

Name: Anish Rane

## Part A

What will the following commands do?

1. `echo "Hello, World!"`

Ans: This command prints Hello, World!"

2. `name="Productive"`

Ans: In name variable 'Productive' value is stored

3. `touch file.txt`

Ans: Create a file.txt file

4. `ls -a`

Ans: List all the files and directory

5. `rm file.txt`

Ans: remove file.txt

6. `cp file1.txt file2.txt`

Ans: copy content of file1.txt to file2.txt

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

Ans: move the file.txt to this directory

8. `chmod 755 script.sh`

Ans: giving access rwx to owner , rx to group and other

9. `grep "pattern" file.txt`

Ans: display line containing word "pattern" in file.txt

10. `kill PID`

Ans: This would attempt to terminate the process with the PID

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

Ans: make a mydir directory and then go mydir also create a file.txt file and save echo

"Hello, World!" in file.txt and display the data of file.txt

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

Ans: will list all files in the current directory, including their detailed information

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

Ans: Display Merges file1.txt and file2.txt, sorts the lines, and removes any duplicate lines, displaying the unique sorted content.

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

Ans: Listing all .txt files in the current directory, showing detailed information about them

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

Ans: Display line containing pattern in this file directory

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

Ans: merge file.txt and file.txt sort it and only display uniq values

17. `chmod 644 file.txt`

Ans: giving rw access to owner ,r access to group and r access to other user

18. `cp -r source_directory destination_directory`

Ans: Copy the directories recursively from source directory to destination directory

19. `find /path/to/search -name "*.txt"`

Ans: Find files containing .txt

20. `chmod u+x file.txt`

Ans: changing owner access to execute

21. `echo $PATH`

Ans: print value stored in path

## Part B

Identify True or False:

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

Ans: True

2. mv is used to move files and directories.

Ans: True

3. cd is used to copy files and directories.

Ans: False

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

Ans: False

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

Ans: True

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

Ans: True

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

Ans: True

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

Ans: False

Identify the Incorrect Commands:

1. chmodx is used to change file permissions.

Ans: chmod is command x is added extra

2. cpy is used to copy files and directories.

Ans: cp is the command

3. mkfile is used to create a new file.

Ans: touch is used to create new file

4. catx is used to concatenate files.

Ans: cat is used to concatenate files

5. rn is used to rename files.

Ans: mv is used to rename

## Part C

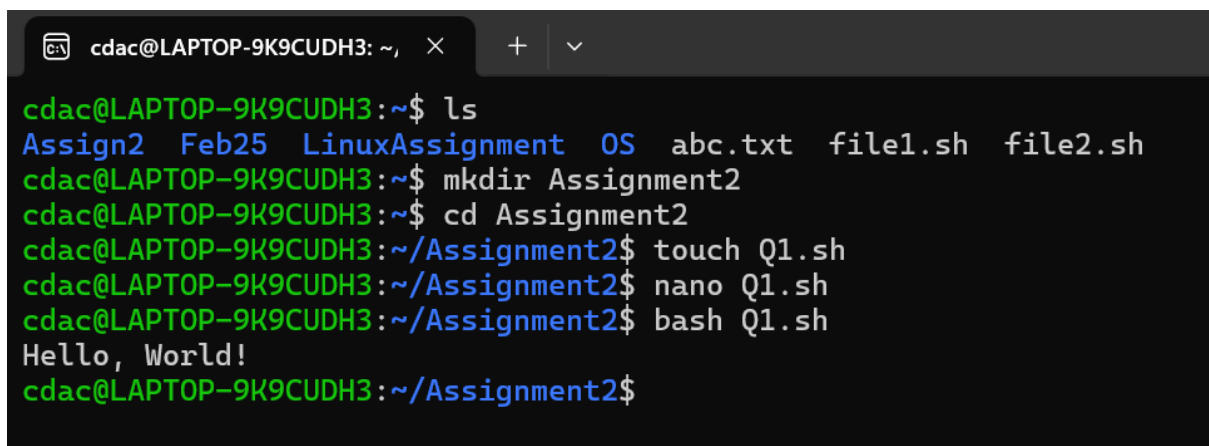
Question 1: Write a shell script that prints "Hello, World!" to the terminal.

Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input: `echo "Hello, World!" // print Hello, World!`

Output:

A terminal window with a dark background and light-colored text. The prompt is 'cdac@LAPTOP-9K9CUDH3: ~'. The user enters 'ls', showing a directory listing including 'Assign2', 'Feb25', 'LinuxAssignment', 'OS', 'abc.txt', 'file1.sh', and 'file2.sh'. Then, the user enters 'mkdir Assignment2', 'cd Assignment2', 'touch Q1.sh', 'nano Q1.sh', and 'bash Q1.sh'. The output of the script is 'Hello, World!'. The prompt then changes to 'cdac@LAPTOP-9K9CUDH3: ~/Assignment2\$'.

