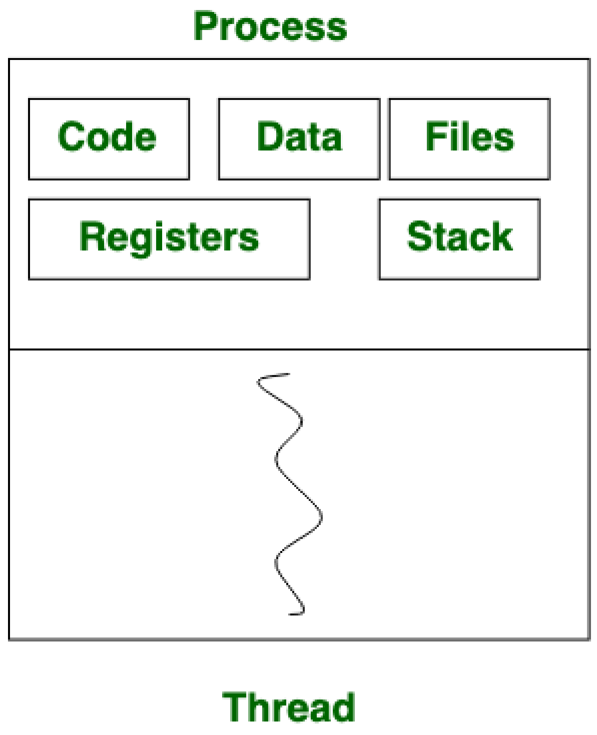
1). Differentiate between process and threads ?

S.NO	Process	Thread
1.	Process means any program is in execution.	Thread means a segment of a process.
2.	The process takes more time to terminate.	The thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
6.	Multiprogramming holds the concepts of multi-process.	We don’t need multi programs in action for multiple threads because a single process consists of multiple threads.
7.	The process is isolated.	Threads share memory.
8.	The process is called the heavyweight process.	A Thread is lightweight as each thread in a process shares code, data, and resources.
9.	Process switching uses an interface in an operating system.	Thread switching does not require calling an operating system and causes an interrupt to the kernel.
10.	If one process is blocked then it will not affect the execution of other processes	If a user-level thread is blocked, then all other user-level threads are blocked.
11.	The process has its own Process Control Block,	Thread has Parents’ PCB, its own Thread Control Block, and Stack and common

S.NO	Process	Thread
	Stack, and Address Space.	Address space.
12.	Changes to the parent process do not affect child processes.	Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process.
13.	A system call is involved in it.	No system call is involved, it is created using APIs.
14.	The process does not share data with each other.	Threads share data with each other.



2). Explain fork() system call with example.

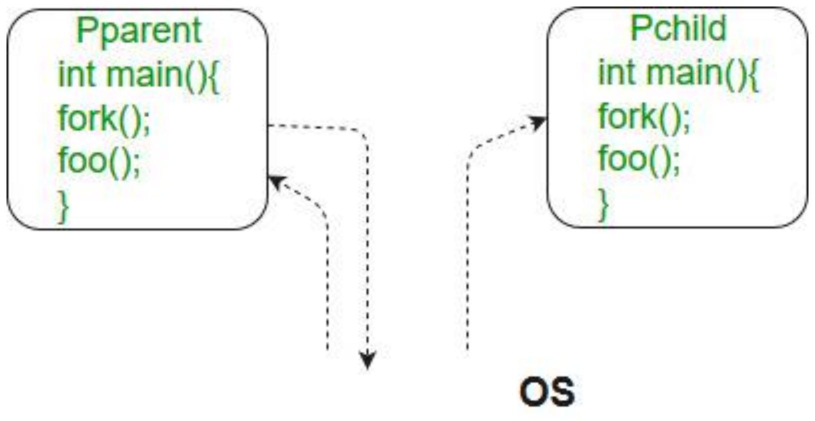
→Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the →fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

→It takes no parameters and returns an integer value. Below are different values returned by fork().

→Negative Value: creation of a child process was unsuccessful.

→Zero: Returned to the newly created child process.

→Positive value: Returned to parent or caller. The value contains process ID of newly created child process.



Ex :

int main()

{

   fork();

```
    fork();

    fork();

    printf("hello\n");

    return 0;

}
```

Explanation :

The number of times ‘hello’ is printed is equal to number of process created. Total Number of Processes = 2n, where n is number of fork system calls. So here n = 3, 23 = 8

```
fork ();   // Line 1

fork ();   // Line 2

fork ();   // Line 3
```

```
     L1              // There will be 1 child process

   /    \            // created by line 1.

  L2      L2         // There will be 2 child processes

 /  \   /  \         //  created by line 2

L3  L3  L3  L3  // There will be 4 child processes

                     // created by line 3
```

→So there are total eight processes (new child processes and one original process).

