CDAC MUMBAI

Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

- **echo "Hello, World!"**

  This will print the Output: Hello. World!

- **name="Productive"**

  It will assign the value “Productive” to the variable name, and it will remember it to use when needed.

- **touch file.txt**

  This will create an empty file named file1.txt if it doesn’t exist.

  If the file already exists, it updates the files timestamp without altering the contents.

- **ls -a**

  List all the files and directories in the current directory, including the hidden ones(files starting with .)

- **rm file.txt**

  It will remove/delete the file.txt named file.

  This action is permanent and cannot be undone.

- **cp file1.txt file2.txt**

  This command will copy the contents of file1.txt to file2.txt

- **mv file.txt /path/to/directory/**

  Mv is used for rename or move a file.

  It will move the file named file.txt to the specified directory.

- **chmod 755 script.sh**

  Read only – 4

  Write only – 2

  Execute only – 1

  It changes the permissions of script.sh as:

  Owner – Read, write, and execute (7) - rwx

  Group – Read and execute (5) - rx

  Others – Read and execute (5) – rx

- **grep "pattern" file.txt**

  It searches the specified name: “pattern” in file.txt and displays the matching lines present in the file.txt

- **kill PID**

  Terminates the process with the given Process ID (PID). Since the above command doesn’t contain any process id, it will result in error.

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

  This is a series of commands that:

  Makes a directory called mydir.

  Walks into mydir.

Makes an empty file called file.txt.

Writes "Hello, World!" to file.txt.

Prints the contents of file.txt.

Output: Hello, World! .

- **ls -l | grep ".txt"**

Displays files with full details and shows only those containing.txt in their names.

- **cat file1.txt file2.txt | sort | uniq**

Function: Merges the lines of file1.txt and file2.txt, sorts the lines, and deletes duplicate lines.

- **ls -l | grep "^d"**

Function: Displays all the directories in the present location.
Explanation: Directories begin with a d in the long list format (-l).

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

Function: Searches recursively for "pattern" in all files under the given directory

- **cat file1.txt file2.txt | sort | uniq –d**

Function: Merges and sorts the contents of file1.txt and file2.txt and shows only duplicate lines.

- **chmod 644 file.txt**

Function: Assigns permissions on file.txt as:
Owner: Read and write (6)
Group: Read-only (4)
Others: Read-only (4)

- **cp -r source_directory destination_directory**

  Function: Recursively copies the source_directory and its contents to destination_directory.

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

  Function: Finds all .txt files in the given directory and its subdirectories

- **chmod u+x file.txt**

  Function: Permission to execute on owner (u) of file.txt.

- **echo $PATH**

  Function: Outputs the current system's PATH environment variable, listing directories where executable programs/files are searched.

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
   It 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.
   FALSE
   -r(recursive option) is used for directories, not files

Identify the Incorrect Commands:

1. **chmodx** is used to change file permissions.
   Incorrect
   Answer - chmod

2. **cpy** is used to copy files and directories.
   Incorrect
   Answer - cp

3. **mkfile** is used to create a new file.
   Incorrect
   Answer - touch

4. **catx** is used to concatenate files.
   Incorrect
   Answer - cat

5. **rn** is used to rename files.
   Incorrect
   Answer – mv

Part C

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

echo "Hello, World!"

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ nano file1.txt
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file1.txt
echo "Hello, World!"
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file1.txt
Hello, World!
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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

name="CDAC Mumbai"

echo "The value of name is: $name"