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

Steps:

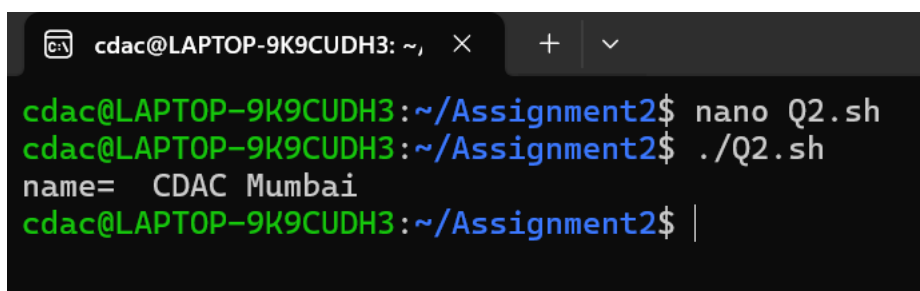
- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

`name= "CDAC Mumbai" // assigning value cdac Mumbai to name`

`echo "name = $name" // Print name and its value`

Output:

A terminal window with a dark background and light-colored text. The prompt is 'cdac@LAPTOP-9K9CUDH3: ~/Assignment2\$'. The user enters 'nano Q2.sh', then './Q2.sh'. The output of the script is 'name= CDAC Mumbai'. The prompt then changes to 'cdac@LAPTOP-9K9CUDH3: ~/Assignment2\$'.

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

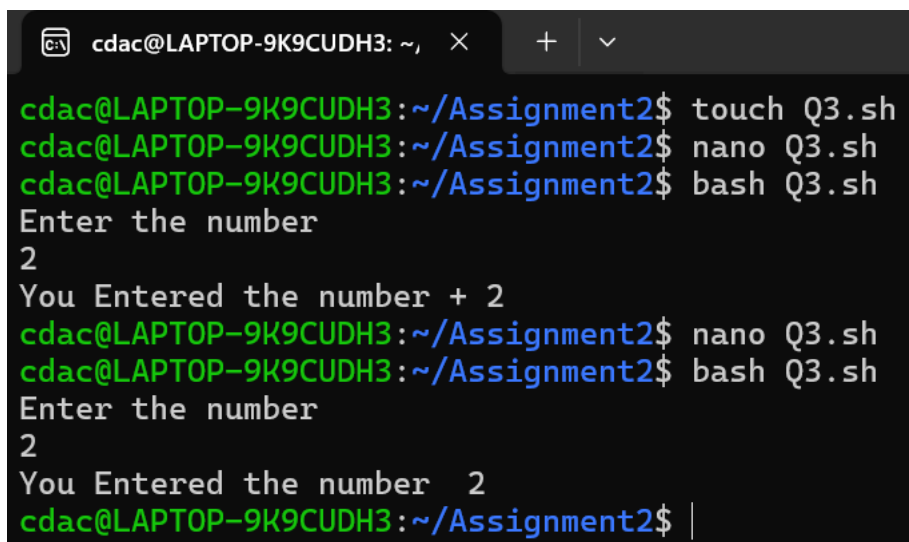
Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
echo "Enter the number"           // Asking user to enter a number
read n                           // assigning user value to n
echo "You entered number" n       // printing the value user has given
```

Output:



```
cdac@LAPTOP-9K9CUDH3: ~/Assignment2$ touch Q3.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nano Q3.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ bash Q3.sh
Enter the number
2
You Entered the number + 2
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nano Q3.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ bash Q3.sh
Enter the number
2
You Entered the number 2
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ |
```

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

Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
a=5                               //assigning 5 to a
b=3                               // assigning 3 to b
sum=$((a + b))                   // assigning addition of a+b value to sum
echo "5+3=" "$sum"               // printing the result
```

Output:

```
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nano Q4.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ ./Q4.sh
5+3= 8
```

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

Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
echo "Enter a number"           //Asking user to enter the number
read num                       // assigning user value to num
if ((num%2 == 0))               //assigning condition to check if it is even or odd
then
echo "$num is even number"      // if num%2 is equal to 0 then it is even
else
echo "$num is odd number"       // if num%2 is equal to 0 then it is odd
fi                               // end of if condition
```

Output:

