

# Concepts of Operating System

## Assignment 2

### Part A

What will the following commands do?

- `echo "Hello, World!"`

```
cdac@Patil: ~  
cdac@Patil:~$ echo "hello world"  
hello world  
cdac@Patil:~$ |
```

- `name="Productive"`

```
cdac@Patil:~$ name="Productive"  
cdac@Patil:~$ echo $name  
Productive  
cdac@Patil:~$ |
```

- `touch file.txt`

```
cdac@Patil:~$ touch file.txt  
cdac@Patil:~$ ls -l  
total 40  
drwxr-xr-x 4 user1 cdac 4096 Feb 27 15:24 Feb25  
drwxr-xr-x 4 cdac cdac 4096 Feb 27 18:38 LinuxAssignment  
-rw-rw-r-- 1 cdac cdac 40 Feb 28 19:16 a13  
-rw-rw-r-- 1 cdac cdac 112 Feb 28 19:20 a14  
-rw-rw-r-- 1 cdac cdac 98 Feb 28 19:28 a15  
-rw-rw-r-- 1 cdac cdac 351 Feb 28 19:39 a17  
-rw-rw-r-- 1 cdac cdac 428 Feb 28 19:42 a18  
lrwxrwxrwx 1 cdac cdac 5 Feb 26 19:38 demo3 -> demo2  
-rw-rw-r-- 1 cdac cdac 0 Mar 1 17:47 file.txt  
drwxr-xr-x 2 cdac cdac 4096 Feb 26 19:37 linked  
-rw-r--r-- 1 cdac cdac 55 Feb 27 15:03 myfile.txt  
drwxr-xr-x 2 cdac cdac 4096 Feb 27 19:33 test  
cdac@Patil:~$ |
```

- ls -a

```
cdac@Patil:~$ ls -a
.          .config      .profile      a13  demo3
..         .landscape   .sudo_as_admin_successful a14  file.txt
.bash_history .lessht      .viminfo      a15  linked
.bash_logout .local       Feb25         a17  myfile.txt
.bashrc      .motd_shown  LinuxAssignment a18  test
cdac@Patil:~$ |
```

- rm file.txt

```
cdac@Patil:~$ rm file.txt
cdac@Patil:~$ ls -l
total 40
drwxr-xr-x 4 user1 cdac 4096 Feb 27 15:24 Feb25
drwxr-xr-x 4 cdac cdac 4096 Feb 27 18:38 LinuxAssignment
-rw-rw-r-- 1 cdac cdac 40 Feb 28 19:16 a13
-rw-rw-r-- 1 cdac cdac 112 Feb 28 19:20 a14
-rw-rw-r-- 1 cdac cdac 98 Feb 28 19:28 a15
-rw-rw-r-- 1 cdac cdac 351 Feb 28 19:39 a17
-rw-rw-r-- 1 cdac cdac 428 Feb 28 19:42 a18
lrwxrwxrwx 1 cdac cdac 5 Feb 26 19:38 demo3 -> demo2
drwxr-xr-x 2 cdac cdac 4096 Feb 26 19:37 linked
-rw-r--r-- 1 cdac cdac 55 Feb 27 15:03 myfile.txt
drwxr-xr-x 2 cdac cdac 4096 Feb 27 19:33 test
cdac@Patil:~$ |
```

- cp file1.txt file2.txt

```
cdac@Patil:~$ mkdir ass2
cdac@Patil:~$ cd ass2
cdac@Patil:~/ass2$ touch file1.txt
cdac@Patil:~/ass2$ cp file1.txt file2.txt
cdac@Patil:~/ass2$ ls
file1.txt  file2.txt
cdac@Patil:~/ass2$ ls -l
total 0
-rw-r--r-- 1 cdac cdac 0 Mar 1 17:49 file1.txt
-rw-r--r-- 1 cdac cdac 0 Mar 1 17:50 file2.txt
cdac@Patil:~/ass2$ |
```

- mv file.txt /path/to/directory/

Moves file.txt to the specified directory.

```
cdac@Patil:~$ mv /home/cdac/file.txt /home/user/documents/
cdac@Patil:~$ ls /home/user/documents/
file.txt
cdac@Patil:~$ |
```

- chmod 755 script.sh

```
cdac@Patil:~/ass2$ touch script.sh
cdac@Patil:~/ass2$ ls -l
total 0
-rw-r--r-- 1 cdac cdac 0 Mar  1 17:49 file1.txt
-rw-r--r-- 1 cdac cdac 0 Mar  1 17:50 file2.txt
-rw-r--r-- 1 cdac cdac 0 Mar  1 17:52 script.sh
cdac@Patil:~/ass2$ chmod 755 script.sh
cdac@Patil:~/ass2$ ls -l
total 0
-rw-r--r-- 1 cdac cdac 0 Mar  1 17:49 file1.txt
-rw-r--r-- 1 cdac cdac 0 Mar  1 17:50 file2.txt
-rwxr-xr-x 1 cdac cdac 0 Mar  1 17:52 script.sh
cdac@Patil:~/ass2$ |
```

- grep "pattern" file.txt

```
cdac@Patil:~/ass2$ cat file1.txt
cdac@Patil:~/ass2$ nano file1.txt
cdac@Patil:~/ass2$ grep "Ashwin" file1.txt
Ashwin
cdac@Patil:~/ass2$ |
```

