Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

- echo "Hello, World!"
  This will display string Hello, World!

- name="Productive"
  variable name is assigned the value Productive

- touch file.txt
  Creates a new file in a directory named file.txt

- ls -a
  Lists all the files and directories including hidden ones

- rm file.txt
  Removes the file - file.txt

- cp file1.txt file2.txt
  Copies file (file1.txt) and paste it as file2.txt

- mv file.txt /path/to/directory/
  Moves file.txt to the directory

- chmod 755 script.sh
  The chmod 755 command gives the owner full permissions while others can only read & execute it.

- grep "pattern" file.txt
  Searches for the string "pattern" in file.txt and displays the lines that contain it.

- kill PID
  Sends a termination signal to the process with the process ID PID. This will terminate the process.

- mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
  Creates a directory named mydir
  Changes into mydir directory
  Creates a new empty file file.txt
  Writes Hello, World! Into file.txt
  Display the content of file.txt

- ls -l | grep ".txt"

Lists all files & directories in the current directory in long format then filters the o/p to show only lines containing .txt

- cat file1.txt file2.txt | sort | uniq
  Concatenates the contents of file1.txt & file2.txt
  Sorts the combined output
  Filters out duplicate lines and showing only unique lines

- ls -l | grep "^d"
  Lists all files n directories in the current directory in long format then filters the output to show only directories (lines starting with d)

- grep -r "pattern" /path/to/directory/
  Searches for the string "pattern" in all files within /path/to/directory/ and also its subdirectories

- cat file1.txt file2.txt | sort | uniq –d
  Concatenates the contents of file1.txt & file2.txt
  Sorts the combined output
  The -d option only prints duplicate lines

- chmod 644 file.txt
  Changes the permissions of file.txt to 644 means the owner can read n write the file while others can only read it.

- cp -r source_directory destination_directory
  Copies the directory source_directory and its contents to the destination_directory

- find /path/to/search -name "*.txt"
  Searches for files with the .txt extension within /path/to/search and its subdirectories

- chmod u+x file.txt
  Adds execute permissions for the owner (user) of file.txt

- echo $PATH
  Allows us to view the current value of the $PATH variable in a Linux system
  Finds which directories our shell is set to check 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

→ used to change 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 because -p option ensures that directory1 is created first followed by directory2 inside it

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

→ rm -f file.txt deletes file. txt without asking for confirmation.

Identify the Incorrect Commands:

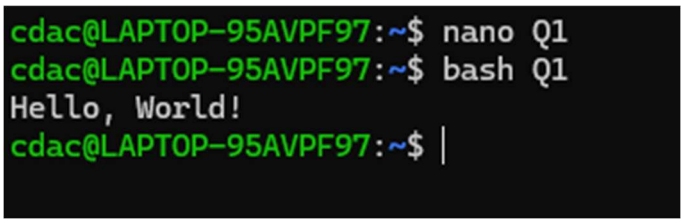
1. chmodx: The correct command is chmod. chmod is used to change file permissions.
2. cpy: The correct command is cp. cp is used to copy files and directories.
3. mkfile: The correct command is touch or echo (for creating an empty file). mkfile is not a standard command for creating files.
4. catx: The correct command is cat. cat is used to concatenate and display the contents of files.
5. rn: The correct command is mv. mv is used to rename files and also to move files and directories.

Part C

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

#!/bin/bash

echo "Hello, World!"



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

#!/bin/bash

name="CDAC Mumbai"

echo "$name"

```
cdac@LAPTOP-95AVPF97:~$ nano Q2
cdac@LAPTOP-95AVPF97:~$ bash Q2
CDAC Mumbai
```

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

```
echo "Enter number:"
read num
echo $num
```

```
cdac@LAPTOP-95AVPF97:~$ nano Q3
cdac@LAPTOP-95AVPF97:~$ bash Q3
Enter number:
21
21
```

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

```
#!/bin/bash
a=5
b=3
sum=`expr $a + $b`
echo $sum
```

```
cdac@LAPTOP-95AVPF97:~$ nano Q4
cdac@LAPTOP-95AVPF97:~$ bash Q4
8
```

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 num
if [ $((num % 2)) -eq 0 ]
then
   echo "$num is an even number"
else
   echo "$num is an odd number"
fi
```

```
cdac@LAPTOP-95AVPF97:~$ nano Q5
cdac@LAPTOP-95AVPF97:~$ bash Q5
Enter a number:
6
 6 is an even number
```

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

```
a=0

for a in 1 2 3 4 5

do

echo "Helloo!!"

echo $a

done
```

```
cdac@LAPTOP-95AVPF97:~$ nano Q
cdac@LAPTOP-95AVPF97:~$ bash Q
1
2
3
4
5
```

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

```
a=1

while [ $a -le 5 ]

do

echo $a

a=$((a + 1))

done
```

```
cdac@LAPTOP-95AVPF97:~$ nano Q7
cdac@LAPTOP-95AVPF97:~$ bash Q7
1
2
3
4
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.

