

## Assignment2

### Part A -

What will the following commands do?

1: echo "Hello, World!"

Ans: Prints Hello, World! to the terminal.

2: name="Productive"

Ans: Creates a variable name and assigns it the value Productive .

3: touch file.txt

Ans: Creates an empty file named file.txt or updates its timestamp if it already exists.

4: ls -a

Ans: Lists all files and directories in the current directory, including hidden ones (those starting with .

5: rm file.txt

Ans: Removes the file file.txt permanently.

6:cp file1.txt file2.txt

Ans: Copies file1.txt to file2.txt . If file2.txt exists, it will be overwritten.

6: mv file.txt /path/to/directory/

Ans: Moves file.txt to the specified directory.

7: chmod 755 script.sh

Ans: Grants the owner full permissions (read, write, execute) and gives others read and execute permissions on script.sh

8:grep "pattern" file.txt

Ans: Searches for occurrences of "pattern" in file.txt and prints matching lines.

9: kill PID

Ans: Terminates the process with the specified Process ID (PID).

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

Ans: creates a directory mydir > changes into mydir > creates an empty file file.txt > writes "Hello, World!" into file.txt > displays the contents of file.txt

11: ls -l | grep ".txt"

Ans: Lists files in long format and filters only those containing ". Txt" in their names.

12: cat file1.txt file2.txt | sort | uniq

Ans: Concatenates file1.txt and file2.txt , sorts them, and removes duplicate lines.

13: ls -l | grep "^d"

Ans: Lists directories (entries starting with d in long format output).

14: grep -r "pattern" /path/to/directory/

Ans: Searches for "pattern" recursively in all files under /path/to/directory/ .

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

Ans: Concatenates file1.txt and file2.txt , sorts them, and displays only duplicate lines.

16: chmod 644 file.txt

Ans: Grants the owner read and write permissions, while others get read-only access to file.txt .

17: cp -r source\_directory destination\_directory

Ans: Recursively copies source\_directory to destination\_directory , preserving contents.

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

Ans: Finds all .txt files in /path/to/search and its subdirectories.

19: chmod u+x file.txt

Ans: Gives the owner ( u ) execute permission on file.txt .

20: echo \$PATH

Ans: Displays the system's PATH environment variable, listing directories where executable files are searched for.

## Part B

Identify True or False

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

Identify the Incorrect Commands:

1. Incorrect - chmodx is not a valid command. The correct command to change file permissions is chmod .
2. Incorrect - cpy is not a valid command. The correct command to copy files and directories is cp .
3. Incorrect - mkfile is not a standard Linux command. To create a new file, use touch filename .
4. Incorrect - catx is not a valid command. The correct command to concatenate files is cat .
5. Incorrect - rn is not a valid command. To rename files, use the mv command ( mv oldname newname ).

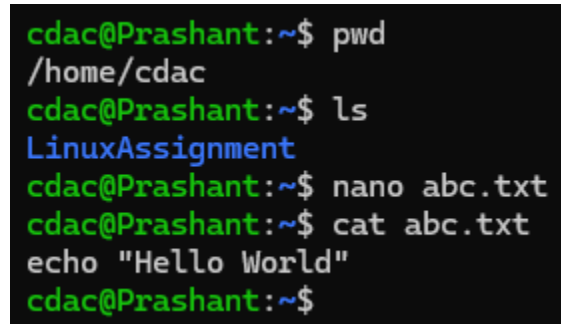
## Part C

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

Ans:

```
#!/bin/bash
```

```
echo "Hello, World!"
```

A terminal window with a black background and green text. The prompt is 'cdac@Prashant:~\$'. The user enters 'pwd', and the output is '/home/cdac'. The user enters 'ls', and the output is 'LinuxAssignment'. The user enters 'nano abc.txt'. The user enters 'cat abc.txt', and the output is 'Hello World'. The prompt returns to 'cdac@Prashant:~\$'.

```
cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
Hello World
cdac@Prashant:~$
```

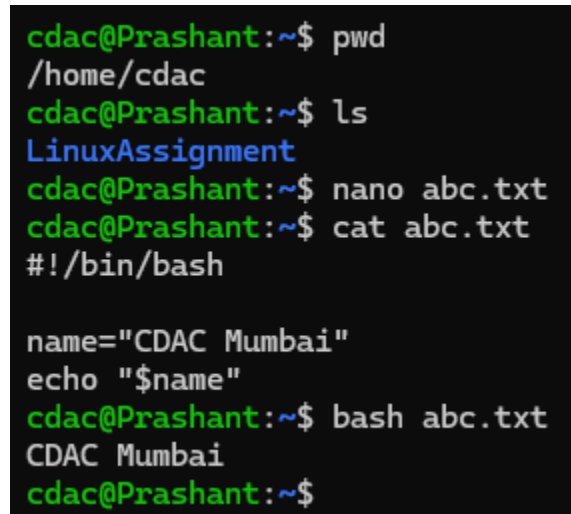
Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.