- kill PID

```
cdac@Patil:~$ ps aux | grep <process_name>
-bash: syntax error near unexpected token `newline'
cdac@Patil:~$ pgrep <process_name>
-bash: syntax error near unexpected token `newline'
cdac@Patil:~$ kill 1234
-bash: kill: (1234) - No such process
cdac@Patil:~$ ps aux | grep firefox
cdac 58 0.0 0.0 4024 2044 pts/0 S+ 20:09 0:00 grep --color=auto firefox
cdac@Patil:~$ pgrep firefox
cdac 61 0.0 0.0 4024 2112 pts/0 S+ 20:09 0:00 grep --color=auto firefox
cdac@Patil:~$ kill 1234
-bash: kill: (1234) - No such process
cdac@Patil:~$ pgrep firefox
cdac@Patil:~$ kill 1234
-bash: kill: (1234) - No such process
cdac@Patil:~$ kill -9 1234
-bash: kill: (1234) - No such process
cdac@Patil:~$ |
```

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

```
cdac@Patil:~/ass2/mydir$ mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
Hello, World!
cdac@Patil:~/ass2/mydir/mydir$ |
```

- `ls -l | grep ".txt"`

```
cdac@Patil:~/ass2/mydir/mydir$ ls -l | grep ".txt"
-rw-r--r-- 1 cdac cdac 14 Mar  1 17:56 file.txt
cdac@Patil:~/ass2/mydir/mydir$ |
```

- `cat file1.txt file2.txt | sort | uniq`

```
cdac@Patil:~/ass2$ cat file1.txt file2.txt | sort | uniq

Ashwin
ashwin
patil
cdac@Patil:~/ass2$ |
```

- `ls -l | grep "^d"`

```
cdac@Patil:~/ass2$ ls -l | grep "^d"
drwxr-xr-x 3 cdac cdac 4096 Mar  1 17:56 mydir
cdac@Patil:~/ass2$ |
```

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

```
cdac@Patil:~$ grep -r "error" /home/user/logs/
/home/user/logs/file2.log:No errors here
/home/user/logs/file1.log:This is an error message
cdac@Patil:~$ |
```

- cat file1.txt file2.txt | sort | uniq -d

```
cdac@Patil:~/ass2$ cat file1.txt
apple
banana
cherry
cdac@Patil:~/ass2$ cat file2.txt
banana
cherry
date
cdac@Patil:~/ass2$ cat file1.txt file2.txt | sort | uniq -d
banana
cherry
cdac@Patil:~/ass2$ |
```

- chmod 644 file.txt

```
cdac@Patil:~/ass2$ chmod 644 file1.txt
cdac@Patil:~/ass2$ ls -l
total 12
-rw-r--r-- 1 cdac cdac 20 Mar 1 18:06 file1.txt
-rw-r--r-- 1 cdac cdac 19 Mar 1 18:06 file2.txt
drwxr-xr-x 3 cdac cdac 4096 Mar 1 17:56 mydir
-rwxr-xr-x 1 cdac cdac 0 Mar 1 17:52 script.sh
cdac@Patil:~/ass2$ |
```

- cp -r source\_directory destination\_directory

```
cdac@Patil:~/ass2$ cp -r mydir destination_directory
cdac@Patil:~/ass2$ mkdir source_directory
mkdir: cannot create directory 'source_directory': File exists
cdac@Patil:~/ass2$ cp -r source_directory destination_directory
cdac@Patil:~/ass2$ |
```

- find /path/to/search -name "\*.txt"

```
cdac@Patil:~$ mkdir user
cd user
mkdir document
touch myfile.txt
cdac@Patil:~/user$ cd ~/ass2/user/document
cdac@Patil:~/ass2/user/document$ touch file1.txt file2.txt
cdac@Patil:~/ass2/user/document$ find ~/ass2/user/document -name "*.txt"
/home/cdac/ass2/user/document/file1.txt
/home/cdac/ass2/user/document/file2.txt
```

- chmod u+x file.txt

```
cdac@Patil:~/ass2$ chmod u+x file1.txt
cdac@Patil:~/ass2$ ls -l
total 12
-rwxr--r-- 1 cdac cdac  20 Mar  1 18:06 file1.txt
-rw-r--r-- 1 cdac cdac  19 Mar  1 18:06 file2.txt
drwxr-xr-x 3 cdac cdac 4096 Mar  1 17:56 mydir
-rwxr-xr-x 1 cdac cdac   0 Mar  1 17:52 script.sh
cdac@Patil:~/ass2$ |
```

- echo \$PATH

```
cdac@Patil:~/ass2$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/local/games:/usr/lib/wsl/lib:/mnt/c/Program Files/Common Files/Orac
le/Java/javapath:/mnt/c/WINDOWS/system32:/mnt/c/WINDOWS:/mnt/c/WINDOWS/S
ystem32/Wbem:/mnt/c/WINDOWS/System32/WindowsPowerShell/v1.0:/mnt/c/WIND
OWS/System32/OpenSSH:/mnt/c/Program Files/Java/jdk-11/bin:/mnt/c/Users/p
atil/AppData/Local/Microsoft/WindowsApps:/mnt/c/Program Files/Java/jdk-
11:/snap/bin
cdac@Patil:~/ass2$ |
```

## 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

**Answer>>** cp is use for copy file

Cd is use for directories

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. >>>>> **mkdir**
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

Code//

```
echo "hello word"
```

```
cdac@Patil:~$ cd ass2
cdac@Patil:~/ass2$ touch sh1
cdac@Patil:~/ass2$ nano sh1
cdac@Patil:~/ass2$ bash sh1
hello word
cdac@Patil:~/ass2$ |
```

