Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

echo "Hello, World!"

* It prints the text Hello, World!.

 name="Productive"

* Name is variable of value productive.

 touch file.txt

* Create new blank file with naming file.txt

 ls -a

* Showing all the files and directories with hidden files.

 rm file.txt

* Delete file name file.txt

 cp file1.txt file2.txt

* Copy the file1.txt in file2.txt

 mv file.txt /path/to/directory/

* Move file file.txt in the given directory.

 chmod 755 script.sh

* It shows owner has read , write and execute permission and group and other users has permission read and execute the script.sh file.

 grep "pattern" file.txt

* It shows the lines where this pattern is found

 kill PID

* Stop the process with specified PID number.

 mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt

* Create directory of name mydir, move to that directory and create touch.txt blank file and write Hello, World! and display it.

 ls -l | grep ".txt"

* List all the files which are .txt with the detailed information like permission, owner, size, date

 cat file1.txt file2.txt | sort | uniq

* Combine contents of two files, sort them, and remove duplicates.

 ls -l | grep "^d"

* List all the directories with the detailed information like permission, owner, size, date

 grep -r "pattern" /path/to/directory/

* Recursively searching for "pattern" in all files under the given path.

 cat file1.txt file2.txt | sort | uniq –d

* Combine contents of two files, sort them, and show only duplicates.

 chmod 644 file.txt

* Owner has read and write permissions, while group and others have only read permission on file.txt

 cp -r source\_directory destination\_directory

* Recusively copy the directory and all its contents.

 find /path/to/search -name "\*.txt"

* Find all .txt file in the given path.

 chmod u+x file.txt

* Changing the permissions of user/owner and give the permission of executing file.txt.

 echo $PATH

* Shows the folders where your computer looks for commands/programs to run.

Part B

Identify True or False:

1. ls is used to list files and directories in a directory.

=> True

2. mv is used to move files and directories.

=> True

3. cd is used to copy files and directories.

=> False

4. pwd stands for "print working directory" and displays the current directory.

=> True

5. grep is used to search for patterns in files.

=> True

6. chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute permissions to group and others.

=> True

7. mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1 if directory1 does not exist.

=> True

8. rm -rf file.txt deletes a file forcefully without confirmation.

=> True.

Identify the Incorrect Commands:

1. chmodx is used to change file permissions.

=> chmod

2. cpy is used to copy files and directories

=> cp

3. mkfile is used to create a new file.

=> touch

4. catx is used to concatenate files.

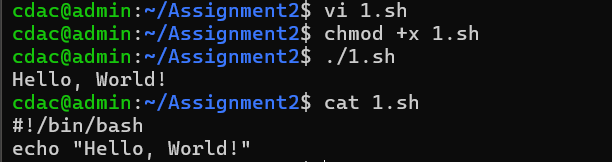
=> cat

5. rn is used to rename files.

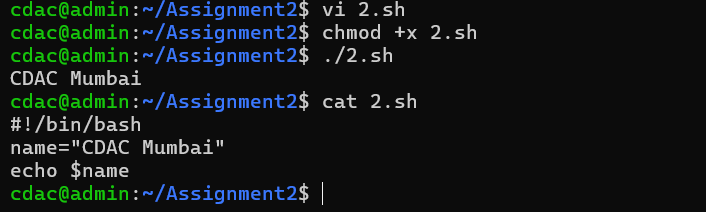
=> mv

Part C

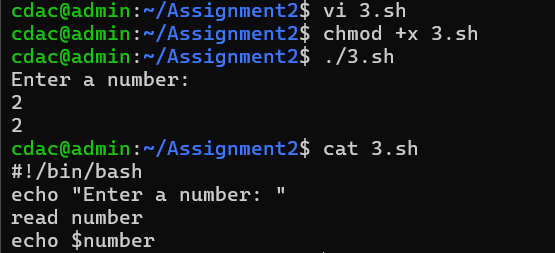
Question 1: Write a shell script that prints "Hello, World!" to the terminal.



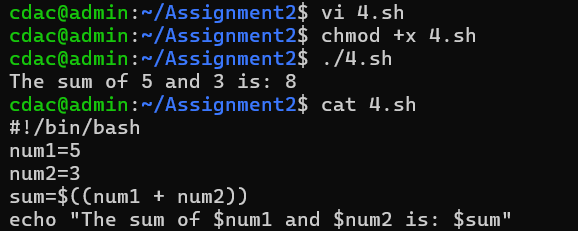
Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.



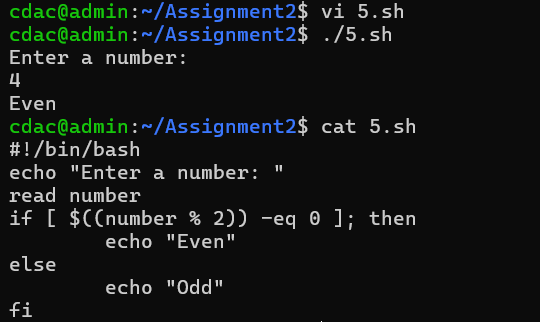
Question 3: Write a shell script that takes a number as input from the user and prints it.



