Process & Threads In Operating System

Marwadi University

Department of Computer Engineering

Unit - 2
Processes & Threads

Operating System - 01CE0401

Topics Covered

- Program
- Process
- Address Space
- Process Model
- Process States / Process Life Cycle
- Process Control Block (PCB)
- Process Table
- Process Creation
- Process Termination
- Process Scheduling
- Process Queues
- Process Schedulers
- Context Switching / Process Switching

Program

- Program contains a set of instructions designed to complete a specific task.
- Program exists at a single place and continues to exist until it is deleted.
- A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.
- A part of a computer program that performs a well-defined task is known as an algorithm. A collection of computer programs, libraries and related data are referred to as a software.

Program

- A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language –

Program:

```
#include <stdio.h> int main()

{

   printf("Hello, World! \n");

   return 0;

}
```

Process Concept

- A process is a program in execution.
- A process generally also includes the process **stack**, which contains temporary data (such as function parameters, return addresses, and local variables), and a **data section**, which contains global variables.
- Or an entity that can be assigned and execute on a processor.
- It can be also defined as instance of the program running in the computer.

Process Concept

- Program is passive entity stored on disk (executable file), process is active
- Program becomes process when executable file loaded into memory.
- Processes that are running in the background mode (e.g.checking email) are known as Daemons.
- Daemons are processes that run unattended. They are constantly in the background and are available at all times. Daemons are usually started when the system starts, and they run until the system stops. A daemon process typically performs system services and is available at all times to more than one task or user.

Eg: Crond, syslogd

Difference B/W zombie, orphan and daemon: https://www.tutorialspoint.com/zombie-vs-orphan-vs-daemon-processes

How is a program converted into the process?

- The program can be converted into the process by following three actions
  - Compiler
  - Linker
  - Loader

How is a program converted into the process?

Compiler
The compiler converts the high-level language code into the low-level language code or object code.
The compiler creates the .o file.

Linker
Linker converts the object code or low-level language code into the executable code.
The linker creates the .exe file.

Loader
The loader loads the executable code into the main memory and allocates the address space for the process.
A program contains code and data sections.

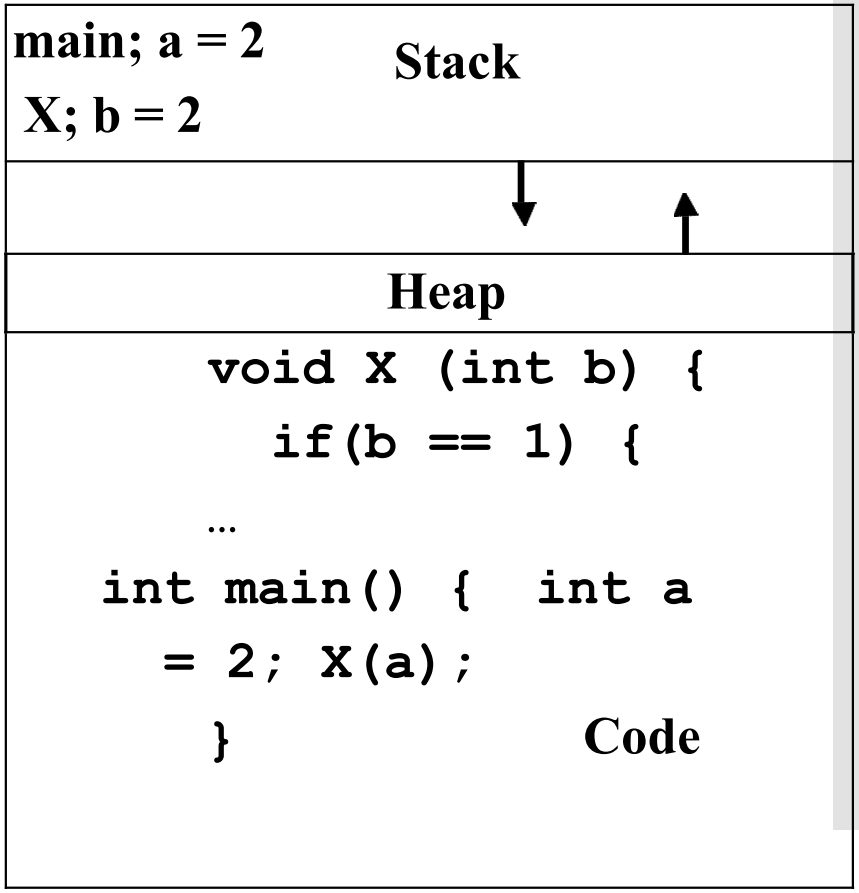
Process in Memory

- Program to process.
  - What you wrote

```
void X (int b) {
  if(b == 1) {
…
 int main()
   {  int a
   =      2;
   X(a);
}
```

  - What must the OS track for a process?

- What is in memory.



Process VS Program

S. 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.	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 does not have any control block.	Process has its own control block called Process Control Block.

Address Space

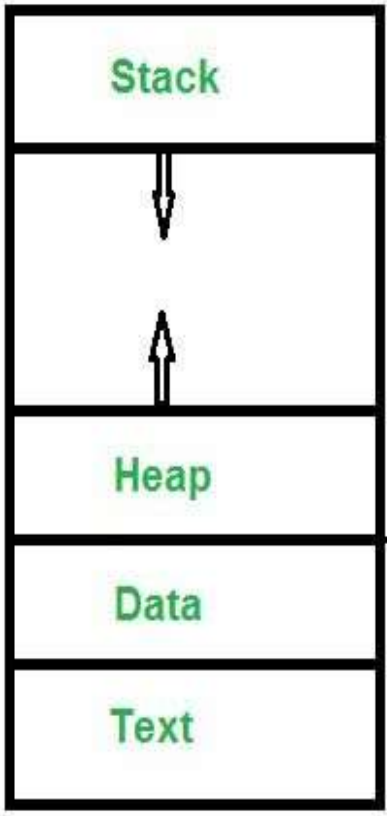
What does a process look like in memory?

Stack: The stack contains temporary data, such as method/function parameters, returns addresses, and local variables.

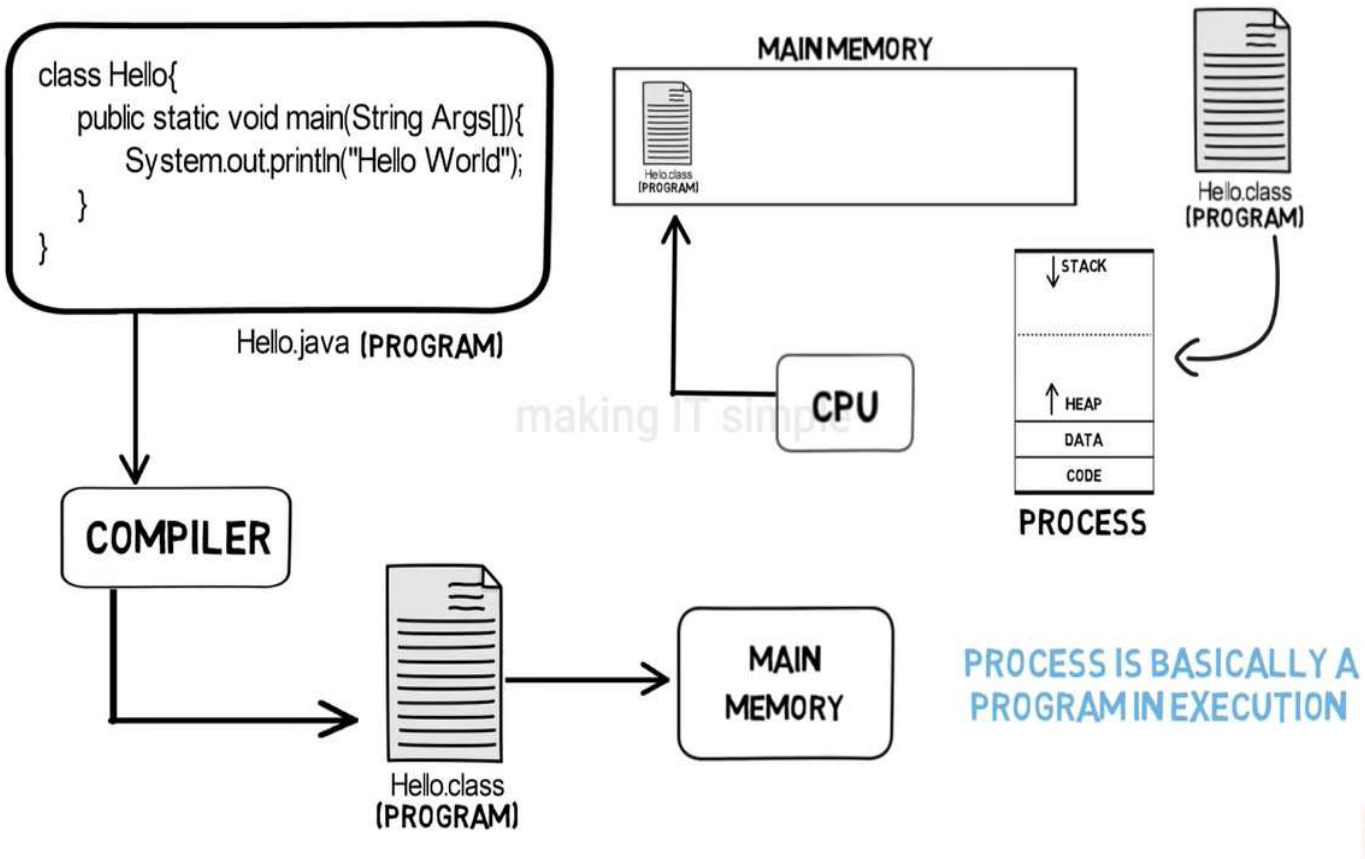
Heap Section: Dynamically allocated memory to process during its run time.

Data Section: Contains the global variable.

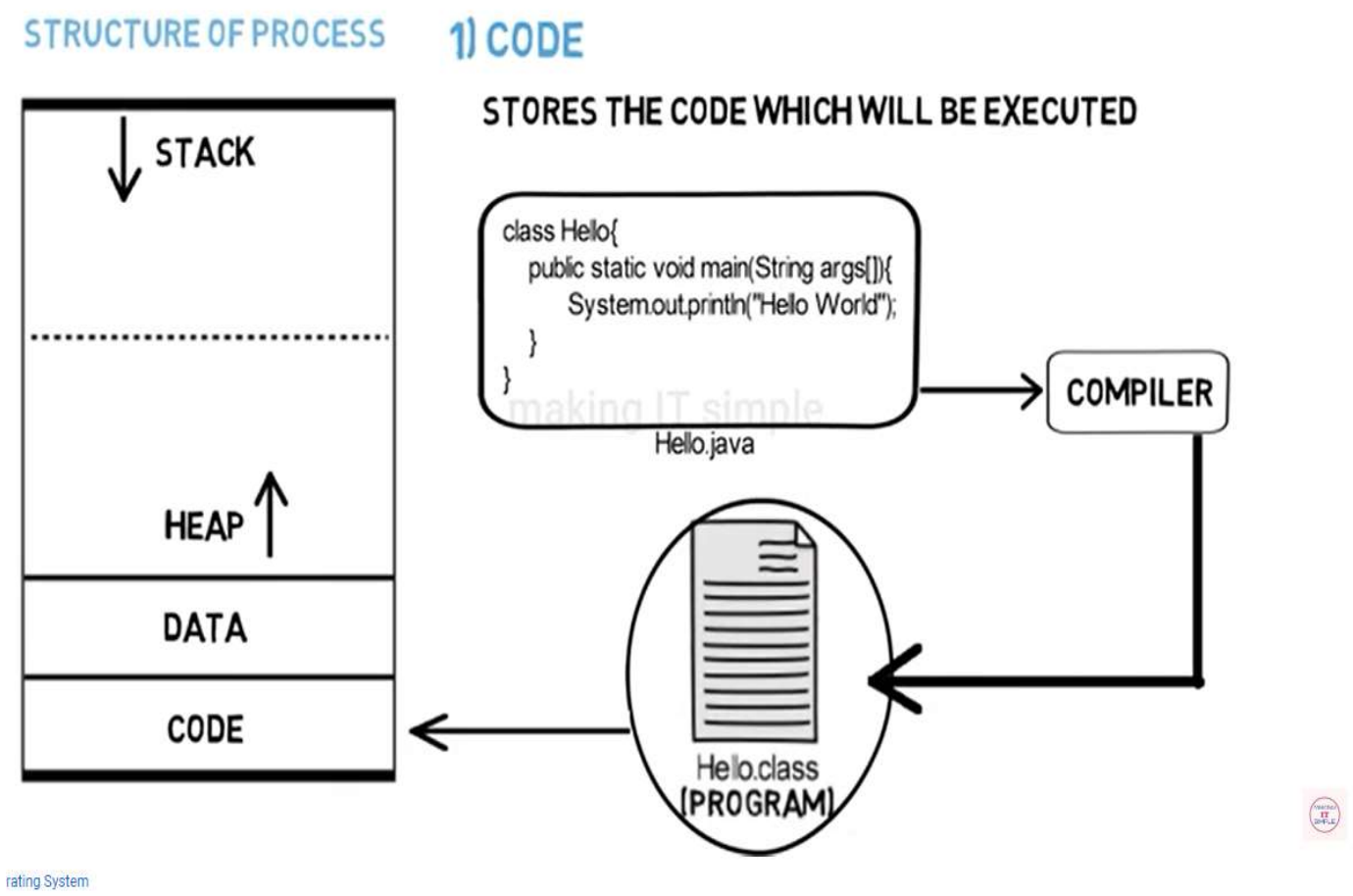
Text / Code Section: A Process, sometimes known as the Text Section, also includes the current activity represented by the value of the Program Counter.



Address Space

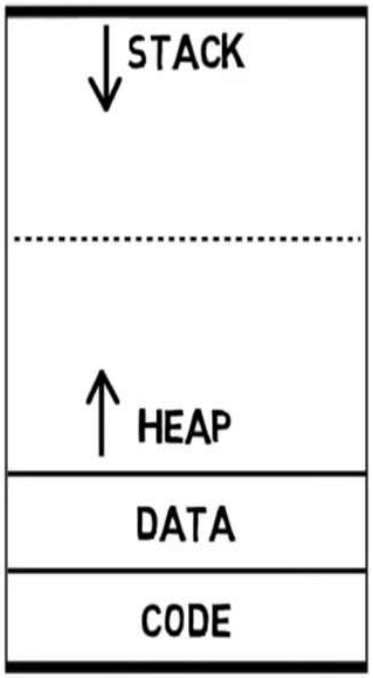


Address Space



Address Space

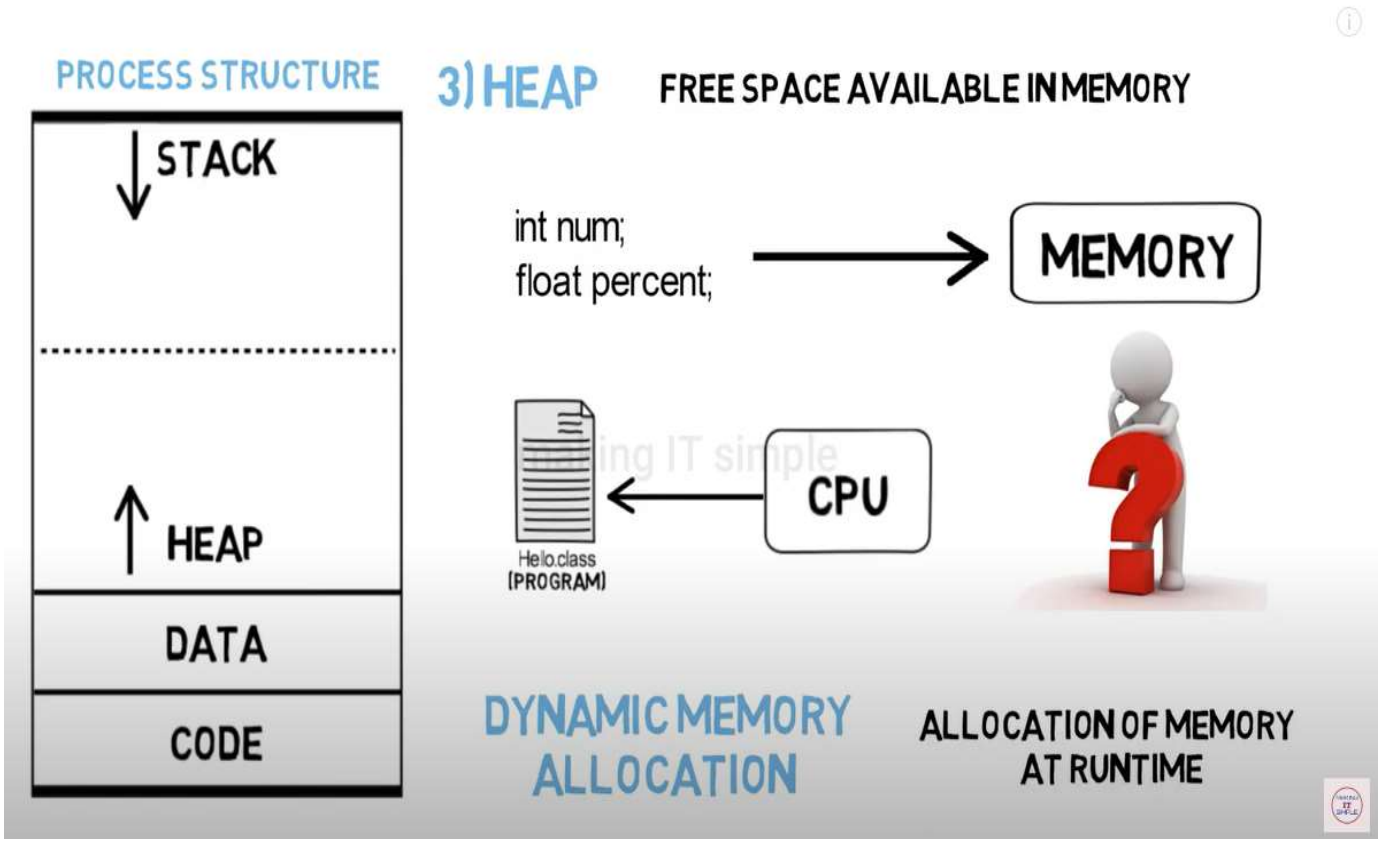
STRUCTURE OF PROCESS



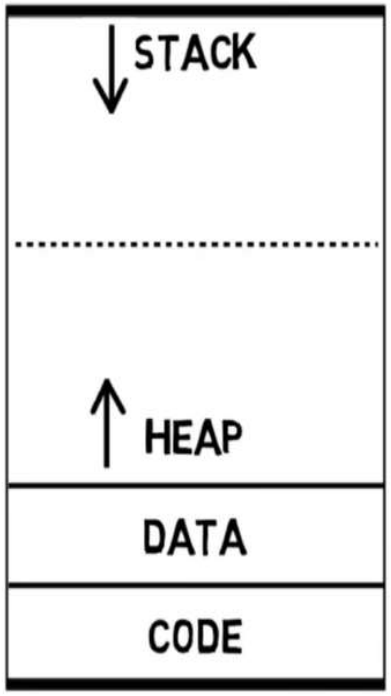
2) DATA

STATIC VARIABLE	GLOBAL VARIABLE
THESE ARE DECLARED AT COMPILE TIME	GLOBAL VARIABLE HAS A GLOBAL SCOPE
THEY REMAIN UNTILL COMPLETE RUN OF PROGRAM	IT CAN BE ACCESSED ANYWHERE IN THE PROGRAM
VALUE ASSIGNED TO STATIC VARIABLE IS PRESERVED	

Address Space



Address Space



4) STACK

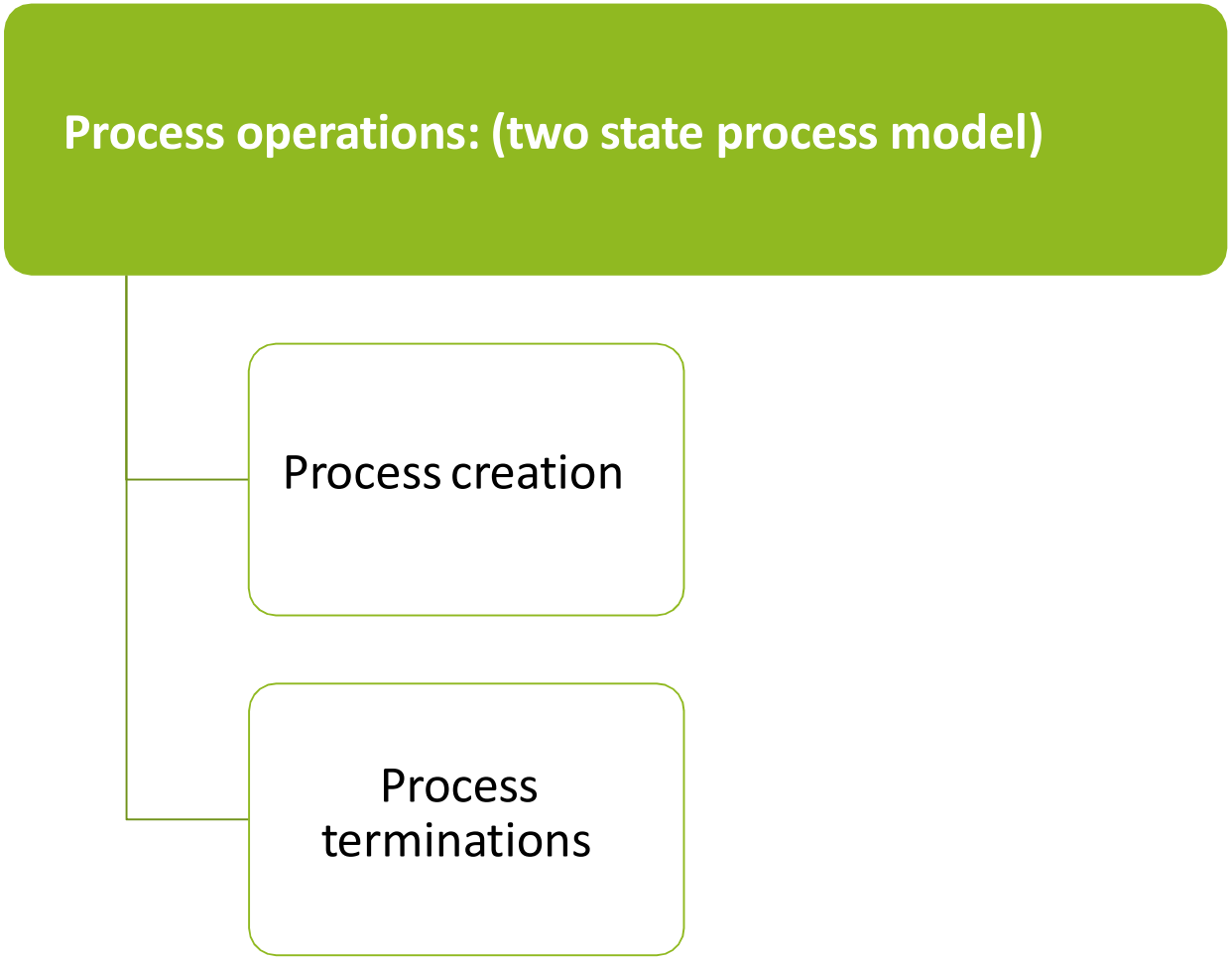
```
int number;
float percentage;

public static int max( int a, int b ){
        .
        .
        return maxnum;
}

public int fact ( int num)
{
        if ( num == 0 )
                return 1;
        else
                return num * fact( num - 1 );
}
```