→if we want to represent the relationship between the processes as a tree hierarchy it would be the following:

The main process: P0

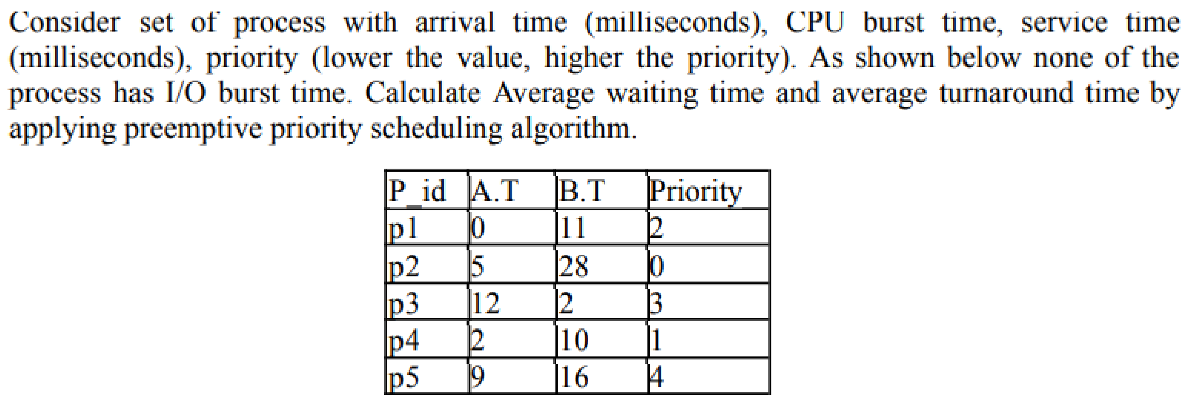
Processes created by the 1st fork: P1

Processes created by the 2nd fork: P2, P3

Processes created by the 3rd fork: P4, P5, P6, P7

```
          P0

       /  |  \

     P1    P4   P2

   / \        \

  P3    P6        P5

 /

P7
```

3)

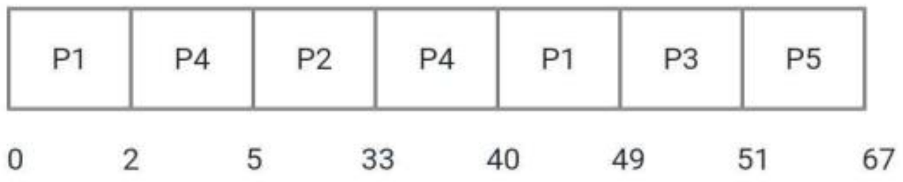


Consider set of process with arrival time (milliseconds), CPU burst time, service time (milliseconds), priority (lower the value, higher the priority). As shown below none of the process has I/O burst time. Calculate Average waiting time and average turnaround time by applying preemptive priority scheduling algorithm.

P_id	A.T	B.T	Priority
p1	0	11	2
p2	5	28	0
p3	12	2	3
p4	2	10	1
p5	9	16	4

Ans:

Gantt chart:



Process Table:

Process	Arrival Time	Burst Time	Priority	Completion time (CT)	Turnaround time (TAT) TAT= CT – AT	Waiting time (WT) WT = TAT-BT
P_1	0	11	2	49	49	38
P_2	5	28	0	33	28	0
P_3	12	2	3	51	39	37
P_4	2	10	1	40	38	28
P_5	9	16	4	67	58	42

Average waiting time = $\frac{\sum waiting\ time\ of\ all\ the\ processes}{number\ of\ processes}$

