Future Institute of Technology

Department of CSE (AI & ML)

Operating system Lab (PCC-CS592)

Student Name:…..…………..…………………………………………………

University Roll Number: ……..…………………..…………………………………

University Registration Number: …..……………………………………………..

Subject Name: …..…………………………………………………………….

Subject Code: …...……………………………………………………………………

Remarks: ………..………………………………………………………………………..

……………………………….. **………………………………..**

Student's Signature **Faculty Signature**

INDEX

Serial No.	***Experiment No.***	***Name of the Experiments***	***Remarks***

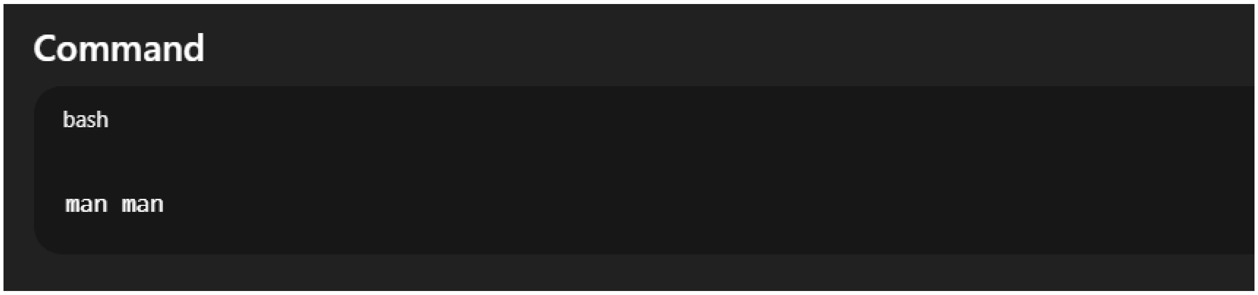
Assignment -1:

Perform the following tasks in the Linux shell with the help of appropriate commands:

a) Check the manual for the ‘man’ command.

Ans:

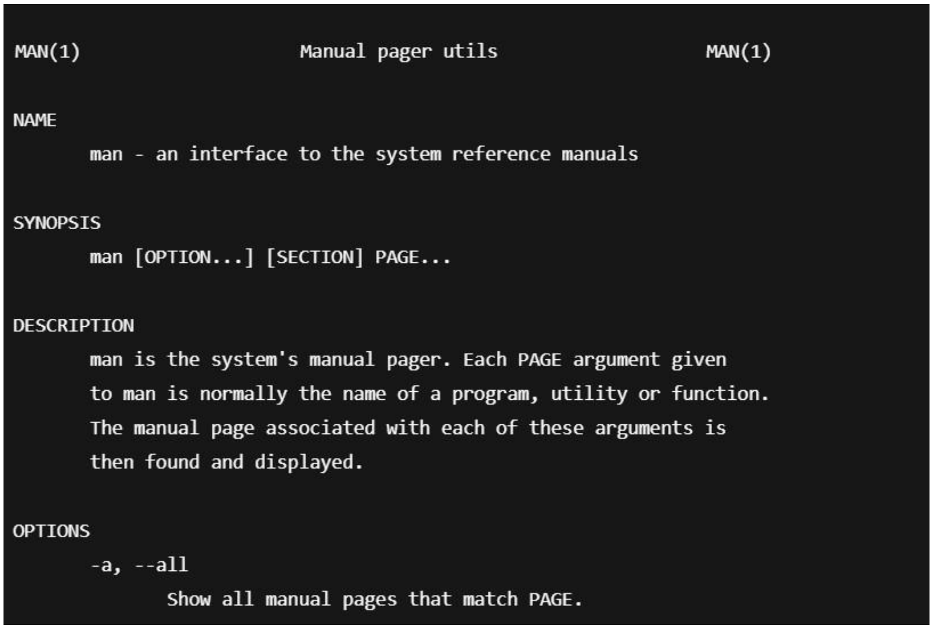
Code:



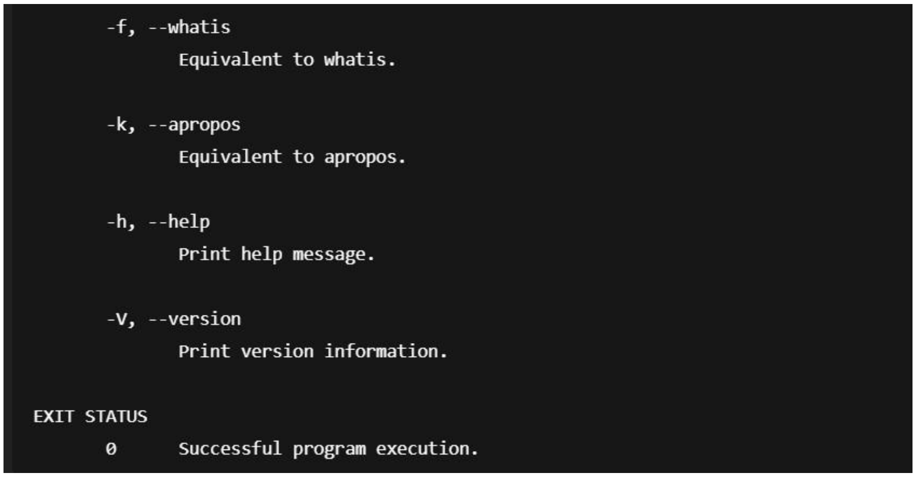
Command

```bash
man man
```

Output:



```
MAN(1)                    Manual pager utils                    MAN(1)

NAME
       man - an interface to the system reference manuals

SYNOPSIS
       man [OPTION...] [SECTION] PAGE...

DESCRIPTION
       man is the system's manual pager. Each PAGE argument given
       to man is normally the name of a program, utility or function.
       The manual page associated with each of these arguments is
       then found and displayed.

OPTIONS
       -a, --all
              Show all manual pages that match PAGE.
```

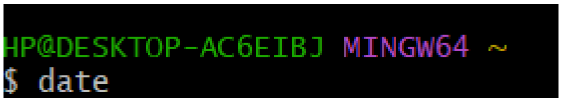


```
       -f, --whatis
              Equivalent to whatis.

       -k, --apropos
              Equivalent to apropos.

       -h, --help
              Print help message.

       -V, --version
              Print version information.

EXIT STATUS
       0      Successful program execution.
```

b) Display the current date and time.

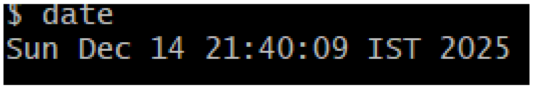
Ans:

Code:



```
HP@DESKTOP-AC6EIBJ MINGW64 ~
$ date
```

Output:

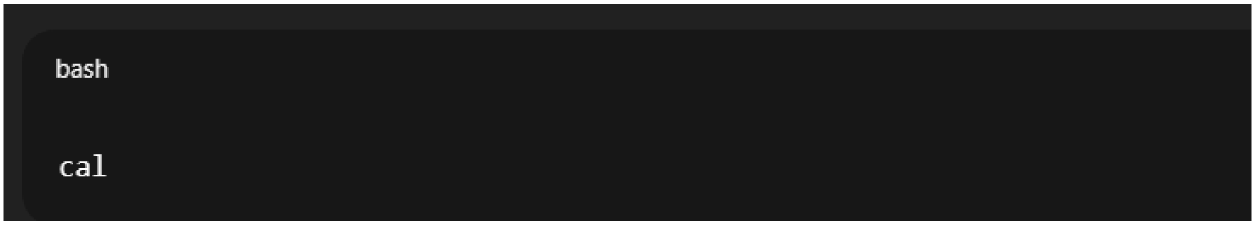


```
$ date
Sun Dec 14 21:40:09 IST 2025
```

c) Display the calendar of the current month.

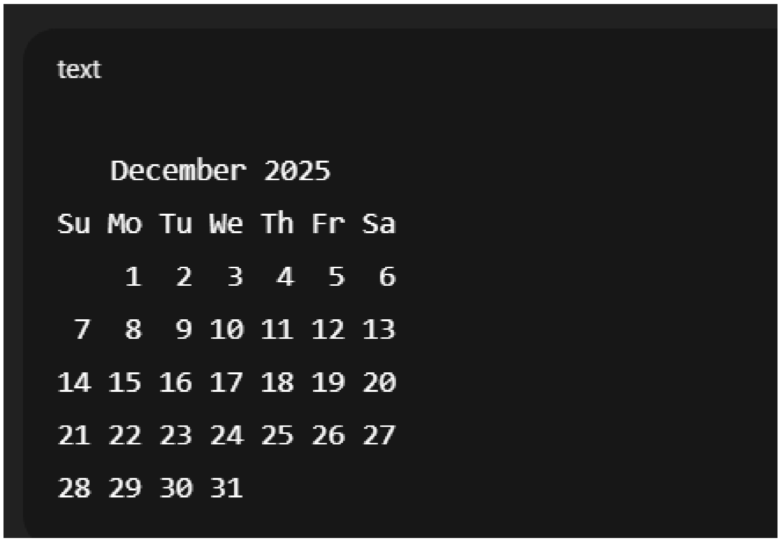
Ans:

Code:



```
bash

cal
```

Output:

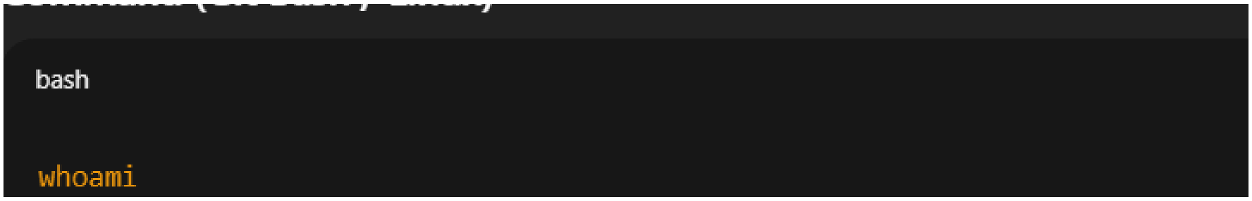


```
text

   December 2025
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

d) Display the current logged-in user.

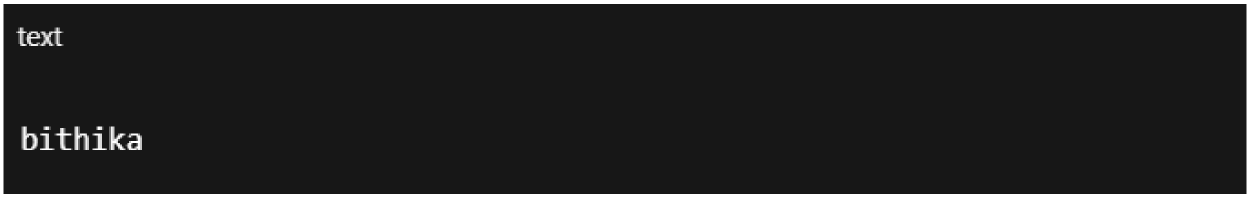
Ans:

Code:



```bash
whoami
```

Output:

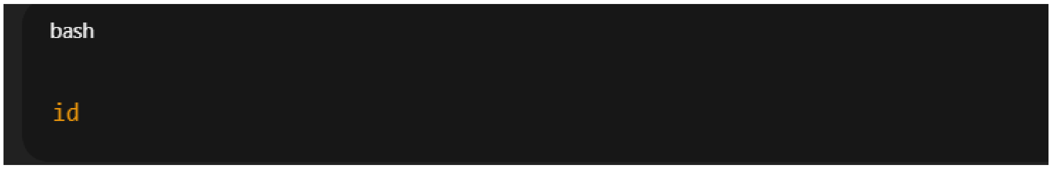


```
text

bithika
```

e) Display more information about the current logged-in user.

Ans:

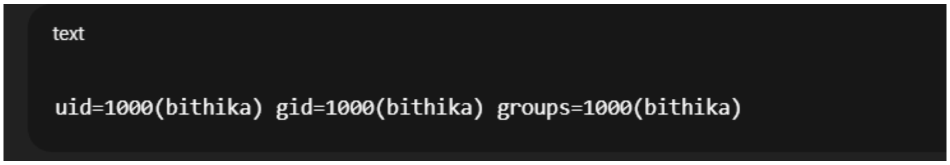
Code:



bash

```bash
id
```

Output:



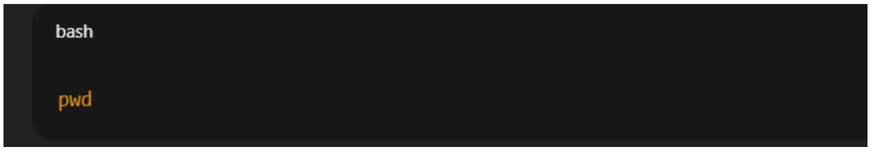
text

```text
uid=1000(bithika) gid=1000(bithika) groups=1000(bithika)
```

f) Display the current working directory.

Ans:

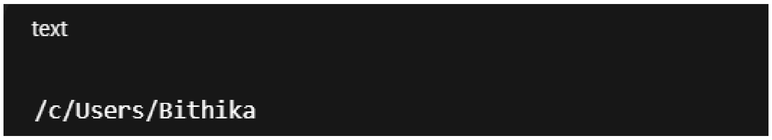
Code:



bash

```bash
pwd
```

Output:



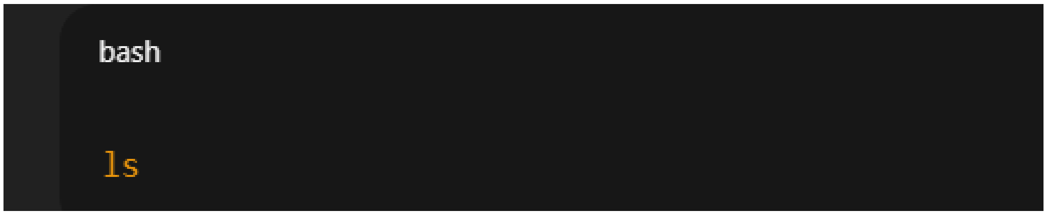
text

```text
/c/Users/Bithika
```

g) Check the contents of the current working directory.

Ans:

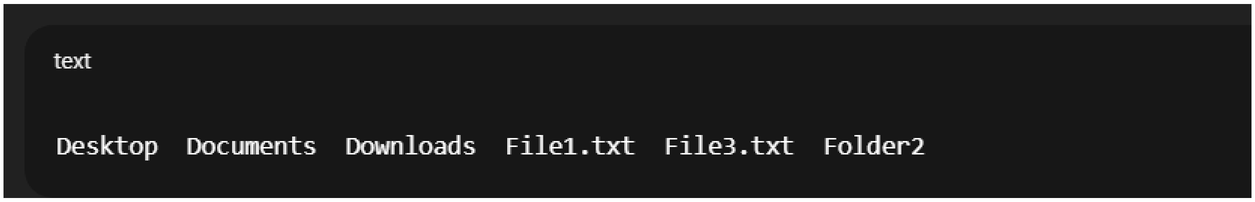
Code:



bash

```bash
ls
```

Output:



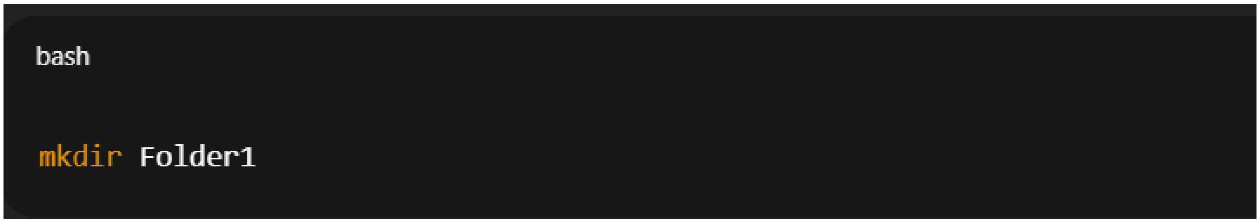
text

```
Desktop  Documents  Downloads  File1.txt  File3.txt  Folder2
```

h) Make a new directory ‘Folder1’ in your current working directory.

Ans:

Code:

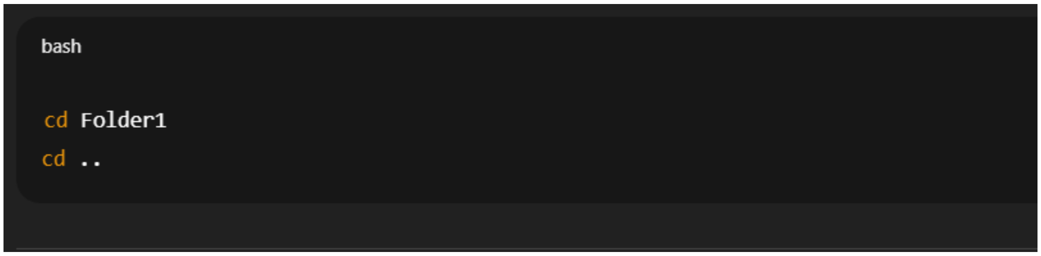


bash

```bash
mkdir Folder1
```

i) Enter the newly created directory Folder1 and go up one level.

Ans:



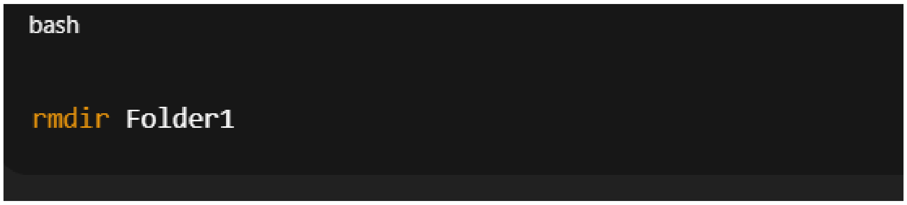
bash

```bash
cd Folder1
cd ..
```

j) Delete the newly created directory Folder1.

Ans:

Code:



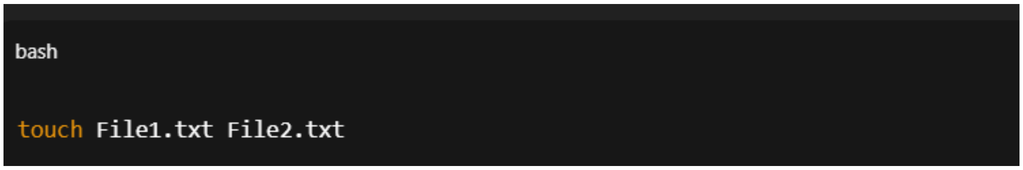
```bash
rmdir Folder1
```

k) Concatenate 2 files and display the resulting content:

(i) Create 2 new files, name them ‘File1.txt’ and ‘File2.txt’ respectively.

Ans:

Code:

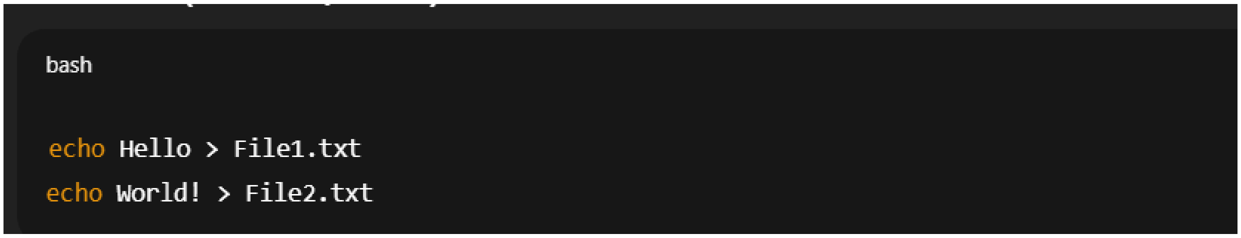


```bash
touch File1.txt File2.txt
```

(ii) Write ‘Hello’ in File1.txt and ‘World!’ in File2.txt.

Ans:

Code:

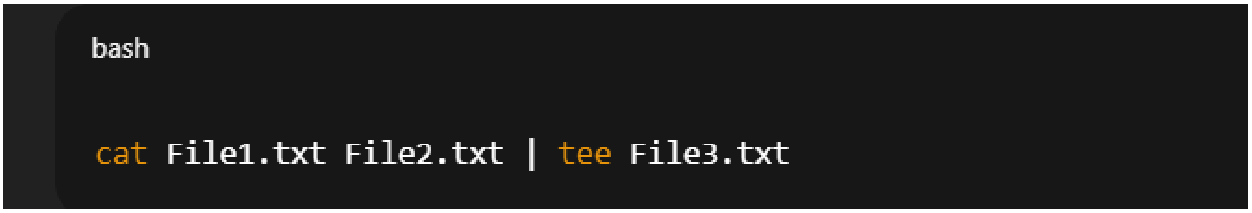


```bash
echo Hello > File1.txt
echo World! > File2.txt
```

(iii) Concatenate these 2 files, display the content, and save the result in ‘File3.txt’.

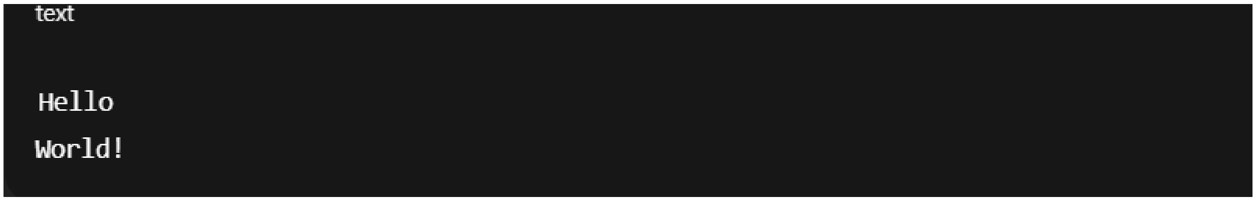
Ans:

Code:



```bash
cat File1.txt File2.txt | tee File3.txt
```

Output:

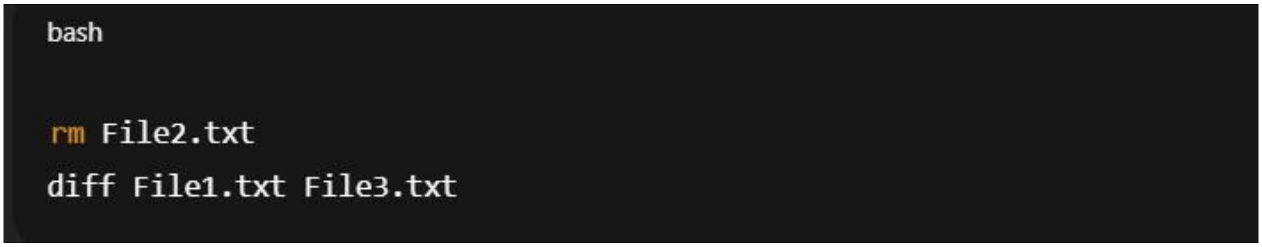


```text
Hello
World!
```

iv) Delete File2.txt. Now compare File1.txt and File3.txt.

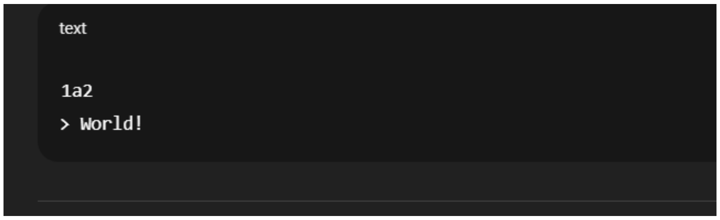
Ans:

Code:



```bash
rm File2.txt
diff File1.txt File3.txt
```

Output:

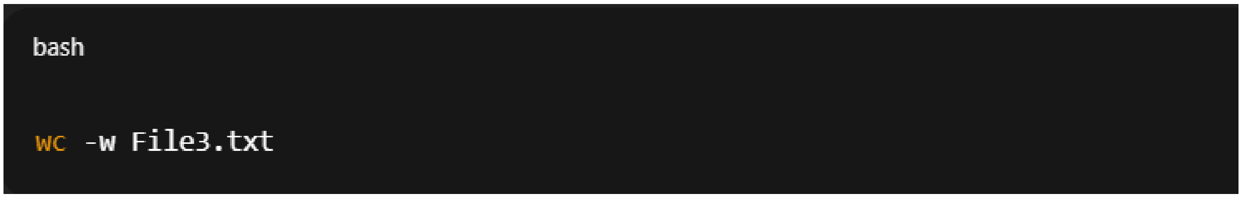


```text
1a2
> World!
```

(l) Check and display the word count in File3.txt.

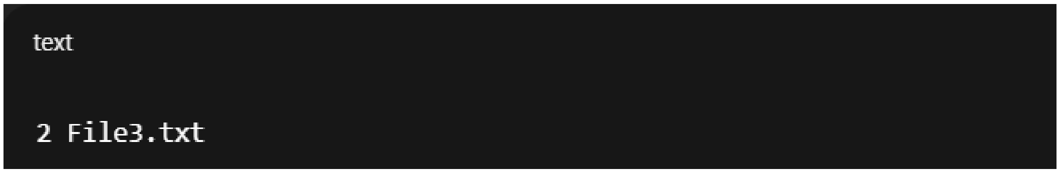
Ans:

Code:



```
bash

wc -w File3.txt
```

Output:

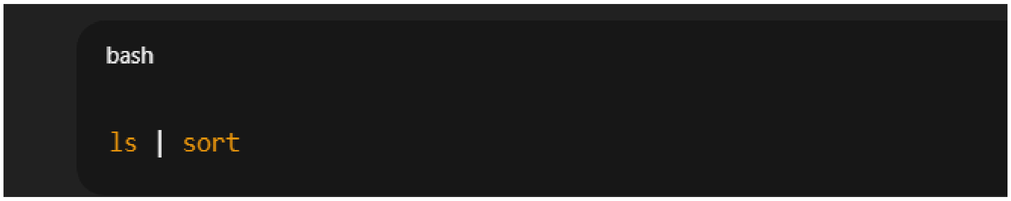


```
text

2 File3.txt
```

m) Lexicographically sort the files and folders in the current working directory using the pipe operator along with other appropriate commands.

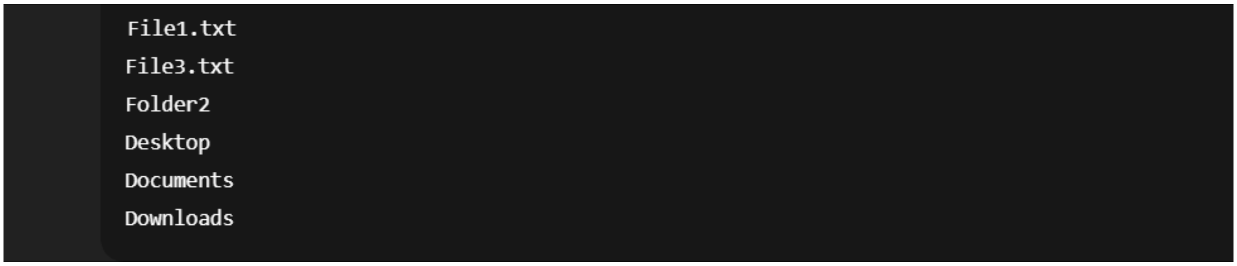
Ans:

Code:



```
bash

ls | sort
```

Output:

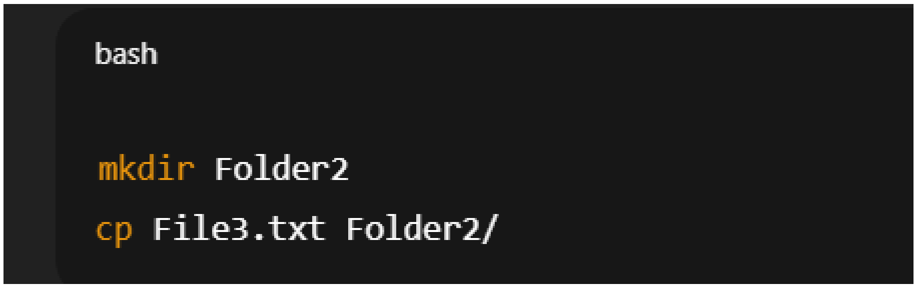


```
File1.txt
File3.txt
Folder2
Desktop
Documents
Downloads
```

n) Make a new directory ‘Folder2’ in your current working directory. Copy and paste File3.txt into the aforementioned directory.

Ans:

Code:

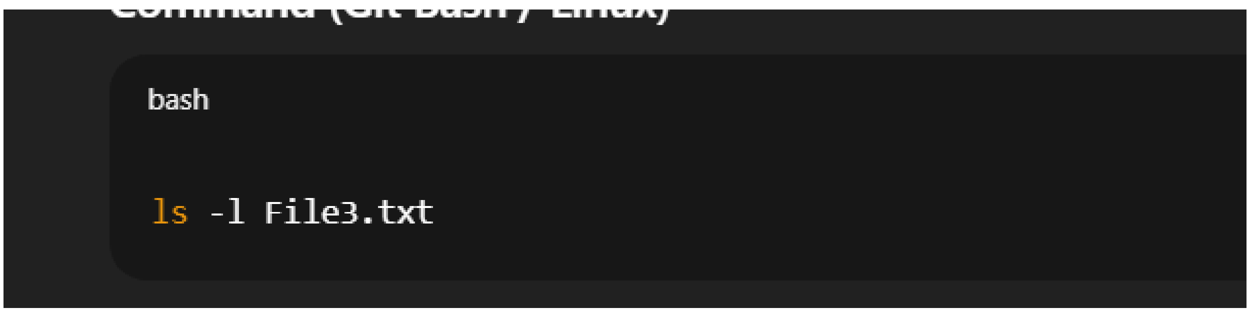