LOCAL VARIABLES

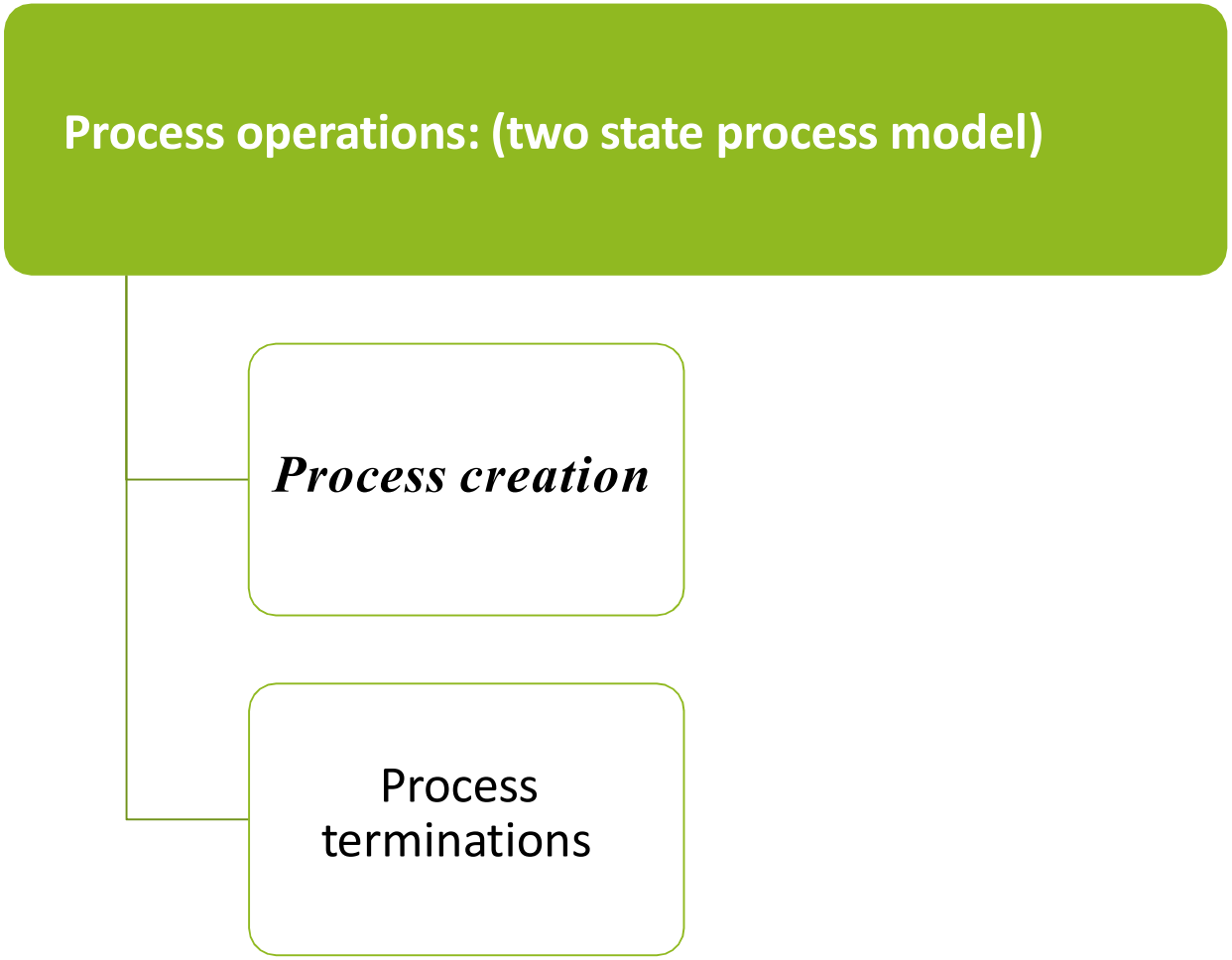
PARAMETERS AND RETURN ADDRESS

RECURSIVE CALLS

Process Operations



Process Operations



Process Operations

System initialization:

- When OS is booted, several system process are created. They provide system services. They do not need user interaction it just execute in background.

Execution of a process creation system call:

- A running process can issue system call to create new process. i.e in unix system call name “fork” is used to create new process.

Process Operations

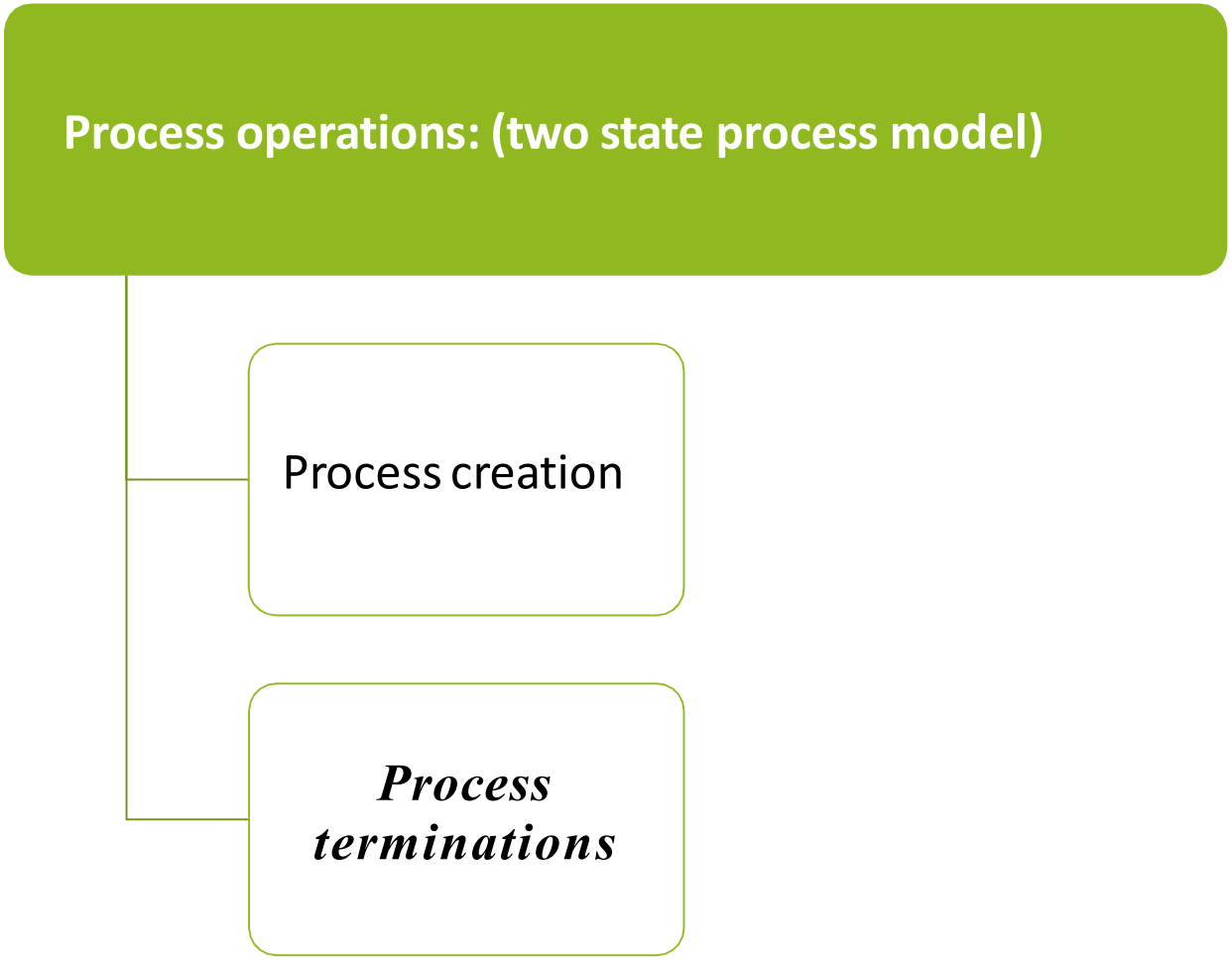
A user request to create a new process

- User can start new process by typing command or double click on some application icon.

Initiation of a batch job:

- This applies only to the batch system. Here user submit batch jobs to the system. When there are enough resources are free to execute a next job a new job is selected from batch queue.

Process Operations



Process Operations

Normal exit (voluntary)

- Terminate when done their job. i.e. when all the instructions from program get executed.
- Even user can select option to terminate the process.

Error exit (voluntary):

- Process even terminated when some error occurred. Example divide by zero error.

Process Operations

Fatal error (involuntary):

- Fatal error are generated due to user mistake in executing program. Such as file name not found error.

Killed by another process (involuntary):

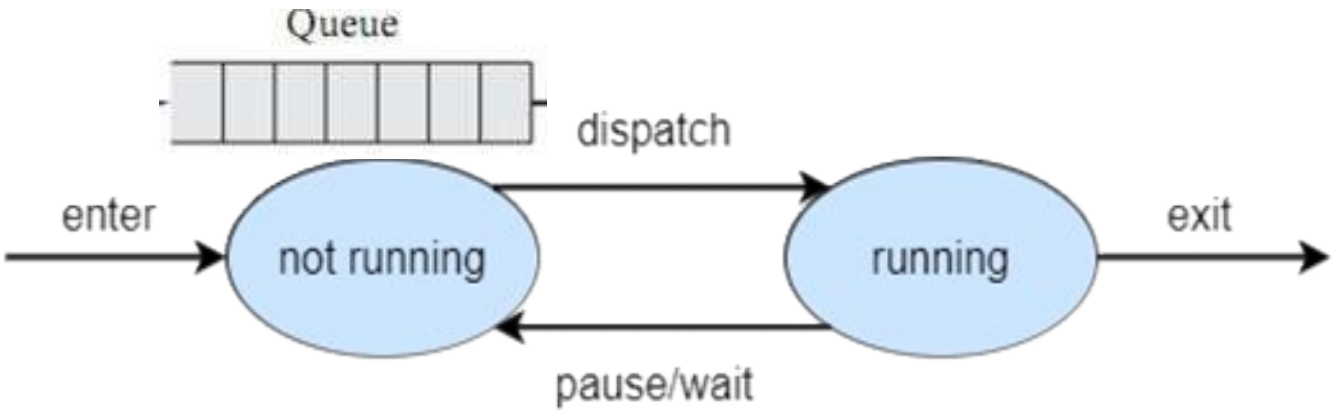
- If some other process request for OS to kill the process. Then UNIX it can be done by “kill” system call while in windows “TerminateProcess”

Process States / Process Life Cycle

- When a process executes, it passes through different states.
- These stages may differ in different operating systems, and the names of these states are also not standardized.
- When process executes, it changes state.
- Process state is defined as the current activity of the process.

Two State Process Model

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



Two State Process Model

- **NOT RUNNING**

  When O.S. is create the process and enter into the main memory for execution this time its state will be NOT RUNNING and these processes are waiting for the execution

- **RUNNING**

  When process is going for execution its state is called RUNNING.

Two State Process Model

- When a process is first created by the OS, it initializes the program control block for the process and the new process enters the system in Not-running state.
- After some time, the currently running process will be interrupted by some events, and the OS will move the currently running process from Running state to Not-running state.
- The dispatcher then selects one process from Not-running processes and moves the process to the Running state for execution.

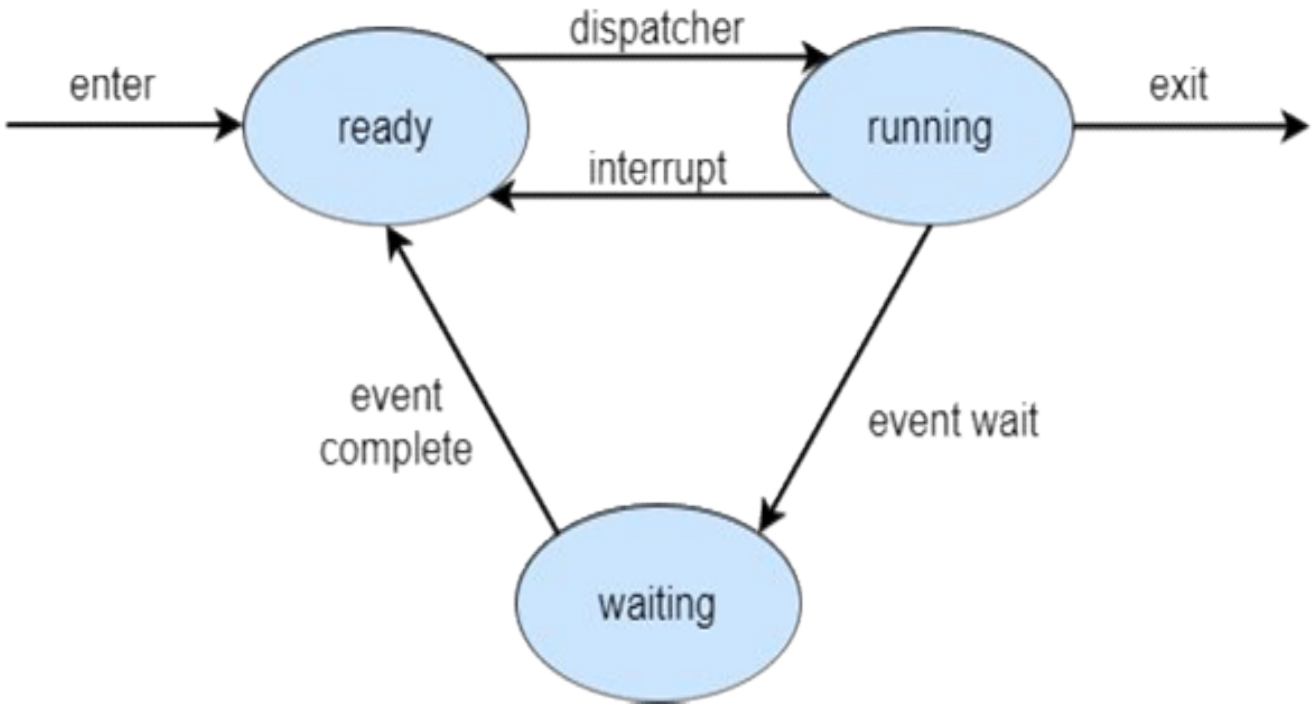
Two State Process Model

- For example, if we implement round-robin scheduling then the running process moves to not-running state after the time quantum.
- When a process finishes the execution, the process exits the system and the dispatcher again selects a new process and moves it to Running state.
- All the processes in the Not-running state are maintained in a queue.

Three State Process Model

- There is one major drawback of two state process model. When dispatcher brings a new process from not-running state to running state, the process might still be waiting for some event or I/O request.
- So, the dispatcher must traverse the queue and find a not-running process that is ready for execution. It can degrade performance.

Three State Process Model



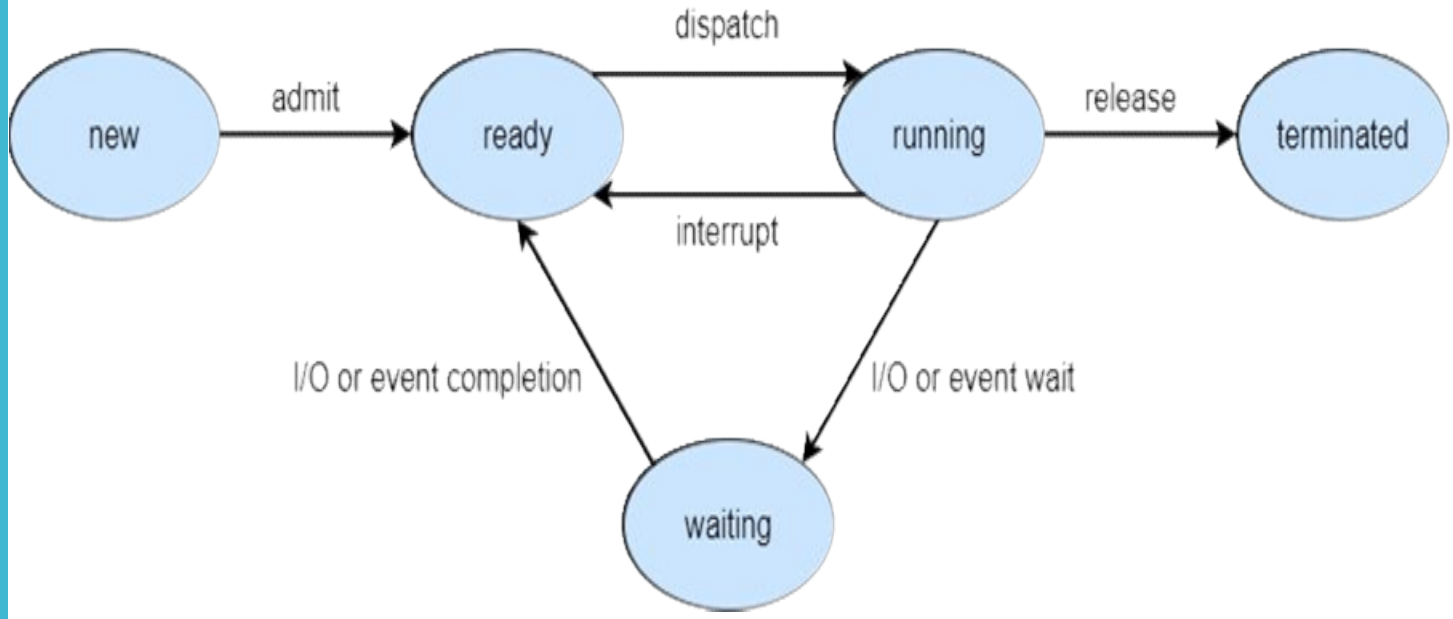
Three State Process Model Transition Diagram

Three State Process Model

- To overcome this problem, we split the not-running state into two states: Ready State and Waiting (Blocked) State.
  - Ready State: The process in the main memory that is ready for execution.
  - Waiting or Blocked State: The process in the main memory that is waiting for some event.
- The OS maintains a separate queue for both Ready State and Waiting State.
- A process moves from Waiting State to Ready State once the event it's been waiting for completes.

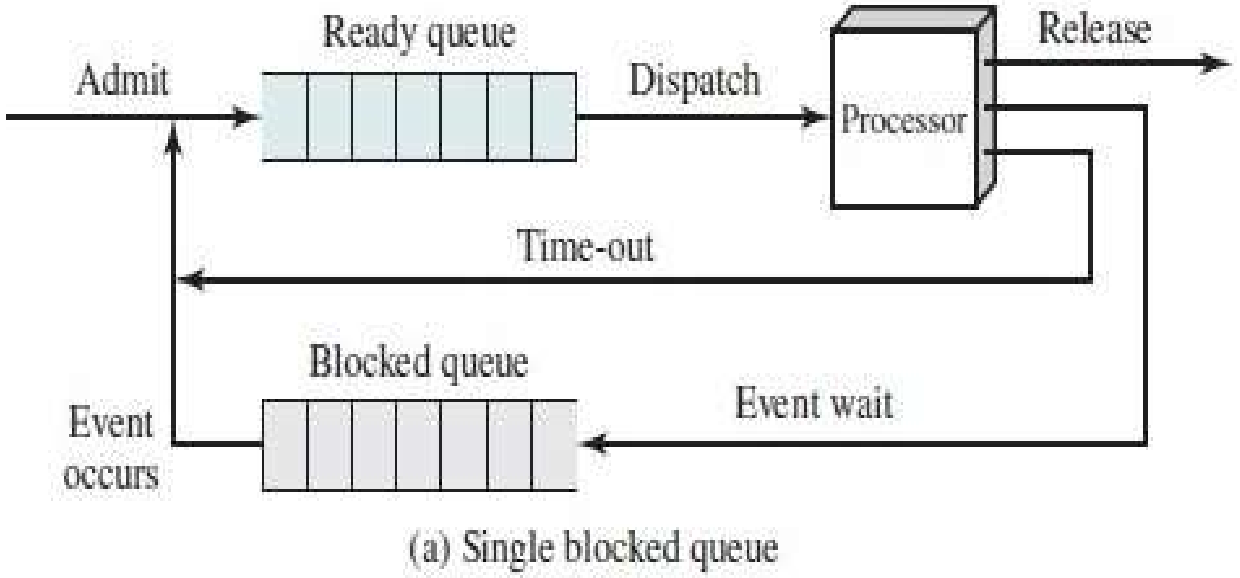
Five State Process Model

In the five state model, we introduce two new states: new state and terminated state.



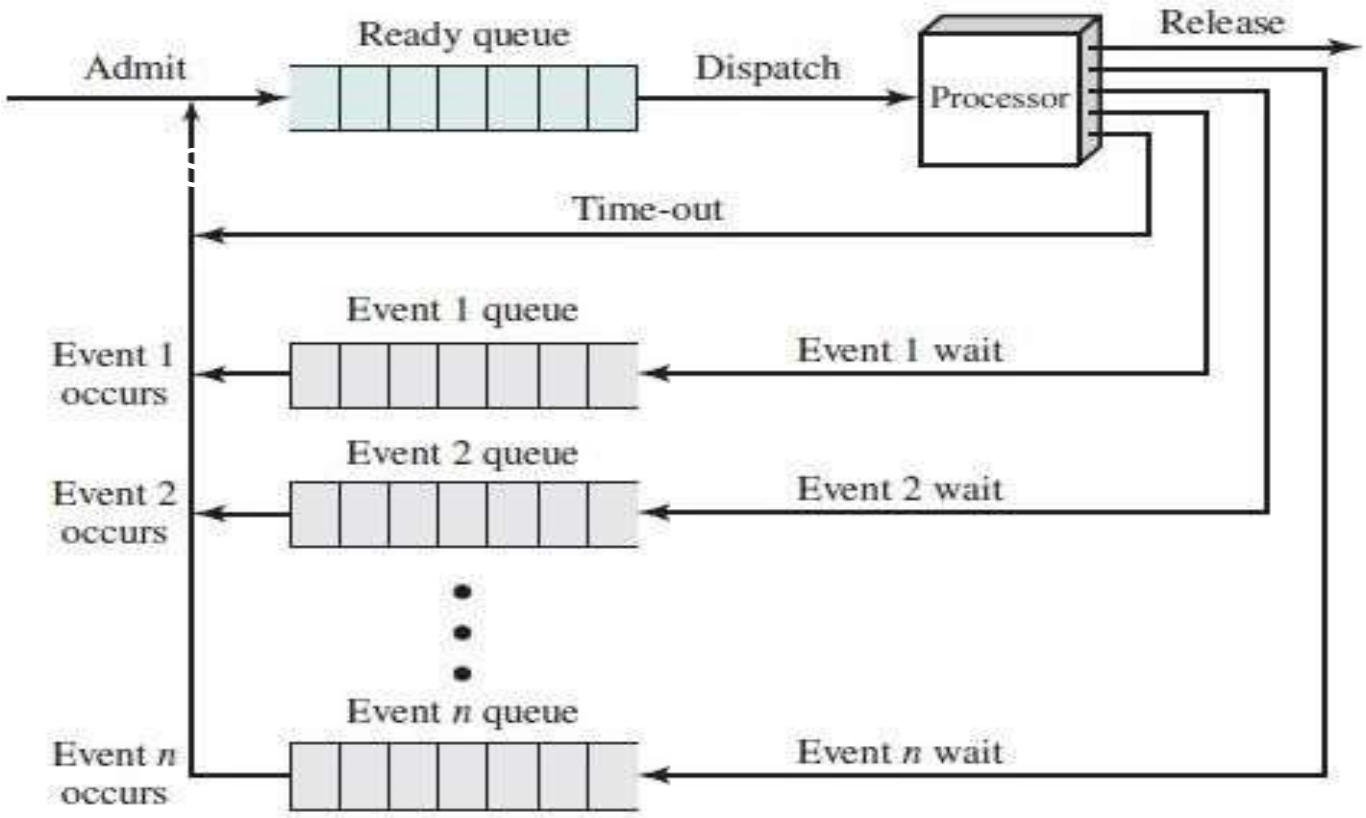
Five-State Process Model State Transition Diagram

Five State Process Model



Five State Process Model Queuing Diagram (With single blocked queue)

Five State Process Model



Five State Process Model Queuing Diagram (With multiple blocked queue)

Five State Process Model

Reason for New State

- In the previous models, we assumed that the main memory is large enough to accommodate all programs but this is not true. Modern programs are very large. Loading all processes in the main memory is not possible.
- When a new process is made, its program is not loaded in the main memory. The OS only stores some information about the process in the main memory. The long term scheduler moves the program to the main memory when sufficient space is available. Such a process is said to be in new state.

