Assignment 2 OS

Name : Utsav Vijay Gavli_KH

Part A :

1.echo "Hello, World!" - Prints "Hello, World!" to the terminal.

2.name="Productive" - Assigns the value "Productive" to the variable name (only for the current shell session).

3.touch file.txt - Creates an empty file named file.txt or updates its last modified timestamp if it already exists.

4.ls -a - Lists all files and directories, including hidden ones (those starting with .).

5.rm file.txt - Deletes the file file.txt.

6.cp file1.txt file2.txt - Copies the contents of file1.txt to file2.txt. If file2.txt exists, it will be overwritten.

7.mv file.txt /path/to/directory/ - Moves file.txt to the specified directory.

8.chmod 755 script.sh - Changes the file permissions of script.sh to rwxr-xr-x, making it executable for the owner and readable/executable for others.

9.grep "pattern" file.txt - Searches for occurrences of "pattern" in file.txt and prints matching lines.

10.kill PID -Terminates the process with the given PID (Process ID).

11. mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt -

Creates a directory mydir, navigates into it, creates file.txt, writes "Hello, World!" into it, and then displays the file's contents.

12.ls -l | grep ".txt" - Lists all files in long format and filters results to display only those containing .txt in their names.

13.cat file1.txt file2.txt | sort | uniq - Concatenates file1.txt and file2.txt, sorts the lines, and removes duplicates.

14.ls -l | grep "^d" - Lists only directories (^d indicates entries starting with d, which represents directories in ls -l output).

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

16.cat file1.txt file2.txt | sort | uniq -d - Concatenates file1.txt and file2.txt, sorts the lines, and prints only duplicate lines.

17.chmod 644 file.txt - Sets permissions for file.txt to rw-r--r--, allowing the owner to read/write and others to only read.

18.cp -r source_directory destination_directory - Recursively copies source_directory and its contents to destination_directory.

19.find /path/to/search -name "*.txt" - Searches for all .txt files in /path/to/search and its subdirectories.

20.chmod u+x file.txt - Grants the owner (u) execute (+x) permission for file.txt.

21.echo $PATH -Displays the system's PATH environment variable, which lists directories where executable files are searched for.

Part B :

Identify True or False:

1. 'ls' is used to list files and directories in a directory. - True

2. mv is used to move files and directories. - True

3. cd is used to copy files and directories. - False

4. pwd stands for "print working directory" and displays the current directory. - True

5. grep is used to search for patterns in files. - True

6. chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute permissions to group and others. - True

7. mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1 - True if directory1 does not exist.

8. rm -rf file.txt deletes a file forcefully without confirmation. - True

Identify the Incorrect Commands:

1. chmodx is used to change file permissions. - Incorrect (The correct command to change file permissions is chmod.)

2. cpy is used to copy files and directories. - Incorrect (The correct command to copy files and directories is cp.)

3. mkfile is used to create a new file. - Incorrect (The correct way to create a new file is touch filename or echo "" > filename.)

4. catx is used to concatenate files. - Incorrect (The correct command to concatenate and display file contents is cat.)

5. rn is used to rename files. - Incorrect (The correct command to rename (move) files is mv oldname newname.)

Part C

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

```
cdac@DESKTOP-UGL28VA:~$ echo "Hello World!"
Hello World!
```

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

```
cdac@DESKTOP-UGL28VA:~$ name="CDAC Mumbai"
cdac@DESKTOP-UGL28VA:~$ echo $name
CDAC Mumbai
```

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

```
cdac@DESKTOP-UGL28VA:~$ cat i
echo "Enter your number :"
read n
echo "you have entered the number $n"
cdac@DESKTOP-UGL28VA:~$ bash i
Enter your number :
12
you have entered the number 12
```

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

```
cdac@DESKTOP-UGL28VA:~$ cat i
a=10
b=12
sum=$(( a+b ))
echo "the sum of $a and $b is $sum"
cdac@DESKTOP-UGL28VA:~$ bash i
the sum of 10 and 12 is 22
```

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

```
cdac@DESKTOP-UGL28VA:~$ cat i
echo "Enter your number"
read n
if (( n%2==0 ))
then
echo "number $n is even"
else
echo "number $n is odd"
fi
cdac@DESKTOP-UGL28VA:~$ bash i
Enter your number
11
number 11 is odd
```

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

```
cdac@DESKTOP-UGL28VA:~$ cat i
for a in {1..5}
do
echo "$a"
done
cdac@DESKTOP-UGL28VA:~$ bash i
1
2
3
4
5
```