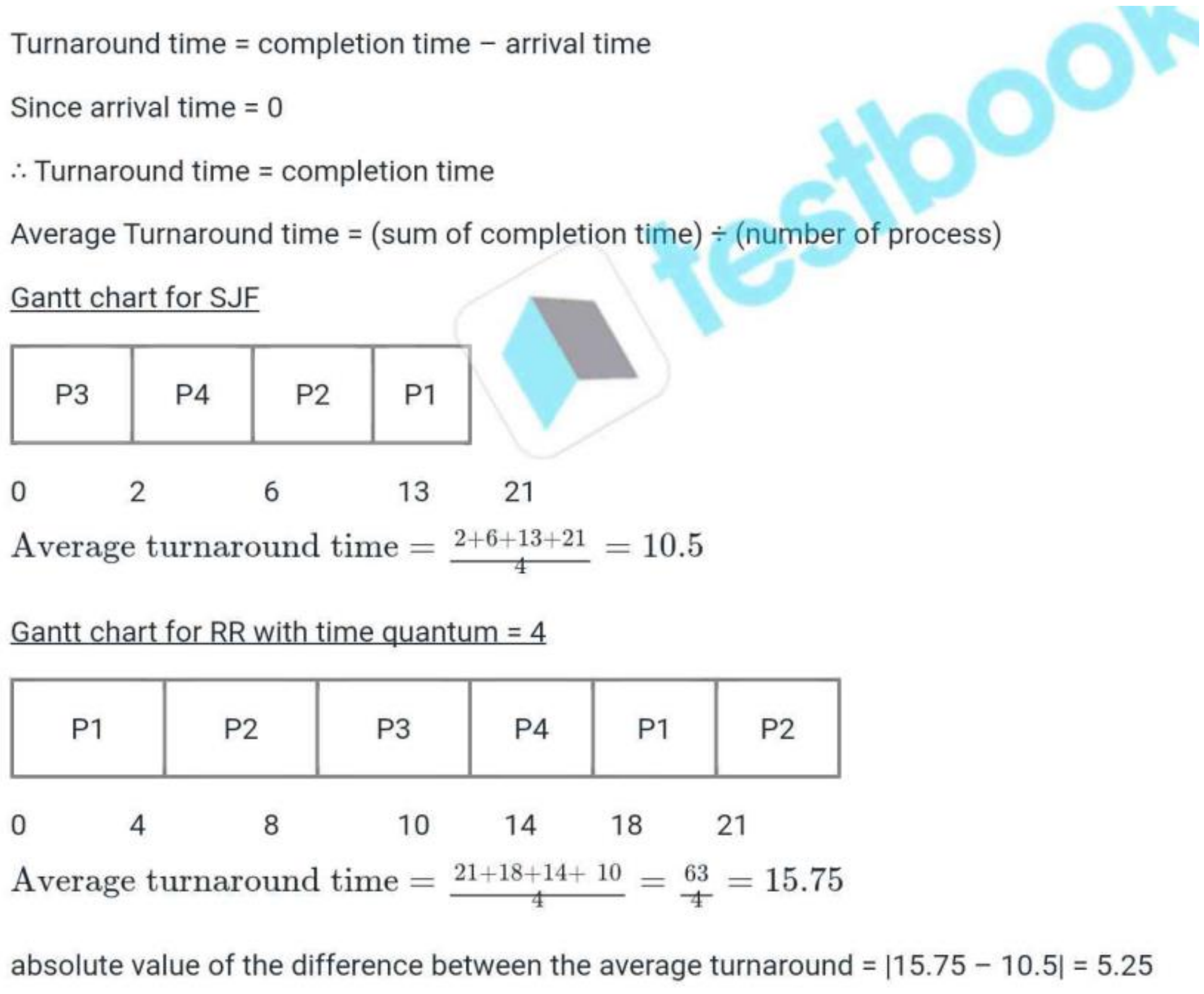
$$= \frac{38+0+37+28+42}{5} = 29 \text{ milliseconds}$$

4).

Consider the following set of processes, assumed to have arrived at time 0. Consider the CPU scheduling algorithms Shortest Job First (SJF) and Round Robin (RR). For RR, assume that the processes are scheduled in the order P_1, P_2, P_3, P_4.

Processes	P_1	P_2	P_3	P_4
Burset Time (in ms)	8	7	2	4

Ans:



Turnaround time = completion time − arrival time

Since arrival time = 0

∴ Turnaround time = completion time

Average Turnaround time = (sum of completion time) ÷ (number of process)

Gantt chart for SJF

P3	P4	P2	P1

0 2 6 13 21

$$\text{Average turnaround time} = \frac{2+6+13+21}{4} = 10.5$$

Gantt chart for RR with time quantum = 4

P1	P2	P3	P4	P1	P2

0 4 8 10 14 18 21

$$\text{Average turnaround time} = \frac{21+18+14+10}{4} = \frac{63}{4} = 15.75$$

absolute value of the difference between the average turnaround = |15.75 − 10.5| = 5.25

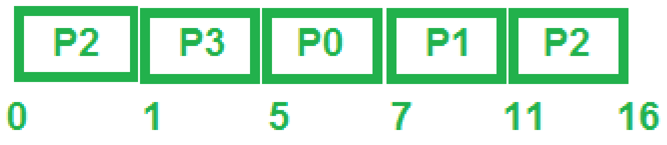
5). What is preemptive and non-preemptive scheduling? Under which circumstances CPU scheduling take place?

1. Preemptive Scheduling:

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

→Algorithms based on preemptive scheduling are: Round Robin (RR),Shortest Remaining Time First (SRTF), Priority (preemptive version), etc.

Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4

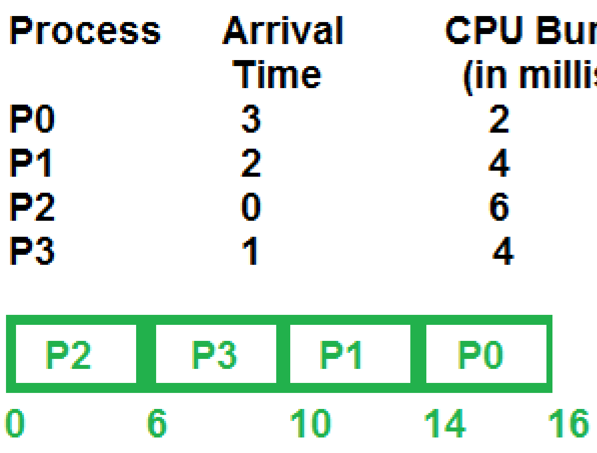


Preemptive Scheduling

2. Non-Preemptive Scheduling:

→Non-preemptive Scheduling is used when a process terminates, or a process switches from running to the waiting state. In this scheduling, once the resources (CPU cycles) are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state. In the case of non-preemptive scheduling does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then it can allocate the CPU to another process.

→Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non preemptive) and Priority (non preemptive version), etc.



Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4

Non-Preemtive Scheduling

CPU scheduling decisions may take place when a process:

1.Switches from running state to waiting state

2.Switches from running state to ready state

3.Switches from waiting state to ready state

4.When a process terminates

For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.

There is a choice, however, for situations 2 and 3.

6). Discuss various scheduling criteria ?

→**CPU utilization** – keep the CPU as busy as possible

→**Throughput** – number of processes that complete their execution per time unit (e.g., 5 per second)

→**Turnaround time** – amount of time to execute a particular process (Submission of Process to Completion)i.e.(Waiting to get into memory+ Waiting in ready queue+ Executing on the CPU+ Doing IO)

→**Waiting time** – total amount of time a process has been waiting in the ready queue (Sum of periods spent waiting in the ready queue)

→**Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

7). Define the following

a)Race condition:

Race Condition or Race Hazard is an undesirable situation of software, electronics, or other systems. When the output of the system or program depends on the sequence or timing of other uncontrolled events, this condition is called Race Condition.

This condition occurs mainly in the logic circuits, distributed and multithreaded software programs.

A race condition is categorized as either a critical or non-critical race condition. The critical race conditions are those conditions that occur when the order of the internal variable regulates the last state of the machine. On the other hand, the non-critical race conditions are those conditions which occur when the order of internal variables does not regulate the last state of the machine.

B). Critical Section :

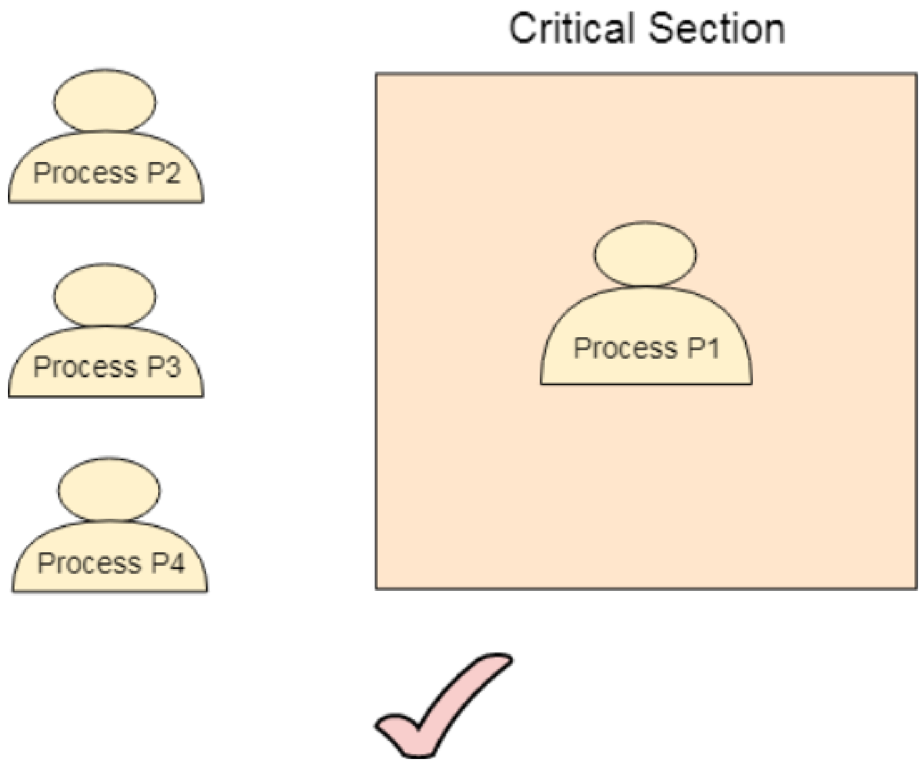
Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