Five State Process Model

Reason for Terminated State

- In the previous models, when a process finishes execution, its resources are immediately freed. But there might be some other process that may need its data in the future.
- For example, when a child process finishes execution, the OS preserves its data until the parent call wait(). The child process is still in memory but not available for execution. The child process is said to be in terminated state.

Five State Process Model

- **Running:** The currently executing process.
- **Waiting/Blocked:** Process waiting for some event such as completion of I/O operation, waiting for other processes, synchronization signal, etc.
- **Ready:** A process that is waiting to be executed.
- **New:** The process that is just being created. The Program Control Block is already being made but the program is not yet loaded in the main memory. The program remains in the new state until the long term scheduler moves the process to the ready state (main memory).
- **Terminated/Exit:** A process that is finished or aborted due to some reason.

Five State Process Model (Cont.)

State Transitions

- **New -> Ready:**
  - The long term scheduler picks up a new process from second memory and loads it into the main memory when there are sufficient resources available. The process is now in ready state, waiting for its execution.
- **Ready -> Running:**
  - The short term scheduler or the dispatcher moves one process from ready state to running state for execution.

Five State Process Model (Cont.)

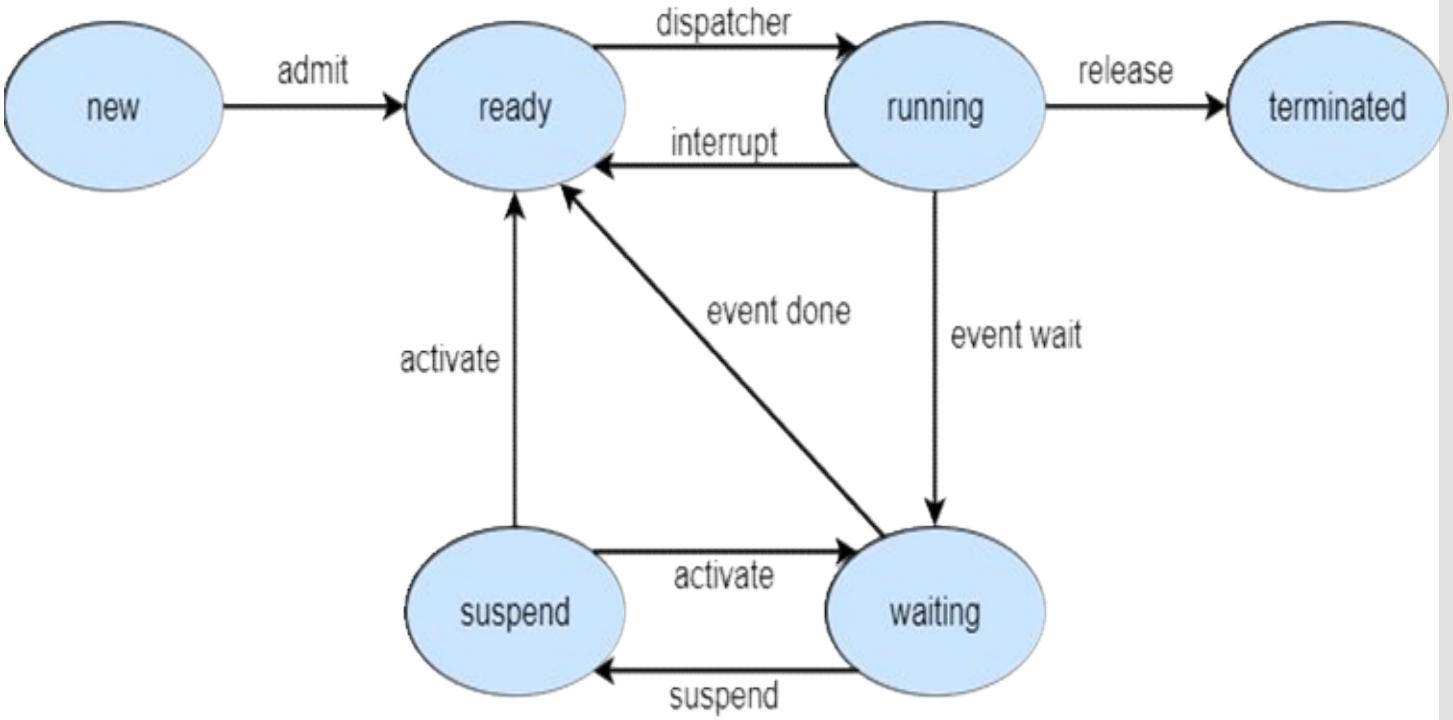
- **Running -> Terminated:**
  - The OS moves a process from running state to terminated state if the process finishes execution or if it aborts.
- **Running -> Ready:**
  - This transition can occur when the process runs for a certain amount of time running without any interruption. For example, if we use round-robin to schedule processes, then the running process will move to the ready state after time quantum. Another example is if the priority of a process in the ready state is more than the priority of the currently running process, then OS may preempt the running process and move it to ready state.

Five State Process Model (Cont.)

- **Running -> Waiting:**
  - A process is put in the waiting state if it must wait for some event. For example, the process may request some resources or memory which might not be available. The process may be waiting for an I/O operation or it may be waiting for some other process to finish before it can continue execution.
- **Waiting -> Ready:**
  - A process moves from waiting state to ready state if the event the process has been waiting for, occurs. The process is now ready for execution.

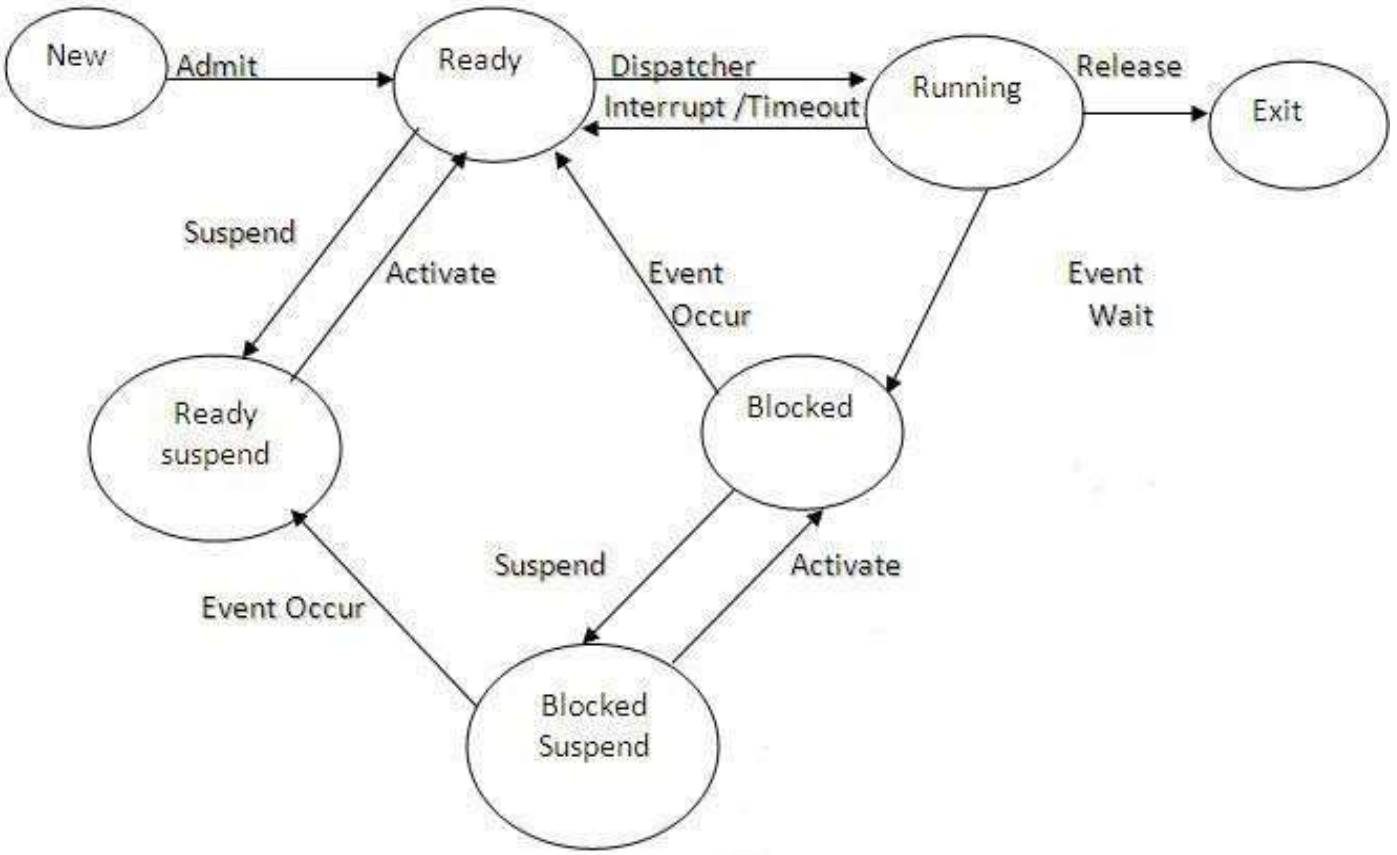
Six State Process Model

It is commonly as Five state process model with suspend state.



Six State Process Model Transition Diagram

Seven State Process Model



Animated video of 7state process model : https://www.youtube.com/watch?v=OP4WG5dRIXA

Seven State Process Model (Cont.)

- It is commonly known as Five state process model with two suspended states.
- There is one major drawback in the previous process state model. That is, the CPU doesn't know which process in the suspend queue is ready for execution. CPU may swap a process that is still waiting for event completion from secondary memory back to the main memory. There is no point in moving a blocked process back to the main memory. The performance suffers.

Seven State Process Model (Cont.)

- To avoid this, we divide the suspend state into 2 states:
  - Blocked/Suspend: The process is in secondary memory but not yet ready for execution.
  - Ready/Suspend: The process is in secondary memory and ready for execution.

Seven State Process Model (Cont.)

State Transitions:

Blocked-> Blocked/Suspend :

- If all the processes in the main memory are in the waiting state, the processor swaps out at least one waiting process back to secondary memory to free memory to bring another process.

- **Blocked/Suspend -> Blocked:**
  - This transition might look unreasonable but if the priority of a process in Blocked/Suspend state is greater than processes in Ready/Suspend state then CPU may prefer process with higher priority.

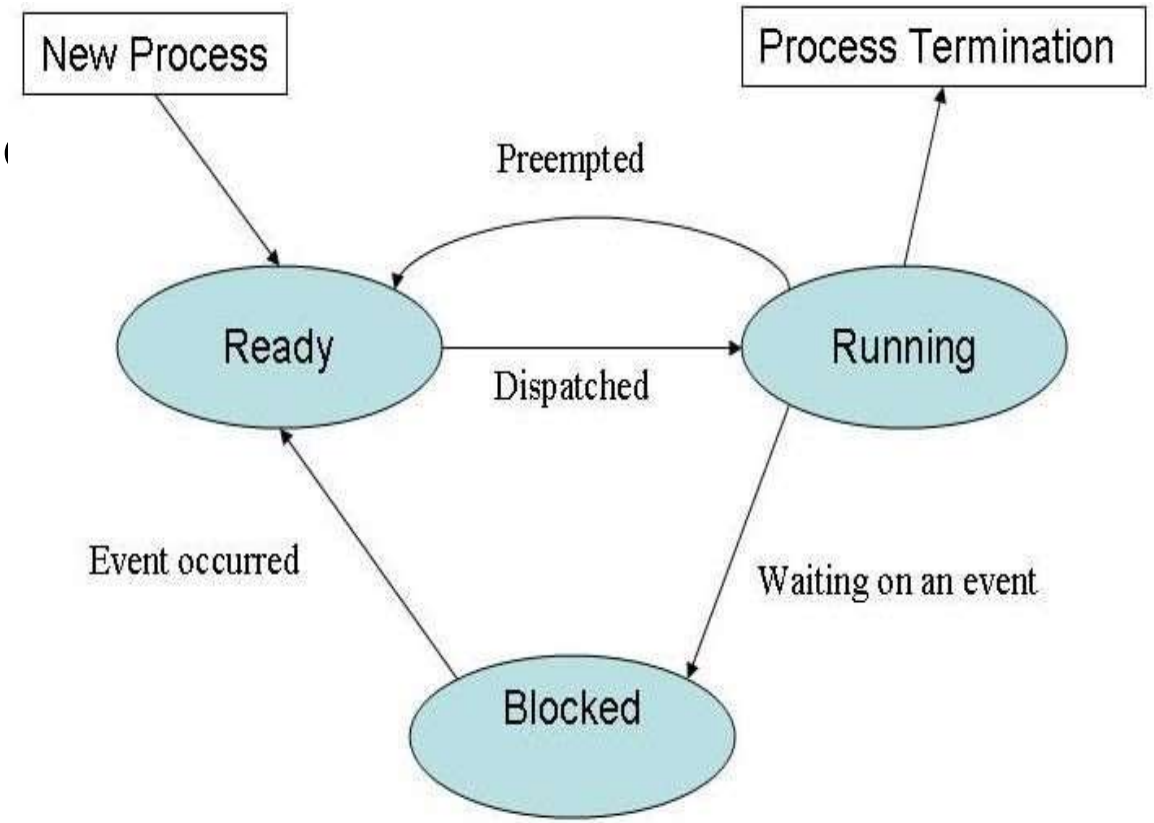
Seven State Process Model (Cont.)

- **Blocked/Suspend -> Ready/Suspend:**
  - The process moves from Blocked/Suspend to Ready/Suspend state if the event, the process has been waiting for occurs.
- **Ready/Suspend -> Ready:**
  - The OS moves a process from secondary memory to the main memory when there is sufficient space available. Also, if there is a high priority process in Ready/Suspend state, then OS may swap it with a lower priority process in the main memory.
- **Ready -> Ready/Suspend:**
  - The OS moves a process from the ready state to ready/suspended to free main memory for a higher priority process.

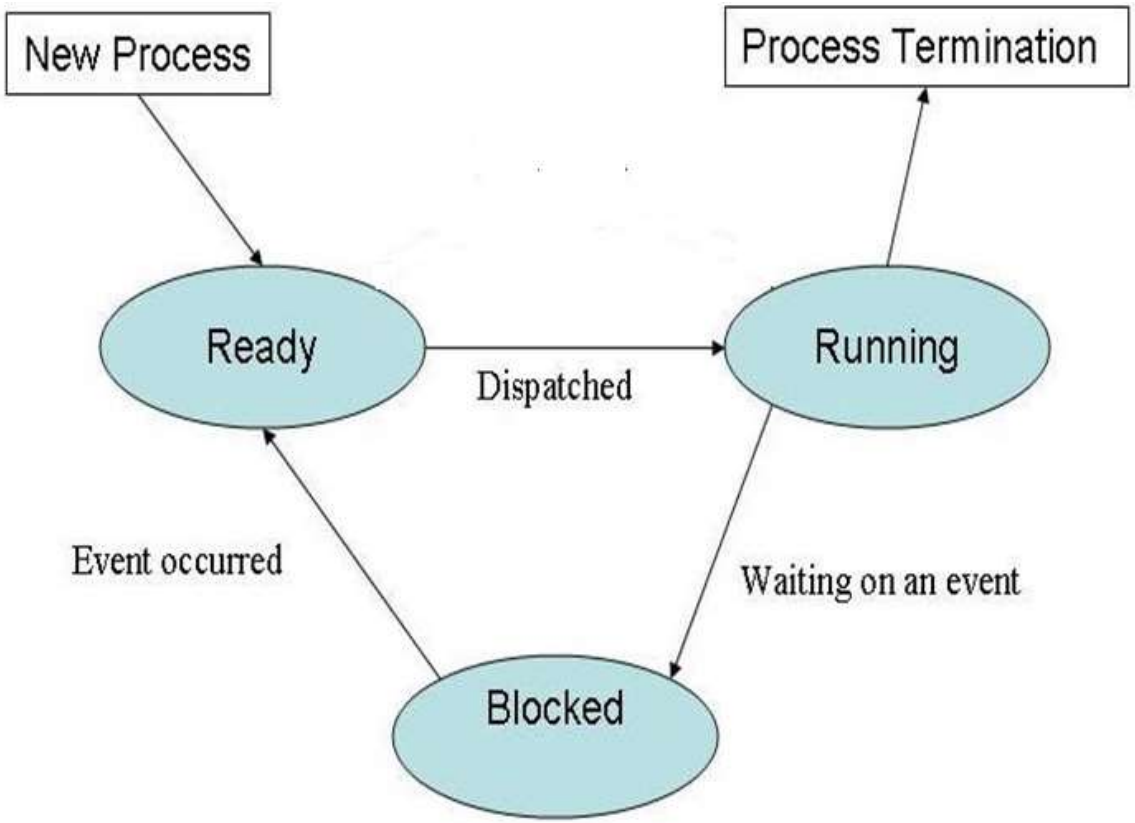
Seven State Process Model (Cont.)

In all the process state models, a process can directly move from any state to terminated state. This is because the parent process can terminate the child process at any moment.

Seven State Process Model (Cont.)



CPU Non-preemption process state diagram



Q - 01

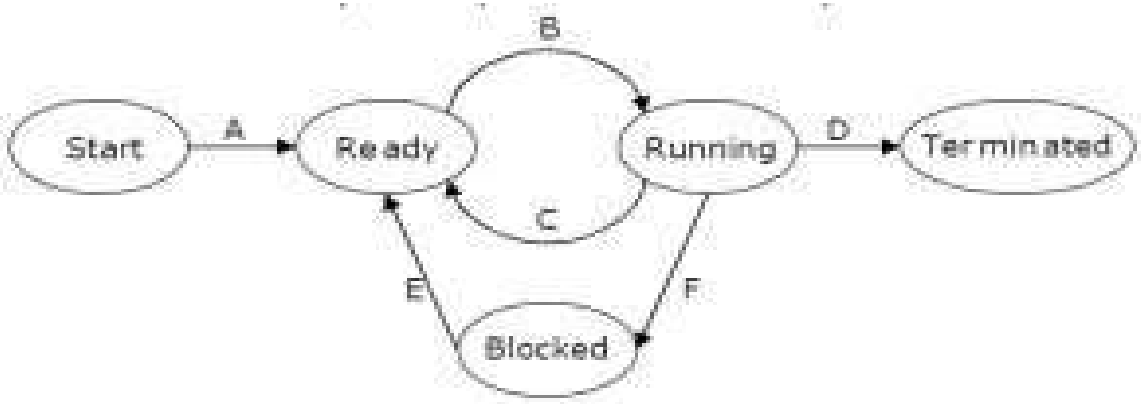
I. If a process makes a transition D, it would result in another process making transition A immediately.

II. A process P2 in blocked state can make transition E while another process P1 is in running state.

III. The OS uses preemptive scheduling.

IV. The OS uses non-preemptive scheduling.

Which of the above statements are TRUE?



Q- 1
Solution

- If a process makes a transition DD, it would result in another process making transition A immediately. - This is false. It is not said anywhere that one process terminates, another process immediately come into Ready state. It depends on availability of process to run & Long term Scheduler.
- A process P2 in blocked state can make transition E while another process P2 is in running state. - This is correct. There is no dependency between running process & Process getting out of blocked state.
- The OS uses preemptive scheduling. :- This is true because we got transition CC from Running to Ready.
- The OS uses non-preemptive scheduling. Well as previous statement is true, this becomes false.

So answer is II and III .

Q - 02

Process can get blocked from which of the following?

(A) Running or Suffered

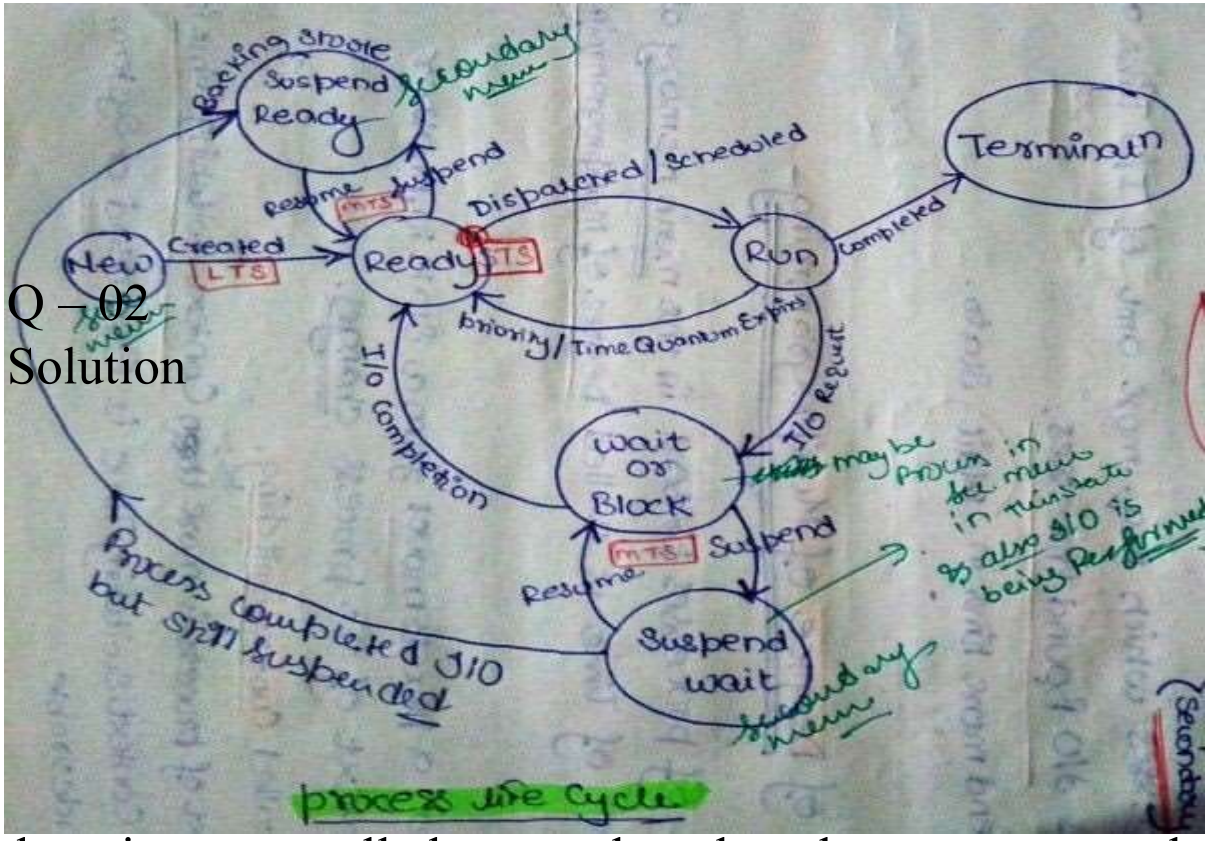
(B) Ready or New

(C) Ready or Running

(D) terminate or New

Q – 02
Solution

Q – 02
Solution



here is a state called suspend ready, when system needs to free up resources, some processes are moved to that state from ready state

Answer-C

Process Control Block (PCB)

OR

Task Control Block (TCB)

- PCB is a data structure in the OS and it contains all the information about a process.
- While creating a process the operating system performs several operations. To identify the processes, it assigns a process identification number (PID) to each process.
- As the operating system supports multiprogramming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process execution status.

Process Control Block (PCB) OR Task Control Block (TCB) (Cont.)

Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc.

All these information is required and must be saved when the process is switched from one state to another. When the process makes a transition from one state to another, the operating system must update information in the process's PCB.

