Published: 02/23/67

Identification

Overview of Traffic Control
J.H. Saltzer

Purpose

This section presents a general summary of the procedures of the central supervisor that perform processor multiplexing, interrupt management, and inter-process signaling. The procedures are known collectively as the traffic controller.

References

Basic concepts of the traffic controller are set forth in the Project MAC Technical Report "Traffic Control in a Multiplexed Computer System", by J.H. Saltzer, MAC-TR-30, published July, 1966. This thesis presents the design approach to the traffic controller and is useful for background information.

Disclaimer

The present section BJ.0 is merely an edited version of earlier design documents on the Traffic Controller. It is issued at this time to provide some accurate overview information since the earlier documents are no longer completely correct or easily accessible.

Terminology

A process is basically a program in execution. The tangible evidence of a process is a processor stateword (a set of machine conditions) and an associated two dimensional address space (a core image.) The address space of a process, defined by a Descriptor Segment, determines the region of accessibility of the processor, both in execution of instructions and in obtaining data. A dynamic linking mechanism allows the process to change the contents and extent of its own address space, but this does not alter the fundamental view of a process as the execution of a program contained in the address space.

Within the system every process known to the system is identified by a unique number, its process I.D. This number is a key to a table of all known processes, which contains further information about each process.

Every process is in one of three execution states: running, ready, or blocked. A running process is at this instant executing in some processor. A ready process is one which would be running if a processor were available. A blocked process is one which has no use for a processor; it is waiting for some event to happen. The event might be arrival of a signal from elsewhere in the system, or perhaps completion of a computation by another process.

Every process either is or is not loaded into core memory. The definition of loaded is entirely an operational one. The "core image" part of a process may be stored in core memory, or an secondary storage, or split between the two. A process is defined as loaded only if enough of it is present in core memory that it may operated within critical supervisor modules. A precise definition of "loaded" is given in sections BJ.1.00 and BJ.5.02.

An active process is one for which there is sufficient information in core storage to allow it to enter the ready or running states. The necessary information for an inactive process is stored on secondary storage, and must be retreived before the process is allowed to run. Operationally, an active process is one which appears in the Active Process Table.

A number of things can happen to divert a process from its programmed course. These diversions have been variously termed traps, interrupts, and faults. We use the term interrupt when referring to hardware signals coming from outside the processor which cause a processor to depart from the procedure it was executing. Interrupts are distinguished from faults, which are triggered by hardware signals generated within the processor.

Processor multiplexing includes both the sharing of processors among many users to provide interactive response (sometimes called time-sharing) and switching among several procedures in response to interrupts so as to keep both processors and I/O devices as efficiently used as possible (sometimes called multiprogramming).

The Traffic Controller

The Traffic Controller is a set of procedures appearing within the address space of a process. Although every process in the system must have a working, compatible, Traffic Controller, it is not necessary that all processes have an identical Traffic Controller. Those processes which have identical Traffic Controllers use a common copy of the procedures involved as shared segments.

The functions provided by the Traffic Controller are intentionally primitive; it is viewed as the innermost layer of a multilayered supervisor existing within a process. In fact, it is unlikely that any user's program would ever be permitted to call the Traffic Controller entries directly. Instead, the user's program would call some outer supervisor layer which, for example, checks the authority of a call to signal another process.

The rest of this document will describe the Traffic Controller as though it is used directly by some "customer." It is understood, however, that its only "customers" are actually other supervisor procedures.

The Traffic Controller can be conveniently broken into two distinct parts which perform its major functions:

1. The system interrupt interception routines.

2. The Process Exchange.

The three major functions of the Traffic Controller are the following:

1. Perform multiplexing of processors among processes.

2. Provide an interface with the system interrupt hardware.

3. Allow one process to signal another.

An important function of the Traffic Controller is processor multiplexing. To visualize this multiplexing, consider the progress of a process, as seen by the system. As time passes, the process goes back and forth among the running, ready, and blocked states as in the time diagram below.

run	ready	run	ready	run	blocked	ready	run	ready	run

time ——>

The Traffic Controller has inserted the ready states in order to multiplex, or share, the processor among all the processes presently demanding service. The process, however, does not normally observe the times spent in "ready" status. From the point of view of this particular process, the above diagram looks like this:

run	run	run	blocked	run	run

with dotted lines indicating points at which the calendar clock takes a quantum jump. Multiplexing is arranged so that, except for the real time clock jumps, it is basically "invisible" to the affected process. This means that a process can completely ignore the multiplexing being performed by the supervisor. It also means that a process must be substantially independent of timing. A further implication is that service to critically timing dependent hardware functions must be provided by the Traffic Controller itself.

<u>The Interfaces of the Traffic Controller</u>.

The Traffic Controller has two interfaces: on the one side with the system interrupt hardware, and on the other with the rest of the supervisor and the user's program. The hardware interface is described in detail in the section on interrupt handling, BK.2.

The interface with the rest of the process consists primarily of three calls into the Traffic Controller. (There are also several less important entry points concerned with process creation, etc.; these entries do not affect the significance of this discussion and can be ignored for the moment.) The three calls in are to entries named Block, Wakeup, and Quit. Figure one is a block diagram of the Traffic Controller. It shows there three external entries going to a module named the Process Exchange, the interrupt hardware interface, and also calls between those two major sections.

