CDAC MUMBAI

Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

- echo "Hello, World!"
- name="Productive"
- touch file.txt
- ls -a
- rm file.txt
- cp file1.txt file2.txt
- mv file.txt /path/to/directory/
- chmod 755 script.sh
- grep "pattern" file.txt
- kill PID
- mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
- ls -l | grep ".txt"
- cat file1.txt file2.txt | sort | uniq
- ls -l | grep "^d"
- grep -r "pattern" /path/to/directory/
- cat file1.txt file2.txt | sort | uniq –d
- chmod 644 file.txt
- cp -r source_directory destination_directory
- find /path/to/search -name "*.txt"
- chmod u+x file.txt
- echo $PATH

Part A

What will the following commands do?

- echo "Hello, World!" → This will print Hello, World
- name = "Productive" → It will store productive in name variable.
- touch file.txt → It will make a empty file named file.txt, if it doesn't exist, or updates time stamp of file.txt if it already exists.
- ls -a → It will list all files and directories in current directory, including the hidden files.
- rm file.txt → It removes the file named file.txt
- cp file1.txt file2.txt → It copies content of file1.txt to new file named 'file2.txt'.
- 'mv file1.txt /path/to/directory → It moves 'file1.txt' to the directory path/to/directory
- chmod 755 script.sh → It changes the permission of 'script.sh' to 755.

  which means → owner : read, write & execute
  → group : read & execute
  → others : read & execute

- grep "pattern" file.txt → It searches for the the string "pattern" in 'file.txt' & displays lines containing pattern.

- kill PID → It sends the termination signal to the process with the process ID 'PID'. Replace 'PID' with actual process ID.

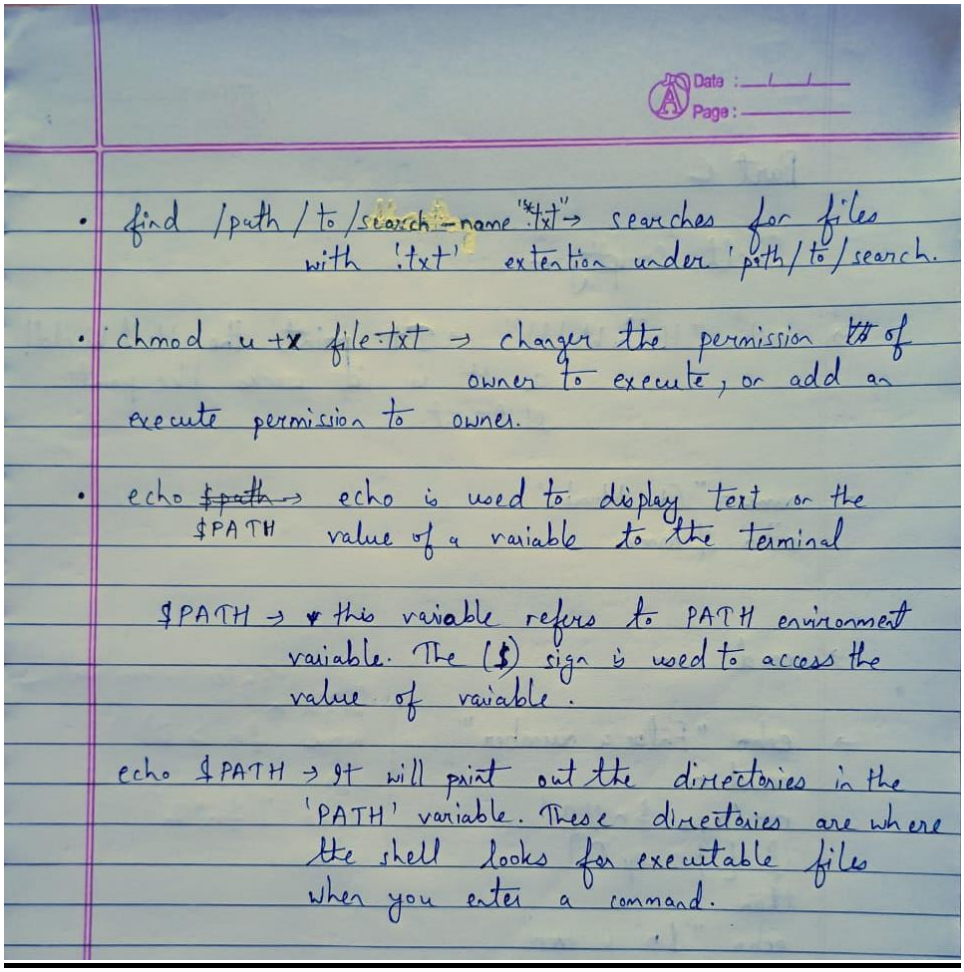
- ls -l | grep ".txt" → It lists files in long format & filters the list to show only lines containing '.txt' which will typically display only '.txt' files.