```bash
mkdir Folder2
cp File3.txt Folder2/
```

o) Check the read / write permission of File3.txt.

Ans:

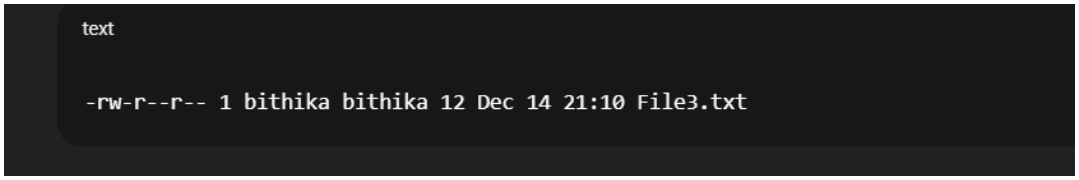
Code:



Command (Git Bash / Linux)

```bash
ls -l File3.txt
```

Output:



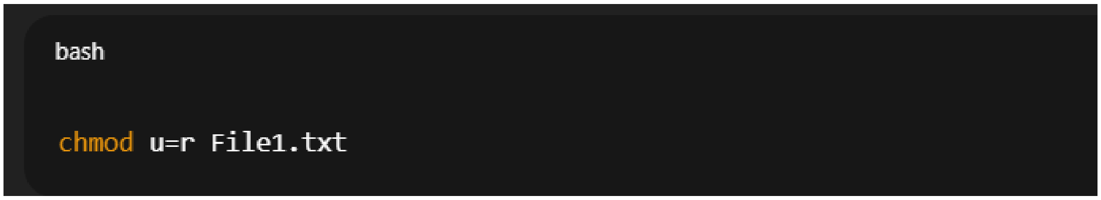
```text
-rw-r--r-- 1 bithika bithika 12 Dec 14 21:10 File3.txt
```

p) Set the following permissions for File1.txt:

(i) File owner permission as read-only.

Ans:

Code:



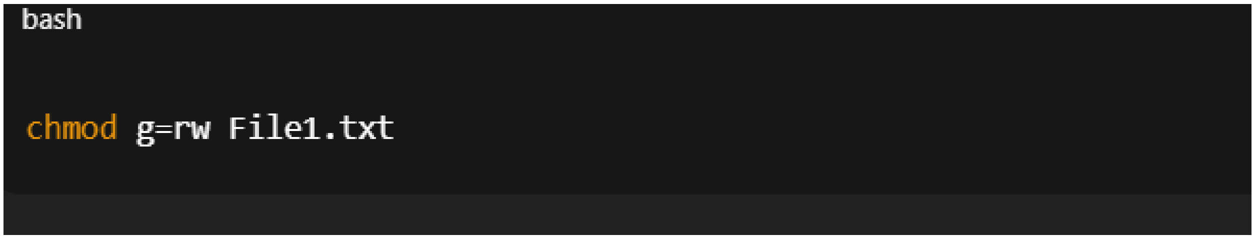
bash

```
chmod u=r File1.txt
```

(ii) Group file owner permission as read and write.

Ans:

Code:



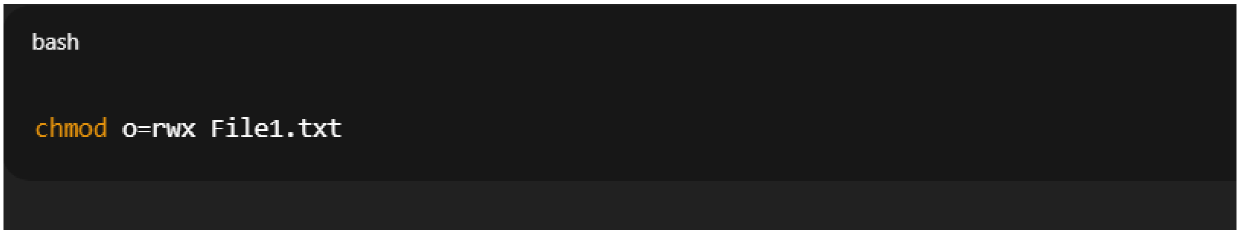
bash

```
chmod g=rw File1.txt
```

iii) Other user permission as read, write and execute.

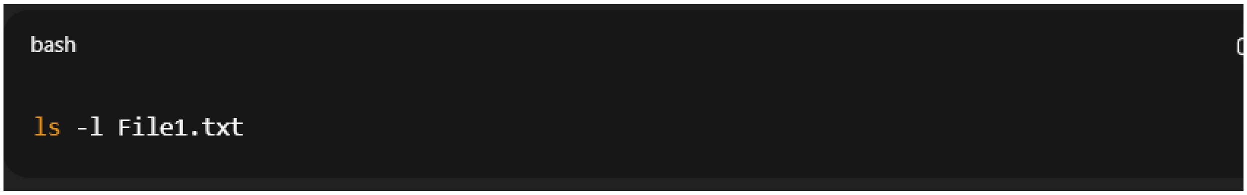
Ans:

Code:



bash

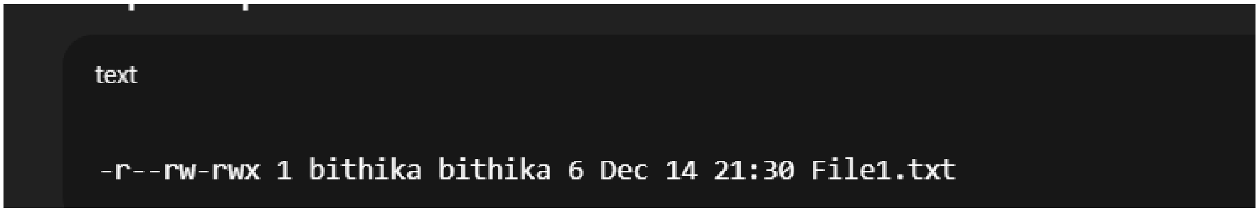
```
chmod o=rwx File1.txt
```



bash

```bash
ls -l File1.txt
```

Output:



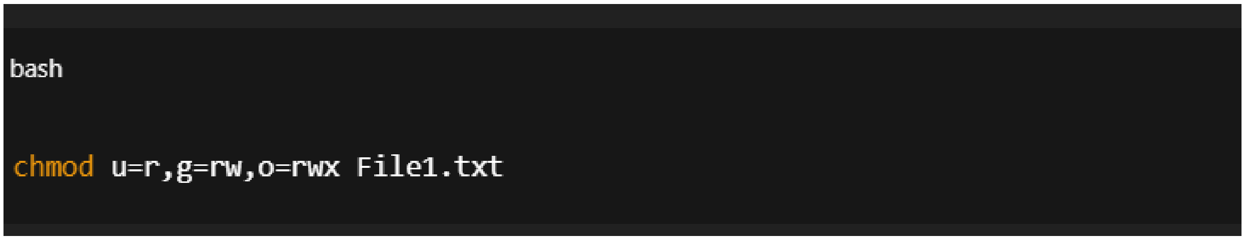
text

```text
-r--rw-rwx 1 bithika bithika 6 Dec 14 21:30 File1.txt
```

iv) All other system user permissions as read, write and execute.

Ans:

Code:



bash

```bash
chmod u=r,g=rw,o=rwx File1.txt
```

Assignment-2

(a)Use the ‘ps’ command to perform the following operations:

a. Display the process status of the current shell.

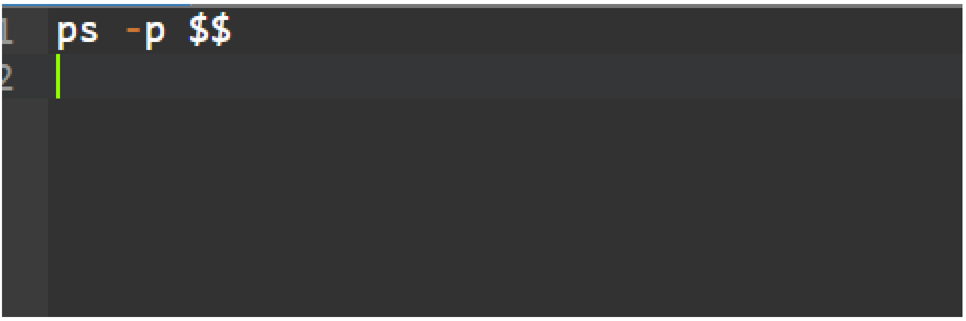
b. Display the process status of all the running processes.

c. Display the process status of all the running processes in a full format.

d. Display the process status of all the running processes except session leader

Ans:

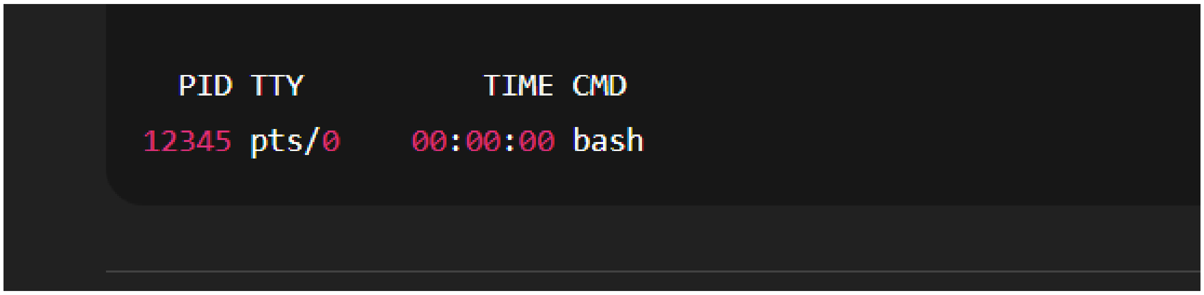
(a)Code:



```
ps -p $$
```

;

Output:



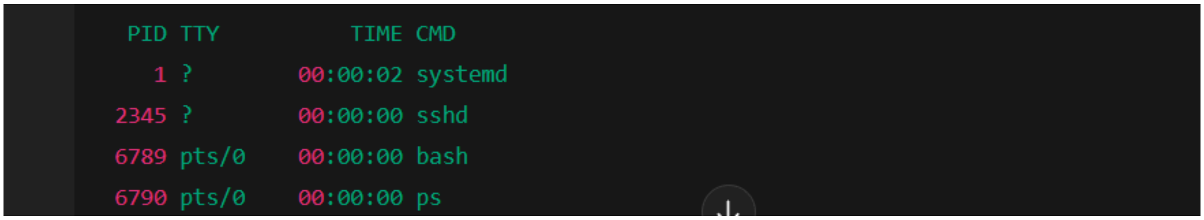
```
  PID TTY          TIME CMD
12345 pts/0    00:00:00 bash
```

(b)

Code:

```
ps -e
```

Output:



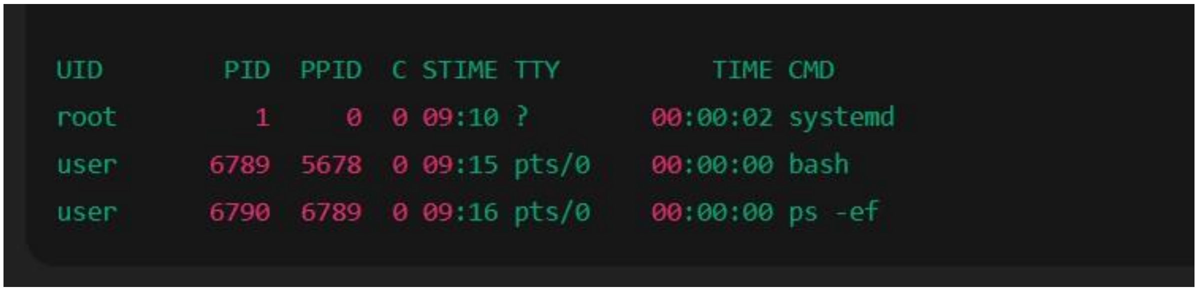
```
  PID TTY          TIME CMD
    1 ?        00:00:02 systemd
 2345 ?        00:00:00 sshd
 6789 pts/0    00:00:00 bash
 6790 pts/0    00:00:00 ps
```

(c)

Code:

```
ps -ef
```

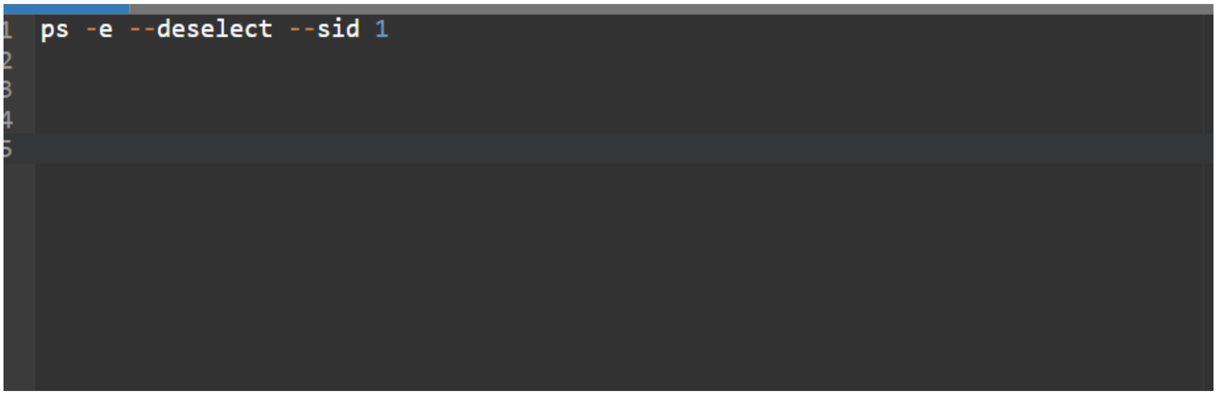
Output:



```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 09:10 ?        00:00:02 systemd
user        6789  5678  0 09:15 pts/0    00:00:00 bash
user        6790  6789  0 09:16 pts/0    00:00:00 ps -ef
```

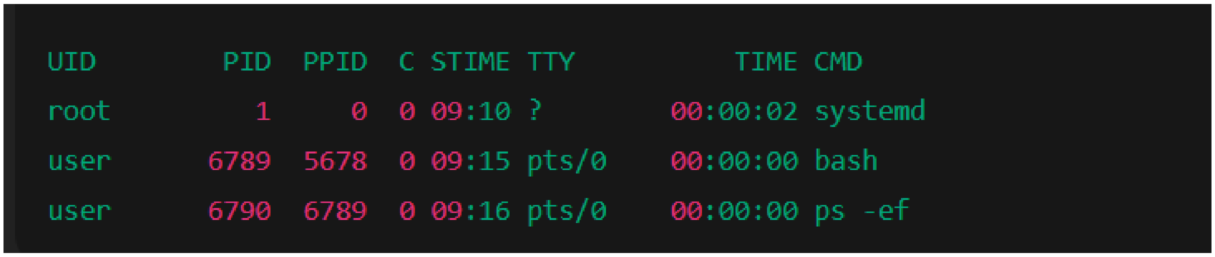
(d)

Code:



```
ps -e --deselect --sid 1
```

Output:



```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 09:10 ?        00:00:02 systemd
user      6789  5678  0 09:15 pts/0    00:00:00 bash
user      6790  6789  0 09:16 pts/0    00:00:00 ps -ef
```

(b) Open the Firefox browser. Now using the ‘htop’ program, display the following:

a. The PID(s) of the Firefox process / processes.

b. The owner of the process / processes.

c. Virtual memory being consumed by the process / processes.

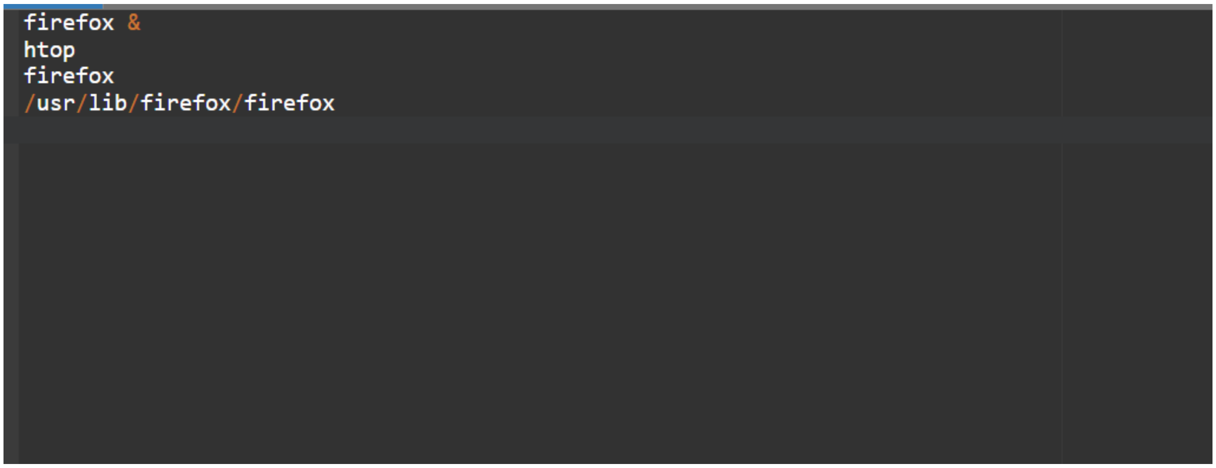
d. The percentage of the processor time used by the process / processes.

e. The percentage of physical RAM used by the process / processes.

f. The name of the command that started the process / processes.

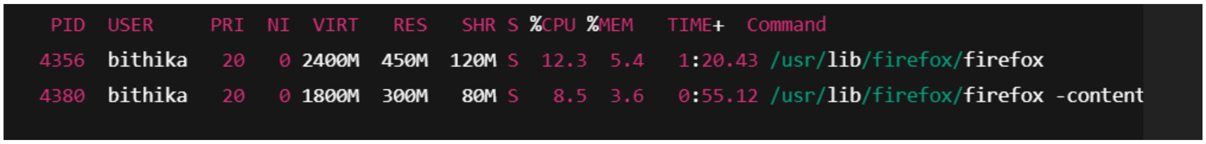
Ans:

Code:



```
firefox &
htop
firefox
/usr/lib/firefox/firefox
```

Output:

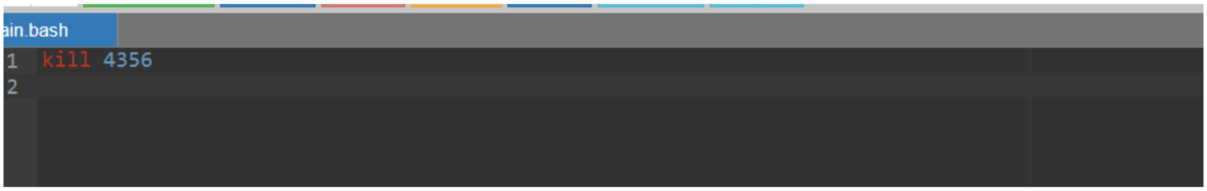


```
  PID  USER      PRI  NI  VIRT    RES    SHR S %CPU %MEM    TIME+  Command
 4356  bithika    20   0 2400M  450M  120M S  12.3  5.4   1:20.43 /usr/lib/firefox/firefox
 4380  bithika    20   0 1800M  300M   80M S   8.5  3.6   0:55.12 /usr/lib/firefox/firefox -content
```

(c) Kill a specific process by using its PID

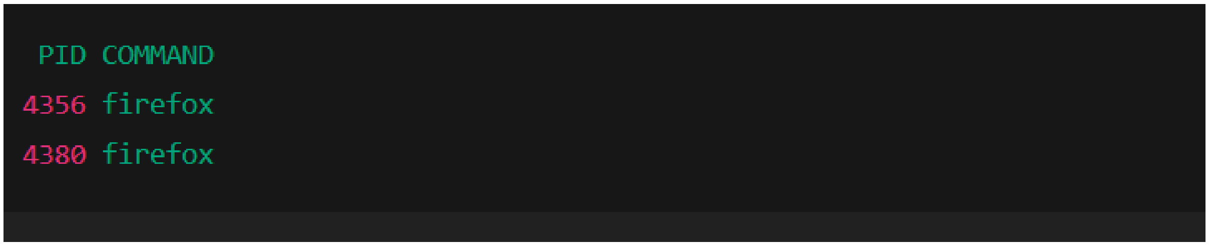
Ans:

Code:

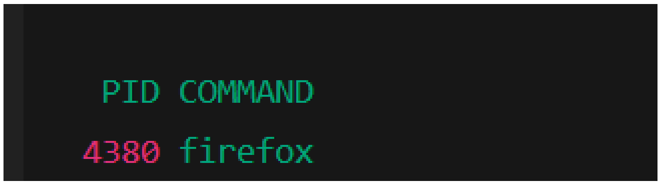


```
kill 4356
```

Output:



```
 PID COMMAND
4356 firefox
4380 firefox
```

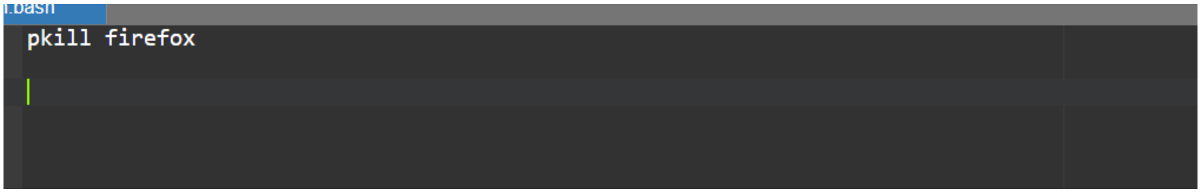


```
 PID COMMAND
4380 firefox
```

(d) Kill all the processes in the system.

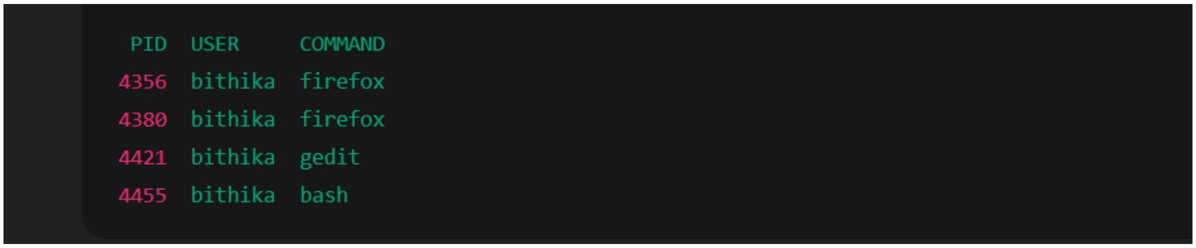
Ans:

Code:

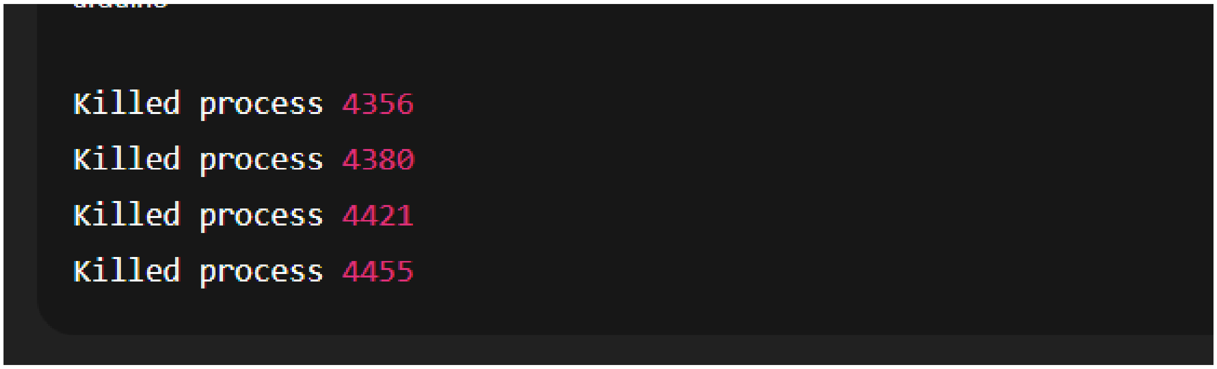


```
pkill firefox
```

Output:



```
 PID  USER     COMMAND
4356  bithika  firefox
4380  bithika  firefox
4421  bithika  gedit
4455  bithika  bash
```



