Scheduling

1. Introduction

In a multiprogramming system, frequently multiple process competes for the CPU at the same time. When two or more process are simultaneously in the ready state a choice has to be made which process is to run next. This part of the operating system is called Scheduler and the algorithm is called scheduling algorithm. Process execution consists of cycles of CPU execution and I/O wait. Processes alternate between these two states. Process execution begins with a CPU burst that is followed by 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.

Long term and short term scheduling:

If we consider batch systems, there will often be more processes submitted than the number of processes that can be executed immediately. So, incoming processes are spooled (to a disk).

The long-term scheduler selects processes from this process pool and loads selected processes into memory for execution. **The short-term scheduler** selects the process to get the processor from among the processes which are already in memory.

The short-time scheduler will be executing frequently (mostly at least once every 10 milliseconds). So it has to be very fast in order to achieve better processor utilization. Time-sharing systems (mostly) have no long-term scheduler. The stability of these systems either depends upon a physical limitation (number of available terminals) or the self-adjusting nature of users (if you can't get response, you quit). It can sometimes be good to reduce the degree of multiprogramming by removing processes from memory and storing them on disk. These processes can then be reintroduced into memory by the medium-term scheduler. This operation is also known as swapping. Swapping may be necessary to improve the process mix or to free memory.

Process Behavior:

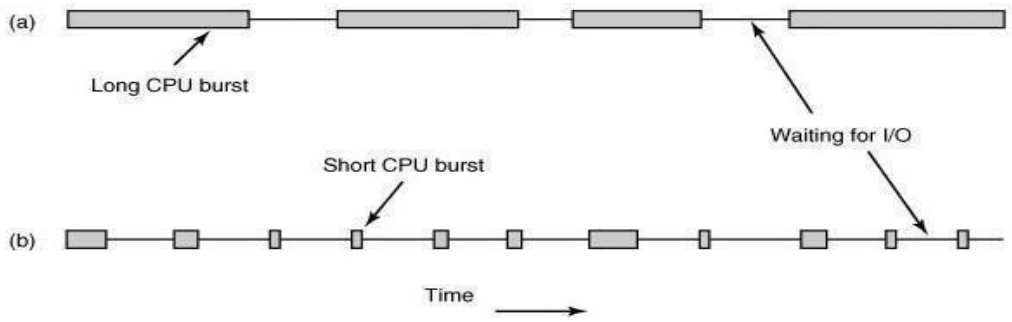


Fig: Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O bound process

- **CPU Bound (or compute bound):** Some processes such as the one shown fig a. above spend most of their time computing. These processes tend to have long CPU burst and thus infrequent I/O waits. example: Matrix multiplication
- **I/O Bound:** Some process such as the one shown in fig. b above spend most of their time waiting for I/O. They have short CPU bursts and thus frequent I/O waits. example: Firefox

2. Types of Scheduling:

I. **Non-preemptive scheduling algorithm:** It picks a process to run and then just lets it run until it blocks (either on I/O or waiting for another process) or until it voluntarily releases the CPU. Even it runs for hours, it will not be forcibly suspended. In effect no scheduling decisions are made during clock interrupts.

II. **Preemptive scheduling algorithm:** It picks a process and lets it run for a maximum of some fixed time. If it is still running at the end of the time interval, it is suspended and the scheduler picks another process to run (if one is available). Doing preemptive scheduling requires having a clock interrupt occur at the end of time interval to give control of the CPU back to the scheduler. If no clock is available, non-preemptive scheduling is only the option.

III. **Batch scheduling:** Batch processing is the execution of a series of programs ("jobs") on a computer without manual intervention. An operating environment is termed as “batch processing” because the input data are collected into batches or sets of records and each batch is processed as a unit. The output is another batch that can be reused for computation.

IV. **Interactive scheduling:** Interactive processes spend more time waiting for I/O and generally experience short CPU bursts. A text editor is an example of an interactive process with short CPU bursts. It works like short term scheduler. The short-term scheduler selects the process to get the processor from among the processes which are already in memory. They are executed frequently.

