OS Mid 1

OS Important Questions

Multiple-Choice Questions:

1. To access the services of the operating system, the interface is provided by the ___
   - [ ] a) Library
   - [x] b) System calls
   - [ ] c) Assembly instructions
   - [ ] d) API
2. Network operating system runs on ___
   - [ ] a) every system in the network
   - [x] b) server
   - [ ] c) both
   - [ ] d) none
3. Which one of the following is not a real-time operating system?
   - [ ] a) RTLinux
   - [x] b) Palm OS
   - [ ] c) QNX
   - [ ] d) VxWorks
4. Which of these doesn't interrupt a running process?
   - [ ] a) Power failure
   - [x] b) Scheduler process
   - [ ] c) Timer
   - [ ] d) A device

5. Which one of these statements about kernel-level threads and user-level threads is FALSE?
   - [x] a) The context switch time is comparatively longer for the kernel-level threads, as compared to the ones for the user-level threads.
   - [ ] b) The user-level threads don't require any support for hardware.
   - [ ] c) We can schedule the related kernel-level threads in a multi-processor system on different processors.
   - [ ] d) When one kernel-level thread is blocked, then all the related threads will be blocked.
6. If a process fails, most operating systems write the error information to a __
   - [ ] a) new file
   - [ ] b) another running process
   - [x] c) log file
   - [ ] d) none
7. CPU scheduling is the basis of ___
   - [x] a) multiprogramming OS
   - [ ] b) larger memory-sized systems
   - [ ] c) multiprocessor systems
   - [ ] d) none of the mentioned
8. In a timeshare operating system, when the time slot assigned to a process is completed, the process switches from the current state to?
   - [ ] a) Suspended state
   - [ ] b) Terminated state
   - [x] c) Ready state
   - [ ] d) Blocked state

9. Cascading termination refers to the termination of all child processes if the parent process terminates __
   - [x] a) Normally or abnormally
   - [ ] b) Abnormally
   - [ ] c) Normally
   - [ ] d) None of the mentioned
10. The FCFS algorithm is particularly troublesome for___

- [ ] a) operating systems
- [ ] b) multiprocessor systems
- [x] c) time-sharing systems
- [ ] d) multiprogramming systems

11. A deadlock avoidance algorithm dynamically examines the _ _ to ensure that a circular wait condition can never exist.
   - [ ] a) operating system
   - [ ] b) resources
   - [ ] c) system storage state
   - [x] d) resource allocation state
12. Process synchronization can be done on
   - [x] a) Hardware level
   - [x] b) Software Level
   - [ ] c) Other
   - [ ] d) None
13. Out of the following, which one needs a device driver?
   - [ ] a) Main memory
   - [x] b) Disk
   - [ ] c) Register
   - [ ] d) Cache
14. What is the main function of the command interpreter?

- [ ] a) to provide the interface between the API and application program
- [ ] b) to handle the files in the operating system
- [x] c) to get and execute the next user-specified command
- [ ] d) none of the mentioned

15. In Unix, which system call creates the new process?
    - [ ] a) create
    - [x] b) fork
    - [ ] c) new
    - [ ] d) none of the mentioned
16. The total time taken for switching between the user and the kernel modes of execution is t1, while the total time taken for switching between two processes is t2. Out of the following, which one is TRUE?
    - [x] a) t2 > t1
    - [ ] b) t2 = t1
    - [ ] c) t2 < t1
    - [ ] d) We cannot say anything about the relation between t2 and t1
17. Process synchronization can be done on
    - [x] a) Hardware level
    - [x] b) Software Level
    - [ ] c) Other
    - [ ] d) None
18. Which of the following is a synchronization tool
    - [ ] a) Thread
    - [ ] b) Pipe
    - [x] c) Semaphore
    - [ ] d) Sockets

