CDAC MUMBAI

Concepts of Operating System
Assignment 2

Part A

What will the following commands do?

- echo "Hello, World!"
  Print hello worls
- name="Productive"
  Name is variable productive is value
- touch file.txt
  to create file
- ls -a
  list the files hidden ones also
- rm file.txt
  remove files
- cp file1.txt file2.txt
  copy file 1 to file 2
- mv file.txt /path/to/directory/
  move file to given path
- chmod 755 script.sh
  give permission to r/w/x/to owner and r/x to others
- grep "pattern" file.txt
  search for patterns in file
- kill PID
  terminate the process in pid
- mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
  create mydir in this create file print hello world and then display file content
- ls -l | grep ".txt"
  list files and give txt files only
- cat file1.txt file2.txt | sort | uniq
  conbine both sort them and show duplicates only
- ls -l | grep "^d"
  list dir only files starting with d
- grep -r "pattern" /path/to/directory/
  recursilvly searches pattern in given dir
- cat file1.txt file2.txt | sort | uniq –d
  show duplicate files within files
- chmod 644 file.txt
  give permission to r/w to owner and read to others only
- cp -r source_directory destination_directory
  copy dir recursivly
- find /path/to/search -name "*.txt"
  fint txt file for given path
- chmod u+x file.txt

give permission to execute to user

- echo $PATH
  displays sys executable search pat

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. **TRUE**
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**

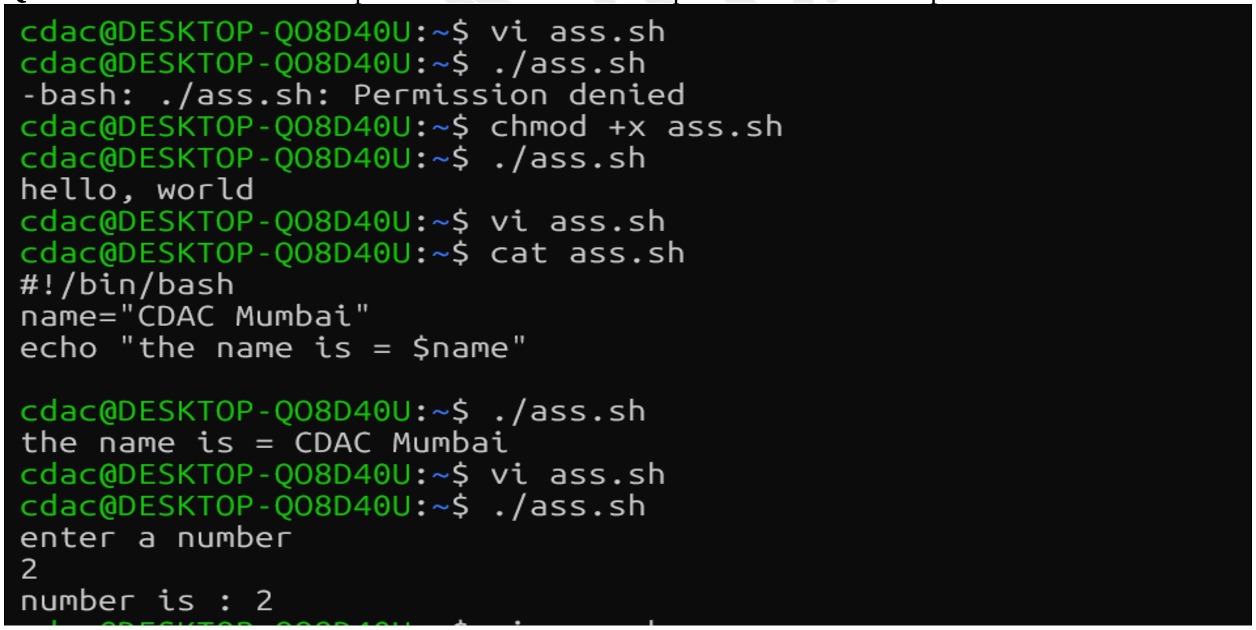
Identify the Incorrect Commands:

1. **chmodx** is used to change file permissions.
2. **cpy** is used to copy files and directories.
3. **mkfile** is used to create a new file.
4. **catx** is used to concatenate files.
5. **rn** is used to rename files.

All the listed commands are incorrect

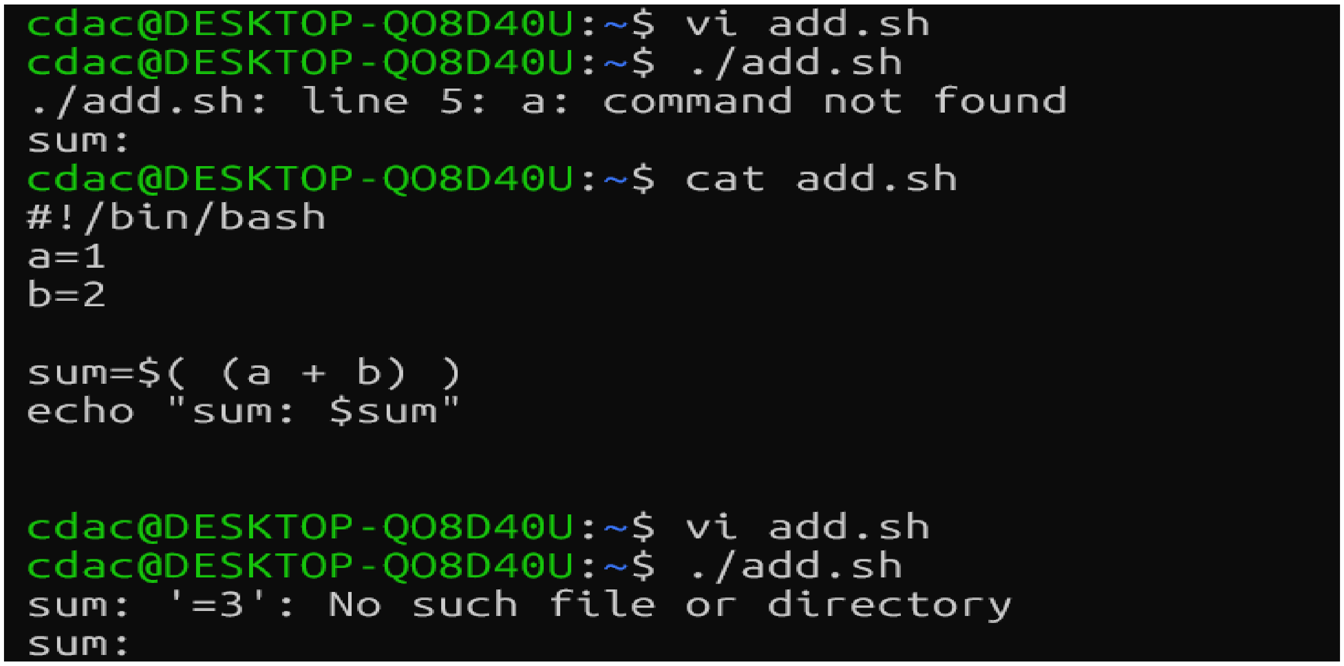
Part C

Question 1: Write a shell script that prints "Hello, World!" to the terminal.
Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.
Question 3: Write a shell script that takes a number as input from the user and prints it.