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

Code//

variable. Question

```
name="CDAC Mumbai"
```

```
echo "The name is :" $name
```

```
cdac@Patil:~/ass2$ bash sh2
The name is : CDAC Mumbai
cdac@Patil:~/ass2$ |
```

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

Code//

```
echo "Enter the number ":
```

```
read number
```

```
echo "The number is :" $number
```

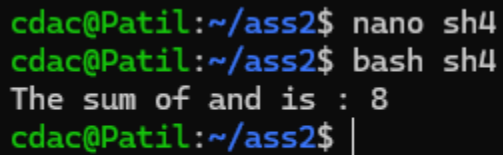
```
cdac@Patil:~/ass2$ bash sh3
Enter the number :
12
The number is : 12
cdac@Patil:~/ass2$ 3|
```



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

**Code//**

```
num=5
num2=3
sum=$((num + num2))
echo The sum of $num1 and $sum2 is : $sum
```

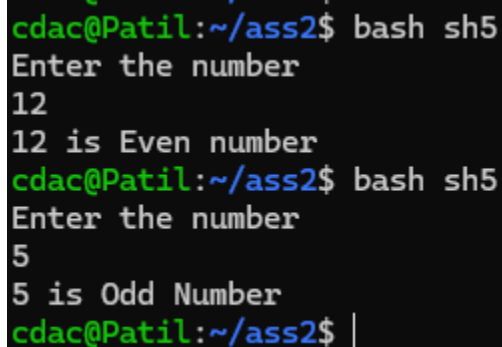
A terminal window showing the execution of a shell script. The prompt is 'cdac@Patil:~/ass2\$'. The user enters 'nano sh4', then 'bash sh4'. The script outputs 'The sum of and is : 8'. The prompt returns to 'cdac@Patil:~/ass2\$' with a cursor.

```
cdac@Patil:~/ass2$ nano sh4
cdac@Patil:~/ass2$ bash sh4
The sum of and is : 8
cdac@Patil:~/ass2$ |
```

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

**Code//**

```
echo Enter the number
read num
if [ $((num % 2)) -eq 0 ]
then
    echo $num is Even number
else
    echo $num is Odd Number
fi
```

A terminal window showing the execution of a shell script. The prompt is 'cdac@Patil:~/ass2\$'. The user enters 'bash sh5'. The script prompts 'Enter the number' and the user enters '12'. The script outputs '12 is Even number'. The user enters 'bash sh5' again. The script prompts 'Enter the number' and the user enters '5'. The script outputs '5 is Odd Number'. The prompt returns to 'cdac@Patil:~/ass2\$' with a cursor.

```
cdac@Patil:~/ass2$ bash sh5
Enter the number
12
12 is Even number
cdac@Patil:~/ass2$ bash sh5
Enter the number
5
5 is Odd Number
cdac@Patil:~/ass2$ |
```

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

**Code//**

```
for num in {1..5}
do
    echo Number : $num
done
```

```
cdac@Patil:~/ass2$ nano sh6
cdac@Patil:~/ass2$ bash sh6
Number : 1
Number : 2
Number : 3
Number : 4
Number : 5
cdac@Patil:~/ass2$ |
```

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

**Code//**

```
num=1
while [ $num -le 5 ]
do
    echo Number $num
    num=$((num + 1))
done
```

```
cdac@Patil:~/ass2$ nano sh8
cdac@Patil:~/ass2$ bash sh8
Number 1
Number 2
Number 3
Number 4
Number 5
cdac@Patil:~/ass2$ |
```

**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".

**Code//**

```
if [ -f "file1.txt" ]
then
    echo "file Exits"
else
    echo file does not exist:
fi
```

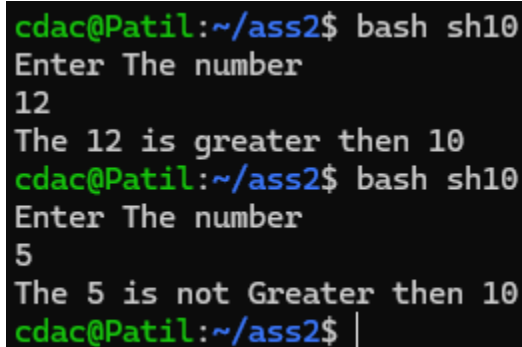
```
cdac@Patil:~/ass2$ nano sh9
cdac@Patil:~/ass2$ bash sh9
file Exits
cdac@Patil:~/ass2$ |
```

**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.

**Code//**

```
echo Enter The number
read num
```

```
if [ $num -gt 10 ]
then
    echo The $num is greater then 10
else
    echo The $num is not Greater then 10
fi
```



A terminal window showing the execution of the shell script. The prompt is 'cdac@Patil:~/ass2\$'. The user enters 'bash sh10'. The script prompts 'Enter The number' and the user enters '12'. The script outputs 'The 12 is greater then 10'. The user enters 'bash sh10' again. The script prompts 'Enter The number' and the user enters '5'. The script outputs 'The 5 is not Greater then 10'. The prompt returns to 'cdac@Patil:~/ass2\$'.

