Concepts of Operating System
Assignment 2

Part A Solution

What will the following commands do?

- **echo "Hello, World!"**
  - The 'echo' command is used to print the text, line, string on the terminal. The text inside the " " is the string you want to display. In this case the Hello, World! Will be printed on the screen.

- **name="Productive"**
  - The 'name' in this command is the variable name which we are defining and "Productive " is the string value which is assign to variable using '=' operator.

- **touch file.txt**
  - The touch command is used to create an empty file. In this case it will create a empty file named file.txt.

- **ls –a**
  - This command is used to list 'l' all '-a' files including hidden files and directories in the terminal.

- **rm file.txt**
  - It is specifically used to remove/delete a file from a specific directory. Here it will remove the file.txt from a current directory.
- **cp file1.txt file2.txt**
  - The 'cp' command is used to copy a file or directory. Here the file1.txt is the source file which we want to copy and file2.txt is the destination file, if the file2.txt doesn't exist it will be created with same content as file1.txt.

- **mv file.txt /path/to/directory/**
  - The 'mv' command is used to move or rename a file or directory. In this case the file.txt is the source file and /path/to/directory/ is destination where we want to move the file.

- **chmod 755 script.sh**
  - The 'chmod' is command used to change the permission for the owner, group, user
  - This is a numeric represents that he file permissions is set.
  - The number 755 is represented as :
  - 7 (Owner): The owner of the file has read (4), write (2), and execute (1) permissions.
  - 5 (Group): Members of the group have read (4) and execute (1) permissions, totaling 5.
  - 5 (Others): All other users (everyone else) have read (4) and execute (1) permissions.

- **kill PID**
  - The ‘kill’ command is used to terminate the process by its ID(Process Id).

- **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**
  - In this command we used series of chained commands, each command separated ‘&&’ and it means each command runs only if previous one gets run successfully.
  - ‘mkdir mydir’ will create new directory.
  - ‘cd’ will change the directory to mydir.
  - ‘touch’ will create new empty file named with file.txt.
  - echo "Hello, World!" > file.txt will write the provided string into file.txt
  - ‘cat’ command will output the content from file.txt.

- **ls -l | grep ".txt"**
  - ls –l will lists all files and directories in the current directory in long format. The long format shows detail explanation such as permissions, owner, group, size, and modified date for each file or directory.
  - | (pipe) will Passes the output of the ls -l command as input to the next command (grep).
  - grep ".txt” will filters the output to show only the lines that contain the string .txt. This will typically match any files with a .txt extension.

- **cat file1.txt file2.txt | sort | uniq**
  - The ‘cat file1.txt file2.txt’ will concatenates the contents of file1.txt and file2.txt and shows them together as one stream of text.
  - And then | (pipe) Sends this combined text to the next command (sort).
  - ‘sort’ command will arrange the lines in order like alphabetically or by numbers.
  - ‘uniq’ command will removes repeated lines, but only if they are next to each other.
  - So basically produces a sorted list of unique lines from the combined contents of file1.txt and file2.txt.

- **ls -l | grep "^d"**
  - The ‘ls –l’ is used to lists all files and directories in the current directory in long format, showing detailed information for each.
  - And | (pipe) passes the output of ls -l to the next command, grep.
  - So ‘grep "^d"’ will filters and displays only the lines that start with the letter d and in this scenario it is going to display the detailed information of only the directories in the current directory.

- **grep -r "pattern" /path/to/directory**
  - grep -r "pattern" will search recursively through files for the specified "pattern".
  - The -r (or --recursive) option tells grep to read all files under each directory, starting from the specified directory.
  - /path/to/directory specifies the directory where the recursive search should start.
  - In this case it finds the specified "pattern" in all files within /path/to/directory and its subdirectories, displaying matching lines along with their filenames.

- **cat file1.txt file2.txt | sort | uniq –d**
  - The ‘cat file1.txt file2.txt’ will concatenates the contents of file1.txt and file2.txt and shows them together as one stream of text.
  - And then | (pipe) Sends this combined text to the next command (sort).
  - ‘sort’ command will arrange the lines in order like alphabetically or by numbers.
  - ‘uniq –d’ will check the sorted output to display only the duplicate lines, i.e., lines that appear more than once across the two files. The -d option ensures that only lines that are repeated are shown.
  - So it finds and displays the lines that appear in both file1.txt and file2.txt.

- **chmod 644 file.txt**
  - The command chmod 644 file.txt changes the permissions of file.txt to 644, which gives the file's permissions like :
  - 6 (read and write for the owner): The owner of the file has read and write permissions.
  - 4 (read-only for the group): The group associated with the file has read-only permissions.
  - 4 (read-only for others): All other users have read-only permissions.
  - It will set the permissions of file.txt so that the owner can read and write the file, while everyone else can only read it.

- **cp -r source_directory destination_directory**
  - In this the ‘cp –r’ will copy files and directories recursively. The -r option ensures that the entire directory structure, including all files and subdirectories, is copied.
  - ‘source_directory’ specifies the directory to be copied.
  - ‘destination_directory’ specifies the location where source_directory should be copied to.
  - Copy the entire source_directory, including all its contents, to destination_directory. If destination_directory does not exist, it will be created.

- **find /path/to/search -name "*.txt"**
  - Locate and list all files with a .txt extension within /path/to/search and its subdirectories.

