Published: 10/01/68

Identification

Multi-programming Control
Robert L. Rappaport, Michael J. Spier

Purpose

In order to achieve optimum system performance, a mechanism is needed to coordinate the (otherwise independent) core-control and processor-multiplexing mechanisms. This coordination is done by the Traffic Controller in conjunction with the Traffic Controller System Process (TCSP). This section is an overview of System Resource Management and provides a complete discussion of the issues involved. The Traffic Controller System Process is described in section BJ.6.01.

Introduction

The Traffic Controller's function is to multiplex the system's processor resources among all competing processes. To briefly summarize, multiplexing is achieved by giving a processor to a process with the understanding that the process is to willingly abandon the processor as soon as it does not have immediate use for it (wait, block, or stop) and with the further understanding that if it does not willingly abandon the processor within certain time limits then the processor will be forcibly taken away from it (timer runout).

Core Control does the memory-resource management. Whenever a process is reaching out for a page that is not in core, Core Control brings that page in. If necessary, it first throws some other page out of core in order to provide the necessary space. "Page turning" is relatively expensive in processor time as it involves computations as well as I/O.

A process is expected to be thrifty with processor time. Whenever it has no immediate use for a processor, rather than loop until some condition be satisfied, it calls wait or block to give its processor away.

By giving the processor away, the process saves the amount of time that would otherwise have been wasted looping. However, by the time that that process is made to run once more, it may find that one or more of its pages have been thrown out of core, and that the expense of recovering those pages might be completely out of proportion with the original saving. In other words, we find ourselves

in a situation where we pay dollars for the satisfaction of saving pennies. The problem arises from the fact that a process is given not only a processor, but the totality of system resources (processor, core, secondary storage, I/O devices) to dispose of. By polarizing oneself on efficiently managing one of these resources, one may easily over-burden and wastefully mismanage the others.

The system must be managed economically, similar to any commercial business enterprise. The cost of saving must be known, and only when the expense of saving some resource is known to be fractional in comparison with the expected return should such saving be implemented.

Discussion

A process may be in one of the following five execution states: Running, Ready, Waiting, Blocked and Stopped. Also, a process may be in one of the following three loading states: Unloaded, Being loaded/unloaded, Loaded. These two state variables are associated with processor and core resources respectively and are independent of one another except for the prerequisite that a running process must be loaded; otherwise, any combination of the two states is possible. When a process is made to run for the very first time, it must spend a considerable amount of processor-time in establishing its working memory space, (making segments known and bringing pages into core memory). A process needs a certain minimum amount of free core in which to put its pages so that it could execute for a certain amount of time without incurring any page faults. Unless that much free core is provided, the process will spend the larger part of its life thrashing around handling page faults. Consequently, only a very limited number of processes can be allowed to execute concurrently if this condition is to be maintained. Moreover, if one of those concurrently executing processes reaches a point in which it knows that it will not be able to go on executing until some event happens in the very near future, it seems unadvisable to let it give the processor away to some other process, because we then risk the loss of all the time invested in establishing the current working memory space. So, it seems more economically sound to let a few milliseconds of processor time to go to waste rather than risk the loss of those invested seconds.

In view of all this, if one wants to maintain an overall system efficiency one has to be able to exercise control over processes in a way such as to maintain a reasonable relationship between the actual work performed by the process and the overhead in managing its memory space.

Control is exercised by allowing only a limited number of processes to actually participate in the race for a processor at any given time. Such processes are said to be "eligible for multiprogramming". An eligible process is a process in which an investment has been made and which is therefore guaranteed to execute long enough in order to make that investment profitable. Similar to the above-made definition of processor multiplexing, a process is made eligible with the understanding that it will willingly abandon this state whenever it reaches a point where it must give its processor away for an undetermined length of time (block or stop) and with further understanding that if it does not willingly abandon its state within certain time limits then the state will be forcibly taken away from it.

A process is said to be loaded when it has enough information in core to enable it to take segment and/or page faults; this includes a wired-down ring-0 descriptor segment, a wired-down PDS and the activation of its PDF and KST segments. A loaded process is using up wired-down core, so the number of loaded processes must also be restricted.

Two system variables, tc_data$max_loaded and tc_data$max_eligible delimit the number of loaded and eligible processes, respectively. These variables can be changed by the system administrator to allow him to control and "finetune" system performance. These variables are related to one another so that

tc_data$max_loaded $\geq$ tc_data$max_eligible

The relationship between execution-state, loading-state and eligibility is as follows:

1. Only a ready process can be made eligible (however, once eligible that process can assume the ready or waiting states without affecting its eligibility).

2. An eligible process must be loaded (an ineligible process which is a candidate for eligibility must first be loaded before it can assume the eligible state).

3. A process which assumes the blocked or stopped state (or which has suffered pre-emption) must give up its eligibility.

4. Only processes which are eligible and ready and loaded are considered as candidates for running. When a processor is available and none of the currently ready processes is eligible and loaded, then the processor is given to the idle process (see BJ.7).

Implementation

When a process is put on the ready list a check is made to see how many eligible processes there are, currently. If their number is less than the maximum number of eligible processes then a wakeup is sent to the Traffic Controller System Process. Also, when a process gives up its eligibility it sends a wakeup to the Traffic Controller System Process. The TCSP wakes up and tries to confer eligibility upon the topmost ready & ineligible process(es) on the ready list, within the limits of tc_data$max_eligible. Then, it looks for unloaded eligible processes, and tries to load those which it finds. In attempting to load, it checks to see whether the maximum number of loaded processes has been over-reached, in which case it first unloads the oldest ineligible loaded process.

The following is a resumé of the above-mentioned logic:

1. A process is made eligible if it is on top of the ready list by the time that an eligibility-vacancy occurs.

2. A process loses its eligibility when its eligibility-time-limit has run out or when it calls block or stop or when it has suffered pre-emption, thus causing an eligibility-vacancy.

3. An eligible process is always automatically loaded by the Traffic Controller System Process.

4. An eligible process is unloaded only if this is necessary in order to allow the loading of an unloaded eligible process.

5. A process is chosen to run only if it is the topmost

   ready & eligible & loaded

   process on the ready list.

Certain system processes are always eligible, such as the Traffic Controller System Process, or the idle processes. The maximum number of eligible processes is therefore adjusted to include all those system processes.