Unit-2
Process and Threads Management

Outline

- Process concept
- Process states
- Process state transitions
- Process Control Block (PCB)
- Context switching
- Threads
- Comparison between process and thread
- Benefits/Advantages of threads
- Types of threads
- Multi Threading Models
- Pthread function calls
- System calls

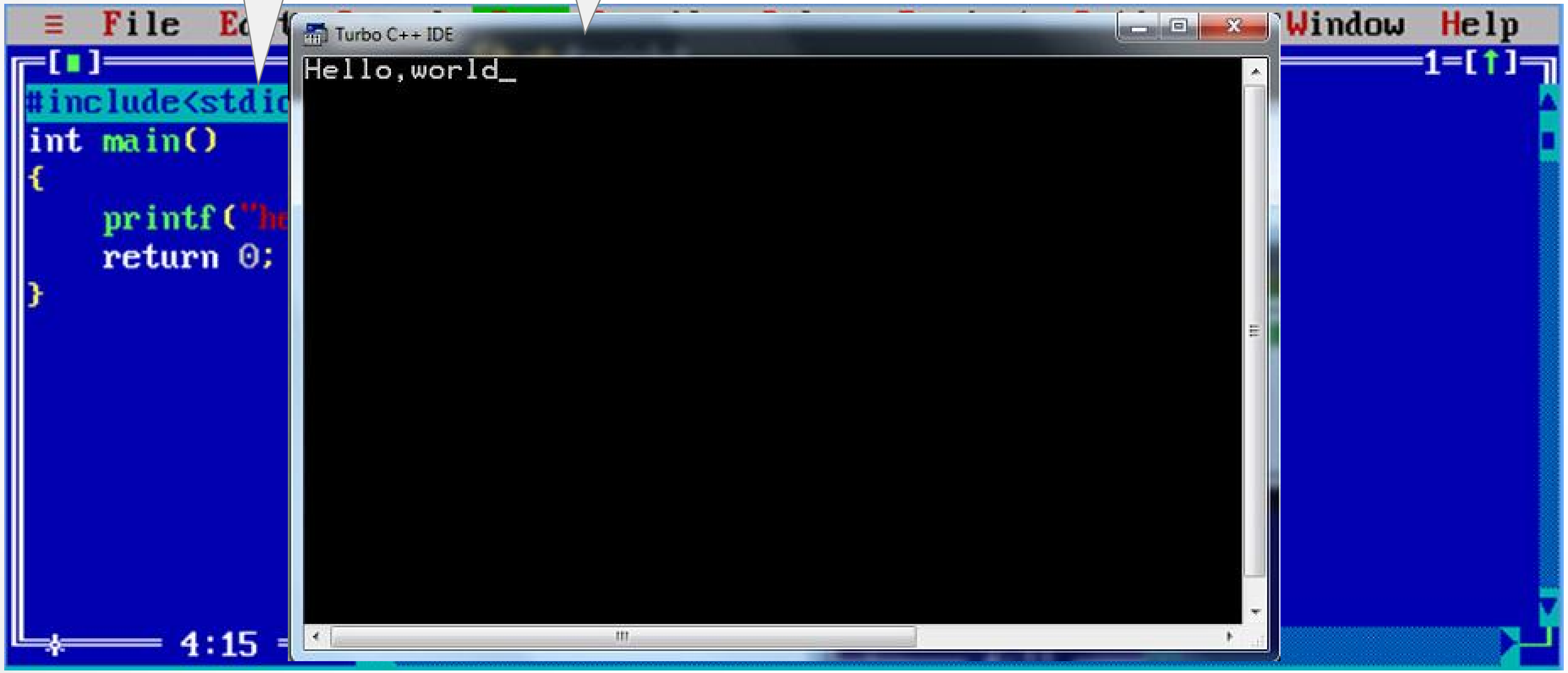
Process concept

Section - 1

What is Process?

Program

Process

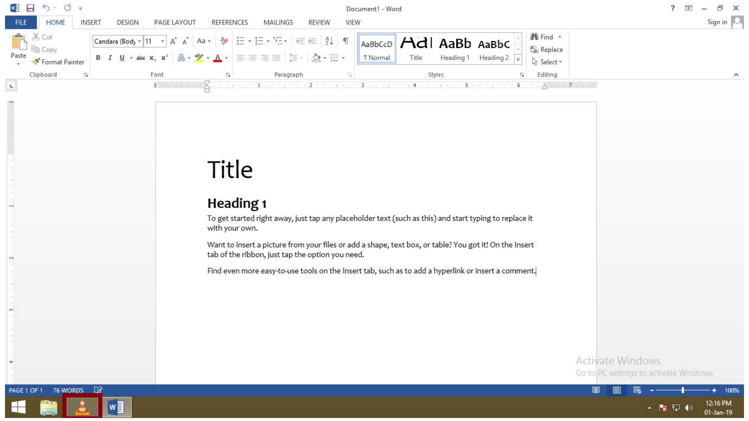


What is Process?

- Process is a **program under execution**.
- Process is an **abstraction of a running program**.
- Process is an **instance of an executing program**, including the current values of the program counter, registers & variables.
- **Each process has its own virtual CPU**.

Multiprogramming

- The real **CPU switches** back and forth from process to process.
- This **rapid switching back and forth is called multiprogramming**.
- The **number of processes loaded simultaneously in memory is called degree of multiprogramming**.



Multiprogramming execution

Memory

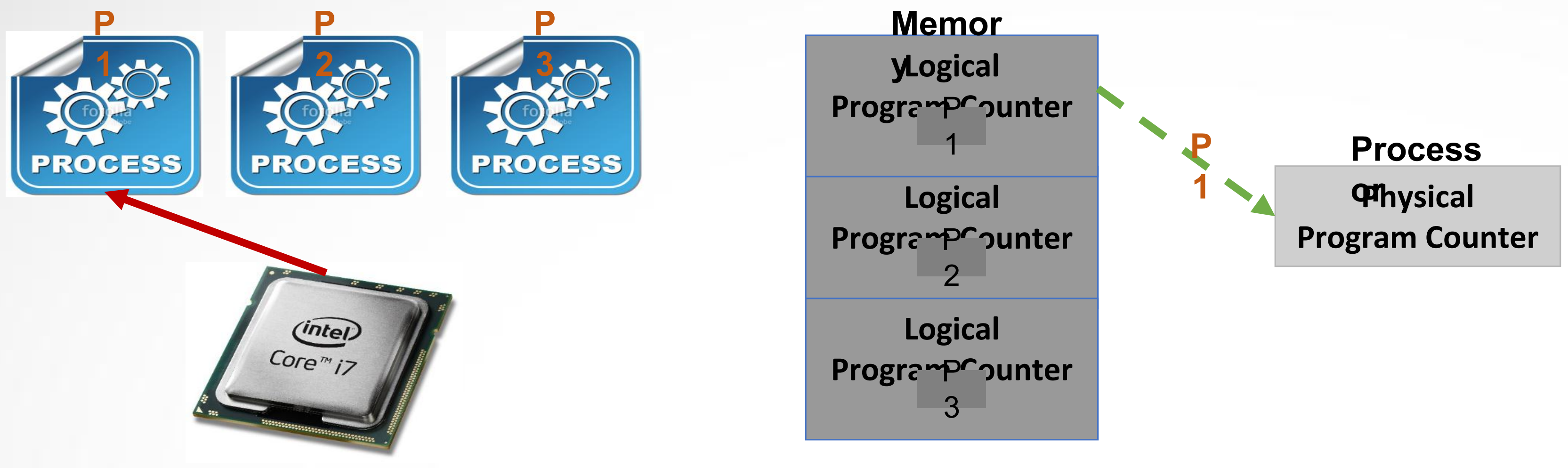
Logical Program Counter P1
Logical Program Counter P2
Logical Program Counter P3

Processor

Physical Program Counter

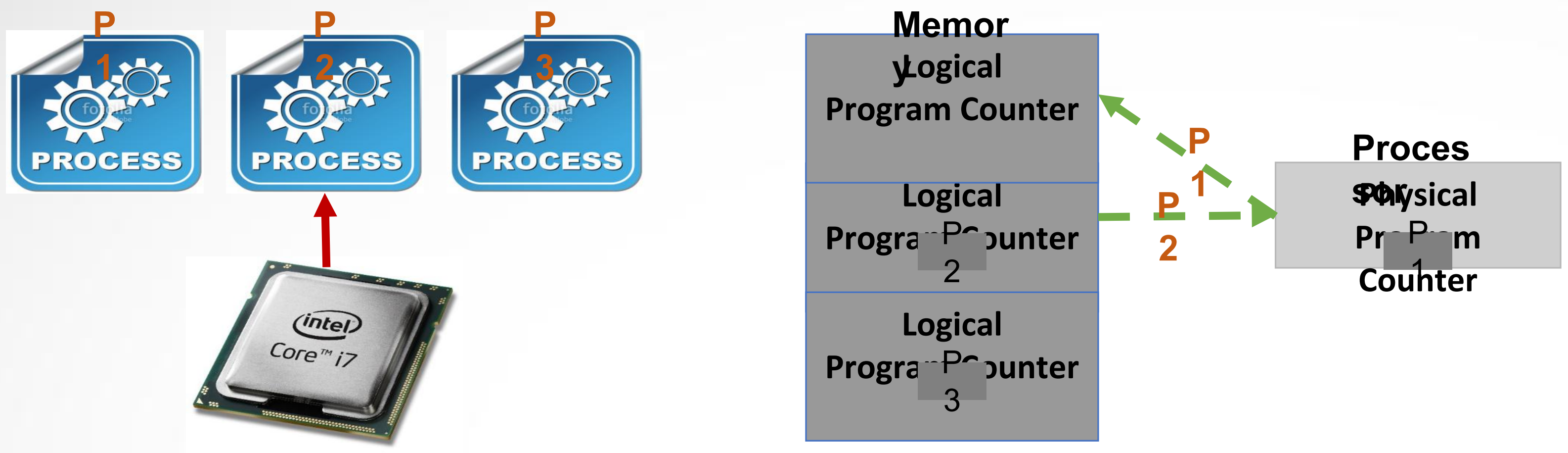
- There are **three processes**, **one processor** (CPU), **three logical program counter** (one for each processes) in memory and one physical program counter in processor.
- Here **CPU is free** (no process is running).
- No data in physical program counter.

Multiprogramming execution



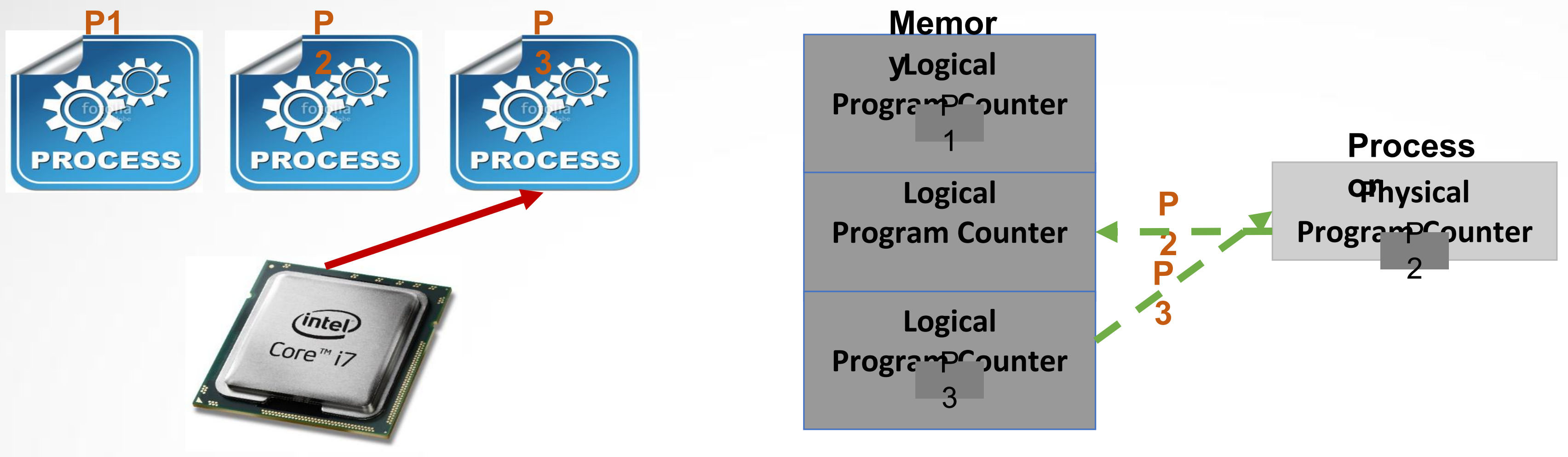
- CPU is **allocated to process P1** (process P1 is running).
- **Data of process P1 is copied** from its logical program counter to the physical program counter.

Multiprogramming execution



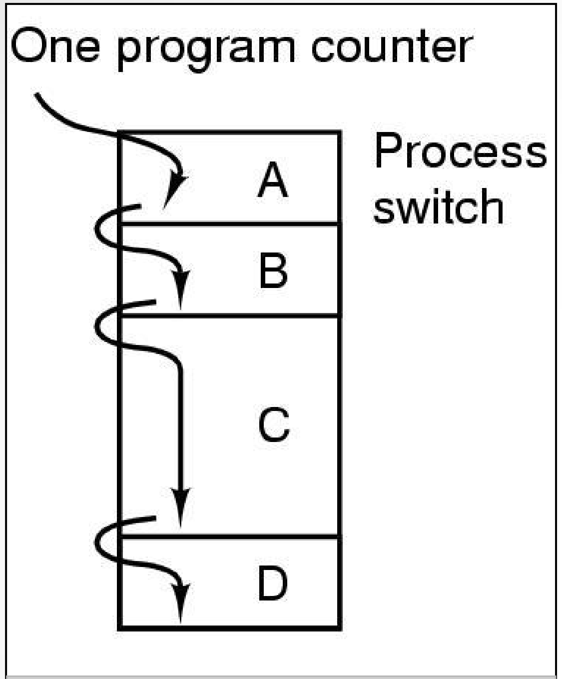
- CPU **switches** from process **P1 to** process **P2**.
- CPU is **allocated to process P2** (process P2 is running).
- **Data of process P1 is copied back** to its logical program counter.
- **Data of process P2 is copied** from its logical program counter to the physical program counter.

Multiprogramming execution

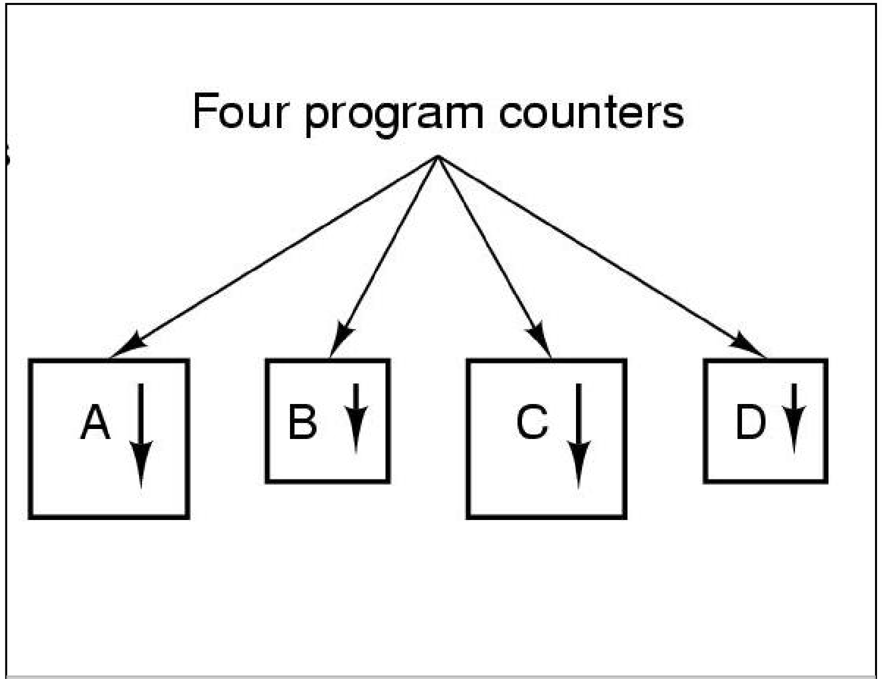


- CPU **switches** from process **P2 to** process **P3**.
- CPU is **allocated to process P3** (process P3 is running).
- **Data of process P2 is copied back** its logical program counter.
- **Data of process P3 is copied** from its logical program counter to the physical program counter.

