Process and scheduling

Kashiram Pokharel

Unit 3 Process Management **15 Hrs.**

Process Concepts(3 Hrs.): Definitions of Process, The Process Model, Process States, Process State Transition, The Process Control Block, Operations on Processes (Creation, Termination, Hierarchies, Implementation), Cooperating Processes, System Calls (Process Management, File management, Directory Management).

Threads (1 Hr): Definitions of Threads, Types of Thread Process (Single and Multithreaded Process), Benefits of Multithread, Multithreading Models (Many-to-One Model, One-to-One Model, Many-to Many Model).

Inter-Process Communication and Synchronization(6 Hrs.): Introduction, Race Condition, Critical Regions, Avoiding Critical Region: Mutual Exclusion And Serializability; Mutual Exclusion Conditions, Proposals for Achieving Mutual Exclusion: Disabling Interrupts, Lock Variable, Strict Alteration (Peterson's Solution), The TSL Instruction, Sleep and Wakeup, Types of Mutual Exclusion (Semaphore, Monitors, Mutexes, Message Passing, Bounded Buffer), Serializability: Locking Protocols and Time Stamp Protocols; Classical IPC Problems (Dining Philosophers Problems, The Readers and Writers Problem, The Sleeping Barber's Problem)

Process Scheduling(5 Hrs): Basic Concept, Type of Scheduling (Preemptive Scheduling, Nonpreemptive Scheduling, Batch, Interactive, Real Time Scheduling), Scheduling Criteria or Performance Analysis, Scheduling Algorithm (Round-Robin, First Come First Served, Shortest-Job- First, Shortest Process Next, Shortest Remaining Time Next, Real Time, Priority Fair Share, Guaranteed, Lottery Scheduling, HRN, Multiple Queue, Multilevel Feedback Queue); Some Numerical Examples on Scheduling.

Program:

- Program is a static entity made up of program statement. Program contains the instructions.
- A program exists at single place in space and continues to exist. A program does not perform the action by itself.
- A program is a passive entity as it resides in the secondary memory, such as the contents of a file stored on disk. One program can have several processes
- A program is an executable file which contains a certain set of instructions written to complete the specific job on your computer. For example, Google browser chrome.exe is an executable file which stores a set of instructions written in it which allow you to view web pages.

Program:

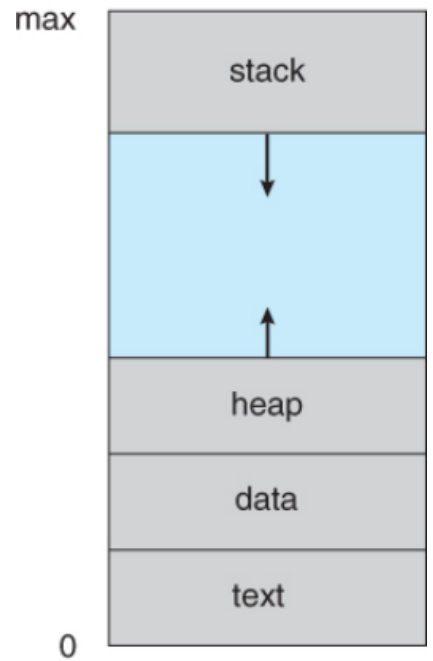
- Programs are never stored on the primary memory in your computer. Instead, they are stored on a disk or secondary memory on your PC or laptop. They are read from the primary memory and executed by the kernel.

Features of Program

- A program is a **passive entity**. It stores a group of instructions to be executed.
- Various processes may be related to the same program.
- A user may run multiple programs where the operating systems simplify its internal programmed activities like memory management.
- The program can't perform any action without a run. It needs to be executed to realize the steps mentioned in it.
- The operating system allocates main memory to store programs instructions.

Process

- A process is a program in execution. A process defines the fundamental unit of computation for the computer.
- a **process** is an instance of a computer program that is being executed.
- It contains the program code and its current activity.
- Depending on the **operating system**(**OS**), a **process** may be made up of multiple threads of execution that execute instructions concurrently.
- Process is a **dynamic entity** that is a program in execution.
- A process is a **sequence of information executions**.
- Process **exists in a limited span of time**.
- Two or more processes could be executing the same program, each using their own data and resources.
- Process is **active entity and may provides response to the user**.



Components of process are:

- Object Program:
- Data
- Resources:
- Status of the process execution.

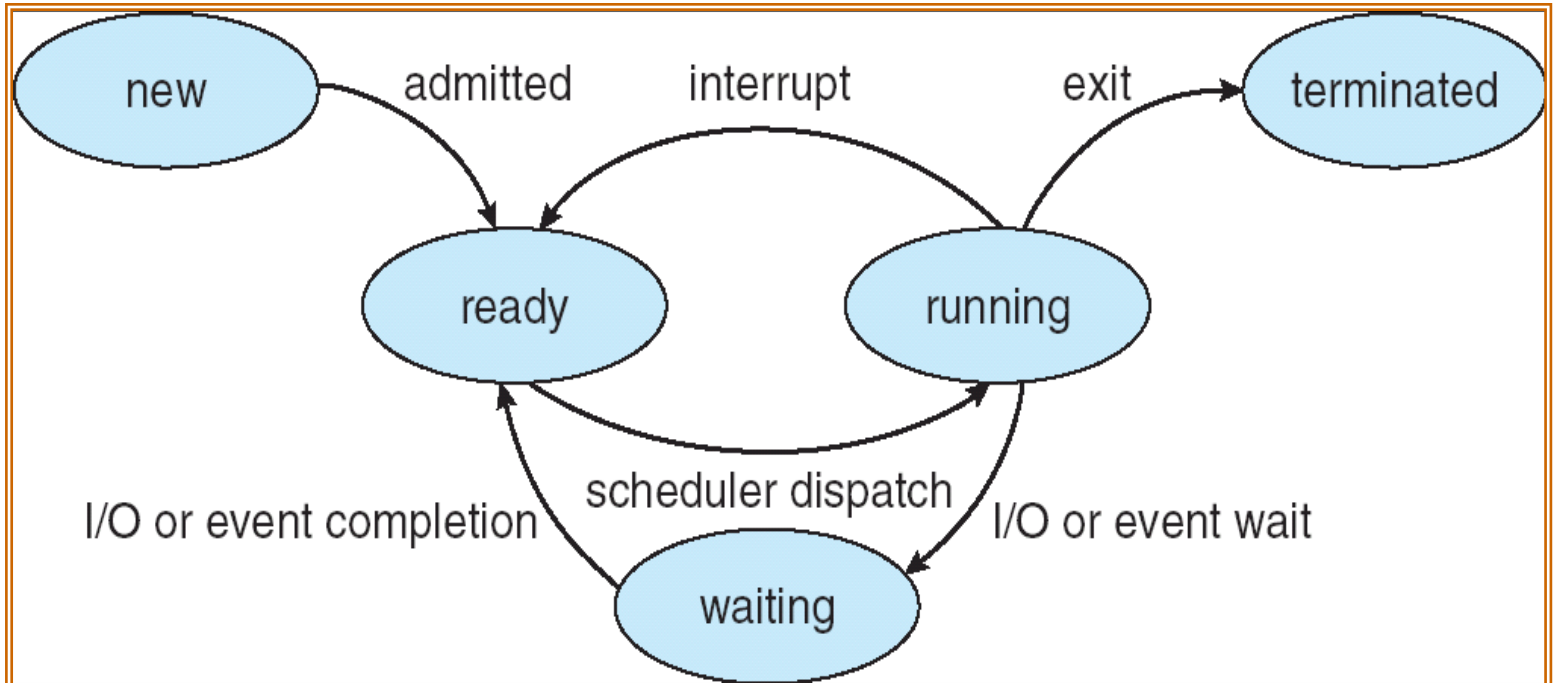
- Process memory is divided into four sections for efficient working :
  - The **text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
  - The **data section** is made up the global and static variables, allocated and initialized prior to executing the main.
  - The **heap** is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc , free, etc.
  - The **stack** is used for local variables. Space on the stack is reserved for local variables when they are declared.

Program vs Process:

Sr.No	Program	Process
1.	Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
2.	Program is a passive entity as it resides in the secondary memory in the form of file.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
4.	Program is a static entity.	Process is a dynamic entity.
5.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
6.	Program is loaded into secondary storage and does not have any control block.	Process is loaded into main memory and has its own control block called Process Control Block

Process States diagram / process transition diagram:

- When process executes, it changes state. Process state is defined as the **current activity** of the process. Fig. below shows the general form of the process state transition diagram. Process state contains five states. Each process is in one of the states. The states are listed below.



- ***New***: A process that has just been created but has not yet been admitted to the pool of executable processes by the operating system.
- ***Ready***: Ready processes are waiting to have the processor allocated to them by the operating system so that they can run.
- ***Running***: The process that is currently being executed. A running process possesses all the resources needed for its execution, including the processor.
- ***Waiting***: A process that cannot execute until some event occurs such as the completion of an I/O operation. The running process may become suspended by invoking an I/O module.
- ***Terminated(exit)***: A process that has been released from the pool of executable processes by the operating system either because it halted or because it aborted for some reason.

Some transition of process are:

Admitted (new → running)

- This transition occur when process are loaded into main memory and waiting for cpu for execution.

Dispatch(ready ->running)

- It occurs when all other process has had their fair share and it is time for first process to get CPU to run again.

Timer-run-out/interrupt (running ->ready)

- It occurs when scheduler decides that the running process has run long enough and it is time to let another process have same CPU time.

