UNIT-I PART - 2
PROCESS MANAGEMENT

PROCESS CONCEPT

The process concept includes the following:

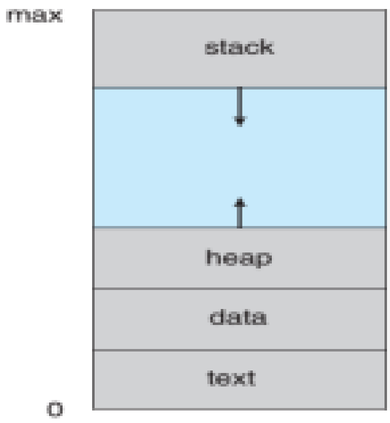
1. Process
2. Process state
3. Process Control Block
4. Threads

Process

A process can be thought of as a program in execution (or) A process is the unit of work in a modern time-sharing system.

A process will need certain resources such as CPU time, memory, files and I/O devices to accomplish its task.

These resources are allocated to the process either when it is created or while it is executing. The below figure shows the structure of process in memory:



The process contains several sections: Text, Data, Heap and Stack.

· **Text Section** contains the program code. It also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers.

· **Process stack** contains temporary data such as function parameters, return addresses and local variables.

· **Data section** contains global variables.

· **Heap** is memory that is dynamically allocated during process run time.

Difference between Program and Process:

· A program is a ***passive*** entity, such as a file containing a list of instructions stored on disk often called an **executable file**.

· A process is an ***active*** entity with a program counter specifying the next instruction to execute and a set of associated resources.

· A program becomes a process when an executable file is loaded into memory.

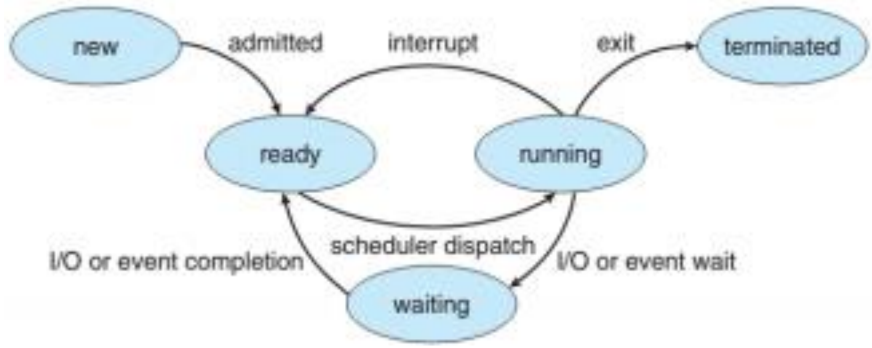
Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line as in prog.exe or a.out.

Although two processes may be associated with the same program, they are considered as two separate execution sequences. For instance, several users may be running different copies of the mail program or the same user may invoke many copies of the web browser program. Each of these is considered as a separate process.

Process State

As a process executes, it changes **state**. The process state defines the current activity of that process.

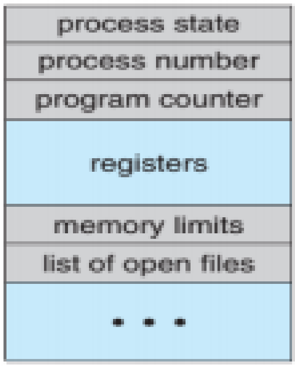
A process may be in one of the following states:
· **New**: The process is being created.
· **Ready**: The process is waiting to be assigned to a processor.
· **Running**: Instructions are being executed.
· **Waiting**: The process is waiting for some event to occur such as an I/O completion or reception of a signal.
· **Terminated**: The process has finished execution.
Note: Only one process can be *running* on any processor at any instant of time.



Process Control Block

Each process is represented in the operating system by a **Process Control Block (PCB)**. It is also called a **Task Control Block**.

PCB serves as the repository for any information that may vary from process to process.



The PCB contains information related to process such as:

· **Process state**: The state may be new, ready, running, waiting and terminated. · **Program counter**: The counter indicates the address of the next instruction to be executed for this process.

· **CPU registers**: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers and general purpose registers etc. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

· **CPU-scheduling information**: This information includes a process priority, pointers to scheduling queues and any other scheduling parameters.

· **Memory-management information**: This information includes the base and limit registers values, the page tables or the segment tables depending on the memory system used by the operating system.

· **Accounting information**: This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers and so on.