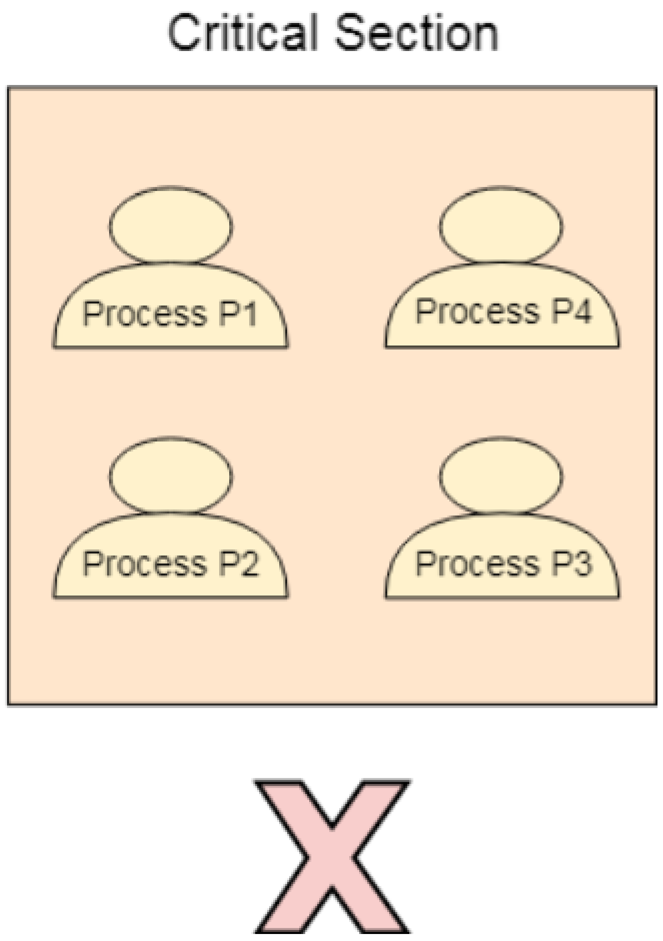
The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

C). Mutual Exclusion :

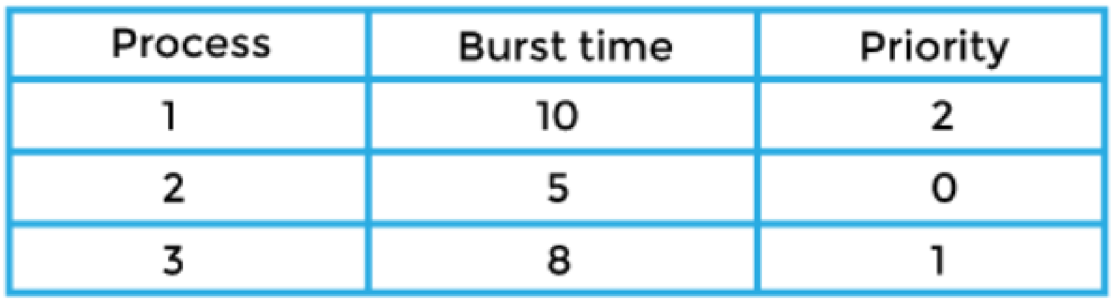
Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.



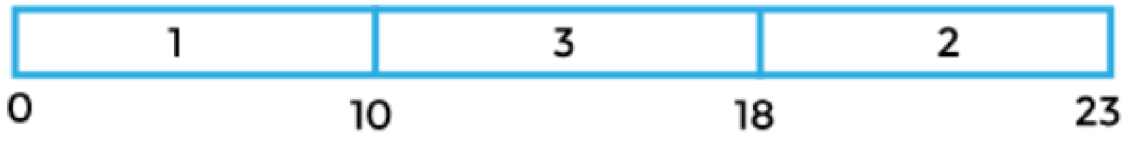


D).Starvation :

Starvation or indefinite blocking is a phenomenon associated with the Priority scheduling algorithms. A process that is present in the ready state and has low priority keeps waiting for the CPU allocation because some other process with higher priority comes with due respect time. Higher-priority processes can prevent a low-priority process from getting the CPU.



Process	Burst time	Priority
1	10	2
2	5	0
3	8	1



For example, the above image process has higher priority than other processes getting CPU earlier. We can think of a scenario in which only one process has very low priority (for example, 127), and we are giving other processes high priority. This can lead to indefinitely waiting for the process for CPU, which is having low-priority, which leads to **Starvation**.

E). 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.

Kernel loads first into memory when an operating system is loaded and remains into memory until operating system is shut down again. It is responsible for various tasks such as disk management, task management, and memory management.

Kernel has a process table that keeps track of all active processes

• Process table contains a per process region table whose entry points to entries in region table.

Kernel loads an executable file into memory during 'exec' system call'.

(f)System call :

A system call is a method for a computer program to request a service from the kernel of the operating system on which it is running. A system call is a method of interacting with the operating system via programs. A system call is a request from computer software to an operating system's kernel.

The **Application Program Interface (API)** connects the operating system's functions to user programs. It acts as a link between the operating system and a process, allowing user-level programs to request operating system services. The kernel system can only be accessed using system calls. System calls are required for any programs that use resources.

