1) What is an Operating System? Discuss the role/functions of OS as a resource manager.

- An Operating System is a program that acts as an intermediate/interface between a user of a computer and the computer hardware.
- It manages hardware resources and provides various services to users
- It provides a user-friendly environment in which a user may *easily* develop and execute programs.
- Otherwise, hardware knowledge would be mandatory for computer programming.
- So, it can be said that an OS *hides the complexity* of hardware from uninterested users.
- *To manage Resources, OS performs many tasks such as:*
  - Install drivers required for input and output, memory, and power.
  - Coordination among peripherals.
- A computer is a set of resources that perform the functions such as storing, process, and transferring the data.
- If a computer system is used by multiple application(or users),then they will compete for these resources.
- OS is responsible for managing the resources to control this functions.

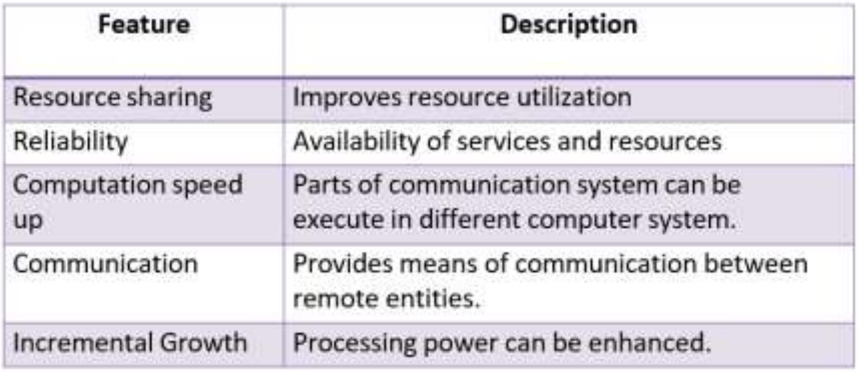
2) Identify various types of operating system and describe it with suitable example.

a) Multi-programming OS
- The concurrent residency of more than one program in the main memory is called as multiprogramming.
- Number of processor: one
- One process is executed at a time.

b) Multi-Tasking OS
- The execution of more than one task simultaneously is called as multitasking.
- Number of processor: one
- One by one job is being executed at a time.

c) Multi-processing OS
- The availability of more than one processor per system, which can execute several set of instructions in parallel is called as multiprocessing.
- Number of processor: more than one
- More than one process can be executed at a time.

d) Distributed OS
- A Distributed system is collection of independent computers which are connected through network.
- Examples :- **Web Search Engines:**
  - Major growth industry in the last decade.
  - 10 billion per month for global number of searches.
  - e.g. Google distributed infrastructure



Feature	Description
Resource sharing	Improves resource utilization
Reliability	Availability of services and resources
Computation speed up	Parts of communication system can be execute in different computer system.
Communication	Provides means of communication between remote entities.
Incremental Growth	Processing power can be enhanced.

e) Real Time OS

- A system is said to be Real Time if it is required to complete it’s work & deliver it’s services on time.
- A real time OS defines the completion of job within the
- *rigid time constraints* otherwise job looses its meaning.
- Example – Flight Control System,Air line reservation system.
- All tasks in that system must execute on time.

➢ **The Real Time OS is of two types:**

- Hard Real Time OS
- Soft Real Time OS

▪ **Hard Real time**

- It ensures the complication of critical tasks within the well defined constraints.
- It can not afford to miss even a single deadline, single miss can lead to the critical failure.
- Example: Flight controller system.

▪ **Soft Real time**

- Late completion of jobs is undesirable but not fatal(does not leads to any critical failure) .
- System performance degrades as more & more jobs miss deadlines.
- Example: Online Databases, DVD player cannot process a frame.

3) Describe RTOS its types with suitable examples.

- A system is said to be Real Time if it is required to complete it’s work & deliver it’s services on time.
- A real time OS defines the completion of job within the
- *rigid time constraints* otherwise job looses its meaning.
- Example – Flight Control System,Air line reservation system.
- All tasks in that system must execute on time.

➢ **The Real Time OS is of two types:**

- Hard Real-Time OS
- Soft Real Time OS

▪ **Hard Real time**

- It ensures the complication of critical tasks within the well defined constraints.
- It can not afford missing even a single deadline, single miss can lead to the critical failure.
- Example: Flight controller system.

▪ **Soft Real time**

- Late completion of jobs is undesirable but not fatal(does not leads to any critical failure) .
- System performance degrades as more & more jobs miss deadlines.
- Example: Online Databases, DVD player cannot process a frame.

4) Explain Monolithic, Micro Kernel and Layered system of Operating System Structure.

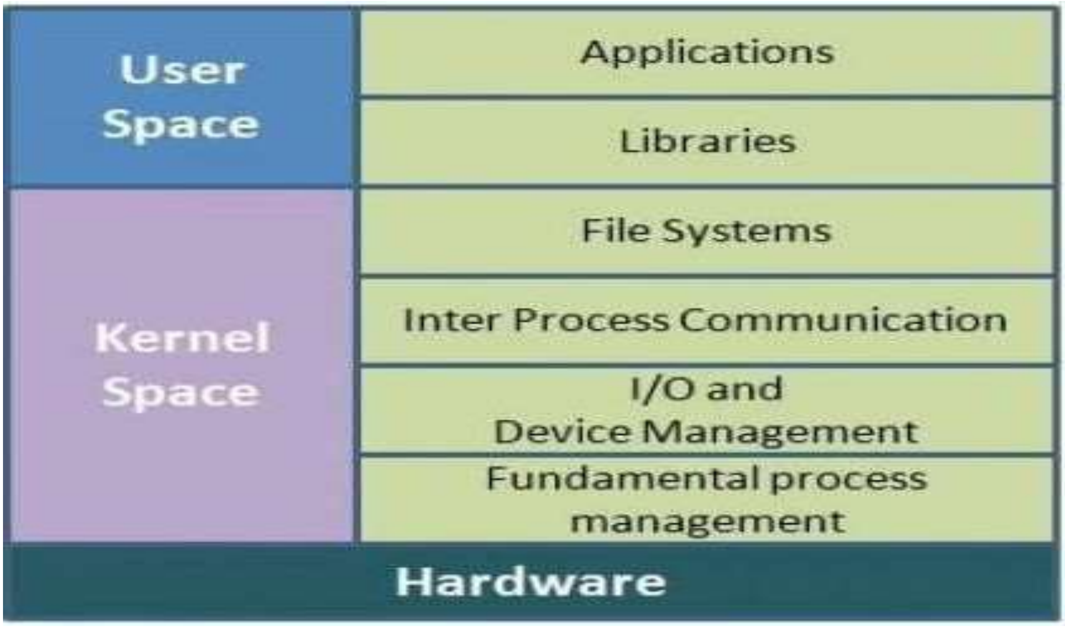
I. Monolithic Structure

▪ Monolithic systems – basic structure:

- o the entire operating system is working in kernel space.
- o A set of primitives or system calls implement all operating system services such as process management, concurrency, and memory management.
- o Device drivers can be added to the kernel as modules.
- o **user services** and **kernel services** are implemented under same address space. As both services are implemented under same address space, this makes operating system

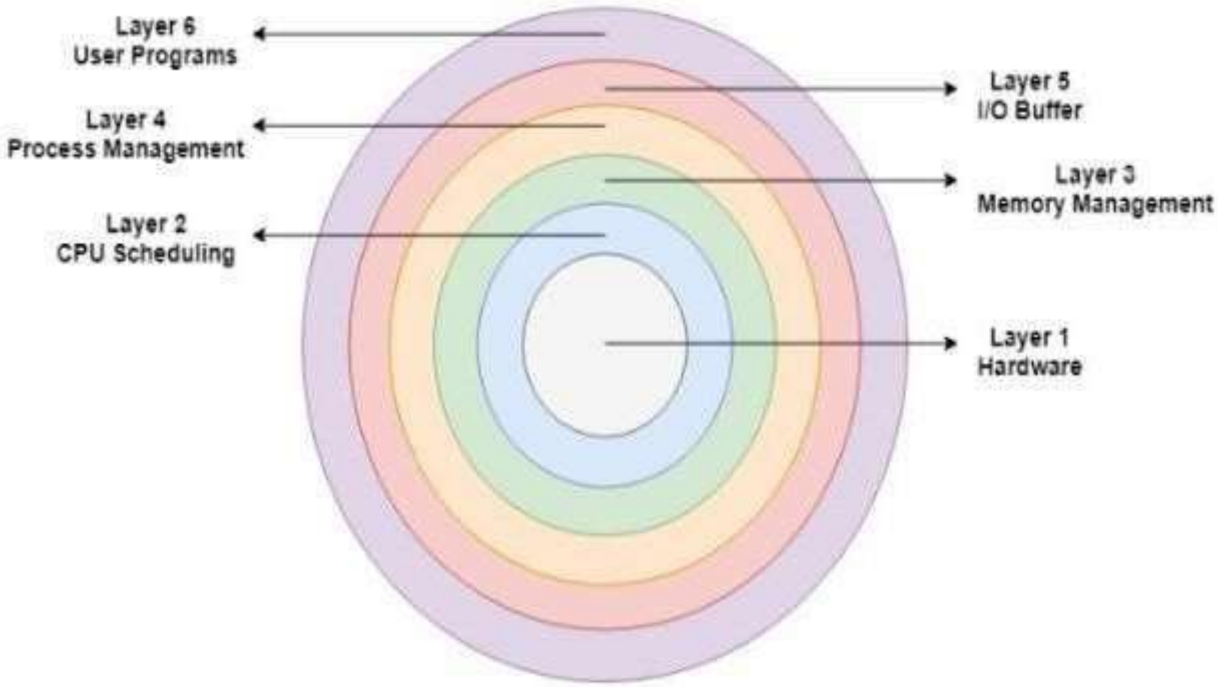
execution faster.

- If any service fails the entire system crashes, and it is one of the drawbacks of this kernel. The entire operating system needs modification if user adds a new service.
- Acts as a virtual machine which controls all hardware parts
- Examples: Linux, Unix like kernels, BSD, OS/360


User Space
Applications
Libraries
Kernel Space
File Systems
Inter Process Communication
I/O and Device Management
Fundamental process management
Hardware

II. Layered Structure

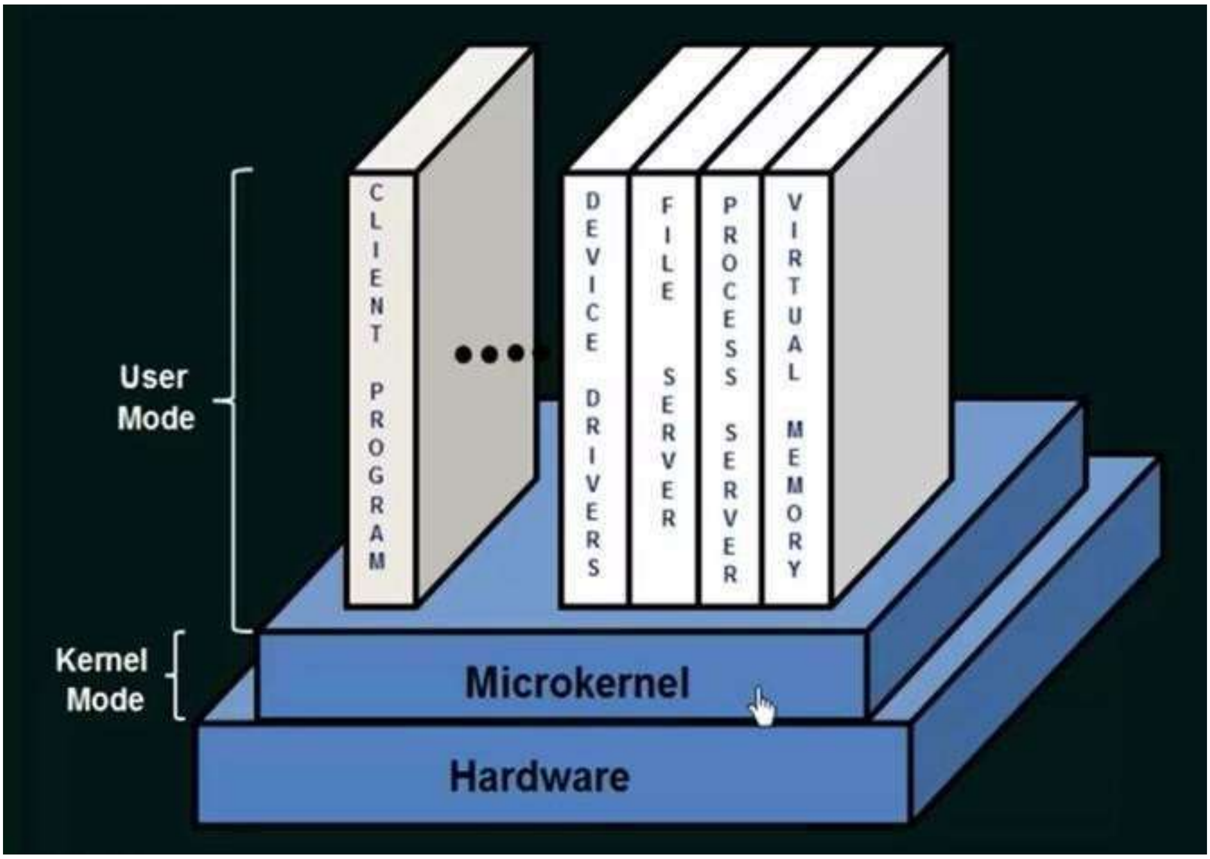
- A layered system consists of a **series of layers**, each of which depends only on the correct operation of the layer immediately beneath it.
- The lowest layer represents the hardware interface, and the highest layer the applicationinterface.
- All the layers can be defined separately and interact with each other as required. Also, it is easier to create, maintain and update the system if it is done in the form of layers.
- Change in one layer specification does not affect the rest of the layers.
- **A key problem in a layered system is deciding on the ordering of the layers.** (This is critical because of the requirement that each layer can ultimately only use services provided by layers below it - so the ordering cannot have anycycles.)
- In a strictly-layered structure, efficiency can also become a problem because when a higher-level layer requires a lower-level operation the request must work its way down layer by layer.
- Examples OS/2, window NT



