CHAPTER 3 : CPU SCHEDULING

- In a single-processor system, only one process can run at a time.
- Others must wait until the CPU is free and can be rescheduled.
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- The idea is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request.
- In a simple computer system, the CPU then just sits idle. All this waiting time is wasted, no useful work is accomplished.
- With multiprogramming, several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues.
- Every time one process has to wait, another process can take over use of the CPU.
- **Scheduling** of this kind is a **fundamental operating-system** function.
- Almost all computer resources are scheduled before use.

CPU–I/O Burst Cycle

- The success of CPU scheduling depends on an observed property of processes:
- process execution consists of a **cycle** of CPU execution and I/O wait. Processes alternate between these two states.
- Process execution begins with a **CPU burst**. That is followed by an **I/O burst**, which is followed by another CPU burst, then another I/O burst, and so on.
- Eventually, the final CPU burst ends with a system request to terminate execution as shown in below fig.

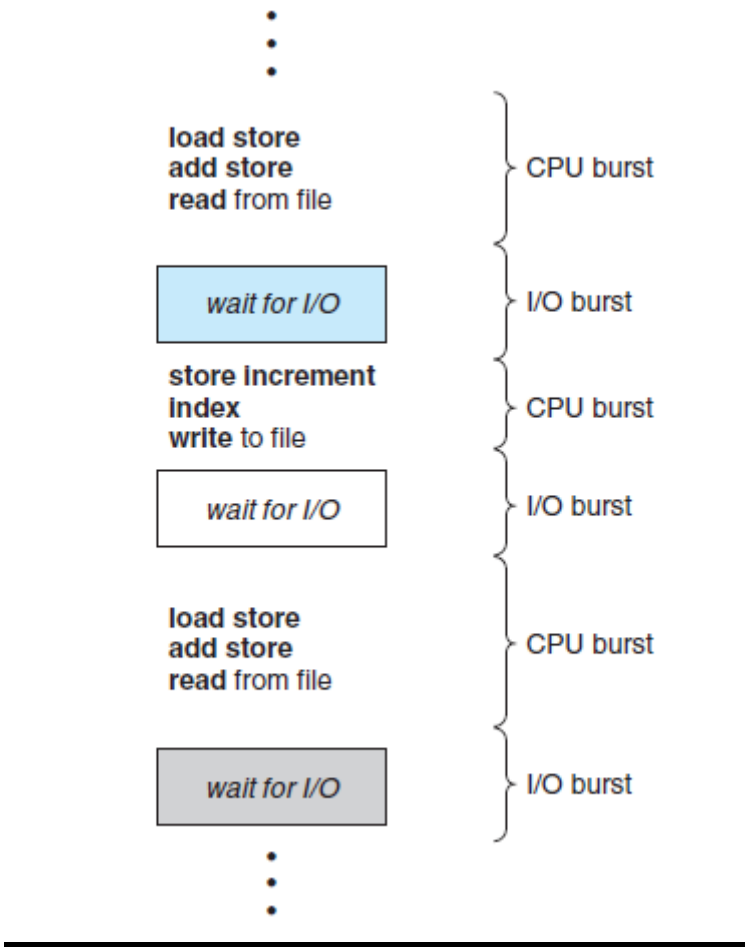

Fig: Alternating sequence of CPU and I/O bursts.

CPU Scheduler

- Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed.
- The selection process is carried out by the **short-term scheduler**, or CPU scheduler.
- The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Preemptive Scheduling

- Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state.
- The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time

remaining. That process stays in ready queue till it gets next chance to execute.

- Algorithms based on preemptive scheduling are : Round Robin (RR), Shortest Job First (SJF basically non preemptive) and Priority (non preemptive version), etc.

- CPU-scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process)
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs)
3. When a process switches from the waiting state to the ready state (for example, at completion of I/O)
4. When a process terminates

Non pre-emptive Scheduling :

- Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state.
- In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.
- In case of non-preemptive, scheduling does not interrupt a process running CPU in middle of the execution.
- Instead, it waits till the process complete its CPU burst time and then it can allocate the CPU to another process.

