Published: 08/24/67

Identification

stop procedure
J. J. Donovan

Purpose

Stop is called by the overseer procedure (See BQ.3.01) in response to a quit, automatic logout, or suspension event. Stop halts the execution of a user's work as soon as possible and establishes a state in which the Overseer may alter, destroy or save the user's work. Stop is a ring 1 procedure, callable only in ring 1 and called by the overseer. Stop operates only in the Overseer Process.

The user's work is being done by processes in his process group known as Working Processes. Asynchronously the users' I/O is being done by Universal Device Managers operating in other groups and (possibly) by Device Managers in the user's own group. (See Section BQ.3.00 on the process-group). The functions of stop are to halt the Working Processes and I/O of the user and prevent the halted processes from waking up. The Overseer may then alter, destroy or save the halted processes.

Basic Outline

The stop procedure must perform the following functions:

1. Halt all working processes.

2. Prevent the halted processes of a process group from receiving any wakeups, except those initiated by a start command. (See Section BX.3.06)

3. Halt all I/O processing for this process group.

Stop Implementation

The stop procedure will be called by the following:

```
call stop;
```

Figure 1 depicts the Flowchart for the implementation of stop.

To halt the working process, the stop procedure calls the quit_process entry of the Traffic Controller (See Section BJ.3.03) for each Working Process of the process group. The calling sequence used to quit a working process is

```
call quit_process (A);
```

where A is the process I.D. of the process to be quit. Stop finds the I.D. of all working processes in the overseer data base which contains a list of all working processes in the process group. (See Section BQ.3.01 for description of the overseer data base.)

To prevent the halted processes from receiving wakeups, stop must ensure that the sources of wakeups are inhibited. In Multics there are two sources of wakeups, the interprocess communication facility (See Section BQ.6.04) in the administrative ring and the process wait and notify module in the hardcore ring. (See Section BF.15.01). The quit_process procedure in the Process Exchange ensures that wakeups are not sent from the process wait and notify module. The quit_process procedure causes the target process to block itself. If, however, just prior to attempting quit the target process was executing in ring zero, quit_process establishes the necessary mechanism to run the target process out of ring zero and have the target process go blocked as soon as it leaves ring zero. The quit_process procedure returns only after the target process has run out of ring zero and then called block. Since the process wait and notify module only sends wakeups to a process while the process is in ring zero, no wakeups will be sent to a quit process from the process wait and notify module. The only remaining source of possible wakeups is, then, the interprocess communication facility.

To prevent the quit Working Processes from receiving any wakeups, stop sets a flag (wakeup inhibit flag) in the event table of each process that is to be quit. The interprocess communication facility examines the wakeup inhibit flag before issuing any wakeups. To avoid races, stop must set this flag before calling the quit_process entry in the traffic controller.

Stop sets the wakeup inhibit flag in the event table of a process that is to be quit by the following call to the event channel manager (See Section BQ.6.04):

```
call set_wakeup_sw (A, "0"b);
```

where the argument A is the process I.D., and the argument "0"b is the flag setting, indicating on.

Before setting the wakeup inhibit flag in the event table, the stop procedure reads the status of the quit flag which is set in the overseer data base. If this flag is on, indicating that the Working Process has previously been quit, then stop sets a destroy flag associated with this Working Process in the overseer data base. The destroy flag may be interpreted by stop's caller and perhaps some appropriate action taken. For example, if the quit procedure was stop's caller, then the quit procedure destroys the processes with the flag on. This destruction is a mechanism used so as to control the number of processes in a process group.

To summarize the above, stop obtains the ID of a Working Process from the overseer data base. If the process has previously been quit, stop sets the destroy flag on for this purpose. If the Working Process has not been quit, stop sets the wakeup inhibit flag in the event channel of the Working Process, thus preventing a quit process from receiving wakeups. Stop calls the quit_process procedure which stops the execution of the Working Process.

After quitting each working process, stop sets the quit flag associated with the working process in the overseer data base.

Finally to halt all I/O processing for this process group (Step 3 of basic outline), the stop procedure calls the io_ctl$stop procedure of the I/O System (See Section BF.3.01). The io_ctl procedure returns after all I/O is stopped.

A summary of the information in the overseer data base which the stop procedure uses is the following: Associated with each working process there is a process id, quit bit, destroy bit, and a list of channels over which processes may signal the overseer.