· **I/O status information**: This information includes the list of I/O devices allocated to the process, a list of open files and so on.

Threads

In a single processor system a process is a program that performs a single **thread** of execution.

· **Example:** When a process is running a word-processor program, a single thread of instructions is being executed.

· This single thread of control allows the process to perform only one task at a time. · The user cannot simultaneously type in characters and run the spell checker within the same process.

In multicore system or multi-processor system allows a process to run multiple threads of execution in parallel.

On a system that supports threads, the PCB is expanded to include information for each thread.

Process Scheduling

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.

To meet these objectives, the **Process Scheduler** selects an available process for program execution on the CPU.

Process scheduling involves three things:

1. Scheduling Queues
2. Schedulers
3. Context Switch

Scheduling Queues

There are several queues are implemented in operating system such as Job Queue, Ready Queue, Device Queue.

· **Job Queue:** It consists of all processes in the system. As processes enter the system, they are put into a **job queue**.

· **Ready Queue:** The processes that are residing in main memory and they are ready and waiting to execute are kept on a list called the **Ready Queue**. Ready queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.· **Device Queue:** Each device has its own device queue. It contains the list of processes waiting for a particular I/O device.

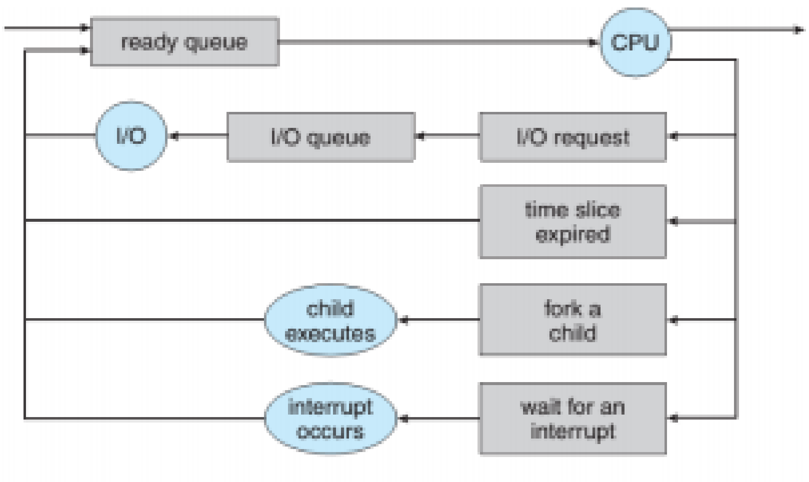


Figure 3.6 Queueing-diagram representation of process scheduling.

Consider the above Queuing Diagram:

· Two types of queues are present: the **Ready Queue** and a set of **Device Queues**. CPU and

I/O are the resources that serve the queues.

· A new process is initially put in the ready queue. It waits there until it is selected for execution or **dispatched**.

Once the process is allocated the CPU and is executing, one of several events could occur: · The process could issue an I/O request and then be placed in an I/O queue. · The process could create a new child process and wait for the child's termination. · The process could be removed forcibly from the CPU, as a result of an interrupt and be put back in the ready queue.

<u>Schedulers</u>

A process migrates among the various scheduling queues throughout its lifetime. For scheduling purpose, the operating system must select processes from these queues. The selection process is carried out by the **Scheduler.**

There are three types of Schedulers are used:

1. Long Term Scheduler
2. Short Term Scheduler
3. Medium Term Scheduler

Long Term Scheduler (New to ready state)

· Initially processes are spooled to a mass-storage device (i.e Hard disk), where they are kept for later execution.

· Long-term scheduler or job scheduler selects processes from this pool and loads them into main memory for execution. (i.e. from Hard disk to Main memory).

· The long-term scheduler executes much less frequently, there may be minutes of time between creation of one new process to another process.

· The long-term scheduler controls the **degree of multiprogramming** (the number of processes in memory).

