Published: 11/30/66

Identification

Quit
R. L. Rappaport

Purpose

Entry point quit in the Traffic Controller provides the mechanism whereby one process can halt the execution of another process.

Preface

The description of quit that follows is divided into three 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 were never unloaded and that execution of the subroutine would take place while:

1) The processor was completely masked against interrupts.

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

The second section presents the necessary additions to the basic outline that enable the unloading of processes to be accomplished. The final section is a complete specification that describes the steps that must be taken to allow more than one process to be concurrently executing in the Process Exchange.

Introduction

A process that has been "quit" appears to be a normal process that is in the "blocked" state, and, in fact, it is. The difference between a "quit" process and another blocked process is that a process going into the blocked state normally arranges to be sent a wakeup signal, but a process that has been "quit" will not receive a wakeup signal until the process which "quit" it decides to send it one. A process that is trying to quit another one then must go through two steps:

1. It must cause the process to enter the blocked state.

2. It must prevent any wakeup signals from reaching the blocked process.

Entry point quit in the Process Exchange is concerned with the first of these steps. The discussion of quit handling in BO.1.08 describes the second step.

The calling sequence for quit is:

call quit (A);

where A is the process I.D. of the process to be quit (the target process).

The strategy used in causing a process to go into the "blocked" state depends upon its current execution state. If the process is "ready", it must simply be removed from the ready list. If the process is running, it must be sent a process interrupt that will cause it to call entry point block (see Section BJ.3.01) in the Process Exchange. Since subroutine block does not cause a process to become blocked if its wakeup-waiting switch is on, quit turns off the wakeup-waiting switch of the target before interrupting this process. If the process is already "blocked", nothing need be done. Therefore, quit is basically simple; it determines the execution state of the target process and takes the appropriate action as described above.

Figure 1 is an illustration of the basic outline of quit.

Additions to enable unloading of processes.

The reason for "quitting" a process is to force it to stop running as soon as possible. If the process is not currently loading, as soon as possible means immediately. If the process is loading itself, as soon as possible means immediately after the loading is complete. This is because a process in the midst of loading itself temporarily uses several pages of "wired down" core storage. If the process were stopped indefinitely, the core would remain "wired down" indefinitely. (See Section BJ.1.02.) What this means is that quit must be able to notify a loading target process that it has been quit and that this loading process must still be able to complete the loading operation.

If the target process is loading (as can be determined by the state of the target's process-loading switch) quit will set on the target's quit-waiting switch and then return, regardless of the current execution state of the target. All processes when resetting their state from loading to loaded in swap-dbr (Section BJ.5.01) are required to check the contents of the quit-waiting switch and go blocked if it is on. In this way the quit is delayed until the loading is complete. Figure 2 illustrates the basic outline of quit with the addition necessary to enable unloading of processes.

Complete Specification of Quit

With several processes possibly executing in the Process

Exchange concurrently, steps must be taken to coordinate their actions. In particular, two general steps have been taken. First, certain interlocks and switches have been placed 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. Secondly, at certain times while some of these interlocks are set, the processor referencing the locked data must be masked against all interrupts. This is to prevent the possibility of putting a processor into an infinite loop. (For a complete discussion of coordination in the Process Exchange see section BJ.6)

The main coordination problem faced by _quit_ occurs when the target process is in the ready state. The problem arises from the fact that the target may have already been chosen to run in subroutine _getwork_ (Section BJ.4.02). If this is the case the target, although in the ready state, may change its execution state to running at any instant. In this case the course of action that _quit_ should follow is not clear. Therefore _quit_ should defer action until either the target's execution state is changed to running on until it is determined that _swap-dbr_ (Section BJ.5.01) will be unable to switch control of a processor to this target. _Quit_ determines whether or not a ready process is "chosen" by testing the status of the process-chosen switch in the process' Active Process Table entry. This switch is turned _on_ in _getwork_ when a process is picked to run. It is turned _off_ in _swap-dbr_ if the switching is successful. If the switching is unsuccessful it is turned _off_ by _getwork_ when _swap-dbr_ performs an error-return.

There are two other coordination problems which must be faced in _quit_. The first is that _quit_ makes use of several data items which other Traffic Controller modules might attempt to make simultaneous use of. In order to prevent fatal mishaps an interlock has been created which controls access to these data items. The interlock is the process-state lock and it exists as a data item in the process' Active Process Table entry. The items to which it governs access are:

1. The running switch
2. The ready switch
3. The quit-waiting switch
4. The wakeup-waiting switch

The first two items define the process' execution state, and the use of the second two has already been discussed. In quit, none of the above switches may be referred to or altered unless the appropriate process' process-state lock has been locked.

