# Assignment 2

## PART D

**1. What is an operating system, and what are its primary functions?**

An operating system (OS) is software that manages a computer's hardware and provides a platform for applications to run. Its primary functions include:

1. Managing Resources: It allocates and controls CPU, memory, and storage resources.

2. File Management: It handles the creation, deletion, and access of files on storage devices.

3. Process Management: It manages the execution of programs, including starting, stopping, and multitasking.

4. User Interface: It provides a way for users to interact with the computer, typically through a graphical interface or command line.

5. Device Management: It controls and communicates with hardware devices like printers, keyboards, and displays.

In essence, the OS is the bridge between the user, the software, and the hardware.

**2. Explain the difference between process and thread.**

A process is an task/work/program which is being executing by CPU,

Whereas thread is A lightweight unit of a process that shares the same memory and resources.

**3. What is virtual memory, and how does it work?**

Virtual memory is a technique that extends the physical memory of a computer by using part of the hard drive as additional memory.

How it works:

1. Paging: Data is divided into blocks (pages). If physical RAM is full, less-used pages are moved to a file on the hard drive (swap space).
2. Swapping: When needed, pages are swapped between RAM and the hard drive, allowing more programs to run simultaneously than the physical RAM alone would support.

**4. Describe the difference between multiprogramming, multitasking, and multiprocessing**.

**Multiprogramming:**

Running multiple programs on a single CPU by switching between them, aiming to maximize CPU usage.

**Multitasking:**

Running multiple tasks or processes simultaneously on a single CPU using time-sharing.

**Multiprocessing:**

Using multiple CPUs or cores to execute multiple processes simultaneously.

**5. What is a file system, and what are its components?**

A file system is a method used by operating systems to manage and organize data on storage devices like hard drives, SSDs, and USB drives. It determines how data is stored, retrieved, and managed.

Components of a File System:

1. Files: Basic units of storage that contain data.
2. Directories/Folders: Structures that organize files and other directories.
3. File Metadata: Information about files, such as name, size, type, and permissions.
4. Inodes: Data structures that store metadata about files and directories.
5. Superblock: Contains information about the file system, like size, status, and layout.
6. File Allocation Table (FAT) or Index Nodes (inodes): Tracks the location of files on the disk.
7. Boot Block: Contains the boot loader, which is necessary to start the operating system.
8. Mount Point: Location in the directory structure where a file system is mounted.

**6. What is a deadlock, and how can it be prevented?**

Deadlock happens when two or more processes get stuck because each one is waiting for the other to release a resource, and no one can move forward.

Preventing Deadlock:

1. Don’t Share Resources Exclusively: Let processes share resources whenever possible.
2. Request All Resources at Once: Make processes ask for all the resources they need in one go.
3. Take Back Resources: If needed, take resources from one process and give them to another.
4. Avoid Circular Waiting: Make processes request resources in a specific order to avoid circular dependency.

**7. Explain the difference between a kernel and a shell.**

The kernel is the core part of an operating system. It directly interacts with the hardware and manages system resources like CPU, memory, and devices.

The shell is a user interface that allows you to interact with the operating system. It takes commands from you, passes them to the kernel, and then shows the results.

In short, the kernel does the heavy lifting behind the scenes, while the shell is what you use to communicate with the system.

**8. What is CPU scheduling, and why is it important?**

CPU scheduling is the process of deciding which tasks (or processes) the CPU should work on and in what order. It determines how the CPU's time is divided among different processes.

Importance:

1. Efficiency: Ensures the CPU is used effectively, minimizing idle time.
2. Responsiveness: Helps the system respond quickly to user requests and other tasks.
3. Fairness: Allocates CPU time to processes in a balanced way so no single process hogs resources.
4. Performance: Improves overall system performance by optimizing how tasks are executed.

**9. How does a system call work?**

A system call is a way for a program to request a service from the operating system. Here’s how it works:

1. Request: A program makes a system call to request an OS service (like reading a file or accessing hardware).
2. Switch to Kernel Mode: The system call triggers a switch from user mode to kernel mode, where the OS can execute privileged operations.
3. Execution: The OS processes the request and performs the required action.
4. Return: The result is sent back to the program, and control switches back to user mode.

