**Concepts of Operating System**

**Assignment 2**

Name – Dhirajkumar Sirsat\_KH

**Part A**

**What will the following commands do?**

• echo "Hello, World!"

* Prints Hello, World

• name="Productive"

* Assigns String “Productive” to the “name” variable

echo $name will print the String “Productive”

• touch file.txt

* It creates the file with name as “file.txt” if doesn’t exists

• ls -a

* It list out all the files and directories including hidden files

• rm file.txt

* Deletes the file with file name “file.txt”

• cp file1.txt file2.txt

* It copies the contents from file1.txt to file2.txt. If file2.txt does not exist it will get created.

• mv file.txt /path/to/directory/

* It moves the “file.txt” file to the specified path

• chmod 755 script.sh

=> Changes the permissions for script.sh

-rwxr-xr-x 1 cdac cdac 20 Aug 30 15:43 script.sh

• grep "pattern" file.txt

* It searches for the “pattern” string in file.txt and prints that line

• kill PID

* It will kill the process with specified Process ID

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

* mkdir mydir - will create the directory with “mydir” name

cd mydir – change the directory to “mydir”

touch file.txt – create the file with filename as “file.txt “ under mydir directory.

echo "Hello, World!" – writes the “Hello, World!” into file.txt file

cat file.txt – displays the file contents

• ls -l | grep ".txt"

* It lists the files with file names having “.txt”

• cat file1.txt file2.txt | sort | uniq

* It prints the sorted content from both file1.txt and file2.txt only unique entries. It will remove the duplicates

• ls -l | grep "^d"

* It lists Directories only

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

* It search for pattern in all the directories and subdirectories.

• cat file1.txt file2.txt | sort | uniq –d

* Display only duplicate values from both file1.txt and file2.txt

• chmod 644 file.txt

* It adds the permissions ‘rw’ to user/owner, ‘r’ to group and other

?? • cp -r source\_directory destination\_directory

* Copies the structure of source\_directory to destination\_directory

• find /path/to/search -name "\*.txt"

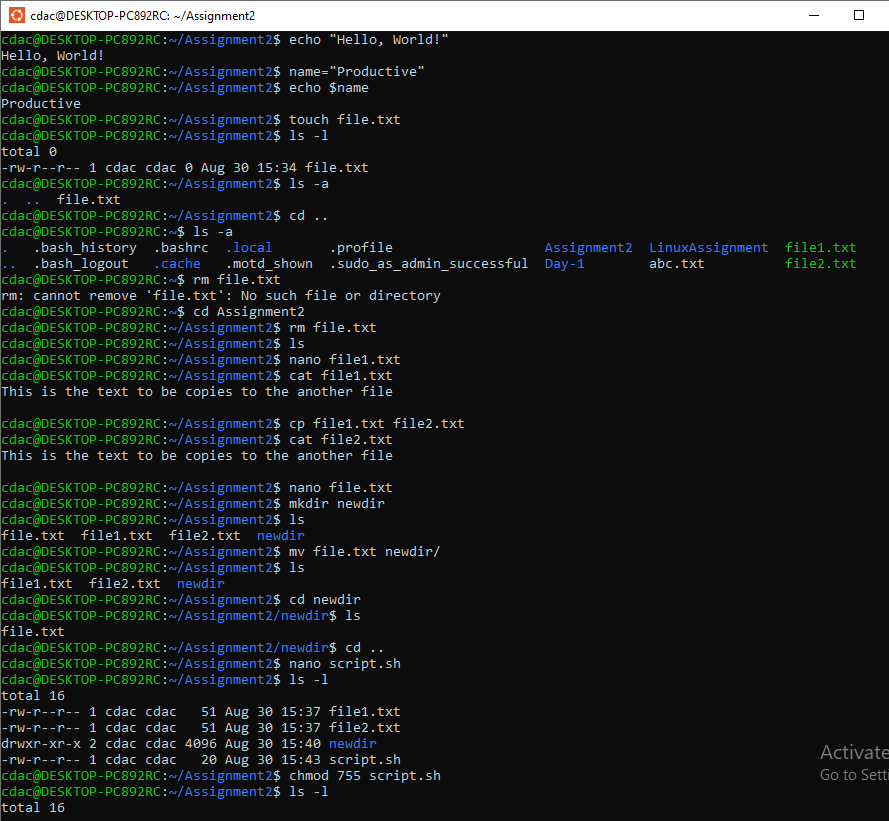
* It finds all the “.txt” files under current directory and sub-directories