Short Term Scheduler (Ready to Running)

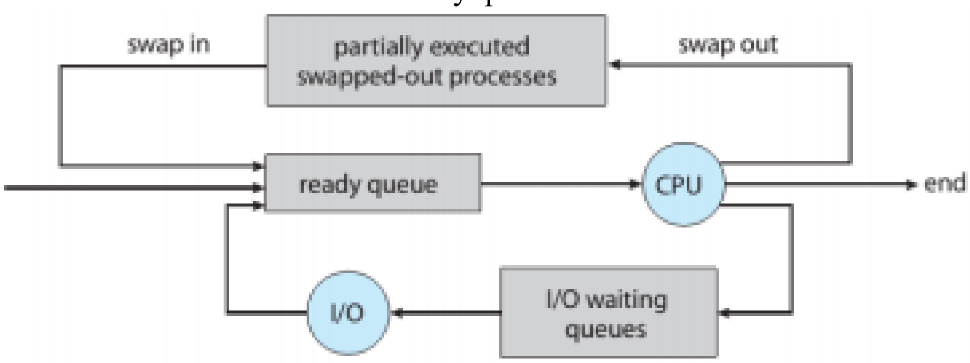
· Short-term scheduler or CPU scheduler selects from among the processes that are ready to execute and allocates the CPU to one of them. (i.e. a process that resides in main memory will be taken by CPU for execution).

· The short-term scheduler must select a new process for the CPU frequently. · The short term scheduler must be very fast because of the short time between executions of processes.

Medium Term Scheduler

Medium Term Scheduler does two tasks:

1. **Swapping:** Medium-term scheduler removes a process from main memory and stores it into the secondary storage. After some time, the process can be reintroduced into main memory and its execution can be continued where it left off. This procedure is called Swapping.
2. Medium Term Scheduler moves a process from CPU to I/O waiting queue and I/O queue to ready queue.



The processes can be described as two types:

1. I/O bound process is one that spends more of its time doing I/O than it spends doing

computations.

2. CPU Bound process using more of its time doing computations and generates I/O requests infrequently.

The long-term scheduler selects a good ***process mix*** of I/O-bound and CPU-bound processes. · If all processes are I/O bound, the ready queue will almost always be empty and the CPU will remain idle for long time because I/O device processing takes a lot of time. · If all processes are CPU bound, the I/O waiting queue will almost always be empty. I/O devices will be idle and CPU is busy for most of the time.

· Thus if the system maintains the combination of CPU bound and I/O bound processes then the system performance will be increased.

Note: Time-sharing systems such as UNIX and Microsoft Windows systems often have no long-term scheduler but simply put every new process in memory for the short-term scheduler.

Context Switching

· Switching the CPU from one process to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **Context Switch**.

· The context is represented in the PCB of the process. It includes the value of the CPU registers, the process state and memory-management information.

· When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

· Context-switch time is pure overhead, because the system does no useful work while switching. Context switch time may be in few milliseconds.

CPU SCHEDULING

CPU scheduling is the basis of Multi-programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. · 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 CPU will sit idle and waiting for a process that needs an I/O operation to complete. If the I/O operation completes then only the CPU will start executing the process. A lot of CPU time has been wasted with this procedure.

· The objective of multiprogramming is to have some process running at all times to maximize CPU utilization.

· When several processes are in main memory, if one processes is waiting for I/O then the operating system takes the CPU away from that process and gives the CPU to another process. Hence there will be no wastage of CPU time.

Concepts of CPU Scheduling

1. CPU–I/O Burst Cycle
2. CPU Scheduler
3. Pre-emptive Scheduling
4. Dispatcher

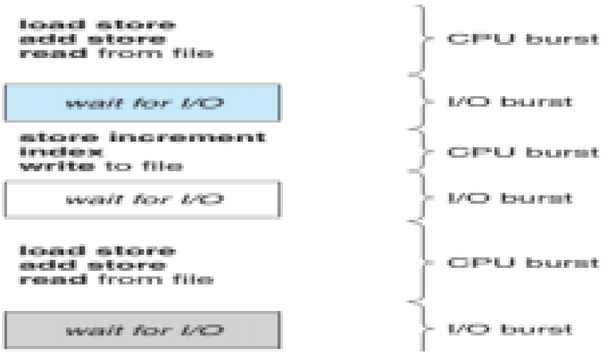
CPU–I/O Burst Cycle

Process execution consists of a **cycle** of CPU execution and I/O wait.

· Process execution begins with a **CPU burst**. That is followed by an **I/O burst.** Processes alternate between these two states.

· The final CPU burst ends with a system request to terminate execution. ·

Hence the **First cycle** and **Last cycle** of execution must be CPU burst.



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**.

Preemptive Scheduling

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

1. When a process switches from the running state to the waiting state. 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.
   Example: when an interrupt occurs
3. When a process switches from the waiting state to the ready state.
   Example: at completion of I/O.
4. When a process terminates.