19. An operating system can be designed to be either volatile or non-volatile.
20. Multiprocessor is also called as Parallel Processor.
21. _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ is usually used to provide high availability service.
22. Applications must be written specifically to take advantage of the cluster by using parallel processing technique.
23. No. of processes completed per unit time is known as throughput.
24. Entry of all the PCBS of the current process is in Process Table.
25. The context of the process in PCB of a process doesn't contain actual data and context switch time.
26. A Parent process calling wait() system call will be suspended until the children process terminates.
27. The wait function decreases the semaphore value.
28. Semaphores are mostly used to implement interprocess communication (IPC) mechanisms.
29. The link between processes P and Q to send and receive messages is called communication link.
30. Bounded capacity and Unbounded capacity queues are referred to as Automatic buffering.
31. An operating system halts the execution of the user to prevent errors and improper use of the computer.
32. The interaction between the user and hardware is called The operating system (OS).
33. The operating System is one program running at all times on the computer usually called as the kernel.
34. The occurrence of an event is usually signaled by an Interrupt from either the hardware or the software.

35. General-purpose computers run most of their programs from rewritable memory called random-access memory (RAM) [main memory].
36. Time-sharing operating system requires an interactive computer system that provides direct communication between the user and the system.
37. In UNIX each process is identified by its Process Identifier (PID).