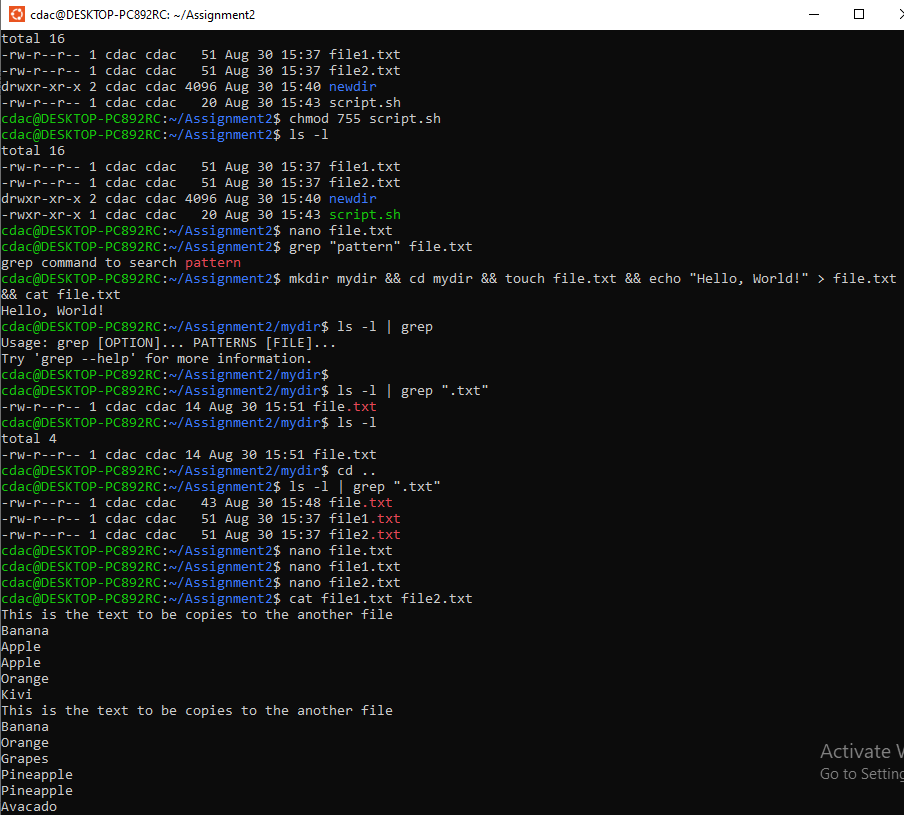
• chmod u+x file.txt

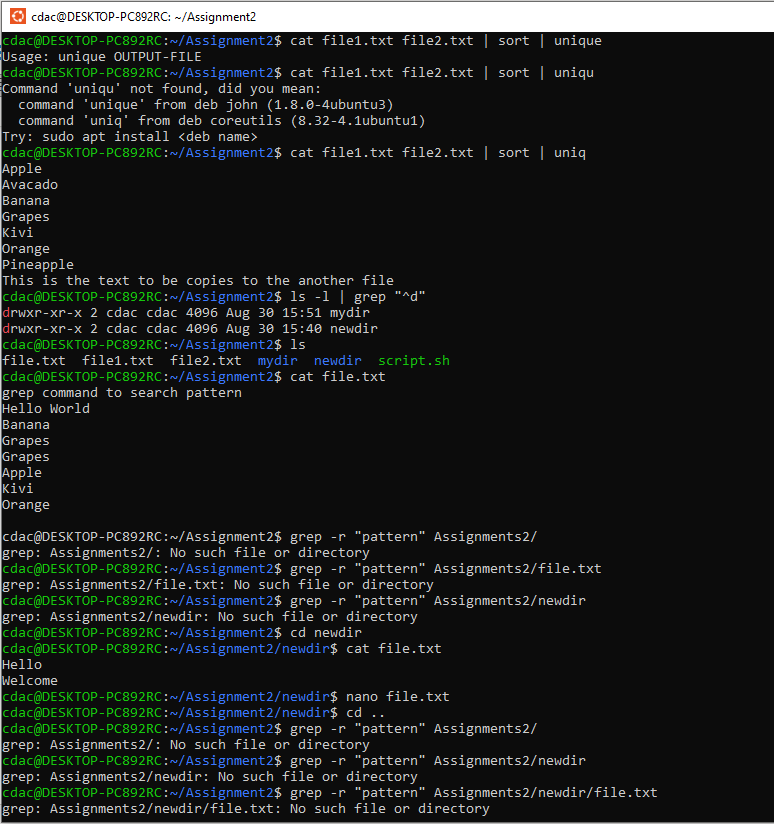
* Adds execute permission to the owner of file.txt