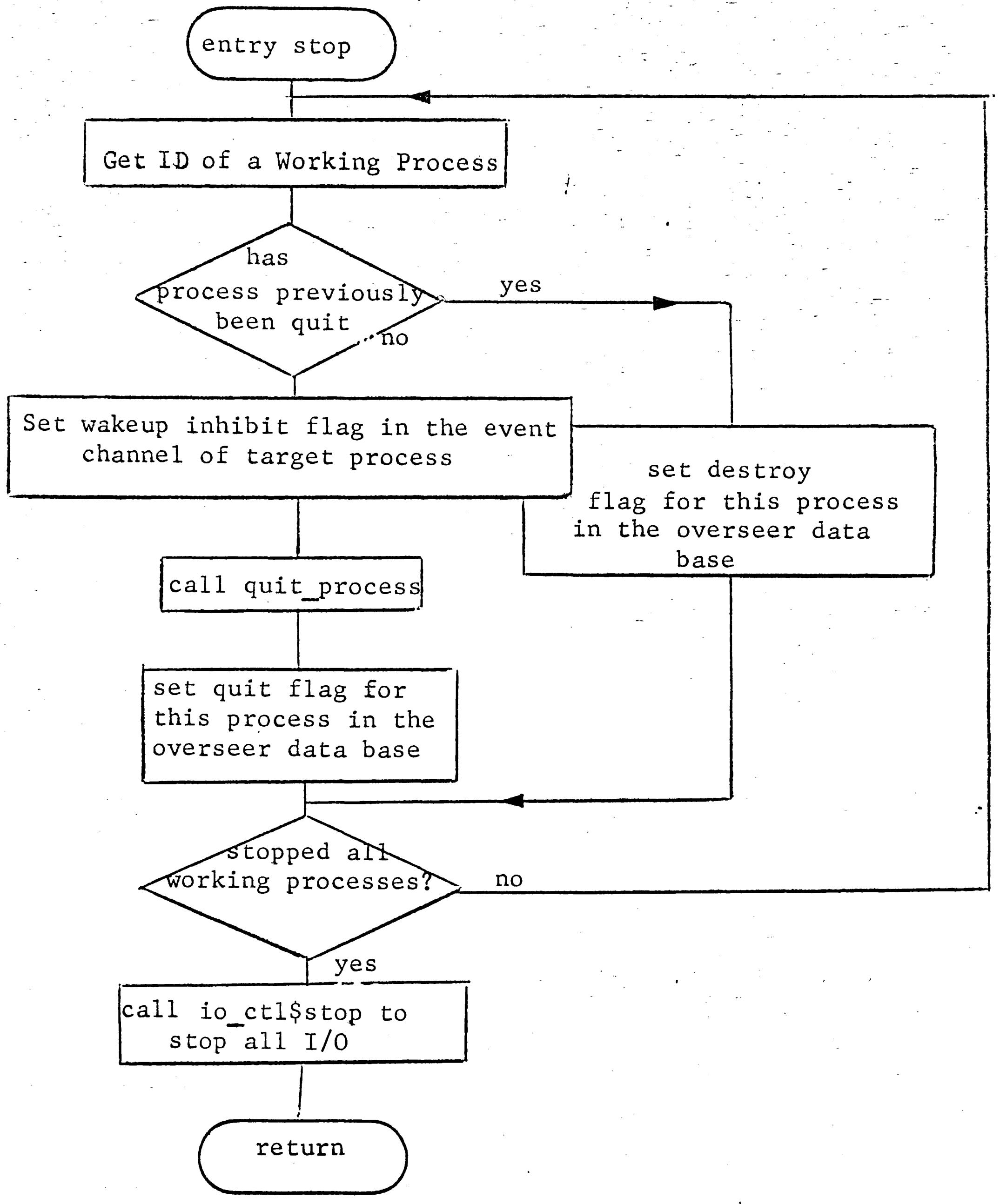
entry stop
Get ID of a Working Process
has process previously been quit
yes
no
Set wakeup inhibit flag in the event channel of target process
set destroy flag for this process in the overseer data base
call quit_process
set quit flag for this process in the overseer data base
stopped all working processes?
no
yes
call io_ctl$stop to stop all I/O
return

Figure 1 Implementation of stop