PANIHATI MAHAVIDYALAYA

Lecture Notes On

CPU SCHEDULING

Prepared by,
Prof. Biswajit Patwari,
M.Tech in CSE,
Department of Computer Science

 # lecture-1 #

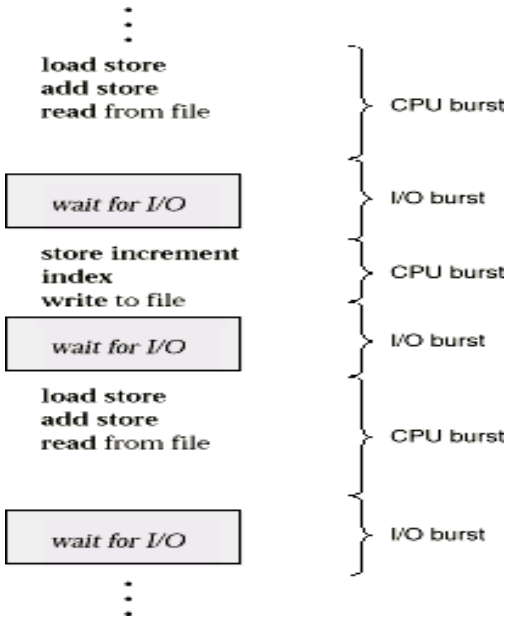
CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.
- CPU burst distribution

Alternating Sequence of CPU And I/O Bursts



CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
    1. Switches from running to waiting state.
    2. Switches from running to ready state.
    3. Switches from waiting to ready.
    4. Terminates.

Dispatcher

Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:

switching context

switching to user mode
Jumping to the proper location in the user program to restart that program

Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

Why do we need scheduling?

A typical process involves both I/O time and CPU time. In a uniprogramming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

Various criteria of good scheduling algorithm are:

- **CPU Utilization** − A scheduling algorithm should be designed so that CPU remains busy as possible. It should make efficient use of CPU.
- **Throughput** − Throughput is the amount of work completed in a unit of time. In other words throughput is the processes executed to number of jobs completed in a unit of time. The scheduling algorithm must look to maximize the number of jobs processed per time unit.
- **Response time** − Response time is the time taken to start responding to the request. A scheduler must aim to minimize response time for interactive users.
- **Turnaround time** − Turnaround time refers to the time between the moment of submission of a job/ process and the time of its completion. Thus how long it takes to execute a process is also an important factor.
- **Waiting time** − It is the time a job waits for resource allocation when several jobs are competing in multiprogramming system. The aim is to minimize the waiting time.
- **Fairness** − A good scheduler should make sure that each process gets its fair share of the CPU.

Objectives of Process Scheduling Algorithm:

Max CPU utilization [Keep CPU as busy as possible]
Fair allocation of CPU.
Max throughput [Number of processes that complete their execution per time unit]
Min turnaround time [Time taken by a process to finish execution]
Min waiting time [Time a process waits in ready queue]
Min response time [Time when a process produces first response]

Optimization Criteria
Max CPU utilization
Max throughput
Min turnaround time
Min waiting time
Min response time

 # lecture-2 #

Different Scheduling Algorithms:

1. 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 Remaining Time First (SRTF), Priority (preemptive version), etc.

2. Non-Preemptive 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.
Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non preemptive) and Priority (non preemptive version), etc.

Key Differences Between Preemptive and Non-Preemptive Scheduling:

- In preemptive scheduling the CPU is allocated to the processes for the limited time whereas in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting state.
- The executing process in preemptive scheduling is interrupted in the middle of execution when higher priority one comes whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution and wait till its execution.
- In Preemptive Scheduling, there is the overhead of switching the process from ready state to running state, vise-verse, and maintaining the ready queue. Whereas in case of non-preemptive scheduling has no overhead of switching the process from running state to ready state.
- In preemptive scheduling, if a high priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. On the other hands, in the non-preemptive scheduling, if CPU is allocated to the process having larger burst time then the processes with small burst time may have to starve.
- Preemptive scheduling attain flexible by allowing the critical processes to access CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.
- The Preemptive Scheduling has to maintain the integrity of shared data that's why it is cost associative as it which is not the case with Non-preemptive Scheduling.

First-Come, First-Served Scheduling:

By far the simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. The code for FCFS scheduling is simple to write and understand. The average waiting time under the FCFS policy, however, is often quite long.