Block: (running →waiting)

- It occurs when a process discovers that it can't continue.
- When process needs some i/o signal or user response

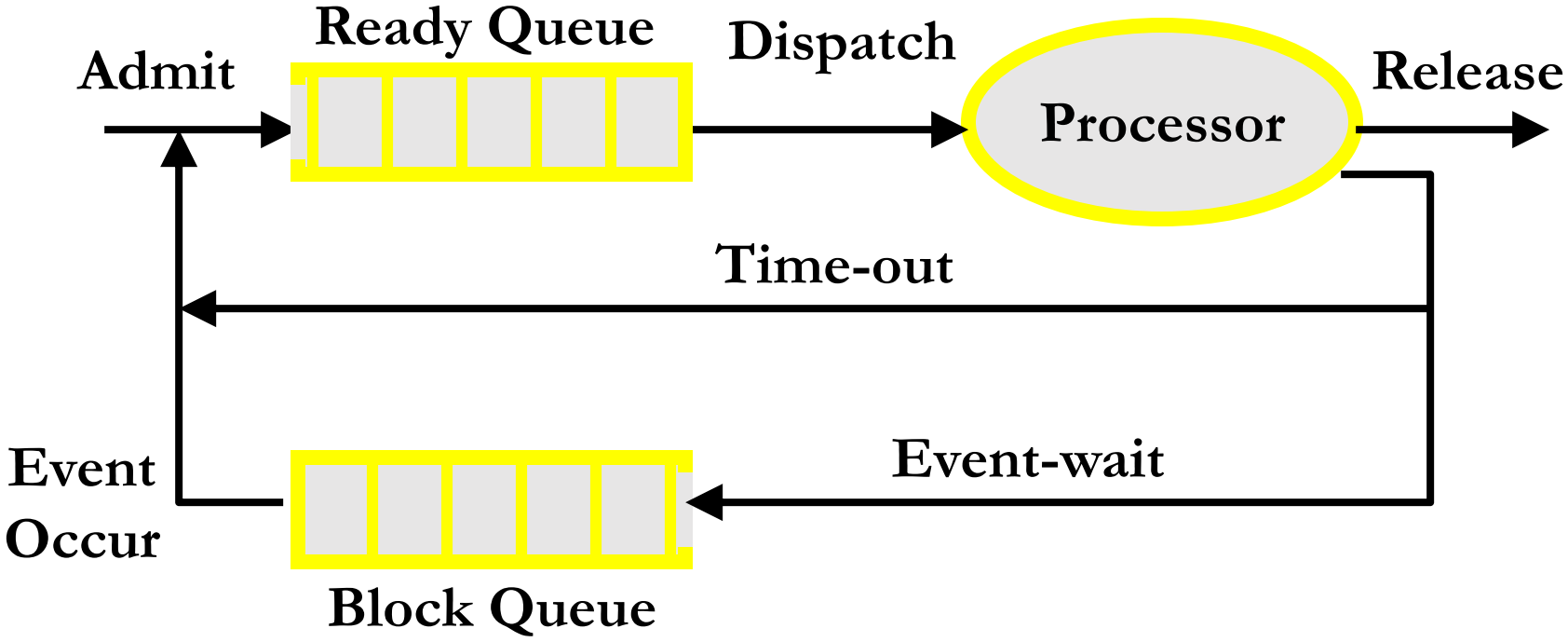
Wake-up: (waiting → ready)

- It occurs when the external event for which a process was waiting happens or i/o is complete.

Exit: (running →terminate)

- This transition occurs when process complete its execution and remove from main memory.

- Whenever processes changes state, the operating system reacts by placing the process PCB in the list that corresponds to its new state. Only one process can be running on any processor at any instant and many processes may be ready and waiting state.



Operation On process:

- Several operations are possible on the process. Process must be created and deleted dynamically. Operating system must provide the environment for the process operation. We discuss the two main operations on processes.
  - Create a process
  - Terminate a process
  - Process Hierarchy

Creating a process:

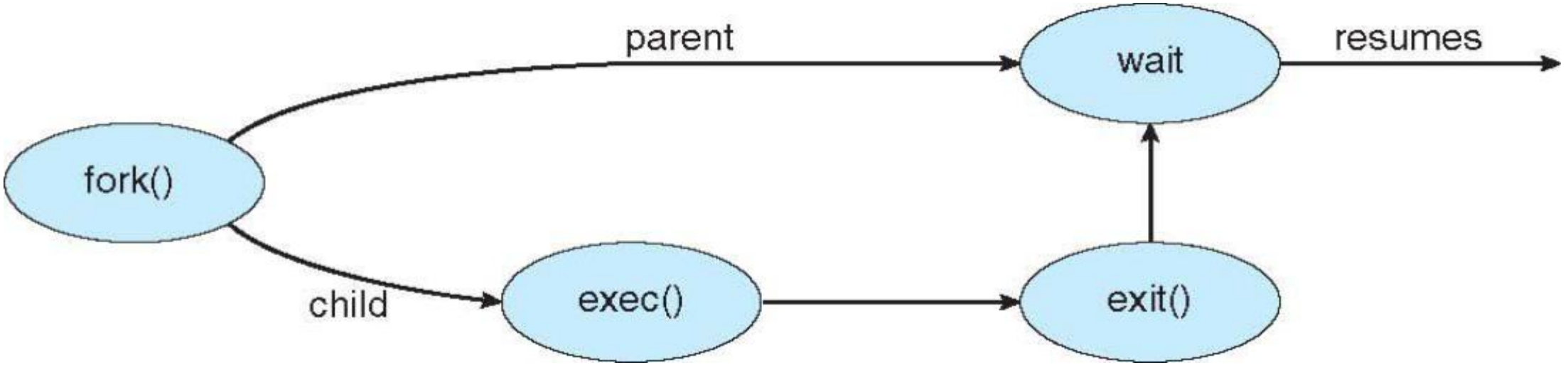
- Operating system creates a new process with the specified or default attributes and identifier. A process may create several new sub processes. Syntax for creating new process is:

  CREATE (processed, attributes)

- Two names are used in the process they are parent process and child process. Parent process is a creating process. Child process is created by the parent process. Child process may create another sub-process. So it forms a tree of processes. When operating system issues a CREATE system call, it obtains a new process control block from the pool of free memory, fills the fields with provided and default parameters, and insert the PCB into the ready list. Thus it makes the specified process eligible to run the process.
- When a process is created, it requires some parameters. These are priority, level of privilege, requirement of memory, access right, memory protection information etc. Process will need certain resources, such as CPU time, memory, files and I/O devices to complete the operation. When process creates a sub-process, that sub-process may obtain its resources directly from the operating system. Otherwise it uses the resources of parent process.

UNIX examples

- fork() system call creates new process
- exec () system call used after a fork() to replace the process' memory space with a new program.



The Creation and Termination of Processes

Reasons for Process Creation

1. **New Batch Job:** In a batch environment, a process is created in response to the submission of a job.
2. **Interactive Log On:** A user at a terminal logs onto the system
3. **Created by OS to Provide a Service:** The OS can create a process to perform a function on behalf of a user program, without the user having to wait e.g. printing.
4. **Spawned by Existing Processes:** For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes. When a process is created by the OS at the explicit request of another process, the action is referred as **process spawning**.

- When one process spawns another process, the spawning process is called the parent process and the spawned process is called the child process.
- Typically, the "related" processes need to "communicate" and "cooperate" with each other. Achieving this cooperation is a difficult task for the programmers of the OS.

Termination of Process

- **While executing a process , Process executes last statement and then asks the operating system to delete it using the exit () system call.**
  - **Returns status data from child to parent (via wait())**
  - **Process' resources are deallocated by operating system**
- Parent may terminate the execution of children processes using the abort () system call
- A batch job should indicate a "Halt" instruction, which generates an interrupt to alert the operating system that a process has completed.
- For an interactive application, the action of the user will indicate when the process is completed

Reasons for Process Termination

1. **Normal Termination:** The process executes an OS service call to indicate that it has completed running.
2. **Time Limit Exceeded:** The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include (i) total time elapsed, (ii) amount of time spent executing, and (iii) in case of an interactive process, the amount of time since the user last provided any input.
3. **Memory Unavailable:** The process requires more memory than the system can provide.
4. **Bounds Violation:** The process tries to access memory locations that it is not allowed to access.

5. **Protection Error:** The process attempts to use a resource or a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
6. **Arithmetic Error:** The process tries a prohibited computation, such as division by zero!
7. **Time Overrun:** The process has waited longer than a specified maximum for a certain event to occur.
8. **An I/O Failure:** An error occurs in input or output, such as inability to find a file, failure to read or write after specified number of tries (when, for example, a defective area is encountered on a diskette), or invalid operation (such as reading from the line printer).

Process Hierarchy:

- In a computer system, we require to run many processes at a time and some processes need to create other processes while their execution.
- A process may create several new processes during its time of execution. The creating process is called "Parent Process", while new processes are called "Child Processes". This parent-child like structure of processes form a hierarchy, called Process Hierarchy.
- The process hierarchy model creates a shared and systematic understanding of high-level, key and critical processes. It enables your organisation to see and better manage processes. It can help ensure resources are deployed more effectively to deliver value in current services and meet future corporate aspirations.

Suspended Processes

- Suspended process is not immediately available for execution.
- The process may or may not be waiting on an event.
- For preventing the execution, process is suspend by OS, parent process, process itself and an agent.
- Process may not be removed from the suspended state until the agent orders the removal.
- Swapping is used to move all of a process from main memory to disk. When all the process by putting it in the suspended state and transferring it to disk.
- **First steps to creating a process hierarchy**
  - Step 1: Start with the standards and systems you already have.
  - Step 2: Workshop critical processes with senior management.
  - Step 3: Focus on critical processes, to begin with.
  - Step 4: Assign responsibility for each process.
  - Step 5: Decide on your format and tools.

Reasons for process suspension

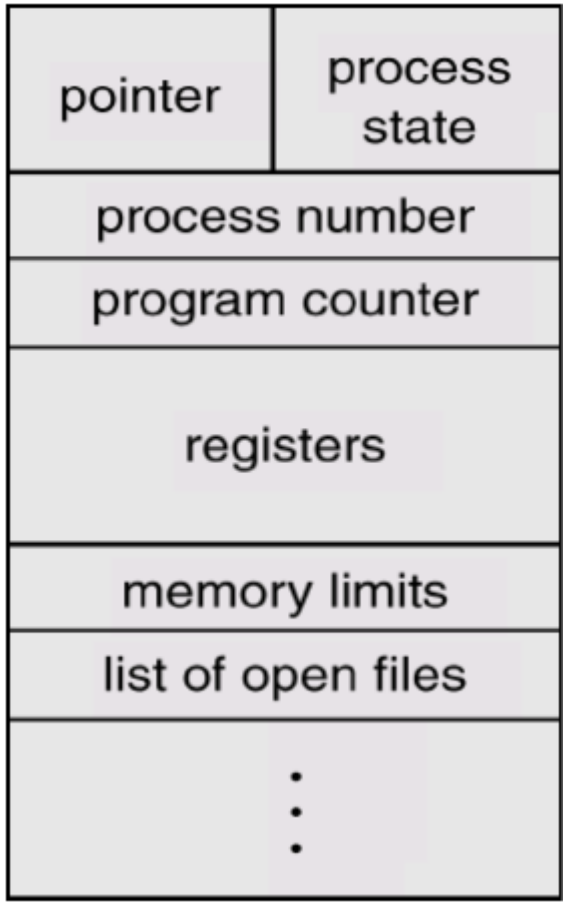
- ***Swapping:*** OS needs to release required main memory to bring in a process that is ready to execute.
- ***Timing:*** Process may be suspended while waiting for the next time interval.
- **Interactive user request**: Process may be suspended for debugging purpose by user.
- ***Parent process request:*** To modify the suspended process or to coordinate the activity of various descendants.