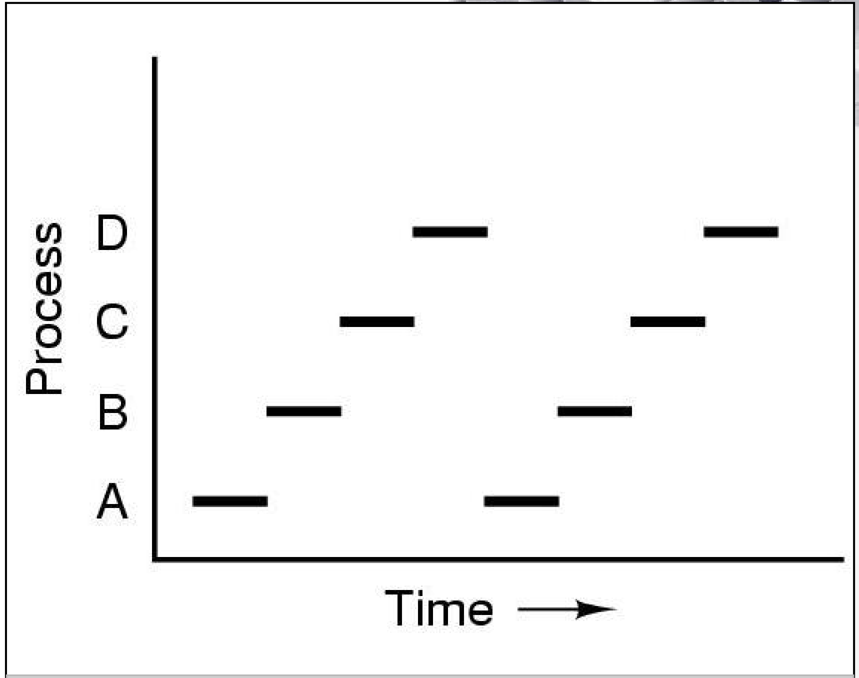
Process Model



Multiprogramming of
four programs in memory



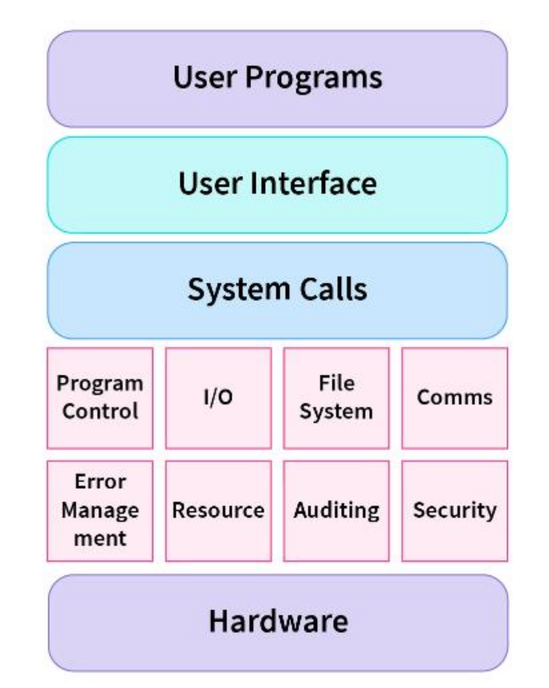
Conceptual model of 4 independent, sequential processes, each with its own flow of control (i.e., its own logical program counter) and each one running independently of the other ones.



Over a long period of time interval, all the processes have made progress, but at any given instant only one process is actually running.

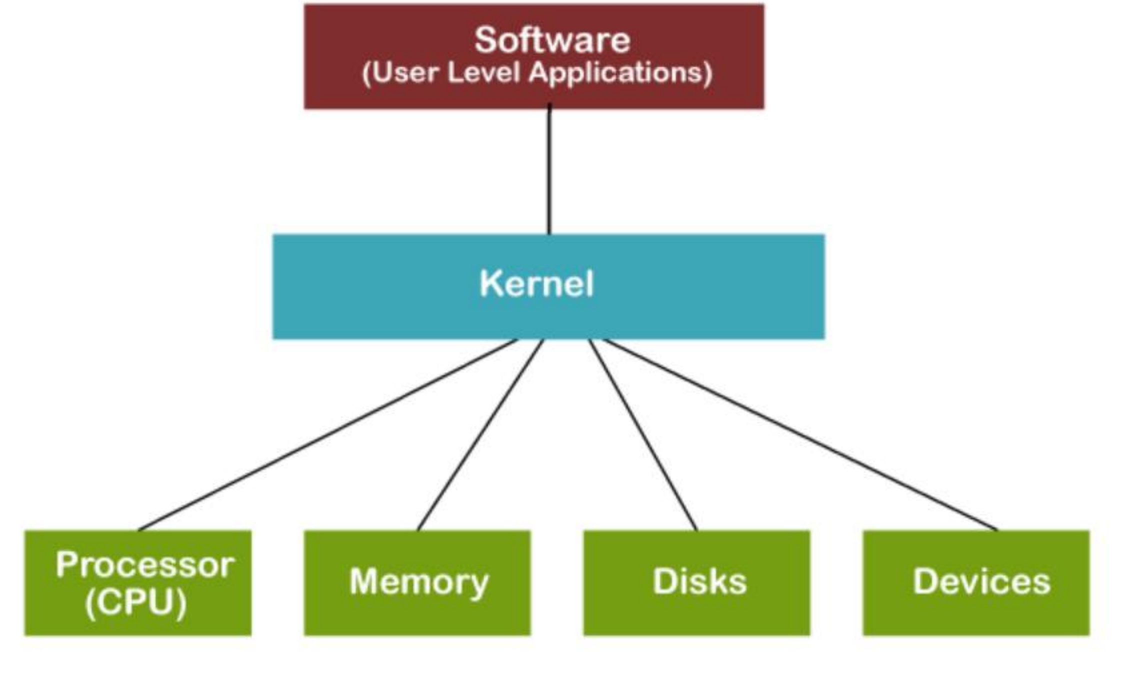
System call

A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API).



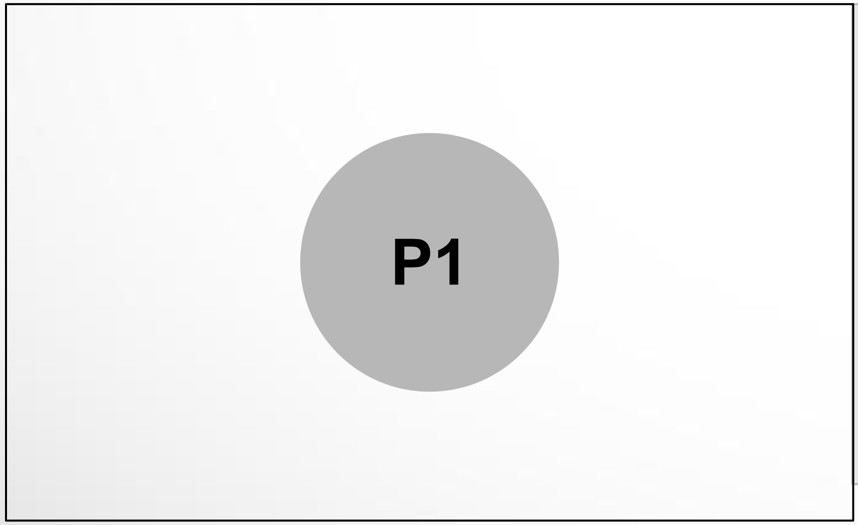
Kernel

- Kernel is central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time. It is core component of an operating system. Kernel acts as a bridge between applications and data processing performed at hardware level using inter-process communication and system calls.



Process Creation

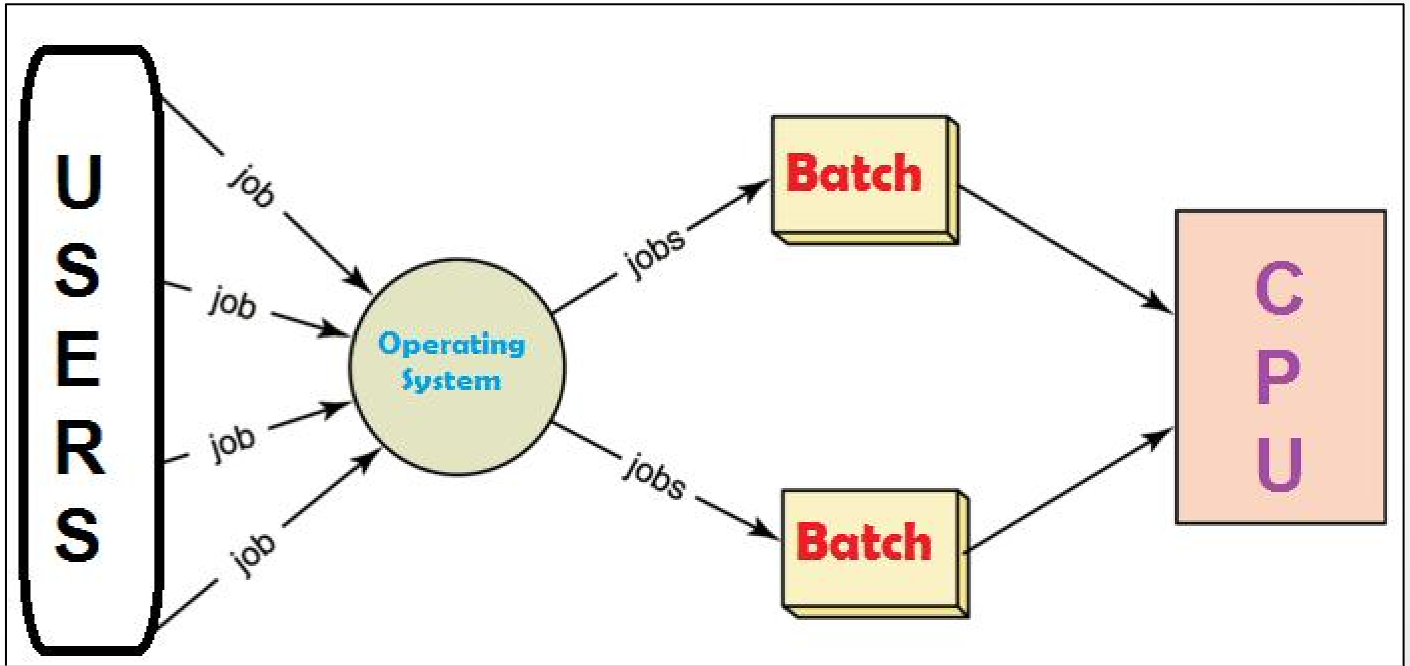
1. **System initialization**
   - At the time of **system (OS) booting** various processes are created
   - Foreground and background processes are created
   - **Background process** – that **do not interact with user** e.g. process to accept mail
   - **Foreground Process** – that **interact with user**



2. **Execution of a process creation system call (fork) by running process**
   - Running process will issue system call (fork) to **create one or more new process to help it**.
   - A process fetching large amount of data and execute it will create two different processes one for fetching data and another to execute it.

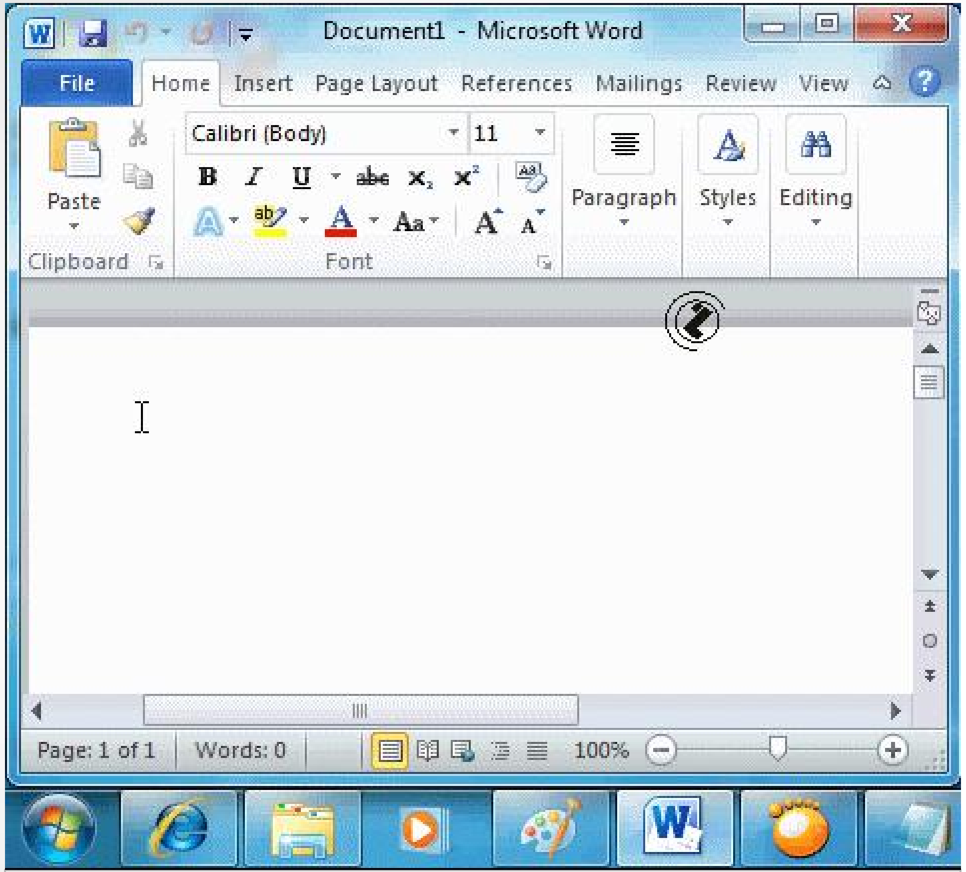
Process Creation

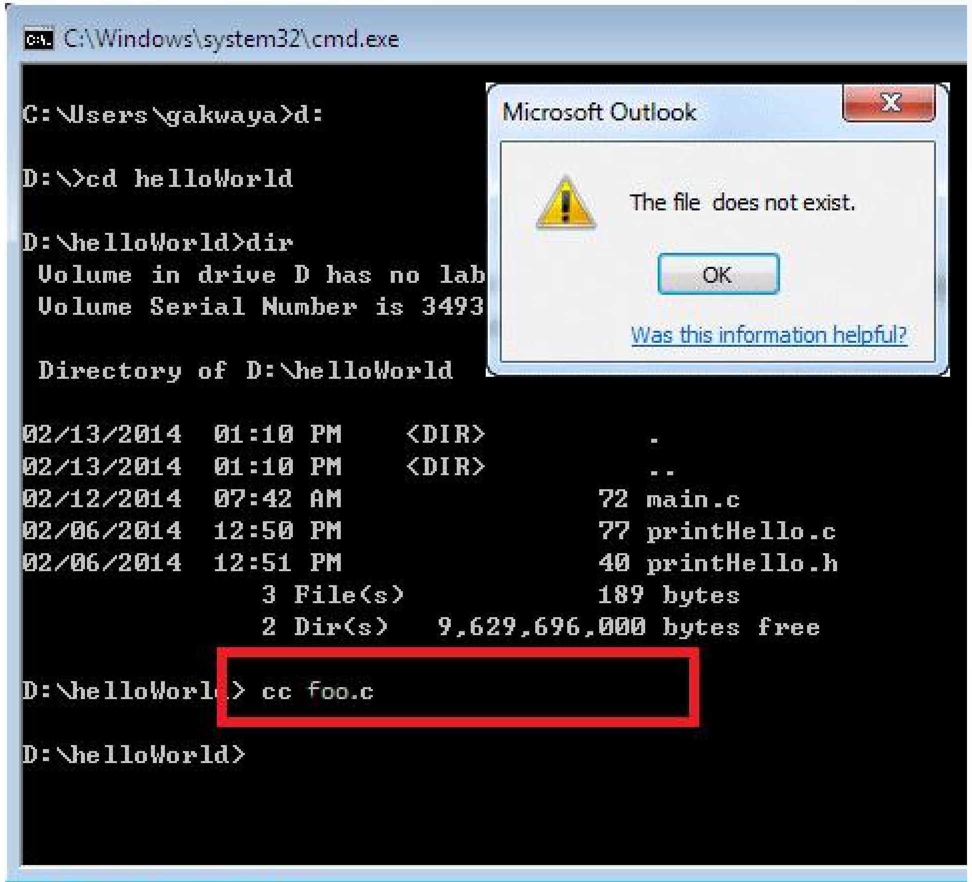
3. A **user request to create a new process**
   - Start process by clicking an icon (opening word file by double click) or by typing command.