Ans:

```
#!/bin/bash
```

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

```
echo "$name"
```

A terminal window with a black background and green text. The prompt is 'cdac@Prashant:~\$'. The user enters 'pwd', and the output is '/home/cdac'. The user enters 'ls', and the output is 'LinuxAssignment'. The user enters 'nano abc.txt'. The user enters 'cat abc.txt'. The user enters '#!/bin/bash'. The user enters 'name="CDAC Mumbai"'. The user enters 'echo "\$name"', and the output is 'CDAC Mumbai'. The user enters 'bash abc.txt'. The prompt returns to 'cdac@Prashant:~\$'.

```
cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash

name="CDAC Mumbai"
echo "$name"
cdac@Prashant:~$ bash abc.txt
CDAC Mumbai
cdac@Prashant:~$
```

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

Ans:

```
#!/bin/bash
```

```
read -p "Enter a number: " num
```

```
echo "You entered: $num"
```

```

cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
read -p "Enter a number: " num
echo "You entered: $num"
cdac@Prashant:~$ bash abc.txt
Enter a number: 12
You entered: 12
cdac@Prashant:~$

```

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

Ans

```
#!/bin/bash
```

```
a=5
```

```
b=3
```

```
sum=$((a + b)) echo "Sum: $sum"
```

```

cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
a=5
b=3
sum=$((a + b))
echo "Sum: $sum"
cdac@Prashant:~$ bash abc.txt
Sum: 8
cdac@Prashant:~$

```

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

Ans:

```
#!/bin/bash read -p "Enter a number: " num
```

```
if ((num % 2 == 0)); then
echo "Even"
else
echo "Odd"
fi
```

```
cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat ac
cat: ac: No such file or directory
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
read -p "Enter a number: " num
if ((num % 2 == 0)); then
echo "Even"
else
echo "Odd"
fi
cdac@Prashant:~$ bash abc.txt
Enter a number: 24
Even
cdac@Prashant:~$
```

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

Ans:

```
#!/bin/bash
for i in {1..5}; do
echo "$i"
done
```

```
cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
for i in {1..5}; do
echo "$i"
done
cdac@Prashant:~$ bash abc.txt
1
2
3
4
5
cdac@Prashant:~$
```

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

Ans:

```
#!/bin/bash
```

```
i=1
```

```
while [ $i -le 5 ]; do
```

```
echo "$i"
```

```
((i++))
```

```
Done
```

```

cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
i=1
while [ $i -le 5 ]; do
echo "$i"
((i++))
done
cdac@Prashant:~$ bash abc.txt
1
2
3
4
5
cdac@Prashant:~$

```

Question 8: Write a shell script that checks if a file named "file. Txt" exists in the current directory.

Ans:

```
#!/bin/bash
```

```
if [ -f "file.txt" ]; then
```

```
echo "File exists"
```

```
else
```

```
echo "File does not exist"
```

```
fi
```



```

cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ touch abc.txt def.txt ghi.txt jkl.txt
cdac@Prashant:~$
cdac@Prashant:~$ ls
LinuxAssignment abc.txt def.txt ghi.txt jkl.txt
cdac@Prashant:~$
cdac@Prashant:~$ nano jkl.txt
cdac@Prashant:~$ cat jkl.txt
#!/bin/bash
if [ -f "file.txt" ]; then
echo "File exists"
else
echo "File does not exist"
fi
cdac@Prashant:~$ bash jkl.txt
File does not exist
cdac@Prashant:~$ touch file.txt
cdac@Prashant:~$ bash jkl.txt
File exists
cdac@Prashant:~$

```

Question 9: Write a shell script that checks if a number is greater than 10 and prints a message accordingly.

Ans:

```

#!/bin/bash
if [ -f "file.txt" ]; then
echo "File exists"
else
echo "File does not exist"
fi

```

```
cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
read -p "Enter a number: " num
if [ $num -gt 10 ]; then
echo "Number is greater than 10"
else
echo "Number is 10 or less"
fi
cdac@Prashant:~$ bash abc.txt
Enter a number: 12
Number is greater than 10
cdac@Prashant:~$ 8
8: command not found
cdac@Prashant:~$
```

Question 10: Write a shell script that prints a multiplication table for numbers from 1 to 5.

Ans:

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

```

cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
for i in {1..5}; do
for j in {1..5}; do
printf "%4d" $((i * j))
done
echo
done
cdac@Prashant:~$ bash abc.txt
 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
cdac@Prashant:~$

```

Question 11: Write a shell script that reads numbers from the user until a negative number is entered.

