Name: Mrunali Kale_KH

CDAC MUMBAI

Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

1. **echo "Hello, World!"**
   This command prints the string "Hello, World!" to the terminal.
2. **name="Productive"**
   This command assigns the value "Productive" to a variable named name. It doesn't display anything unless you reference the variable later.
3. **touch file.txt**
   Creates an empty file named file.txt if it doesn't already exist, or updates its timestamp if it does.
4. **ls -a**
   Lists all files in the current directory, including hidden files (those starting with a dot .).
5. **rm file.txt**
   Deletes the file named file.txt from the current directory.
6. **cp file1.txt file2.txt**
   Copies the contents of file1.txt to a new file called file2.txt.
7. **mv file.txt /path/to/directory/**
   Moves the file file.txt to the specified directory /path/to/directory/.
8. **chmod 755 script.sh**
   Changes the permissions of script.sh to 755, meaning the owner has read, write, and execute permissions, while the group and others have read and execute permissions.
9. **grep "pattern" file.txt**
   Searches for the string "pattern" in file.txt and prints matching lines.
10. **kill PID**
    Terminates the process with the specified Process ID (PID).
11. **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**
    This is a chain of commands executed sequentially:

- Creates a directory named mydir.
- Changes into that directory (cd mydir).
- Creates an empty file file.txt.

- Writes "Hello, World!" into file.txt.
- Displays the contents of file.txt (Hello, World!).

12. **ls -l | grep ".txt"**
    Lists files in long format (ls -l) and filters the output to show only files that have .txt in their name.
13. **cat file1.txt file2.txt | sort | uniq**
    Concatenates the contents of file1.txt and file2.txt, then sorts the combined content and removes duplicate lines using uniq.
14. **ls -l | grep "^d"**
    Lists files in long format and filters the output to show only directories (because directories start with d in the long listing format).
15. **grep -r "pattern" /path/to/directory/**
    Recursively searches for "pattern" in all files within the specified directory /path/to/directory/.
16. **cat file1.txt file2.txt | sort | uniq -d**
    Concatenates the contents of file1.txt and file2.txt, sorts the combined content, and then filters only the duplicate lines using uniq -d.
17. **chmod 644 file.txt**
    Changes the permissions of file.txt to 644, meaning the owner has read and write permissions, and the group and others have read-only permissions.
18. **cp -r source_directory destination_directory**
    Copies the entire directory source_directory and its contents recursively to destination_directory.
19. **find /path/to/search -name "*.txt"**
    Searches for all files with a .txt extension under the directory /path/to/search.
20. **chmod u+x file.txt**
    Adds execute permission for the user (owner) to the file file.txt.
21. **echo $PATH**
    Displays the current system's PATH environment variable, which contains a colon-separated list of directories where executable files are located.

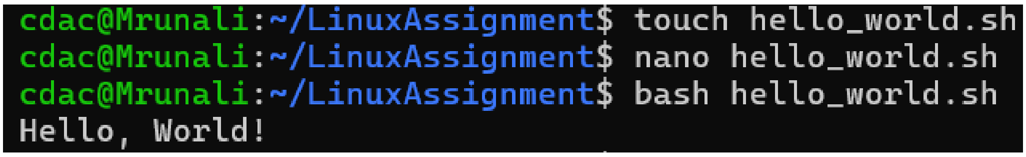
Part – B

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**
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**

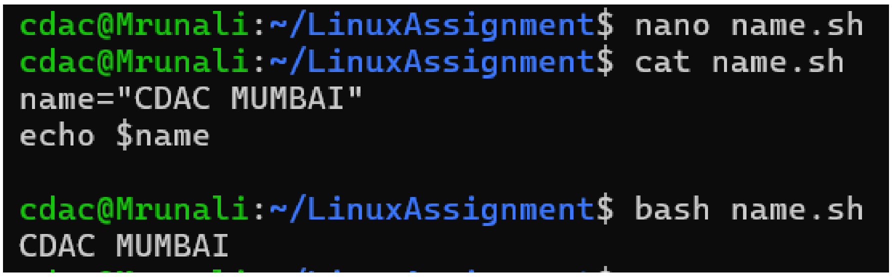
Part – C

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



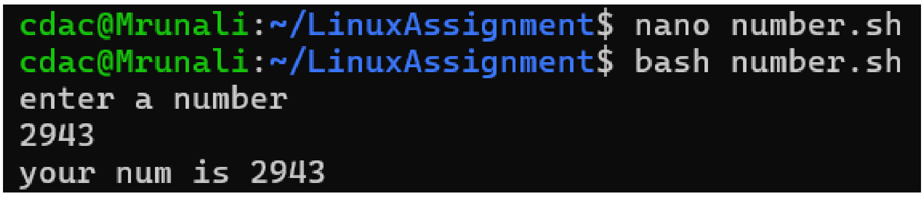
```
cdac@Mrunali:~/LinuxAssignment$ touch hello_world.sh
cdac@Mrunali:~/LinuxAssignment$ nano hello_world.sh
cdac@Mrunali:~/LinuxAssignment$ bash hello_world.sh
Hello, World!
```