Process control block(PCB)

- Every process is represented in the operating system by a process control block, which is also called a **task control block.**
- PCB stands for Process Control Block. It is a **data structure** that is maintained by the Operating System for every process. The PCB should **be identified by an integer Process ID (PID).** It helps you to **store all the information required to keep track of all the running processes.**
- It is also accountable for **storing the contents of processor registers**. These are saved when the process moves from the running state and then returns back to it. The information is quickly updated in the PCB by the OS as soon as the process makes the state transition.

PCB:

- **Pointer –** It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
- **Process state –** It stores the respective state of the process.
- **Process number –** Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
- **Program counter –** It stores the counter which contains the address of the next instruction that is to be executed for the process.
- **Register –** These are the CPU registers which includes: accumulator, base, registers and general purpose registers.
- **Memory limits –** This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.
- **Open files list –** This information includes the list of files opened for a process.
- **Memory Management Information**: This information may include the value of base and limit register. This information is useful for deallocating the memory when the process terminates.
- **Accounting Information**: This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers etc.



Context switch

- Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier.
- 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

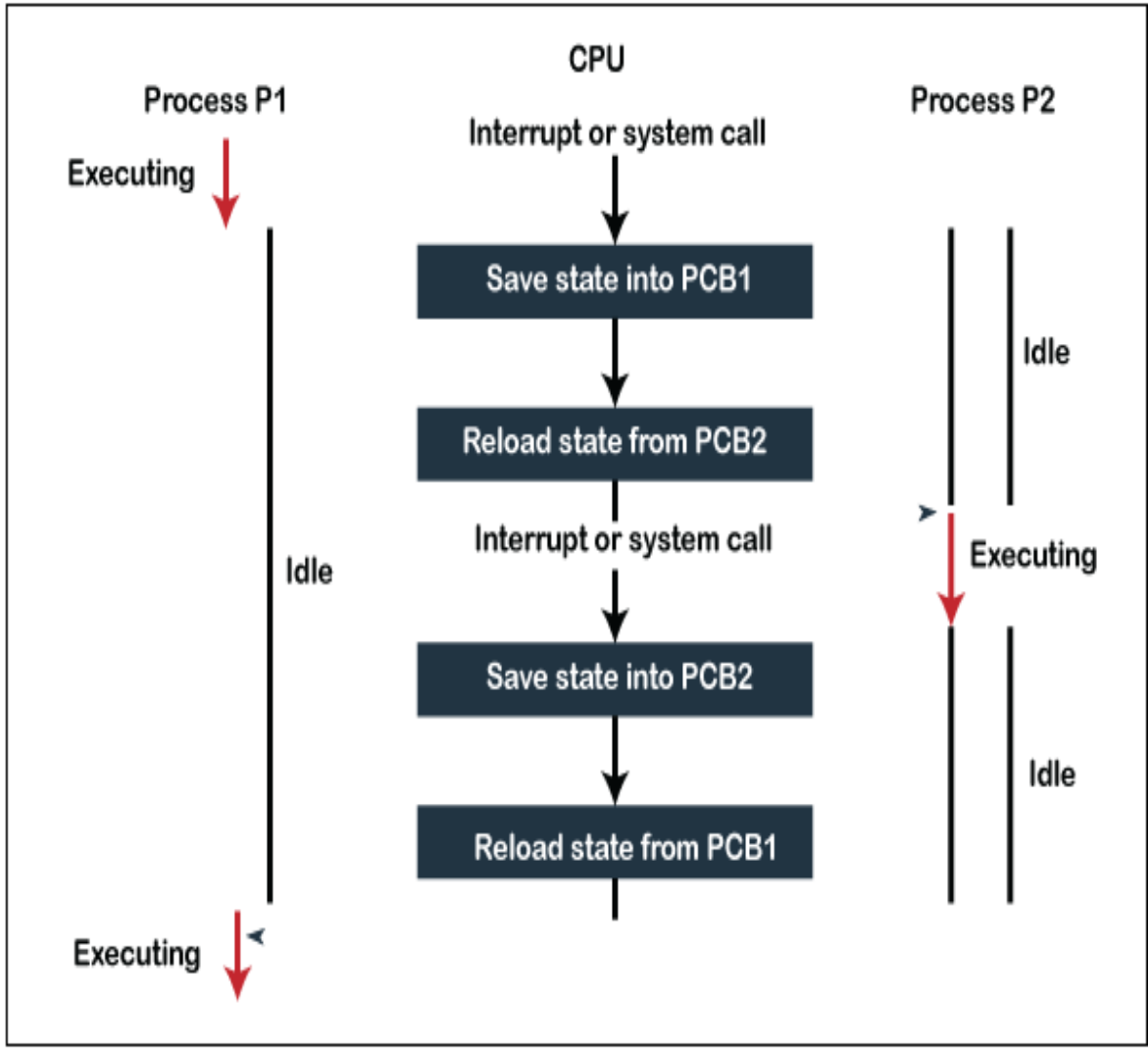
Following are the three types of context switching triggers as follows.

  - Interrupts
  - Multitasking
  - Kernel/User switch

- **Interrupts**: A CPU requests for the data to read from a disk, and if there are any interrupts, the context switching automatic switches a part of the hardware that requires less time to handle the interrupts.
- **Multitasking**: A context switching is the characteristic of multitasking that allows the process to be switched from the CPU so that another process can be run. When switching the process, the old state is saved to resume the process's execution at the same point in the system.
- **Kernel/User Switch**: It is used in the operating systems when switching between the user mode, and the kernel/user mode is performed.

The steps involved in context switching

- **Save the context** of the process that is currently running on the CPU. **Update the process control block** and other important fields.
- **Move the process control block** of the above process into the relevant queue such as the ready queue, I/O queue etc.
- **Select a new process** for execution.
- **Update the process control block of the selected process**. This includes updating the process state to running.
- Update the memory management data structures as required.
- Restore the context of the process that was previously running when it is loaded again on the processor. This is done by loading the previous values of the process control block and registers.



Co-operating process:

- Co-operating processes **share the information**: Such as a file, memory etc. System must provide an environment to allow concurrent access to these types of resources.
- Co-operating process is a process **that can affect or be affected by the other processes while executing**. If suppose any process is sharing data with other processes, then it is called co-operating process.
- Benefit of the co-operating processes are:
  - Sharing of information
  - Increases computation speed
  - Modularity
  - Convenience
  - elements are connected together. System is constructed in a modular fashion.
  - System function is divided into number of modules.
  - Cooperating process can affect or be affected by the execution of another process
  - Share data with another process/es

Inter Process Communication:

- A process can be of two types:
  - Independent process.
  - Co-operating process.
- An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes
- Inter-process communication (IPC) is a mechanism **that allows processes to communicate with each other and synchronize their actions**. The communication between these processes can be seen as a method of co-operation between them.

Different approaches of IPC:

1. Shared memory
2. Message passing
3. Pipes
4. Direct and indirect communication
5. Message queue
6. FIFO

SHARED MEMORY	MESSAGE PASSING
A region of memory is shared among the processes.	Messages are exchanged among the processes.
Faster than message-passsing systems.	Useful for exchanging smaller amounts of data.
Only require to establish shared-memory regions.	Require more time consuming task of kernel intervention.

1. IPC in Shared-Memory Systems

- A process creates the shared-memory region in it's own address space. Other processes communicate by attaching the address space to their own address space.
- This type of memory requires to protected from each other by synchronizing access across all the processes.
- Processes communicate by Reading and Writing data in the shared area. Operating system does not have any control over data or location. It is solely determined by the processes.

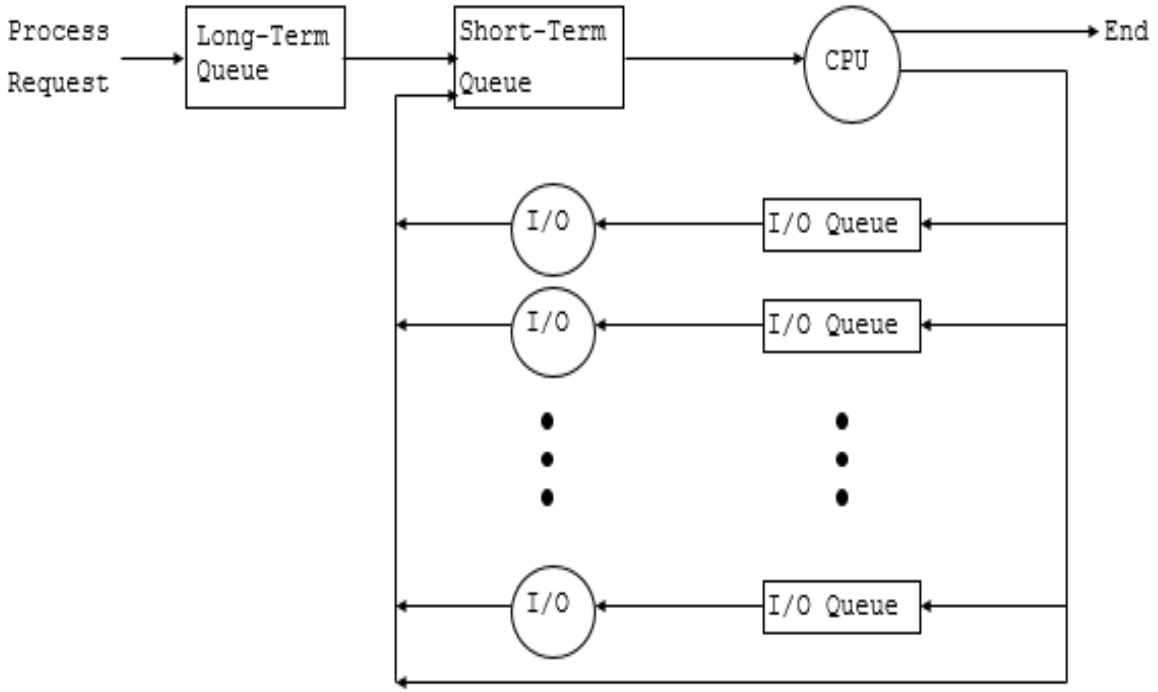
Message passing

- Message passing provides a mechanism to **allow processes to communicate and to synchronize their actions without sharing the same address space.**
- It is very useful in case where the tasks or **processes reside on different computers and are connected by a network**.
- PC facility provides two operations:
  - send(message)
  - receive(message)
- The message size is either fixed or variable. If processes P and Q wish to communicate, they need to:
  - Establish a communication link between them
  - Exchange messages via send/receive
- Message passing: either **blocking or non-blocking**
  - Blocking is considered synchronous
    - o Blocking send has the sender block until the message is received
    - o Blocking receive has the receiver block until a message is available
  - Non-blocking is considered asynchronous
    - o Non-blocking send has the sender send the message and continue
    - o Non-blocking receive has the receiver receive a valid message or null

CPU scheduler:

- **Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed**.
- The selection process is carried out by the short-term scheduler (or CPU scheduler).
- The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process. Note that the ready queue is not necessarily a first-in, first-out (FIFO) 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 –
- Long Term Scheduler
- Short Term Scheduler
- Medium Term Scheduler

Three level scheduling:



Long Term Scheduler

- It is also called job scheduler.
- **Long term scheduler determines which programs are admitted to the system for processing.**
- **Job scheduler selects processes from the queue and loads them into memory for execution.**
- Process loads into the memory for CPU scheduler.
- 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.
- When process changes the state from new to ready, then there is a long term scheduler.