V. **Real-time scheduling:** They focuses on the meeting the time constraints. They prevent simultaneous access to shared resources and devices.

3. Scheduling Objective and Criteria or Performance Analysis:

Objectives:

a) **Efficient utilization of resources**. In general, we want to avoid having a resource sit idle when there is some process/thread that could use it.

Example: Suppose that the system contains a mix of CPU bound and IO bound processes (or threads). In this case, if the CPU bound processes/threads are allowed to "hog" the CPU, then utilization of the IO devices will be low, since the primary processes that use them will have to spend a lot of time waiting for the CPU in order to be able to generate work for the IO devices to do.

b) **Maximizing throughput**. The throughput of a system is the number of processes (or threads) that actually complete in a period of time.
c) Minimizing average **turnaround time.**
d) **Minimizing response time.** The response time is the time between when an interactive users submits a request and when the system begins to respond to the request. (Response time is only an issue when the user is interactive).

Many criteria have been suggested for comparison of CPU scheduling algorithms. They are;

- **CPU utilization**: we have to keep the CPU as busy as possible. It may range from 0 to 100%. In a real system it should range from 40 –90 % for lightly and heavily loaded system.
- **Throughput:** It is the measure of work in terms of number of process completed per unit time. For long process this rate may be 1 process per hour, for short transaction, throughput may be 10 processes per second.
- **Turnaround Time:** It is the sum of time periods spent in waiting to get into memory, waiting in ready queue, execution on the CPU and doing I/O. The interval from the time of submission of a process to the time of completion is the turnaround time.

  Turnaround time= Waiting time + service time or burst time
- **Waiting time:** It is the sum of periods waiting in the ready queue.
- **Response time:** in interactive system the turnaround time is not the best criteria. Response time is the amount of time it takes to start responding, not the time taken to output that response.

4. Scheduling Algorithms:

I. First come First Serve:

FCFS is the simplest **non-preemptive** algorithm. Processes are assigned the CPU in the order they request it. That is the process that requests the CPU first is allocated the CPU first. The implementation of FCFS is policy is managed with a FIFO (First in first out) queue. When the first job enters the system from the outside in the morning, it is started immediately and allowed to run as long as it wants to. As other jobs come in, they are put onto the end of the queue. When the running process blocks, the first process on the queue is run next. When a blocked process becomes ready, like a newly arrived job, it is put on the end of the queue.

Advantages:

- Easy to understand and program. With this algorithm a single linked list keeps track of all ready processes.
- Equally fair.
- Suitable especially for Batch Operating system.

Disadvantages:

- FCFS is not suitable for time-sharing systems where it is important that each user should get the CPU for an equal amount of arrival time.

Consider the following set of processes having their burst time mentioned in milliseconds. CPU burst time indicates that for how much time, the process needs the CPU.

Process	Burst Time
P1	24
P2	3
P3	3

Calculate the average waiting time if the processes arrive in the order of:

a) P1, P2, P3
b) P2, P3, P1

a) The process arrives at the order P1, P2, P3. Let us assume they arrive in the same time at 0 ms in the system. We get the following Gantt chart.

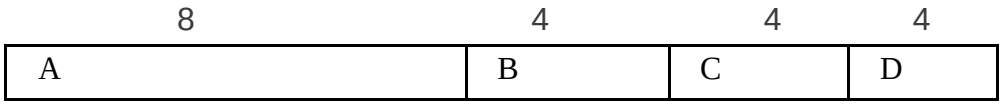
Waiting time for P1= 0ms , for P2 = 24 ms for P3 = 27ms
Average waiting time: (0+24+27)/3= 17

b) If the process arrive in the order P2,P3, P1

Average waiting time: (0+3+6)/3=3. Average waiting time vary substantially if the process CPU burst time vary greatly.