4. **Initialization of batch process**
   - Applicable to only batch system found on large mainframe

Process Termination

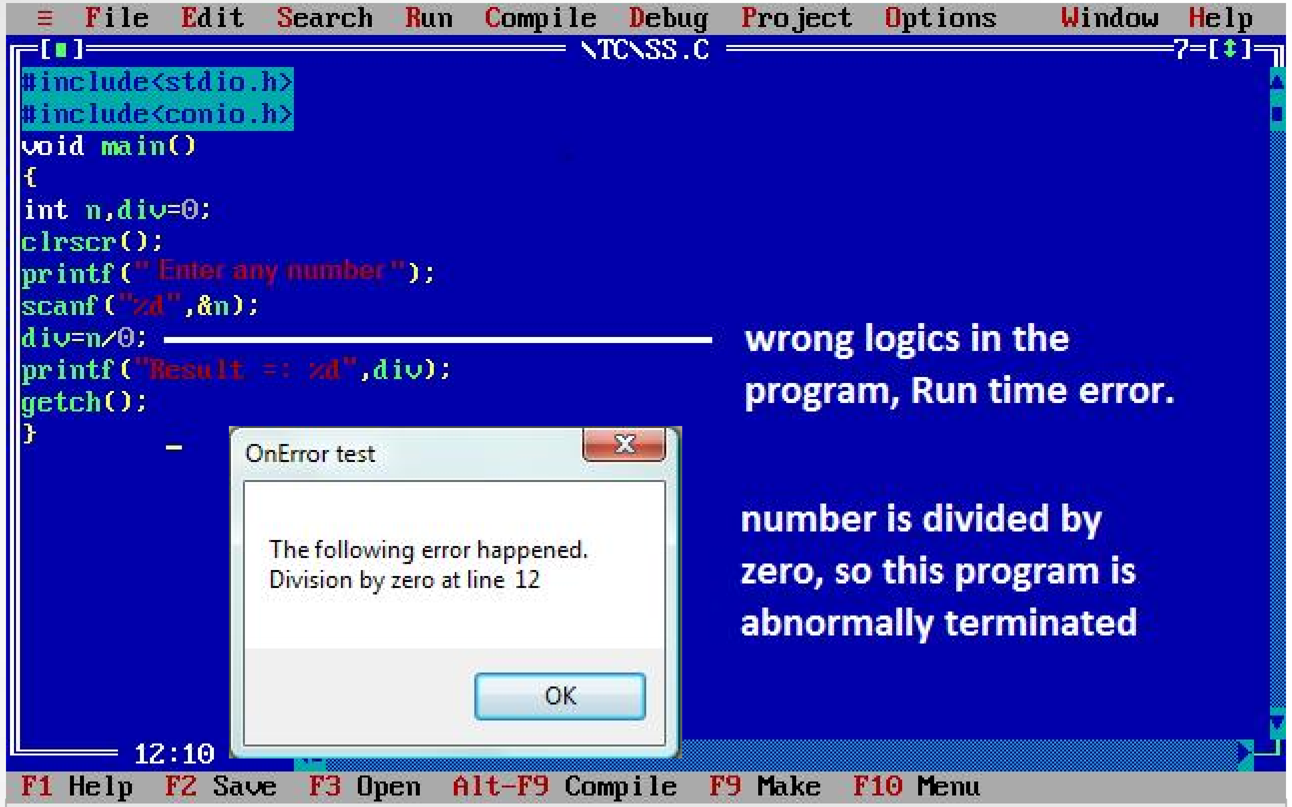




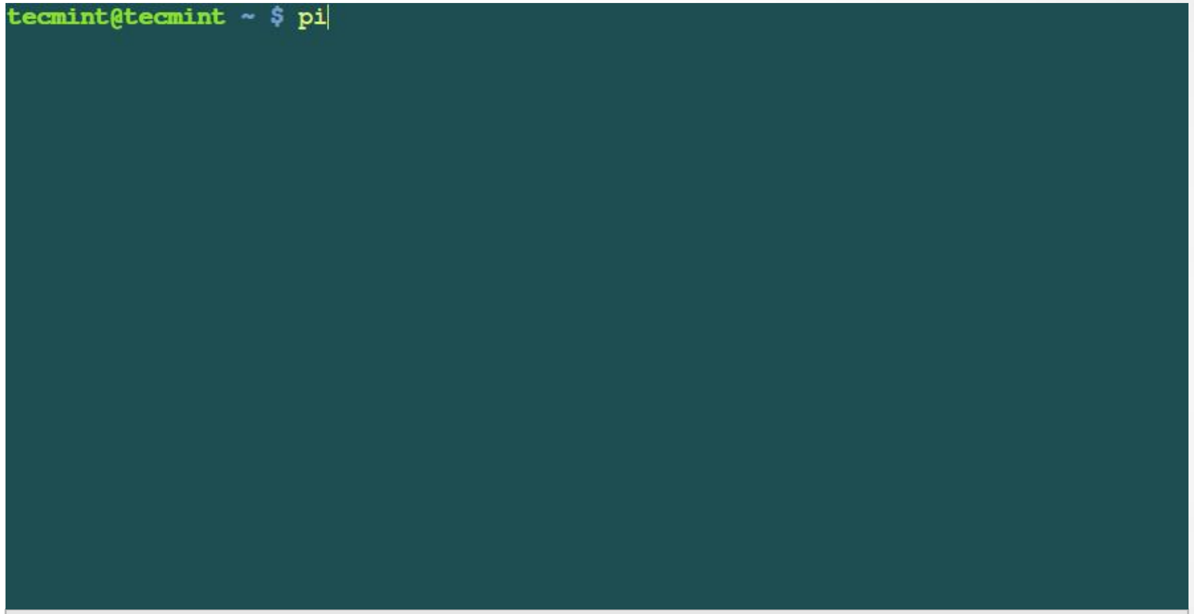
1. **Normal exit (voluntary)**
   - Terminated because process has done its work.

2. **Error exit (voluntary)**
   - The process discovers a fatal error e.g. user types the command cc foo.c to compile the program foo.c and no such file exists, the compiler simply exit.

Process Termination



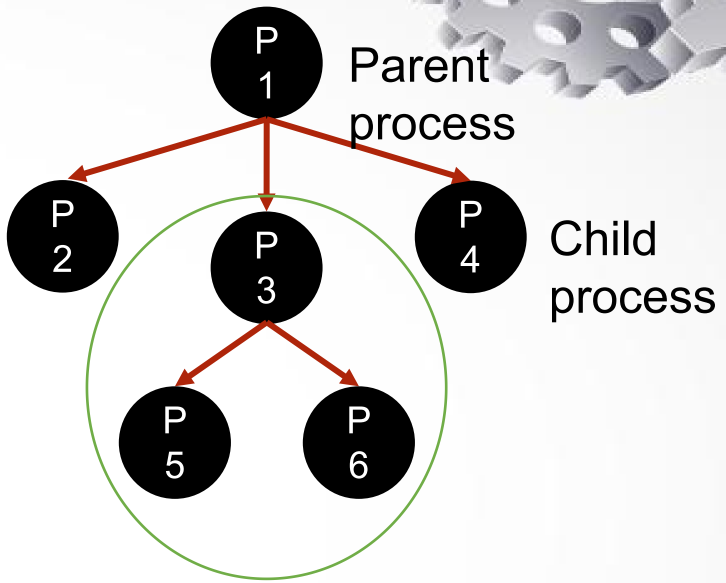
3. **Fatal error (involuntary)**
   - An error caused by a process often due to a program bug e.g. executing an illegal instruction, referencing nonexistent memory or divided by zero.

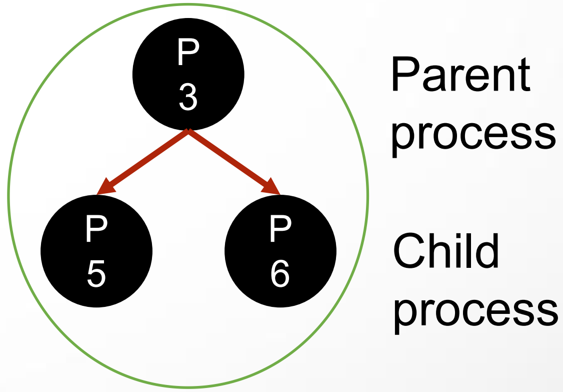


4. **Killed by another process (involuntary)**
   - A process executes a system call telling the OS to kill some other process using kill system call.

Process Hierarchies

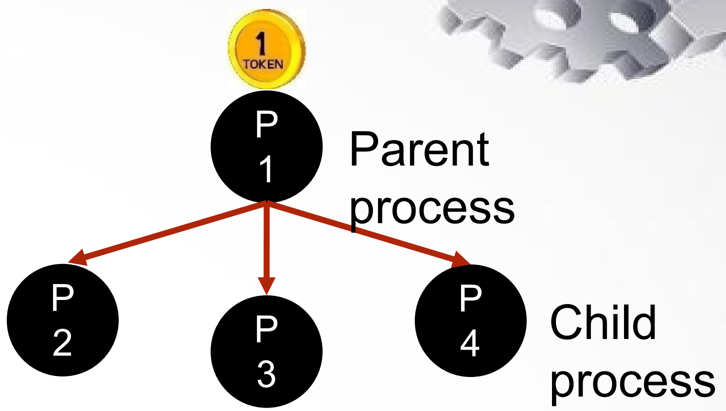
- Parent process can create child process, child process can create its own child process.
- **UNIX has hierarchy concept** which is known as process group
- Windows has no concept of hierarchy
  - All the process as treated equal (**use handle** concept)





Handle

- When a process is created, the **parent process is given a special token called handle**.
- This **handle is used to control the child process**.
- A **process is free to pass this token** to some other process.



Process states

Section - 2

Process states

Running

Running:

Process is actually using the CPU

Ready

Ready:

Process is runnable, temporarily stopped to let another process to run

Blocked

Blocked:

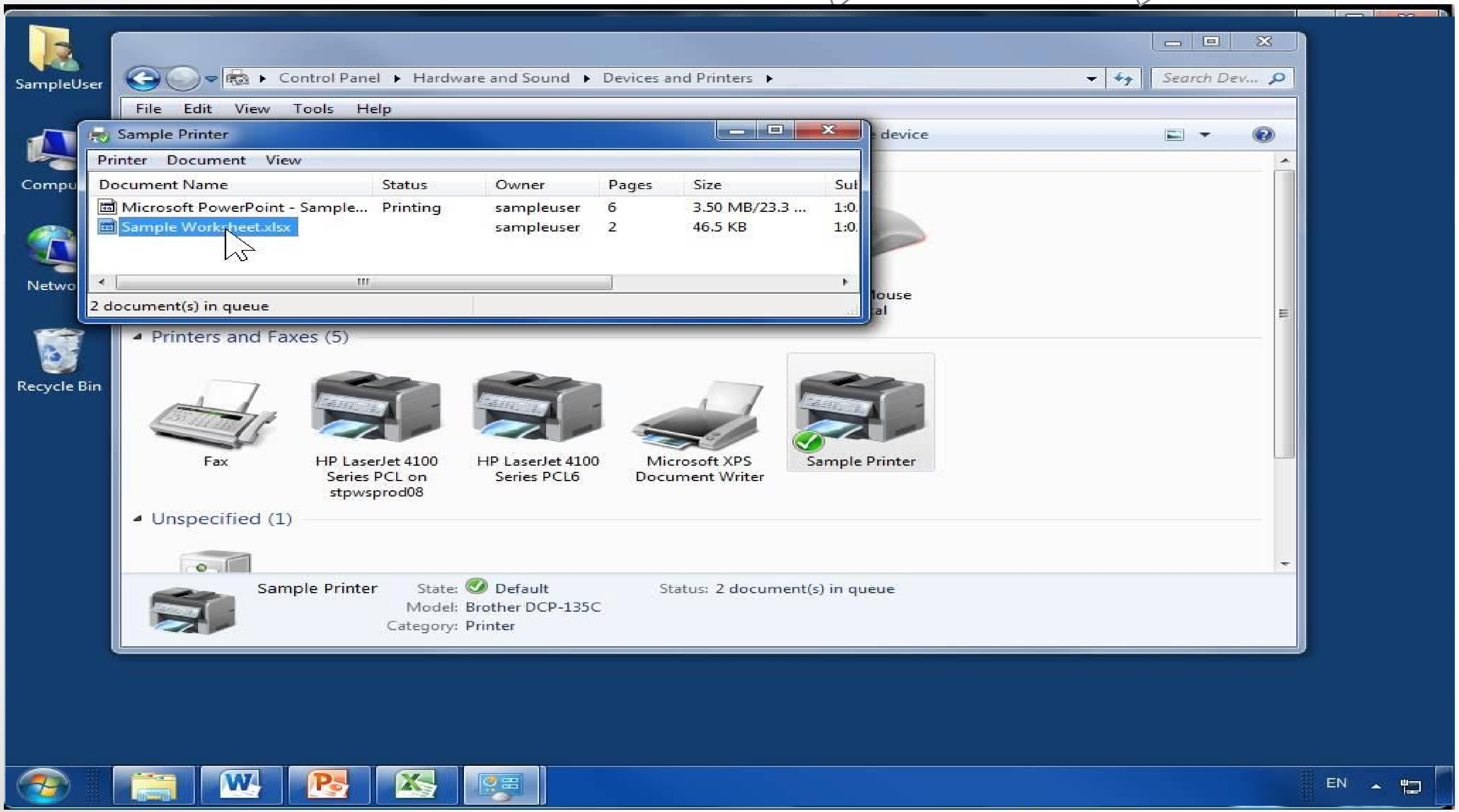
Process is unable to run until some external event happens

Process states

Ready

Blocked

Running

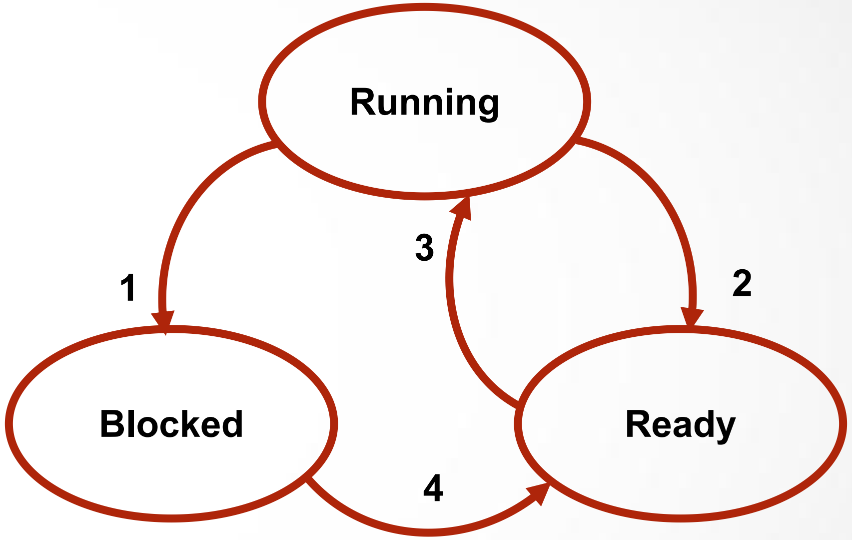


Process states transitions

Section - 3

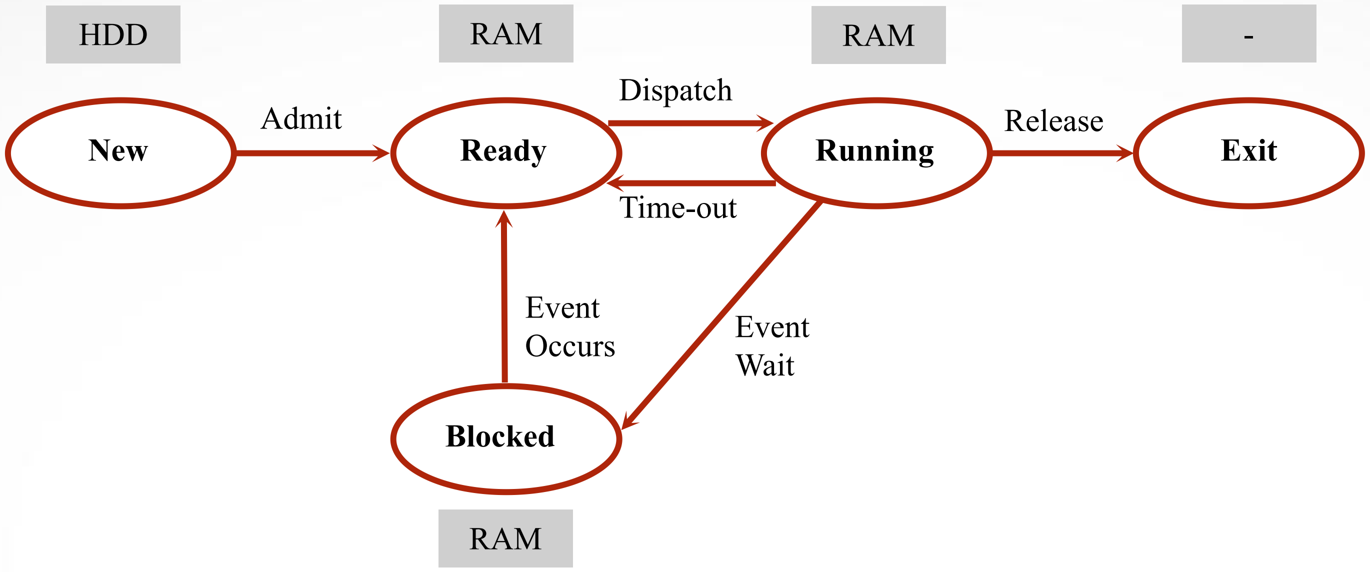
Process State Transitions

- When and how these transitions occur (process moves from one state to another)?
  1. Process blocks for input or waits for an event (i.e. printer is not available)
  2. Scheduler picks another process
     - End of time-slice or pre-emption.
  3. Scheduler picks this process
  4. Input becomes available, event arrives (i.e. printer become available)



Processes are always either executing (running) or waiting to execute (ready) or waiting for an event (blocked) to occur.