Short Term Scheduler

- It is also called CPU scheduler.
- Main objective is increasing 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 from among the processes that are ready to execute and allocates the CPU to one of them.
- Short term scheduler **also known as dispatcher**, execute most frequently and makes the fine grained decision of which process to execute next. Short term scheduler is faster than long term scheduler.

Medium Term Scheduler

- Medium term scheduling is **part of the swapping function.**
- 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.
- Running process may become suspended by making an I/O request. Suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other process. Suspended process is move to the secondary storage is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison between scheduler.

S.N.	Long Term	Short Term	Medium Term
1	It is job scheduler	It is CPU Scheduler	It is swapping
2	Speed is less than short term scheduler	Speed is very fast	Speed is in between both
3	It controls degree of multiprogramming	Less control over degree of multiprogramming	Reduce the degree of multiprogramming.
4	Absent or minimal in time sharing system.	Minimal in time sharing system.	Time sharing system use medium term scheduler.
5	It select processes from pool and load them into memory for execution.	It select from among the processes that are ready to execute.	Process can be reintroduced into memory and its execution can be continued.
6	Process state is (New to Ready)	Process state is (Ready to Running)	swapping
7	Select a good process, mix of I/O bound and CPU bound.	Select a new process for a CPU quite frequently.	

Process scheduling:

Process Scheduling(5 Hrs): Basic Concept, Type of Scheduling (Preemptive Scheduling, Nonpreemptive Scheduling, Batch, Interactive, Real Time Scheduling), Scheduling Criteria or Performance Analysis, Scheduling Algorithm (Round-Robin, First Come First Served, Shortest-Job- First, Shortest Process Next, Shortest Remaining Time Next, Real Time, Priority Fair Share, Guaranteed, Lottery Scheduling, HRN, Multiple Queue, Multilevel Feedback Queue); Some Numerical Examples on Scheduling.

- Multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time multiplexing.
- 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 scheduling mechanism is the part 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 particular strategy.**
  - CPU scheduling decisions may take place when a process:
    1. Switches from running to waiting state
    2. Switches from running to ready state
    3. Switches from waiting to ready
    4. Terminates
  - Scheduling under 1 and 4 is *nonpreemptive*
  - All other scheduling is *preemptive*

- **CPU scheduling deals with the problem of deciding which of the process in the ready queue is to be allocated the CPU.**
- The major decision among different scheduling techniques is that whether they support preemptive or non-preemptive scheduling schemes

Non-Preemptive Scheduling	Preemptive scheduling
Once the CPU is allocated to a process, it cannot be taken away from that process until termination	once the CPU has been allocated to a process, the process can be taken out before the completion
No preference for higher priority job	CPU switches to higher preference job from lower preference job from middle
Fair treatment to all the processes	Not fair, CPU either switches because of time constraints or due to higher priority process
Cheap to implement	Costlier to implement
FCFS algorithm	Round Robin algorithm

Performance/scheduling criteria:

CPU utilization:

- Average time during which CPU is busy executing user programs and/or system programs. We have to keep the CPU as busy as possible
- It determines how much CPU is kept busy .CPU utilization may range from 0 to 100 percent .In real system, it should range from 40 percent(for a lightly loaded system) to 90 percent(for a heavily used system).
- In short, it is the percentage of the time the processor is busy.

Throughput:

- It is the measure of work in terms of number of process completed per unit time
- One way to measure throughput is by means of processes that are completed in a unit of time.
- In short, it is the number of processes completed.

Turn around time:

- It may be defined as the interval from the time of submission of a process to the time of it's completion. In other words it is the sum of Waiting time and the service time.
- ***Turn around time=actual execution time(CPU time)+time spent waiting for resources***
- In short ,it is how long it takes to complete its task.

Fairness:

- It is the determination that how much each process gets the fair share of the CPU. That is, it determines or checks that no process suffers starvation.

Waiting time:

- Amount of time a process has been waiting in the ready queue.
- In multiprogramming, as several jobs reside in memory at a time, CPU executes only one job at a time. The rest of the jobs wait for the CPU.

Waiting time=turnaround time - actual processing time(CPU time)

- In short, it is how long a process spends waiting in the ready queue.

Response time:

- It is the interval of time from the submission of a request until the first response is produced .This actually is the amount of time it takes to start responding, but not the time it takes to output that response .It is mostly considered in time sharing and real time systems

Objective of CPU scheduling/process scheduling:

- increase the CPU utilization and increase processing speed.
- Increase throughput.
- Decrease Turn Around Time.
- Decrease Response time and Waiting Time.
- To provide fair share of resource among number of process.

CPU scheduling algorithm:

Process Scheduling(5 Hrs): Basic Concept, Type of Scheduling (Preemptive Scheduling, Nonpreemptive Scheduling, Batch, Interactive, Real Time Scheduling), Scheduling Criteria or Performance Analysis, Scheduling Algorithm (Round-Robin, First Come First Served, Shortest-Job- First, Shortest Process Next, Shortest Remaining Time Next, Real Time, Priority Fair Share, Guaranteed, Lottery Scheduling, HRN, Multiple Queue, Multilevel Feedback Queue); Some Numerical Examples on Scheduling.

Scheduling in batch system:

- **first come first serve (FCFS) algorithm**
- **Shortest job first (SJF) scheduling algorithm:**
- **Shortest remaining time first(SRTF)**

Scheduling in Interactive system:

- **Priority Based algorithm**
- **Round robin algorithm**
- **Multiple queue scheduling**
- **Shortest process Next**
- Guaranteed scheduling
- Lottery scheduling
- Fair share scheduling

And other scheduling are :

- Two level scheduling
- Policy vs mechanism
- **Real time scheduling**
- **Optimal scheduling**
- **Multilevel feedback queue scheduling**

first come first serve (FCFS) algorithm

- Jobs are executed on **first come, first serve** basis. i.e. Processes are assigned the CPU in the order they request it. It is a **non-preemptive** scheduling algorithm.
- Easy to understand and implement. Its implementation is based on **FIFO queue**.
- **Poor in performance** as average wait time is high
- The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters a ready queue, the PCB is linked into the tail of the queue. when the CPU is free, it is allocated to the process at the head of the queue. In this algorithm once the process has the CPU allocated ,it runs to it’s completion

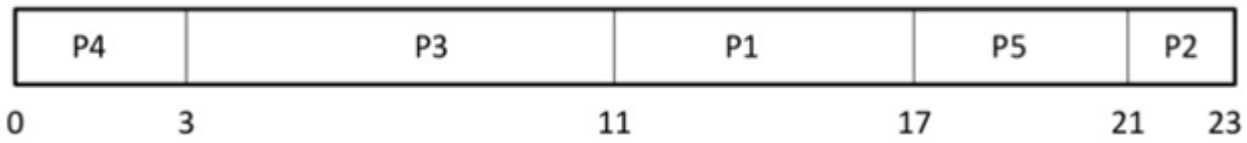
Advantages:

- Easy to understand and program. With this algorithm a single linked list keeps track of all ready processes.
- Equally fair.
- Suitable especially for Batch Operating system.

Disadvantages:

- FCFS is not suitable for time-sharing systems where it is important that each user should get the CPU for an equal amount of arrival time.
- which usually results in **poor performance.as** a consequence, there **is low rate of CPU utilization** and system throughput.
- short jobs may suffer considerable turn around delays and waiting times when the CPU has been allocated to longer jobs.

- Gantt chart becomes:



Process	Burst time	Arrival time
P1	6	2
P2	3	5
P3	8	1
P4	3	0
P5	4	4

Turn around time:

Find your self:

Waiting time = Start time - Arrival time
= TAT- execution time.

P4 = 0-0 = 0
P3 = 3-1 = 2
PI = 11-2 = 9
P5= 17-4 = 13
P2= 21-5= 16
Average Waiting Time=40/5=8.

Shortest job first (SJF) scheduling algorithm:

- Allocate the CPU to the process with least CPU burst time. With this scheduling algorithms the scheduler always chooses the process whose remaining run time is shortest.
- Two schemes:
  - **Non-preemptive** – once CPU given to the process it cannot be preempted until process completes its CPU burst.
  - **preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process is get preempt known as **Shortest-Remaining-Time-First (SRTF).**
  - **When a new job arrives its total time is compared to the current process remaining time. If the new job needs less time to finish than the current process, the current process is suspended and the new job is started. This scheme allows new short jobs to get good service.**
- Optimal for minimizing queueing time, but impossible to implement. Tries to predict the process to schedule based on previous history.
- Predicting the time the process will use on its next schedule:

SJF Scheduling Algorithm:

- scheduling of a job for a process is done on the basis of its having shortest execution time. if the two processes have the same CPU time ,then FCFS scheduling is used among them.
- Shortest job first scheduling is an optimum scheduling algorithm in terms of minimizing the average waiting time of the given set of processes.
- The shortest job first algorithm works optimally only when the exact future execution time of jobs are known at the time of scheduling.

SJF Advantages:

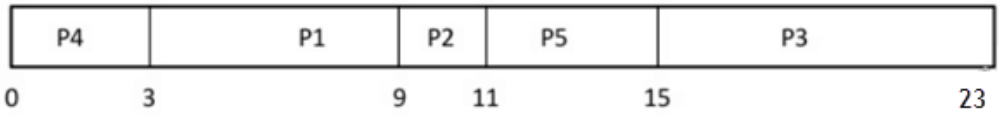
- SJF is optimal – gives minimum average waiting time for a given set of processes

SJF Disadvantages:

- Difficult to know the length of next CPU burst time
- Big jobs are waiting for CPU, which may result in aging problem.