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



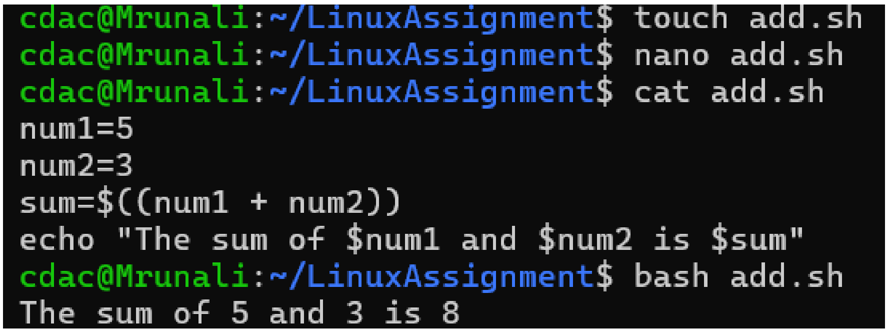
```
cdac@Mrunali:~/LinuxAssignment$ nano name.sh
cdac@Mrunali:~/LinuxAssignment$ cat name.sh
name="CDAC MUMBAI"
echo $name

cdac@Mrunali:~/LinuxAssignment$ bash name.sh
CDAC MUMBAI
```

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



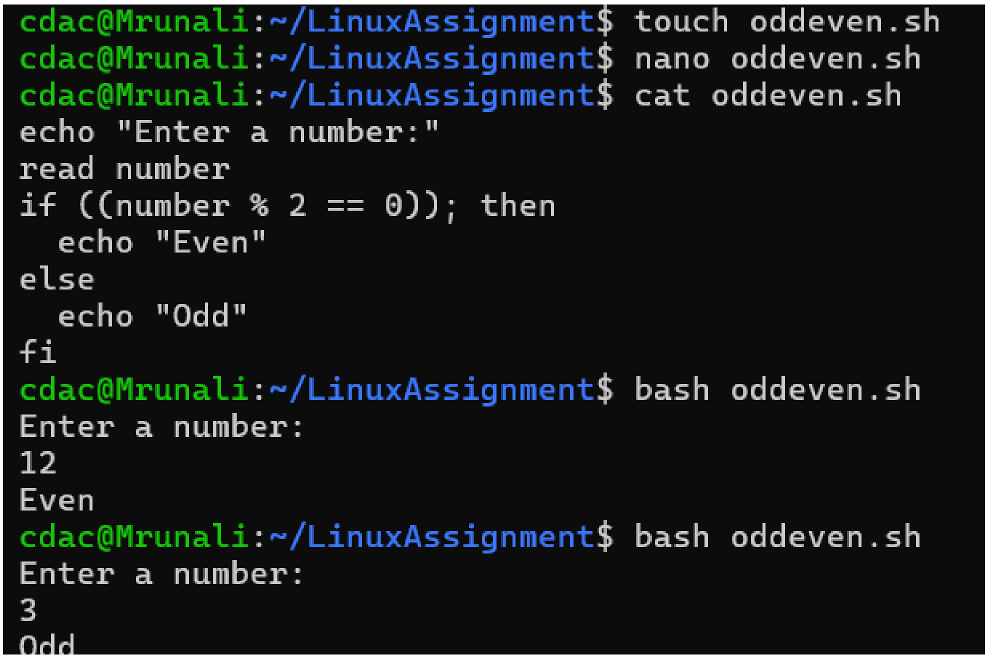
```
cdac@Mrunali:~/LinuxAssignment$ nano number.sh
cdac@Mrunali:~/LinuxAssignment$ bash number.sh
enter a number
2943
your num is 2943
```