```
cdac@LAPTOP-9K9CUDH3:~$ cd Assignment2/
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ touch Q5.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nanao Q5.sh
Command 'nanao' not found, did you mean:
  command 'nano' from snap nano (7.2+pkg-4057)
  command 'nano' from deb nano (7.2-2ubuntu0.1)
See 'snap info <snapname>' for additional versions.
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nano Q5.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ bash Q5.sh
Enter a number
20
20 is even number
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ |
```

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

Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

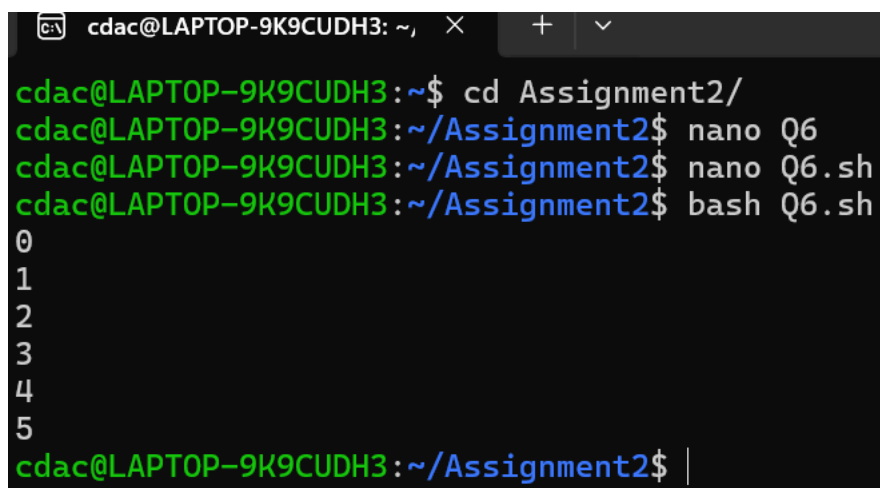
```
for((a=0;a<6; a++))           //assigning condition
```

```
do
```

```
echo $a                       //printing a value
```

```
done
```

Output:

A terminal window screenshot showing a user named 'cdac' on a machine 'LAPTOP-9K9CUDH3'. The user navigates to a directory 'Assignment2' and creates a file 'Q6' using 'nano'. Then, they create a shell script 'Q6.sh' using 'nano' and execute it using 'bash Q6.sh'. The output of the script is a list of numbers from 0 to 5, each on a new line.

```
cdac@LAPTOP-9K9CUDH3: ~$ cd Assignment2/  
cdac@LAPTOP-9K9CUDH3: ~/Assignment2$ nano Q6  
cdac@LAPTOP-9K9CUDH3: ~/Assignment2$ nano Q6.sh  
cdac@LAPTOP-9K9CUDH3: ~/Assignment2$ bash Q6.sh  
0  
1  
2  
3  
4  
5  
cdac@LAPTOP-9K9CUDH3: ~/Assignment2$ |
```

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

Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
a=1                           //assigning 1 to a
```

```
while [ $a -le 5 ]           // applying condition of a<5
```

```
do
```

```
echo $a                      // print a
```

```
a=$((a + 1))                 // a increment
```

```
done                //end of loop
```

Output:

```
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nano Q7.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ bash Q7.sh
1
2
3
4
5
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ |
```

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".

Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
if [ -e "file.txt" ]; then          //applying condition of searching file.txt file
    echo "File exists"              // if found print file exist
else
    echo "File does not exist"      // not found print file does not exist
fi                                  // end of loop
```

Output:

```
cdac@LAPTOP-9K9CUDH3: ~/ × + v
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ touch Q8.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nano Q8.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ bash Q8.sh
File does not exist
cdac@LAPTOP-9K9CUDH3:~/Assignment2$
```

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.

Steps:

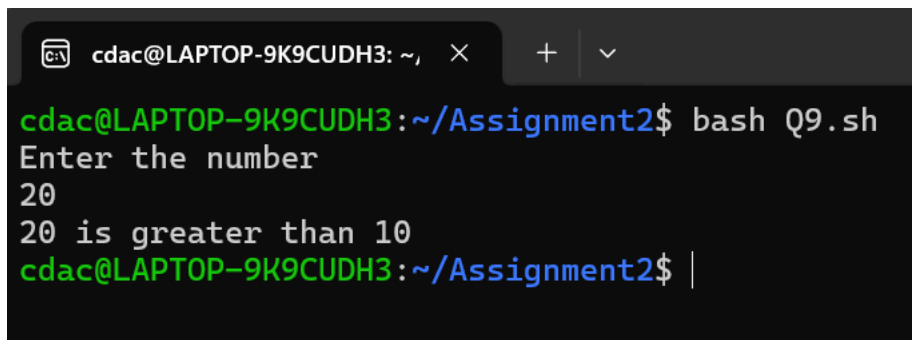


- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
echo "Enter the number"           //asking user a number
read num                         // assigning user input to num
if(($num > 10))                   //applying if and giving condtion num>10
then
echo $num "is greater than 10"    // printing result
elif(($num == 10))               // checking if num value is 10
then
echo $num "=10"                  // if yes printing num=10
else
echo $num "is less than 10"      //id none of both then print this
fi                                // end of loop
```

Output:



```
cdac@LAPTOP-9K9CUDH3: ~/Assignment2$ bash Q9.sh
Enter the number
20
20 is greater than 10
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ |
```

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.