- **Non preemptive sjf Gantt chart becomes,**

23



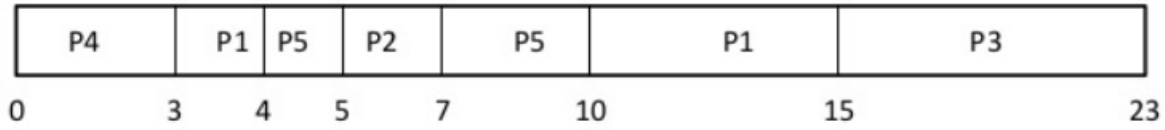
Wait time
P4= 0-0=0
P1= 3-2=1
P2= 9-5=4
P5= 11-4=7
P3= 15-1=14

Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

SRTF or

- **Preemptive SJF , Gantt chart:**

23



- .

TAT find your self

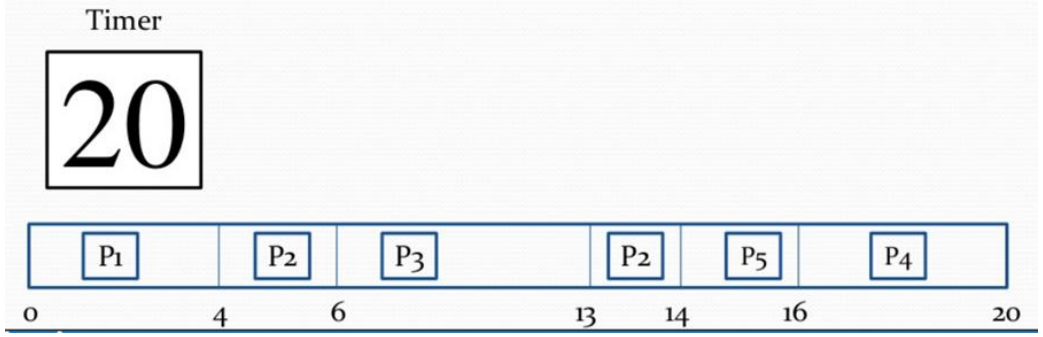
Wait time
P4= 0-0=0
P1= (3-2) + 6 =7
P2= 5-5 = 0
P5= 4-4+2 =2
P3= 15-1 = 14
Average Waiting Time = 0+7+0+2+14/5 = 23/5 =4.6

Priority Based algorithm

- A priority is associated with each process, and the CPU is allocated to process with the highest priority. Equal priority processes are scheduled in FCFS order.
- The priority determination is affected by:
  - Resource requirement
  - Processing time
  - Total spent time on the system
- Priorities can be defined either internally or externally .Internally defined priorities use some measurable quantity. for example. Time limits, memory requirements, number of open files etc.
- External priorities are set by criteria that are external to the operating system, such as the importance of the process.

- There are two Types of priority scheduling:
  - Preemptive Priority Scheduling
  - Non-preemptive Priority Scheduling
- Priority scheduling can suffer from a major problem known as ***indefinite blocking***, or ***starvation***, in which a low-priority task can wait forever because there are always some other jobs around that have higher .
- One common solution to this problem is ***aging***, in which priorities of jobs increase the longer they wait. Under this scheme a low-priority job will eventually get its priority raised high enough that it gets run.

- Gantt chart becomes.



Process	Priority	Burst time	Arrival time
P1	1	4	0
P2	2	3	0
P3	1	7	6
P4	3	4	11
P5	2	2	12

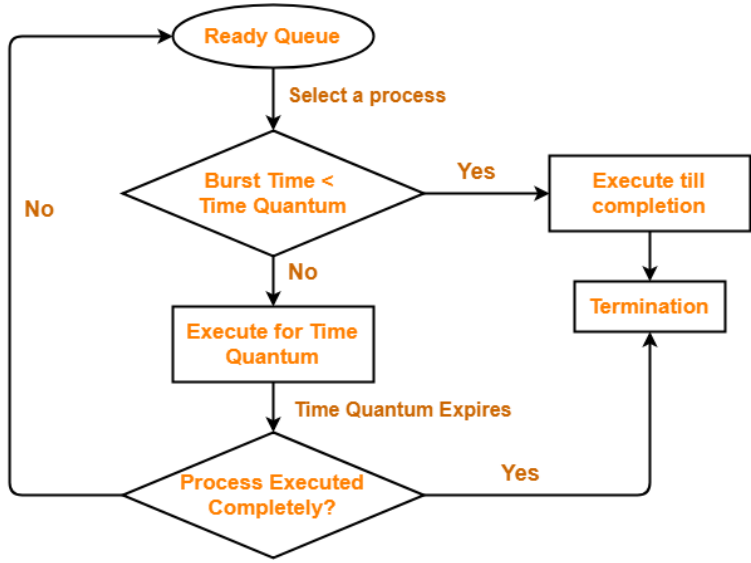
Turn around Time:
execution completion time-arrival time
P1=
P2=
P3=
P4=

Waiting Time = start time - arrival time + wait time for next burst
P1 = o - o = o
P2 =4 - o + 7 =11
P3= 6-6=0
P4= 16-11=5
Average Waiting time = (0+11+0+5+2)/5 = 18/5= 3.6

Round Robin Algorithm:

- Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called ***time quantum*** and it is preemptive in nature.
- When a process is given the CPU, a timer is set for whatever value has been set for a time quantum.
  - If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU just like the normal FCFS algorithm.
  - If the timer goes off first, then the process is swapped out of the CPU and moved to the back end of the ready queue.

.



Round Robin Scheduling

- The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and so on.
- RR scheduling can give the effect of all processors sharing the CPU equally, although the average wait time can be longer than with other scheduling algorithms
- The performance of RR is sensitive to the time quantum selected. If the quantum is large enough, then RR reduces to the FCFS algorithm; If it is very small, then each process gets 1/nth of the processor time and share the CPU equally.

Round Robin Example:

- .

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2

0 10

P1 waiting time : 4

P2 waiting time: 6

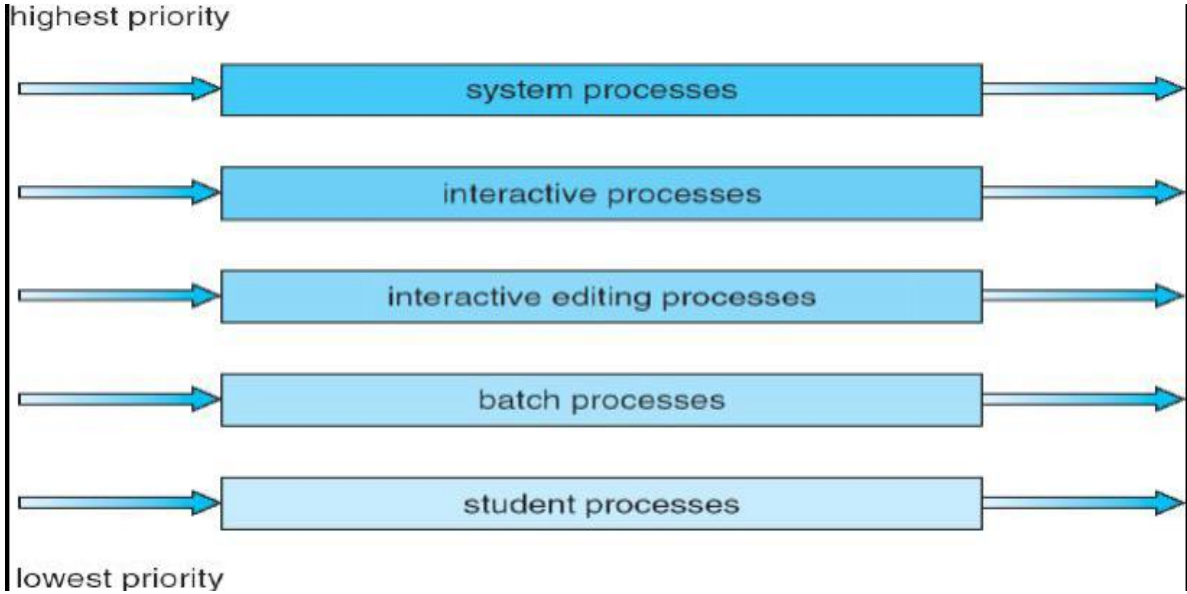
P3 waiting time: 6

The average waiting time(AWT) : (4+6+6)/3=5.33

Multi-level queue scheduling:

- A multi-level queue scheduling algorithm partitions the ready queue into several separate queues.
- The processes are divided on basis of their intrinsic characteristics such as memory size, priority etc.
- Each queue has its own scheduling algorithm.
- Since, processes do not move between queues, it has low scheduling overhead and is inflexible.
- **For example,** separate queues might be used for foreground and background processes. The foreground queue might be scheduled by the Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

Multilevel Queue Scheduling

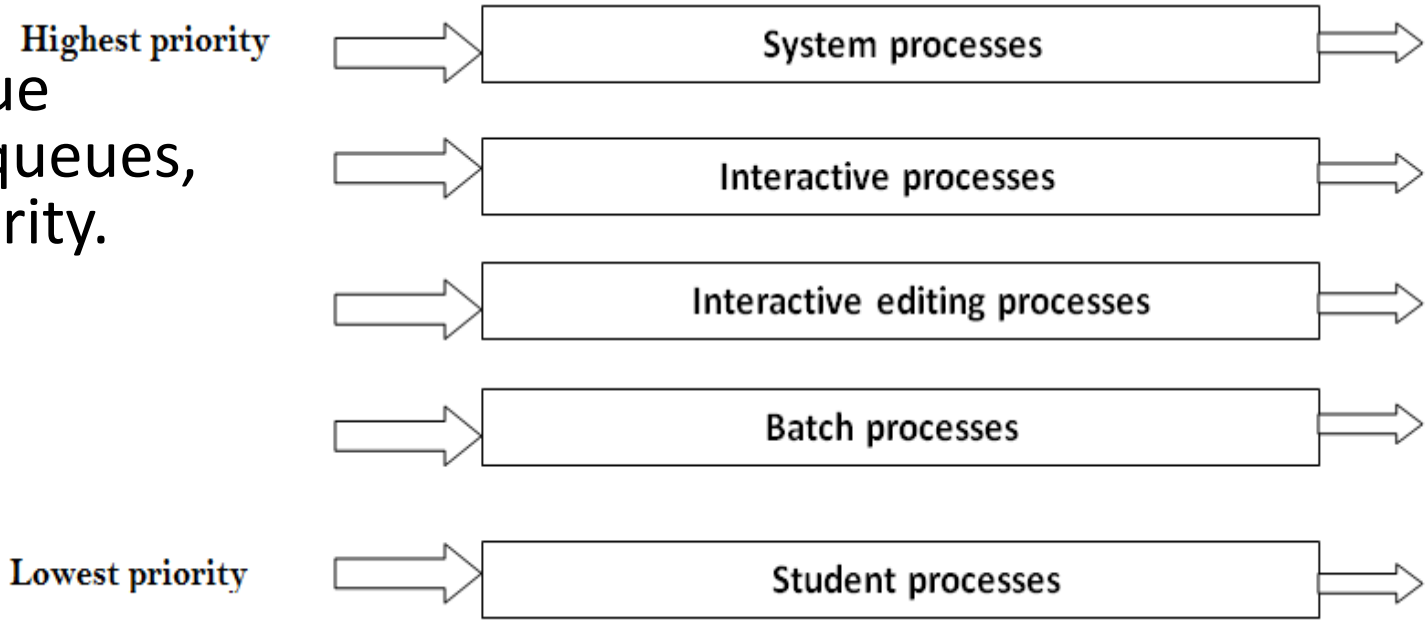


- Example : Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)

and Each queue has its own scheduling algorithm, for example

  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues.
- Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
- Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR; 20% to background in FCFS