LAYERED OPERATING SYSTEM

III. Microkernel Structure

- Kernel is made as small as possible only the most important services are put inside the kernel and rest of the OS service are present in the system application program.
- The user can easily interact with those not-so important services within the system applications kernel is solely responsible for the three most important services ofoperating system namely:
  - Inter-Process communication
  - Memory management
  - CPU scheduling
- Microkernel and system applications can interact with each other by *message passing* as and when required.
- This is extremely advantageous architecture since burden of kernel is reduced and less crucial services are accessible to the user and hence security is improved too. It is being highly adopted in the present-daysystems.
- MacOSX, Eclipse IDE is a good example of Microkernel Architecture.



5) Classify the operating system and Explain different views of the operating system. Discuss various services (tasks) of OS.

- ➢ An Operating System is a program that acts as an intermediate/interface between a user of a computer and the computer hardware.
- ➢ It manages hardware resources and provides various service to users
- ➢ It provides a user-friendly environment in which a user may *easily* develop and execute programs.
- ➢ Otherwise, hardware knowledge would be mandatory for computer programming.
- ➢ So, it can be said that an OS *hides the complexity* of hardware from uninterested users.

❖ **Functions/Services/Tasks of OS**

I. Process Management
- o *By process management OS manages many kinds of activities:*
- o All process from start to shut down i.e. open, save, copy, install, print.
- o Creation and deletion of user and system processes.

II. Memory Management
- o *The major activities of an operating regard to memory-management are:*
- o Decide which process are loaded into memory when memory space becomes available.
- o Allocate and deallocate memory space as needed.

III. File Management
- o *The file management system allows the user to perform such tasks:*
  - • Creating files and directories
  - • Renaming files

- Coping and moving files
- Deleting files

IV. Security Management

- *By security management OS manages many tasks such as:-*
  a. Alert messages
  b. Virus protection
  c. Dialogue boxes
  d. Firewall
  e. Passwords

V. Resource Management

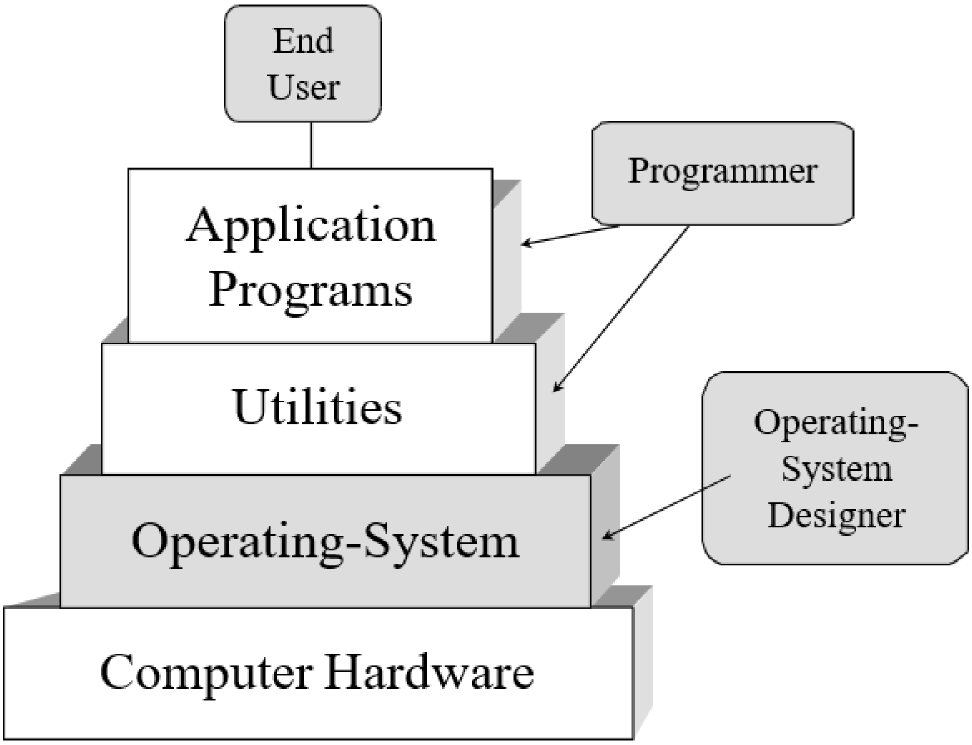
- *To manage Resources, OS perform many tasks such as:*
  a. Install drivers required for input and output, memory, power.
  b. Coordination among peripherals.

VI. Communication Management

- *For proper coordination OS performs communication management:*
  a. Communication between *user-application software- hardware.*
  b. One computer to another via LAN or WAN.

❖ Views of OS

1) OS as a User/Computer Interface



- **End User:-** End user only sees the computer system in terms of the set of applications, he/she is not concerned with the details of the computer hardware.
- **Application Programs:-** An application can be expressed in a programming language and is developed by an application programmer.
- **Utilities :-** System Program Facilities are known as utilities.
- **Programmer :-** A programmer uses system program facilities (Utilities) in developing

an application.

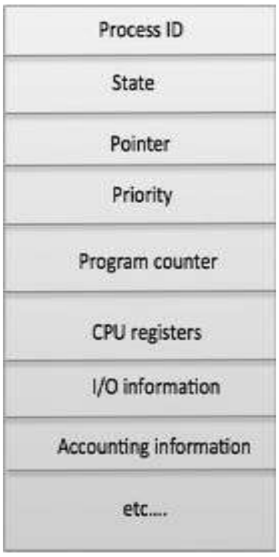
- **Operating Systems:-** OS is made up of the most important collection of system programs OS acts as a mediator. Make it easier for the Programmer and application program to access the facilities and services of OS.

2) OS as a Resource manager

- A computer is a set of resources that perform the functions such as storing, process, and transferring data.
- If a computer system is used by multiple applications (or users), then they will compete for these resources.
- OS is responsible for managing the resources to control these functions.

6) Explain PCB in detail. Differentiate various types of schedulers.(From Book)

- To implement the process model, the operating system maintains a table called the process table.
- Process table is *array* of structure
- Another name for process table is process control block (PCB)
- PCB is the key to multiprogramming.
- PCB must be saved when a process is switched from *running* to *ready* or *blocked* states.



Process ID
State
Pointer
Priority
Program counter
CPU registers
I/O information
Accounting information
etc.....

- Process ID - Unique identification for each of the processes 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, its typically numeric value. A process is assigned a priority at its creation.
- Program Counter - Program Counter is a pointer to the address of the next instruction to be executed for this process.
- CPU registers – contains the content of the register when the CPU was last released by the process.
- IO status information - This includes a list of I/O devices allocated to the process.
- 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 a snapshot i.e. the image of the PSW when the CPU was last voluntarily leave by the process.
- Memory management information - it includes values of base and limit registers, information about page table or segment table.
- 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.

- Types of Scheduler

1) **Long-term scheduler**
   - When the process is created.
   - Selects process and loads it into the ready queue (memory) for execution.
   - This is a decision whether to add a new process to the set of processes that are currently active.

2) **Medium term scheduler**
   - Memory manager.
   - Swap in, Swap out from main memory non-active processes.

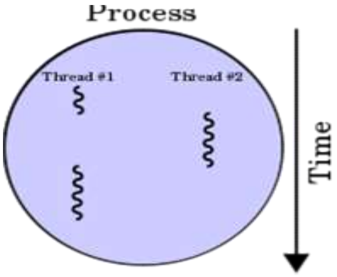
3) **Short term scheduler**
   - Deals with processes among ready processes.
   - Very fast, similar to action at every clock interrupt
   - if a process requires a resource (or input) that it does not have, it is removed from the ready list (and enters the WAITING state).
   - Short-term scheduling is the actual decision of which ready process to execute next

7) What do you mean by thread? Explain threads with example. Write short note on Multithreading.

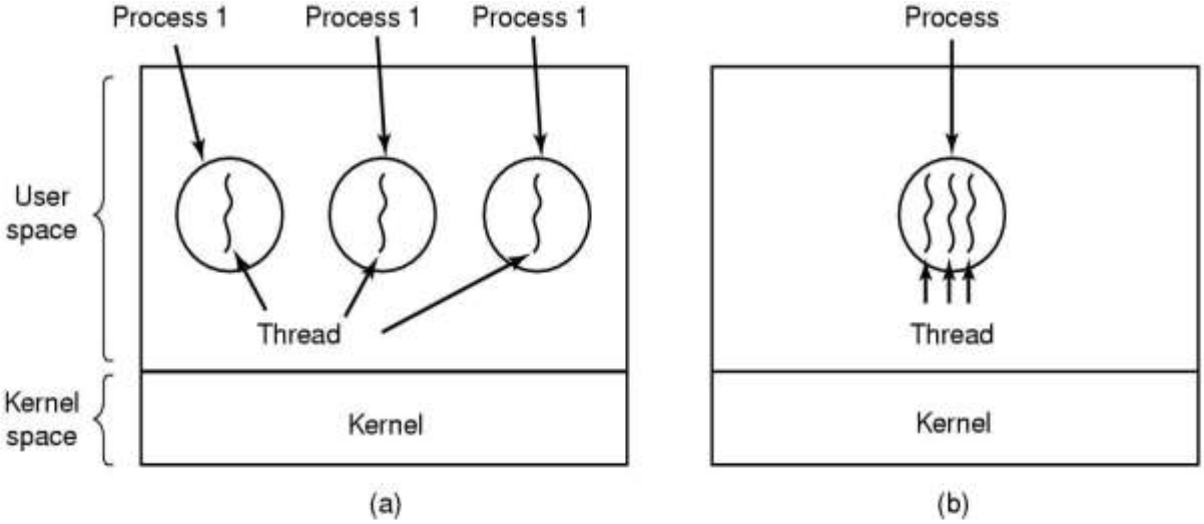
Definition:-

"A thread is the smallest unit of execution."

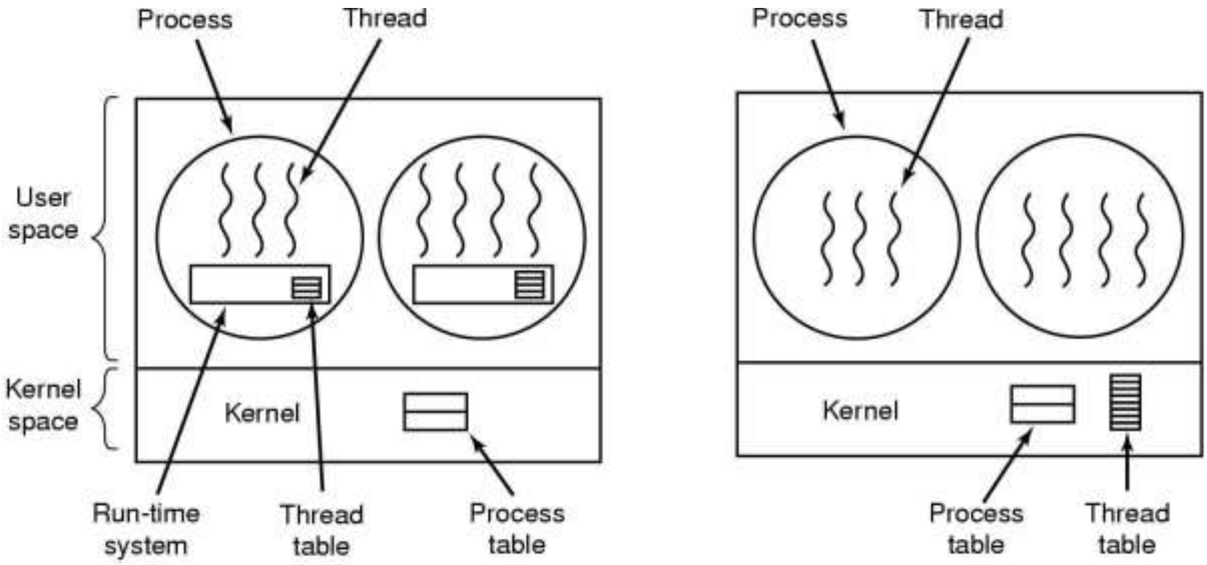
- Figure: A process with two threads of execution on a single processor machine.
- A thread is a light-weight process.



