Name: 117_Swapnil Dhotre_KH

Concepts of Operating Systems

Assignment 2

Part A:

1) What will the following commands do?

a) echo "Hello, World!"

This is a command used to display a line of text or a string that is Hello World.

b) name="Productive"

The command name="Productive" assigns the string value "Productive" to the variable name.

c) touch file.txt

If file.txt does not already exist in the current directory, touch will create an empty file named file.txt. If file.txt already exists, touch will update the file's last modified timestamp to the current date and time.

d) ls -a

This Command is used to lists all files and directories in the current directory, including hidden files.

e) rm file.txt

The command rm file.txt is used to delete a file named file.txt from the current directory

f) cp file1.txt file2.txt

This command is used to is copy files in file1.txt to file2.txt.

g) mv file.txt /path/to/directory

This command is used to is move or rename files and directories in file1.txt to the another directory.

h) chmod 755 script.sh:

The command chmod 755 script.sh is used to change the permissions of a file.

7 (rwx) for the owner: The owner of the file can read, write, and execute the fie.

5 (r-x) for the group: Members of the file's group can read and execute the file, but cannot write to it.

5 (r-x) for others: All other users can read and execute the file, but cannot write to it.

script.sh**:** This is the file whose permissions we are changing.

i)grep "pattern" file.txt:

The command grep "pattern" file.txt is used to search for a specific text pattern within a file.txt

j) kill PID :

The Kill PID command used to send signals to processes ID or PID

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

The command mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt is a series of commands connected by &&.

mkdir mydir: Creates a new directory named mydir in the current location.

cd mydir: Changes the current working directory to mydir, so all subsequent commands will operate within this directory.

touch file.txt: Creates an empty file named file.txt in the mydir directory.

echo "Hello, World!" > file.txt: Writes the text "Hello, World!" into file.txt. The > operator overwrites any existing content in file.txt with this new text.

cat file.txt: Displays the contents of file.txt in the terminal. Since you just wrote "Hello, World!" to the file, this text will be shown as the output.

l) ls -l | grep ".txt"

The command ls -l | grep ".txt" is a combination of two commands connected by a pipe (|). The ls command lists files and directories in the current directory. The -l flag provides a detailed (long) listing format, showing additional information such as file permissions, number of links, owner, group, file size, and the last modified date and time, along with the file name. The pipe (|) takes the output of the command on the left (ls -l) and passes it as input to the command on the right (grep ".txt").The grep command searches for lines that match a specific pattern. The pattern ".txt" is used to filter the output, so only lines that contain .txt are displayed.

m)cat file1.txt file2.txt | sort | uniq

The command sequence cat file1.txt file2.txt | sort | uniq performs several operations on the contents of two files, file1.txt and file2.txt.

cat file1.txt file2.txt: The cat command concatenates and displays the contents of file1.txt and file2.txt. It outputs the combined contents of both files to the terminal.

The pipe (|) passes the output of the cat command as input to the next command (sort). The sort command sorts the combined lines from file1.txt and file2.txt in alphabetical order.

The uniq command filters out any duplicate lines that are adjacent in the sorted output.

n) ls -l | grep "^d"

ls: Lists the contents of a directory.

-l: Uses the long listing format, which provides detailed information about each item, including: File type and permissions, Number of links Owner name, Group name, file size, Modification date and time filename.|(pipe): The pipe takes the output of ls -l command and passes it as input to the next command. grep "^d": grep: Searches for patterns within text.

"^d": The ^ character signifies the start of a line. So, ^d matches any line that begins with the letter "d".

In the context of ls -l, lines that start with "d" represent directories.

o) grep -r "pattern" /path/to/directory/

grep: This is a command-line utility for searching plain-text data for lines that match a specified pattern.

-r (or --recursive): This option tells grep to search recursively through all files in the specified directory and its subdirectories.

"pattern": This is the specific text string or regular expression you're searching for within the files.

/path/to/directory/: This specifies the directory where grep should start searching. It will search through this directory and all its subdirectories.

p) cat file1.txt file2.txt | sort | uniq –d

The cat command concatenates (cat stands for "concatenate") and displays the contents of file1.txt and file2.txt together as a single stream of text.

| (Pipe): The pipe (|) takes the output of the command on the left (cat file1.txt file2.txt) and feeds it as input to the command on the right (sort).

sort: The sort command sorts the combined lines from file1.txt and file2.txt in alphabetical order. Sorting is necessary because uniq only works on adjacent lines.

uniq -d: The uniq command removes duplicate lines, but with the -d option, it only prints lines that are duplicated, meaning it will show lines that appear more than once in the sorted output. Since the files were concatenated before sorting, uniq -d will identify lines that are present in both file1.txt and file2.txt.

q)chmod 644 file.txt

The command chmod 644 file.txt is used to change the permissions of the file file.txt.

chmod: Stands for "change mode" and is used to change the file permissions.

644: This is an octal (base-8) representation of the file permissions.

Permissions in Unix are divided into three categories:

1. Owner (User): The person who owns the file.
2. Group: The group that the file belongs to.
3. Others (World): Everyone else.

The numbers 644 are interpreted as follows:

- 6 (Owner): Read and write permissions (4 + 2).
- 4 (Group): Read-only permission.
- 4 (Others): Read-only permission.

Permission Details:

- 6 means 4 (read) + 2 (write) = 6, so the owner can read and write the file.

- 4 means 4 (read) = 4, so the group can only read the file.
- 4 means 4 (read) = 4, so others can only read the file.

r) cp -r source_directory destination_directory

cp: The cp command is used for copying files and directories.

-r (or -R): This option stands for "recursive" and tells the cp command to copy directories recursively. This means it will copy the specified directory and all of its contents, including any subdirectories and the files they contain.

source_directory: This is the directory you want to copy.

destination_directory: This is the location where you want to copy the source_directory. If destination_directory does not exist, it will be created. If it exists, the source_directory will be copied inside it.

s) find /path/to/search -name "*.txt"

find: This command searches for files and directories based on various criteria within a specified location.

/path/to/search: This specifies the directory where the search should begin. The find command will recursively search through this directory and all of its subdirectories.

-name "*.txt": The -name option tells find to look for files with a name that matches the pattern "*.txt".

"*.txt" is the pattern where * is a wildcard that matches any number of characters, meaning it will find any file that ends with .txt.

t) chmod u+x file.txt

chmod: Stands for "change mode" and is used to change the permissions of a file or directory.

u: Refers to the user (owner) of the file.

+x: Adds the execute (x) permission to the file for the user. The execute permission allows the user to run the file as a program or script.

file.txt: The name of the file whose permissions you want to modify.

u) echo $PATH

The command echo $PATH is used to display the current value of the PATH environment variable in a Unix-like operating system.

echo: This command prints text or the value of a variable to the terminal.

$PATH: $PATH is an environment variable that contains a list of directories separated by colons (:). The $ symbol is used to reference the value of the variable PATH

Part B

Identify True or False:

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

True. The ls command lists files and directories in the current directory

2. mv is used to move files and directories.

