Central Processor Unit (CPU) Scheduling and How Memory is Managed

By

Kenneth G. Foster

May 2, 2010

Abstract

This paper briefly discusses the symbiosis between central processor scheduling and memory management. Four processor scheduling algorithms are briefly discussed along with Rate Monotonic Scheduling and its applicability to both single processor and multitasking implementations.

The processor scheduling and memory management algorithm's complement each other, each providing a unique interdependent services to the other. Designers can implement a variety of combinations dependent on the needs of the design. The applications were designed and are constantly refined to streamline a computers central processor unit (CPU) output potential. In this paper we will review the differences between the various key choices within the realm of processor scheduling algorithms.

A processor scheduling algorithm acts as the manager for the workload that the CPU is requested to execute (Stallings, 2009 P. 406). The memory manager works in conjunction with the scheduling algorithm to flow instructions to the CPU. These applications work together to perform the function of determining a weighted, value-based, decision matrix. This matrix is intended to render the right instructions, at the right time, with the least amount of idle time as possible into main memory based on an assigned priority system (figure 1). There are four primary types of processor scheduling algorithms; First Come First Served (FCFS), Round Robin (RR), Shortest Process Next (SPN), and Shortest Remaining Time (SRT) (Stallings, 2009 P. 414-423). Each algorithm is designed with emphasis on different methods and priorities of execution. These priorities may include support for preemption as well as time scheduling for multi-user systems.

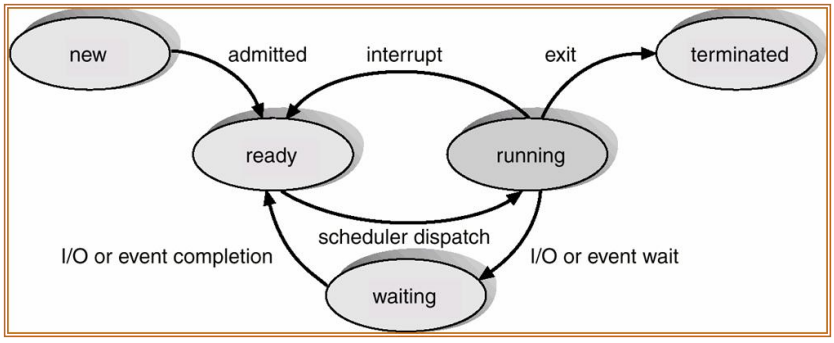

Figure 1. CPU Scheduling Cycle Overview

The First Come – First Served (FCFS) algorithm, which is sometimes also referred to as First Come, First Served, is the simplest of the scheduling policies to implement. It functions just as its title implies, executing the processes based on an age calculation (figure 2) with priority of oldest to newest (Stallings, 2009 P. 414). This scheduler favors longer processes, placing shorter ones at a disadvantage. The scheduler also favors processor processes over Input / Output (I/O) processes (Stallings, 2009 p.416). This can cause inefficient utilization of both the processor and the I/O devices (Stallings, 2009 p. 417). This scheduling policy is not well suited for single processors and has a turnaround time that is unfavorable, rending an average of over one third of the processes with a service time ten times higher than normal (Stallings, 2009 p. 428).

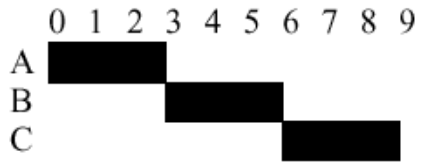

Figure 2. First Come – First Served Scheduling Wait Pipeline

The Round Robin (RR) scheduling algorithm uses a preemptive scheduling routine and bases its interrupt intervals on a clock (Stallings, 2009 p. 417). This results in a reduced penalty for short jobs when compared to the FCFS scheduling algorithm (figure 3). This particular scheduling is considered a general purpose scheduler and is very well suited for time sharing systems (University of Bridgeport, nd). Using a technique called time slicing, the scheduler allocates a set increment of time to each process in the queue, rotating access to processes in a cyclic order until its satisfies all the processes, returns again to service the process, or the process terminates (Yaashuwanth & Ramesh, 2010). There are several drawbacks to the type of scheduler. The primary drawback of this scheduler is that execution intensive processes receive

a higher portion of access than those that require I/O access (Stallings, 2009 p. 417). Enhancements to this scheduler include a suggested Virtual Round Robin (VRR) as a way to increase the fairness of process allocation (Stallings, 2009 p. 418).