- Threads in three process (a) and one process (b)



- There are two types of threads to be managed in a modern system: **User threads and kernel threads**.
- **User threads** are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.
- **Kernel threads** are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.



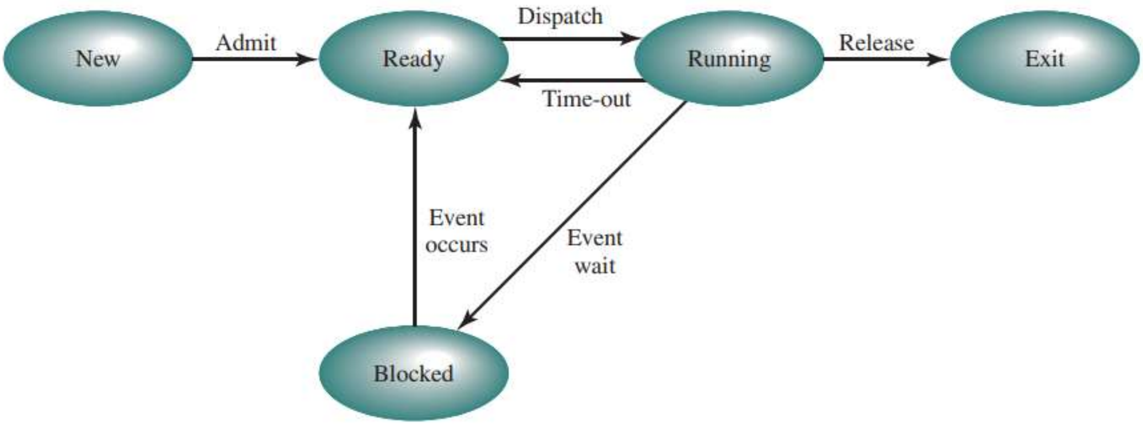
Threads of User Space

- **Advantages**
- Fast switching among threads.
- Thread scheduling can be application specific.
- **Disadvantages**
- When a user level thread executes a system call, not only that thread but all of the threads within that process are blocked.
- Advantages of multiprocessing can not achieve because kernel assigns one process to one processor. (kernel schedules processes not threads).

Multithreading :

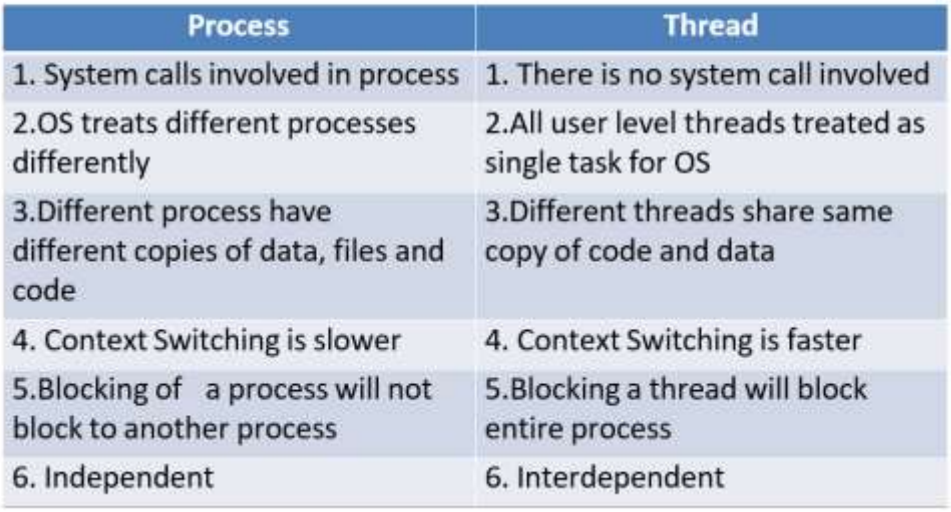
- Multithreading is a programming and operating system concept that allows multiple threads to exist within the context of a single process. Each thread represents a separate flow of control, enabling concurrent execution of tasks within a program.
- Multithreading is essential for maximizing the utilization of modern multi-core processors, as it allows programs to perform multiple tasks simultaneously. By dividing a program into multiple threads, developers can achieve improved performance and responsiveness, as well as better resource utilization.
- However, multithreading introduces challenges such as race conditions, synchronization issues, and deadlocks, which need to be carefully managed through synchronization mechanisms like locks, semaphores, and monitors. Proper synchronization ensures that threads can safely access shared resources without conflicts.
- Multithreading is widely used in various applications, including web servers, database systems, multimedia processing, and real-time systems. It enables developers to create efficient, responsive, and scalable software that can take advantage of the parallel processing capabilities of modern hardware.
- Overall, understanding and effectively implementing multithreading is crucial for developing high-performance and concurrent software systems that can leverage the full potential of modern computing architectures.

8) What is process? What are the different type of states of any process?



- Running: The process that is currently being executed. For this chapter, we will assume a computer with a single processor, so at most one process at a time can be in this state.
- Ready: A process that is prepared to execute when given the opportunity.
- Blocked/Waiting: A process that cannot execute until some event occurs, such as the completion of an I/O operation.
- New: A process that has just been created but has not yet been admitted to the pool of executable processes by the OS. Typically, a new process has not yet been loaded into main memory, although its process control block has been created.
- Exit: A process that has been released from the pool of executable processes by the OS, either because it halted or because it aborted for some reason.

9) Difference between process and thread.



Process	Thread
1. System calls involved in process	1. There is no system call involved
2.OS treats different processes differently	2.All user level threads treated as single task for OS
3.Different process have different copies of data, files and code	3.Different threads share same copy of code and data
4. Context Switching is slower	4. Context Switching is faster
5.Blocking of a process will not block to another process	5.Blocking a thread will block entire process
6. Independent	6. Interdependent

10) What is scheduler? Explain queuing diagram representation of process scheduler with figure.

- Yet Not Decided

11) What is thread? Explain thread structure.

- Yet Not Decided

12) Explain process creation and process termination.

- Process operations: (two state process model)
  1) Process creation
  - **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**.**
  - **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.

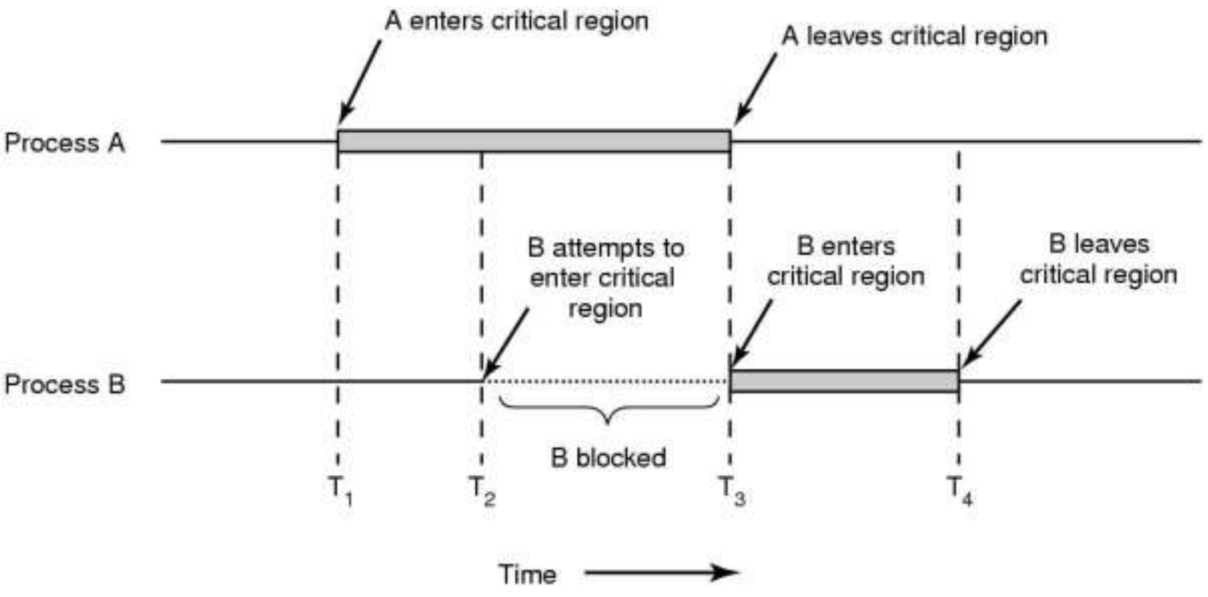
  2) Process terminations
  - **Normal exit (voluntary)** :- Terminate when done their job. i.e. when all the instructions from program get executed. Event user can select option to terminate the process.
  - **Error exit (voluntary)** :- Process even terminated when some error occurred. Example divide by zero error.
  - **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 “Terminate Process”

13) Explain types of thread.

- There are two types of threads to be managed in a modern system: **User threads and kernel threads**.
  - **User threads** are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.
  - **Kernel threads** are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

14) What is critical section problem? Explain with example?

- The **part of program where the shared resource is accessed** is called critical section or critical region.
- Sometimes a process has to access shared memory or files, or do other critical things that can lead to races. This type of **part of program** where shared memory is accessed is called the **critical region** or **critical section.**
- If no two processes are ever in their critical regions at the same time then we could avoid race.



- Any solution to the CS problem must satisfy following criteria:

1. **Mutual Exclusion**
    - Only one process at an instant of time can enter into critical section.
    - Also, critical section must be accessed in mutual exclusion fashion.
2. **Progress**
    - Progress says, only consider those processes should compete which actually Wants to access or enter into the critical section.
    - Also, progress is your mandatory criteria.
3. **Bounded Wait**
    - There should be a maximum bound up to which a process can wait.
    - No process should have to wait forever to enter a critical section.

15) What is Semaphore? Explain producer consumer problem using semaphore?

❖ **Semaphore :-**

- A semaphore is a synchronization construct used in concurrent programming to control access to a common resource by multiple processes or threads. It is essentially a variable or abstract data type that is used for controlling access to shared resources in a multi-threaded environment. Semaphores were introduced by Dutch computer scientist Edsger W. Dijkstra in the late 1960s.
- A semaphore maintains a count or a value which represents the number of available resources or the number of processes that can access a particular resource simultaneously. It supports two fundamental operations: "wait" (also known as "P" or "down") and "signal" (also known as "V" or "up").
    a) Wait Operation (P): When a process wants to access the shared resource, it performs a wait operation on the semaphore. If the semaphore's count is greater than zero, indicating that resources are available, the process decrements the semaphore count and proceeds to access the resource. If the count is zero, indicating that no resources are currently available, the process is blocked (or put to sleep) until a resource becomes available.
    b) Signal Operation (V): After a process has finished using the shared resource, it performs a signal operation on the semaphore. This increments the semaphore count, indicating that a resource has been released and is now available for use by another process. If there are processes waiting on the semaphore (due to a previous wait operation), one of them is awakened.

- Semaphores can be used to solve various synchronization problems, including the critical section problem, producer-consumer problem, and readers-writers problem.
- There are two types of semaphores:
  - **a)** Binary Semaphore: Also known as mutex (mutual exclusion) semaphore, it can only take on the values 0 and 1. It is typically used to control access to a single resource where only one process can access it at a time.
  - **b)** Counting Semaphore: This type of semaphore can take on any non-negative integer value. It is used to control access to a pool of identical resources, allowing multiple processes to access them simultaneously up to a certain limit.

- Semaphores provide a powerful mechanism for coordinating concurrent processes and ensuring that they can safely access shared resources without causing conflicts or inconsistencies.