Difference between Pre-emptive and Non-Preemptive Scheduling

Non Preemptive Scheduling	Preemptive Scheduling
Once the Cpu is given to a process , it cannot be taken away from that process	The cpu can be taken away
Shorter jobs must wait for completion of longer jobs.	Shorter jobs needs to wait
Cost is low	Cost is high.
Overheads are low	Overheads are high due to storage of non-running programs in the main memory.
Suitable for batch processing	Suitable for real time and interactive timesharing systems.
It occurs when the process either switches from running state to waiting state or when it terminates.	It occurs when the process either switches from running state to ready state or from waiting state to ready state.
There is no need of Context Switching	Context switching becomes necessary when a process is pre-empted and a new process has to be scheduled to the cpu.
Job is completed according to the allocated time.	Completion time of the process in execution cannot be computed accurately.
Scheduling is done once.	Rescheduling is necessary
If an interrupt occurs, the process is terminated.	When an interrupt occurs the process is temporarily suspended to be resumed later.

Dispatcher

- Another component involved in the CPU-scheduling function is the **dispatcher**.
- The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.
- This function involves the following:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

Scheduling Criteria

✓ **CPU utilization**. We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).

- ✓ **Throughput** : Measure of work is the number of processes that are completed per time unit, called **throughput**. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.

- ✓ **Turnaround time** : Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

- ✓ **Waiting time** : Waiting time is the sum of the periods spent waiting in the ready queue.

- ✓ **Response time** : The time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response.

Scheduling Algorithms

1. First Come First Serve (FCFS)

- First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm.
- FIFO simply queues processes in the order that they arrive in the ready queue.
- In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

Important Points:

1. Non-preemptive
2. Average Waiting Time is not optimal
3. Cannot utilize resources in parallel: Results in **Convoy effect**

What is Convoy Effect?

Convoy Effect is a situation where many processes, who need to use a resource for short time are blocked by one process holding that resource for a long time.

This essentially leads to poor utilization of resources and hence poor performance.

2. Shortest-Job-First Scheduling

- A different approach to CPU scheduling is the **shortest-job-first (SJF)** scheduling algorithm.
- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.
- The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes. Moving a short process before a long one decreases the waiting time of the short process more than it increases the waiting time of the long process.
- The average waiting time decreases.
- The real difficulty with the SJF algorithm is knowing the length of the next CPU request.
- **A preemptive SJF algorithm** will preempt the currently executing process, whereas a **nonpreemptive SJF** algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first.**

3. Priority Scheduling

- The SJF algorithm is a special case of the general **priority-scheduling** algorithm.
- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.
- scheduling in terms of ***high*** priority and ***low*** priority.
- Priority scheduling can be either preemptive or nonpreemptive.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
- A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

- A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.
- A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**. A process that is ready to run but waiting for the CPU can be considered blocked.
- A solution to the problem of indefinite blockage of low-priority processes is **aging**.
- Aging involves gradually increasing the priority of processes that wait in the system for a long time.

4. Round-Robin Schedulig(RR)

- The **round-robin (RR)** scheduling algorithm is designed especially for timesharing systems.
- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- A small unit of time, called a **time quantum** or **time slice**, is defined.
- A time quantum is generally from 10 to 100 milliseconds in length.
- The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- To implement RR scheduling, we again treat the ready queue as a FIFO queue of processes.
- New processes are added to the tail of the ready queue.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

NOTE :

Problems : Refer class notes

5. Multilevel Queue Scheduling

- Processes are easily classified into different groups
- common division is made between **foreground** (interactive) processes and **background** (batch) processes.
- These two types of processes have different response-time requirements and so may have different scheduling needs.

- In addition, foreground processes may have priority (externally defined) over background processes.