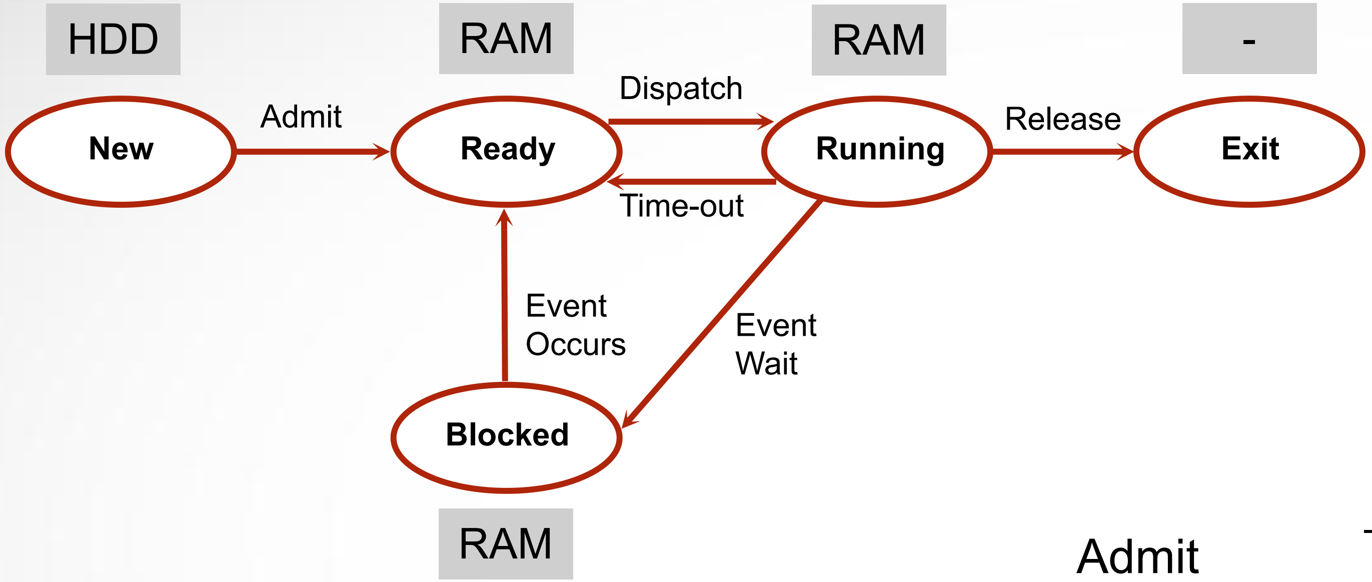
Five State Process Model and Transitions

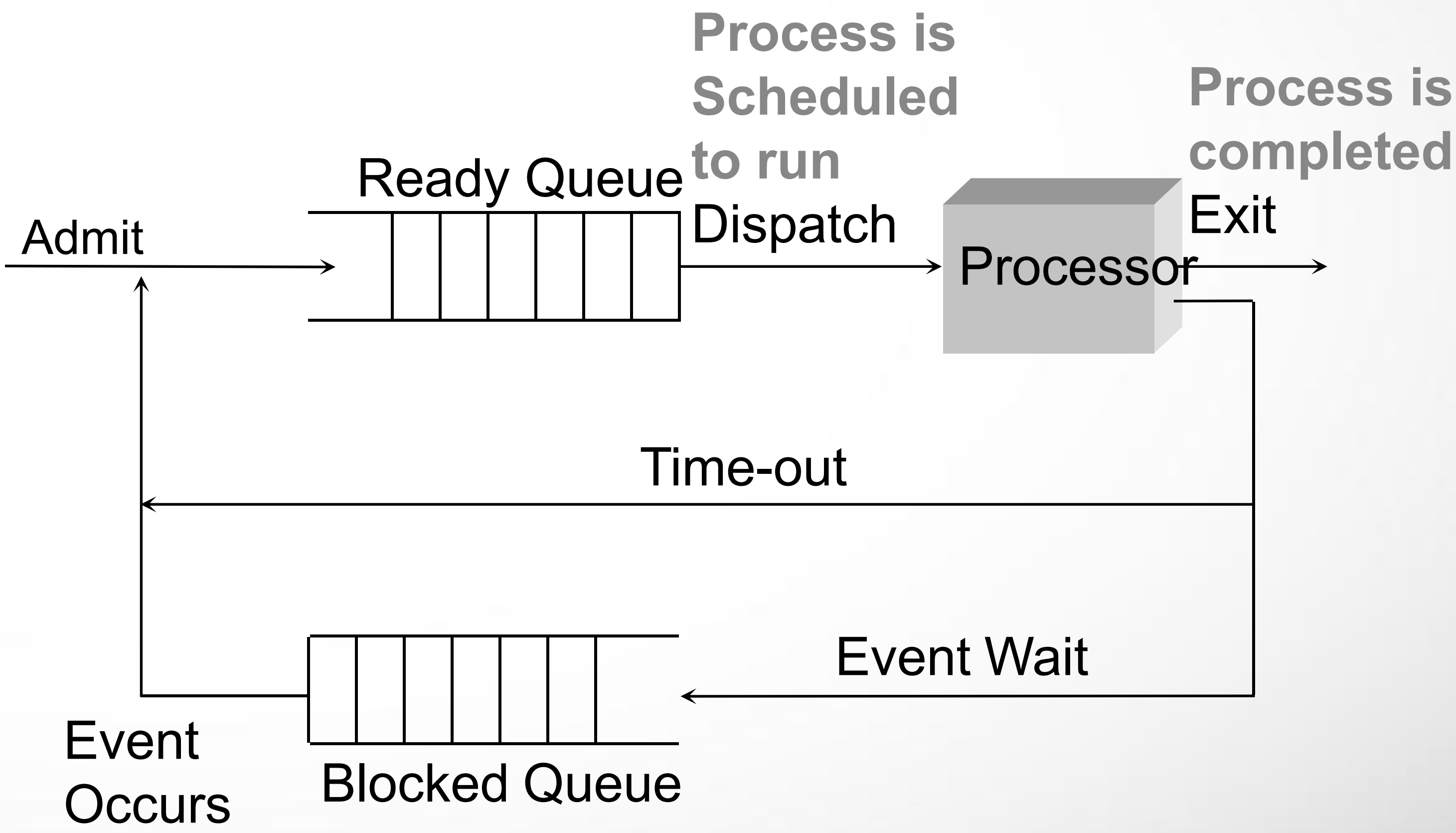


- **New** – process is being **created**
- **Ready** – process is **waiting to run (runnable)**, temporarily stopped to let another process run
- **Running** – process is actually **using the CPU**
- **Blocked** – **unable to run** until some external event happens
- **Exit (Terminated)** – process has **finished the execution**

Exercise A process resides in which memory during different state?

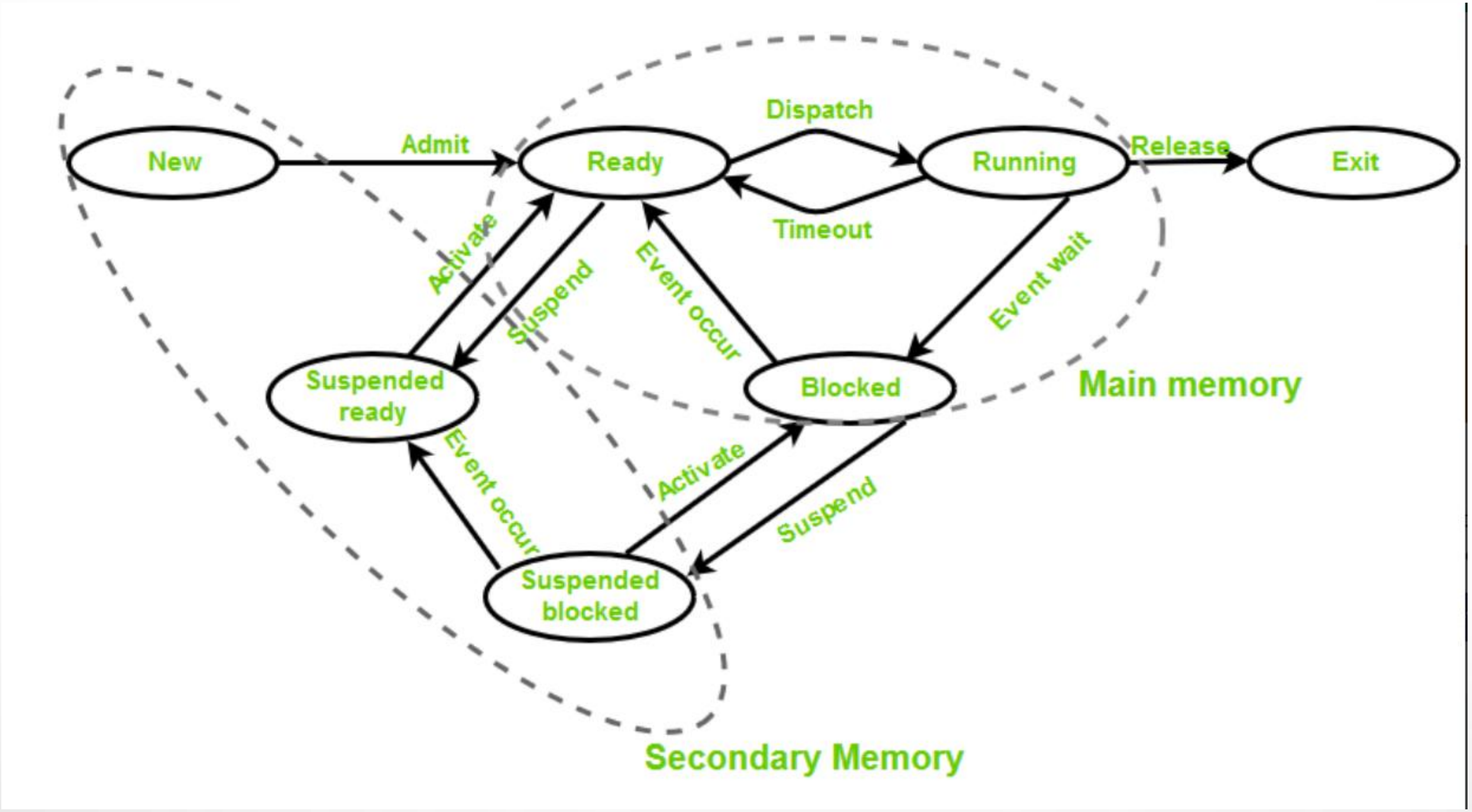
Queue Diagram





7 States of a Process

A process has several stages that it passes through from beginning to end. There must be a minimum of five states. Even though during execution, the process could be in one of these states, the names of the states are not standardized. Each process goes through several stages throughout its life cycle.



- **Suspend Ready:** Process that was initially in the ready state but was swapped out of main memory(refer to Virtual Memory topic) and placed onto external storage by the scheduler is said to be in suspend ready state. The process will transition back to a ready state whenever the process is again brought onto the main memory.
- **Suspend wait or suspend blocked:** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready.

Process Control Block (PCB)

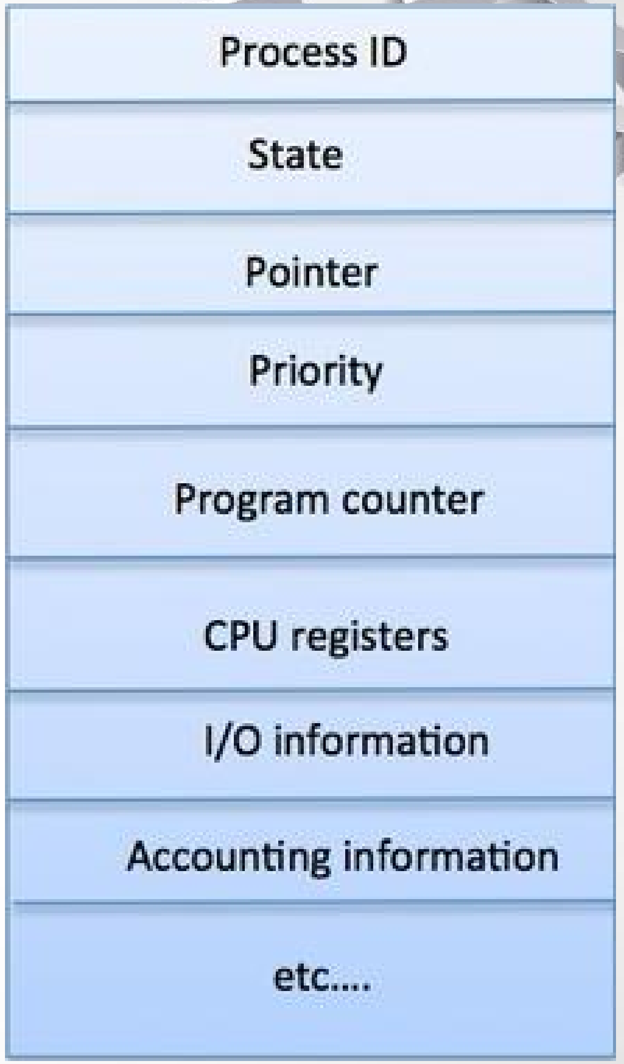
Section - 4

What is Process Control Block (PCB)?

- A Process Control Block (PCB) is a **data structure maintained by the operating system for every process**.
- PCB is used for **storing the collection of information about the processes**.
- The PCB is **identified by an integer process ID (PID)**.
- A PCB **keeps all the information needed to keep track of a process**.
- The PCB is maintained for a process **throughout its lifetime** and is **deleted once the process terminates**.
- The **architecture** of a PCB is completely **dependent on operating system** and may contain different information in different operating systems.
- PCB **lies in kernel memory space**.

Fields of Process Control Block (PCB)

- **Process ID** - Unique identification for each of the process in the operating system.
- **Process State** - The current state of the process i.e., whether it is ready, running, waiting.
- **Pointer** - A pointer to parent process.
- **Priority** - Priority of a process.
- **Program Counter** - Program Counter is a pointer to the address of the next instruction to be executed for this process.
- **CPU registers** - Various CPU registers where process need to be stored for execution for running state.
- **IO status information** - This includes a list of I/O devices allocated to the process.
- **Accounting information** - This includes the amount of CPU used for process execution, time limits etc.

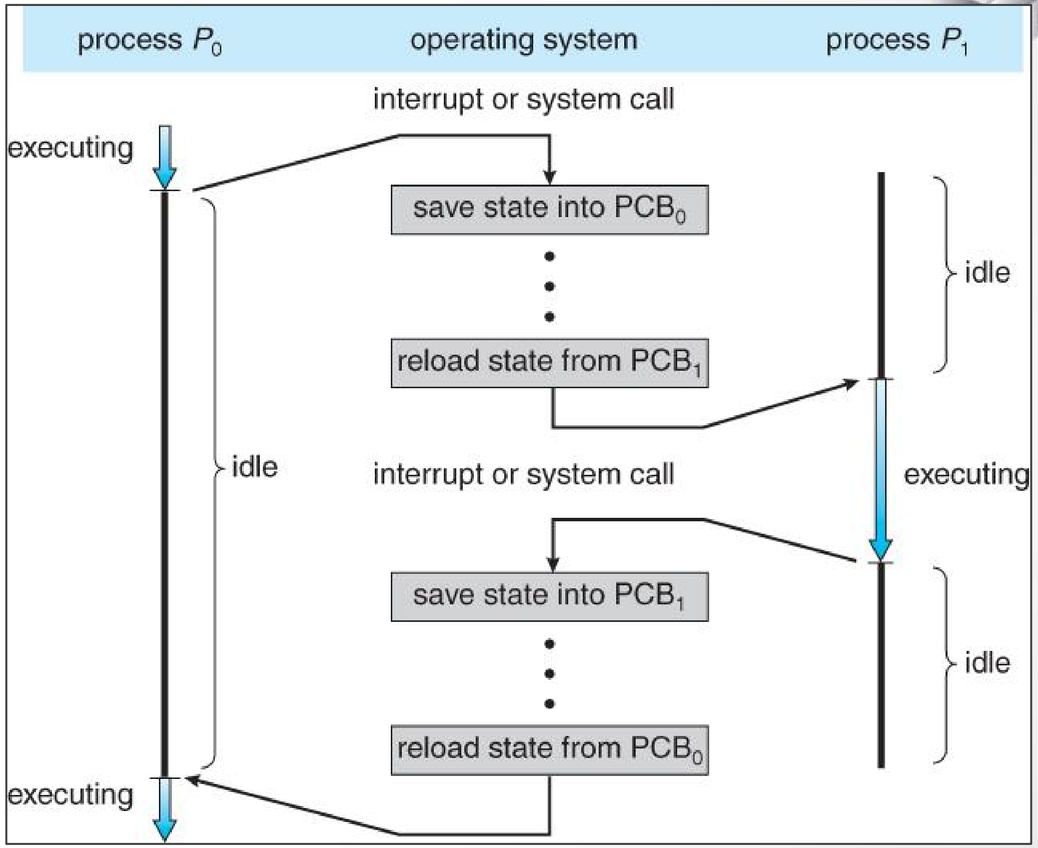


Context switching

Section - 5

Context switching

- Context switch means **stopping one process** and **restarting another process**.
- When an event occur, the **OS saves the state of an active process (into its PCB)** and **restore the state of new process (from its PCB)**.
- Context switching is **purely overhead** because system does not perform any useful work while context switch.
- Sequence of action:
  1. **OS takes control** (through interrupt)
  2. **Saves context of running process** in the process PCB
  3. **Reload context of new process** from the new process PCB
  4. **Return control** to new process



Exercise What causes (Reasons for) context switching?