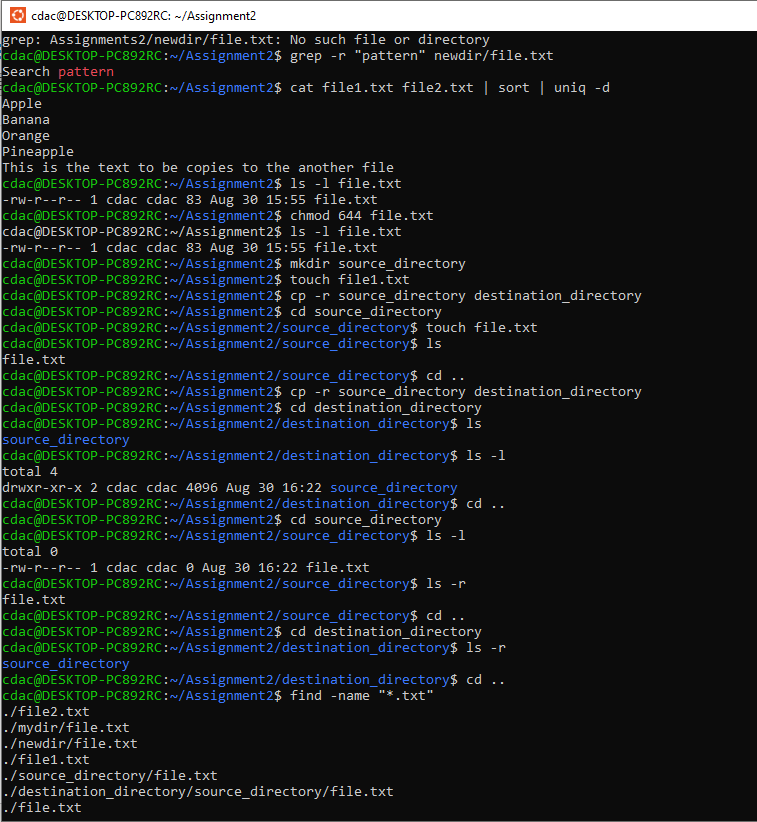
• echo $PATH

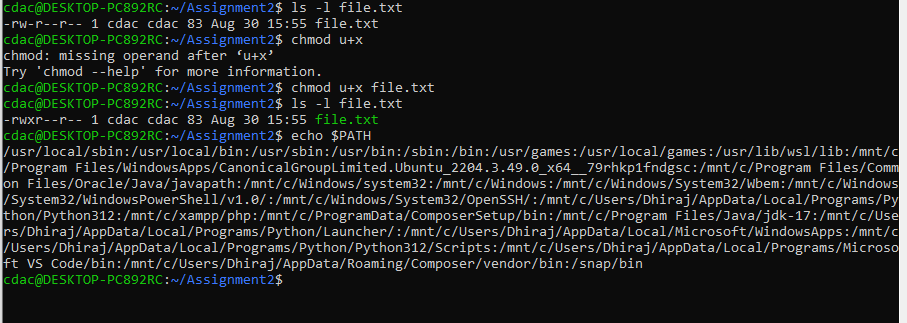
* Lists all the directories where the system looks for executable files.