While the process-state lock is set in quit the processor must be masked against all interrupts. This is to prevent an interrupt from being serviced whose handler might attempt to set this same process-state lock. Therefore the processor must be completely masked before the process-state lock is set in quit and cannot be unmasked until this lock is reset.

The final coordination problem has to do with the fact that quit makes use of the ready list. The ready list has a global interlock which limits access to one process at a time. Therefore quit must lock the ready list prior to referring to it. This will ensure that no other process is using this data base. Since getwork uses the ready list in choosing processes, quit should lock the ready list prior to testing the status of the target's process-chosen switch. The ready list lock will prevent getwork from choosing the target after its chosen switch has been tested. Finally, it should be stated that the ready list can only be locked while the processor is completely masked. However, in quit the ready list is locked only while the target's state is locked and this first locking can only be accomplished while the processor is masked. Hence, no more masking is needed. Figure 3 is a complete flow diagram of quit.

In process B, call quit (A);

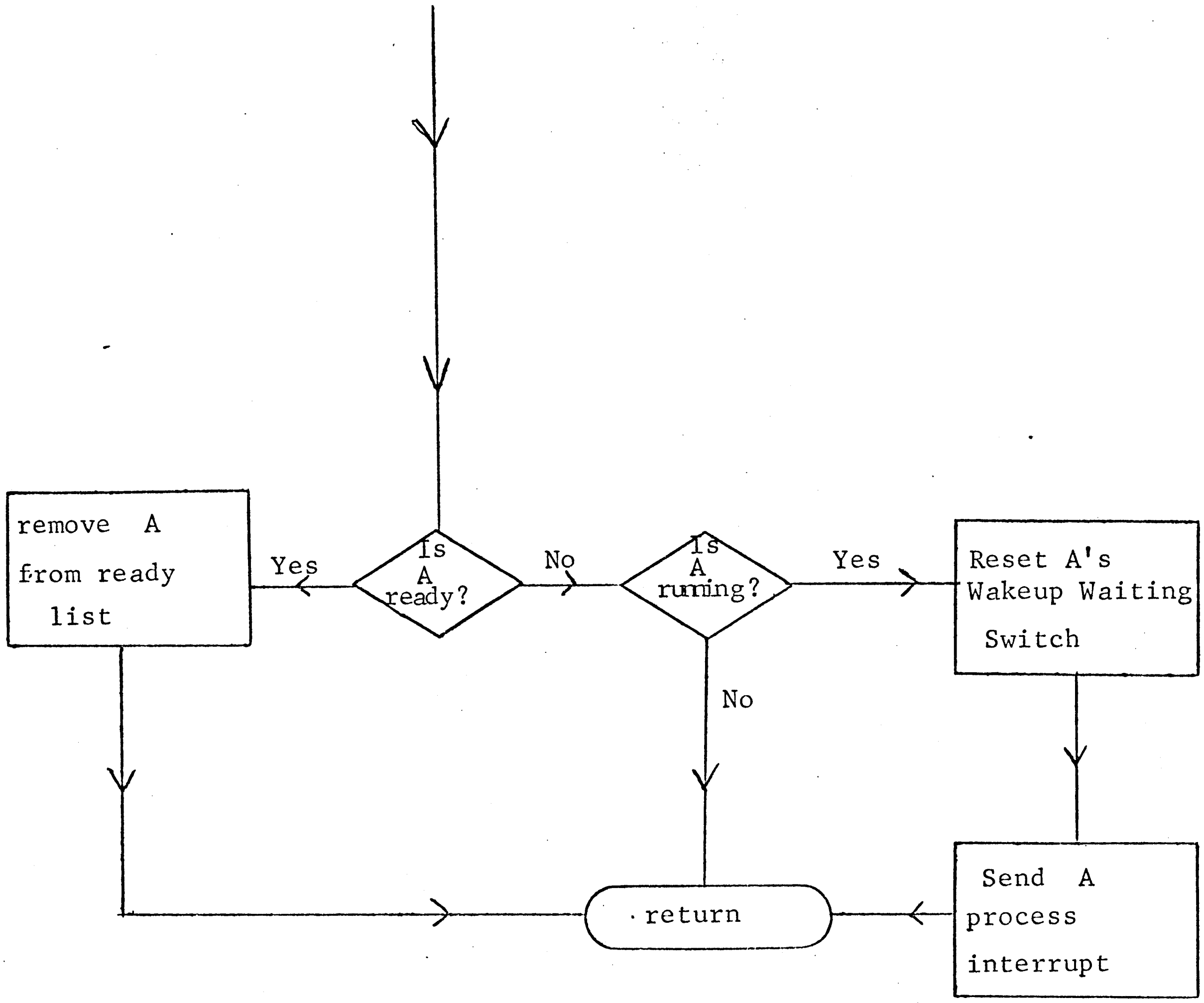

Figure 1. Basic Outline of quit.

In process B, call quit (A,);

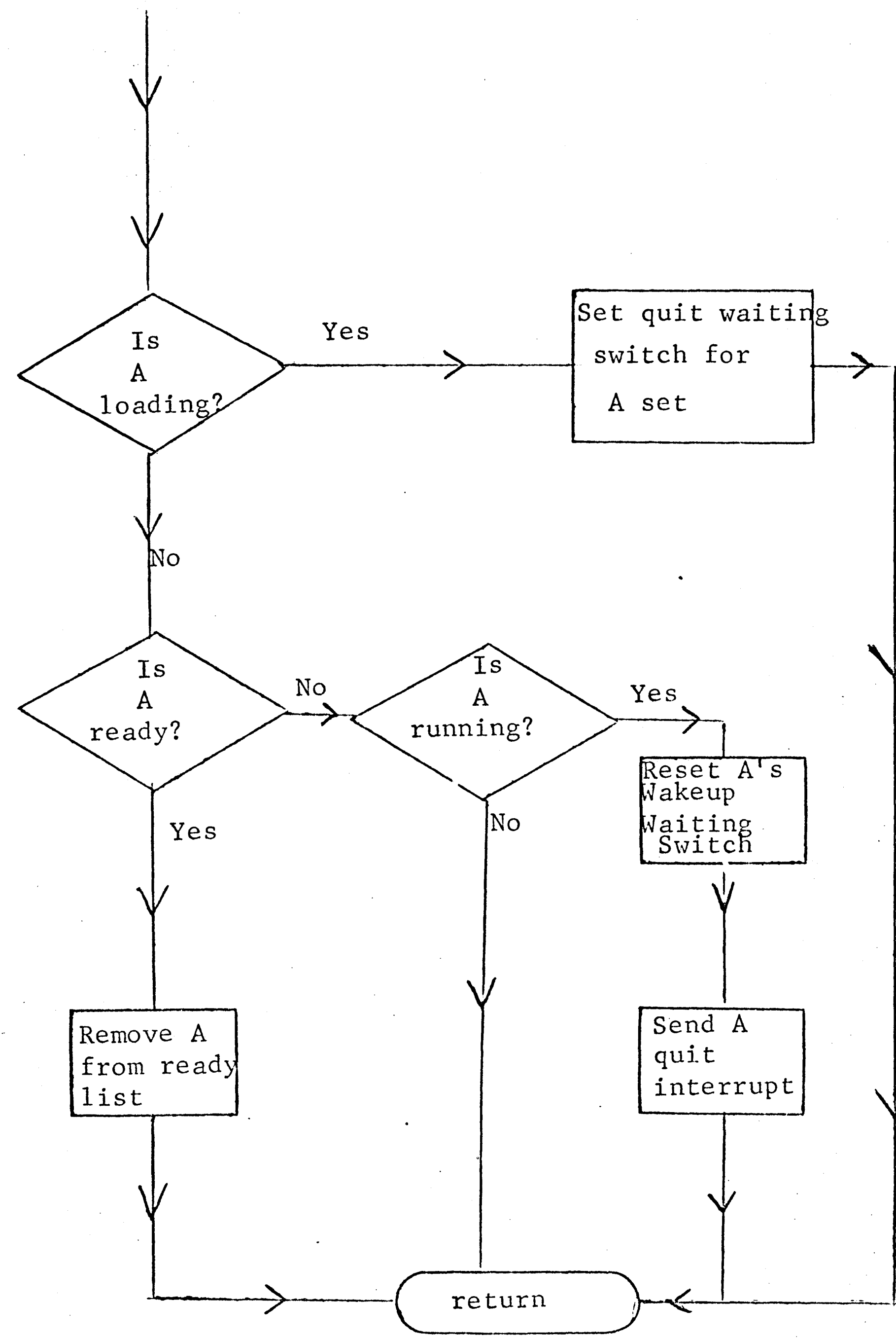

Figure 2. Basic outline of <u>quit</u> with additions

In process B, call quit (A);