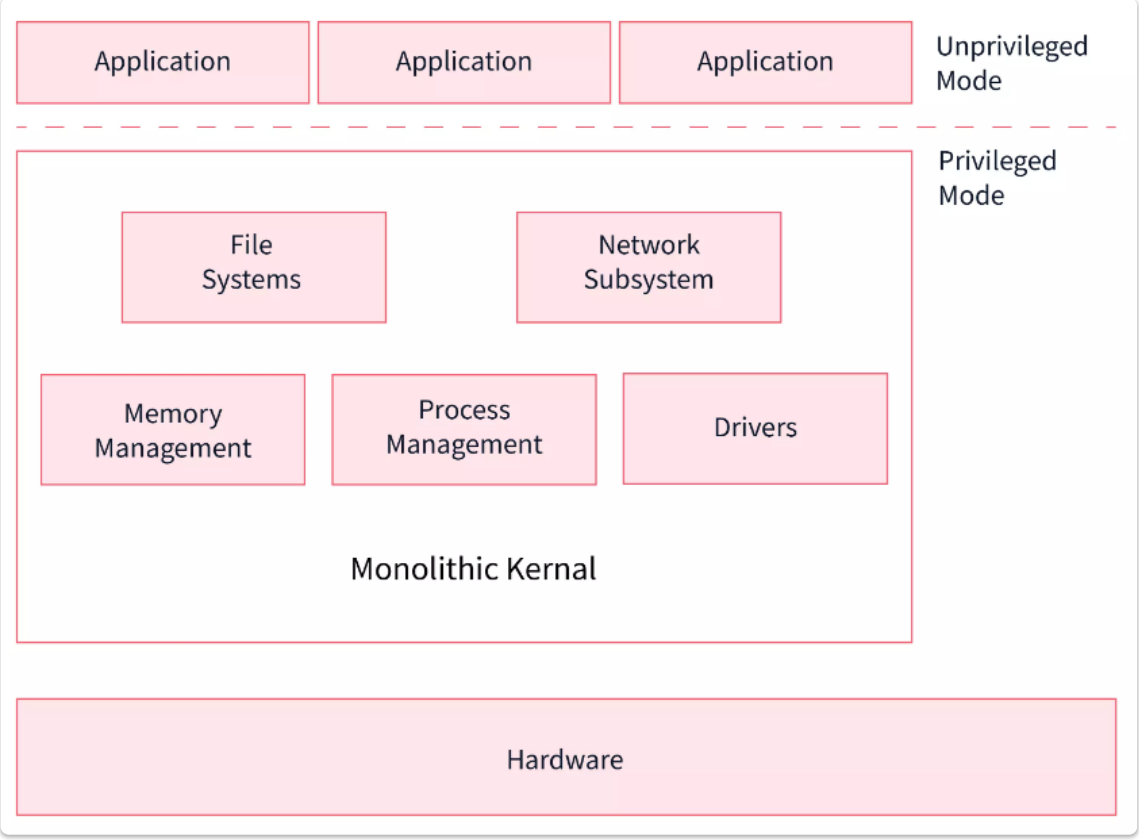
PART – B

1. What are the various components of the operating system structure and explain with a neat sketch.

1. Kernel:
    - The brain of the operating system.
    - Manages tasks like running programs, handling memory, and talking to devices.
2. Process Management:
    - Takes care of running programs.
    - Decides which program gets to use the computer's resources.
3. Memory Management:
    - Deals with computer memory (RAM).
    - Makes sure programs have enough space to run.
4. File System Management:
    - Organizes and keeps track of files.
    - Handles creating, deleting, and finding files.
5. Device Drivers:
    - Helps the operating system talk to hardware like printers or keyboards.

- Manages input and output operations.

6. **Security and Protection:**
   - Keeps the system safe.
   - Controls who can access the computer and what they can do.
7. **User Interface:**
   - How you interact with the computer.
   - Can be graphical (with icons and windows) or text-based (using commands).
8. **Networking:**
   - Lets computers talk to each other.
   - Handles things like internet connections and data transfer.
9. **System Libraries:**
   - Pre-made tools for programs.
   - Helps programs do common tasks without starting from scratch.



2. Explain distributed OS and time-sharing OS in detail.

Distributed Operating System

Distributed Operating System:

- **Description:** Distributed operating systems are a recent advancement in computer technology. Various interconnected computers communicate using a shared communication network. Independent systems have their memory units and CPUs, forming loosely coupled or distributed systems.
- **Types of Distributed Systems:**
  - **Client/Server Systems:** Clients request resources, and servers provide them. Servers can serve multiple clients simultaneously, and they communicate via a computer network.
  - **Peer-to-Peer Systems:** Nodes in these systems share resources and tasks equally, interacting via a network.
- **Advantages:**
  - Failure of one system doesn't affect network communication.
  - Enables remote access to files and software from other networked systems.
  - Enhances data exchange speed, computation speed, and scalability.
  - Reduces the load on host computers and minimizes data processing delays.
- **Disadvantages:**
  - Network failure can disrupt the entire communication.
  - Lack of well-defined languages for establishing distributed systems.

- High cost and complexity, including the underlying software.

- **Example:** LOCUS

Time-Sharing Operating Systems

Types of Operating Systems (Continued):

5. **Time-Sharing Operating Systems:**
   - **Description:** Time-sharing operating systems allocate specific time slices (quantums) to each task, ensuring all tasks run smoothly. Each user gets a share of the CPU's time in these multitasking systems.
   - **Advantages:**
     - Equal opportunities for all tasks.
     - Reduced chances of software duplication.
     - Minimized CPU idle time.
     - Efficient resource sharing, reducing hardware costs.
     - Improved productivity and user experience.
   - **Disadvantages:**
     - Reliability issues.
     - Security and data integrity concerns.
     - Data communication problems.
     - Higher overhead and complexity.
     - Increased security risks.
   - **Examples:**
     - IBM VM/CMS.
     - TSO (Time Sharing Option) by IBM.
     - Windows Terminal Services.

3. Consider 3 processes P1, P2, and P3, which require 5, 7, and 4 time units and arrive at times 0, 1, and 3.

Draw the Gantt chart, process completion sequence, and average waiting time for.
i) Round-robin scheduling with CPU quantum of 2 time units.
ii) FCFS

Round-Robin Scheduling

Gantt chart

Time	P1	P2	P3	Burst Time P1	Burst Time P2	Burst Time P3
0				5	7	4
0-2	P1			3	7	4
2-4		P2		3	5	4
4-6	P1			1	5	4
6-8			P3	1	5	2
8-10		P2		1	3	2
10-11	P1			0	3	2
11-13			P3	-	3	0
13-15		P2		-	1	-
15-16		P2		-	0	-

$$\text{Waiting Time} = \text{Completion Time} - \text{Arrival Time} - \text{Burst Time}$$

