CHAPTER 2: Process Management

- Introduction to Process
  - Process description
  - Process states
  - Process control
- Threads
- Processes and Threads
- Scheduling
  - Types of scheduling
  - Scheduling in batch system
  - Scheduling in Interactive System
  - Scheduling in Real Time System
  - Thread Scheduling
- Multiprocessor Scheduling concept

Process:

- A process is defined as an entity which represents the basic unit of work to be implemented in the system i.e. a process is a program in execution.
- The execution of a process must progress in a sequential fashion.
- In general, a process will need certain resources such as the CPU time, memory, files, I/O devices and etc. to accomplish its task. As a process executes, it changes state.

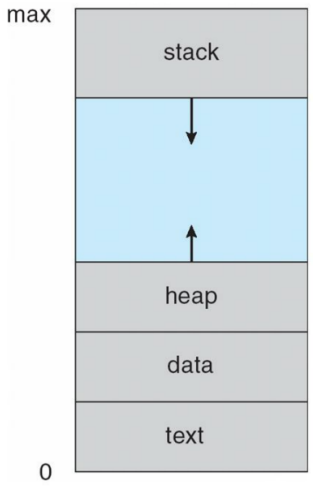


Fig: Sections of process in OS

- **Stack**: stores temporary data such as functions, parameters, local variables.
- **Heap**: dynamically allocates memory to a process during its run time.
- **Data**: value of PC and the contents of the processor's registers.
- **Text**: global and static variables.

Program VS Process

There is a hairline difference between the program and process in the sense that a program is a passive entity that does not initiate anything by itself whereas a process is an active entity that performs all the actions specified in a particular program.

Program	Process
1. It is a set of instructions.	1. It is a program in execution.
2. It is a passive entity.	2. It is an active entity.
3. It has longer lifespan.	3. It has limited lifespan.
4. A program needs memory space on disk to store all instructions.	4. A process contains many resources like a memory address, disk, printer, etc.
5. Program is loaded into secondary storage device.	5. Process is loaded into main memory.
6. It is a static object existing in a file form.	6. It is a dynamic object (i.e., program in execution)
7. No such duplication is needed.	7. New processes require duplication of parent process.

Operation on Process

1. Process Creation
2. Destroy of a process
3. Run a process
4. Change a process priority
5. Get process information
6. Set process information

Process Creation

- When a new process is to be added to those currently being managed, the OS builds the date structures that are used to manage the process and allocates address space in main memory to the process. These actions constitute the creation of a new process.
- There are four ways achieving it:
  1. For a batch environment a process is created in response to submission of a job.
  2. In Interactive environment, a process is created when a new user attempt to log on.
  3. The OS can create process to perform functions on the behalf a user program.
  4. A number of process can be generated from the main process. For the purpose of modularity or to exploit parallelism a user can create numbers of process.

- A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. Each of these new processes may in turn create other processes forming a tree of processes.
- A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files and environment strings. However, they have distinct address spaces.
- **When a process creates new process, two possibilities exists in terms of execution:**
  - The parent continues to execute concurrently with its children.
  - The parent waits until some or all of its children have terminated.
- There are also two address-space possibilities for the new process:

- The child process is a duplicate of the parent process (it has the same program and data as the parent).
- The child process has a new program loaded into it.

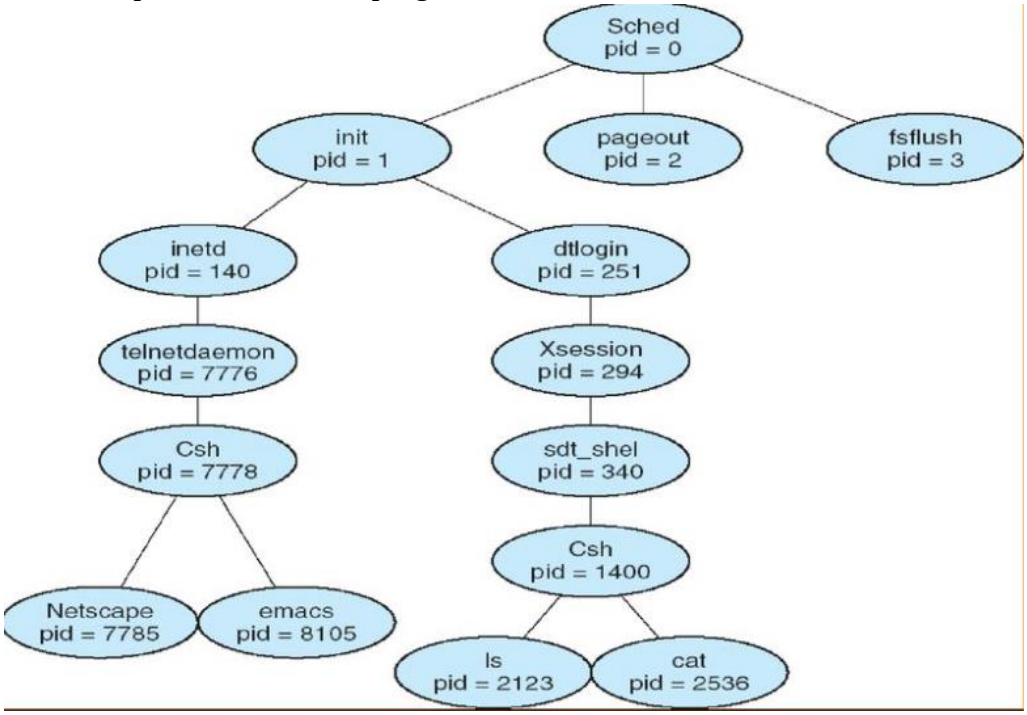
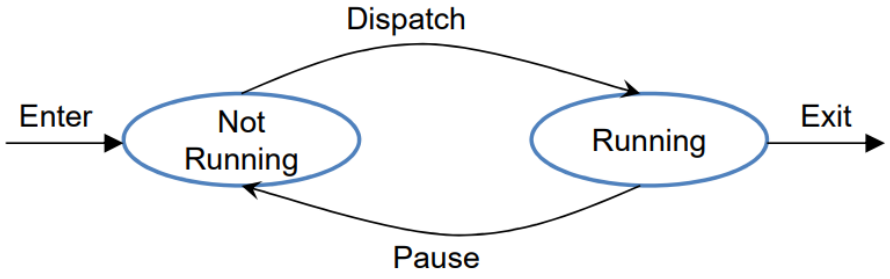


Fig: A tree of processes in Solaris System

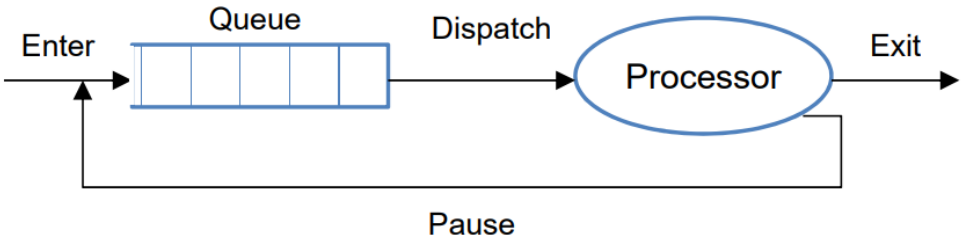
Process Termination

- A process terminates when it finishes executing its last statement.
- Its resources are returned to the system, it is purged from any system lists or tables, and its process control block (PCB) is erased.
- All the resources of the process including physical and virtual memory, open files and I/O buffers are deallocated by Operating System.
- The new process terminates the existing process, usually due to following reasons:
  - **Normal Exit (Voluntary):** Most processes terminate because they have done their job.
  - **Fetal Error (Involuntary):** When process discovers a fatal error. For example, a user tries to compile a program that does not exist.
  - **Error Exit (Voluntary):** An error caused by process due to a bug in program for example, executing an illegal instruction, referring non-existing memory or dividing by zero.
  - **Killed by another Process (Industrial):** A process executes a system call telling the Operating Systems to terminate some other process. In UNIX, this call is kill-9. In some systems when a process kills all processes it created are killed as well (UNIX does not work this way).