True. The mv command is used to move files and directories

3. cd is used to copy files and directories.

False. The cd command is used to change directories

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

True. The pwd command stands for "print working directory" and displays the full path of the current working directory.

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

True. The grep command is used to search for specific patterns

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

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

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

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

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

True: rm -rf file.txt deletes a file forcefully without confirmation.

Identify the Incorrect Commands:

1. chmodx is used to change file permissions.
   Incorrect, The correct command to change file permissions is chmod, not chmodx. The chmod command modifies the permissions of a file or directory.

2. cpy is used to copy files and directories.
   Incorrect, The correct command to copy files and directories is cp, not cpy. The cp command is used for copying files and directories

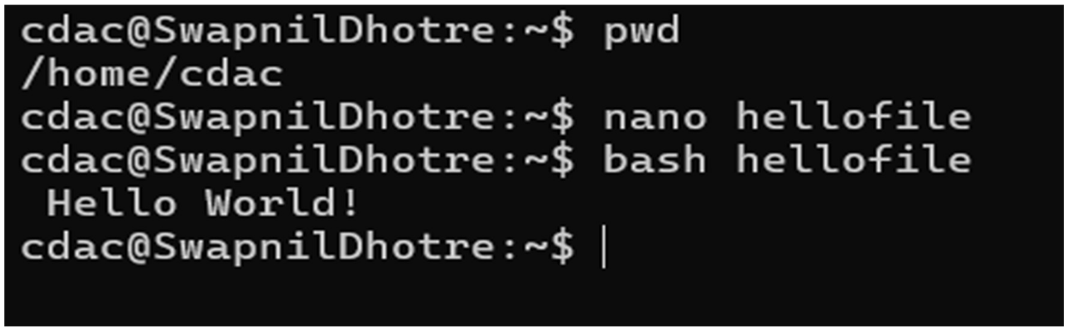
3. mkfile is used to create a new file.
   Incorrect, The mkfile command is not used to create a new file. The correct way to create a new file is generally to use touch or simply redirect output to a file

4. catx is used to concatenate files
   Incorrect, The correct command to concatenate files is cat, not catx. The cat command reads and concatenates file contents.

5. rn is used to rename files.
   Incorrect, Correct command to rename file is mv and mv command moves files and directories, and can also be used to rename them.

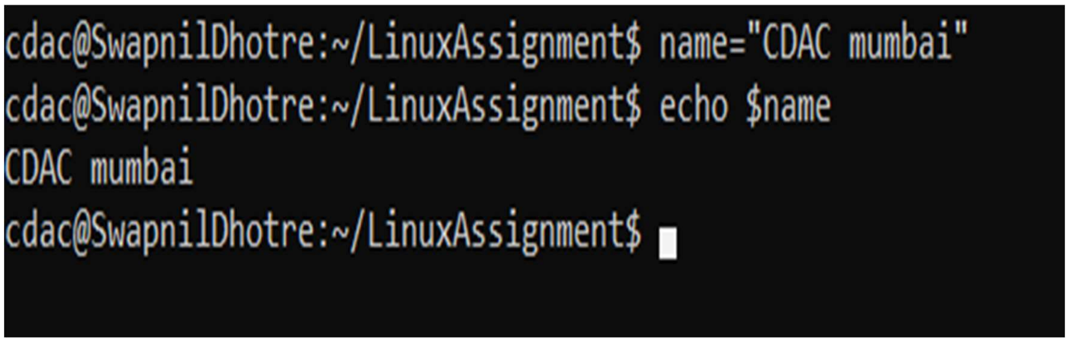
Part C

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



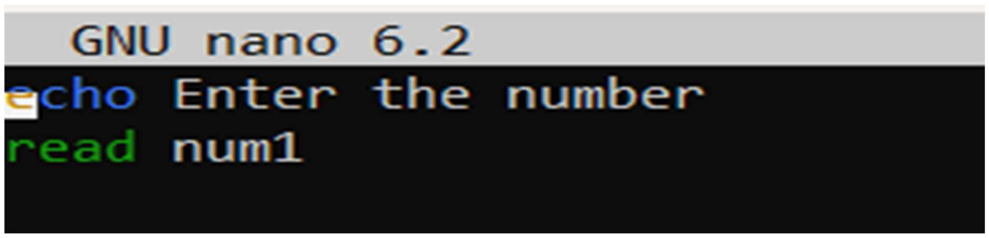
```
cdac@SwapnilDhotre:~$ pwd
/home/cdac
cdac@SwapnilDhotre:~$ nano hellofile
cdac@SwapnilDhotre:~$ bash hellofile
 Hello World!
cdac@SwapnilDhotre:~$ |
```

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

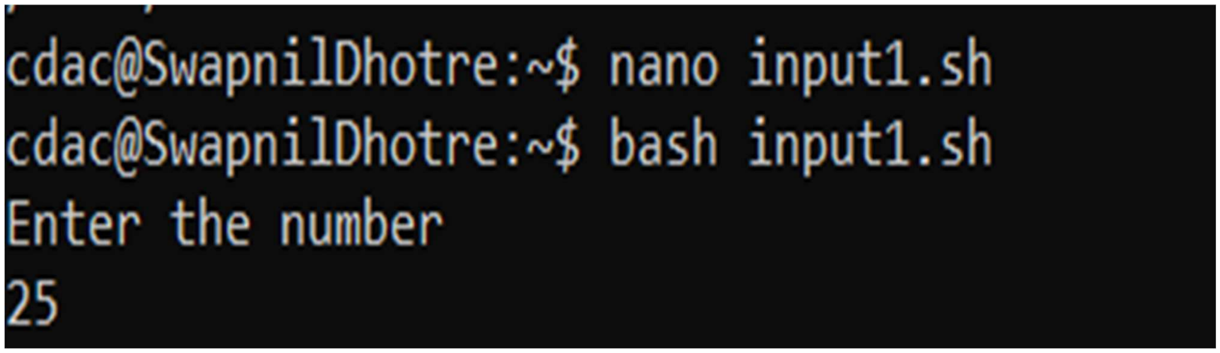


```
cdac@SwapnilDhotre:~/LinuxAssignment$ name="CDAC mumbai"
cdac@SwapnilDhotre:~/LinuxAssignment$ echo $name
CDAC mumbai
cdac@SwapnilDhotre:~/LinuxAssignment$
```

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

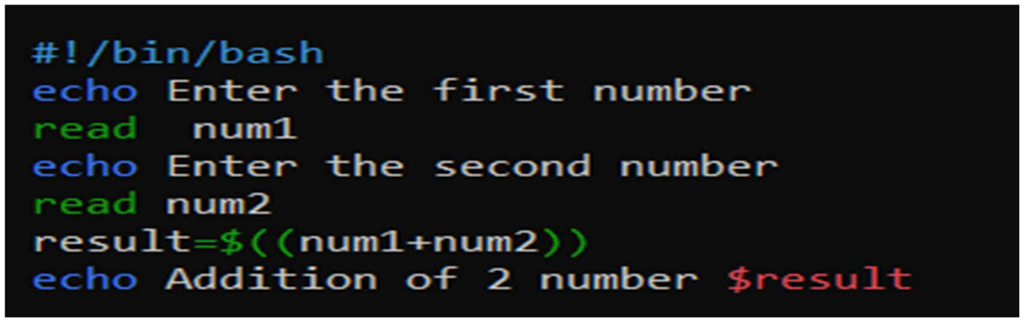


```
  GNU nano 6.2
echo Enter the number
read num1
```