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



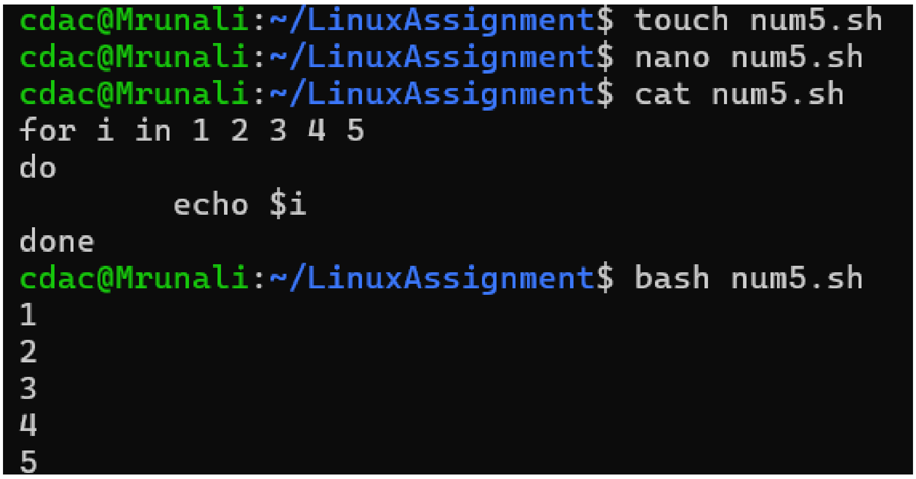
```
cdac@Mrunali:~/LinuxAssignment$ touch add.sh
cdac@Mrunali:~/LinuxAssignment$ nano add.sh
cdac@Mrunali:~/LinuxAssignment$ cat add.sh
num1=5
num2=3
sum=$((num1 + num2))
echo "The sum of $num1 and $num2 is $sum"
cdac@Mrunali:~/LinuxAssignment$ bash add.sh
The sum of 5 and 3 is 8
```

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



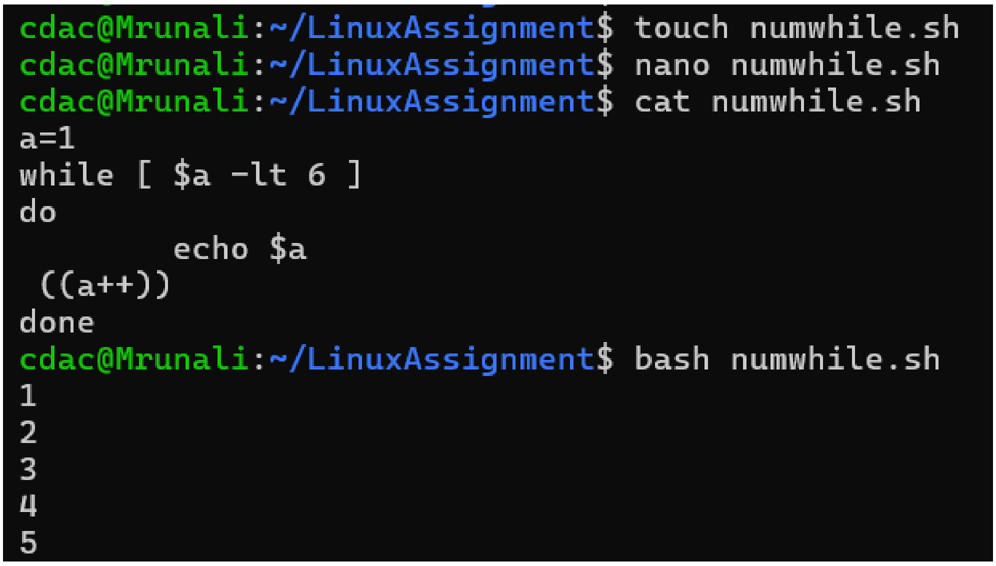
```
cdac@Mrunali:~/LinuxAssignment$ touch oddeven.sh
cdac@Mrunali:~/LinuxAssignment$ nano oddeven.sh
cdac@Mrunali:~/LinuxAssignment$ cat oddeven.sh
echo "Enter a number:"
read number
if ((number % 2 == 0)); then
  echo "Even"
else
  echo "Odd"
fi
cdac@Mrunali:~/LinuxAssignment$ bash oddeven.sh
Enter a number:
12
Even
cdac@Mrunali:~/LinuxAssignment$ bash oddeven.sh
Enter a number:
3
Odd
```

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



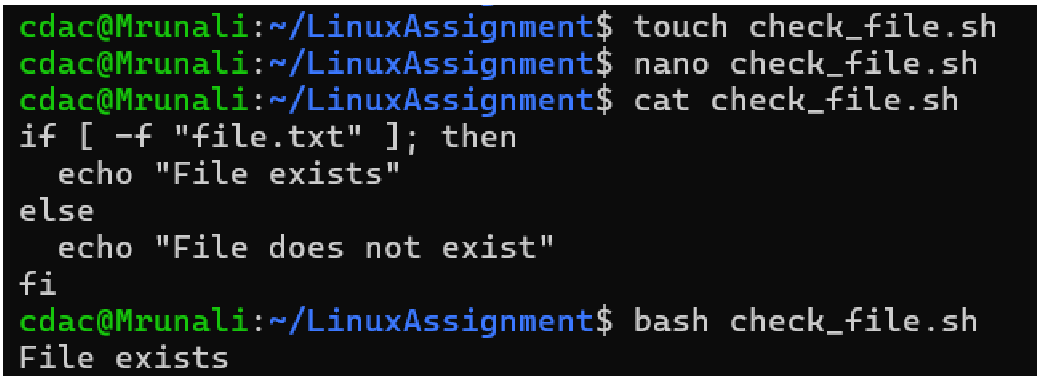
```
cdac@Mrunali:~/LinuxAssignment$ touch num5.sh
cdac@Mrunali:~/LinuxAssignment$ nano num5.sh
cdac@Mrunali:~/LinuxAssignment$ cat num5.sh
for i in 1 2 3 4 5
do
        echo $i
done
cdac@Mrunali:~/LinuxAssignment$ bash num5.sh
1
2
3
4
5
```