For situations 2 and 4 are considered as **Pre-emptive scheduling** situations. Mach OS X, WINDOWS 95 and all subsequent versions of WINDOWS are using Preemptive scheduling.

Dispatcher

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. Dispatcher function involves:

1. Switching context
2. Switching to user mode
3. Jumping to the proper location in the user program to restart that program. The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another process running is known as the **Dispatch Latency**.

SCHEDULING CRITERIA

Different CPU-scheduling algorithms have different properties and the choice of a particular algorithm may favor one class of processes over another.

Many criteria have been suggested for comparing CPU-scheduling algorithms: · **CPU utilization:** CPU must be kept as busy as possible. CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 to 90 percent. · **Throughput:** The number of processes that are completed per time unit. · **Turn-Around Time:** It is the interval from the time of submission of a process to the time of completion. 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:** It is the amount of time that a process spends waiting in the ready queue. · **Response time:** It is the time from the submission of a request until the first response is produced. Interactive systems use response time as its measure.

Note: It is desirable to maximize CPU utilization and Throughput and to minimize Turn Around Time, Waiting time and Response time.

CPU SCHEDULING ALGORITHMS

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. Different CPU-scheduling algorithms are: 1. First-Come, First-Served Scheduling (FCFS)
2. Shortest-Job-First Scheduling (SJF)
3. Priority Scheduling
4. Round Robin Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

Gantt Chart is a bar chart that is used to illustrates a particular schedule including the start and finish times of each of the participating processes.

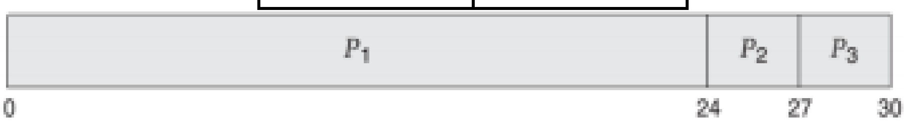
First-Come, First-Served Scheduling (FCFS)

In FCFS, the process that requests the CPU first is allocated the CPU first.
· FCFS scheduling algorithm is Non-preemptive.
· Once the CPU has been allocated to a process, it keeps the CPU until it releases the CPU. · FCFS can be implemented by using FIFO queues.
· 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.

Example:1 Consider the following set of processes that arrive at time 0. The processes are arrived in the order P1, P2, P3, with the length of the CPU burst given in milliseconds.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for FCFS is:



The average waiting time under the FCFS policy is often quite long.
· 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.

Convoy Effect in FCFS

Convoy effect means, when a big process is executing in CPU, all the smaller processes must have to wait until the big process execution completes. This will effect the performance of the system.

Example:2 Let us consider same example above but with the processes arrived in the order P2, P3, P1.

The processes coming at P2, P3, P1 the average waiting time (6 + 0 + 3)/3 = 3 milliseconds whereas the processes are came in the order P1, P2, P3 the average waiting time is 17 milliseconds.

Disadvantage of FCFS:

FCFS scheduling algorithm is Non-preemptive, it allows one process to keep CPU for long time. Hence it is not suitable for time sharing systems.

Shortest-Job-First Scheduling (SJF)

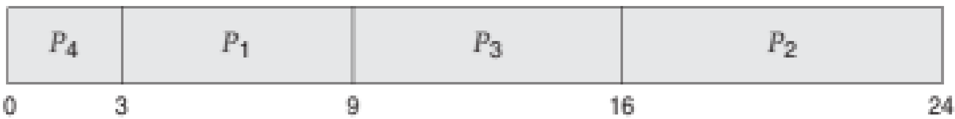
SJF algorithm is defined as **“**when the CPU is available, it is assigned to the process that has the smallest next CPU burst**”.** If the next CPU bursts of two processes are the same, FCFS scheduling is used between two processes.

SJF is also called as **Shortest-Next CPU-Burst** algorithm, because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

Example: Consider the following processes and CPU burst in milliseconds:

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

Gantt Chart of SJF algorithm:



Waiting Time for Processes:

Process	Burst Time (ms)	Waiting Time
P1	6	3
P2	8	16
P3	7	9
P4	3	0
Average Waiting Time		7 ms

· By looking at the above table the average waiting time by using SJF algorithm is 7ms. · SJF gives the minimum average waiting time for a given set of processes. SJF is optimal. · The average waiting time decreases because moving a short process before long process decrease the waiting time of the short process more than it increases the waiting time of the long process.