Ans:

```
#!/bin/bash
```

```
for num in {1..5}; do
```

```
echo "Multiplication table for $num: "
```

```
for i in {1..10}; do
```

```
echo "$num * $i = $((num * i))"
```

```
Done
```

```
echo ""
```

```
done
```

```

cdac@Prashant:~$ pwd
/home/cdac
cdac@Prashant:~$ ls
LinuxAssignment
cdac@Prashant:~$ nano abc.txt
cdac@Prashant:~$ cat abc.txt
#!/bin/bash
for num in {1..5}; do
    echo "Multiplication table for $num: "
    for i in {1..10}; do
        echo "$num * $i = $((num * i))"
    done
    echo ""
done
cdac@Prashant:~$ bash abc.txt
Multiplication table for 1:
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10

Multiplication table for 2:
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20

Multiplication table for 3:
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30

Multiplication table for 4:
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

Multiplication table for 5:
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

cdac@Prashant:~$

```

## Part D - Common Interview Questions Operating Systems Q&A

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

Ans: An operating system (OS) is system software that manages hardware resources and provides services for computer programs. Its primary functions include:

- Process management: Scheduling and controlling processes.
- Memory management: Allocating and deallocating memory.

- File system management: Organizing and managing data storage.
- Device management: Controlling hardware devices.
- Security and access control: Ensuring proper access to resources.

2. Explain the difference between process and thread.

- Ans: Process: A program in execution, having its own memory space.
- Thread: A lightweight process that shares the same memory space with other threads in the same process. Multiple threads can exist within a single process.

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

Ans: Virtual memory is an abstraction that allows programs to use more memory than physically available by swapping data between RAM and disk storage. It works by using a page table to map virtual addresses to physical addresses.

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

- Ans: Multiprogramming: Running multiple programs simultaneously by switching between
- them.
- Multitasking: The OS executes multiple tasks (processes or threads) concurrently.
- Multiprocessing: Use of multiple processors (CPUs) to run tasks in parallel.

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

Ans: A file system is a method for storing and organizing computer files. Its components include:

- File control block (FCB): Information about files.
- Directories: Organize files.
- Data blocks: Store actual file data.

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

Ans: A deadlock occurs when two or more processes are blocked forever, waiting for each other. It can be prevented by using techniques like resource allocation graphs or the Banker's algorithm.

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

Ans:

- Kernel: The core part of the OS that manages hardware and system resources.
- Shell: A user interface that allows users to interact with the OS.

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

Ans: CPU scheduling is the method the OS uses to decide which process runs next on the CPU. It is important for optimizing CPU usage and system performance.

9. How does a system call work?

Ans: A system call is a request from a program to the OS for services such as file manipulation, process control, or I/O operations.

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

Ans: Device drivers are programs that enable the operating system to communicate with hardware devices (e.g., printers, hard drives).

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

Ans: A page table maps virtual memory addresses to physical memory addresses, enabling efficient memory management.

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

Ans: Thrashing occurs when the OS spends more time swapping data between disk and memory than executing processes. It can be avoided by managing the degree of multiprogramming or using page replacement algorithms.

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

Ans: A semaphore is a synchronization tool used to control access to a shared resource by multiple processes in concurrent programming.

14. How does an operating system handle process synchronization?

Ans: The OS ensures that processes coordinate and avoid conflicts when accessing shared resources, typically using locks, semaphores, or monitors.

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

Ans: An interrupt is a signal to the processor indicating an event that needs immediate attention, allowing the OS to pause the current process and handle the interrupt.

16. Explain the concept of a file descriptor.

Ans: A file descriptor is a unique identifier used by the OS to access an open file or other input/output resources.

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

Ans: After a system crash, the OS typically uses a technique like journaling to recover data or performs a file system check to restore consistency.

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

Ans:

- Monolithic Kernel: A large, single kernel that directly controls hardware and system resources.
- Microkernel: A smaller kernel that only provides essential services, with other services running in user space.

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

Ans:

- Internal Fragmentation: Wasted space within allocated memory blocks.
- External Fragmentation: Wasted space between allocated memory blocks.

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

Ans: The OS manages I/O operations through device drivers, buffering, and scheduling to ensure efficient data transfer.

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

Ans:

- Preemptive: The OS can interrupt a running process to allocate CPU time to another process.
- Non-preemptive: A process runs until it voluntarily relinquishes control.

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

Ans: Round-robin scheduling is a preemptive scheduling algorithm where each process is assigned a fixed time slice (quantum) to run before being swapped out.

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

Ans: Processes are assigned priorities, and the OS executes the process with the highest priority. Priorities can be static or dynamic.