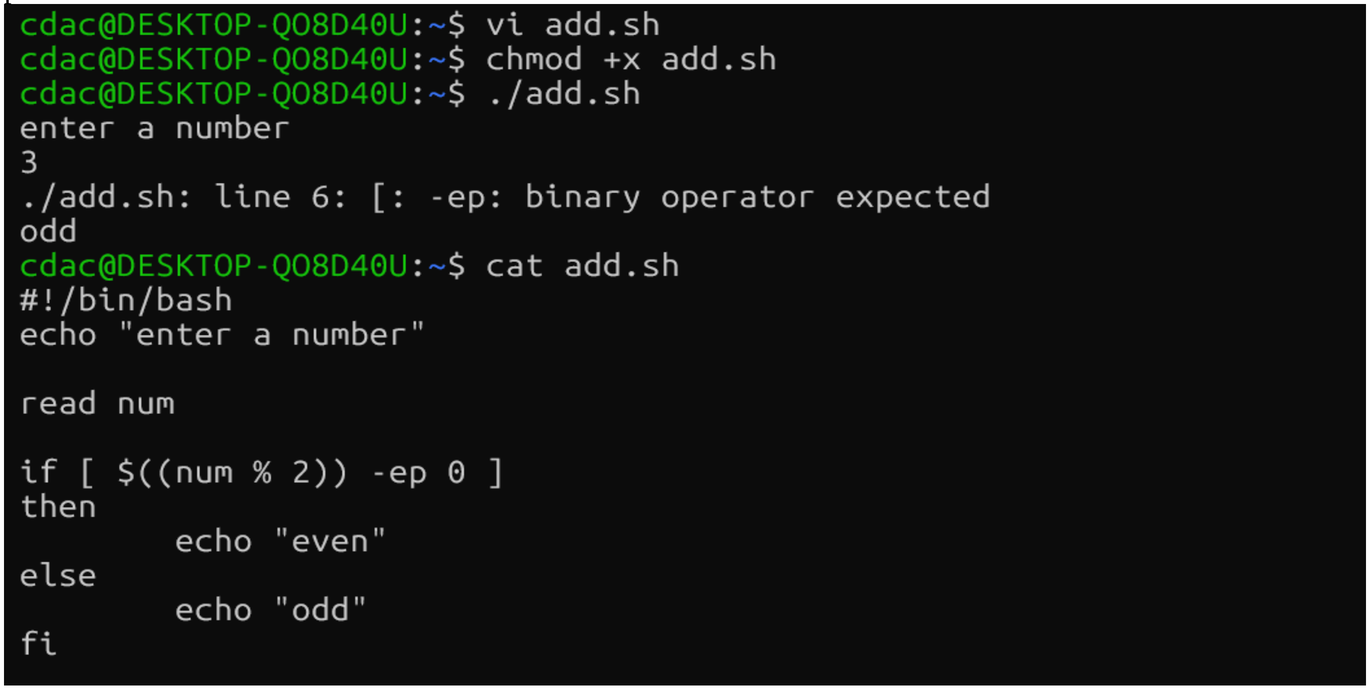
```
cdac@DESKTOP-QO8D40U:~$ vi ass.sh
cdac@DESKTOP-QO8D40U:~$ ./ass.sh
-bash: ./ass.sh: Permission denied
cdac@DESKTOP-QO8D40U:~$ chmod +x ass.sh
cdac@DESKTOP-QO8D40U:~$ ./ass.sh
hello, world
cdac@DESKTOP-QO8D40U:~$ vi ass.sh
cdac@DESKTOP-QO8D40U:~$ cat ass.sh
#!/bin/bash
name="CDAC Mumbai"
echo "the name is = $name"

cdac@DESKTOP-QO8D40U:~$ ./ass.sh
the name is = CDAC Mumbai
cdac@DESKTOP-QO8D40U:~$ vi ass.sh
cdac@DESKTOP-QO8D40U:~$ ./ass.sh
enter a number
2
number is : 2
```

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



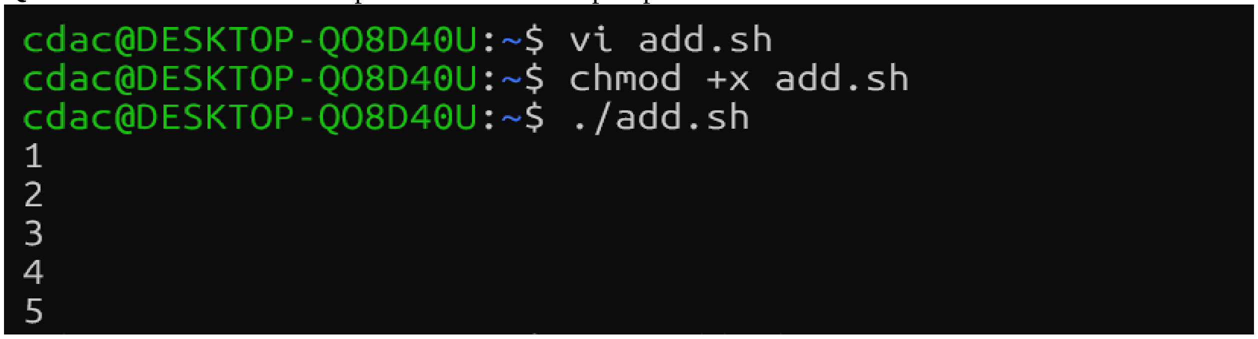
```
cdac@DESKTOP-QO8D40U:~$ vi add.sh
cdac@DESKTOP-QO8D40U:~$ ./add.sh
./add.sh: line 5: a: command not found
sum:
cdac@DESKTOP-QO8D40U:~$ cat add.sh
#!/bin/bash
a=1
b=2

sum=$( (a + b) )
echo "sum: $sum"


cdac@DESKTOP-QO8D40U:~$ vi add.sh
cdac@DESKTOP-QO8D40U:~$ ./add.sh
sum: '=3': No such file or directory
sum:
```