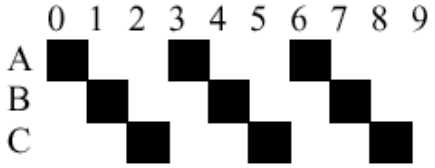

Figure 3. Round Robin Schedule Processing Overview

The Shortest Process Next (SPN) scheduling algorithm is a non-preemptive policy implementation of a scheduler (Stallings, 2009 p. 418-419). This means that once the process begins the execution stage, it continues till complete (Stallings, 2009 p. 413). This reduces the bias demonstrated in FCFS scheduling for longer processes. This scheduler is weighted to execute the process with the shortest anticipated processing time (University of Bridgeport, nd). This process is superior to RR scheduling except when it comes to short processes (Stallings, 2009 p. 428). This scheduler has several disadvantages which include the overhead to perform calculations to determine the estimated time to execute. It calculated incorrectly, this can cause jobs to be incorrectly prioritized or aborted (Stallings, 2009 p. 419). This scheduler can also starve longer processes and is not recommended for time sharing systems (Stallings, 2009 p. 422).

The Shortest Remaining Time (SRT) algorithm is a variation of SPN and has a preemptive interrupt sequence added for efficiency (Stallings, 2009 p. 422). This scheduler weights processes with the shortest expected processing time remaining for execution. Each time a new process is accepted, the schedule checks the new process against the current executing process and updates the table of calculations in regards to the shortest time remaining

(Callari, 1996). This is in contrast to FCFS which favors longer processes. Unlike RR, SRT has a reduced overhead by not adding additional interrupts to the scheduling (Stallings, 2009 p. 422). One of the risks associated with this algorithm is larger processes can be left in a ready state in favor of smaller processes (University of Bridgeport, nd). In situations where idle time is low, the larger processes run the risk of not being served.

Rate Monotonic Scheduling (RMS) is a scheduling solution that places an absolute premium on reliability and task execution through deadline management. Originally written for uniprocessor systems, RMS scales well for multitasking systems as long as the processors share a priority queue and bus access that is not unduly inhibited (Carnegie Mellon University, nd). All operating systems execute based on the processor cycles or clock ticks (Stallings, 2009 P. 467). In a RMS or real-time scheduling, the system calculates the required time for completion of the task and compares it to the queue of tasks and deadlines (Carnegie Mellon University, nd). An RMS system never denies a task due to a lack of sufficient resources (Stallings, 2009 p. 477). These types of systems are commonly used in critical applications such as lab experiments, industrial control systems, financial system, robotics, air traffic control, telecommunications, and military applications (Stallings, 2009 p. 466). RMS systems are designed to prevent the failure which may result in the lost critical data or hardware damage (Stallings, 2009 p. 468). Their design is such that scheduling efficiency rates are normally over the 90 percentile, allowing for high processor utilization (Sha & Sathaye, 1995).

RMS uses an execution priority scheduling calculation that places premium upon time a processes completion requirements. In an RMS system if a high priority task fails to complete by its designated time interval, the system could corrupt, crash or suffer unrecoverable consequences. Its avoidance mechanism is to assign processes for execution a priority value,

based on the determined required time boundary for execution (Stallings, 2009 p. 476). Because real-time processing tasks have an associate urgency time value (figure 4), each task has specified parameters such as a start and ending criteria (Carnegie Mellon University, nd). The system schedules tasks for completion based on their deadlines, placing highest priority on those closest to expiration, following sequentially by the next furthest out and so on (Carnegie Mellon University, nd). This type of scheduling works well on both real-time and time-sharing systems.