- **chmod u+x file.txt**
  - chmod command is used to give permission to the file and here it will give excute permission to the owner of the file.

- **echo $PATH**
  - Display the current value of the PATH environment variable, showing all directories where the shell looks for executable commands.

Part B Solution

Identify True or False:

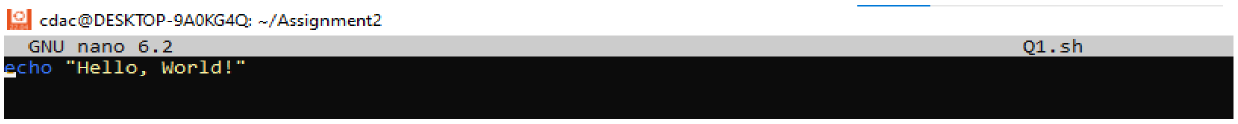
1. **ls** is used to list files and directories in a directory………………………………………… **TRUE**
2. **mv** is used to move files and directories……………………………………………….... **FALSE**
3. **cd** is used to copy files and directories…………………… ……………………………..**FALSE**
4. **pwd** stands for "print working directory" and displays 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:

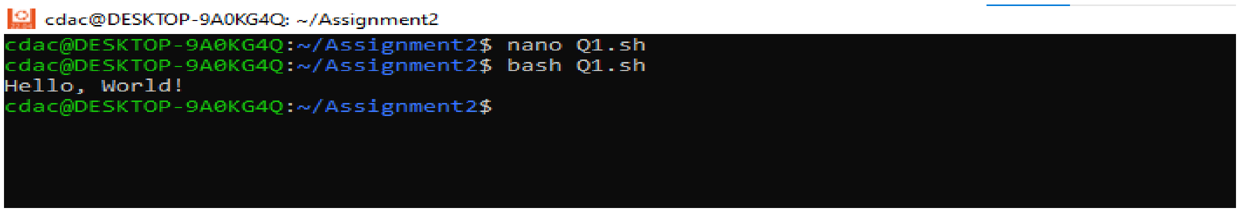
1. **chmodx: Incorrect.** The correct command is chmod, which is used to change file permissions.
2. **cpy: Incorrect.** The correct command is cp, which is used to copy files and directories.
3. **mkfile: Incorrect.** There is no standard command called mkfile for creating new files in most Unix-like systems. The correct command for creating an empty file is touch.
4. **catx: Incorrect.** The correct command is cat, which is used to concatenate and display the contents of files.
5. **rn: Incorrect.** The correct command is mv, which is used to rename (as well as move) files.

Part C Solution

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

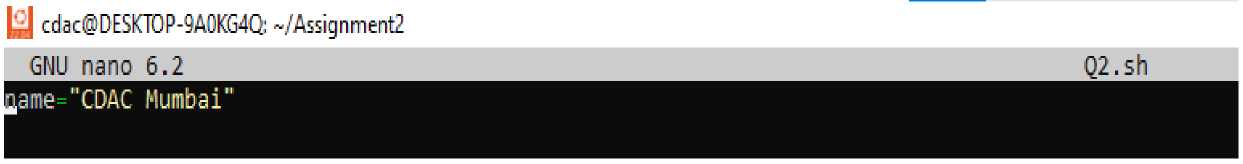


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                          Q1.sh
echo "Hello, World!"
```

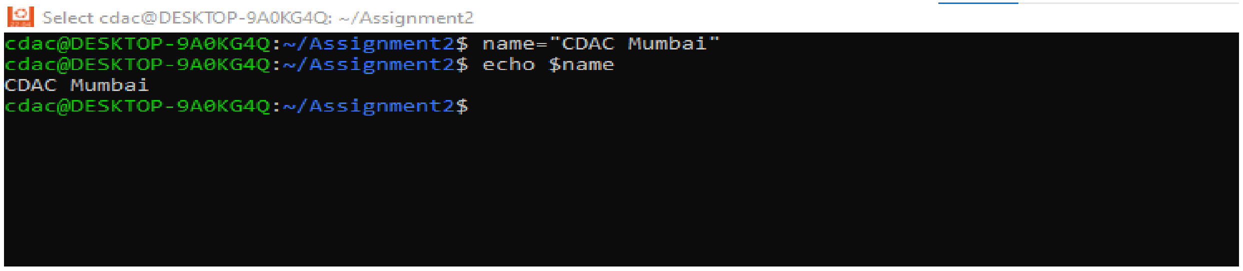


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q1.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q1.sh
Hello, World!
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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

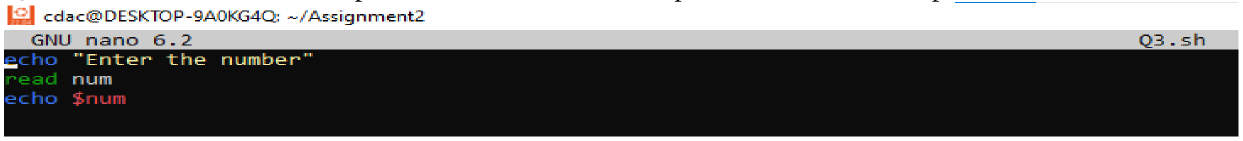


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                          Q2.sh
name="CDAC Mumbai"
```

