**Part A**

 echo "Hello, World!" // print hello, world!

 name="Productive" //assign Productive to the variable name

 touch file.txt // crete empty file name

 ls -a // list out all file and directories

 rm file.txt // can remove file.txt

 cp file1.txt file2.txt //copy content of file1 to file2

 mv file.txt /path/to/directory/ // move file.txt to specific directory

 chmod 755 script.sh // give permission of script.sh to readable and executable by everyone but only writable by the owner

 grep "pattern" file.txt // search the string and return it also return all similar string

 kill PID // kill the process

 mkdir mydir (make dir of mydir )&& cd mydir(enter the mydir dir) && touch file.txt(create eempty file) && echo "Hello, World!" > file.txt (create file and writtened hello world)&& cat file.txt(show the data in written in file.txt)

 ls -l | grep ".txt" // list of directories in long format in txt format

 cat file1.txt file2.txt | sort | uniq // concat file1.txt and file2.txt and display only duplicate line

 ls -l | grep "^d" // list of directories in long format

 grep -r "pattern" /path/to/directory/ //recursively search for the string pattern in all file

 cat file1.txt file2.txt | sort | uniq –d // concat file1.txt and file2.txt then sorted combine output and remove duplicate

 chmod 644 file.txt // change the permission of file.txt tobe readable and writable by owner and reader by other

 cp -r source\_directory destination\_directory // copy recursively sorce dir to destination dir

 find /path/to/search -name "\*.txt"

 chmod u+x file.txt // adds execute permission for the owner of file.txt

 echo $PATH // display current value of path in details

Part B – T/F

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

**Identify the Incorrect Commands:**

1. chmodx is used to change file permissions. // chmod is use to change file permission

2. cpy is used to copy files and directories. // cp is use to copy file and dir

3. mkfile is used to create a new file. // mkfile is use to create specific file of specific size - b,k,m,g

4. catx is used to concatenate files. // cat is use to display data

5. rn is used to rename files. // rn use to remove files

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}

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

cdac@LAPTOP-EUG8ANV8:~/LinuxAssignment$ nano script.sh

#!/bin/bash

echo "Enter number"

read number

echo "$number"

cdac@LAPTOP-EUG8ANV8:~/LinuxAssignment$ chmod +x script.sh

cdac@LAPTOP-EUG8ANV8:~/LinuxAssignment$ ./script.sh

Enter number

18

18

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

result.

#!/bin/bash

echo "Enter number1"

read number1

echo "Enter number2"

read number2

sum =$((number1+number2))

echo "$sum"

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

prints "Odd".

#!/bin/bash

echo "Enter num"

read num

if [ $((num % 2)) -eq 0 ];

then

echo "number is even"

else

echo "number is odd"

fi

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

#!/bin/bash

for i in 1 2 3 4 5

do

echo " loop no $i"

done

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

#!/bin/bash

counter=1

while [ $counter -le 5 ]

do

echo $counter

((counter++))

Done

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".

#!/bin/bash

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

echo "file exists"

else

echo "file does not exists"

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.

#!/bin/bash

echo "num : "

read num

if [ $num -gt 10 ]; then

echo "num is greater than 10"

else

echo "num is smaller than 10"

fi

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.

#!/bin/bash

echo -e "\t1\t2\t3\t4\t5"

for i in {1..5}

do

echo -n -e "$i\t"

for j in {1..5}

do

echo -n -e" $((i \* j))\t"

done

echo ""

done

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

#!/bin/bash

echo "Enter number ( negative number to exit)"

while true

do

read num

if [ $num -lt 0 ]; then

break

fi

square=$((num \* num))

echo " The square of $num is : $square"

done

echo "Exited loop"

# 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 | Arrival Time | Burst time | Completion Time | Turn Around Time | Waiting Time |
| P1 | 0 | 5 | 5 | 5 | 0 |
| P2 | 1 | 3 | 8 | 7 | 4 |
| P3 | 2 | 6 | 14 | 12 | 6 |

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 | Arrival Time | Burst Time | Completion Time | Turn Around Time |
| P1 | 0 | 3 | 3 | 3 |
| P2 | 1 | 5 | 13 | 12 |
| P3 | 2 | 1 | 4 | 2 |
| P4 | 3 | 4 | 8 | 5 |

Average Turn Around Time = 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 | Arrival Time | Burst Time | Priority | Completion Time | Turn Around Time | Waiting Time |
| P1 | 0 | 5 | 3 | 12 | 12 | 0 |
| P2 | 1 | 4 | 1 | 5 | 4 | 0 |
| P3 | 2 | 7 | 4 | 19 | 17 | 10 |
| P4 | 3 | 2 | 2 | 7 | 4 | 2 |

1. 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 | Arrival Time | Burst Time | Completion Time | Turn Around Time |
| P1 | 0 | 4 | 10 | 10 |
| P2 | 1 | 5 | 14 | 13 |
| P3 | 2 | 2 | 6 | 4 |
| P4 | 3 | 3 | 13 | 10 |

Average Turn Around Time = 37/4 = 9.25

1. 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?

The final value of x in both the parent and child processes will be 6.