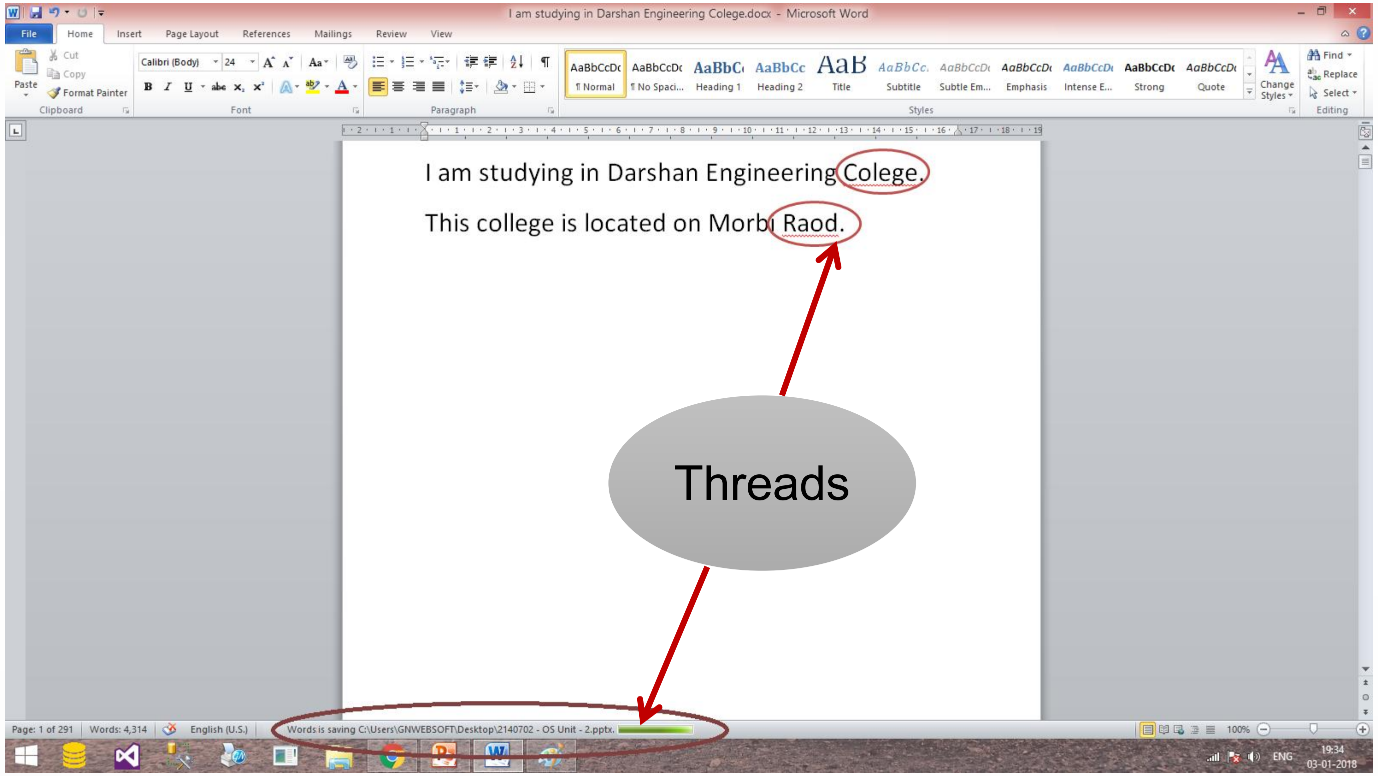
1. Time slice has elapsed
2. Process with a higher priority has become ready to run.

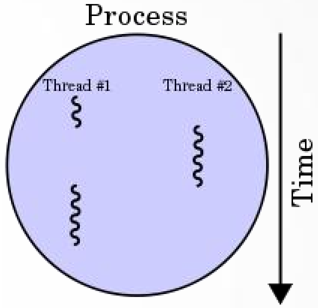
Threads

Section - 6

What is Threads?

- Thread is **light weight process** created by a process.

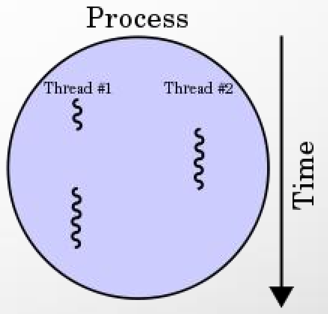




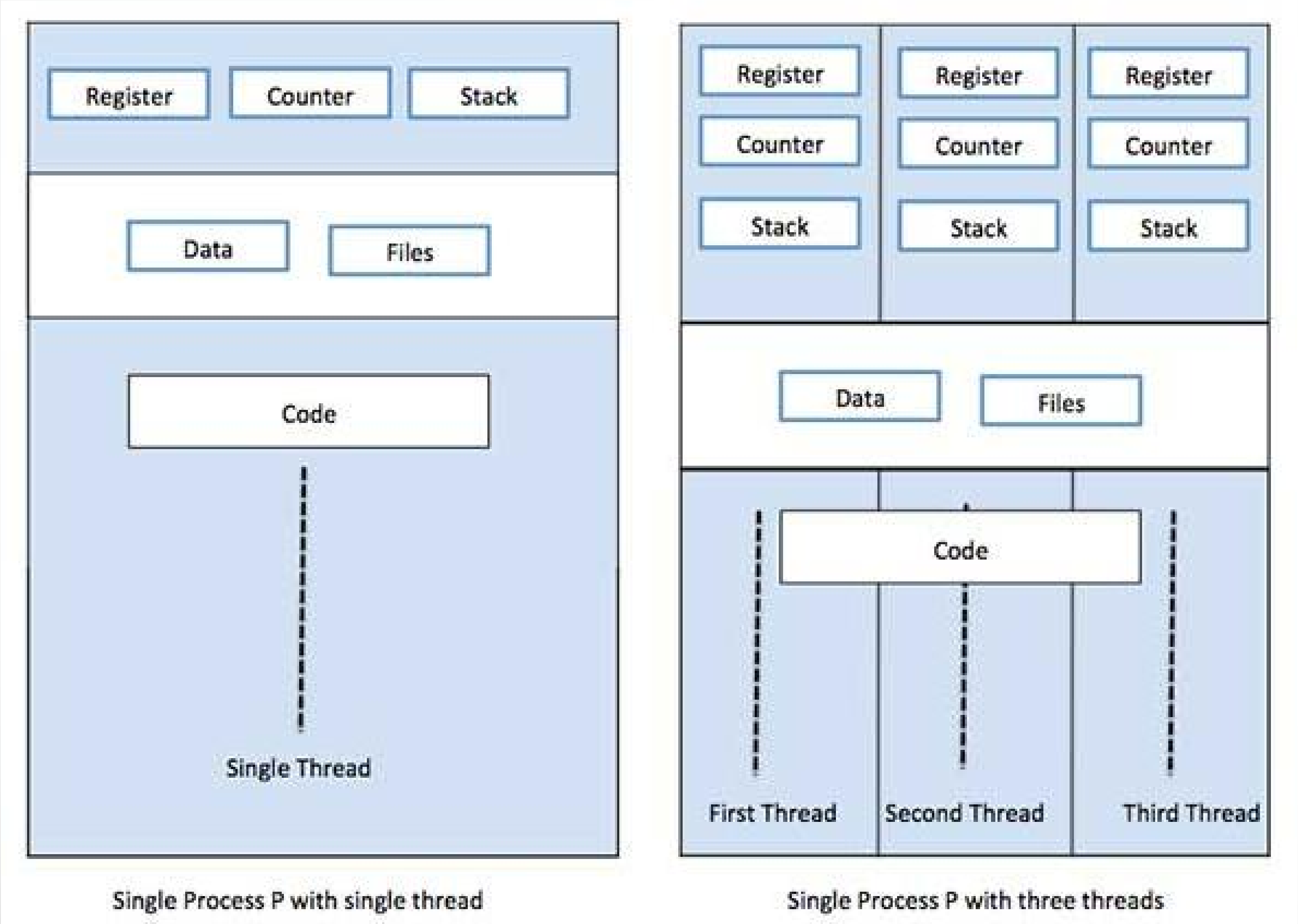
Processes are used to execute large, 'heavyweight' jobs such as working in word, while threads are used to carry out smaller or 'lightweight' jobs such as auto saving a word document.

What is Threads?

- Thread is **light weight process** created by a process.
- Thread is a **single sequence stream** within a process.
- Thread has it own
  - **program counter** that **keeps track of which instruction to execute next**.
  - **system registers** which **hold its current working variables**.
  - **stack** which **contains the execution history**.



Single Threaded Process VS Multiple Threaded Process



- A single-threaded process is a process with a single thread.
- A multi-threaded process is a process with multiple threads.
- The multiple threads have its own registers, stack and counter but they share the code and data segment.

Comparison between process and thread

Section - 7

Similarities between Process & Thread

- Like processes threads **share CPU and only one thread is running at a time**.
- Like processes threads **within a process execute sequentially**.
- Like processes thread **can create children's**.
- Like a traditional process, a thread **can be in any one of several states: running, blocked, ready or terminated**.
- Like process threads **have Program Counter, Stack, Registers and State**.

Dissimilarities between Process & Thread

- Unlike processes threads are **not independent of one another**.
- Threads within the same process **share an address space**.
- Unlike processes all threads **can access every address in the task**.
- Unlike processes threads are **design to assist one other**. Note that processes might or might not assist one another because processes may be originated from different users.

Benefits/Advantages of threads

Section - 8

Benefits/Advantages of Threads

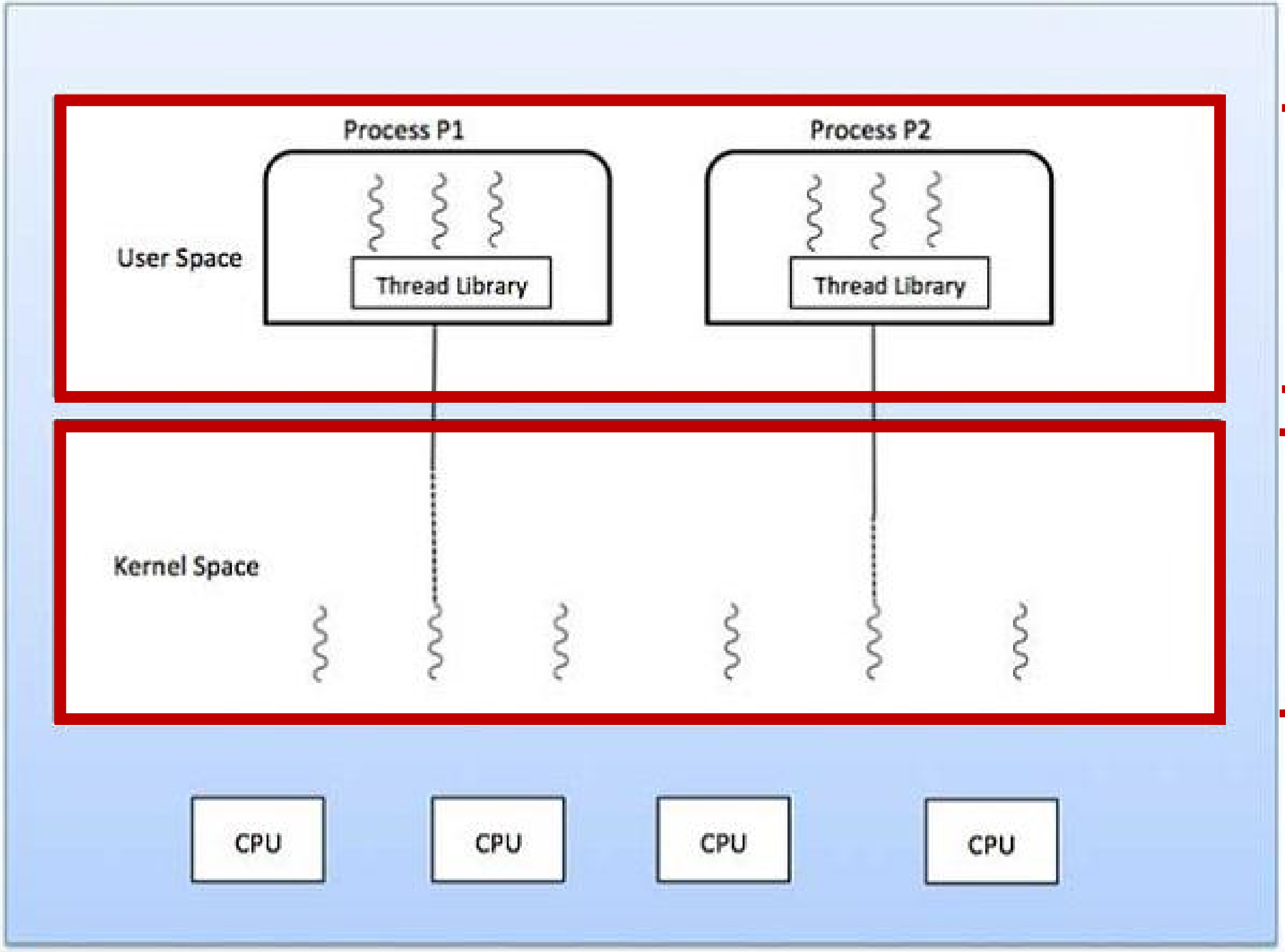
- Threads **minimize** the **context switching time**.
- Use of threads **provides concurrency** within a process.
- **Efficient communication**.
- It is more **easy to create** and **context switch** threads.
- Threads can **execute in parallel** on multiprocessors.
- With threads, an application can **avoid per-process overheads**
  - Thread creation, deletion, switching easier than processes.
- Threads have **full access to address space** (easy sharing).

Types of threads

Section - 9

Types of Threads

1. Kernel Level Thread
2. User Level Thread



User Level Threads

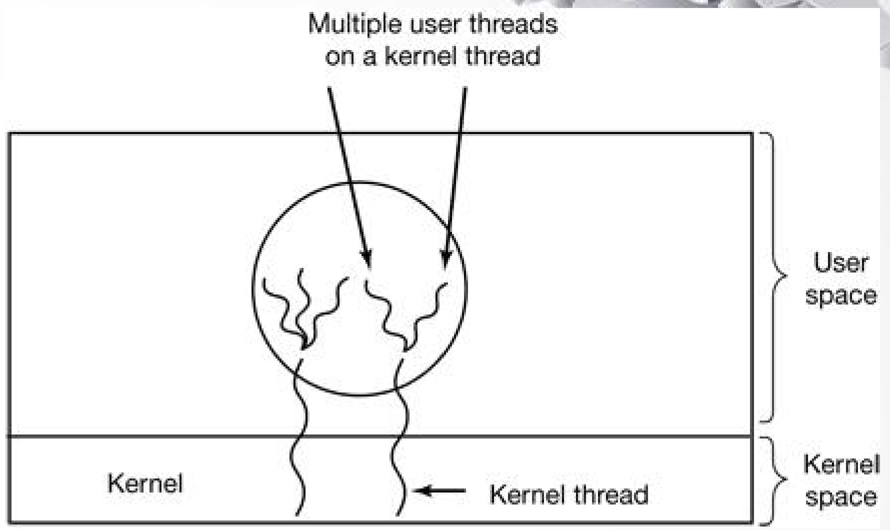
Kernel Level Threads

Types of Threads

User Level Thread	Kernel Level Thread
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn’t recognize user level threads.	Kernel threads are recognized by OS.
Implementation of user threads is easy.	Implementation of kernel thread is complex.
Context switch requires no hardware support.	Context switch requires hardware support.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread with in same process can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris

Hybrid Threads

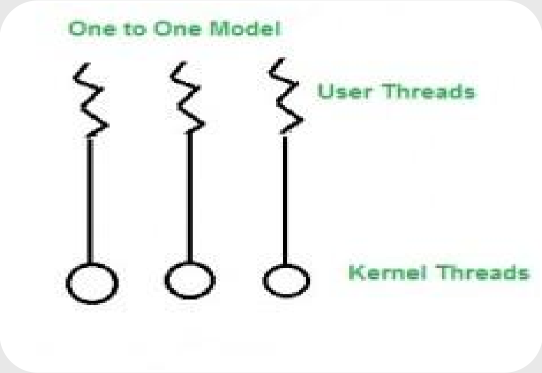
- Combines the advantages of user level and kernel level thread.
- It **uses kernel level thread** and then **multiplex user level thread on to some or all of kernel threads**.
- **Gives flexibility** to programmer that how many kernel level threads to use and how many user level thread to multiplex on each one.
- **Kernel is aware of only kernel level threads** and schedule it.

