Advanced Types Of Scheduling

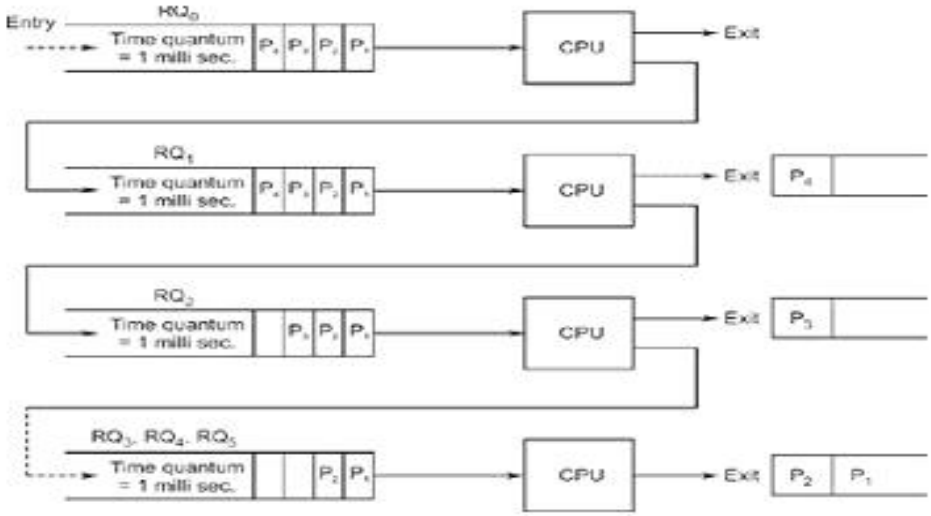
In the previous article I discussed about some of the basic types of scheduling algorithms. In this article I will discuss about some other advanced scheduling algorithms.

Highest Response Ratio Next (HRRN)

When the present running process completed or it is blocked, choose the ready process with the greatest value of Response Ratio (RR). It is attractive because it considers the age of the process also. Here expected service time must be estimated.

This is depicted in the figure below:

Selection Function is Response ratio(RR)=W+S/S
Where
W=Waiting_Time
S=Service_Time_Required



When a process first enters minimum value of RR is 1.0.

Decision mode is non-preemptive.
At time 10 RR of P3=5+3/3 and P4=2+2/2 .
Here RR of P3 is greater than P4, thus P3 is serving first.

Advantages

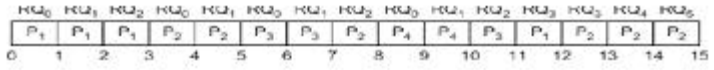
- Compromises FCFS for long processes and SPN for short processes by considering waiting and service times.
- Throughput is high.
- Good response time.

Disadvantage

- There is a overhead on processor.

Multilevel Feedback Queue Scheduling

In this algorithm dynamic priority mechanism is used i.e., when a process first enters in to the system, it is placed in RQ0 when it returns to the Ready State after its first execution, it is placed in RQ1. After each subsequent execution, it is moved to the next lower priority queue. This is depicted in the figure below:



A shorter process will complete quickly, without migrating to the lower levels of the hierarchy. The scheduler first executes all processes present in the queue RQ0. Processes present in queue RQ1 will execute only when no processes is present in RQ0. Similarly RQ2, RQ3 and so on will execute.

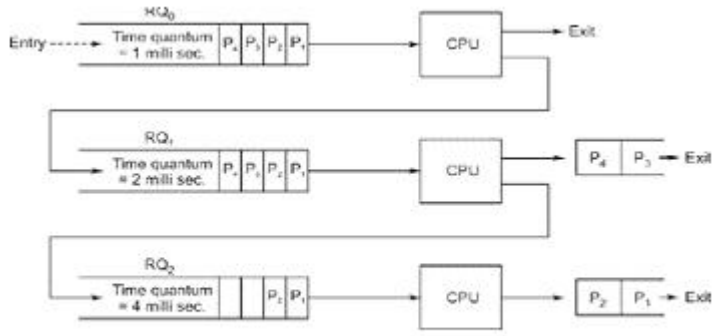
For a quantum of one unit, the behavior is some what similar to round robin scheduling.