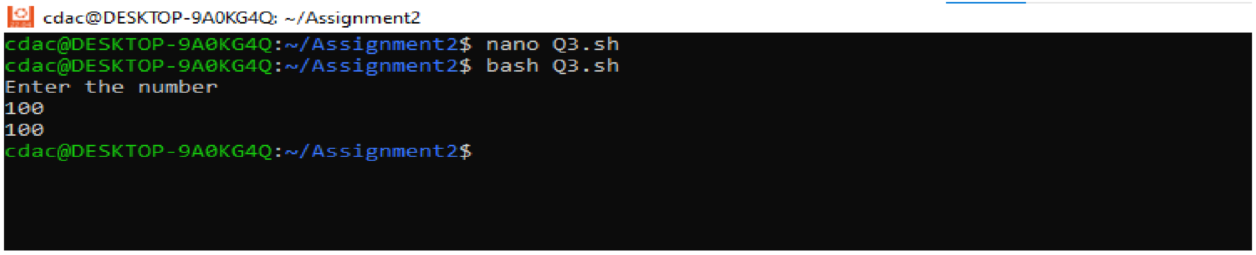


```
Select cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ name="CDAC Mumbai"
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ echo $name
CDAC Mumbai
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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

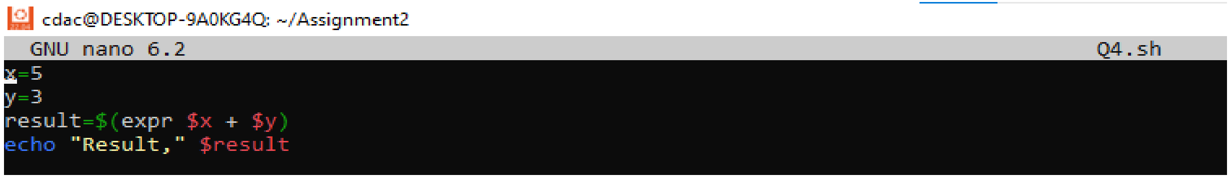


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                          Q3.sh
echo "Enter the number"
read num
echo $num
```

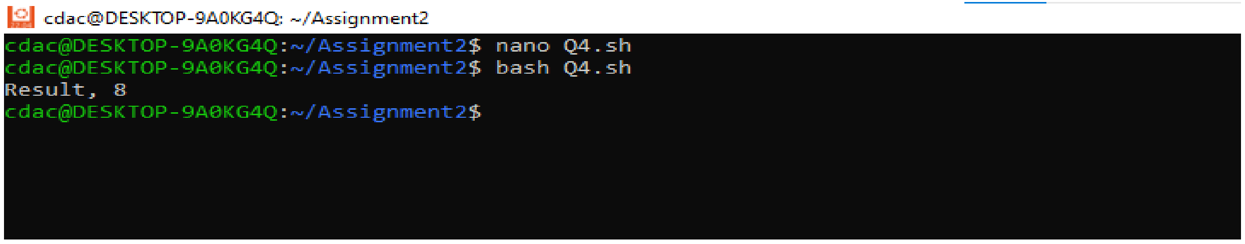


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q3.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q3.sh
Enter the number
100
100
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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