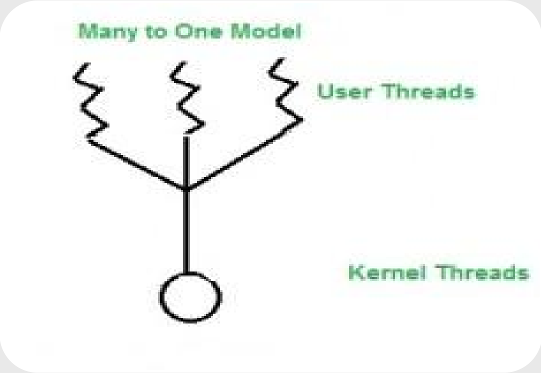


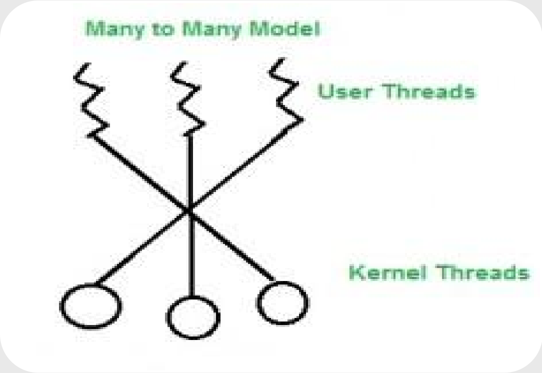
Multi Threading Models

Section - 10

Multi Threading Models







<u>One to One Model</u>

Each user threads mapped to one kernel thread.

Problem with this model is that creating a user thread requires the corresponding kernel thread.

<u>Many to One Model</u>

Multiple user threads mapped to one kernel thread.

Problem with this model is that a user thread can block entire process because we have only one kernel thread.

<u>Many to Many Model</u>

Multiple user threads multiplex to more than one kernel threads.

Advantage with this model is that a user thread can not block entire process because we have multiple kernel thread.

Pthread function calls

Section - 11

Pthread function calls

1. Pthread_create:- Create a new thread
2. Pthread_exit:- Terminate the calling thread
3. Pthread_join:- Wait for a specific thread to exit
4. Pthread_yield:- Release the CPU to let another thread run
5. Pthread_attr_init:- Create and initialize a thread's attribute structure
6. Pthread_destroy:- Remove a thread's attribute structure

System calls

Section - 12

System calls

- A system call is the **programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on**.
- A system call is a **way for programs to interact with the operating system**.
- A **computer program makes a system call when it makes a request to the operating system's kernel**.
- System call **provides the services of the operating system to the user programs via Application Program Interface(API)**.
- It **provides an interface** between a process and operating system to allow user-level processes to request services of the operating system.
- System calls are the **only entry points into the kernel system**.
- **All programs needing resources must use system calls**.

System calls

- ps (process status):- The ps (process status) command is used to **provide information about the currently running processes**, including their process identification numbers (PIDs).
- fork:- Fork system call is used for **creating a new process, which is called child process**, which runs concurrently with the process that makes the fork() call (parent process).
- wait:- Wait system call **blocks the calling process until one of its child processes exits** or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.
- exit:- Exit system call **terminates the running process normally**.
- exec family:- The exec family of functions **replaces the current running process with a new process**.

Questions asked in examiantion

1. Explain Process/Thread Life Cycle with diagram.
2. Explain process control block (PCB) with diagram.
3. Difference between process and thread.
4. Write various multi threading models.
5. Write benefits of threads.

Outline-II

- What is scheduling
- Objectives of scheduling
- Types of scheduler
- Scheduling algorithms
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Shortest Remaining Time Next (SRTN)
  - Round Robin (RR)
  - Priority
    - Non-Preemptive Priority
    - Preemptive Priority
- Real Time Operating System

What is process scheduling?

Section - 1

What is process scheduling?

- Process scheduling is the **activity of the process manager** that handles **suspension of running process from CPU and selection of another process** on the basis of a particular strategy.
- The **part of operating system** that **makes** the **choice** is called **scheduler**.
- The **algorithm used by** this **scheduler** is called scheduling **algorithm**.
- Process scheduling is an essential part of a multiprogramming operating systems.

Objectives (goals) of scheduling

Section - 2

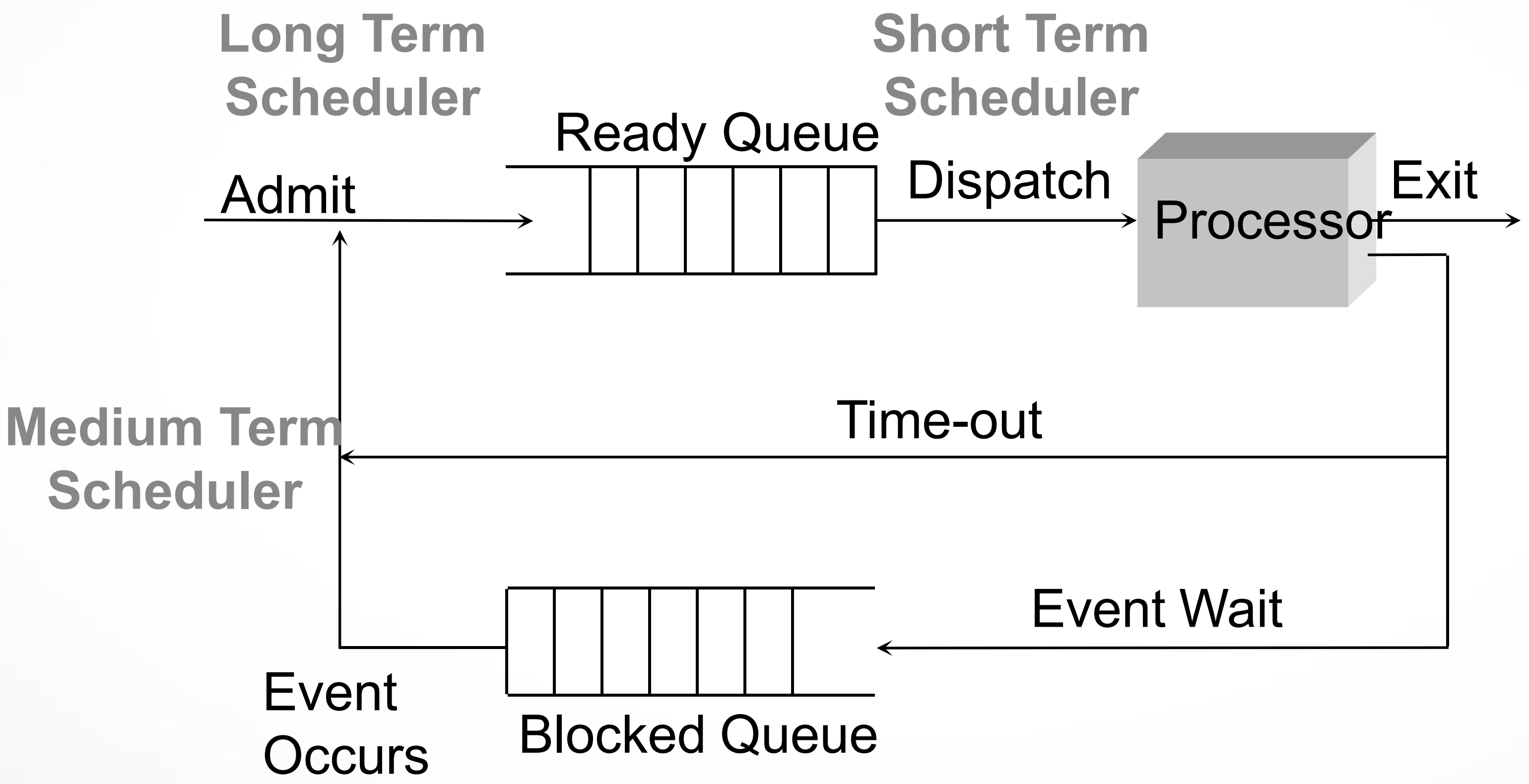
Objectives (goals) of scheduling

- **Fairness**: giving each process a fair share of the CPU.
- **Balance**: keeping all the parts of the system busy (Maximize).
- **Throughput**: no of processes that are completed per time unit (Maximize).
- **Turnaround time**: time to execute a process from submission to completion (Minimize).
  - **Turnaround time** = Process finish time – Process arrival time
- **CPU utilization**: percent of time that the CPU is busy in executing a process.
  - keep CPU as busy as possible (Maximized).
- **Response time**: time between issuing a command and getting the result (Minimized).
- **Waiting time**: amount of time a process has been waiting in the ready queue (Minimize).
  - **Waiting time** = Turnaround time – Actual execution time

Types of schedulers

Section - 3

Types of schedulers



Types of schedulers

Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
It is a job scheduler.	It is a CPU scheduler.	It is a process swapping scheduler.
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.
Speed is lesser than short term scheduler.	Speed is fastest among other two schedulers.	Speed is in between both short and long term scheduler.

Scheduling algorithms

Section - 4

Scheduling algorithms

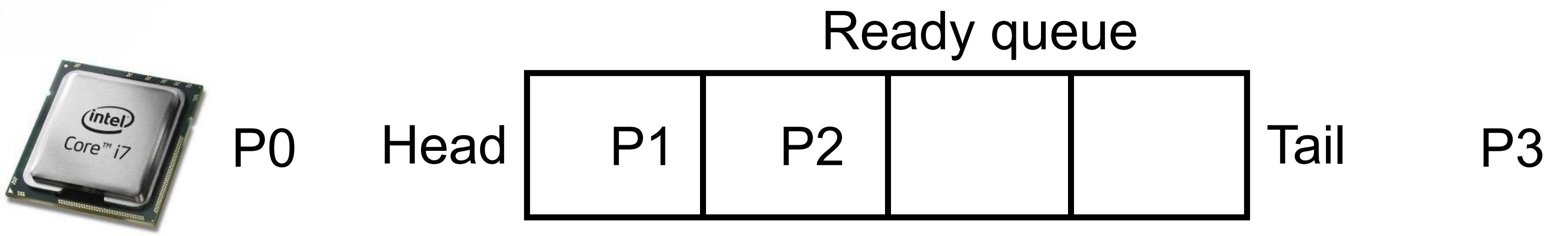
1. First Come First Served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time Next (SRTN)
4. Round Robin (RR)
5. Priority
   I. Preemptive
   II. Non-Preemptive

First Come First Served (FCFS)

Section – 4.1

First Come First Served (FCFS)

- Selection criteria:
  - The **process that request first is served first**.
  - It means that **processes are served in the exact order of their arrival**.

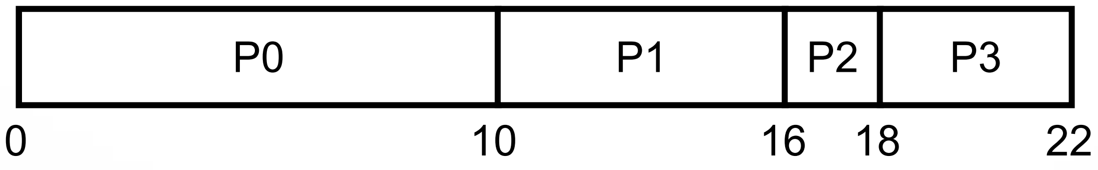


- Decision Mode:
  - **Non preemptive**: **Once a process is selected, it runs until it is blocked** for an I/O or some other event or it is terminated.
- Implementation:
  - This strategy can be easily **implemented by using FIFO** (First In First Out) queue.
  - When CPU becomes free, a process from the first position in a queue is selected to run.

First Come First Served (FCFS)

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - ΔT)
P0	0	10	10	10	0
P1	1	6	16	15	9
P2	3	2	18	15	13
P3	5	4	22	17	13

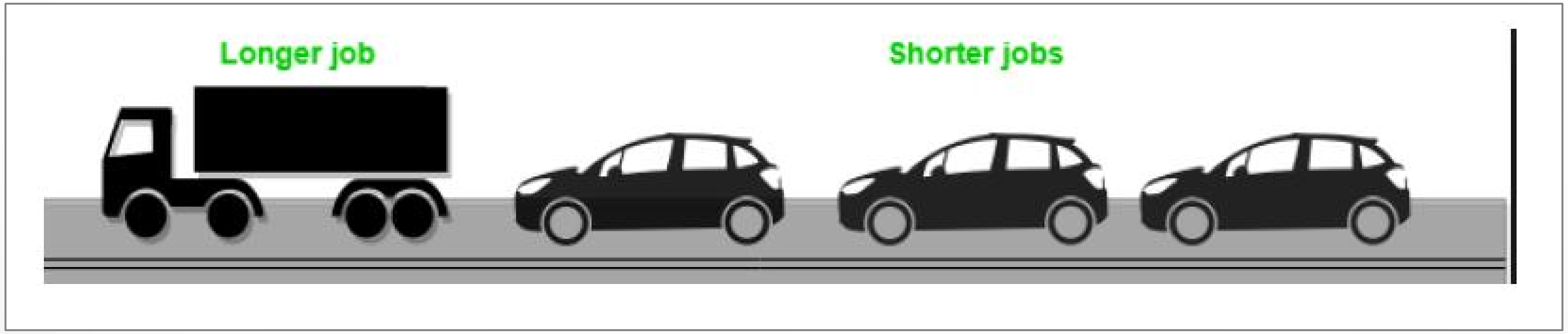
- Gantt Chart



- Average Turnaround Time: (10+15+15+17)/4 = 14.25 ms.
- Average Waiting Time: (0+9+13+13)/4 = 8.75 ms.

First Come First Served (FCFS)

- Advantages
  - **Simple** and **fair**.
  - **Easy to understand** and **implement**.
  - Every process will get a chance to run, so **starvation doesn't occur**.
    - Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.
- Disadvantages
  - **Not efficient** because average waiting time is too high.
  - **Convoy effect is possible**. All small I/O bound processes wait for one big CPU bound process to acquire CPU.



  - **CPU utilization may be less efficient** especially when a CPU bound process is running with many I/O bound processes.

Shortest Job First (SJF)

Section – 4.2

Shortest Job First (SJF)

- Selection criteria:
  – The process, that **requires shortest time to complete execution, is served first**.

Ready queue

P0 (4) Head

P1 (2)	P2 (6)		

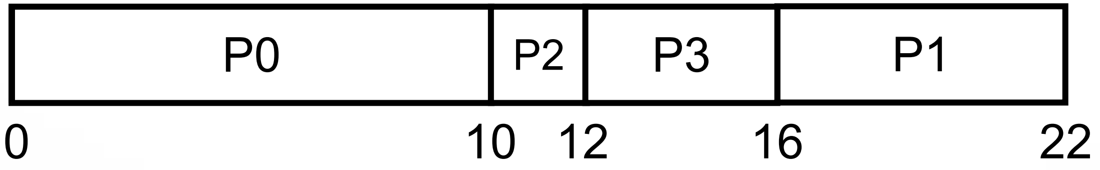
Tail P3 (3)

- Decision Mode:
  – **Non preemptive**: **Once a process is selected, it runs until it is blocked** for an I/O or some other event or it is terminated.
- Implementation:
  – This strategy can be easily **implemented by using sorted FIFO** (First In First Out) queue.
  – All processes in a queue are **sorted in ascending order based on their required CPU bursts**.
  – When CPU becomes free, a process from the first position in a queue is selected to run.