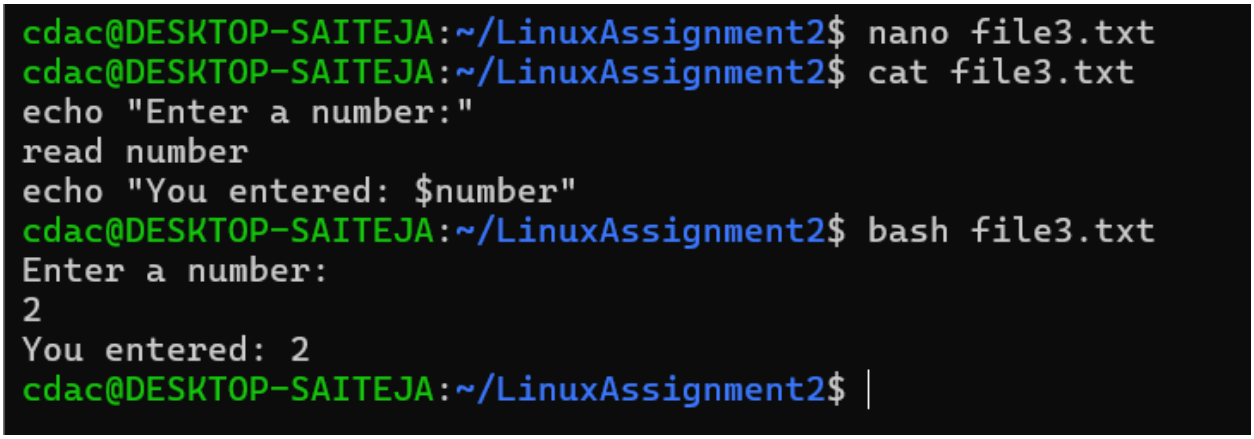
```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ nano file2.txt
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file2.txt
name="CDAC Mumbai"
echo $name
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file2.txt
CDAC Mumbai
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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

```
echo "Enter a number:"

read number

echo "You entered: $number"
```



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

```
echo "Enter a number"

read num1

echo "Enter a number"

read num2

sum=`expr $num1 + $num2`

echo sum of $num1 and $num2 is $sum
```

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ nano file4.txt
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file4.txt
echo "Enter a number"
read num1
echo "Enter a number"
read num2
sum=`expr $num1 + $num2`
echo sum of $num1 and $num2 is $sum
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file4.txt
Enter a number
2
Enter a number
3
sum of 2 and 3 is 5
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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

```
echo "Enter a number:"
read number
if [ `expr $number % 2` -eq 0 ]
then
    echo "$number is an even number"
else
    echo "$number is an odd number"
fi
```

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ nano file5.txt
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file5.txt
echo "Enter a number:"
read number
if [ `expr $number % 2` -eq 0 ]
then
        echo "$number is an even number"
else
        echo "$number is an odd number"
fi
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file5.txt
Enter a number:
4
4 is an even number
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file5.txt
Enter a number:
3
3 is an odd number
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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

for i in {1..5}

do

  echo $i

done

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file6.txt
for i in {1..5}
do
        echo $i
done
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file6.txt
1
2
3
4
5
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

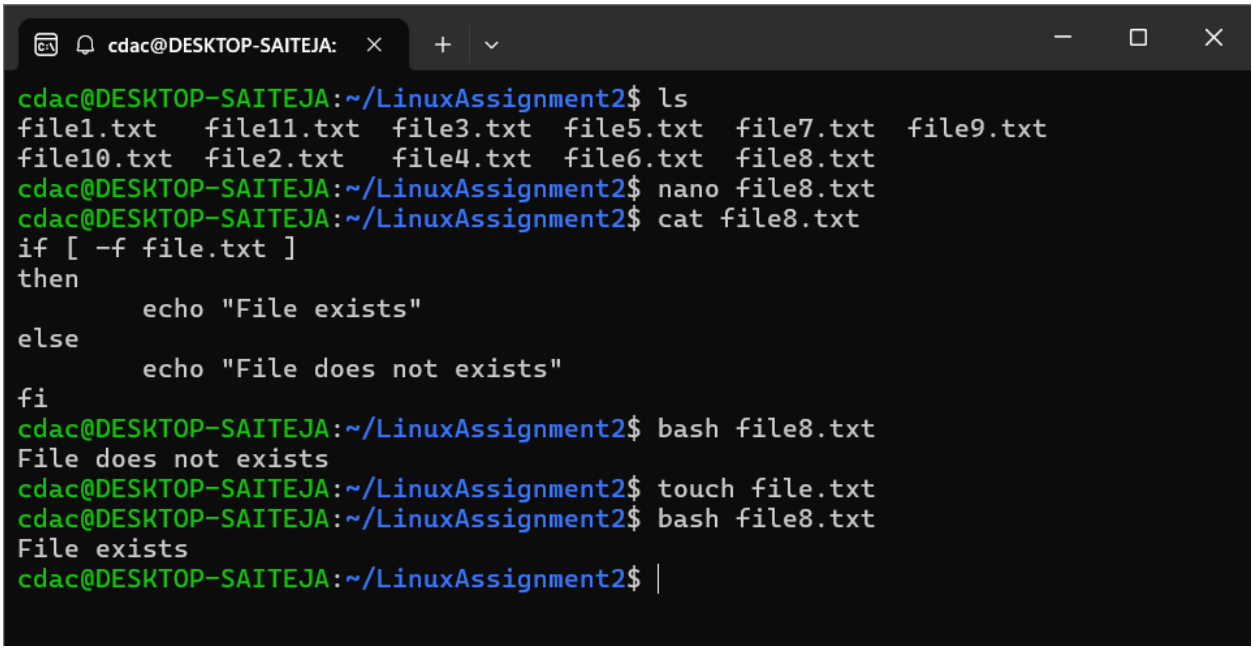
Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.

