MODULE 2

CHAPTER 2 – PROCESS SCHEDULING

CPU SCHEDULING

- The objective of multiprogramming is to have some process running at all times, in order to maximize CPU utilization.
- In a uniprocessor system, only one process may run at a time; any other processes must wait until the CPU is free. With multiprogramming, we try to use this time productively.
- 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.
- CPU scheduling is the process of determining which process will actually run when there are multiple runnable processes.

- For attaining the objectives of multi programming, there must be required proper CPU scheduling.
- 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, then another CPU burst, then another I/O burst, and so on.
- Eventually, the last CPU burst will end with a system request to terminate execution.

❖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 from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- A ready queue may be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list.
- All the processes in the ready queue are lined up waiting for a chance to run on the CPU. The records in the queues are generally process control blocks (PCBs) of the processes.

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

1. When a process switches from the running state to the waiting state (I/O request)

2. When a process switches from the running state to the ready state (when an interrupt occurs)

3. When a process switches from the waiting state to the ready state (for example, completion of I/O)

4. When a process terminates

- In circumstances 1 and 4, CPU is in idle state. So if a process is available in ready queue, the process must be selected for the execution.

- In 2 and 3, CPU is in working state, which means that CPU is not free.
- When the scheduling take place only under circumstances 1 and 4, we say the scheduling is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.
- Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

❖**Non-preemptive scheduling:** Once the CPU has been allocated to a process, the process keeps the CPU until its termination or its transition to the blocked state. This means that once CPU is allocated to a process, this process can use the CPU for its own execution till it willingly surrenders or leave the CPU.

❖**Preemptive scheduling:** Here, even if the CPU has been allocated to a certain process, it can be snatched from this process any time either due to time constraint or due to priority reasons. It implies that if a process with a higher priority becomes ready for its execution, the process which is currently using the CPU will be forced to give up the CPU so that higher priority job has run fast.

❖**Preemptive scheduling problems:**

- Preemptive scheduling is costly as compared to non-preemptive scheduling. Consider the case of two processes sharing data. One may be in the midst of updating the data when it is preempted and the second process is run. The second process may try to read the data, which are currently in an inconsistent state.
- Preemption also has an effect on the design of the operating-system kernel. During the processing of a system call, the kernel may be busy with an activity on behalf of a process. Such activities may involve changing important kernel data (for instance, I/O queues). If scheduling is done in kernel level, it modifies the important codes of kernel. So preemption in kernel must be avoided.

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:

✓ **Switching context:** Switching of CPU from one process to another.

✓ **Switching to user mode:** When the system is in kernel mode, dispatcher switches the kernel mode into user mode and vice versa.

✓ Jumping to the proper location in the user program to restart that program

- The dispatcher should be as fast as possible, given that it is invoked during every process switch.

❖ **Dispatcher latency**

- The time it takes for the dispatcher to stop one process and start another process execution is known as the dispatch latency.

SCHEDULING CRITERIA

- **CPU utilization:** We want to keep the CPU as busy as possible. CPU utilization may range from to 100 percent.
- **Throughput:** It is the number of processes completed per time unit (No. of process completed /second). For long processes, this rate may be 1 process per hour; for short transactions, throughput might be 1 process per second.
- **Turnaround time:** The interval from the time of submission of a process to the time of completion is the 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.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 66

- **Waiting time:** Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response time:** it is the time from the submission of a request until the first response is produced.

➢We want to

- **Maximize** CPU utilization and throughput
- **Minimize** turnaround time, waiting time, and response time.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 67

SCHEDULING ALGORITHMS

1. First-Come, First-Served Scheduling (FCFS)

- It is the simplest of all the scheduling algorithms. Key concept of this algorithm is —allocate the CPU in the order in which the processes arrive. It assumes that ready queue is managed as FIFO (First in first out).This algorithm is non-preemptive.
- Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

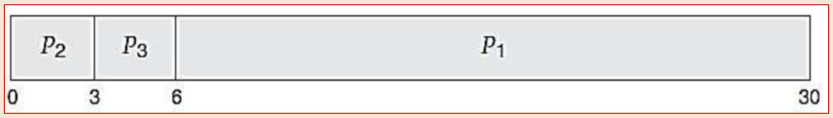
Process	Burst Time
P_1	24
P_2	3
P_3	3

- If the processes arrive in the order P1, P2, P3, and are served in FCFS order, we get the result shown in the following **Gantt chart**, which is a bar chart that illustrates a particular schedule, including the start and finish times of each of the participating processes:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The waiting time is 0 milliseconds for process P1, 24 milliseconds for process P2, and 27 milliseconds for process P3.
- Thus, the average waiting time is **(0 + 24 + 27)/3 = 17 milliseconds**.