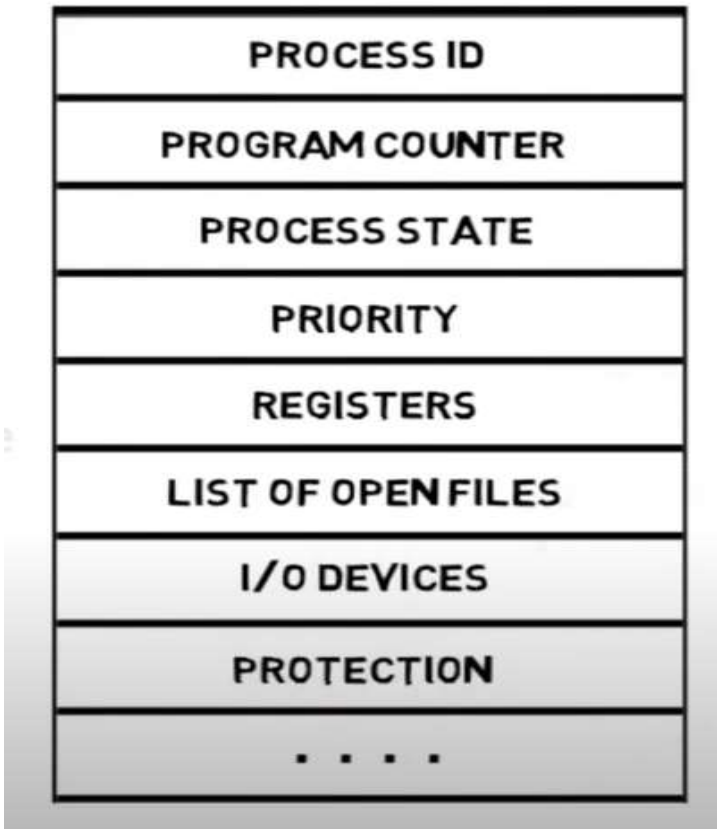
Process Control Block (PCB)

OR
Task Control Block (TCB)
(Cont.)

How PCB of a process is created in OS?

When a process is created, the operating system creates a corresponding process control block for storing the information of that process.

Process Control Block (PCB)



Process Control Block (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.

Process Control Block (PCB)

- **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.
- **Miscellaneous accounting and status data** – This field includes information about the amount of CPU used, time constraints, jobs or process number, etc.

Process Control Block (PCB)

- The process Control block stores the register content also known as execution content of the processor when it was blocked from running.
- This execution content architecture enables the operating system to restore a process's execution context when the process returns to the running state.
- When the process makes a transition from one state to another, the operating system updates its information in the process's PCB.

Process Control Block (PCB)

Process ID	• it process identifier (PID), parent process identifier and user identifier.
Priority	• its typically numeric value. A process is assigned a priority at its creation.
Process state	• the current state of the process.
Program counter	• indicate the address of the next instruction to be executed for process.
CPU register	• content of the register when the CPU was last released by the process.

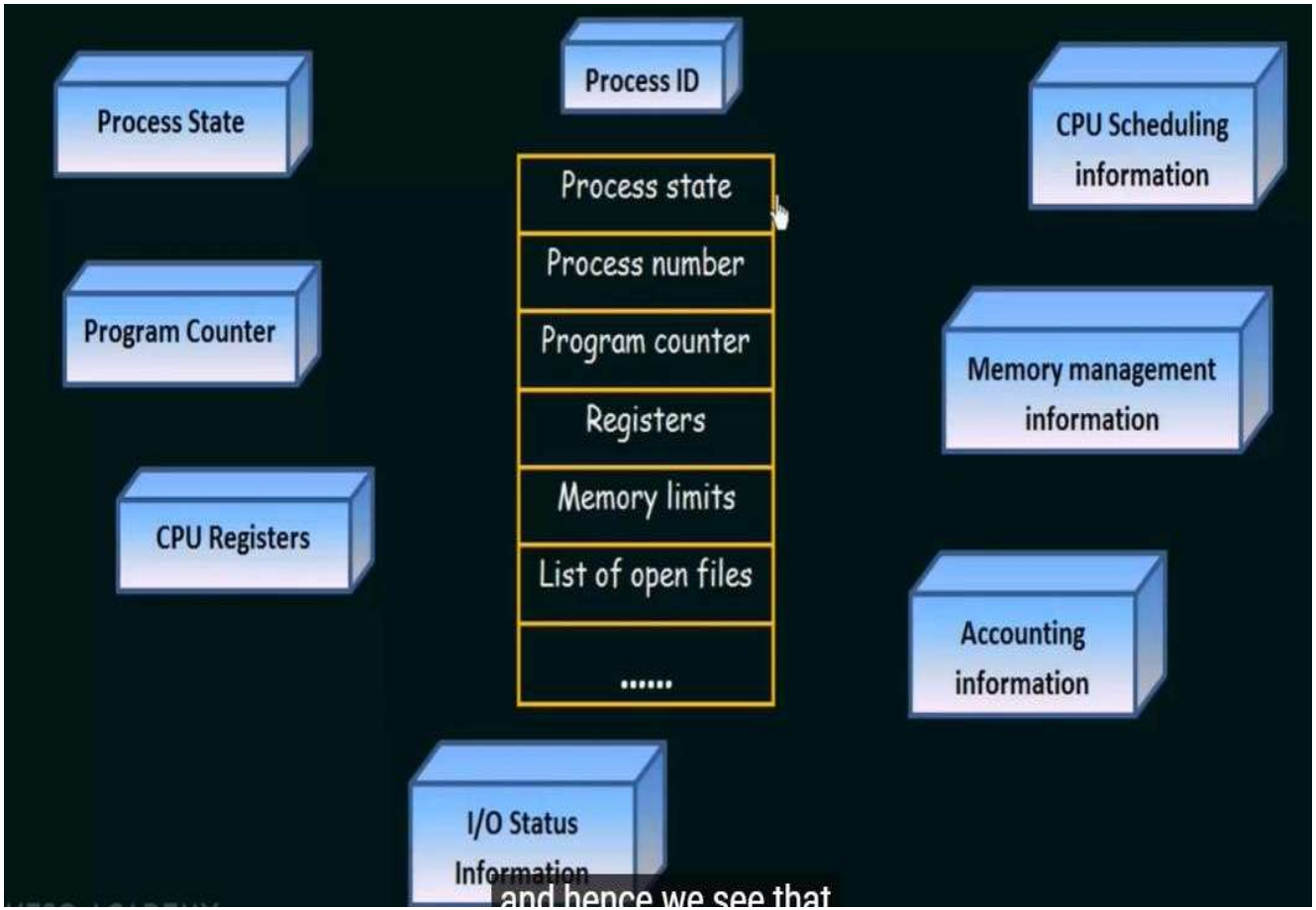
Process Control Block (PCB)

CPU scheduling information	• it includes process priority, pointer to various scheduling queue, information about events on which process is waiting and other parameters.
Program Status Word (PSW)	• this is snapshot i.e. the image of the PSW when the CPU was last voluntarily leave by the process. Loading this snapshot back into the PSW would resume execution of the program.
Memory management information	• it includes values of *base and limit registers*, information about page table or segment table.

Process Control Block (PCB)

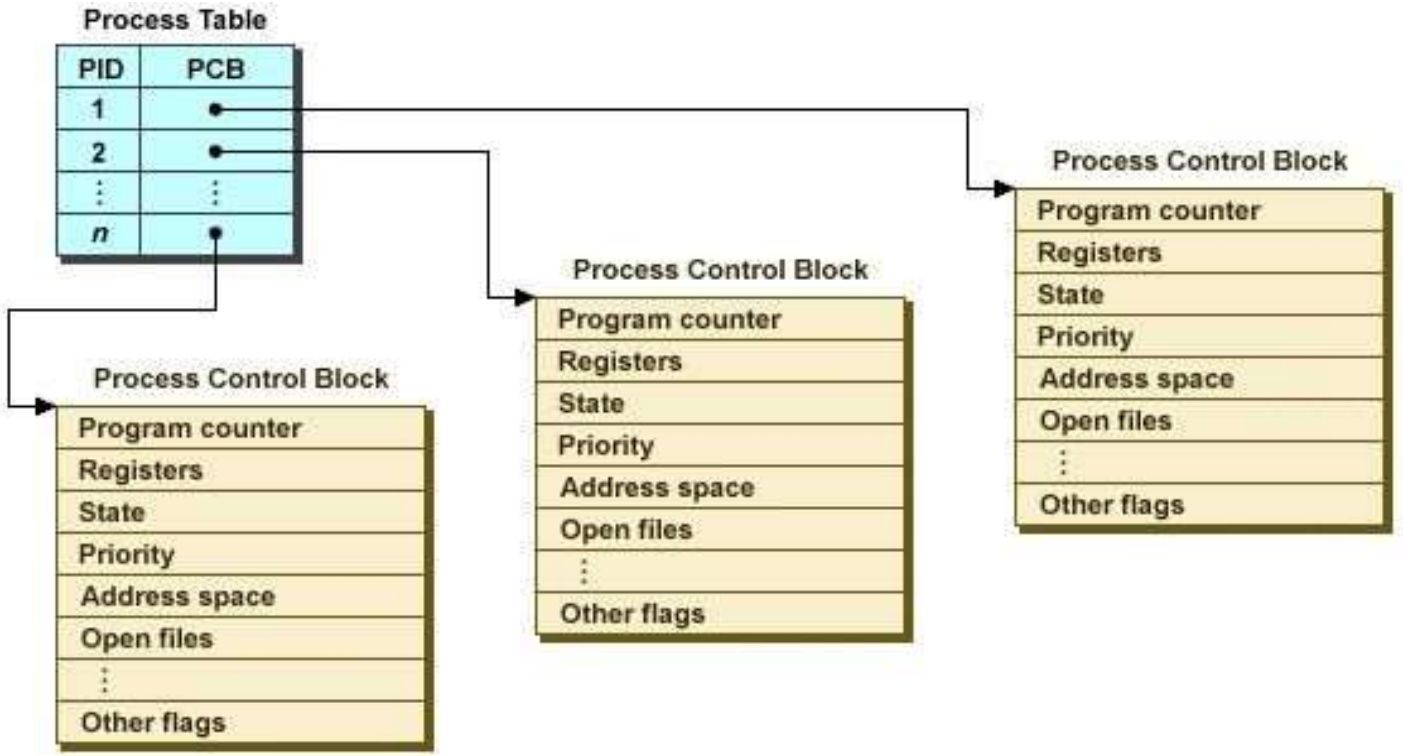
PCB pointer	• this field is used to form a list of PCBs. The kernel maintains several lists of PCBs.
Event information	• for a process in the *blocked state*, this field contains information about *event for which the process is waiting,* when an event occurs kernel uses this information.

Process Control Block



Process Table

The operating system maintains pointers to each process's PCB in a process table so that it can access the PCB quickly.



Process Scheduling

- The act of determining which process is in the ready state, and should be moved to the running state is known as Process Scheduling.
- 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.

Process Scheduling

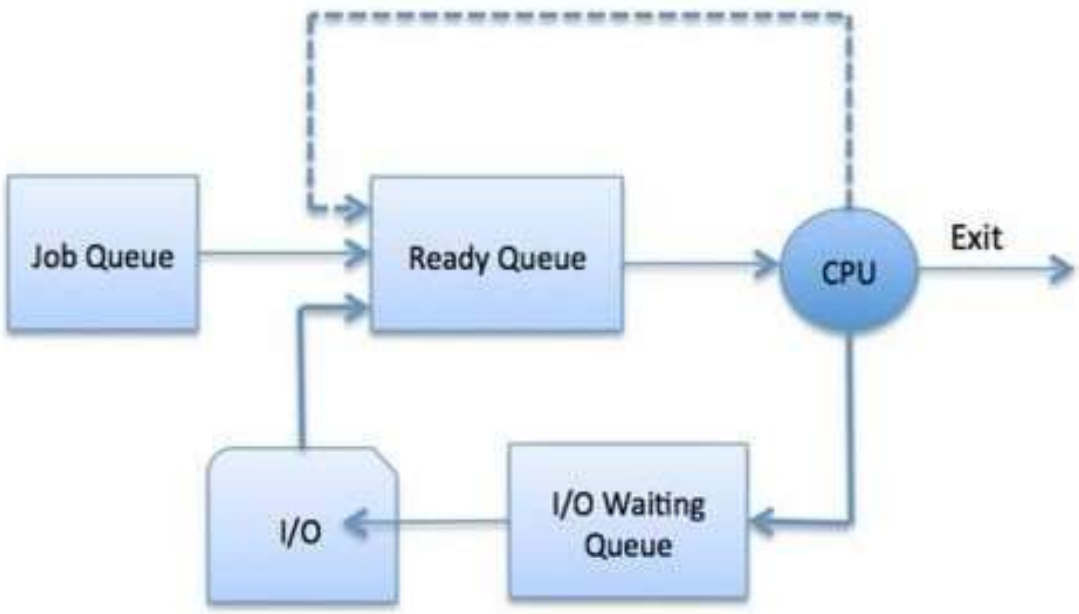
- Scheduling fell into one of the two general categories:
  - **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.
  - **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.

Process Scheduling

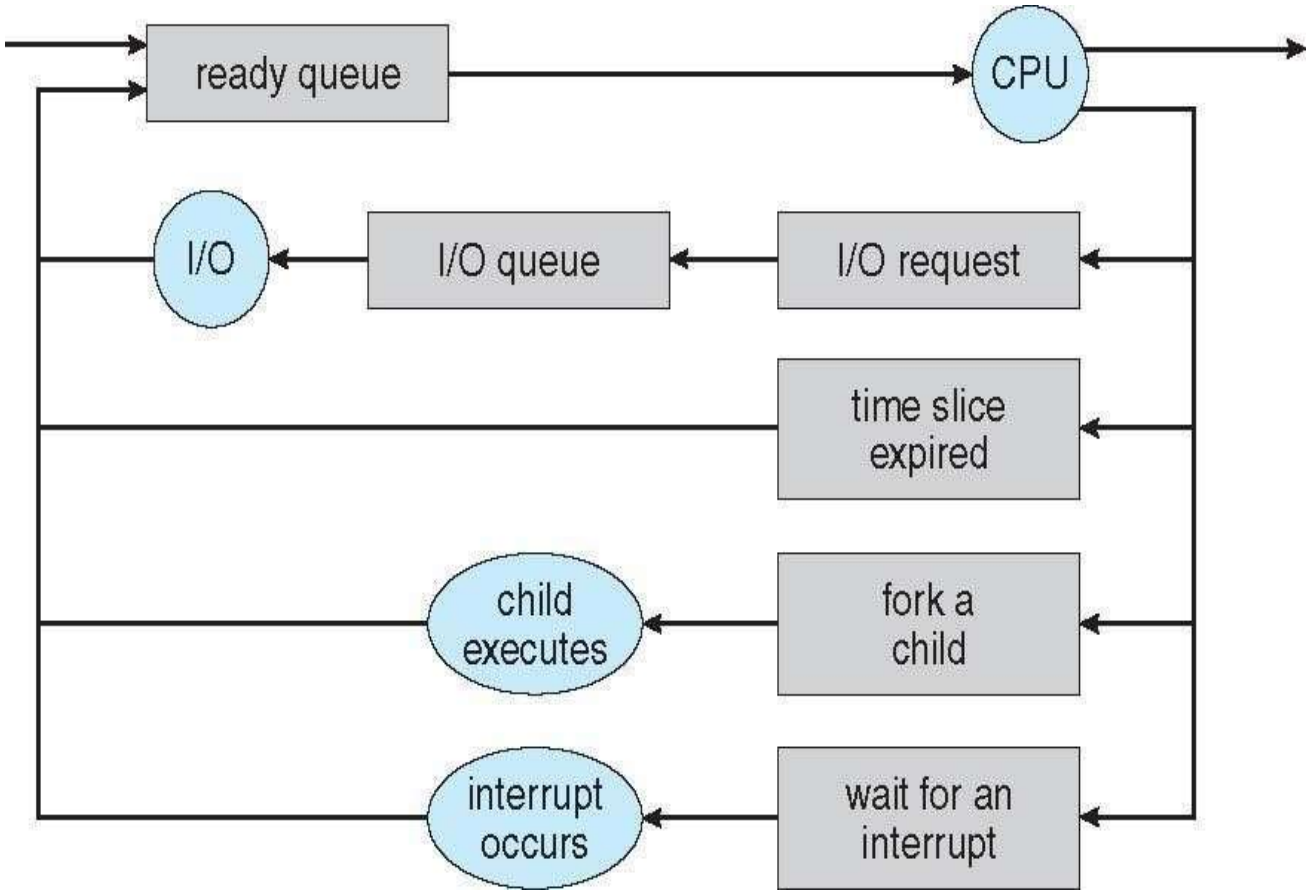
Parameter	Preemptive Scheduling	Non-Preemptive Scheduling
Basic	In this resources (CPU Cycle) are allocated to a process for a limited time.	Once resources (CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process cannot be interrupted until it terminates itself or its time is up.
Starvation	If a process having high priority frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then later coming process with less CPU burst time may starve.
Overhead	It has overheads of scheduling the processes.	It does not have overheads.
Flexibility	flexible	rigid
Cost	cost associated	no cost associated
CPU Utilization	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.
Examples	Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First.

Scheduling Queues / Process Queues

- Maintains scheduling queues of processes
  - Job queue
  - Ready queue
  - Device queues - I/O Queue
- Processes migrate among the various queues



Scheduling Queues / Process Queues (Cont.)



Queuing Diagram

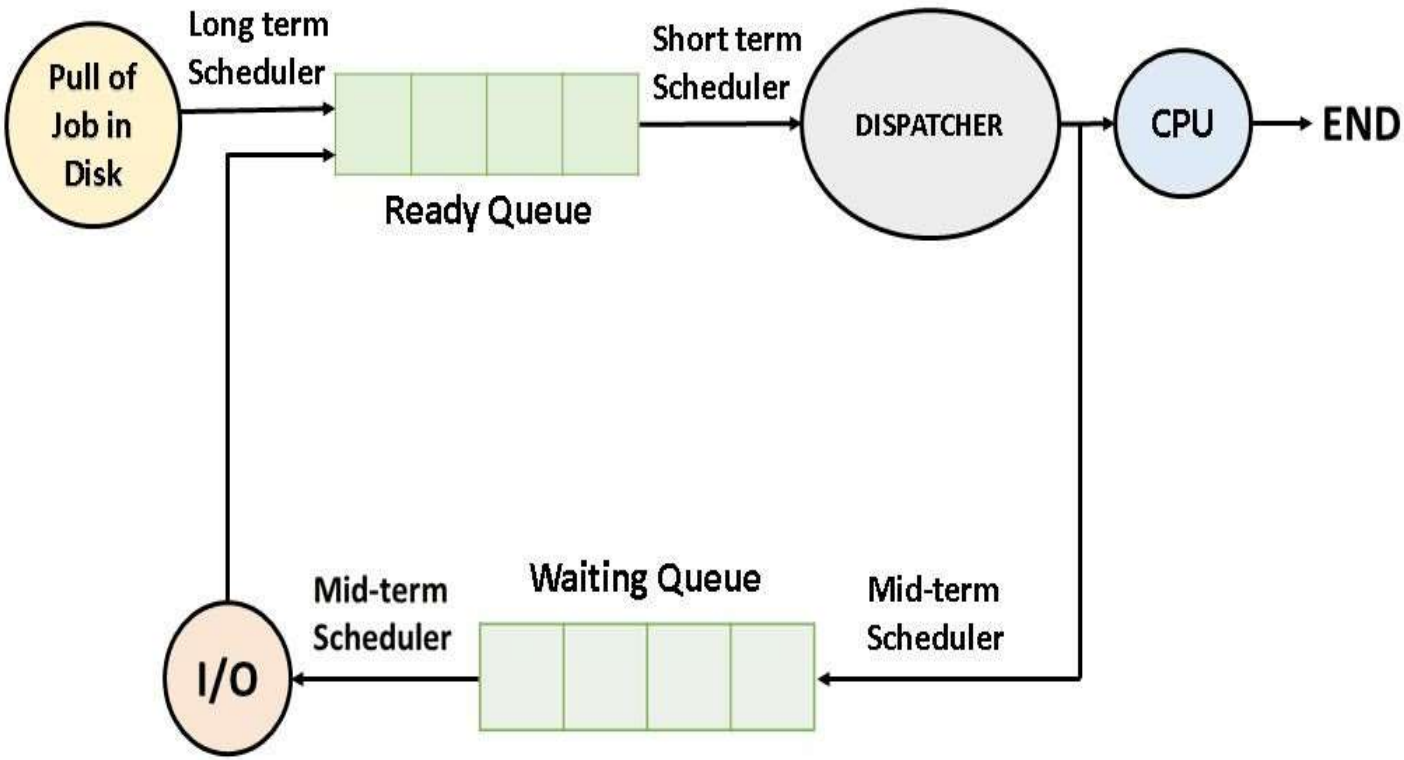
Scheduling Queues / Process Queues (Cont.)

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues.** There are unique device queues available for each I/O device.

Process 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.
- Schedulers are of three types
  - Long-Term Scheduler / job scheduler
  - Short-Term Scheduler / CPU scheduler
  - Medium-Term Scheduler

Process Scheduler



Long Term Scheduler

- Long term scheduler runs less frequently.
- Long Term Schedulers decide which program must get into the job queue.
- From the job queue, the Job Processor, selects processes and loads them into the memory for execution.
- Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming.
- An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.

Short Term Scheduler

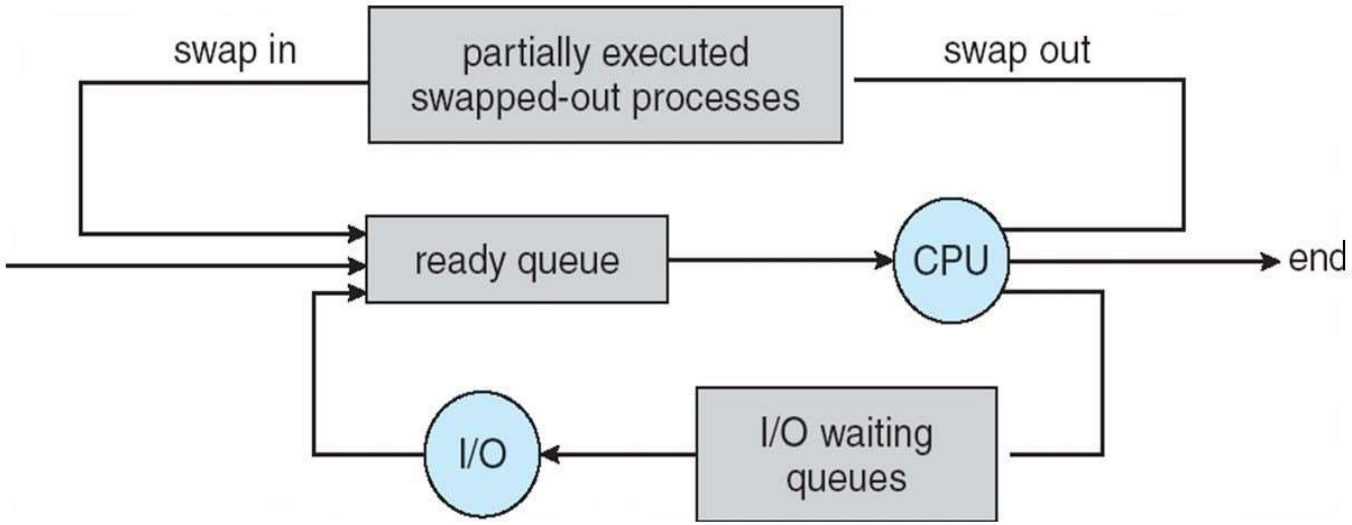
- This is also known as CPU Scheduler and runs very frequently.
- The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

Medium Term Scheduler

- This scheduler removes the processes from memory (and from active contention for the CPU), and thus reduces the degree of multiprogramming.
- At some later time, the process can be reintroduced into memory and its execution van be continued where it left off. This scheme is called swapping.
- The process is swapped out, and is later swapped in, by the medium term scheduler.