Disadvantage

It is possible for starvation, if new jobs are continuously entering.

To compensate this vary the preemption times. That is RQ0 is allowed to execute for 1 time unit and then process is preempted to RQ1. A process scheduled for RQ1 is allowed to execute 2 time units and so on. In general a process scheduled from RQi

is allowed to execute 2i time units before preemption from that queue. The next figure is showing the multilevel feedback queue scheduling with time quantum = 2n.



Longer processes may still suffer with starvation. For overcoming this problem promote a process to a higher-priority queue after it spends a certain amount of time in waiting for execution.

Multilevel Queue Scheduling

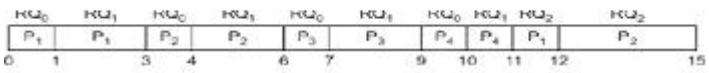
The ready queue is partitioned into number of ready queues. Each ready queue should have certain priority and can have its own scheduling algorithm.This is created for situations in which

processes are easily classified into groups:

- Foreground(Interactive) processes
- Background(Batch) processes

These two type of process have different response time requirements and thus different scheduling requirements. The processes are permanently assigned to one queue based on some property of the process(e.g. memory size,priority type).Each queue has its own scheduling algorithm. For e.g. the foreground process might be scheduled by an RR algorithm while the background queue is scheduled by a FCFS algorithm. There must be scheduling between the queues commonly implemented as fixed priority preemptive scheduling i.e. foreground process have absolute priority over the background process. It could also use a time slice algorithm where each queue gets a certain amount of CPU time which it can schedule among its processes.

Eg. 80% to foreground queue for RR and 20% to background queue for FCFS.



Multiple-Processor Scheduling

Multiple-processor scheduling (in parallel systems) is more complex and it concentrate on systems where the processors are identical (i.e., homogeneous) in terms of their functionality.

Multiprocessor systems provide higher throughput than uniprocessor systems because multiple programs can run concurrently on different processors. However, some overhead is incurred in dividing a task amongst many CPUs as wel as during contention from shared resources. As a result, the speed obtained by using multiprocessor systems is not linear, that is, it is not directly proportional to the number of CPUs.

Limitations

- Consider a system with an Input/Output device attached to a private bus of one processor then processes wishing to use that device must be scheduled to run only on that processor.
- If several identical processors are present, then load sharing (by using memory and peripherals) can occur. In this case when we provide a separate queue for each processor then few processors may be idle, with an empty queue.
- For overcoming the problem in the above limitation use a common ready queue for all the processors. Here two or more processors may choose the same process. Avoid this problem by using one processor exclusively for scheduling the processes for the other processors with a master slave structure.

Symmetric multiprocessing is simple when compared with the asymmetric multiprocessing, because asymmetric multiprocessing is to perform all scheduling decisions such as Input/Output processing and other system activities should be handled by only one processor (i.e., master), while other processors execute user code, where as in symmetric multiprocessing any process or thread can be assigned to any processor. Even though the symmetric multiprocessing is simple it is not efficient and effective.

Real-Time Scheduling

We have already seen the real-time systems under types of operating systems and these are hard-real-time systems and soft real-time systems.