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

```
cdac@DESKTOP-UGL28VA:~$ cat i
a=1
while [ $a -lt 6 ]
do
echo "$a"
a=$(( a+1))
done
cdac@DESKTOP-UGL28VA:~$ bash i
1
2
3
4
5
```

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-UGL28VA:~$ ls
LinuxAssignment  command  docs  fibonnaci  file1  file2  i  output.txt  snap  utsav  y
cdac@DESKTOP-UGL28VA:~$ cat i
if [ -f "file.txt" ]
then
echo "file exists"
else
echo "file doesn't exist"
fi
cdac@DESKTOP-UGL28VA:~$ bash i
file doesn't exist
```

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-UGL28VA:~$ cat i
echo -n "enter your number:"
read n

if [ $n -gt 10 ]
then
echo "The number $n is greater than 10"
else
echo "The number $n is not greater than 10"
fi
cdac@DESKTOP-UGL28VA:~$ bash i
enter your number:11
The number 11 is greater 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.

```
cdac@DESKTOP-UGL28VA:~$ cat i
for i in {1..5}
do
echo "the mutliplication table of $i"
for j in {1..10}
do
echo "$i x $j"= $((i * j))
done
echo
done


cdac@DESKTOP-UGL28VA:~$ bash i
the mutliplication table of 1
1 x 1= 1
1 x 2= 2
1 x 3= 3
1 x 4= 4
1 x 5= 5
1 x 6= 6
1 x 7= 7
1 x 8= 8
1 x 9= 9
1 x 10= 10
```

```
the mutliplication table of 2
2 x 1= 2
2 x 2= 4
2 x 3= 6
2 x 4= 8
2 x 5= 10
2 x 6= 12
2 x 7= 14
2 x 8= 16
2 x 9= 18
2 x 10= 20

the mutliplication table of 3
3 x 1= 3
3 x 2= 6
3 x 3= 9
3 x 4= 12
3 x 5= 15
3 x 6= 18
3 x 7= 21
3 x 8= 24
3 x 9= 27
3 x 10= 30
```

```
the mutliplication table of 4
4 x 1= 4
4 x 2= 8
4 x 3= 12
4 x 4= 16
4 x 5= 20
4 x 6= 24
4 x 7= 28
4 x 8= 32
4 x 9= 36
4 x 10= 40

the mutliplication table of 5
5 x 1= 5
5 x 2= 10
5 x 3= 15
5 x 4= 20
5 x 5= 25
5 x 6= 30
5 x 7= 35
5 x 8= 40
5 x 9= 45
5 x 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.

```
cdac@DESKTOP-UGL28VA:~$ cat i
while true; do
 read -p "Enter a number: " num
 if [ $num -lt 0 ]
 then
 echo "the entered number is negative "
 break
 fi
 echo "Square: $((num * num))"
done



cdac@DESKTOP-UGL28VA:~$ bash i
Enter a number: 12
Square: 144
Enter a number: 23
Square: 529
Enter a number: -12
the entered number is negative
```

Part D
Common Interview Questions (Must know)