- 'cat file1.txt file2.txt | cont | uniq → It concatenates the content of file1.txt & file2.txt conts the combined output & removes duplicate lines, displaying only unique lines.

- ls -l | grep "^d" → Lists files in long format & filters the list to show only directories. Directories have 'd' at the begining of permission string.
'^' is a regular expression symbol that matches the begining of a line → Eg → directories data.txt

- grep -r "pattern" /path/to/directory/ → Recursively searches for string "pattern" in all files under /path/to/directory/

- chmod 644 file.txt → Changes permission to
owner : read & write
group : read
other : read



- find /path/to/search -name "*.txt" → searches for files with '.txt' extention under 'path/to/search.
- chmod u+x file.txt → changes the permission of owner to execute, or add an execute permission to owner.
- echo $path → echo is used to display text or the $PATH value of a variable to the terminal

$PATH → this variable refers to PATH environment variable. The ($) sign is used to access the value of variable.

echo $PATH → it will print out the directories in the 'PATH' variable. These directories are where the shell looks for executable files when you enter a command.

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. False

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
2. **cpy** is used to copy files and directories. cp
3. **mkfile** is used to create a new file. mkdir
4. **catx** is used to concatenate files. cat
5. **rn** is used to rename files. rm

Part C

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

```
root@DESKTOP-O9GM2B2:~/shellscripting# nano p5
root@DESKTOP-O9GM2B2:~/shellscripting# bash p5
Hello World!
root@DESKTOP-O9GM2B2:~/shellscripting#
```

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

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                    p5
echo "Hello World!"
name="Cdac Mumbai"
echo $name
```

```
root@DESKTOP-O9GM2B2:~/shellscripting# nano p5
root@DESKTOP-O9GM2B2:~/shellscripting# bash p5
Hello World!
Cdac Mumbai
root@DESKTOP-O9GM2B2:~/shellscripting#
```

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

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                    p5
echo "Enter a number"
read num
echo $num
```

```
root@DESKTOP-O9GM2B2: ~/shellscripting
root@DESKTOP-O9GM2B2:~/shellscripting# nano p5
root@DESKTOP-O9GM2B2:~/shellscripting# bash p5
Enter a number
80
80
root@DESKTOP-O9GM2B2:~/shellscripting#
```

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

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                  p6
x=3
y=5
Res=`expr $x + $y`
echo $Res
```

```
root@DESKTOP-O9GM2B2:~/shellscripting# nano p6
root@DESKTOP-O9GM2B2:~/shellscripting# bash p6
8
root@DESKTOP-O9GM2B2:~/shellscripting#
```

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

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                  p7
echo "Enter a number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
echo "$n is even"
else
echo "$n is odd"
fi
```

```
root@DESKTOP-O9GM2B2: ~/shellscripting
root@DESKTOP-O9GM2B2:~/shellscripting# nano p7
root@DESKTOP-O9GM2B2:~/shellscripting# bash p7
Enter a number
8
8 is even
root@DESKTOP-O9GM2B2:~/shellscripting# bash p7
Enter a number
9
9 is odd
root@DESKTOP-O9GM2B2:~/shellscripting#
```

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

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                  p8 *
a=0
for a in 1 2 3 4 5
do
echo $a
done
```

```
root@DESKTOP-O9GM2B2: ~/shellscripting
root@DESKTOP-O9GM2B2:~/shellscripting# nano p8
root@DESKTOP-O9GM2B2:~/shellscripting# bash p8
1
2
3
4
5
root@DESKTOP-O9GM2B2:~/shellscripting#
```

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

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                  p9
a=0
while [ $a -lt 6 ]
do
echo $a
done
```

```
root@DESKTOP-O9GM2B2:~/shellscripting# nano p9
root@DESKTOP-O9GM2B2:~/shellscripting# bash p8
1
2
3
4
5
root@DESKTOP-O9GM2B2:~/shellscripting#
```

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".

