Draft 5.31.68
Supersedes 2.23.67

Identification

Overview of Traffic Control

J. H. Saltzer, R.L.Rappaport, M.J. Spier

Purpose

This section presents a general summary of the procedures of the central supervisor that perform processor multiplexing, interrupt management, and inter-process signalling. 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 Jerome H. Saltzer, MAC-TR-30, published July, 1966. This thesis presents the design approach to the Traffic Controller and is useful for background information.

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 more information about each process.

Every process is in one of five execution states:

1. running
2. ready
3. waiting
4. blocked
5. quit

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 waiting process does not have immediate use for a processor, it is waiting for a system-event (for example, the arrival of a page into core) to happen within a predictable period of time. A blocked process is one which has no use for a processor, it is waiting for some event to happen some time in the future. The event may be arrival of a signal from elsewhere in the system, or perhaps completion of a computation by another process. A quit process is a blocked process that does not await events and which is guaranteed to have left its hardcore data bases in a predictable state.

Every process 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 in 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 operate within critical supervisor modules.

An active process is one for which there is sufficient information in core storage to allow it to enter the ready state. The necessary information for an inactive process is stored on secondary storage, and must be retrieved before the process is allowed to enter the ready state.

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 processor 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.

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, a user's program is never permitted to call the Traffic Controller entries directly. Instead, the user's program calls 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, waiting and blocked states as in the diagram below:

run	wait	ready	run	ready	run	blocked	ready	run	wait	ready	run

The Traffic Controller has inserted the ready states in order to multiplex, or share, the processor among all the processes 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	wait		run	blocked	run	wait	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.

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.

The interface with the rest of the system consists primarily of six calls into the Traffic Controller. (There are also several less important entrypoints concerned with process synchronization, process creation, and processor-resource management. These entries do not affect the significance of this discussion and can be ignored for the moment.)

The six calls can be classified into three groups as follows:

1. Process Wait and Notify (PWN) calls: wait, notify.
2. Interprocess Communication (IPC) calls: block, wakeup.
3. Process interrupt calls: restart, quit.

The Process Wait and Notify calls

Every process reaches a point in its execution where it has to have information from some other process; if the information is unavailable, it abandons the processor on which it currently executes until such time as the information will become available, or until that event happens. We name "event" anything that is observed by some other process and which is of interest to our process. We distinguish between two classes of events, system-events and user-events. This distinction is made for reasons that are largely implementation-dependent. (Theoretically, events are all of the same natur and can be haddled uniformly.)

System-events are characterized by the fact that they can be observed in the hardcore ring only and that they are guaranteed to happen within a predictable period of time (normally measured in milliseconds.) These include the arrival of a page into core, or the unlocking of a currently-locked systemwide data base.

A process that has to wait for a specific system-event calls

wait (event)

This call puts it into the waiting state and associates it with 'event' so that when some other process observes the occurance of 'event' it calls notify (event)

which causes all the processes which are currently waiting for 'event' to be restored into the ready, and eventually the running state. As can be seen, the PWN calls are event oriented. PWN is discussed in detail in sections BJ.2.

<u>The Interprocess Communication calls</u>

A process may wish to give its processor away until it be notified of the occurance of a user-event. Typical of a user event is that it may happen anytime in the future; also, a user-event is process-oriented (the signalling is done towards a specific process rather than "generally broadcast") and is always associated with some information.

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 wakeup 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 called with two arguments:

call block(interaction_switch, event)

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

The 'interaction_switch' indicates whether or not the process is blocking itself while interacting with a human being, in which case the process will be given a higher-than-usual priority in its race for a processor, when awakened, to insure quick system response to human requests; 'event' is a queue of event-messages, returned by the Traffic controller.

The entry name wakeup is used whenever a process wishes to wake up a blocked process. The wakeup, by definition, is directed to some named process as a result of the observing of some user-event. A typical call from within process 'A' to wake up process 'B' and inform it that event 'E' has happened would be

call wakeup(B,E)

Process 'B' may be running, ready, waiting, blocked or quit at this time. Although the information associated with event 'E' will not be lost to process 'B', the call will have effect only if 'B' is blocked, in which case it will be restored to the ready state, or awakened.

<u>Process Interrupt calls</u>

We recognize two types of process interrupts

1. The timer run-out interrupt
2. The quit interrupt

the first occurs whenever a process depletes its current processor-time allotment, the second whenever some other process wants to deprive this process of its processor-time.

When a process is initially made to run, it is given a certain time allotment which the hardware keeps track of. When this time has been used up, a process-interrupt is generated which diverts the process' execution into an interrupt handler which then calls the Traffic Controller's entry point

call restart (execution_switch)

to reschedule the process, give it a fresh time allotment and put it into the ready state. 'execution_switch' tells the scheduler whether the process was executing in behalf of the system or in behalf of the user when its time allotment ran out.

Sometimes, a process may wish to 'stop' another process' execution. If process 'A' wants to stop process 'B' it calls

call quit (B)

which will put process 'B' into the quit state. By convention, process 'B', if currently executing in behalf of the system, is allowed to finish its current system-task before it is quit. This is done in order to insure that a quit process always leaves hardcore databases in a predictable state.

Entrypoint restart, and the scheduler, are discussed in section BJ.5, quit is discussed in section BJ.4.