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



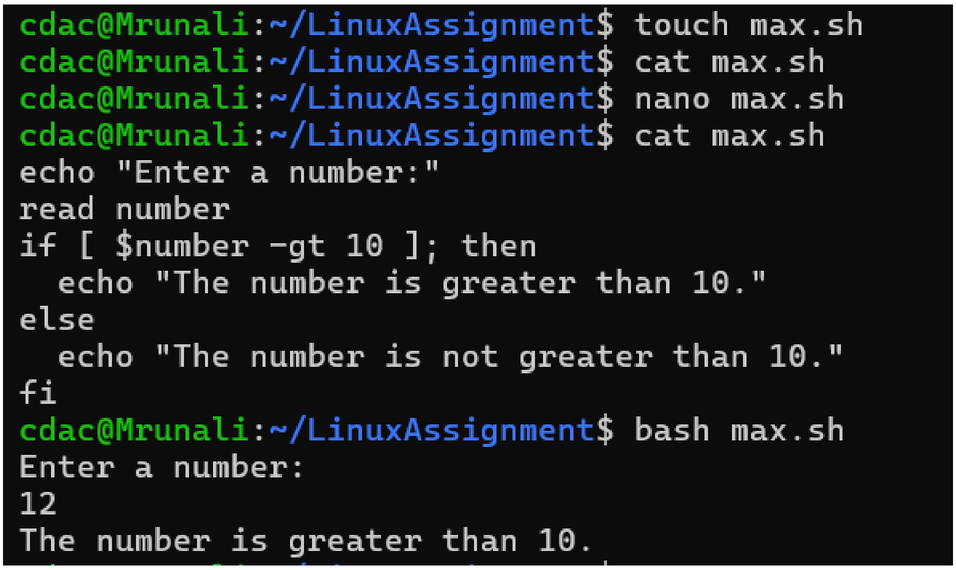
```
cdac@Mrunali:~/LinuxAssignment$ touch numwhile.sh
cdac@Mrunali:~/LinuxAssignment$ nano numwhile.sh
cdac@Mrunali:~/LinuxAssignment$ cat numwhile.sh
a=1
while [ $a -lt 6 ]
do
        echo $a
 ((a++))
done
cdac@Mrunali:~/LinuxAssignment$ bash numwhile.sh
1
2
3
4
5
```

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".



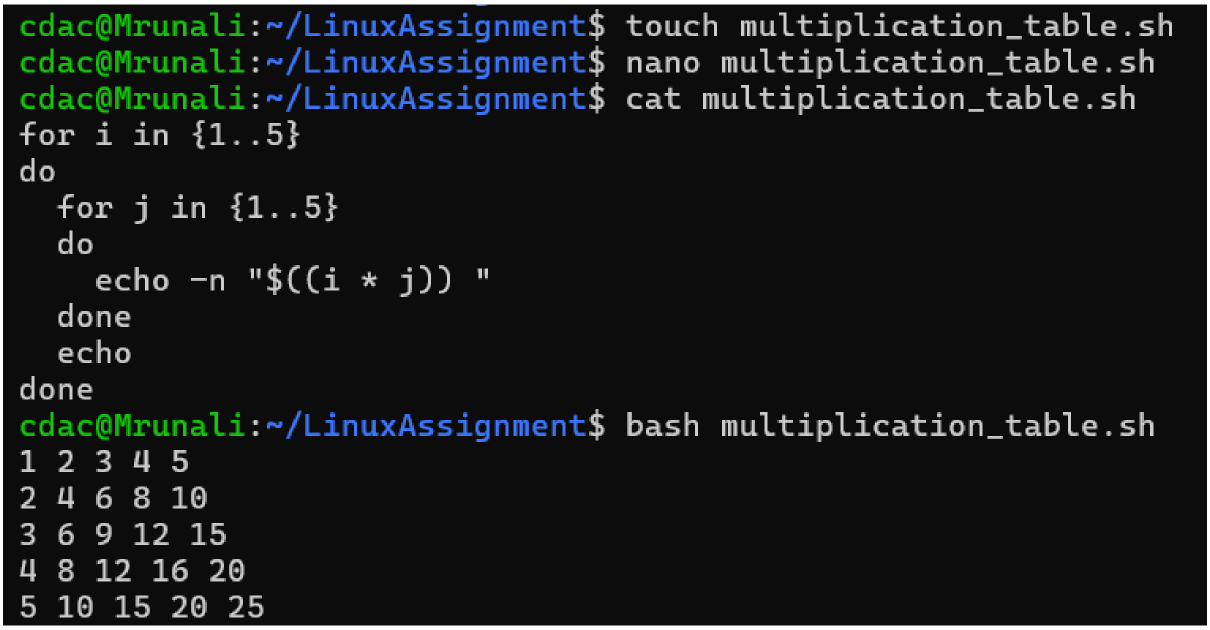
```
cdac@Mrunali:~/LinuxAssignment$ touch check_file.sh
cdac@Mrunali:~/LinuxAssignment$ nano check_file.sh
cdac@Mrunali:~/LinuxAssignment$ cat check_file.sh
if [ -f "file.txt" ]; then
  echo "File exists"
else
  echo "File does not exist"
fi
cdac@Mrunali:~/LinuxAssignment$ bash check_file.sh
File exists
```