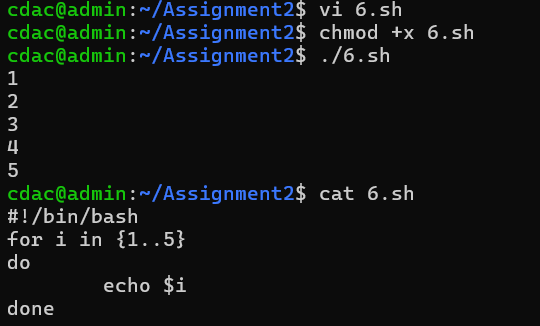
Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.



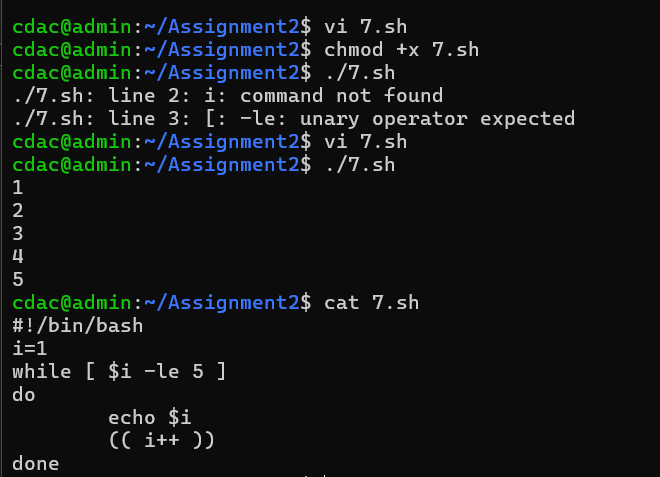
Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".



Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.

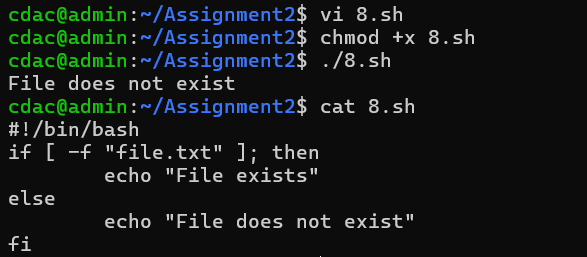


Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.

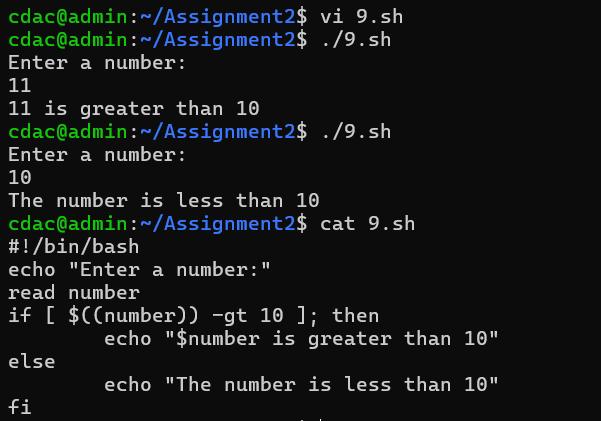


//assign the variables without spacing. E.g i=1 # no spaces

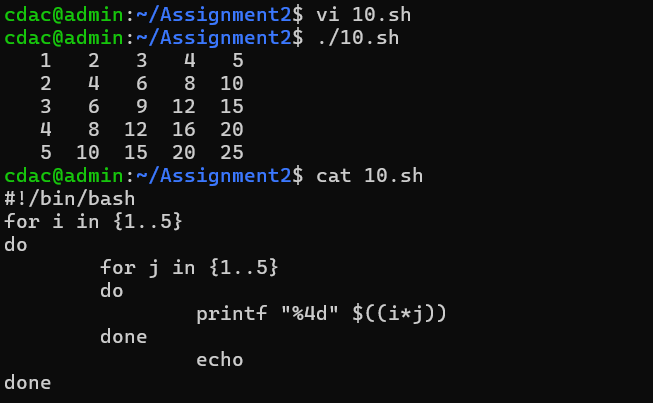
Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".



Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.



Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.