- If the processes arrive in the order P2, P3, P1 the results will be as shown in the following Gantt chart



Process	Burst Time
P_1	24
P_2	3
P_3	3

- The average waiting time is now **(6 + 0 + 3)/3 = 3 milliseconds.**
- This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the processes‘ CPU burst times vary greatly.

2. Shortest-Job-First Scheduling (SJF)

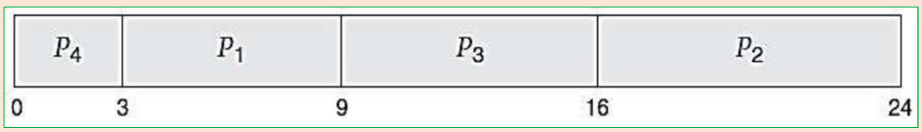
- The key concept of this algorithm is: —CPU is allocated to the process with least CPU burst time.
- Amongst the processes in the ready queue, CPU is always assigned to the process with the least CPU burst requirement.
- If there are two processes with the CPU burst, the one which arrived first, will be taken up first by the CPU.
- This algorithm can be either preemptive or non-preemptive.
- Preemptive SJF scheduling is sometimes called Shortest-Remaining-Time-First (SRTF) scheduling.

- SJF is an optimal algorithm, as it gives the minimum average waiting time.

❖**Example**

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- Using SJF scheduling, we would schedule these processes according to the following Gantt chart:



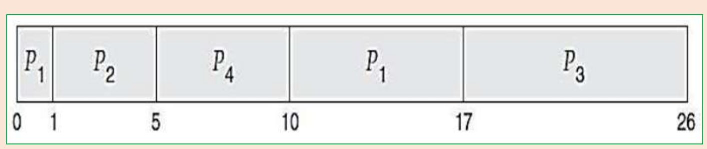
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 72

- The waiting time is 3 milliseconds for process P1, 16 milliseconds for process P2, 9 milliseconds for process P3, and 0 milliseconds for process P4.
- The average waiting time is **(3 + 16 + 9 + 0)/4 = 7** milliseconds.

❖**Shortest-Remaining-Time-First (SRTF)**

Consider the following four processes, with the length of the CPU burst given in milliseconds:

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 73

- Process P1 is started at time 0, since it is the only process in the queue.
- Process P2 arrives at time 1. The remaining time for process P1 (7 milliseconds) is larger than the time required by process P2 (4 milliseconds), so process P1 is preempted, and process P2 is scheduled.
- The average waiting time for this example is [(10 − 1) + (1 − 1) + (17 − 2) + (5 − 3)]/4 = 26/4 = 6.5 milliseconds.
- Non-preemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

3. Priority Scheduling

- The CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.
- Consider the following set of processes, assumed to have arrived at time 0 in the order P1, P2, · · ·, P5, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- The average waiting time is 8.2 milliseconds.

- Priorities can be assigned in two ways:

❖**Internal assignment:**

- It uses some measurable quantity or quantities to compute the priority of a process.
- For example, time limits, memory requirements, the number of open files, and the ratio of average I/O burst to average CPU burst have been used in computing priorities.

❖**External Assignment:**

- It is set by criteria that are external to the operating system, such as the importance of the process, the type and amount of funds being paid for computer use, the department sponsoring the work, and other, often political, factors.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 76

- Priority scheduling can be either preemptive or non-preemptive.
- 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 major problem with priority-scheduling algorithms is indefinite blocking (or **starvation**) - higher-priority processes can prevent a low-priority process from ever getting the CPU.
- A solution to the problem of indefinite blockage of low-priority processes is **aging**. When a process is waiting for a long time, the OS gradually increasing the priority of processes that wait in the system for a long time, and taken it for execution.
- **Starvation** is avoided by **aging.**

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 77

4. Round-Robin Scheduling

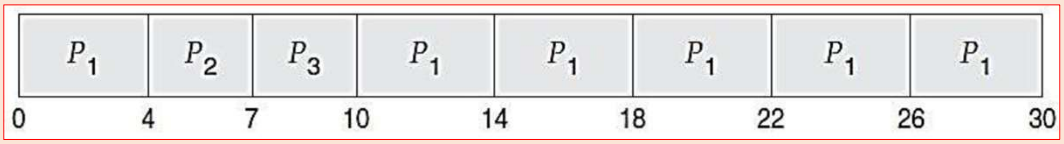
- The round-robin (RR) scheduling algorithm is designed especially for time- sharing systems for getting high responsiveness.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time, called a time quantum (or time slice), is defined. The ready queue is treated as a circular queue.
- To implement RR scheduling, we keep 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.