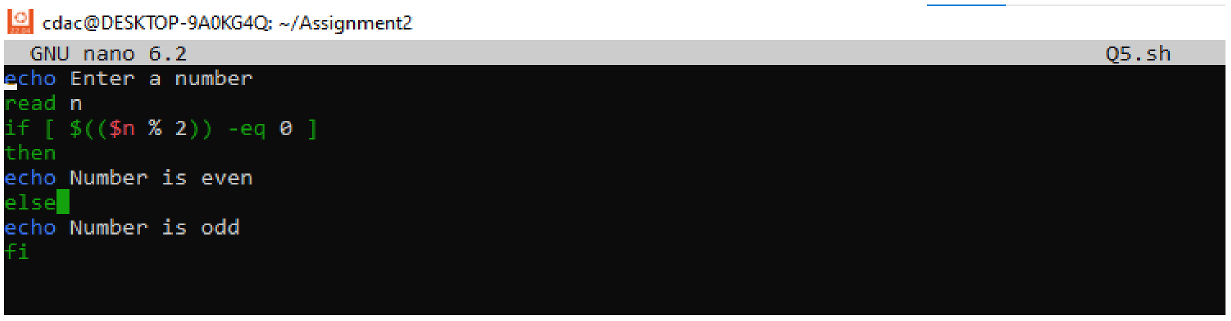


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                          Q4.sh
x=5
y=3
result=$(expr $x + $y)
echo "Result," $result
```

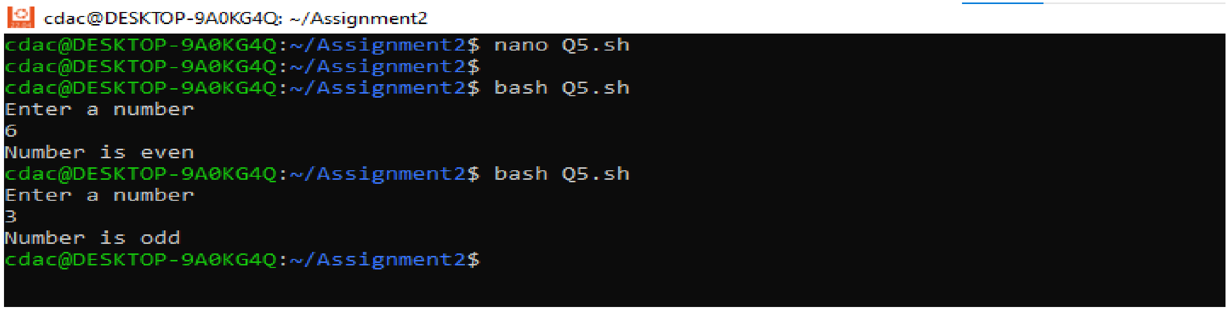


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q4.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q4.sh
Result, 8
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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

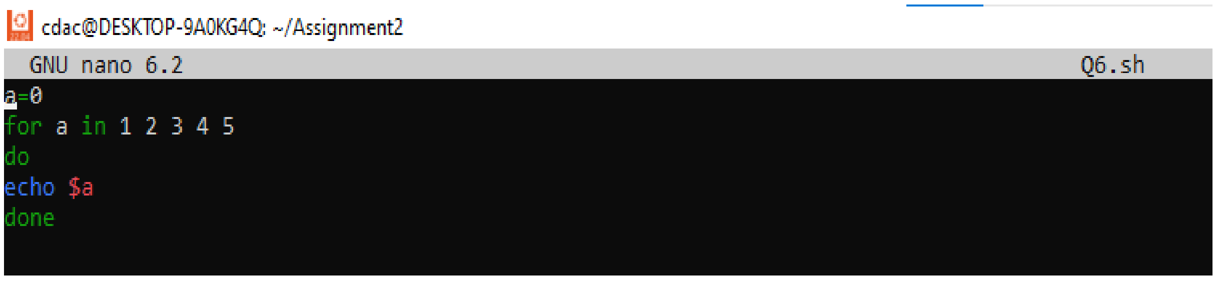


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                    Q5.sh
echo Enter a number
read n
if [ $(($n % 2)) -eq 0 ]
then
echo Number is even
else
echo Number is odd
fi
```

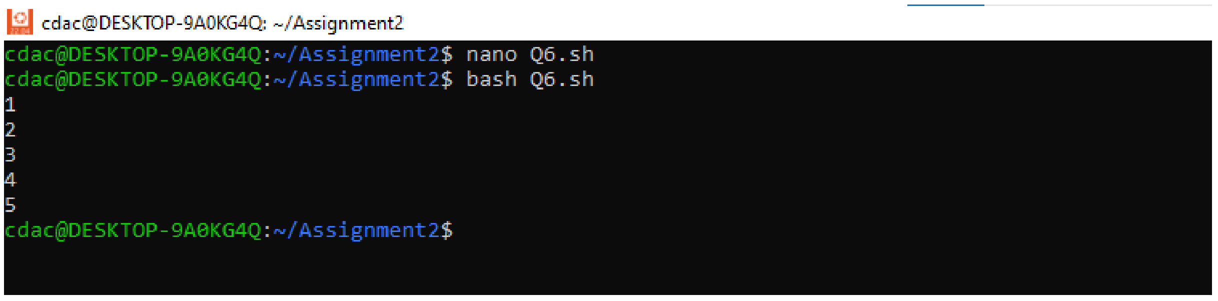


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q5.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q5.sh
Enter a number
6
Number is even
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q5.sh
Enter a number
3
Number is odd
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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