Difficulty with SJF

The difficulty with the SJF algorithm is “knowing the length of the next CPU request”. With Short-Term Scheduling, there is no way to know the length of the next CPU burst. It is not implemented practically.

Shortest Remaining Time First Scheduling (SRTF)

SRTF is the pre-emptive SJF algorithm.

· A new process arrives at the ready queue, while a previous process is still executing.

· The next CPU burst of the newly arrived process may be shorter than the currently executing process.

· SRTF will preempt the currently executing process and executes the shortest job.

Consider the four processes with arrival times and burst times in milliseconds:

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

Gantt Chart for SRTF

P_1	P_2	P_4	P_1	P_3

0 1 5 10 17 26 ·

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 = **26/4 = 6.5 milliseconds.**

<u>Priority Scheduling</u>

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.

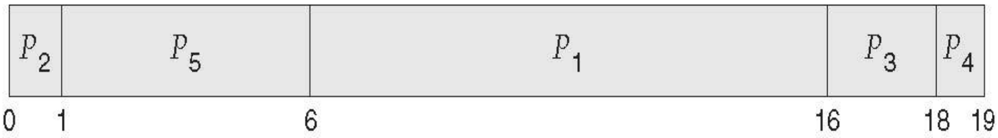
· An SJF algorithm is special kind of priority scheduling algorithm where small CPU burst will have higher priority.

· Priorities can be defined based on time limits, memory requirements, the number of open files etc.

Example: Consider the following processes with CPU burst and Priorities. All the processes are arrived at time t=0 in the same order. Low numbers are having higher priority.

Process	Burst time (ms)	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Gantt chart for Priority Scheduling:



Process	Burst time (ms)	Waiting Time
P1	10	6
P2	1	0
P3	2	16
P4	1	18
P5	5	1
Average Waiting Time		8.2 ms

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.

Problem: Starvation or Indefinite Blocking

· In priority Scheduling when there is a continuous flow of higher priority processes has come to ready queue then all the lower priority processes must have to wait for the CPU until the all the higher priority processes execution completes.

· This leads to lower priority processes blocked from getting CPU for long period of time. This situation is called Starvation or Indefinite blocking.

· In worst case indefinite blocking may take years to execute the process.

Solution: Aging

Aging involves gradually increasing the priority of processes that wait in the system for a long time.

Round-Robin Scheduling (RR)

Round-Robin (RR) scheduling algorithm is designed especially for Timesharing systems. · RR 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**. New processes are added to the tail of the ready queue.

· The CPU scheduler goes around the ready queue by allocating the CPU to each process for a time interval of up to 1 time quantum and dispatches the process.

· If a process CPU burst exceeds 1 time quantum, that process is preempted and is put back in the ready queue.

In RR scheduling one of two things will then happen.

1. The process may have a CPU burst of less than 1 time quantum. The process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the

ready queue.

2. 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.

Consider the following set of processes that arrive at time 0 and the processes are arrived in the order P1, P2, P3 and Time Quantam=4.

Process	**Burst Time**
P1	24
P2	3
P3	3

Gantt chart of Round Robin Scheduling

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

· If we use a time quantum of 4 milliseconds, then process *P*1 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 *P*2.

· CPU burst of Process *P*2 is 3, so it does not need 4 milliseconds then it quits before its time quantum expires. The CPU is then given to the next process *P*3.

· Once each process has received 1 time quantum, the CPU is returned to process *P*1 for an additional time quantum.

The average waiting time under the RR policy is often long.

· *P*1 waits for 6 milliseconds (10 - 4), *P*2 waits for 4 milliseconds and *P*3 waits for 7 milliseconds. Thus, the average waiting time is 17/3 = 5.66 milliseconds.

The performance of the RR algorithm depends on the size of the Time Quantum. · If the time quantum is extremely large, the RR policy is the same as the FCFS policy. · If the time quantum is extremely small (i.e. 1 millisecond) the RR approach can result in a large number of context switches.