**Part B**

**Identify True or False:**

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

=> True

2. **mv** is used to move files and directories.

=>True

3. **cd** is used to copy files and directories.

=>False | cd is used to change the directory

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

=>True

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

=>True

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

=>True

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

=>True

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

**=>True**

**Identify the Incorrect Commands:**

1. **chmodx** is used to change file permissions.

=> Incorrect - chmod

2. **cpy** is used to copy files and directories.

=>Incorrect - cp

3. **mkfile** is used to create a new file.

=> Incorrect – touch or cat

4. **catx** is used to concatenate files.

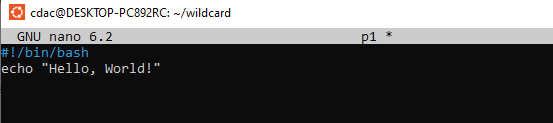
=>Incorrect - cat

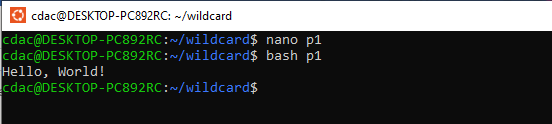
5. **rn** is used to rename files.

=>Incorrect - mv

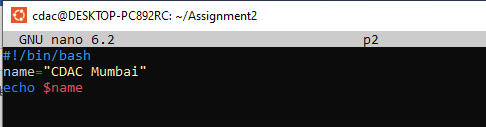
**Part C**

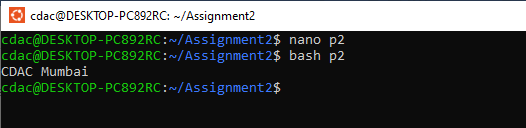
**Question 1:** Write a shell script that prints "Hello, World!" to the terminal.



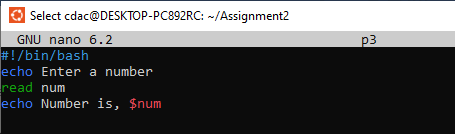


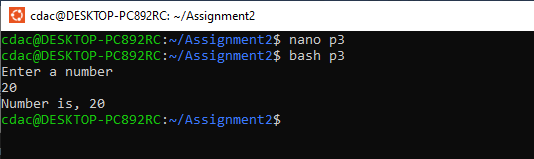
**Question 2:** Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.



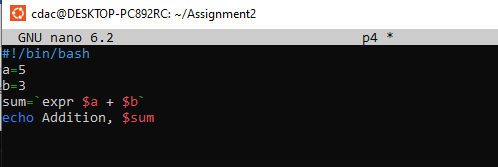


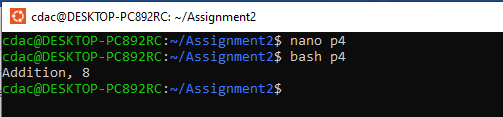
**Question 3:** Write a shell script that takes a number as input from the user and prints it.



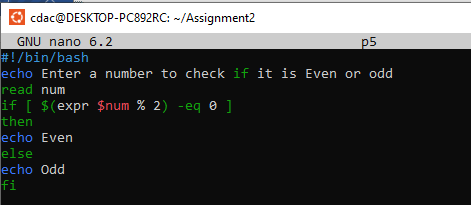


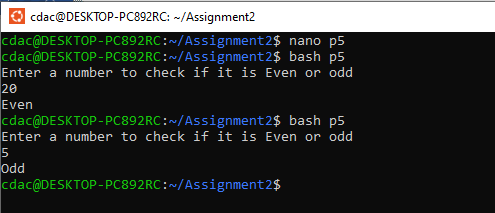
**Question 4:** Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.