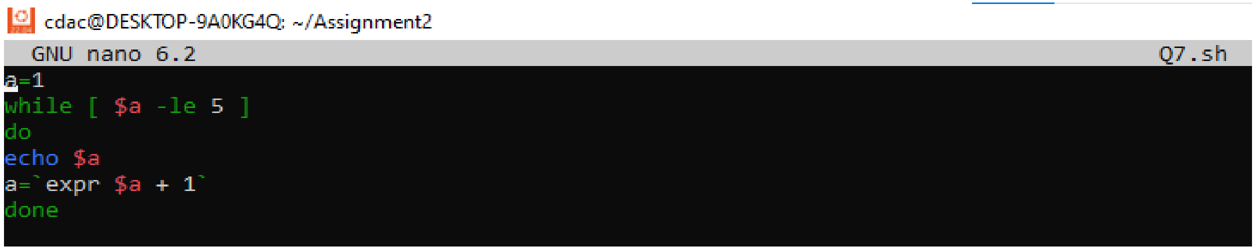


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                    Q6.sh
a=0
for a in 1 2 3 4 5
do
echo $a
done
```

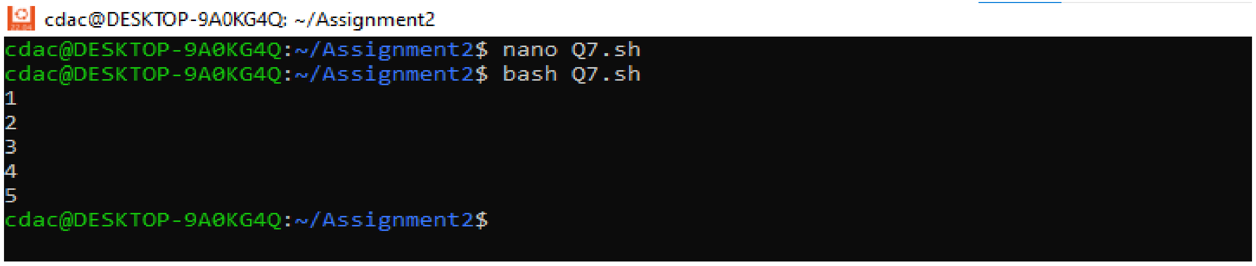


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q6.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q6.sh
1
2
3
4
5
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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

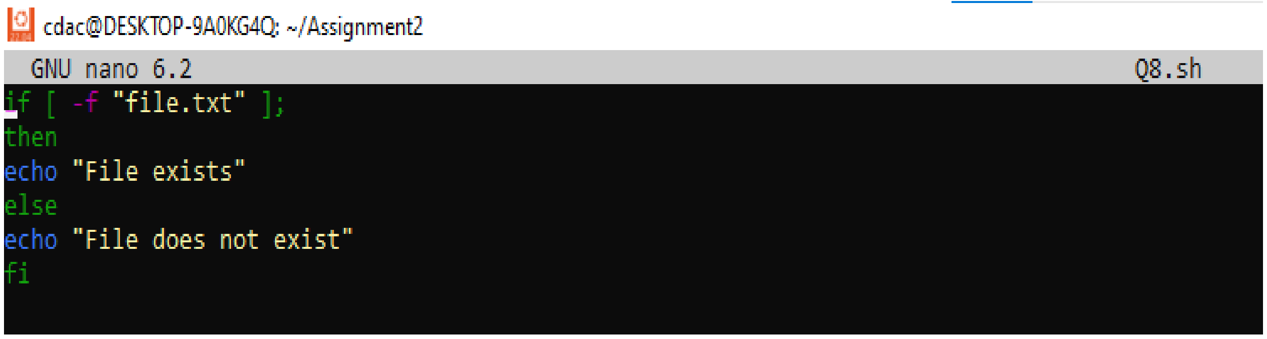


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                                 Q7.sh
a=1
while [ $a -le 5 ]
do
echo $a
a=`expr $a + 1`
done
```

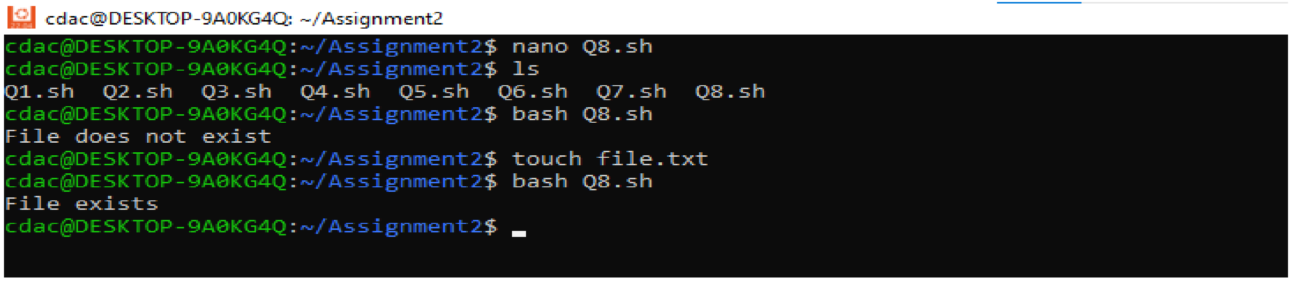


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q7.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q7.sh
1
2
3
4
5
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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".

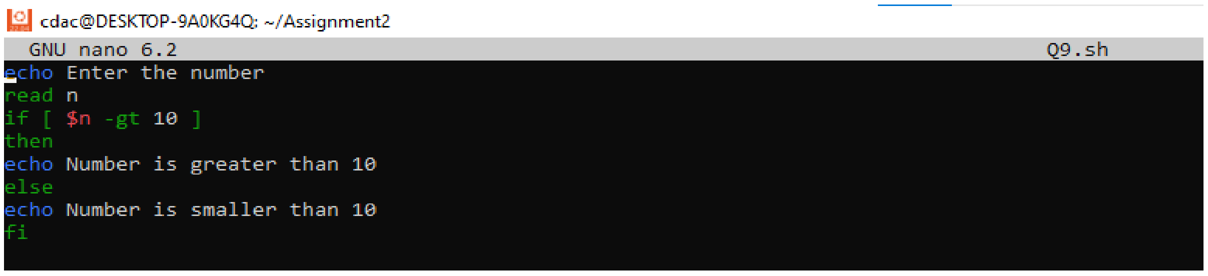


```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                                 Q8.sh
if [ -f "file.txt" ];
then
echo "File exists"
else
echo "File does not exist"
fi
```