❖ **The Producer-Consumer Problem :-**

- It is also known as ***Bounded Buffer Problem*** **in (multi- process synchronization problem).**
- Consider two processes Producer and Consumer , whoshare common, fixed size buffer**.**
- Producer puts information into the buffer
- Consumer consume this information from buffer.
- Trouble arises when the producer wants to put a new item in the buffer, but it is already full**.**
- And consumer wants to remove an item from buffer, but it is already empty**.**

❖ Semaphores are commonly used to solve the producer-consumer problem by controlling access to the shared buffer. Here's how semaphores can be used to implement a solution:

a) Empty Slots Semaphore: This semaphore represents the number of empty slots in the buffer. Initially, it is set to the size of the buffer, indicating that all slots are empty.

b) Full Slots Semaphore: This semaphore represents the number of filled slots in the buffer. Initially, it is set to 0, indicating that there are no items in the buffer initially.

c) Mutex Semaphore: This semaphore is used to ensure mutual exclusion while accessing the buffer. It prevents multiple producers or consumers from accessing the buffer simultaneously, avoiding race conditions.

16) Define process synchronization and explain Peterson solution algorithms?

❖ Process Synchronization
- o Process synchronization is the mechanism of coordinating the execution of multiple processes to ensure that they don't interfere with each other and access shared resources safely. This prevents race conditions, data corruption, and other problems that can arise when multiple processes access shared resources concurrently.

❖ Peterson's Solution Algorithm
- o Peterson's solution is a classic algorithm for solving the critical section problem for two processes. It uses two shared variables, flag and turn, to ensure that only one process can enter the critical section at a time.

❖ Here's how it works:

1) Initialization:
- o Both flag variables are initialized to false.
- o turn is initialized to 0 or 1, indicating which process has the right to enter the critical section first.

2) Entry Section:
- o Each process sets its flag variable to true.
- o Each process checks the other process's flag variable.

- If the other process's flag variable is false, the process enters the critical section.
- If the other process's flag variable is true, the process checks the turn variable.
- If the turn variable is equal to the process's own identifier, the process enters the critical section.
- Otherwise, the process waits until the turn variable changes to its own identifier.

3) Critical Section:

- The process executes its critical code.

4) Exit Section:

- The process sets its flag variable to false.
- The process sets the turn variable to the other process's identifier.
- This algorithm ensures that only one process can be in the critical section at a time, even if both processes attempt to enter the critical section simultaneously.

❖ Advantages of Peterson's Solution

- Simple and elegant: The algorithm is easy to understand and implement.
- Fair: Both processes have an equal chance of entering the critical section.
- Non-blocking: The algorithm does not block either process indefinitely.

❖ Disadvantages of Peterson's Solution

- Overhead: The algorithm requires several shared variables, which can increase overhead.
- Limited to two processes: The algorithm can only be used for two processes.

17) What is Monitor? Explain with any example using monitor?

➢ **Monitors :-**

- Tony Hoare introduced the concept of a monitor in 1974, followed by Per Brinch Hansen's proposal in 1975. Monitors serve as a synchronization structure designed to be incorporated into high-level programming languages.
- A monitor is essentially a special module or package that contains a collection of procedures, variables, and data structures. Processes can call the procedures within the monitor whenever needed, but they are restricted from directly accessing the internal data structures declared outside the monitor.
- Key points about monitors:
  1) Monitors encapsulate procedures, variables, and data structures.
  2) Processes access the procedures within the monitor but cannot directly access its internal data structures.
  3) Monitors facilitate synchronized access to shared resources in concurrent programming environments.
- Processes may call the procedures in a monitor whenever they want to, but they cannot directly access the monitor’s internal data structures from procedures declared outside the monitor.

18) Explain Dining-Philosophers Problem with solution.

➢ Philosophers eat and think.

1) To eat, they must first acquire a left fork and then a right fork.
2) Then they eat.
3) Then they put down the forks.
4) Then they think.
5) Go to 1.

- The dining philosopher's problem is the classical problem of synchronization which says that Five philosophers are sitting around a circular table
- Their job is to think and eat alternatively.
- A bowl of noodles is placed at the center of the table along with five chopsticks for each of the philosophers.
- To eat a philosopher needs both their right and a left chopstick. A philosopher can only eat

if both immediate left and right chopsticks of the philosopher is available.

- In case if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their (either left or right) chopstick and starts thinking again.

```
Void philosopher()
{
        while(T)
        {
                think();
                take_fork(i);
                take_fork((i+1)%n);
                eat();
                put_fork(i);
                put_fork((i+1)%n);
        }
}
```

❖ **Solution (Not correct)**

- Solution of **Dining Philosopher** problem using **Semaphore**

```
Void philosopher()
{
    while(T) {
            think();
            take_fork(Si);
            take_fork((Si+1)%n);
            eat();
            put_fork(Si);
            put_fork((Si+1)%n);
    }
}
```

  - **S[i] five semaphores** (i.e. equal to number of forks) **S1, S2, S3, S4, S5**

P1 →	S1	S2
P2 →	S2	S3
P3 →	S3	S4
P4 →	S4	S5
P5 →	S5	S1

19) Explain about Deadlock Avoidance?

- In most systems, however, resources are requested one at a time. The system must be able to decide whether granting a resource is **safe or not and only make the allocation when it is safe.**

- If the system can allocate resources to the process in such a way that it can avoid deadlock. Then the system is in a safe state.
- If the system can’t allocate resources to the process safely, then the system is in an unsafe state.

Here's an explanation of deadlock avoidance techniques:

1) Resource Allocation Graph: One common approach to deadlock avoidance involves using a resource allocation graph to track resource allocations and requests. Nodes in the graph represent processes and resources, while edges represent resource allocation and requests. By analyzing this graph, it's possible to detect and prevent potential deadlocks.

2) Resource Allocation Policies: Deadlock avoidance relies on resource allocation policies that govern how resources are allocated to processes. These policies may include strategies such as:

- Resource Ordering: Define a global order in which resources must be acquired to prevent circular wait conditions. Processes must request resources in this predefined order.
- Banker's Algorithm: Used for systems with a fixed number of resources. It ensures that resource allocation requests won't lead to a deadlock by simulating potential future resource requests.

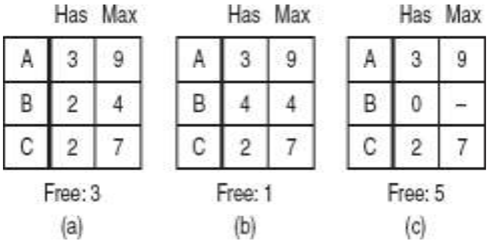
3) Dynamic Resource Allocation: Systems can dynamically allocate and deallocate resources based on resource usage patterns and process demands. By monitoring resource utilization and predicting future resource needs, systems can make intelligent decisions to allocate resources in a way that minimizes the risk of deadlocks.

4) Timeouts and Rollbacks: In distributed systems, deadlock avoidance may involve setting timeouts for resource requests. If a process waits too long for a resource, it may timeout and release its held resources, preventing potential deadlocks. Rollback mechanisms can also be used to revert the state of processes to a safe point to break potential deadlock situations.

5) Deadlock Detection and Recovery: Deadlock avoidance techniques can be complemented with deadlock detection mechanisms. If avoidance measures fail and a deadlock is detected, systems can use techniques such as process termination, resource preemption, or rollback to recover from the deadlock state.

Safe State

- A **total of 10 instance** of the resource exist, so with **7 resources already allocated, there are 3 still free.**
- ***Figure : Demonstration that the state in (a) is safe.***



	Has	Max
A	3	9
B	2	4
C	2	7

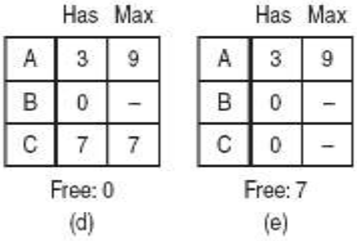
Free: 3
(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1
(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5
(c)



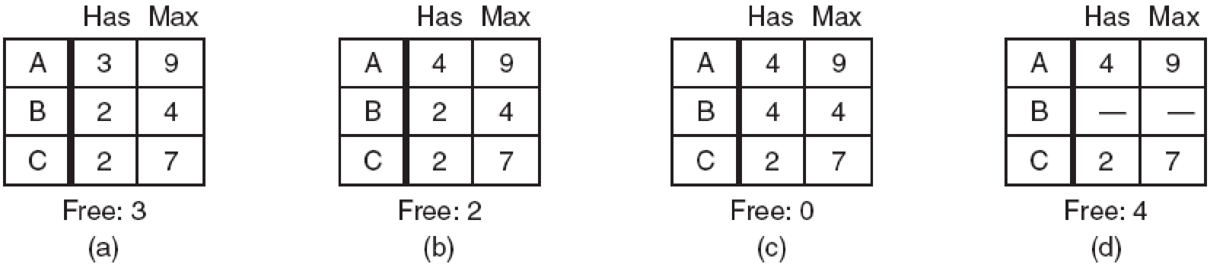
	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0
(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7
(e)

- **Figure :** Demonstration that the state in (b) i



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

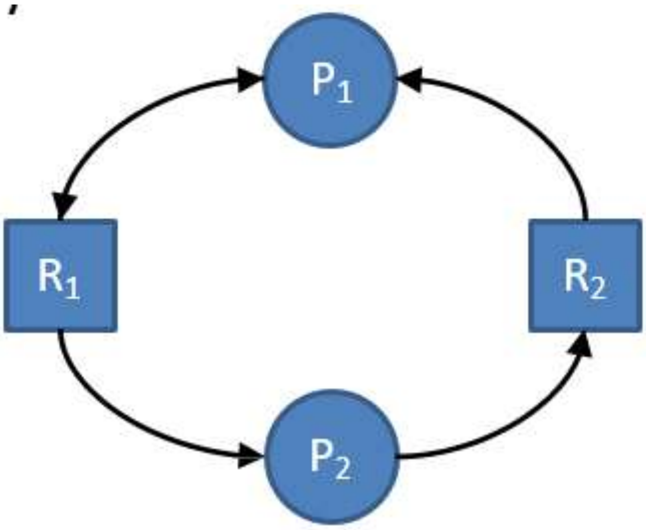
	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

20) Explain how recovery from deadlock?

- ➢ Deadlock Recovery
    - **a)** Recovery through pre-emption
        - o **Recovery through preemption**
            - o In some cases, it may be possible to temporarily take a resource away from its current owner and give it to another process.
            - o Recovering this way is frequently difficult or impossible.
            - o Choosing the process to suspend depends largely on which ones have resources that can easily be taken back.



b) Recovery through rollback

- o The simplest way to break a deadlock is to kill one or more processes.
- o Kill all the processes involved in deadlock.
- o Kill process one by one.
- o After killing each process check for deadlock.
    - o If deadlock is recovered then stop killing more process.
    - o Otherwise kill another process.

c) Recovery through killing processes

- o Checkpoint a process periodically.
- o Check pointing a process means that its state is written to a file
- o so that it can be restarted later.
- o When deadlock is detected, rollback the preempted process up to the previous safe state before it acquired that resource.

21) Explain Dead lock detection (Banker’s Algorithm) with Example?

Deadlock detection is a mechanism used in operating systems to identify the presence of deadlocks within a system. One well-known deadlock detection algorithm is the Banker's Algorithm, which was developed by Edsger W. Dijkstra. It's commonly used in systems with a fixed number of resources to determine if the system has entered a deadlock state.

Here's how the Banker's Algorithm works with an example:

Consider a system with five processes (P1 to P5) and three types of resources (A, B, and C). The system has a fixed number of available resources of each type. Each process can request resources and release them once it's done.

- Available resources: $A=3A=3$, $B=3B=3$, $C=2C=2$
- Maximum demand of each process:

ProcessMaximum Demand (A, B, C)$P1(7,5,3)P2(3,2,2)P3(9,0,2)P4(2,2,2)P5(4,3,3)$Process$P1P2P3P4P5$
Maximum Demand (A, B, C)(7,5,3)(3,2,2)(9,0,2)(2,2,2)(4,3,3)

- Currently allocated resources to each process:

ProcessAllocated Resources (A, B, C)$P1(0,1,0)P2(2,0,0)P3(3,0,2)P4(2,1,1)P5(0,0,2)$Process$P1P2P3P4P5$
Allocated Resources (A, B, C)(0,1,0)(2,0,0)(3,0,2)(2,1,1)(0,0,2)

With this information, we can use the Banker's Algorithm to check if the system is in a safe state or if it's in a deadlock.

1. **Calculate Need Matrix**: The need matrix indicates the remaining resources that each process needs to complete its execution.

Need=Maximum Demand−Allocated ResourcesNeed=Maximum Demand−Allocated Resources
ProcessNeed (A, B, C)$P1(7,4,3)P2(1,2,2)P3(6,0,0)P4(0,1,1)P5(4,3,1)$
Process$P1P2P3P4P5$Need (A, B, C)(7,4,3)(1,2,2)(6,0,0)(0,1,1)(4,3,1)

2. **Check for Safety**: To check for safety, we simulate the allocation of resources to processes and see if there's a sequence in which all processes can complete.

- We start by assuming all resources are available.