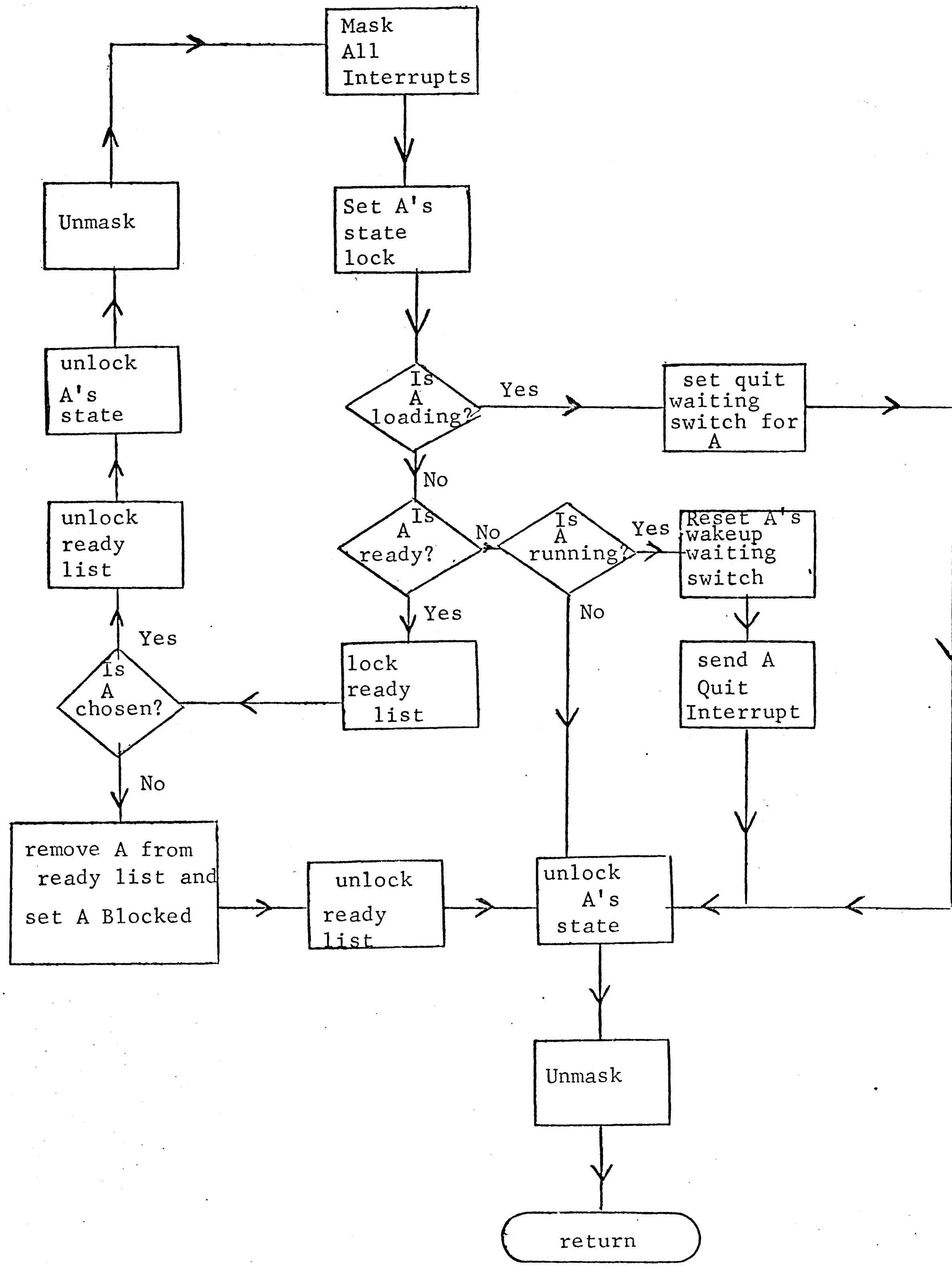

Figure 3. Complete flow diagram of quit.