**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

**code//**

```
echo Multiplication table
echo -n " | "
for i in {1..5}
do
    printf "%4d" $i
done
echo
echo "-----"
for i in {1..5}
do
    printf "%2d | " $i
    for i in {1..5}
    do
        printf "%4d" $((i*i))
    done
    echo
done
```

```
cdac@Patil:~/ass2$ nano sh11
cdac@Patil:~/ass2$ bash sh11
Multiplication table
| 1 2 3 4 5
-----
1 | 1 4 9 16 25
2 | 1 4 9 16 25
3 | 1 4 9 16 25
4 | 1 4 9 16 25
5 | 1 4 9 16 25
cdac@Patil:~/ass2$
```

**Question 11:** Write a shell script that uses a while loop to read numbers from the user until the user enters a negative number. For each positive number entered, print its square. Use the break statement to exit the loop when a negative number is entered.

**Code//**

```
while true
do
    echo "Enter a number (Negative number to exit)"
    read num
    if [ $num -lt 0 ]
    then
        echo Negative number
        break
    fi
    sq=$((num * num))
    echo square of $num is $sq
done
```

```
cdac@Patil:~/ass2$ bash sh12
Enter a number (Negative number to exit)
12
square of 12 is 144
Enter a number (Negative number to exit)
4
square of 4 is 16
Enter a number (Negative number to exit)
-2
Negative number
cdac@Patil:~/ass2$
```

## 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 interface between computer hardware and users. It manages hardware resources and provides services to applications.

**Primary functions:**

- **Process Management:** Scheduling and execution of processes.
  - **Memory Management:** Allocating and deallocating memory.
  - **File System Management:** Organizing and managing files.
  - **Device Management:** Communicating with hardware through drivers.
  - **Security and Access Control:** Authentication and authorization.
  - **User Interface:** Command-line or graphical interaction.
- 

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

- A **process** is an independent program in execution with its own memory space and system resources.
- A **thread** is a lightweight unit of execution within a process that shares memory and resources with other threads in the same process.

Feature	Process	Thread
Memory	Has its own memory space	Shares memory with parent process
Communication	Uses Inter-Process Communication (IPC)	Shares data directly
Context Switching	Slower	Faster
Independence	Independent execution	Dependent on parent process

---

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

**Virtual memory** is a memory management technique that allows programs to use more memory than physically available RAM.

**How it works:**

- The OS divides memory into **pages**.
- Active pages stay in RAM, while inactive pages move to disk (**swap space**).

- When needed, the OS **swaps** pages back into RAM using **paging**.
- This enables **multitasking** and prevents **out-of-memory** errors.

---

#### 4. Difference between multiprogramming, multitasking, and multiprocessing.

Feature	Multiprogramming	Multitasking	Multiprocessing
Definition	Multiple programs loaded into memory, but only one runs at a time	Multiple programs execute seemingly simultaneously by time-sharing CPU	Multiple CPUs execute multiple processes simultaneously
Example	Batch systems	Windows OS running multiple applications	Dual-core or multi-core processors running separate tasks

---

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

A **file system** organizes and manages files stored on storage devices.

##### Components:

- **Files:** Data storage units.
  - **Directories:** Containers for organizing files.
  - **Inodes:** Metadata about files (size, permissions, timestamps).
  - **File Allocation Table (FAT) / Index Nodes:** Structures that track file locations.
  - **Mounting System:** Connects a storage device to the OS.
- 

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

A **deadlock** occurs when processes hold resources and wait indefinitely for other resources held by other processes.

##### Deadlock Prevention Methods:

1. **Avoidance:** Use **Banker's Algorithm** to allocate resources safely.
2. **Prevention:**
  - Remove **mutual exclusion** (share resources if possible).
  - Remove **hold and wait** (allocate resources in advance).
  - Remove **no preemption** (forcefully take back resources if needed).

- Remove **circular wait** (assign priority order to resources).
3. **Detection & Recovery:** Identify and terminate deadlocked processes.
- 

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

- **Kernel:** The core of the OS that manages hardware and system calls.
- **Shell:** A user interface (CLI or GUI) that interacts with the kernel.

Feature	Kernel	Shell
Role	Core OS component managing hardware	Interface between user and OS
Access	Runs in privileged mode	Runs in user mode
Example	Linux Kernel, Windows NT Kernel	Bash, PowerShell, Command Prompt

---

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

**CPU scheduling** determines which process gets CPU time to optimize performance.

### Importance:

- Maximizes **CPU utilization**.
- Reduces **waiting time** for processes.
- Ensures **fairness** among processes.
- Supports **multitasking** efficiently.

### Common CPU Scheduling Algorithms:

- **FCFS (First-Come-First-Serve)**
  - **SJF (Shortest Job First)**
  - **Round Robin**
  - **Priority Scheduling**
  - **Multilevel Queue Scheduling**
- 

## 9. How does a system call work?

A **system call** allows user programs to request OS services like file access, memory allocation, and process control.

### Steps in a System Call:

1. The program makes a **system call request** using a library function (open(), read()).
2. The **user mode** switches to **kernel mode**.
3. The OS **validates the request** and executes it.
4. The OS **returns control** to the user program.

**Examples:**

- **File Operations:** open(), read(), write(), close().
  - **Process Control:** fork(), exec(), exit().
  - **Memory Management:** malloc(), free().
- 

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

**Device drivers** allow the OS to communicate with hardware devices like printers, keyboards, and graphics cards.

**Roles of Device Drivers:**

- Convert **OS commands** into device-specific signals.
  - Ensure **compatibility** between hardware and OS.
  - Manage **I/O operations** efficiently.
  - Allow **plug-and-play functionality**.
- 

**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** to **physical addresses** in memory.

**Role of the Page Table:**

- **Address Translation:** Converts virtual addresses into physical addresses.
- **Memory Protection:** Stores access control information to prevent unauthorized access.
- **Paging Support:** Helps in implementing demand paging.
- **Reduces Fragmentation:** Enables non-contiguous memory allocation.