**10. What is the purpose of device drivers in an operating system?**

Device drivers are essential software components that enable the operating system to communicate with and control hardware devices such as printers, graphics cards, or keyboards. They serve as a bridge between the operating system and the hardware by translating OS commands into actions that the hardware can understand. Device drivers manage the specific functions of each hardware device and provide a standardized interface for the OS to interact with different types of hardware. This abstraction and management ensure that various devices can operate smoothly with the operating system, maintaining compatibility and allowing the hardware and software to work together effectively.

**11. Explain the role of the page table in virtual memory management.**

The page table is a key component in managing virtual memory. It helps translate virtual addresses used by programs into physical addresses in the computer’s RAM. When a program needs to access data, the page table looks up where that data is stored in physical memory or on disk. This system allows programs to use more memory than is physically available, by efficiently managing how memory is allocated and used. Essentially, the page table helps the operating system keep track of where everything is and ensures that programs can access their data without running into memory issues.

**12. What is thrashing, and how can it be avoided?**

Thrashing occurs when the operating system spends too much time swapping data between RAM and disk, rather than executing processes. This happens when there’s not enough physical memory to handle the workload, causing excessive paging.

Avoiding Thrashing:

1. Increase RAM: Add more physical memory to reduce the need for swapping.
2. Optimize Processes: Ensure that processes are well-designed to use memory efficiently.

**13. Describe the concept of a semaphore and its use in synchronization.**

A semaphore is a synchronization tool used in computing to manage access to shared resources by multiple processes or threads. It helps prevent conflicts and ensures that processes do not interfere with each other.

By using semaphores, multiple processes can coordinate their access to shared resources, preventing conflicts and ensuring that each process gets a fair opportunity to use the resource.

**14. How does an operating system handle process synchronization?**

An operating system handles process synchronization to make sure that multiple processes or threads can work together without causing problems, especially when they need to access shared resources. Here’s how it does that:

1. Mutual Exclusion: Ensures that only one process or thread can use a resource at a time.
2. Locks: Uses tools to prevent more than one process from accessing a resource simultaneously.
3. Condition Variables: Allows processes to wait for certain conditions before continuing.
4. Semaphores: Uses counters to manage how many processes can access a resource at once.
5. Monitors: Combines mutual exclusion with condition waiting in one tool.

**15. What is the purpose of an interrupt in operating systems?**

An interrupt in an operating system is a signal that tells the CPU to stop what it's currently doing and handle a more urgent task.

This is important because it allows the system to quickly respond to important events, like input from a keyboard or messages from a network. By using interrupts, the operating system can efficiently manage multiple tasks, making sure that critical actions are addressed promptly without waiting for the current process to finish. This helps keep the system running smoothly and ensures that urgent tasks are handled quickly.

**16. Explain the concept of a file descriptor.**

A file descriptor is a unique identifier used by an operating system to manage and keep track of open files and other input/output resources. When a program opens a file or a device, the operating system assigns a file descriptor to it. This number is used to refer to the file or resource in subsequent operations, like reading from or writing to it.

In simple terms, think of a file descriptor as a handle or reference number that the system uses to keep track of files and resources so that programscan interact with them efficiently.

**17. How does a system recover from a system crash?**

* Identify the cause: Try to determine what caused the crash.
* Restart in safe mode: Restart your computer in safe mode, also known as recovery mode. This will boot the system with minimal drivers, which can help you identify the cause of the crash.
* Restore from a backup: You can restore your system from a backup.
* Repair system files: You can try repairing the system files.
* Reinstall the operating system: You can reinstall the operating system.
* Test and verify: After making any changes, test and verify the system

**18. Describe the difference between a monolithic kernel and a microkernel.**

a monolithic kernel integrates many services into a single large kernel for efficiency, while a microkernel keeps the kernel small and moves many services to user space for greater modularity and safety.

**19. What is the difference between internal and external fragmentation?**

* Fragmentation Occurs when allocated memory blocks are larger than the memory requested by a process, leaving unused space within the allocated block.

