Process and Thread Management

Process is a program in execution. It is the unit of work in a modern time-sharing system. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers. A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables. A process may also include a heap, which is memory that is dynamically allocated during process run time. The structure of a process in memory is shown in following Figure:

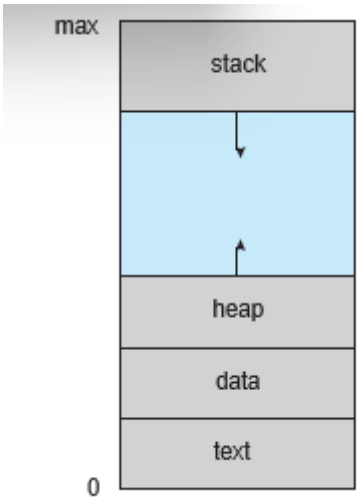


Figure. Process in memory

A program is a passive entity, such as a file containing a list of instructions stored on disk (called an executable file). In contrast, a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

Process State:

A process may be in one of the following states:

- **New**. The process is being created.
- **Running**. Instructions are being executed.
- **Waiting**. The process is waiting for some event to occur (such as an I/Ocompletion or reception of a signal).
- **Ready**. The process is waiting to be assigned to a processor.
- **Terminated**. The process has finished execution.

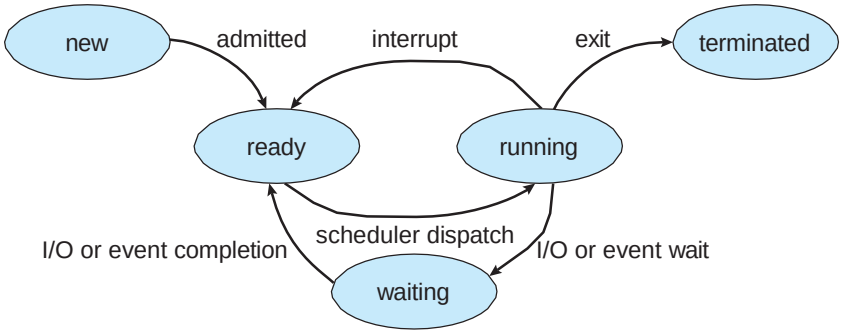


Figure Process States

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 as shown in following Figure:

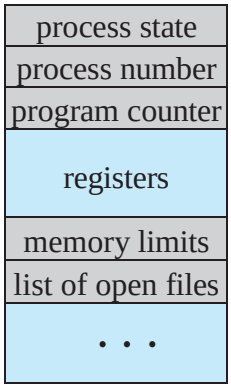


Figure. Process control block (PCB).

- **Process state.** The state may be new, ready, running, waiting, halted, and so on.
- **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, 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 (Figure 3.4).
- **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters. (Chapter 6 describes process scheduling.)
- **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.
- **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.

Process Scheduler:

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 (possibly from a set of several available processes) for program execution on the CPU. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. This 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. In addition to this there may be some processes waiting for I/O request to a shared device, such as a disk. The list of processes waiting for a particular I/O device is called a device queue as shown in following figure.