The **Translation Lookaside Buffer (TLB)** is a special cache that stores recent page table mappings to speed up memory access.

---

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



**Thrashing** occurs when excessive paging (swapping between RAM and disk) reduces system performance.

**Causes of Thrashing:**

- Insufficient **physical memory**.
- Too many processes competing for memory.
- Poor page replacement policies.

**Avoiding Thrashing:**

- **Increase RAM** to accommodate more processes.
  - **Use Working Set Model** to allocate memory based on process demand.
  - **Improve Page Replacement Algorithms** (e.g., LRU, Optimal).
  - **Reduce Multiprogramming** if memory is limited.
- 

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

A **semaphore** is a synchronization primitive used to control access to shared resources in concurrent programming.

**Types of Semaphores:**

- **Binary Semaphore (Mutex):** Can only be 0 or 1 (used for locking).
- **Counting Semaphore:** Used to allow multiple accesses to a resource.

**Use in Synchronization:**

- Prevents **race conditions** in critical sections.
  - Ensures **mutual exclusion** by allowing only one process to access a resource at a time.
  - Used for **process synchronization** (e.g., producer-consumer problem).
- 

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

The OS provides **synchronization mechanisms** to ensure that multiple processes or threads execute correctly when accessing shared resources.

**Methods for Synchronization:**

1. **Semaphores:** Used for resource access control.
2. **Mutex (Mutual Exclusion):** Locks and unlocks shared resources.
3. **Monitors:** High-level synchronization construct combining locks and condition variables.

4. **Atomic Operations:** Ensures operations execute without interruption.
5. **Inter-Process Communication (IPC):** Allows processes to coordinate actions.

#### Common Problems & Solutions:

- **Race Conditions** → Use Locks/Semaphores
  - **Deadlocks** → Implement Deadlock Prevention Algorithms
- 

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

An **interrupt** is a signal sent to the CPU to notify it of an event that requires immediate attention.

##### Purpose of Interrupts:

- Allows the CPU to respond to **asynchronous events** (e.g., I/O completion, errors).
- Improves system **efficiency** by allowing multitasking.
- Supports **hardware interactions** (e.g., keyboard, mouse, network).

##### Types of Interrupts:

1. **Hardware Interrupts:** Triggered by devices (e.g., pressing a key).
  2. **Software Interrupts:** Triggered by system calls (e.g., `int 0x80` in Linux).
  3. **Timer Interrupts:** Used in CPU scheduling for preemptive multitasking.
- 

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

A **file descriptor** is an integer used by the OS to reference an open file.

##### Common File Descriptors in Linux/Unix:

---

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

A system crash occurs due to hardware failures, software bugs, or resource exhaustion.

##### Recovery Methods:

- **Rebooting:** Restarting the system to restore normal operations.
- **Filesystem Check (fsck in Linux, chkdsk in Windows):** Repairs corrupted file systems.
- **Process Checkpointing:** Saves process states periodically to resume from a crash.
- **Journaling File Systems:** Logs transactions before execution to prevent data loss (e.g., ext4, NTFS).

- **Backup & Restore:** Restores lost data from backups.

---

## 18. Difference between a monolithic kernel and a microkernel.

Feature	Monolithic Kernel	Microkernel
Structure	All OS services in one large kernel	Only essential services in the kernel
Performance	Fast (direct communication)	Slower (message passing overhead)
Security	Less secure (entire OS runs in kernel mode)	More secure (only essential parts in kernel mode)
Example	Linux, Windows	QNX, Minix

In **Monolithic kernels**, drivers, file systems, and process management all reside in the kernel.

In **Microkernels**, only core functions remain in the kernel, while everything else runs in **user space**.

---

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

Type	Internal Fragmentation	External Fragmentation
Definition	Unused space <b>inside</b> allocated memory blocks	Unused space <b>between</b> allocated memory blocks
Cause	Fixed-size memory allocation	Dynamic memory allocation
Solution	Use <b>paging</b>	Use <b>compaction</b> or <b>segmentation</b>

Example:

- **Internal Fragmentation:** A 512-byte block is allocated for a 500-byte file, leaving 12 bytes wasted.
  - **External Fragmentation:** Multiple small free spaces exist, but none are large enough for a new allocation.
- 

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

The OS handles I/O operations using **device drivers, buffering, and scheduling**.

### I/O Management Components:

1. **Device Drivers:** Provide an interface between hardware and OS.
2. **Interrupt Handlers:** Manage device interrupts for efficient processing.
3. **Buffering & Caching:** Temporarily store data to optimize I/O speed.

4. **Spooling:** Queues I/O requests to improve performance (used in printing).
5. **I/O Scheduling:** Determines the order in which I/O requests are handled.

#### I/O Scheduling Algorithms:

- **FCFS (First Come, First Serve)**
- **SSTF (Shortest Seek Time First)**
- **SCAN (Elevator Algorithm)**

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

Feature	Preemptive Scheduling	Non-Preemptive Scheduling
Definition	The CPU can be taken away from a running process.	The CPU is not taken away until the process completes.
Interrupts	A process can be interrupted mid-execution.	A process runs until it voluntarily releases the CPU.
Efficiency	More responsive, but more overhead due to context switching.	Less overhead but less responsive.
Example	Round Robin (RR), Shortest Remaining Time First (SRTF), Priority Preemptive	First Come First Serve (FCFS), Shortest Job First (SJF) Non-Preemptive

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

**Round-Robin (RR) Scheduling** is a **preemptive CPU scheduling algorithm** where each process is assigned a fixed time slice (time quantum).