II. Shortest Job First:

It is also known as shortest job next (SJN) or shortest process next algorithm. SJN is a non-preemptive algorithm. When several equally important jobs are sitting in an Input queue waiting to be started, the scheduler picks the shortest jobs first.



Original order: (Turn Around time)

Here we have four jobs A, B, C, D with run times of 8, 4, 4 and 4 minutes respectively. By running them in that order the turnaround time for A is 8 minutes, for B 12 minutes, for C 16 minutes and for D 20 minutes for an average of 14 minutes. Now let us consider running these jobs in the Shortest Job First. B, C, D and then A. The turnaround times are now, 4, 8, 12 and 20 minutes giving the average of 11. Shortest job first is probably optimal.

Consider the four jobs with run times of a, b, c, d. The first job finished at a, the second at a + b and so on. So the mean turnaround time is (4a+3b+2c+d)/4. It is clear that the „a" contributes more to the average than any other. So it should be the shortest one.

The disadvantage of this algorithm is the problem to know the length of time for which CPU is needed by a process. The SJF is optimal when all the jobs are available simultaneously.

The SJF is either preemptive or non-preemptive. Preemptive SJF scheduling is sometimes called **Shortest Remaining Time next** scheduling. With this scheduling algorithm the scheduler always chooses the process whose remaining run time is shortest.

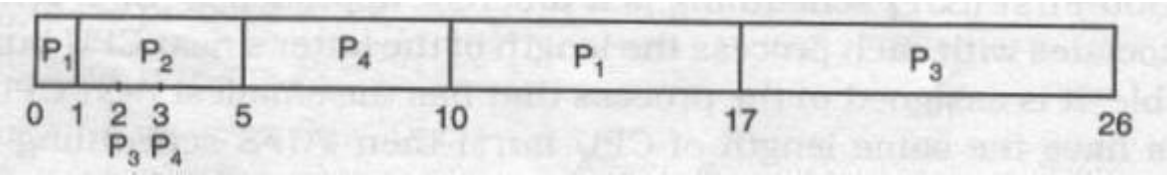
When a new job arrives its total time is compared to the current process remaining time. If the new job needs less time to finish than the current process, the current process is suspended and the new job is started. This scheme allows new short jobs to get good service.

Q). Calculate the average waiting time in
1) Preemptive SJF and
2) Non Preemptive SJF

Note: SJF Default: (Non Preemptive)

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

a. Preemptive SJF (Shortest Remaining Time First):

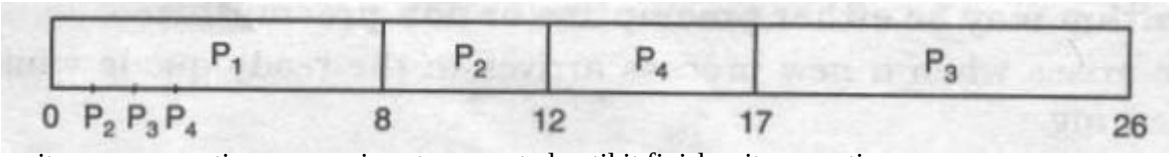


At t=0ms only one process P1 is in the system, whose burst time is 8ms; starts its execution. After 1ms i.e., at t=1, new process P2 (Burst time= 4ms) arrives in the ready queue. Since its burst time is less than the remaining burst time of P1 (7ms) P1 is preempted and execution of P2 is started.

Again at t=2, a new process P3 arrive in the ready queue but its burst time (9ms) is larger than remaining burst time of currently running process (P2 3ms). So P2 is not preempted and continues its execution. Again at t=3, new process P4 (burst time 5ms) arrives . Again for same reason P2 is not preempted until its execution is completed.

Waiting time of P1: 0ms + (10 –1) ms = 9ms
Waiting time of P2: 1ms –1ms = 0ms Waiting time of P3: 17ms –2ms = 15ms Waiting time of P4: 5ms –3ms = 2ms Average waiting time: (9+0+15+2)/4 = 6.5ms

Non-preemptive SJF:



Since its non-preemptive process is not preempted until it finishes its execution.