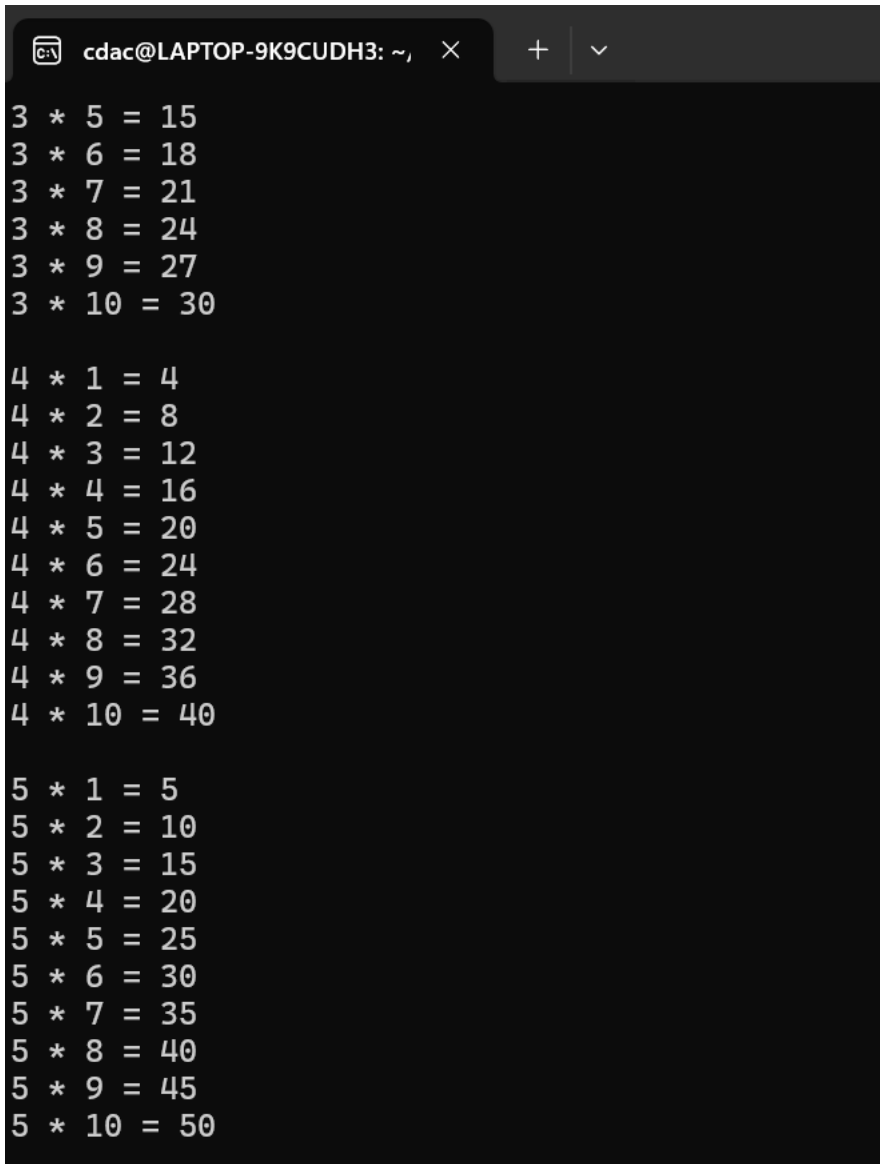
Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
for i in {1..5}          // printing table 1 to 5
do
    for j in {1..10}      // printing table upto 10
    do
        echo $i "*" $j "=" $((i * j)) // multiplying and printing result
    done                  // end of inner loop
done                      // printing blank space
echo                     // printing blank space
done                     // end of outer loop
```

Output:



```
cdac@LAPTOP-9K9CUDH3: ~, X + v
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30

4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40

5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

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.

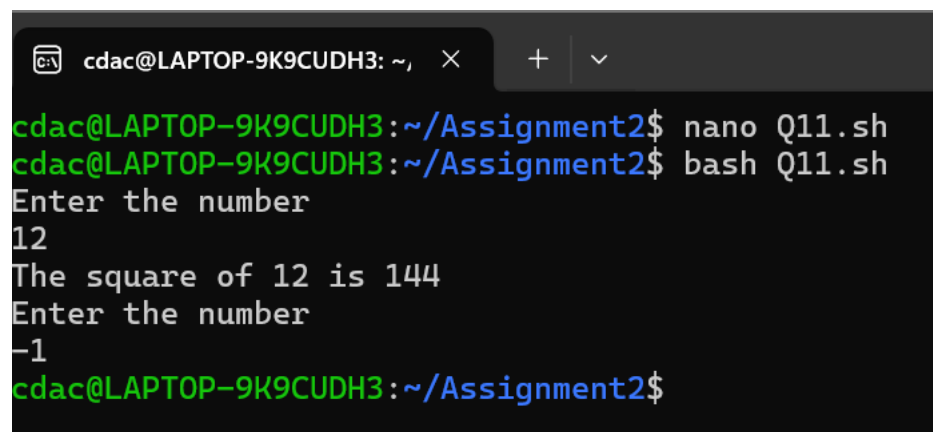
Steps:

- create a new .sh file
- enter the input and save the file
- execute the .sh file by using bash command or ./ (shell)command

Input:

```
num=0                                // assigning num value 0
while (( num >= 0 ))                  //go to loop until num is positive
do                                   //start of loop
    echo "Enter the number"          // asking user a number
    read num                         // storing value in num
    if (( num >= 0 ))                 //checking if num is positive
    then                             // if yes
        s=$(( num * num ))           // squaring num and storing it in variable s
        echo "The square of $num is $s" // printing square of number
    else                             // if num is negative
        break                        // break the while loop
    fi                               // end of if condition
done
```

Output:



```
cdac@LAPTOP-9K9CUDH3: ~, × + ∨
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ nano Q11.sh
cdac@LAPTOP-9K9CUDH3:~/Assignment2$ bash Q11.sh
Enter the number
12
The square of 12 is 144
Enter the number
-1
cdac@LAPTOP-9K9CUDH3:~/Assignment2$
```

## Part D

### Common Interview Questions (Must know)

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

Ans: An operating system is a program that acts as an interface between the user and the computer. Following is some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users
- hardware and controls the execution of all kinds of programs.

2. Explain the difference between process and thread.

Ans:

Process	Thread
Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

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

Ans: A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Virtual memory allows an OS to use both physical memory (RAM) and disk space to run programs that require more memory than

what is available. It creates an illusion of a large, continuous block of memory by managing and mapping pages between RAM and disk, ensuring efficient multitasking and larger application handling.

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

Ans:

multiprogramming	multitasking	multiprocessing
Running multiple programs on a single CPU.	Running multiple tasks (processes) simultaneously.	Using multiple CPUs or cores to execute processes simultaneously.
Goal is to maximize CPU utilization by keeping it busy with multiple jobs	Goal is to Improve user experience by allowing multiple tasks to operate concurrently.	Goal is to Enhance performance by parallel processing.
Example: Running a text editor and a compiler on the same machine.	Example: Browsing the web, listening to music, and editing a document at the same time.	Example: Modern servers and high-performance computing systems with multiple processors.

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

Ans: A file system is a method used by an operating system (OS) to control how data is stored and retrieved on a storage device, such as a hard drive or SSD. It ensures that files are organized, accessible, and managed efficiently. Here are the main components of a file system:

- Files: Units of data storage containing user or system data.
- Directories/Folders: Organizational structures that store and group files.
- File Descriptors: Data structures containing information about open files, such as their location and access mode.
- Superblock: A metadata structure containing information about the file system's layout and properties.
- Inodes: Data structures that store information about files and directories, including attributes like file size, permissions, and timestamps.
- File Allocation Table (FAT): A table that maps file fragments to physical storage locations, used in some file systems like FAT32.
- Journaling: A feature that logs changes to the file system to help recover from crashes and ensure data integrity.
- Metadata: Information about files and directories, such as their names, sizes, permissions, and creation dates.
- Blocks: Fixed-size units of storage used to store file data on the disk.

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

Ans: A deadlock is a situation in which two or more processes are unable to proceed because each is waiting for the other to release resources. This can cause the processes to remain indefinitely blocked. Deadlocks typically occur in systems that allow concurrent processes and resource sharing. Ways to Prevent Deadlocks:

- Avoid Mutual Exclusion: Ensure that some resources can be shared among processes, reducing the risk of deadlock.
- Hold and Wait Prevention: Require processes to request all needed resources at once, ensuring they do not hold onto resources while waiting for others.
- No Preemption: If a process holding resources is denied further resources, it must release its current resources and request them again.
- Circular Wait Prevention: Impose an ordering on resource types and ensure that each process requests resources in a predetermined order.

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

Ans: The kernel is the core component of an operating system, responsible for managing system resources and hardware at a low level. It handles tasks like memory management, process management, and device management. On the other hand, the shell is the user interface that allows users to interact with the operating system. It interprets and executes user commands, operating at a higher level to provide a user-friendly environment. In essence, the kernel is like the engine of the OS, while the shell is the interface that lets users control the OS.