Medium Term Scheduler

- Swapping may be necessary to improve the process mix, or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

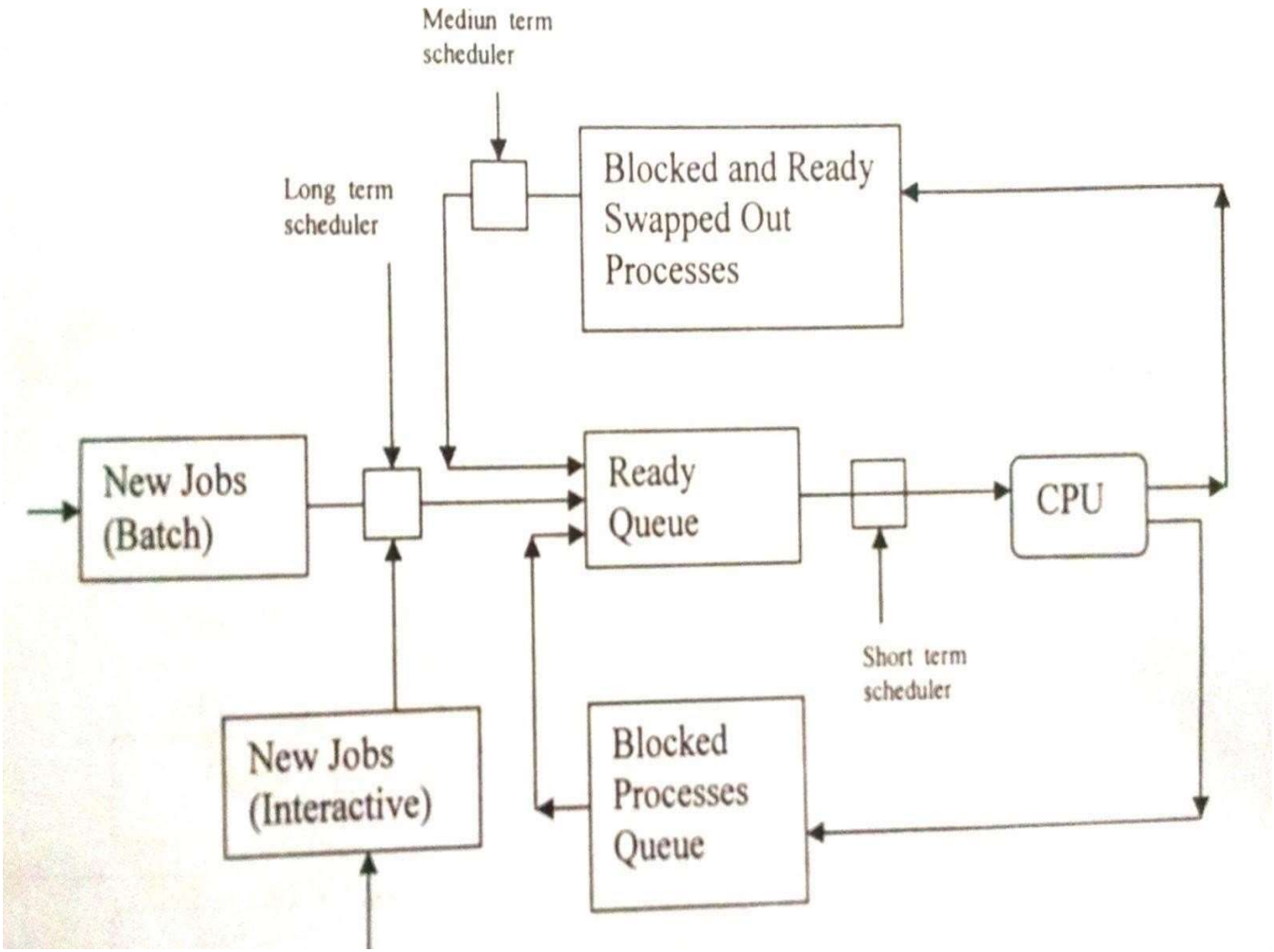


Addition of Medium-term scheduling to the queueing diagram.

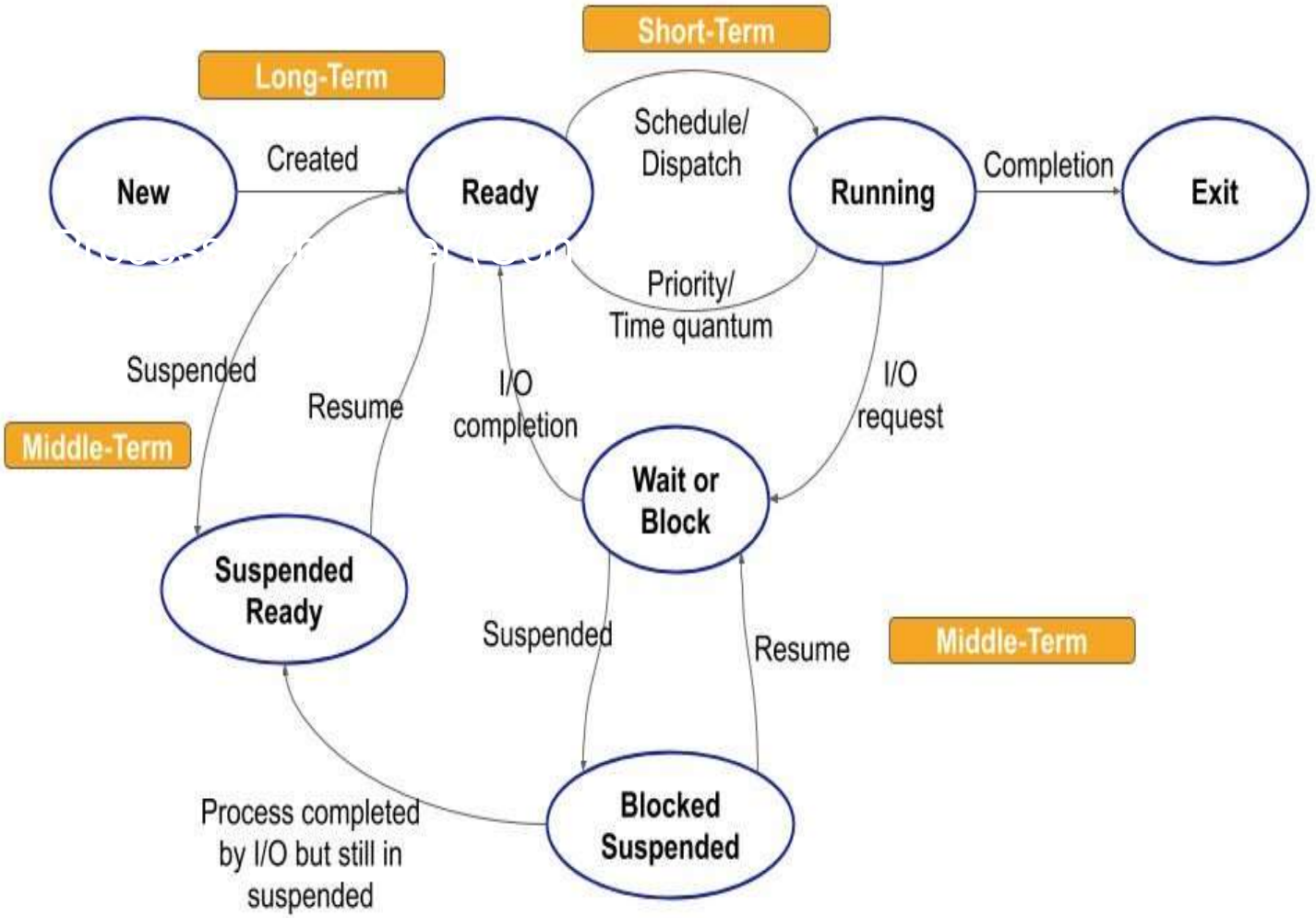
Comparison among Scheduler

S.N	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Process Scheduler (Cont.)

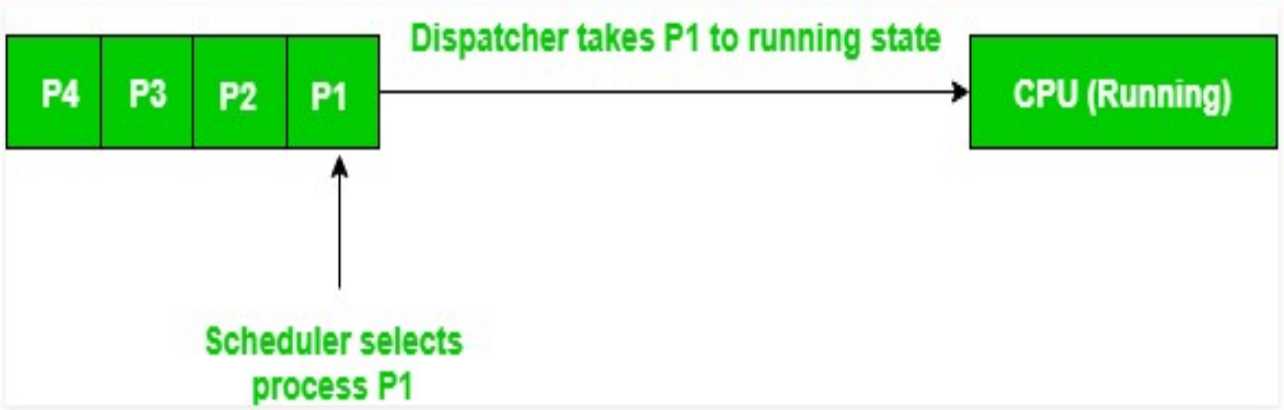


Process Scheduler (Cont.)



Dispatcher

- A dispatcher is a special program which comes into play after the scheduler.
- When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue.



Dispatcher (Cont.)

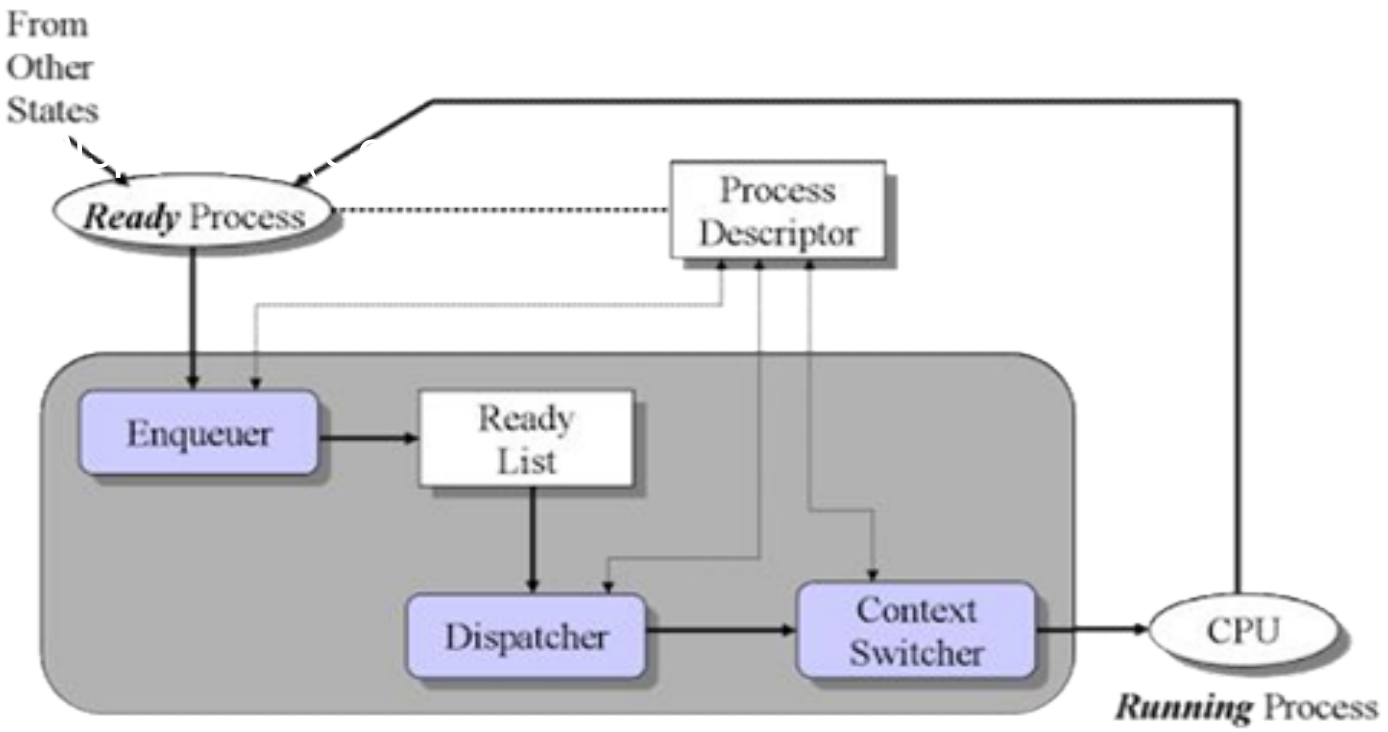
- The dispatcher is the module that gives a process control over the

  CPU after it has been selected by the short-term scheduler.

  This function involves the following:

  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program

Dispatcher (Cont.)



Dispatcher Vs Scheduler

Properties	Dispatcher	Scheduler
Definition	Dispatcher is a module that gives control of CPU to the process selected by short term scheduler	Scheduler is something which selects a process among various processes
Types	There are no different types in dispatcher. It is just a code segment	There are 3 types of scheduler i.e., Long-term, Short-term, Medium-term
Dependency	Working of dispatcher is dependent on scheduler. Means dispatcher have to wait until scheduler selects a process.	Scheduler works independently. It works immediately when needed
Algorithm	Dispatcher has no specific algorithm for its implementation	Scheduler works on various algorithm such as FCFS, SJF, RR etc.
Time Taken	The time taken by dispatcher is called dispatch latency	Time taken by scheduler is usually negligible. Hence we neglect it
Functions	Dispatcher is also responsible for: Context Switching, Switch to user mode, Jumping to proper location when process again restarted	The only work of scheduler is selection of processes

Dispatcher VS Scheduler

Context Switching / Process Switching

- The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system.
- It defines the characteristics of a multitasking operating system in which multiple processes shared the same CPU to perform multiple tasks without the need for additional processors in the system.
- In simple terms, it is like loading and unloading the process from running state to ready state.

Context Switching / Process Switching

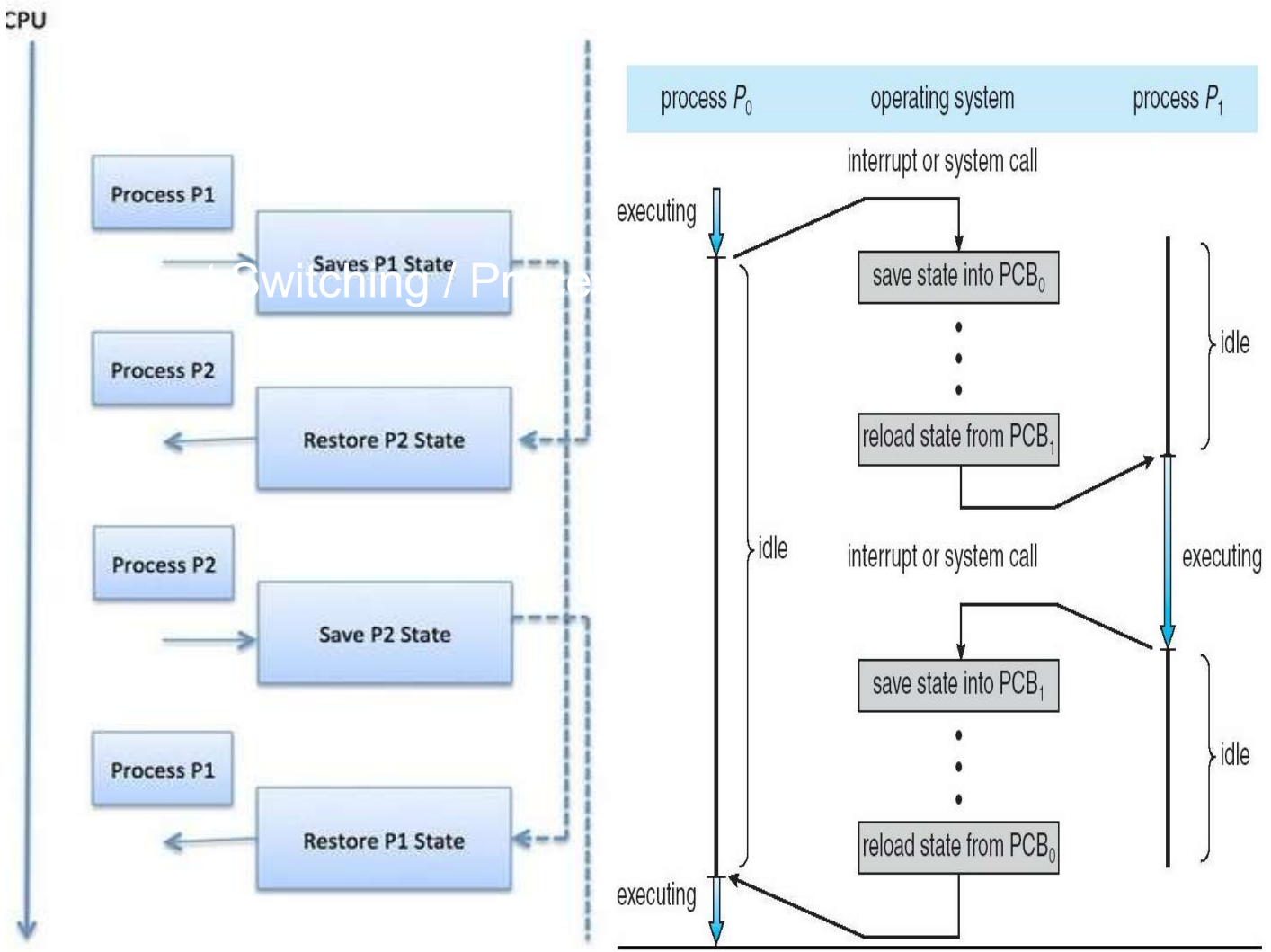
- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a Context Switch.
- When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

Context Switching / Process Switching

When does context switching happen?

- When a high-priority process comes to ready state (i.e. with higher priority than the running process)
- An Interrupt occurs
- User and kernel mode switch (It is not necessary though)
- Pre-emptive CPU scheduling used

Context Switching / Process Switching



CPU Switch From Process to Process

Context Switching / Process Switching

- Context switch time is pure overhead, because the system does no useful work while switching.
- Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions(such as a single instruction to load or store all registers). Typical speeds range from 1 to 1000 microseconds.
- Context Switching has become such a performance bottleneck that programmers are using new structures(threads) to avoid it whenever and wherever possible.

Context Switching / Process Switching

- **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.

Process & Threads In Operating System

Marwadi University

Department of Computer Engineering

Unit - 2
Processes & Threads

Operating System - 01CE0401

Part 2 – Scheduling Criteria and Scheduling Algorithm: Outline

- Scheduling Criteria
  - CPU utilization
  - Throughput
  - Turnaround time
  - Waiting time
  - Response time
- Scheduling Algorithm
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Longest Job First (LJF)
  - Priority Scheduling
  - Round Robin (RR)

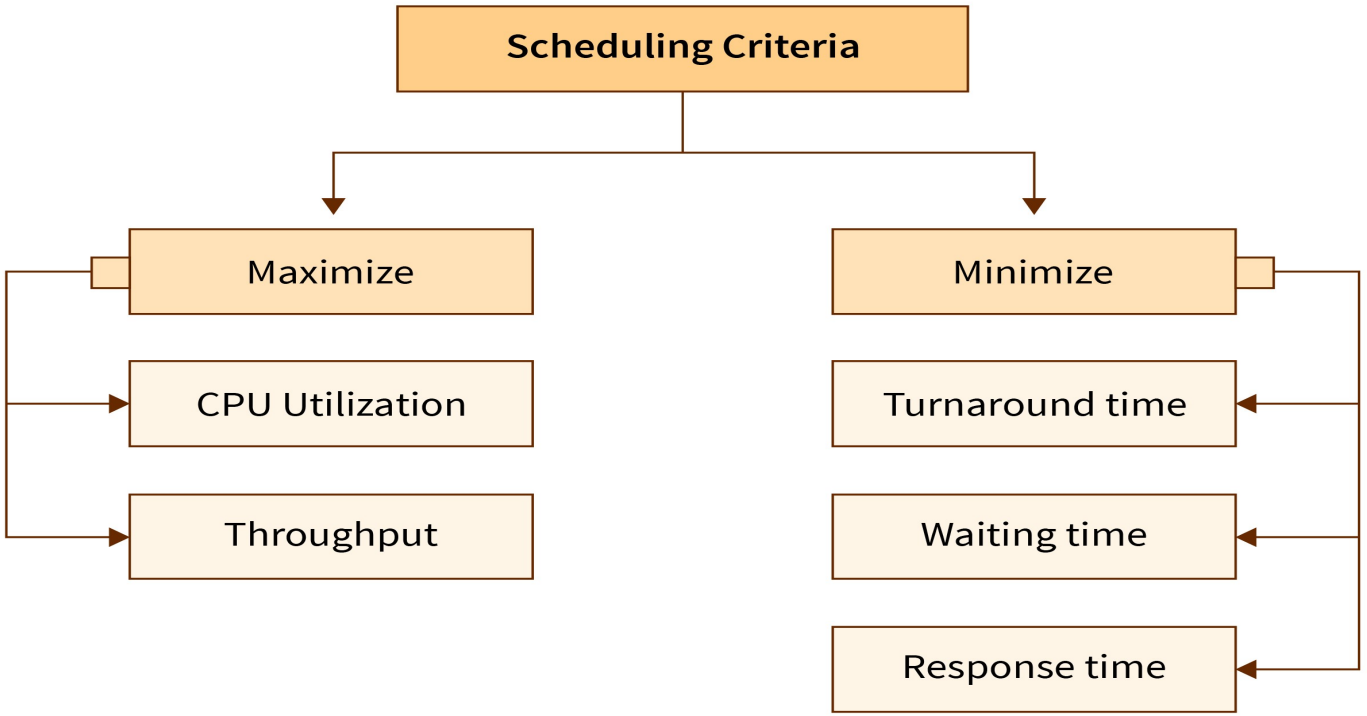
Scheduling Criteria

- **CPU utilization:**
  - It makes sure that the *CPU* is operating at its peak and is busy.
- **Throughput:**
  - A measure of the work done by the CPU is the number of processes being executed and completed per unit of time.
  - It is the number of processes that complete their execution per unit of time.
- **Turnaround time:**
  - It is the amount of time required to execute a specific process.
  - The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time.
  - Turn-around time is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU, and waiting for I/O.
  - The formula to calculate Turn Around Time = Compilation Time – Arrival Time / Burst Time + Waiting Time

Scheduling Criteria (Cont.)

- **Waiting time:**
  - It is the amount of waiting time in the queue.
  - The formula for calculating Waiting Time = Turnaround Time – Burst Time.
- **Response time:**
  - Time retired for generating the first request after submission.
  - The formula to calculate Response Time = CPU Allocation Time(when the CPU was allocated for the first) – Arrival Time

Scheduling Criteria (Cont.)



Preemptive Scheduling VS Non Preemptive Scheduling

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

Scheduling Algorithm

- First Come First Served (FCFS)
  - Non-preemptive.
- Shortest Job First (SJF)
  - Non-preemptive
  - Preemptive (Shortest Remaining Time First (SRTF))
- Longest Job First (LJF)
  - Non-preemptive
  - Preemptive (Longest Remaining Time First (LRTF))
- Priority Scheduling (Small Number = High Priority)
  - Non-preemptive
  - Preemptive
- Round Robin (RR)
  - Preemptive

First Come First Served (FCFS)

Working:

- Jobs are executed on first come, first serve basis.
- Its implementation is based on FIFO queue.
- It is Non-preemptive Scheduling Algorithm

- Advantage:
  - Easy to understand and implement.