Waiting time for P1: 0ms
Waiting time for P2: (8-1)ms = 7ms
Waiting time for P3: (17 –2) ms = 15ms
Waiting time for P4: (12 –3)ms = 9ms
Average waiting time: (0+7+15+9)/4 = 7.75ms

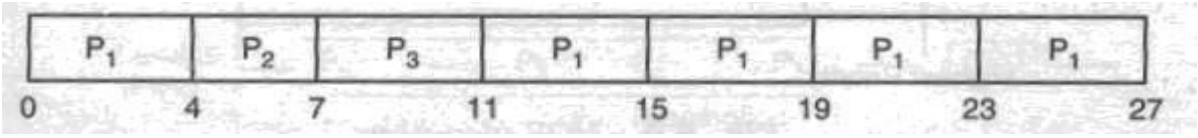
III. Round-Robin Scheduling Algorithms:

One of the oldest, simplest, fairest and most widely used algorithms is round robin (RR).

- In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum.
- If a process does not complete before its CPU-time expires, the CPU is preempted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list.
- If the process has blocked or finished before the quantum has elapsed the CPU switching is done.
- Round Robin Scheduling is preemptive (at the end of time-slice) therefore it is effective in time-sharing environments in which the system needs to guarantee reasonable response times for interactive users.
- The only interesting issue with round robin scheme is the length of the quantum. Setting the quantum too short causes too many context switches and lower the CPU efficiency. On the other hand, setting the quantum too long may cause poor response time and approximates FCFS.
- In any event, the average waiting time under round robin scheduling is on quite long. Consider the following set of processes that arrives at time 0 ms.

Process	Burst Time
P1	20
P2	3
P3	4

If we use time quantum of 4ms then calculate the average waiting time using R-R scheduling.



According to R-R scheduling processes are executed in FCFS order. So, firstly P1 (burst time=20ms) is executed but after 4ms it is preempted and new process P2 (Burst time = 3ms) starts its execution whose execution is completed before the time quantum. Then next process P3 (Burst time=4ms) starts its execution and finally remaining part of P1 gets executed with time quantum of 4ms.

- Waiting time of Process P1: 0 ms + (11 –4) ms = 7 ms
- Waiting time of Process P2: 4ms
- Waiting time of Process P3: 7ms
- Average Waiting time: (7+4+7)/3=6ms

IV. Priority Scheduling:

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 the FCFS order.

Assigning priority:

- To prevent high priority process from running indefinitely the scheduler may decrease the priority of the currently running process at each clock interrupt. If this causes its priority to drop below that of the next highest process, a process switch occurs.
- Each process may be assigned a maximum time quantum that is allowed to run. When this quantum is used up, the next highest priority process is given a chance to run.

Priorities can be assigned statically or dynamically. For UNIX system there is a command nice for assigning static priority.

It is often convenient to group processes into priority classes and use priority scheduling among the classes but round-robin scheduling within each class.

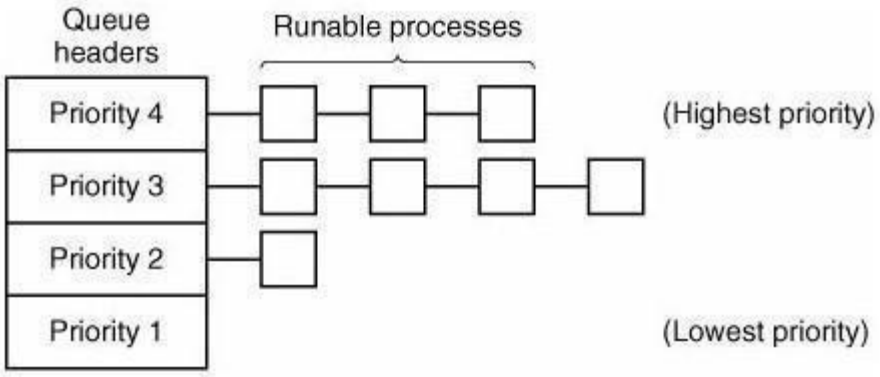


Fig: A scheduling algorithm with four Priority classes

A problem in Priority Scheduling is Starvation: Low priority process may never execute.