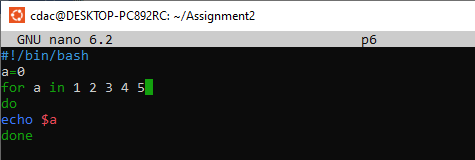


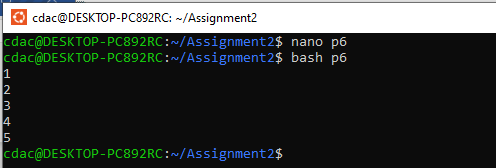
**Question 5:** Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".



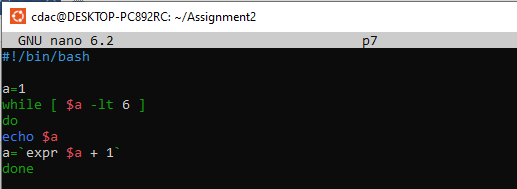


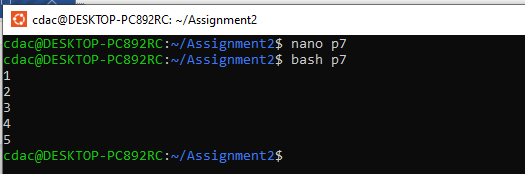
**Question 6:** Write a shell script that uses a for loop to print numbers from 1 to 5.



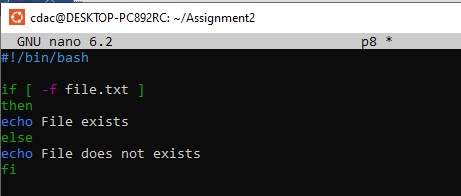


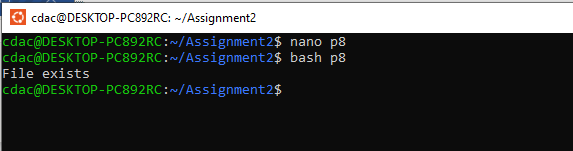
**Question 7:** Write a shell script that uses a while loop to print numbers from 1 to 5.



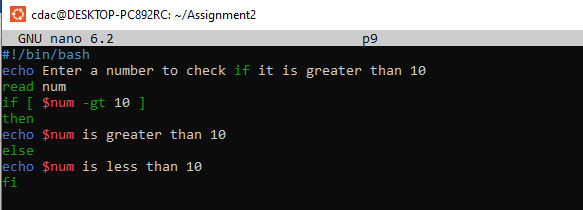


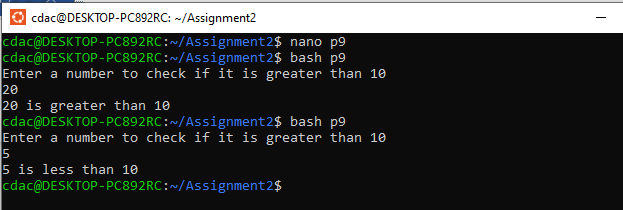
**Question 8:** Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".



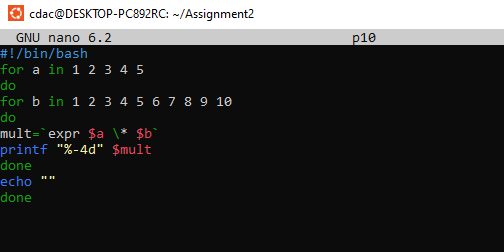


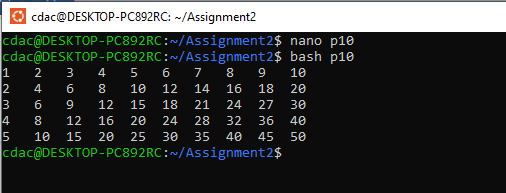
**Question 9:** Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.