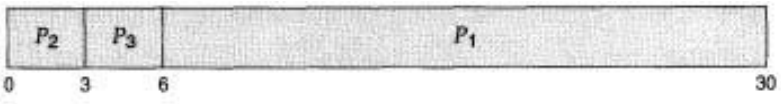
Consider the following set of processes that arrive at time 0, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time
PI	24
P2	3
P3	3

If the processes arrive in the order PI, P2, P3, and are served in FCFS order, we get the result shown in the

following Gantt chart:

The waiting time is 0 milliseconds for process PI, 24 milliseconds for process PZ, 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, Pl, however, the results will be as shown in the following Gantt chart:



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

In addition, consider the performance of FCFS scheduling in a dynamic situation. Assume we have one CPU-bound process and many I/O-bound processes. As the processes flow around the system, the following scenario may result. The CPU-bound process will get the CPU and hold it. During this time, all the other processes will finish their I/O and move into the ready queue, waiting for the CPU. While the processes wait in the ready queue, the I/O devices are idle. Eventually, the CPU-bound process finishes its CPU burst and moves to an I/O device. All the I/O-bound processes, which have very short CPU bursts, execute quickly and move, back to the I/O queues. At this point, the CPU sits idle. The CPU-bound process will then move back to the ready queue and be allocated the CPU. Again, all the I/O processes end up waiting in the ready queue until the CPU-bound process is done. There is a convoy effect, as all the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

The FCFS scheduling algorithm is non-preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O. The FCFS algorithm is particularly troublesome for time-sharing systems, where each user needs to get a share of the CPU at regular intervals. It would be disastrous to allow one process to keep the CPU for an extended period.

 # lecture-3 #

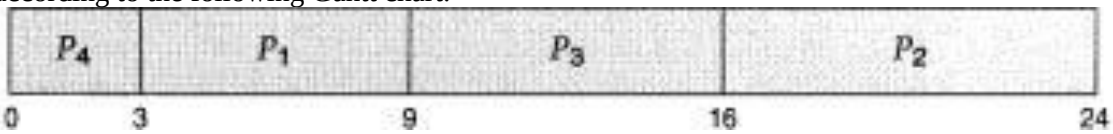
Shortest-Job-First Scheduling:

A different approach to CPU scheduling is the shortest-job-first (SJF) scheduling algorithm. This algorithm associates with each process the length of the latter's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. Note that a more appropriate term would be the shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of a process, rather than its total length. We use the term SJF because most people and textbooks refer to this type of scheduling discipline as SJF.

As an example, consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time
PI	6
p2	8
p3	7
p4	3

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



The waiting time is 3 milliseconds for process PI, 16 milliseconds for process P2,9 milliseconds for process PS, and 0 milliseconds for process Pq. Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = 7 milliseconds. If we were using the FCFS scheduling scheme, then the average waiting time would be 10.25 milliseconds.

The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes. By moving a short process before a long one, the waiting time of the short process decreases more than it increases the waiting time of the long process. Consequently, the average waiting time decreases.

The real difficulty with the SJF algorithm is knowing the length of the next CPU request. For long-term (or job) scheduling in a batch system, we can use as the length the process time limit that a user specifies when he submits the job.

Thus, users are motivated to estimate the process time limit accurately, since a lower value may mean faster response. (Too low a value will cause a time-limit exceeded error and require resubmission.) SJF scheduling is used frequently in long-term scheduling.

Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU scheduling. There is no way to know the length of the next CPU burst. One approach is to try to approximate SJF scheduling. We may not know the length of the next CPU burst, but we may be able to predict its value. We expect that the next CPU burst will be similar in length to the previous ones.

Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

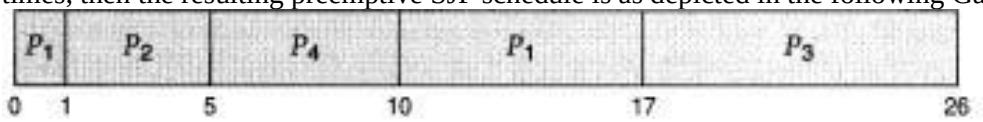
The SJF algorithm may be either preemptive or nonpreemptive. The choice arises when a new process arrives at the ready queue while a previous process is executing. The new process may have a shorter next CPU burst than what is left of the currently executing process. A preemptive SJF algorithm will preempt 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 scheduling.

As an example, consider the following four processes, with the length of the CPU-burst

time given in milliseconds:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
p4	3	5

If the processes arrive at the ready queue at the times shown and need the indicated burst times, then the resulting preemptive SJF schedule is as depicted in the following Gantt chart:



Process PI is started at time 0, since it is the only process in the queue. Process PZ 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. A nonpreemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

 # lecture-4 #

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.

An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa.

