CHAPTER 2

PROCESS MANAGEMENT

- ***Basic Concepts of Process***
- ***Operation on Process***
- ***Process State Model and Transition***
- ***Process Control Block***
- ***Context Switching***
- ***Introduction to Threads***
- ***Types of Threads***
- ***Thread Models***
- ***Basic Concepts of Scheduling***
- ***Types of Schedulers, Scheduling Criteria***
- ***Scheduling Algorithms***

➢ BASIC CONCEPTS OF PROCESS

A process is essentially running software. The execution of any process must occur in a specific order. A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory:

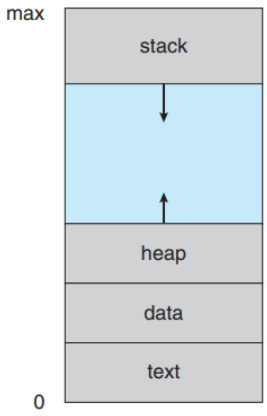


Fig.: Layout of Process in memory

1. Stack: The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2. Heap: This is dynamically allocated memory to a process during its run time.
3. Text: This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4. Data: This section contains the global and static variables.

➢ OPERATION ON PROCESS

1. **Creation:** Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.
2. **Scheduling:** Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

3. **Execution:** Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

4. **Deletion/killing:** Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system.

➢ PROCESS STATE MODEL AND TRANSITION

Effective way to manage multiple process at a time

The state of a process is an effective way to manage the processes in a multi-programming environment. A process may be running and another process may be waiting for an I/O device at the same time. Therefore, through the states of the processes, the situation of every process at a given time is identified and every process is managed such that it gets a uniform execution. Various events happening in the system cause a process to change its state. Thus, a process moves through a series of discrete states. The states of a process are depicted as follows:

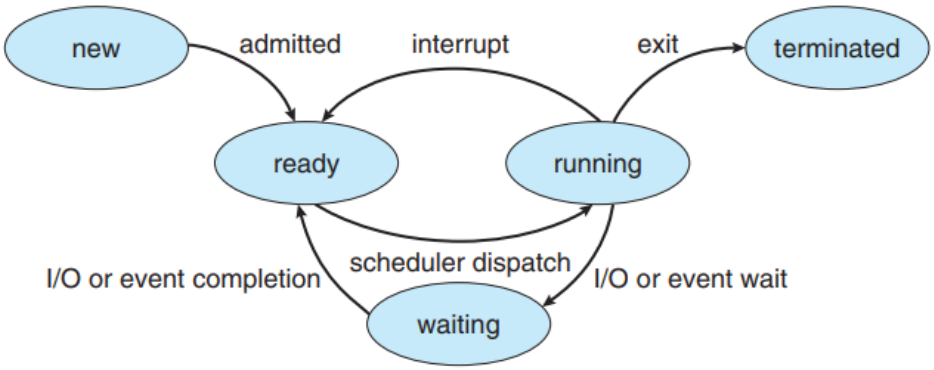


Fig.: Process state diagram.

1. **New:** A program which is going to be picked up by the OS into the main memory is called a new process.
2. **Ready:** Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

   secondary --> main memory

   The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. **Running:** One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.
4. **Block or wait:** From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

5. **Completion or termination:** When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

- **State Transitions:** There can be various events that lead to a state transition for a process. The possible state transitions are given below:

1. **Null -> New:** A new process is created for the execution of a process.

2. **New -> Ready:** The system will move the process from new to ready state and now it is ready for execution. Here a system may set a limit so that multiple processes can't occur otherwise there may be a performance issue.

3. **Ready -> Running:** The OS now selects a process for a run and the system chooses only one process in a ready state for execution.

4. **Running -> Exit:** The system terminates a process if the process indicates that is now completed or if it has been aborted.

5. **Running -> Ready:** The reason for which this transition occurs is that when the running process has reached its maximum running time for uninterrupted execution.

6. **Running -> Blocked:** A process is put in the blocked state if it requests for something it is waiting. Like, a process may request some resources that might not be available at the time to finish before the process can continue.

7. **Blocked -> Ready:** A process moves from blocked state to the ready state when the event for which it has been waiting.

8. **Ready -> Exit:** This transition can exist only in some cases because, in some systems, a parent may terminate a child's process at any time.

➢ PROCESS CONTROL BLOCK

Each process is represented in the operating system by a process control block (PCB), also called a task control block. A PCB is shown in Figure, it contains many pieces of information associated with a specific process.