##### Working Mechanism:

1. All processes are placed in a circular queue.
2. The CPU executes each process for a fixed **time quantum**.
3. If a process doesn't finish, it is **preempted** and moved to the end of the queue.
4. The next process in the queue gets the CPU.
5. This cycle repeats until all processes are completed.

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

**Priority Scheduling** assigns a priority to each process, and the **CPU is allocated to the highest-priority process** first.

##### Types:

- **Preemptive Priority Scheduling:** If a higher-priority process arrives, it **interrupts** the current process.
- **Non-Preemptive Priority Scheduling:** The CPU finishes executing the current process before switching.

#### Priority Assignment:

- **Static Priority:** Assigned at process creation (e.g., system vs. user processes).
- **Dynamic Priority:** Adjusted based on process behavior (e.g., aging to prevent starvation).

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

**Shortest Job Next (SJN)** or **Shortest Job First (SJF)** is a scheduling algorithm that selects the process with the **smallest burst time** to execute first.

#### Types:

- **Non-Preemptive SJN:** The process runs until completion.
- **Preemptive SJN (Shortest Remaining Time First - SRTF):** A shorter process can interrupt a longer one.

#### When is SJN used?

- **Used in batch processing** where execution time is known.
- **Minimizes average waiting time**, but suffers from **starvation** (longer processes may never get CPU time).

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

**Multilevel Queue Scheduling** divides processes into different **priority-based queues**, and each queue has its own scheduling algorithm.

#### Example of Queues:

1. **System Processes (High Priority) → Round Robin**
2. **Interactive Processes → Shortest Job First**
3. **Background Processes (Low Priority) → FCFS**

#### Characteristics:

- **Fixed priority order:** Higher-priority queues get executed first.
- **Preemption between queues:** High-priority queues can interrupt lower-priority queues.
- **Time-sharing within queues:** Different scheduling algorithms can be applied in each queue.

**Use Case:** Used in **real-time operating systems** where system tasks need priority over user applications.

---

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

A **Process Control Block (PCB)** is a **data structure** that stores information about a process.

**Contents of a PCB:**

- **Process ID (PID)** – Unique identifier.
- **Process State** – Ready, Running, Blocked, etc.
- **Program Counter (PC)** – Address of the next instruction.
- **CPU Registers** – Stores execution context.
- **Memory Management Info** – Page tables, segment tables.
- **I/O Status** – Open files, I/O devices.
- **Priority** – Scheduling information.

The **PCB helps the OS manage processes** by keeping track of process states and resources.

---

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

**Process States:**

1. **New:** Process is created.
2. **Ready:** Process is waiting for CPU execution.
3. **Running:** Process is executing on the CPU.
4. **Waiting (Blocked):** Process is waiting for I/O or other events.
5. **Terminated:** Process has finished execution.

**State Transitions:**

- **New → Ready:** When the process is admitted.
  - **Ready → Running:** When the scheduler assigns CPU.
  - **Running → Waiting:** If the process requests I/O.
  - **Waiting → Ready:** If I/O is completed.
  - **Running → Terminated:** If the process completes execution.
- 

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

Processes communicate using **Inter-Process Communication (IPC)** methods:

**1. Shared Memory:**

- Processes access the same memory segment.
- Fast, but requires synchronization (e.g., semaphores).

**2. Message Passing:**

- Uses `send()` and `receive()` system calls.
- Slower than shared memory but safer.

**3. Pipes:**

- One-way communication (e.g., between parent and child processes).
- Example: `ls | grep txt`

**4. Sockets:**

- Allows communication between processes on different machines.
- Used in networking.

---

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

Process Synchronization ensures that multiple processes **execute in a coordinated manner** when accessing shared resources.

**Why is it important?**

- Prevents **race conditions** where multiple processes modify shared data inconsistently.
- Ensures **data consistency** and **mutual exclusion**.
- Used in **multithreading** to avoid conflicts.

**Synchronization Mechanisms:**

- **Semaphores**
- **Mutex Locks**
- **Monitors**
- **Atomic Variables**

Example: **Producer-Consumer Problem**, where a buffer is shared between a producer adding items and a consumer removing items.

---

### 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 because its **parent has not read its exit status**.

#### How is it created?

1. A child process completes execution (`exit()`), but the parent does not call `wait()`.
2. The process remains in the **zombie state** until the parent retrieves the exit status.

#### How to prevent zombies?

- Use **`wait()` or `waitpid()`** in the parent process.
- Make the parent **ignore `SIGCHLD` signals** (`signal(SIGCHLD, SIG_IGN);`).
- If the parent terminates, the zombie process is adopted by **`init`**, which cleans it up.

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

Fragmentation Type	Internal Fragmentation	External Fragmentation
Definition	Unused memory <b>inside</b> allocated blocks.	Unused memory <b>outside</b> allocated blocks.
Cause	Memory blocks are allocated larger than requested.	Free memory is fragmented into non-contiguous blocks.
Example	If a process requests <b>28 KB</b> , but memory is allocated in <b>32 KB</b> blocks, 4 KB is wasted.	If a process of <b>8 KB</b> is freed from a <b>32 KB</b> block, but new processes require exactly <b>32 KB</b> , they cannot fit.
Solution	Use smaller, more precise allocation units (Paging).	Use <b>compaction, segmentation, or paging</b> to combine free memory.

