CDAC MUMBAI

**Concepts of Operating System Assignment 2**

# Part A

## What will the following commands do?

**1. echo "Hello, World!"**

* This command simply prints the message "Hello, World!" to the console. It's a basic way to output text.

**2. name="Productive"**

* This creates a variable named name and assigns the string value "Productive" to it. Variables can store data for later use in scripts.

**3. touch file.txt**

* This command creates an empty file named file.txt in the current working directory. If the file already exists, it does nothing.

**4. ls -a**

* This command lists all files and directories in the current directory, including hidden files (those starting with a dot). ls is the basic directory listing command.

**5. rm file.txt**

* This command permanently deletes the file file.txt from the current directory. Use caution, as deleted files cannot be easily recovered.

**6. cp file1.txt file2.txt**

* This command copies the file file1.txt to a new file named file2.txt in the current directory. The contents of file1.txt are copied to file2.txt.

**7. mv file.txt /path/to/directory/**

* This command moves the file file.txt to the specified directory (/path/to/directory/). The file will be relocated to the new location.

**8. chmod 755 script.sh**

* This command changes the file permissions of script.sh. The number 755 represents a specific set of permissions:
  + 7: Owner has read, write, and execute permissions.
  + 5: Group has read and execute permissions.
  + 5: Others have read and execute permissions.

**9. grep "pattern" file.txt**

* This command searches for lines in file.txt that contain the specified pattern ("pattern"). If any matches are found, they are printed to the console. grep is a powerful tool for text searching.

**10. kill PID**

* This command forcefully terminates a running process identified by its process ID (PID). Use this with caution, as it can disrupt ongoing tasks.

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

* This is a compound command using && to chain multiple actions:
  + Creates a directory named mydir.
  + Changes the working directory to mydir.
  + Creates an empty file named file.txt.
  + Writes the text "Hello, World!" to file.txt.
  + Prints the contents of file.txt to the console.

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

* This command uses a pipe (|) to combine two commands:
  + ls -l: Lists files and directories with detailed information (long format).
  + grep ".txt": Filters the output of ls -l to show only lines containing ".txt" (presumably filenames ending in ".txt").

**13. cat file1.txt file2.txt | sort | uniq**

* This command also uses pipes:
  + cat file1.txt file2.txt: Concatenates (combines) the contents of file1.txt and file2.txt.
  + sort: Sorts the combined lines alphabetically.
  + uniq: Removes duplicate lines from the sorted output.

**14. ls -l | grep "^d"**

* This command filters the output of ls -l to show only lines that start with "^d" (lines representing directories).

**15. grep -r "pattern" /path/to/directory/**

* This command searches for the pattern ("pattern") recursively within all files under the specified directory (/path/to/directory/). -r enables recursive searching.

**16. cat file1.txt file2.txt | sort | uniq –d**

* This command is similar to command 13, but the uniq option -d shows only duplicate lines (those that appear more than once) instead of removing them.

**17. chmod 644 file.txt**

* This command changes the file permissions of file.txt to:
  + 6: Owner has read and write permissions.
  + 4: Group has read permission only.
  + 4: Others have read permission only.

## I

## PartB

dentify True or False:

1. **True:** ls is used to list files and directories in a directory.
2. **True:** mv is used to move files and directories.
3. **False:** cd is used to change the current directory, not to copy files.
4. **True:** pwd stands for "print working directory" and displays the current directory.
5. **True:** grep is used to search for patterns in files.
6. **True:** chmod 755 file.txt gives the specified permissions to the owner, group, and others.
7. **True:** mkdir -p directory1/directory2 creates nested directories as described.
8. **True:** rm -rf file.txt deletes the file forcefully without confirmation.

## Identify the Incorrect Commands:

1. **Incorrect:** chmodx is not a valid command to change file permissions. The correct command is chmod.
2. **Incorrect:** cpy is not a valid command to copy files. The correct command is cp.
3. **Incorrect:** mkfile is not a valid command to create a new file. The correct command is touch.
4. **Incorrect:** catx is not a valid command to concatenate files. The correct command is cat.
5. **Incorrect:** rn is not a valid command to rename files. The correct command is mv.