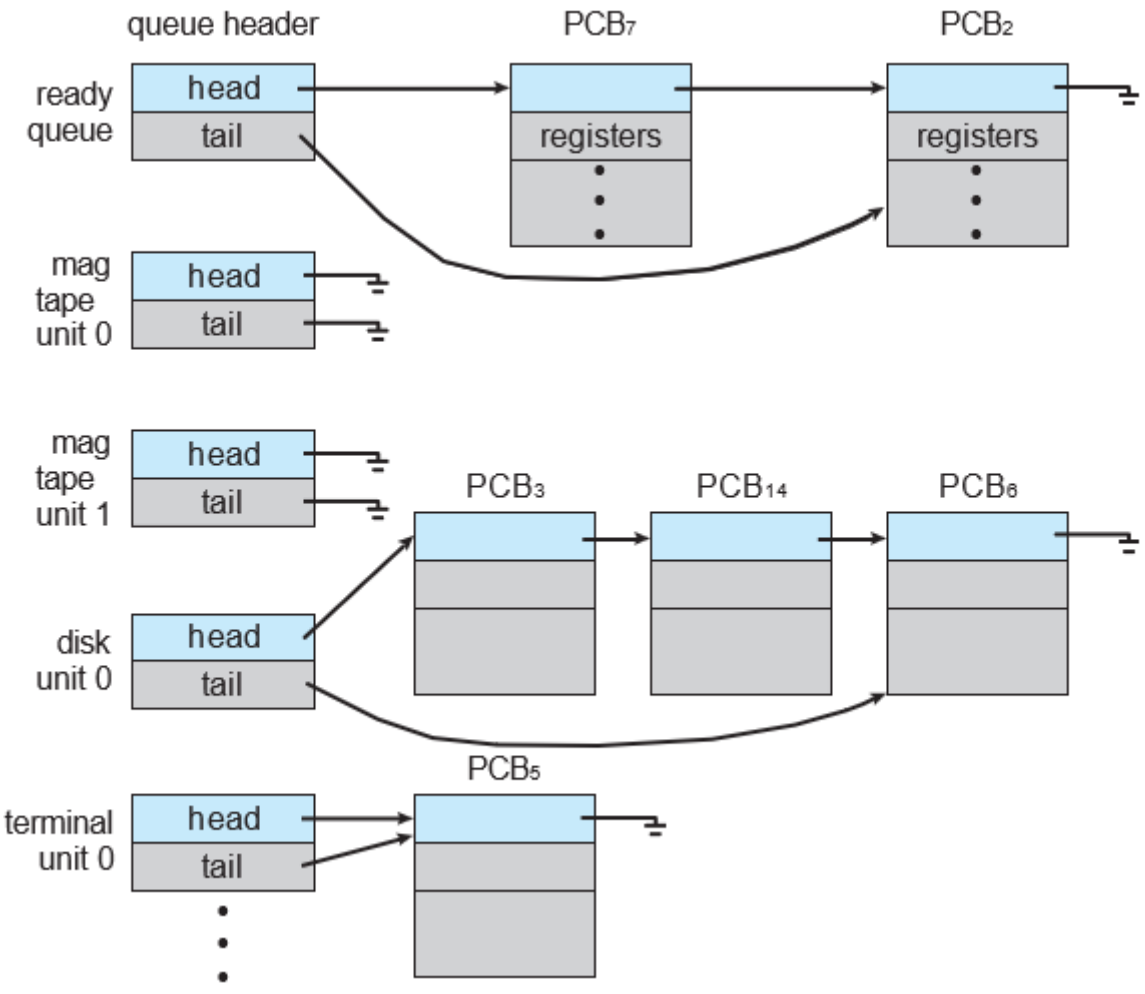


Figure The ready queue and various I/O device queues.

A common representation of process scheduling is a queueing diagram, such as that in following Figure.

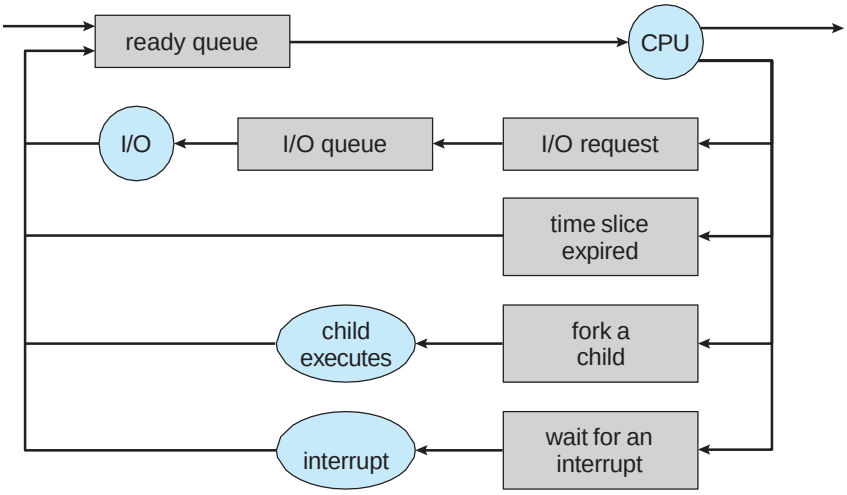


Figure. Queueing-diagram representation of process scheduling.