Shortest Job First (SJF)

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - ΔT)
P0	0	10	10	10	0
P1	1	6	22	21	15
P2	3	2	12	9	7
P3	5	4	16	11	7

- Gantt Chart



- Average Turnaround Time: (10+21+9+11)/4 = 12.75 ms.
- Average Waiting Time: (0+15+7+7)/4 = 7.25 ms.

Shortest Job First (SJF)

- Advantages
  - **Less waiting time.**
  - **Good response for short processes.**
- Disadvantages
  - It is **difficult to estimate time required** to complete execution.
  - **Starvation is possible for long process**. Long process may wait forever.
    - Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.

Shortest Remaining Time Next (SRTN)

Section – 4.3

Shortest Remaining Time Next (SRTN)

- Selection criteria:
  - The process, **whose remaining run time is shortest, is served first**. This is a **preemptive version of SJF** scheduling.

Ready queue

P0 (2)

Head	P1 (2)	P2 (6)			Tail

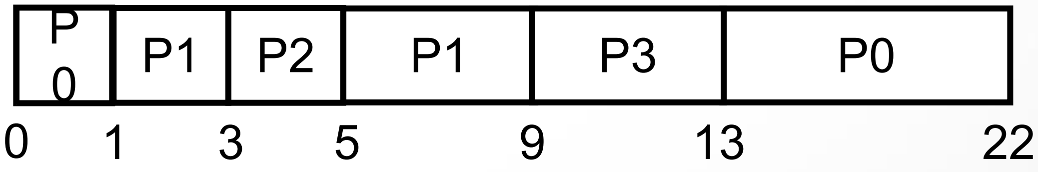
P3 (1)

- Decision Mode:
  - **Preemptive**: **When a new process arrives, its total time is compared to the current process remaining run time**.
  - If the **new process needs less time to finish than the current process, the current process is suspended and the new job is started**.
- Implementation:
  - This strategy can also be implemented by using **sorted FIFO queue**.
  - All processes in a queue are **sorted in ascending order on their remaining run time**.
  - When CPU becomes free, a process from the first position in a queue is selected to run.

Shortest Remaining Time Next (SRTN)

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

- Gantt Chart



- Average Turnaround Time: 10 ms
- Average Waiting Time: 4.5 ms

P0	P1	P2	P1	P3	P0

0 1 3 5 9 13 22

Process	Remaining Time
P1	6
P0	9

Process	Remaining Time
P0	9
P2	2
P1	4

Process	Remaining Time
P0	9
P1	4
P3	4

Process	Remaining Time
P0	9
P3	4

Process	Remaining Time
P0	9

Shortest Remaining Time Next (SRTN)

- Advantages
  - **Less waiting time.**
  - **Quite good response for short processes.**
- Disadvantages
  - It is **difficult to estimate time required** to complete execution.
  - **Starvation is possible for long process**. Long process may wait forever.
    - Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.
  - **Context switch overhead is there**.

Round Robin (RR)

Section – 4.4

Round Robin (RR)

- Selection criteria:
  - Each selected process is **assigned a time interval, called time quantum or time slice**.
  - Process is **allowed to run only for this time interval**.
  - Here, two things are possible:
  - First, **process is either blocked or terminated before the quantum has elapsed**. In this case the **CPU switching is done** and **another process is scheduled** to run.

Ready queue & Quantum = 3

P0 (2) Head | P1 (2) | P2 (6) | | | Tail P3 (1)

  - Second, **process needs CPU burst longer than time quantum**. In this case, process is **running at the end of the time quantum**.
  - Now, it will be **preempted** and **moved to the end of the queue**.
  - CPU will be **allocated to another process**.
  - Here, **length of time quantum is critical to determine**.

Ready queue & Quantum = 3

P0 (4) Head | P1 (2) | P2 (6) | | | Tail P3 (1)

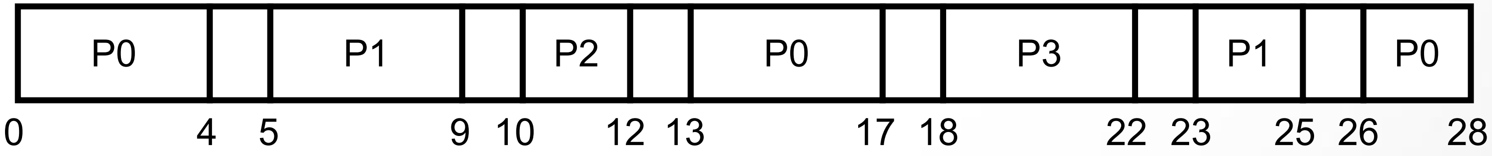
Round Robin (RR)

- Decision Mode:
  - **Preemptive**: **When a new process arrives, its total time is compared to the current process remaining run time**.
  - **Selection of new job is as per FCFS** scheduling algorithm.
- Implementation:
  - This strategy can be implemented by using **circular FIFO queue**.
  - If any process comes, or process releases CPU, or process is preempted. It is moved to the end of the queue.
  - When CPU becomes free, a process from the first position in a queue is selected to run.

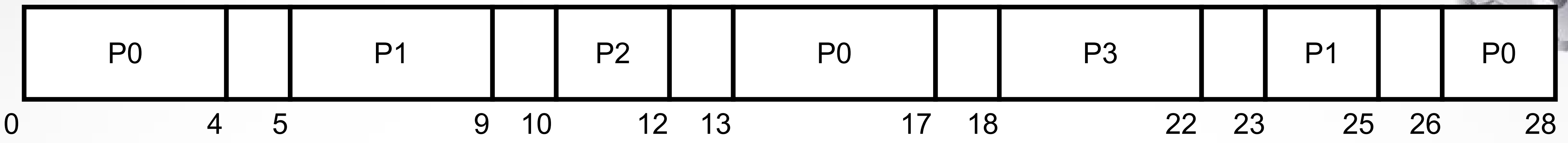
Round Robin (RR)

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - ΔT)
P0	0	10	28	28	18
P1	1	6	25	24	18
P2	3	2	12	9	7
P3	5	4	22	17	13

- Gantt Chart
  - Quantum time is 4 ms &
  - Context switch overhead is 1 ms



- Avg. Turnaround Time: 19.5 ms
- Avg. Waiting Time: 14 ms



Process	Remaining Time
P1	6
P2	2
P0	6

Process	Remaining Time
P2	2
P0	6
P3	4
P1	2

Process	Remaining Time
P0	6
P3	4
P1	2

Process	Remaining Time
P3	4
P1	2
P0	2

Process	Remaining Time
P1	2
P0	2

Ready Queue

Round Robin (RR)

- Advantages
  - **Simplest, fairest and most widely used algorithms.**
- Disadvantages
  - **Context switch overhead is there**.
  - **Determination of time quantum is too critical**.
    - If it is too short, it causes frequent context switches and lowers CPU efficiency.
    - If it is too long, it causes poor response for short interactive process.

Priority (Non-Preemptive Priority)

Section – 4.5.1

Non-Preemptive Priority

- Selection criteria:
  - The process, that **has highest priority, is served first**.

Ready queue

P0 (2) Head

P1 (8)	P2 (6)		

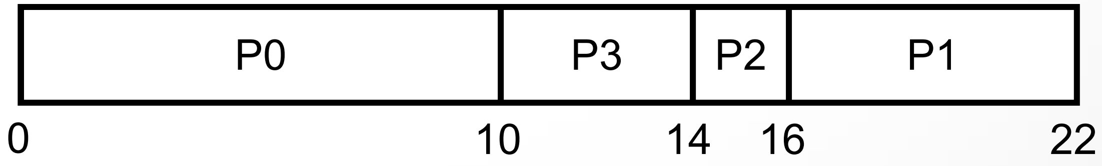
Tail P3 (7)

- Decision Mode:
  - **Non preemptive**: **Once a process is selected, it runs until it is blocked** for an I/O or some other event or it is terminated.
- Implementation:
  - This strategy can also be implemented by using **sorted FIFO queue**.
  - All processes in a queue are **sorted based on their priority with highest priority process at front end**.
  - When CPU becomes free, a process from the first position in a queue is selected to run.

Non-Preemptive Priority

Process	Arrival Time (T0)	Burst Time (ΔT)	Priority	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - ΔT)
P0	0	10	5	10	10	0
P1	1	6	4	22	21	15
P2	3	2	2	16	13	11
P3	5	4	0	14	9	5

- Gantt Chart (small values for priority means higher priority of a process)



- Avg. Turnaround Time: 13.25 ms
- Avg. Waiting Time: 7.75 ms

Non-Preemptive Priority

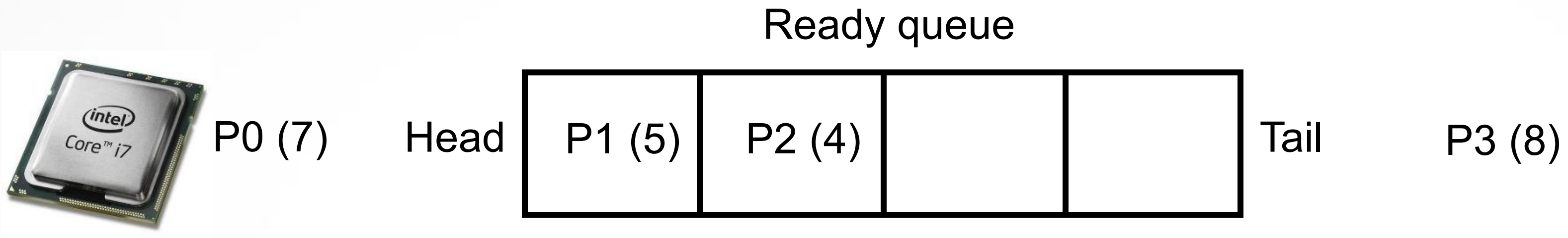
- Advantages
  - **Priority is considered so critical processes can get even better response time.**
- Disadvantages
  - **Starvation is possible for low priority processes.** It can **be overcome by using** technique called **'Aging'.**
  - **Aging: gradually increases the priority of processes that wait in the system for a long time.**

Priority (Preemptive Priority)

Section – 4.5.2

Preemptive Priority

- Selection criteria:
  - The process, that **has highest priority, is served first**.



- Decision Mode:
  - **Preemptive: When a new process arrives, its priority is compared with current process priority.**
  - If the **new process has higher priority than the current, the current process is suspended and new job is started.**
- Implementation:
  - This strategy can also be implemented by using **sorted FIFO queue**.
  - All processes in a queue are **sorted based on their priority with highest priority process at front end**.
  - When CPU becomes free, a process from the first position in a queue is selected to run.

Preemptive Priority

Process	Arrival Time (T0)	Burst Time (ΔT)	Priority	Finish Time (T1)	Turnaround Time (TAT = T1 - T0)	Waiting Time (WT = TAT - ΔT)
P0	0	10	5	22	22	12
P1	1	6	4	13	12	6
P2	3	2	2	5	2	0
P3	5	4	0	9	4	0

- Gantt Chart
  - small values means higher priority
- Avg. Turnaround Time: 10 ms
- Avg. Waiting Time: 4.5 ms

P0	P1	P2	P3	P1	P0
0	1	3	5	9	13

22

Proces s	Proces s	Proces s	Proces s	Proces s	Priorit y
P1	P0	P0	P0	P0	5
P0	P2	P1	P1	4	
	P1	P3	0		

Preemptive Priority

- Advantages
  - Priority is considered so critical processes can get even better response time.
- Disadvantages
  - Starvation is possible for low priority processes. It can be overcome by using technique called 'Aging'.
  - Aging: gradually increases the priority of processes that wait in the system for a long time.
  - Context switch overhead is there.

Highest Response Ratio Next (HRRN) CPU Scheduling

- The Highest Response Ratio Next is a non-preemptive algorithm.
- It is being recognised as one of the most optimal scheduling algorithms.
- As the name suggests here, the CPU time is allocated on the basis of the response ratio of all the available processes, where it selects the process that has the highest Response Ratio.
- The selected process will run till its execution is complete.

Response Ratio

The Response Ratio is calculated as:

Response Ratio = (WT + BT)/ BT

Here, WT - Waiting time, and

BT - Burst time.

Example:

Process	Arrival Time	Burst Time
P1	0	2
P2	2	6
P3	4	7
P4	5	3
P5	7	5

At time 0:

Available processes are: P1 (no other)

So the execution of P1 will start and will continue till its completion.

Process	Arrival Time	Burst Time	Completion Time
P1	0	2	2
P2	2	6	
P3	4	7	
P4	5	3	
P5	7	5	

At time 2:

Available processes are: P2.

Process	Arrival Time	Burst Time	Completion Time
P1	0	2	2
P2	2	6	8
P3	4	7	
P4	5	3	
P5	7	5	

At time 8:

Available processes are: P3, P4, and P5 (since the arrival time of all these three processes from the ready queue is within time = 8).

Since there are three processes in the ready queue, they have to be scheduled to get the CPU time. For this, we shall calculate the response ratio for each process P3, P4, and P5 using the formula given above.

Response ratio for P3 = [(8 - 4) + 7]/ 7 = 1.57

Response ratio for P4 = [(8 - 5) + 3]/ 3 = 2

Response ratio for P5 = [(8 - 7) + 5]/ 5 = 1.2

From above, Response ratio for P4 is the highest, so the process to be scheduled next for execution is P4.

Process	Arrival Time	Burst Time	Completion Time
P1	0	2	2
P2	2	6	8
P3	4	7	
P4	5	3	11
P5	7	5	

- **At time 11:**
- Available processes are: P3, and P5 (since the arrival time of all these two processes from the ready queue is within time = 11)
- Again, since there are two processes ready to get hold of the CPU time, they have to be scheduled and Response ratio calculation again for deciding the process to be executed first from among P3, and P5.
- Response ratio for P3= [(11 - 4) + 7)/ 7 = 2
- Response ratio for P5 = [(11 - 7) + 5)/ 5 = 1.8
- From above, Response ratio for P3 is the highest, so the process to be scheduled next for execution is P3.

Process	Arrival Time	Burst Time	Completion Time
P1	0	2	2
P2	2	6	8
P3	4	7	18
P4	5	3	11
P5	7	5	

- At time 18:
- Now, only one process, P5, is left after completion of P3.

Process	Arrival Time	Burst Time	Completion Time
P1	0	2	2
P2	2	6	8
P3	4	7	18
P4	5	3	11
P5	7	5	23

The final Gantt Chart will be:

Gantt Chart

P1	P2	P4	P3	P5
0 2	8	11	18	23

Thus, the final table is:

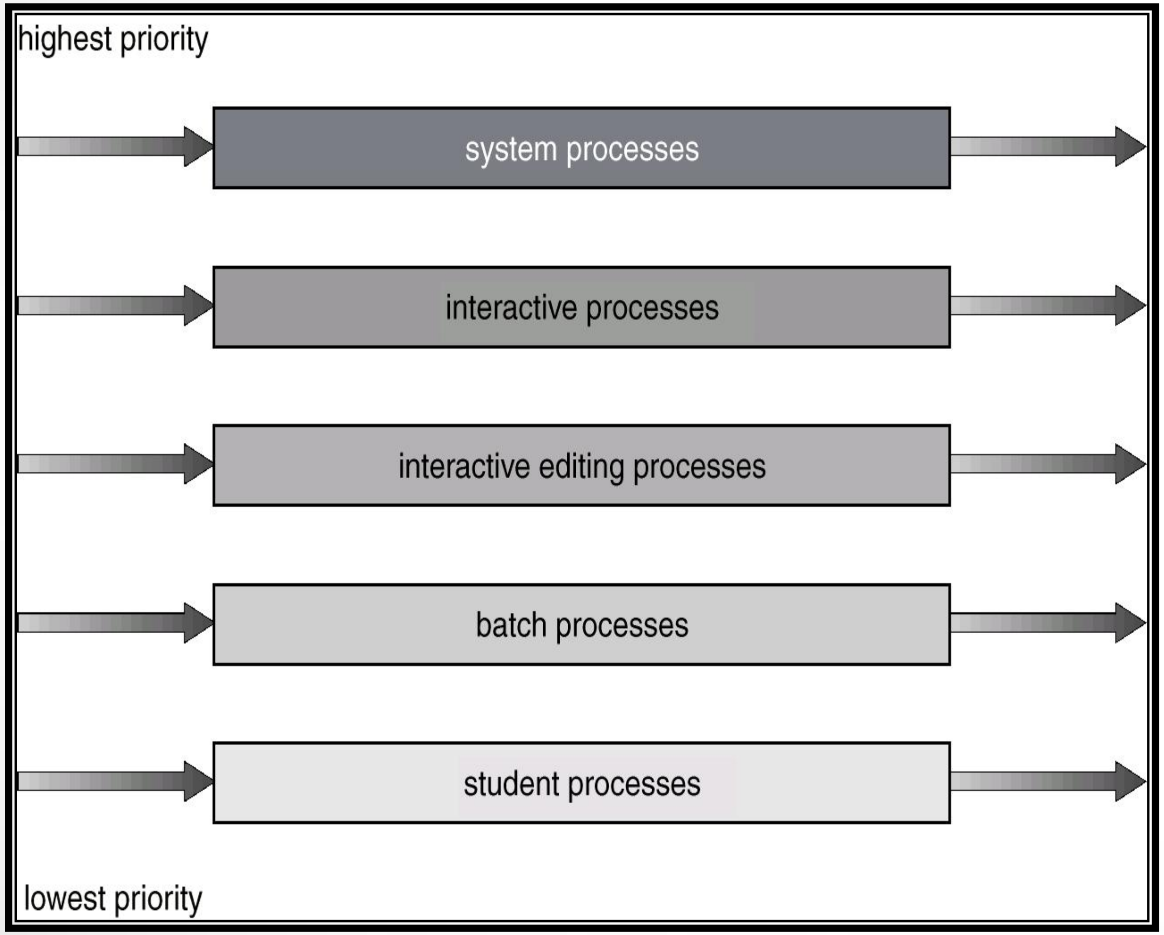
Process	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
P1	0	2	2	2	0	0 - 0 = 0
P2	2	6	8	6	0	2 - 2 = 0
P3	4	7	18	14	7	11 - 4 = 7
P4	5	3	11	6	3	8 - 5 = 3
P5	7	5	23	16	11	18 - 7 = 11