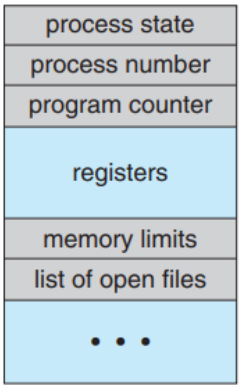


Fig.: Process Control Block (PCB)

1. **Process state:** The state may be new, ready, running, waiting, halted, and so on.

2. **Program counter:** The counter indicates the address of the next instruction to be executed for this process.

3. **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, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward when it is rescheduled to run.

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

5. **Memory-management information:** This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.

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

7. **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.

➢ CONTEXT SWITCHING

The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system.

When switching perform in the system, it stores the old running process's status in the form of registers and assigns the CPU to a new process to execute its tasks. While a new process is running in the system, the previous process must wait in a ready queue.

The execution of the old process starts at that point where another process stopped it. It defines the characteristics of a multitasking operating system in which multiple processes shared the same CPU to perform multiple tasks without the need for additional processors in the system.

There are several steps involves in context switching of the processes. The following diagram represents the context switching of two processes, P1 to P2, when an interrupt, I/O needs, or priority-based process occurs in the ready queue of PCB.

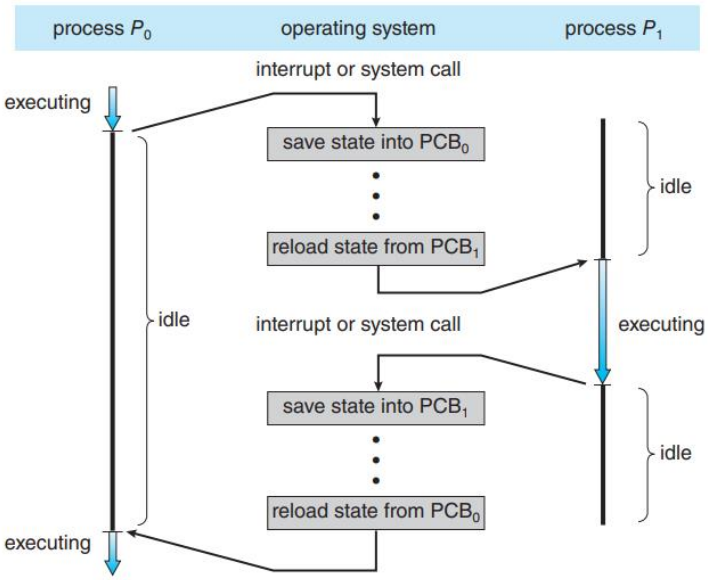


Fig.: Context Switch from process to process.

The following steps are taken when switching Process P1 to Process 2:

1. First, the context switching needs to save the state of process P1 in the form of the program counter and the registers to the PCB (Program Counter Block), which is in the running state.
2. Now update PCB1 to process P1 and moves the process to the appropriate queue, such as the ready queue, I/O queue and waiting queue.
3. After that, another process gets into the running state, or we can select a new process from the ready state, which is to be executed, or the process has a high priority to execute its task.
4. Now, we have to update the PCB (Process Control Block) for the selected process P2. It includes switching the process state from ready to running state or from another state like blocked, exit, or suspend.

5. If the CPU already executes process P2, we need to get the status of process P2 to resume its execution at the same time point where the system interrupt occurs.

- **Need of Context Switching**

1. A context switching helps the operating system that switches between the multiple processes to use the CPU's resource to accomplish its tasks and store its context. We can resume the service of the process at the same point later. If we do not store the currently running process's data or context, the stored data may be lost while switching between processes.
2. If a high priority process falls into the ready queue, the currently running process will be shut down or stopped by a high priority process to complete its tasks in the system.
3. Context switching stores the state of the process to resume its tasks in an operating system. Otherwise, the process needs to restart its execution from the initials level.
4. A context switching allows a single CPU to handle multiple process requests simultaneously without the need for any additional processors.

➢ INTRODUCTION TO THREADS parallelism

Thread is a lightweight process that the operating system can schedule and run concurrently with other threads. The operating system creates and manages threads, and they share the same memory and resources as the program that created them. This enables multiple threads to collaborate and work efficiently within a single program.

The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads.