Here each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.

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.

Schedulers are special system software which handle process scheduling in various ways.

Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

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

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.

- On some systems, the long-term scheduler may not be available or minimal.
- Time-sharing operating systems have no long term scheduler.
- When a process changes the state from new to ready, then there is use of long-term scheduler.

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.

Medium 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.
- A running process may become suspended if it makes an I/O request.
- A suspended process cannot make any progress towards completion.
- In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage.
- This process is called swapping, and the process is said to be swapped out or rolled out.
- Swapping may be necessary to improve the process mix.

Context Switch

Switching the CPU 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. 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. Switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions.

- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time.
- Using this technique, a context switcher enables multiple processes to share a single CPU.
- Context switching is an essential part of a multitasking operating system features.
- When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block.
- After this, the state for the process to run next is loaded from its own PCB and used to

set the PC, registers, etc.

- At that point, the second process can start executing.

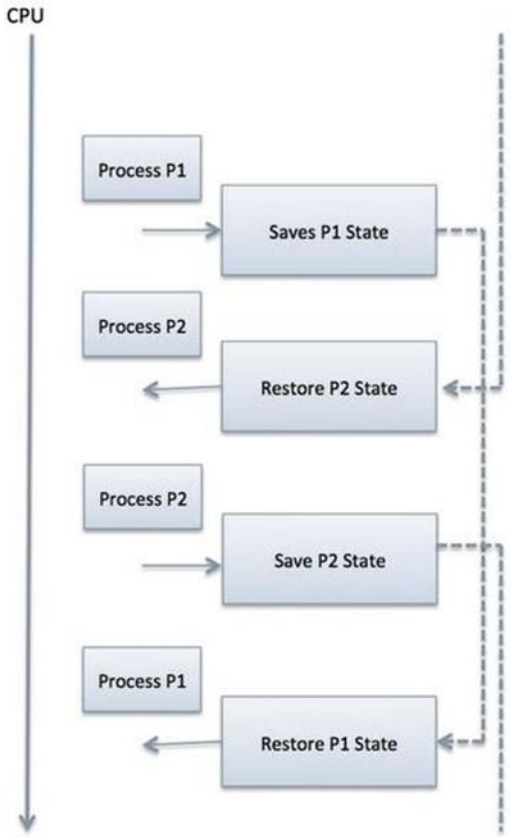


Figure. Context Switch

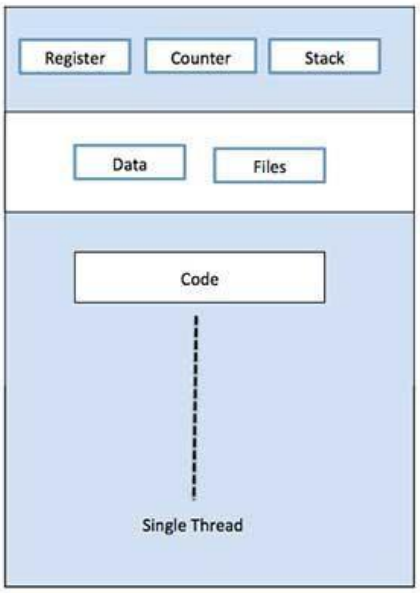
- Context switches are computationally intensive since register and memory state must be saved and restored.
- To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers.
- When the process is switched, the following information is stored for later use.
  - Program Counter
  - Scheduling information
  - Base and limit register value
  - Currently used register
  - Changed State
  - I/O State information
  - Accounting information

Threads:

