OPERATING SYSTEM LAB MANUAL

Kamalesh S P

II-CSE-‘B’

2116230701138

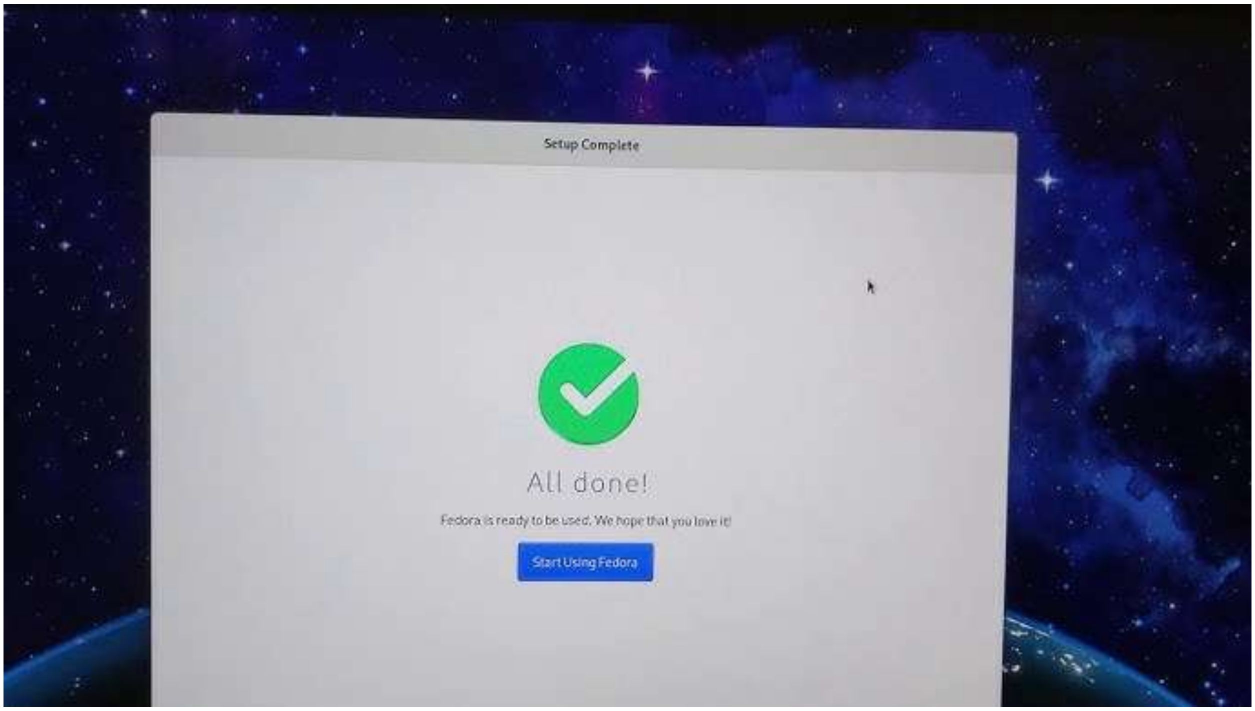
CS23431-Operating System

2024-2025

1a)

Aim: To install and configure Linux operating system in a Virtual Machine.

Output:



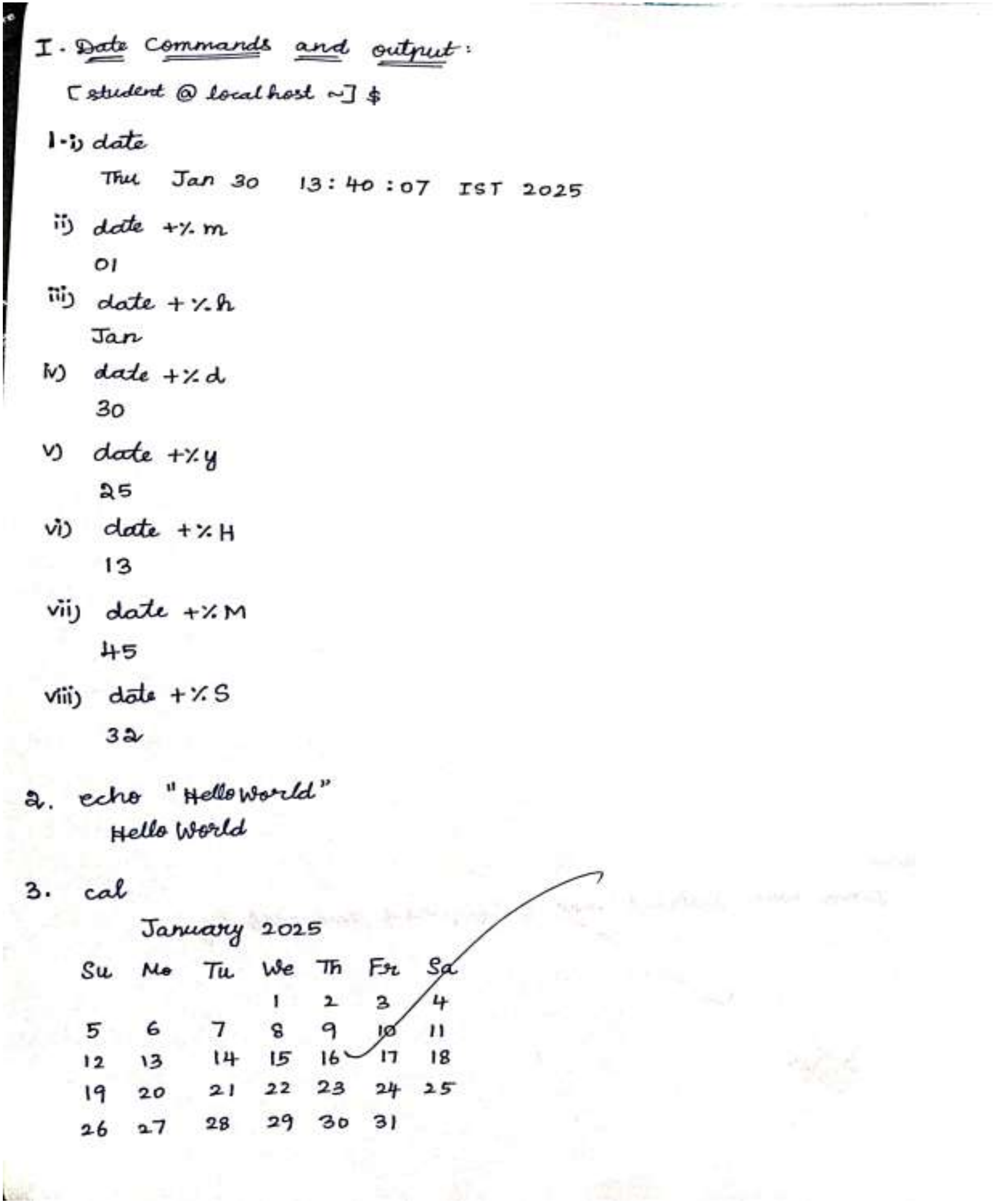
All done!

Fedora is ready to be used. We hope that you love it!

1b)

Aim: To run all the basic commands in fedora OS

Output:



I. Date Commands and output :

```
[student @ localhost ~] $

1.i) date
       Thu   Jan 30    13:40:07   IST 2025
ii)  date +%m
       01
iii) date +%h
       Jan
iv)  date +%d
       30
v)   date +%y
       25
vi)  date +%H
       13
vii) date +%M
       45
viii) date +%S
       32

2.  echo "HelloWorld"
       Hello World

3.  cal
          January 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
```

4. bc

```
16/8
2
```

5. who

```
Student  pts/0    2025-01-30   13:39 (:0)
Student  pts/1    2025-01-30   13:40 (:0)
```

6. who am i

```
student  pts/1   2025-01-30   13:40 (:0)
```

7. id

```
uid = 1000 (student)  gid = 1000 (student)  groups = 1000 (student)
context = unconfined_u : unconfined_r : unonfined_t : S0-S0 : C0.
C1023
```

8. tty

```
/dev/pts/1
```

9. clear

[screen gets cleared]