Process State (Two state model)



(a) State transition diagram

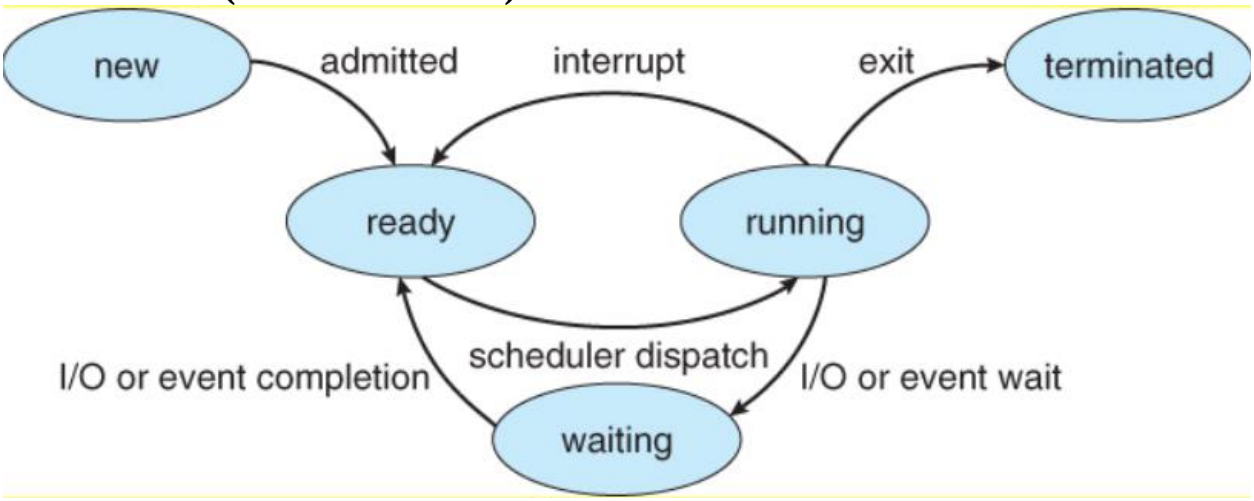


(b) Queuing diagram

- Two State Process Model consists of two states:
  - **Not-running State:** Process waiting for execution.
  - **Running State:** Process currently executing.

- First of all, when a new process is created, then it is in "Not Running State".
- When CPU becomes free, Dispatcher gives control of the CPU to that process which is in "NOT running state" and waiting in a queue.
- Dispatcher is a program that gives control of the CPU to the process selected by the CPU scheduler. Suppose dispatcher allow PROC to execute on CPU.
- When dispatcher allows PROC to execute on CPU then it starts its execution. Here we can say that PROC is in running state.
- Now, if any process PROC2 with high priority wants to execute on CPU, then will be in waiting state and PROC2 will be in running state. Now, when PROC2 terminates then PROC again allows the dispatcher to execute on CPU.

Process State (Five State Model)



The five states in this model are:

- **New**: A newly created process that has not yet been admitted to the pool of executable processes by the OS.
- **Ready**: Processes that prepared to execute when given opportunity.
- **Running**: The process is currently being executed.
- **Waiting**: The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
- **Terminated:** Process has been completed.

Process Control Block (PCB)

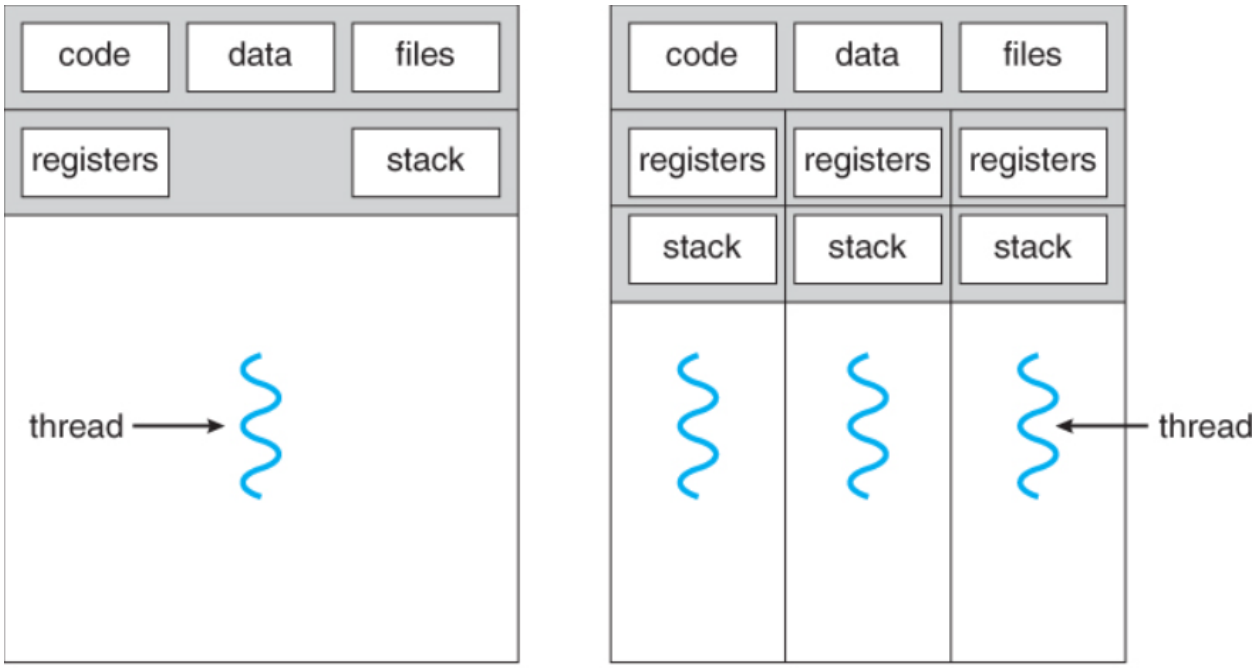
- A Process Control Block is a data structure (process table) maintained by the OS that holds information for every process.
- A PCB stores descriptive information pertaining to a process such as its state, program counter, memory management information, information about its scheduling, allocated resources, accounting information etc. that is required to control and manage a particular process.
- The basic purpose of PCB is to indicate the 'so far' progress of a process.
- The PCB is maintained for a process throughout its lifetime and is deleted once the process terminates.