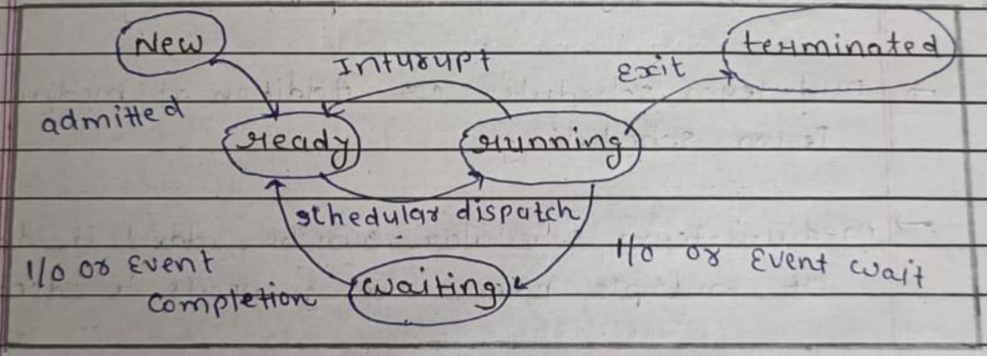
8). Explain process state diagram and various types of schedulers.

* Process state

→ As a process Executes, it changes State.

→ New :- The process is being created.
→ running: Instructions are Executed.
→ waiting :- The process is waiting for some Event to occur
→ ready :- The process is waiting to be assigned to a processor
→ Terminate :- The process has finished Execution.

* Diagram of process State :-

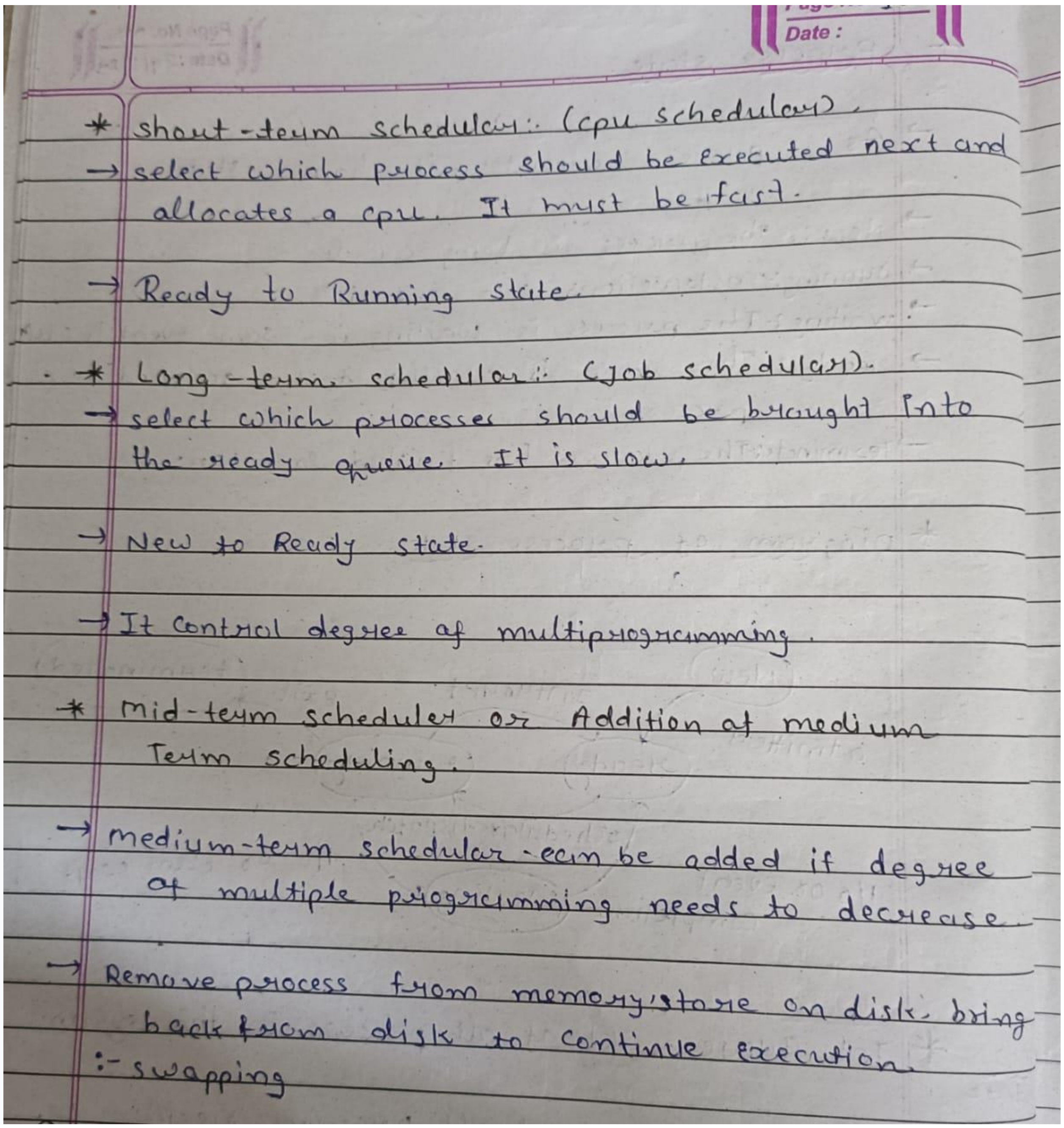


New State * Job Queue :- Set of all process in the system.

ready State * Ready Queue: set of all processes residing in main memory, ready and waiting to Execute

waiting State * Device Queue :- set of process waiting for an I/O device.

processes migrate among the various queues.