The entry named Wakeup is used whenever a process wishes to wake up a blocked process. The wakeup by definition is directed to some named process. A typical call from within process "A" to wake up process "B" would be

Call Wakeup (B)

Process "B" may be running, ready, or blocked at this time. The call has an effect only if B is blocked, in which case B will be unblocked, or awakened.

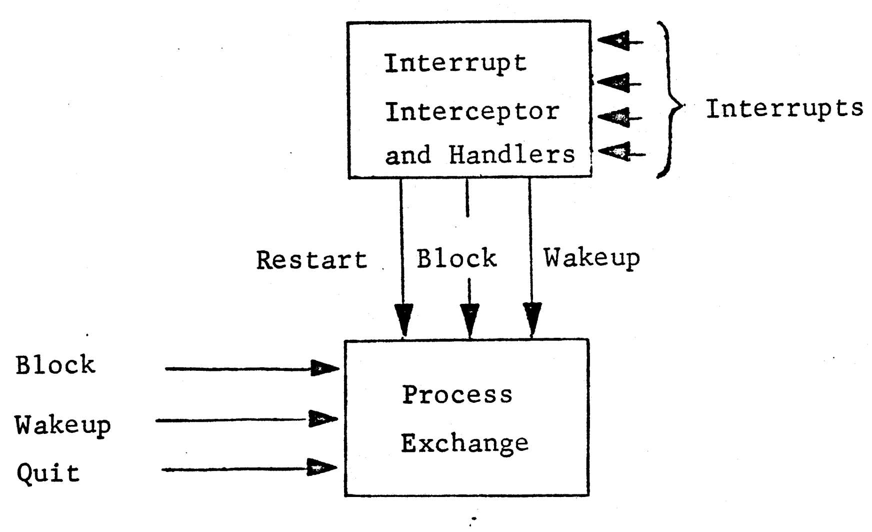

Figure 1 -- Block diagram of Traffic Controller.

The entry point Block of the Traffic Controller is called by a process when that process cannot proceed until a signal in the form of a wake up from another process arrives. It is the responsibility of the process calling Block to insure that some process will indeed wake it up. Block is then called with no arguments:

Call Block

The Traffic Controller will place this process in Blocked status, where it will remain until some wakeup signal arrives for it.

When the process eventually receives a return from its call to Block it can be assured that some process has called entry point Wakeup for this process.

The Quit entry of the Traffic Controller is the inverse of the Wakeup entry; it is used to place another process in Blocked status. If process "A" wishes to block process "B", "A" can

Call Quit (B)

Process "B", if ready or running, will immediately revert to Blocked state. A later call by "A", or any other running process, to entry point Wakeup for process "B" will permit "B" to continue from the point at which it was interrupted.

Interrupt Handling

The underlying philosophy of interrupt handling is that interrupts are signals similar in nature to Wakeup calls, but originating in external hardware equipment. Thus the sole function of the interrupt handling routines is to transform an interrupt into appropriate calls to the Process Exchange. As an example, for an interrupt representing the completion of a write operation on a typewriter, the interrupt handler would call Wakeup for the process which originated output to the typewriter. No other computation is done at the instant of the interrupt. The process "responsible" for the interrupt (in the above example, the process initiating I/O on the typewriter) is scheduled by the Wakeup call; computation in response to the signal (data transformation, redundancy checking, etc.) is not accomplished until the responsible process begins execution.

A comprehensive overview of interrupt handling is provided in section BK.2.01.

Interaction with core control

The operations of processor multiplexing interact with those of core memory multiplexing. A special interface between the basic file system and the traffic controller helps guarantee that the traffic controller will not attempt to multiplex processor capacity among so many processes that memory becomes too crowded.

To this end, a little-used process may be unloaded by core control if space becomes too tight; when an unloaded process comes to the top of the ready list it will not be reloaded until adequate core space is available for it to run efficiently. Unloading is accomplished by paging out its descriptor segment and other segments needed to enter the running state; the process is remembered only by its entry in the Active Process Table.

As a further measure, a process which has not been used for some time may be deactivated, which means that its Active Process Table entry is copied into pageable storage. Since reactivating an inactive process requires a directory search it can only be done at a time when page faults are permitted; this has the result that only blocked processes may be deactivated. Activation of a process is done by a special (and never deactivated) system process which receives all wakeups intended for inactive processes. If the system is overloaded, the activator process can choose to delay activation of some processes.

Details of the interface between the traffic controller and the basic file system are contained in sections BJ.1 and BJ.5.

Process Control

In addition to the Process Exchange and the interrupt handling procedures, the Traffic Controller contains a "housekeeping" module, known as Process Control. This module provides entries to

1. Initialize the rest of the Traffic Controller and the processor hardware.
2. Create new processes.
3. Delete old processes.
4. Interface with the basic file system to perform core memory multiplexing.
5. Simulate an execution meter (processor usage meter) for each process.

An overview of the Process Control module may be found in section BJ.1.00.