- Disadvantage:
  - Poor in performance as average wait time is high.
  - Lower Device Utilization
  - It does not consider the priority or burst time of the processes.
  - It suffers from convoy effect.

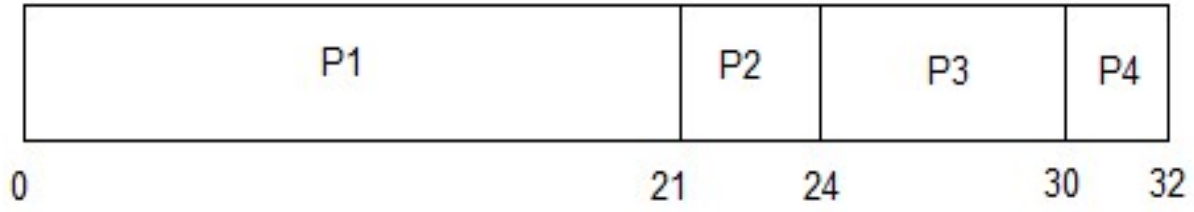
(Convoy Effect - Consider processes with higher burst time arrived before the processes with smaller burst time.Then, smaller processes have to wait for a long time for longer processes to release the CPU.)

First Come First Served (FCFS) Example 1:

- Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time, let's find the average waiting time using the FCFS scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

First Come First Served (FCFS) Example 1 (Cont.) :



Process	Burst Time	Waiting Time	Turn Around Time (Burst Time+Waiting Time)
P1	21	0	21+0 = 21
P2	3	21	3+21 = 24
P3	6	24	6+24 = 30
p4	2	30	2+30 = 32

Average waiting time = (0+21+24+30) / 4 = 18.75

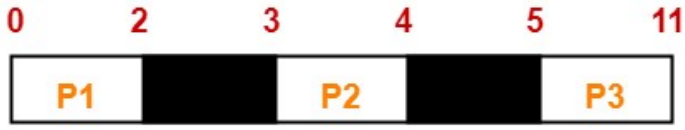
Average turn around time = (21+24+30+32) / 4 = 26.75

First Come First Served (FCFS) Example 2:

- Consider the set of 3 processes whose arrival time and burst time are given below. If the CPU scheduling policy is FCFS, calculate the average waiting time and average turn around time.

Process Id	Arrival time	Burst time
P1	0	2
P2	3	1
P3	5	6

First Come First Served (FCFS) Example 2 (Cont.):



Gantt Chart

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

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	0	2	2	2 – 0 = 2	2 – 2 = 0
P2	3	1	4	4 – 3 = 1	1 – 1 = 0
P3	5	6	11	11- 5 = 6	6 – 6 = 0

Average Turn Around time = (2 + 1 + 6) / 3 = 9 / 3 = 3 unit
Average waiting time = (0 + 0 + 0) / 3 = 0 / 3 = 0 unit

First Come First Served (FCFS) Example 3:

- Consider the set of 5 processes whose arrival time and burst time are given below- If the CPU scheduling policy is FCFS, calculate the average waiting time and average turn around time.

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

First Come First Served (FCFS) Example 3 (Cont.):

- Arrange process in ascending order of Arrival Time

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

Gantt Chart

Here, black box represents the idle time of CPU.

First Come First Served (FCFS) Example 3 (Cont.):

Gantt Chart

Process Id	Arrival time	Burst time	Exit Time	Turn Around time (Exit Time-Arrival Time)	Waiting Time (Turn Around Time – Burst Time)
P1	3	4	7	7-3 = 4	4 – 4 = 0
P2	5	3	13	13-5=8	8 – 3 = 5
P3	0	2	2	2 – 0 = 2	2 – 2 = 0
P4	5	1	14	14 – 5 = 9	9 – 1 = 8
P5	4	3	10	10 – 4 = 6	6 – 3 = 3

Now,
Average Turn Around time = (4 + 8 + 2 + 9 + 6) / 5 = 29 / 5 = 5.8 unit
Average waiting time = (0 + 5 + 0 + 8 + 3) / 5 = 16 / 5 = 3.2 unit

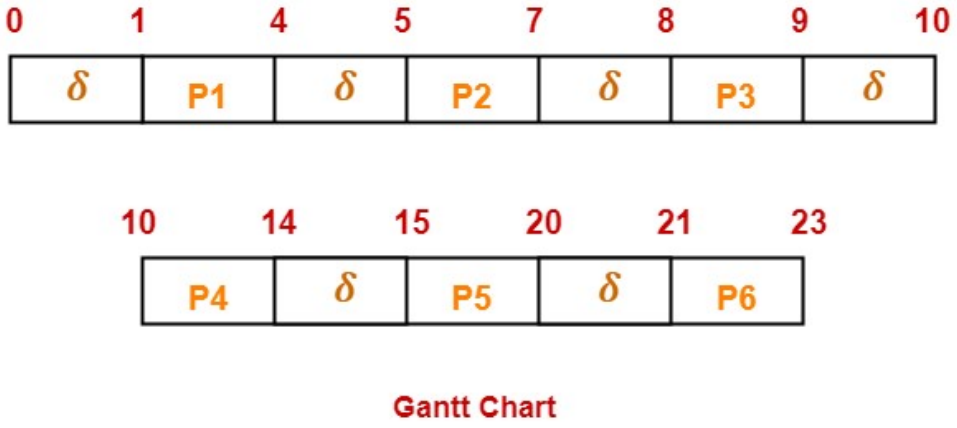
First Come First Served (FCFS) Example 4:

- Consider the set of 6 processes whose arrival time and burst time are given below, If the CPU scheduling policy is FCFS and there is 1 unit of overhead in scheduling the processes, find the efficiency of the algorithm.

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

First Come First Served (FCFS) Example 4 (Cont.):

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



Gantt Chart

Here, δ denotes the context switching overhead.

First Come First Served (FCFS) Example 4 (Cont.):

Now,

Useless time / Wasted time = 6 x δ = 6 x 1 = 6 unit

Total time = 23 unit

Useful time = 23 unit – 6 unit = 17 unit

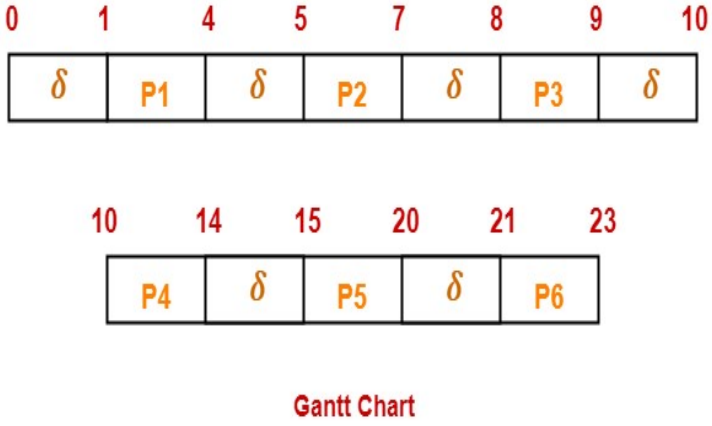
Efficiency (η)

= Useful time / Total

= 17 unit / 23 unit

= 0.7391

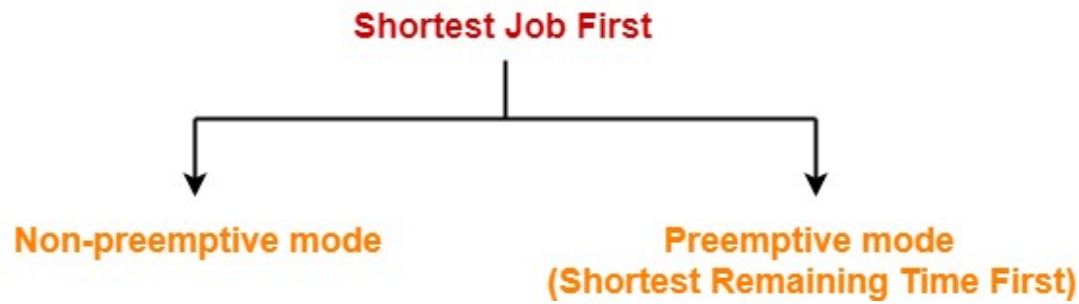
= 73.91%



Gantt Chart

Shortest Job First (SJF)

- Out of all the available processes, CPU is assigned to the process having smallest burst time.



Advantages-

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

Disadvantages-

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

Non-Preemptive Shortest Job First (SJF) Example 1:

- Consider the set of 5 processes whose arrival time and burst time are given below, If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turn around time.

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

Non-Preemptive Shortest Job First (SJF) Example 1 (Cont.):

- Arrange process in ascending order of Arrival Time

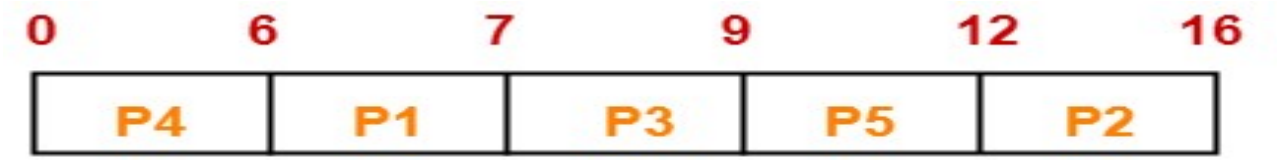
Process Id	Arrival time	Burst time
P_4	0	6
P_2	1	4
P_5	2	3
P_1	3	1
P_3	4	2

Gantt Chart

Non-Preemptive Shortest Job First (SJF) Example 1 (Cont.):

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

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



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

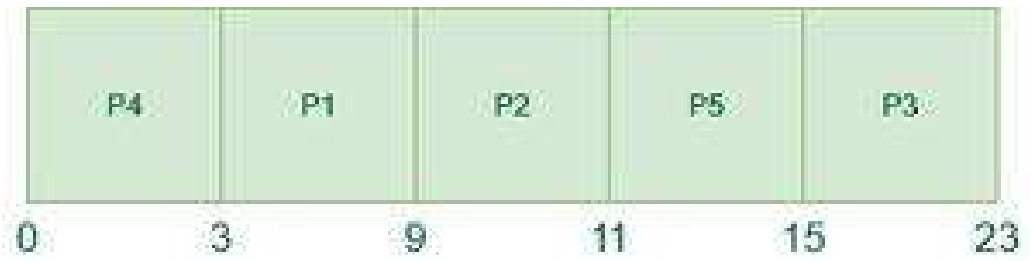
Non-Preemptive Shortest Job First (SJF) Example 2:

- Consider the following table of arrival time and burst time for five processes P1, P2, P3, P4 and P5. calculate the average waiting time.

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

Non-Preemptive Shortest Job First (SJF) Example 2 (Cont.):

Process	Arrival Time	Burst Time
P_4	0 ms	3 ms
P_3	1 ms	8 ms
P_1	2 ms	6 ms
P_5	4 ms	4 ms
P_2	5 ms	2 ms



Non-Preemptive Shortest Job First (SJF) Example 2 (Cont.):

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

Process	Burst Time	Arrival Time	**Exit Time**	**Turn Around time (Exit Time-Arrival Time)**	**Waiting Time (Turn Around Time – Burst Time)**
P1	6 ms	2 ms	9	9-2=7	7-6=1
P2	2 ms	5 ms	11	11-5=6	6-2=4
P3	8 ms	1 ms	23	23-1=22	22-8=14
P4	3 ms	0 ms	3	3-0=3	3-3=0
P5	4 ms	4 ms	15	15-4=11	11-4=7

- ***Average Waiting Time = 0 + 1 + 4 + 7 + 14/5 = 26/5 = 5.2***

Preemptive Shortest Job First (SJF) Example 1:

- Consider the set of 3 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	9
P2	1	4
P3	2	9

- If the CPU scheduling policy is SRTF (Preemptive Shortest Job First - SJF or (Shortest Remaining Time First - SRTF), calculate the average waiting time and average turn around time.

Preemptive Shortest Job First (SJF) Example 1 (Cont.):

Process Id	Arrival time	Burst time
P1	0	9
P2	1	4
P3	2	9

Gantt Chart

Preemptive Shortest Job First (SJF) Example 1 (Cont.):

- Now, we know-
- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Gantt Chart

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	0	9	13	13 – 0 = 13	13 – 9 = 4
P2	1	4	5	5 – 1 = 4	4 – 4 = 0
P3	2	9	22	22- 2 = 20	20 – 9 = 11

Now,

- Average Turn Around time = (13 + 4 + 20) / 3 = 37 / 3 = 12.33 unit
- Average waiting time = (4 + 0 + 11) / 3 = 15 / 3 = 5 unit

Preemptive Shortest Job First (SJF) Example 2:

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

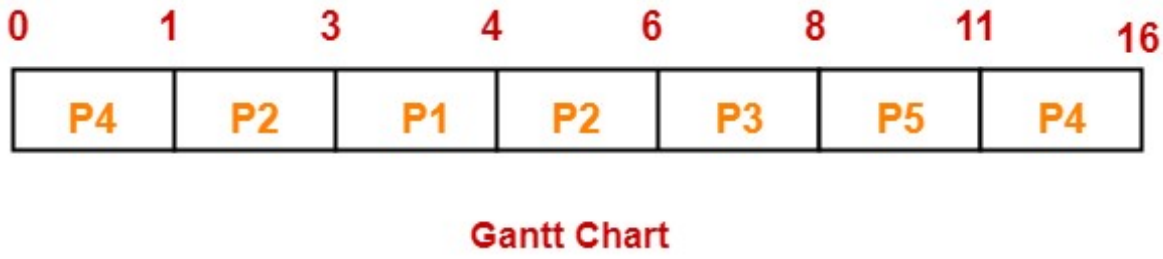
Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

- If the CPU scheduling policy is SJF preemptive, calculate the average waiting time and average turn around time.

Preemptive Shortest Job First (SJF) Example 2 (Cont.):

- Arrange process in ascending order of Arrival Time

Process Id	Arrival time	Burst time
P_4	0	6
P_2	1	4
P_5	2	3
P_1	3	1
P_3	4	2

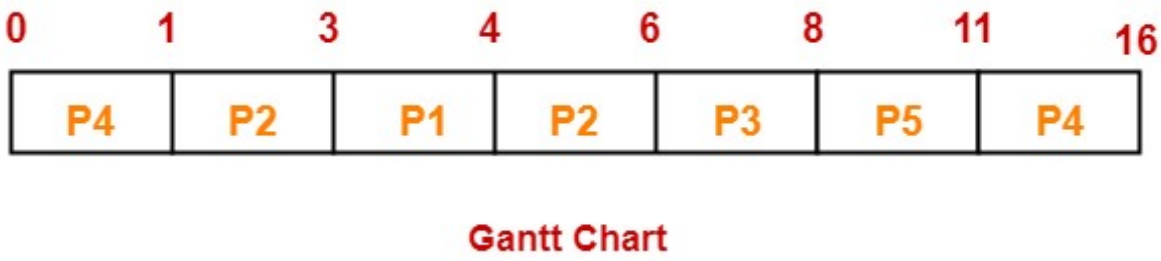


Gantt Chart

Preemptive Shortest Job First (SJF) Example 2 (Cont.):

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

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	3	1	4	4-3 = 1	1-1=0
P2	1	4	6	6-1=5	5-4=1
P3	4	2	8	8-4=4	4-2=2
P4	0	6	16	16-0=16	16-6=10
P5	2	3	11	11-2=9	9-3=6



Gantt Chart

- Now,
- Average Turn Around time = (1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7 unit
- Average waiting time = (0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8 unit

Preemptive Shortest Job First (SJF) Example 3:

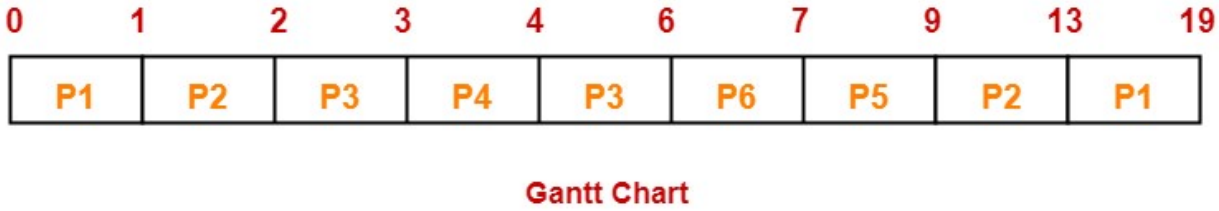
- Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	7
P2	1	5
P3	2	3
P4	3	1
P5	4	2
P6	5	1

- If the CPU scheduling policy is Preemptive shortest remaining time first, calculate the average waiting time and average turn around time.

Preemptive Shortest Job First (SJF) Example 3 (Cont.) :

Process Id	Arrival time	Burst time
P1	0	7
P2	1	5
P3	2	3
P4	3	1
P5	4	2
P6	5	1



Gantt Chart

Preemptive Shortest Job First (SJF) Example 3 (Cont.) :

- Now, we know-
- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	0	7	19	19 – 0 = 19	19 – 7 = 12
P2	1	5	13	13 – 1 = 12	12 – 5 = 7
P3	2	3	6	6 – 2 = 4	4 – 3 = 1
P4	3	1	4	4 – 3 = 1	1 – 1 = 0
P5	4	2	9	9 – 4 = 5	5 – 2 = 3
P6	5	1	7	7 – 5 = 2	2 – 1 = 1

- Average Turn Around time = (19 + 12 + 4 + 1 + 5 + 2) / 6 = 43 / 6 = 7.17 unit
- Average waiting time = (12 + 7 + 1 + 0 + 3 + 1) / 6 = 24 / 6 = 4 unit

Gantt Chart

Preemptive Shortest Job First (SJF) Example 4:

- Consider the set of 4 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	20
P2	15	25
P3	30	10
P4	45	15

- If the CPU scheduling policy is SRTF, calculate the waiting time of process P2.

Preemptive Shortest Job First (SJF) Example 4 (Cont.) :

Gantt Chart

Process Id	Arrival time	Burst time
P1	0	20
P2	15	25
P3	30	10
P4	45	15

Preemptive Shortest Job First (SJF) Example 4 (Cont.) :

- Now, we know-
- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

- Thus,
- Turn Around Time of process P2 = 55 – 15 = 40 unit
- Waiting time of process P2 = 40 – 25 = 15 unit

Priority Scheduling

- In Priority Scheduling,
- Out of all the available processes, CPU is assigned to the process having the highest priority.
- In case of a tie, it is broken by **FCFS Scheduling**.

Priority Scheduling

Advantages-

- It considers the priority of the processes and allows the important processes to run first.
- Priority scheduling in preemptive mode is best suited for real time operating system.

Disadvantages-

- Processes with lesser priority may starve for CPU.
- There is no idea of response time and waiting time.

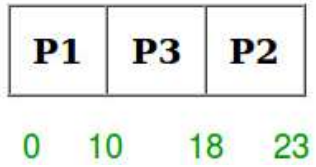
Non-Preemptive Priority Scheduling Example 1:

- Consider the set of 3 processes whose burst time is given below. If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

Process	Burst Time	Priority
P1	10	2
P2	5	0
P3	8	1

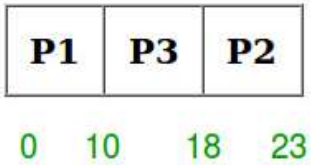
Non-Preemptive Priority Scheduling Example 1 (Cont.):

Process	Burst Time	Priority
P1	10	2
P2	5	0
P3	8	1



Non-Preemptive Priority Scheduling Example 1 (Cont.):

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



- Average waiting time = 9.33333
- Average turn around time = 17

Process Id	Burst time	Priority	Exit time	Turn Around time	Waiting time
P1	10	2	10	10-0=10	10-10=0
P2	5	0	23	23-0=23	23-5=18
P3	8	1	18	18-0=18	18-8=10

Non-Preemptive Priority Scheduling Example 2:

- Consider the set of 5 processes whose arrival time and burst time are given below. If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

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

Non-Preemptive Priority Scheduling Example 2 (Cont.):

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

- (Higher number represents higher priority)

Gantt Chart

Non-Preemptive Priority Scheduling Example 2 (Cont.):

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

Process Id	Arrival time	Burst time	Priority	Exit time	Turn Around time	Waiting time
P1	0	4	2	4	4 – 0 = 4	4 – 4 = 0
P2	1	3	3	15	15 – 1 = 14	14 – 3 = 11
P3	2	1	4	12	12 – 2 = 10	10 – 1 = 9
P4	3	5	5	9	9 – 3 = 6	6 – 5 = 1
P5	4	2	5	11	11 – 4 = 7	7 – 2 = 5

- Average Turn Around time = (4 + 14 + 10 + 6 + 7) / 5 = 41 / 5 = 8.2 unit
- Average waiting time = (0 + 11 + 9 + 1 + 5) / 5 = 26 / 5 = 5.2 unit

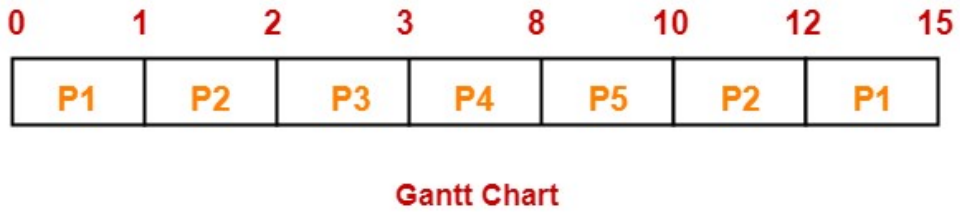
Preemptive Priority Scheduling Example 1:

- Consider the set of 5 processes whose arrival time and burst time are given below. If the CPU scheduling policy is priority preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

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

Preemptive Priority Scheduling Example 1 (Cont.):

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



Gantt Chart

Preemptive Priority Scheduling Example 1 (Cont.):

Gantt Chart

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

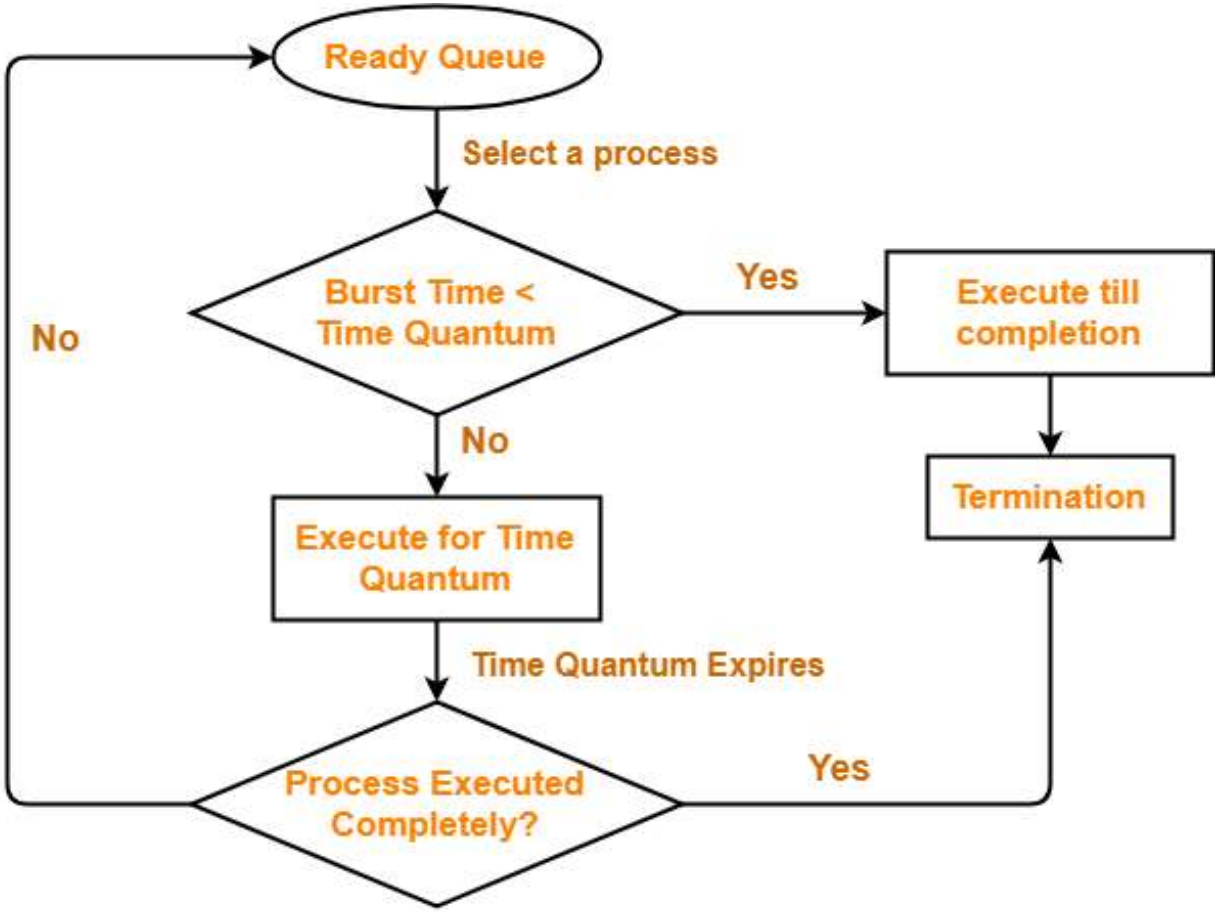
Process Id	Arrival time	Burst time	Priority	Exit time	Turn Around time	Waiting time
P1	0	4	2	15	15 – 0 = 15	15 – 4 = 11
P2	1	3	3	12	12 – 1 = 11	11 – 3 = 8
P3	2	1	4	3	3 – 2 = 1	1 – 1 = 0
P4	3	5	5	8	8 – 3 = 5	5 – 5 = 0
P5	4	2	5	10	10 – 4 = 6	6 – 2 = 4

Average Turn Around time = (15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6 unit
Average waiting time = (11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6 unit

Round Robin Scheduling

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

Round Robin Scheduling (Cont.)