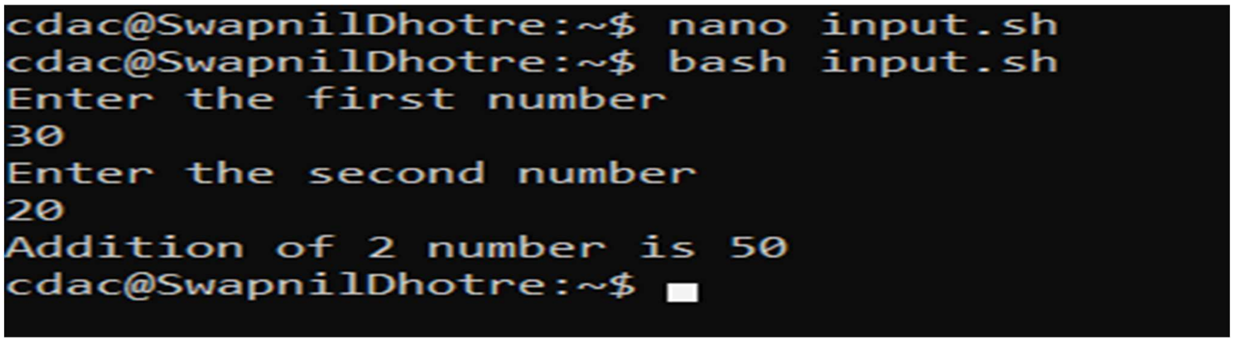


```
cdac@SwapnilDhotre:~$ nano input1.sh
cdac@SwapnilDhotre:~$ bash input1.sh
Enter the number
25
```

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

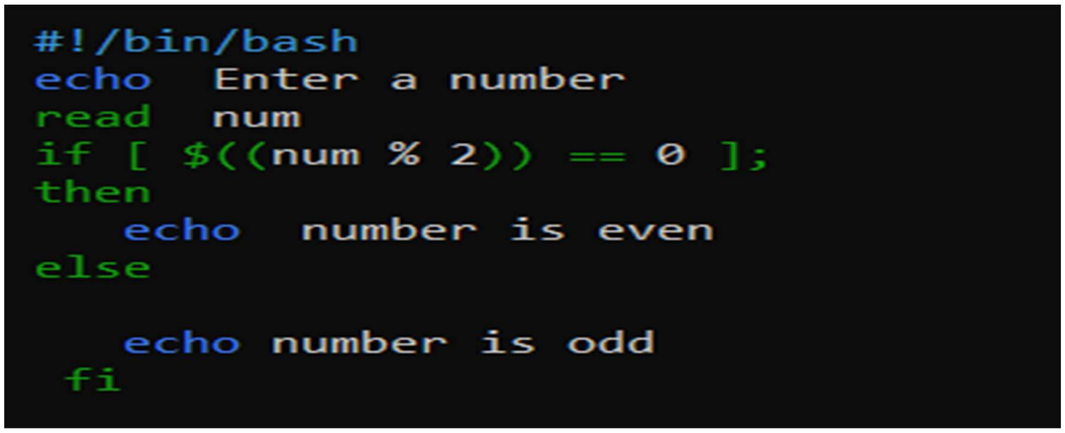


```
#!/bin/bash
echo Enter the first number
read  num1
echo Enter the second number
read num2
result=$((num1+num2))
echo Addition of 2 number $result
```



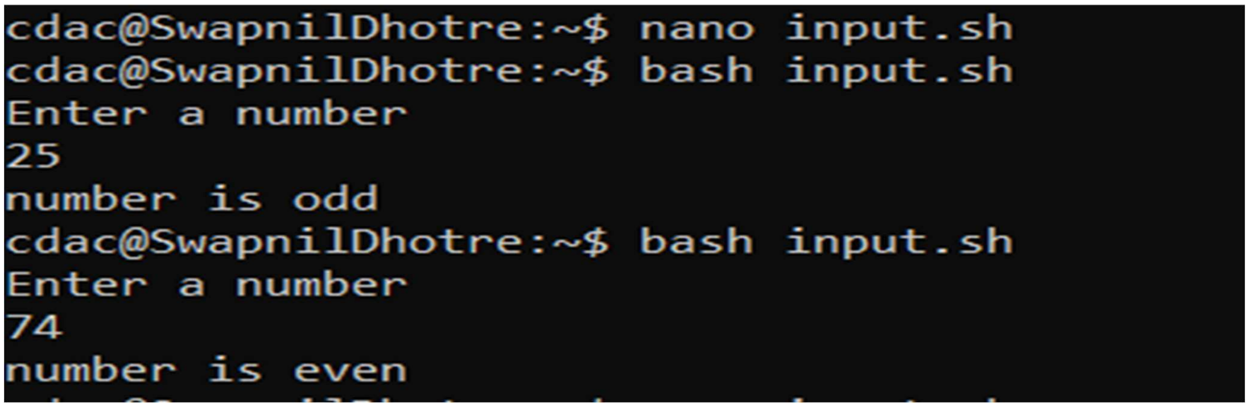
```
cdac@SwapnilDhotre:~$ nano input.sh
cdac@SwapnilDhotre:~$ bash input.sh
Enter the first number
30
Enter the second number
20
Addition of 2 number is 50
cdac@SwapnilDhotre:~$ 
```

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



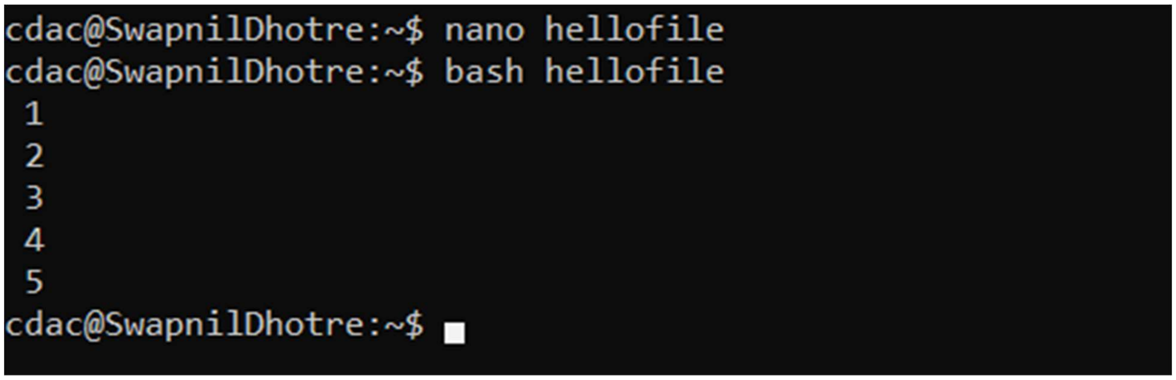
```
#!/bin/bash
echo  Enter a number
read  num
if [ $((num % 2)) == 0 ];
then
    echo  number is even
else

    echo number is odd
 fi
```