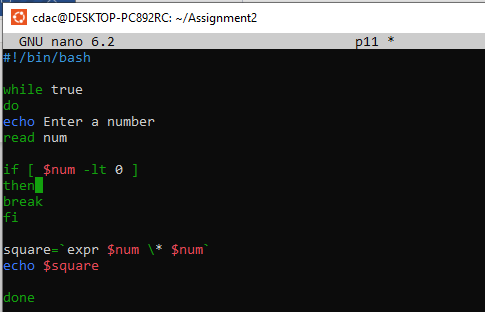


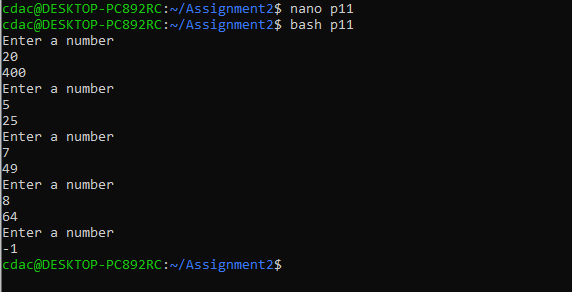
**Question 10:** Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.





**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.





**Part E**

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

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

| P1 | 0 | 5 |

| P2 | 1 | 3 |

| P3 | 2 | 6 |

Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.

=>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | AT | BT | WT | TAT |
| P1 | 0 | 5 | 0 | 5 |
| P2 | 1 | 3 | 4 | 7 |
| P3 | 2 | 6 | 6 | 12 |

P1 P2 P3

0 – 5 – 8 – 14

Average Waiting Time = 10 / 3 = 3.33

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

| Process | Arrival Time | Burst Time | |---------|--------------|------------|

| P1 | 0 | 3 |

| P2 | 1 | 5 |

| P3 | 2 | 1 |

| P4 | 3 | 4 |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | AT | BT | WT | CT | TAT |
| P1 | 0 | 3 | 0 | 3 | 3 |
| P2 | 1 | 5 | 7 | 13 | 12 |
| P3 | 2 | 1 | 1 | 4 | 2 |
| P4 | 3 | 4 | 1 | 8 | 5 |

P1 P3 P4 P2

0 - 3 - 4 - 8 - 13

Average Turn Around Time = (3+12+2+5)/4 = 22/4 = 5.5

3. Consider the following processes with arrival times, burst times, and priorities (lower number

indicates higher priority):

| Process | Arrival Time | Burst Time | Priority | |---------|--------------|------------|----------|

| P1 | 0 | 6 | 3 |

| P2 | 1 | 4 | 1 |

| P3 | 2 | 7 | 4 |

| P4 | 3 | 2 | 2 |

Calculate the average waiting time using Priority Scheduling.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Process | AT | BT | Priority | CT | WT | TAT |
| P1 | 0 | 6 | 3 | 6 | 0 |  |
| P2 | 1 | 4 | 1 | 10 | 5 |  |
| P3 | 2 | 7 | 4 | 16 | 10 |  |
| P4 | 3 | 2 | 2 | 12 | 7 |  |

P1 P2 P4 P3

0 - 6 - 10 - 12 - 16

Average Waiting Time = (0+5+10+7) / 4 = 22/4 = 5.5

4. Consider the following processes with arrival times and burst times, and the time quantum for

Round Robin scheduling is 2 units: | Process | Arrival Time | Burst Time | |---------|--------------|------------|

| P1 | 0 | 4 |

| P2 | 1 | 5 |

| P3 | 2 | 2 |

| P4 | 3 | 3 |

Calculate the average turnaround time using Round Robin scheduling.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | AT | BT | CT | WT | TAT |
| P1 | 0 | 4 | 10 | 6 | 10 |
| P2 | 1 | 5 | 14 | 7 | 13 |
| P3 | 2 | 2 | 6 | 2 | 4 |
| P4 | 3 | 3 | 13 | 7 | 10 |

P1 P2 P3 P4 P1 P2 P4 P2

0 2 4 6 8 10 12 13 14

Average Turn Around Time = (10+13+4+10) / 4 = 37/4 =9.25

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?