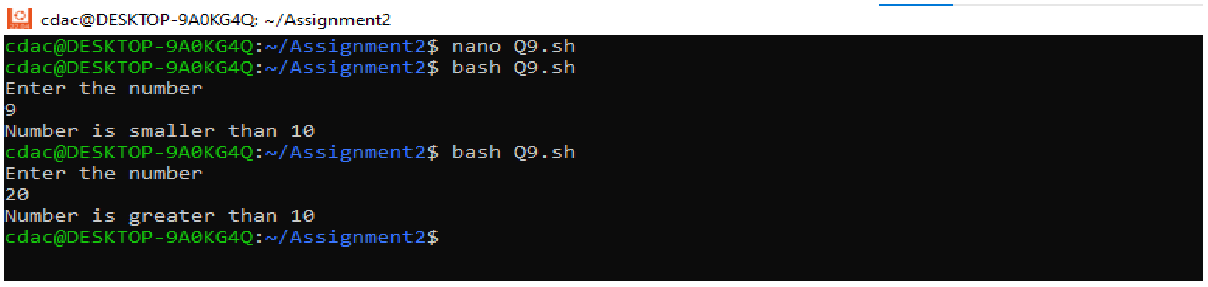
```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q8.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ ls
Q1.sh  Q2.sh  Q3.sh  Q4.sh  Q5.sh  Q6.sh  Q7.sh  Q8.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q8.sh
File does not exist
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ touch file.txt
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q8.sh
File exists
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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-9A0KG4Q: ~/Assignment2

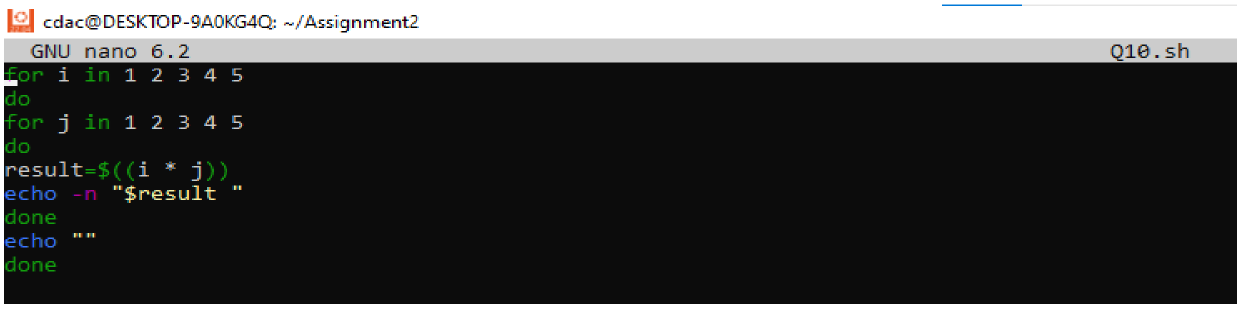
```
  GNU nano 6.2                                                              Q9.sh
echo Enter the number
read n
if [ $n -gt 10 ]
then
echo Number is greater than 10
else
echo Number is smaller than 10
fi
```



cdac@DESKTOP-9A0KG4Q: ~/Assignment2

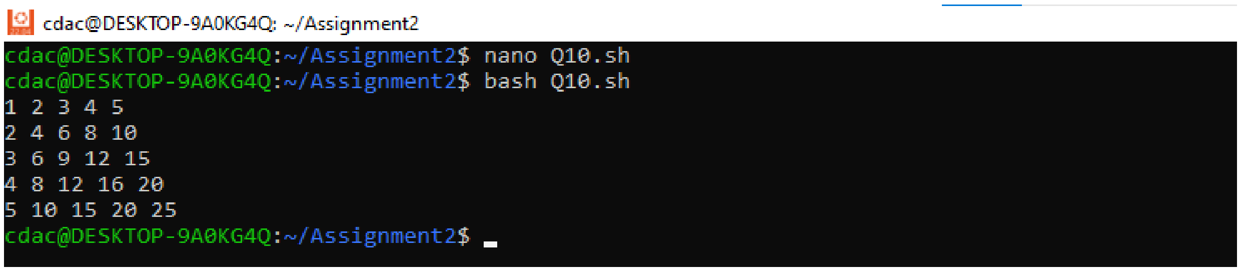
```
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q9.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q9.sh
Enter the number
9
Number is smaller than 10
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q9.sh
Enter the number
20
Number is greater than 10
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

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.



cdac@DESKTOP-9A0KG4Q: ~/Assignment2

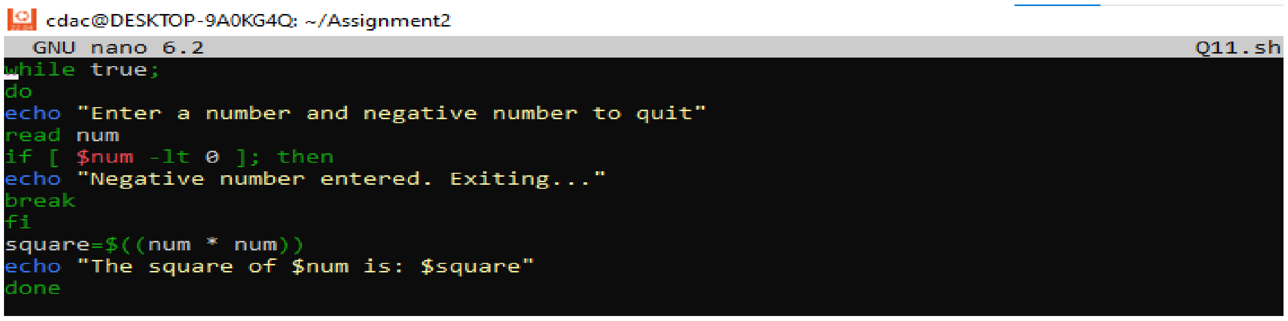
```
  GNU nano 6.2                                                              Q10.sh
for i in 1 2 3 4 5
do
for j in 1 2 3 4 5
do
result=$((i * j))
echo -n "$result "
done
echo ""
done
```