- The process control block typically contains following information:

- **Process ID**: number that identifies the process
- **Process State**: stores current states of a process that can be new, ready, running, waiting or terminated.
- **Parent process ID:** stores PID of parent.
- **Child process ID:** stores PID of all child processes of a parent class.
- **Program counter:** contains address of instruction that is to be executed next in the process.
- **Event information:** If the process is in waiting state, this field contains information about the event for which the process is waiting to happen. For e.g.: if the process is waiting for I/O device, this field stores the ID of that device.
- **Memory management information:** includes information related to memory configuration for a process.
- **CPU registers:** stores contents of registers at the time when the CPU was last released by process.
- **CPU Scheduling information:** includes information used by the scheduling algorithms.
- **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:

- Thread is a single sequential flow of execution of tasks of a process. It is also known as thread of execution or thread of control.
- It is the fundamental unit of CPU utilization consisting of a program counter, stack and set of registers.
- Threads have some of the properties of processes, they are sometimes called lightweight processes.
- Each thread belongs to exactly one process and outside a process no threads exist.
- The process can be split down into so many threads. For example, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.
- Like a traditional process i.e., process with one thread, a thread can be in any of several states (Running, Blocked, Ready or terminated).



single-threaded process multithreaded process

Advantages of threads:

- Threads are cheap to create and use very little resources.
- Since different threads of a process shares the same address space, they can communicate with each other via shared memory which reduces extra overhead.
- Context switching is fast (only have to save/reload PC, Stack, SP, Registers).
- It takes less time to terminate a thread than a process.
- Process can be executed speedily by creating multiple threads in the process and executing them in a quasi-parallel manner (i.e., by rapidly switching CPU among multiple threads.)

Similarities between process and thread

- Like processes, threads share CPU and only one thread active (running) at a time.
- Like processes, threads within processes execute sequentially.
- Like processes, thread can create children.
- Like process, if one thread is blocked, another thread can run

Difference between Process and thread

Process	Thread
1. heavy weight	1. light weight
2. It takes more time to terminate.	2. It takes less time to terminate.
3. More resources are required.	3. Less resources are required.
4. More time required for creation.	4. Less time required for creation.
5. System call is involved in process.	5. System call not involved in threads.

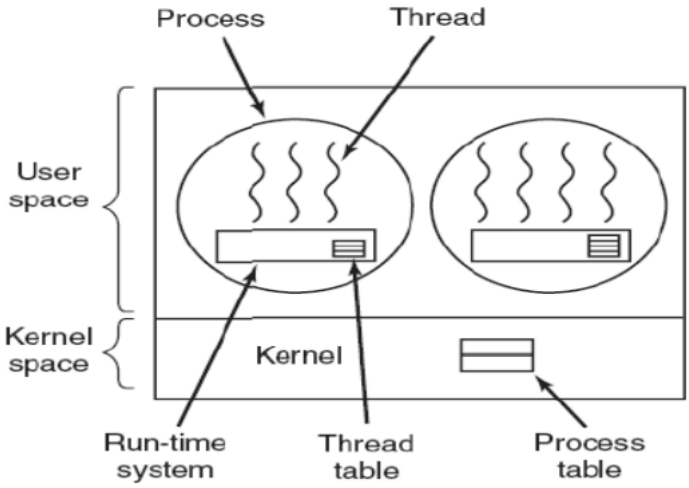
6. If one server process is blocked then other server processes cannot execute until the first process is unblocked.	6. If one thread is blocked and waiting then the second thread in the same task can run.
7. Slow Execution	7. Fast Execution
8. Doesn't share memory (loosely coupled)	8. Shares memories and files (tightly coupled)
9. Takes more time to switch between processes.	9. Takes less time to switch between threads.

Implementation of threads

- Threads can be implemented in different ways depending on the extent to which the process and OS knows about them. It can be implemented in following ways:
  - **User Level Threads**: User managed threads
  - **Kernel Level Threads**: Operating System managed threads acting on kernel, an operating system core

User Level Threads:

- Users are the ones managing these threads. The thread managing kernel is not aware of this thread's existence.
- The thread library has codes for creating, passing messages and data, scheduling execution, shaving and restoring, and destroying threads. These are used at the application level and have no involvement of the OS.



Advantages:

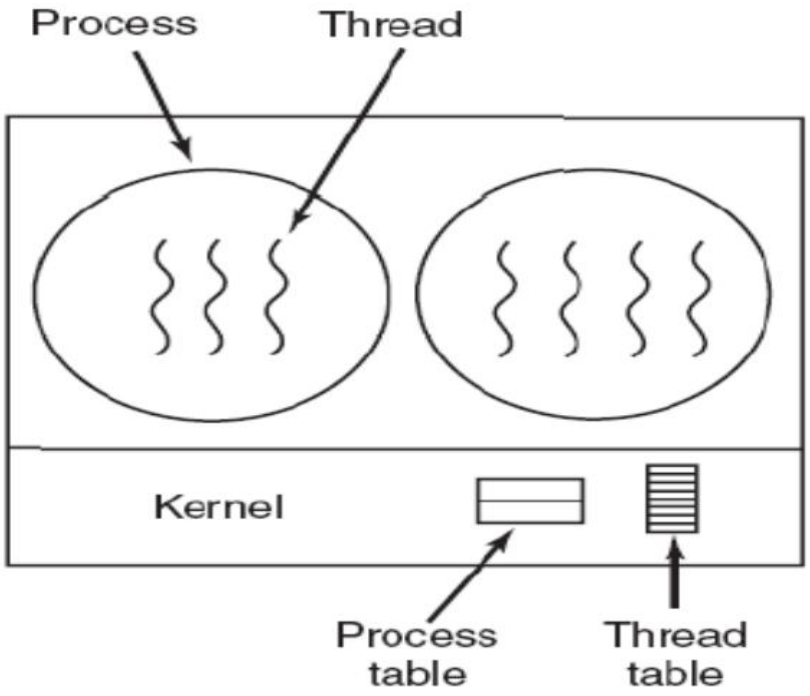
- User level threads can be created and managed faster than kernel level threads.
- Can run on any OS.
- User-level thread does not require modification to operating systems.
- **Simple Representation:** Each thread is represented simply by a PC, registers, stack and a small control block, all stored in the user process address space.
- **Simple Management:** This simply means that creating a thread, switching between threads and synchronization between threads can all be done without intervention of the kernel.
- **Fast and Efficient:** Thread switching is not much more expensive than a procedure call.

Disadvantages:

- User-level threads lack coordination between the thread and the kernel.
- If a thread causes a page fault, the entire process is blocked.

Kernel level threads:

- In this method, the kernel knows about and manages the threads.
- Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system.
- Scheduling by the Kernel is done on a thread basis.
- The Kernel performs thread creation, scheduling and management in Kernel space.
- Kernel threads are generally slower to create and manage than the user threads.



Advantages:

- Kernel can simultaneously schedule multiple threads
- If one thread gets blocked, another can be scheduled.
- Kernel routines multithread themselves.

Disadvantages:

- Slower to create and manage
- Transfer of control within a process requires a mode switch to the Kernel.