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



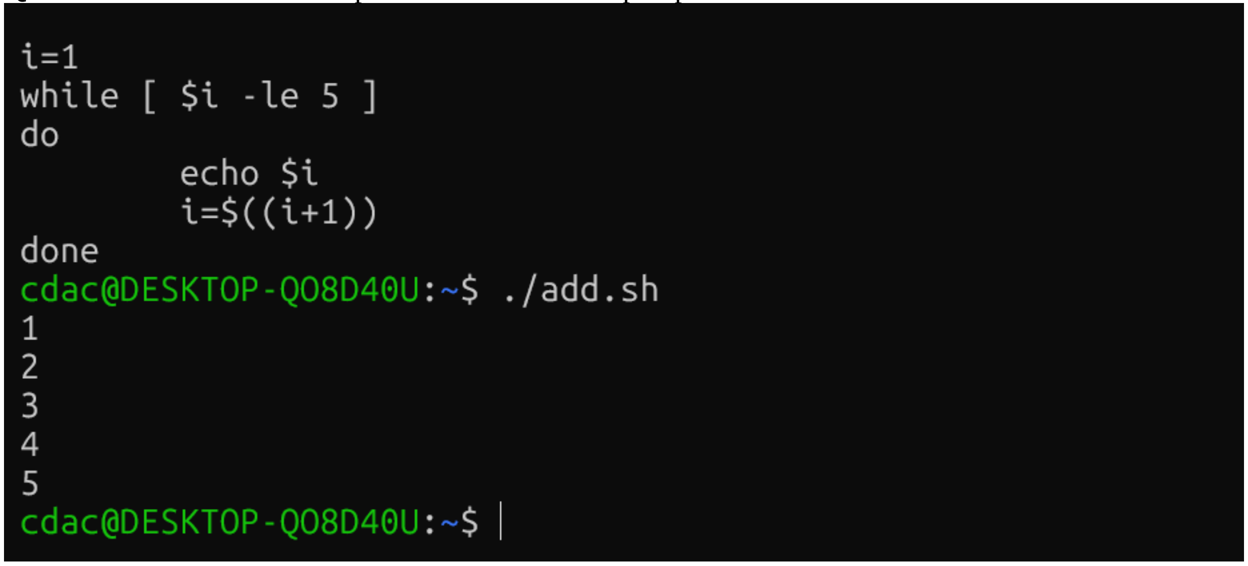
```
cdac@DESKTOP-QO8D40U:~$ vi add.sh
cdac@DESKTOP-QO8D40U:~$ chmod +x add.sh
cdac@DESKTOP-QO8D40U:~$ ./add.sh
enter a number
3
./add.sh: line 6: [: -ep: binary operator expected
odd
cdac@DESKTOP-QO8D40U:~$ cat add.sh
#!/bin/bash
echo "enter a number"

read num

if [ $((num % 2)) -ep 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.



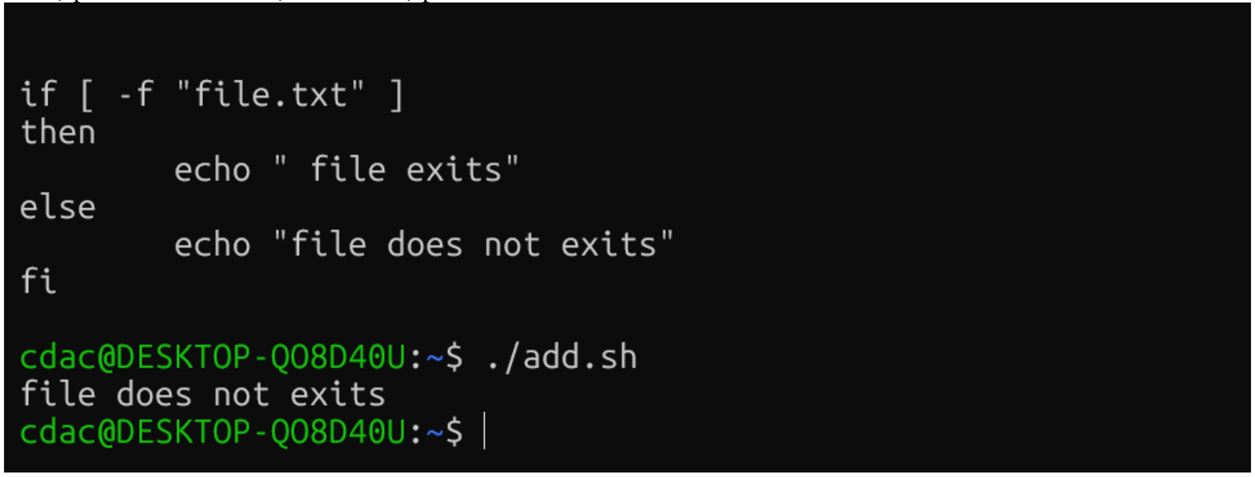
```
cdac@DESKTOP-QO8D40U:~$ vi add.sh
cdac@DESKTOP-QO8D40U:~$ chmod +x add.sh
cdac@DESKTOP-QO8D40U:~$ ./add.sh
1
2
3
4
5
```

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



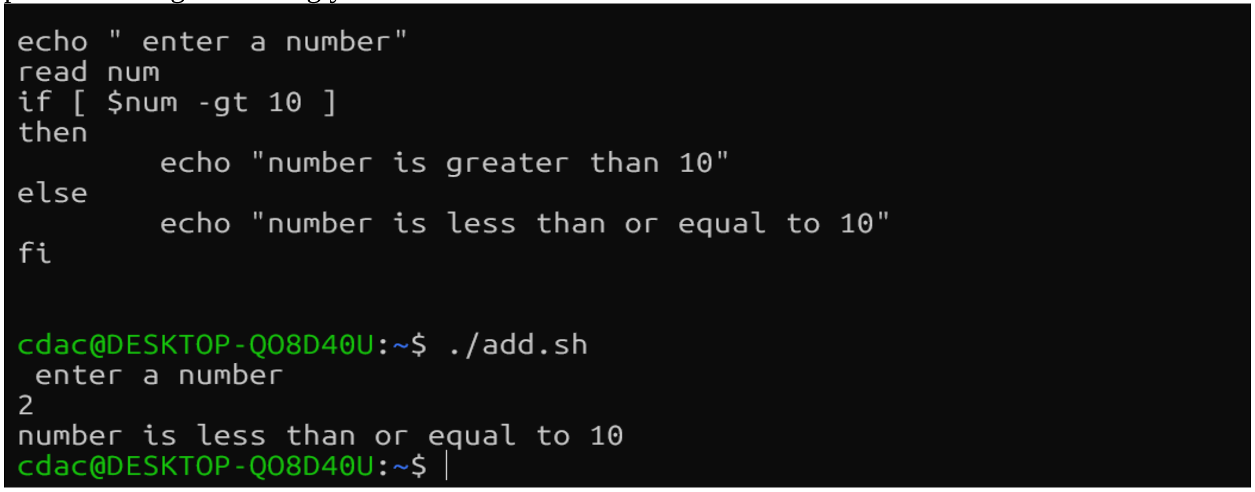
```
i=1
while [ $i -le 5 ]
do
        echo $i
        i=$((i+1))
done
cdac@DESKTOP-QO8D40U:~$ ./add.sh
1
2
3
4
5
cdac@DESKTOP-QO8D40U:~$
```

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".



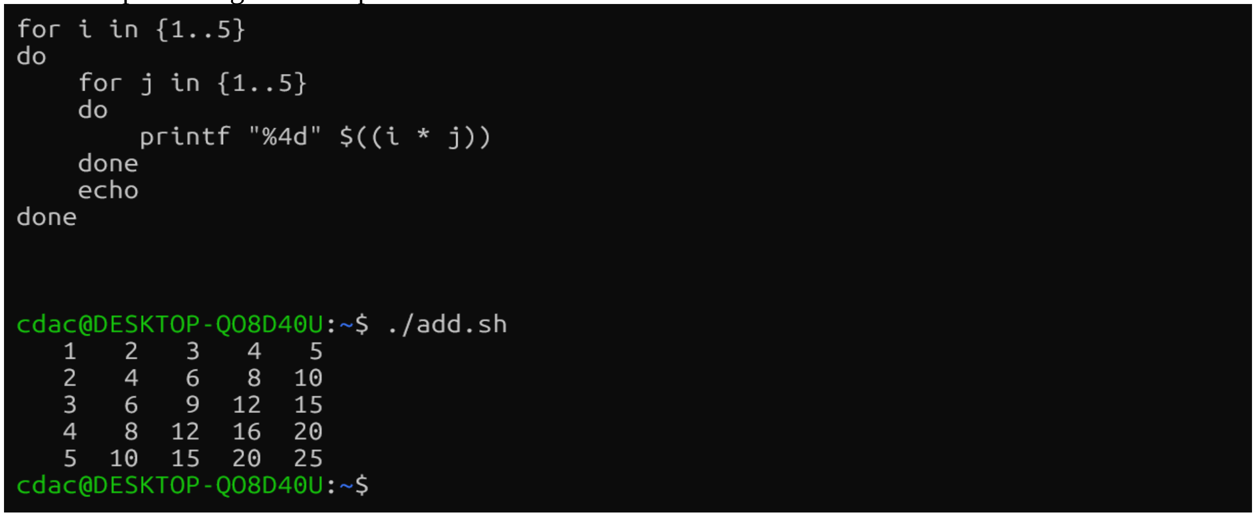
```
if [ -f "file.txt" ]
then
        echo " file exits"
else
        echo "file does not exits"
fi