- An example of a multilevel queue scheduling algorithm with five queues, listed below in the order of priority.
  - System processes
  - Interactive processes
  - Interactive editing processes
  - Batch processes
  - Student processes



- Each queue has absolute priority over lower priority queues. No processes in the batch queue, for example could run unless the queue for System processes, interactive processes and interactive editing processes were all empty.
- If an interactive editing process enters the ready queue while a batch process was running the batch process would be preempted. Another possibility is to time slice between the queues.
- For instance foreground queue can be given 80% of the CPU time for RR scheduling among its processes, whereas the background receives 20% of the CPU time.

Summary:

- Generally, we see in a multilevel queue scheduling algorithm processes are permanently stored in one queue in the system and do not move between the queue.
- There is some separate queue for foreground or background processes but the processes do not move from one queue to another queue and these processes do not change their foreground or background nature, these type of arrangement has the advantage of low scheduling but it is inflexible in nature.

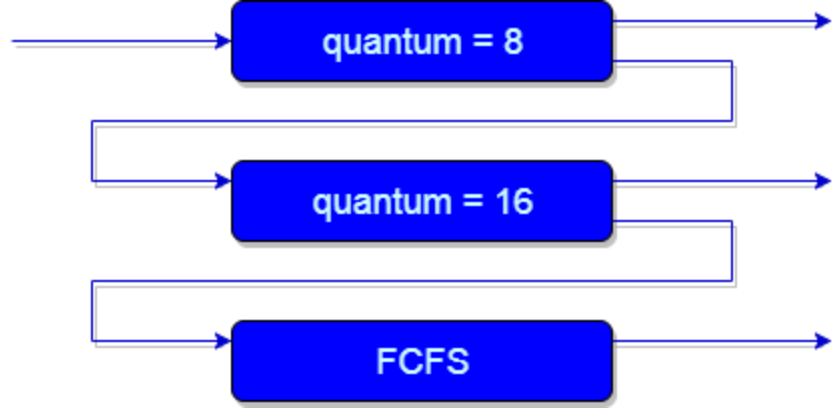
Advantages: Different type of process has different scheduling algorithm, as per requirement.

- **Disadvantages:** Lowest priority process gets starvation for the higher priority process because here priority is static.

Multilevel Feedback Queue:

- To solve the problem of multilevel queue scheduling, it allows A process to move between the various queues
  - aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service.
- Here, queues are classified as higher priority queue and lower priority queues. If process takes longer time in execution it is moved to lower priority queue.
- 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.

Example:



- First of all, Suppose that **queues 1 and 2 follow round robin** with time quantum 8 and 16 respectively and queue **3 follows FCFS**. One of the implementations of Multilevel Feedback Queue Scheduling is as follows:
  1. If any process starts executing then firstly it enters queue 1.
  2. In queue 1, the process executes for 8 unit and if it completes in these 8 units or it gives CPU for I/O operation in these 8 units unit than the priority of this process does not change, and if for some reasons it again comes in the ready queue than it again starts its execution in the Queue 1.
  3. If a process that is in queue 1 does not complete in 8 units then its priority gets reduced and it gets shifted to queue 2.
  4. Above points 2 and 3 are also true for processes in queue 2 but the time quantum is 16 units. Generally, if any process does not complete in a given time quantum then it gets shifted to the lower priority queue.
  5. After that in the last queue, all processes are scheduled in an FCFS manner.
  6. It is important to note that a process that is in a lower priority queue can only execute only when the higher priority queues are empty.
  7. Any running process in the lower priority queue can be interrupted by a process arriving in the higher priority queue.

Advantages of MFQS

- 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.

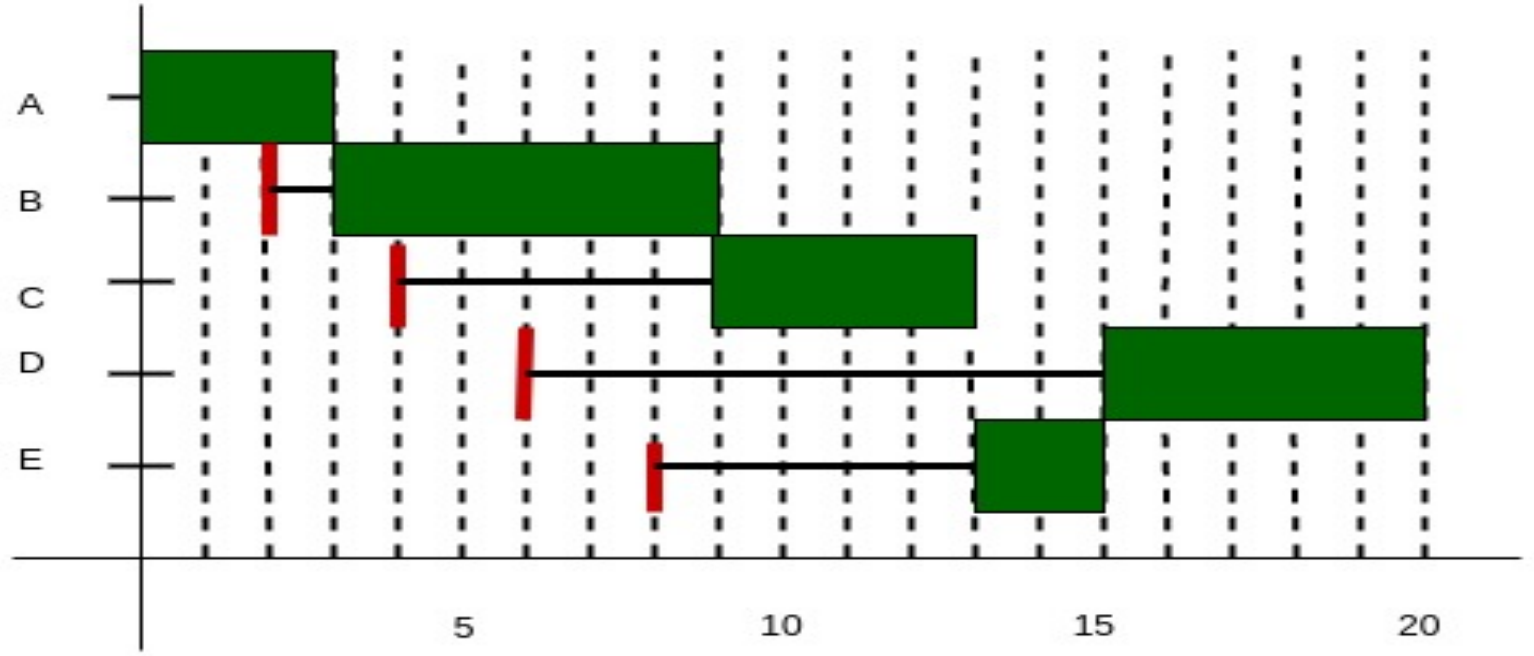
Disadvantages of MFQS

- 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

Highest Response Ratio Next (HRRN) CPU scheduling
OR
HRN

- Given n processes with their Arrival times and Burst times, To find average waiting time and average turn around time using HRRN we need to find the response ratio of all available processes and select the one with the highest Response Ratio.
- A process once selected will run till completion, i.e it is non preemptive in neature.
- **Scheduling Criteria – Response Ratio**
- **Mode – Non-Preemptive**
- Response Ratio = (W + S)/S
  - Here, **W** is the waiting time of the process so far and **S** is the Burst time of the process.
- Shorter Processes are favored for execution.
- Aging without service increases ratio, longer jobs can get past shorter jobs.

Gantt Chart -



Scheduling Example

Thread	Arrival TIme	CPU Burst Length
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Explanation –

- At t = 0 we have only one process available, so A gets scheduled.
- Similarly, at t = 3 we have only one process available, so B gets scheduled.
- Now at t = 9 we have 3 processes available, C, D and E. Since, C, D and E were available after 4, 6 and 8 units respectively. Therefore, waiting time for C, D and E are (9 – 4 =)5, (9 – 6 =)3, and (9 – 8 =)1 unit respectively.
- Using the formula given above we calculate the Response Ratios of C, D and E respectively as 2.25, 1.6 and 1.5.
- Clearly, C has the highest Response Ratio and so it gets scheduled
- Next at t = 13 we have 2 jobs available D and E.
- Response Ratios of D and E are 2.4 and 3.5 respectively.
- So process E is selected next and process D is selected last.

Implementation of HRRN Scheduling

- HRN is **non-preemptive scheduling** algorithm.
- In Shortest Job First scheduling, priority is given to shortest job, which may sometimes indefinite blocking of longer job.HRN Scheduling is used to **correct this disadvantage of SJF.**
- For determining priority, not only the job's service time but the waiting time is also considered.
- In this algorithm, dynamic priorities are used instead of fixed priorities.
- Dynamic priorities in HRN are calculated as

  Priority = (waiting time + service time) / service time.
- So shorter jobs get preference over longer processes because service time appears in the denominator.
- Longer jobs that have been waiting for long period are also give favorable treatment because waiting time is considered in numerator.

Lottery scheduling

- each process in the ready state queue is given some lottery tickets, and then the algorithm picks a winner based on a randomly generated number.
- The winning process is awarded a time slice, in which the process may be complete or the time slice may expire before the process completes.
- The winning process goes back into the ready queue, receives its tickets for the next lottery, and the cycle continues.
- Lottery scheduling can be **preemptive or non-preemptive**. It also solves the problem of starvation.
- Sometimes more important processes can be given extra tickets . To increase their odds of winning because a process having higher number of tickets, then obviously it can get more chance to win to get the cpu for execution.