Process	Arrival	Burst Time	Completion Time	Waiting Time
P1	0	5	11	6

Process	Arrival	Burst Time	Completion Time	Waiting Time
P2	1	7	13	5
P3	3	4	16	9

Now, let's calculate the average waiting time:

$$\text{Average Waiting Time} = \frac{\text{Sum of Waiting Times}}{\text{Number of Processes}}$$

$$\text{Average Waiting Time} = \frac{6+5+9}{3} = \frac{20}{3}$$

Average Waiting Time (RR): $\frac{20}{3}$

FCFS (First-Come, First-Served) Scheduling

Gantt chart

Time	P1	P2	P3
0-5	P1		
5-12		P2	
12-16			P3

$$\text{Waiting Time} = \text{Completion Time} - \text{Arrival Time} - \text{Burst Time}$$

Process	Arrival	Burst Time	Completion Time	Waiting Time
P1	0	5	5	0
P2	1	7	12	4
P3	3	4	16	9

Now, let's calculate the average waiting time:

$$\text{Average Waiting Time} = \frac{\text{Sum of Waiting Times}}{\text{Number of Processes}}$$

$$\text{Average Waiting Time} = \frac{0+4+9}{3} = \frac{13}{3}$$

Average Waiting Time (FCFS): $\frac{13}{3}$ time units

4. What is Deadlock? List the condition that leads to deadlock. How deadlock can be prevented.

DEADLOCK IN OPERATING SYSTEMS

In operating systems, a deadlock is a critical situation where a set of processes is blocked because each process is holding a resource and waiting for another resource acquired by some other process. This situation creates a standstill, much like two trains coming towards each other on the same track with only one track available.

Examples of Deadlock:

1. **Two Tape Drives:**
   - **Scenario:**
     - The system has two tape drives.
     - Processes P1 and P2 each hold one tape drive and need another one.
2. **Semaphores A and B:**
   - **Scenario:**
     - Semaphores A and B, initialized to 1.
     - P0 executes wait(A) and preempts.
     - P1 executes wait(B).
     - Now P0 and P1 enter a deadlock.
3. **Space Allocation:**
   - **Scenario:**
     - The space available for allocation is 200KB.
     - Processes P0 and P1 engage in a deadlock while making requests for space.

Necessary Conditions for Deadlock:

1. **Mutual Exclusion:**
    - Two or more resources are non-shareable, meaning only one process can use a resource at a time.
2. **Hold and Wait:**
    - A process is holding at least one resource and waiting for additional resources.
3. **No Pre-emption:**
    - A resource cannot be taken from a process unless the process willingly releases the resource.
4. **Circular Wait:**
    - A set of processes is waiting for each other in a circular form, creating a cycle of dependencies.

5. Write short notes on the process. Explain various states of the process with a neat diagram.

Process Concepts and Scheduling

The term "process" was first used by the designers of the Multics system in the 1960s. A process is a program in execution, and process execution must progress sequentially. Processes exist for a limited span of time.

Two or more processes could be executing the same program, each using their data and resources. The process memory is divided into four sections for efficient operation:

- **Text:** Composed of integrated program code, which is read from fixed storage when the program is launched.
- **Data:** Made up of global and static variables, distributed and executed before the main action.

