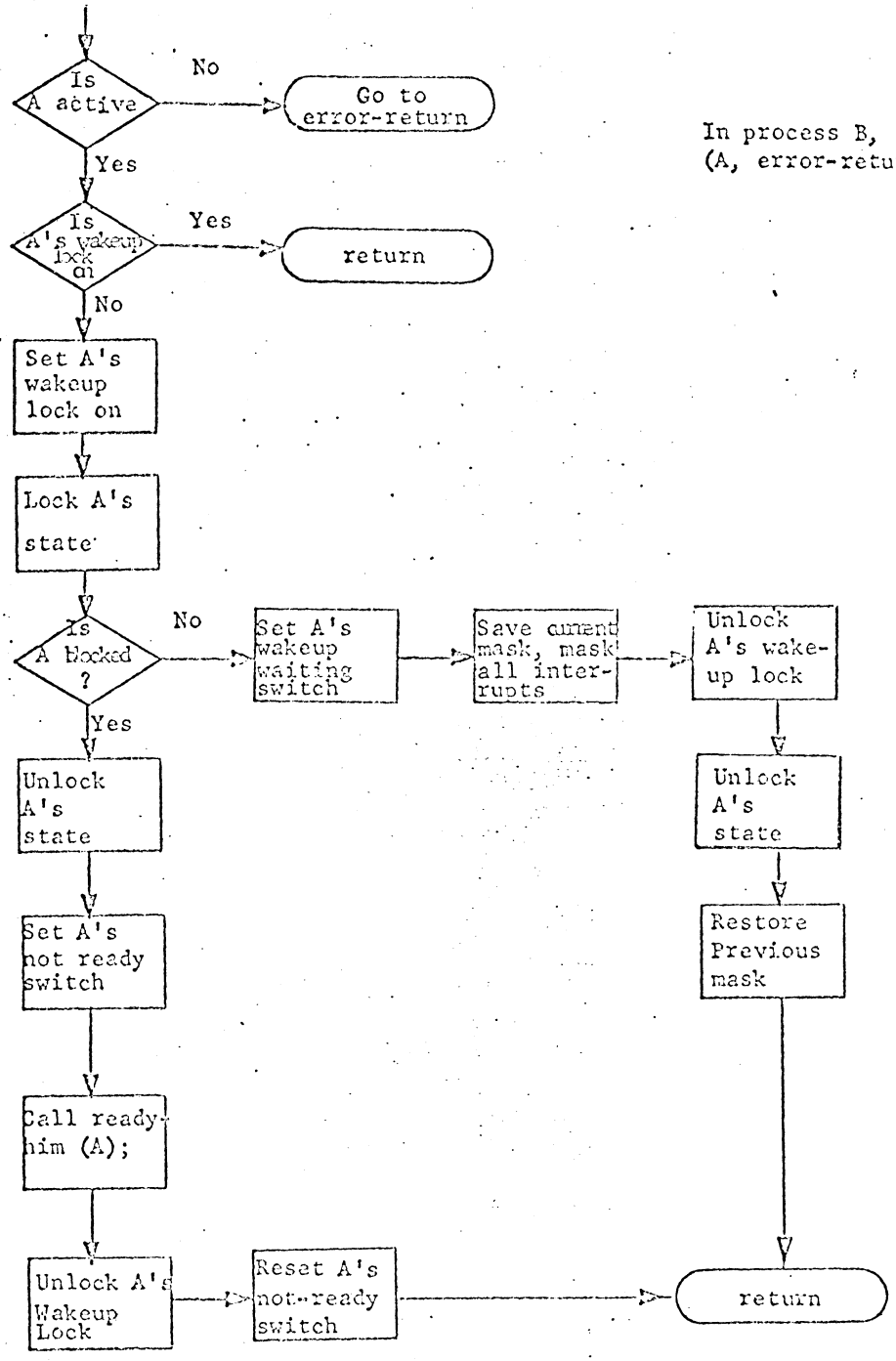


Figure 1. Basic Outline of Wakeup



In process B, call wakeup (A, error-return);

Figure 2. Complete Flow Diagram for Wakeup