Date :

* short-term schedular :- (cpu schedular).
→ select which process should be executed next and allocates a cpu. It must be fast.

→ Ready to Running state.

* Long-term schedular :- (job schedular).
→ select which processes should be brought into the ready queue. It is slow.

→ New to Ready state.

→ It control degree of multiprogramming.

* mid-term scheduler or Addition of medium Term scheduling.

→ medium-term schedular can be added if degree of multiple programming needs to decrease.

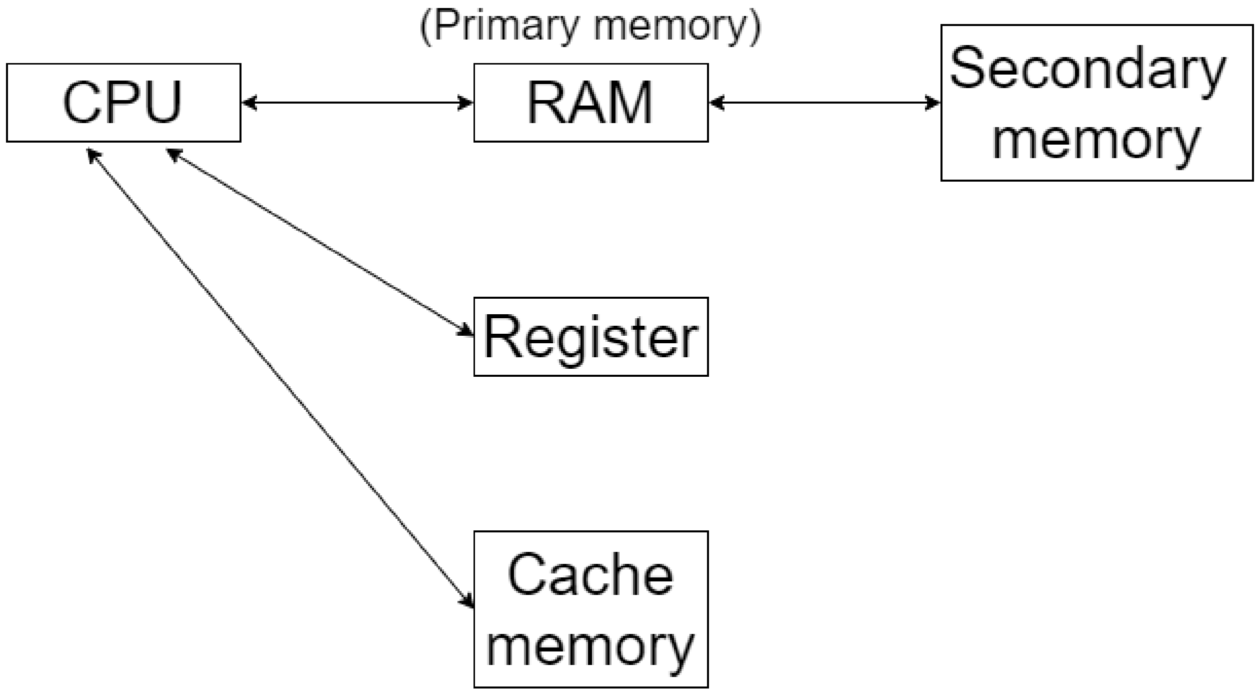
→ Remove process from memory, store on disk, bring back from disk to continue execution.
:- swapping

9). Explain the function of memory management & how the degree of multiprogramming dependson memory management

Memory management:

Memory management in operating system means that it is a method/ functionality to manage different types of memory.

Mainly memory management is method of managing primary memory or RAM.



Degree of multiprogramming :

→CPU is not connected with secondary memory directly because the speed of secondary memory is very low in comparison of CPU.

→All the process/ programs are present in secondary memory, in order to execute them we have to load them in RAM

→From RAM they will be sent to CPU and get executed.

→This is called degree of multiprogramming.

→Multiprogramming means, try to place more and more processes from ---→secondary memory to primary memory. Because then only the utilization of CPU will be at satisfactory level.

If we increase the number of process in RAM than CPU utilization will be more and degree of multiprogramming will be also high.

10). . **What is address binding? Describe all its types with reference to absolute, relocatable, logical, virtual & physical addresses.**

The Address Binding refers to the mapping of computer instructions and data to physical memory locations. Both logical and physical addresses are used in computer memory. It assigns a physical memory region to a logical pointer by mapping a physical address to a logical address known as a virtual address. It is also a component of computer memory management that the OS performs on behalf of applications that require memory access.

Types of Address Binding in Operating System

There are mainly three types of an address binding in the OS. These are as follows:

1). Compile Time Address Binding

2). Load Time Address Binding

3). Execution Time or Dynamic Address Binding

Compile time address binding

Compile time address binding allocates a space in the memory to the machine code of the computer when the program is compiled.
In other words, if the compiler is responsible for address binding, then it is known as compiler time address binding. The compiler interacts with the OS to perform the address binding.
It is done before loading the program in the computer memory.