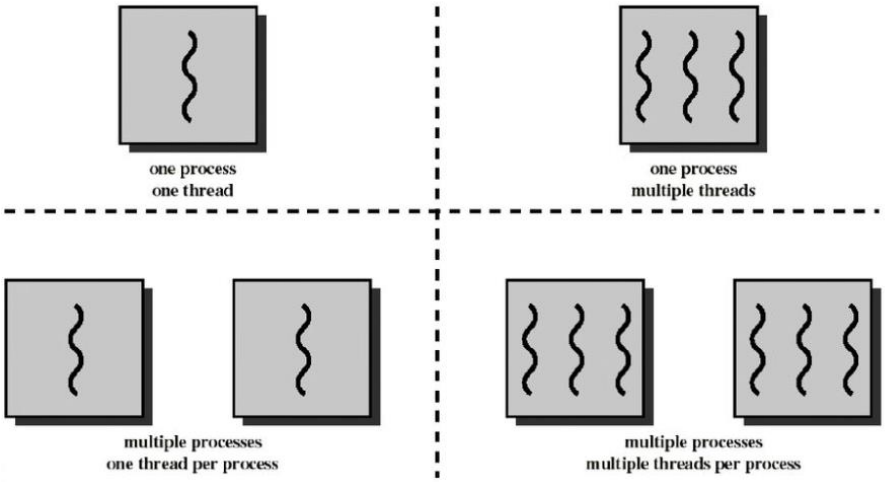
User level threads VS Kernel level threads

User level threads	Kernel level threads
1. They are faster to create and manage.	1. They are slower to create and manage.
2. More efficient	2. Less efficient
3. Context switching time is less.	3. Context switching time is more.
4. It can run on any OS.	4. They are specific to OS.

5. It cannot take full advantage of multiprocessing.	5. It takes full advantage of multiprocessing.
6. They are managed without kernel support by run time system.	6. They are managed and supported by Operating system.
7. Implementation of User threads is easy.	7. Implementation of kernel threads is complicated.

Multithreading:

- The use of multiple threads in a single program, all running at the same time and performing different tasks is known as multithreading.
- Multithreading is a type of execution model that allows multiple threads to exist within the context of a process such that they execute independently but share their process resources.
- It enables the processing of multiple threads at one time, rather than multiple processes.
- Based on functionality, threads are put under 4 categories:
  - Single Process, Single Thread (MS-DOS)
  - Single Process, Multiple Threads (JAVA Runtime)
  - Multiple Process, Single Thread per process (UNIX)
  - Multiple Process, Multiple Threads (Solaris, UNIX)



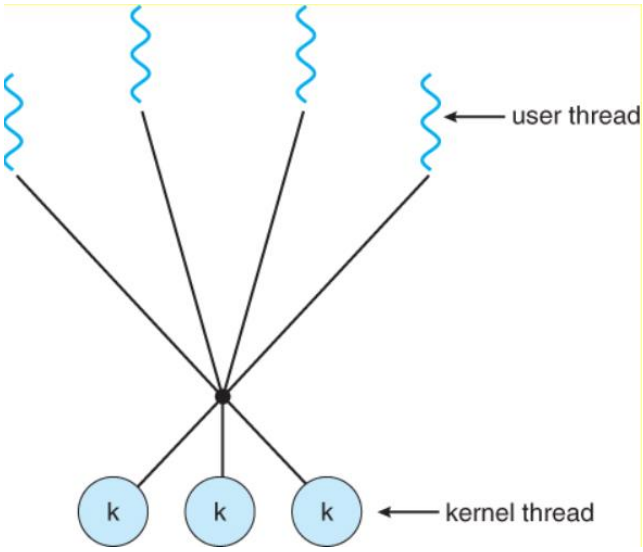
Multithreading Models:

- Many system supports hybrid thread model that contains both user and kernel level threads along with a relationship between these threads.
- Multithreading models are three types:
  - Many to many relationships
  - Many to one relationship
  - One to one relationship

Many to Many relationship:

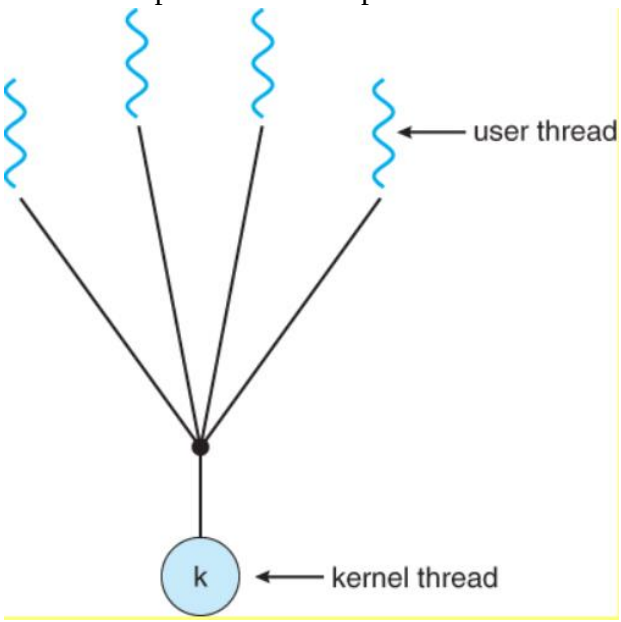
- Many-to-many model multiplexes equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users have no restrictions on the number of threads created.
- Blocking kernel system calls do not block the entire process.

- Processes can be split across multiple processors.
- Individual processes may be allocated variable numbers of kernel threads, depending on the number of CPUs present and other factors.



Many to one relationship:

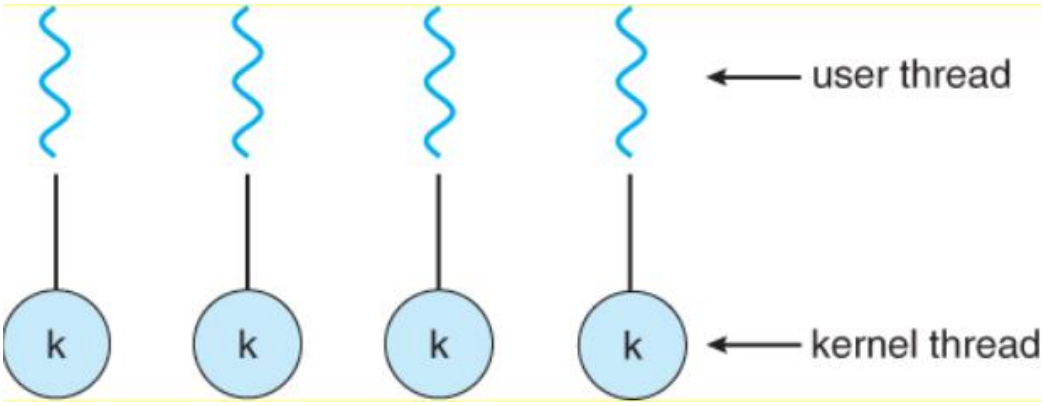
- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is very efficient.
- Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.
- The entire process will be blocked if a thread makes a blocking system call.
- Because a single kernel thread can operate only on a single CPU, the many-to-one model does not allow individual processes to be split across multiple CPUs.



One to One relationship

- One to one model creates a separate kernel thread to handle each user thread.
- This model provides more concurrency than the many to one model by allowing another thread to run when a thread makes blocking system call.
- It support multiple thread to execute in parallel on microprocessors.
- One to one model overcomes the problems listed above involving blocking system calls and the splitting of processes across multiple CPUs.

- Most implementations of this model place a limit on how many threads can be created.



Scheduling:

- Process Scheduling is a task in the operating system that schedules processes in various states such as ready, waiting, and running. Process scheduling allows the operating system to assign each process a time interval for CPU execution.
- The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs.
- For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.
- It 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.

Scheduler:

- 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.