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



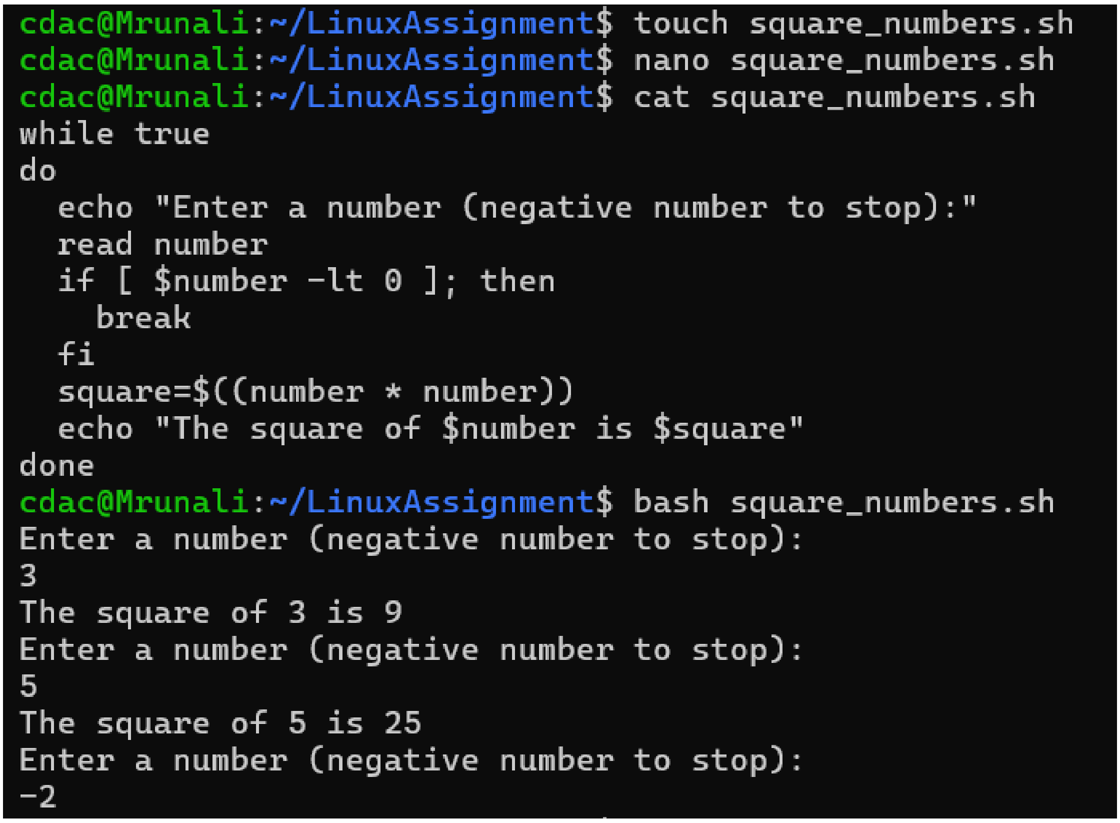
```
cdac@Mrunali:~/LinuxAssignment$ touch max.sh
cdac@Mrunali:~/LinuxAssignment$ cat max.sh
cdac@Mrunali:~/LinuxAssignment$ nano max.sh
cdac@Mrunali:~/LinuxAssignment$ cat max.sh
echo "Enter a number:"
read number
if [ $number -gt 10 ]; then
  echo "The number is greater than 10."
else
  echo "The number is not greater than 10."
fi
cdac@Mrunali:~/LinuxAssignment$ bash max.sh
Enter a number:
12
The number is greater than 10.
```

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.