Solution: **Aging**: As time progress increase the priority of Process.

V. Scheduling in Real Time System:

A real-time system is one in which time plays an essential role. Typically, one or more physical devices external to the computer generate activity, and the computer must react appropriately to them within a fixed amount of time. For example; the computer in a compact disc player gets the bits as they come off the drive and must convert them into music with a very tight time interval. If the calculation takes too long the music sounds strange. Other example includes;

- Auto pilot in Aircraft
- Robot control in automated factory.
- Patient monitoring in Factory (ICU)

Real time systems are of two types:

i. **Hard Real Time system:** There are absolute deadline that must be met.
ii. **Soft Real Time system:** Missing an occasional deadline is undesirable but nevertheless tolerable.

In both cases real time behavior is achieved by dividing the program into a number of processes, each of whose behavior is predictable & known in advance. These processes are short lived and can run to completion. It‟s the job of schedulers to schedule the process in such a way that all deadlines are met.

If there are m periodic events and event i occur with period P_i and require C_i seconds of CPU time to handle each event, then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

A real-time system that meets these criteria is said to be schedulable.

As an example, consider a soft real-time system with three periodic events, with periods of 100, 200, and 500 ms, respectively. If these events require 50, 30, and 100 ms of CPU time per event, respectively, the system is schedulable because 0.5 + 0.15 + 0.2 < 1. If a fourth event with a period of 1 sec is added, the system will remain schedulable as long as this event does not need more than 150 ms of CPU time per event. Implicit in this calculation is the assumption that the context-switching overhead is so small that it can be ignored.

VI. Highest-Response Ration Next (HRN) Scheduling

Highest Response Ratio Next (HRRN) scheduling is a non-preemptive discipline, in which the priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement (process starvation). It selects a process with the largest ratio of waiting time over service time. This guarantees that a process does not starve due to its requirements. In fact, the jobs that have spent a long time waiting compete against those estimated to have short run times.

Priority = waiting time + estimated runtime / estimated runtime

(Or)

Ratio = (waiting time + service time) / service time

Advantages

- Improves upon SPF scheduling
- Still non-preemptive
- Considers how long process has been waiting
- Prevents indefinite postponement

Disadvantages

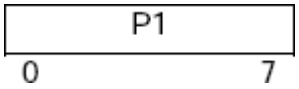
- Does not support external priority system. Processes are scheduled by using internal priority system.

Example: Consider the Processes with following Arrival time, Burst Time and priorities

Process	Arrival time	Burst time	Priority
P1	0	7	3 (High)
P2	2	4	1 (Low)
P3	3	4	2

Solution: HRRN

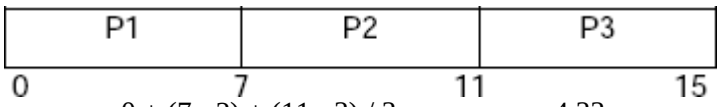
At time 0 only process p1 is available, so p1 is considered for execution



Since it is Non-preemptive, it executes process p1 completely. It takes 7 ms to complete process p1 execution. Now, among p2 and p3 the process with highest response ratio is chosen for execution.