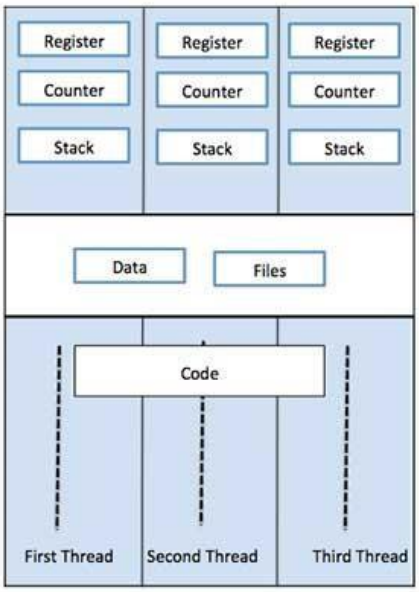
- A thread is a flow of execution through the process code, with its own program

counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

- A thread shares with its peer threads few information like code segment, data segment and open files.
- When one thread alters a code segment memory item, all other threads see that.
- A thread is also called a **lightweight process**.
- Threads provide a way to improve application performance through parallelism.
- Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.
- Threads have been successfully used in implementing network servers and web server.
- They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.
- The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Difference between Process and Thread:

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Threads:

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

CPU Scheduling and Algorithm:

CPU and I/O Burst Cycle:

- 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, followed by an I/O burst, then another CPUburst ...etc
- The last CPU burst will end with a system request to terminate execution rather thanwith another I/O burst.
- The duration of these CPU burst have been measured.
- An I/O-bound program would typically have many short CPU bursts, A CPU-

boundprogram might have a few very long CPU bursts.

- This can help to select an appropriate CPU-scheduling algorithm.

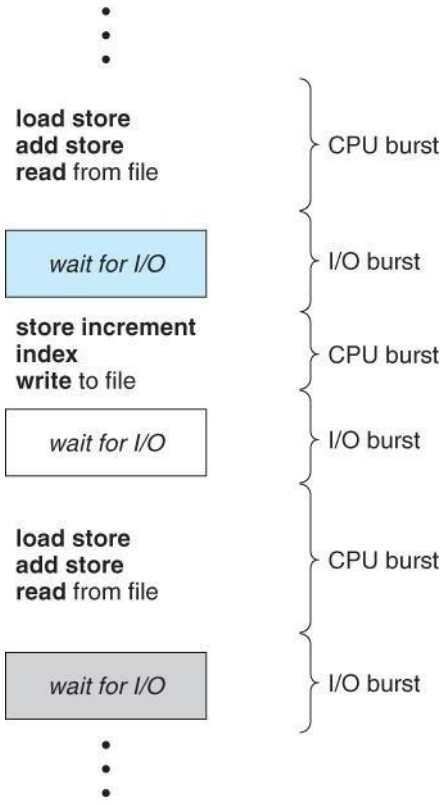


Figure. CPU and I/O Burst Cycle

Preemptive Scheduling:

- Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state.
- The resources (mainly CPU cycles) are allocated to the process for the limited amountof time and then is taken away, and the process is again placed back in the ready queueif that process still has CPU burst time remaining.
  That process stays in ready queue till it gets next chance to execute.

Non-Preemptive Scheduling:

- Non-preemptive Scheduling is used when a process terminates, or a process switchesfrom running to waiting state.
- In this scheduling, once the resources (CPU cycles) is allocated to a process, the processholds the CPU till it gets terminated or it reaches a waiting state.
- In case of non-preemptive scheduling does not interrupt a process running CPU inmiddle of the execution.
- Instead, it waits till the process complete its CPU burst time and then it can allocate theCPU to another process.

Basis for Comparison	Preemptive Scheduling	Non Preemptive Scheduling
Basic	The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
Starvation	If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Overhead	Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Flexibility	Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Cost	Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

Scheduling Criteria:

- There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:

1. **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from40% (lightly loaded) to 90% (heavily loaded.)
2. **Throughput** - Number of processes completed per unit time. May rangefrom 10 / second to 1 / hour depending on the specific processes.
3. **Turnaround time** - Time required for a particular process to complete,from submission time to completion.
4. **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.
5. **Response time** - The time taken in an interactive program from the issuance of a command to the ***commence*** of a response to that command.

- **Arrival Time:** Time at which the process arrives in the ready queue.
- **Completion Time:** Time at which process completes its execution.
- **Burst Time:** Time required by a process for CPU execution.
- **Turn Around Time:** Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
- **Waiting Time (W.T):** Time Difference between turnaround time and burst time. Waiting Time = Turn Around Time – Burst Time

Types of Scheduling Algorithm

a) First Come First Serve (FCFS)

In FCFS Scheduling:

- The process which arrives first in the ready queue is firstly assigned the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always non-preemptive in nature.
- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Advantages-

- It is simple and easy to understand.
- It can be easily implemented using queue data structure.
- It does not lead to starvation.