cdac@DESKTOP-QO8D40U:~$ ./add.sh
file does not exits
cdac@DESKTOP-QO8D40U:~$
```

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.



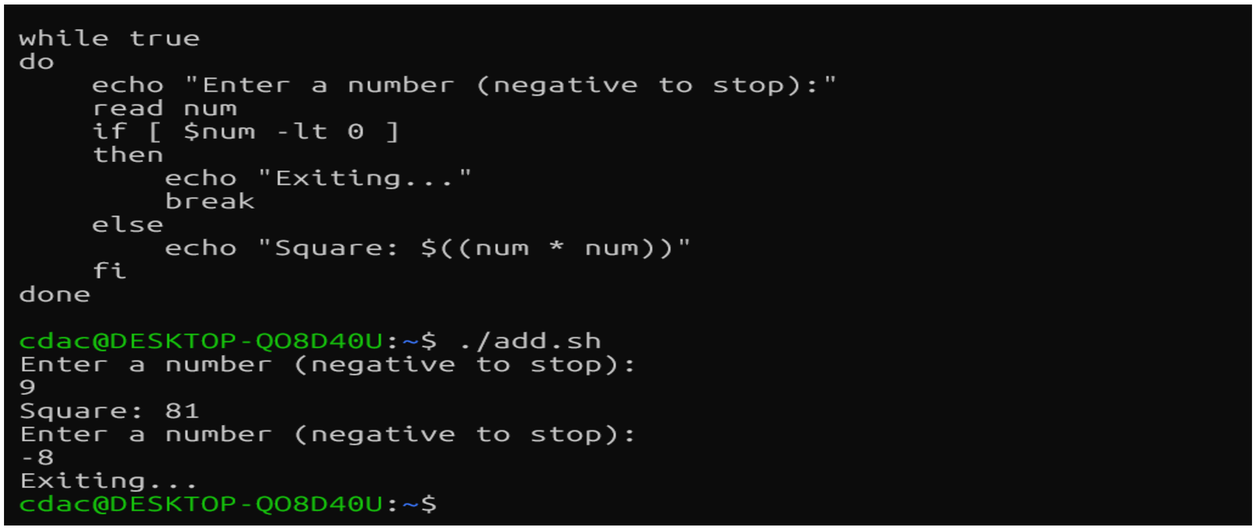
```
echo " enter a number"
read num
if [ $num -gt 10 ]
then
        echo "number is greater than 10"
else
        echo "number is less than or equal to 10"
fi


cdac@DESKTOP-QO8D40U:~$ ./add.sh
 enter a number
2
number is less than or equal to 10
cdac@DESKTOP-QO8D40U:~$ 
```

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 "%4d" $((i * j))
    done
    echo
done



cdac@DESKTOP-QO8D40U:~$ ./add.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-QO8D40U:~$
```