# Part C

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

**echo "Hello, World!"**

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

**name="CDAC Mumbai" echo "Name: $name"**

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

**echo "Enter a number: " read number echo "You entered: $number"**

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

**echo "Enter the first number: " read num1 echo "Enter the second number: " read num2 result=$((num1 + num2)) echo "Sum: $result"**

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

**echo "Enter a number: " read num if ((num % 2 == 0)); then echo "Even" else echo "Odd" fi**

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

**for ((i=1; i<=5; i++)); do echo "$i" done**

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

**i=1 while [[ $i -le 5 ]]; do echo "$i" i=$((i+1)) done**

**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".

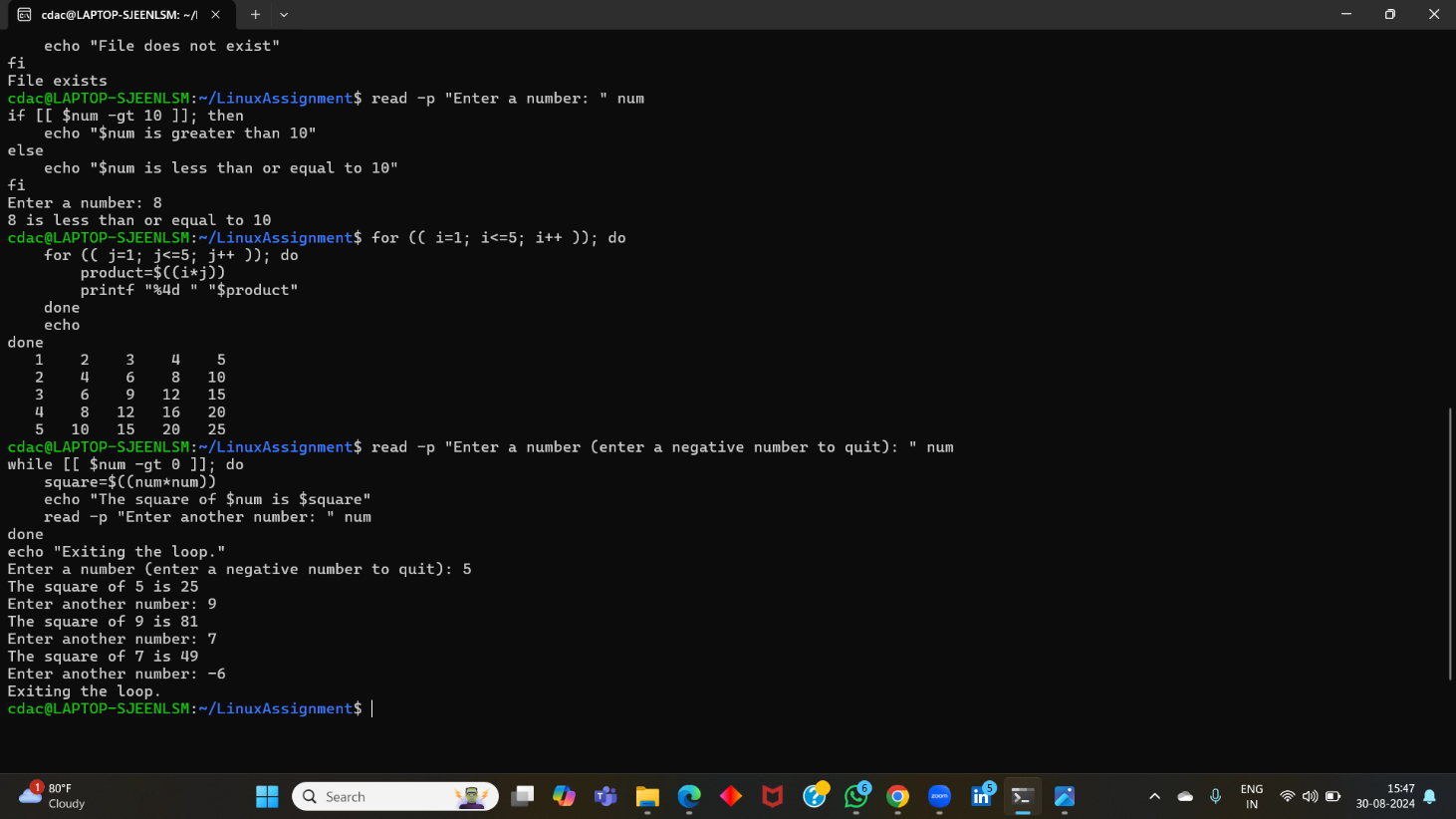