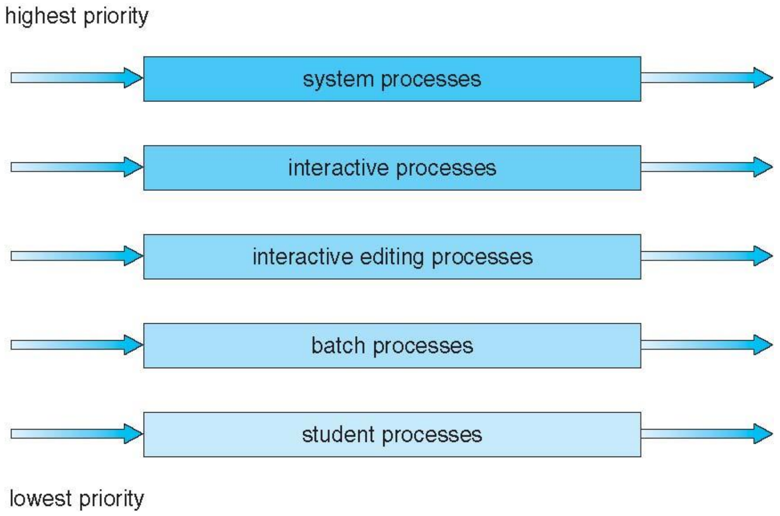
· The time taken for context switch value should be a small fraction of Time quanta then the performance of the RR will be increased.

Note: A rule of thumb is that 80 percent of the CPU bursts should be shorter than the time quantum.

Multi-Level Queue Scheduling (MLQ)

In Multilevel Queue Scheduling algorithm the processes are classified into different groups. · A Multilevel queue scheduling partitions the ready queue into several separate queues. · The processes are permanently assigned to one queue based on memory size, process priority or process type. Each queue has its own scheduling algorithm.

Example: Foreground processes have highest priority over background processes and these processes have different response times hence it needs different scheduling.



The above figure shows Multi-level 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

Each queue has absolute priority over lower-priority queues.

· No lower level queue processes will start executing unless all the processes in higher level queue are empty.
  Example: The interactive processes start executing only when all the processes in system queue are empty.

· If a lower priority process is executing and an higher priority process entered into the queue then lower priority process will be preempted and starts executing a higher priority process.
  Example: If a system process entered the ready queue while an interactive process was running, the interactive process would be preempted.

Disadvantage: Starvation of Lower level queue

In multilevel queue scheduling algorithm is inflexible.

· The processes are permanently assigned to a queue when they enter the system. Process are not allowed to move from one queue to other queue.

· There is a chance that lower level queues will be in starvation because unless the higher level queues are empty no lower level queues will be executing.

· If at any instant of time if there is a process in higher priority queue then there is no chance that lower level process can be executed eternally.

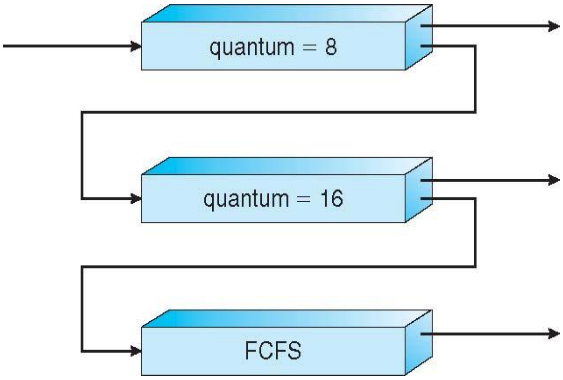
Multilevel Feedback Queue Scheduling is used to overcome the problem of Multi-level queue scheduling.

Multilevel Feedback Queue Scheduling (MLFQ)

Multilevel feedback queue scheduling algorithm allows a process to move between queues. · Processes are separated 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. · A process that waits too long in a lower-priority queue moved to a higher-priority queue. · This form of aging prevents starvation.

Consider a multilevel feedback queue scheduler with three queues: queue0, queue1, queue2. ·

The scheduler first executes all processes in queue0 then queue1 and then queue2. · Only when queue0 and queue1 is empty, scheduler will execute processes in queue2. · A process that arrives for queue1 will preempt a process in queue2. A process in queue1 will in turn be preempted by a process arriving for queue0.



· A process entering the ready queue is put in queue0. A process in queue 0 is given a time quantum of 8ms. 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 queue1 is given a quantum of 16ms. If it does not complete, it is preempted and is put into queue2.

· 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 8ms or less. Such a process will quickly get the CPU and finish its CPU burst and go off to its next I/O burst.

· Processes that need more than 8ms but less than 24ms are also served quickly, although with lower priority than shorter processes.

· Long processes automatically sink to queue2 and are served in FCFS order with any CPU cycles left over from queues0 and queue1.

A Multi-Level 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.