Disadvantages-

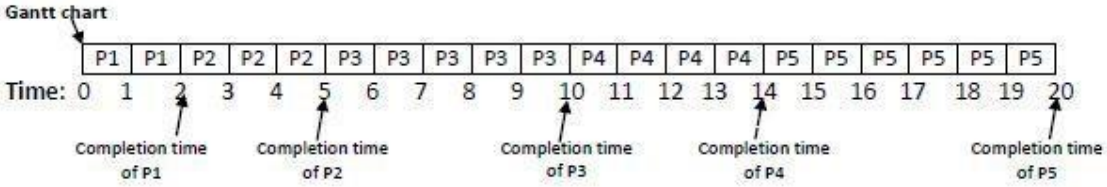
- It does not consider the priority or burst time of the processes.
- It suffers from convoy effect i.e. processes with higher burst time arrived beforethe processes with smaller burst time.

Example 1:

Q. Consider the following processes with burst time (CPU Execution time). Calculate the average waiting time and average turnaround time?

Process id	Arrival time	Burst time/CPU execution time
P1	0	2
P2	1	3
P3	2	5
P4	3	4
P5	4	6

Sol.



Turnaround time= Completion time – Arrival time

Waiting time= Turnaround time – Burst time

Process id	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P1	0	2	2	2-0=2	2-2=0
P2	1	3	5	5-1=4	4-3=1
P3	2	5	10	10-2=8	8-5=3
P4	3	4	14	14-3=11	11-4=7
P5	4	6	20	20-4=16	16-6=10

Average turnaround time= $\sum_{i=0}^{n} \text{Turnaround time(i)}/n$ where, n= no. of process

Average waiting time= $\sum_{i=0}^{n} \text{Wating time(i)}/n$ where, n= no. of process

Average turnaround time= 2+4+8+11+16/5 =41/5 =8.2

Average waiting time= 0+1+3+7+10/5 = 21/5 =4.2

b) Shortest Job First (SJF)

- Process which have the shortest burst time are scheduled first.
- If two processes have the same bust time, then FCFS is used to break the tie.
- This is a non-pre-emptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is notknown.
- The processer should know in advance how much time process will take.
- Pre-emptive mode of Shortest Job First is called as Shortest Remaining TimeFirst (SRTF).

Advantages-

- SRTF is optimal and guarantees the minimum average waiting time.
- It provides a standard for other algorithms since no other algorithm performsbetter than it.

Disadvantages-

- It can not be implemented practically since burst time of the processes can not beknown in advance.
- It leads to starvation for processes with larger burst time.
- Priorities can not be set for the processes.
- Processes with larger burst time have poor response time.

Example:

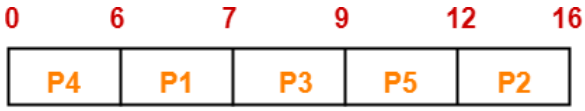
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

Solution-

If the CPU scheduling policy is SJF non-preemptive, calculate the average waitingtime and average turnaround time.