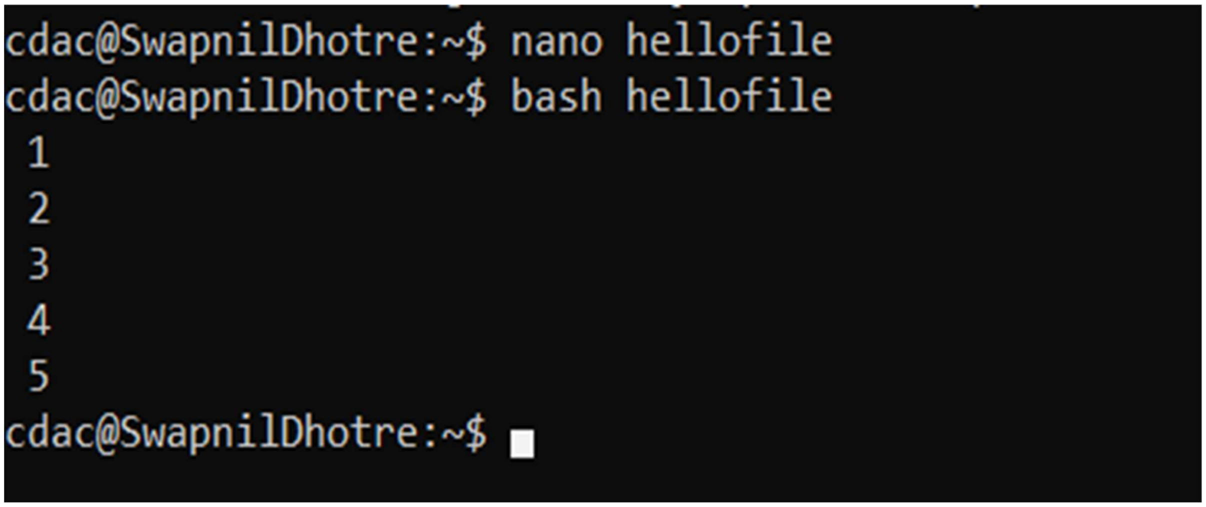
```
cdac@SwapnilDhotre:~$ nano input.sh
cdac@SwapnilDhotre:~$ bash input.sh
Enter a number
25
number is odd
cdac@SwapnilDhotre:~$ bash input.sh
Enter a number
74
number is even
```

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



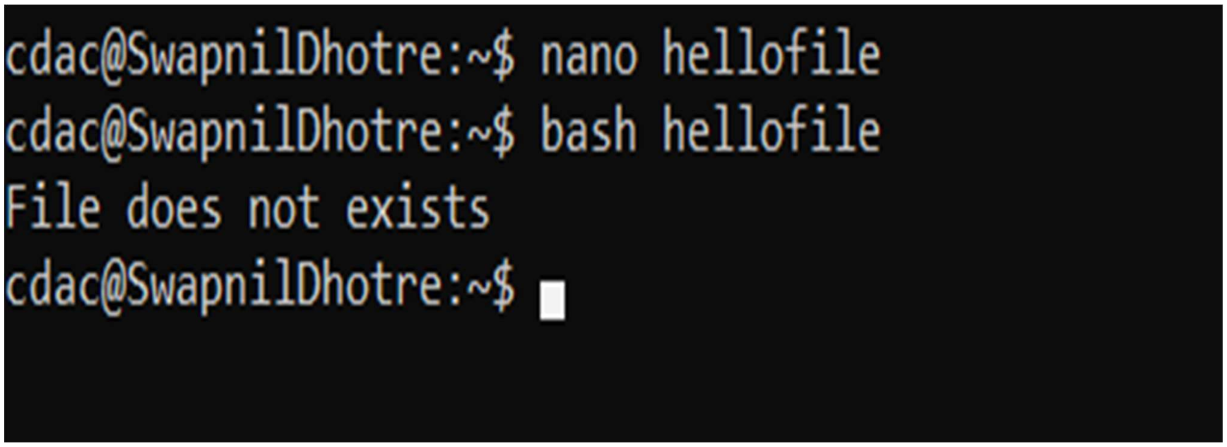
```
cdac@SwapnilDhotre:~$ nano hellofile
cdac@SwapnilDhotre:~$ bash hellofile
 1
 2
 3
 4
 5
cdac@SwapnilDhotre:~$ 
```

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



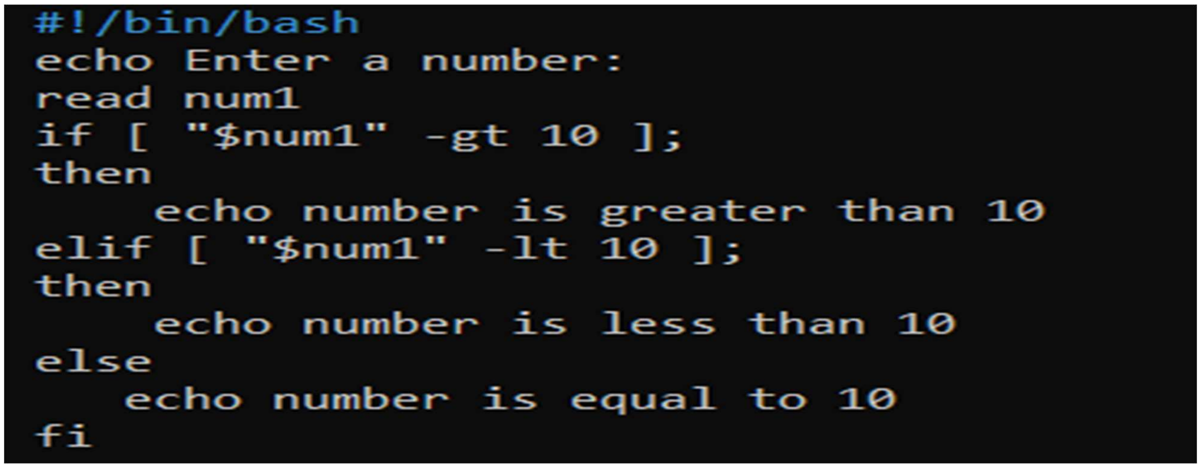
```
cdac@SwapnilDhotre:~$ nano hellofile
cdac@SwapnilDhotre:~$ bash hellofile
 1
 2
 3
 4
 5
cdac@SwapnilDhotre:~$ 
```

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".