A **multilevel queue** scheduling algorithm partitions the ready queue into several separate queues (Figure 6.6). The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm

Let's look at an example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority:

1. System processes
2. Interactive processes
3. Interactive editing processes
4. Batch processes
5. Student processes

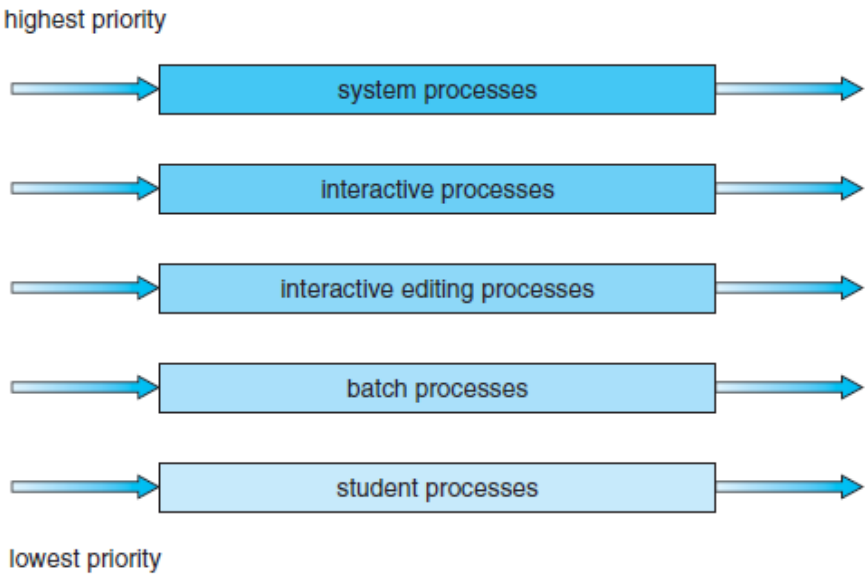

Fig : Multi level queue scheduling.

when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system. If there are separate queues for foreground and background processes, for example, processes do not move from one queue to the other, since processes do not change their foreground or background nature. This setup has the advantage of low scheduling overhead, but it is inflexible.to overcome this we have multilevel feedback queue.

✓ Multilevel Feedback Queue

- The **multilevel feedback queue** scheduling algorithm, allows a process to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
- In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.
- For example, consider a multilevel feedback queue scheduler with three queues, numbered from 0 to 2 (Figure a).

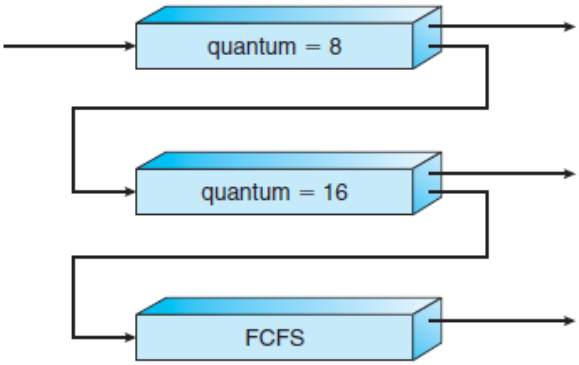

Fig a : Multilevel feedback queue

- The scheduler first executes all processes in queue 0.
- Only when queue 0 is empty will it execute processes in queue 1.
- Similarly, processes in queue 2 will be executed only if queues 0 and 1 are empty.
- A process that arrives for queue 1 will preempt a process in queue 2.
- A process in queue 1 will in turn be preempted by a process arriving for queue 0.
- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds.
- If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds.
- If it does not complete, it is preempted and is put into queue 2.

- Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.
- This scheduling algorithm gives highest priority to any process with a CPU burst of 8 milliseconds or less.
- Such a process will quickly get the CPU, finish its CPU burst, and go off to its next I/O burst.
- Processes that need more than 8 but less than 24 milliseconds are also served quickly, although with lower priority than shorter processes.
- Long processes automatically sink to queue 2 and are served in FCFS order with any CPU cycles left over from queues 0 and 1.

✓ In general, a multilevel feedback queue scheduler is defined by the following parameters

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher priority queue.
- The method used to determine when to demote a process to a lower priority queue.
- The method used to determine which queue a process will enter when that process needs service.