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

**Demand Paging** is a technique where pages are **loaded into memory only when needed**, instead of loading the entire process at once.

#### How it works:

1. A process starts execution with only essential pages loaded.
2. When a required page is missing, a **page fault** occurs.
3. The OS loads the missing page into memory from the disk.

#### Benefits:



- **Reduces memory waste** by keeping only needed pages in RAM.
  - **Improves system performance** by running larger programs with limited memory.
  - **Reduces loading time**, since unnecessary pages are not loaded initially.
- 

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

A **page table** is a data structure used by the OS to **map virtual addresses to physical addresses** in memory.

#### Functions of a Page Table:

- **Keeps track of memory pages** assigned to a process.
- **Translates virtual addresses** into physical addresses.
- **Stores page frame numbers** and protection bits (read/write permissions).

#### Example:

If process **P1** accesses virtual address **0x1234**, the page table determines the **physical location** of that data.

#### Optimization techniques:

- **TLB (Translation Lookaside Buffer)** speeds up address translation.
  - **Multi-level paging** reduces memory overhead for large address spaces.
- 

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

A **Memory Management Unit (MMU)** is a hardware component that **translates virtual addresses into physical addresses**.

#### How MMU works:

1. **CPU generates a virtual address.**
2. **MMU looks up the page table** to find the corresponding **physical address**.
3. If the required page is in memory (**TLB hit**), it is accessed directly.
4. If the page is not in memory (**page fault**), it is loaded from disk.

#### Key Features of MMU:

- **Provides memory protection** (prevents illegal memory access).
- **Supports virtual memory** by enabling paging and segmentation.
- **Enhances multitasking** by isolating process memory spaces.

---

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

**Thrashing** occurs when a system **spends more time swapping pages in and out of memory** rather than executing processes.

#### Causes:

- **Insufficient RAM** for running processes.
- **High page fault rate**, leading to excessive disk I/O.
- **Too many processes competing for memory.**

#### How to Avoid Thrashing:

1. **Increase RAM** – Provides more physical memory.
2. **Use Working Set Model** – Ensures processes have enough frames to execute efficiently.
3. **Use Page Replacement Algorithms** – Such as **LRU (Least Recently Used)** to keep frequently used pages in memory.
4. **Adjust Multiprogramming Level** – Reduce the number of processes competing for memory.

---

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

A **system call** is a function used by user programs to request services from the **operating system kernel**.

#### How System Calls Work:

1. A program requests a service (e.g., reading a file).
2. The CPU switches to **kernel mode** to execute the request.
3. The OS performs the operation and returns the result to the program.

#### Examples of System Calls:

- **Process Control:** `fork()`, `exec()`, `exit()`.
- **File Management:** `open()`, `read()`, `write()`.
- **Memory Management:** `mmap()`, `brk()`.
- **I/O Control:** `ioctl()`, `select()`.

---

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

Feature	Monolithic Kernel	Microkernel
Structure	All OS services (file system, memory, drivers) are in one large kernel.	Only core functions (process, memory, and IPC) are in the kernel. Other services run in user space.
Performance	Fast due to direct communication.	Slower due to message-passing overhead.
Stability	Less stable – a crash can bring down the whole system.	More stable – faulty services don't affect the entire OS.
Example OS	Linux, Windows, Unix.	Minix, QNX, macOS (partially).

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

An OS manages **I/O operations** using the following components:

1. **Device Drivers** – Act as an interface between hardware and OS.
2. **I/O Scheduling** – Determines the order of I/O requests (e.g., disk scheduling algorithms like **FCFS**, **SSTF**).
3. **Interrupts** – Notifies the CPU when an I/O operation is complete.
4. **DMA (Direct Memory Access)** – Allows data transfer between devices and memory **without CPU intervention**, improving performance.

Example: When reading a file,

- The OS sends a request to the **disk driver**.
- The driver fetches data using **DMA**.
- An **interrupt** informs the CPU when data is ready.

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

A **race condition** occurs when **multiple processes or threads access shared data simultaneously**, leading to inconsistent results.

**Example:**

Two threads **increment the same counter** at the same time.

- **Thread A:** Reads counter = 5, increments to 6, writes back.
- **Thread B:** Reads counter = 5, increments to 6, writes back.
- The final value **should be 7**, but it remains **6** due to lost updates.

**How to Prevent Race Conditions:**

1. **Mutex Locks** – Allow only one thread to access the resource at a time.
  2. **Semaphores** – Use counters to manage access.
  3. **Atomic Operations** – Ensure operations are completed without interruption.
  4. **Monitor Constructs** – High-level synchronization primitives.
- 

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

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

##### Functions of Device Drivers:

- **Translate OS requests into hardware commands.**
- **Provide an abstraction layer** between hardware and applications.
- **Handle device-specific protocols** (e.g., USB, PCI, SATA).
- **Optimize performance** by managing buffering and caching.

##### Example:

A **printer driver** translates a document into printer commands (e.g., PCL or PostScript) so the printer can understand it.

##### Types of Device Drivers:

1. **Character Drivers** – For devices like keyboards, mice, serial ports.
  2. **Block Drivers** – For storage devices (e.g., HDD, SSD).
  3. **Network Drivers** – For Ethernet, Wi-Fi cards.
- 

#### 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 **completed execution but still has an entry in the process table** because its parent has not read its exit status.

##### How it occurs:

1. A child process **completes execution** and terminates.
2. The parent **fails to call wait() or waitpid()** to collect the exit status.
3. The process remains in a "zombie" state, occupying a process table slot.