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 (negative to stop):"
    read num
    if [ $num -lt 0 ]
    then
        echo "Exiting..."
        break
    else
        echo "Square: $((num * num))"
    fi
done

cdac@DESKTOP-QO8D40U:~$ ./add.sh
Enter a number (negative to stop):
9
Square: 81
Enter a number (negative to stop):
-8
Exiting...
cdac@DESKTOP-QO8D40U:~$
```

<u>Part D</u>

Common Interview Questions (Must know)

1. What is an operating system, and what are its primary functions?
   An operating system is system software that manages computer hardware and software resources and provides services for applications. Its main functions are process management, memory management, file management, device management, security, and providing a user interface.
2. Explain the difference between process and thread.
   A process is an independent program in execution with its own memory space, while a thread is a lightweight unit of execution inside a process that shares memory and resources with other threads.
3. What is virtual memory, and how does it work?
   Virtual memory is a technique that allows the computer to use both RAM and disk storage to create the illusion of a large memory space. It works by keeping active pages in RAM and moving inactive pages to disk using paging or swapping.
4. Describe the difference between multiprogramming, multitasking, and multiprocessing.
   Multiprogramming means several programs are kept in memory and the CPU executes them one at a time. Multitasking means the CPU switches between tasks so fast that it looks like they run at the same time. Multiprocessing means more than one CPU works in parallel to execute processes.
5. What is a file system, and what are its components?
   A file system controls how data is stored and retrieved in a computer. It consists of files, directories, metadata, and structures like inodes or allocation tables that keep track of file locations.
6. What is a deadlock, and how can it be prevented?
   Deadlock is a situation where processes wait forever for resources held by each other. It can be prevented by avoiding circular waits, using resource ordering, allowing preemption, or using algorithms like the banker's algorithm.
7. Explain the difference between a kernel and a shell.
   The kernel is the core of the operating system that directly manages hardware, while the shell is the user interface, such as the command line, that allows users to communicate with the kernel.
8. What is CPU scheduling, and why is it important?
   CPU scheduling is the method the operating system uses to decide which process gets the CPU next. It is important to ensure efficiency, fairness, and better response times.
9. How does a system call work?
   A system call is a way for a user program to request services from the operating system kernel, such as reading a file. It works by switching from user mode to kernel mode to execute the request.
10. What is the purpose of device drivers in an operating system?

Device drivers are small software programs that allow the operating system to communicate with hardware devices like printers, keyboards, or storage drives.

11. Explain the role of the page table in virtual memory management.
    The page table is used in virtual memory to map virtual addresses generated by programs to physical addresses in RAM.
12. What is thrashing, and how can it be avoided?
    Thrashing happens when the system spends more time swapping pages in and out of memory than doing useful work. It can be avoided by using better page replacement algorithms, increasing RAM, or using working set models.
13. Describe the concept of a semaphore and its use in synchronization.
    A semaphore is a synchronization tool used to control access to shared resources by processes. It helps avoid race conditions and ensures mutual exclusion.
14. How does an operating system handle process synchronization?
    An operating system handles process synchronization by using tools such as locks, semaphores, monitors, and message passing to ensure processes do not interfere with each other.
15. What is the purpose of an interrupt in operating systems?
    An interrupt is a signal to the CPU that an event needs immediate attention, such as input from the keyboard or completion of an I/O operation
16. Explain the concept of a file descriptor.
    A file descriptor is a number that the operating system uses to identify an open file. For example, standard input is 0, standard output is 1, and standard error is 2.
17. How does a system recover from a system crash?
    A system recovers from a crash by using logs, checkpoints, or backups. Journaling file systems help restore consistency by recording changes before applying them.
18. Describe the difference between a monolithic kernel and a microkernel.
    A monolithic kernel runs all operating system services in kernel space, making it fast but less secure. A microkernel runs only essential services in kernel space and the rest in user space, making it more secure and modular but sometimes slower.
19. What is the difference between internal and external fragmentation?
    Internal fragmentation happens when memory is allocated in fixed blocks and some space inside the block is wasted. External fragmentation happens when free memory is scattered in small pieces that cannot be combined for larger requests.
20. How does an operating system manage I/O operations?
    The operating system manages I/O operations by using device drivers, interrupts, buffers, and spooling to handle communication with hardware efficiently.
21. Explain the difference between preemptive and non-preemptive scheduling.
    Preemptive scheduling means the operating system can stop a process and give the CPU to another, while non-preemptive scheduling means a process keeps the CPU until it finishes or blocks itself.
22. What is round-robin scheduling, and how does it work?
    Round robin scheduling gives each process a fixed time slice or quantum in a circular order, ensuring fairness and responsiveness.
23. Describe the priority scheduling algorithm. How is priority assigned to processes?
    Priority scheduling assigns a priority to each process, and the CPU is given to the process with the highest priority. To prevent starvation, priorities can be gradually increased over time (aging).
24. What is the shortest job next (SJN) scheduling algorithm, and when is it used?
    The shortest job next or shortest job first algorithm runs the process with the smallest CPU burst time first. It is often used in batch processing systems.
25. Explain the concept of multilevel queue scheduling.
    Multilevel queue scheduling divides processes into queues based on type, such as system, interactive, or batch processes, and each queue may use a different scheduling method.
26. What is a process control block (PCB), and what information does it contain?
    A process control block (PCB) is a data structure that stores information about a process, including its ID, state, program counter, registers, memory limits, and open files.
27. Describe the process state diagram and the transitions between different process states.

The process state diagram shows how a process moves between states like new, ready, running, waiting, and terminated depending on events such as scheduling, I/O requests, or completion.

28. How does a process communicate with another process in an operating system?
    Processes communicate with each other using inter-process communication methods such as pipes, message queues, shared memory, sockets, or signals.
29. What is process synchronization, and why is it important?
    Process synchronization ensures that processes accessing shared resources do not cause inconsistencies or errors. It is important to avoid race conditions and maintain data integrity.
30. Explain the concept of a zombie process and how it is created.
    A zombie process is a process that has finished execution but still remains in the process table because its parent has not yet collected its exit status.