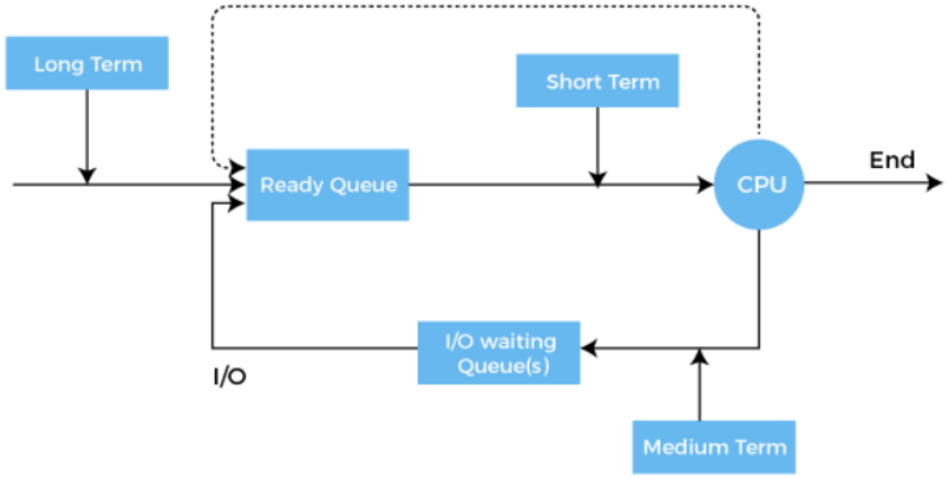
Types of Scheduler

1. **Long term scheduler:**
   - Also called Job scheduler.
   - Those schedulers whose decision will have a long-term effect on the performance.
   - It decide which program must get into the job queue. [Takes process from new state to ready state]
   - Processes of long term scheduler are placed in the ready state because in this state the process is ready to execute waiting for calls of execution from CPU which takes time that's why this is known as long term scheduler.

2. **Medium term Scheduler:**
   - Also called swap scheduler.
   - The task of moving from main memory to secondary memory is called swapping out. The task of moving back a swapped out process from secondary memory to main memory is known as swapping in.
   - The swapping of processes is performed to ensure the best utilization of main memory.

3. **Short term Scheduler:**

- Also called CPU scheduler.
- It decides the priority in which processes is in the ready queue are allocated CPU time for their execution.



Scheduling Criteria

Different CPU scheduling algorithm have different properties .The criteria used for comparing these algorithms include the following:

- **CPU Utilization:** Keep the CPU as busy as possible. It range from 0 to 100%. In practice, it range from 40 to 90%.
- **Throughput**: Throughput is the rate at which processes are completed per unit of time.
- **Turnaround time**: This is the how long a process takes to execute a process. It is calculated as the time gap between the submission of a process and its completion.
- **Waiting time**: Waiting time is the sum of the time periods spent in waiting in the ready queue.
- **Response time**: Response time is the time it takes to start responding from submission time. It is calculated as the amount of time it takes from when a request was submitted until the first response is produced.
- **Fairness**: Each process should have a fair share of CPU.

Dispatcher:

- It is a special program that comes into play after the scheduler.
- Dispatcher is a module that gives control of CPU to the process selected by short term scheduler.
- Dispatcher is also responsible for: Context Switching, Switch to user mode, restart execution of process

Context Switching:

- A context switching is a process that involves switching of the CPU from one process or task to another.
- 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.
- Context switching enables multiple processes to share a single CPU.

- Context switching is an essential part of a multitasking operating system features.
- The speed of context switching depends on:
  - Speed of memory
  - No. of registers that must be copied
  - Existence of some special instruction

Process Scheduling Types

1. **Preemptive Scheduling:**
   - The scheduling in which a running process can be interrupted if a high priority process enters the queue and is allocated to the CPU is called preemptive scheduling.
   - In this case, the current process switches from the running queue to ready queue and the high priority process utilizes the CPU cycle.
   - When a high-priority process comes in the ready queue, it doesn't have to wait for the running process to finish its burst time. However, the running process is interrupted in the middle of its execution and placed in the ready queue until the high-priority process uses the resources. As a result, each process gets some CPU time in the ready queue.

2. **Non- preemptive scheduling:**
   - The scheduling in which a running process cannot be interrupted by any other process is called non-preemptive scheduling.
   - Any other process which enters the queue has to wait until the current process finishes its CPU cycle.
   - When a non-preemptive process with a high CPU burst time is running, the other process would have to wait for a long time, and that increases the process average waiting time in the ready queue.

Preemptive scheduling	Non-preemptive scheduling
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.
Process can be interrupted in between.	Process cannot be interrupted till it terminates or switches to waiting state.
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.
Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

Scheduling Algorithms

The various scheduling algorithms are as follows:

Scheduling in Batch System

- First-Come, First-Served (FCFS) Scheduling
- Shortest Job First (SJF) Scheduling
- Shortest Remaining Time First (SRTF) Scheduling

Scheduling in Interactive System

- Priority Scheduling
- Round Robin (RR) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Multiple Queues
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling

Scheduling in Real Time Systems

- Real Time Scheduling

Thread Scheduling

- Self-Scheduling
- Master- Slave Scheduling

First Come First Served Scheduling:

- One of the simplest scheduling algorithms.
- As the name implies, the processes are executed in the order of their arrival in the ready queue which means process that enters the ready queue first gets CPU first.
- It is non preemptive algorithm. Therefore, once a process gets the CPU, it retains control until it blocks or terminates.
- Implementation of ready queue is managed as FIFO queue.
- When the first process enters the ready queue it immediately gets the CPU and starts executing. Meanwhile, other processes enter the system and are added to the end of queue by inserting their PCBs in queue. When the currently running process completes or blocks, the CPU is allocated to the process at the forefront of the queue and its PCB is removed from the queue.

Advantages:

It is simple and easy to understand.

Disadvantages:

If processes with higher burst time arrived before processes with smaller burst time then smaller processes have to wait for a long time for long processes to release CPU.

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

1. Consider the following set of processes that arrives at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst time
P1	24
P2	3
P3	3

Gantt chart: Suppose processes arrives in order: P1, P2 and P3

P1	**P2**	**P3**

0 24 27 30

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	24	24	0	24	0	0
P2	3	27	0	27	24	24
P3	3	30	0	30	27	27

Average Turn Around time (ATAT) = (24 + 27 + 30)/3 = 27ms
Average Waiting Time (AWT) = (0 + 24 + 27)/3 = 17ms

2. Suppose processes arrives in order P2, P3, P1

Gantt chart:

P2	**P3**	**P1**

0 3 6 30

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	24	30	0	30	6	6
P2	3	3	0	3	0	0
P3	3	6	0	6	3	3

Average Turn Around time (ATAT) = (30 + 3 + 6)/3 = 13ms
Average Waiting Time (AWT) = (6 + 0 + 3)/3 = 3ms

3. Consider the following set of processes that arrives at given time, with the length of the CPU burst given in milliseconds:
Suppose processes arrives in order: P1, P2, P3, P4

Gantt chart:

P1	**P2**	**P3**	**P4**

0 15 21 28 33

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	15	15	0	15	0	0
P2	6	21	2	19	13	13
P3	7	28	3	25	18	18
P4	5	33	5	28	23	23

Average Turn Around time (ATAT) = (15 + 19 + 25 + 28)/4 = 21.75ms
Average Waiting Time (AWT) = (0+ 13 + 18 + 23)/4 = 13.5ms

Shortest Job First (SJF) Scheduling

- Also called shortest process next (SPN) or shortest request next (SRN).
- It is a scheduling algorithm that schedules the processes according to the length of CPU burst they require.
- When the CPU is available, it is assigned to the process that has smallest next CPU burst.
- If the next bursts of two processes are the same, FCFS scheduling is used.
- Shortest job first scheduling can be either preemptive or non-preemptive.

