CDAC MUMBAI

Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

- echo "Hello, World!"
  Ans. echo command displays the line of text .This will print hello world .
- name="Productive"
  Ans. This command assigns string “productive” to variable name.
- touch file.txt
  Ans. touch is used to create an empty file or update the timestamp of an existing file.
- ls -a
  Ans. lists all files and directories in the current directory, including hidden files.
- rm file.txt
  Ans. rm is used to remove (delete) files or directories. This command will remove file.txt file.
- cp file1.txt file2.txt
  Ans This command Copies the file file1.txt to file2.txt. If file2.txt doesn't exist, it will be created.
- mv file.txt /path/to/directory/
  Ans. This command Moves file.txt to the specified directory (/path/to/directory/). If a file with the same name exists in that directory, it will be overwritten.
- chmod 755 script.sh
  Ans. Changes permissions of script.sh to 755,
  Owner: read, write, and execute (7).
  Group: read and execute (5).
  Others: read and execute (5).
- grep "pattern" file.txt
  Ans. This command Searches for the string "pattern" in file.txt and prints matching lines.
- kill PID
  Ans. This command Terminates the process with the specified PID (Process ID).
- mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
  Ans.This command:
  a) Creates a new directory named mydir.
  b) Changes the current directory to mydir.
  c) Creates an empty file file.txt.
  d) Writes "Hello, World!" to file.txt.
  e) Displays the content of file.txt using cat.
- ls -l | grep ".txt"
  Ans. This command Lists files in long format and filters the results to show only files with .txt in their name.
- cat file1.txt file2.txt | sort | uniq
  Ans. This command Concatenates the contents of file1.txt and file2.txt, sorts them, and removes duplicate lines.
- ls -l | grep "^d"
  Ans. This command Lists files and directories in long format and filters to show only directories (lines starting with d).
- grep -r "pattern" /path/to/directory/
  Ans. This command Recursively searches for "pattern" in all files under the specified directory.
- cat file1.txt file2.txt | sort | uniq –d

Ans. This command Concatenates the contents of file1.txt and file2.txt, sorts them, and displays only duplicate lines (lines that appear in both files).

- chmod 644 file.txt
  Ans. This command Changes permissions of file.txt to 644, meaning:
  a) Owner: read and write (6).
  b) Group: read-only (4).
  c) Others: read-only (4).
- cp -r source_directory destination_directory
  Ans. Recursively copies the source_directory and its contents to destination_directory.
- find /path/to/search -name "*.txt"
  Ans. This command Searches for all files ending with .txt within the specified path and its subdirectories.
- chmod u+x file.txt
  Ans. This command Adds execute permission to the file owner for file.txt.
- echo $PATH
  Ans. This command Prints the current value of the PATH environment variable, which contains directories where executable 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. Cd is use to print current 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. The -rf option is typically used for directories, not individual files.

Identify the Incorrect Commands:

1. **chmodx** is used to change file permissions.
   Ans. Incorrect**.** The correct command is **chmod.**
2. **cpy** is used to copy files and directories.
   Ans. Incorrect**.** The correct command is **cp.**
3. **mkfile** is used to create a new file.
   Ans. Incorrect**.** The correct command to create a new file is **touch**.
4. **catx** is used to concatenate files.
   Ans. Incorrect. The correct command is **cat**.
5. **rn** is used to rename files.
   Ans. Incorrect. The correct command is **mv**.

Part C

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

```
#!/bin//bash
echo "Hello., World!"
```

```
cdac@DESKTOP-S7QM7LP:~$ nano hello.sh
cdac@DESKTOP-S7QM7LP:~$ bash hello.sh
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@DESKTOP-S7QM7LP:~$ nano name.sh
cdac@DESKTOP-S7QM7LP:~$ bash name.sh
CDAC Mumbai
```

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

```
echo Enter number
read number
echo "Entered number is" $number
```

```
cdac@DESKTOP-S7QM7LP:~$ nano number.sh
cdac@DESKTOP-S7QM7LP:~$ bash number.sh
Enter number
20
Entered number is 20
cdac@DESKTOP-S7QM7LP:~$
```

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

```
echo Enter first number
read a
echo Enter second number
read b
sum=`expr $a + $b`
echo sum of $a and $b is $sum .
```

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ nano add
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash add
Enter first number
20
Enter second number
80
sum of 20 and 80 is 100 .
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

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 n
if (( n % 2 == 0 ))
then
echo $n is Even.
else
echo $n is odd.
fi
```

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ nano evenodd
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash evenodd
Enter a number
10
10 is Even.
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash evenodd
Enter a number
43
43 is odd.
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

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

```
#!/bin/bash
for a in 1 2 3 4 5
do
echo $a
done
```

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ nano forloop.sh
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash forloop.sh
1
2
3
4
5
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

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
```

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash whileloop.sh
1
2
3
4
5
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

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

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ nano filecheck
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash filecheck
File exists
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

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.