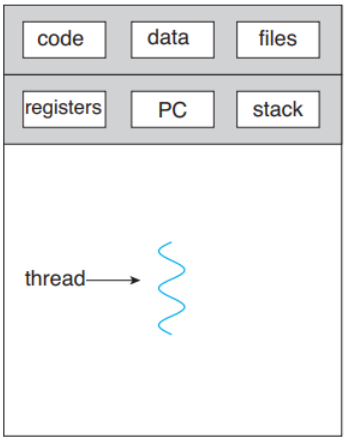


Fig.: Overview of thread

1. Processor registers Every thread works on its designated processor registers. Therefore, when there is a thread switch, these must be saved and restored.

2. Stack Every thread has its own stack. Each frame of the stack stores local variables and returns address of the procedure called by the thread. It may be user stack or kernel stack depending on the type of the thread.
3. PC Every thread has unique program counter value so that they execute independently.
4. Other info: The thread has some other unique information such as its state, priority, and so on.

Advantages of Threads:

- Responsiveness
- Faster Context Switch
- Effective utilization of multiprocessor system
- Resource sharing
- Communication
- Enhanced throughput of the system

➢ TYPES OF THREADS

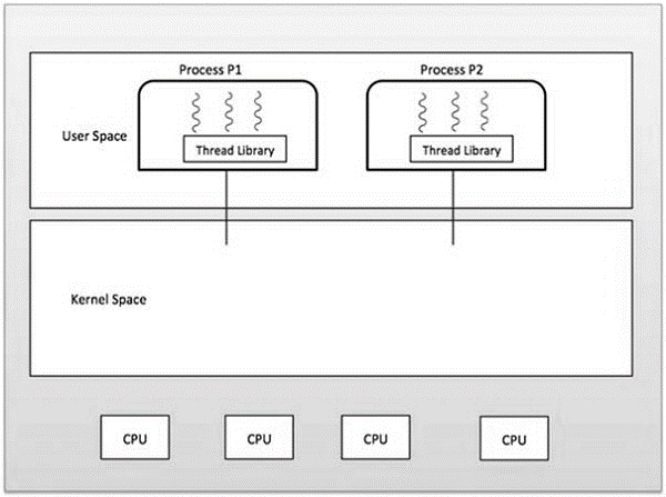


Fig. User Level and Kernel Level Thread

➢ **User-level threads** are managed without kernel support by the run-time system and are supported above the kernel. The kernel does not know anything about the user-level threads. It treats them as if they are single-threaded processes.

POSIX threads

context switching , creating the threads are fast
no hardware
one thread block then it blocks the all threads
easy to use

Advantages:

1. User-level threads are fast to create and efficient.
2. User-level threads are easier to manage than kernel-level threads.
3. The switching between threads takes the same time as a procedural call.

Disadvantages:

1. OS is unaware of user-level threads, so the scheduler cannot schedule them properly.
2. The entire process will get blocked if one user-level thread performs a blocking operation.

3. Multi-processor applications in user-level threads cannot use multiprocessing to their full advantage.

Kernel-level threads are supported and managed by the operating system. No runtime system is required in the case of this type of thread. The kernel has a thread table that keeps track of all threads in the system, rather than having a thread table in each process. In addition to that, the kernel keeps track of processes using a traditional process table. The kernel includes system calls for creating and managing threads.

java and linux

context switching , creating the threads are slow due to os
more hardware
one thread block then no affect on the other process
not easy to use

Advantages

1. The kernel is fully aware of kernel-level threads, so the scheduler handles the process better.
2. The kernel can still schedule another thread for execution if one thread is blocked.
3. This type of thread is suitable for applications that are frequently blocked.

Disadvantages

1. It is slower to create and inefficient.
2. The overhead associated with kernel-level threads requires a thread control block.
3. Kernel-level threads are not generic and are specific to the Operating System.

➢ THREAD MODELS

Multi-threading: It is a process of multiple threads executes at same time. Many operating systems support kernel thread and user thread in a combined way. Example of such system is Solaris. Multi-threading model are of three types.

1. Many to Many Model

In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads. Number of kernel level threads are specific to the machine; advantage of this model is if a user thread is blocked, we can schedule others user thread to other kernel thread. Thus, System doesn't block if a particular thread is blocked.