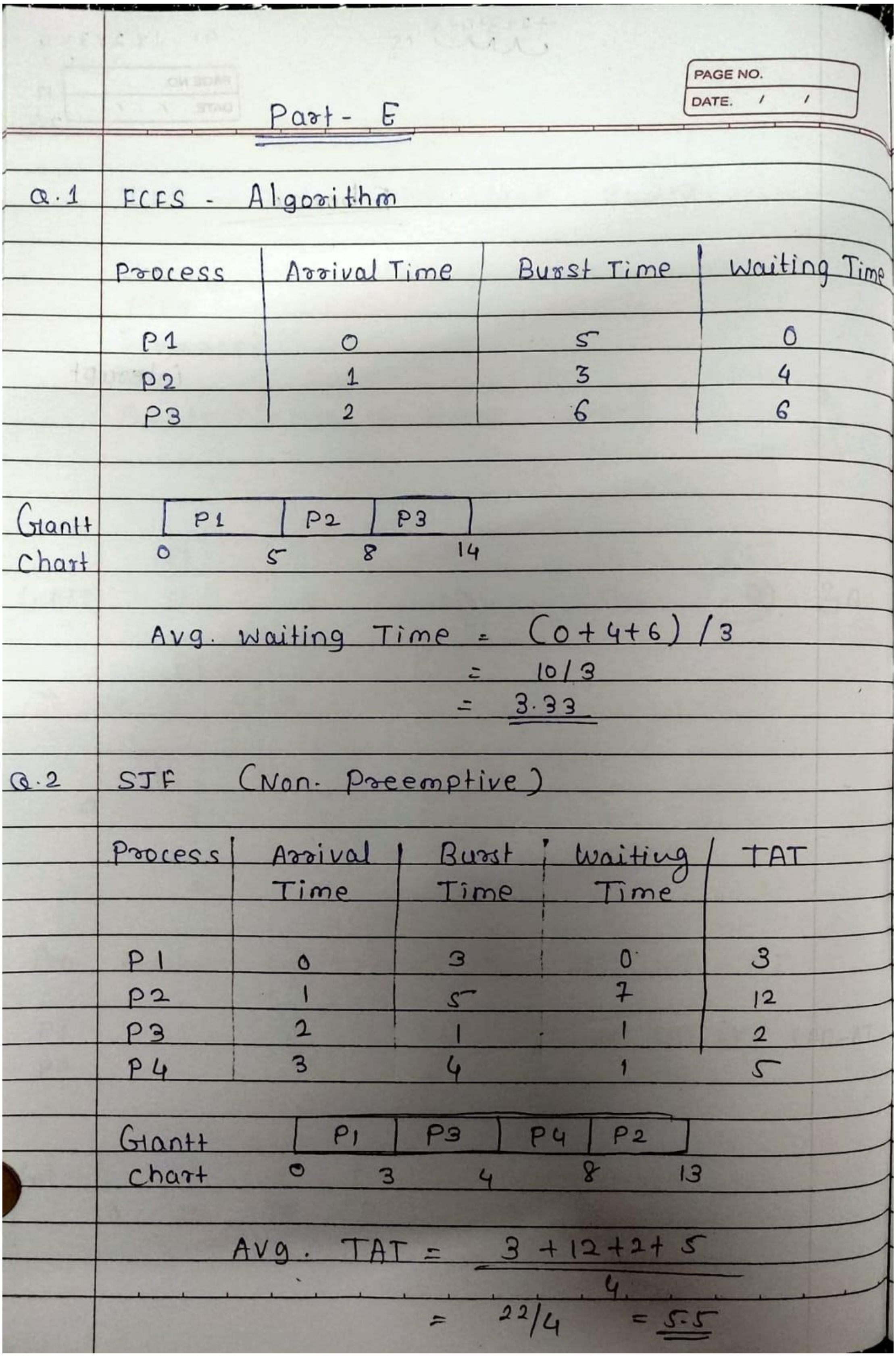
```
cdac@Mrunali:~/LinuxAssignment$ touch multiplication_table.sh
cdac@Mrunali:~/LinuxAssignment$ nano multiplication_table.sh
cdac@Mrunali:~/LinuxAssignment$ cat multiplication_table.sh
for i in {1..5}
do
  for j in {1..5}
  do
    echo -n "$((i * j)) "
  done
  echo
done
cdac@Mrunali:~/LinuxAssignment$ bash multiplication_table.sh
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
```

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.



```
cdac@Mrunali:~/LinuxAssignment$ touch square_numbers.sh
cdac@Mrunali:~/LinuxAssignment$ nano square_numbers.sh
cdac@Mrunali:~/LinuxAssignment$ cat square_numbers.sh
while true
do
  echo "Enter a number (negative number to stop):"
  read number
  if [ $number -lt 0 ]; then
    break
  fi
  square=$((number * number))
  echo "The square of $number is $square"
done
cdac@Mrunali:~/LinuxAssignment$ bash square_numbers.sh
Enter a number (negative number to stop):
3
The square of 3 is 9
Enter a number (negative number to stop):
5
The square of 5 is 25
Enter a number (negative number to stop):
-2
```