```
Killed process 4356
Killed process 4380
Killed process 4421
Killed process 4455
```

(e) Open the Firefox browser. Observe and state the following using appropriate commands:

a. Parent process

b. Child process / processes

c. PID of the associated processes

d. PPID of the associated processes

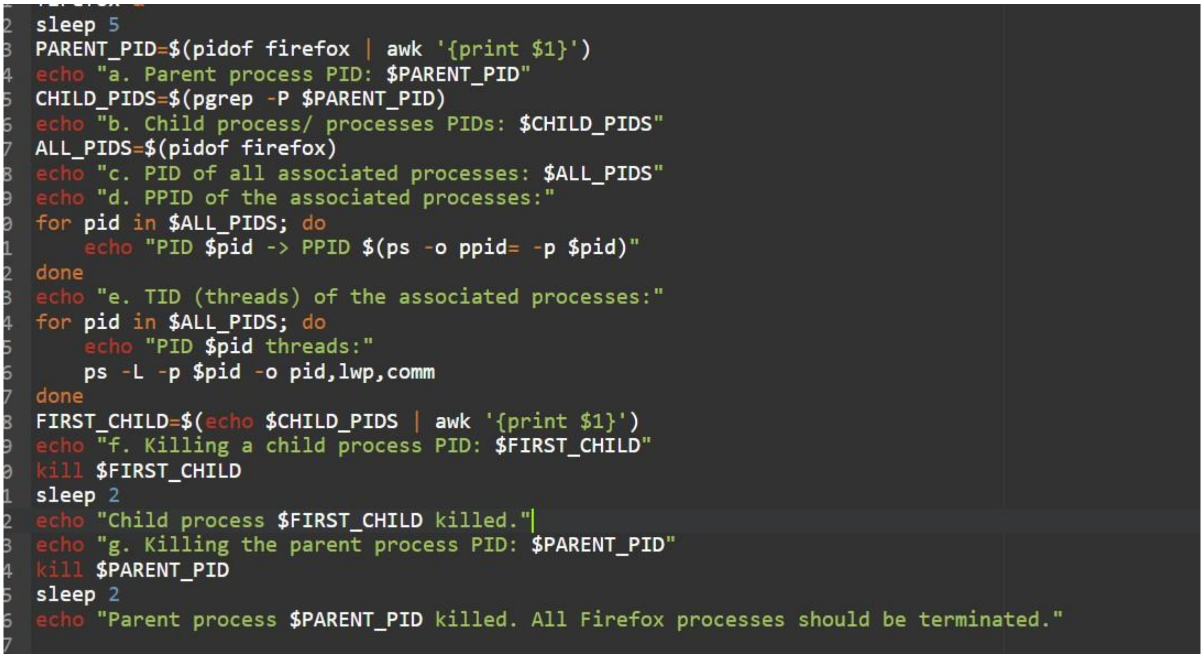
e. TID of the associated processes

f. Kill a child process

g. Kill the parent process

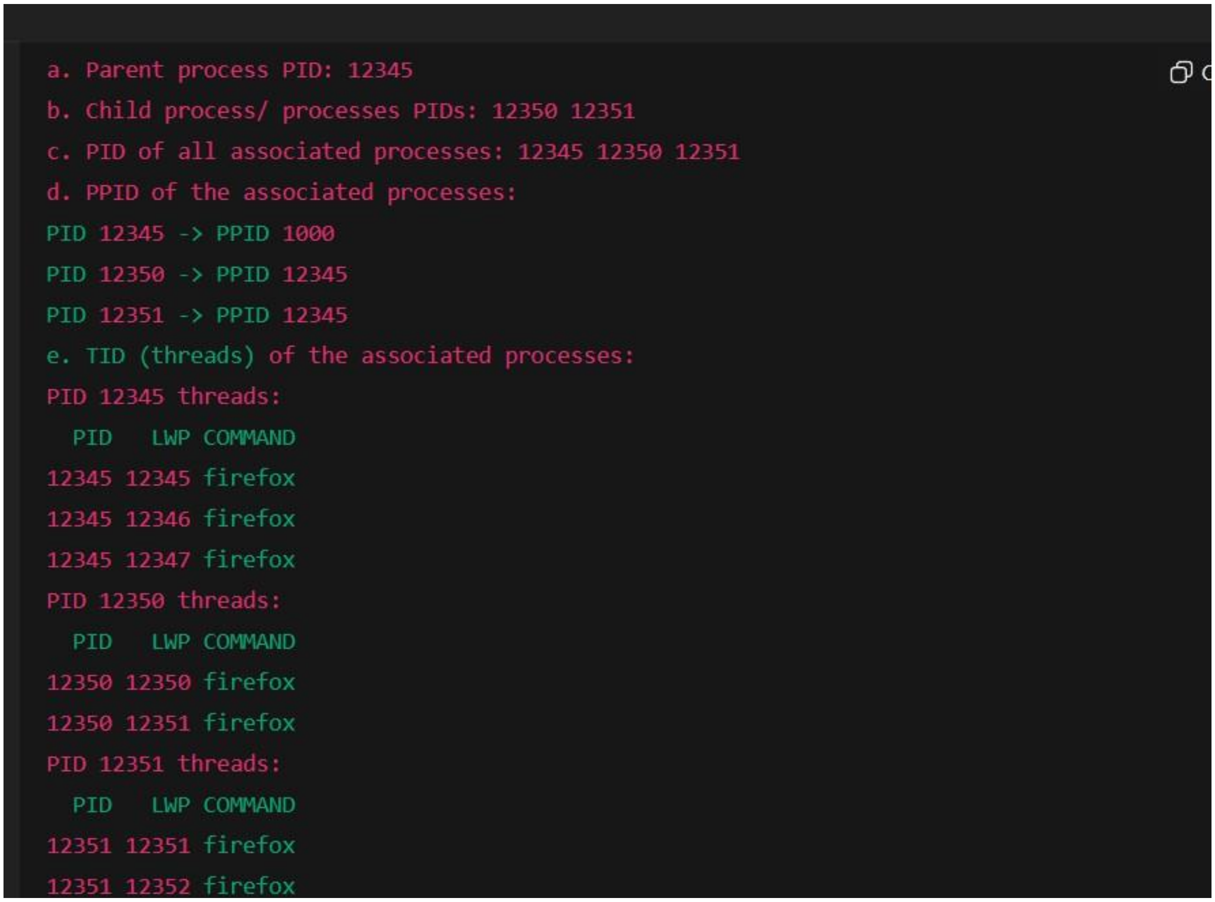
Ans:

Code:

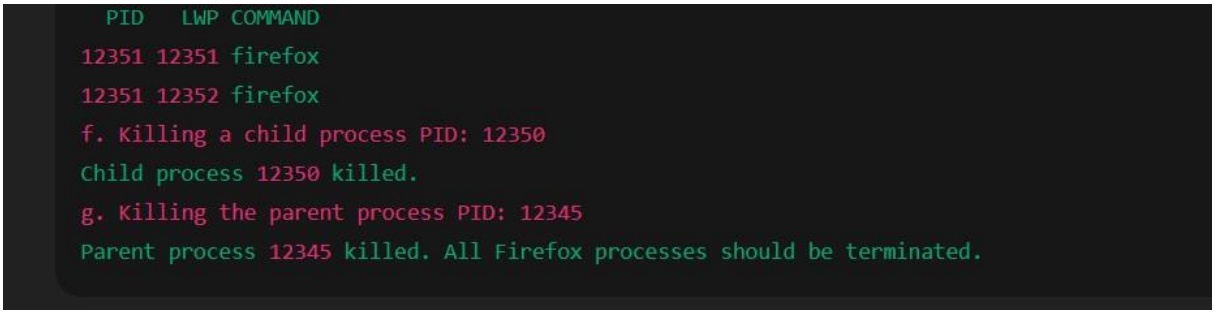


```
firefox &
sleep 5
PARENT_PID=$(pidof firefox | awk '{print $1}')
echo "a. Parent process PID: $PARENT_PID"
CHILD_PIDS=$(pgrep -P $PARENT_PID)
echo "b. Child process/ processes PIDs: $CHILD_PIDS"
ALL_PIDS=$(pidof firefox)
echo "c. PID of all associated processes: $ALL_PIDS"
echo "d. PPID of the associated processes:"
for pid in $ALL_PIDS; do
    echo "PID $pid -> PPID $(ps -o ppid= -p $pid)"
done
echo "e. TID (threads) of the associated processes:"
for pid in $ALL_PIDS; do
    echo "PID $pid threads:"
    ps -L -p $pid -o pid,lwp,comm
done
FIRST_CHILD=$(echo $CHILD_PIDS | awk '{print $1}')
echo "f. Killing a child process PID: $FIRST_CHILD"
kill $FIRST_CHILD
sleep 2
echo "Child process $FIRST_CHILD killed."
echo "g. Killing the parent process PID: $PARENT_PID"
kill $PARENT_PID
sleep 2
echo "Parent process $PARENT_PID killed. All Firefox processes should be terminated."
```

Output:



```
a. Parent process PID: 12345
b. Child process/ processes PIDs: 12350 12351
c. PID of all associated processes: 12345 12350 12351
d. PPID of the associated processes:
PID 12345 -> PPID 1000
PID 12350 -> PPID 12345
PID 12351 -> PPID 12345
e. TID (threads) of the associated processes:
PID 12345 threads:
  PID   LWP COMMAND
12345 12345 firefox
12345 12346 firefox
12345 12347 firefox
PID 12350 threads:
  PID   LWP COMMAND
12350 12350 firefox
12350 12351 firefox
PID 12351 threads:
  PID   LWP COMMAND
12351 12351 firefox
12351 12352 firefox
```

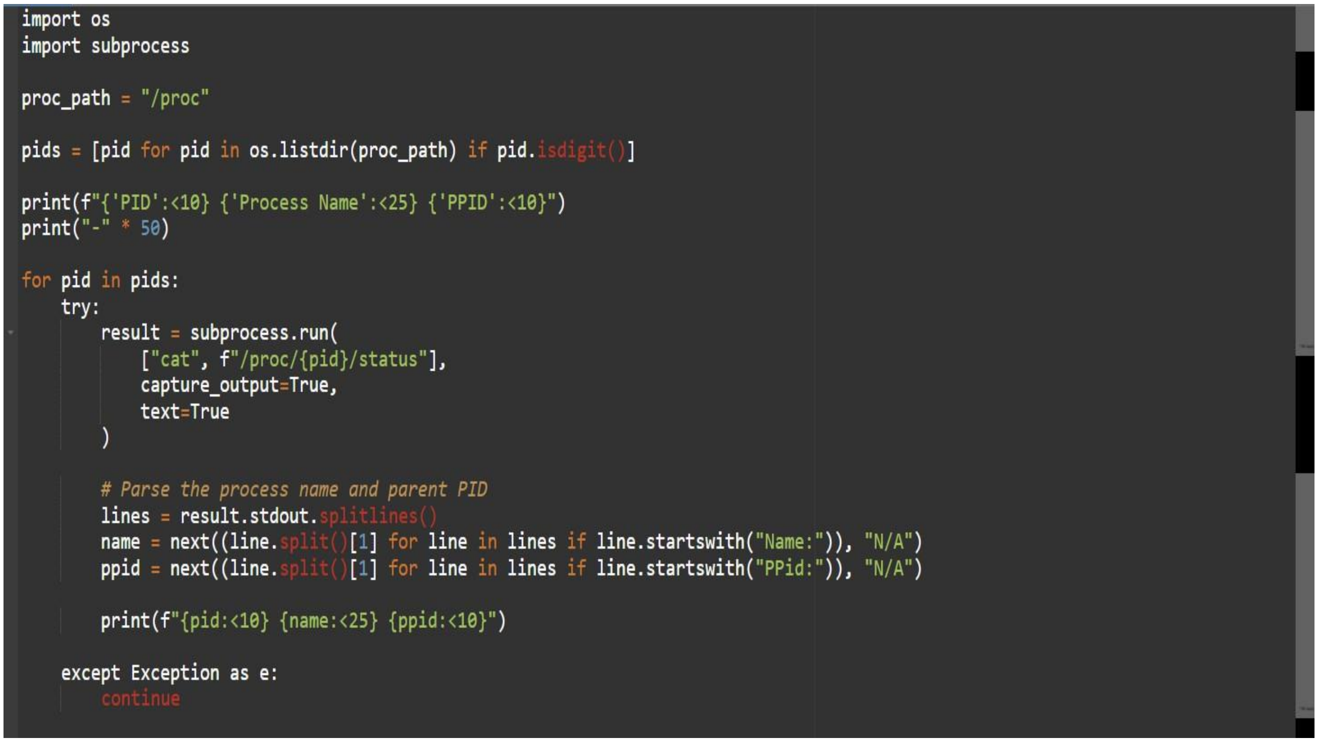


```
  PID   LWP COMMAND
12351 12351 firefox
12351 12352 firefox
f. Killing a child process PID: 12350
Child process 12350 killed.
g. Killing the parent process PID: 12345
Parent process 12345 killed. All Firefox processes should be terminated.
```

(f) Using CAT command, write a program in Python to get a list of all the running processes.

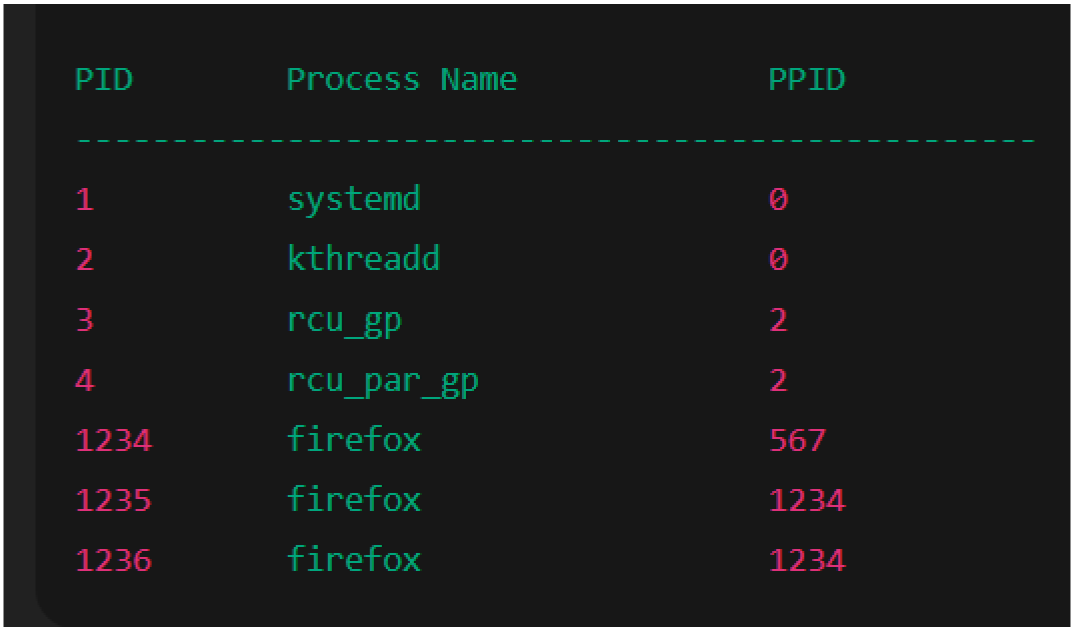
Ans:

Code:



```
import os
import subprocess

proc_path = "/proc"

pids = [pid for pid in os.listdir(proc_path) if pid.isdigit()]

print(f"{'PID':<10} {'Process Name':<25} {'PPID':<10}")
print("-" * 50)

for pid in pids:
    try:
        result = subprocess.run(
            ["cat", f"/proc/{pid}/status"],
            capture_output=True,
            text=True
        )

        # Parse the process name and parent PID
        lines = result.stdout.splitlines()
        name = next((line.split()[1] for line in lines if line.startswith("Name:")), "N/A")
        ppid = next((line.split()[1] for line in lines if line.startswith("PPid:")), "N/A")

        print(f"{pid:<10} {name:<25} {ppid:<10}")

    except Exception as e:
        continue
```

Output:

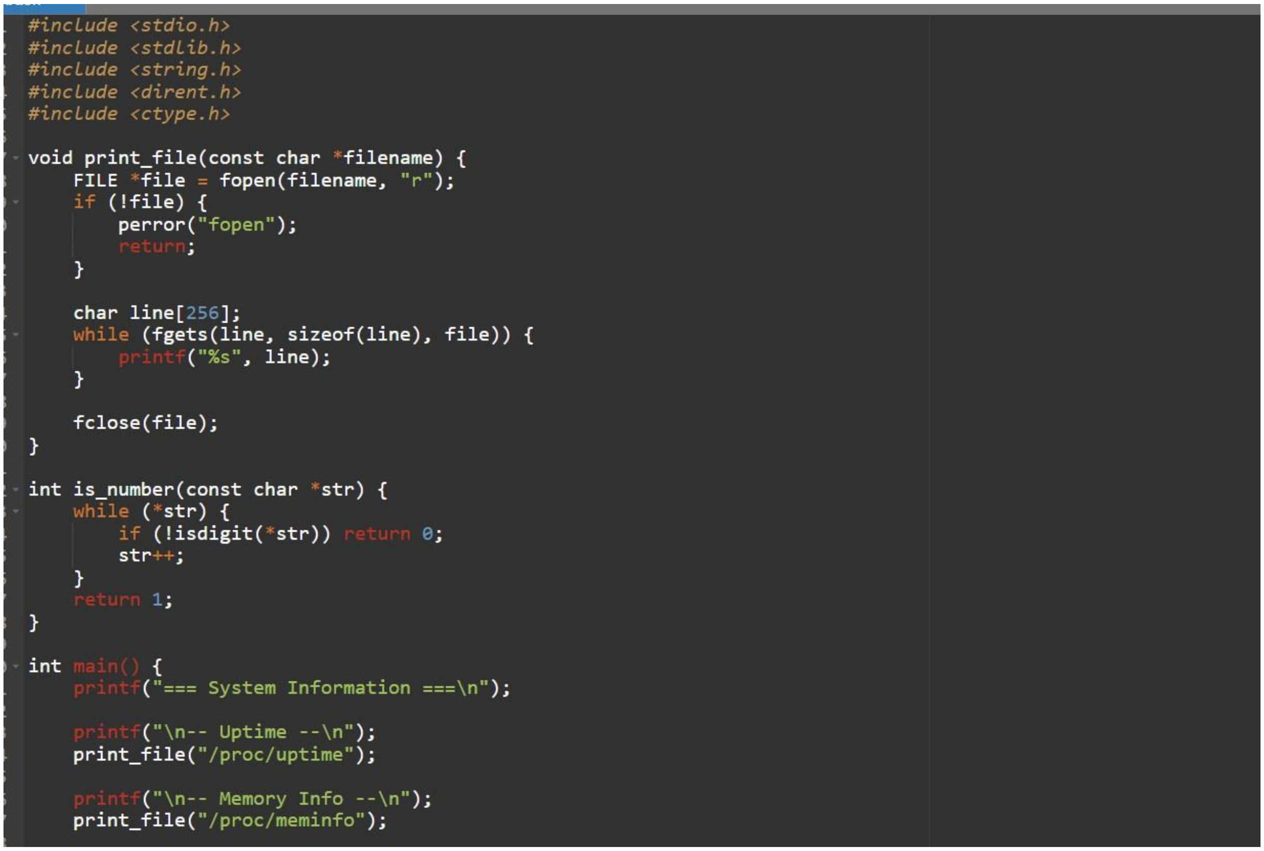


```
PID        Process Name             PPID
--------------------------------------------------
1          systemd                  0
2          kthreadd                 0
3          rcu_gp                   2
4          rcu_par_gp               2
1234       firefox                  567
1235       firefox                  1234
1236       firefox                  1234
```

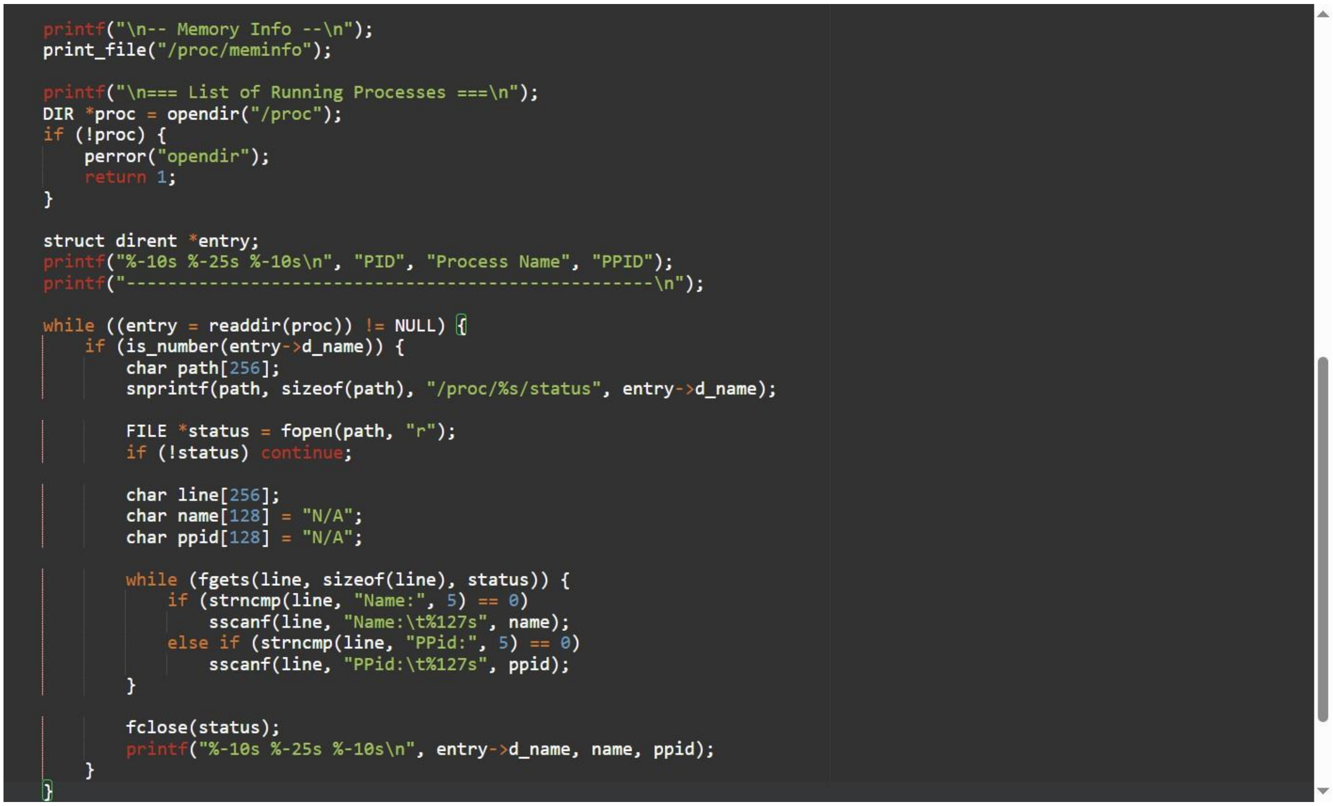
(g) Using CAT command, write a program in C to get the system and process information.

Ans:

Code:

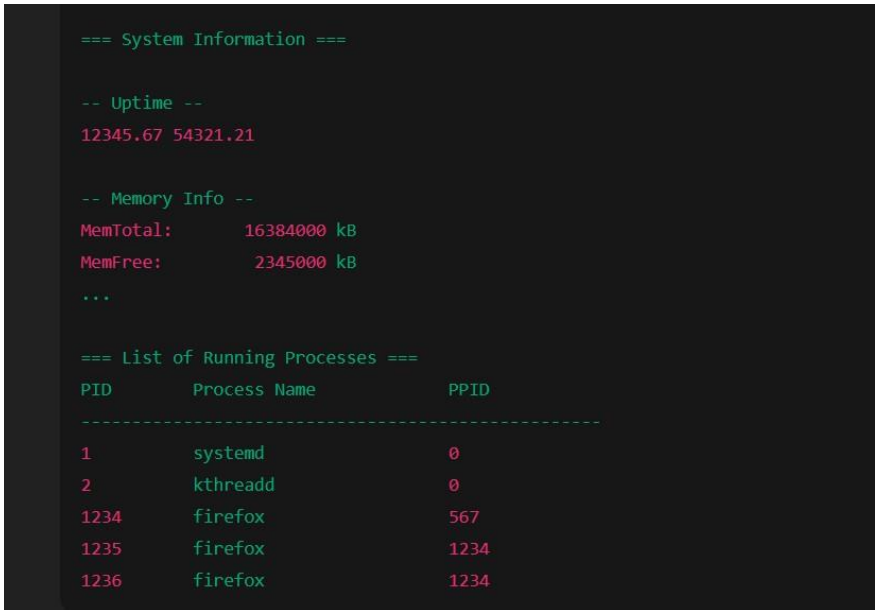


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <ctype.h>

void print_file(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("fopen");
        return;
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        printf("%s", line);
    }

    fclose(file);
}

int is_number(const char *str) {
    while (*str) {
        if (!isdigit(*str)) return 0;
        str++;
    }
    return 1;
}

int main() {
    printf("=== System Information ===\n");

    printf("\n-- Uptime --\n");
    print_file("/proc/uptime");

    printf("\n-- Memory Info --\n");
    print_file("/proc/meminfo");
```



```c
printf("\n-- Memory Info --\n");
print_file("/proc/meminfo");

printf("\n=== List of Running Processes ===\n");
DIR *proc = opendir("/proc");
if (!proc) {
    perror("opendir");
    return 1;
}

struct dirent *entry;
printf("%-10s %-25s %-10s\n", "PID", "Process Name", "PPID");
printf("---------------------------------------------------\n");

while ((entry = readdir(proc)) != NULL) {
    if (is_number(entry->d_name)) {
        char path[256];
        snprintf(path, sizeof(path), "/proc/%s/status", entry->d_name);

        FILE *status = fopen(path, "r");
        if (!status) continue;

        char line[256];
        char name[128] = "N/A";
        char ppid[128] = "N/A";

        while (fgets(line, sizeof(line), status)) {
            if (strncmp(line, "Name:", 5) == 0)
                sscanf(line, "Name:\t%127s", name);
            else if (strncmp(line, "PPid:", 5) == 0)
                sscanf(line, "PPid:\t%127s", ppid);
        }

        fclose(status);
        printf("%-10s %-25s %-10s\n", entry->d_name, name, ppid);
    }
}
```

Output:



```
=== System Information ===

-- Uptime --
12345.67 54321.21

-- Memory Info --
MemTotal:       16384000 kB
MemFree:         2345000 kB
...

=== List of Running Processes ===
PID        Process Name              PPID
---------------------------------------------------
1          systemd                   0
2          kthreadd                  0
1234       firefox                   567
1235       firefox                   1234
1236       firefox                   1234
```

Assignment-3

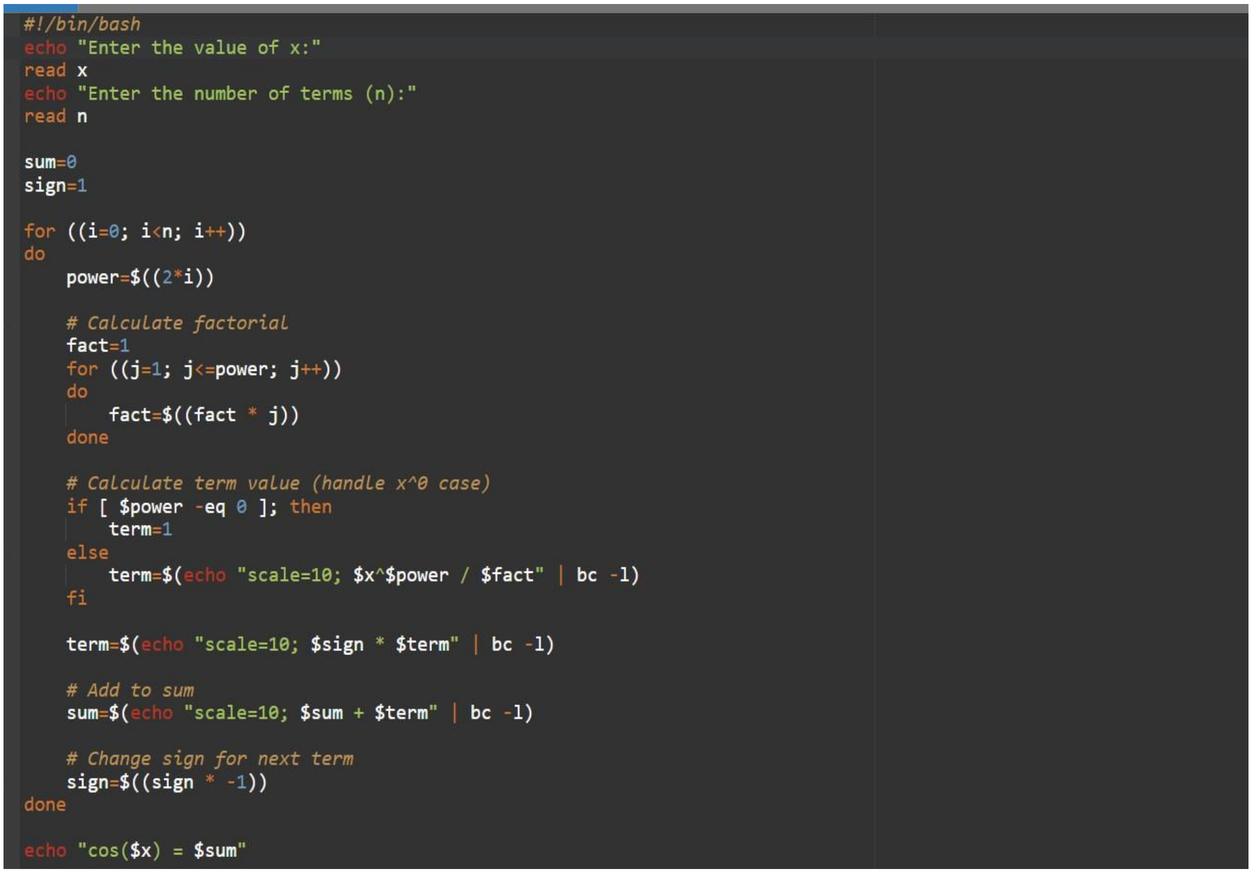
1) Write a shell program to print the sum of the following series:

a. sin(x) = x – x^3 / 3! + x^5 / 5! – x 7 / 7! + ... ± x^n / n!

b. cos(x) = 1 – x^2 / 2! + x^4 / 4! – x^6 / 6! + ... ± x^n / n!