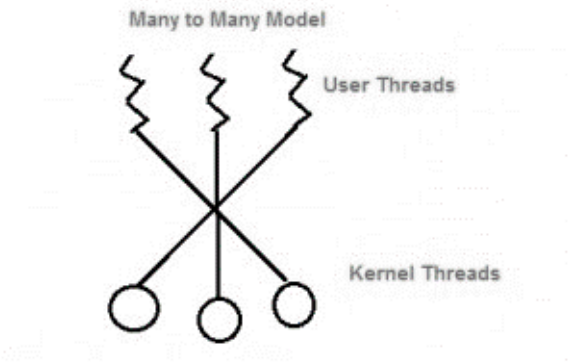


Fig.: Many to many model

2. **Many to One Model**

In this model, we have multiple user threads mapped to one kernel thread. In this model when a user thread makes a blocking system call entire process blocks. As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time. The thread management is done on the user level so it is more efficient.

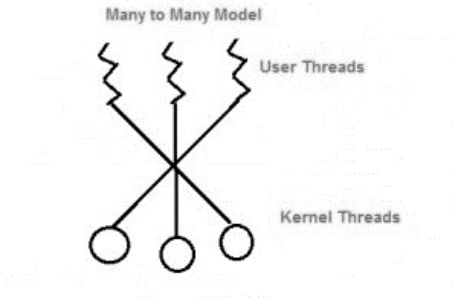


Fig.: Many to one model

3. **One to One Model**

In this model, one to one relationship between kernel and user thread. In this model multiple thread can run on multiple processors. Problem with this model is that creating a user thread requires the corresponding kernel thread. As each user thread is connected to different kernel, if any user thread makes a blocking system call, the other user threads won't be blocked.

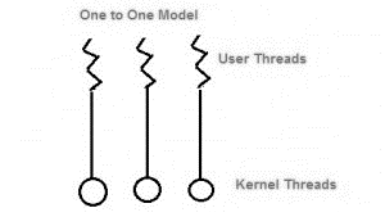


Fig.: One to one model

➢ DIFFERENCE BETWEEN PROCESS AND THREADS

Sr. No.	more time in termination,creation Process	Thread
1.	Process means any program is in execution.	Thread means a segment of a process.
2.	The process takes more time to terminate.	The thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.

4.	It also takes more time for context switching.	It takes less time for context switching.
5.	The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
6.	Multiprogramming holds the concepts of multi-process.	We don't need multi programs in action for multiple threads because a single process consists of multiple threads.
7.	The process is isolated.	Threads share memory.
8.	The process is called the heavyweight process.	A Thread is lightweight as each thread in a process shares code, data, and resources.
9.	Process switching uses an interface in an operating system.	Thread switching does not require calling an operating system and causes an interrupt to the kernel.
10.	The process does not share data with each other.	Threads share data with each other.

➢ BASIC CONCEPTS OF SCHEDULING

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

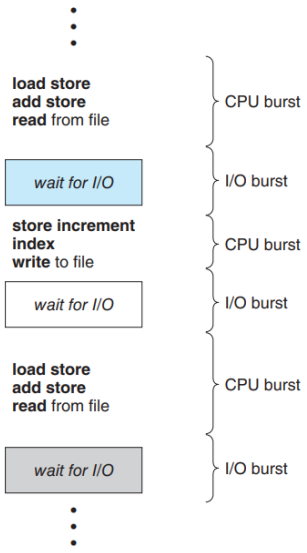


Fig.: Sequence of CPU and IO Burst

The success of CPU scheduling depends on an observed property of processes: 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, 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.

Preemptive scheduling is used when a process switches from the running state to the ready state or from the waiting state to the ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining.

- **Categories of Scheduling**

  There are two categories of scheduling:

1. **Non-pre-emptive:** Here the resource can't be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.
2. **Pre-emptive:** Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

➢ TYPES OF SCHEDULERS, SCHEDULING CRITERIA

1. Long Term Scheduler

It is also called a job scheduler. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

2. Short Term Scheduler

It is also called as CPU scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

3. Mid Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

➢ SCHEDULING CRITERIA

Different CPU scheduling algorithms have different properties and the choice of a particular algorithm depends on various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.

The criteria include the following:

1. **CPU utilization:** The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilization can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.

2. **Throughput:** A measure of the work done by the CPU is the number of processes being executed and completed per unit of time. This is called throughput. The throughput may vary depending on the length or duration of the processes.

3. **Turnaround time:** For a particular process, an important criterion is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU, and waiting for I/O.

   The formula to calculate Turn Around Time = Compilation Time – Arrival Time.

4. **Waiting time:** A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

   The formula for calculating Waiting Time = Turnaround Time – Burst Time.