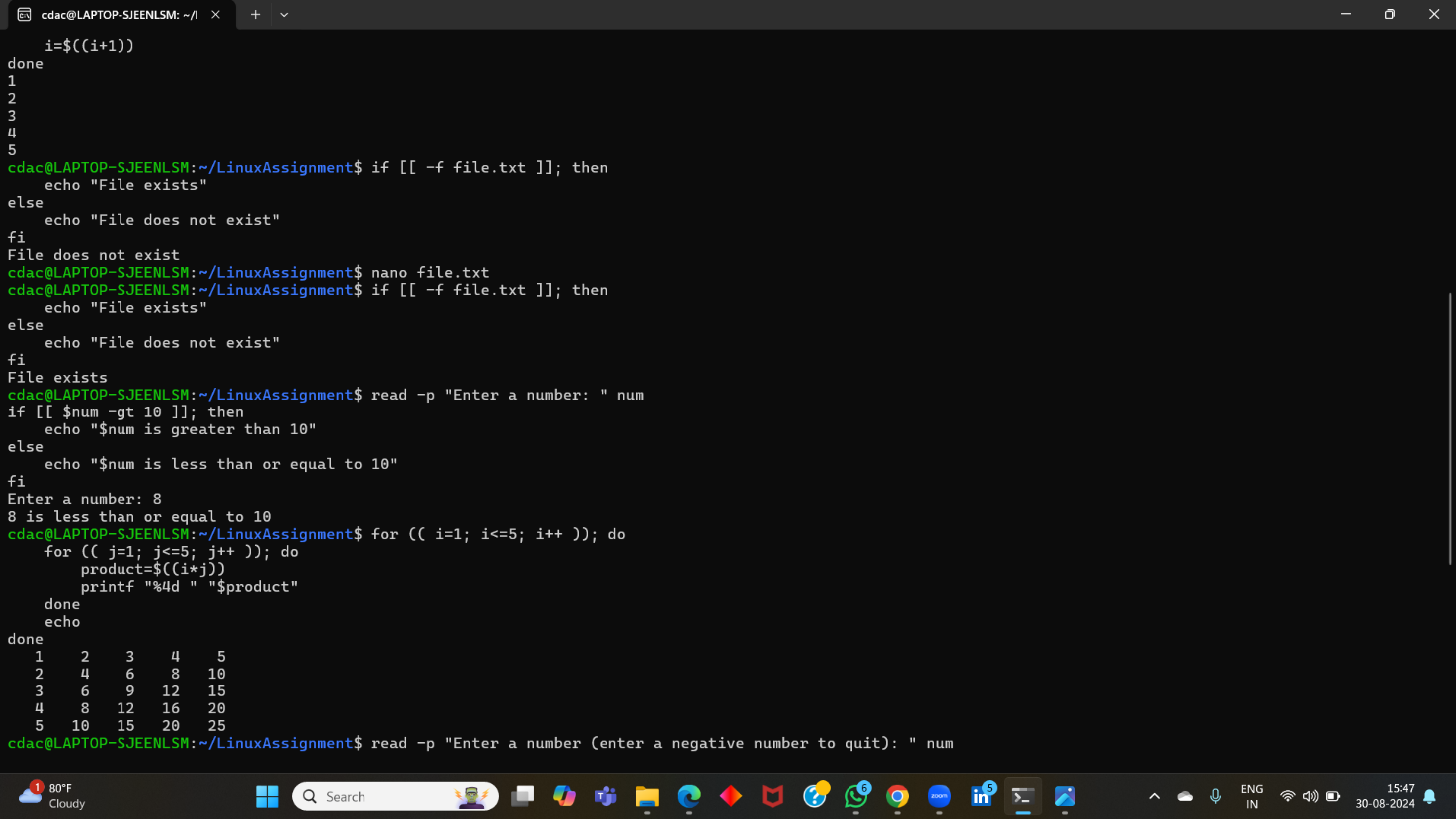
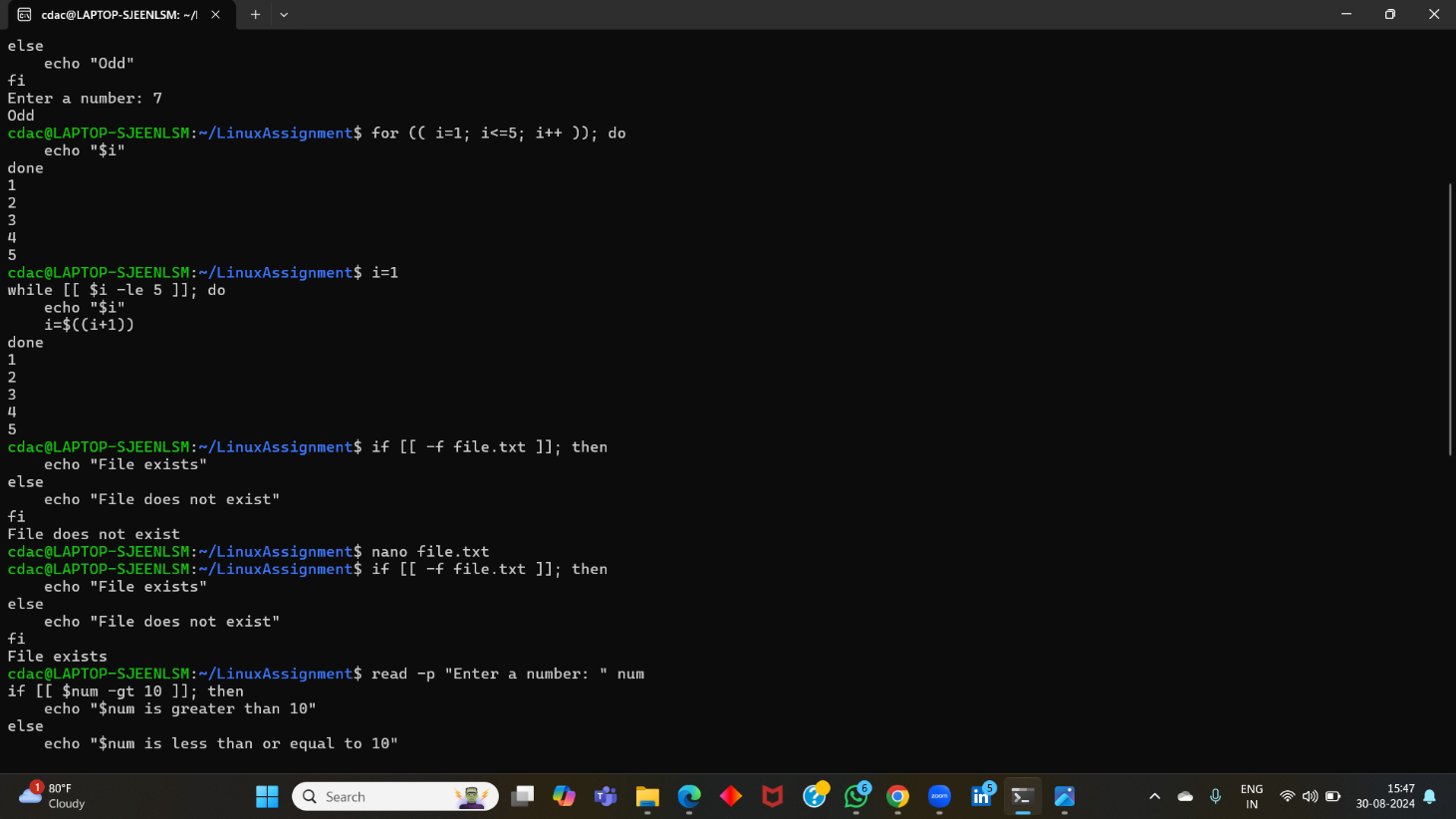
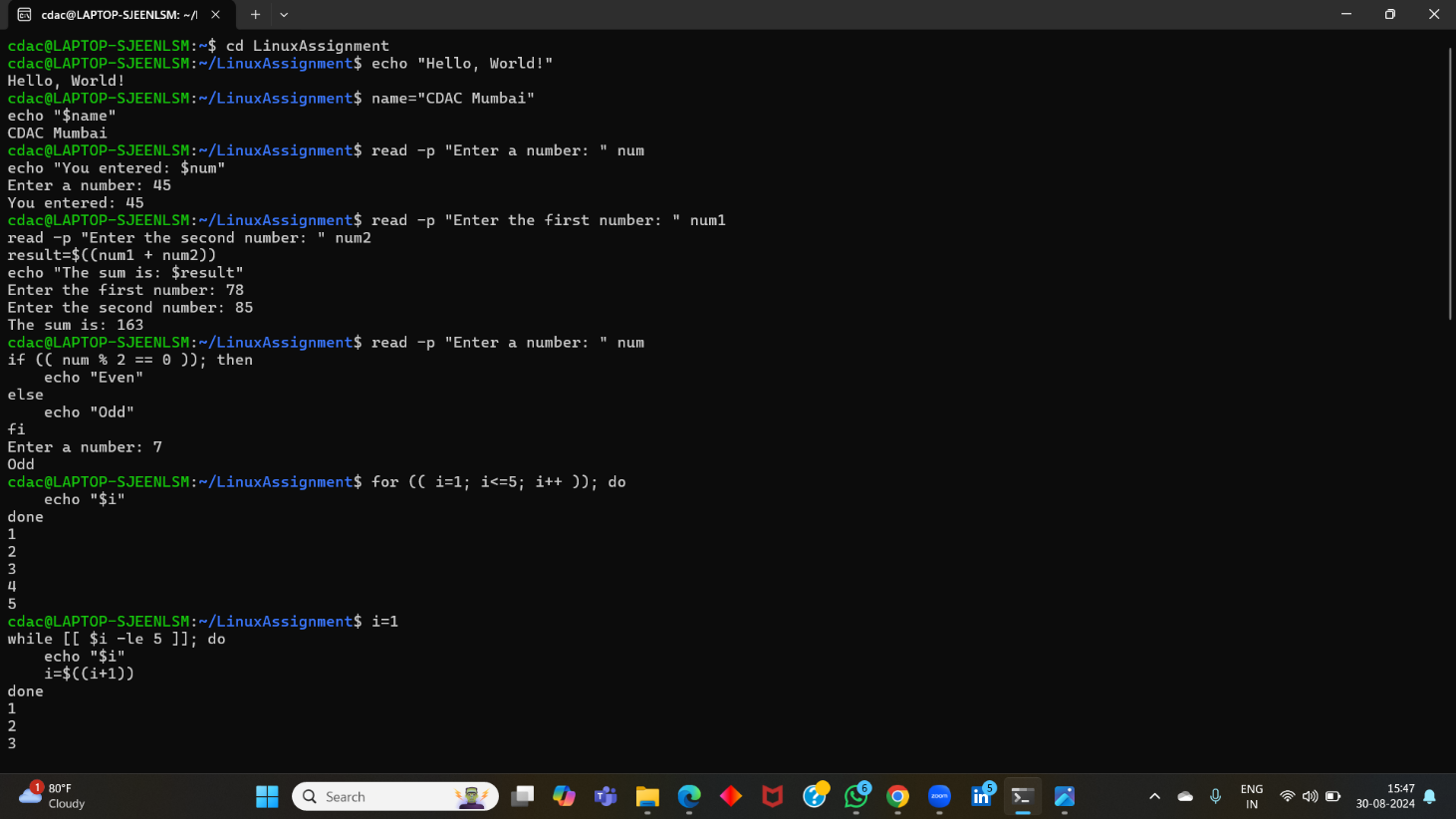
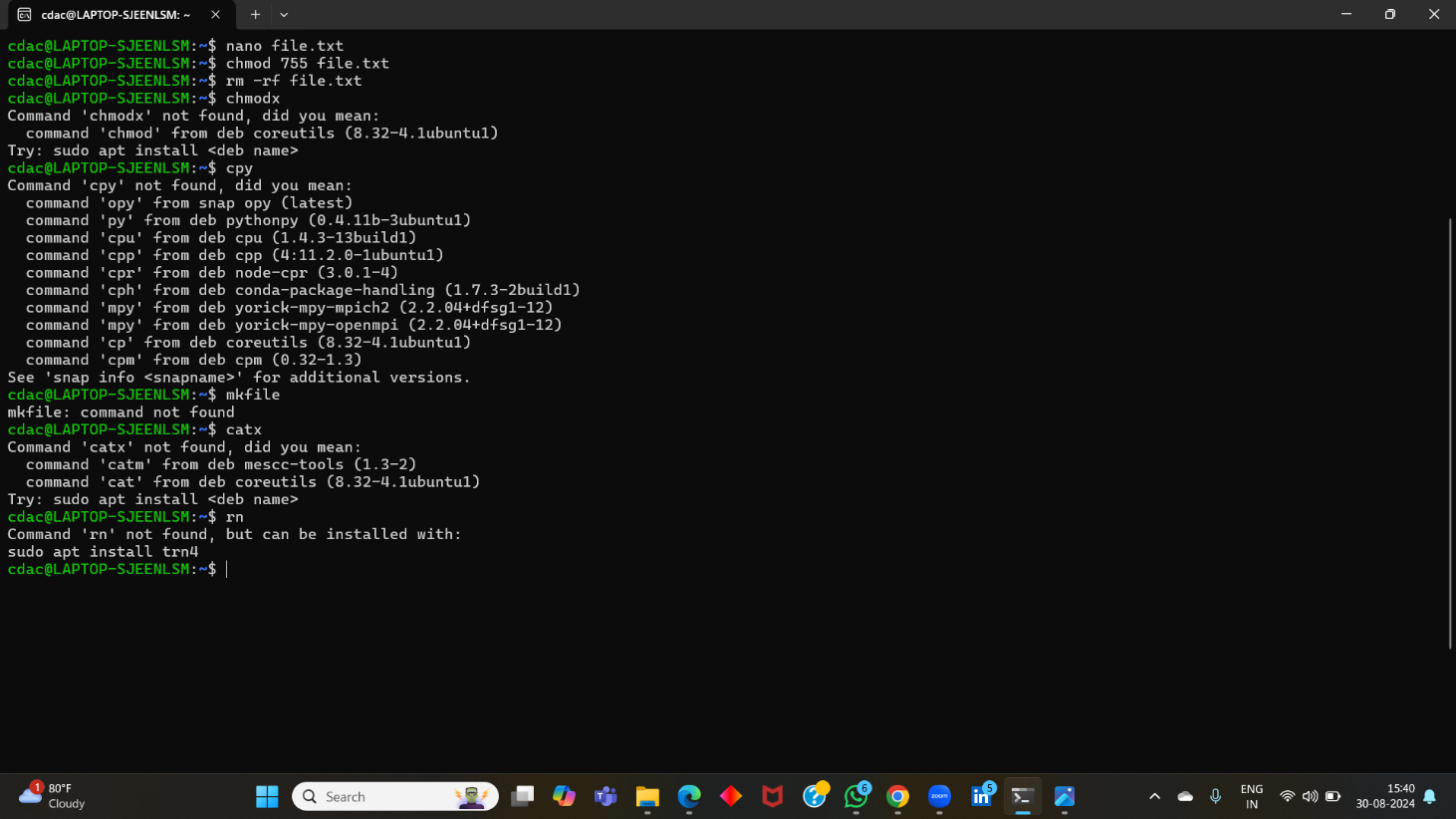
**if [[ -f file.txt ]]; then echo "File exists" else echo "File does not exist" fi**