- We check if any process can complete its execution based on the available resources. If yes, we allocate resources to that process and release them.
- We repeat this process until either all processes complete or no process can be allocated resources without violating the system's constraints.

For example, let's start with process P4:

- Available resources: $A=0+2=2A=0+2=2$, $B=1+0=1B=1+0=1$, $C=1+0=1C=1+0=1$

With available resources, P4 can complete its execution, releasing resources (2, 1, 1).

- Updated available resources: $A=2+2=4A=2+2=4$, $B=1+1=2B=1+1=2$, $C=1+1=2C=1+1=2$

Now, let's check which other processes can proceed. In this example, P2 can proceed.

- Allocate resources to P2 and update available resources.
- Check for next process to proceed until all processes can complete or none of them can proceed.

If a safe sequence exists, the system is in a safe state, and deadlock is not present. If no safe sequence exists, deadlock is detected.

In this example, a safe sequence is P4 -> P2 -> P5 -> P1 -> P3, indicating that the system is in a safe state. Therefore, no deadlock is present.

22) Write about Deadlock Prevention Methods?

- Deadlock can be prevented by violating the one of the four conditions that leads to deadlock.

1) Attacking the Mutual Exclusion Condition

- This method involves allowing resources to be shared among processes rather than exclusively owned. By allowing resources to be accessed concurrently by multiple processes, the mutual exclusion condition is violated, and deadlocks can be prevented.

2) Attacking the Hold and Wait Condition

- To prevent hold and wait, processes must request and hold all required resources simultaneously, or none at all. This approach ensures that a process doesn't hold resources while waiting for others, reducing the likelihood of deadlocks.

3) Attacking the No Preemption Condition

- Preemption involves forcibly taking resources away from processes if necessary. By allowing preemption, a process holding resources can be interrupted to satisfy the resource needs of other processes, thereby preventing deadlock situations.

4) Attacking the Circular Wait Condition

- To prevent circular wait, resources can be assigned unique numerical identifiers, and processes are required to acquire resources in increasing order. This approach ensures that processes cannot form a circular chain of waiting for resources, thereby preventing deadlocks.

23) Explain Semaphores in Detail

Semaphores are synchronization primitives used in concurrent programming to control access to shared resources and coordinate the execution of multiple processes or threads. They were introduced by Dutch computer scientist Edsger W. Dijkstra in the 1960s and have since become fundamental in managing concurrency in operating systems and concurrent applications.

Here's a detailed explanation of semaphores:

1. **Concept**:

- A semaphore is a non-negative integer variable (often referred to as a semaphore counter) that is used to control access to shared resources.
- Semaphores can be thought of as a signaling mechanism between processes or threads. They allow one process to signal the availability of resources to another process.

2. **Operations**:

- **Wait (P) Operation**: Also known as down or decrement operation. When a process wants to access a shared resource, it performs a wait operation on the semaphore.
- If the semaphore's value is positive (greater than 0), indicating that resources are available, the process decrements the semaphore counter and proceeds to access the resource.
- If the semaphore's value is zero, indicating that no resources are currently available, the process is blocked (or put to sleep) until another process signals (increments) the semaphore, indicating that resources have become available.
- **Signal (V) Operation**: Also known as up or increment operation. After a process has finished using the shared resource, it performs a signal operation on the semaphore.
- This operation increments the semaphore counter, indicating that a resource has been released and is now available for use by another process.
- If there are processes waiting on the semaphore (due to a previous wait operation), one of them is awakened to proceed.

3. **Types of Semaphores**:

- **Binary Semaphore**: Also known as mutex semaphore (mutual exclusion). It can only take on the values 0 and 1. It is typically used to control access to a single resource where only one process can access it at a time.

- **Counting Semaphore**: This type of semaphore can take on any non-negative integer value. It is used to control access to a pool of identical resources, allowing multiple processes to access them simultaneously up to a certain limit.

4. **Applications**:

- Semaphores are used in various synchronization problems, including the producer-consumer problem, reader-writer problem, dining philosophers problem, and more.
- They are fundamental in operating system kernels for managing access to system resources such as files, memory, and I/O devices.
- In concurrent programming, semaphores are used to coordinate access to critical sections of code to ensure mutual exclusion and prevent race conditions.

5. **Implementation**:

- Semaphores can be implemented using atomic operations provided by the hardware or using software constructs such as locks, mutexes, or condition variables.
- In modern operating systems, semaphores are typically provided as part of the operating system's API and are managed by the kernel.

24) Explain IPC Problem – Readers & Writers Problem.

The Readers-Writers problem is a classic synchronization problem in computer science that deals with the coordination of multiple processes accessing shared resources. In particular, it focuses on scenarios where there are multiple reader processes and multiple writer processes accessing a shared data structure, such as a database, file, or memory location.

The problem is defined as follows:

1. **Readers**:

- Multiple reader processes can read from the shared resource simultaneously without any issues.
- Readers do not modify the shared resource, so they can read concurrently without interfering with each other.

2. **Writers**:

- Only one writer process can write to the shared resource at a time.
- Writers need exclusive access to the shared resource to ensure data integrity and consistency.

- Writers must be prevented from writing to the resource when readers are accessing it, and vice versa.

The objective of the Readers-Writers problem is to design synchronization mechanisms to ensure that:

- Readers can read concurrently without interfering with each other.
- Writers can write exclusively, ensuring that no readers are reading and no other writers are writing simultaneously.

Various solutions exist for the Readers-Writers problem, each with its own trade-offs and considerations. Some common solutions include:

1. **First-Come First-Served (FCFS)**:

- Maintain a queue of requests for access to the resource.
- Readers and writers are serviced in the order they arrive.
- This solution can lead to starvation for writers if readers are continually accessing the resource.

2. **Readers Preference**:

- Allow multiple readers to access the resource simultaneously.
- Writers must wait until all readers have finished reading before writing.
- This solution prioritizes readers over writers, which can lead to starvation for writers if there are always readers.

3. **Writers Preference**:

- Writers are given priority over readers.
- When a writer arrives, it blocks all subsequent readers until it has finished writing.
- This solution can lead to starvation for readers if writers are continually accessing the resource.

4. **Semaphore-based Solution**:

- Use semaphores or mutexes to control access to the shared resource.
- Maintain separate counters for readers and writers to track the number of processes accessing the resource.

- Use signaling mechanisms to coordinate access between readers and writers, ensuring mutual exclusion when necessary.

25) What is Deadlock? List the conditions that lead to deadlock. How Deadlock can be prevented

- ➢ A deadlock consists of a **set** of blocked processes, each **holding** a resource and **waiting** to acquire a resource held by another process in the set.
- ➢ **For Example:**
- o two processes each want to record a scanned document on a CD.
- o process 1 request for scanner & get it
- o process 2 requests for CD writer & get it
- o process 1 request for CD writer but is blocked
- o Process 2 requests for the scanner but is blocked.
- o At this point both processes are blocked and will remain so forever,
- o **This situation is called a deadlock**

❖ Necessary Conditions for Deadlock

1) **Mutual exclusion:**
   - o **At least one resource type** in the system that can be used in
   - o **non-shareable mode** (i.e. one at a time). for example printer.
   - o Because many resources can be shared by more than one process at a time .(e.g., Memory location).
2) **Hold and Wait:**
   - o Processes are allowed to request new resources without releasing the resources they are currently holding.
   - o Hold and wait is a condition in which a process is holding one resource while simultaneously waiting for another resource that is being held by another process.
3) **No pre-emption:**
   - o A resource can be released only voluntarily by the process holding it after that process has completed its task.
   - o Preemption is temporarily interrupting an executing task and later resuming it.
   - o **For example,** if process P1 is using a resource and a high-priority process P2 requests for the resource, process P1 is stopped and the resources are allocated to P2.
4) **Circular wait:**
   - o Two or more processes must form a circular chain in which each process is waiting for a resource that is held by the next member of the chain.
   - o In circular wait, two or more processes wait for resources in a circular order.

❖ **Deadlock Prevention**

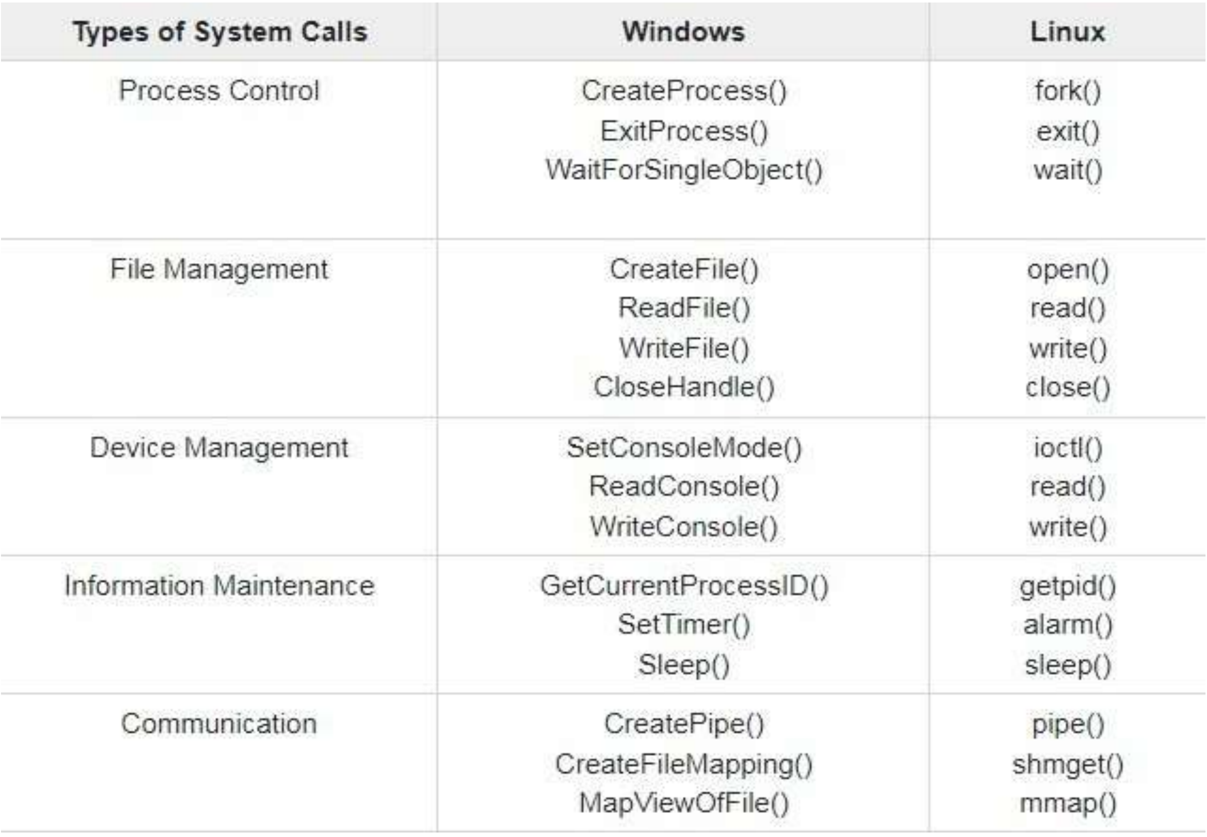
- o Deadlock can be prevented by violating the one of the four conditions that leads to deadlock.
  1) Attacking the Mutual Exclusion Condition
     - o This method involves allowing resources to be shared among processes rather than exclusively owned. By allowing resources to be accessed concurrently by multiple processes, the mutual exclusion condition is violated, and deadlocks can be prevented.
  2) Attacking the Hold and Wait Condition
     - o To prevent hold and wait, processes must request and hold all required resources simultaneously, or none at all. This approach ensures that a process doesn't hold resources while waiting for others, reducing the likelihood of deadlocks.
  3) Attacking the No Preemption Condition
     - o Preemption involves forcibly taking resources away from processes if necessary. By allowing preemption, a process holding resources can be interrupted to satisfy the resource needs of other processes, thereby preventing deadlock

situations.

4) Attacking the Circular Wait Condition
   - To prevent circular wait, resources can be assigned unique numerical identifiers, and processes are required to acquire resources in increasing order. This approach ensures that processes cannot form a circular chain of waiting for resources, thereby preventing deadlocks.

26) What is system call? Explain the types of system call.

- A system call is a *request* that a program makes to the OS.
- OS and user program communicate with help of system call. System call are provided by the OS.
- Example : User program needs to read the file from the hard disk but as the user program is working in the user mode it can not have direct access to any computer hard ware, thus program makes system call instruction to transfer control to the operating system.
- Then OS process the system call and returns control to the instruction.



Types of System Calls	Windows	Linux

Process Control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Management	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device Management	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

27) Write different operating system services.

I. Process Management
   - *By process management OS manages many kinds of activities:*
   - All process from start to shut down i.e. open, save, copy, install, print.
   - Creation and deletion of user and system processes.