Part – E



PAGE NO.
DATE. / /

Part- E

Q.1 FCFS - Algorithm

Process	Arrival Time	Burst Time	Waiting Time
P1	0	5	0
P2	1	3	4
P3	2	6	6

Gantt Chart

P1	P2	P3

0 5 8 14

Avg. waiting Time = (0+4+6)/3

= 10/3

= 3.33

Q.2 SJF (Non-Preemptive)

Process	Arrival Time	Burst Time	Waiting Time	TAT
P1	0	3	0	3
P2	1	5	7	12
P3	2	1	1	2
P4	3	4	1	5

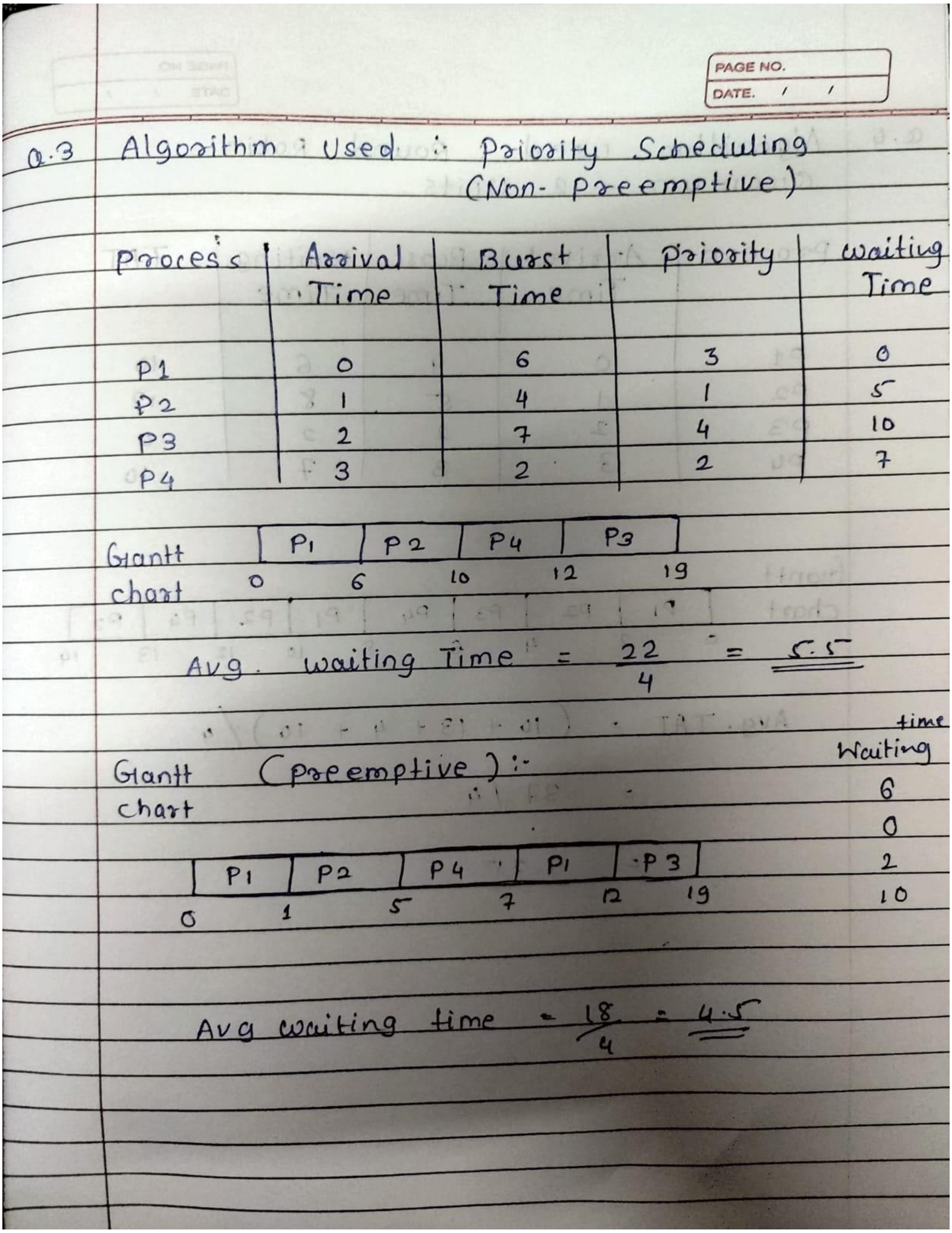
Gantt chart

P1	P3	P4	P2

0 3 4 8 13

$$\text{Avg. TAT} = \frac{3+12+2+5}{4}$$

$$= 22/4 = \underline{5.5}$$



PAGE NO.
DATE. / /

Q.3 Algorithm Used : Priority Scheduling (Non- preemptive)