##### How to prevent zombie processes:

- The parent process should **call wait() or waitpid()** to remove the zombie.
- Use a **signal handler (SIGCHLD)** to automatically reap child processes.

- If the parent terminates before the child, **init (PID 1)** collects the **exit status**, preventing zombies.
- 

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

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

**How the OS handles orphan processes:**

- The **init process (PID 1)** adopts orphan processes in **Unix/Linux**.
  - **init ensures proper cleanup** by waiting for orphan processes to terminate, preventing them from becoming zombies.
- 

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

- A **parent process** creates a **child process** using `fork()`.
  - The child **inherits** a copy of the parent's memory space.
  - The parent **can control or monitor** the child using `wait()` or `kill()`.
  - Both processes **execute independently** unless synchronized.
- 

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

The `fork()` system call **creates a new child process by duplicating the parent process**.

**How it works:**

1. The parent calls `fork()`.
  2. The OS **creates a copy of the parent's address space**.
  3. The child **receives a new PID and runs independently**.
  4. Both parent and child execute the next instruction **with different return values** from `fork()`.
- 

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

A parent process **uses `wait()` or `waitpid()`** to wait for a child process to finish execution.

Here, `wait(NULL)`; makes the parent **pause until the child exits**.

---

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

- The exit status tells the **parent why the child terminated** (normal exit, error, or signal).
  - The **parent can retrieve this status using wait() or waitpid()**.
  - A successful exit typically returns **0**, while errors return **non-zero values**.
- 

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

The **parent process** can terminate a child using the kill() system call.

This forcefully stops the child **before it completes execution**.

---

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

Feature	Process Group	Session
Definition	A collection of related processes.	A collection of process groups.
Leader	One process is the <b>group leader</b> .	One process is the <b>session leader</b> .
Control	Signals can be sent to the <b>entire group</b> .	Sessions manage <b>multiple terminals</b> .
Example	Shell pipeline (`cmd1	cmd2

---

**49. Describe how the exec() family of functions replaces the current process image with a new one.**

The exec() functions replace the **current process with a new program**, keeping the same PID.

- =
- 

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

Feature	wait()	waitpid()
Waits for any child	Yes	No, only a specific child.
Non-blocking option	No	Yes (WNOHANG flag).
Can specify a child process	No	Yes.

---

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

A process terminates when:

- It calls **exit()** or **returns from main()**.
  - The **parent sends kill()** to it.
  - It receives an **unhandled fatal signal** (e.g., SIGKILL).
- 

**52. What is the role of the long-term scheduler in the process scheduling hierarchy?**

- The **long-term scheduler (job scheduler)** decides **which processes enter the system**.
  - It **controls the degree of multiprogramming** (number of processes in memory).
  - It **balances CPU-bound and I/O-bound processes** for system efficiency.
- 

**53. How does the short-term scheduler differ from the long-term and medium-term schedulers?**

Scheduler	Function	Frequency
Long-term	Decides which jobs to admit.	Infrequent.
Medium-term	Swaps processes in/out of memory.	Occasionally.
Short-term	Selects process to run on CPU.	Very frequent (milliseconds).

---

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

**Scenario:**

- A system **experiences high memory usage** due to too many active processes.
- The **medium-term scheduler swaps out some processes** to free memory.
- Once memory becomes available, **it swaps the process back in**.

**How it helps:**

- **Prevents thrashing** by limiting the number of processes in RAM.
- **Improves CPU efficiency** by ensuring active processes get memory.

## 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.

Process	Arrival Time	BurstTime	compltion time	TAT	Wating time	Response Time
p1	0	5	5	5	0	0
p2	1	3	8	7	4	4
p3	2	6	14	12	6	6
			AT=9	AT=8	AT=3.33	AT=3.33
Ganth Chat						
	0	5	8	14		



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

Process	Arrival Time	BurstTime	compltion time	TAT	Wating time	Response Time
p1	0	3	3	3	0	0
p2	1	5	13	12	7	7
p3	2	1	4	2	1	1
p4	3	4	8	5	1	1
			AT=7	AT =5.5	AT=2.22	AT=2.22
Ganth Chat	p1	p3	p4	p2	p2	
	0	3	4	8	13	

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.

Process	Arrival Time	Priority	BurstTime	compltion time	TAT	Wating time	Response Time
p1	0	6	3	6	6	0	0
p2	1	4	1	10	9	5	5
p3	2	7	4	19	7	10	10
p4	3	2	2	12	9	7	7
				AT=11.72	AT =7.75	AT=5.5	AT=5.5
Ganth Chat	p1	p2	p4	p3			
	0	6	10	12	19		

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

Process	Arrival Time	BurstTime	completion time	TAT	Waiting time	Response Time
p1	0	4	10	10	4	0
p2	1	5	14	13	8	1
p3	2	2	6	4	2	2
p4	3	3	13	10	7	3
			AT=10.75	AT=9.25	AT=5.25	AT=1.5
Gantt Chart						
	p1	p2	p3	p4	p1	p2
	0	2	4	6	8	10
						12
						13
						14

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() c

### Answer

- The parent process has a variable x with an initial value of 5.
- The fork() system call is executed, creating a child process.
- Both the parent and child processes have their own copies of the variable x with the initial value of 5.
- The parent and child processes are independent, so changes to x in one process do not affect the other.
- Both the parent and child processes increment their respective copies of **x + 1**.
- **Parent Process:**  
The parent process increments its copy of x from 5 to 6.
- **Child Process:**  
The child process also increments its copy of x from 5 to 6.