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                  p11
if [ -f "file.txt" ]
then
echo "File exists"
else
echo "File does not exist"
fi
```

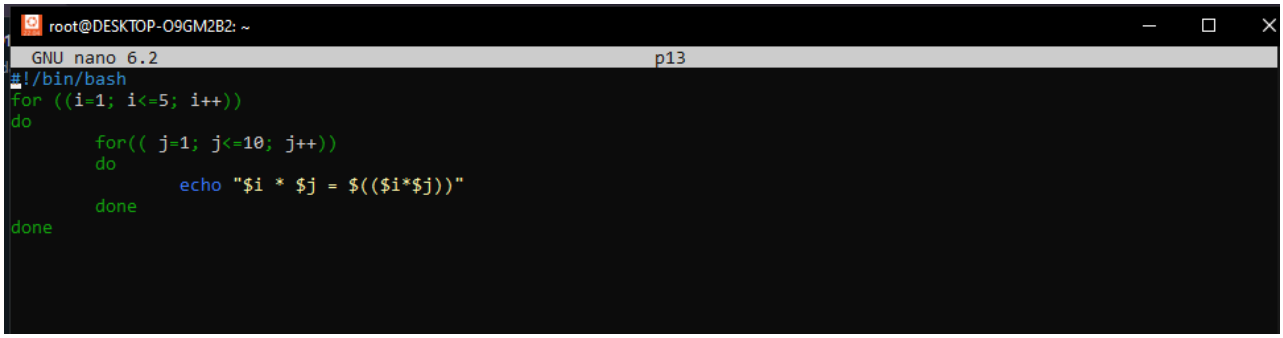
```
root@DESKTOP-O9GM2B2: ~/shellscripting
root@DESKTOP-O9GM2B2:~/shellscripting# bash p11
File does not exist
root@DESKTOP-O9GM2B2:~/shellscripting# touch file.txt
root@DESKTOP-O9GM2B2:~/shellscripting# bash p11
File exists
root@DESKTOP-O9GM2B2:~/shellscripting#
```

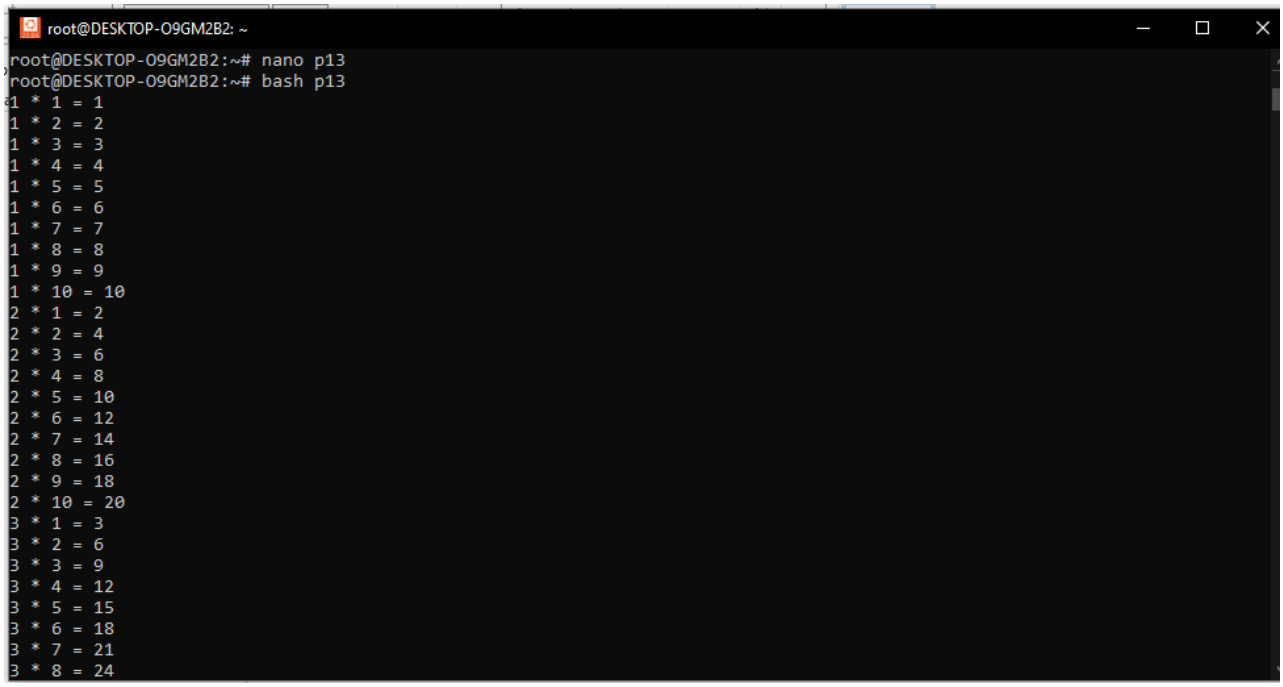
Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.