31. Describe the difference between internal fragmentation and external fragmentation.
    Internal fragmentation wastes space inside allocated blocks, while external fragmentation wastes space because free memory is broken into small scattered parts.
32. What is demand paging, and how does it improve memory management efficiency?
    Demand paging loads pages into memory only when they are needed, reducing wasted memory space and improving efficiency.
33. Explain the role of the page table in virtual memory management.
    The page table is the data structure that maps virtual memory addresses to physical memory addresses during execution.
34. How does a memory management unit (MMU) work?
    The memory management unit (MMU) is hardware that translates virtual addresses from programs into physical addresses in RAM using the page table.
35. What is thrashing, and how can it be avoided in virtual memory systems?
    Thrashing in virtual memory means the CPU is overloaded with page swaps. It can be avoided by efficient memory allocation, better replacement policies, and adding more RAM.
36. What is a system call, and how does it facilitate communication between user programs and the operating system?
    A system call is the interface between a user program and the kernel, allowing user programs to request services like reading files or creating processes.
37. Describe the difference between a monolithic kernel and a microkernel.
    In a monolithic kernel, all OS services run in kernel mode, while in a microkernel only the essential parts run in kernel mode and other services run in user mode.
38. How does an operating system handle I/O operations?
    The operating system handles I/O operations using interrupts, device drivers, and scheduling methods. A race condition occurs when two or more processes access shared resources at the same time and the result depends on the order of execution. It can be prevented by using synchronization techniques like locks and semaphores.
39. Explain the concept of a race condition and how it can be prevented.
    race condition happens when two or more processes or threads access a shared resource at the same time, and the final result depends on the order in which they run. synchronization mechanisms
40. Describe the role of device drivers in an operating system.
    Device drivers are special programs that act as a bridge between the operating system and hardware devices. They allow the OS to communicate with hardware such as printers, keyboards, disks, or network cards, by translating high-level OS commands into device-specific instructions.
41. What is a zombie process, and how does it occur? How can a zombie process be prevented?
    A zombie process is a process that has completed execution but still remains in the process table because its parent has not collected its exit status. It occurs when the parent process fails to call wait() or waitpid() after the child exits. To prevent zombies, the parent should properly wait for child processes, or the system init process (PID 1) will eventually clean them up if the parent terminates.

42. Explain the concept of an orphan process. How does an operating system handle orphan processes?
An orphan process is a child process whose parent has terminated before it finishes execution. The operating system handles orphan processes by automatically reassigning them to the init process (or systemd in modern Linux), which then takes care of cleaning them up when they finish.
43. What is the relationship between a parent process and a child process in the context of process management?
In process management, a parent process is the one that creates another process, called the child, usually using the fork() system call. The parent and child have separate process IDs but may share resources like files or memory depending on how they were created.
44. How does the fork() system call work in creating a new process in Unix-like operating systems?
The fork() system call is used in Unix-like systems to create a new process. It duplicates the parent process into a child process with its own unique process ID. After fork(), both parent and child continue executing the same program, but they can be distinguished because fork() returns 0 to the child and the child's PID to the parent.
45. Describe how a parent process can wait for a child process to finish execution.
A parent process can wait for a child process to finish using the wait() system call. This suspends the parent until the child terminates, allowing the parent to collect the child's exit status and release system resources.
46. What is the significance of the exit status of a child process in the wait() system call?
The exit status of a child process is important in the wait() system call because it tells the parent how the child finished—whether it completed successfully, failed, or was terminated abnormally. This helps the parent handle errors or continue processing based on results.
47. How can a parent process terminate a child process in Unix-like operating systems?
A parent process can terminate a child process in Unix-like systems by sending it a signal, usually with the kill command or kill() system call. For example, sending SIGKILL immediately stops the child process.
48. Explain the difference between a process group and a session in Unix-like operating systems.
A process group is a collection of one or more processes that can be signaled together, usually for job control. A session is a collection of process groups that are associated with a controlling terminal. Sessions are useful for managing login shells and background jobs.
49. Describe how the exec() family of functions is used to replace the current process image with a new one.
The exec() family of functions is used to replace the current process image with a new program. After exec() is called, the current program code is completely replaced by the new one, while the process ID stays the same. It is often used right after fork() to let the child run a different program.
50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?
The waitpid() system call allows a parent to wait for a specific child process to finish, identified by its process ID. Unlike wait(), which waits for any child, waitpid() gives the parent more control, such as waiting for a particular child or checking the status without blocking.
51. How does process termination occur in Unix-like operating systems?
Process termination in Unix-like systems happens when a process finishes execution and calls exit(), or when it is killed by a signal. The process's resources are released, but its exit status remains in the process table until the parent collects it.
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?
The long-term scheduler, also called the job scheduler, controls which jobs are admitted into the system for processing. Its main role is to decide the degree of multiprogramming by controlling how many processes are in memory, balancing system load and performance.

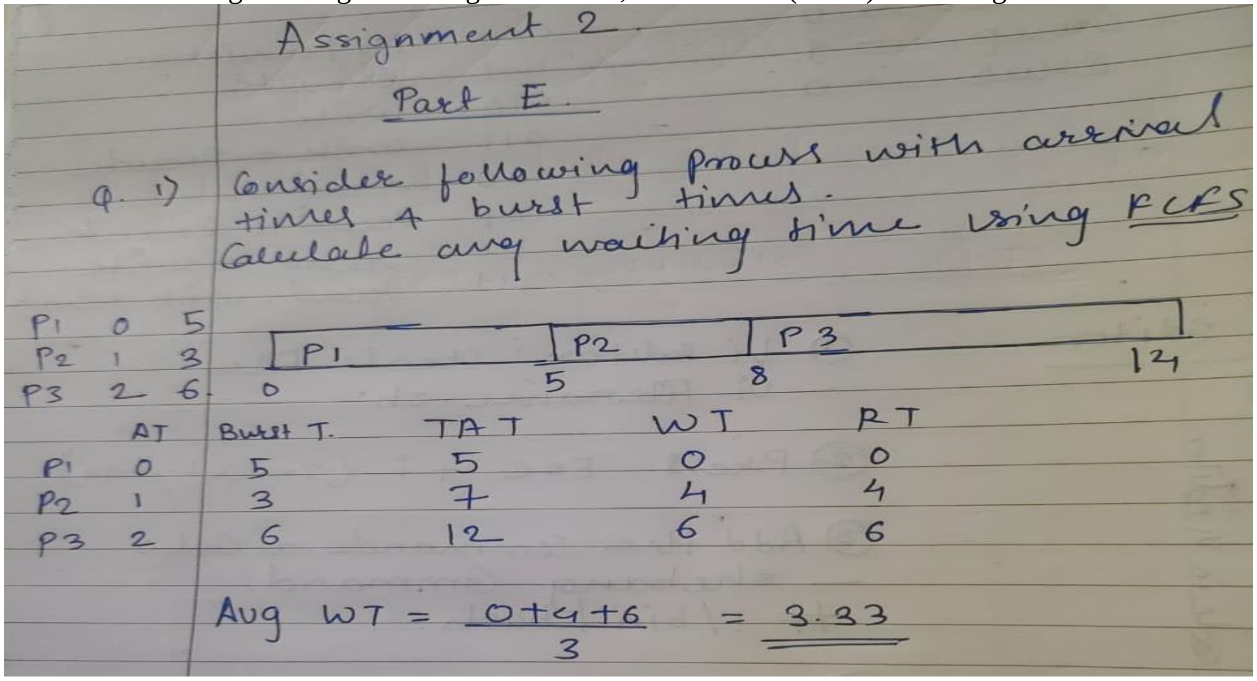
53. How does the short-term scheduler differ from the long-term and medium-term schedulers in terms of frequency of execution and the scope of its decisions?
The short-term scheduler, or CPU scheduler, decides which process in the ready queue gets the CPU next. It runs very frequently (milliseconds) compared to the long-term scheduler, which runs less often, and the medium-term scheduler, which runs occasionally to suspend or resume processes.
54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.
The medium-term scheduler is used when the system is overloaded and resources are scarce. For example, it can suspend a process and move it to secondary storage to free up memory for other processes. This helps improve responsiveness and ensures better overall resource utilization.

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.



Assignment 2.

Part E.

Q. 1) Consider following process with arrival times & burst times. Calculate avg waiting time using FCFS

P1	0	5
P2	1	3
P3	2	6