Advantages:

- Shortest Jobs are favored.
- It gives minimum average waiting time for a given set of processes.

Disadvantages:

- It is difficult to implement as it needs to know the length of CPU burst of processes in advance.
- It does not favor processes having longer CPU burst. This is because long processes will not be allowed to get CPU as long as short processes continue to enter ready queue. This results in **starvation** of long processes.

Examples:

1. Consider the following set of processes that arrives at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst time
P1	6
P2	8
P3	7
P4	3

Gantt chart:

P4	**P1**	**P3**	**P2**

0 3 9 16 24

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	6	9	0	9	3	3
P2	8	24	0	24	16	16
P3	7	16	0	16	9	9
P4	3	3	0	3	0	0

Average Turn Around time (ATAT) = (9 + 24 + 16 + 3)/4 = 13ms
Average Waiting Time (AWT) = (3+ 16 + 9 + 0)/4 = 7ms

2. Consider the following set of processes that arrives at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst time	Arrival time
P1	7	0
P2	5	1
P3	2	3
P4	3	4

Gantt chart:

P1	**P3**	**P4**	**P2**

0 7 9 12 17

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	7	7	0	7	0	3
P2	5	17	1	16	11	11
P3	2	9	3	6	4	4
P4	3	12	4	8	5	5

Average Turn Around time (ATAT) = (7 + 16 + 6 + 8)/4 = 9.25ms
Average Waiting Time (AWT) = (0+ 11 + 4 + 5)/4 = 5ms

3. Consider the following set of processes that arrives at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst time	Arrival time
P1	7	0
P2	4	2
P3	1	4

P4	4	5

Gantt chart:

P1	P3	P2	P4

0 7 8 12 16

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	7	7	0	7	0	0
P2	4	12	2	10	6	6
P3	1	8	4	4	3	3
P4	4	16	5	11	7	7

Average Turn Around time (ATAT) = (7 + 10 + 4 + 11)/4 = 9.25ms
Average Waiting Time (AWT) = (0+ 6 + 3 + 7)/4 = 5ms

Shortest Remaining Job (SRTN) Scheduling

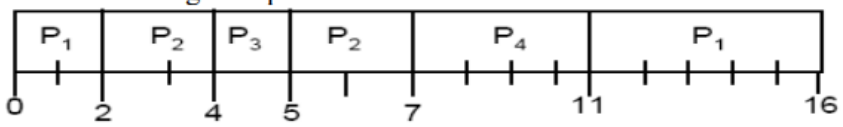
- Also called shortest time to go (STG)
- It is preemptive version of SJF scheduling algorithm where the remaining processing time is considered for assigning CPU to the next process.
- It takes into account the length of remaining CPU burst of the processes rather than the whole length in order to schedule them. The scheduler always chooses processes for execution which has the shortest remaining processing time.
- While a process is being executed, the CPU can be taken back from it and assigned to some newly arrived process provided the CPU burst of the new process is shorter than the remaining CPU burst of the current process. Note that if at any point of time, the remaining CPU burst of two processes becomes equal, they are scheduled in the FCFS order.

1. Consider the following set of processes that arrives at given time, with the length of the CPU burst given in milliseconds: c) Calculate AWT d) Calculate ATAT

Process	**Arrival Time**	**Burst time**
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Time Unit	Queue of Process
0	P_1 7
2	P_2 4 P_1 5
4	P_1 5 P_3 1 P_2 2
5	P_1 5 P_2 2 P_4 4

Gantt Chart for given question



Process	Arrival Time (T_0)	Burst Time (ΔT)	Finish Time (T_1)	TAT ($T_1 - T_0$)	WT (TAT-ΔT)	Response Time (RT)
P_1	0	7	16	16	9	0
P_2	2	4	7	5	1	0
P_3	4	1	5	1	0	0
P4	5	4	11	6	2	2

Average Turn Around time (ATAT) = (16 + 5 + 1 + 6)/4 = 7ms
Average Waiting Time (AWT) = (9 + 1 + 0 + 2)/4 = 3 ms

2. Let us consider following four processes with the length of the CPU burst given in milliseconds.
a) Calculate AWT b) Calculate ATAT

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

Time Unit	Queue of Process
0	P_1^{8}
1	P_2^{4} P_1^{7}
2	P_1^{7} P_3^{9} P_2^{3}
3	P_1^{7} P_3^{9} P_4^{5} P_2^{2}

Gantt chart is:

P1	P2	P4	P1	P3	
0	1	5	10	17	26

Process	Arrival Time (T_0)	Burst Time (ΔT)	Finish Time (T_1)	TAT ($T_1 - T_0$)	WT (TAT-ΔT)	Response Time (RT)
P_1	0	8	17	17	9	0
P_2	1	4	5	4	0	0
P_3	2	9	26	24	15	15
P_4	3	5	10	7	2	2

Average Turn Around time (ATAT) = (17 + 4 + 24 + 7)/4 = 13ms
Average Waiting Time (AWT) = (9 + 0 + 15 + 2)/4 = 6.5 ms

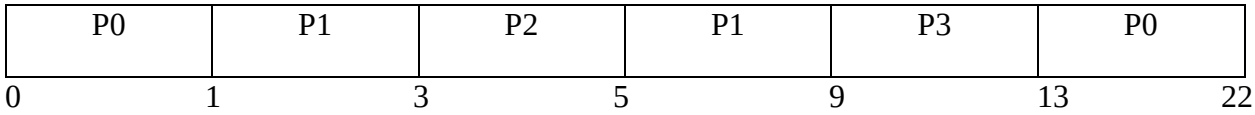
3. Let us consider following four processes with the length of the CPU burst given in milliseconds.
   a) Calculate AWT b) Calculate ATAT

Process	**Arrival Time**	**Burst time**
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Time Unit	Queue
0	P_0^{10}
1	P_1^{6} P_0^{9}
3	P_2^{2} P_0^{9} P_1^{4}
5	P_0^{9} P_1^{4} P_3^{4}

9	$P_0{}^9 P_3{}^4$
13	$P_0{}^9$

Gantt chart:



Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT -ΔT)	Response Time (RT)
P0	10	22	0	22	12	0-0=0
P1	6	9	1	8	2	1-1=0
P2	2	5	3	2	0	3-3=0
P3	4	13	5	8	4	9-5=4

Average Turn Around time (ATAT) = (22 + 8 + 2 + 8)/4 = 10ms
Average Waiting Time (AWT) = (12 + 2 + 0 + 4)/4 = 4.5 ms

4. Let us consider following four processes with the length of the CPU burst given in milliseconds.
   a) Calculate AWT b) Calculate ATAT

Process	**Arrival Time**	**Burst time**
P1	0	5
P2	1	3
P3	2	4
P4	4	1

Time Unit	Queue
0	$P_1{}^5$
1	$P_2{}^3 P_1{}^4$
2	$P_1{}^4 P_3{}^4 P_2{}^2$
4	$P_3{}^4 P_4{}^1 P_1{}^2$
5	$P_1{}^4 P_3{}^4$
9	$P_3{}^4$

Gantt chart:

0 1 2 4 5 9 13

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P0	5	9	0	9	4	0-0=0
P1	3	4	1	3	0	1-1=0
P2	4	13	2	11	7	9-2=7
P3	1	5	4	1	0	4-4=0

Average Turn Around time (ATAT) = (9 + 3 + 11 + 1)/4 = 6ms
Average Waiting Time (AWT) = (4 + 0 + 7 + 0)/4 = 2.75 ms

Priority Scheduling:

- 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.
- A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.
- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal priorities are scheduled in FCFS order.
- Priorities are generally indicated by some fixed range of numbers and there is no general method of indicating which is the highest or lowest priority, it may be either increasing or decreasing order.
- The priority ca be assigned to a process either internally defined by the system depending on the process characteristics like memory usage, I/O frequency, usage cost etc. or externally defined by the user executing that process.