- The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue.
- If the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system.
- A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.
- **The RR scheduling algorithm is preemptive**.

Eg: Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds.

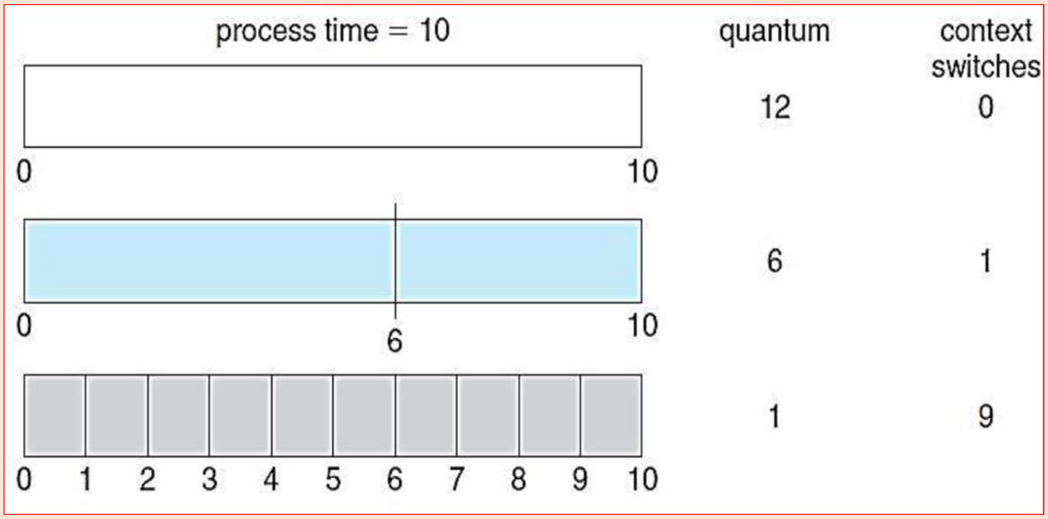
Process	Burst Time
P_1	24
P_2	3
P_3	3

- time quantum of **4** milliseconds



- Let's calculate the average waiting time for this schedule.
- P1 waits for 6 milliseconds (10 - 4)
- P2 waits for 4 milliseconds
- P3 waits for 7 milliseconds.
- Thus, the average waiting time is **17/3 = 5.66 milliseconds**.
- The performance of the RR algorithm depends heavily on the size of the time quantum.
- If the time quantum is very large (infinite), the RR policy is the same as the FCFS policy.
- If the time quantum is very small (say 1 microsecond), the RR approach is called processor sharing

Figure: Showing how a smaller time quantum increases context switches.



- If the quantum is 12 time units, the process finishes in less than 1 time quantum, with no overhead.
- If the quantum is 6 time units, however, the process requires 2 quanta, resulting in 1 context switch. If the time quantum is 1 time unit, then 9 context switches will occur, slowing the execution of the process accordingly.
- If the context-switch time is approximately 10 percent of the time quantum, then about 10 percent of the CPU time will be spent in context switch.
- Turnaround time also depends on the size of the time quantum.
- When time quantum increases, the turnaround time increases and context switching decreases.

5. Multilevel Queue Scheduling

- Processes are easily classified into different groups. Foreground and background processes.
- Foreground (or interactive) processes are directly interact with users. In background (or batch) processes, no user interaction is occurred.
- Foreground processes have high priority, but interactive processes have high responsiveness.
- A multilevel queue-scheduling algorithm partitions the ready queue into several separate queues.