```
i=1
while [ $i -le 5 ]
do
   echo $i
   i=`expr $i + 1`
done
```

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file7.txt
i=1
while [ $i -le 5 ]
do
        echo $i
        i=`expr $i + 1`
done
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file7.txt
1
2
3
4
5
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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".

```
if [ -f file.txt ]; then
   echo "File exists"
else
   echo "File does not exist"
fi
```



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.

```
echo "Enter a number:"
read number
if [ $number -gt 10 ]
then
    echo "$number is greater than 10"
else
    echo "$number is less than or equal than 10"
fi
```

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ nano file9.txt
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file9.txt
echo "Enter a number:"
read number
if [ $number -gt 10 ]
then
        echo "$number is greater than 10"
else
        echo "$number is less than or equal than 10"
fi
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file9.txt
Enter a number:
2
2 is less than or equal than 10
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file9.txt
Enter a number:
22
22 is greater than 10
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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
        result=`expr $i \* $j`
        echo -n "$result      "
    done
    echo
done
```

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ nano file10.txt
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file10.txt
for i in {1..5}
do
        for j in {1..5}
        do
                result=`expr $i \* $j`
                echo -n "$result         "
        done
        echo
done
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file10.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@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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.

```
while [ true ]
do
    echo "Enter a number:"
    read number
    if [ $number -lt 0 ]
    then
        echo "Negative number entered. Exiting..."
        break
    fi
    square=$((number * number))
    echo "Square of $number is: $square"
done
```

```
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ nano file11.txt
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ cat file11.txt
while [ true ]
do
        echo "Enter a number:"
        read number
        if [ $number -lt 0 ]
        then
                echo "Negative number entered. Exiting..."
                break
        fi
        square=$((number * number))
        echo "Square of $number is: $square"
done
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$ bash file11.txt
Enter a number:
2
Square of 2 is: 4
Enter a number:
5
Square of 5 is: 25
Enter a number:
-3
Negative number entered. Exiting...
cdac@DESKTOP-SAITEJA:~/LinuxAssignment2$
```

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.

P1 P2 P3

0 5 8 14

Process	Starting Time (ST)	Arrival Time (AT)	Waiting Time (WT)
P1	0	0	0
P2	5	1	4
P3	8	2	6

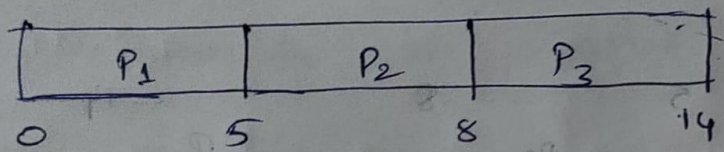
WT = ST – AT

AVG WT = (0+4+6)/3 = 10/3 = 3.33

Q1) FCFS

Process	Arrival time	Burst time	Waiting time
P_1	0	5	0
P_2	1	3	4
P_3	2	6	6

Gantt Chart :-



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

= 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.