Part E

1. Consider the following processes with arrival times and burst times:

| Process | Arrival Time | Burst Time |

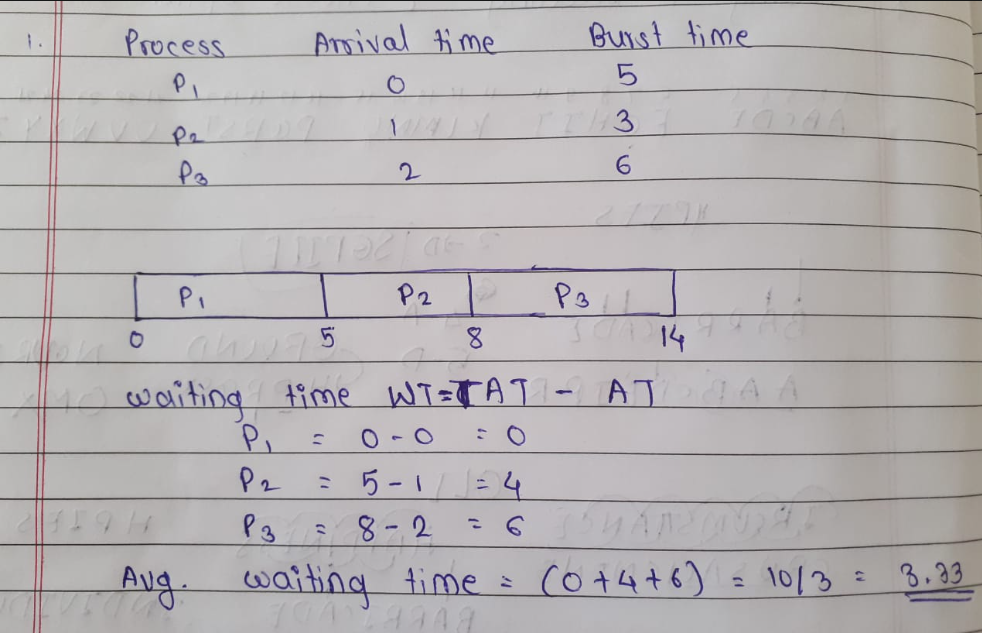
|---------|--------------|------------|

| P1 | 0 | 5 |

| P2 | 1 | 3 |

| P3 | 2 | 6 |

Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.



2. Consider the following processes with arrival times and burst times:

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

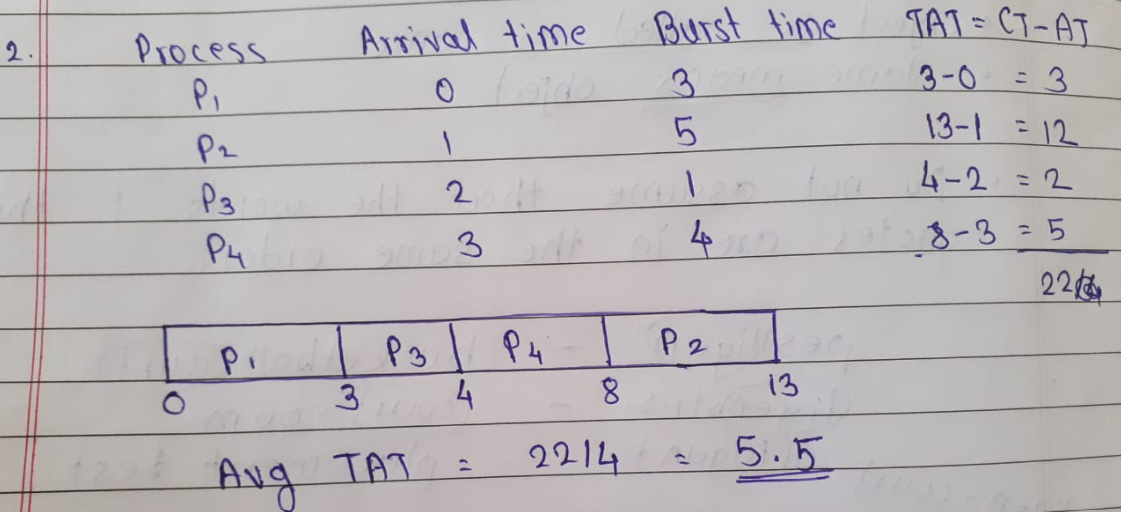
| P1 | 0 | 3 |

| P2 | 1 | 5 |

| P3 | 2 | 1 |

| P4 | 3 | 4 |

Calculate the average turnaround time using Shortest Job First (SJF) scheduling.