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ nano num
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash num
Enter a number
5
The number is not greater than 10.
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash num
Enter a number
11
The number is greater than 10.
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

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
for j in {1..10}; do
  for i in {1..5}; do
    result=$((i * j))
    printf "%2d * %2d = %2d    " $i $j $result
  done
  echo
done
```

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ nano table
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash table
 1 *  1 =  1     2 *  1 =  2     3 *  1 =  3     4 *  1 =  4     5 *  1 =  5
 1 *  2 =  2     2 *  2 =  4     3 *  2 =  6     4 *  2 =  8     5 *  2 = 10
 1 *  3 =  3     2 *  3 =  6     3 *  3 =  9     4 *  3 = 12     5 *  3 = 15
 1 *  4 =  4     2 *  4 =  8     3 *  4 = 12     4 *  4 = 16     5 *  4 = 20
 1 *  5 =  5     2 *  5 = 10     3 *  5 = 15     4 *  5 = 20     5 *  5 = 25
 1 *  6 =  6     2 *  6 = 12     3 *  6 = 18     4 *  6 = 24     5 *  6 = 30
 1 *  7 =  7     2 *  7 = 14     3 *  7 = 21     4 *  7 = 28     5 *  7 = 35
 1 *  8 =  8     2 *  8 = 16     3 *  8 = 24     4 *  8 = 32     5 *  8 = 40
 1 *  9 =  9     2 *  9 = 18     3 *  9 = 27     4 *  9 = 36     5 *  9 = 45
 1 * 10 = 10     2 * 10 = 20     3 * 10 = 30     4 * 10 = 40     5 * 10 = 50
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

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
  read -p "Enter a number (negative to exit): " num

  if [ $num -lt 0 ]; then
    echo "Negative number entered, exiting."
    break
  fi

  square=$((num * num))
  echo "Square of $num is: $square"
done
```

```
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ nano program
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$ bash program
Enter a number (negative to exit): 10
Square of 10 is: 100
Enter a number (negative to exit): 2
Square of 2 is: 4
Enter a number (negative to exit): 33
Square of 33 is: 1089
Enter a number (negative to exit): 345
Square of 345 is: 119025
Enter a number (negative to exit): -90
Negative number entered, exiting.
cdac@DESKTOP-S7QM7LP:~/home/shellprogramming$
```

Part D

Common Interview Questions (Must know)

1. What is an operating system, and what are its primary functions?
2. Explain the difference between process and thread.
3. What is virtual memory, and how does it work?
4. Describe the difference between multiprogramming, multitasking, and multiprocessing.
5. What is a file system, and what are its components?
6. What is a deadlock, and how can it be prevented?
7. Explain the difference between a kernel and a shell.
8. What is CPU scheduling, and why is it important?
9. How does a system call work?
10. What is the purpose of device drivers in an operating system?
11. Explain the role of the page table in virtual memory management.
12. What is thrashing, and how can it be avoided?
13. Describe the concept of a semaphore and its use in synchronization.
14. How does an operating system handle process synchronization?
15. What is the purpose of an interrupt in operating systems?
16. Explain the concept of a file descriptor.
17. How does a system recover from a system crash?
18. Describe the difference between a monolithic kernel and a microkernel.
19. What is the difference between internal and external fragmentation?
20. How does an operating system manage I/O operations?
21. Explain the difference between preemptive and non-preemptive scheduling.
22. What is round-robin scheduling, and how does it work?
23. Describe the priority scheduling algorithm. How is priority assigned to processes?
24. What is the shortest job next (SJN) scheduling algorithm, and when is it used?
25. Explain the concept of multilevel queue scheduling.
26. What is a process control block (PCB), and what information does it contain?
27. Describe the process state diagram and the transitions between different process states.
28. How does a process communicate with another process in an operating system?
29. What is process synchronization, and why is it important?
30. Explain the concept of a zombie process and how it is created.
31. Describe the difference between internal fragmentation and external fragmentation.
32. What is demand paging, and how does it improve memory management efficiency?
33. Explain the role of the page table in virtual memory management.
34. How does a memory management unit (MMU) work?
35. What is thrashing, and how can it be avoided in virtual memory systems?
36. What is a system call, and how does it facilitate communication between user programs and the operating system?
37. Describe the difference between a monolithic kernel and a microkernel.
38. How does an operating system handle I/O operations?
39. Explain the concept of a race condition and how it can be prevented.