II. Memory Management
   - *The major activities of an operating regard to memory-management are:*
   - Decide which process are loaded into memory when memory space becomes available.
   - Allocate and deallocate memory space as needed.

III. File Management

- *The file management system allows the user to perform such tasks:*
    - Creating files and directories
    - Renaming files
    - Coping and moving files
    - Deleting files

IV. Security Management

- *By security management OS manages many tasks such as:-*
    - Alert messages
    - Virus protection
    - Dialogue boxes
    - Firewall
    - Passwords

V. Resource Management

- *To manage Resources, OS perform many tasks such as:*
    - Install drivers required for input and output, memory, power.
    - Coordination among peripherals.

VII. Communication Management

- *For proper coordination OS performs communication management:*
    a. Communication between *user-application software- hardware.*
    b. One computer to another via LAN or WAN.

28) What is spooling?

Spooling, which stands for "Simultaneous Peripheral Operation On-Line," is a computing term that refers to a technique used to improve the efficiency of input/output (I/O) operations between a computer and peripheral devices such as printers, disk drives, or network devices. The main purpose of spooling is to minimize the idle time of the CPU and peripheral devices by overlapping the I/O processing with other computing tasks.

In spooling, instead of sending data directly to the peripheral device for processing, the data is temporarily stored in a buffer or a queue called a "spool." This spool acts as an intermediate storage area where data can be queued up for processing by the peripheral device at its own pace, while the CPU can continue executing other tasks.

The spooling process typically involves the following steps:

1. **Input Spooling**: Data from input devices (such as keyboards or network connections) is captured and stored in a spool for processing. This ensures that the input data is not lost and can be processed in the order it was received.

2. **Output Spooling**: Similarly, output data destined for peripheral devices (such as print jobs or data to be written to disk) is stored in a spool before being sent to the output device. This allows the CPU to continue processing other tasks while the output device processes data from the spool.

3. **Spool Management**: The operating system manages the spooling process, including prioritizing and scheduling jobs in the spool, allocating resources, and ensuring proper synchronization between the CPU and peripheral devices.

Spooling offers several benefits, including:

- **Improved Performance**: By allowing the CPU to perform other tasks while data is being transferred between the computer and peripheral devices, spooling helps maximize system performance and minimize idle time.
- **Enhanced Efficiency**: Spooling helps manage I/O operations more efficiently, reducing the likelihood of bottlenecks and improving overall system throughput.
- **Resource Sharing**: Spooling enables multiple users or processes to share access to peripheral devices without interfering with each other's tasks.

29) What is kernel? Explain types of kernel.

The kernel is the core component of an operating system (OS) that acts as a bridge between the computer hardware and software applications. It provides essential services and functionalities required for managing system resources, executing processes, and facilitating communication between hardware components and software programs.

Here are the main types of kernels:

1. **Monolithic Kernel**:

- In a monolithic kernel architecture, all kernel services, device drivers, and system call implementations run in a single address space.
- Monolithic kernels are designed to maximize performance by minimizing the overhead associated with inter-process communication.
- Examples include the Linux kernel and earlier versions of the Windows NT kernel.

2. **Microkernel**:

- A microkernel architecture aims to keep the kernel as small and lightweight as possible by moving non-essential services, such as device drivers and file system implementations, out of the kernel space and into user space.
- Microkernels typically provide only basic functionalities, such as inter-process communication and memory management, while more complex services run as separate user-space processes.

- This design enhances system reliability and security by reducing the amount of code running in privileged kernel mode.
- Examples include the Mach kernel used in macOS and the QNX kernel.

3. **Hybrid Kernel**:

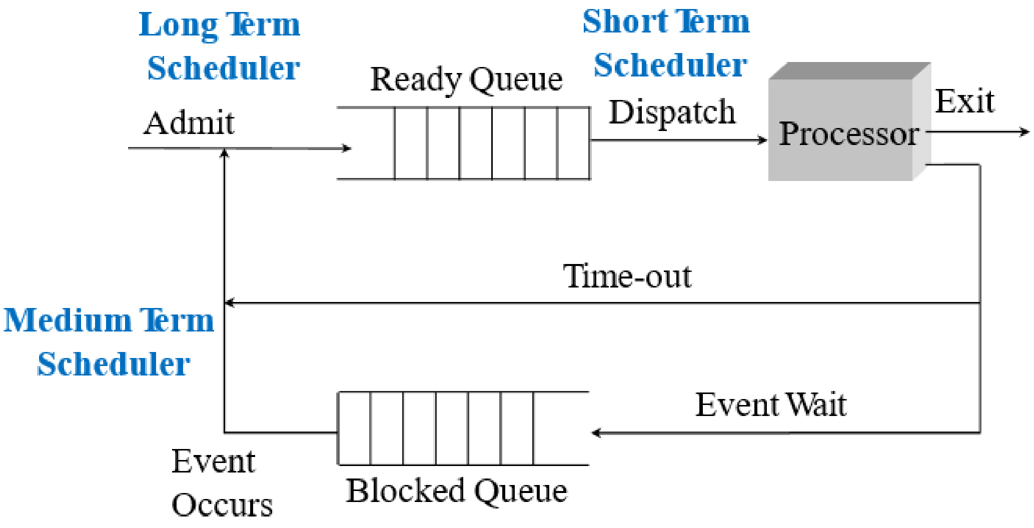
- Hybrid kernels combine elements of both monolithic and microkernel architectures, aiming to strike a balance between performance and modularity.
- In a hybrid kernel, core OS services like process management and memory management typically run in kernel space for performance reasons, while other services such as device drivers may run in user space.
- This architecture provides flexibility in designing the OS, allowing for better performance and easier development and maintenance compared to traditional monolithic kernels.
- Examples include the Windows NT kernel used in modern versions of Windows (such as Windows 10) and the XNU kernel used in macOS.

30) What is Process? Give the difference between a process and a program.

- ➢ A process is a program in execution. It represents a running instance of a program along with its associated resources, such as memory, CPU time, and I/O devices. Processes are fundamental to multitasking operating systems, which allow multiple programs to run concurrently by time-sharing the CPU among them.

Aspect	**Process**	**Program**
Definition	A process is an instance of a program in execution. It includes program code, data, and resources such as memory and CPU time.	A program is a set of instructions or code stored in a file on disk. It is a passive entity and does not execute until it is loaded into memory and executed by the operating system.
Execution	Actively executing and utilizing system resources, such as CPU time and memory.	Passive entity stored on disk until loaded into memory and executed.
Life Cycle	Created, executed, and eventually terminated or suspended by the operating system.	Static entity that exists as a file on disk until loaded into memory for execution.
Resource Usage	Utilizes system resources such as CPU time, memory, I/O devices, and network connections.	Does not consume system resources until loaded into memory and executed as a process.
Dynamic State	Exhibits dynamic states such as running, ready, blocked, or terminated, depending on its execution stage.	Static entity without dynamic states; does not change its state during execution.

31) What is scheduler? Explain the queuing diagram representation of the process scheduler with the figure.



- **Long term scheduler**
  - When process is created.
  - selects process and loads it into ready queue (memory) for execution.
  - This is a decision whether to add a new process to the set of processes that are currently active.
- **Medium term scheduler**
  - Memory manager.
  - Swap in, Swap out from main memory non-active processes.
- **Short term scheduler**
  - Deals with processes among ready processes.
  - Very fast, similar to action at every clock interrupt
  - if a process requires a resource (or input) that it does not have, it is removed from the ready list (and enters the WAITING state).
  - Short-term scheduling is the actual decision of which ready process to execute next

32) Define: Mutual Exclusion.

- Mutual exclusion is a concept in computer science and concurrent programming that ensures that only one process or thread accesses a shared resource at a time. It prevents multiple processes from simultaneously modifying the shared resource, which could lead to data inconsistency or corruption.

- In other words, mutual exclusion guarantees that if one process is accessing or modifying a resource, other processes are temporarily prevented from accessing or modifying the same resource until the first process has finished its operation and released the resource.

- Achieving mutual exclusion typically involves the use of synchronization mechanisms such as locks, semaphores, or mutexes. These synchronization primitives enforce a protocol where processes or threads must acquire exclusive access to the shared resource before performing any operation on it. Once a process has finished using the resource, it releases the lock or semaphore, allowing other processes to acquire access.

- Mutual exclusion is crucial for preventing race conditions and ensuring the consistency and correctness of concurrent programs where multiple processes or threads are accessing shared resources simultaneously.

33) Explain: Race conditions, Semaphore and Monitor.

- ❖ Race Conditions:

  - A race condition occurs in concurrent programming when the outcome of a program depends on the non-deterministic ordering of operations performed by multiple threads or processes.
  - It arises when multiple threads or processes access shared resources concurrently, and the final outcome of the program becomes unpredictable because it depends on the relative timing and interleaving of their operations.
  - Race conditions can lead to unexpected and erroneous behavior, such as data corruption, deadlock, or inconsistent program state.
  - To prevent race conditions, synchronization mechanisms such as locks, semaphores, or monitors are used to coordinate access to shared resources and enforce mutual exclusion.

- ❖ Semaphore:

  - A semaphore is a synchronization primitive used in concurrent programming to control access to shared resources and coordinate the execution of multiple threads or processes.
  - It consists of a counter and two atomic operations: wait (P) and signal (V).
  - The wait operation decrements the semaphore counter. If the counter becomes negative, the calling thread or process is blocked until the counter becomes non-negative.
  - The signal operation increments the semaphore counter. If there are threads or processes waiting on the semaphore, one of them is unblocked.
  - Semaphores can be used to implement various synchronization patterns, such as mutual exclusion, producer-consumer synchronization, and reader-writer synchronization.

- ❖ Monitor:

  - A monitor is a high-level synchronization construct used in concurrent programming to encapsulate shared data and operations on that data within a single module.
  - It provides mutual exclusion by automatically managing access to shared resources within its scope, ensuring that only one thread can execute monitor procedures or methods at a time.
  - Monitors typically support two types of operations: entry procedures, which acquire exclusive access to the monitor and execute critical sections of code, and condition variables, which allow threads to wait for specific conditions to become true.
  - Monitors abstract away the complexity of low-level synchronization primitives such as locks and semaphores, making it easier to write correct and thread-safe concurrent programs.

34) Define Critical Section, Race Condition.

- ❖ Critical Section:

  - A critical section refers to a part of a concurrent program where shared resources are accessed or modified by multiple threads or processes.
  - In the critical section, operations must be executed atomically, meaning that they should appear as if they were executed sequentially, without interference from other threads or processes.
  - It is crucial to ensure mutual exclusion in critical sections to prevent race conditions and maintain data integrity.
  - Synchronization mechanisms such as locks, semaphores, or monitors are used to coordinate access to critical sections and enforce mutual exclusion, allowing only one thread or process to execute the critical section at a time.

❖ Race Condition:

- A race condition occurs in concurrent programming when the outcome of a program depends on the non-deterministic order or timing of operations performed by multiple threads or processes accessing shared resources.
- It arises when multiple threads or processes access shared resources concurrently, and the final outcome of the program becomes unpredictable because it depends on the relative timing and interleaving of their operations.
- Race conditions can lead to unexpected and erroneous behavior, such as data corruption, deadlock, or inconsistent program state.
- To prevent race conditions, synchronization mechanisms such as locks, semaphores, or monitors are used to coordinate access to shared resources and enforce mutual exclusion in critical sections of code.

35) What is Semaphore? Give the implementation of the Readers-Writers Problem using Semaphore.
36) Write short note: Mutual Exclusion

- Mutual exclusion is a concept in computer science and concurrent programming that ensures that only one process or thread accesses a shared resource at a time. It prevents multiple processes from simultaneously modifying the shared resource, which could lead to data inconsistency or corruption.

- In other words, mutual exclusion guarantees that if one process is accessing or modifying a resource, other processes are temporarily prevented from accessing or modifying the same resource until the first process has finished its operation and released the resource.