Interrupt Handling

The underlying philosophy of interrupt handling is that interrupts are signals similiar in nature to wakeup calls, but originating on 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, signalling an event-name which is associated with that 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 that typewriter) is restored into the ready state by the wakeup call; computation in response to the signal (data transformation, redundancy checking, etc.) is not accomplished until the responsible process begins execution.

Interaction with the File System

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 the File System if space becomes too tight; when an unloaded process comes to the top of the ready list it will not be re-loaded until adequate space is available for it to run efficiently. Unloading is accomplished by paging out the remainder of 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 for result that only blocked or quit processes may be deactivated.

Loading, unloading, activating and deactivating are done by a special (and never unloaded or deactivated) system process known as the Traffic Controller Daemon Process and which receives all signals intended for inactive processes. This daemon process is described in section BJ.6.

Draft 5.24.68

Supersedes 8.3.67

Identification

Overview of Traffic Controller Data Bases

R.L. Rappaport, M.J. Spier

Purpose

Associated with the Traffic Controller are certain data bases, some of which are per-system and other of which are per process. (Per processor data bases are discussed in sections BK.) The details of these various data bases are discussed in the remaining sections of BJ.1. The present section is an overview and a discussion of the strategies used for selecting a particular data base for a particular item.

Per System Data Bases

All per-system data bases are maintained by the Traffic Controller in segment <tc_data> . There are five major parts of this segment:

1. The Traffic Controller Data Block (TCDB)
2. The Active Process Table (APT)
3. The Active Event Table (AET)
4. The APT Hash Table (APTHSH)
5. The Activator Table

The Traffic Controller Data Block is a collection of miscellaneous items in segment tc_data; some of these items are internal to the Traffic Controller (pointers to the various tables, table descriptions), others (such as the variable containing the number of loaded processes) are of general interest.

There are two main categories of items in this data block: Certain per-system Traffic Controller items have to go somewhere and seem most naturally to go here. There are also system parameters which presumably will not be changed other than occasionally by a system administrator (for example, the number of levels in the multi-level ready-list). Rather than have these parameters built into code as constants, it seems more appropriate to collect them all in a single place. The TCDB is described in BJ.1.1.

The Active Process Table contains certain information about each process which is currently active. An item must go into the APT if it needs to be accessed from other processes. For example, sending a wakeup to a process involves knowing that process' execution-state; likewise, selecting the next process-to run from the top of the ready-list involves knowing that process' loading-state. Consequently, those two state variables have to be kept in the process' APT entry. The APT is discussed in BJ.1.2.

All processes waiting for a particular system event are threaded into a single list (associated with this event) and the head of the thread is kept in the Active Event Table. The AET is a table containing a group of pointers to a collection of event threads running through the APT. The AET is described in BJ.1.3.

Communication between processes is done on the basis of (symbolic) process identification, but the communication itself requires knowing the location of the target-process' APT entry. To make efficient looking up the APT, given a process-id, a hash-table of process-ids is maintained by the Traffic Controller; APTHSH is a typical hash table associating process-ids and relative pointers into the APT. The APTHSH is described in BJ.1.4.

An "inactive" process does not have an APT entry. A signal (wakeup, quit, unquit) sent to an inactive process is diverted to the Traffic Controller Daemon Process (loades/activator daemon) which has "power of attorney" for inactive processes. Whenever subroutines wakeup, quit and unquit are unable to locate an entry in the APTHSH for a certain process-id, they write that process-id into the Activator Table which is the daemon's mailbox. The Traffic Controller Daemon Process retrieves these process-ids from the Activator Table and activates the corresponding processes. The Activator Table is described in BJ.1.5.

All of these data bases are accessed in the hardcore ring at times when page faults cannot be tolerated, so segment tc_data (as well as the procedures constituting the Traffic Controller) is wired down. It is accessible for reading and writing in the hardcore ring only.

Segment tc_data is pre-assembled, and loaded during system initialization time from the Multics System Tape (MST). It is initialized by a procedure named tc_data_init which allocates space in tc_data for the various above-mentioned tables, initializes the tables and puts values into the variables in TCDB. By convention, mainly for reasons of clarity, all tables begin at an address that is 0 mod 64, and all APT entries begin at an address that is 0 mod 16.

Per Process Data Bases

In addition to the per-system data bases described above, the Traffic Controller maintains certain data bases for each process in the system. In general, there are two types of data: Taht which must be wired down, and that which need not be. The former is kept in the Process Data Segment (PDS) and the latter in the Process Definition Segment (PDF).

The process data segment contains two basic items: The process' concealed stack and a block of miscellaneous data referred to as the Process Data Block (PDB). The PDS is discussed in BJ.1.6.

The process definition segment is similiar to the process data segment but it is not wired down. It also contains two items: The fault stack and the Process Definition Block. The PDF is discussed in BJ.1.7.

Strategies

All Traffic Controller data items of a per-system nature are kept in segment tc_data. Per-process items can be kept in either the APT, the PDS or the PDF. If the item must be accessed by any other process it is kept in the APT. Otherwise, the decision as to which block to put it in is based on whether or not it must be wired-down: If so, it must go into the process data block; while if not, it may go into the process definition block. Clearly, it is desirable to put as few items as possible into the process data block so as to minimize the amount of wired-down core.

For convenience, certain items are kept in both the APT and the PDS. These are items which other processes need to know, but which the current process must access frequently. Accessing the PDB is more efficient.