- Example – **If we have two processes A and B having 60 and 40 tickets respectively out of total 100 tickets. CPU share of A is 60% and that of B is 40%.These shares are calculated probabilistically and not deterministically**.

Explanation –

- We have two processes A and B. A has 60 tickets (ticket number 1 to 60) and B have 40 tickets (ticket no. 61 to 100).
- Scheduler picks a random number from 1 to 100. If the picked no. is from 1 to 60 then A is executed otherwise B is executed.
- An example of 10 tickets picked by Scheduler may look like this –
- Ticket number - 73 82 23 45 32 87 49 39 12 09.
- Resulting Schedule - B B A A A B A A A A.
- A is executed 7 times and B is executed 3 times. As you can see that A takes 70% of CPU and B takes 30% which is not the same as what we need as we need A to have 60% of CPU and B should have 40% of CPU.This happens because shares are calculated probabilistically but in a long run(i.e when no. of tickets picked is more than 100 or 1000) we can achieve a share percentage of approx. 60 and 40 for A and B respectively.

Guaranteed Scheduling:

- Make real promises to the users about performance.
- If there are n users logged in while you are working, you will receive about 1 /n of the CPU power. Similarly, on a single-user system with n processes running, all things being equal, each one should get 1 /n of the CPU cycles.
- To make good on this promise, the system must keep track of how much CPU each process has had since its creation.
- CPU Time entitled= (Time Since Creation)/n
- Then compute the ratio of Actual CPU time consumed to the CPU time entitled.
- A ratio of 0.5 means that a process has only had half of what it should have had, and a ratio of 2.0 means that a process has had twice as much as it was entitled to.
- •The algorithm is then to run the process with the lowest ratio until its ratio has moved above its closest competitor

Real time scheduling,

- In real-time systems **processes are referred to as tasks** and these have certain temporal qualities and restrictions. **All tasks will have a deadline, an execution time and a release time.**
- **There are generally three approaches of real time scheduling**
  - **Clock driven approach**
    - applicable only when the system is by and large deterministic, except for a few aperiodic and sporadic jobs to be accommodated in the deterministic framework.
  - **Weighted round-robin**
  - **Priority driven approach**

- Real-time scheduling algorithms are grouped into two primary categories:
- **Static Scheduling**
- With static scheduling, decisions about what to run next are not made in real time. We can do this by:
- Feeding the system a pre-made list of processes and the order in which they should run. This can help us save time that would otherwise be lost due to an inefficient scheduling algorithm or real-time decision-making (if there are lots of processes to run).
- Building scheduling decisions into our code by means of control mechanisms like locks and semaphores to allow threads to take turns and to share resources like the CPU, RAM, buffers, files, and so on. When a thread attempts to claim a lock on a resource that's currently in use, it will simply be blocked until that resource is freed.

- **Dynamic Scheduling**
- Dynamic scheduling is a broad category of scheduling that employs any of the algorithms we've looked at so far. However, in the context of real-time systems, it prioritizes scheduling according to the system's deadlines:
- ⚠ **Hard real-time deadlines** are ones that we simply can't afford to miss; doing so may result in a disaster, such as a jet's flight controls failing to respond to a pilot's input.
- ☹ **Soft real-time deadlines** are ones that will produce a minor inconvenience if they aren't met. An example of this is a video whose audio isn't properly synced, causing a noticeable delay between what is shown and what's actually heard.
- Thus, while static scheduling is employed before the system ever receives its first process, dynamic scheduling is employed on the fly, making decisions about which processes to schedule as they arrive in real time. Leaving things up to the hardware like this can have its advantages. For one, it guarantees some sort of scheduling optimization among threads regardless of how the code was originally written.

System call

Kernel Mode of cpu

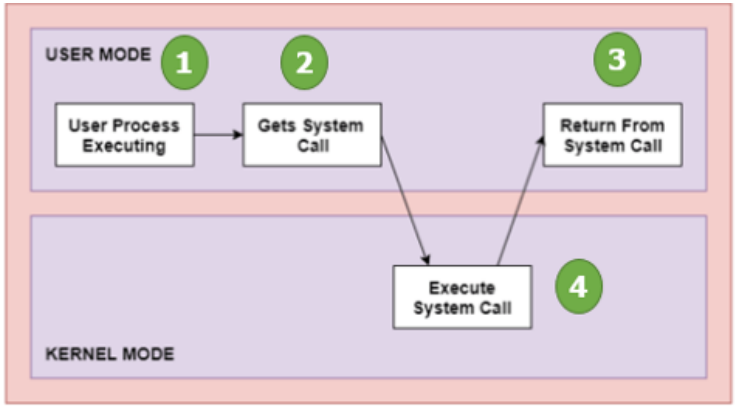
- When CPU is in **kernel mode**, the code being executed can access any memory address and any hardware resource.
- Hence kernel mode is a very privileged and powerful mode.
- If a program crashes in kernel mode, the entire system will be halted.

User Mode of cpu

- When CPU is in **user mode**, the programs don't have direct access to memory and hardware resources.
- In user mode, if any program crashes, only that particular program is halted.
- That means the system will be in a safe state even if a program in user mode crashes.Hence, most programs in an OS run in user mode.

System call

- When a program in user mode requires access to hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a **system call**.
- a **system call** is the programmatic way in which a **User program requests serveral service from the kernel of the operating system** it is executed on and the OS responds by invoking a series of system calls to satisfy the request.
- System call offers the services of the operating system to the user programs via API (Application Programming Interface)



Step 1) The processes executed in the user mode till the time a system call interrupts it.
Step 2) After that, the system call is executed in the kernel-mode on a priority basis.
Step 3) Once system call execution is over, control returns to the user mode.,
Step 4) The execution of user processes resumed in Kernel mode.

Services Provided by System Calls :

- Process creation and management
- Main memory management
- File Access, Directory and File system management
- Device handling(I/O)
- Protection
- Networking, etc.

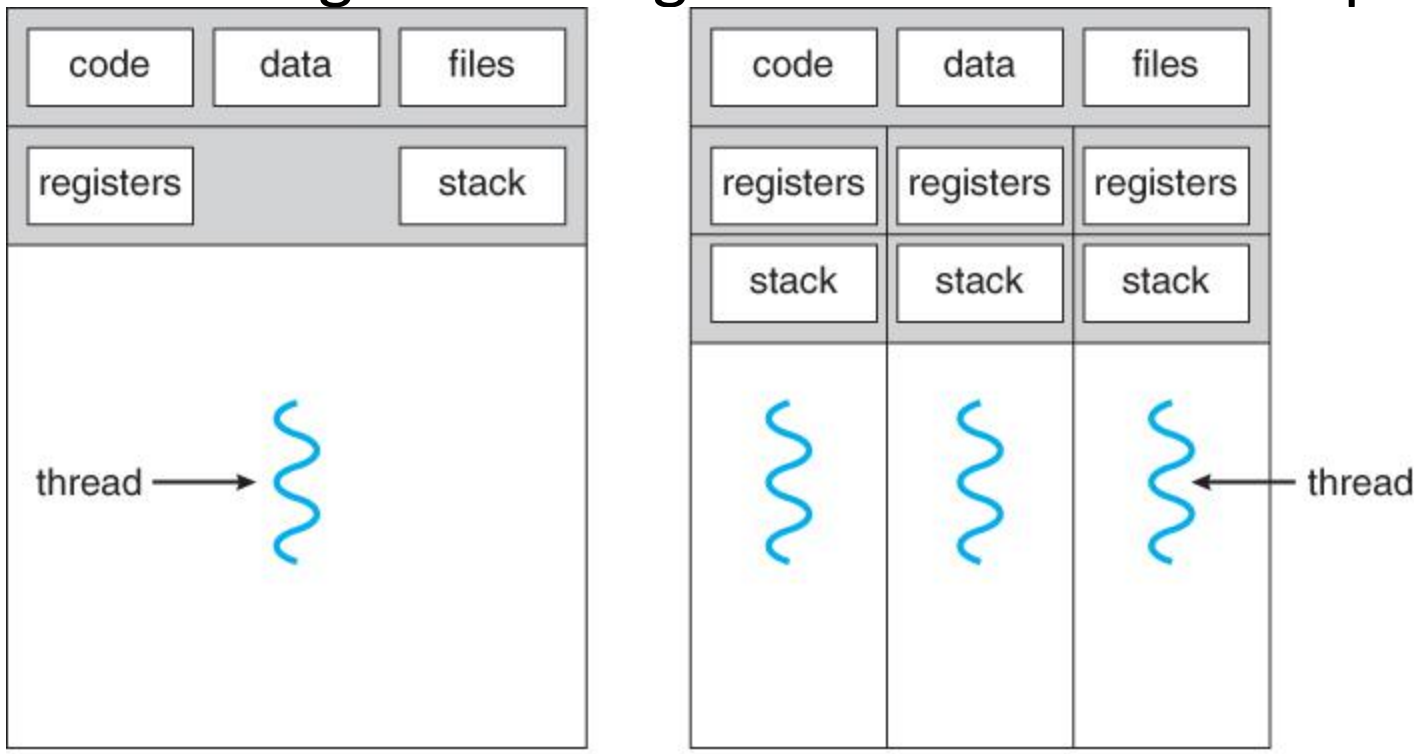
Examples of Windows and Unix System Calls –

Type of system call	**Windows**	Unix
Process Control	**CreateProcess()** **ExitProcess()** **WaitForSingleObject()**	fork() exit() wait()
File Manipulation	**CreateFile()** **ReadFile()** **WriteFile()** **CloseHandle()**	open() read() write() close()
Device Manipulation	**SetConsoleMode()** **ReadConsole()** **WriteConsole()**	ioctl() read() write()
Information Maintenance	**GetCurrentProcessID()** **SetTimer()** **Sleep()**	getpid() alarm() sleep()
Communication	**CreatePipe()** **CreateFileMapping()** **MapViewOfFile()**	pipe() shmget() mmap()
Protection	**SetFileSecurity()** **InitlializeSecurityDescriptor()** **SetSecurityDescriptorGroup()**	chmod() umask() chown()

Thread:

- **A thread is a flow of execution through the process code, with its own program counter, system registers and stack.**
- A thread is also called a **light weight 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**. Following figure shows the working of the single and multithreaded processes



single-threaded process multithreaded process

Advantage of thread:

- Thread minimizes context switching time.
- Increase responsiveness i.e. with multiple threads in a process, if one threads blocks then other can still continue executing
- Use of threads provides concurrency within a process.
- Efficient communication. Sharing of common data reduce requirement of inter-process communication
- Economy- It is more economical to create and context switch threads.
- Utilization of multiprocessor architectures to a greater scale and efficiency.
- Threads are cheap to create and use very little resources