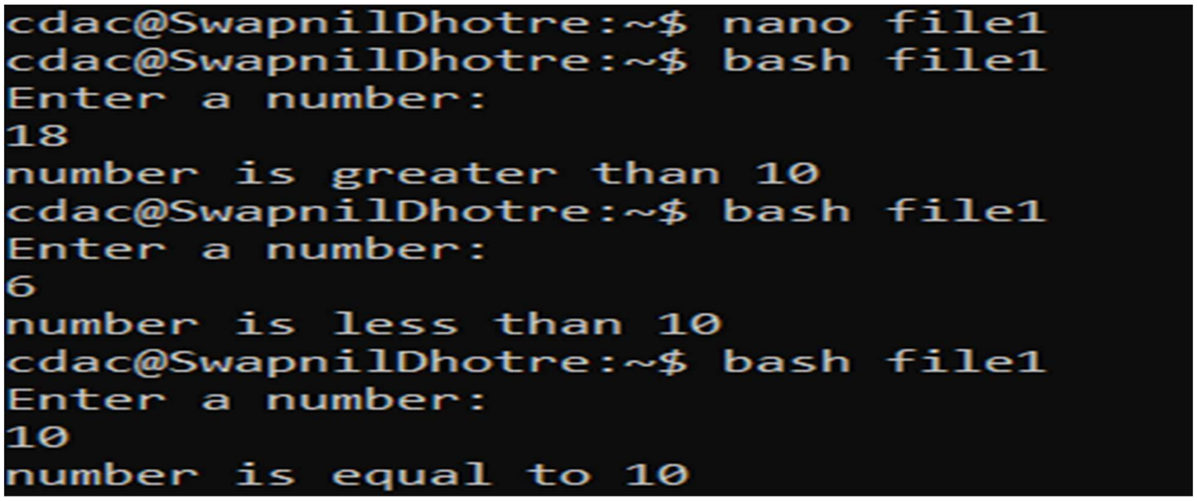


```
cdac@SwapnilDhotre:~$ nano hellofile
cdac@SwapnilDhotre:~$ bash hellofile
File does not exists
cdac@SwapnilDhotre:~$
```

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.

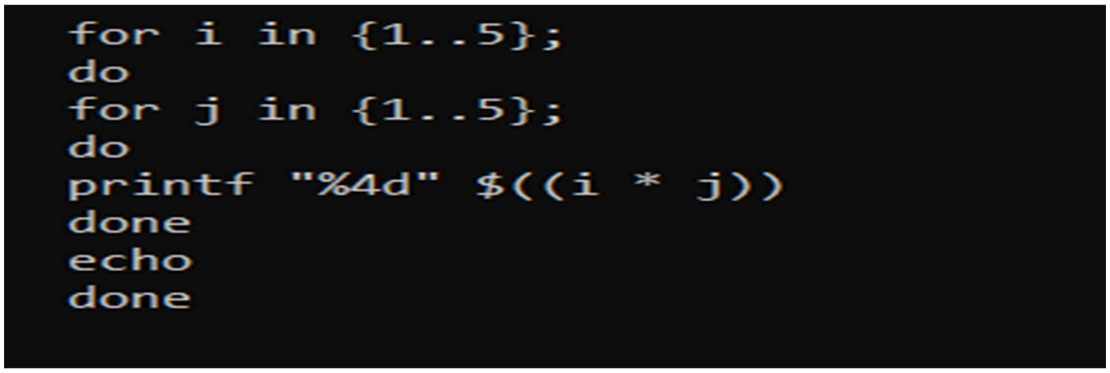


```
#!/bin/bash
echo Enter a number:
read num1
if [ "$num1" -gt 10 ];
then
    echo number is greater than 10
elif [ "$num1" -lt 10 ];
then
    echo number is less than 10
else
   echo number is equal to 10
fi
```

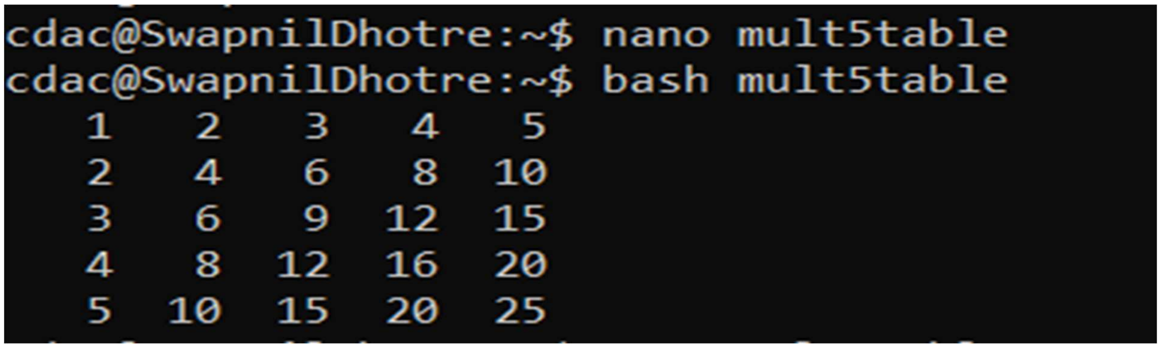


```
cdac@SwapnilDhotre:~$ nano file1
cdac@SwapnilDhotre:~$ bash file1
Enter a number:
18
number is greater than 10
cdac@SwapnilDhotre:~$ bash file1
Enter a number:
6
number is less than 10
cdac@SwapnilDhotre:~$ bash file1
Enter a number:
10
number is equal to 10
```

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.

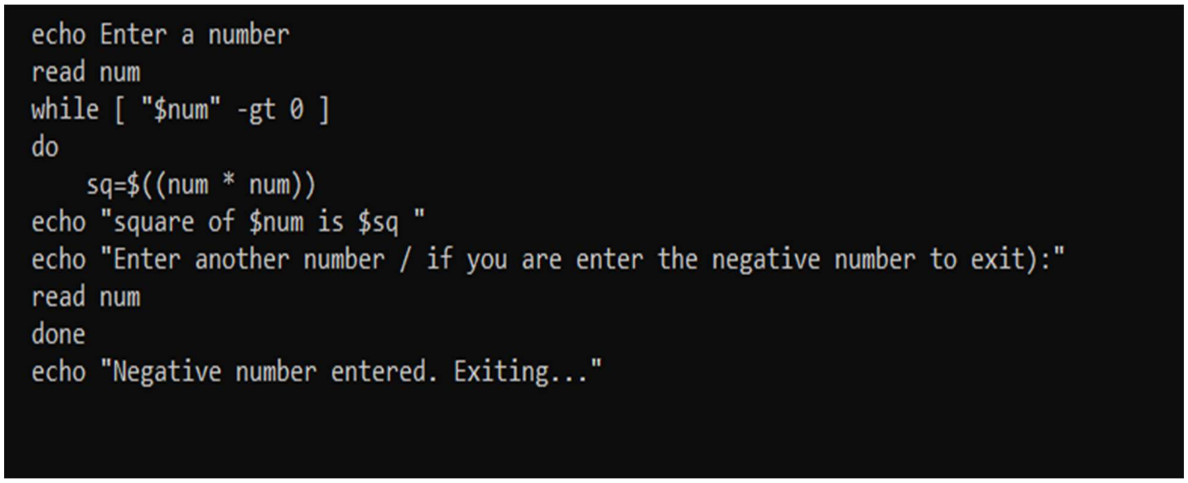


```
for i in {1..5};
do
for j in {1..5};
do
printf "%4d" $((i * j))
done
echo
done
```