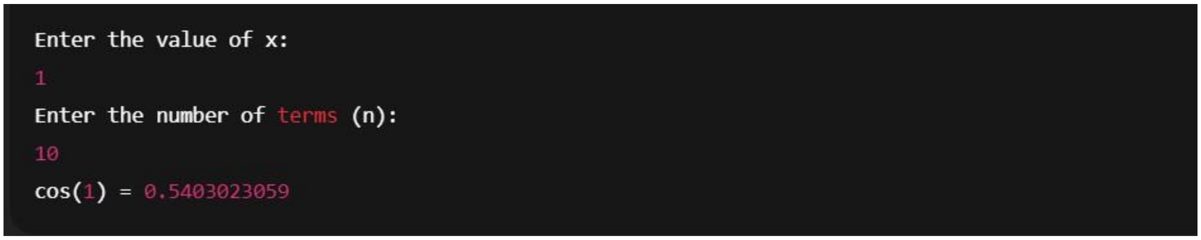
Ans:

Code:



```
#!/bin/bash
echo "Enter the value of x:"
read x
echo "Enter the number of terms (n):"
read n

sum=0
sign=1

for ((i=0; i<n; i++))
do
    power=$((2*i))

    # Calculate factorial
    fact=1
    for ((j=1; j<=power; j++))
    do
        fact=$((fact * j))
    done

    # Calculate term value (handle x^0 case)
    if [ $power -eq 0 ]; then
        term=1
    else
        term=$(echo "scale=10; $x^$power / $fact" | bc -l)
    fi

    term=$(echo "scale=10; $sign * $term" | bc -l)

    # Add to sum
    sum=$(echo "scale=10; $sum + $term" | bc -l)

    # Change sign for next term
    sign=$((sign * -1))
done

echo "cos($x) = $sum"
```

Output:

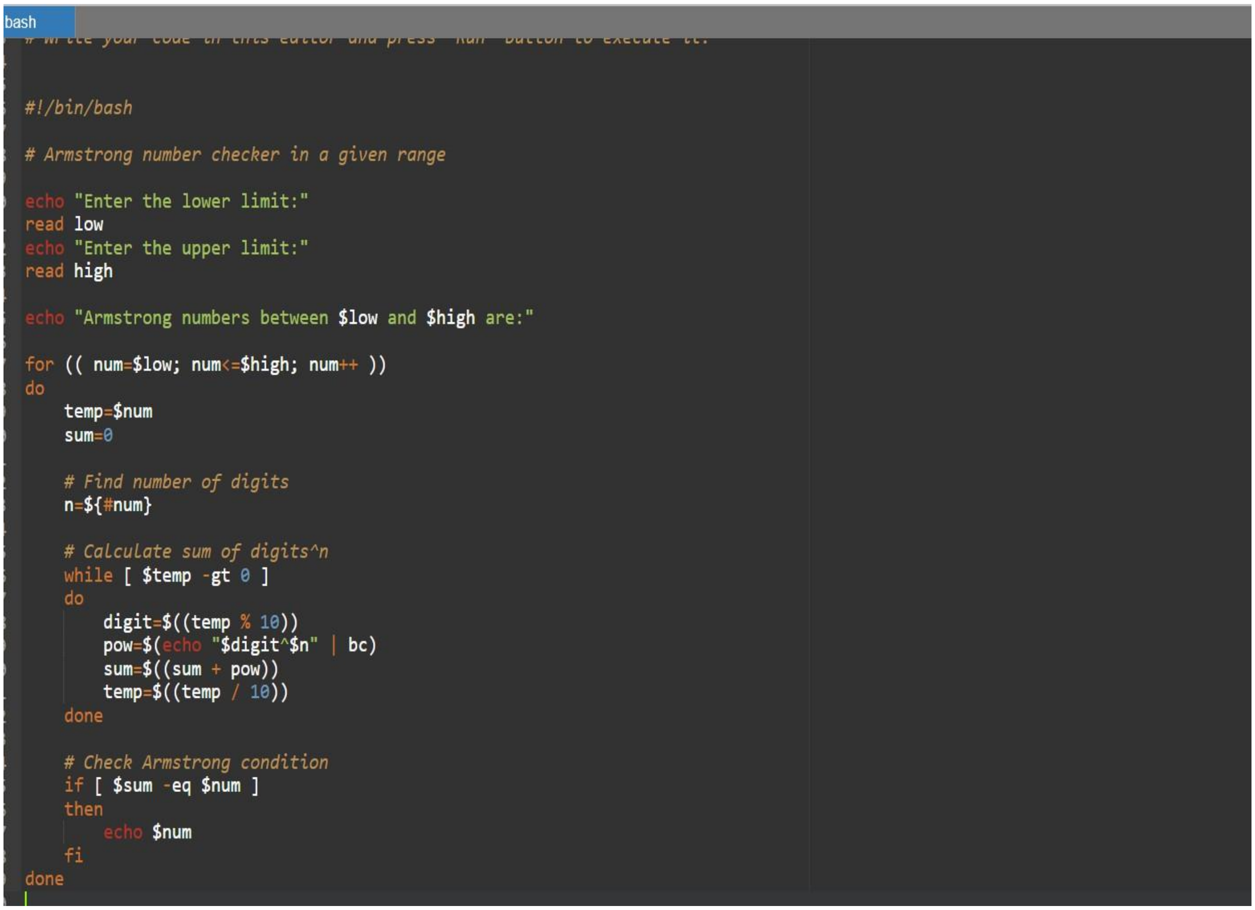


```
Enter the value of x:
1
Enter the number of terms (n):
10
cos(1) = 0.5403023059
```

2) Write a shell program to display Armstrong numbers within a user-defined range.

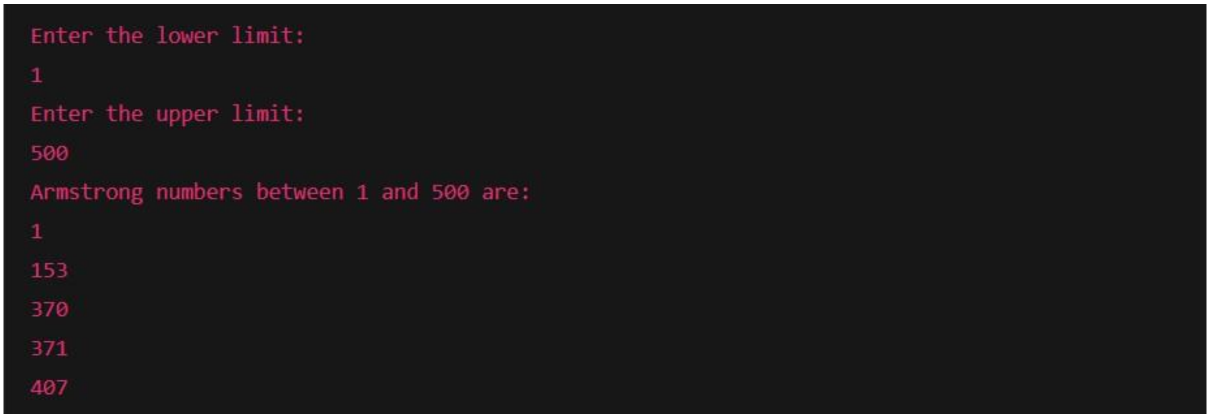
Ans:

Code:



```
# Write your code in this editor and press "Run" button to execute it.


#!/bin/bash

# Armstrong number checker in a given range

echo "Enter the lower limit:"
read low
echo "Enter the upper limit:"
read high

echo "Armstrong numbers between $low and $high are:"

for (( num=$low; num<=$high; num++ ))
do
    temp=$num
    sum=0

    # Find number of digits
    n=${#num}

    # Calculate sum of digits^n
    while [ $temp -gt 0 ]
    do
        digit=$((temp % 10))
        pow=$(echo "$digit^$n" | bc)
        sum=$((sum + pow))
        temp=$((temp / 10))
    done

    # Check Armstrong condition
    if [ $sum -eq $num ]
    then
        echo $num
    fi
done
```

Output:

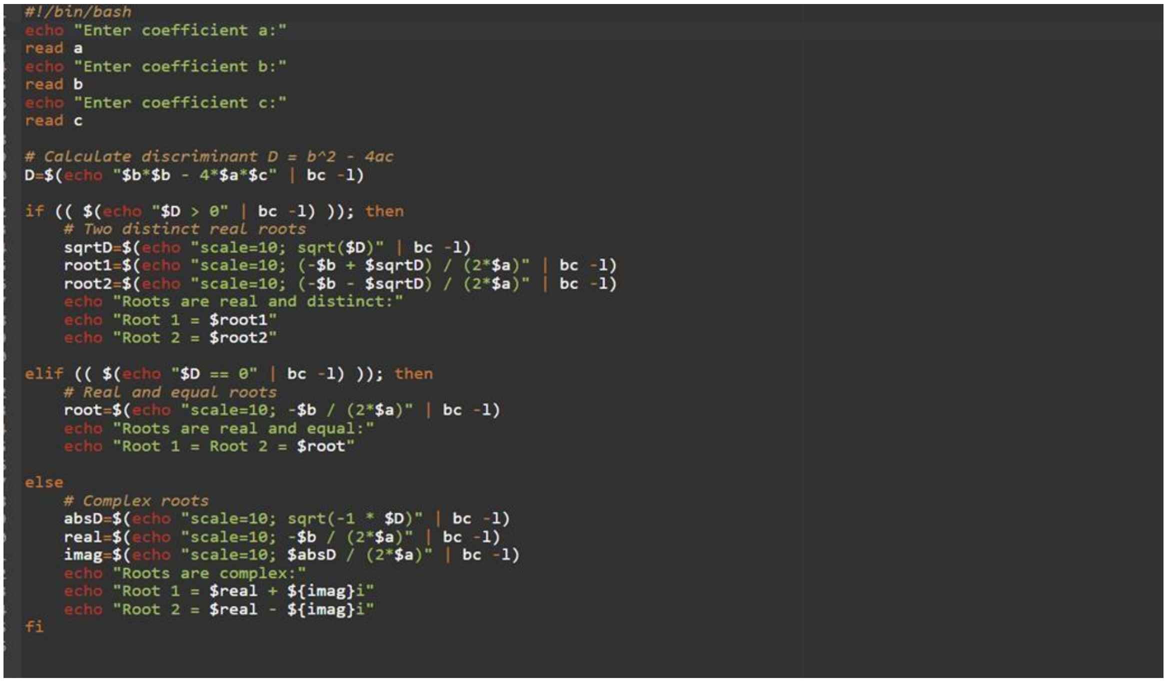


```
Enter the lower limit:
1
Enter the upper limit:
500
Armstrong numbers between 1 and 500 are:
1
153
370
371
407
```

3) Write a shell program that takes coefficients of a quadratic equation as input and calculates its roots. Handle cases of real and complex roots. [If the discriminant D = b 2 – 4ac is negative, D will yield an imaginary number. For example, the quadratic equation x^2 – 2x + 10 = 0, has a = 1, b = −2, c = 10. Therefore, D = √−36 = 6i and root α = (−b + √D) / 2a = 1 + 3i and root β = (−b − √D) / 2a = 1 – 3i.]

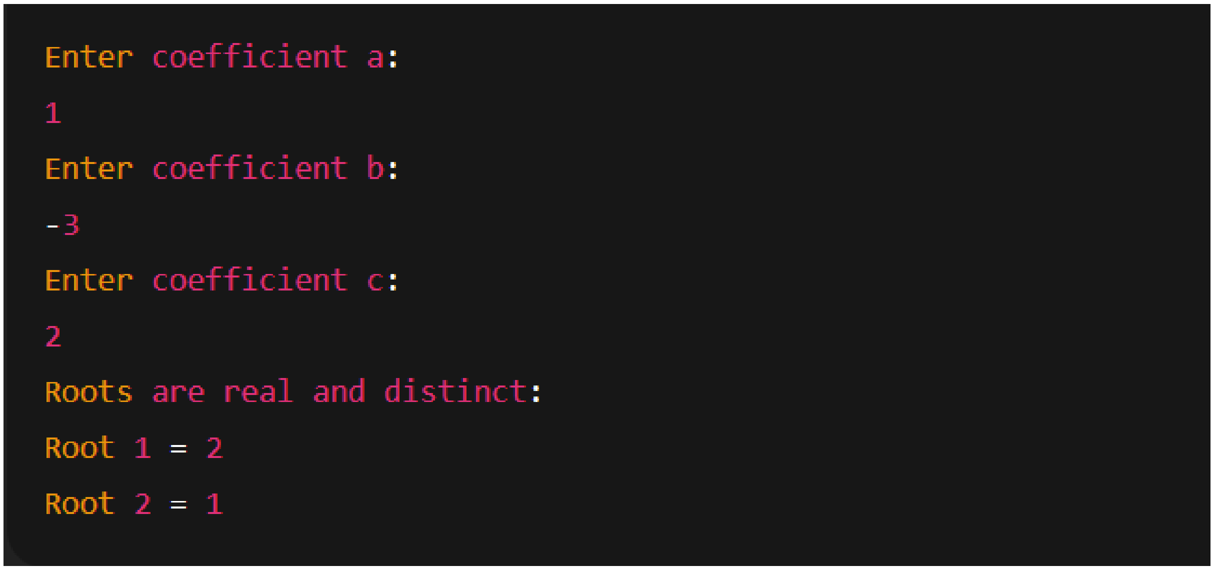
Ans:

Code:



```
#!/bin/bash
echo "Enter coefficient a:"
read a
echo "Enter coefficient b:"
read b
echo "Enter coefficient c:"
read c

# Calculate discriminant D = b^2 - 4ac
D=$(echo "$b*$b - 4*$a*$c" | bc -l)

if (( $(echo "$D > 0" | bc -l) )); then
    # Two distinct real roots
    sqrtD=$(echo "scale=10; sqrt($D)" | bc -l)
    root1=$(echo "scale=10; (-$b + $sqrtD) / (2*$a)" | bc -l)
    root2=$(echo "scale=10; (-$b - $sqrtD) / (2*$a)" | bc -l)
    echo "Roots are real and distinct:"
    echo "Root 1 = $root1"
    echo "Root 2 = $root2"

elif (( $(echo "$D == 0" | bc -l) )); then
    # Real and equal roots
    root=$(echo "scale=10; -$b / (2*$a)" | bc -l)
    echo "Roots are real and equal:"
    echo "Root 1 = Root 2 = $root"

else
    # Complex roots
    absD=$(echo "scale=10; sqrt(-1 * $D)" | bc -l)
    real=$(echo "scale=10; -$b / (2*$a)" | bc -l)
    imag=$(echo "scale=10; $absD / (2*$a)" | bc -l)
    echo "Roots are complex:"
    echo "Root 1 = $real + ${imag}i"
    echo "Root 2 = $real - ${imag}i"
fi
```

Output:



```
Enter coefficient a:
1
Enter coefficient b:
-3
Enter coefficient c:
2
Roots are real and distinct:
Root 1 = 2
Root 2 = 1
```

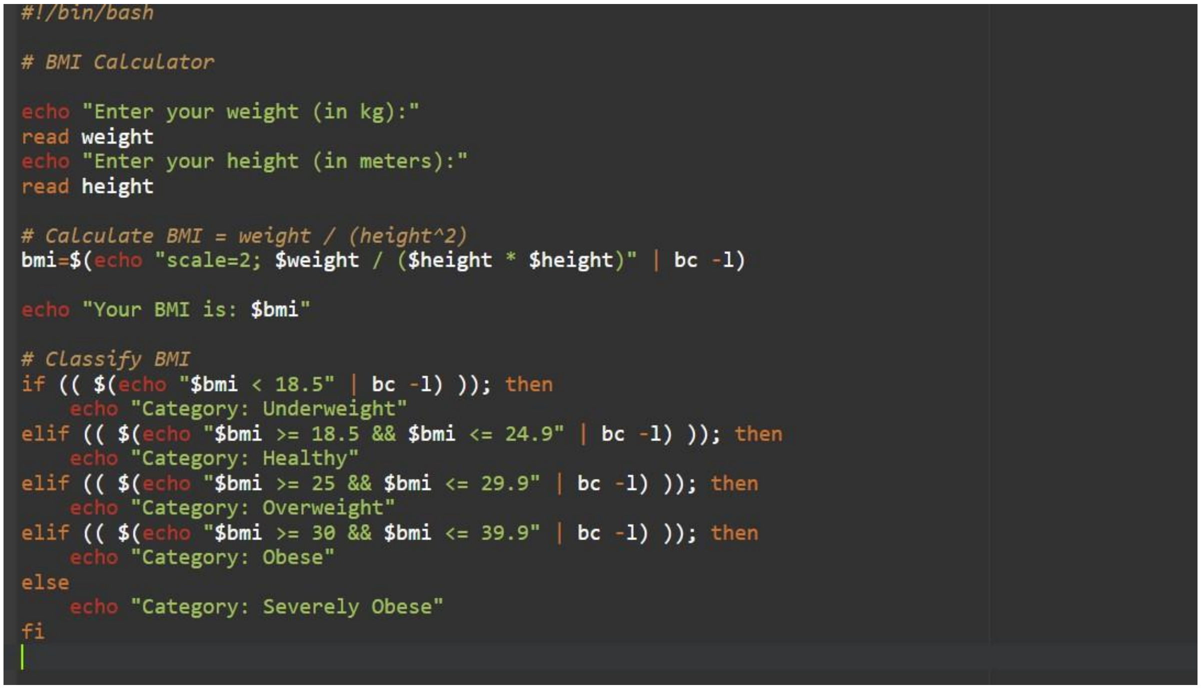
4) Implement a BMI (Body Mass Index) calculator. Accept the weight (in kg) from the user

and their height (in m). The formula to calculate BMI is weight (kg) / [height (m)]^2. Basedon the calculated BMI, classify the person's health status according to the given table:

BMI	Category
< 18.5	Underweight
18.5 – 24.9	Healthy
25 – 29.9	Overweight
30 – 39.9	Obese
>= 40	Severely Obese

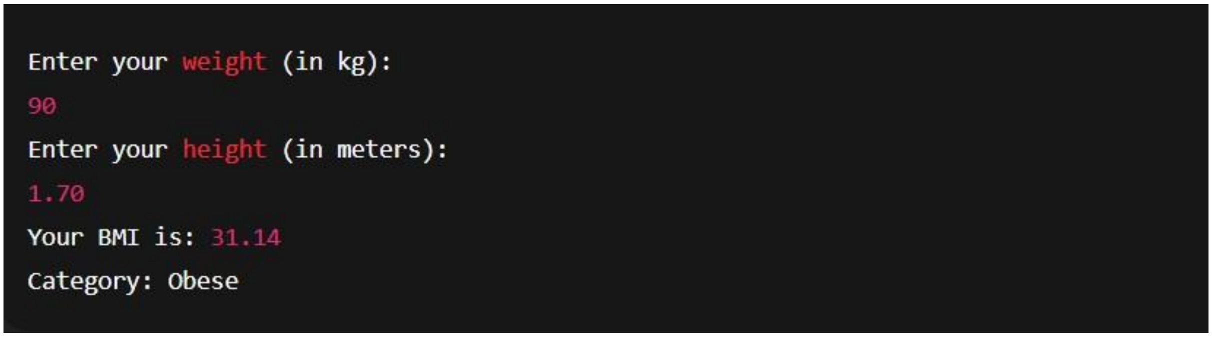
Ans:

Code:



```
#!/bin/bash

# BMI Calculator

echo "Enter your weight (in kg):"
read weight
echo "Enter your height (in meters):"
read height

# Calculate BMI = weight / (height^2)
bmi=$(echo "scale=2; $weight / ($height * $height)" | bc -l)

echo "Your BMI is: $bmi"

# Classify BMI
if (( $(echo "$bmi < 18.5" | bc -l) )); then
    echo "Category: Underweight"
elif (( $(echo "$bmi >= 18.5 && $bmi <= 24.9" | bc -l) )); then
    echo "Category: Healthy"
elif (( $(echo "$bmi >= 25 && $bmi <= 29.9" | bc -l) )); then
    echo "Category: Overweight"
elif (( $(echo "$bmi >= 30 && $bmi <= 39.9" | bc -l) )); then
    echo "Category: Obese"
else
    echo "Category: Severely Obese"
fi
```

Output:

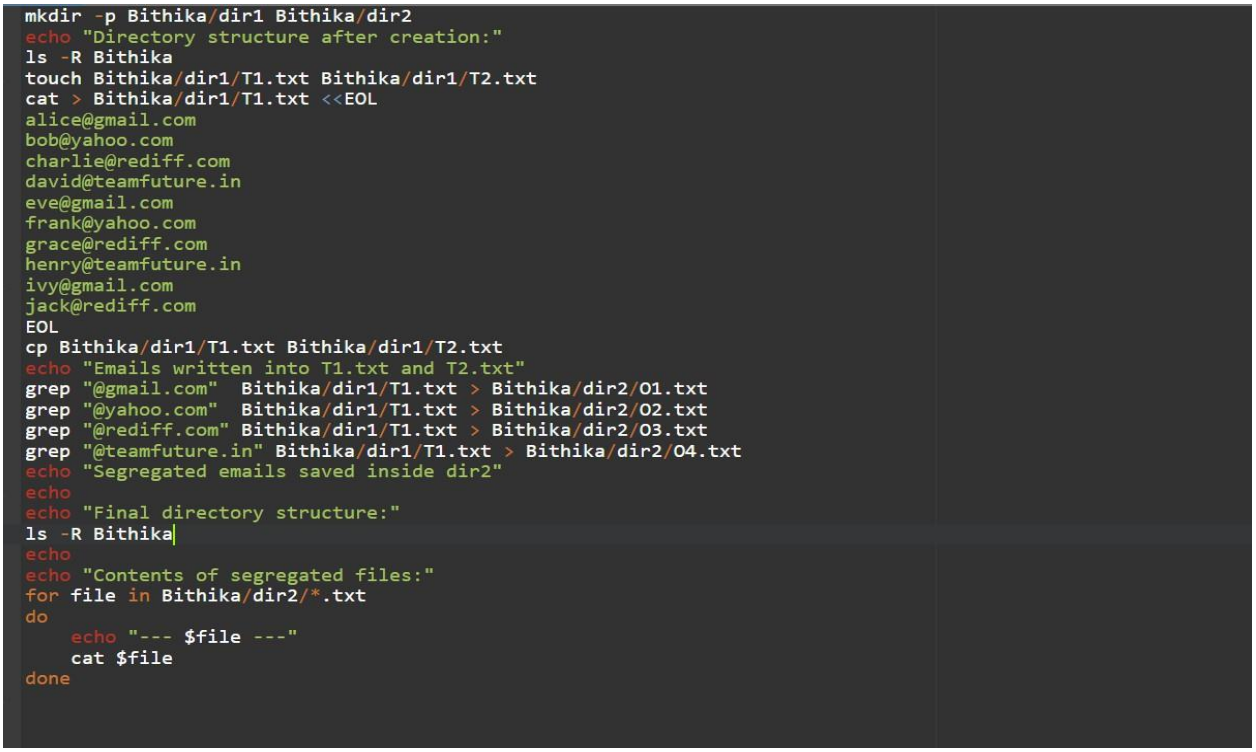


```
Enter your weight (in kg):
90
Enter your height (in meters):
1.70
Your BMI is: 31.14
Category: Obese
```

5) Create a directory with your name. Inside the directory create two sub-directories dir1 anddir2 respectively. Show the list of files and directories. Create two text files T1.txt andT2.txt inside dir1. Write ten email addresses corresponding to four different domain names(@gmail.com, @yahoo.com, @rediff.com, @teamfuture.in). Segregate the email addresses with respect to their domain name and save the email addresses inside four different text files O1.txt, O2.txt, O3.txt and O4.txt inside dir2.

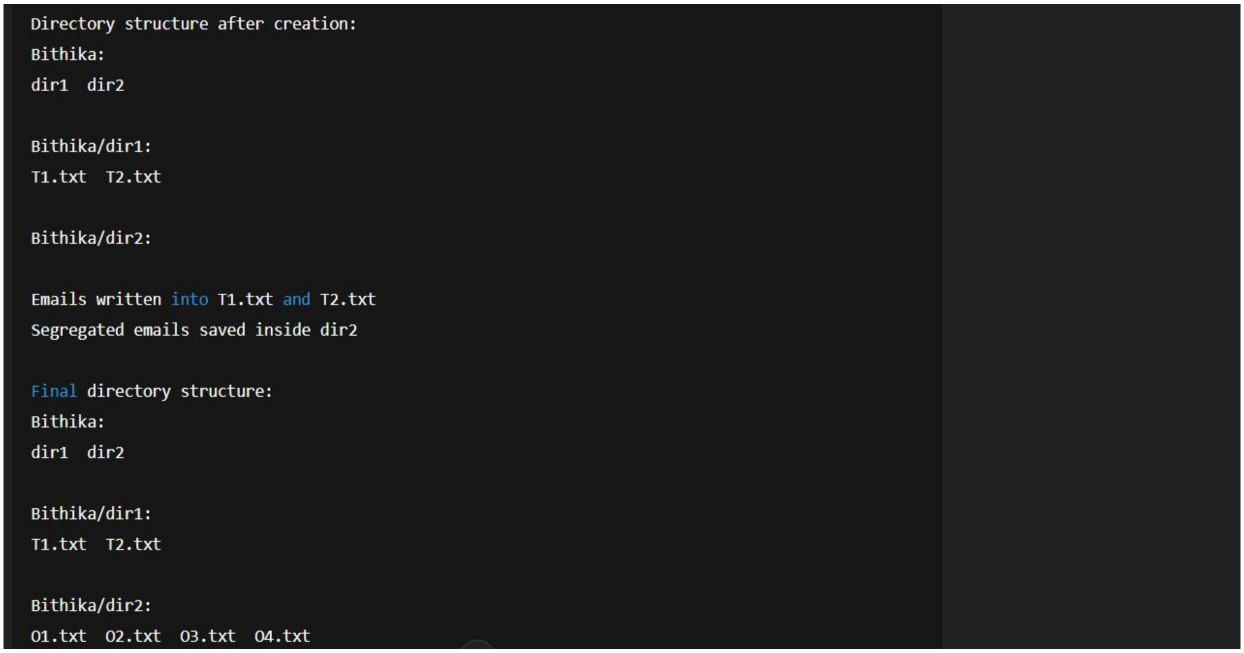
Ans:

Code:

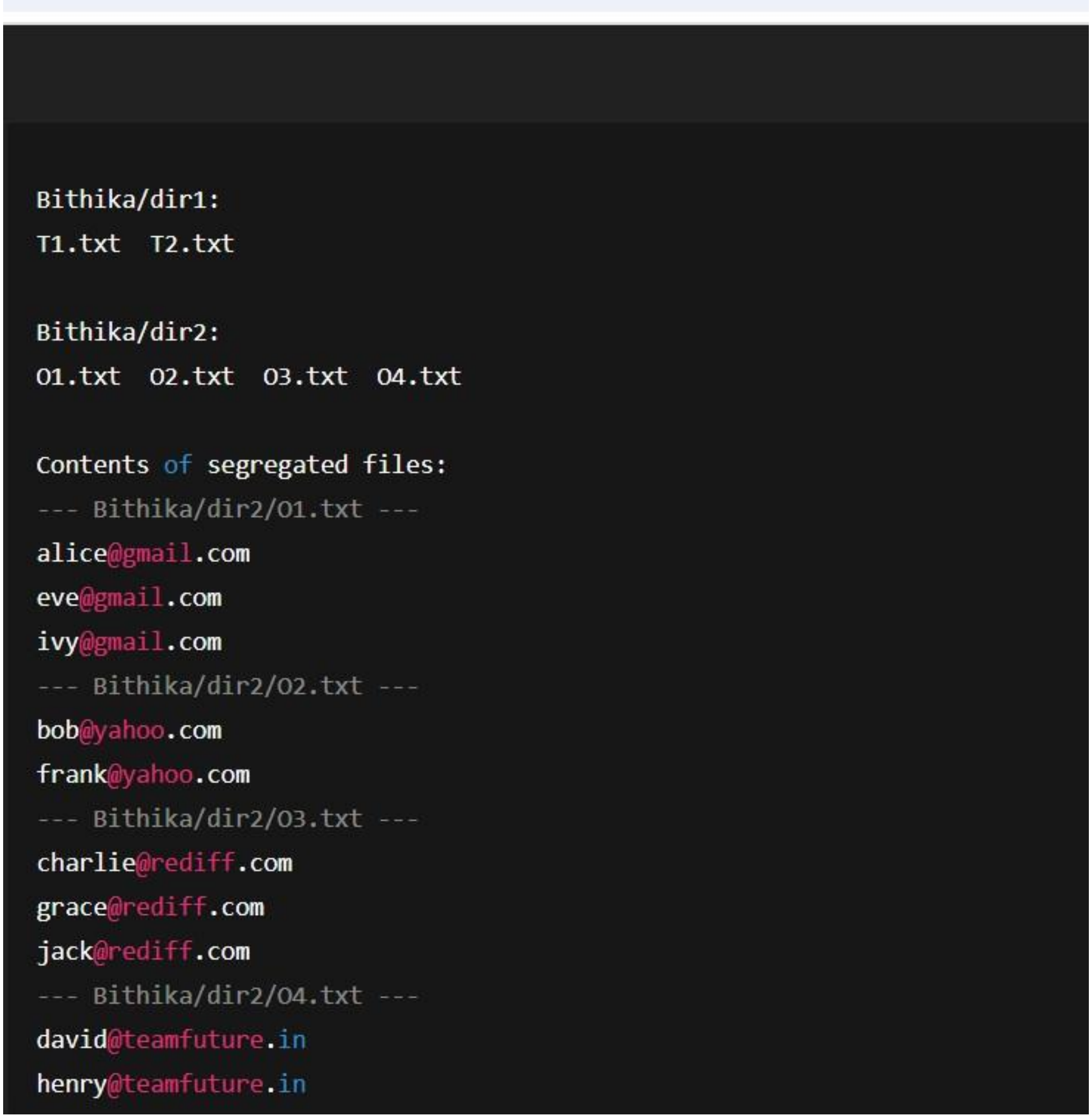


```
mkdir -p Bithika/dir1 Bithika/dir2
echo "Directory structure after creation:"
ls -R Bithika
touch Bithika/dir1/T1.txt Bithika/dir1/T2.txt
cat > Bithika/dir1/T1.txt <<EOL
alice@gmail.com
bob@yahoo.com
charlie@rediff.com
david@teamfuture.in
eve@gmail.com
frank@yahoo.com
grace@rediff.com
henry@teamfuture.in
ivy@gmail.com
jack@rediff.com
EOL
cp Bithika/dir1/T1.txt Bithika/dir1/T2.txt
echo "Emails written into T1.txt and T2.txt"
grep "@gmail.com"  Bithika/dir1/T1.txt > Bithika/dir2/O1.txt
grep "@yahoo.com"  Bithika/dir1/T1.txt > Bithika/dir2/O2.txt
grep "@rediff.com" Bithika/dir1/T1.txt > Bithika/dir2/O3.txt
grep "@teamfuture.in" Bithika/dir1/T1.txt > Bithika/dir2/O4.txt
echo "Segregated emails saved inside dir2"
echo
echo "Final directory structure:"
ls -R Bithika
echo
echo "Contents of segregated files:"
for file in Bithika/dir2/*.txt
do
    echo "--- $file ---"
    cat $file
done
```