Process	Arrival Time	Burst Time	priority	waiting Time
P1	0	6	3	0
P2	1	4	1	5
P3	2	7	4	10
P4	3	2	2	7

Gantt chart

P1	P2	P4	P3

0 6 10 12 19

Avg. waiting Time = $\frac{22}{4}$ = 5.5

Gantt chart (preemptive) :-

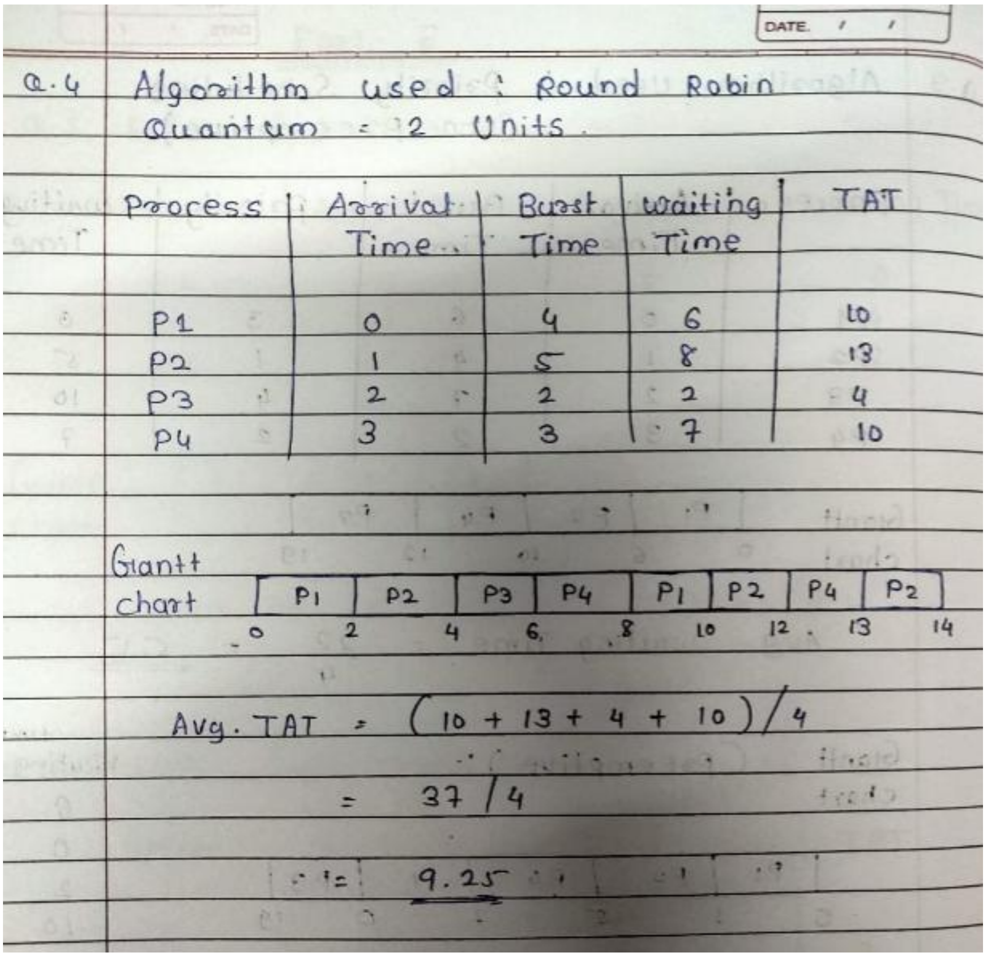
P1	P2	P4	P1	P3

0 1 5 7 12 19

Waiting time
6
0
2
10

Avg waiting time = $\frac{18}{4}$ = 4.5

Name: Mrunali Kale_KH



DATE. / /

Q.4 Algorithm used : Round Robin
Quantum = 2 Units.

Process	Arrival Time	Burst Time	waiting Time	TAT
P1	0	4	6	10
P2	1	5	8	13
P3	2	2	2	4
P4	3	3	7	10

Gantt chart

P1	P2	P3	P4	P1	P2	P4	P2

0 2 4 6 8 10 12 13 14

Avg. TAT = (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?

- When the fork() system call is used, it creates a child process that has its own copy of the parent's memory.
- Before forking, the parent has a variable x = 5. After the fork, both the parent and child have separate copies of x, still equal to 5.
- Each process then increments x by 1, so both the parent and child have x = 6, but in their own separate memory.
- In parent process, x=6. In child process, x=6