CPU scheduling is the basis of multi-programmed operating systems. By switching the CPU among different processes or threads, the operating system can make the computer more productive. This chapter introduces the basic scheduling concepts and discusses in great length CPU scheduling algorithms including FCFS, SJF, SRTF, Round-Robin, Priority, Multi-level (feedback) queue and the other scheduling algorithms.

Basic on Scheduling

- Each process or thread during its life time of execution typically goes through a series of *CPU bursts* and *I/O bursts*. Thus, a process can be modeled as switching between bursts of CPU execution and I/O wait
- **The length of the next CPU burst** is critical in some scheduling algorithms such as SJF and SRTF. The next CPU length can be estimated based on historical CPU burst lengths and by using an **Exponential Averaging** algorithm. The estimated (predicted) value for the next CPU burst length depends, 1) all actual CPU burst lengths of this process prior to the current estimation, 2) the initial estimation.
- There are many criteria or factors relevant to scheduling, they may be conflicting with one another, including (1) CPU utilization, which is calculated by the fraction of time that CPU is busy within a certain time duration; (2) throughput is defined as the number of processes completed per unit time, which is computed by the total number of processes completed divided by the time period; This metric apparently favors short process over long process; (3) turn-around time, all waiting time + all CPU burst times; if only one CPU burst time is given, the turn-around time is the waiting time plus the CPU burst time; (4) waiting time, the sum of all waiting times of a process; and (5) response time (between arrival time and the first response time typical in RR).
- Turnaround time is the sum of the periods that a process is spent on waiting, executing on the CPU, and doing I/O. It essentially measures the amount of time it takes to execute a process. Response time, on the other hand, is a measure of the time that elapses between a process request and the first response produced, typical the first CPU burst time or the end of first quantum if a RR is used.
- CPU scheduling is done for processes currently residing on the ready queue(s). The **dispatcher** gives control of the CPU to the process selected by the CPU scheduling. To perform this task, *a context switch*, a switch to user mode, and a jump to the proper location in the user program are all required. The dispatcher should be made as fast as possible. The time elapse is termed *dispatch latency*.
- **Non-preemptive** scheduling policy is evoked only when the current process running on the CPU gives up the CPU voluntarily either due to the termination of the process or the completion of its current CPU burst (for example waiting for I/O). Otherwise, the scheduling is **preemptive** in nature. For instance, scheduling might occur when a new process with higher priority joins the ready queue either as a new process or as a process from waiting to ready.

FCFS

- The scheduling is based solely on the arrival orders of the processes in the ready queue. Typically this is non-preemptive in that each process once occupying the CPU is allowed to run to the completion of its current CPU burst.
- The main problem with FCFS is that a short process might get stuck behind a long process. This phenomenon is referred as the **convoy effect**. This can result in excessively long average waiting time.

SJF

- Shortest Job First or SJF scheduling is based on the CPU burst lengths of the processes currently in the ready queue. This is proved to be the optimal scheduling algorithm (non-preemptive) with respect to the minimum average waiting time.
- Shortest Remaining Time First or SRTF is the preemptive version of the SJF, in which a newly arrived process (from new to ready) or a process just joining the ready queue (from waiting to ready) can preempt a process currently running on CPU if its CPU burst length is shorter than the remaining CPU burst time of the process running on the CPU. Please notice it is obvious that the CPU burst lengths of all processes in the ready queue are larger than the CPU burst length of the process in running state (occupying the CPU), that is the SJF part of the SRTF.

RR

- Round Robin scheduling is done typically based on FCFS order but limits each process to occupy the CPU no more than a prior-determined **quantum**. This allows each process to receive a fair share of CPU usage.
- By ignoring the context switching time overhead, RR scheduling results in better response time for each process, which is calculated as the time between the process arrival time to the ready queue and the completion of the first quantum for that process. The value of the quantum has to be carefully designed based on the characteristics of the CPU bursts of all processes.
- At one extreme, if the time quantum is extremely large, the RR policy is the same as the FCFS policy. If the time quantum is extremely small, the RR approach is called processor sharing and creates the appearance that each of n processes has its own processor running at 1/n the speed of the real processor (supposing that the context switch time is ignored).