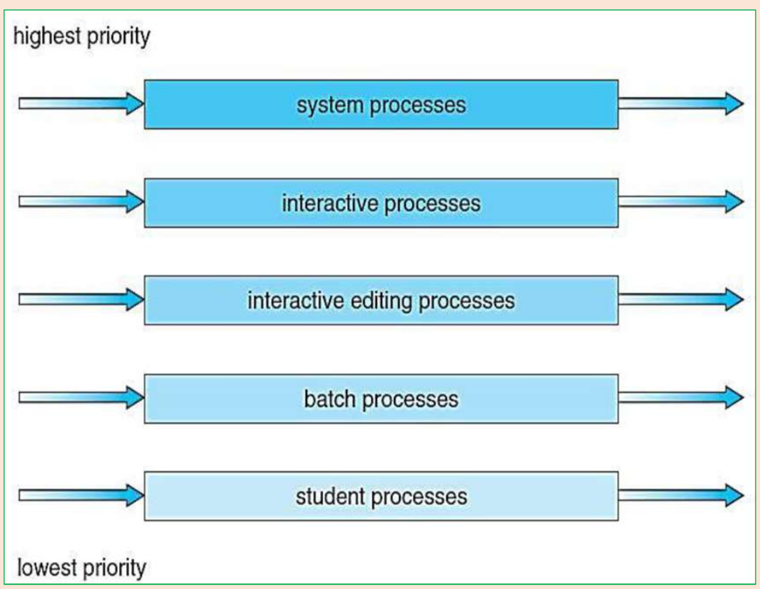


Figure: Multilevel queue scheduling

- Here ready queue is divided in to 5 queues. Each one is priority queue. If a process comes, the nature of that process is determined, and assigns it to the queue.
- If a process is assigned to a particular queue, that process cannot move in to another queue. That is a fixed priority is given to the incoming process.
- Key concept of this algorithm is —Multi level queue scheduling was created for situations in which processes are easily classified in to different groups
- Each queue has its own scheduling algorithm. For example, the foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.

- Here starvation problem may occur.
- For avoiding starvation, time slicing principle is used.
- Each queue has given a particular time quantum, and the times quantum is again sliced and distribute it to the processes reside in that queue.

6. Multilevel Feedback Queue Scheduling

- In multi level queue scheduling, processes are permanently assigned to a queue on entry to the system. Processes do not move from one queue to the other, since processes do not change their foreground or background nature.
- For avoiding starvation Multilevel feedback queue scheduling, allows a process to move between queues.
- The idea is to separate processes with different CPU-burst characteristics.
- 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.

- Similarly, 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.

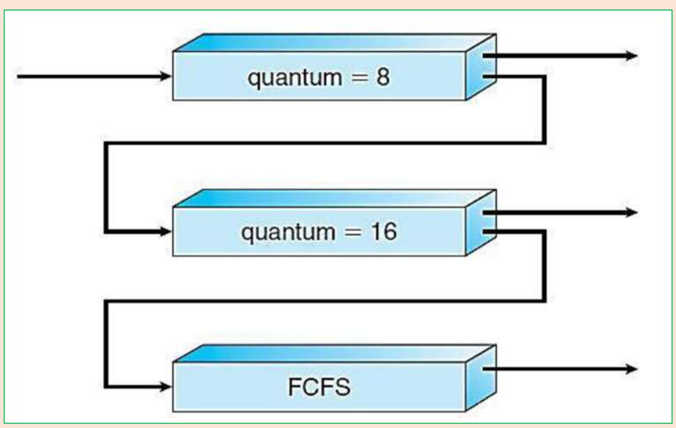


Figure: Multilevel feedback queues

- The processes are classifies on the basis of CPU burst time.
- Highest priority is given to the I/O bound processes and interactive processes.
- The incoming processes are entered in Q0. In Q0, each process has same priority.
- The first process in Q0 is taken and allocates CPU for execution. After completing 8 milliseconds, the process is preempted and it is added to the end of Q1.
- After completing one turn in Q0, the processes are taken from Q1. After completing 16 time quantum, next go to FCFS.
- Last performs FCFS.

7. Multiple-Processor Scheduling

- Load balancing is possible in multiple processor system. In multiple processors scheduling, two cases arises.

✓Each processor has its own ready queue.

✓A common ready queue is available.

- In the first case, the load balancing concept does not occur.
- Two types of multiprocessors are asymmetric and symmetric multiprocessors. Load sharing is only possible in symmetric multi-processor.

❖ Processor Affinity

- consider what happens if the process migrates to another processor.
- The contents of cache memory must be invalidated for the first processor, and the cache for the second processor must be repopulated.
- Because of the high cost of invalidating and repopulating caches, most SMP systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor.
- This is known as **processor affinity**—that is, a process has an affinity for the processor on which it is currently running.

➢ Two types of processor affinities are:

- **Soft:** It is not a strict one. The process can be moved into another processor on a critical situation.
- **Hard:** It is very strict. Here a particular process must be executed in a particular processor.

❖ Load balancing

- Load balancing must be done when separate ready queues are implemented. Two types of load balancing are:
- **Push migration:** Here, a dedicated process is check the CPU periodically. If the CPU is in a waiting state, the checking process takes two or more processes from a queue that have more processes, and push them in to the waiting CPU's ready queue.
- **Pull migration:** Here, the idle processor takes processes from a busy CPU.