Load time address binding
Load time address binding is done after loading the program in the memory.
It is done by the OS memory manager or the loader.

Execution time address binding
Execution time address binding is mostly only applicable to variables and is usually used for script binding, which generally doesn't get compiled.
It is postponed even after loading the program into the memory. The program keeps changing the memory location until the time of program execution.
Dynamic address binding is not done until program execution.
In majority of the Operating System, like Windows and Linux, dynamic address binding, dynamic loading, and dynamic linking is used.

11). What is the role of relocation registers in MMU? How is memory protection used between one process to another?

→ Relocation registers are used by the Memory Management Unit (MMU) in a computer system to translate virtual addresses used by programs into physical addresses in memory. They contain a base address that is added to the virtual address during translation to determine the physical address in memory.

→The value stored in the relocation register is determined by the operating system at runtime and is typically different for each process that is running. Relocation registers enable the MMU to provide a protected memory environment and efficient memory management for programs running on a computer system.

→ Memory protection is used in a multi-process system to ensure that one process cannot access or modify memory belonging to another process. Each process is assigned its own virtual address space, which is protected by the operating system using hardware memory protection mechanisms such as memory protection bits and address space layout randomization (ASLR).

→ Memory protection bits are set in the page tables used by the Memory Management Unit (MMU) to determine the access rights granted to each process for a given memory page. ASLR randomly assigns the base address of a process's virtual address space to make it more difficult for attackers to locate and exploit specific memory locations. These mechanisms are used to maintain the security and stability of modern operating systems.

12). Explain all contiguous memory allocation techniques with its advantages & disadvantages.

What is Contiguous Memory Allocation?

It is the type of ***memory allocation method.*** When a process requests the memory, a single contiguous section of memory blocks is allotted depending on its requirements.

It is completed by partitioning the memory into fixed-sized partitions and assigning every partition to a single process. However, it will limit the degree of multiprogramming to the number of fixed partitions done in memory.

This allocation also leads to internal fragmentation. For example, suppose a fixed-sized memory block assigned to a process is slightly bigger than its demand. In that case, the remaining memory space in the block is referred to as internal fragmentation.

Advantages

1. It is simple to keep track of how many memory blocks are left, which determines how many more processes can be granted memory space.
2. The read performance of contiguous memory allocation is good because the complete file may be read from the disk in a single task.
3. The contiguous allocation is simple to set up and performs well.

Disadvantages

1. Fragmentation isn't a problem because every new file may be written to the end of the disk after the previous one.
2. When generating a new file, it must know its eventual size to select the appropriate hole size.
3. When the disk is filled up, it would be necessary to compress or reuse the spare space in the holes.

What is Non-Contiguous Memory Allocation?

It allows a process to obtain multiple memory blocks in various locations in memory based on its requirements. The non-contiguous memory allocation also reduces memory wastage caused by ***internal*** and ***external*** fragmentation because it uses the memory holes created by internal and external fragmentation.

The two methods for making a process's physical address space non-contiguous are paging and segmentation. Non-contiguous memory allocation divides the process into blocks ***(pages or segments)*** that are allocated to different areas of memory space based on memory availability.

Non-contiguous memory allocation can decrease memory wastage, but it also raises address translation overheads. As the process portions are stored in separate locations in memory, the memory execution is slowed because time is consumed in address translation.

Advantages

1. It has the advantage of reducing memory waste, but it increases overhead because of the address translation.
2. It slows down the memory execution because time is consumed in address translation.

Disadvantages

1. The downside of this memory allocation is that the access is slow because you must reach the other nodes using pointers and traverse them.

13) **Consider five memory partitions of size 100 KB, 500 KB, 200 KB, 450 KB and 600 KB in the same order. If a sequence of requests for blocks of size 212 KB, 417 KB, 112 KB and 426 KB in the same order come, then which of the following algorithms makes the efficient use of memory? (A)Best fit algorithm (B) First fit algorithm (C) Next fit algorithm (D)Both next fit and best fit results in the same**

Answer is C.

100 KB	500 KB	200 KB	450 KB	600 KB

	212	112	417	426
First fit :-	⇓	⇓	⇓	⇓
ext. frag ⇒	288 kB	78 kB	33 kB	174 kB

Total = 573 kB (wastage)

Best fit :-

500 kB	200 kB	450 kB	600 kB
417	112	212	426
⇓	⇓	⇓	⇓
83 kB	88 kB	238 kB	174 kB

Total = 573 kB (wastage)

Next fit :-

500 kB	200 kB	450 kB	600 kB
212 ✓		417 ✓	112 ✓ | 426 ✓
⇓		⇓	⇓
288 kB		33 kB	62 kB

Total ⇒ 383 KB (wastage) $\longrightarrow$ Minimum in all