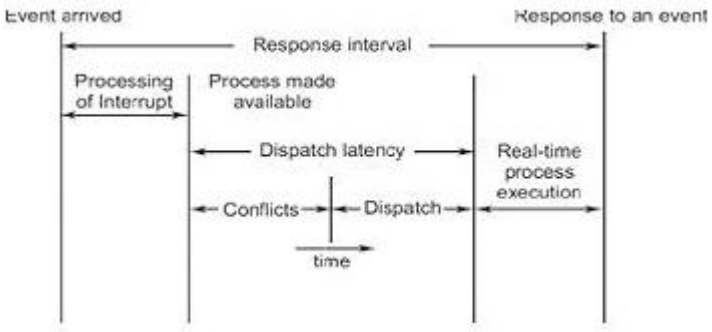
In hard real-time systems process should submit along with a statement of the amount of time in which it needs to complete. These are composed of special-purpose software running (in-built) on hardware.

Soft real-time system is less restrictive. It is a general purpose system, which supports multimedia, high-speed interactive graphics and variety of tasks.

Real-time processes must have the highest priority. Many operating systems including UNIX are forced to wait for either a system call to complete or for an Input/Output block to take place before performing context switch. This dispatch latency in these systems can be long. Some system calls are complex and some Input/Output devices are slow.

For maintaining low dispatch latency, allow system calls to be preemptible. Different methods are there for achieving this (i.e., low dispastch latency). One method is to insert preemption points in long-duration system calls. It checks whether a high-priority process needs to be run.

Preemption points place only at ‘safe’ locations in the kernel.



If the higher-priority process needs to read or modify kernel data during the lower-priority access, the higher-priority process must wait for the completion of the lower-priority process.

This situation is known as "priority inversion". Once when they finish their work, their priority reverts to its natural situation.

The conflict phase of a dispatch latency has three components:

- Preemption of any process running in the kernel.
- Lower-priority processes releasing resources needed by the higher-priority process.
- Context switching from the current process to the higher-priority process.

Real-time Embeded Systems

Now a days these devices are found everywhere, from car engines and manufacturing robots to VCRs, DVDs and microwave ovens. These devices should have very clear specific tasks.

Real-time embeded systems may have little or no user interface. These systems spend their time in monitoring and managing hardware devices, such as robot arms movement, automobile engines, etc., by using sensors.Embeded systems almost run in real-time operating system. A real-time system is to use when rigid time requirements are to follow as the part of its operation of a processor or the flow of data.

E.g., Entire house can be computerized, so that a central computer, either a general-purpose computer or an embeded system can control the heating and lighting, alarm systems and even coffe makers.

Disk-Scheduling Policies

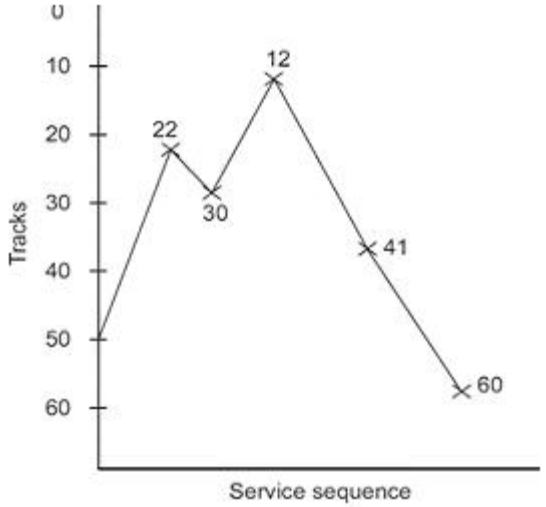
Disk-scheduling policies are useful for reducing the average time spent on seeks. That is the time required to move the disk arm (or head) to the required track by using movable head. For doing this activity different disk scheduling policies are available. For e.g., Requests arrived in sequence are 22, 30, 12, 41 and 60.

First Come First Served (FCFS)

The simplest form of scheduling is first come first served or first in first out scheduling.