```
root@DESKTOP-O9GM2B2: ~/shellscripting
  GNU nano 6.2                                  p10
echo "Enter a number"
read Num
if [ $Num -gt 10 ]
then
echo "$Num is greater than 10"
else
echo "$Num is smaller than 10"
fi
```

```
root@DESKTOP-O9GM2B2:~/shellscripting# nano p10
root@DESKTOP-O9GM2B2:~/shellscripting# bash p10
Enter a number
60
60 is greater than 10
root@DESKTOP-O9GM2B2:~/shellscripting# bash p10
Enter a number
9
9 is smaller than 10
```

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.





```
root@DESKTOP-O9GM2B2: ~
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

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.

```
root@DESKTOP-O9GM2B2: ~
 GNU nano 6.2                                    p12
#!/bin/bash
while true

do

echo "Enter any number, if you type negative number then you are out"
read num

if [ $num -lt 0 ]
then

echo "Negative number you are out!"
break

fi

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

```
root@DESKTOP-O9GM2B2: ~
root@DESKTOP-O9GM2B2:~# bash p12
Enter any number, if you type negative number then you are out
40
Square of the number is: 1600
Enter any number, if you type negative number then you are out
-4
Negative number you are out!
root@DESKTOP-O9GM2B2:~#
```

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.

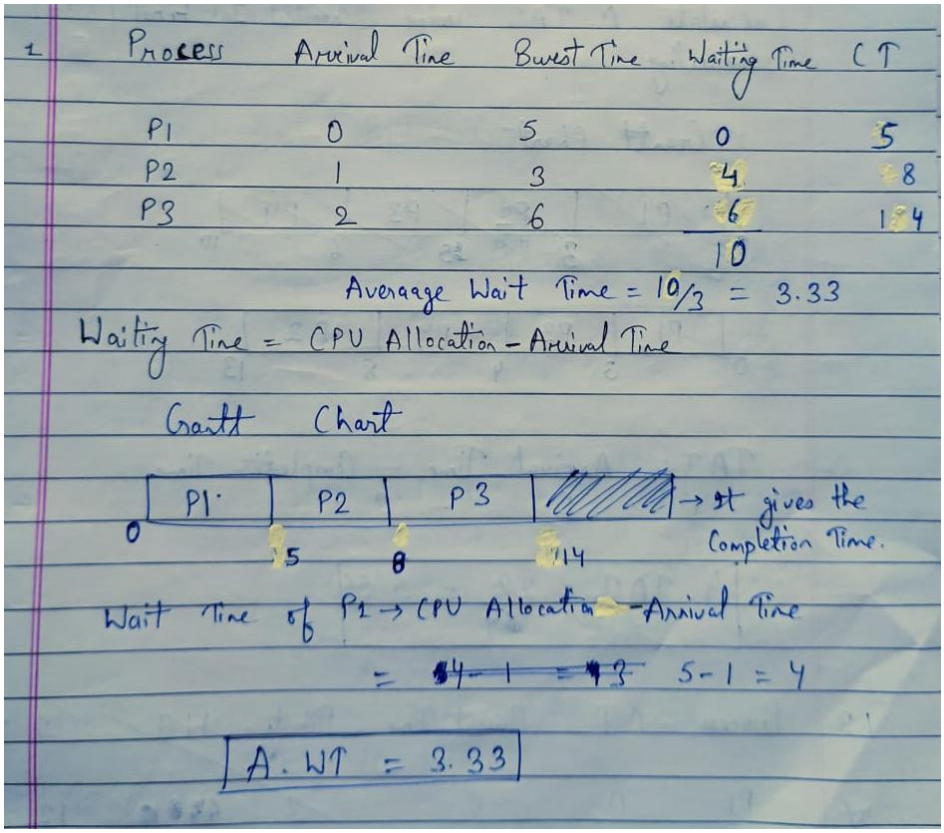
40. Describe the role of device drivers in an operating system.
41. What is a zombie process, and how does it occur? How can a zombie process be prevented?
42. Explain the concept of an orphan process. How does an operating system handle orphan processes?
43. What is the relationship between a parent 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.