Round Robin Scheduling

Round Robin Scheduling (Cont.)

Advantages-

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

Disadvantages-

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

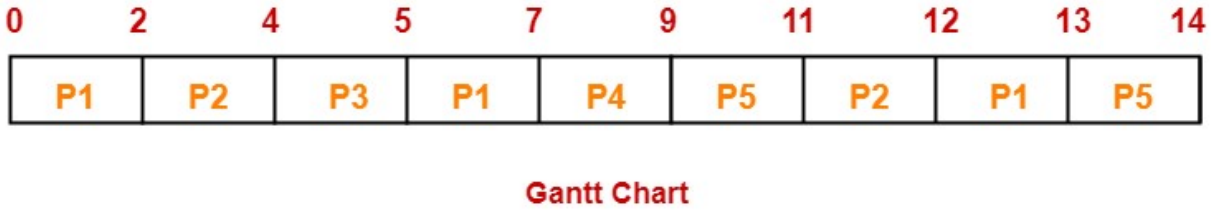
Round Robin Scheduling Example 1:

- Consider the set of 5 processes whose arrival time and burst time are given below. If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time.

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

Round Robin Scheduling Example 1: (Cont.)

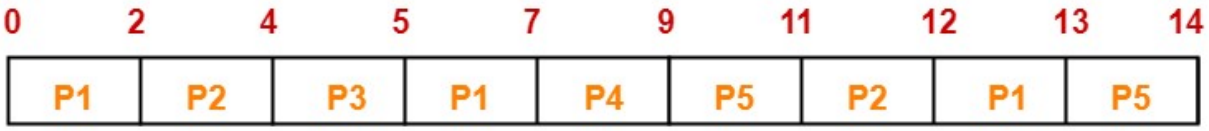
Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3



Gantt Chart

Round Robin Scheduling Example 1: (Cont.)

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



Gantt Chart

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	0	5	13	13 – 0 = 13	13 – 5 = 8
P2	1	3	12	12 – 1 = 11	11 – 3 = 8
P3	2	1	5	5 – 2 = 3	3 – 1 = 2
P4	3	2	9	9 – 3 = 6	6 – 2 = 4
P5	4	3	14	14 – 4 = 10	10 – 3 = 7

Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit
Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

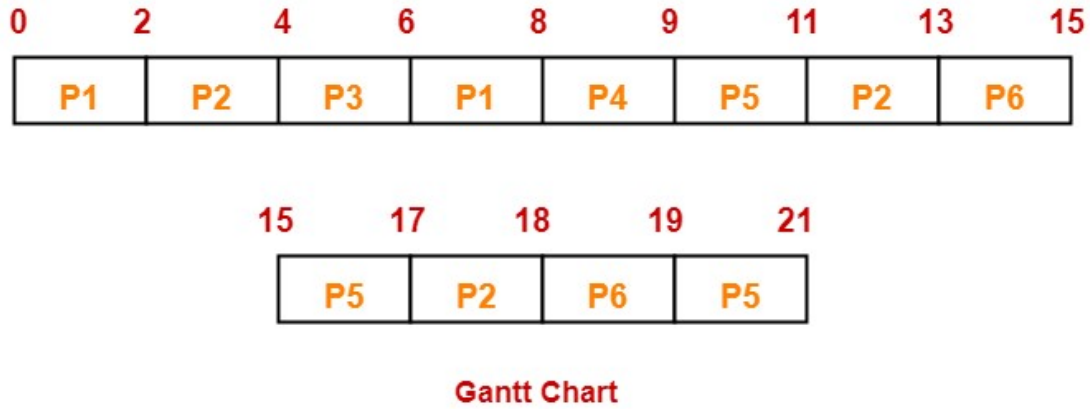
Round Robin Scheduling Example 2:

- Consider the set of 6 processes whose arrival time and burst time are given below. If the CPU scheduling policy is Round Robin with time quantum = 2, calculate the average waiting time and average turn around time.

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

Round Robin Scheduling Example 2 (Cont.):

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



Gantt Chart

Round Robin Scheduling Example 2 (Cont.):

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

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	0	4	8	8 – 0 = 8	8 – 4 = 4
P2	1	5	18	18 – 1 = 17	17 – 5 = 12
P3	2	2	6	6 – 2 = 4	4 – 2 = 2
P4	3	1	9	9 – 3 = 6	6 – 1 = 5
P5	4	6	21	21 – 4 = 17	17 – 6 = 11
P6	6	3	19	19 – 6 = 13	13 – 3 = 10

- Average Turn Around time = (8 + 17 + 4 + 6 + 17 + 13) / 6 = 65 / 6 = 10.84 unit
- Average waiting time = (4 + 12 + 2 + 5 + 11 + 10) / 6 = 44 / 6 = 7.33 unit

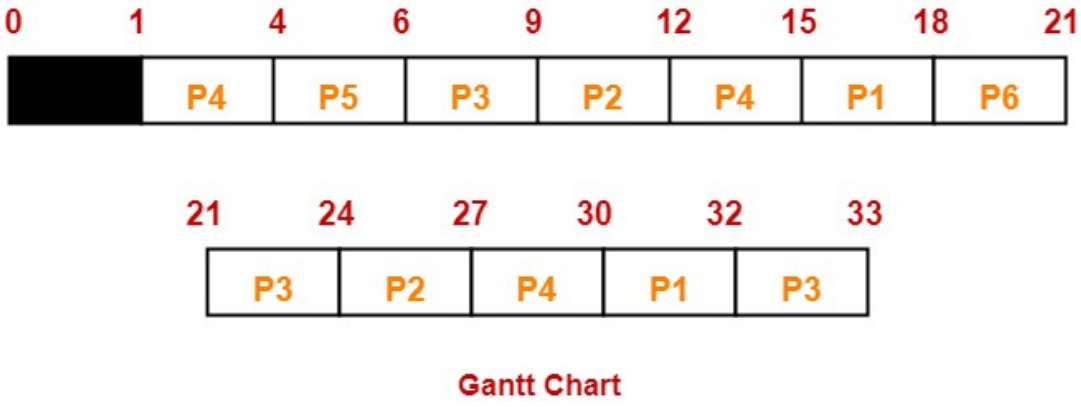
Round Robin Scheduling Example 3 :

- Consider the set of 6 processes whose arrival time and burst time are given below. If the CPU scheduling policy is Round Robin with time quantum = 3, calculate the average waiting time and average turn around time.

Process Id	Arrival time	Burst time
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

Round Robin Scheduling Example 3 (Cont.):

Process Id	Arrival time	Burst time
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3



Gantt Chart

Round Robin Scheduling Example 3 (Cont.):

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

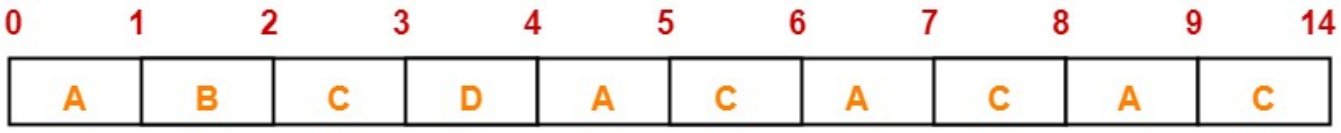
Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	5	5	32	32 – 5 = 27	27 – 5 = 22
P2	4	6	27	27 – 4 = 23	23 – 6 = 17
P3	3	7	33	33 – 3 = 30	30 – 7 = 23
P4	1	9	30	30 – 1 = 29	29 – 9 = 20
P5	2	2	6	6 – 2 = 4	4 – 2 = 2
P6	6	3	21	21 – 6 = 15	15 – 3 = 12

- Average Turn Around time = (27 + 23 + 30 + 29 + 4 + 15) / 6 = 128 / 6 = 21.33 unit
- Average waiting time = (22 + 17 + 23 + 20 + 2 + 12) / 6 = 96 / 6 = 16 unit

Round Robin Scheduling Example 4:

- Four jobs to be executed on a single processor system arrive at time 0 in the order A, B, C, D. Their burst CPU time requirements are 4, 1, 8, 1 time units respectively. The completion time of A under round robin scheduling with time slice of one time unit is-
  - 10
  - 4
  - 8
  - 9

Round Robin Scheduling Example 4 (Cont.):

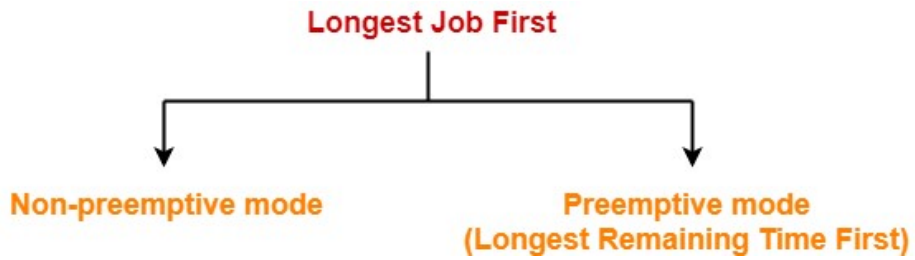


Gantt Chart

Process Id	Arrival time	Burst time
A	0	4
B	0	1
C	0	8
D	0	1

Longest Job First Scheduling

- Out of all the available processes, CPU is assigned to the process having largest burst time.
- In case of a tie, it is broken by FCFS Scheduling.



Longest Job First Scheduling (Cont.)

Advantages-

- No process can complete until the longest job also reaches its completion.
- All the processes approximately finishes at the same time.

Disadvantages-

- The waiting time is high.
- Processes with smaller burst time may starve for CPU.

Non-Preemptive Longest Job First Scheduling Example 1:

- Consider the set of 5 processes whose arrival time and burst time are given below. If the CPU scheduling policy is LJF non-preemptive, calculate the average waiting time and average turn around time.

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

Non-Preemptive Longest Job First Scheduling Example 1 (Cont.):

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

Gantt Chart

Non-Preemptive Longest Job First Scheduling Example 1 (Cont.):

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

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	0	3	3	3 – 0 = 3	3 – 3 = 0
P2	1	2	20	20 – 1 = 19	19 – 2 = 17
P3	2	4	18	18 – 2 = 16	16 – 4 = 12
P4	3	5	8	8 – 3 = 5	5 – 5 = 0
P5	4	6	14	14 – 4 = 10	10 – 6 = 4

Gantt Chart

- Average Turn Around time = (3 + 19 + 16 + 5 + 10) / 5 = 53 / 5 = 10.6 unit
- Average waiting time = (0 + 17 + 12 + 0 + 4) / 5 = 33 / 5 = 6.6 unit

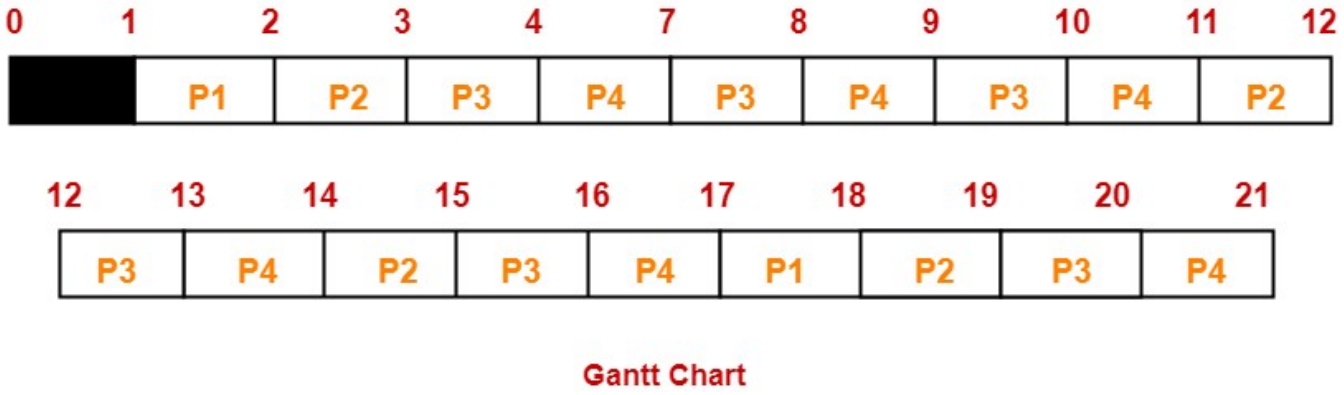
Preemptive Longest Job First Scheduling Example 1:

- Consider the set of 4 processes whose arrival time and burst time are given below. If the CPU scheduling policy is LJF preemptive, calculate the average waiting time and average turn around time.

Process Id	Arrival time	Burst time
P1	1	2
P2	2	4
P3	3	6
P4	4	8

Preemptive Longest Job First Scheduling Example 1 (Cont.):

Process Id	Arrival time	Burst time
P1	1	2
P2	2	4
P3	3	6
P4	4	8



Gantt Chart

Preemptive Longest Job First Scheduling Example 1 (Cont.):

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

Process Id	Arrival time	Burst time	Exit time	Turn Around time	Waiting time
P1	1	2	18	18 – 1 = 17	17 – 2 = 15
P2	2	4	19	19 – 2 = 17	17 – 4 = 13
P3	3	6	20	20 – 3 = 17	17 – 6 = 11
P4	4	8	21	21 – 4 = 17	17 – 8 = 9

- Average Turn Around time = (17 + 17 + 17 + 17) / 4 = 68 / 4 = 17 unit
- Average waiting time = (15 + 13 + 11 + 9) / 4 = 48 / 4 = 12 unit

Preemptive Longest Job First Scheduling Example 2:

- Consider three processes (process id 0, 1, 2 respectively) with compute time bursts 2, 4 and 8 time units. All processes arrive at time zero. Consider the longest remaining time first (LRTF) scheduling algorithm. In LRTF, ties are broken by giving priority to the process with the lowest process id. The average turn around time is-
  - 13 unit
  - 14 unit
  - 15 unit
  - 16 unit

Preemptive Longest Job First Scheduling Example 2 (Cont.):

- We have the set of 3 processes whose arrival time and burst time are given below-

Gantt Chart

Process Id	Arrival time	Burst time	Exit time	Turn Around time
P1	0	2	12	12 – 0 = 12
P2	0	4	13	13 – 0 = 13
P3	0	8	24	14 – 0 = 14

- Now,
- Average Turn Around time = (12 + 13 + 14) / 3 = 39 / 3 = 13 unit
- Thus, Option (A) is correct.

Process & Threads

Unit #2

Marwadi University

Department of Computer Engineering

Operating System

Sem 4

01CE1401

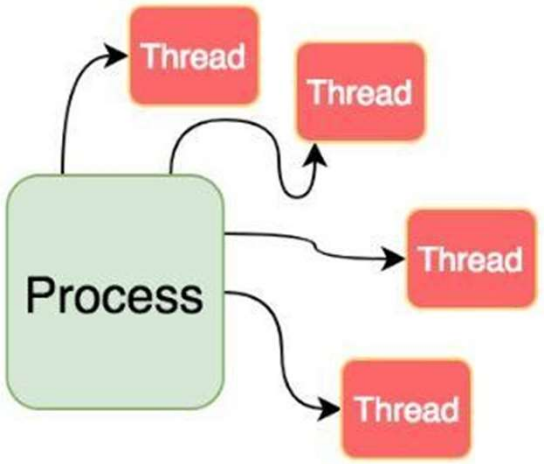
4 Credits

Part 3 – Thread: Outline

- Thread
- Types of Thread
- Concept of multithreading
- Multithreading Models
- Multithreading Issues
- Thread Life Cycle
- Thread Control Block

What is Thread?

- A thread is a path of execution within a process.
- A process can contain multiple threads.
- A thread is also known as lightweight process.



Why Multithreading?

- The idea is to achieve parallelism by dividing a process into multiple threads.
- For example, in a browser, multiple tabs can be different threads.
- MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Comparison Basis	Thread	Process
Weight	A thread is a lightweight.	A Process is a heavy weight.
Context switching	Threads require less time for context switching	Processes require more time for context switching/
Termination Time	Threads require less time for termination.	Processes require more time for termination.
Blocked	If a user level thread gets blocked, all of its peer threads also get blocked.	If a process gets blocked, remaining processes can continue execution.
Code and data sharing	A thread shares the data segments, and files etc. with its peer threads.	Processes have independent data and code segments.
Dependent or independent	Threads are dependent on process.	Individual processes are independent.
Memory Sharing	A thread may share some memory with its peer threads.	Processes don't share memory.
Resource Consumption	Threads are Lightweight so need less resources..	Processes are Heavy weight so need more resources..
Creation Time	Threads require less time for creation.	Processes require more time for creation.
Communication	Faster	Slower

Similarities of thread and process:

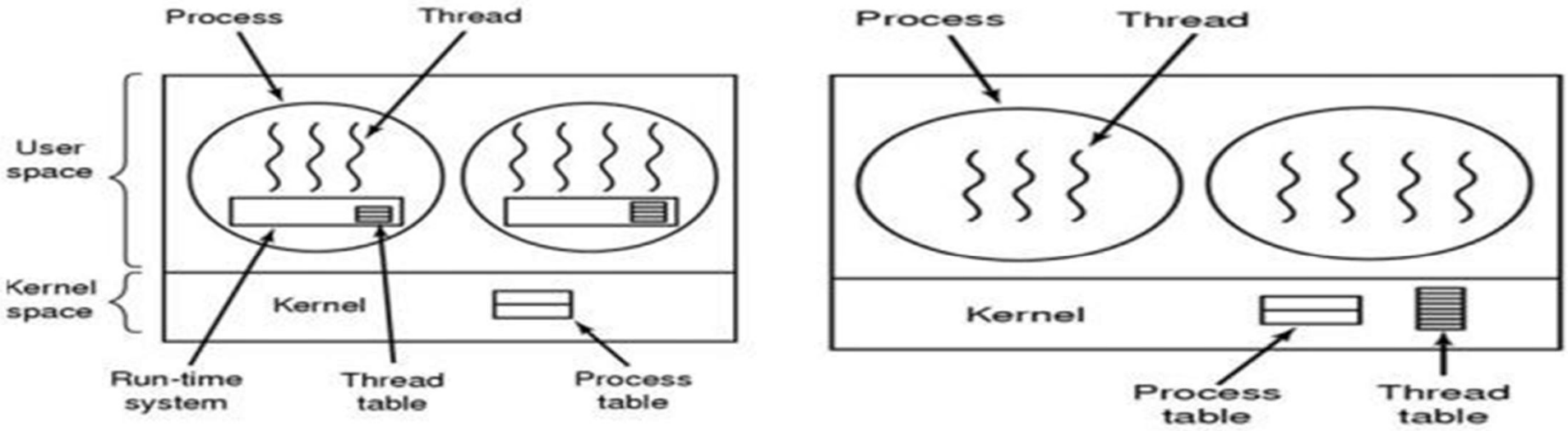
- Both can share CPU
- Both can create a child
- If one block then other can run

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.
- Increase processing speed.
- Don’t need for inter-process communication

Types of Thread

- There are two types of Threads
  - User-level Threads
    - User managed threads
  - Kernel-level Threads
    - Operating System managed threads acting on kernel, an operating system core



(a) A user-level threads package. (b) A threads package managed by the kernel.

User-Level Thread VS Kernel-Level Thread

User-Level Thread	Kernel-Level Thread
The thread which is created with the help of a user is known as User-level thread.	The threads which are created with the help of an operating system are known as Kernel-Level thread.
In the User-level thread, the time of context switching is less.	In the kernel-level threads, the time of context switching is more.
The operating system cannot recognize the User-level threads	The operating system can recognize the kernel-level threads.
In User-level threads, for context switching, there is no need for hardware support.	In kernel-level threads for context switching, hardware support is required.
Examples of User-level threads are POSIX threads, java threads.	An example of a kernel-level thread is window Solaris.
The User-Level threads are implemented in an easy way.	The Kernel-Level threads are difficult to Implement.
In the User-Level-Thread, if any of the thread performs the blocking operations, then due to blocking operation, the whole process will be blocked.	In the kernel-level thread, when one kernel-level thread performs blocking operation, then due to of blocking operations, other thread will not affect the other thread so, the remaining thread continues its execution.

User-Level Threads:

- User-level threads are those types of thread that are created in the user-level library.
- These threads are not implemented with the help of the system calls.
- If we want to switch thread, then there is no need to interrupt the kernel and call operating system.
- In the user-level thread, there is no need for the kernel to know about the user-level thread and handle threads, if a process containing a single thread.
- Examples of a User-level threads are POSIX threads, java threads.

User-Level Threads (Cont.):

Advantages of User-Level Threads

- User-level thread is easy to create because in this, there is no involvement of kernel.
- It is fast because there is no need for operating system calls.
- It is easy to run a user-level thread in any operating system.

Disadvantages of User-Level Threads

- If in one thread page Fault is caused, then the whole process is blocked.
- In the User-level thread, there is no coordination between threads and the kernel.

Kernel-Level Threads

- Kernel-level threads are those threads that are directly managed by the operating system, and the kernel does the management of the thread.
- In this, rather creating a thread table for each process, kernel has its own table that is a master table that is used to keep track of all the threads in the system.
- It is also keeping track of the classical process table to maintain track
- of the processes.
- The Operating system kernel offers a system call to handle and implement a thread.
- Example of a kernel-level thread is window Solaris.

Kernel-Level Threads (Cont.)

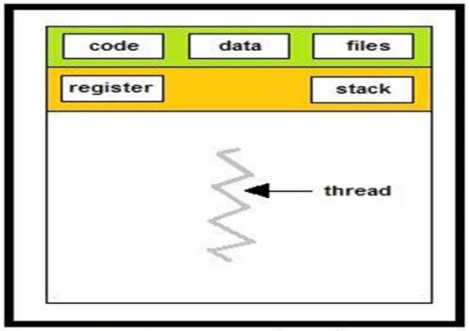
Advantages of Kernel-Level Thread

- Kernel-level thread is useful for applications that are blocked frequently.
- Because the kernel has complete knowledge of the system's threads, the scheduler may decide to give more time to those processes which have a different number of threads.