3. Consider the following processes with arrival times, burst times, and priorities (lower number

indicates higher priority):

| Process | Arrival Time | Burst Time | Priority |

|---------|--------------|------------|----------|

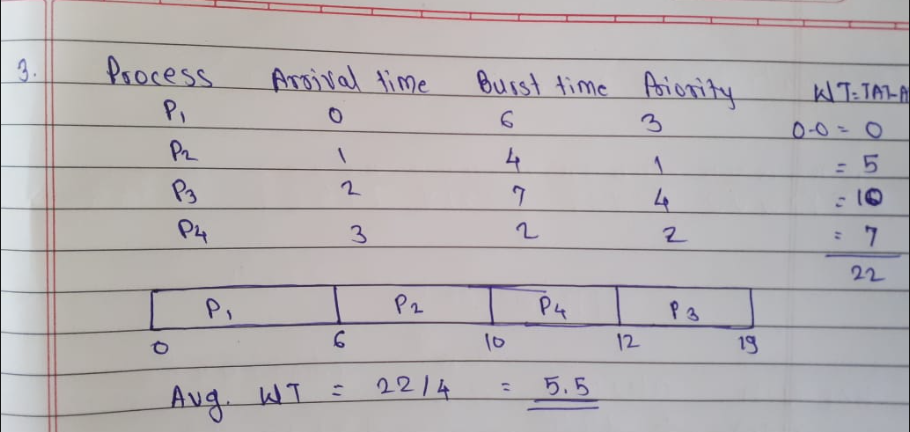
| P1 | 0 | 6 | 3 |

| P2 | 1 | 4 | 1 |

| P3 | 2 | 7 | 4 |

| P4 | 3 | 2 | 2 |

Calculate the average waiting time using Priority Scheduling.



4. Consider the following processes with arrival times and burst times, and the time quantum for

Round Robin scheduling is 2 units:

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

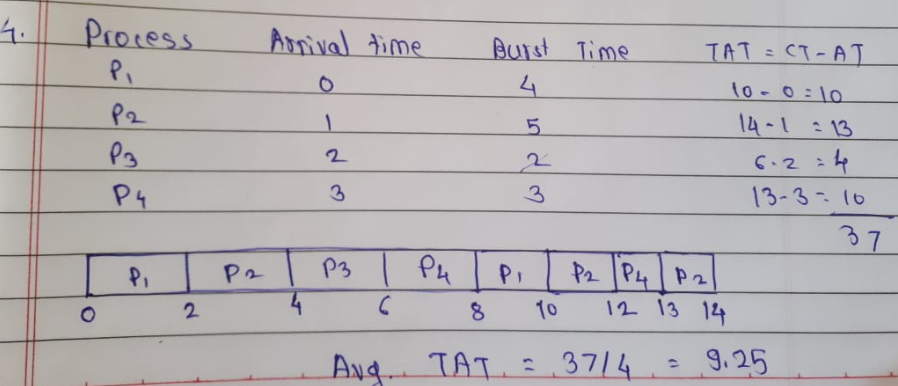
| P1 | 0 | 4 |

| P2 | 1 | 5 |

| P3 | 2 | 2 |

| P4 | 3 | 3 |

Calculate the average turnaround time using Round Robin scheduling.



5. Consider a program that uses the fork() system call to create a child process. Initially, the parent process has a variable x with a value of 5. After forking, both the parent and child processes increment the value of x by 1.

What will be the final values of x in the parent and child processes after the fork() call?

* When a program uses the fork() system call, it creates a child process which is a copy of the parent process. This means that the child also gets its own copy of the parent’s variables. In this case, the parent has a variable x = 5. After the fork, both the parent and the child have x = 5 in their own memory. When both processes increment x by 1, the parent’s value of x becomes 6 and the child’s value of x also becomes 6. These values are stored separately, so changing x in one process does not affect the other.

Part D

Common Interview Questions (Must know)

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

=> An operating system (OS) is system software that acts as an intermediary between users, applications, and computer hardware. It provides an environment where applications can run and interact with hardware efficiently and safely. The main functions of an OS include: **process management** (scheduling, creation, termination), **memory management** (allocation, protection, virtual memory), **file system management** (organizing and accessing data), **device management** (using device drivers for I/O), and **security & protection** (user authentication, resource protection). Examples: Windows, Linux, macOS.

2. Explain the difference between process and thread.

=>**Process**

* A process is an independent program in execution.
* Each process has its own **memory space** (code, data, heap, stack).
* Processes do not share memory by default; communication between processes requires **Inter-Process Communication (IPC)** like pipes, message queues, or shared memory.
* Creating a process is slower because the OS must allocate separate resources.
* If one process crashes, it usually does **not affect others**.

**Thread**

* A thread is a **lightweight subunit** of a process.
* Multiple threads can exist within a single process.
* Threads **share the same memory space** (code, data, heap) but have their own **stack and registers**.
* Communication between threads is faster, as they share memory directly.
* Creating a thread is faster and requires fewer resources.
* If one thread crashes, it may affect the whole process (and other threads in it).

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

=> Virtual memory is a memory management technique that gives an application the illusion of having large, continuous memory, even if the physical RAM is smaller. It uses both main memory (RAM) and secondary storage (disk) to provide this abstraction. The OS uses **paging or segmentation** to map virtual addresses to physical addresses, handled by the **Memory Management Unit (MMU)**. When a program accesses a page not in memory, a **page fault** occurs, and the required page is loaded from disk. This allows large programs to run efficiently, enables multiprogramming, and ensures memory protection between processes. Example: Running multiple applications on 8 GB RAM while total memory demand exceeds it.