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

Ans: CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair. Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

9. How does a system call work?

Ans: A system call is a mechanism that allows user programs to interact with the operating system. When a program makes a system call, it switches from user mode to kernel mode, allowing it to request services like file I/O, memory allocation, or process management from the OS.

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

Ans: Device drivers are specialized programs that allow the operating system to communicate with hardware devices. They serve as intermediaries, translating OS commands into device-specific instructions and vice versa.

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

Ans: The page table maps virtual addresses to physical addresses. It keeps track of where each page of a program's memory is located in physical memory or on disk, enabling efficient memory management and address translation.

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

Ans: Thrashing occurs when a system spends more time swapping pages in and out of memory than executing tasks. It can be avoided by using proper page replacement algorithms, increasing physical memory, or adjusting the system's multiprogramming level.

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

Ans: Semaphores are synchronization primitives used to manage access to shared resources in a concurrent system. They can be used to signal and wait for conditions, preventing race conditions and ensuring proper coordination between processes.

14. How does an operating system handle process synchronization?

Ans: The operating system uses various mechanisms like semaphores, mutexes, and monitors to handle process synchronization. These tools ensure that multiple processes can access shared resources without interfering with each other.

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

Ans: Interrupts are signals that notify the operating system of events requiring immediate attention, such as hardware failures or user inputs. They allow the OS to respond quickly to critical events.

16. Explain the concept of a file descriptor.

Ans: A file descriptor is a unique identifier assigned to an open file within a process. It is used to perform operations like reading, writing, and closing the file.

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

Ans: System recovery involves restoring the system to a stable state after a crash. This may include restarting the system, restoring files from backups, and using journaling mechanisms to recover lost data.

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

Ans: Monolithic Kernel: All OS components run in a single address space, providing better performance but less modularity.

Microkernel: The core functions run in kernel space, while other services run in user space, offering better modularity and stability.

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

Ans:

- External fragmentation: Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
- Internal fragmentation: Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

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

Ans: The OS manages I/O operations using buffering, caching, spooling, and device drivers to ensure efficient and coordinated access to hardware devices.

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

Ans:

- Preemptive: The OS can interrupt and switch between processes, ensuring fair CPU allocation.
- Non-preemptive: Processes run until completion or voluntarily yield the CPU.

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

Ans: A scheduling algorithm that assigns a fixed time slice (quantum) to each process, cycling through them to ensure fair CPU allocation.

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

Ans: Processes are assigned priorities, and the CPU is allocated to the highest-priority process. Priority can be based on factors like process importance, resource requirements, or aging.

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

Ans: Selects the process with the shortest execution time. It is used to minimize average waiting time but can lead to starvation of longer processes.

25. Explain the concept of multilevel queue scheduling.

Ans: Processes are divided into different queues based on priority or type. Each queue has its scheduling algorithm, allowing for tailored management of various process types.



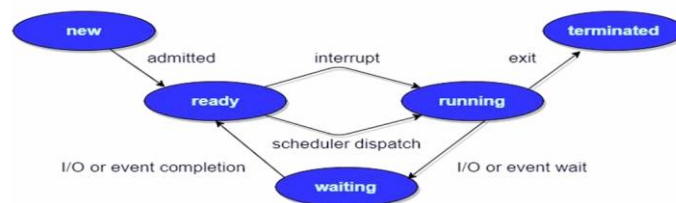
26. What is a process control block (PCB), and what information does it contain?

Ans: A data structure containing information about a process, such as its ID, state, priority, program counter, registers, memory limits, and I/O status.

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

Ans: Processes in the operating system can be in any of the following states:

- NEW- The process is being created.
- READY- The process is waiting to be assigned to a processor.
- RUNNING- Instructions are being executed.
- WAITING- The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- TERMINATED- The process has finished execution.



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

Ans: Mechanisms like pipes, message queues, shared memory, and sockets allow processes to communicate and synchronize their actions.

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

Ans: Ensures that processes sharing resources do not interfere with each other, preventing race conditions and ensuring data consistency.

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

Ans: A process that has completed execution but still has an entry in the process table, awaiting the parent process to read its exit status. It is created when a child process terminates, but the parent has not yet called wait().

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

Ans:

- External fragmentation: Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
- Internal fragmentation: Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

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

Ans: Loads pages into memory only when they are needed, reducing memory usage and improving efficiency. It helps manage memory dynamically.

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

Ans: Maps virtual addresses to physical addresses, tracking page locations in memory or on disk.

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

Ans: Hardware component that handles virtual-to-physical address translation, enabling virtual memory management.

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

Ans: Occurs when excessive paging happens. Avoided by proper page replacement algorithms, increasing physical memory, or adjusting multiprogramming levels.