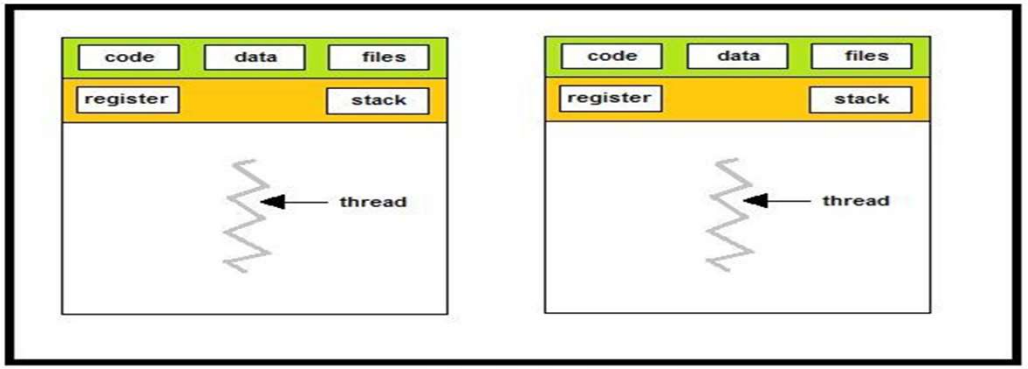
Disadvantages of Kernel-Level Thread

- Kernel-level threads are slow
- Kernel-level threads are not efficient.
- Kernel-level thread is overhead because it needed a thread control block.

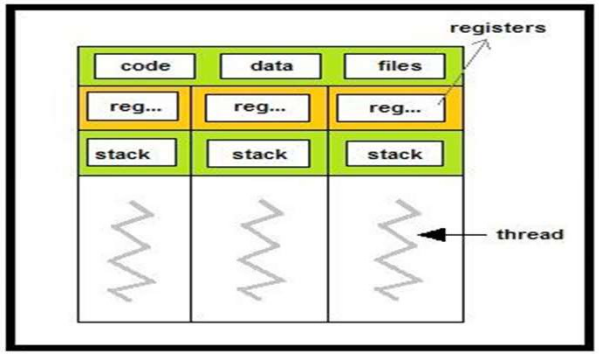
Concept of Multithreading



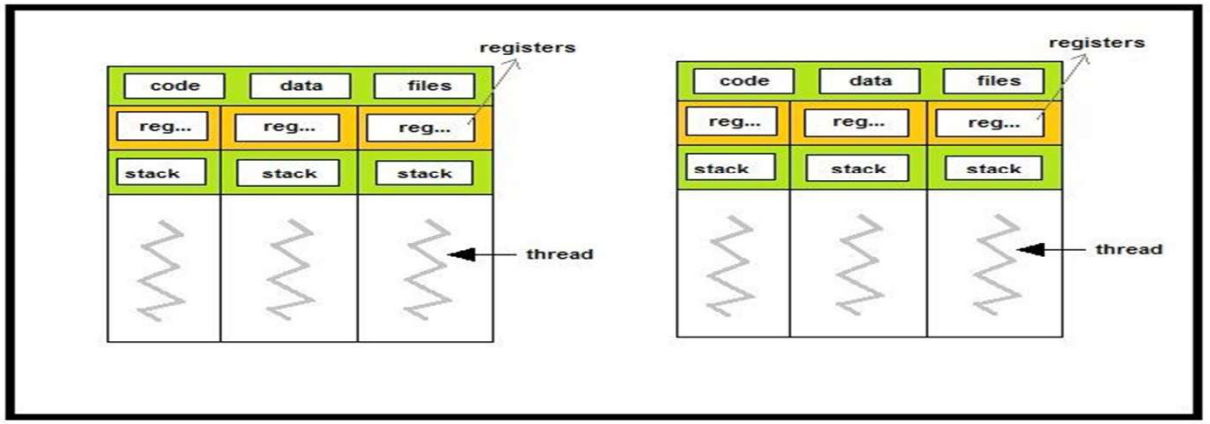
One Process One Thread



Multi Process One Thread Per Process



One Process Multiple Thread
(Multi-Threading)



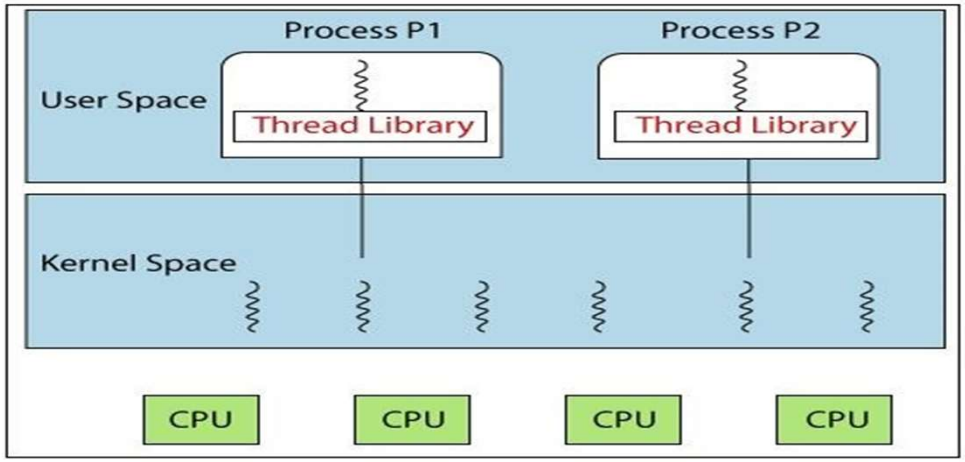
Multi Process and Multiple Thread Per Process
(Multitasking and Multithreading)

Concept of Multithreading (Cont.)

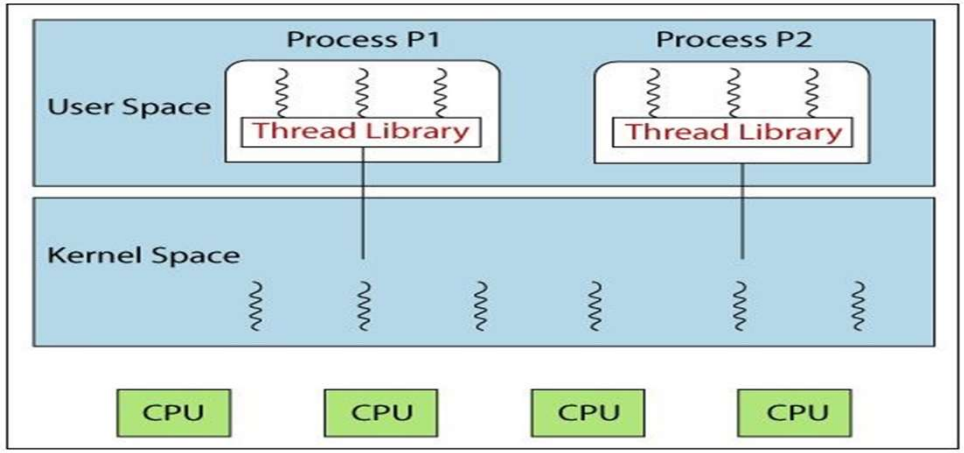
- One Process - One Thread
  - Example: MS-DOC
- Multi Process - One Thread per Process (Multi-Tasking)
  - Example: Unix
- One Process - Multiple Thread (Multi-Threading)
  - Example: Java
- Many Process - Multiple Threads (Multi- Tasking with Multi-Threading)
  - Example: Windows, MaC

Multithreading Models

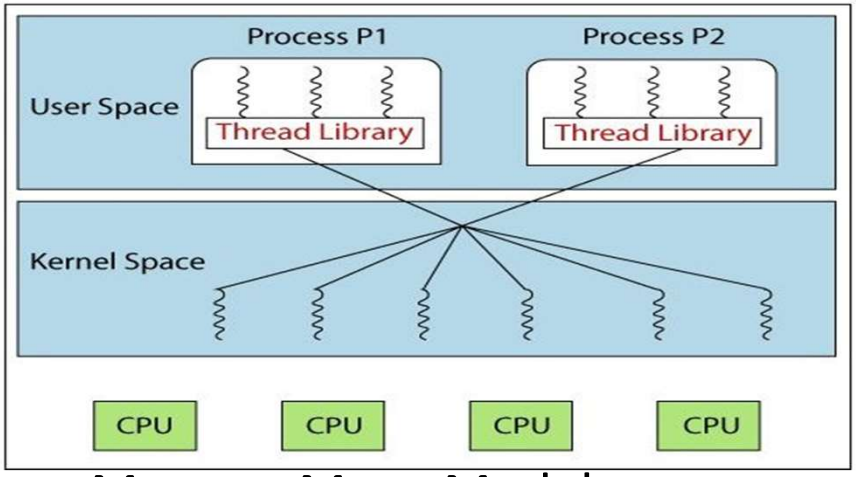
- Multithreading meansusing both types of threads User-Level as well as Kernel-Level Thread.
- An Example of multithreading is Solaris.
- Multithreading Models can be classified into three types:
  - One to One Model
  - Many to One Model
  - Many to Many Model



One to One Model



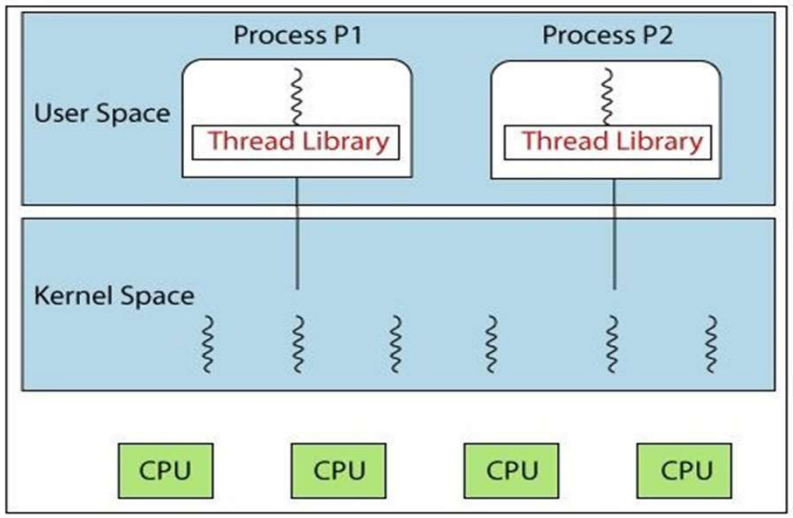
Many to One Model



Many to Many Model

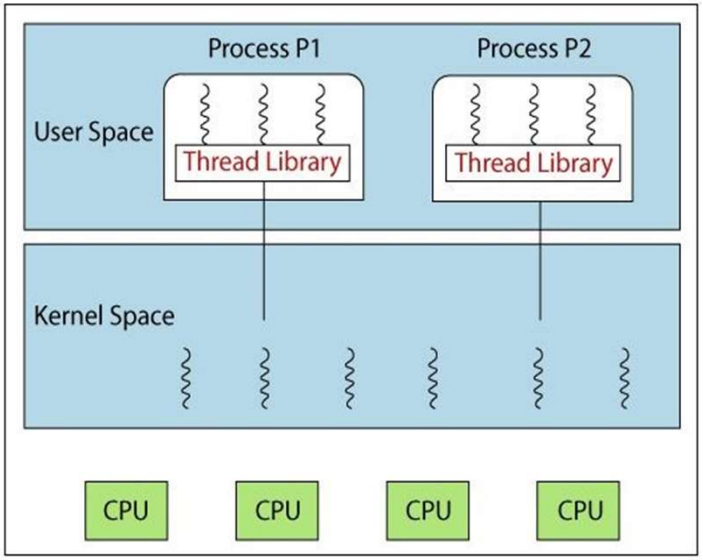
One to One Model

- One to One model is a Multithreading Model in which One to One relationship exists between the kernel level and User Level Thread.
- One to One Models can run multiple threads on multiple processors.
- In one-to-one Model, to create a User-Level thread we also need kernel thread. This is a problem in one to one model.



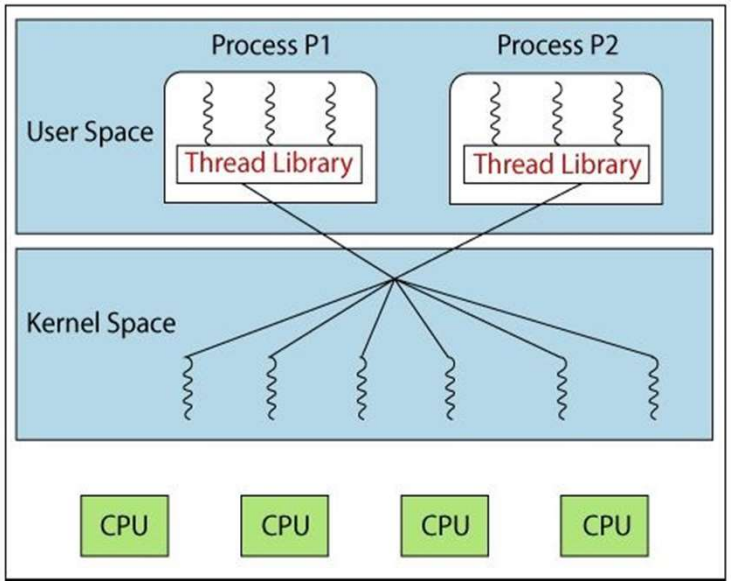
Many to One Model

- In Many to One Multithreading model, many user threads are mapped to a single kernel.
- If due to user thread, the system call is blocked, then the whole process is blocked.
- In this, the task of thread management is managed by the thread library in the convenient user space.



Many to Many Model

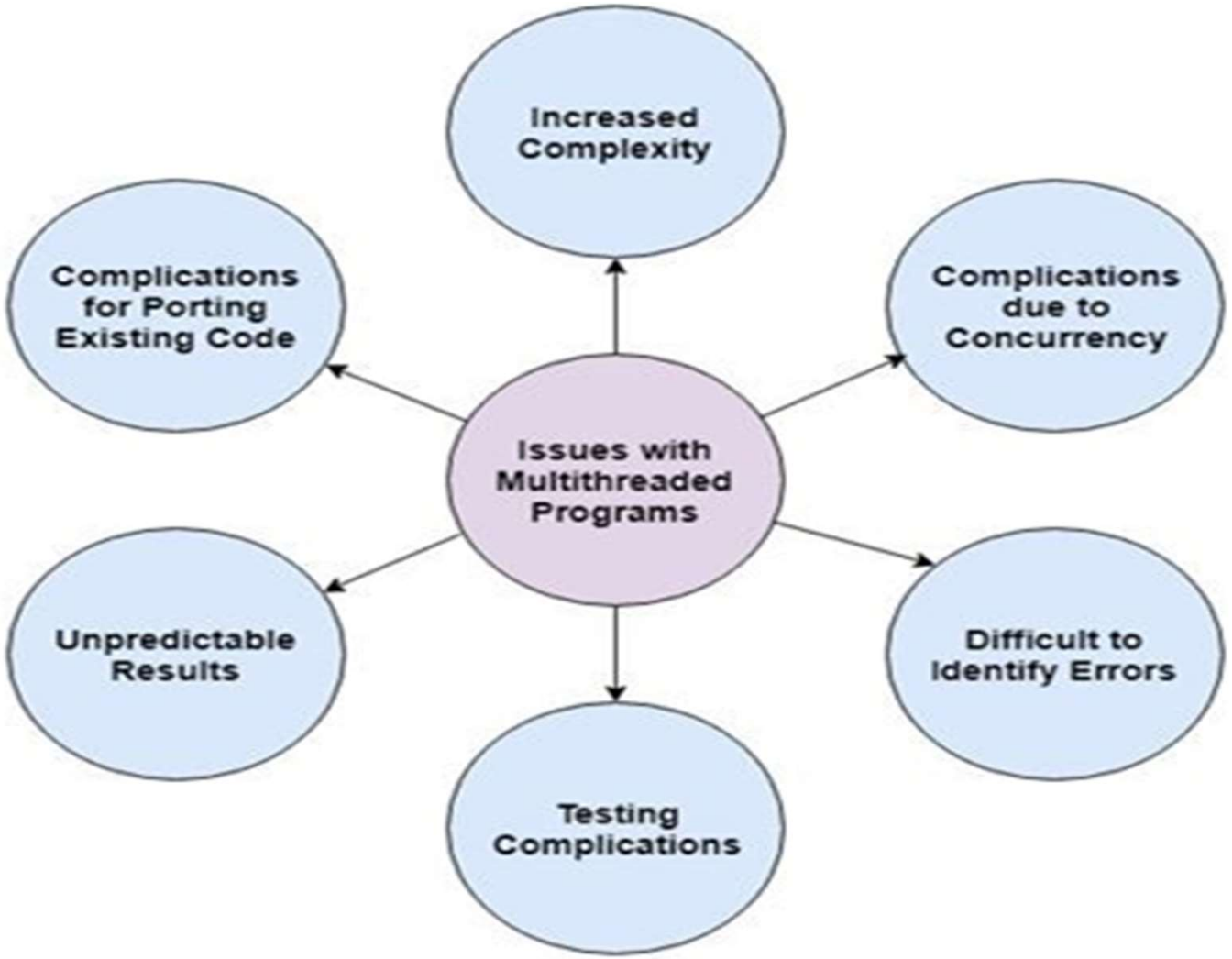
- In Many to Many Multithreading models, the user can create multiple threads, and if the kernel system call is blocked, then the entire process does not block.
- This model multiplexes multiple user-level threads onto smaller or an equal number of kernel-level threads.
- In this, processes can be divided across multiple processors.



Advantages of Multithreading

- Context switching is easy in Multithreading.
- Responsiveness
- Resource sharingis easy, so it offers better utilization of resources.
- Implementing and managing threads is quite simple.

Multithreading Issues



Multithreading Issues (Cont.)

- **Increased Complexity** − Multithreaded processes are quite complicated. Coding for these can only be handled by expert programmers.
- **Complications due to Concurrency** − It is difficult to handle concurrency in multithreaded processes. This may lead to complications and future problems.
- **Difficult to Identify Errors**− Identification and correction of errors is much more difficult in multithreaded processes as compared to single threaded processes.

Multithreading Issues (Cont.)

- **Testing Complications**– Testing is a complicated process is multithreaded programs as compared to single threaded programs. This is because defects can be timing related and not easy to identify.
- **Unpredictable results**– Multithreaded programs can sometimes lead to unpredictable results as they are essentially multiple parts of a program that are running at the same time.
- **Complications for Porting Existing Code** – A lot of testing is required for porting existing code in multithreading. Static variables need to be removed and any code or function calls that are not thread safe need to be replaced.

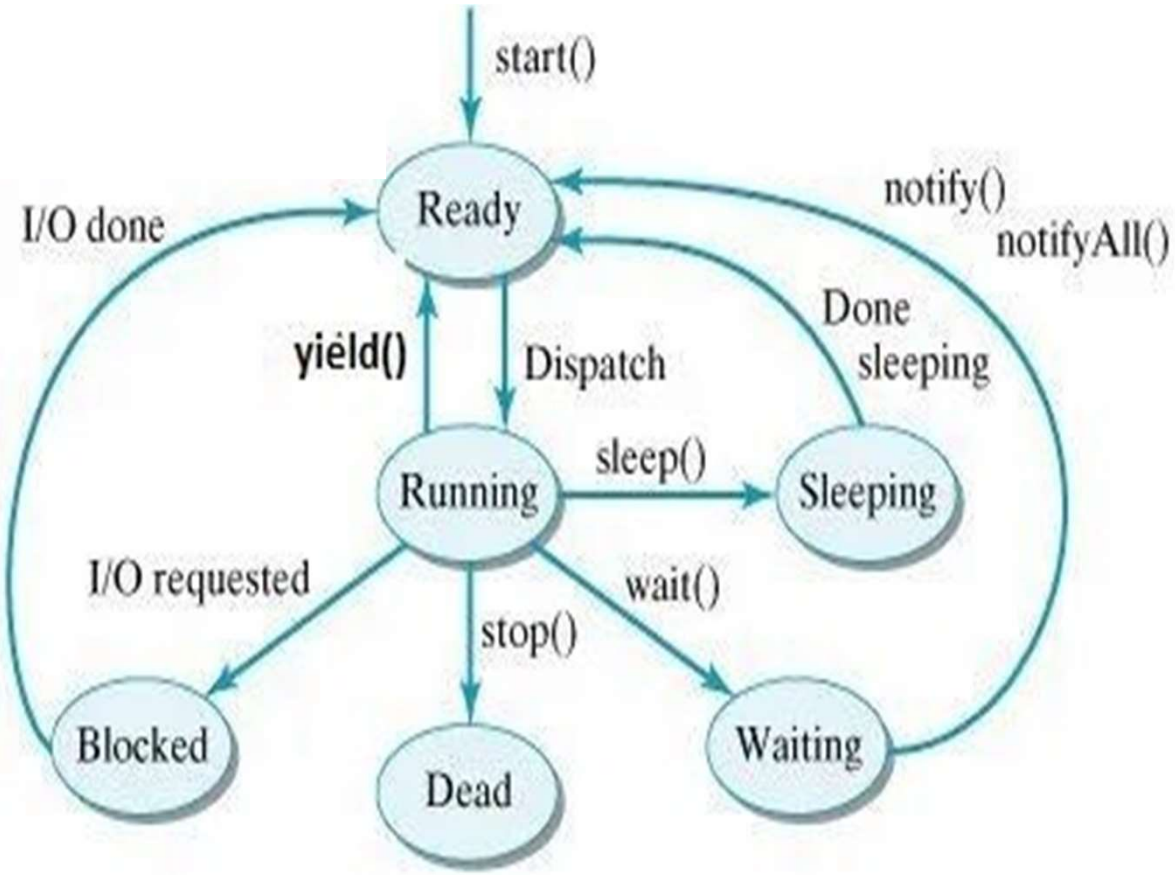
Multithreading Issues (Cont.)

- **Thread Cancellation**
  - Thread cancellation means terminating a thread before it has finished working.
  - There can be two approaches for this
    - Asynchronous cancellation - cancels the thread immediately.
    - Deferred cancellation - sets a flag indicating the thread should cancel itself when it is convenient. It is then up to the cancelled thread to check this flag periodically and exit nicely when it sees the flag set.

Multithreading Issues (Cont.)

- **Signal Handling**
  - Signals are used in UNIX systems to notify a process that a particular event has occurred.
  - Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all or a single thread.
- **fork() System Call**
  - fork() is a system call executed in the kernel through which a process creates a copy of itself.
  - Now the problem in the Multithreaded process is, if one thread forks, will the entire process be copied or not?
- **Security Issues**
  - Yes, there can be security issues because of the extensive sharing of resources between multiple threads.

Thread Life Cycle



Thread Life Cycle (Cont.)

1. Born State: A thread that has just created.
2. Ready State: The thread is waiting for the processor (CPU).
3. Running: The System assigns the processor to the thread means that the thread is being executed.
4. Blocked State: The thread is waiting for an event to occur or waiting for an I/O device.
5. Sleep: A sleeping thread becomes ready after the designated sleep time expires.
6. Dead: The execution of the thread is finished.

Thread Control Block (TCB)

- Very similar to Process Control Blocks (PCBs) which represents processes, Thread Control Blocks TCBs represents threads generated in the system.

Thread ID
Thread state
CPU information : Program counter Register contents
Thread priority
Pointer to process that created this thread
Pointer(s) to other thread(s) that were created by this thread

Thread Control Block (TCB) (Cont.)

- **Thread ID:** It is a unique identifier assigned by the Operating System to the thread when it is being created.
- **Thread states:** These are the states of the thread which changes as the thread progresses through the system
- **CPU information:** It includes everything that the OS needs to know about, such as how far the thread has progressed and what data is being used.
- **Thread Priority:** It indicates the weight (or priority) of the thread over other threads which helps the thread scheduler to determine which thread should be selected next from the READY queue.
- **A pointer which points to the process** which triggered the creation of this thread.
- **A pointer which points to the thread**(s) created by this thread.