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.

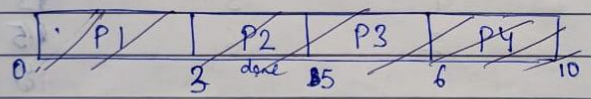
Date : / /
Page :

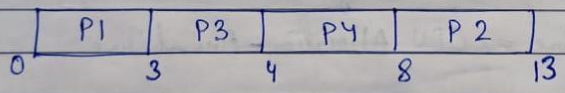
2.

Process	Arrival Time	Burst Time	Wait Time	TAT	CT
P1	0	3	0	3	3
P2	1	5	7	12	13
P3	2	1	1	2	4
P4	3	4	1	5	8
				22	

Calculate A. TAT using Shortest Job First (SJF) scheduling.

Gantt Chart





TAT = Arrival Time − Completion Time

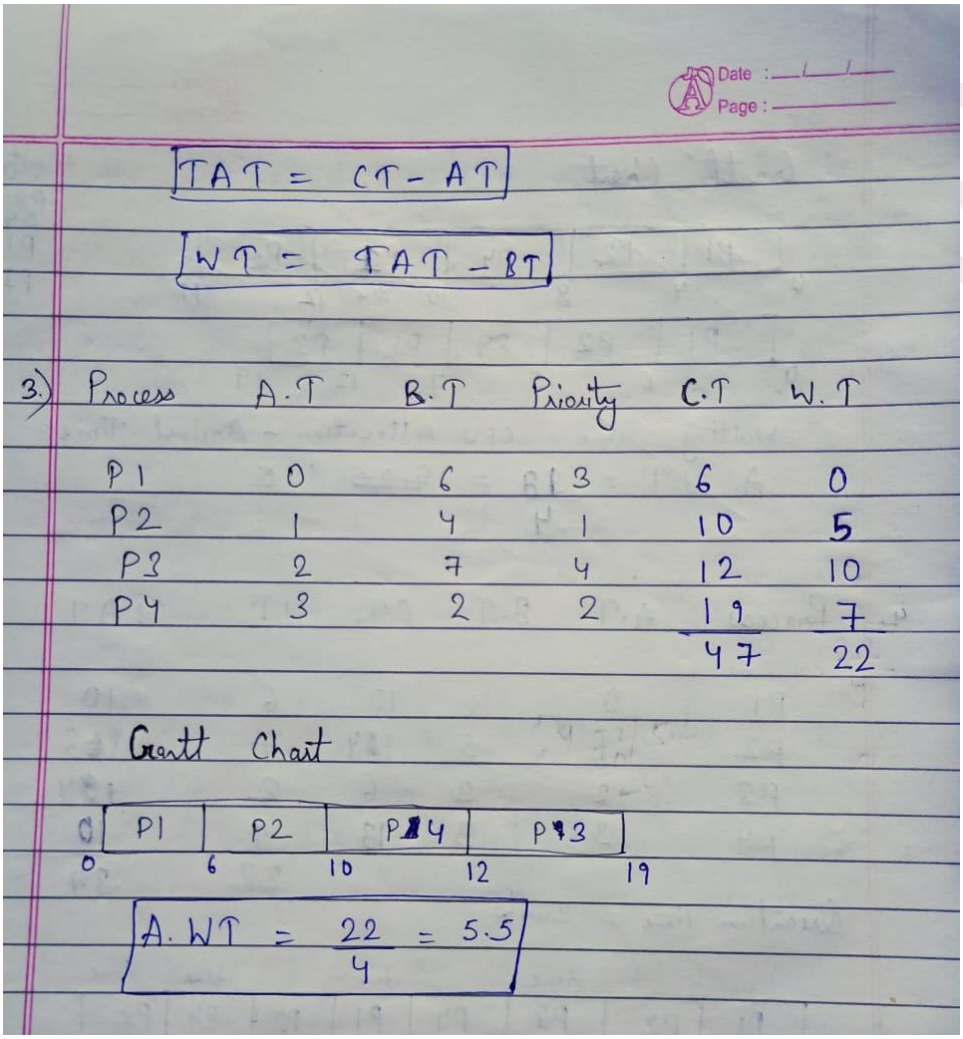
P1 = 0 − 3

$$\text{A. TAT} = \frac{22}{4} = 5.56$$

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

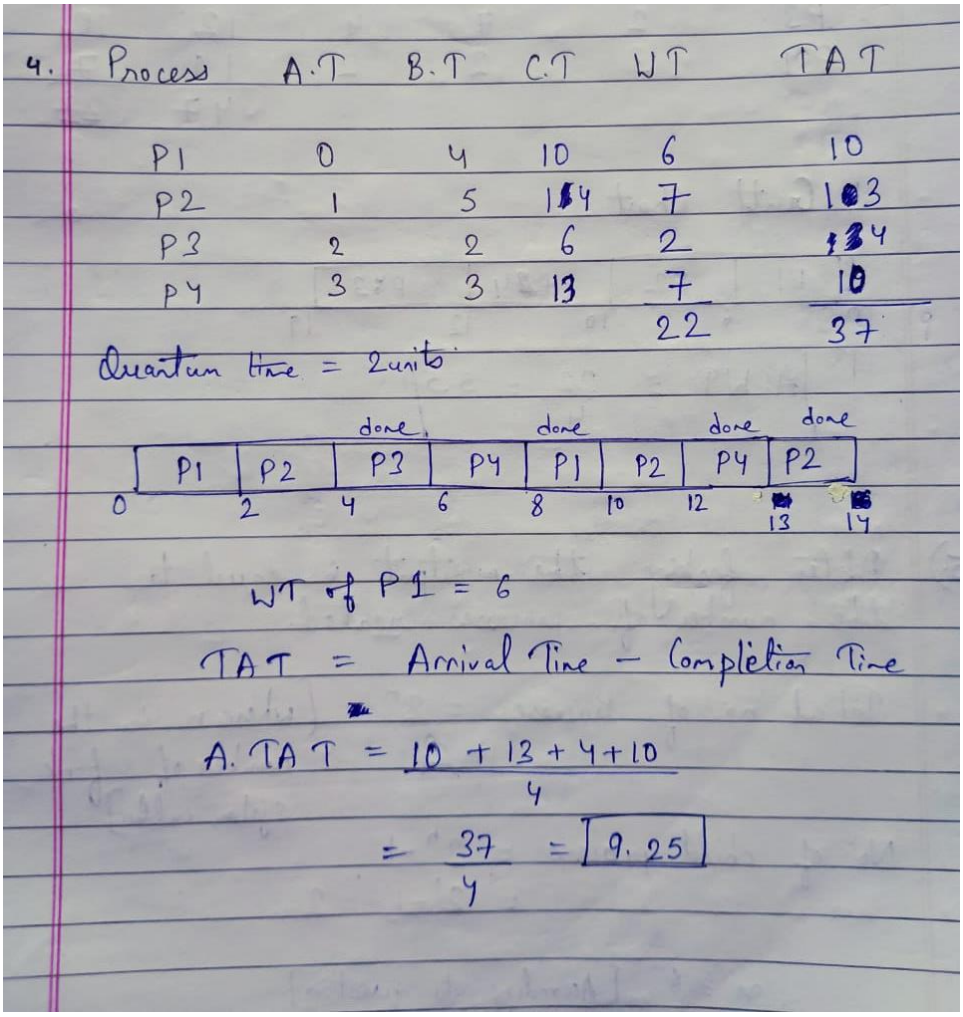
Calculate the average waiting time using Priority Scheduling.



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.



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;
print("%d\n",x);
return 0;
}
```

Output -> 6
6

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.