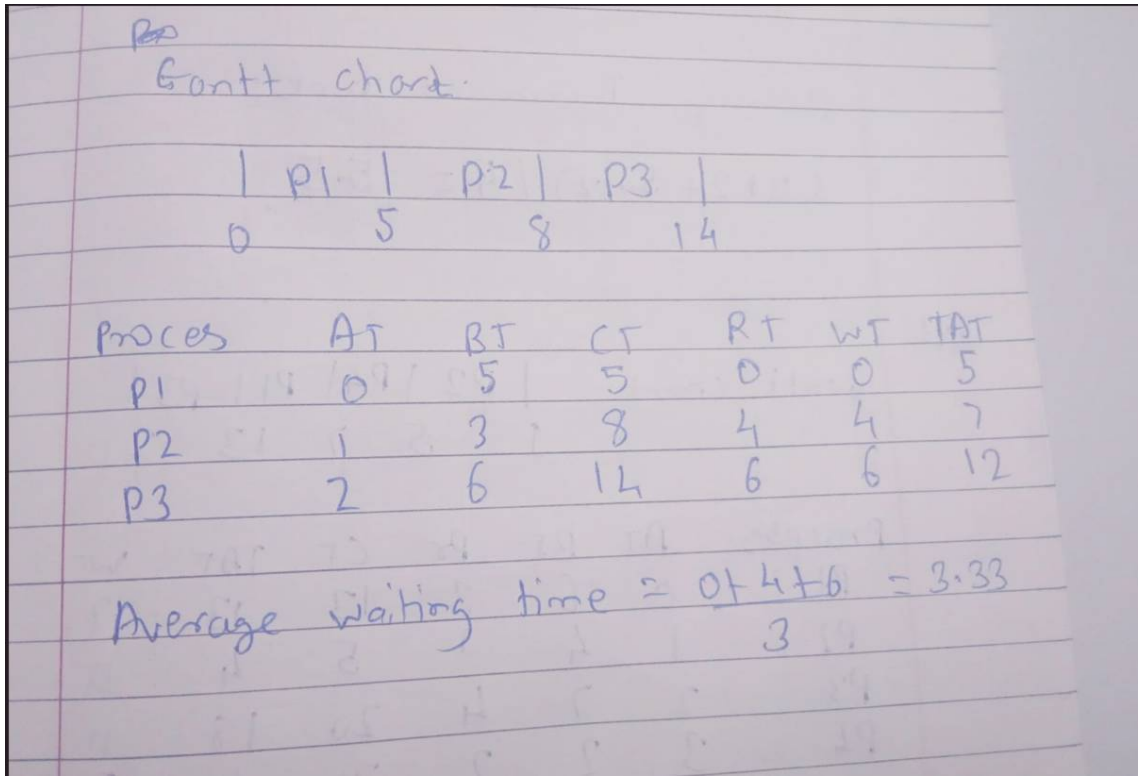
1. What is an operating system, and what are its primary functions?
2. Explain the difference between process and thread.
3. What is virtual memory, and how does it work?
4. Describe the difference between multiprogramming, multitasking, and multiprocessing.
5. What is a file system, and what are its 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.



Gantt chart

| P1 | P2 | P3 |
0 5 8 14

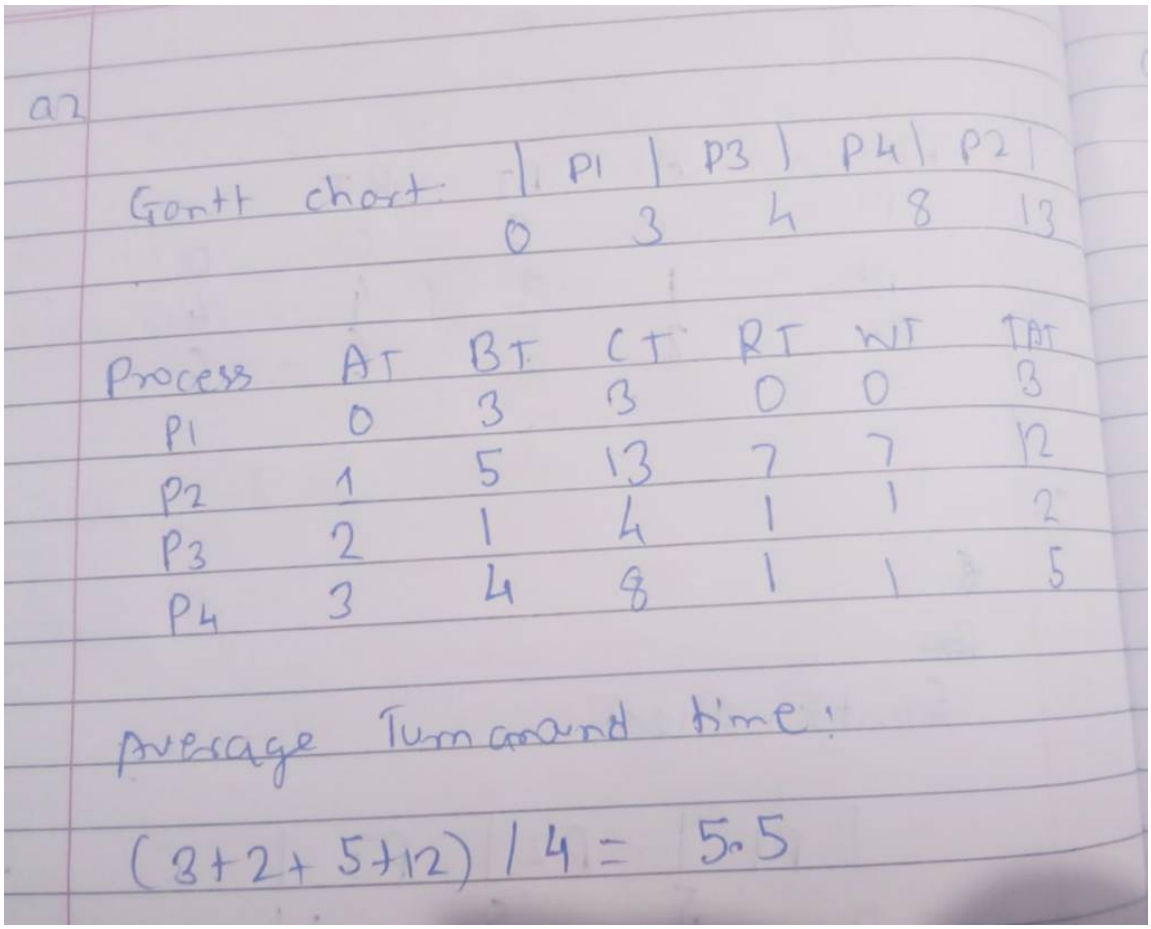
Proces	AT	BT	CT	RT	WT	TAT
P1	0	5	5	0	0	5
P2	1	3	8	4	4	7
P3	2	6	14	6	6	12