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

* **Multiprogramming:** Technique where multiple programs are loaded into memory, and the CPU switches between them to keep itself busy. Only one program runs at a time, but switching increases efficiency.
* **Multitasking:** An extension of multiprogramming where the CPU gives each process a time slice, creating the illusion of simultaneous execution. Useful in interactive systems.
* **Multiprocessing:** Involves multiple CPUs (or cores) working together to execute multiple processes simultaneously. Improves true parallelism.  
  **Example:** A single-core system with multiprogramming may run Word and Music Player alternately, while a multicore CPU can run them simultaneously.

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

=>A file system is a method used by the operating system to store, organize, and manage files on storage devices. It provides a way to create, delete, read, write, and organize files. Major components are:

1. **Files** – basic unit of storage containing data.
2. **Directories** – structures that organize files hierarchically.
3. **Metadata** – information about files (size, permissions, timestamps).
4. **File allocation methods** – define how disk space is allocated (contiguous, linked, indexed).
5. **Free space management** – tracks available space on disk.  
   **Examples of file systems:** FAT32, NTFS, ext4.

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

=> A deadlock is a situation in which a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by another process. For example, Process A holds a printer and waits for a scanner, while Process B holds the scanner and waits for the printer.  
**Conditions for deadlock (Coffman):** Mutual exclusion, Hold and wait, No preemption, Circular wait.  
**Prevention strategies:**

* Avoid mutual exclusion (make resources sharable).
* Prevent hold & wait (request all resources at once).
* Allow preemption (forcefully take resources).
* Avoid circular wait (use resource ordering).

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

=> The **kernel** is the core component of the operating system. It manages hardware resources, memory, CPU scheduling, device drivers, and system calls. It operates in privileged mode (kernel mode).  
The **shell** is the interface between the user and the kernel. It interprets user commands and communicates with the kernel to execute them. The shell can be **command-line (CLI)** like Bash or **graphical (GUI)** like Windows Explorer.

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

CPU scheduling is the process of selecting which ready process in the queue should be assigned to the CPU next. It is important because:

* It maximizes CPU utilization.
* Provides fairness to processes.
* Minimizes waiting time, turnaround time, and response time.
* Ensures efficient time-sharing in multitasking systems.  
  **Algorithms used:** FCFS, SJF, Priority, Round Robin, Multilevel Queue, etc.

9. How does a system call work?

=> A system call is a request made by a user program to the operating system for performing tasks that require privileged access (like I/O operations).  
**Steps:**

1. User program executes a library function (e.g., read()).
2. A **trap instruction** switches control to kernel mode.
3. The kernel executes the requested service.
4. The result is returned to the user program.  
   **Example:** Opening a file requires a system call to interact with the file system.

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

Device drivers are special programs that allow the operating system to communicate with hardware devices. Since hardware manufacturers design devices differently, the OS cannot directly handle them all.

* Drivers act as a **translator** between OS and hardware.
* They provide standard interfaces so applications can use devices without worrying about hardware details.  
  **Example:** Printer drivers, graphics card drivers, USB drivers.

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

The page table is a data structure used by the OS to map **virtual addresses** (used by programs) to **physical addresses** (in RAM).

* Each process has its own page table.
* It stores frame numbers corresponding to page numbers.
* The MMU (Memory Management Unit) uses the page table to translate addresses quickly.
* Special techniques like TLB (Translation Lookaside Buffer) speed up lookup.  
  **Role:** Enables virtual memory, memory protection, and process isolation.

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

Thrashing occurs when a system spends more time **swapping pages between RAM and disk** than executing processes. This happens when there is too little RAM and too many processes.  
**Effects:** CPU utilization drops drastically.  
**Avoidance:**

* Use working set model (allocate enough frames to active processes).
* Reduce degree of multiprogramming.
* Add more physical memory.
* Use better page replacement policies.

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

A semaphore is a synchronization tool used to control access to shared resources in a concurrent system.

* A semaphore is essentially a counter variable.
* **Types:**
  + Binary semaphore (mutex) – value 0 or 1.
  + Counting semaphore – value can range above 1 (multiple resources).
* **Operations:** wait(P) (decrement), signal(V) (increment).
* Used to solve problems like producer-consumer, reader-writer.  
  **Example:** Two processes trying to print to the same printer use a semaphore to avoid conflict.

14. How does an operating system handle process synchronization?

Process synchronization ensures that multiple processes can safely access shared resources.  
**Techniques used:**

* Semaphores
* Mutex locks
* Monitors
* Condition variables
* Spinlocks  
  The OS uses these primitives to avoid problems like **race conditions**, **deadlocks**, and **inconsistent data states**.