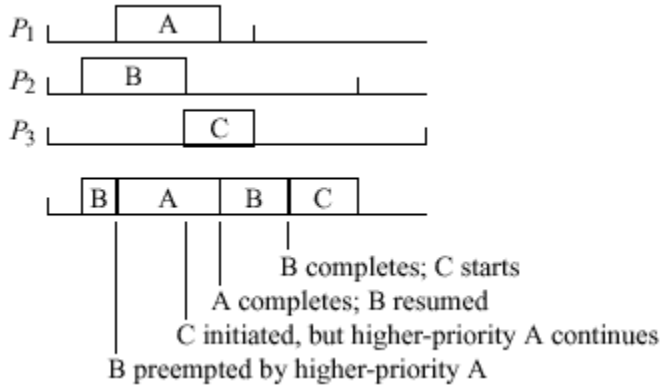

Figure 4. RMS Scheduling Preemption Example

Not all tasks in an RMS system will cause fatal errors. There are two types of tasks classifications, Hard and Soft tasks (Stallings, 2009 p. 467). In order to establish a high level of reliability, critical tasks are considered a Hard Tasks and their failure would be catastrophic to the system. A Soft Task provides the system a mechanism to recover from a deadline breach without the loss of data (Stallings, 2009 p. 468). This is accomplished as the result of other integrity mechanisms available to the system (Carnegie Mellon University, nd). There are also two types of scheduling tasks, periodic and aperiodic (Stallings, 2009 p. 467). Periodic tasks are predictable and occur on a scheduled rate of time, whereas aperiodic tasks are more random in nature (Stallings, 2009 p. 467). Another characteristic of an RMS system is that User Control is far broader in scope and berth. This provides the user a high granularity of control over RMS

based system (Stallings, 2009 p. 468). Since RMS systems are extremely sensitive to time of process execution, one of the metrics used to measure success is the Interrupt Service Routine (ISR). This gages the length of time after an ISR is issued till it is serviced. As evidenced above, RMS systems accommodate both single processors and multithreaded environments and excel in high demand critical environments where execution surety takes precedent.

System designers have many complex decisions that they are required to make before finalizing their designs. The selections of the appropriate processor scheduling and memory management algorithms allow them to tailor the design to perform at the highest level of performance, reducing overall idle time. Design specifications will drive decision such as responsiveness, preemption, and reliability. These combinations can benefit both single processor and multitasking systems if selected in the proper combination.

References:

Callari, F. (1996). *Shortest remaining time next (SRTN)*. Retrieved May 2, 2010, from Operating Systems 304: http://www.cim.mcgill.ca/~franco/OpSys-304-427/lecture-notes/node44.html

Carnegie Mellon University. (nd). *Rate Monotonic Analysis.* Retrieved May 2, 2010, from Software Engineering Institute: http://www.cs.fsu.edu/~baker/realtime/restricted/notes/s6-r.pdf

Sha, L., & Sathaye, S. S. (1995, September). *Distributed System Design Using Generalized Rate Monotonic Theory*. Retrieved May 2, 2010, from Defense Technical Information Center: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.9376&rep=rep1&type=pdf

Stallings, W. (2009). *Operating Systems.* Upper Saddle River, NJ: Pearson Prentice Hall.

University of Bridgeport. (nd). *Operating Systems (CS/CPE 408) : CPU Scheduling* . Retrieved May 2, 2010, from http://www1bpt.bridgeport.edu/sed/projects/cs503/Spring_2001/kode/os/scheduling.htm

Yaashuwanth, C., & Ramesh, R. (2010). Intelligent Time Slice for Round Robin in Real Time Operating Systems. *International Journal of Robotics Research* , 126-131.