**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.

**echo "Enter a number: " read num if [[ $num -gt 10 ]]; then echo "$num is greater than 10" else echo "$num is less than or equal to 10" fi**

**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.

**for ((i=1; i<=5; i++)); do for ((j=1; j<=5; j++)); do result=$((i \* j)) printf "%4d " "$result" done echo done**

****

**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.

**echo "Enter numbers (enter a negative number to quit): " while true; do read num if [[ $num -lt 0 ]]; then break fi square=$((num \* num)) echo "$num squared is: $square" done**

# Part D

## Common Interview Questions (Must know)

1. What is an operating system, and what are its primary functions?
2. Explain the difference between process and thread.
3. What is virtual memory, and how does it work?
4. Describe the difference between multiprogramming, multitasking, and multiprocessing.
5. What is a file system, and what are its components?
6. What is a deadlock, and how can it be prevented?
7. Explain the difference between a kernel and a shell.
8. What is CPU scheduling, and why is it important?
9. How does a system call work?
10. What is the purpose of device drivers in an operating system?
11. Explain the role of the page table in virtual memory management.
12. What is thrashing, and how can it be avoided?
13. Describe the concept of a semaphore and its use in synchronization.
14. How does an operating system handle process synchronization?
15. What is the purpose of an interrupt in operating systems?
16. Explain the concept of a file descriptor.
17. How does a system recover from a system crash?
18. Describe the difference between a monolithic kernel and a microkernel.
19. What is the difference between internal and external fragmentation?
20. How does an operating system manage I/O operations?
21. Explain the difference between preemptive and non-preemptive scheduling.
22. What is round-robin scheduling, and how does it work?
23. Describe the priority scheduling algorithm. How is priority assigned to processes?
24. What is the shortest job next (SJN) scheduling algorithm, and when is it used?
25. Explain the concept of multilevel queue scheduling.
26. What is a process control block (PCB), and what information does it contain?
27. Describe the process state diagram and the transitions between different process states.
28. How does a process communicate with another process in an operating system?
29. What is process synchronization, and why is it important?
30. Explain the concept of a zombie process and how it is created.
31. Describe the difference between internal fragmentation and external fragmentation.
32. What is demand paging, and how does it improve memory management efficiency?
33. Explain the role of the page table in virtual memory management.
34. How does a memory management unit (MMU) work?
35. What is thrashing, and how can it be avoided in virtual memory systems?
36. What is a system call, and how does it facilitate communication between user programs and the operating system?
37. Describe the difference between a monolithic kernel and a microkernel.
38. How does an operating system handle I/O operations?
39. Explain the concept of a race condition and how it can be prevented.
40. Describe the role of device drivers in an operating system.
41. What is a zombie process, and how does it occur? How can a zombie process be prevented?
42. Explain the concept of an orphan process. How does an operating system handle orphan processes?
43. What is the relationship between a parent process and a child process in the context of process management?
44. How does the fork() system call work in creating a new process in Unix-like operating systems?
45. Describe how a parent process can wait for a child process to finish execution.
46. What is the significance of the exit status of a child process in the wait() system call?
47. How can a parent process terminate a child process in Unix-like operating systems?
48. Explain the difference between a process group and a session in Unix-like operating systems.
49. Describe how the exec() family of functions is used to replace the current process image with a new one.
50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?
51. How does process termination occur in Unix-like operating systems?
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?
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?
54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.

# 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.

To calculate the average waiting time using FCFS scheduling, we need to first determine the waiting time for each process and then calculate the average.

**FCFS Scheduling:**

In FCFS scheduling, processes are executed in the order they arrive.

**Process Execution:**

1. **P1:** Arrives at time 0, starts executing immediately.
2. **P2:** Arrives at time 1, starts executing after P1 finishes.
3. **P3:** Arrives at time 2, starts executing after P2 finishes.