Gantt Chart-



Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

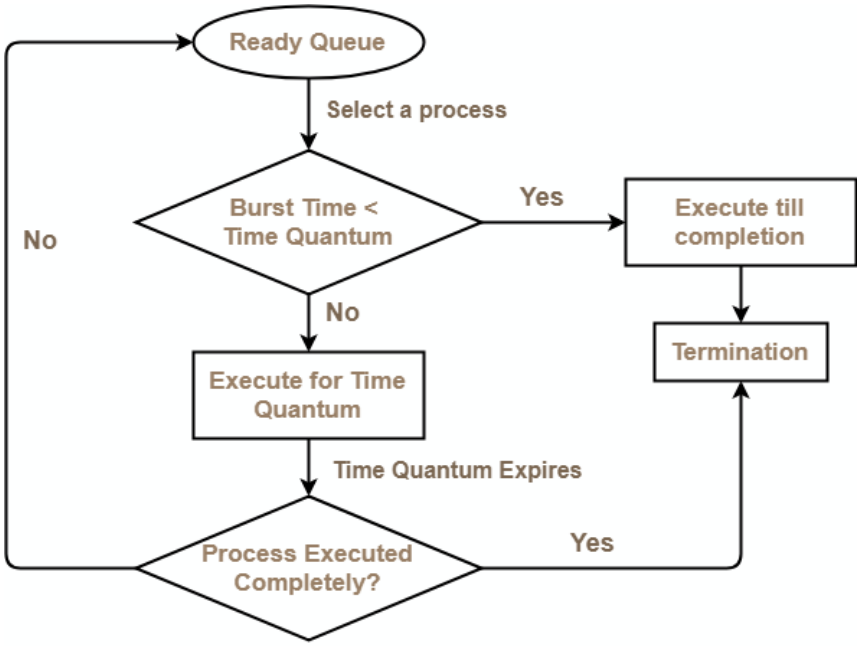
Process Id	Exit time	Turn Around time	Waiting time
P1	7	7 – 3 = 4	4 – 1 = 3
P2	16	16 – 1 = 15	15 – 4 = 11
P3	9	9 – 4 = 5	5 – 2 = 3
P4	6	6 – 0 = 6	6 – 6 = 0
P5	12	12 – 2 = 10	10 – 3 = 7

Now,

- Average Turn Around time = (4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8 unit
- Average waiting time = (3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8 unit

c) Round Robin Scheduling:

- CPU is assigned to the process on the basis of FCFS for a fixed amount of time.
- This fixed amount of time is called as time quantum or time slice.
- After the time quantum expires, the running process is preempted and sent to theready queue.
- Then, the processor is assigned to the next arrived process.
- It is always preemptive in nature.



Round Robin Scheduling

Advantages-

- It gives the best performance in terms of average response time.
- It is best suited for time sharing system, client server architecture andinteractive system.

Disadvantages-

- It leads to starvation for processes with larger burst time as they have to repeatthe cycle many times.
- Its performance heavily depends on time quantum.
- Priorities can not be set for the processes.

With decreasing value of time quantum,

- Number of context switch increases
- Response time decreases
- Chances of starvation decreases

Thus, smaller value of time quantum is better in terms of response time.

With increasing value of time quantum,

- Number of context switch decreases
- Response time increases
- Chances of starvation increases

Thus, higher value of time quantum is better in terms of number of context switch.

With increasing value of time quantum, Round Robin Scheduling tends to become FCFS Scheduling.

- When time quantum tends to infinity, Round Robin Scheduling becomes FCFSScheduling.
- The performance of Round Robin scheduling heavily depends on the value oftime quantum.
- The value of time quantum should be such that it is neither too big nor too small.

The average waiting time under the RR policy is often long. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Example:

Process	Burst Time
P1	24
P2	3
P3	3

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

Let's calculate the average waiting time for this schedule. P1 waits for 6 milliseconds (10 - 4), P2 waits for 4 milliseconds, and P3 waits for 7 milliseconds**. Thus, the average waiting time is 17/3 = 5.66 milliseconds.**

In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row (unless it is the only runnable process). If a process's CPU burst exceeds 1 time quantum, that process is preempted and is put back in the ready queue. The RR scheduling algorithm is thus preemptive. If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units. Each process must wait no longer than (n 1) q time units until its next time quantum. For example, with five processes and a time quantum of 20 milliseconds, each process will get up to 20 milliseconds every 100 milliseconds.

The performance of the RR algorithm depends heavily on the size of the time quantum. At one extreme, if the time quantum is extremely large, the RR policy is the same as the FCFS policy. In contrast, if the time quantum is extremely small (say, 1 millisecond), the RR approach can result in a large number of context switches.

Although the time quantum should be large compared with the context- switch time, it should not be too large. As pointed out earlier, if the time quantum is too large, RR scheduling degenerates to an FCFS policy. A rule of thumb is that 80 percent of the CPU bursts should be shorter than the time quantum.