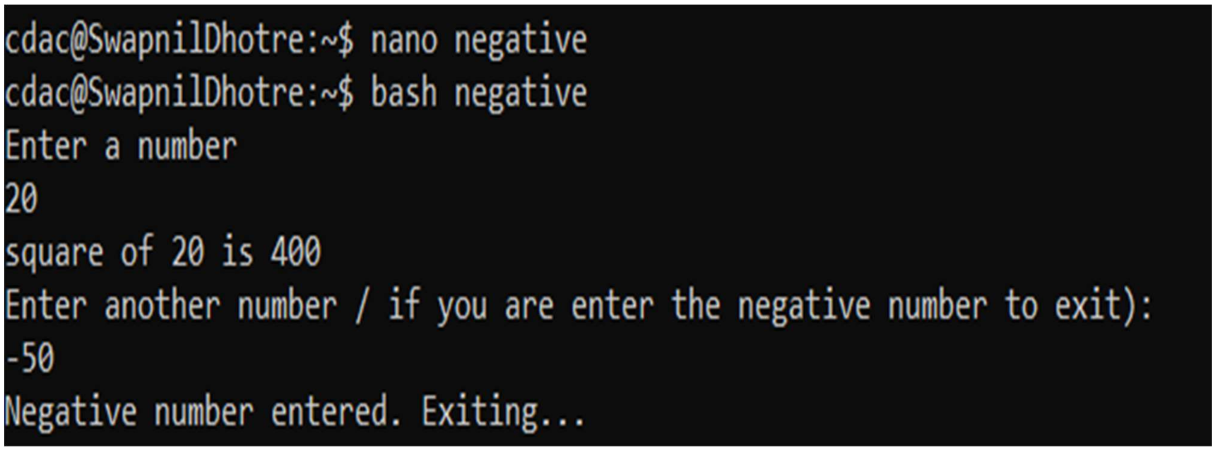


```
cdac@SwapnilDhotre:~$ nano mult5table
cdac@SwapnilDhotre:~$ bash mult5table
   1   2   3   4   5
   2   4   6   8  10
   3   6   9  12  15
   4   8  12  16  20
   5  10  15  20  25
```

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 a number
read num
while [ "$num" -gt 0 ]
do
    sq=$((num * num))
echo "square of $num is $sq "
echo "Enter another number / if you are enter the negative number to exit):"
read num
done
echo "Negative number entered. Exiting..."
```