Example: If a process needs 30 KB of memory but the system allocates a 40 KB block, 10 KB of that block remains unused and wasted.

* External Fragmentation Occurs when free memory is available but is scattered in small, non-contiguous chunks, making it difficult to allocate a large block of memory.

Example: If there are several small free blocks of memory spread out, a process requiring a large, contiguous block might not be able to find enough space even though the total free memory is sufficient.

**20. How does an operating system manage I/O operations?**

An operating system manages I/O (Input/Output) operations by:

1. Device Drivers: Using special software to control and communicate with hardware devices like printers and disks.
2. Buffers: Storing data temporarily while it’s being moved between the computer and the devices, so everything runs smoothly.
3. Interrupts: Notifying the CPU when an I/O task is done or needs attention, so the system can manage tasks efficiently.
4. I/O Scheduling: Organizing the order in which I/O requests are handled to keep things running smoothly.
5. System Calls: Allowing programs to request I/O operations, like reading from a file or sending data to a printer.

**21. Explain the difference between preemptive and non-preemptive scheduling.**

PreEmptive : In preemptive scheduling, the operating system can interrupt and suspend a currently running process in order to start or resume another process. This is typically based on priority or other scheduling criteria.

Non-PreEmptive: In non-preemptive scheduling, once a process starts its execution, it cannot be interrupted or suspended until it finishes or voluntarily yields control of the CPU (e.g., it enters a waiting state).

**22. What is round-robin scheduling, and how does it work?**

Round-Robin Scheduling is a preemptive CPU scheduling algorithm commonly used in time-sharing systems. Here's how it works:

How It Works:

1. Time Quantum: A fixed time slice, called a time quantum or time slice, is assigned to each process in the ready queue.
2. Process Execution: The CPU executes each process for a duration equal to the time quantum. If a process finishes during its time quantum, it releases the CPU.
3. Preemption: If a process does not finish within its time quantum, it is preempted (paused), and the CPU is assigned to the next process in the queue. The preempted process is then placed at the back of the ready queue to wait for its next turn.
4. Cycle Repeats: This cycle continues, giving each process a fair share of CPU time.

**23. Describe the priority scheduling algorithm. How is priority assigned to processes?**

Priority Scheduling is a CPU scheduling algorithm where each process is assigned a priority, and the CPU is allocated to the process with the highest priority. This can be either preemptive or non-preemptive.

How Priority Scheduling Works:

* Priority Assignment: Each process is given a priority value, typically a number. The process with the highest priority (lowest number, if lower numbers indicate higher priority) is selected for execution first.
* Execution:
  + In preemptive priority scheduling, if a new process arrives with a higher priority than the currently running process, the CPU is preempted and allocated to the new process.
  + In non-preemptive priority scheduling, the currently running process will continue until it finishes, and then the CPU will be allocated to the highest-priority process in the ready queue.
* Tie-Breaking: If two processes have the same priority, tie-breaking can be done using another scheduling algorithm like First-Come, First-Served (FCFS).

**24. What is the shortest job next (SJN) scheduling algorithm, and when is it used?**

Shortest Job Next (SJN), also known as Shortest Job First (SJF), is a non-preemptive CPU scheduling algorithm. In SJN, the process with the shortest burst time (the time required to complete the process) is selected for execution next.

How SJN Works:

* Process Selection: Among the processes in the ready queue, the one with the shortest burst time is selected for execution.
* Execution: Once a process starts executing, it runs to completion without being interrupted.
* Tie-Breaking: If two processes have the same burst time, another criterion like FCFS (First-Come, First-Served) can be used to decide which process to run first.

**25. Explain the concept of multilevel queue scheduling.**

Multilevel Queue Scheduling is a CPU scheduling algorithm that partitions the ready queue into multiple separate queues, each with its own scheduling algorithm. Processes are assigned to a queue based on certain characteristics, such as priority, process type, or resource requirements, and each queue can have its own scheduling policy.