Output:



```
Directory structure after creation:
Bithika:
dir1  dir2

Bithika/dir1:
T1.txt  T2.txt

Bithika/dir2:

Emails written into T1.txt and T2.txt
Segregated emails saved inside dir2

Final directory structure:
Bithika:
dir1  dir2

Bithika/dir1:
T1.txt  T2.txt

Bithika/dir2:
O1.txt  O2.txt  O3.txt  O4.txt
```



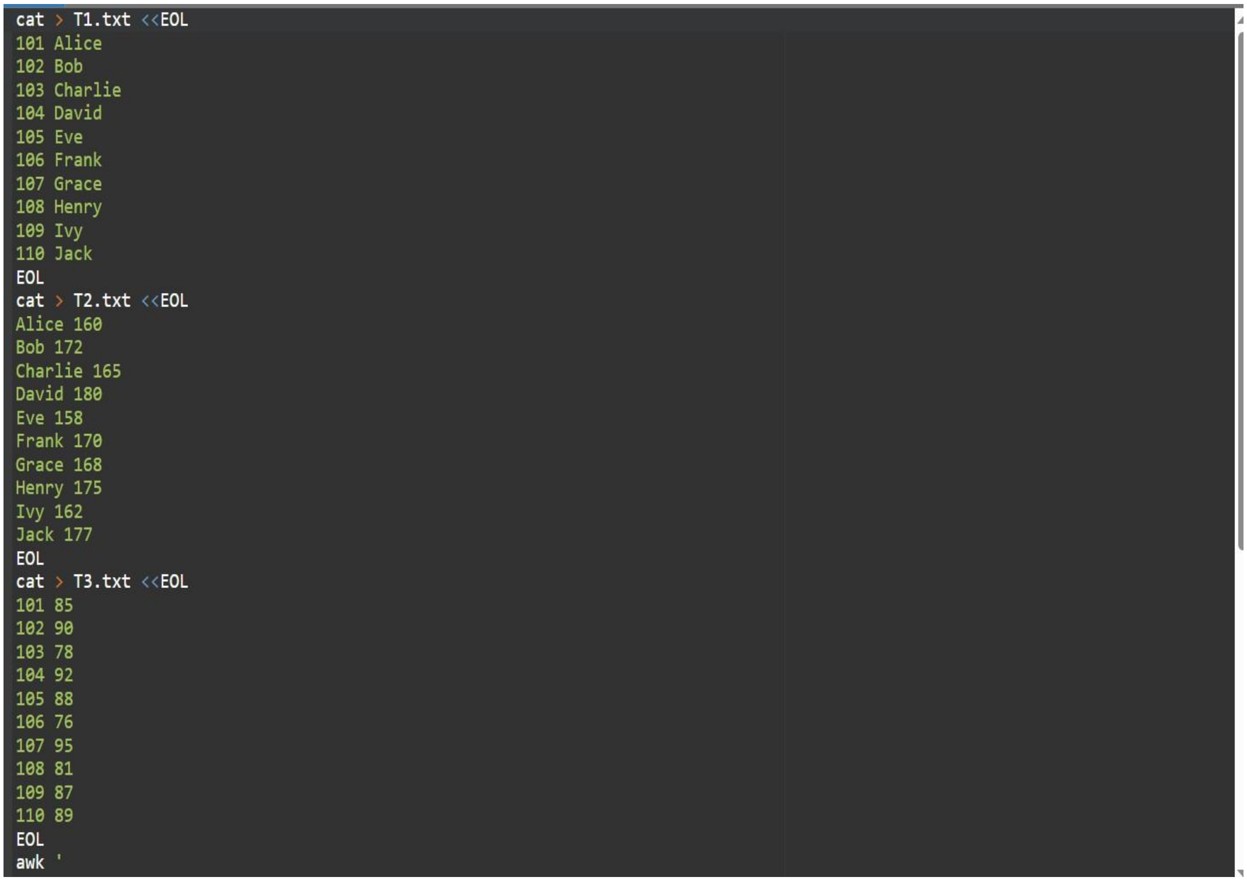
```
Bithika/dir1:
T1.txt  T2.txt

Bithika/dir2:
O1.txt  O2.txt  O3.txt  O4.txt

Contents of segregated files:
--- Bithika/dir2/O1.txt ---
alice@gmail.com
eve@gmail.com
ivy@gmail.com
--- Bithika/dir2/O2.txt ---
bob@yahoo.com
frank@yahoo.com
--- Bithika/dir2/O3.txt ---
charlie@rediff.com
grace@rediff.com
jack@rediff.com
--- Bithika/dir2/O4.txt ---
david@teamfuture.in
henry@teamfuture.in
```

6) Create three text files T1.txt containing the roll numbers and names of 10 students, T2.txt containing the names and heights of 10 students and T3.txt containing the roll numbers and grades of these same students. Find the highest grade, lowest

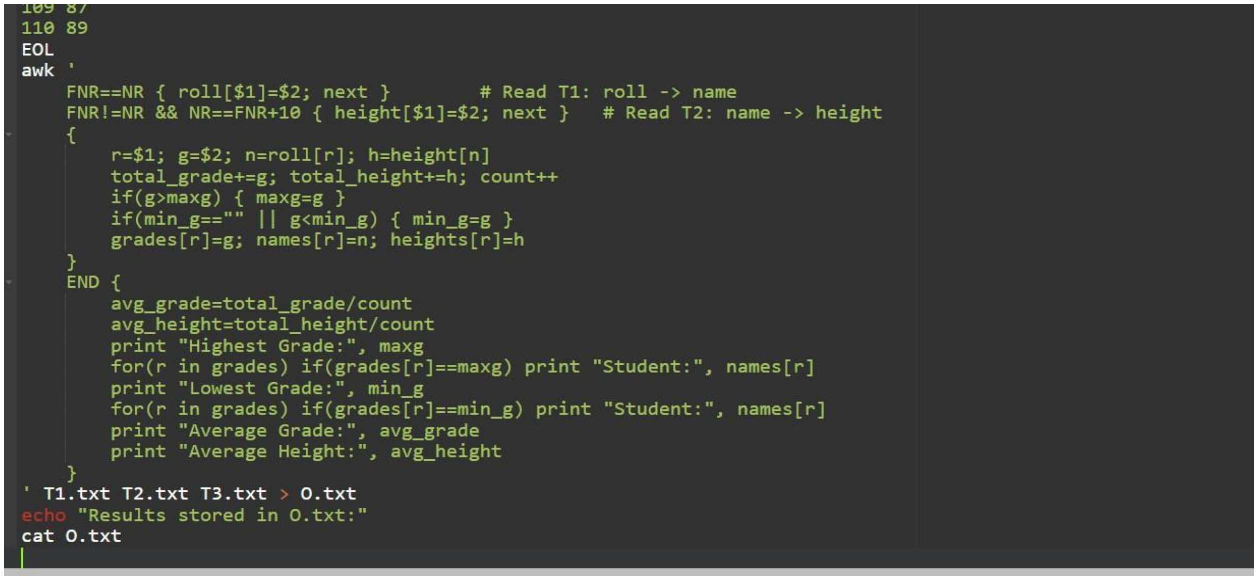
grade, average grade and average height from the given data, and also find the corresponding student name(s). Store this output in a new text file O.txt.

Ans:

Code:

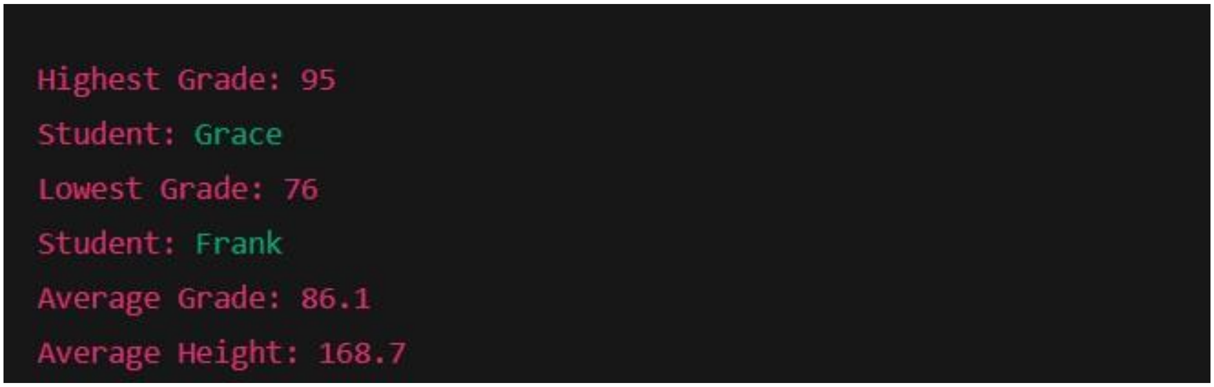


```
cat > T1.txt <<EOL
101 Alice
102 Bob
103 Charlie
104 David
105 Eve
106 Frank
107 Grace
108 Henry
109 Ivy
110 Jack
EOL
cat > T2.txt <<EOL
Alice 160
Bob 172
Charlie 165
David 180
Eve 158
Frank 170
Grace 168
Henry 175
Ivy 162
Jack 177
EOL
cat > T3.txt <<EOL
101 85
102 90
103 78
104 92
105 88
106 76
107 95
108 81
109 87
110 89
EOL
awk '
```



```
109 87
110 89
EOL
awk '
    FNR==NR { roll[$1]=$2; next }          # Read T1: roll -> name
    FNR!=NR && NR==FNR+10 { height[$1]=$2; next }   # Read T2: name -> height
    {
        r=$1; g=$2; n=roll[r]; h=height[n]
        total_grade+=g; total_height+=h; count++
        if(g>maxg) { maxg=g }
        if(min_g=="" || g<min_g) { min_g=g }
        grades[r]=g; names[r]=n; heights[r]=h
    }
    END {
        avg_grade=total_grade/count
        avg_height=total_height/count
        print "Highest Grade:", maxg
        for(r in grades) if(grades[r]==maxg) print "Student:", names[r]
        print "Lowest Grade:", min_g
        for(r in grades) if(grades[r]==min_g) print "Student:", names[r]
        print "Average Grade:", avg_grade
        print "Average Height:", avg_height
    }
' T1.txt T2.txt T3.txt > O.txt
echo "Results stored in O.txt:"
cat O.txt
```

Output:



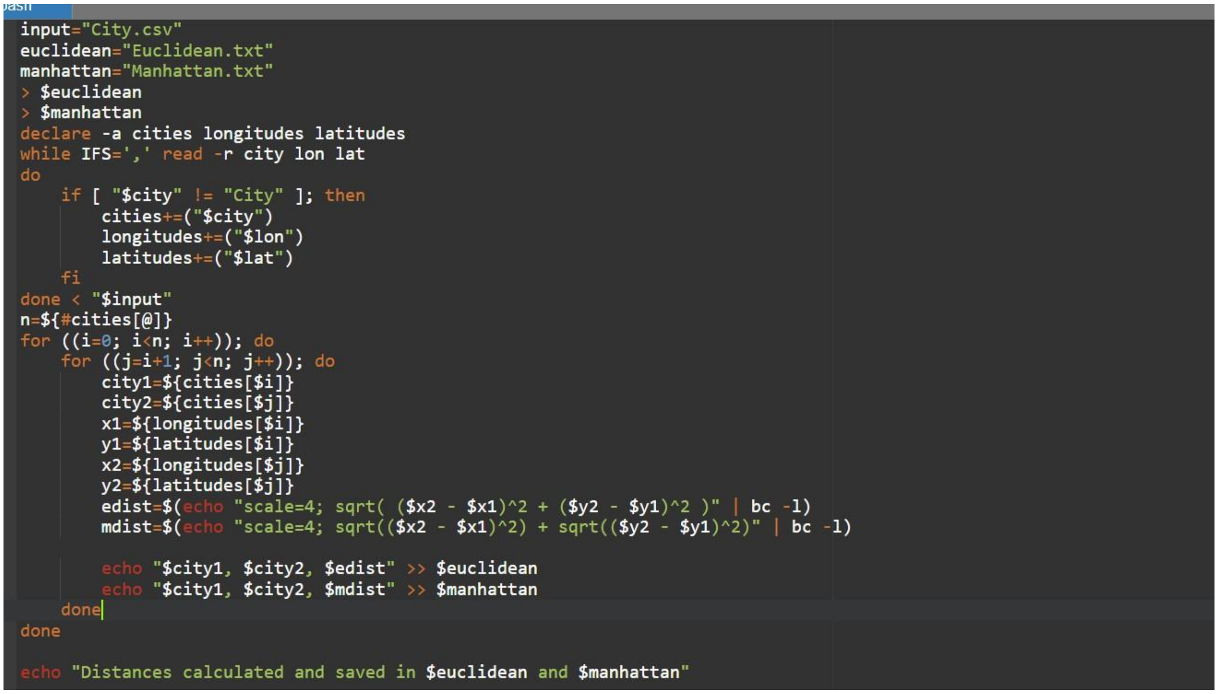
```
Highest Grade: 95
Student: Grace
Lowest Grade: 76
Student: Frank
Average Grade: 86.1
Average Height: 168.7
```

7) Calculate the Euclidean distance and Manhattan distance in between the cities. Coordinate information of the cities is provided in a file named City.csv. Euclidean and Manhattan distance between the cities

will be calculated and stored in Euclidean.txt and Manhattan.txt respectively. [Coordinate information format for City.csv: city_name, longitude, latitude;Information format for Euclidean.txt and Manhattan.txt: source_city, destination_city,distance]

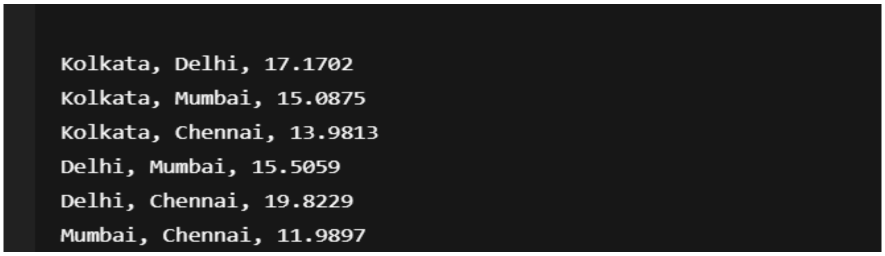
Ans:

Code:



```
input="City.csv"
euclidean="Euclidean.txt"
manhattan="Manhattan.txt"
> $euclidean
> $manhattan
declare -a cities longitudes latitudes
while IFS=',' read -r city lon lat
do
    if [ "$city" != "City" ]; then
        cities+=("$city")
        longitudes+=("$lon")
        latitudes+=("$lat")
    fi
done < "$input"
n=${#cities[@]}
for ((i=0; i<n; i++)); do
    for ((j=i+1; j<n; j++)); do
        city1=${cities[$i]}
        city2=${cities[$j]}
        x1=${longitudes[$i]}
        y1=${latitudes[$i]}
        x2=${longitudes[$j]}
        y2=${latitudes[$j]}
        edist=$(echo "scale=4; sqrt( ($x2 - $x1)^2 + ($y2 - $y1)^2 )" | bc -l)
        mdist=$(echo "scale=4; sqrt(($x2 - $x1)^2) + sqrt(($y2 - $y1)^2)" | bc -l)

        echo "$city1, $city2, $edist" >> $euclidean
        echo "$city1, $city2, $mdist" >> $manhattan
    done
done

echo "Distances calculated and saved in $euclidean and $manhattan"
```

Output:



```
Kolkata, Delhi, 17.1702
Kolkata, Mumbai, 15.0875
Kolkata, Chennai, 13.9813
Delhi, Mumbai, 15.5059
Delhi, Chennai, 19.8229
Mumbai, Chennai, 11.9897
```

Assignment -4

Implement the following scheduling algorithms using Python:

i. First Come First Serve (FCFS)

ii. Shortest Job First (SJF)

iii. Round Robin (RR)

I/P format

For FCFS algorithm:

a) Process array

b) Arrival Time (AT) array

c) Burst Time (BT) array

For SJF algorithm:

a) Process array

b) Arrival Time (AT) array

c) Burst Time (BT) array

d) Pre-emptive or non-pre-emptive choice

For RR algorithm:

a) Process array

b) Arrival Time (AT) array

c) Burst Time (BT) array

d) Time quantum

O/P format

i. Gantt chart

ii. Table columns: PID, AT, BT, CT, TAT, RT

iii. Display AWT, ATAT, ART

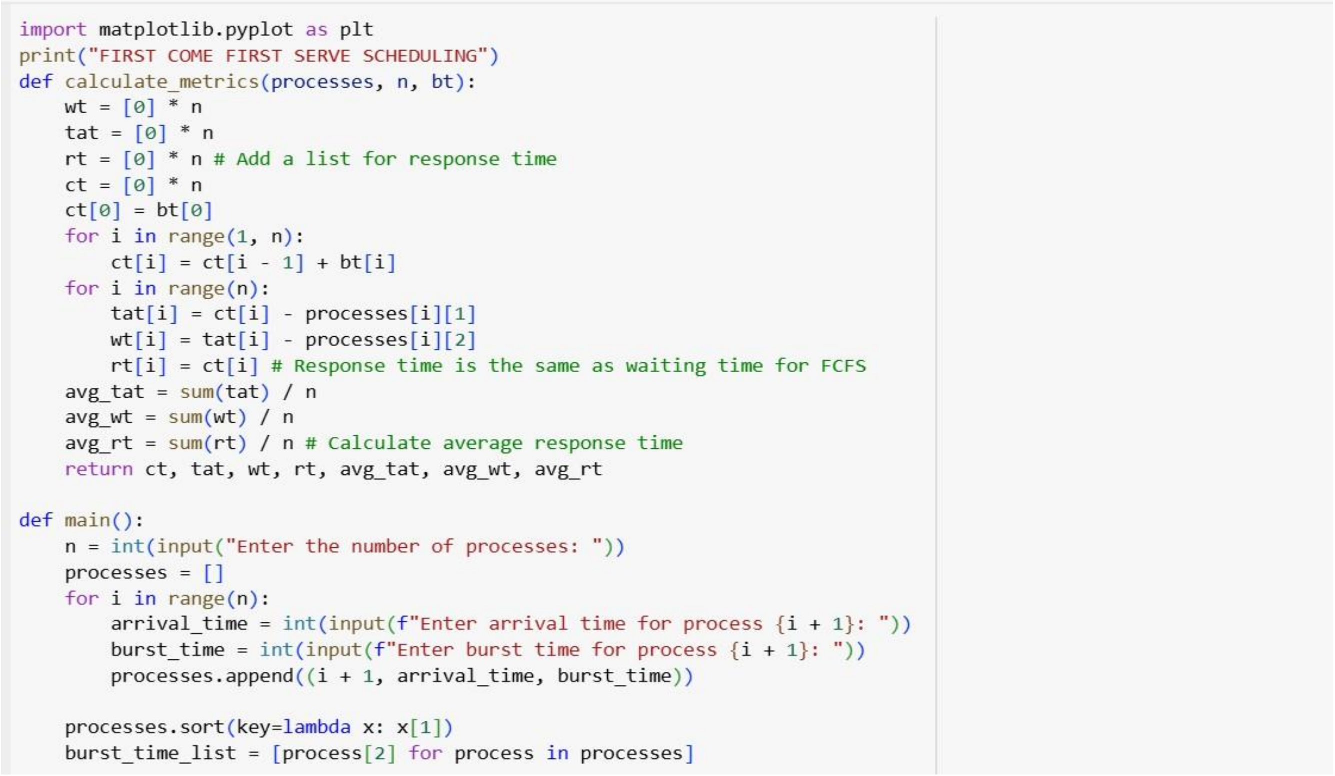
iv. For RR algorithm, change the time quantum and check ATAT, AWT, ART. Plot graphs

for each of them.

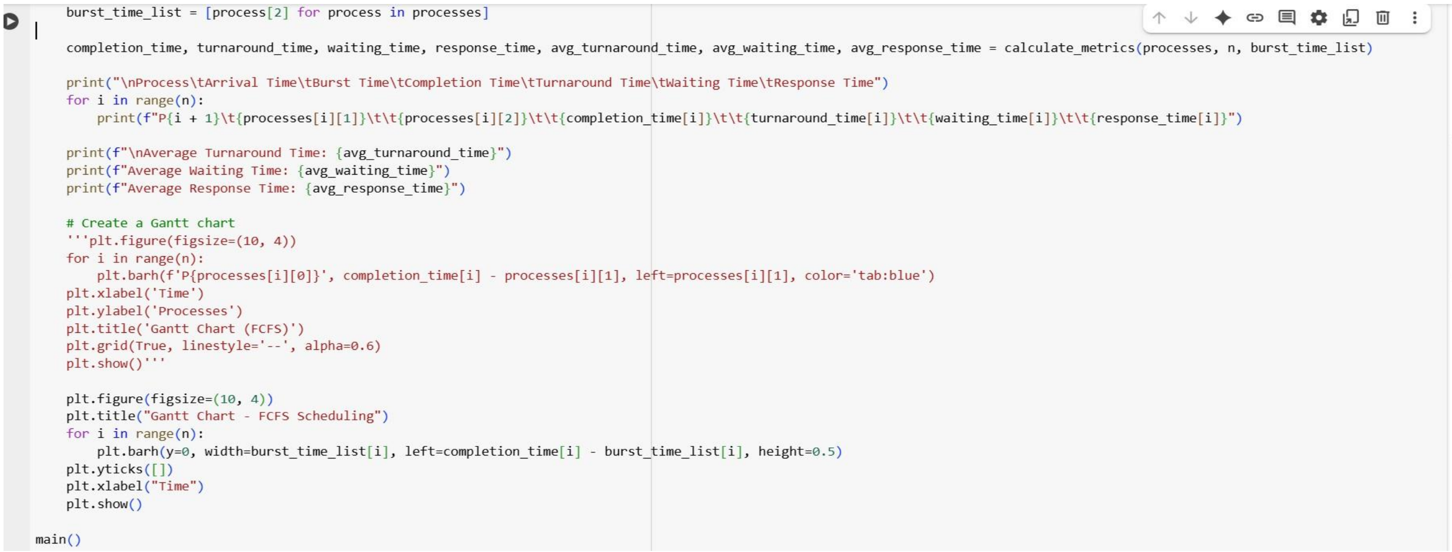
Ans:

FCFS:

Code:

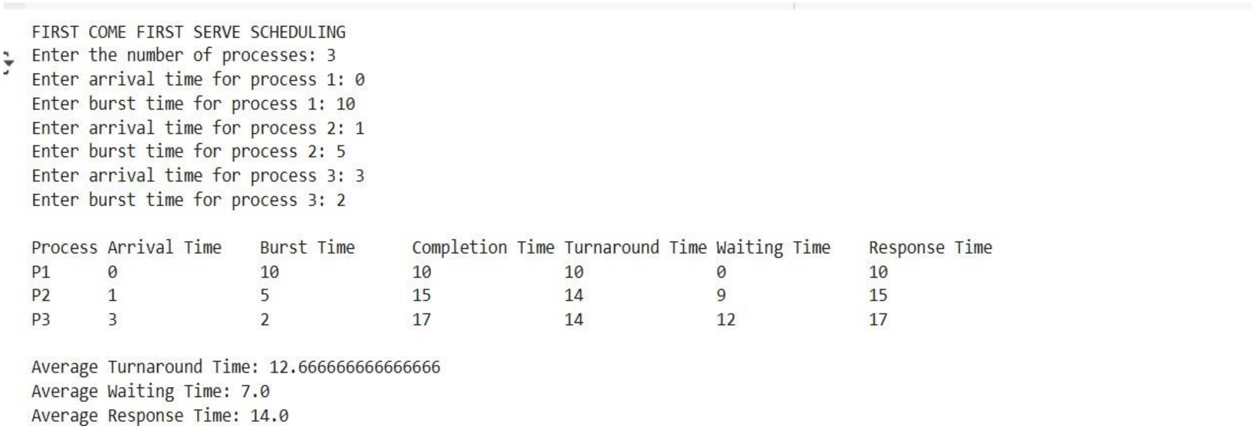


```
import matplotlib.pyplot as plt
print("FIRST COME FIRST SERVE SCHEDULING")
def calculate_metrics(processes, n, bt):
    wt = [0] * n
    tat = [0] * n
    rt = [0] * n # Add a list for response time
    ct = [0] * n
    ct[0] = bt[0]
    for i in range(1, n):
        ct[i] = ct[i - 1] + bt[i]
    for i in range(n):
        tat[i] = ct[i] - processes[i][1]
        wt[i] = tat[i] - processes[i][2]
        rt[i] = ct[i] # Response time is the same as waiting time for FCFS
    avg_tat = sum(tat) / n
    avg_wt = sum(wt) / n
    avg_rt = sum(rt) / n # Calculate average response time
    return ct, tat, wt, rt, avg_tat, avg_wt, avg_rt

def main():
    n = int(input("Enter the number of processes: "))
    processes = []
    for i in range(n):
        arrival_time = int(input(f"Enter arrival time for process {i + 1}: "))
        burst_time = int(input(f"Enter burst time for process {i + 1}: "))
        processes.append((i + 1, arrival_time, burst_time))

    processes.sort(key=lambda x: x[1])
    burst_time_list = [process[2] for process in processes]
```

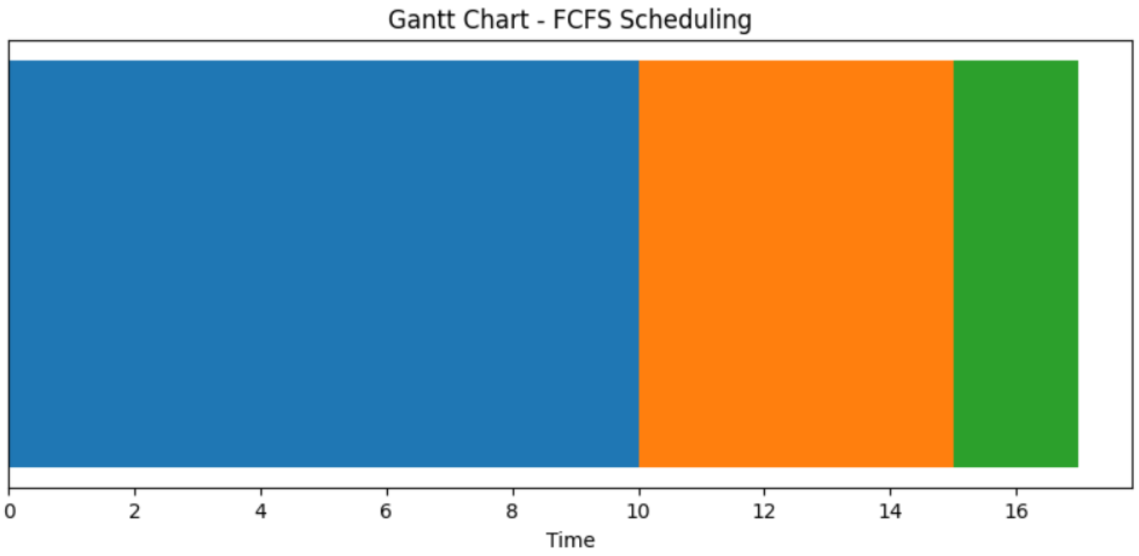


```
    burst_time_list = [process[2] for process in processes]

    completion_time, turnaround_time, waiting_time, response_time, avg_turnaround_time, avg_waiting_time, avg_response_time = calculate_metrics(processes, n, burst_time_list)

    print("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\tResponse Time")
    for i in range(n):
        print(f"P{i + 1}\t{processes[i][1]}\t\t{processes[i][2]}\t\t{completion_time[i]}\t\t{turnaround_time[i]}\t\t{waiting_time[i]}\t\t{response_time[i]}")

    print(f"\nAverage Turnaround Time: {avg_turnaround_time}")
    print(f"Average Waiting Time: {avg_waiting_time}")
    print(f"Average Response Time: {avg_response_time}")

    # Create a Gantt chart
    '''plt.figure(figsize=(10, 4))
    for i in range(n):
        plt.barh(f'P{processes[i][0]}', completion_time[i] - processes[i][1], left=processes[i][1], color='tab:blue')
    plt.xlabel('Time')
    plt.ylabel('Processes')
    plt.title('Gantt Chart (FCFS)')
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.show()'''

    plt.figure(figsize=(10, 4))
    plt.title("Gantt Chart - FCFS Scheduling")
    for i in range(n):
        plt.barh(y=0, width=burst_time_list[i], left=completion_time[i] - burst_time_list[i], height=0.5)
    plt.yticks([])
    plt.xlabel("Time")
    plt.show()

main()
```