Average waiting time = $\frac{0+4+6}{3} = 3.33$

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

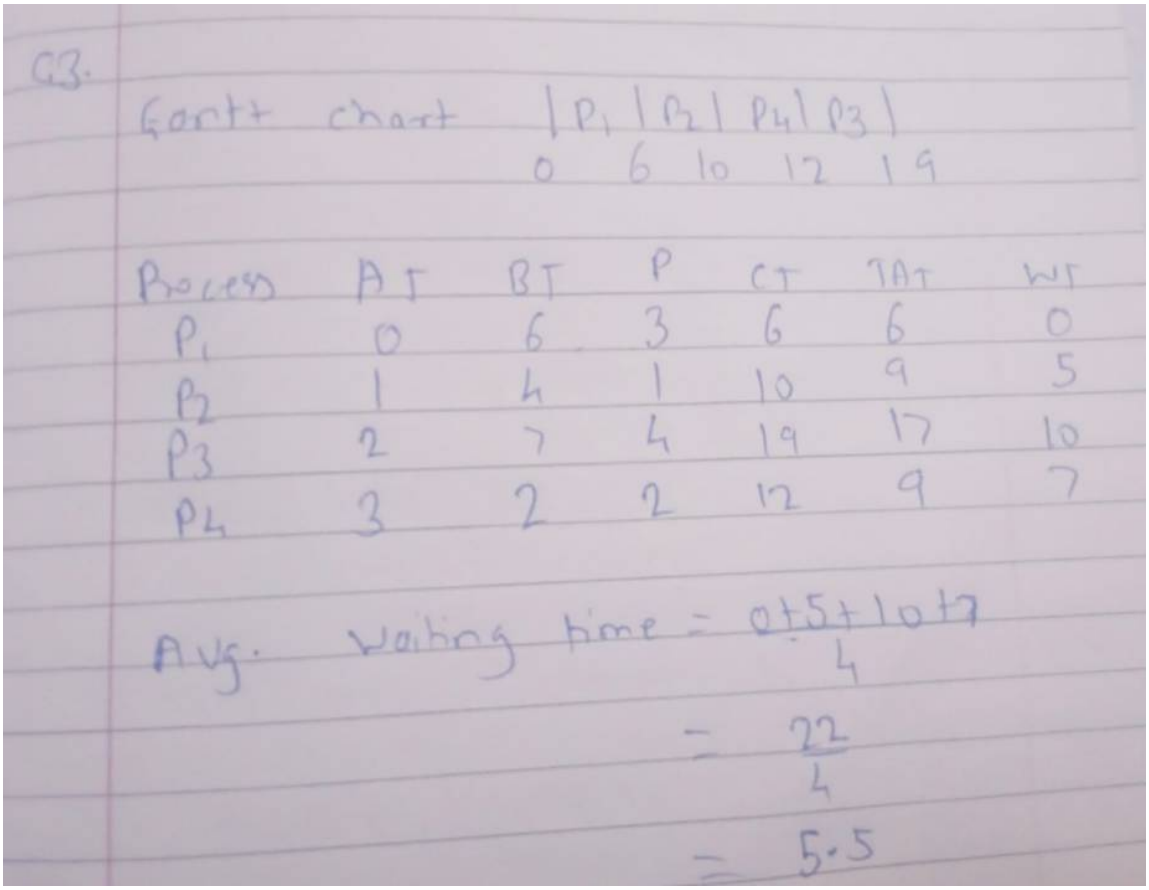
Calculate the average turnaround time using Shortest Job First (SJF) scheduling.