Ratio for p2 = (5 + 4) / 4 = 2.25
Ratio for p3 = (4 + 4) / 4 = 2

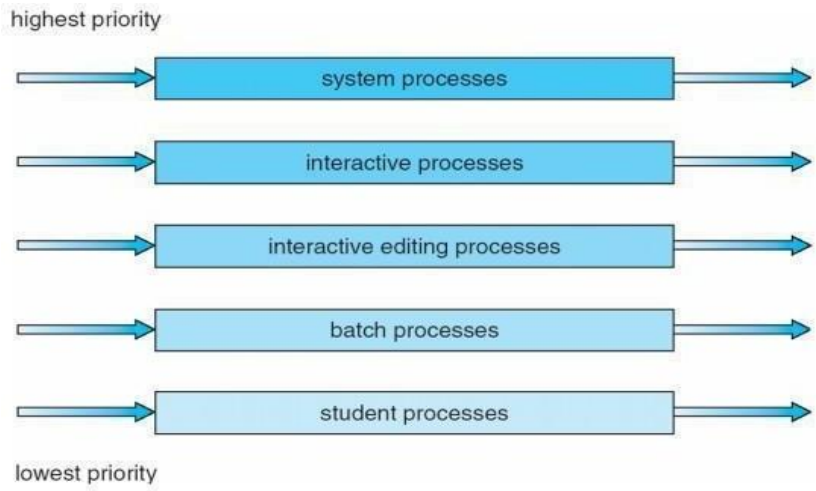
As process p2 is having highest response ratio than that of p3. Process p2 will be considered for execution and then followed by p3.



Average waiting time = 0 + (7 - 2) + (11 - 3) / 3 = 4.33
Average Turnaround time = 7 + (11 - 2) + (15 - 3) / 3 = 9.33

VII. Multilevel Queue Scheduling:

In this scheduling, processes are classified into different groups. A common example may be foreground (or Interactive processes) or background (or batch processes).



Ready queue is partitioned into separate queues:

- foreground (interactive)
- background (batch)

Let us look at an example of a multilevel queue scheduling algorithm with five queues, listed below in the order of priority.

i. System processes
ii. Interactive processes
iii. Interactive editing processes
iv. Batch processes
v. Student processes

Each queue has absolute priority over lower priority queues. No processes in the batch queue, for example could run unless the queue for System processes, interactive processes and interactive editing processes were all empty. If an interactive editing process enters the ready queue while a batch process was running the batch process would be preempted.

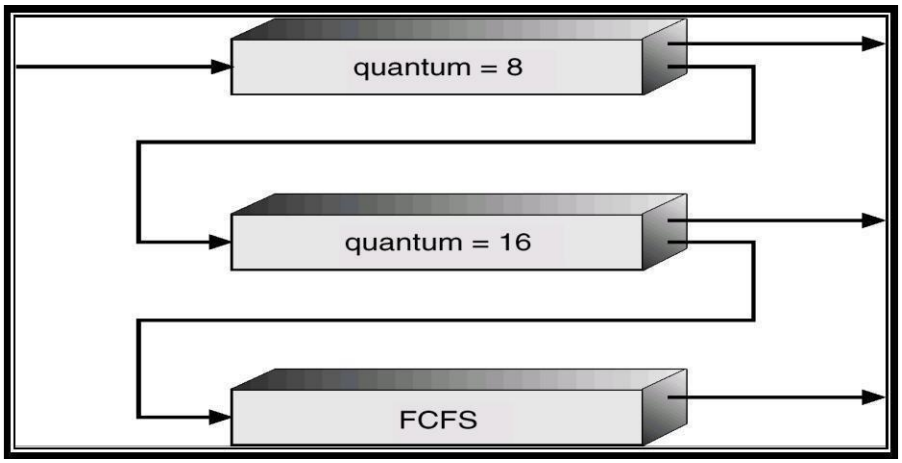
Another possibility is to time slice between the queues. For instance foreground queue can be given 80% of the CPU time for RR scheduling among its processes, whereas the background receives 20% of the CPU time.

VIII. Multilevel Feedback Queue:

A process can move between the various queues. It is defined by the following parameters:

- number of queues
- scheduling algorithms for each queue
- method used to determine when to upgrade a process
- method used to determine when to demote a process
- method used to determine which queue a process will enter when that process needs service

For example let us consider three queues;



- Q_0 –time quantum 8 milliseconds
- Q_1 –time quantum 16 milliseconds
- Q_2 –FCFS

CPU scheduling is done in following ways;

- When a new job enters queue Q_0 it is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
- At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.