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

Ans: SJN schedules the process with the shortest burst time next. It is ideal for minimizing average waiting time but requires knowing process durations in advance.

25. Explain the concept of multilevel queue scheduling.

Ans: Processes are divided into multiple queues based on priority or other criteria. Each queue may have its own scheduling algorithm.

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

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

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

Ans: A diagram illustrating the different states a process can be in (e.g., running, waiting, ready, terminated) and transitions between them.

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

Ans: Processes communicate through mechanisms such as pipes, message queues, shared memory, or sockets.

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

Ans: Process synchronization ensures that processes do not interfere with each other when accessing shared resources, typically using locks, semaphores, or condition variables.

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

Ans: A zombie process is a process that has completed execution but still has an entry in the process table. It occurs when the parent doesn't read the exit status.

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



Ans:

- Internal Fragmentation: Wasted space within allocated memory blocks.
- External Fragmentation: Wasted space between allocated memory blocks.

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

Ans: Demand paging is a memory management technique where pages are only loaded into memory when needed, improving memory efficiency.

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

Ans: The MMU is a hardware component that maps virtual addresses to physical addresses using the page table, enabling virtual memory.

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

Ans: A system call allows user programs to request services such as file manipulation, process control, or I/O operations from the OS.

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

Ans: A race condition occurs when the outcome of a program depends on the order of execution of concurrent processes. It can be prevented using synchronization techniques like locks.

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

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

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

Ans: When a process terminates, its resources are reclaimed, and the process state is set to terminated. The parent is notified through signals or the wait() system call.

38. What is the role of the long-term scheduler in process scheduling?

Ans: The long-term scheduler decides which processes should be admitted into the ready queue, influencing the degree of multiprogramming.

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

Ans: The parent process can use the wait() system call to wait for the child process to finish execution.

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

Ans: The exit status indicates whether a child process completed successfully or encountered an error.

## Part E - Scheduling and Process Management Questions

41. 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:

Process	Arrival Time	Burst Time	Completion Time	Response Time	Waiting Time	Turn around Time
P <sub>1</sub>	0	5	5	0	0	5
P <sub>2</sub>	1	3	8	4	4	7
P <sub>3</sub>	2	6	14	6	6	12
Avg: $\frac{10}{3} = 3.33$				$\frac{10}{3} = 3.33$	$\frac{24}{3} = 8$	

Gantt chart =

P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>
0	5	8
		14

42. 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:

Process	Arrival Time	Burst Time	Completion Time	Response Time	Waiting Time	Turn around Time
P <sub>1</sub>	0	3	3	0	0	3
P <sub>2</sub>	1	5	13	7	7	12
P <sub>3</sub>	2	1	4	1	1	2
P <sub>4</sub>	3	4	8	1	1	5
Avg:				$= \frac{9}{4} = 2.25$	$= \frac{9}{4} = 2.25$	$= \frac{22}{4} = 5.5$

Gantt chart =

0      P<sub>1</sub>      P<sub>3</sub>      P<sub>4</sub>      P<sub>2</sub>

          3      4      8      13

43. 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. Primitive Method  
Ans:

Page No.:

Process	Arrival Time	Burst Time	Priority	Completion Time	TAT	Waiting Time	Response Time
P <sub>1</sub>	0	6	3	13	13	7	0
P <sub>2</sub>	1	4	1	5	4	0	0
P <sub>3</sub>	2	7	4	20	18	11	11
P <sub>4</sub>	3	2	2	7	4	2	2
				$\frac{39}{4} = 9.75$	$\frac{20}{4} = 5$	$\frac{13}{4} = 3.25$	

Gantt Chart = P<sub>1</sub> P<sub>2</sub> P<sub>2</sub> P<sub>2</sub> P<sub>2</sub> P<sub>4</sub> P<sub>4</sub> P<sub>3</sub> P<sub>3</sub>

0 1 2 3 4 5 6 7 13 20

44. 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.

Ans:

Process	Arrival Time	Burst Time	Completion Time	TAT	Waiting Time	Response Time
P <sub>1</sub>	0	4	10	10	6	0
P <sub>2</sub>	1	5	14	13	8	1
P <sub>3</sub>	2	2	6	4	2	2
P <sub>4</sub>	3	3	13	10	7	3

$$= \frac{37}{4} = 9.25 = \frac{23}{4} = 5.75 = \frac{6}{4} = 1.5$$

Gantt chart = P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> P<sub>1</sub> P<sub>2</sub> P<sub>4</sub> P<sub>2</sub>

0 2 4 6 8 10 12 13 14

45. Fork System Call Scenario 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:

step 1: Before `fork()` is called

`int x = 5;`

step 2: Calling `fork()`

- `fork()` sys call creates a new child process.
- Both parent & child have separate.