Advantages:

- This provides a good mechanism where the relative importance of each process maybe precisely defined.

Disadvantages:

- If high priority processes use up a lot of CPU time, lower priority processes may starve and be postponed indefinitely.
- Difficulty in setting process priority.

Example:

1. Let us consider following five processes with arrival time 0ms and length of the CPU burst given in milliseconds. Consider 1 is highest priority.
   a) Calculate AWT b) Calculate ATAT.

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

[Non Preemptive Way]

Gantt chart:

P2	P5	P1	P3	P4

0 1 6 16 18 19

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	10	16	0	16	6	6
P2	1	1	0	1	0	0
P3	2	18	0	18	16	16
P4	1	19	0	19	18	18
P5	5	6	0	6	1	1

Average Turn Around time (ATAT) = (16 + 1 + 18 + 19 + 6)/5 = 12ms

Average Waiting Time (AWT) = (6 + 0 + 16 + 18 + 1)/5 = 8.2 ms

2. **[Preemptive Way]**

Process	Burst time	Priority	Arrival time
P1	10	3	0
P2	1	1	1
P3	2	4	2
P4	1	5	4
P5	5	2	6

Time Unit	Queue of processes
0	$P_1{}^{10}$
1	$P_2{}^{1}\ P_1{}^{9}$
2	$P_1{}^{9}\ P_3{}^{2}$
4	$P_3{}^{2}\ P_4{}^{1}\ P_1{}^{7}$
6	$P_3{}^{2}\ P_4{}^{1}\ P_5{}^{2}\ P_1{}^{5}$
8	$P_3{}^{2}\ P_4{}^{1}\ P_1{}^{5}$
13	$P_3{}^{2}\ P_4{}^{1}$
14	$P_3{}^{2}$

Gantt chart:

P1	P2	P1	P1	P5	P1	P4	P3

0 1 2 4 6 8 13 14 16

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	10	13	0	13	3	0-0=0
P2	1	2	1	1	0	1-1=0
P3	2	16	2	14	12	14-2=12
P4	1	14	4	10	9	13-4=9
P5	5	8	6	2	0	6-6=0

Average Turn Around time (ATAT) = (13 + 1 + 14 + 10 + 2)/5 = 8ms
Average Waiting Time (AWT) = (3 + 0 + 12 + 9 + 0)/5 = 4.8 ms

3. Let us consider following five processes with given arrival time and length of the CPU burst given in milliseconds. Consider 1 is highest priority.
   a) Calculate AWT b) Calculate ATAT.

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

[Non preemptive Way]
Gantt chart:

P1	P3	P4	P2

0 7 10 12 16

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	7	7	0	7	0	0
P2	4	16	1	15	11	11
P3	3	10	3	7	4	4
P4	2	12	4	8	6	6

Average Turn Around time (ATAT) = (7 + 15 + 7 + 8)/4 = 9.25ms
Average Waiting Time (AWT) = (0 + 11 + 4 + 6)/4 = 5.25 ms

4. **[Preemptive Way of Q.No 3]**

Time Unit	Queue of processes
0	P_1^7
1	$P_1^6 P_2^4$
3	$P_1^6 P_2^2 P_3^3$
4	$P_1^6 P_2^2 P_3^2 P_4^2$
6	$P_1^6 P_2^2 P_4^2$
8	$P_1^6 P_2^2$
10	P_1^6

Gantt chart:

P1	**P2**	**P3**	**P3**	**P4**	**P2**	**P1**

0 1 3 4 6 8 10 16

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	7	16	0	16	9	0-0=0
P2	4	10	1	9	5	1-1=0
P3	3	6	3	3	0	3-3=0
P4	2	8	4	4	2	6-4=0

Average Turn Around time (ATAT) = (16 + 9 + 3 + 4)/4 = 8ms
Average Waiting Time (AWT) = (9 + 5 + 0 + 2)/4 = 4ms

Round Robin (RR) Scheduling:

- Most widely used preemptive scheduling algorithm
- Especially designed for time sharing systems
- It considers all processes equally important and treats in an impartial manner.
- Each process in the ready queue gets a fixed amount of CPU time (generally from 10 to 100 milliseconds) known as time slice or **time quantum** for its execution.
- CPU is assigned to the process on the basis of FCFS for a fixed amount of time called Quantum time. After the time quantum expires, the running process is preempted and sent to the ready queue and then processor is assigned to the next arrived process. This process continuous in circular fashion.

Advantages:

- Every process gets an equal share of the CPU.
- RR is cyclic in nature, so there is no starvation.

Disadvantages:

- Setting the quantum too short, increases the overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short processes.
- Average waiting time under the RR policy is often long.

Examples:

1. Let us consider following three processes with arrival time 0ms and length of the CPU burst given in milliseconds. Consider Quantum time = 4 ms.
   a) Calculate AWT b) Calculate ATAT

Process	Burst time
P1	24
P2	3
P3	3

Gantt chart:

P1	**P2**	**P3**	**P1**	**P1**	**P1**	**P1**	**P1**

0 4 7 10 14 18 22 26 30

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	24	30	0	30	6	0-0=0
P2	3	7	0	7	4	4-0=4
P3	3	10	0	10	7	7-0=7

Average Turn Around time (ATAT) = (30 + 7 + 10)/3 = 8ms
Average Waiting Time (AWT) = (6 + 4 + 7)/3 = 4ms

2. Let us consider following three processes with given arrival time and length of the CPU burst given in milliseconds. Consider Quantum time = 2 ms.
   a) Calculate AWT b) Calculate ATAT

Process	Burst time	Arrival time
P1	4	0
P2	5	1
P3	2	2
P4	1	3
P5	6	4
P6	3	6

Ready Queue:

Time Unit	Queue of processes
0	$P_1{}^4$
2	$P_2{}^5\ P_3{}^2\ P_1{}^2$
4	$P_3{}^2\ P_1{}^2\ P_4{}^1\ P_5{}^6\ P_2{}^3$
6	$P_1{}^2\ P_4{}^1\ \ P_5{}^6\ P_2{}^3\ P_6{}^3$
8	$P_4{}^1\ P_5{}^6\ P_2{}^3\ P_6{}^3$
9	$P_5{}^6\ P_2{}^3\ P_6{}^3$
11	$P_2{}^3\ P_6{}^3\ P_5{}^4$
13	$P_6{}^3\ P_5{}^4\ P_2{}^1$
15	$P_5{}^4\ P_2{}^1\ P_6{}^1$
17	$P_2{}^1\ P_6{}^1\ P_5{}^2$
18	$P_6{}^1\ P_5{}^2$
19	$P_5{}^2$

Gantt chart:

P1	P2	P3	P1	P4	P5	P2	P6	P5	P2	P6	P5

0 2 4 6 8 9 11 13 15 17 18 19 21

Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	4	8	0	8	4	0-0=0
P2	5	18	1	17	12	2-1=1
P3	2	6	2	4	2	4-2=2
P4	1	9	3	6	5	8-3= 5
P5	6	11	4	7	1	9-4= 5
P6	3	15	6	9	6	13-6= 7