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

```
cdac@LAPTOP-95AVPF97:~$ nano Q9
cdac@LAPTOP-95AVPF97:~$ bash Q9
 Enter a number :
8
 8 is less than 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.

```
echo "Enter the number:"
read num
i=1
while [ $i -le 10 ]
do
res=`expr $i \* $num`
echo "$num * $i = $res"
((++i))
done
```

```
cdac@LAPTOP-95AVPF97:~$ nano Q10
cdac@LAPTOP-95AVPF97:~$ bash Q10
Enter the number:
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
```

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.

➔

End Time for :

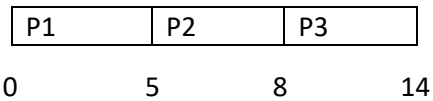
P1 = 5, P2 = 8, P3 = 14

Waiting Time Calculation:

- P1: Waiting time = Start Time - Arrival Time = 0 - 0 = 0
- P2: Waiting time = Start Time - Arrival Time = 5 - 1 = 4
- P3: Waiting time = Start Time - Arrival Time = 8 - 2 = 6

Average Waiting Time = 30+4+6= 10/3 = **3.33**

Sequence of Execution:



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.

➔

End Time for :

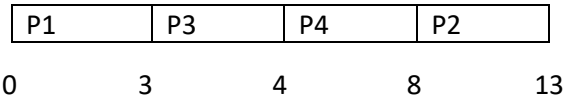
P1 = 3, P3 = 4, P4 = 8, P2 = 13

Turnaround Time Calculation:

- P1: Turnaround time = End Time - Arrival Time = 3 - 0 = 3
- P3: Turnaround time = End Time - Arrival Time = 4 - 2 = 2
- P4: Turnaround time = End Time - Arrival Time = 8 - 3 = 5
- P2: Turnaround time = End Time - Arrival Time = 13 - 1 = 12

Average Turnaround Time= 43+2+5+12 = 22/4 = **5.5**

Sequence of Execution:



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.

➔

The Priority Order for processes to execute :

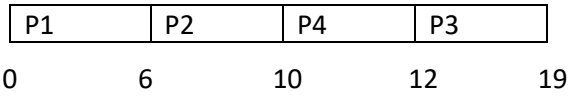
Time	Process Execute
0 – 6	P1
6 – 10	P2
10 – 12	P4
12 – 19	P3

Calculating Waiting Time:

- P1: Waiting time = Start Time - Arrival Time = 0 - 0 = 0
- P2: Waiting time = Start Time - Arrival Time = 6 - 1 = 5
- P4: Waiting time = Start Time - Arrival Time = 10 - 3 = 7
- P3: Waiting time = Start Time - Arrival Time = 12 - 2 = 10

Average Waiting Time = 40+5+7+10 = 22/4 = **5.5**

Sequence of Execution:



0 6 10 12 19

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.

➔

Time	**Action**	**Remaining Burst Times**
0-2	P1 executes	P1: 2, P2: 5, P3: 2, P4: 3
2-4	P2 executes	P1: 2, P2: 3, P3: 2, P4: 3
4-6	P3 executes	P1: 2, P2: 3, P3: 0, P4: 3
6-8	P4 executes	P1: 2, P2: 3, P3: 0, P4: 1
8-10	P1 executes	P1: 0, P2: 3, P3: 0, P4: 1

Step-by-Step Execution:

10-12	P2 executes	P1: 0, P2: 1, P3: 0, P4: 1
12-13	P4 executes	P1: 0, P2: 1, P3: 0, P4: 0
13-14	P2 executes	P1: 0, P2: 0, P3: 0, P4: 0

Turnaround Time for Each Process:

Process	Arrival Time	Completion Time	Turnaround Time
P1	0	10	10 - 0 = 10
P2	1	14	14 - 1 = 13
P3	2	6	6 - 2 = 4
P4	3	13	13 - 3 = 10

Average Turnaround Time = 10+13+4+10 = 37/4 = **9.25**

Sequence of Execution:

P1	P2	P3	P4	P1	P2	P4	P2

0 2 4 6 8 10 12 13 14

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?

➔

Final Values of x:

- Parent Process: After incrementing x by 1, the value of x in the parent process will be 6.
- Child Process: The child process gets a copy of x, which initially is 5. After incrementing by 1, the value of x in the child process will be 6.

So, The final value in both Parent and Child process will be 6.