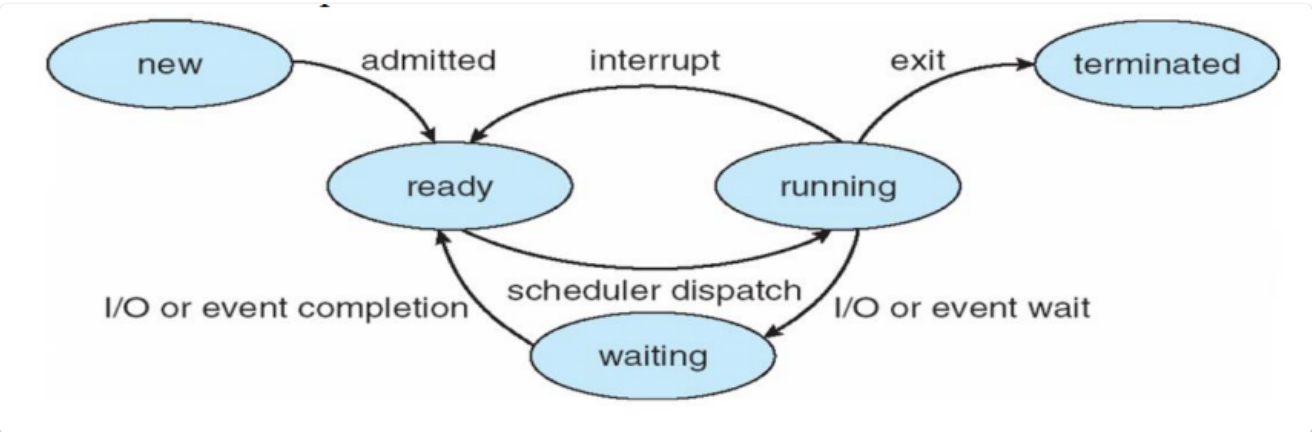
- **Heap:** Used for flexible or dynamic memory allocation and managed by calls to new, delete, malloc, free, etc.
- **Stack:** Used for local variables, with space reserved for local variables when announced.

Process State

When a process executes, it changes state. The process state is defined as the current activity of the process, and it contains five states. Each process is one of the following states:

- **New:** The process is being created.
- **Running:** Instructions are being executed.
- **Waiting:** The process is waiting for some event to occur.
- **Ready:** The process is waiting to be assigned to a processor.
- **Terminated:** The process has finished execution.

Diagram of Process State



6. Explain the concepts of semaphores and critical section problems?

CRITICAL SECTION PROBLEM

A critical section is a code segment accessible by only one process at a time. It contains shared variables that need synchronization to maintain data consistency. The critical section problem involves designing a way for cooperative processes to access shared resources without creating data inconsistencies.

Entry Section

In the entry section, a process requests entry into the critical section.

Any solution to the critical section problem must satisfy three requirements:

1. **Mutual Exclusion:**
    - If a process is executing in its critical section, no other process is allowed to execute in the critical section.
2. **Progress:**
    - If no process is executing in the critical section, and other processes are waiting outside the critical section, only those processes not executing in their remainder section can participate in deciding who will enter the critical section next. The selection can't be postponed indefinitely.
3. **Bounded Waiting:**
    - A bound must exist on the number of times other processes are allowed to enter their critical sections after a process has made a request and before that request is granted.

SEMAPHORES

Semaphore is a hardware solution given to the critical section problem, represented as a normal integer. It cannot be negative,

with the minimum value being zero. Semaphores typically have two operations:

Semaphore Operations:

1. **Wait ( ) Operation:**
   - If the Semaphore value is greater than zero, the process can enter the critical section.
   - If the Semaphore value is zero, the process has to wait.
   - The Semaphore value is reduced when the process exits the critical section.

Definition of wait()

```
wait(Semaphore S)
{
  while (S<=0) ;        //no operation
  S--;
}
```

2. **Signal ( ) Operation:**
   - Executed only when the process exits the critical section.
   - The Semaphore value cannot be incremented before the process exits the critical section.

Definition of signal()

```
signal(S)
{
  S++;
}
```

7. Write about deadlock conditions and the banker's algorithm in detail.

> Necessary Conditions for Deadlock:
>
> 1. **Mutual Exclusion:**
>   - Two or more resources are non-shareable, meaning only one process can use a resource at a time.
> 2. **Hold and Wait:**
>   - A process is holding at least one resource and waiting for additional resources.
> 3. **No Pre-emption:**
>   - A resource cannot be taken from a process unless the process willingly releases the resource.
> 4. **Circular Wait:**
>   - A set of processes is waiting for each other in a circular form, creating a cycle of dependencies.

BANKER'S ALGORITHM

> BANKER'S ALGORITHM
>
> The Banker's Algorithm is used to avoid deadlock and allocate resources safely to each process in a computer system. It examines all possible tests or activities in the 'S-State' before deciding whether the allocation should be allowed to each process. Named after its analogy to a banker approving a loan, it helps the operating system share resources among processes successfully.
>
> #### Working Principle
>
> - The algorithm checks whether a person should be sanctioned a loan amount to simulate allocation resources

safely.