```
cdac@SwapnilDhotre:~$ nano negative
cdac@SwapnilDhotre:~$ bash negative
Enter a number
20
square of 20 is 400
Enter another number / if you are enter the negative number to exit):
-50
Negative number entered. Exiting...
```

Part D: Common Interview Questions

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

An operating system (OS) is system software that manages hardware resources and provides services for application software. Its primary functions include managing hardware, providing a user interface, managing files and directories, facilitating networking, and managing processes and memory.

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

A **process** is an independent program in execution, with its own memory space. A **thread** is the smallest unit of execution within a process, sharing the process's memory and resources but running independently.

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

Virtual memory is a memory management technique that gives an application the impression it has contiguous memory, while physically it might be fragmented and even partly stored on disk. It works by paging and swapping data between RAM and disk as needed.

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

Multiprogramming: Running multiple programs on a single CPU by switching between them.

Multitasking: Extends multiprogramming by allowing multiple tasks to be performed simultaneously by rapidly switching between them.

Multiprocessing: Using multiple CPUs to execute multiple processes simultaneously.

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

A file system is a way of organizing and storing files on storage devices. Components include directories, files, and metadata (like file size, permissions, timestamps).

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

A deadlock is a situation where two or more processes cannot proceed because each is waiting for the other to release resources. It can be prevented by resource ordering, avoiding hold-and-wait conditions, or preemptively killing processes.

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

The **kernel** is the core part of an OS, managing system resources and communication between hardware and software. The **shell** is an interface that allows users to interact with the kernel via commands.

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

CPU scheduling determines the order in which processes run on the CPU. It's important for ensuring efficient use of the CPU, reducing wait times, and improving overall system performance.

9. How does a system call work?

A system call is a way for programs to interact with the OS. It acts as an interface between user-space applications and OS services, allowing programs to request tasks like file handling, process control, or networking.

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

Device drivers are software components that allow the OS to communicate with hardware devices. They translate OS commands into device-specific actions.

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

The page table maps virtual addresses to physical addresses in memory. It helps the CPU translate a program’s virtual address space into physical memory addresses.

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

Thrashing occurs when excessive paging and swapping deplete system performance, leaving little time for actual computation. It can be avoided by optimizing memory allocation, using proper paging strategies, or adding more physical memory.

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

A semaphore is a synchronization tool used to control access to shared resources by multiple processes. It uses signaling mechanisms (wait and signal operations) to prevent race conditions and ensure proper execution order.

14. How does an operating system handle process synchronization?

An OS handles process synchronization by using synchronization primitives like semaphores, mutexes, and condition variables to coordinate the execution of processes and prevent conflicts in accessing shared resources.

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

An interrupt is a signal to the CPU indicating an event that needs immediate attention. It allows the CPU to temporarily halt the current task, address the interrupt, and then resume normal operations.

16. Explain the concept of a file descriptor.

A file descriptor is an integer that uniquely identifies an open file within a process. It's used by the OS to access files and I/O resources.

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

A system recovers from a crash by rebooting and restoring from backups or using file system consistency checks like fsck. Some systems use journaling or checkpointing to minimize data loss.

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

A **monolithic kernel** has all OS services integrated into one large block of code running in a single address space. A **microkernel** minimizes the kernel by running most services in user space, with only essential services like IPC and basic scheduling in the kernel.

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

Internal fragmentation occurs when allocated memory has unused space. **External fragmentation** happens when free memory is scattered in small blocks, preventing allocation of larger blocks.

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

An OS manages I/O operations using device drivers, buffering, caching, and scheduling to efficiently handle data transfer between hardware devices and memory.

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

Preemptive scheduling allows the OS to interrupt a running process to assign CPU to another process, while **non-preemptive scheduling** lets the current process run until it completes or voluntarily yields control.

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

Round-robin scheduling assigns each process a fixed time slice to run on the CPU. If a process doesn't finish within its time slice, it's moved to the back of the queue, and the next process runs.

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 process with the highest priority. Priorities can be static or dynamic, based on factors like process importance or resource requirements.

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

SJN (or SJF for Shortest Job First) schedules the process with the shortest execution time next. It minimizes average wait time but requires knowledge of process execution times, making it ideal for batch processing.

25. Explain the concept of multilevel queue scheduling.

Multilevel queue scheduling divides processes into several queues based on priority or process type. Each queue can have its own scheduling algorithm, and the CPU is allocated according to predefined rules between queues.

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

A PCB is a data structure that stores all information about a process, including its process ID, state, CPU registers, memory management information, open files, and scheduling information.

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

The process state diagram includes states like **New**, **Ready**, **Running**, **Waiting**, and **Terminated**. Transitions occur as processes are created, scheduled, blocked, or terminated.

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

Processes communicate via Inter-Process Communication (IPC) mechanisms like pipes, message queues, shared memory, and sockets.

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

Process synchronization ensures that multiple processes execute in a coordinated manner, especially when accessing shared resources, to prevent race conditions and ensure data consistency.

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 hasn't read its exit status. It's created when the child terminates before the parent.

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

Internal fragmentation: Wasted space within allocated memory blocks.