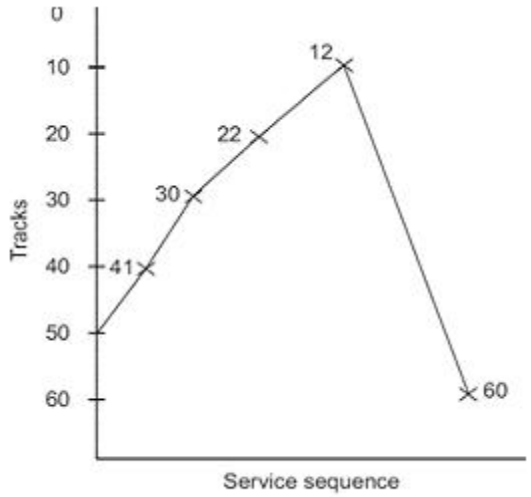
Assume that a disk containing 100 tracks and disk requests queue has random requests in it. That is the requests in the order received are 22, 30, 12, 41 and 60. In

FCFS method these requests are serviced in the order they are received as shown in the figure below. Initial head position is at 50th track.



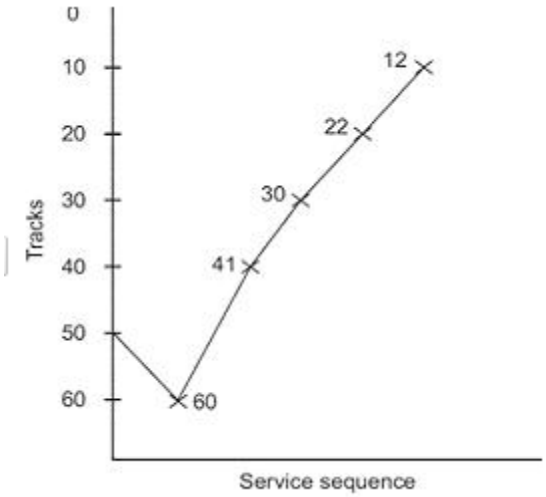
Shortest Service Time First (SSTF)

This method is also called as shortest seek time first. Here we will assume the requests for servicing the tracks 22, 30, 12, 41 and 60 are arrived at the zeroth time interval and the initial head position is at track 50. This method chooses the next request to serve, where the seek time is the shortest-with respect to the current head position. Here the request with track number 41 is nearer or shorter from the initial position of head i.e., 50. First it will serve 41 then 30 then 22 then 12 and finally 60. In the mean time, if any request (i.e., after starting the servicing) has to arrive and if it is very nearer or in the path of the servicing it will serve otherwise it will serve in the next cycle. It is Shown in the figure below:



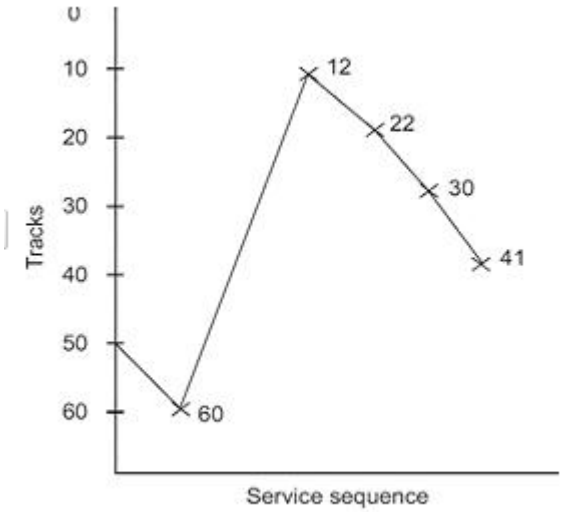
SCAN Method

In this method, the head starts a scan in some direction by picking up all the requests on the way, and then reverses the direction and serve the requests present in that direction. During the service if new requests are to arrive, it will process similar to the above method. It is shown in the figure below:



Circular (C)-SCAN Method

As shown in the figure below it restricts scanning to one direction only. That is when the last track has been visited in one direction, the arm is returned to the opposite end of the disk and then scan begins again.



Source: http://www.go4expert.com/articles/advanced-types-scheduling-t22349/