- It manages resources in a way similar to a bank system, ensuring that a person's request for a loan is approved based on available resources.

Operating System Application

- In an operating system, when a new process is created, it provides information about upcoming processes, resource requests, counts, and delays.
- The operating system uses this information to decide the execution sequence of processes, avoiding deadlock scenarios.

Banker's Algorithm Requirements

When working with the banker's algorithm, it requires information about three things:

1. **Maximum Request:**
    - How much each process can request for each resource in the system. Denoted by [MAX] request.
2. **Currently Allocated Resources:**
    - How much each process is currently holding each resource in the system. Denoted by [ALLOCATED] resource.
3. **Available Resources:**
    - The number of each resource currently available in the system. Denoted by [AVAILABLE] resource.

Data Structures

The banker's algorithm uses several data structures:

1. **Available:**

- An array defining each type of resource available in the system.

2. **Max:**
   - A matrix indicating each process's maximum resource requirements.
3. **Allocation:**
   - A matrix indicating the type of resources currently allocated to each process.
4. **Need:**
   - A matrix representing the number of remaining resources for each process.
5. **Finish:**
   - A vector indicating whether a process has been allocated requested resources and has released them after finishing its task.

The Banker's Algorithm combines the safety algorithm and the resource request algorithm to control processes and avoid deadlock.

8. Write short notes on PCB and context switching?

9. Explain about a race condition? How to avoid a race condition? Explain any two methods to avoid a race condition.

RACE CONDITION

A race condition occurs when multiple processes share data, access the data concurrently, and the output of execution depends on the specific sequence in which they share and access the data.

Definition

When more than one process executes the same code or accesses the same memory or shared variable simultaneously, there is a possibility of incorrect output or value of the shared variable. In this condition, processes race to claim the correctness of their output.

Example

Consider two processes, P1 and P2, sharing a common variable (shared = 10). Both processes are in the ready queue, waiting for execution. The processes manipulate the shared variable concurrently, and the outcome depends on the order of access.

1. Process P1 initializes X=10 and increments it by 1 (X=11). It then goes into waiting for 1 second.
2. CPU switches to Process P2, which initializes Y=10 and decrements it by 1 (Y=9). P2 goes into waiting for 1 second, leaving the CPU idle.
3. After 1 second, CPU resumes Process P1, increments X, and shared becomes 11.
4. After another second, CPU resumes Process P2, decrements Y, and shared becomes 9.

Actual Meaning of Race-Condition

- If the order of execution is P1 first, then P2, the final shared value is 9.
- If the order of execution is P2 first, then P1, the final shared value is 11.

Avoiding Race Conditions

To avoid race conditions, proper synchronization is essential. Two methods to avoid race conditions are:

1. Mutual Exclusion:

- Ensure that only one process can access the shared data or resource at a time. This prevents simultaneous modifications and guarantees consistency.

2. Locks and Semaphores:

- Use locks or semaphores to control access to shared resources. Processes acquire a lock before accessing the resource and release it afterward. This ensures exclusive access.

10. List out the types of operating systems and explain batch OS and Real-time sharing OS in brief.

Types of Operating Systems:

- Batch Operating System
- Multi-Programming Operating System
- Multi-Processing Operating System
- Multi-Tasking Operating System
- Time-Sharing Operating Systems
- Personal Computer
- Parallel Operating System
- Distributed Operating System
- Network Operating System

Batch Operating System

Batch Operating System:

- **Description:** In this type of operating system, the computer doesn't interact directly with the user. Instead, an operator groups similar jobs with the same requirements into batches, and the operator is responsible for sorting these jobs.
- **Advantages:**
  - Predictable job durations.
  - Multiple users can share the system.
  - Minimal idle time.
  - Efficient for handling repetitive tasks.
- **Disadvantages:**
  - Requires knowledgeable operators.
  - Challenging to debug.
  - Can be costly.
  - If a job fails, other jobs have to wait.
  - Difficult to estimate job completion times.
- **Examples:** Payroll systems, bank statements, etc.