Average Turn Around time (ATAT) = (8 + 17 + 4 + 6 + 7 + 9)/6 = 8.5ms
Average Waiting Time (AWT) = (4 + 12 + 2 + 5 + 1 + 6)/6 = 5ms

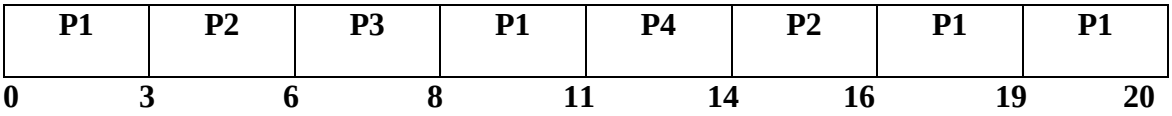
3. Let us consider following three processes with given arrival time and length of the CPU burst given in milliseconds. Consider Quantum time = 3 ms.
a) Calculate AWT b) Calculate ATAT

Process	Burst time	Arrival time
P1	10	0
P2	5	1
P3	2	3
P4	3	4

Ready Queue:

Time Unit	Queue of processes
0	$P_1{}^{10}$
3	$P_2{}^5\ P_3{}^2\ P_1{}^7$
6	$P_3{}^2\ P_1{}^7\ P_4{}^3\ P_2{}^2$
8	$P_1{}^7\ P_4{}^3\ P_2{}^2$
11	$P_4{}^3\ P_2{}^2\ P_1{}^4$
14	$P_2{}^2\ P_1{}^4$
16	$P_1{}^4$
19	$P_1{}^1$

Gantt chart:



Processes	Burst Time(ΔT)	Finish Time (T1)	Arrival Time (T0)	TAT (T1 – T0)	WT (TAT-ΔT)	Response Time (RT)
P1	10	20	0	20	10	0-0=0
P2	5	16	1	15	10	3-1=2
P3	2	8	3	5	3	6-3=3
P4	3	14	4	10	7	11-4= 7

Average Turn Around time (ATAT) = (20 + 15 + 5 + 10)/4 = 12.5ms
Average Waiting Time (AWT) = (10 + 10 + 3 + 7)/4 = 7.5ms

Highest Response Ratio Next (HRRN) Scheduling:

- Non preemptive scheduling algorithm that schedules the process according to their response ratio.
- Whenever the CPU becomes available, the process having the highest value of response ratio among all the ready processes is scheduled next.
- Jobs gain higher priority the longer they wait, which prevents indefinite postponement (process starvation).
- The response ratio of a process in the queue is computed by using the following equation:

  Response Ratio= (Time since arrived + CPU burst) / CPU burst

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

Disadvantages:

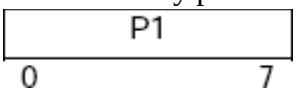
- Does not support external priority system. Processes are scheduled by using internal priority system.

Example:

1) Let us consider following Three processes with given arrival and length of the CPU burst given in milliseconds. a) Calculate AWT b) Calculate ATAT

Process	Arrival time	Burst time	Priority
P1	0	7	3
P2	2	4	1
P3	3	4	2

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



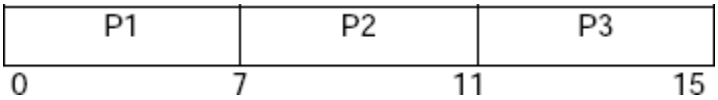
Since it is Non-preemptive, it executes process p1 completely. It takes 7 ms to complete process p1 execution.

Now, among p2 and p3 the process with highest response ratio is chosen for execution.

Response Ration for P2 = (5+4)/4 = 2.25

Response Ration for P3 = (4+4)/4 = 2

As process p2 is having highest response ratio than that of p3. Process p2 will be considered for execution and then followed by P3.



Average Waiting Time = 0+ (7-2) + (11-3) / 3 = 4.33

Average Turnaround Time = 7 + (11-2) + (15-3) /3 = 9.33

Process	Arrival time	Burst time	Complete time	Turn Around time	Waiting time	Response time
P1	0	7	7	7	0	0
P2	2	4	11	9	5	5
P3	3	4	15	12	8	8

ATAT = (7 + 9 + 12) / 3 = 9.33ms

AWT = (0 + 5 + 8) / 3 = 4.33ms

2) Let us consider following five processes with given arrival and length of CPU burst given in milliseconds. (a) Calculate AWT (b) Calculate ATAT

Process	Arrival time	Burst time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2

At time 0 only process p1 is available, so p1 is considered for execution
At time 3 only process p2 is available, so p2 is considered for execution
Since at time 9 all process has arrived so we have to calculate Response Ratio for these process

P1	P2

3 9 13

Response Ratio for P3 = (5 + 4)/4 = 2.25
Response Ratio for P4 = (3 + 5)/5 = 1.6
Response Ratio for P5 = (1+2)/2 = 1.5

P1	P2	P3

3 9 13 15

Response Ratio for P4= (7+ 5)/5= 2.4
Response Ratio for P5 = (5+2)/2 = 3.5

Process	Arrival time	Burst time	Complete time	Turn Around time	Waiting time	Response time
P1	0	3	3	3	0	0
P2	2	6	9	7	1	1
P3	4	4	13	9	5	5
P4	6	5	20	14	9	9
P5	8	2	15	7	5	5

ATAT = (3 + 7 + 9 + 14 + 7) / 5 = 8ms
AWT = (0 + 1 + 5 + 9 + 5) / 5 =4ms

3) Let us consider following five processes with given arrival and length of CPU burst given in milliseconds. (a) Calculate AWT (b) Calculate ATAT

Process	**Arrival time**	**Burst time**
P1	1	3
P2	3	6
P3	5	8
P4	7	4
P5	8	5

At time=0, there is no process available in the ready queue. So from 0 to 1 CPU is idle. Thus 0 to 1 is considered as CPU idle time.

At time=1, only process P1 is available in ready queue. So, process P1 executes till its completion.

CPU idle time	P1

0 1 4

After process P1, at time=4, only process P2 arrived, so P2 gets executed.

CPU idle time	P1	P2

0 1 4 10

At time=10, process P3, P4 and P5 were in ready queue.
So, in order to schedule next process after P2, we need to calculate response ratio.

Response Ratio for P3 = (5 + 8)/8 = 1.625
Response Ratio for P4 = (3 + 4)/4 = 1.75
Response Ratio for P5 = (2+5)/5 = 1.4

CPU idle time	P1	P2	P4

0 1 4 10 14

At time t=10, P4 executes due to its large response ratio. Now in ready queue, we have two processes P3 and P5. After the execution of P4, let us calculate response ratio of P3 and P5.
Response Ratio for P3 = (9 + 8)/8 = 2.125
Response Ratio for P4 = (6 + 5)/5 = 2.2

CPU idle time	P1	P2	P4	P5	P3

0 1 4 10 14 19 27

Process	Arrival time	Burst time	Complete time	Turn Around time	Waiting time	Response time
P1	1	3	4	3	0	0
P2	3	6	10	7	1	1
P3	5	8	27	22	14	14
P4	7	4	14	7	3	3
P5	8	5	19	11	6	6

ATAT = (3 + 7 + 22 + 7 + 11) / 5 = 10ms
AWT = (0 + 1 + 14 + 3 + 6) / 5 =4.8ms