CMPSCI 377 Operating Systems **Fall 2008**

Lecture 7: September 23

Lecturer: Prashant Shenoy *Scribe: Shashi Singh*

7.1 Scheduling

Scheduling is a key concept in computer multitasking and multiprocessing operating system design, and in real-time operating system design. It refers to the way processes are assigned priorities in a priority queue. This assignment is carried out by software known as a scheduler. It consists of mainly, CPU utilization - to keep the CPU as busy as possible. Throughput - number of process that complete their execution per time unit. Turnaround- amount of time to execute a particular process. Waiting time - amount of time a process has been waiting in the ready queue. Response time - amount of time it takes from when a request was submitted until the first response is produced.

7.1.1 Types of Schedulers

Operating systems may feature up to 3 distinct types of schedulers: a long-term scheduler (also known as an admission scheduler or high-level), a mid-term or medium-term scheduler and a short-term scheduler (also known as a dispatcher). The names suggest the relative frequency with which these functions are performed.

7.1.1.1 Long-term Scheduler

The long-term, or admission, scheduler decides which jobs or processes are to be admitted to the ready queue; that is, when an attempt is made to execute a program, its admission to the set of currently executing processes is either authorized or delayed by the long-term scheduler. Thus, this scheduler dictates what processes are to run on a system and the degree of concurrency to be supported at any one time - ie: whether a high or low amount of processes are to be executed concurrently, and how the split between IO intensive and CPU intensive processes is to be handled. Typically for a desktop computer, there is no long-term scheduler as such, and processes are admitted to the system automatically. However this type of scheduling is very important for a real-time operating system, as the system's ability to meet process deadlines may be compromised by the slowdowns and contention resulting from the admission of more processes than the system can safely handle.

7.1.1.2 Mid-term Scheduler

The mid-term scheduler, present in all systems with virtual memory, temporarily removes processes from main memory and places them on secondary memory (such as a disk drive) or vice versa. This is commonly referred to as ”swapping out” or ”swapping in” (also incorrectly as ”paging out” or ”paging in”). The mid-term scheduler may decide to swap out a process which has not been active for some time, or a process which has a low priority, or a process which is page faulting frequently, or a process which is taking up a large amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is available, or when the process has been unblocked and is no longer waiting for a resource.

7.1.1.3 Short-term Scheduler

The short-term scheduler (also known as the dispatcher) decides which of the ready, in-memory processes are to be executed (allocated a CPU) next following a clock interrupt, an IO interrupt, an operating system call or another form of signal. Thus the short-term scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers - a scheduling decision will at a minimum have to be made after every time slice, and these are very short. This scheduler can be preemptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process, or non-preemptive, in which case the scheduler is unable to ”force” processes off the CPU.

7.2 First Come First Served (FCFS)

Perhaps, First-Come-First-Served algorithm is the simplest scheduling algorithm. Processes are dispatched according to their arrival time on the ready queue. Being a nonpreemptive discipline, once a process has a CPU, it runs to completion. The FCFS scheduling is fair in the formal sense or human sense of fairness but it is unfair in the sense that long jobs make short jobs wait and unimportant jobs make important jobs wait. FCFS is more predictable than most of other schemes since it offers time. FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. The code for FCFS scheduling is simple to write and understand. One of the major drawback of this scheme is that the average time is often quite long. The First-Come-First-Served algorithm is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

7.3 Round Robin Scheduling

Round-robin (RR) is one of the simplest scheduling algorithms for processes in an operating system, which assigns time slices to each process in equal portions and in order, handling all processes without priority. Round-robin scheduling is both simple and easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks.

Round-robin job scheduling may not be desirable if the size of the jobs or tasks are strongly varying. A process that produces large jobs would be favored over other processes. This problem may be solved by time-sharing, i.e. by giving each job a time slot or quantum (its allowance of CPU time), and interrupt the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process.

Example: The time slot could be 100 milliseconds. If a job1 takes a total time of 250ms to complete, the round-robin scheduler will suspend the job after 100ms and give other jobs their time on the CPU. Once the other jobs have had their equal share (100ms each), job1 will get another allocation of CPU time and the cycle will repeat. This process continues until the job finishes and needs no more time on the CPU.

Advantage: Fairness (each job gets an equal amount of the CPU) Disadvantage: Average waiting time can be bad (especially when the number of processes is large)

7.4 Shortest Job First (SJF)

Shortest Job First (SJF) is a scheduling policy that selects the waiting process with the smallest execution time to execute next. Shortest job first is advantageous because of its simplicity and because it maximizes process throughput (in terms of the number of processes run to completion in a given amount of time).

However, it has the potential for process starvation for processes which will require a long time to complete if short processes are continually added. Highest response ratio next is similar but provides a solution to this problem. Shortest job next scheduling is rarely used outside of specialized environments because it requires accurate estimations of the runtime of all processes that are waiting to execute. It is provably optimal with respect to minimizing the average waiting time. It works for both preemptive and non-preemptive schedulers.

7.4.1 Shortest Remaining Time First (SRTF)

Shortest remaining time first is a method of CPU scheduling that is a preemptive version of shortest job first scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time. Shortest remaining time is advantageous because short processes are handled very quickly. The system also requires very little overhead since it only makes a decision when a process completes or a new process is added, and when a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute. However, it has the potential for process starvation for processes which will require a long time to complete if short processes are continually added, though this threat can be minimal when process times follow a heavy-tailed distribution. Like shortest job first scheduling, shortest remaining time first scheduling is rarely used outside of specialized environments because it requires accurate estimations of the runtime of all processes that are waiting to execute.

7.5 Multilevel Feedback Queues (MLFQ)

Multilevel feedback queue-scheduling algorithm allows a process to move between queues. It uses many ready queues and associate a different priority with each queue. The Algorithm chooses to process with highest priority from the occupied queue and run that process either preemptively or non-preemptively. If the process uses too much CPU time it will moved to a lower-priority queue. Similarly, a process that wait too long in the lower-priority queue may be moved to a higher-priority queue may be moved to a highest-priority queue. Note that this form of aging prevents starvation. MLFQ use past behavior to predict the future and assign job priorities.

7.5.1 Adjusting Priorties in MLFQ

The idea is to favor I/O bound jobs over CPU bound jobs. This is for a better response time for I/O bound jobs. A job starts in the highest priority queue. If its time-slice expires, drop its priority one level. If the time-slice does not expire (the context switch comes from an I/O request instead), then increase its priority one level, up to the top priority level.

7.6 Lottery Scheduling

Lottery Scheduling is a probabilistic scheduling algorithm for processes in an operating system. Processes are each assigned some number of lottery tickets, and the scheduler draws a random ticket to select the next process. The distribution of tickets need not be uniform; granting a process more tickets provides it a relative higher chance of selection. This technique can be used to approximate other scheduling algorithms, such as Shortest job next and Fair-share scheduling. Lottery scheduling solves the problem of starvation. Giving each process at least one lottery ticket guarantees that it has non-zero probability of being selected at each scheduling operation. On average, CPU time is proportional to the number of tickets given to each job. For approximating SJF, most tickets are assigned to short running jobs, and fewer to longer runnning jobs. To avoid starvation, every job gets at least one ticket. Implementations of lottery scheduling should take into consideration that there could be a large number of tickets distributed among a large pool of threads. To have an array tickets with each ticket corresponding to a thread may be highly inefficient.