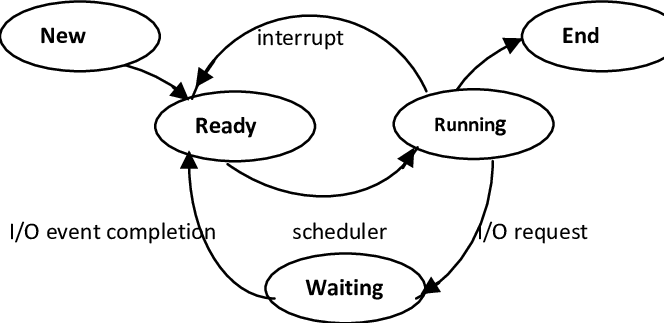
**26. What is a process control block (PCB), and what information does it contain?**

A Process Control Block (PCB) is a data structure used by the operating system to store all the information about a process. The PCB is crucial for process management, as it contains the necessary details for the OS to keep track of process execution and manage multiple processes.

Information Contained in a PCB:

1. Process ID (PID)
2. Process State:
3. Program Counter:
4. CPU Registers:
5. Memory Management Information:
6. Scheduling Information:

**27. Describe the process state diagram and the transitions between different process states.**

****

**New:**

* The process is being created.

**Ready**:

* The process is loaded into memory and is waiting for CPU time.

**Running**:

* The process is currently being executed by the CPU.

**Waiting (or Blocked):**

* The process is waiting for some event to occur, such as I/O completion or a resource becoming available.

**Terminated (or Exit):**

* The process has finished execution or has been terminated by thesystem**.**

**28. How does a process communicate with another process in an operating system?**

process communicate with another process Through System calls in an operating system

**29. What is process synchronization, and why is it important? \**

Process synchronization is about making sure that multiple processes or threads work together without causing problems, especially when they use the same resources. It prevents issues like:

* Data Races: Where multiple processes mess up shared data because they access it at the same time.
* Deadlocks: Where processes get stuck waiting for each other and nothing progresses.

By using tools like locks or semaphores, synchronization ensures that only one process or thread can use a resource at a time, keeping everything running smoothly and correctly.

Importance

* **Prevents Errors**: Stops multiple processes from messing up shared data, ensuring the system works correctly.
* **Avoids Deadlocks**: Prevents processes from getting stuck in endless waiting loops, keeping the system functional.
* **Ensures Consistency**: Makes sure that shared data remains accurate and reliable, avoiding conflicts and corruption.
* **Improves Efficiency**: Coordinates processes to run smoothly without interfering with each other, leading to better system performance.

**30. Explain the concept of a zombie process and how it is created.**

A zombie process is a process that has finished running but still appears in the system's process table. This happens because its parent process hasn't yet read its exit status.

Creation: When a process ends, it becomes a zombie if the parent doesn't collect its exit status.

**31. Describe the difference between internal fragmentation and external fragmentation.**

Refer Q19

**32. What is demand paging, and how does it improve memory management efficiency?**

Demand paging is a memory management technique where pages of a process are loaded into physical memory only when they are needed, rather than loading the entire process at once.

**How It Works:**

1. When a process starts, only a few of its pages are loaded into memory, while others remain on the disk.
2. When a process tries to access a page that is not currently in memory, a page fault occurs. The operating system then loads the required page from disk into memory.
3. If memory is full, the OS may use a page replacement algorithm to swap out a page from memory to make space for the new page.

**33. Explain the role of the page table in virtual memory management.**

The page table is table in virtual memory to map virtual addresses to physical addresses. It plays a key role in translating virtual addresses, used by processes, into physical addresses, used by the hardware. Each entry in the page table holds information about a specific virtual page, including its corresponding physical page frame

Role:

* The page table keeps track of where virtual pages are located in physical memory.
* It maintains mappings for each virtual page to a physical page frame in memory. This helps in managing and accessing the memory efficiently.