External fragmentation: Wasted space outside allocated blocks, leading to scattered free memory.

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

Demand paging loads pages into memory only when they're needed, reducing memory usage and improving efficiency by keeping only active parts of processes in memory.

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

The page table translates virtual addresses to physical addresses, enabling the OS to use virtual memory by keeping track of where each page is located in physical memory.

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

The MMU handles the translation of virtual addresses to physical addresses using the page table, enabling processes to run in virtual memory spaces.

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

Thrashing occurs when excessive paging leads to low CPU utilization. It can be avoided by optimizing memory allocation, using working set models, or adding more physical memory.

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

A system call allows user programs to request services from the OS kernel, such as file operations, process control, and communication.

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

A **monolithic kernel** integrates all OS services within a single large kernel, while a **microkernel** only includes essential services, with other services running in user space.

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

The OS manages I/O operations using device drivers, buffering, caching, and I/O scheduling to efficiently transfer data between devices and memory.

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 resources simultaneously, leading to unpredictable outcomes. It can be prevented using synchronization mechanisms like locks, semaphores, or atomic operations.

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

Device drivers act as intermediaries between the OS and hardware devices, translating OS commands into actions the device can perform and vice versa.

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

A zombie process occurs when a child process terminates but its parent process hasn't read its exit status. It can be prevented by the parent process using wait() to clean up the child's resources.

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 has terminated. The OS handles orphans by reassigning them to the init process (or its equivalent), which then becomes the new parent and reaps them when they terminate.

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

A parent process creates a child process, and the child inherits some resources and attributes from the parent. The parent can control or communicate with the child, and it's responsible for cleaning up the child process after it terminates.

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 process by duplicating the calling process. The new process (child) is identical to the parent but has a different process ID.

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

A parent process can use the wait() or waitpid() system calls to wait for a child process to finish execution and retrieve its exit status.

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

The exit status returned by wait() allows the parent process to determine how the child process terminated (e.g., normal exit, error, signal) and take appropriate actions.

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

A parent process can terminate a child process using the kill() system call, sending a termination signal to the child process.

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

A **process group** is a collection of processes that can be signaled as a group. A **session** is a collection of process groups, typically associated with a user session, with one process acting as the session leader.

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

The exec() functions replace the current process image with a new one, specified by a file path. The new process image starts executing from its entry point, effectively transforming the existing process into a new program.

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

waitpid() allows a parent process to wait for a specific child process to terminate, providing more control over which child process it waits for, unlike wait(), which waits for any child process.

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

A process terminates when it calls exit(), returns from main(), or receives a termination signal. The OS then cleans up the process's resources and notifies the parent process.

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

The long-term scheduler controls the degree of multiprogramming by determining which processes are admitted to the ready queue from the job pool. It influences system load and resource utilization.

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

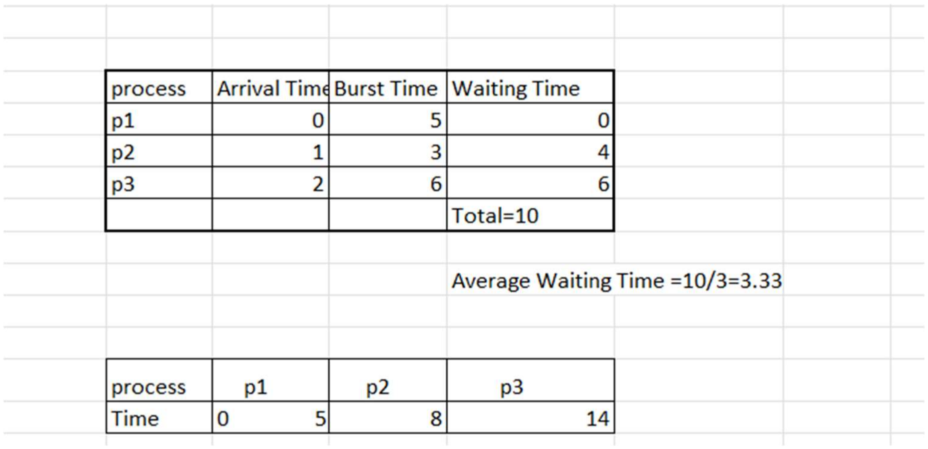
The **short-term scheduler** (CPU scheduler) runs frequently to select the next process to run. The **long-term scheduler** runs less frequently to manage the pool of jobs in the system. The **medium-term scheduler** manages swapping processes in and out of memory to balance load.

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

The medium-term scheduler might be invoked when the system is low on memory, swapping out an inactive process to free up RAM for active processes, thereby improving system performance and resource management.

Part E

1. **Consider the following processes with arrival times and burst times: |Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.**

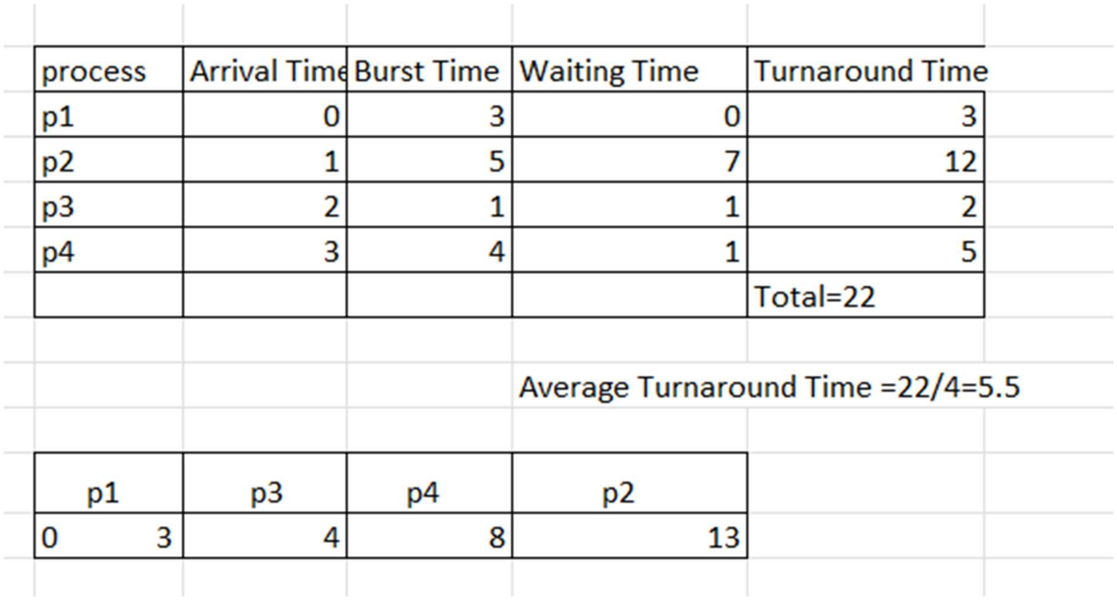


process	Arrival Time	Burst Time	Waiting Time
p1	0	5	0
p2	1	3	4
p3	2	6	6
			Total=10

Average Waiting Time =10/3=3.33

process	p1	p2	p3
Time	0 5	8	14

2. **Consider the following processes with arrival times and burst times: | Calculate the average turnaround time using Shortest Job First (SJF) scheduling.**

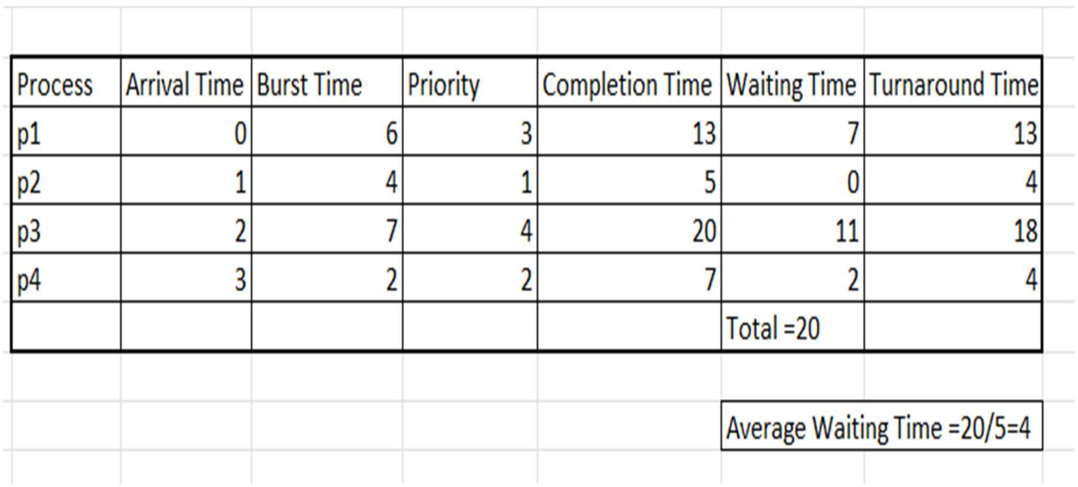


process	Arrival Time	Burst Time	Waiting Time	Turnaround Time
p1	0	3	0	3
p2	1	5	7	12
p3	2	1	1	2
p4	3	4	1	5
				Total=22

Average Turnaround Time =22/4=5.5

p1	p3	p4	p2
0 3	4	8	13

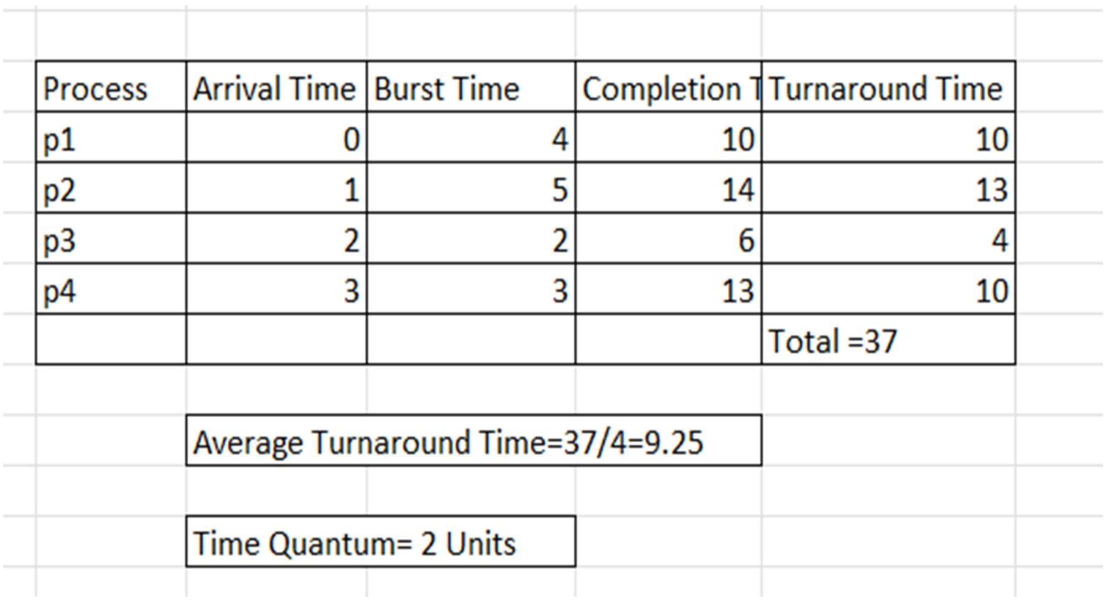
3. **Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority).Calculate the average waiting time using Priority Scheduling.**



Process	Arrival Time	Burst Time	Priority	Completion Time	Waiting Time	Turnaround Time
p1	0	6	3	13	7	13
p2	1	4	1	5	0	4
p3	2	7	4	20	11	18
p4	3	2	2	7	2	4
					Total =20	

Average Waiting Time =20/5=4

4. **Consider the following processes with arrival times and burst times, and the c for Round Robin scheduling is 2 units. Calculate the average turnaround time using Round Robin scheduling**



Process	Arrival Time	Burst Time	Completion T	Turnaround Time
p1	0	4	10	10
p2	1	5	14	13
p3	2	2	6	4
p4	3	3	13	10
				Total =37

Average Turnaround Time=37/4=9.25

Time Quantum= 2 Units

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

When a program uses the fork() system call to create a child process, the following happens:

1. **Forking Behavior**: The fork() system call creates a new child process that is a duplicate of the parent process. After the fork(), both the parent and child processes have their own separate copies of the variables and memory.
2. **Initial State**: Before the fork, the parent process has a variable x with a value of 5.
3. **After Forking**: Both the parent and child processes have their own separate copies of the variable x with the initial value of 5.
4. **Modification of x**: Both processes execute instructions that increment the value of x by 1.

Final Values:

In the Parent Process: After the fork, the parent process increments its own copy of x by 1. Therefore, the final value of x in the parent process will be 5 + 1 = 6.

In the Child Process: The child process also increments its own copy of x by 1.Therefore, the final value of x in the child process will be 5 + 1 = 6.