Output:

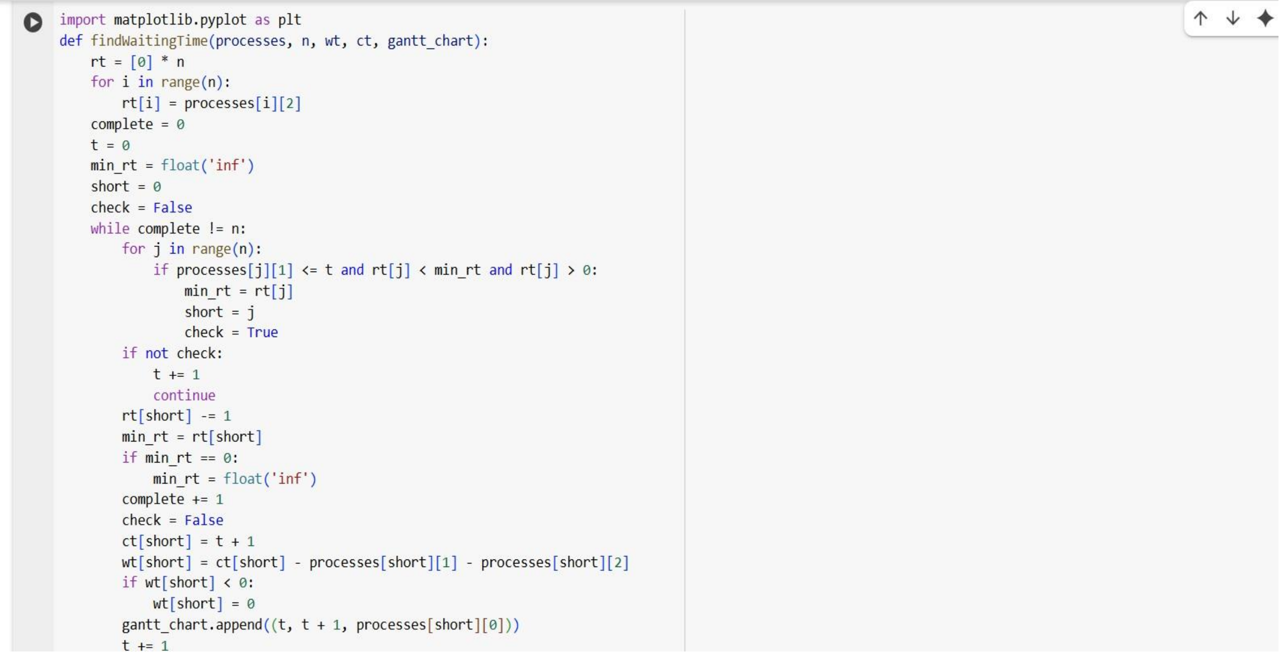


```
FIRST COME FIRST SERVE SCHEDULING
Enter the number of processes: 3
Enter arrival time for process 1: 0
Enter burst time for process 1: 10
Enter arrival time for process 2: 1
Enter burst time for process 2: 5
Enter arrival time for process 3: 3
Enter burst time for process 3: 2

Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time    Response Time
P1      0               10              10              10              0               10
P2      1               5               15              14              9               15
P3      3               2               17              14              12              17

Average Turnaround Time: 12.666666666666666
Average Waiting Time: 7.0
Average Response Time: 14.0
```

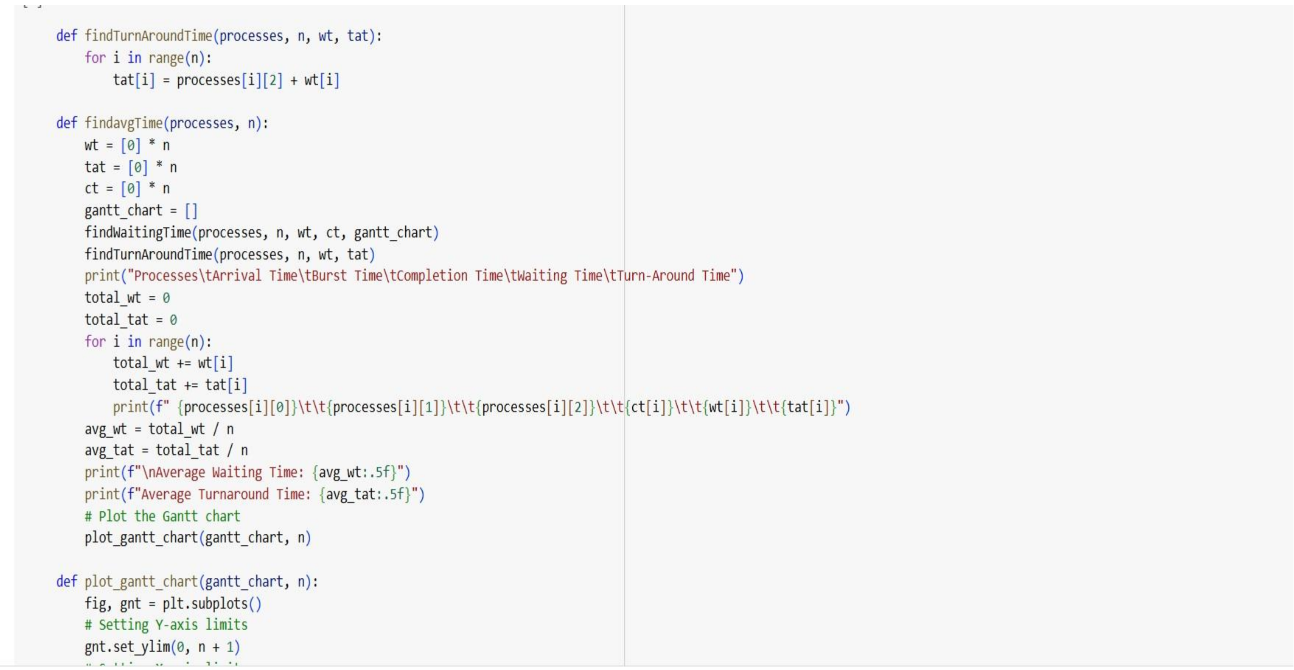


SJF:

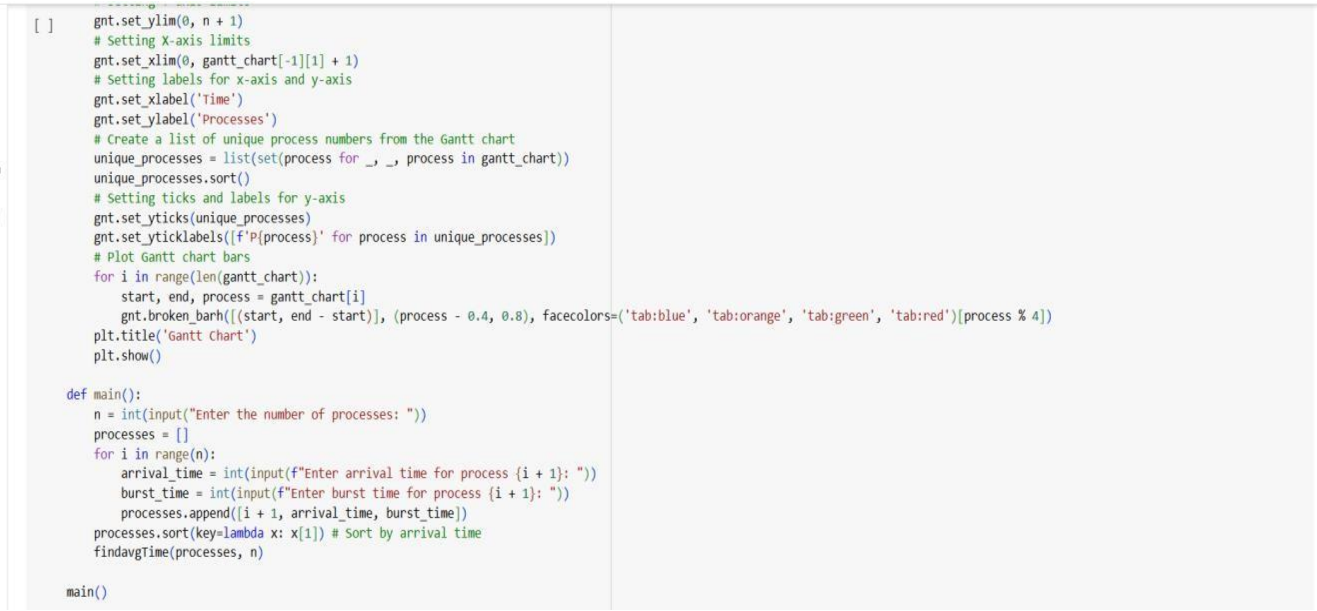
Code:



```
import matplotlib.pyplot as plt
def findWaitingTime(processes, n, wt, ct, gantt_chart):
    rt = [0] * n
    for i in range(n):
        rt[i] = processes[i][2]
    complete = 0
    t = 0
    min_rt = float('inf')
    short = 0
    check = False
    while complete != n:
        for j in range(n):
            if processes[j][1] <= t and rt[j] < min_rt and rt[j] > 0:
                min_rt = rt[j]
                short = j
                check = True
        if not check:
            t += 1
            continue
        rt[short] -= 1
        min_rt = rt[short]
        if min_rt == 0:
            min_rt = float('inf')
        complete += 1
        check = False
        ct[short] = t + 1
        wt[short] = ct[short] - processes[short][1] - processes[short][2]
        if wt[short] < 0:
            wt[short] = 0
        gantt_chart.append((t, t + 1, processes[short][0]))
        t += 1
```

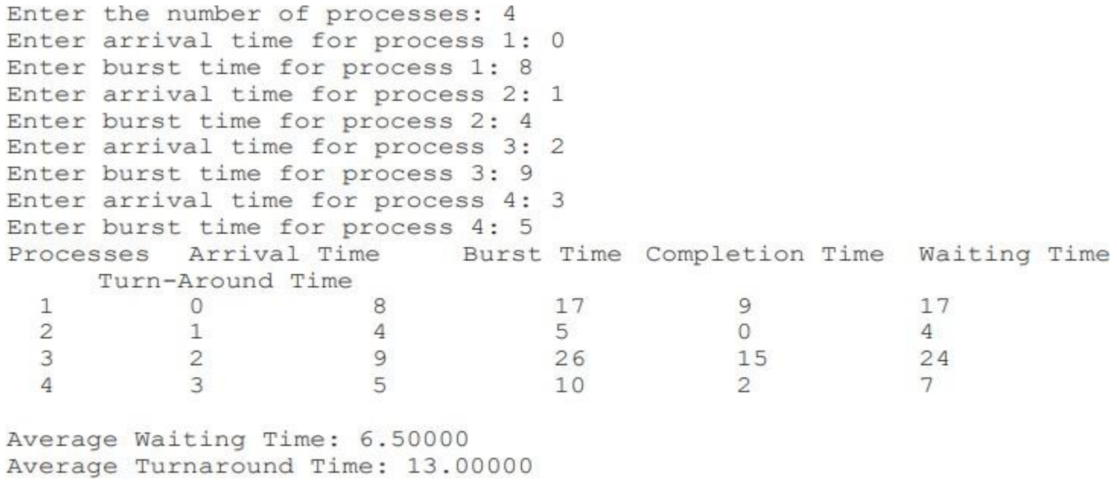


```
def findTurnAroundTime(processes, n, wt, tat):
    for i in range(n):
        tat[i] = processes[i][2] + wt[i]

def findavgTime(processes, n):
    wt = [0] * n
    tat = [0] * n
    ct = [0] * n
    gantt_chart = []
    findWaitingTime(processes, n, wt, ct, gantt_chart)
    findTurnAroundTime(processes, n, wt, tat)
    print("Processes\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurn-Around Time")
    total_wt = 0
    total_tat = 0
    for i in range(n):
        total_wt += wt[i]
        total_tat += tat[i]
        print(f" {processes[i][0]}\t\t{processes[i][1]}\t\t{processes[i][2]}\t\t{ct[i]}\t\t{wt[i]}\t\t{tat[i]}")
    avg_wt = total_wt / n
    avg_tat = total_tat / n
    print(f"\nAverage Waiting Time: {avg_wt:.5f}")
    print(f"Average Turnaround Time: {avg_tat:.5f}")
    # Plot the Gantt chart
    plot_gantt_chart(gantt_chart, n)

def plot_gantt_chart(gantt_chart, n):
    fig, gnt = plt.subplots()
    # Setting Y-axis limits
    gnt.set_ylim(0, n + 1)
```

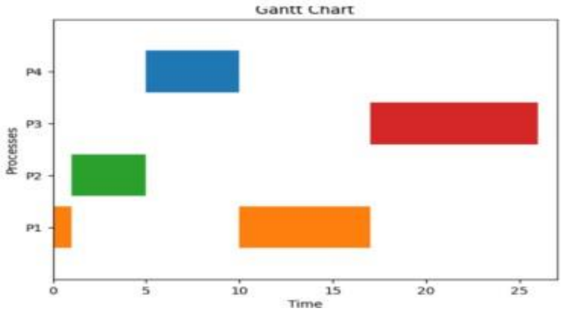


```
    gnt.set_ylim(0, n + 1)
    # Setting X-axis limits
    gnt.set_xlim(0, gantt_chart[-1][1] + 1)
    # Setting labels for x-axis and y-axis
    gnt.set_xlabel('Time')
    gnt.set_ylabel('Processes')
    # Create a list of unique process numbers from the Gantt chart
    unique_processes = list(set(process for _, _, process in gantt_chart))
    unique_processes.sort()
    # Setting ticks and labels for y-axis
    gnt.set_yticks(unique_processes)
    gnt.set_yticklabels([f'P{process}' for process in unique_processes])
    # Plot Gantt chart bars
    for i in range(len(gantt_chart)):
        start, end, process = gantt_chart[i]
        gnt.broken_barh([(start, end - start)], (process - 0.4, 0.8), facecolors=('tab:blue', 'tab:orange', 'tab:green', 'tab:red')[process % 4])
    plt.title('Gantt Chart')
    plt.show()

def main():
    n = int(input("Enter the number of processes: "))
    processes = []
    for i in range(n):
        arrival_time = int(input(f"Enter arrival time for process {i + 1}: "))
        burst_time = int(input(f"Enter burst time for process {i + 1}: "))
        processes.append([i + 1, arrival_time, burst_time])
    processes.sort(key=lambda x: x[1]) # Sort by arrival time
    findavgTime(processes, n)

main()
```

Output:

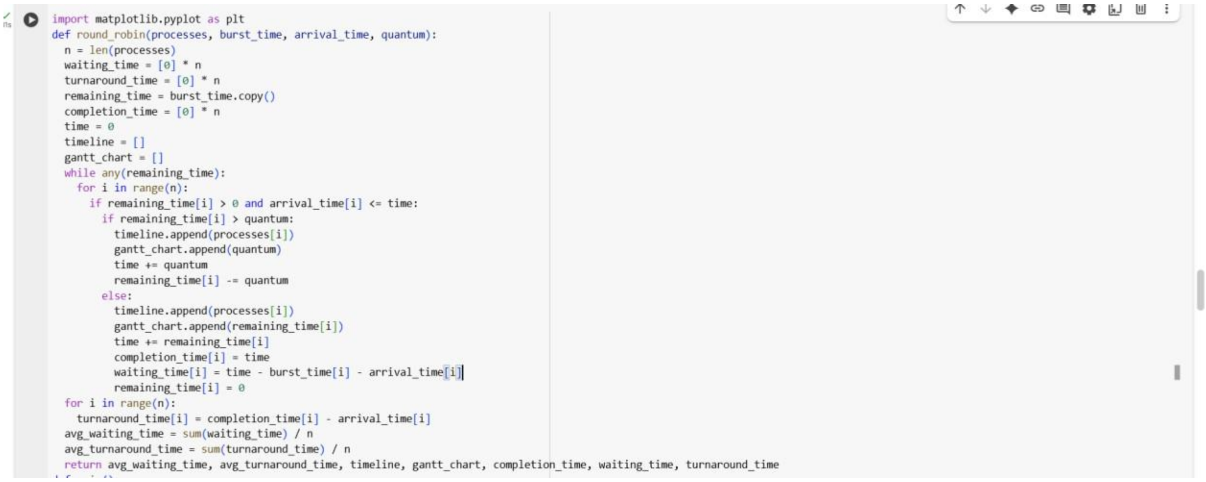


```
Enter the number of processes: 4
Enter arrival time for process 1: 0
Enter burst time for process 1: 8
Enter arrival time for process 2: 1
Enter burst time for process 2: 4
Enter arrival time for process 3: 2
Enter burst time for process 3: 9
Enter arrival time for process 4: 3
Enter burst time for process 4: 5
Processes  Arrival Time    Burst Time  Completion Time  Waiting Time
     Turn-Around Time
  1          0              8           17             9             17
  2          1              4           5              0             4
  3          2              9           26             15            24
  4          3              5           10             2             7

Average Waiting Time: 6.50000
Average Turnaround Time: 13.00000
```

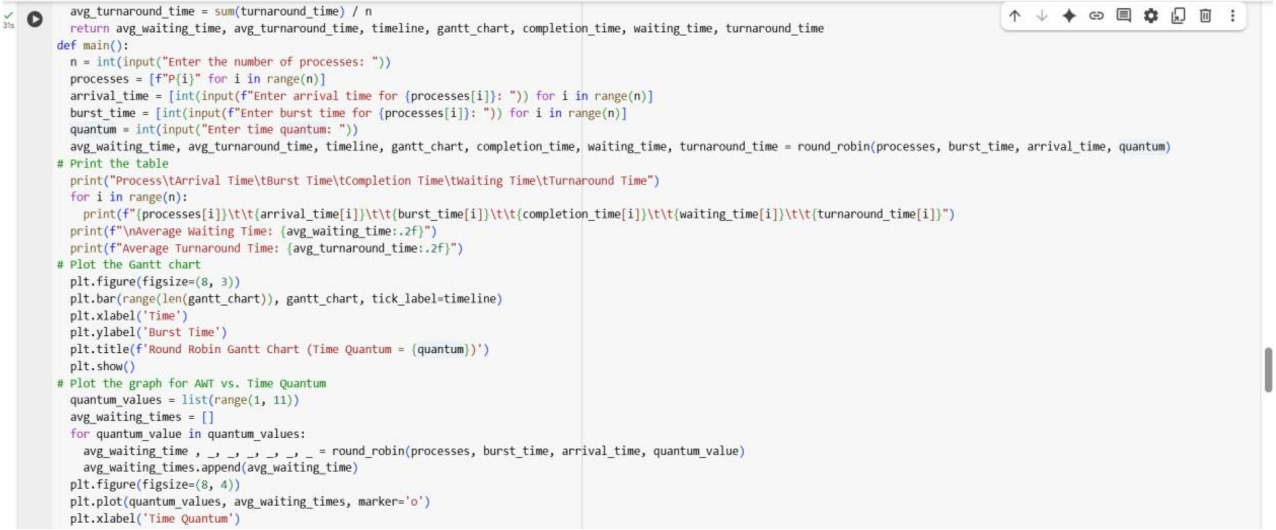


RR:

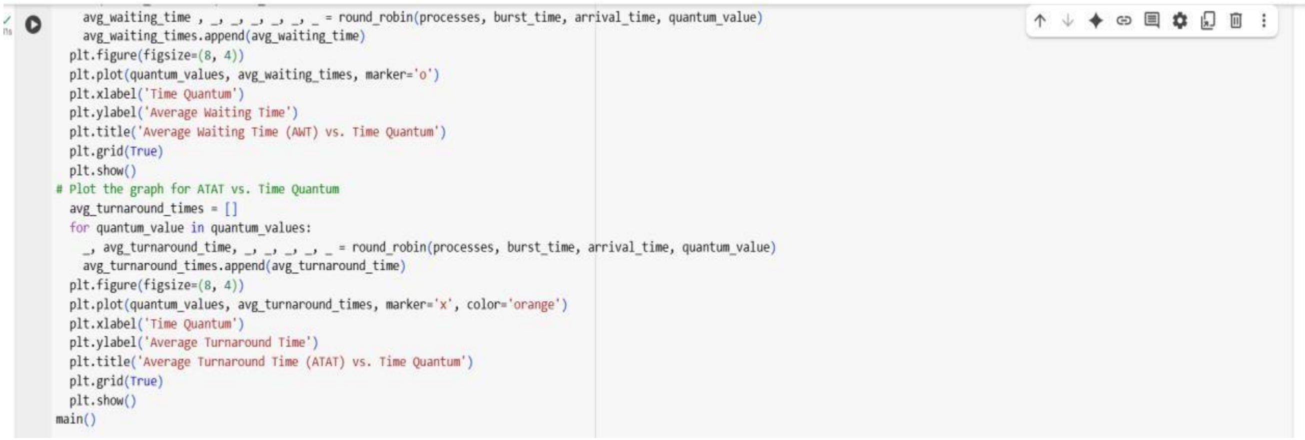
Code:



```
import matplotlib.pyplot as plt
def round_robin(processes, burst_time, arrival_time, quantum):
  n = len(processes)
  waiting_time = [0] * n
  turnaround_time = [0] * n
  remaining_time = burst_time.copy()
  completion_time = [0] * n
  time = 0
  timeline = []
  gantt_chart = []
  while any(remaining_time):
    for i in range(n):
      if remaining_time[i] > 0 and arrival_time[i] <= time:
        if remaining_time[i] > quantum:
          timeline.append(processes[i])
          gantt_chart.append(quantum)
          time += quantum
          remaining_time[i] -= quantum
        else:
          timeline.append(processes[i])
          gantt_chart.append(remaining_time[i])
          time += remaining_time[i]
          completion_time[i] = time
          waiting_time[i] = time - burst_time[i] - arrival_time[i]
          remaining_time[i] = 0
  for i in range(n):
    turnaround_time[i] = completion_time[i] - arrival_time[i]
  avg_waiting_time = sum(waiting_time) / n
  avg_turnaround_time = sum(turnaround_time) / n
  return avg_waiting_time, avg_turnaround_time, timeline, gantt_chart, completion_time, waiting_time, turnaround_time
```

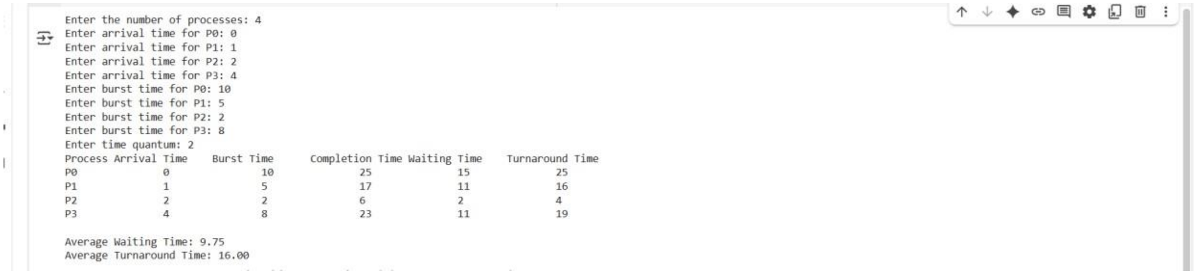


```
  avg_turnaround_time = sum(turnaround_time) / n
  return avg_waiting_time, avg_turnaround_time, timeline, gantt_chart, completion_time, waiting_time, turnaround_time
def main():
  n = int(input("Enter the number of processes: "))
  processes = [f"P{i}" for i in range(n)]
  arrival_time = [int(input(f"Enter arrival time for {processes[i]}: ")) for i in range(n)]
  burst_time = [int(input(f"Enter burst time for {processes[i]}: ")) for i in range(n)]
  quantum = int(input("Enter time quantum: "))
  avg_waiting_time, avg_turnaround_time, timeline, gantt_chart, completion_time, waiting_time, turnaround_time = round_robin(processes, burst_time, arrival_time, quantum)
# Print the table
  print("Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time")
  for i in range(n):
    print(f"{processes[i]}\t\t{arrival_time[i]}\t\t{burst_time[i]}\t\t{completion_time[i]}\t\t{waiting_time[i]}\t\t{turnaround_time[i]}")
  print(f"\nAverage Waiting Time: {avg_waiting_time:.2f}")
  print(f"Average Turnaround Time: {avg_turnaround_time:.2f}")
# Plot the Gantt chart
  plt.figure(figsize=(8, 3))
  plt.bar(range(len(gantt_chart)), gantt_chart, tick_label=timeline)
  plt.xlabel('Time')
  plt.ylabel('Burst Time')
  plt.title(f'Round Robin Gantt Chart (Time Quantum = {quantum})')
  plt.show()
# Plot the graph for AWT vs. Time Quantum
  quantum_values = list(range(1, 11))
  avg_waiting_times = []
  for quantum_value in quantum_values:
    avg_waiting_time , _, _, _, _, _, _ = round_robin(processes, burst_time, arrival_time, quantum_value)
    avg_waiting_times.append(avg_waiting_time)
  plt.figure(figsize=(8, 4))
  plt.plot(quantum_values, avg_waiting_times, marker='o')
  plt.xlabel('Time Quantum')
```

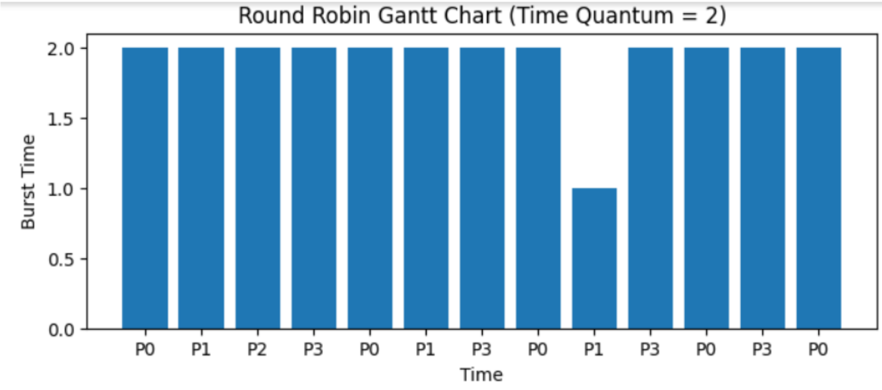


```
    avg_waiting_time , _, _, _, _, _, _ = round_robin(processes, burst_time, arrival_time, quantum_value)
    avg_waiting_times.append(avg_waiting_time)
  plt.figure(figsize=(8, 4))
  plt.plot(quantum_values, avg_waiting_times, marker='o')
  plt.xlabel('Time Quantum')
  plt.ylabel('Average Waiting Time')
  plt.title('Average Waiting Time (AWT) vs. Time Quantum')
  plt.grid(True)
  plt.show()
# Plot the graph for ATAT vs. Time Quantum
  avg_turnaround_times = []
  for quantum_value in quantum_values:
    _, avg_turnaround_time, _, _, _, _, _ = round_robin(processes, burst_time, arrival_time, quantum_value)
    avg_turnaround_times.append(avg_turnaround_time)
  plt.figure(figsize=(8, 4))
  plt.plot(quantum_values, avg_turnaround_times, marker='x', color='orange')
  plt.xlabel('Time Quantum')
  plt.ylabel('Average Turnaround Time')
  plt.title('Average Turnaround Time (ATAT) vs. Time Quantum')
  plt.grid(True)
  plt.show()
main()
```