**34. How does a memory management unit (MMU) work?**

A memory management unit (MMU) is a hardware component that translates virtual addresses to physical addresses in a computer's memory system. It does this to ensure that the processor and virtual memory interact smoothly. The MMU also performs other tasks, such as:

* Memory distribution: Moving memory resources to where they are needed most
* Monitoring: Tracking all memory resources
* Efficiency: Conserving memory resources
* System integrity: Reducing the risk of memory fragmentation and poor allocation
* Data integrity: Maintaining data integrity
* Data risk: Minimizing the risk of data corruption

**35. What is thrashing, and how can it be avoided in virtual memory systems?**

Thrashing is a condition in virtual memory systems where excessive paging occurs, causing the system to spend most of its time swapping pages in and out of physical memory rather than executing processes. This results in poor performance and can make the system appear unresponsive.

* Use Better Algorithms: Implement page replacement algorithms like Least Recently Used (LRU) or Optimal that better manage which pages to keep in memory.
* Limit Active Processes: Reduce the number of processes running simultaneously to ensure that each process gets adequate memory.
* Upgrade RAM: Adding more physical memory can help accommodate the working sets of processes, reducing the need for frequent paging.
* Optimize Code: Design applications to improve their locality of reference so that they access memory in a more predictable and efficient manner.

**36. What is a system call, and how does it facilitate communication between user programs and the operating system?**

A system call is a mechanism that allows user programs to request services from the operating system. It serves as an interface between user-level applications and the kernel, enabling programs to perform operations that are not directly accessible in user mode.

**How It Works:**

1. **Request**: The program makes a system call to request an action.
2. **Kernel Access**: The operating system takes over and performs the action because it has the necessary permissions.
3. **Result**: Once the action is done, the result is sent back to the program.

**37. Describe the difference between a monolithic kernel and a microkernel.**

Refer Q18

**38. How does an operating system handle I/O operations?**

The operating system handles I/O operations by:

1. Using Device Drivers: Special software that controls hardware devices.
2. Processing System Calls: Programs request I/O actions (like reading files) through system calls.
3. Buffering: Temporary storage to manage data while it's being transferred.
4. Scheduling: Organizing I/O tasks to improve efficiency and fairness.
5. Handling Interrupts: Responding to signals from devices to manage I/O events.
6. Direct Memory Access (DMA): Allowing devices to transfer data directly to memory, speeding up operations.

**39. Explain the concept of a race condition and how it can be prevented.**

A race condition happens when two or more processes try to use the same resource at the same time, causing unexpected results. To prevent it:

1. Locks: Make sure only one process can use the resource at a time.
2. Semaphores: Limit how many processes can use the resource at once.
3. Atomic Operations: Ensure operations are completed fully without interference.
4. Synchronization: Manage the order in which things access the resource.

**40. Describe the role of device drivers in an operating system.**

Device drivers are special programs in an operating system that allow the OS to communicate with hardware devices like printers, keyboards, or graphics cards. They act as a translator between the OS and the hardware, ensuring that commands and data are properly understood and handled by the device. Without drivers, the OS wouldn't be able to control or use the hardware effectively.

**41. What is a zombie process, and how does it occur? How can a zombie process be prevented?**

Refer Q30

**42. Explain the concept of an orphan process. How does an operating system handle orphan process?**

An orphan process is a process whose parent has terminated before it has finished. As a result, the orphan process is left running without a parent.

Handling Orphan Processes:

1. Adoption by Init: The operating system’s init process (or its equivalent) adopts orphan processes. This ensures that the orphan process still has a parent process to handle its termination and resource cleanup.
2. Reaping: The adopted orphan process will be properly terminated by init, which also reclaims its resources.

**43. What is the relationship between a parent process and a child process in the context of process management?**

In process management, a parent process is the process that creates one or more child processes. The relationship between them includes:

* The parent process uses system calls like fork() (in Unix-based systems) to create child processes.
* Child processes inherit certain attributes from the parent, such as environment variables and file descriptors.
* The parent process can control and monitor child processes, including waiting for them to complete or terminating them.
* When a parent process terminates, its child processes are typically adopted by the init process (or an equivalent), which ensures proper cleanup and resource management.

**44. How does the fork() system call work in creating a new process in Unix-like operating systems?**

The fork() system call in Unix-like operating systems creates a new process by duplicating the calling process. Here's how it works:

1. Creation: When a process calls fork(), the operating system creates a new process (the child) that is a copy of the calling process (the parent).
2. Process IDs: The child process receives a unique process ID (PID). In the parent process, fork() returns the PID of the child. In the child process, fork() returns 0.
3. Memory Space: Both processes share the same memory space initially, but they can have different execution paths. Changes made to memory in one process do not affect the other.