40. Describe the role of device drivers in an operating system.
41. What is a zombie process, and how does it occur? How can a zombie process be prevented?
42. Explain the concept of an orphan process. How does an operating system handle orphan processes?
43. What is the relationship between a parent process and a child process in the context of process management?
44. How does the fork() system call work in creating a new process in Unix-like operating systems?
45. Describe how a parent process can wait for a child process to finish execution.
46. What is the significance of the exit status of a child process in the wait() system call?
47. How can a parent process terminate a child process in Unix-like operating systems?
48. Explain the difference between a process group and a session in Unix-like operating systems.
49. Describe how the exec() family of functions is used to replace the current process image with a new one.
50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?
51. How does process termination occur in Unix-like operating systems?
52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?
53. How does the short-term scheduler differ from the long-term and medium-term schedulers in terms of frequency of execution and the scope of its decisions?
54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.

Part E

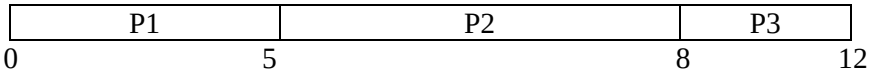
1. Consider the following processes with arrival times and burst times:

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.

Ans.

Process No.	Arrival Time	Burst Time	Completion time	Waiting time	TAT
P1	0	5	5	0	5
P2	1	3	8	4	7
P3	2	6	12	6	10



Avg Waiting time = 0+4+6/3 = 3.33ms

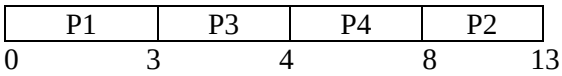
2. Consider the following processes with arrival times and burst times:

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.

ANS.

Process No	AT	BT	CT	TAT	WT
P1	0	3	3	3	0
P2	1	5	13	12	7
P3	2	1	4	2	1
P4	3	4	8	5	1



AVG TAT = 3+12+2+5/4 = 5.5

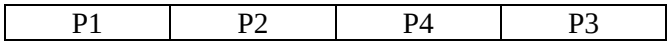
3. Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority):

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.

ANS.

P No	AT	BT	P	CT	TAT	WT
P1	0	6	3	6	6	0
P2	1	4	1	10	9	5
P3	2	7	4	19	17	10
P4	3	2	2	12	9	7



0 6 10 12 19

AVG Waiting time = 5+10+7/4 = 5.5

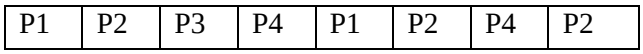
4. Consider the following processes with arrival times and burst times, and the time quantum for Round Robin scheduling is 2 units:

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.

ANS.

P NO	AT	BT	CT	TAT	WT
P1	0	4	10	10	6
P2	1	5	14	13	8
P3	2	2	6	4	2
P4	3	3	13	10	7



0 2 4 6 8 10 12 13 14

Avg. TAT = 10+13+4+10/4 = 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?

   Ans.

```
#!/bin/bash
x=5
(
x=$((x + 1))
echo "Child process: x = $x"
) &
x=$((x + 1))
```

echo "Parent process: x = $x"

```
cdac@DESKTOP-S7QM7LP:~$ nano forkexp
cdac@DESKTOP-S7QM7LP:~$ bash forkexp
Parent process: x = 6
Child process: x = 6
cdac@DESKTOP-S7QM7LP:~$
```

Submission Guidelines:

- Document each step of your solution and any challenges faced.
- Upload it on your GitHub repository

Additional Tips:

- Experiment with different options and parameters of each command to explore their functionalities.
- This assignment is tailored to align with interview expectations, CCEE standards, and industry demands.
- If you complete this then your preparation will be skyrocketed.