5. **Response time:** In an interactive system, turn-around time is not the best criterion. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus, another criterion is the time taken from submission of the process of the request until the first response is produced. This measure is called response time.

   The formula to calculate Response Time = CPU Allocation Time (when the CPU was allocated for the first) – Arrival Time

6. **Completion time:** The completion time is the time when the process stops executing, which means that the process has completed its burst time and is completely executed.

7. **Priority:** If the operating system assigns priorities to processes, the scheduling mechanism should favour the higher-priority processes.

➢ SCHEDULING ALGORITHMS

A process scheduler schedules different process to be assigned to the CPU based on particular scheduling algorithm, the different scheduling algorithms are listed below:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time Next (SRTN)
4. Priority Scheduling
5. Round Robin (RR)
6. Multi-Level Queue Scheduling

➢ First Come First Serve (FCFS)

The simplest of all scheduling algorithms is non pre-emptive first-come first-served. With this algorithm, processes are assigned the CPU in the order they request it. Basically, there is a single queue of ready processes.

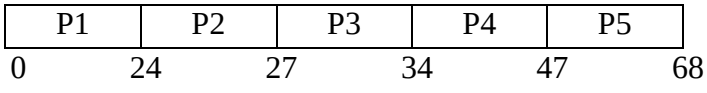
When the running process blocks, the first process on the queue is run next. When a blocked process becomes ready, like a newly arrived job, it is put on the end of the queue.

Example: Consider the following set of processes that arrives at 0, with the length of CPU-burst time given in milliseconds, draw Gantt chart, and calculate average waiting time.

Process	Arrival Time	Burst Time
P1	0	24
P2	1	3
P3	2	7
P4	3	13
P5	4	21

Solution:

1. Gantt chart



2. Average Waiting time is = (0+24+27+34+47)/5 = 132/5= 26.4 milliseconds.

Advantages:

- The simplest and basic form of CPU Scheduling algorithm.
- Easy to implement.
- First come first serve method.
- It is well suited for batch systems where the longer time periods for each process are often acceptable.

Disadvantages:

- As it is a Non-pre-emptive CPU Scheduling Algorithm, hence it will run till it finishes the execution.
- The average waiting time in the FCFS is much higher than in the others
- It suffers from the Convoy effect.
- Not very efficient due to its simplicity.

- **Shortest Job First (SJF)**

The shortest job first (SJF) is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJF, also known as Shortest Job Next (SJN), can be pre-emptive or non-pre-emptive.

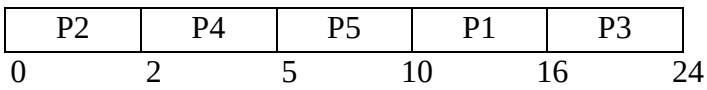
It is practically infeasible as Operating System may not know burst times and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times.

Example: Consider the following set of processes that arrives at 0, with the length of CPU-burst time given in milliseconds, draw Gantt chart, and calculate average waiting time.

Process	Burst Time
P1	6
P2	2
P3	8
P4	3
P5	4

Solution:

1. Gantt chart



2. Average Waiting time is = (0+2+5+10+16)/5 = 33/5= 6.6 milliseconds.

Advantages:

- SJF is better than the First come first serve (FCFS) algorithm as it reduces the average waiting time.
- SJF is generally used for long term scheduling.
- It is suitable for the jobs running in batches, where run times are already known.
- SJF is probably optimal in terms of average turnaround time.

Disadvantages:

- SJF may cause very long turn-around times or starvation.
- In SJF job completion time must be known earlier, but sometimes it is hard to predict.
- Sometimes, it is complicated to predict the length of the upcoming CPU request.
- It leads to the starvation that does not reduce average turnaround time.

- **Shortest Remaining Time Next (SRTN)**

A Shortest remaining Time Next scheduling algorithm is also referred as pre-emptive SJF scheduling algorithm. When a new process arrives at ready queue while one process is still executing then SRTN algorithm is performed to decide which process will execute next.

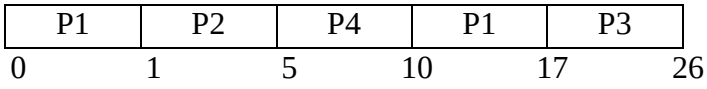
This algorithm compare CPU burst time of newly arrived process with remaining (left) CPU burst time of currently executing process. If CPU burst time of new process is less than remaining time of current process then SRTN algorithm pre-empts current process execution and starts executing new process.