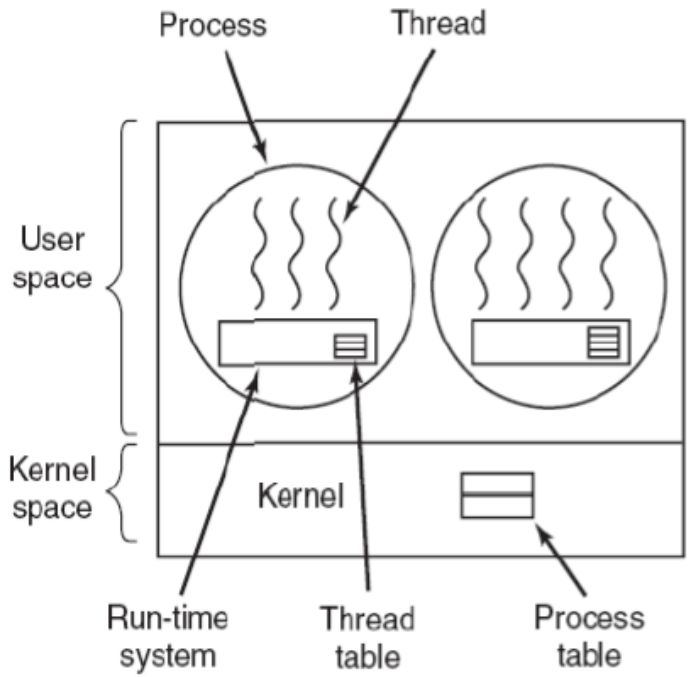
Difference between thread and process.

Process	Thread
Process is **heavy weight or resource intensive**.	Thread is **light weight taking lesser resources than a process.**
Process switching needs interaction with operating system	**Thread switching** does not need to interact with operating system
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
If one process is blocked then no other process can execute until the first process is unblocked	While one thread is blocked and waiting, second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.
Doesn't share memory (loosely coupled)	Shares memories and files (tightly coupled).

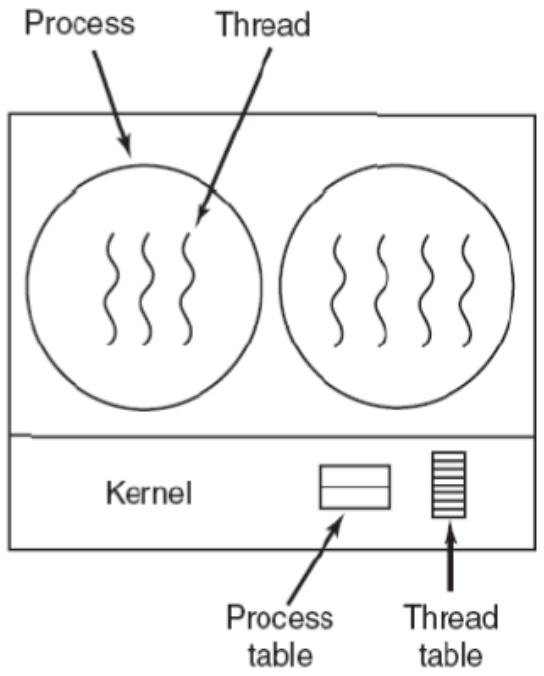
Types of thread:

Threads are implemented in following two ways

- User Level Threads
- Kernel Level Threads



Kernel level thread



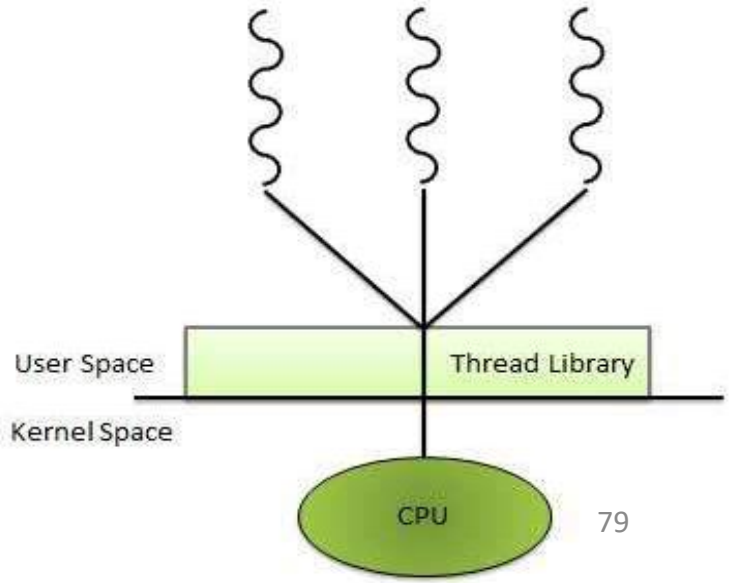
User level thread

User Level Threads

- User managed threads/implemented by user and kernel is not aware of existence of these threads
- Small and much faster than kernel level threads
- Can run in any operating system
- Threads switching doesn't require kernel mode privilege.

- **application manages thread management**
- **kernel is not aware of the existence of threads**.
- The thread library contains code for creating and des passing message and data between threads, for sche execution and for saving and restoring thread

contexts. The application begins with a single thread and begins running in that thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing

Kernel Level Threads

- Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel.
- Slower to create and manage.
- Specific to operating system
- The context information for the process as well as the process threads is all managed by the kernel. Because of this, kernel-level threads are slower than user-level threads.

- In this case, **thread management done by the Kernel**.
- **There is no thread management code in the application area**.
- Kernel threads are **supported directly by the operating system**. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process
- The **Kernel maintains context information for the process as a whole and for individuals' threads within the process**. 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 from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can multithreaded.

Disadvantages

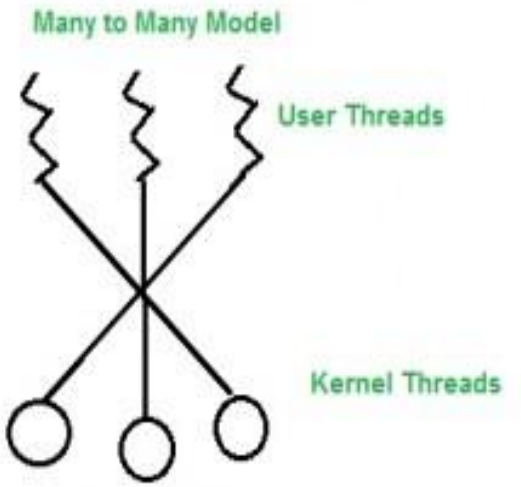
- Kernel threads are generally **slower to create and manage** than the user threads.
- Transfer of control from one thread to another within same process requires a mode switch to the Kernel

User Level Thread	Kernel Level Thread
User-level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
Implemented by a thread_library user level.	Operating system support directly to Kernel threads.
User level thread can run on any operating system.	Kernel level threads are specific to .the operating system.
Support provided at the user level called user-level thread	Support may be provided by Kernel is called Kernel level threads.
Multithread applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
User level threads are also called many to one mapping thread.	Kernel level thread support one to one thread mapping.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
Example: User-thread libraries include POSIX Pthreads, Mach C-threads, and Solaris 2 UI-threads.	Example: Windows NT, Windows 2000, Solaris 2, BeOS, and Tru64 UNIX (formerly Digital UNIX)-support kernel threads.

Multithreading Models:

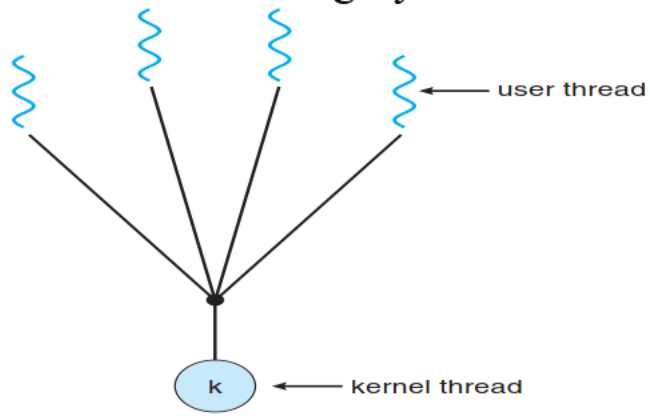
- Multi threading-It is a process of multiple threads executes at same time.
- Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach.
- In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.
- Multithreading models are three types:
  - Many to many relationship.
  - Many to one relationship.
  - One to one relationship.

Many to Many Model



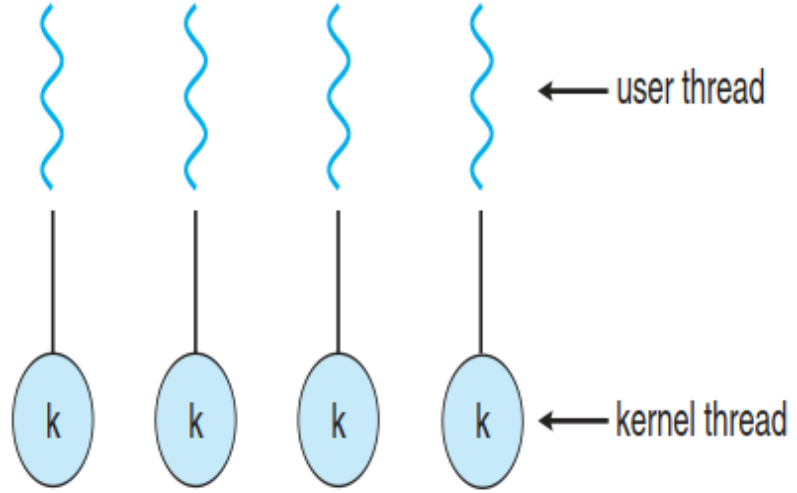
- In this model, many user level threads multiplex to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.
- Fig. shows the many to many models. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.

Many to One model:



- Many to one model maps many user level threads to one Kernel level thread.
- Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.
- If the user level thread libraries are implemented in the operating system, that system does not support Kernel threads use the many to one relationship modes

One to One model:



- The one-to-one model (one user thread to one kernel thread) is among the earliest implementations of true multithreading.
- In this implementation, each user-level thread created by the application is known to the kernel, and all threads can access the kernel at the same time. This model provides more concurrency than the many to one model.
- Problem with this model is that creating a user thread requires the corresponding kernel thread.
- It also another thread to run when a thread makes a blocking system call. It support multiple thread to execute in parallel on microprocessors.
- Disadvantage of this model is that creating user thread requires the corresponding Kernel thread.