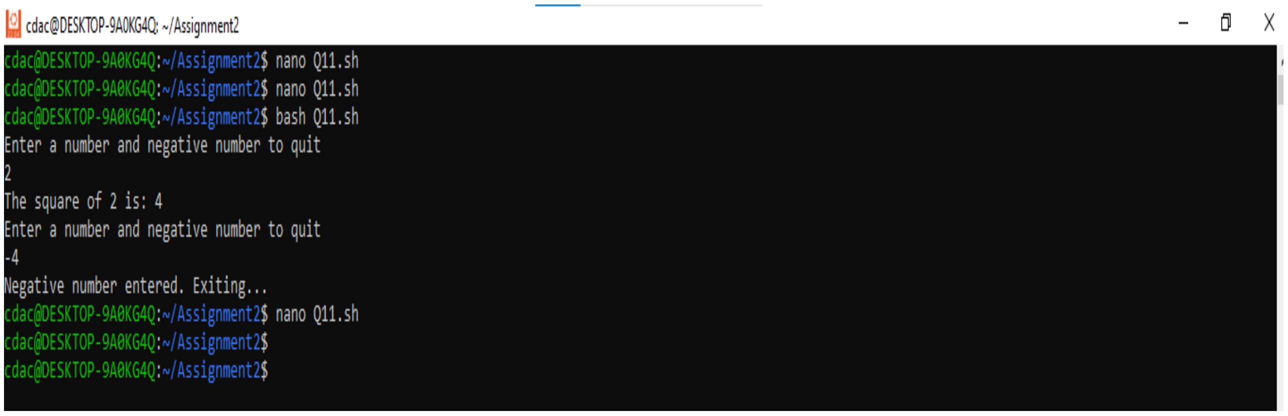
cdac@DESKTOP-9A0KG4Q: ~/Assignment2

```
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q10.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q10.sh
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-9A0KG4Q:~/Assignment2$
```

Question 11: Write a shell script that uses a while loop to read numbers from the user until the user entersa negative number. For each positive number entered, print its square. Use the **break** statement to exit theloop when a negative number is entered.



```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
  GNU nano 6.2                                                                  Q11.sh
while true;
do
echo "Enter a number and negative number to quit"
read num
if [ $num -lt 0 ]; then
echo "Negative number entered. Exiting..."
break
fi
square=$((num * num))
echo "The square of $num is: $square"
done
```



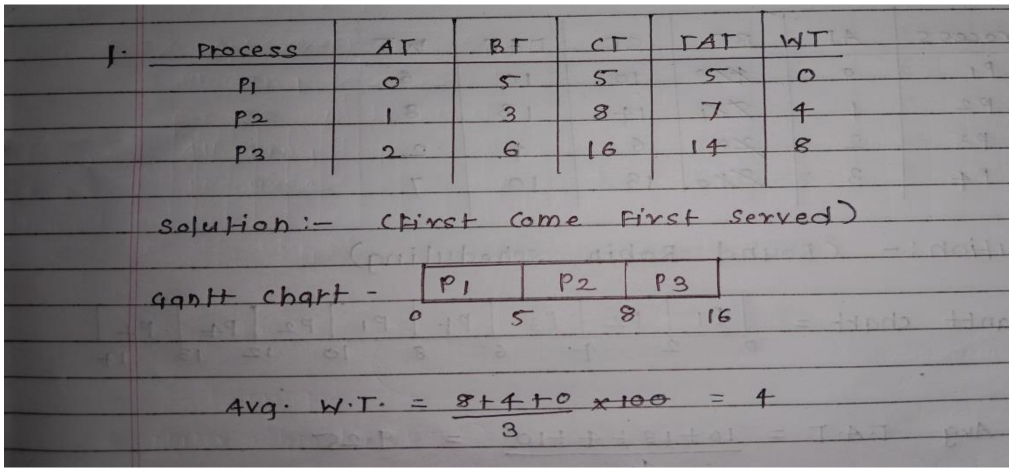
```
cdac@DESKTOP-9A0KG4Q: ~/Assignment2
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q11.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q11.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ bash Q11.sh
Enter a number and negative number to quit
2
The square of 2 is: 4
Enter a number and negative number to quit
-4
Negative number entered. Exiting...
cdac@DESKTOP-9A0KG4Q:~/Assignment2$ nano Q11.sh
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
cdac@DESKTOP-9A0KG4Q:~/Assignment2$
```

Part D Solution

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.



1.

Process	AT	BT	CT	TAT	WT
P1	0	5	5	5	0
P2	1	3	8	7	4
P3	2	6	16	14	8

Solution :- (First come First served)