The Estimation of Next CPU Burst – Exponential Average Algorithm

- Recall each process execution goes through many rounds of *CPU bursts* and *I/O burst*s during its life time. The estimation of the next CPU burst length is determined by the actual CPU burst length measured and estimated in the previous round.
- Simple Algebra can expand the formula, which states that the estimation is determined by the actual CPU burst lengths in all past rounds and the initial estimation value. The CPU burst lengths in the recent rounds weight more than those in earlier rounds, i.e., the exponential factor of 1-alpha.

Multi-level Queues

- Each queue can has its own scheduling algorithm.
- The CPU is allocated to different queues, typically either by priority or a percentage (time slice) of CPU time.
- A process is assigned permanently to one queue, it cannot be moved to a different queue during its life time of execution.

Multi-level Feedback Queues

- Multi-level feedback queue scheduling algorithm allows a process to move from one queue to another.
- There are several parameters that need to be specified for this scheduling algorithm, including, number of queues, which queue to join initially at a specific time, scheduling algorithm for each queue, and policy on how to move a process between different queues.
- **Starvation** occurs when a process is ready to run but is stuck waiting indefinitely for the CPU. This occurs, for example, when there are always higher-priority processes that prevent low-priority processes from acquiring the CPU. **Aging** involves gradually increasing the priority of a waiting process so that the process will eventually achieve a priority high enough to execute if it has waited for a long period of time.

Thread Scheduling

- On operation systems that support user-level and kernel-level threads, it is the kernel-level threads, not processes, which are being scheduled by the OS. User-level threads are managed by the thread library.
- The thread library schedules user-level threads to run on an available LWP. This scheme is known as **process-contention scope** (PCS). Typically PCS is done according to priority. The kernel uses **system-contention scope** (CSC) to determine which kernel-level thread to schedule onto a CPU.

Multiprocessor Scheduling

- In **asymmetric multiprocessing**, all scheduling decisions, I/O, and other system activities are handled by a single processor (the master server). The other processors execute only user codes. This is simple in that only one processor accesses the kernel data structure, reducing the need for data sharing.
- In **SMP** or **symmetric multiprocessing**, each processor is self-scheduling. All processes can be in a common ready queue or in a separate ready queue for each processor. It must ensure that two separate processors do not choose to schedule the same process and that processes are not lost from the queue(s).
- Two general method of load balancing in SMP systems: (1) **push migration**, a specific task periodically checks the load on each processor and — if it finds an imbalance—evenly distributes the load by moving processes from overloaded processors to idle or less-busy processors. (2) **Pull migration** occurs when an idle processor pulls a waiting task from a busy processor. Both push and pull

migration are often implemented in parallel on load-balancing systems

- There is a high cost such as contents of cache memory during the migration of processes from one processor to another. **Processor affinity** implies that a process has an affinity for the processor on which it is currently running. Some systems provide system calls to support **hard affinity**, thereby allowing a process to specify a subset of processors on which it may run. When an OS has a policy of attempting to keep a process running on the same processors, but not guaranteeing that, this situation is known as **soft affinity**.
- If an architecture features non-uniform memory access (**NUMA**), in which a CPU has faster access to some parts of main memory (for example, CPU and parts of the main memory are on the same board) than other parts, this could also result affinity to processor and main memory.

Algorithm Evaluation

- Analytical methods use mathematical analysis to determine the performance of a scheduling algorithm, which requires the exact input sequences are known. Queueing model captures the performance of a scheduling algorithm with a few known distribution. Simulation methods determine performance by imitating the scheduling algorithm on a "representative" sample of processes and computing the resulting performance. But simulation can at best provide an approximation of actual system performance.