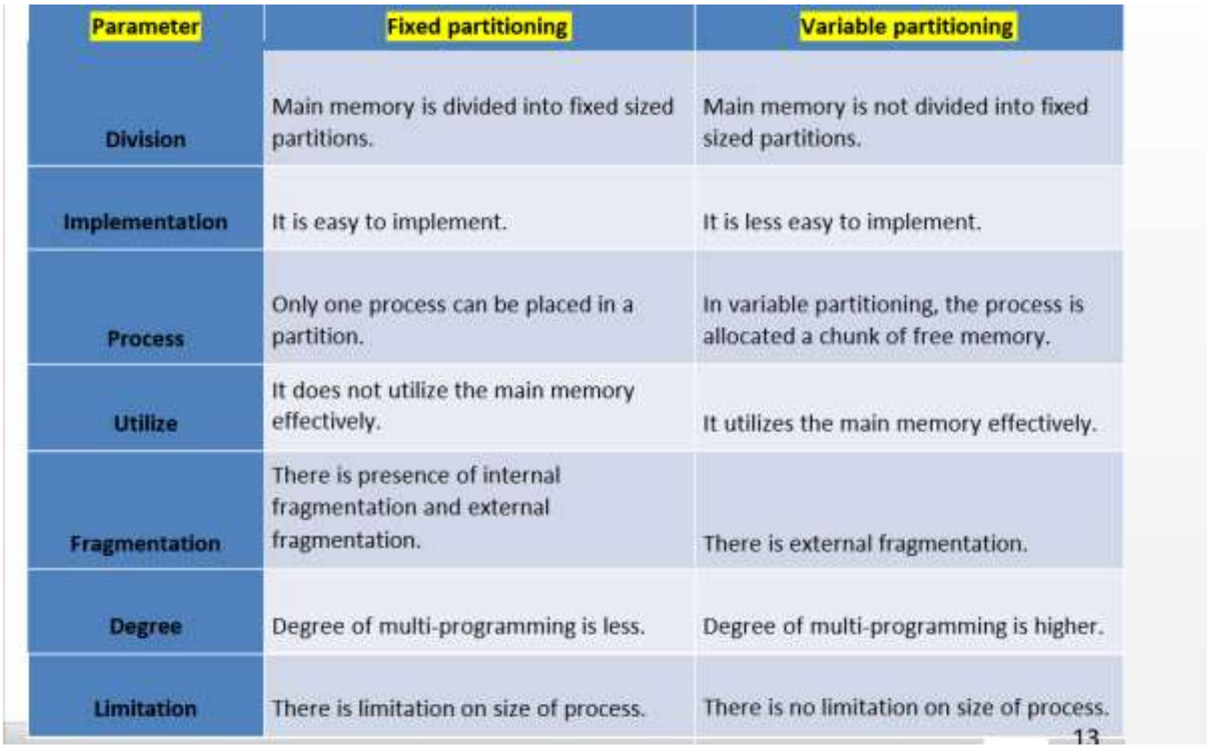
- Achieving mutual exclusion typically involves the use of synchronization mechanisms such as locks, semaphores, or mutexes. These synchronization primitives enforce a protocol where processes or threads must acquire exclusive access to the shared resource before performing any operation on it. Once a process has finished using the resource, it releases the lock or semaphore, allowing other processes to acquire access.

- Mutual exclusion is crucial for preventing race conditions and ensuring the consistency and correctness of concurrent programs where multiple processes or threads are accessing shared resources simultaneously.

37) Discuss Peterson‟s solution for the race condition with an algorithm
38) What is Virtual Memory? Explain.

- A virtual memory is technique that allows a process to execute even though it is partially loaded in main memory.
- The basic idea behind virtual memory is that the combined size of the program, data, and stack may exceed the amount of physical memory(main memory) available for it.
- The operating system keeps those parts of the program currently in use in main memory, and the rest on the disk.
- These program-generated addresses are called virtual addresses and form the virtual address space.
- **MMU (Memory Management Unit)** maps the virtual addresses onto the physical memory addresses

39) Compare Multiprogramming with Fixed Partition and multiprogramming with Variable Partitions with diagram.

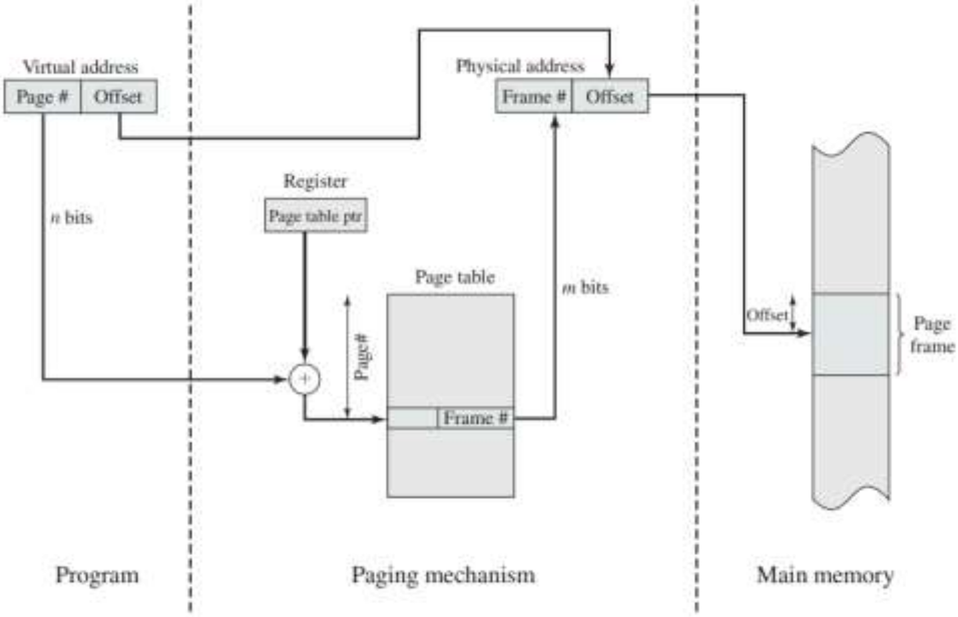


Parameter	Fixed partitioning	Variable partitioning
Division	Main memory is divided into fixed sized partitions.	Main memory is not divided into fixed sized partitions.
Implementation	It is easy to implement.	It is less easy to implement.
Process	Only one process can be placed in a partition.	In variable partitioning, the process is allocated a chunk of free memory.
Utilize	It does not utilize the main memory effectively.	It utilizes the main memory effectively.
Fragmentation	There is presence of internal fragmentation and external fragmentation.	There is external fragmentation.
Degree	Degree of multi-programming is less.	Degree of multi-programming is higher.
Limitation	There is limitation on size of process.	There is no limitation on size of process.

13

40) What is paging? What is Page Table? Explain the conversion of Virtual Address to Physical Address in Paging with example.

- Physical memory (Main Memory)is divided into fixed sized block called frames.
- Logical address space (Secondary Memory) is divided into blocks of fixed size called pages.
- Page and frame will be of same size.
- Whenever a process needs to get execute on CPU, its pages are moved from hard disk to available frame in main memory.
- Thus here **memory management task** is:
- ✓ to **find free frame** in main memory,
- ✓ **allocate appropriate frame** to the page,
- ✓ **keeping track** of which **page belong** to **which frame.**
- ▪ OS maintains a **table called page table,** for each process.
- ▪ Page table is index by page number and stores the information about frame number.

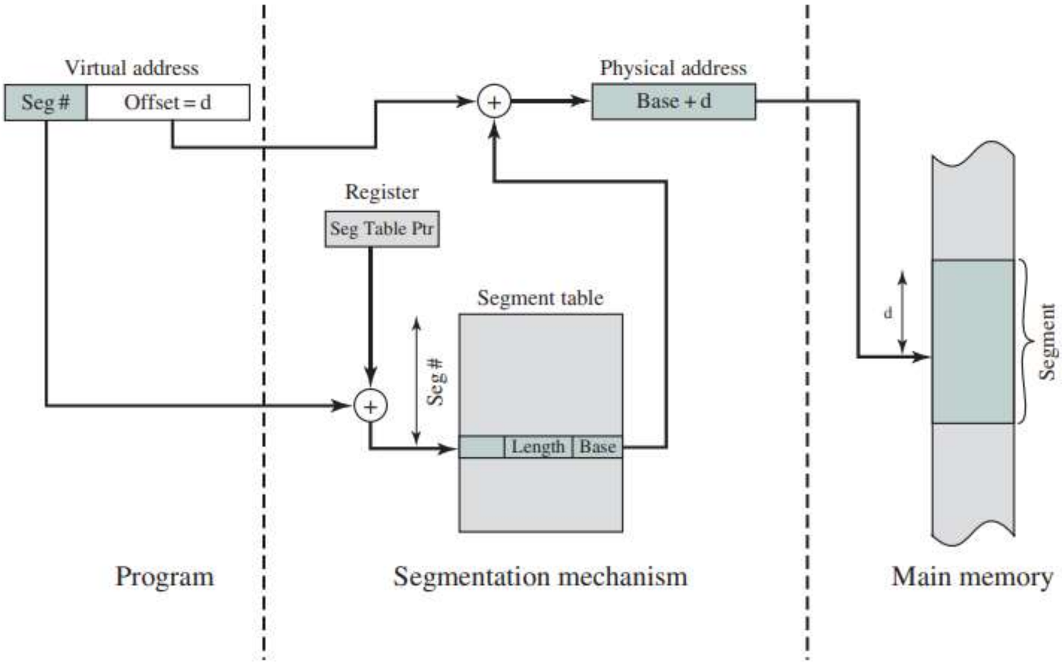


- The CPU always generates a logical address.
- In order to access the main memory always a physical address is needed.
- The **logical address generated by CPU always consists of two parts:**
- Page Number(p)
- Page Offset (d)
- **Page Number** is used to specify the specific page of the process from which the CPU wants to read the data and it is also used as an index to the page table.
- **Page offset** is mainly used to specify the specific word on the page that the CPU wants to read.
- Page table is a data structure.
- It maps the page number referenced by the CPU to the frame number where that page is stored.

41) Explain the concept of Segmentation for Memory Management. Explain why combined Paged Segmentation is used with illustration.

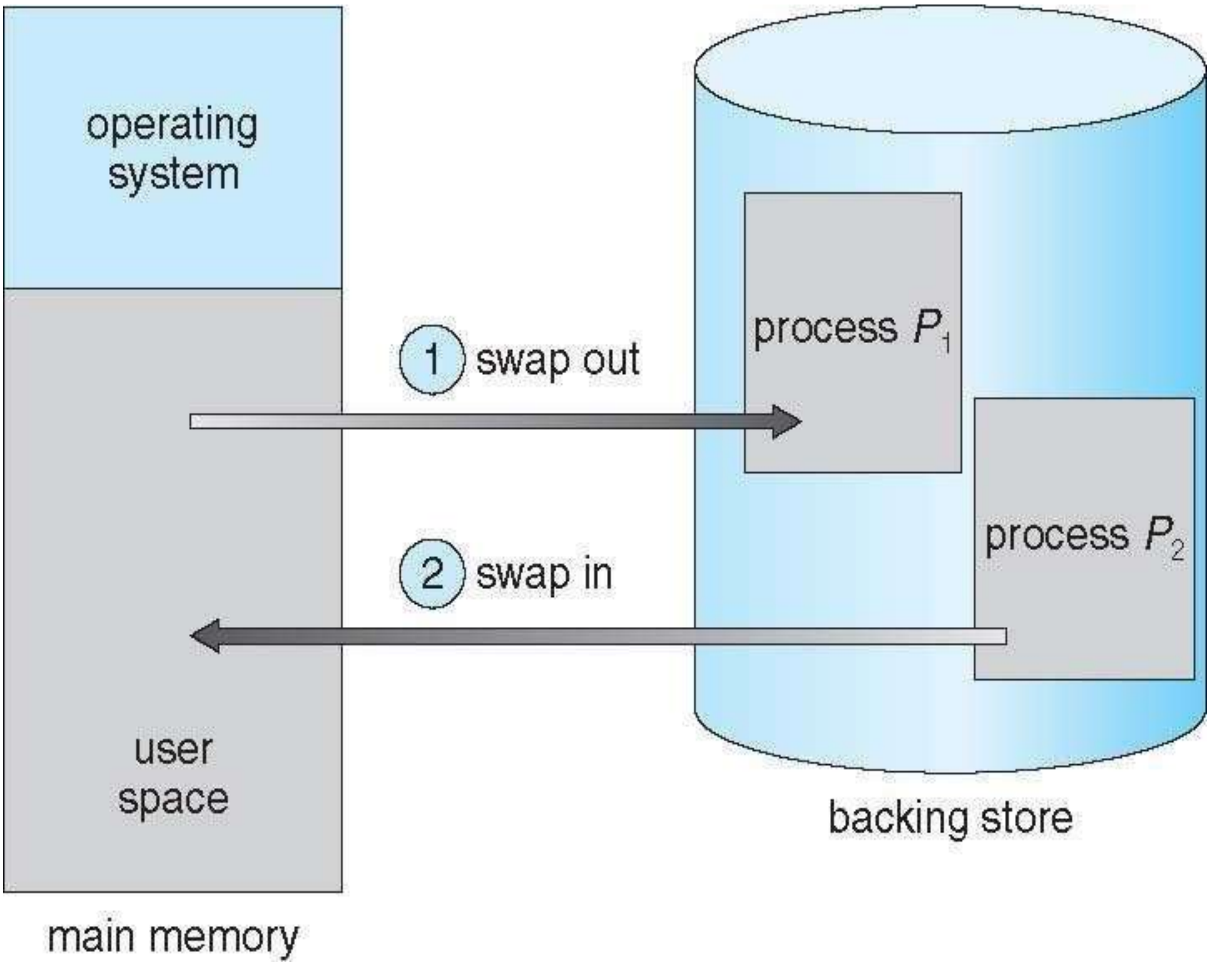
- Here the **logical address space** of a process is divided into **blocks of varying size, called segments.**
- **Each segment** contains a **logical unit** of process.
- Whenever a process is to be executed, **its segments** are moved from **secondary storage** to the **main memory.**
- **Each segment** is allocated a ***chunk of free memory*** *of the* ***size equal to that segment.***
- OS maintains **one table** known as ***segment table,*** for each process. It includes **size of segment** and **location in memory** where the segment has been loaded.
- Logical address is divided in to two parts:

1) **Segment number:** identifier for segment.
2) **Offset:** actual location within a segment.