**45. Describe how a parent process can wait for a child process to finish execution.**

A parent process can wait for a child process to finish execution by using the wait() system call. When the parent process calls wait(), it pauses its own execution until one of its child processes terminates. Once the child process completes, the wait() call returns, providing the parent process with information about the child’s termination status. This includes whether the child exited normally or was terminated by a signal. This mechanism allows the parent to handle the child’s exit properly and clean up any associated resources. By waiting for child processes to finish, the parent process ensures orderly execution and resource management.

**46. What is the significance of the exit status of a child process in the wait() system call?**

The exit status of a child process is significant in the wait() system call because it provides crucial information about how the child process terminated. When a child process ends, it returns an exit status code that indicates whether it completed successfully or if there were errors.

**47. How can a parent process terminate a child process in Unix-like operating systems?**

A parent process can terminate a child process in Unix-like operating systems using the kill() system call.

The parent process needs the process ID (PID) of the child process it wants to terminate. This PID is typically obtained when the child process is created.

**48. Explain the difference between a process group and a session in Unix-like operating systems.**

In Unix-like operating systems, a process group and a session are ways to organize processes:

1. Process Group:
   * A process group is a set of related processes grouped together. It allows you to manage multiple processes as a single unit, such as sending signals (like termination requests) to all processes in the group at once. For example, when you run a command in a shell that spawns several processes, they can be grouped so that they can be controlled together.
2. Session:
   * A session is a larger grouping that includes one or more process groups. It starts with a session leader, which is typically the first process in the session. Sessions help manage related process groups, like those associated with a user’s login session or a terminal. They make it easier to control and manage processes that are linked to a specific user or terminal session.

**49. Describe how the exec() family of functions is used to replace the current process image with a new one.**

The exec() family of functions replaces the current process with a new program. When a process calls exec(), it loads a new program into its memory and starts running it. The original program is completely replaced, and the process ID remains the same. If exec() is successful, the original program does not continue; if it fails, the original program continues running.

**50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?**

The waitpid() system call allows a parent process to wait for a specific child process to finish and get its status. Unlike wait(), which waits for any child process. waitpid() can target a specific child and offers additional options for handling different scenarios.

**51. How does process termination occur in Unix-like operating systems?**

In Unix-like operating systems, when a process finishes its execution or is terminated, it undergoes a series of steps. First, the process ends either by completing its tasks, calling the exit() function, or receiving a termination signal. Next, the system cleans up by freeing the resources allocated to the process, such as memory and file handles. The process then briefly enters a "zombie" state while waiting for the parent process to collect its exit status. Once the parent process retrieves this status using wait() or waitpid(), the system completely removes the process and its entry from the process table. This orderly process ensures that resources are properly managed and the system remains efficient.

**52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?**

The long-term scheduler decides which processes from the job pool get loaded into memory and made ready for execution. It controls the degree of multiprogramming by managing how many processes are in memory at once. If it lets in more processes, it increases system utilization but can cause resource competition. If it lets in fewer, it reduces system load but might underuse resources.

**53. How does the short-term scheduler differ from the long-term and medium-term schedulers in terms of frequency of execution and the scope of its decisions?**

The short-term scheduler operates very frequently, making decisions every few milliseconds about which process from the ready queue will use the CPU next.

In contrast, the long-term scheduler runs less often and focuses on managing which processes should be loaded into memory from the job pool.

The medium-term scheduler works between these two, running as needed to manage memory. It decides which processes should remain in memory and which should be swapped out to disk storage.

**54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.**

A scenario where the medium-term scheduler would be invoked is when the system is running out of memory due to a high number of active processes. Here’s how it works and how it helps manage resources:

Scenario:

Imagine a server running multiple applications, each with several active processes. As more applications are started or existing processes become more memory-intensive, the system may start to run low on available RAM. This can cause performance issues or prevent new processes from being loaded into memory.