```
| P1          | P2     | P3           |
0             5        8              14
```

	AT	Burst T.	TAT	WT	RT
P1	0	5	5	0	0
P2	1	3	7	4	4
P3	2	6	12	6	6

Avg WT = $\frac{0+4+6}{3}$ = 3.33

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

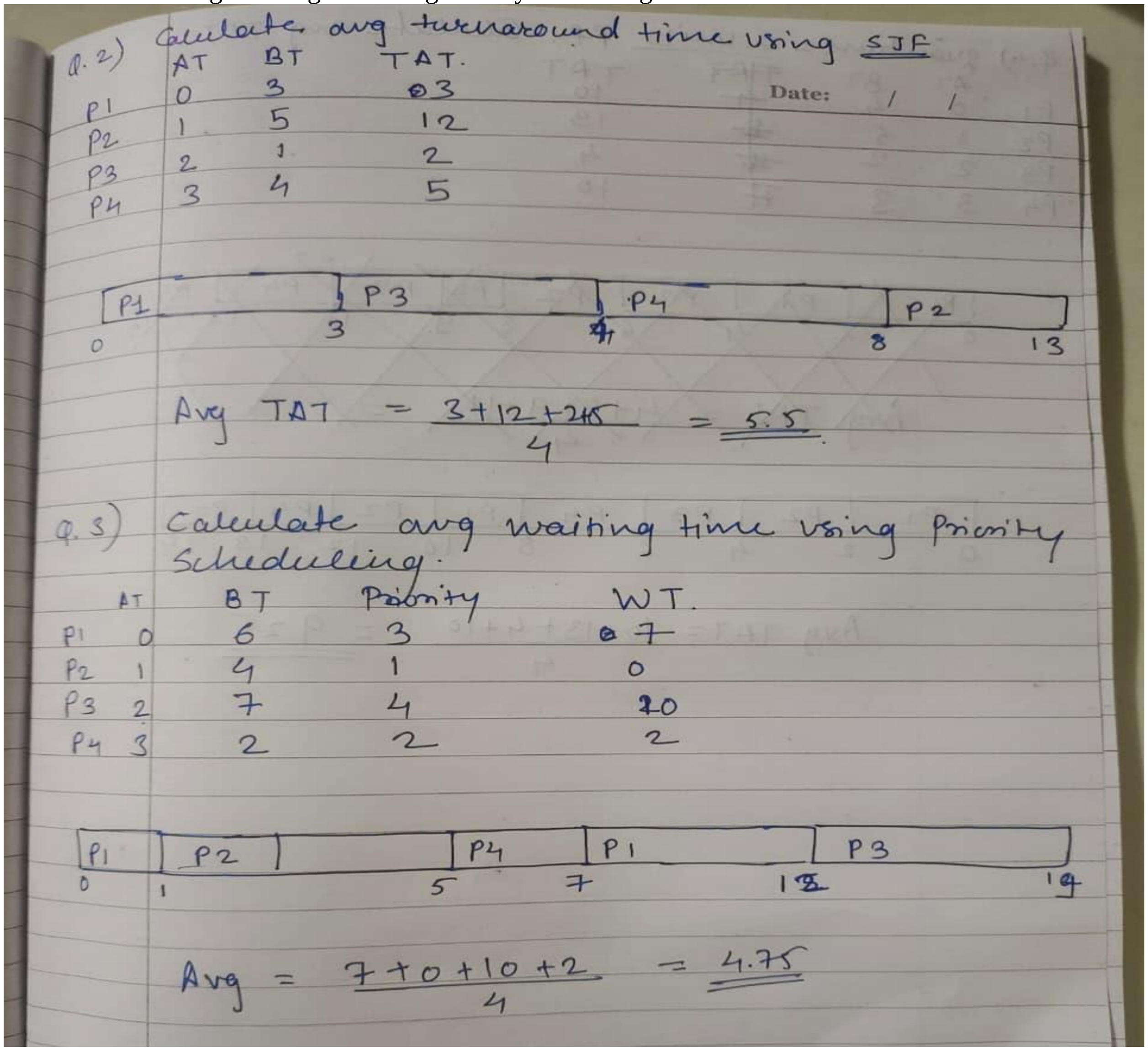
Process	Arrival Time	Burst Time
P1	0	3
P2	1	5
P3	2	1
P4	3	4

3. Calculate the average turnaround time using Shortest Job First (SJF) scheduling.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.



Q.2) Calculate avg turnaround time using SJF

Date: / /

	AT	BT	TAT.
P1	0	3	3
P2	1	5	12
P3	2	1	2
P4	3	4	5

| P1 | P3 | P4 | P2 |

0 3 4 8 13

$$\text{Avg TAT} = \frac{3+12+2+5}{4} = \underline{\underline{5.5}}$$

Q.3) Calculate avg waiting time using Priority Scheduling.

	AT	BT	Priority	WT.
P1	0	6	3	7
P2	1	4	1	0
P3	2	7	4	10
P4	3	2	2	2

| P1 | P2 | P4 | P1 | P3 |

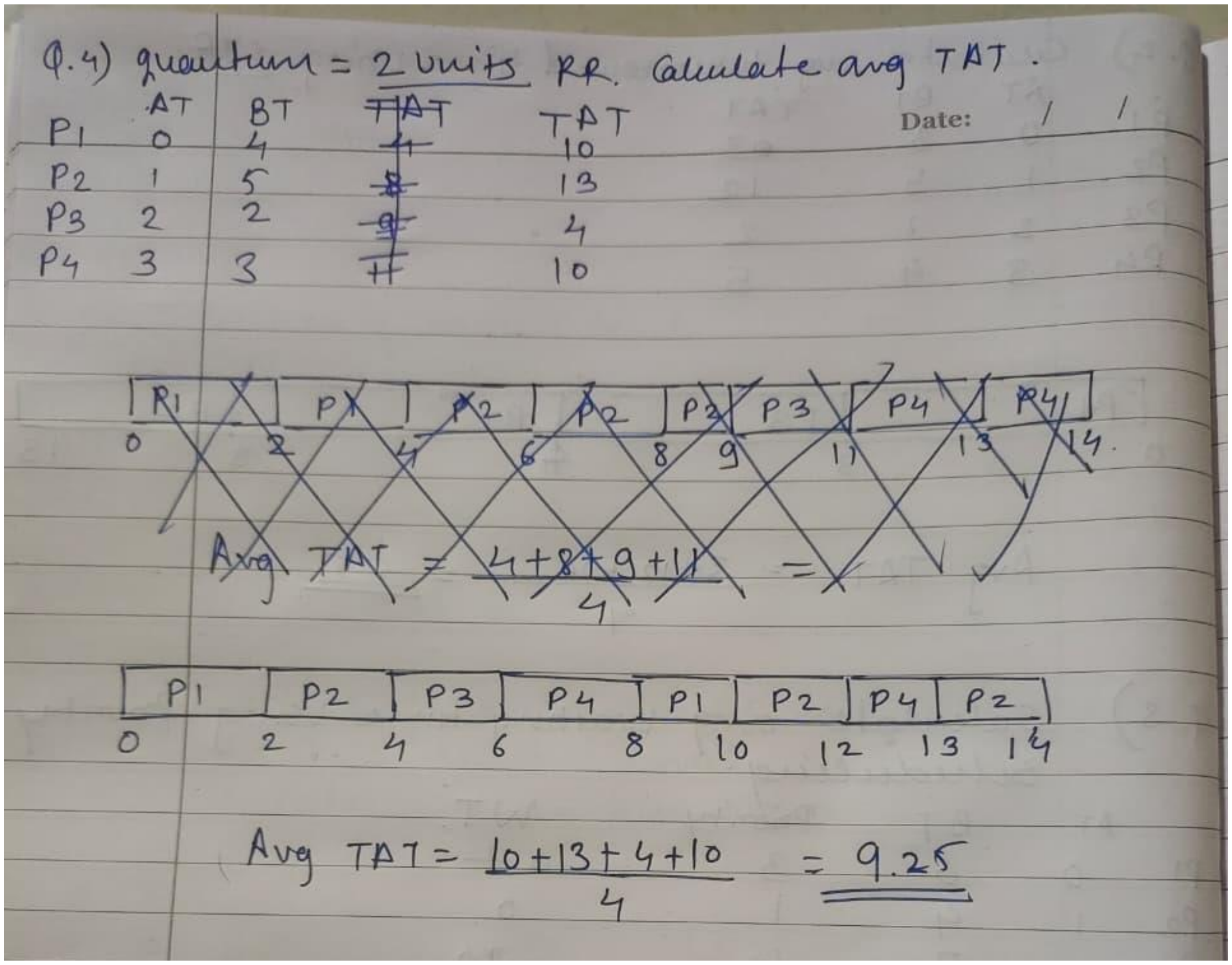
0 1 5 7 12 19

