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/**

  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).

- **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

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
   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!"
```

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"
```

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.

```
num1=5

num2=3

sum=$((num1 + num2))

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

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 ((number % 2 == 0)); then
   echo "Even"
else
   echo "Odd"
fi
```

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
```

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++))
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".

```
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 > 10)); then
   echo "The number is greater than 10"
else
   echo "The number is not greater 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.

```
for i in {1..5}
do
   for j in {1..5}
   do
      printf "%d\t" $((i * j))
   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.

```
while true
do
    echo "Enter a number:"
    read number
    if ((number < 0)); then
        echo "Negative number entered. Exiting the loop."
        break
    fi
    square=$((number * number))
    echo "The square of $number is $square"
done
```

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

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**.

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.

Sorted by Priority and Arrival Time:

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

Step 2: Calculate Waiting Time for Each Process

We use the Gantt chart to visualize the scheduling:

- **P1** starts at time 0, but it has a lower priority.
- **P2** is the first to execute because it has the highest priority.

Gantt Chart:

Time	Process
0	-
1	P2
5	P4
7	P1
13	P3
20	-

Calculations:

- **P2:**
  - Starts at time 1, completes at time 5.
  - Waiting Time: Start Time - Arrival Time = 1 - 1 = 0
- **P4:**
  - Starts at time 5, completes at time 7.
  - Waiting Time: Start Time - Arrival Time = 5 - 3 = 2
- **P1:**
  - Starts at time 7, completes at time 13.
  - Waiting Time: Start Time - Arrival Time = 7 - 0 = 7
- **P3:**
  - Starts at time 13, completes at time 20.
  - Waiting Time: Start Time - Arrival Time = 13 - 2 = 11

Step 3: Calculate Average Waiting Time

Process	**Waiting Time**
P1	7
P2	0
P3	11
P4	2

Average Waiting Time=7+0+11+2/4=20/4=5
So, the average waiting time is **5 units**.

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 Table:

Process	Arrival Time	Burst Time	Remaining Time	Completion Time
P1	0	4	0	6
P2	1	5	0	12
P3	2	2	0	8
P4	3	3	0	13

Execution Order:

1. P1 (0 to 2) - Remaining: 2
2. P2 (2 to 4) - Remaining: 3
3. P3 (4 to 6) - Remaining: 0 (completed at 6)
4. P1 (6 to 8) - Remaining: 0 (completed at 8)
5. P4 (8 to 10) - Remaining: 1
6. P2 (10 to 12) - Remaining: 1
7. P4 (12 to 13) - Remaining: 0 (completed at 13)
8. P2 (13 to 14) - Remaining: 0 (completed at 14)

Turnaround Time Calculation:

- P1: Completion Time (6) - Arrival Time (0) = 6

- P2: Completion Time (14) - Arrival Time (1) = 13
- P3: Completion Time (8) - Arrival Time (2) = 6
- P4: Completion Time (13) - Arrival Time (3) = 10

Average Turnaround Time:

Average Turnaround Time=6+13+6+10/4=35/4=8.75

So, the average turnaround time is **8.75 units**.

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