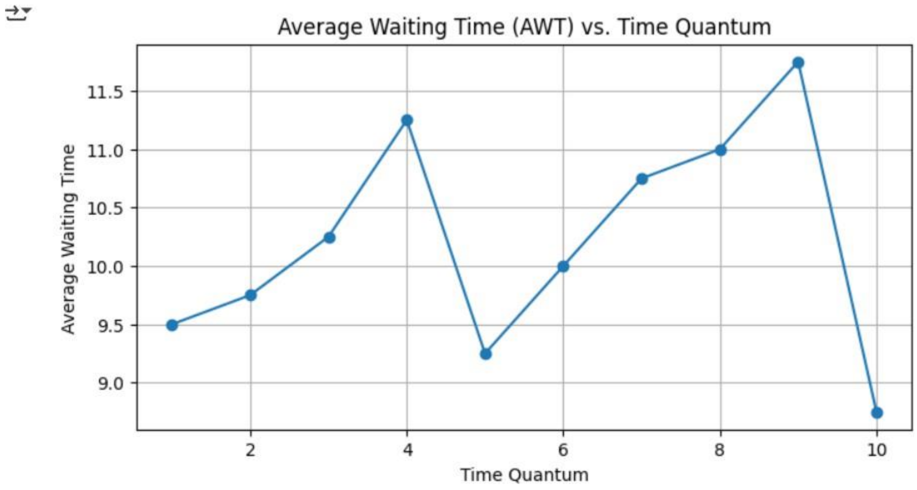
Output:



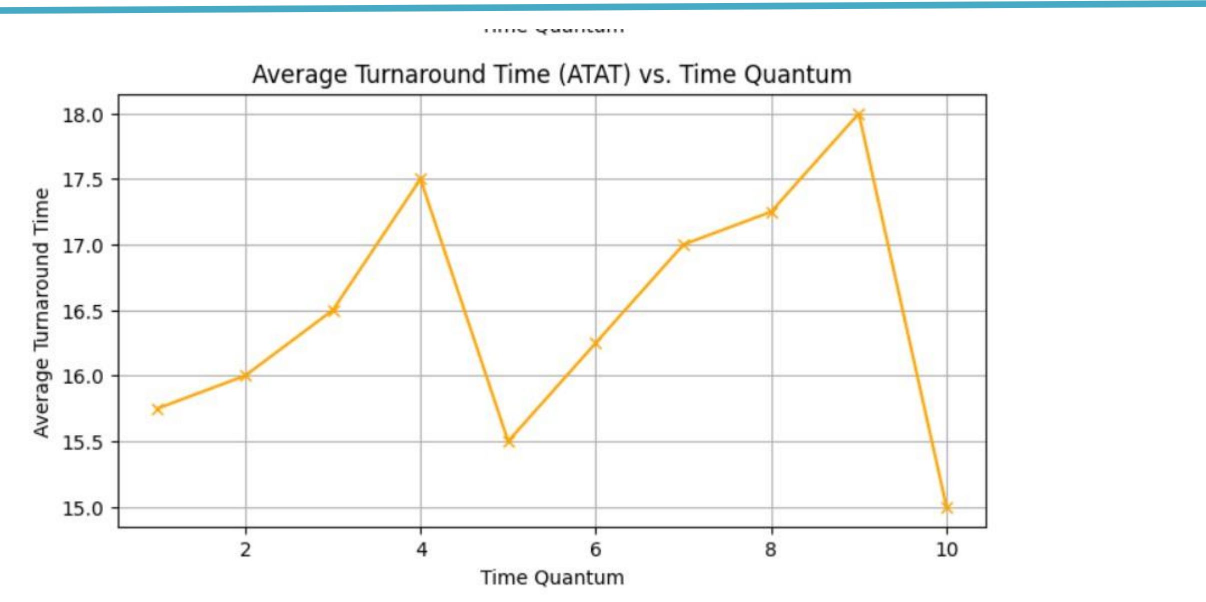
```
Enter the number of processes: 4
Enter arrival time for P0: 0
Enter arrival time for P1: 1
Enter arrival time for P2: 2
Enter arrival time for P3: 4
Enter burst time for P0: 10
Enter burst time for P1: 5
Enter burst time for P2: 2
Enter burst time for P3: 8
Enter time quantum: 2
Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
P0              0               10              25              15              25
P1              1               5               17              11              16
P2              2               2               6               2               4
P3              4               8               23              11              19

Average Waiting Time: 9.75
Average Turnaround Time: 16.00
```



Round Robin Gantt Chart (Time Quantum = 2)



Average Waiting Time (AWT) vs. Time Quantum


Time Quantum
Average Turnaround Time (ATAT) vs. Time Quantum
Average Turnaround Time
18.0
17.5
17.0
16.5
16.0
15.5
15.0
2
4
6
8
10
Time Quantum

Assignment-5:

1.

i. What is system call?

ii. Briefly demonstrate fork() system call, and its working principle.

Ans:

(i) A system call is a programmatic way for a user-level program to request services from the operating system (kernel).It provides an interface between user programs and the kernel, allowing operations like file handling, process control, and communication.

Examples: open(), read(), write(), fork(), exec(), exit().

Key Points:

- System calls are privileged instructions, executed in kernel mode.
- User programs cannot access hardware directly, so they use system calls.

(ii) fork() is a system call used to create a new process by duplicating the calling (parent) process.The new process is called the child process. Both parent and child continue execution from the point of fork().

```
#include <unistd.h>

pid_t fork(void);
```

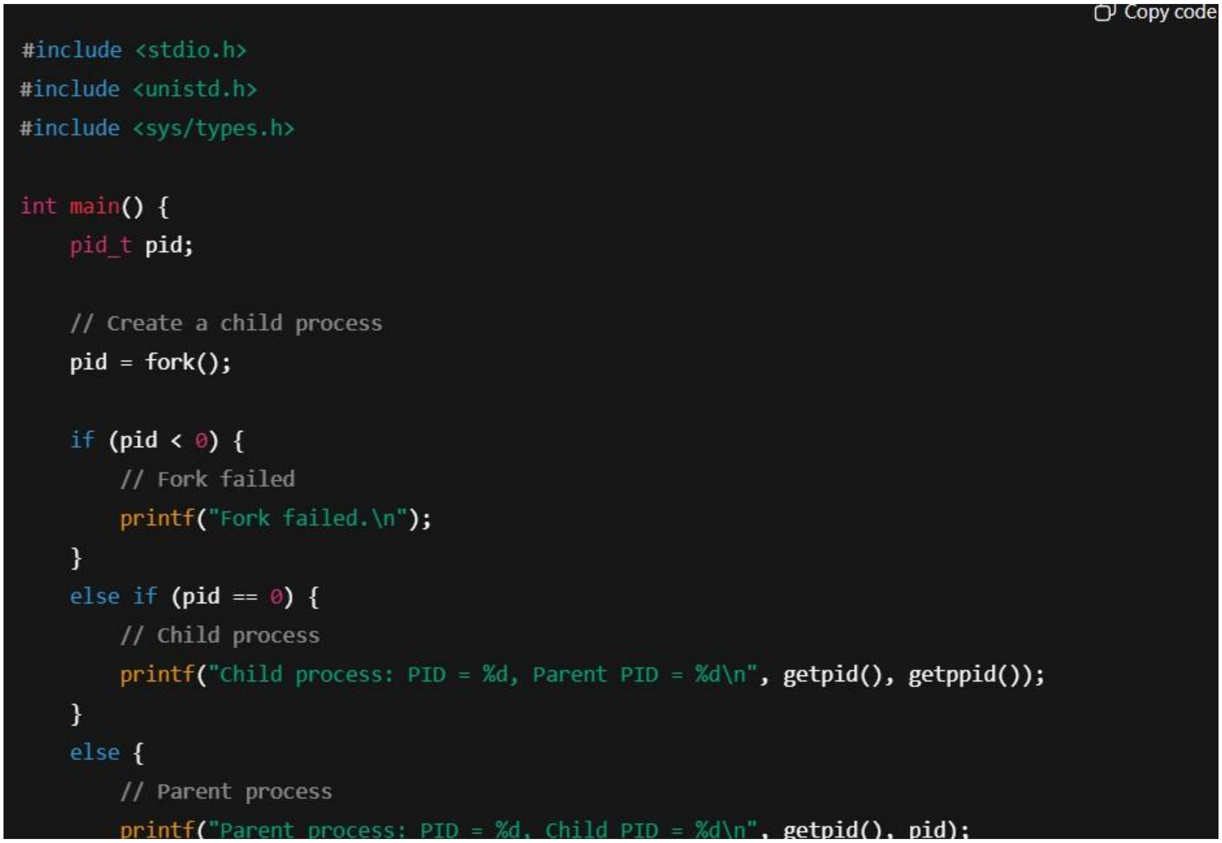
Working Principle

1. Parent process calls fork().
2. Kernel creates a copy of the parent process's address space for the child.
3. Both processes run concurrently.
4. Each process can execute different code using the return value of fork().

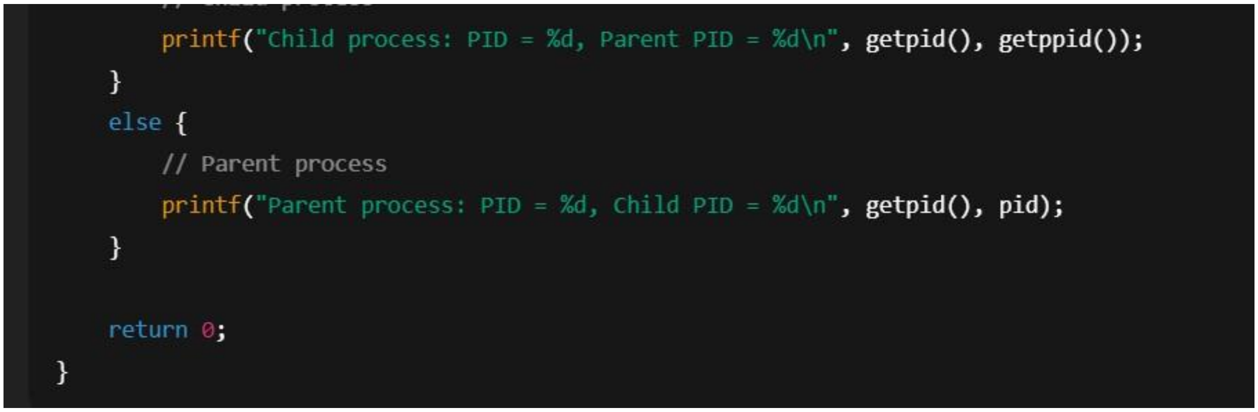
2 (i). Write a program in C, which will illustrate the fork() system call and getpid() system call.

Ans:

Code:

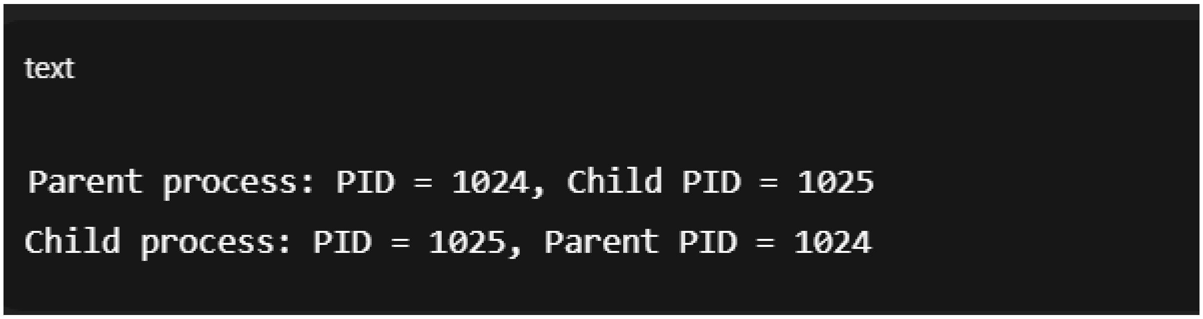


```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        printf("Fork failed.\n");
    }
    else if (pid == 0) {
        // Child process
        printf("Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());
    }
    else {
        // Parent process
        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
```



```
        // Child process
        printf("Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());
    }
    else {
        // Parent process
        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
    }

    return 0;
}
```

Output:

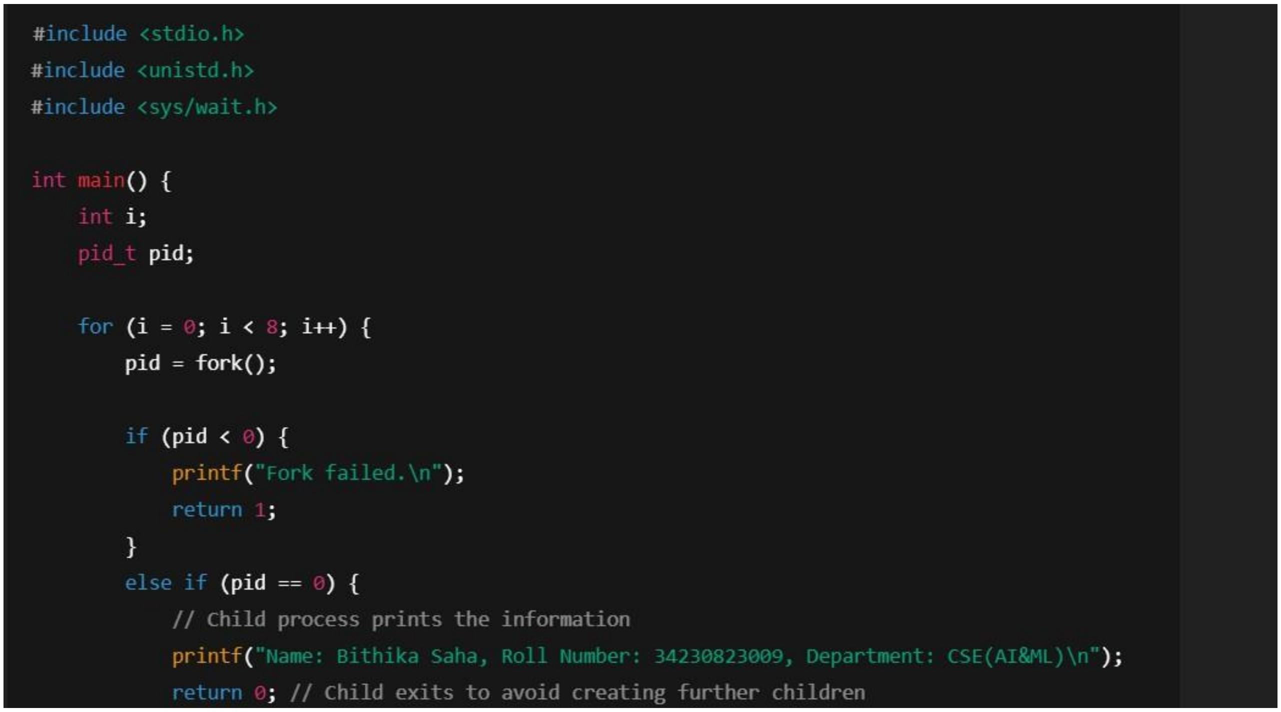


```
text

Parent process: PID = 1024, Child PID = 1025
Child process: PID = 1025, Parent PID = 1024
```

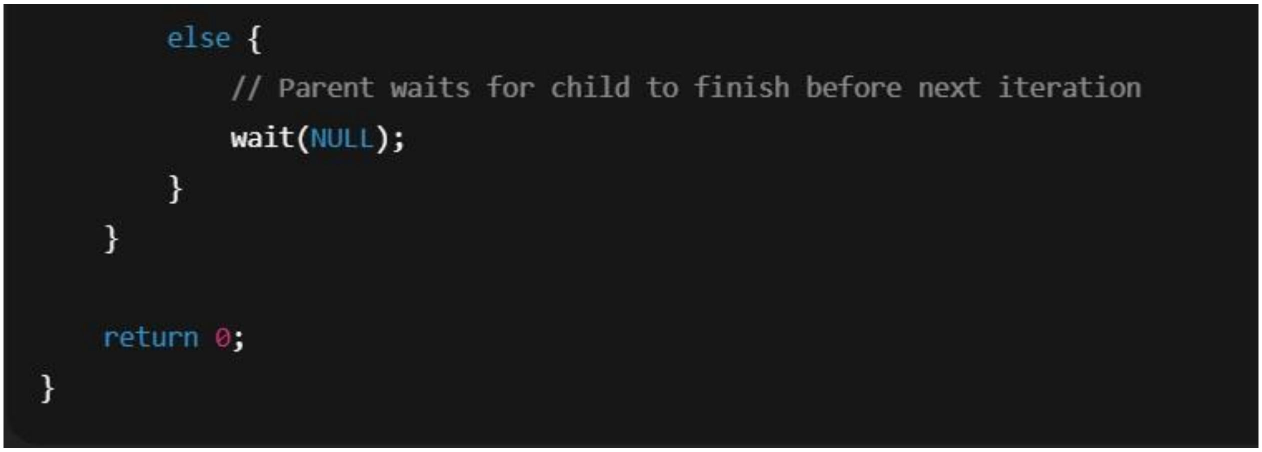
(ii) Write a program in C, which will copy your name, roll_number and department name exactly 8 times, by using fork() system call.

Ans:

Code:

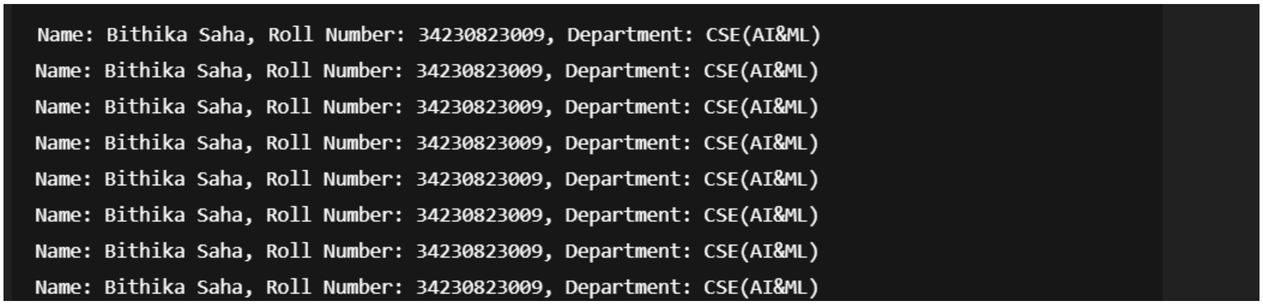


```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int i;
    pid_t pid;

    for (i = 0; i < 8; i++) {
        pid = fork();

        if (pid < 0) {
            printf("Fork failed.\n");
            return 1;
        }
        else if (pid == 0) {
            // Child process prints the information
            printf("Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)\n");
            return 0; // Child exits to avoid creating further children
```



```
        else {
            // Parent waits for child to finish before next iteration
            wait(NULL);
        }
    }

    return 0;
}
```

Output:

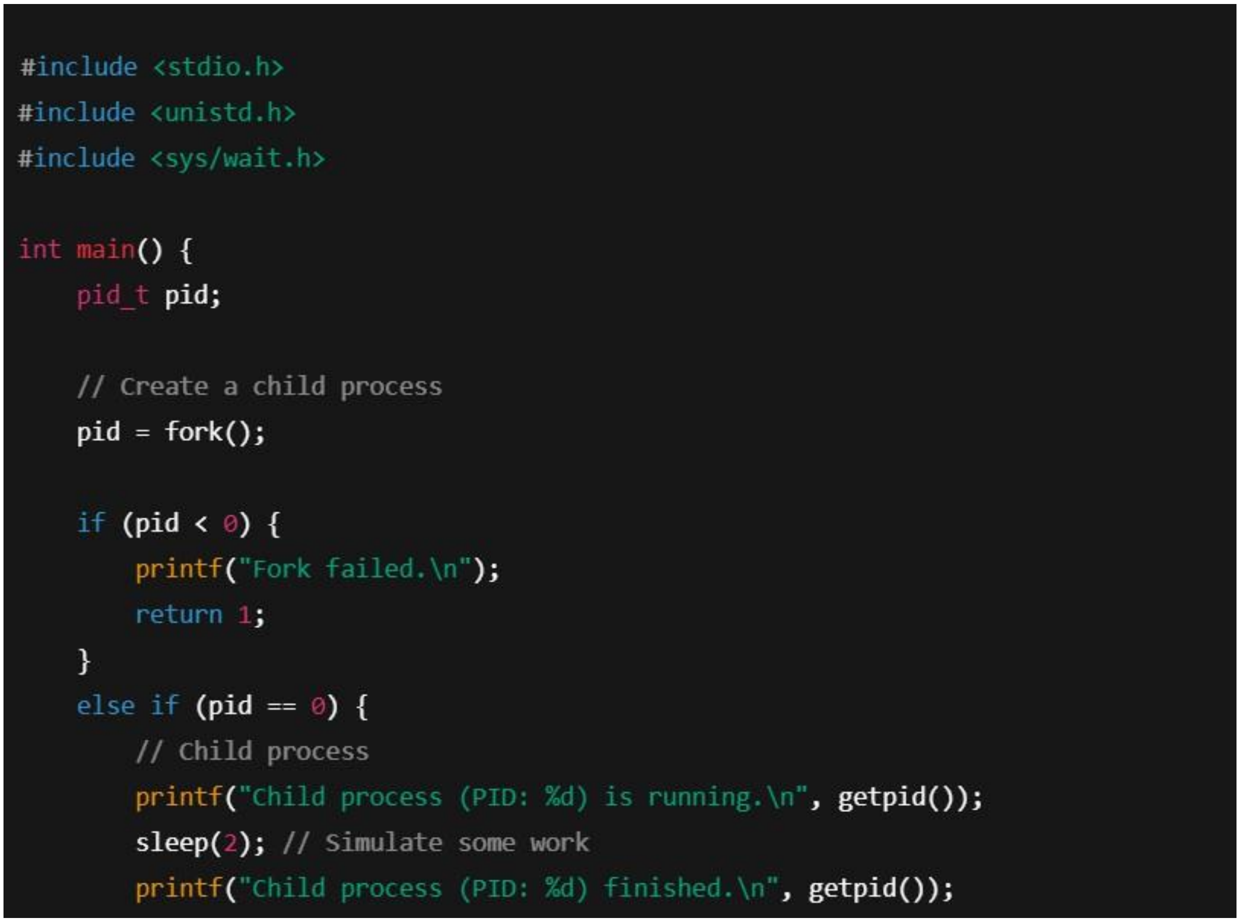


```
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
Name: Bithika Saha, Roll Number: 34230823009, Department: CSE(AI&ML)
```

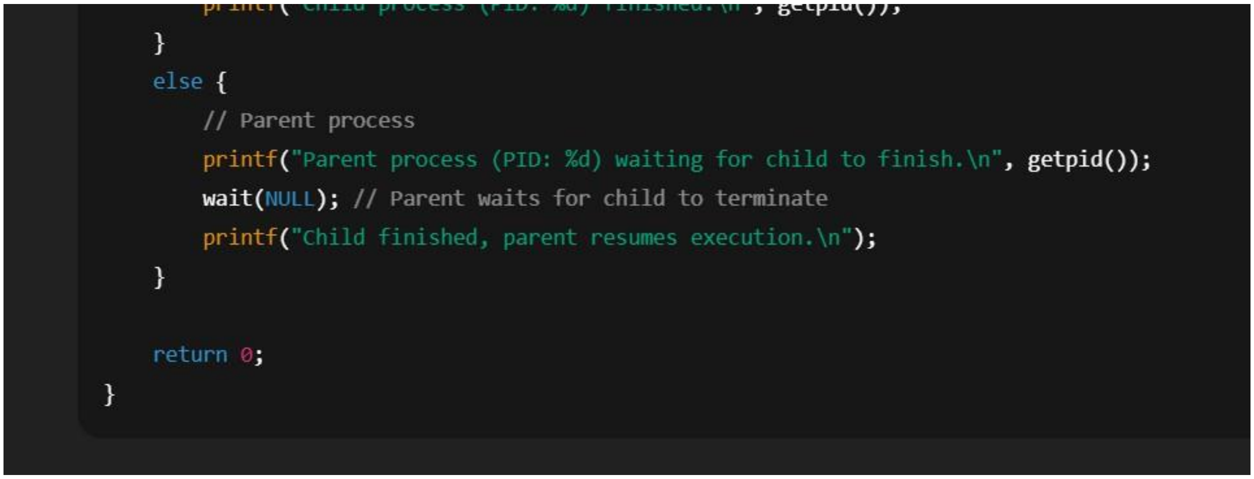
(iii) Write a program in C, which will illustrate wait() system call.

Ans:

Code:

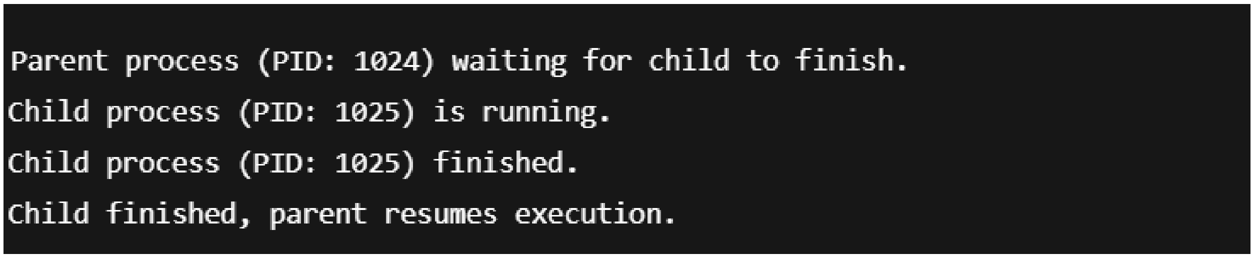


```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        printf("Fork failed.\n");
        return 1;
    }
    else if (pid == 0) {
        // Child process
        printf("Child process (PID: %d) is running.\n", getpid());
        sleep(2); // Simulate some work
        printf("Child process (PID: %d) finished.\n", getpid());
```



```
        printf("Child process (PID: %d) finished.\n", getpid());
    }
    else {
        // Parent process
        printf("Parent process (PID: %d) waiting for child to finish.\n", getpid());
        wait(NULL); // Parent waits for child to terminate
        printf("Child finished, parent resumes execution.\n");
    }

    return 0;
}
```

Output:

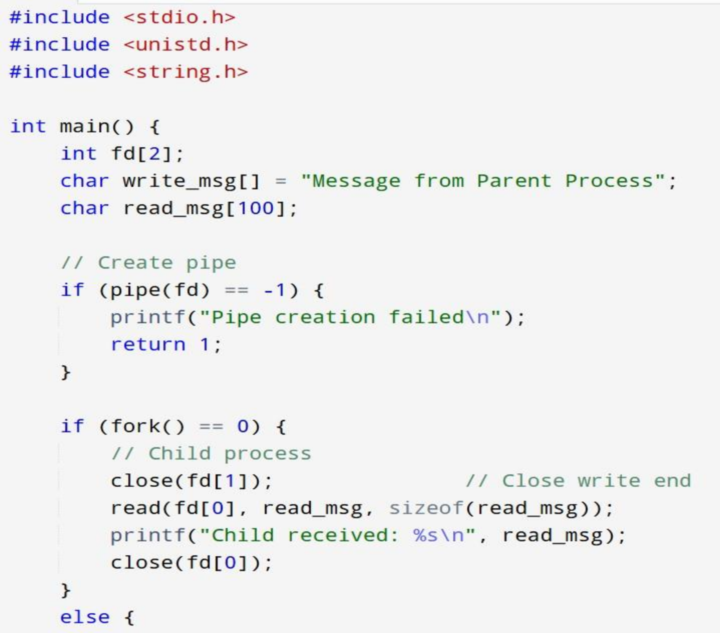


```
Parent process (PID: 1024) waiting for child to finish.
Child process (PID: 1025) is running.
Child process (PID: 1025) finished.
Child finished, parent resumes execution.
```

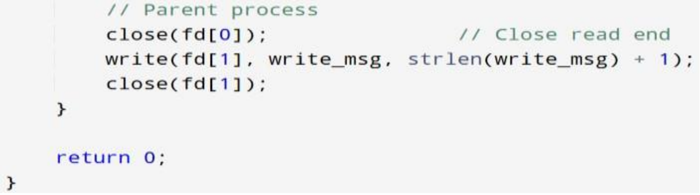
Assignment -6

1.

i. Write a program in c, which will illustrate the unidirectional inter process communication using pipe () system call.