$$\text{Avg} = \frac{7+0+10+2}{4} = \underline{\underline{4.75}}$$

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


Q.4) quantum = 2 units RR. Calculate avg TAT.

	AT	BT	~~TAT~~	TAT
P1	0	4	~~4~~	10
P2	1	5	~~8~~	13
P3	2	2	~~9~~	4
P4	3	3	~~11~~	10

~~| P1 | P1 | P2 | P2 | P2 | P3 | P4 | P4 |~~
~~0 2 4 6 8 9 11 13 14.~~

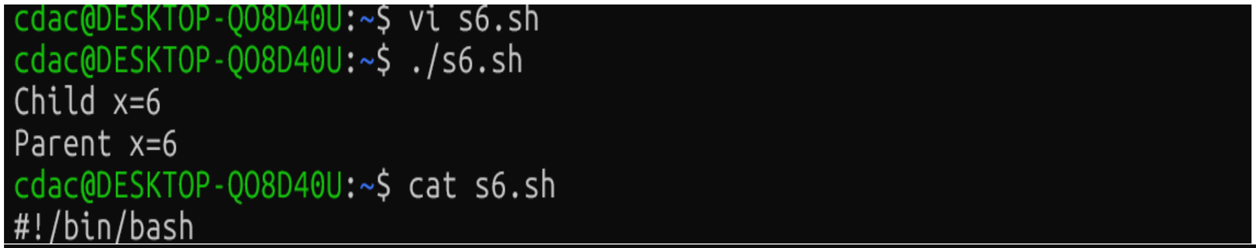
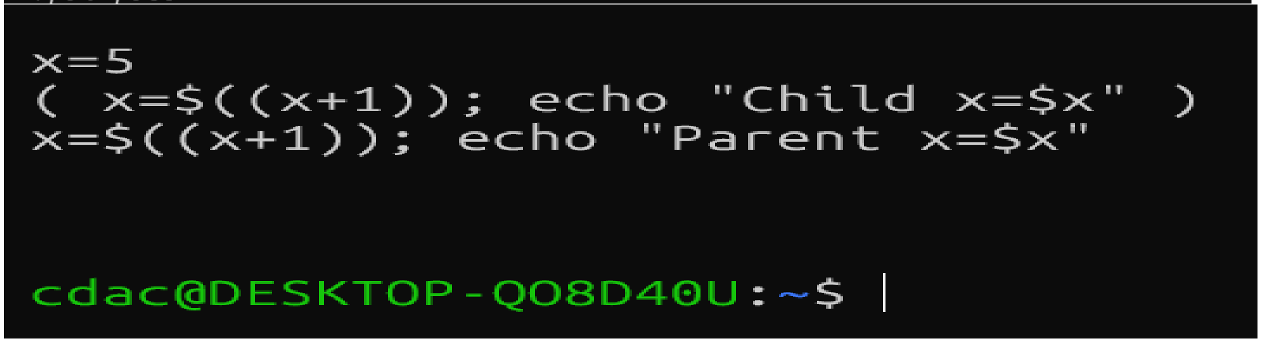
~~$\text{Avg TAT} = \frac{4+8+9+11}{4} =$~~

| P1 | P2 | P3 | P4 | P1 | P2 | P4 | P2 |
0 2 4 6 8 10 12 13 14

$$\text{Avg TAT} = \frac{10+13+4+10}{4} = \underline{\underline{9.25}}$$

.

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?


```
cdac@DESKTOP-QO8D40U:~$ vi s6.sh
cdac@DESKTOP-QO8D40U:~$ ./s6.sh
Child x=6
Parent x=6
cdac@DESKTOP-QO8D40U:~$ cat s6.sh
#!/bin/bash
```

```
x=5
( x=$((x+1)); echo "Child x=$x" )
x=$((x+1)); echo "Parent x=$x"


cdac@DESKTOP-QO8D40U:~$ |
```

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.