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.



Q3.

Gantt chart | P1 | P2 | P4 | P3 |
0 6 10 12 19

Process	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

Avg. Waiting time = $\frac{0+5+10+7}{4}$

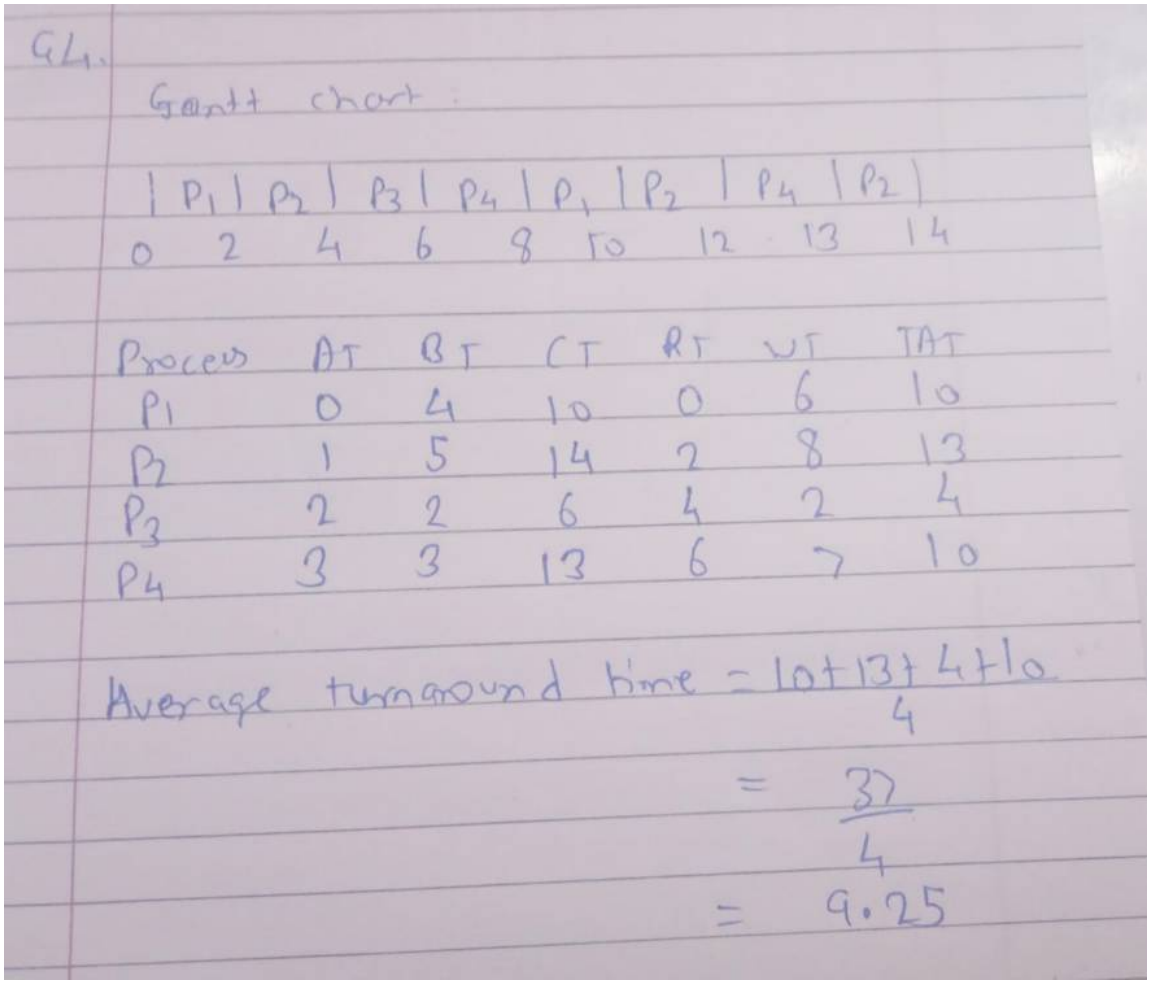
$= \frac{22}{4}$

$= 5.5$

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.



Q4.

Gantt chart:

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

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

Average turnaround time $= \frac{10+13+4+10}{4}$

$= \frac{37}{4}$

$= 9.25$

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?