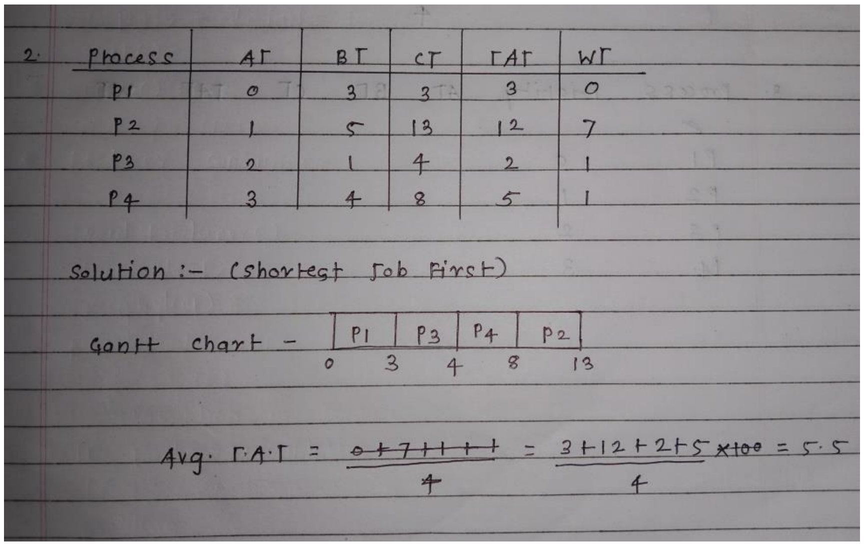
Gantt chart - | P1 | P2 | P3 |
0 5 8 16

Avg. W.T. = $\frac{8+4+0}{3}$ ~~×100~~ = 4

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.



2.

Process	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

Solution :- (Shortest Job First)

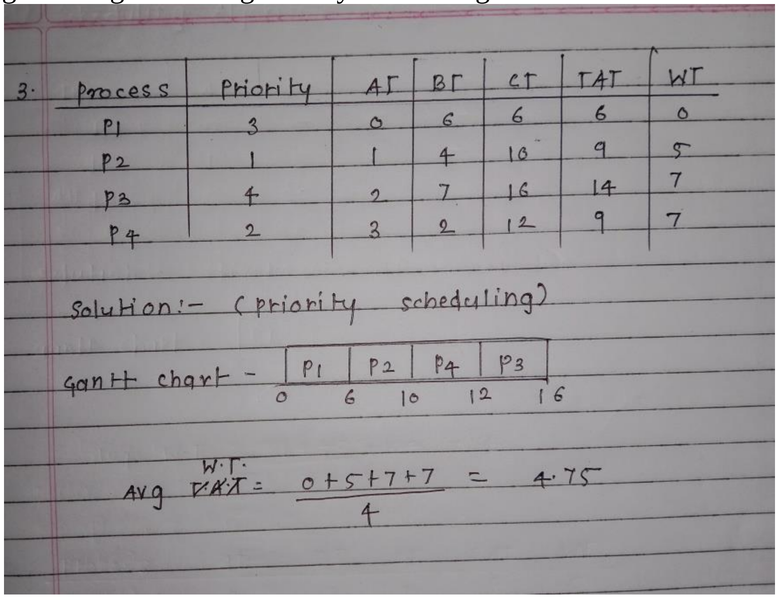
Gantt chart - | P1 | P3 | P4 | P2 |
0 3 4 8 13

Avg. T.A.T = $\frac{\cancel{0+7+1+1}}{4}$ = $\frac{3+12+2+5}{4}$ ~~×100~~ = 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.



3.

Process	Priority	AT	BT	CT	TAT	WT
P1	3	0	6	6	6	0
P2	1	1	4	10	9	5
P3	4	2	7	16	14	7
P4	2	3	2	12	9	7

Solution:- (priority scheduling)

Gantt chart -

P1	P2	P4	P3

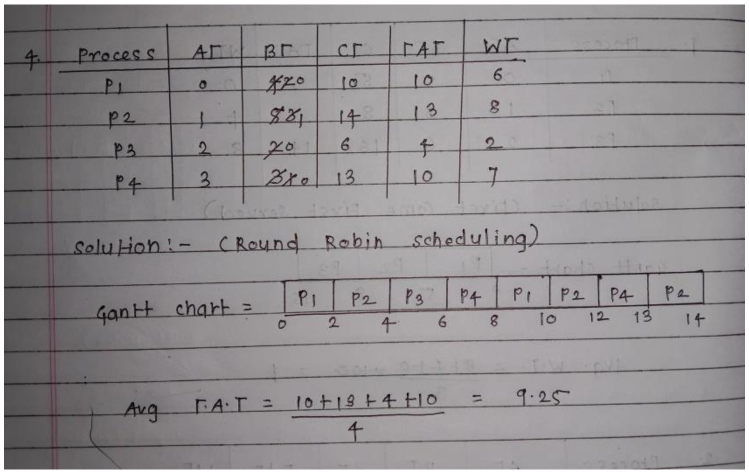
0 6 10 12 16

Avg ~~T.A.T~~ W.T. = $\frac{0+5+7+7}{4}$ = 4.75

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.



4.

Process	AT	BT	CT	TAT	WT
P1	0	~~4~~ ~~2~~ 0	10	10	6
P2	1	~~5~~ ~~3~~ 1	14	13	8
P3	2	~~2~~ 0	6	4	2
P4	3	~~3~~ ~~1~~ 0	13	10	7

Solution:- (Round Robin scheduling)

Gantt chart =

P1	P2	P3	P4	P1	P2	P4	P2

0 2 4 6 8 10 12 13 14

Avg T.A.T = $\frac{10+13+4+10}{4}$ = 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?

```
#include <stdio.h>

void main() {

        int x = 5;
        fork();
        x = x+1;
        printf("x = %d\n",x);
}
```

1. **Parent Process:**

- Initially, `x = 5`.
- After `fork()`, `x` is incremented to 6 in the parent process.

2. **Child Process:**

- Initially, x = 5 (copied from the parent process at the time of fork).
- After fork(), x is incremented to 6 in the child process.

So, in both the parent and child processes, the final value of `x` will be 6.

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.