Real-Time System (RTOS)

Real-Time Operating System (RTOS):

- **Description:** Real-time operating systems are designed for real-time systems where processing and responding to inputs must occur within strict time constraints. These systems are used in applications like missile systems, air traffic control, and robots.
- **Types:**
- **Hard Real-Time Systems:** Meant for applications where time constraints are extremely strict, and even the shortest delay is

unacceptable. Used in life-saving applications like automatic parachutes or airbags.
- Soft Real-Time Systems: Intended for applications with less strict time constraints.
- Advantages:
- Maximum resource utilization and task shifting.
- Focus on running applications and error-free operation.
- Efficient memory allocation and suitability for embedded systems.
- Disadvantages:
- Limited multitasking capability and heavy system resource usage.
- Complex algorithms and specific device drivers are required.
- Inflexible thread priority and less task switching.
- Examples: Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

11. What is a System call? Explain the various types of system calls provided by an operating system.

System Calls

A system call is a fundamental interface that allows user programs to interact with the operating system. These calls enable user programs to request services from the operating system's kernel. System calls are essential for various operations, such as file handling, network communication, process management, and hardware access. Here are some key points about system calls:

1. User-OS Interaction:

- A system call serves as a bridge between user programs and the operating system.
- User programs request specific services, and the operating system responds by invoking a series of system calls to fulfill these requests.

2. **Programming Languages:**
   - System calls can be written in low-level languages like assembly or high-level languages such as C, C++, or Pascal.
   - When using high-level languages, system calls are often predefined functions that the operating system can directly invoke.
3. **Kernel Interaction:**
   - A system call allows a computer program to request a service from the kernel of the operating system.
   - These requests include tasks like file operations, process control, and hardware access.
4. **Interaction Method:**
   - System calls are a means of interacting with the operating system through user programs.
   - They provide a structured way for programs to access essential services provided by the OS.
5. **Request Mechanism:**
   - When software needs to access the kernel's services, it initiates a system call.
   - System calls rely on an Application Program Interface (API) to expose the operating system's services to user-level programs.
6. **Execution Time:**

- System call execution time varies based on the complexity of the task.
- Simple system calls, like retrieving the system date and time, can complete in a few nanoseconds.
- More complex operations, such as connecting to a network device, may take a few seconds.
- To avoid bottlenecks, most modern operating systems employ multi-threading to handle multiple system calls simultaneously.

Types of System Calls

There are commonly five types of system calls. These are as follows:

1. Process Control
2. File Management
3. Device Management
4. Information Maintenance
5. Communication

Process Control

Process control is the system call that is used to direct the processes. Some process control examples include creating, loading, aborting, ending, executing, processing, terminating the process, etc.

File Management

File management is a system call that is used to handle files. Some file management examples include creating files, deleting files, opening, closing, reading, writing, etc.

Device Management

Device management is a system call that is used to deal with devices. Some examples of device management include reading devices, writing, getting device attributes, releasing devices, etc.

Information Maintenance

Information maintenance is a system call that is used to maintain information. Some examples of information maintenance include getting system data, setting time or date, getting time or date, setting system data, etc.

Communication

Communication is a system call that is used for communication. Some examples of communication include creating, deleting communication connections, sending, receiving messages, etc.

Examples of Windows and Unix system calls

Process	Windows	Unix
Process Control	CreateProcess()	Fork()
	ExitProcess()	Exit()
	WaitForSingleObject()	Wait()
File Manipulation	CreateFile()	Open()
	ReadFile()	Read()
	WriteFile()	Write()
	CloseHandle()	Close()
Device Management	SetConsoleMode()	Ioctl()
	ReadConsole()	Read()
	WriteConsole()	Write()
Information Maintenance	GetCurrentProcessID()	Getpid()

Process	Windows	Unix
	SetTimer()	Alarm()
	Sleep()	Sleep()
Communication	CreatePipe()	Pipe()
	CreateFileMapping()	Shmget()
	MapViewOfFile()	Mmap()
Protection	SetFileSecurity()	Chmod()
	InitializeSecurityDescriptor()	Umask()
	SetSecurityDescriptorgroup()	Chown()

12. Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P1	27	5
P2	12	1
P3	37	2
P4	19	4
P5	10	3

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0.
Draw the Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, and Priority. Also, determine the average waiting time and average turnaround time for each of the algorithms.