- Output from the table:
- Total Turn Around Time = 44
- Thus, Average Turn Around Time = 8.8
- Total Waiting Time = 21
- Thus, Average Waiting Time = 4.2
- Total Response Time = 21
- Thus, Average Response Time = 4.2

Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- Each queue has its own scheduling algorithm, e.g.,
  - 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
    - 80% to foreground in RR
    - 20% to background in FCFS

Multilevel Queue Scheduling



- ML queue, 2 levels
  - RR @ 10 units
  - FCFS
  - RR gets priority over FCFS
-

Proc	Arrival	Burst	Queue
P_1	0	12	FCFS
P_2	4	12	RR
P_3	8	8	FCFS
P_4	20	10	RR

- Non-preemptive and preemptive

Multilevel Feedback Queue

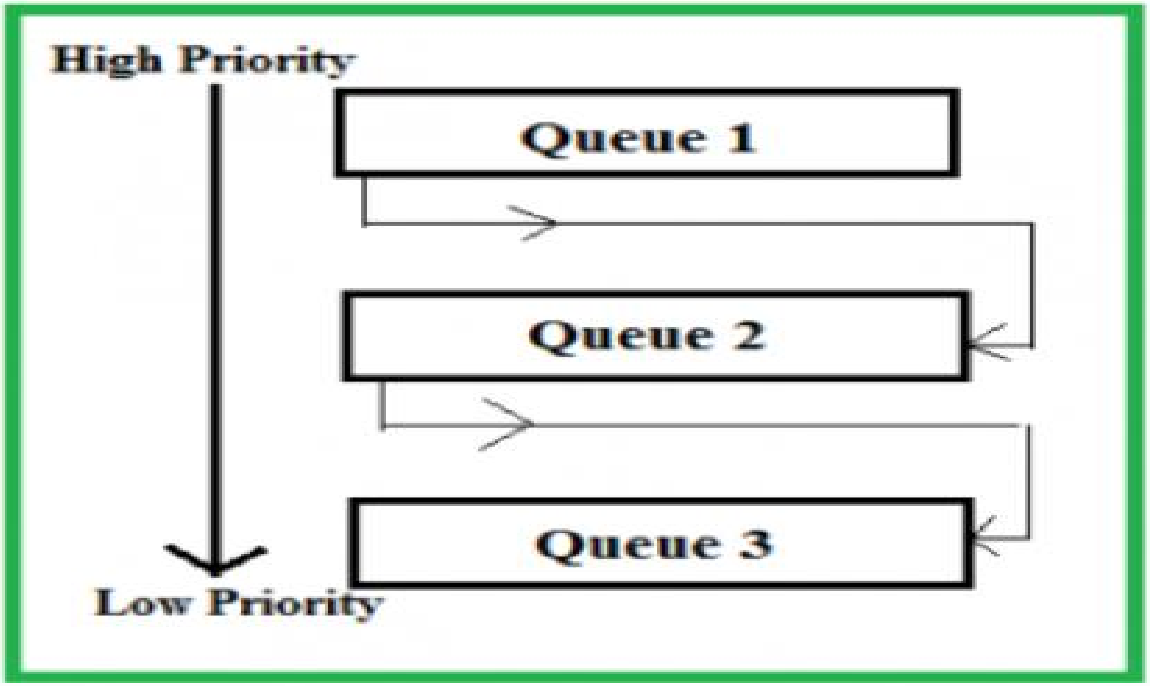
- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine which queue a process will enter when that process needs service
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process

Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling

- **Multilevel Feedback Queue Scheduling (MLFQ)** CPU Scheduling is like Multilevel Queue(MLQ) Scheduling but in this processes can move between the queues. And thus, much more efficient than multilevel queue scheduling.
- **Advantages of Multilevel Feedback Queue Scheduling:**
- It is more flexible.
- It allows different processes to move between different queues.
- It prevents starvation by moving a process that waits too long for the lower priority queue to the higher priority queue.
- **Disadvantages of Multilevel Feedback Queue Scheduling:**
- For the selection of the best scheduler, it requires some other means to select the values.
- It produces more CPU overheads.
- It is the most complex algorithm.

Cont..

- **Multilevel feedback queue scheduling**, however, allows a process to move between queues. Multilevel Feedback Queue Scheduling keeps analyzing the behavior (time of execution) of processes and according to which it changes its priority.



Cont..

- **Example:** Consider a system that has a CPU-bound process, which requires a burst time of 40 seconds. The multilevel Feed Back Queue scheduling algorithm is used and the queue time quantum ‘2’ seconds and in each level it is incremented by ‘5’ seconds. Then how many times the process will be interrupted and in which queue the process will terminate the execution?
- **Solution:**
- Process P needs 40 Seconds for total execution.
- At Queue 1 it is executed for 2 seconds and then interrupted and shifted to queue 2.
- At Queue 2 it is executed for 7 seconds and then interrupted and shifted to queue 3.
- At Queue 3 it is executed for 12 seconds and then interrupted and shifted to queue 4.
- At Queue 4 it is executed for 17 seconds and then interrupted and shifted to queue 5.
- At Queue 5 it executes for 2 seconds and then it completes.
- Hence the process is interrupted 4 times and completed on queue 5.

Performance Analysis of the CPU scheduling algorithms

- **Performance of Round-Robin scheduling:**
- Performance of Round-Robin scheduling depends on time quantum.
- When TQ is small, then there is more context switch overhead.
- Larger TQ makes the system less responsive.
- The value of TQ should neither be too large, wherein it degenerates to work like FCFS, nor the TQ should be too small , where the efficiency of the scheduler tends to become almost nearing to zero.

Example:

Consider the following dataset:

P.No.	A.T	B.T
1	0	2
2	0	4
3	0	5

What will be the result:

(a) When time quantum is very large.
TQ = 10

(b) When time quantum is very small.
TQ = 0.1 (Let $\delta = 2$)

Cont..

Solution:

(a) RQ $\boxed{P_1 ; P_2 ; P_3}$

CPU

P_1	P_2	P_3

0 2 6 11

Since, (time quantum > BT of all processes)

So, Round Robin behaves as FCFS.

(b) TQ = 0.1 δ = 2

T

δ	P_1	δ	P_2	δ	P_3

0 2 2.1 4.1 4.2 6.2 6.3

- During this time 'T', CPU performed some useful work and some overhead activities.
- Processor was busy only for .3 units of time

$$\text{Efficiency} = \frac{.3}{6.3} \simeq \frac{1}{20}$$

- Therefore, only 5% of time processor was doing useful activity.
- So, when TQ becomes very small efficiency of system becomes almost zero.

Cont..

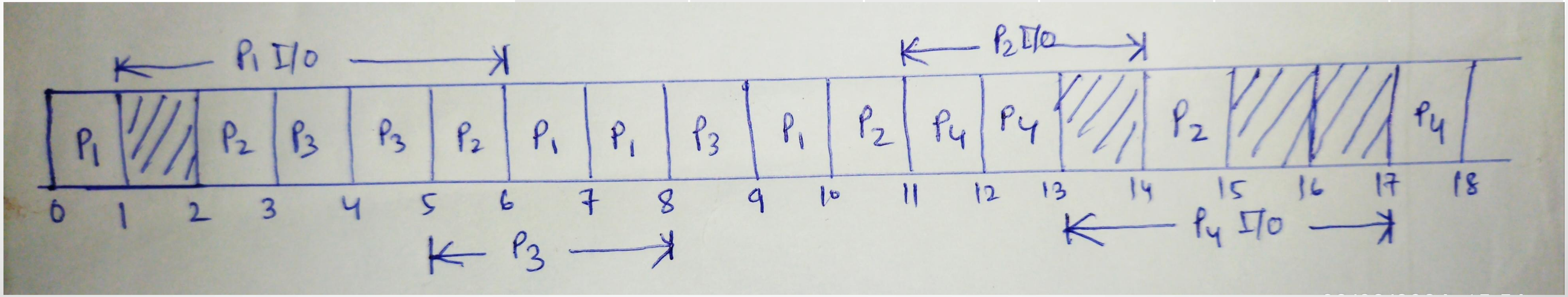
Comparison Table of Scheduling Algorithms

	FCFS	Round Robin	SJF	SRTF	HRRN	Feedback
Selection function	Max [w]	Constant	min [s]	min [s – e]	$\max\left(\frac{w+s}{s}\right)$	–
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, espacially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

I/O event based numerical on Scheduling algorithms

- **Consider the following processes with AT and BT (CPU+IO):**
- **Mode: Preemptive**
- **Criteria: Priority Based**

Process	AT	Priority	CPU	IO	CPU
P1	0	2	1	5	3
P2	2	3	3	3	1
P3	3	1	2	3	1
P4	3	4	2	4	1

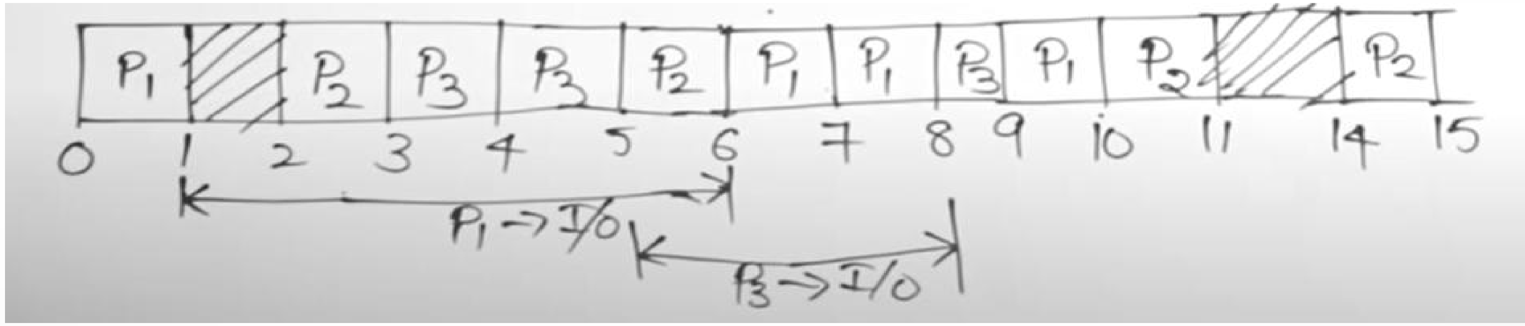


Cont..

- **Consider the following processes with AT and BT (CPU+IO):**

Process	AT	Priority	CPU	IO	CPU
P1	0	2	1	5	3
P2	2	3	3	3	1
P3	3	1	2	3	1

- **Calculate average TAT and WT using preemptive priority scheduling.**



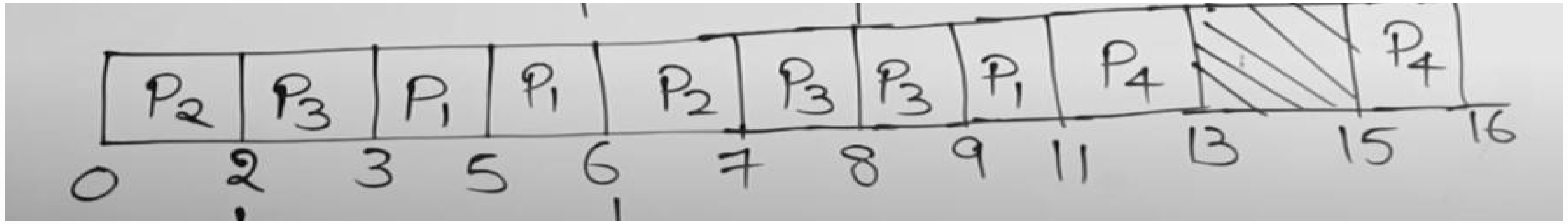
- **Avg. TAT = (10+13+6)/3 = 9.67**
- **Avg. WT = (6+9+3)/3 = 6**

Cont..

- **Consider the following processes with AT and BT (CPU+IO):**

Process	AT	CPU	IO	CPU
P1	0	3	2	2
P2	0	2	4	1
P3	2	1	3	2
P4	5	2	2	1

- **Calculate average TAT and WT using SRTF scheduling.**



- **Avg. TAT = (11+7+7+11)/4 = 9**
- **Avg. WT = (6+4+4+8)/4 = 5.5**

Real Time Operating System

Section – 5

Real Time Operating System

- A real-time system is one in which **time plays an essential role**.
- Real time computing may be defined as that type of computing in which the **correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced**.
- Some of the real-time systems are **patient monitoring in a hospital intensive-care unit, the autopilot in an aircraft and robot control in an automated factory**.
- In all these cases, having the right answer but having it too late is often just as bad as not having it at all.
- Real time task may be classified as hard and soft.
  – A **hard real time task is one that must meet its deadline**; otherwise it will cause unacceptable damage or a fatal error to the system.
  – A **soft real time task has an associated deadline that is desirable but not mandatory**; it will not cause unacceptable damage or a fatal error on missing deadline.

Real Time Operating System

- The events that a real-time system may have to respond to can be further categorized as **periodic (occurring at regular intervals)** or **aperiodic (occurring unpredictably)**.
- A system may have to respond to multiple periodic event streams. Depending on how much time each event requires for processing, it may not even be possible to handle them all.
- Real-time scheduling algorithms can be **static or dynamic**.
  - Static: The former make their scheduling decisions before the system starts running.
  - Dynamic: The latter make their scheduling decisions at run time.
  - Static scheduling only works when there is per-fect information available in advance about the work to be done and the deadlines that have to be met.
  - Dynamic scheduling algorithms do not have these restrictions.

Exercise

1. Five batch jobs A to E arrive at same time. They have estimated running times 10,6,2,4 and 8 minutes. Their priorities are 3,5,2,1 and 4 respectively with 5 being highest priority. For each of the following algorithm determine mean process turnaround time. Ignore process swapping overhead. Quantum time is 2 minute.
   - Round Robin, Priority Scheduling, FCFS, SJF.

2. Suppose that the following processes arrive for the execution at the times indicated. Each process will run the listed amount of time. Assume preemptive scheduling.

Process	Arrival Time (ms)	Burst Time (ms)
P1	0.0	8
P2	0.4	4
P3	1.0	1

   - What is the turnaround time for these processes with Shortest Job First scheduling algorithm?

Exercise

3. Consider the following set of processes with length of CPU burst time given in milliseconds.

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

- Assume arrival order is: P1, P2, P3, P4, P5 all at time 0 and a smaller priority number implies a higher priority. Draw the Gantt charts illustrating the execution of these processes using preemptive priority scheduling.

Questions asked in examination.

1. Define term Scheduler, Scheduling and Scheduling Algorithm with example.
2. Define terms. 1) Throughput 2) Waiting Time 3) Turnaround Time 4) Response Time 5) Granularity 6) Short Term Scheduler 7) CPU Utilization
3. What is scheduler? Explain queuing diagram representation of process scheduler with figure.
4. Write various scheduling criteria.
5. Consider Five Processes P1 to P5 arrived at same time. They have estimated running time 10, 2, 6, 8 and 4 seconds, respectively. Their Priorities are 3, 2, 5, 4 and 1, respectively with 5 being highest Priority. Find the average turnaround time and average waiting time for Round Robin (quantum time=3) and Priority Scheduling algorithm.
6. Consider the processes P1, P2, P3, P4 with burst time is 21, 3, 6 and 2 respectively, arrives for execution in the same order, with arrival time 0, draw GANTT chart and find the average waiting time using the FCFS, SJF, SRTN and Round Robin (quantum time=3) scheduling algorithm.