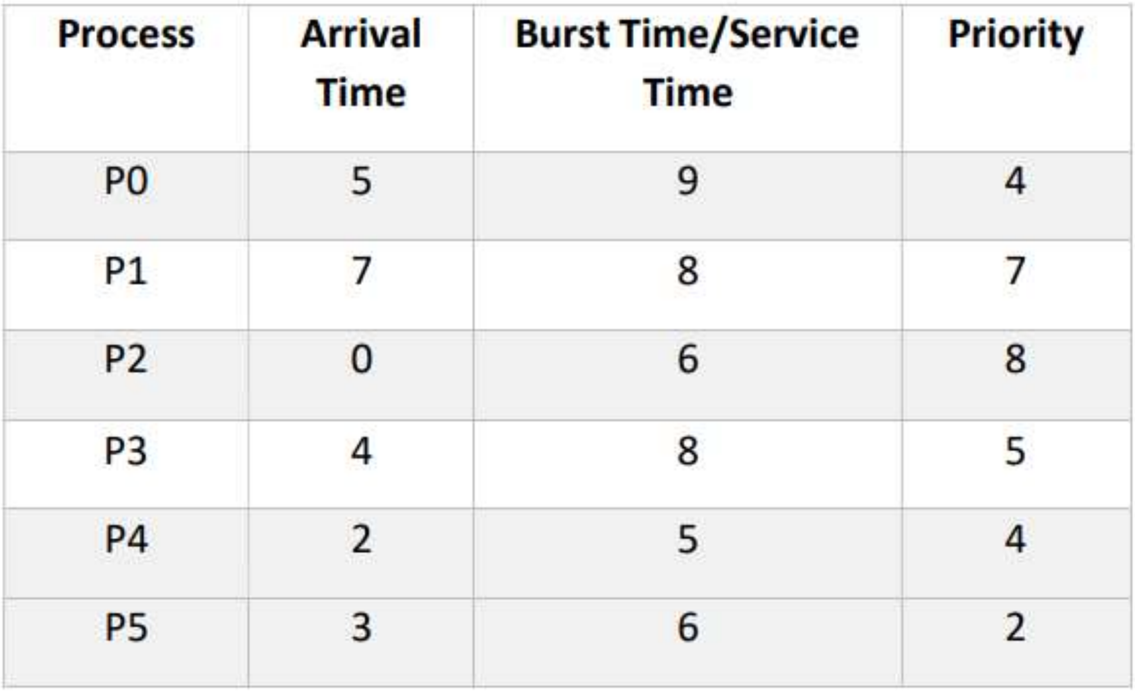


42) Explain swapping in memory management.

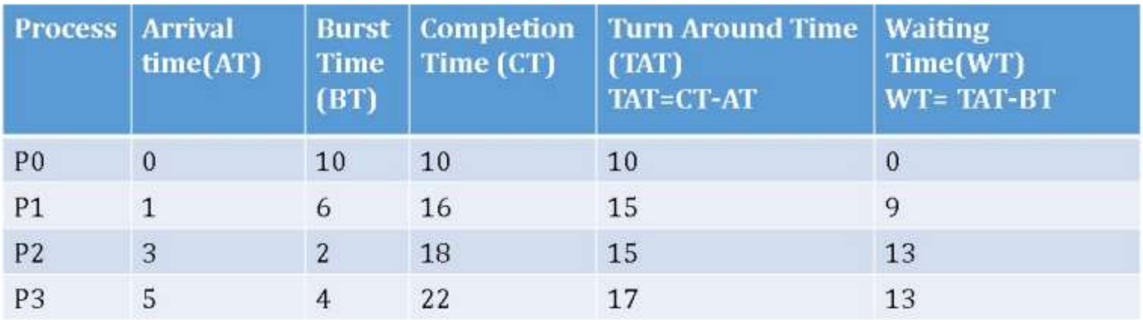
- Swapping is a memory management technique used by operating systems to efficiently utilize available physical memory (RAM) by temporarily transferring data between main memory and secondary storage (usually a hard disk or SSD). It allows the operating system to free up memory space by moving inactive or less frequently used portions of a process's memory from RAM to disk, thus making room for other processes that require memory.
- Example : A system has a physical memory of size 32-MB. Now suppose there are 5 process each having size 8MB that all want to execute simultaneously. How it is possible???
- **The solution is to use swapping.** Swapping is technique in which **process are moved** between **main memory** and **secondary memory or disk.**
- Swapping use some portion of secondary memory as backing store known as swapping area.
- **Operation of moving process** from **memory to swap area** is called **"swap out".** And **moving from swap area** to **memory** is known as **"swap in".**



43) Find out the average waiting time and average turnaround time for given example for all CPU Scheduling algorithms – FCFS, SJF,LJF, PRIORITY(Non primitive), SRTF, LRTS, ROUND ROBBIN, PRIORITY(Primitive). Note: Higher number is higher priority. Time quantum is 2 unit.



Process	Arrival Time	Burst Time/Service Time	Priority
P0	5	9	4
P1	7	8	7
P2	0	6	8
P3	4	8	5
P4	2	5	4
P5	3	6	2



Process	Arrival time(AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) TAT=CT-AT	Waiting Time(WT) WT= TAT-BT
P0	0	10	10	10	0
P1	1	6	16	15	9
P2	3	2	18	15	13
P3	5	4	22	17	13

To calculate the average waiting time and average turnaround time for each CPU scheduling algorithm, we'll go through each algorithm one by one and compute the waiting time and turnaround time for the given processes.

Given data:
- Time quantum for Round Robin: 2 units
- Process Arrival Time, Burst Time/Service Time, Priority:
  - P0: 5, 9, 4
  - P1: 7, 8, 7
  - P2: 0, 6, 8
  - P3: 4, 8, 5
  - P4: 2, 5, 4
  - P5: 3, 6, 2

FCFS (First Come First Serve)
1. Arrange processes in the order of their arrival.
2. Calculate waiting time and turnaround time for each process.
3. Average the waiting times and turnaround times.

FCFS Schedule:
```
P2 P4 P5 P3 P0 P1
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time
P2	6	0	0	6
P4	5	2	6	11
P5	6	3	11	17
P3	8	4	17	25
P0	9	5	25	34
P1	8	7	34	42

Average waiting time = (0 + 6 + 11 + 17 + 25 + 34) / 6 = 14.17 units
Average turnaround time = (6 + 11 + 17 + 25 + 34 + 42) / 6 = 20.83 units

SJF (Shortest Job First)
1. Arrange processes in ascending order of their burst time.
2. Calculate waiting time and turnaround time for each process.
3. Average the waiting times and turnaround times.

SJF Schedule:
```
P2 P4 P5 P3 P1 P0
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time
P2	6	0	0	6
P4	5	2	6	11
P5	6	3	11	17
P3	8	4	17	25
P1	8	7	25	33

P0	9	5	33	42

Average waiting time = (0 + 6 + 11 + 17 + 25 + 33) / 6 = 14.5 units
Average turnaround time = (6 + 11 + 17 + 25 + 33 + 42) / 6 = 20.67 units

LJF (Longest Job First)
LJF is essentially the opposite of SJF. We'll calculate it similarly to SJF but in descending order of burst time.

LJF Schedule:

```
P0 P1 P3 P5 P4 P2
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time
P0	9	5	0	9
P1	8	7	9	17
P3	8	4	17	25
P5	6	3	25	31
P4	5	2	31	36
P2	6	0	36	42

Average waiting time = (0 + 9 + 17 + 25 + 31 + 36) / 6 = 14.67 units
Average turnaround time = (9 + 17 + 25 + 31 + 36 + 42) / 6 = 23.33 units

We'll continue with the other algorithms in subsequent messages.

Let's calculate the average waiting time and average turnaround time for Round Robin and Priority (Primitive) scheduling algorithms.

Round Robin (RR)
Given data:
- Time quantum: 2 units
- Process Arrival Time, Burst Time/Service Time, Priority:
  - P0: 5, 9, 4
  - P1: 7, 8, 7
  - P2: 0, 6, 8
  - P3: 4, 8, 5
  - P4: 2, 5, 4
  - P5: 3, 6, 2

For Round Robin, we'll use a time quantum of 2 units.

RR Schedule:

```
P2 P4 P5 P3 P0 P1 P2 P4 P3 P0 P1 P3 P0 P1
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time
P2	6	0	14	20
P4	5	2	17	22
P5	6	3	18	24
P3	8	4	24	32
P0	9	5	30	39
P1	8	7	36	44

Average waiting time = (14 + 17 + 18 + 24 + 30 + 36) / 6 = 23.17 units
Average turnaround time = (20 + 22 + 24 + 32 + 39 + 44) / 6 = 29.83 units

Priority (Primitive)

In this algorithm, the process with the highest priority is executed first.

Given data:
- Priority:
  - P0: 4
  - P1: 7
  - P2: 8
  - P3: 5
  - P4: 4
  - P5: 2

Priority Schedule:

```
P5 P0 P4 P3 P1 P2
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Priority	Waiting Time	Turnaround Time
P5	6	3	2	0	6
P0	9	5	4	6	15
P4	5	2	4	15	20
P3	8	4	5	20	28
P1	8	7	7	28	36

P2	6	0	8	36	42

Average waiting time = (0 + 6 + 15 + 20 + 28 + 36) / 6 = 17.5 units
Average turnaround time = (6 + 15 + 20 + 28 + 36 + 42) / 6 = 23.167 units

These are the average waiting time and average turnaround time for Round Robin and Priority (Primitive) scheduling algorithms.

Let's calculate the average waiting time and average turnaround time for Priority (Non-primitive), Shortest Remaining Time First (SRTF), and Longest Remaining Time Slice (LRTS) scheduling algorithms.

Priority (Non-primitive)
In this algorithm, the priority is given to processes with higher priority. If two processes have the same priority, the process that arrives first gets executed first.

Given data:
- Process Arrival Time, Burst Time/Service Time, Priority:
  - P0: 5, 9, 4
  - P1: 7, 8, 7
  - P2: 0, 6, 8
  - P3: 4, 8, 5
  - P4: 2, 5, 4
  - P5: 3, 6, 2

Priority Schedule:
```
P2 P4 P5 P3 P0 P1
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Priority	Waiting Time	Turnaround Time
P2	6	0	8	0	6
P4	5	2	4	6	11
P5	6	3	2	11	17
P3	8	4	5	17	25
P0	9	5	4	25	34
P1	8	7	7	34	42

Average waiting time = (0 + 6 + 11 + 17 + 25 + 34) / 6 = 14.17 units
Average turnaround time = (6 + 11 + 17 + 25 + 34 + 42) / 6 = 20.83 units

Shortest Remaining Time First (SRTF)
In SRTF, the process with the shortest remaining burst time is scheduled next.

SRTF Schedule:
```
P2 P4 P5 P3 P0 P1
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time
P2	6	0	0	6
P4	5	2	6	11
P5	6	3	11	17
P3	8	4	17	25
P0	9	5	25	34
P1	8	7	34	42

Average waiting time = (0 + 6 + 11 + 17 + 25 + 34) / 6 = 14.17 units
Average turnaround time = (6 + 11 + 17 + 25 + 34 + 42) / 6 = 20.83 units

Longest Remaining Time Slice (LRTS)
In LRTS, the process with the longest remaining burst time is scheduled next.

LRTS Schedule:
```
P0 P1 P3 P5 P4 P2
```

Calculating waiting time and turnaround time:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time
P0	9	5	0	9
P1	8	7	9	17
P3	8	4	17	25
P5	6	3	25	31
P4	5	2	31	36
P2	6	0	36	42

Average waiting time = (0 + 9 + 17 + 25 + 31 + 36) / 6 = 14.67 units
Average turnaround time = (9 + 17 + 25 + 31 + 36 + 42) / 6 = 23.33 units

These are the average waiting time and average turnaround time for Priority (Non-primitive), Shortest Remaining Time First (SRTF), and Longest Remaining Time Slice (LRTS) scheduling algorithms.