FCFS (First-Come, First-Served) Scheduling:

Gantt Chart:

P1	P2	P3	P4	P5
0	27	39	76	95

Process	Burst Time	Arrival Time	Completion Time	Turnaround Time	Waiting Time
P1	27	0	27	27	0
P2	12	0	39	39	27
P3	37	0	76	76	39
P4	19	0	95	95	76
P5	10	0	105	105	95

Average Waiting Time:

$$\text{Average Waiting Time} = \frac{0+27+39+76+95}{5} = \frac{237}{5} = 47.4$$

Average Turnaround Time:

$$\text{Average Turnaround Time} = \frac{105+12+49+78+59}{5} = \frac{303}{5} = 60.6$$

SJF (Shortest Job First) Scheduling:

Gantt Chart:

P2	P5	P4	P3	P1
0	12	22	61	98

Process	Burst Time	Arrival Time	Completion Time	Turnaround Time	Waiting Time
P5	10	0	10	10	0
P2	12	0	22	22	10
P4	19	0	41	41	22
P1	27	0	68	68	41
P3	37	0	105	105	68

$$\text{Average Waiting Time} = \frac{\text{Sum of Waiting Times}}{\text{Number of Processes}}$$

$$\text{Average Waiting Time} = \frac{0+10+22+41+68}{5} = \frac{141}{5} = 28.2$$

$$\text{Average Turnaround Time} = \frac{\text{Sum of Turnaround Times}}{\text{Number of Processes}}$$
$$\text{Average Turnaround Time} = \frac{10+22+41+68+105}{5} = \frac{246}{5} = 49.2$$

So, the Average Waiting Time is 28.2 time units, and the Average Turnaround Time is 49.2 time units for SJF without priority.

Priority Scheduling:

Assuming lower priority numbers indicate higher priority:

Gantt Chart:

P2	P3	P5	P4	P1
0	12	49	88	105

1. **P2 (Priority 1):**
   - Completion Time: 12 (Burst Time)
   - Turnaround Time: 12 - 0 = 12
   - Waiting Time: 12 - 12 = 0
2. **P3 (Priority 2):**
   - Completion Time: 12 + 37 = 49
   - Turnaround Time: 49 - 0 = 49
   - Waiting Time: 49 - 37 = 12
3. **P5 (Priority 3):**
   - Completion Time: 49 + 10 = 59
   - Turnaround Time: 59 - 0 = 59
   - Waiting Time: 59 - 10 = 49
4. **P4 (Priority 4):**
   - Completion Time: 59 + 19 = 78
   - Turnaround Time: 78 - 0 = 78
   - Waiting Time: 78 - 19 = 59
5. **P1 (Priority 5):**
   - Completion Time: 78 + 27 = 105

- Turnaround Time: 105 - 0 = 105
- Waiting Time: 105 - 27 = 78

Process	Burst Time	Priority	Arrival Time	Completion Time	Turnaround Time	Waiting Time
P1	27	5	0	105	105	78
P2	12	1	0	12	12	0
P3	37	2	0	49	49	12
P4	19	4	0	78	78	59
P5	10	3	0	59	59	49

$$\text{Average Waiting Time} = \frac{\text{Sum of Waiting Times}}{\text{Number of Processes}}$$
$$\text{Average Turnaround Time} = \frac{\text{Sum of Turnaround Times}}{\text{Number of Processes}}$$

$$\text{Average Waiting Time} = \frac{78+0+12+59+49}{5} = \frac{198}{5} = 39.6$$
$$\text{Average Turnaround Time} = \frac{105+12+49+78+59}{5} = \frac{303}{5} = 60.6$$

So, the Average Waiting Time is 39.6 time units, and the Average Turnaround Time is 60.6 time units for the provided SJF table with priority.