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

Ans: Allows user programs to request services from the OS, facilitating communication and resource access

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

Ans:

monolithic kernel	Micro kernel
All OS components (e.g., file system, device drivers, memory management) run in a single address space.	Only essential core functions run in kernel space (e.g., basic memory management, IPC), while other services run in user space.
Faster due to direct communication between components.	Slightly slower due to communication overhead between kernel and user space.
Less modular and harder to maintain or extend.	More modular, easier to maintain and extend.

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

Ans: An operating system handles I/O (Input/Output) operations by using a combination of software and hardware mechanisms to ensure efficient and coordinated access to peripheral devices. Here are the key steps involved:

- Device Drivers: Specialized software that translates OS commands into device-specific instructions, allowing the OS to communicate with hardware devices.

- **I/O Scheduling:** The OS schedules I/O requests to optimize performance and ensure fair access to devices. It may use algorithms like FCFS (First-Come, First-Served), SSTF (Shortest Seek Time First), or SCAN.
- **Buffering:** Temporary storage areas, or buffers, are used to hold data during I/O operations. Buffering helps smooth out differences in data transfer rates between devices and the CPU.
- **Caching:** Frequently accessed data is stored in a cache to reduce I/O latency and improve performance.
- **Spooling:** The OS queues I/O requests, particularly for devices like printers, to manage them more efficiently and ensure sequential processing.
- **Interrupts:** Hardware signals, called interrupts, notify the OS when an I/O device requires attention, allowing the OS to respond promptly and handle I/O operations efficiently.

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

Ans: Occurs when multiple processes access shared resources concurrently, leading to unpredictable results. Prevented using synchronization mechanisms like mutexes and semaphores.

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

Ans: Allow the OS to communicate with hardware devices, translating commands between the OS and devices.

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

Ans: A zombie process is a process that has completed its execution but still has an entry in the process table. It occurs when a child process terminates, but its parent process has not yet read its exit status using the `wait()` or `waitpid()` system calls.

Prevention:

- **Parent Process Handling:** Ensure that the parent process properly calls `wait()` or `waitpid()` to read the child's exit status and remove its entry from the process table
- **Reaping Orphan Processes:** If the parent process terminates before the child, the init process (PID 1) adopts the orphaned child and handles its termination, preventing zombie processes.

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

Ans: A child process whose parent process has terminated. The OS reassigns the orphan process to the init process, ensuring it continues to execute.

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

Ans: The parent process creates the child process, sharing resources and maintaining control over it. The child process can execute independently but reports back to the parent.

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

Ans: Creates a new process by duplicating the existing process. The new process (child) runs concurrently with the original process (parent).

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

Ans: The parent process can use the wait() or waitpid() system calls to wait for the child process to complete, retrieving its exit status.

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

Ans: Indicates the reason for the child process's termination, allowing the parent process to handle it appropriately.

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

Ans: The parent process can use the kill() system call to terminate a child process, specifying the process ID and signal to be sent.

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

Ans: A process group is a collection of related processes, while a session is a collection of process groups. Sessions provide a higher-level organization for managing processes.

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

Ans: Replaces the current process image with a new one, allowing a process to execute a different program.

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

Ans: Allows the parent process to wait for a specific child process to terminate, providing more control over process management compared to wait().

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

Ans: Occurs when a process completes execution or is explicitly terminated. The OS cleans up resources and removes the process from the process table.

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?

Ans: Decides which processes to admit to the system, influencing the degree of multiprogramming and overall system performance. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the 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?

Ans: Selects which process to execute next from the ready queue. Executes more frequently than the long-term scheduler, making quick decisions.

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

Ans: When the system becomes overloaded with active processes, the medium-term scheduler might be invoked to temporarily swap out some processes from RAM to disk, freeing up memory. By doing this, it ensures that the system can continue to run smoothly without thrashing and can better manage the available resources for active processes