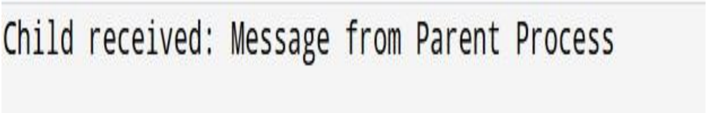


```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd[2];
    char write_msg[] = "Message from Parent Process";
    char read_msg[100];

    // Create pipe
    if (pipe(fd) == -1) {
        printf("Pipe creation failed\n");
        return 1;
    }

    if (fork() == 0) {
        // Child process
        close(fd[1]);               // Close write end
        read(fd[0], read_msg, sizeof(read_msg));
        printf("Child received: %s\n", read_msg);
        close(fd[0]);
    }
    else {
```



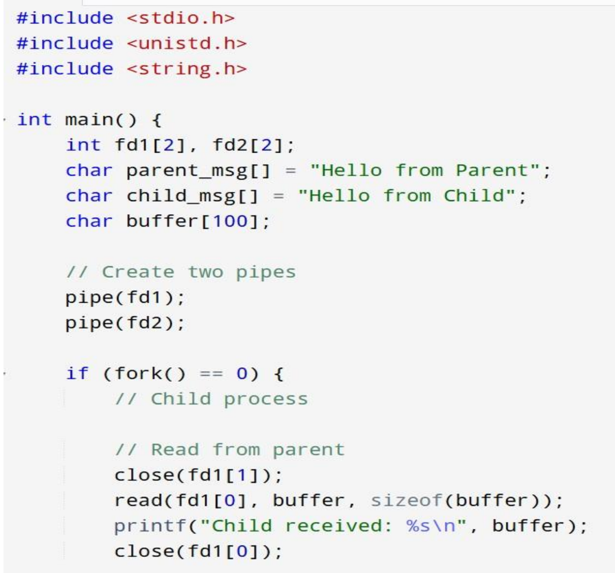
```c
        // Parent process
        close(fd[0]);               // Close read end
        write(fd[1], write_msg, strlen(write_msg) + 1);
        close(fd[1]);
    }

    return 0;
}
```

Output

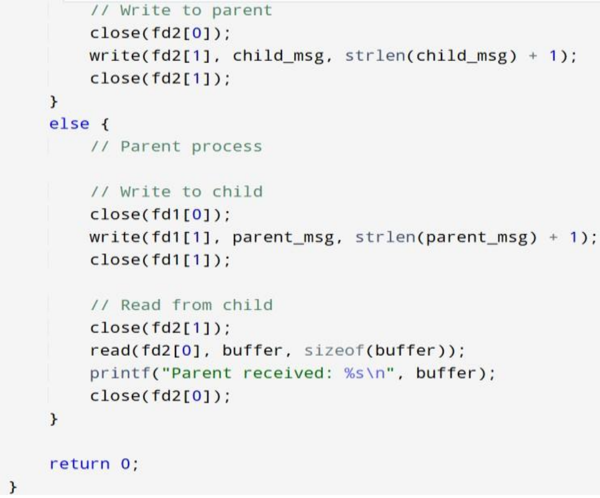


```
Child received: Message from Parent Process
```

ii. Write a program in c, which will illustrate the bidirectional inter process communication using pipe () system call.

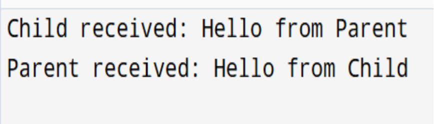


```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd1[2], fd2[2];
    char parent_msg[] = "Hello from Parent";
    char child_msg[] = "Hello from Child";
    char buffer[100];

    // Create two pipes
    pipe(fd1);
    pipe(fd2);

    if (fork() == 0) {
        // Child process

        // Read from parent
        close(fd1[1]);
        read(fd1[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(fd1[0]);
```



```c
        // Write to parent
        close(fd2[0]);
        write(fd2[1], child_msg, strlen(child_msg) + 1);
        close(fd2[1]);
    }
    else {
        // Parent process

        // Write to child
        close(fd1[0]);
        write(fd1[1], parent_msg, strlen(parent_msg) + 1);
        close(fd1[1]);

        // Read from child
        close(fd2[1]);
        read(fd2[0], buffer, sizeof(buffer));
        printf("Parent received: %s\n", buffer);
        close(fd2[0]);
    }

    return 0;
}
```

Output

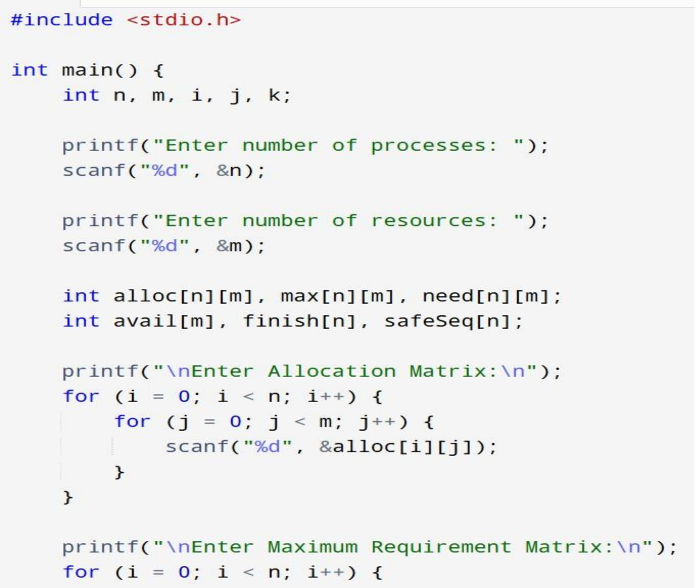


```
Child received: Hello from Parent
Parent received: Hello from Child
```

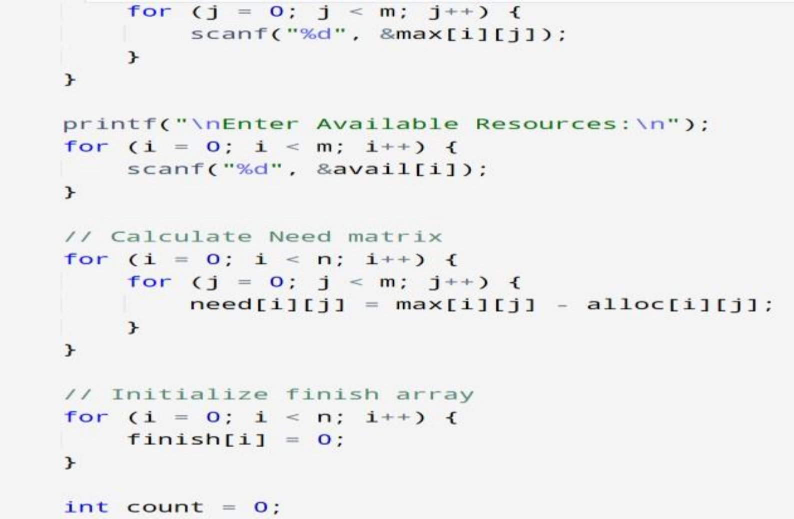
Assignment -7

1. Write a program, which will simulate banker's algorithm.

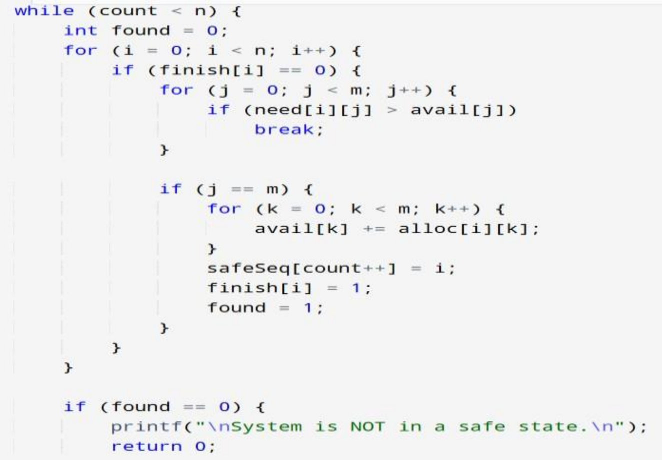
Show the safe sequence (If available).



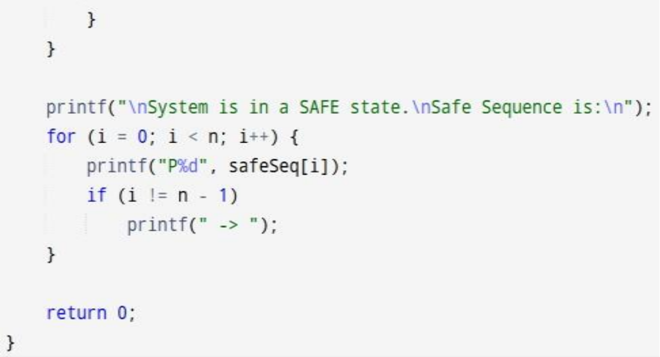
```
#include <stdio.h>

int main() {
    int n, m, i, j, k;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter number of resources: ");
    scanf("%d", &m);

    int alloc[n][m], max[n][m], need[n][m];
    int avail[m], finish[n], safeSeq[n];

    printf("\nEnter Allocation Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("\nEnter Maximum Requirement Matrix:\n");
    for (i = 0; i < n; i++) {
```



```
        for (j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("\nEnter Available Resources:\n");
    for (i = 0; i < m; i++) {
        scanf("%d", &avail[i]);
    }

    // Calculate Need matrix
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    // Initialize finish array
    for (i = 0; i < n; i++) {
        finish[i] = 0;
    }

    int count = 0;
```

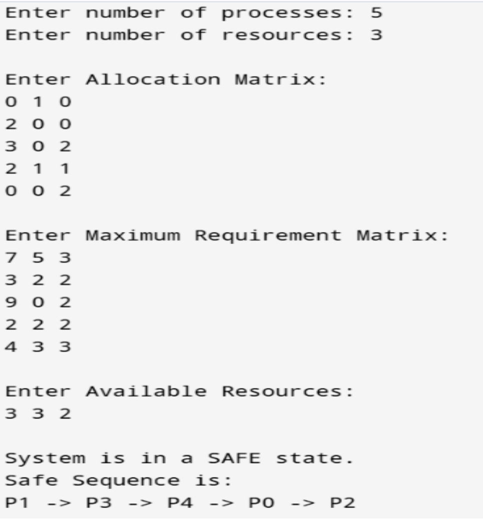


```
while (count < n) {
    int found = 0;
    for (i = 0; i < n; i++) {
        if (finish[i] == 0) {
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j])
                    break;
            }

            if (j == m) {
                for (k = 0; k < m; k++) {
                    avail[k] += alloc[i][k];
                }
                safeSeq[count++] = i;
                finish[i] = 1;
                found = 1;
            }
        }
    }

    if (found == 0) {
        printf("\nSystem is NOT in a safe state.\n");
        return 0;
```



```
        }
    }

    printf("\nSystem is in a SAFE state.\nSafe Sequence is:\n");
    for (i = 0; i < n; i++) {
        printf("P%d", safeSeq[i]);
        if (i != n - 1)
            printf(" -> ");
    }

    return 0;
}
```

Output



```
Enter number of processes: 5
Enter number of resources: 3

Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter Maximum Requirement Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter Available Resources:
3 3 2

System is in a SAFE state.
Safe Sequence is:
P1 -> P3 -> P4 -> P0 -> P2
```

Assignment- 8

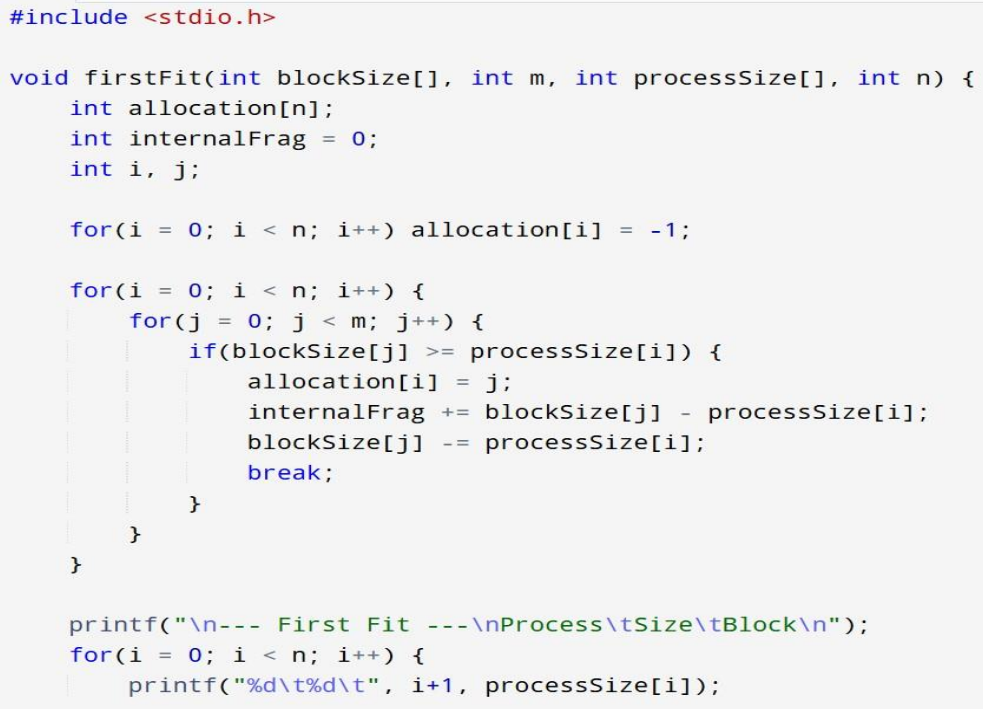
1. Write a program, which will illustrate different memory allocation techniques.

i. First Fit

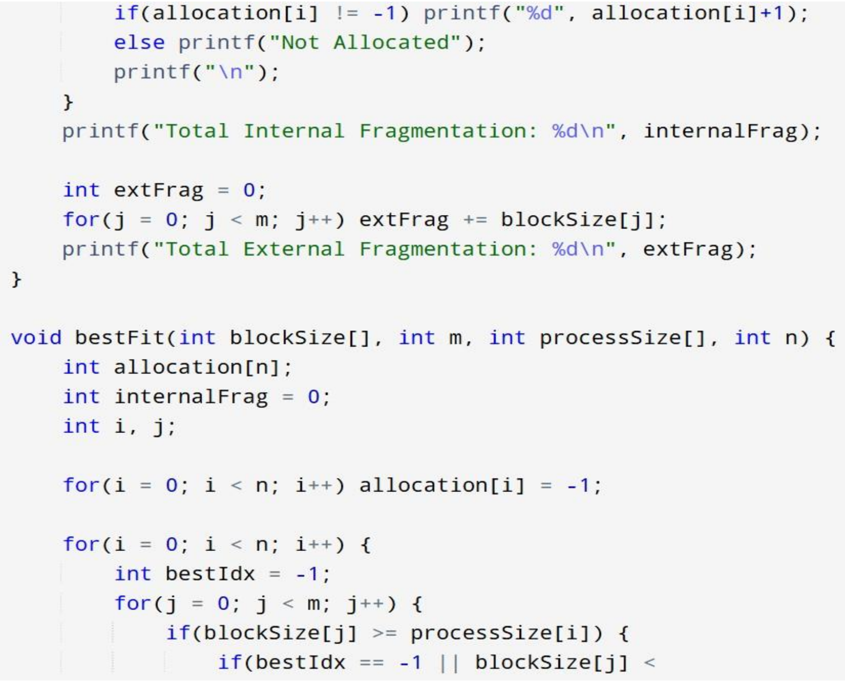
ii. Best Fit

iii. Worst Fit

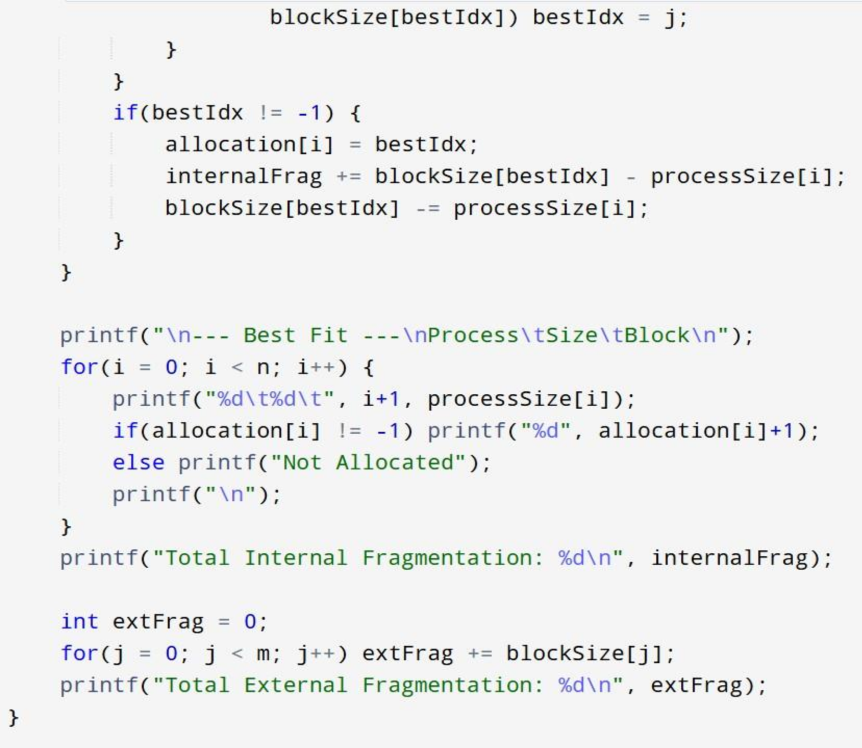
Show the total internal fragmentation and total external fragmentation in each case.



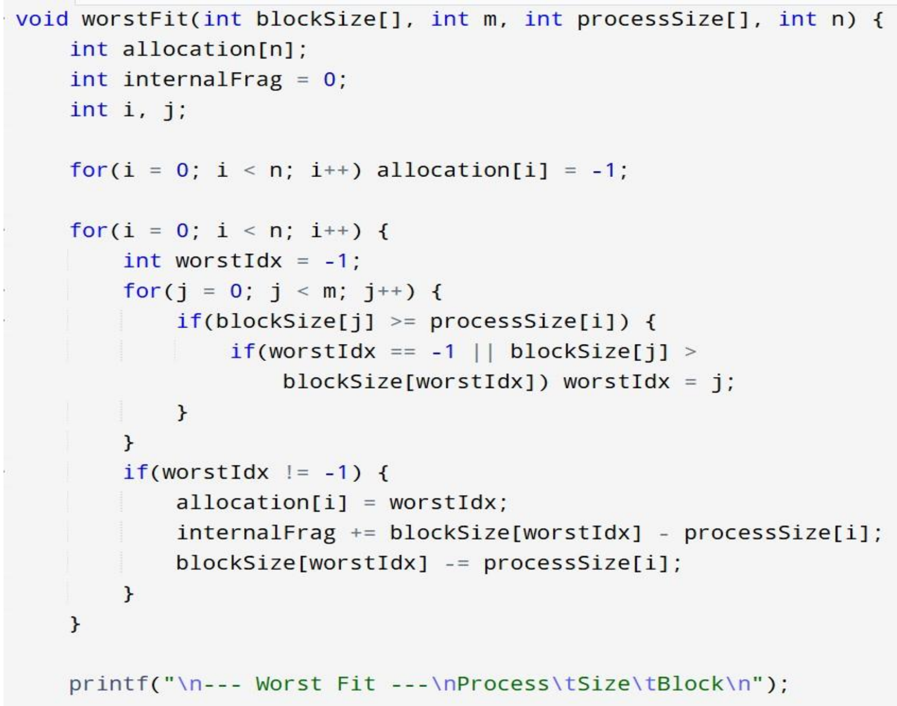
```
#include <stdio.h>

void firstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    int internalFrag = 0;
    int i, j;

    for(i = 0; i < n; i++) allocation[i] = -1;

    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            if(blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                internalFrag += blockSize[j] - processSize[i];
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }

    printf("\n--- First Fit ---\nProcess\tSize\tBlock\n");
    for(i = 0; i < n; i++) {
        printf("%d\t%d\t", i+1, processSize[i]);
```



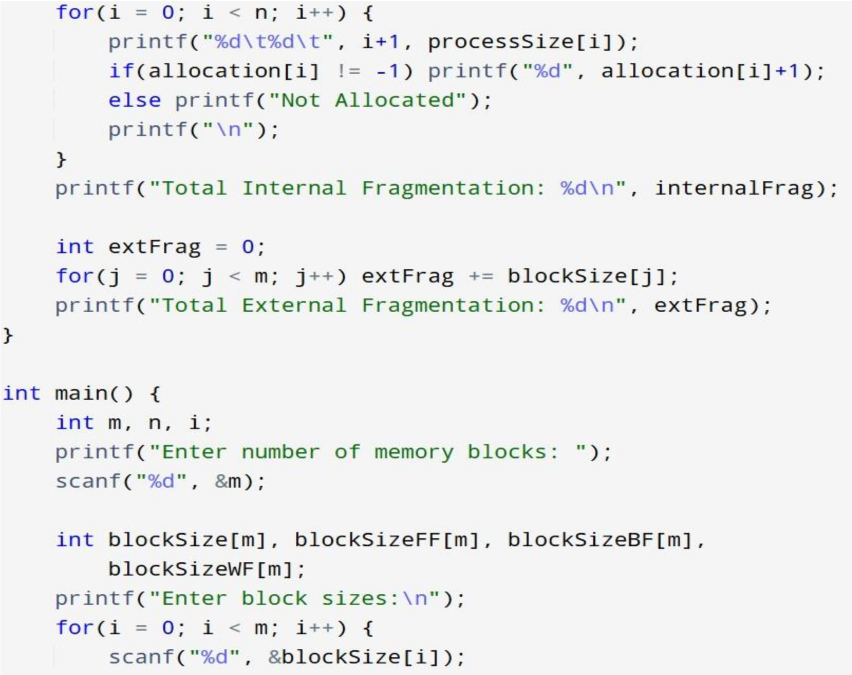
```
        if(allocation[i] != -1) printf("%d", allocation[i]+1);
        else printf("Not Allocated");
        printf("\n");
    }
    printf("Total Internal Fragmentation: %d\n", internalFrag);

    int extFrag = 0;
    for(j = 0; j < m; j++) extFrag += blockSize[j];
    printf("Total External Fragmentation: %d\n", extFrag);
}

void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    int internalFrag = 0;
    int i, j;

    for(i = 0; i < n; i++) allocation[i] = -1;

    for(i = 0; i < n; i++) {
        int bestIdx = -1;
        for(j = 0; j < m; j++) {
            if(blockSize[j] >= processSize[i]) {
                if(bestIdx == -1 || blockSize[j] <
```



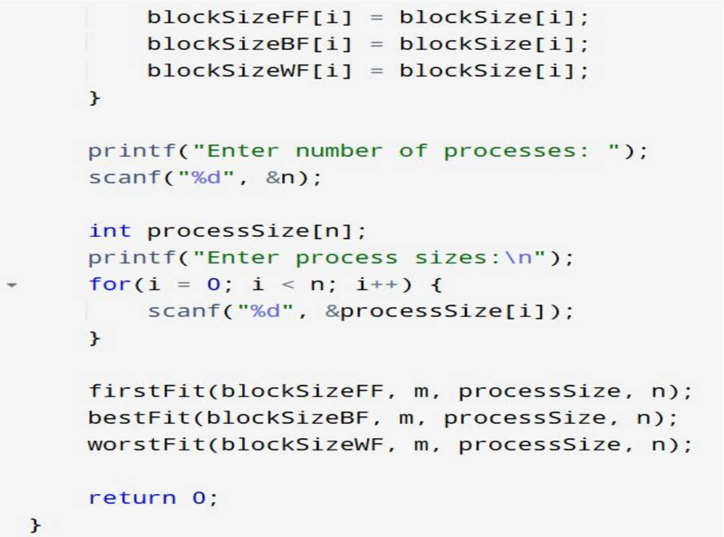
```
                    blockSize[bestIdx]) bestIdx = j;
            }
        }
        if(bestIdx != -1) {
            allocation[i] = bestIdx;
            internalFrag += blockSize[bestIdx] - processSize[i];
            blockSize[bestIdx] -= processSize[i];
        }
    }

    printf("\n--- Best Fit ---\nProcess\tSize\tBlock\n");
    for(i = 0; i < n; i++) {
        printf("%d\t%d\t", i+1, processSize[i]);
        if(allocation[i] != -1) printf("%d", allocation[i]+1);
        else printf("Not Allocated");
        printf("\n");
    }
    printf("Total Internal Fragmentation: %d\n", internalFrag);

    int extFrag = 0;
    for(j = 0; j < m; j++) extFrag += blockSize[j];
    printf("Total External Fragmentation: %d\n", extFrag);
}
```



```
void worstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    int internalFrag = 0;
    int i, j;

    for(i = 0; i < n; i++) allocation[i] = -1;

    for(i = 0; i < n; i++) {
        int worstIdx = -1;
        for(j = 0; j < m; j++) {
            if(blockSize[j] >= processSize[i]) {
                if(worstIdx == -1 || blockSize[j] >
                    blockSize[worstIdx]) worstIdx = j;
            }
        }
        if(worstIdx != -1) {
            allocation[i] = worstIdx;
            internalFrag += blockSize[worstIdx] - processSize[i];
            blockSize[worstIdx] -= processSize[i];
        }
    }

    printf("\n--- Worst Fit ---\nProcess\tSize\tBlock\n");
```

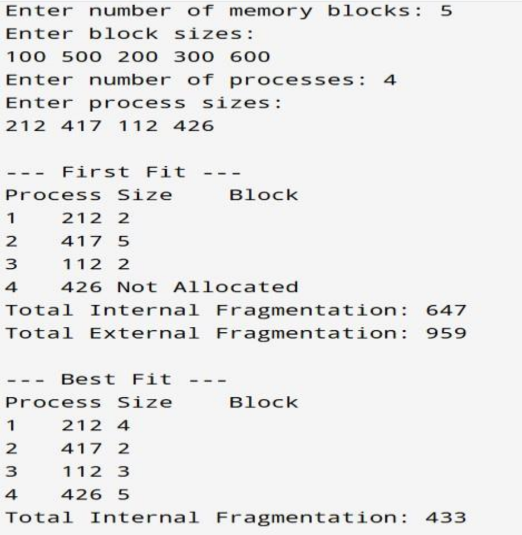


```
    for(i = 0; i < n; i++) {
        printf("%d\t%d\t", i+1, processSize[i]);
        if(allocation[i] != -1) printf("%d", allocation[i]+1);
        else printf("Not Allocated");
        printf("\n");
    }
    printf("Total Internal Fragmentation: %d\n", internalFrag);

    int extFrag = 0;
    for(j = 0; j < m; j++) extFrag += blockSize[j];
    printf("Total External Fragmentation: %d\n", extFrag);
}

int main() {
    int m, n, i;
    printf("Enter number of memory blocks: ");
    scanf("%d", &m);

    int blockSize[m], blockSizeFF[m], blockSizeBF[m],
        blockSizeWF[m];
    printf("Enter block sizes:\n");
    for(i = 0; i < m; i++) {
        scanf("%d", &blockSize[i]);
```

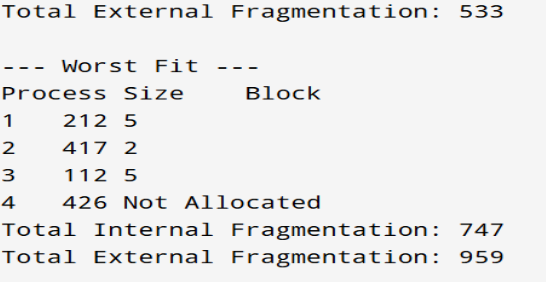


```
        blockSizeFF[i] = blockSize[i];
        blockSizeBF[i] = blockSize[i];
        blockSizeWF[i] = blockSize[i];
    }

    printf("Enter number of processes: ");
    scanf("%d", &n);

    int processSize[n];
    printf("Enter process sizes:\n");
    for(i = 0; i < n; i++) {
        scanf("%d", &processSize[i]);
    }

    firstFit(blockSizeFF, m, processSize, n);
    bestFit(blockSizeBF, m, processSize, n);
    worstFit(blockSizeWF, m, processSize, n);

    return 0;
}
```

Output



```
Enter number of memory blocks: 5
Enter block sizes:
100 500 200 300 600
Enter number of processes: 4
Enter process sizes:
212 417 112 426

--- First Fit ---
Process Size    Block
1    212 2
2    417 5
3    112 2
4    426 Not Allocated
Total Internal Fragmentation: 647
Total External Fragmentation: 959

--- Best Fit ---
Process Size    Block
1    212 4
2    417 2
3    112 3
4    426 5
Total Internal Fragmentation: 433
```



```
Total External Fragmentation: 533

--- Worst Fit ---
Process Size    Block
1    212 5
2    417 2
3    112 5
4    426 Not Allocated
Total Internal Fragmentation: 747
Total External Fragmentation: 959
```