P1 P3 P4 P2
0 3 4 8 13

Process	**Arrival Time**	**Completion Time**	**Turnaround Time**
P1	0	3	3
P2	1	13	12
P3	2	4	2
P4	3	8	5

TAT = CT - AT

Calculate Average Turnaround Time:

Average Turnaround Time= 3+12+2+5/4 = 22/4 = 5.5

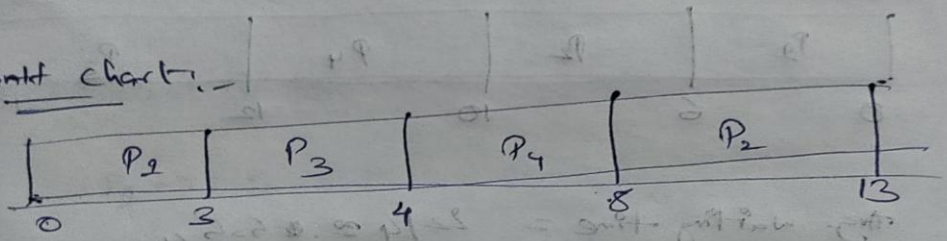
So, the average turnaround time is **5.5 units**.

Q) ~~algorithm~~

Q) SJF (non-preemptive)

Process	Arrival time	Burst time	Waiting time	TurnAround time
P_1	0	3	0	3
P_2	1	5	7	12
P_3	2	1	1	2
P_4	3	4	1	5

Gantt chart:-



Avg. Turn Around time (TAT) $= \frac{3+12+2+5}{4} = 22/4 = 5.5$ //

TAT = CT - AT

CT - Completion time, AT = Arrival time

P	CT	AT	TAT = CT - AT
P_1	3	0	3
P_2	13	1	12
P_3	4	2	2
P_4	8	3	5

3. Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority):

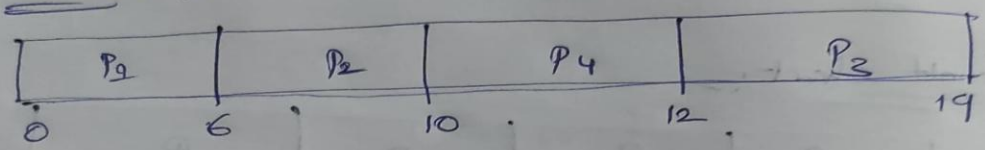
Process	Arrival Time	Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

Calculate the average waiting time using Priority Scheduling.

Eg) Priority Scheduling (non-preemptive):

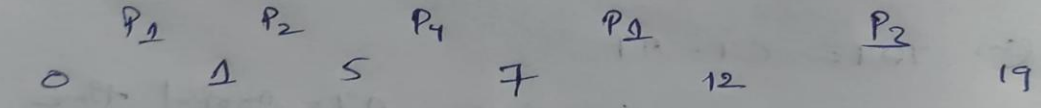
Process	Arrival time	Burst time	priority	Waiting time
P_1	0	6	3	0
P_2	1	4	1	5
P_3	2	7	4	10
P_4	3	2	2	7

Gantt chart :-



Avg. waiting time = 22/4 ≈ 5.5 ms

Gantt chart (preemptive):-



Waiting time	Process
6	P_1
0	P_2
10	P_3
2	P_4

Avg. waiting time
= 18/4
= 4.5 ms

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.

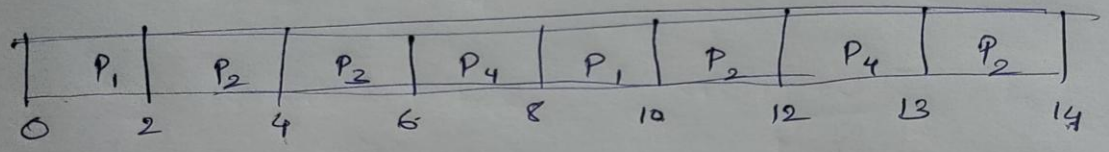
Q4) Round Robin (RR)

Quantum = 2 units

Process	Arrival time	Burst time	Waiting time	TurnAround time
P_1	0	4	6	10
P_2	1	5	8	13
P_3	2	2	2	4
P_4	3	3	7	10

Gantt chart :- CPU not kept idle.

$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_2$



Avg. TurnAround time = (10+13+4+10)/4

= 37/4 = 9.25

(or)

Process	Completion time	Arrival time	TAT (CT−AT)
P_1	10	0	10
P_2	14	1	13
P_3	6	2	4
P_4	13	3	10

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?

   After the fork() system call, both the parent and child processes will have their own separate copies of the variable x. Both increment their x by 1.
   Final Values:

- **Parent Process:** x = 6
- **Child Process:** x = 6