**Waiting Time Calculation:**

* **P1:** Waiting time = 0 (it starts immediately)
* **P2:** Waiting time = 5 (P1 finishes at time 5)
* **P3:** Waiting time = 8 (P2 finishes at time 8)

**Average Waiting Time:**

Average Waiting Time = (Waiting Time of P1 + Waiting Time of P2 + Waiting Time of P3) / Number of Processes

Average Waiting Time = (0 + 5 + 8) / 3 = 13 / 3 = 4.33

**Therefore, the average waiting time using FCFS scheduling is 4.33 units.**

1. 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.

To calculate the average waiting time using Priority Scheduling, we need to first determine the waiting time for each process and then calculate the average.

**Priority Scheduling:**

In Priority Scheduling, processes are executed based on their priority. Processes with higher priority are executed before processes with lower priority. If two processes have the same priority, they are executed in the order they arrive.

**Process Execution:**

1. **P2:** Arrives at time 1 with priority 1, starts executing immediately.
2. **P4:** Arrives at time 3 with priority 2, starts executing after P2 finishes.
3. **P1:** Arrives at time 0 with priority 3, starts executing after P4 finishes.
4. **P3:** Arrives at time 2 with priority 4, starts executing after P1 finishes.

**Waiting Time Calculation:**

* **P1:** Waiting time = 10 (P2 and P4 finish at time 5, P1 starts at time 5)
* **P2:** Waiting time = 0 (it starts immediately)
* **P3:** Waiting time = 11 (P2, P4, and P1 finish at time 5, P3 starts at time 5)
* **P4:** Waiting time = 1 (P2 finishes at time 5)

**Average Waiting Time:**

Average Waiting Time = (Waiting Time of P1 + Waiting Time of P2 + Waiting Time of P3 + Waiting Time of P4) / Number of Processes

Average Waiting Time = (10 + 0 + 11 + 1) / 4 = 22 / 4 = 5.5

**Therefore, the average waiting time using Priority Scheduling is 5.5 units.**



1. 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.

To calculate the average turnaround time using Round Robin scheduling, we need to first determine the turnaround time for each process and then calculate the average.

**Round Robin Scheduling:**

In Round Robin scheduling, each process is given a fixed time quantum to execute. If a process doesn't finish within the time quantum, it is preempted and placed at the end of the ready queue.

**Process Execution:**

**Time 0-2:** P1 executes (4 units remaining) **Time 2-4:** P2 executes (3 units remaining) **Time 4-6:** P3 executes (0 units remaining) **Time 6-8:** P4 executes (1 unit remaining) **Time 8-10:** P1 executes (3 units remaining) **Time 10-12:** P2 executes (2 units remaining) **Time 12-14:** P4 executes (0 units remaining) **Time 14-16:** P1 executes (0 units remaining) **Time 16-18:** P2 executes (0 units remaining)

**Turnaround Time Calculation:**

* **P1:** Turnaround time = 16 (finishes at time 16)
* **P2:** Turnaround time = 18 (finishes at time 18)
* **P3:** Turnaround time = 6 (finishes at time 6)
* **P4:** Turnaround time = 14 (finishes at time 14)

**Average Turnaround Time:**

Average Turnaround Time = (Turnaround Time of P1 + Turnaround Time of P2 + Turnaround Time of P3 + Turnaround Time of P4) / Number of Processes

Average Turnaround Time = (16 + 18 + 6 + 14) / 4 = 54 / 4 = 13.5

**Therefore, the average turnaround time using Round Robin scheduling is 13.5 units.**

1. 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?

**Initial value:** The parent process starts with x = 5.

**Forking:**

The fork() system call creates a child process that is an exact copy of the parent process at the time of the call. This means that both the parent and child processes will initially have x = 5.

**Incrementing x:**

After the fork(), both the parent and child processes increment the value of x by 1. This means that x will become 6 in both processes.

Since each process has its own copy of the variable x, the changes made by one process do not affect the other process. Therefore, both the parent and child processes will have the same final value of x, which is 6.