Example: Consider the following set of processes that arrives at 0, with the length of CPU-burst time given in milliseconds, draw Gantt chart, and calculate average waiting time.

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

Solution:

Gantt chart



Average Waiting time is = (0+2+15+9)/4 = 26/5= 5.2 milliseconds.

Advantages:

- SRTF algorithm makes the processing of the jobs faster than SJN algorithm, given its overhead charges are not counted.

Disadvantages:

- The context switch is done a lot more times in SRTF than in SJN, and consumes CPU's valuable time for processing. This adds up to its processing time and diminishes its advantage of fast processing.

➢ **Round Robin Scheduling**

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the pre-emptive version of first come first serve scheduling.

The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum.

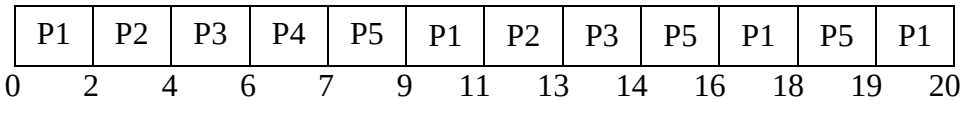
Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

Example: Consider the following set of processes that arrives at 0, with the length of CPU-burst time given in milliseconds, draw Gantt chart, and calculate average waiting time. Quantum =2

Process	Arrival Time	Burst Time
P1	0	7
P2	1	4
P3	2	3
P4	3	1
P5	4	2

Solution:

Gantt chart



Average Waiting time is = (12+8+8+1+9)/4 = 38/5= 7.6 milliseconds.

Advantages:

- It can be actually implementable in the system because it is not depending on the burst time.
- It doesn't suffer from the problem of starvation or convoy effect.
- All the jobs get a fare allocation of CPU.

Disadvantages:

- The higher the time quantum, the higher the response time in the system.
- The lower the time quantum, the higher the context switching overhead in the system.
- Deciding a perfect time quantum is really a very difficult task in the system.

- **Priority Scheduling**

In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority.

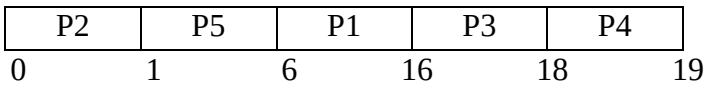
The Process with the higher priority among the available processes is given the CPU. There are two types of priority scheduling algorithm exists. One **is** Pre-emptive priority scheduling while the other is Non Pre-emptive Priority scheduling.

Example: Consider the following set of processes that arrives at 0, with the length of CPU-burst time given in milliseconds, draw Gantt chart, and calculate average waiting time.

Process	Priority	Burst Time
P1	3	10
P2	1	1
P3	4	2
P4	5	1
P5	2	5

Solution:

3. Gantt chart



4. Average Waiting time is = (0+1+6+16+18)/5 = 41/5= 8.2 milliseconds.

Advantages:

- to use scheduling method
- Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
- Suitable for applications with fluctuating time and resource requirements.

Disadvantages:

- If the system eventually crashes, all low priority processes get lost.
- If high priority processes take lots of CPU time, then the lower priority processes may starve and will be postponed for an indefinite time.
- This scheduling algorithm may leave some low priority processes waiting indefinitely.

- **Multilevel Queue Scheduling**

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, 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. 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.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the most general scheme, it is also the most complex.

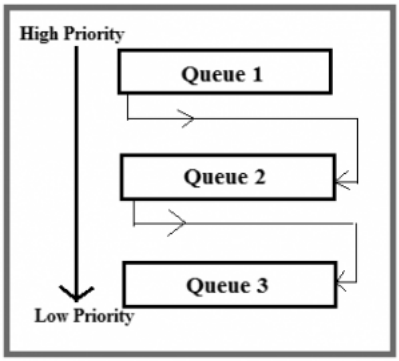


Fig.: Multilevel Feedback Queue Scheduling

Advantages

- This is a flexible Scheduling Algorithm.
- This scheduling algorithm allows different processes to move between different queues.
- In this algorithm, A process that waits too long in a lower priority queue may be moved to a higher priority queue which helps in preventing starvation.

Disadvantages

- This algorithm is too complex.
- As processes are moving around different queues which leads to the production of more CPU overheads.
- In order to select the best scheduler this algorithm requires some other means to select the values