Note that we discuss scheduling in terms of high priority and low priority. Priorities are generally some fixed range of numbers, such as 0 to 7, or 0 to 4,095. However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion. In this text, we use low numbers to represent high priority. As an example, consider the following set of processes, assumed to have arrived at time 0, in the order PI, P2,
..., Ps, with the length of the CPU-burst time given in milliseconds:

Process	Burst	Time Priority
P1	10	3
p2	1	1
p3	2	4
P4	1	5
P5	5	2

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

The average waiting time is 8.2 milliseconds. Priorities can be defined either internally or externally.

Internally defined priorities use 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 priorities are 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.

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 lacking the CPU can be considered blocked-waiting for the CPU. A priority-scheduling algorithm can leave some low-priority processes waiting indefinitely for the CPU.A solution to the problem of indefinite blockage of low-priority processes are aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priorities range from 127 (low) to 0 (high), we could decrement the priority of a waiting process by 1 every 15 minutes. Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed. In fact, it would take no more than 32 hours for a priority 127 process to age to a priority 0 process.

 # lecture-5 #

Round-Robin Scheduling:

The round-robin (RR) scheduling algorithm is designed especially for timesharing systems. 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. A time quantum is generally from 10 to 100 milliseconds. 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 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.

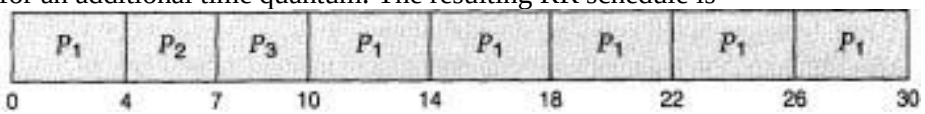
One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is 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 average waiting time under the RR policy, however, is often quite long.

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

Process	Burst Time
PI	24
P2	3
P3	3

If we use a time quantum of 4 milliseconds, then process P1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P2. Since process P2 does not need 4 milliseconds, it quits before its time quantum expires. The CPU is then given to the next process, process P3. Once each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum. The resulting RR schedule is



The average waiting time is 17/3 = 5.66 milliseconds.

In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row. If a process' CPU burst exceeds 1 time quantum, that process is preempted and is put back in the ready queue. The RR scheduling algorithm is preemptive.

If there are n processes in the ready queue and the time quantum is q, then each process gets l/n of the CPU time in chunks of at most q time units. Each process must wait no longer than (n - 1) x q time units until its next time quantum. For example, if there are five processes, with a time quantum of 20 milliseconds, then each process will get up to 20 milliseconds every 100 milliseconds.

The performance of the RR algorithm depends heavily on the size of the time quantum. At one extreme, 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, and appears (in theory) to the users as though each of n processes has its own processor running at l/n the speed of the real processor. This approach was used in Control Data Corporation (CDC) hardware to implement 10 peripheral processors with only one set of hardware and 10 sets of registers.

 # lecture-6 #

Multilevel Queue Scheduling:

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups. For example, a common division is made between foregrounds (or interactive) Processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority (or externally defined) over background processes.

Highest Response Ratio Next (HRRN):

In this scheduling, processes with highest response ratio are scheduled. This algorithm avoids starvation.

Response Ratio = (Waiting Time + Burst time) / Burst time

Multilevel Queue Scheduling:

According to the priority of process, processes are placed in the different queues. Generally high priority processes are placed in the top level queue. Only after completion of processes from top level queue, lower level queued processes are scheduled. It can suffer from starvation.

Multi level Feedback Queue Scheduling:

It allows the process to move in 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 is moved to a lower-priority queue.

Some useful facts about Scheduling Algorithms:

- FCFS can cause long waiting times, especially when the first job takes too much CPU time.
- Both SJF and Shortest Remaining time first algorithms may cause starvation. Consider a situation when the long process is there in the ready queue and shorter processes keep coming.
- If time quantum for Round Robin scheduling is very large, then it behaves same as FCFS scheduling.
- SJF is optimal in terms of average waiting time for a given set of processes.

 # lecture-7 #

Problem-01:

Consider the following table of arrival time and burst time for three processes P0, P1 and P2.

Process	Arrival time	Burst Time
P0	0 ms	9 ms
P1	1 ms	4 ms
P2	2 ms	9 ms

The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?

Problem-02:

Consider the set of processes with arrival time(in milliseconds), CPU burst time (in milliseconds), and priority(0 is the highest priority) shown below. None of the processes have I/O burst time.

Process	Arrival Time	Burst Time	Priority
P_1	0	11	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is

Problem-03:

Consider the following set of processes with corresponding arrival times and burst times:

Process	Arrival Time (units)	CPU Burst Time (Units)
P1	**0**	**6**
P2	**3**	**10**
P3	**5**	**8**
P4	**7**	**5**
p5	**10**	**6**

Draw the Gantt chart considering the Round Robin scheduling policy with time quantum = 4 units. Calculate individual turnaround time and average waiting time.