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

An interrupt is a signal sent to the CPU by hardware or software indicating an event that needs immediate attention.

* **Hardware interrupts:** Generated by I/O devices (e.g., keyboard input, timer).
* **Software interrupts (traps):** Generated by programs (e.g., system calls, errors).  
  **Purpose:**
* Makes CPU respond to events quickly.
* Improves efficiency (CPU doesn’t need to constantly poll devices).
* Enables multitasking and preemptive scheduling.

16. Explain the concept of a file descriptor.

A file descriptor is a small integer value used by the operating system to identify an open file or I/O resource.

* When a process opens a file, the OS returns a file descriptor.
* It is used in system calls like read(), write(), close().
* In UNIX/Linux, standard descriptors are:
  + 0 → Standard Input (stdin)
  + 1 → Standard Output (stdout)
  + 2 → Standard Error (stderr)

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

System recovery after a crash involves several techniques:

* **Checkpointing:** The OS periodically saves system states (memory, process table). After crash, system restarts from last checkpoint.
* **File system recovery:** Journaling file systems (like ext4, NTFS) log operations, so after a crash, incomplete operations are rolled back.
* **Process recovery:** Processes that were running may be restarted, or incomplete transactions may be rolled back.
* **Hardware support:** Watchdog timers reset the system if it becomes unresponsive.  
  **Goal:** Minimize data loss and restore system to consistent state.

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

**Monolithic Kernel:**

* All OS services (device drivers, file systems, memory, CPU scheduling) run in kernel mode.
* Faster performance because services directly interact.
* Harder to maintain (a bug in one service can crash entire system).
* Example: Linux, UNIX.

**Microkernel:**

* Only essential services (like IPC, memory management, scheduling) run in kernel mode.
* Other services (drivers, file systems) run in user space.
* More secure and modular, easier to extend.
* Slightly slower due to more context switching.
* Example: Minix, QNX.

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

**Internal fragmentation:** Occurs when allocated memory block is larger than required, leaving unused space **inside** the block.  
Example: Allocating 8 KB for a process needing 6 KB leaves 2 KB wasted.

**External fragmentation:** Occurs when free memory is split into small scattered holes, preventing allocation of large continuous blocks.  
Example: Free spaces of 10 KB, 5 KB, 8 KB exist, but a process requires 15 KB.

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

I/O management ensures smooth communication between CPU and I/O devices. The OS handles:

* **Buffering:** Temporary storage of data before transfer.
* **Caching:** Storing frequently used data in faster storage.
* **Spooling:** Overlapping of I/O operations with CPU execution (e.g., print queue).
* **Device drivers:** Allow OS to interact with hardware.
* **Interrupts & DMA (Direct Memory Access):** Improve efficiency by reducing CPU involvement.

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

**Preemptive Scheduling:**

* CPU can be taken away from a process before it finishes.
* Provides better response time for interactive processes.
* Examples: Round Robin, Preemptive Priority, SRTF.

**Non-preemptive Scheduling:**

* Once a process gets CPU, it cannot be interrupted until completion or waiting.
* Simpler but less responsive.
* Examples: FCFS, Non-preemptive Priority, SJF.

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

Round-robin (RR) is a **preemptive scheduling algorithm** used in time-sharing systems.

* Each process is assigned a fixed **time quantum** (say 10 ms).
* CPU cycles through processes in the ready queue.
* If a process does not finish within its time slice, it is moved to the back of the queue.  
  **Advantages:** Fair to all processes, good for interactive systems.  
  **Disadvantage:** Performance depends heavily on time quantum (too small → overhead, too large → behaves like FCFS).

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

In **priority scheduling**, each process is assigned a priority number.

* The CPU is allocated to the process with the **highest priority**.
* Priorities may be assigned based on:
  + Process type (system vs. user process).
  + Memory/CPU needs.
  + Importance of task.
* Can be **preemptive** (higher-priority process interrupts) or **non-preemptive**.  
  **Problem:** Starvation (low-priority processes may never run).  
  **Solution:** Aging (increase priority of waiting processes).

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

Shortest Job Next (SJN), also called Shortest Job First (SJF), selects the process with the **smallest execution time**.

* **Non-preemptive:** Once process starts, it cannot be stopped.
* **Preemptive version:** Shortest Remaining Time First (SRTF).
* Optimal for minimizing average waiting time.
* Problem: Requires knowledge of execution time in advance (practical in batch systems, not real-time).

25. Explain the concept of multilevel queue scheduling.

In **multilevel queue scheduling**, processes are divided into different queues based on priority or type:

* Example queues:
  + **Foreground (interactive)** → uses Round Robin.
  + **Background (batch)** → uses FCFS.
* Each queue has its own scheduling algorithm.
* CPU time is divided among queues using **fixed priority** or **time-slicing**.