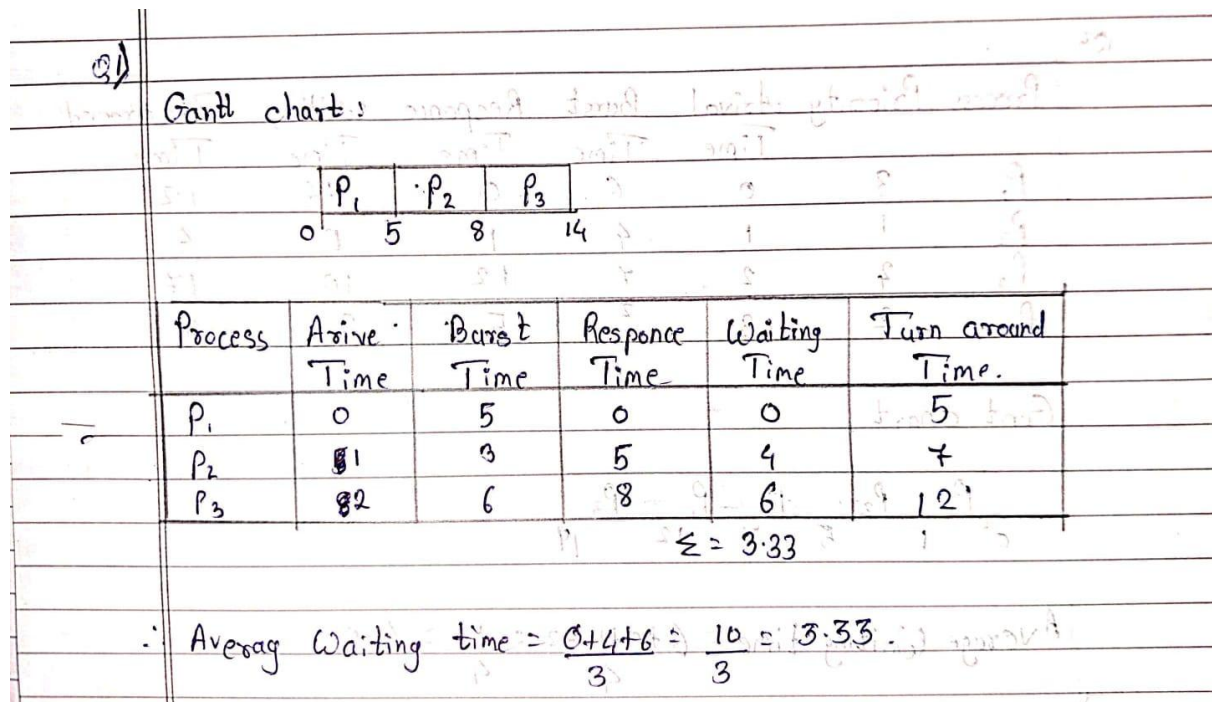
#### Part E

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.

Ans:



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

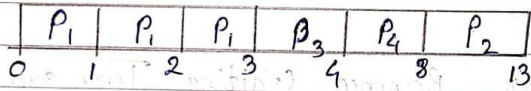
Process	Arrival Time	Burst Time
P1	0	3
P2	1	5
P3	2	1
P4	3	4

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

Ans:

Q2)

Gantt Chart



Process	Arrive Time	Burst Time	Response Time	Waiting Time	Turn around Time
P <sub>1</sub>	0	3	0	0	3
P <sub>2</sub>	1	5	8	4	9
P <sub>3</sub>	2	1	3	3	4
P <sub>4</sub>	3	4	4	8	12

Turn around

$$\text{Average waiting time} = \frac{0+4+3+8}{4} = \frac{3+9+4+12}{4} = \frac{28}{4} = 7$$

ANSWER

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.

Ans:

Q4							
Process	Priority	Arrival Time	Burst Time	Response Time	Waiting Time	Turn around Time	
P <sub>1</sub>	3	0	6	0	6	12	
P <sub>2</sub>	1	1	4	1	0	4	
P <sub>3</sub>	2	2	2	12	10	17	
P <sub>4</sub>	2	3	2	5	2	4	
Gantt chart.							
<pre> 0   P1   P2   P4   P1   P3      --- --- --- --- ---  0   1   5   7   12   19           </pre>							
Average Waiting time = $\frac{6+0+10+2}{4} = \frac{18}{4} = 4.5$							

4. Consider the following processes with arrival times and burst times, and the time quantum for Round Robin scheduling is 2 units:

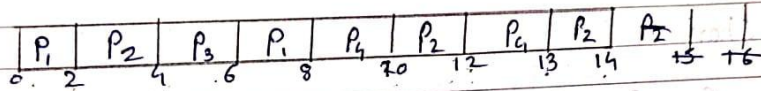
Process	Arrival	Burst
P1	0	4
P2	1	5
P3	2	2
P4	3	3

Calculate the average turnaround time using Round Robin scheduling.

Ans:



### Q5) Gantt Chart



Process	Arrival Time	Burst Time	Response Time	Waiting Time	Turn around Time
P <sub>1</sub>	0	4	0	4	8
P <sub>2</sub>	1	5	2	8	13
P <sub>3</sub>	2	2	4	2	4
P <sub>4</sub>	3	3	5	8	11

$$\text{Average Turn around time} = \frac{8 + 13 + 4 + 11}{4} = \frac{36}{4} = 9$$

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?

Ans:

#### 1. Before the fork() call:

- The parent process has a variable x with a value of 5.

#### 2. After the fork() call:

- The parent process and the child process both have a variable x with an initial value of 5.

#### 3. Incrementing the value of x by 1 in both processes:

- In the parent process, x becomes 6.
- In the child process, x also becomes 6.

Therefore, the final value of x in both the parent and child processes after the fork() call and the increment operations will be 6.