for publication

TO: Multics Distribution

FROM: Carla Marceau

DATE: December 18, 1967

SUBJ: BQ.3.03

The attached minor revision of BQ.3.03 on the stop procedure allows ring 1 procedures to declare themselves uniquittable for short periods of time, according to conventions soon to be described in BQ.3.06 on inhibiting quits.

thornheaffing

DRAFT TYPED: 12/4/67

Identification

stop procedure
J.J. Donovan, P.A. Belmont

Purpose

Stop is called by the overseer procedure (See BQ.3.01) in response to a quit, logout automatic logout, out of funds 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 user 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 the start procedure. (See Sections BQ.3.04 and BX.3.06)
3. Halt all I/O processing for this process group.
4. Help in housekeeping by marking for destruction unwanted previously quit working processes.

Stop Implementation

The stop procedure ~~will be~~ is called by the following:

call stop;

See flow chart for stop in figure 1.

The stop procedure is a table driven routine; the table is the Working Process Table. The Working Process Table (see BQ.3.01) is a per User Process Group data base used by the overseer and its subroutines in administrating the User Process Group. It contains one entry per working process in the group. Each entry contains the following information of interest to the stop procedure:

The process id

The quit_flag: set when a process is quit and reset by the start and hold procedures. (See BQ.3.04)

The quit_pending_flag: set by stop when ~~it~~ and destroy-wp desires to quit a working process.

The destroy_flag: set by stop if the corresponding quit_flag is found set; the Overseer's quit procedure (which calls stop) destroys all working processes whose destroy_flags are set.

The quit_inhibit_counter: a count of the level of unquittability of the working process. Level zero means "quittable". (See BQ.3.06)

The i_am_quittable event channel: channel on which a working process signals upon becoming quittable and finding its quit_pending_flag on.

Stop sees that all of the working processes of the group are quit in two steps. First it passes through the Working Process Table quitting quittable processes, marking for destruction the processes whose quit_flags it finds set, and listing the i_am_quittable event channels of the processes it finds unquittable. Second, it waits for these i_am_quittable events. When one is received, its corresponding

quit_inhibit_counter is again tested. If zero, the process is quit and the count of unquit processes is decremented. Stop continues to wait for i_am_ quittable events until all the processes have been quit.

When all working processes have been quit, stop calls io_control$stop to stop all the I/O attached to the process group and then returns.

Let us look in some detail at how stop quits a working process (with process_id=A). Quitting the process means bringing it to a blocked state (that is, removing it from the running or ready state), insuring that it receives no wakeups and thus remains blocked, and setting its quit_flag.

In MULTICS there are two freely used 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). Direct calls to wakeup are used sparingly, for example by "start" (See BQ.3.04). 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 to 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 Process from receiving any wakeups, stop sets a flag (wakeup inhibit flag) in the event table of the 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.

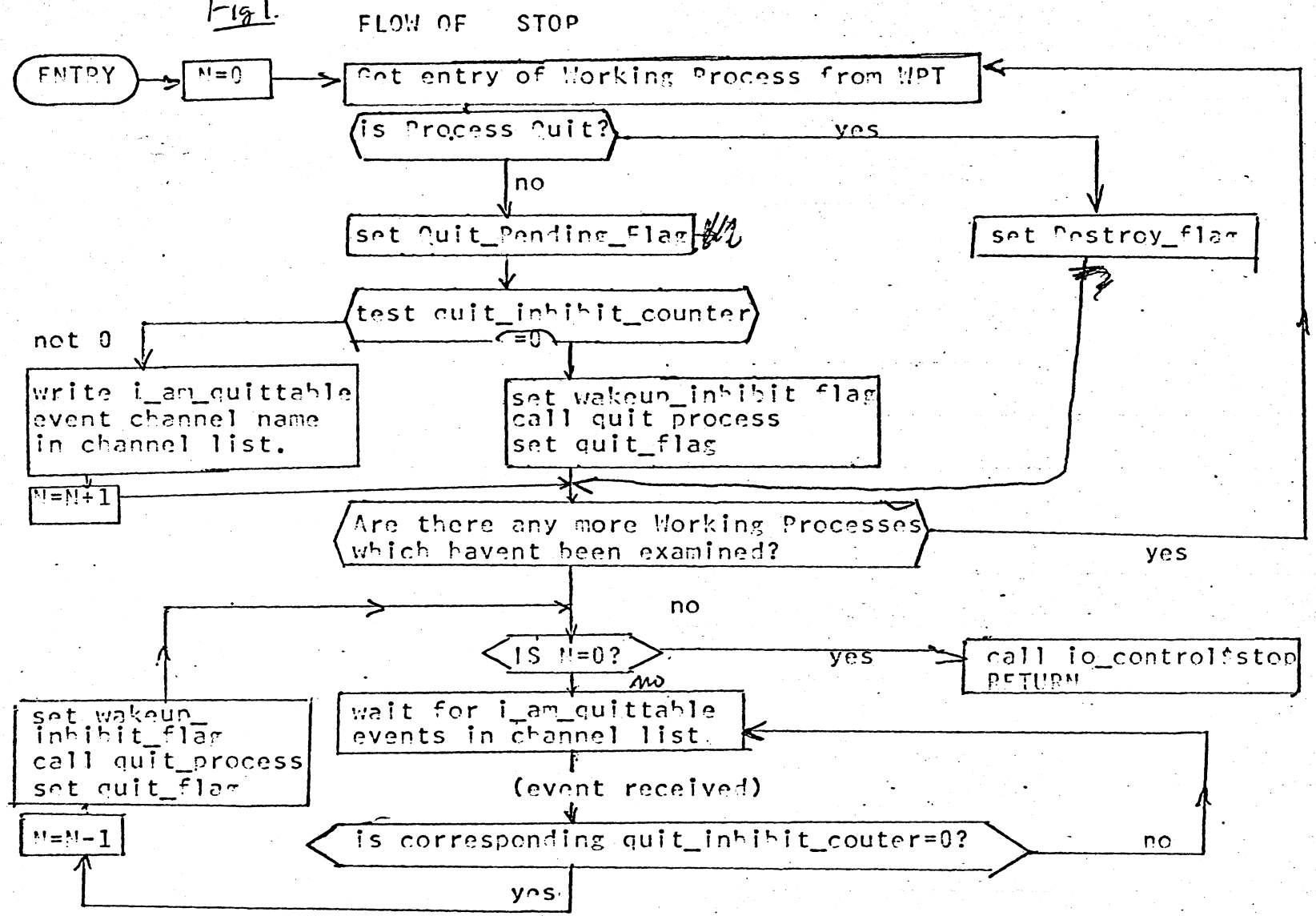
Stop sets the wakeup inhibit flag in the event table of a process that is to be quit by the following call to the event channel manager (See Section BQ.6.04):

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

Stop quits the process by a call to the traffic control module (See BJ.3.03);

```
call     quit_process (A).
```

Fig 1.


FLOW OF STOP
ENTRY
N=0
Get entry of Working Process from WPT
is Process Quit?
yes
no
set Quit_Pending_Flag
set Destroy_flag
test quit_inhibit_counter
not 0
=0
write i_am_quittable event channel name in channel list.
set wakeup_inhibit flag
call quit process
set quit_flag
N=N+1
Are there any more Working Processes which havent been examined?
yes
no
IS N=0?
yes
call io_control$stop
RETURN
no
set wakeup_ inhibit_flag
call quit_process
set quit_flag
wait for i_am_quittable events in channel list.
(event received)
N=N-1
is corresponding quit_inhibit_couter=0?
no
yes