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

The **Process Control Block (PCB)** is a data structure maintained by the operating system to store information about a process. It acts like the "identity card" of a process.

**Contents of PCB:**

* **Process ID (PID):** Unique identifier.
* **Process State:** Ready, Running, Waiting, etc.
* **CPU Registers:** Stored when a process is switched out.
* **Program Counter:** Address of next instruction to execute.
* **Memory Management Info:** Page tables, base/limit registers.
* **Accounting Info:** CPU time used, job number.
* **I/O Status:** List of files and devices allocated.

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

A process goes through different states in its lifecycle:

**States:**

1. **New:** Process created but not ready.
2. **Ready:** Waiting to be assigned CPU.
3. **Running:** Currently executing on CPU.
4. **Waiting/Blocked:** Waiting for I/O or event.
5. **Terminated:** Finished execution.

**Transitions:**

* New → Ready (admitted by OS).
* Ready → Running (scheduler selects process).
* Running → Waiting (if waiting for I/O).
* Running → Ready (preemption by scheduler).
* Running → Terminated (process completes).

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

Inter-Process Communication (IPC) methods:

* **Shared Memory:** Processes share a region of memory for fast data exchange.
* **Message Passing:** Processes send messages via system calls (send/receive).
* **Pipes:** One-way communication channels between processes.
* **Sockets:** Communication between processes on different systems (network).
* **Signals:** Used to notify a process of an event (e.g., kill -9).

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

**Process synchronization** ensures that multiple processes accessing shared resources do so in a controlled manner.

**Why important?**

* Prevents **race conditions** (when outcome depends on execution order).
* Maintains **data consistency** in shared memory.
* Avoids deadlocks and ensures fairness.

**Techniques:** Semaphores, Monitors, Mutex locks, Condition variables.

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

A **zombie process** is a process that has completed execution but still has an entry in the process table.

* Created when a child process finishes, but the parent has not yet read its exit status using wait().
* The process is dead, but its PCB remains.
* Too many zombies can exhaust the process table.

**Solution:** Parent must use wait() or waitpid() to collect child’s exit status.

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

**Internal fragmentation:** Wasted space **inside** allocated blocks (due to block being bigger than required).

**External fragmentation:** Wasted space **outside**, as free memory is split into small holes.

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

**Demand paging** is a memory management technique where pages are loaded into memory **only when required**.

* Initially, no page is loaded.
* If a process accesses a page not in memory, a **page fault** occurs → OS loads it from disk.
* Reduces memory usage since only needed pages are loaded.
* Improves multiprogramming (more processes fit in memory).

**Drawback:** High page faults if locality of reference is low.

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

The **page table** maps **virtual addresses** (used by processes) to **physical addresses** (in RAM).

* Each process has its own page table.
* Contains entries like:
  + Frame number (location in RAM).
  + Valid/Invalid bit (whether page is in memory).
  + Protection bits (read/write permissions).
* Managed by **Memory Management Unit (MMU)**.

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

The **MMU** is a hardware component that translates virtual addresses to physical addresses.

**Working:**

* CPU generates a virtual address.
* MMU uses the process’s page table to find the corresponding physical address.
* If page not in memory → Page fault → OS loads page from disk.

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

**Thrashing** occurs when the CPU spends more time handling page faults than executing processes.

* Happens when the working set of a process is larger than available memory.
* Too much paging in/out reduces performance drastically.

**Avoidance techniques:**

* Use **working set model** (allocate enough frames for active pages).
* Use **page replacement algorithms** wisely (LRU, Clock).
* Increase physical memory.
* Use **load control** (reduce number of active processes).

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

A **system call** is the interface between a user program and the operating system. Since user programs cannot directly access hardware, they request services via system calls.

**Examples of system calls:**

* Process control: fork(), exec(), exit()
* File management: open(), read(), write()
* Device management: ioctl()
* Communication: pipe(), socket()

**How it works:**

1. User program calls a library function.
2. The function executes a **trap instruction** to switch from user mode to kernel mode.
3. The OS performs the requested service.
4. Control returns to the user process.

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

**Monolithic Kernel:**

* Entire OS (file system, memory management, device drivers) runs in **kernel mode**.
* Example: Linux, UNIX.
* Advantages: Fast performance.
* Disadvantages: Large size, less secure (a bug in one part may crash the whole system).

**Microkernel:**

* Only core services (scheduling, memory management, IPC) are in kernel.
* Other services like file system, drivers run in **user mode**.
* Example: MINIX, QNX.
* Advantages: More secure, modular, easier to maintain.
* Disadvantages: Slower due to frequent user-kernel switches.

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

The OS acts as a **manager** between applications and hardware devices.

**Steps in I/O Handling:**

1. Application makes an I/O request via system call.
2. OS checks validity and forwards request to device driver.
3. Device driver communicates with hardware controller.
4. Device signals completion via **interrupt**.
5. OS updates the process’s I/O status.

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

A **race condition** occurs when multiple processes/threads access shared data simultaneously, and the final result depends on execution timing.

**Example:** Two threads updating a shared bank balance at the same time → inconsistent result.

**Prevention:**

* **Locks/Mutexes** – only one thread can access a resource at a time.
* **Semaphores** – control access to shared resources.
* **Monitors & Condition variables** – higher-level synchronization.
* **Atomic operations** – indivisible operations at hardware level.

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

A **device driver** is a software component that allows the OS and applications to interact with hardware devices.

**Functions:**

* Provides abstraction (applications don’t need to know hardware details).
* Converts OS requests into device-specific commands.
* Handles interrupts from hardware.
* Manages buffering and error handling.

**Examples:** Printer driver, Network card driver, Graphics driver.

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

A **zombie process** is a process that has finished execution but still remains in the process table because its parent hasn’t read its exit status.

**How it occurs:**

* Child process terminates.
* Parent doesn’t call wait() or waitpid().
* Child remains as a zombie.

**Prevention:**

* Parent should call wait() to clean up child entries.
* Use **signal handler (SIGCHLD)** to automatically reap child processes.

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

processes?

An **orphan process** is a process whose parent has terminated before the child finishes execution.

**Handling by OS:**

* Orphan processes are **adopted by init (PID 1)** in Unix/Linux.
* init becomes their new parent and reaps them when they finish.

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

A **parent process** creates a **child process** using fork().

The child inherits some properties from the parent (open files, environment variables, code).

Both run independently but may communicate via **IPC**.

Parent may **wait()** for the child to finish or continue execution.

If parent dies → child becomes an **orphan**.

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

* fork() creates a new process (child) that is a copy of the parent.
* Both parent and child continue execution after fork().
* Return values:
  + Parent gets child’s **PID**.
  + Child gets 0.
  + On failure, -1 is returned.
* The child has its own **PID**, memory space, and program counter.

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

The parent can use the **wait() or waitpid()** system call.

* **wait()** – Blocks the parent until any child terminates.
* **waitpid(pid, &status, options)** – Allows parent to wait for a specific child or check child status without blocking.
* Parent collects the child’s **exit status**, preventing zombies.

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

When a child process terminates, it returns an **exit status** to its parent.

The parent retrieves this using **wait() or waitpid()**.

The exit status tells whether the child:

* **Exited normally** (with a return code).
* **Terminated abnormally** (due to a signal).

Significance:

* Helps parent decide next steps.
* Prevents zombie processes.
* Provides debugging information.

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

* The parent can use the **kill(pid, signal)** system call.
* Example:
  + kill(pid, SIGTERM) → asks child to terminate gracefully.
  + kill(pid, SIGKILL) → forcefully kills the child (cannot be ignored).
* Parent must also **wait()** for the child to avoid zombies.

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

**Process Group:**

* A collection of related processes.
* Identified by a process group ID (PGID).
* Used for job control (e.g., stopping/continuing jobs in shell).

**Session:**

* A collection of process groups.
* Identified by a session ID.
* Created by a login shell.
* Used for terminal management.

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

**exec() family** (execl, execv, execp, etc.) replaces the current process code with a new program.

PID remains the same, but the code, data, and stack are replaced.

Used after fork() → child calls exec() to run a different program.

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

**wait()** → Waits for **any** child process to terminate.

**waitpid(pid, &status, options)** → Waits for a **specific child**.

Differences:

* wait() blocks until any child exits.
* waitpid() gives more control (choose child PID, use options like WNOHANG for non-blocking).

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

1. Process calls exit(status) or is killed by a signal.

2. All resources (memory, open files) are released.

3. Exit status is stored in PCB for parent to collect.

4.Process enters **zombie state** until parent calls wait().

5.Once parent collects status, PCB is removed → process fully terminated.

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?

**Long-term scheduler (job scheduler):**

* Decides **which jobs** from the job queue are admitted into the system.
* Controls **degree of multiprogramming** (how many processes are loaded in memory).

If too many processes → CPU thrashing.

If too few processes → CPU underutilized.

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?

**Long-term scheduler:**

* Executes **rarely**.
* Controls job admission (multiprogramming).

**Medium-term scheduler:**

* Executes **occasionally**.
* Suspends/resumes processes (swapping).

**Short-term scheduler (CPU scheduler):**

* Executes **very frequently** (milliseconds).
* Chooses which process runs next on CPU.

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

**Scenario:** In a time-sharing system, too many processes are in ready queue, causing CPU to switch frequently.

**Medium-term scheduler action:**

* Suspends some processes (swaps them to disk).
* Frees CPU and memory for others.
* Later resumes the suspended processes when resources are available.