10. man who

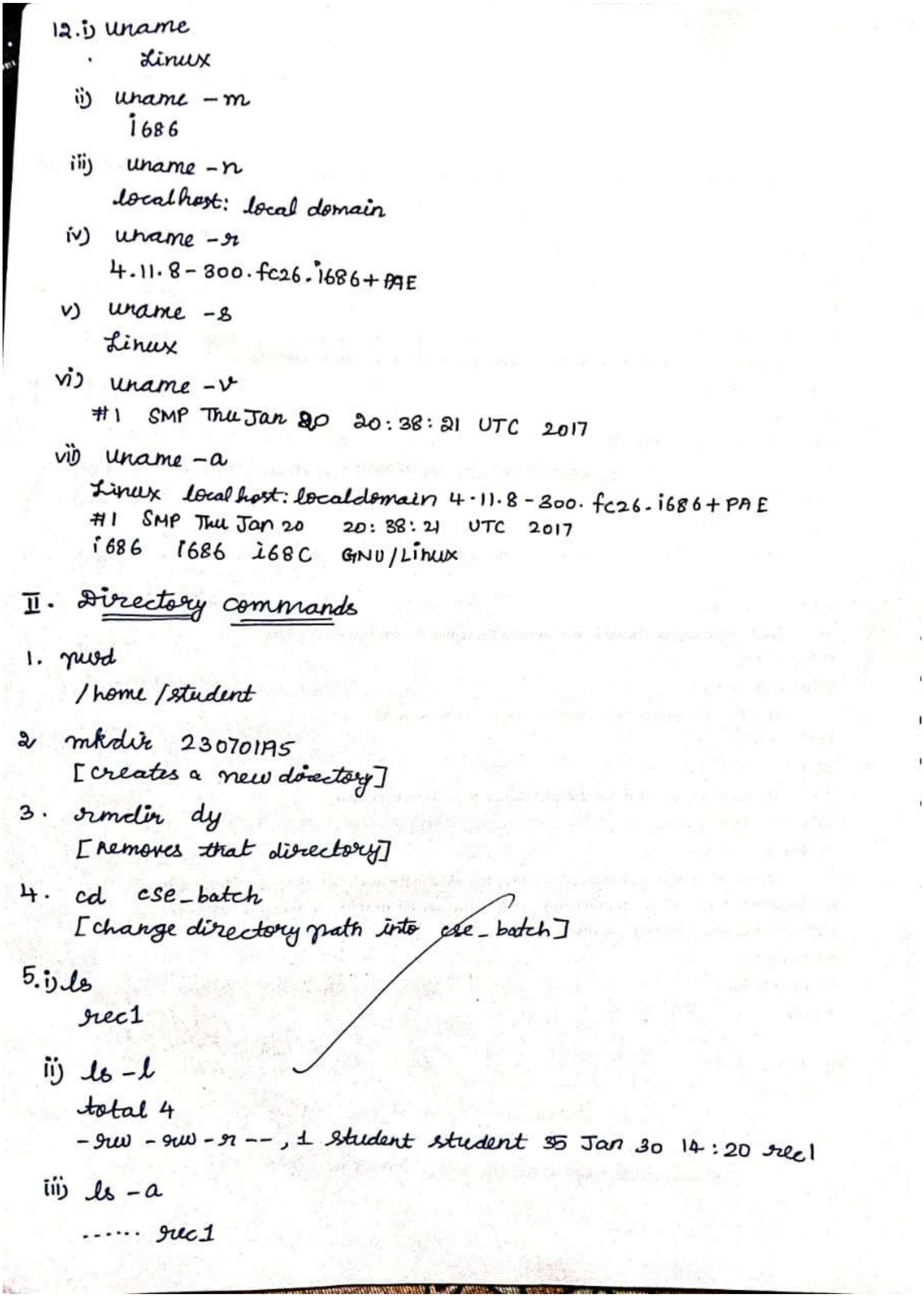
```
Name
     who - show the login details of the system
Synopsis
     who [option] .... [FILE|ARG1 ARG2] ....
```

11. i) ps

```
PID    TTY     TIME       CMD
1602   pts/1   00:00:00   bash
1699   pts/1   00:00:00   ps
```

ii) PS -aux

[shows the info about the running processes]



12.i) uname

```
Linux
```

ii) uname -m

```
i686
```

iii) uname -n

```
localhost: local domain
```

iv) uname -r

```
4.11.8-300.fc26.i686+PAE
```

v) urame -s

```
Linux
```

vi) uname -v

```
#1 SMP Thu Jan 20 20:38:21 UTC 2017
```

vii) uname -a

```
Linux localhost: localdomain 4-11.8-300. fc26-i686+PAE
#1 SMP Thu Jan 20  20: 38: 21  UTC  2017
i686  i686  i68C  GNU/Linux
```

II. Directory commands

1. pwd

```
/home/student
```

2 mkdir 230701A5

[creates a new directory]

3. rmdir dy

[Removes that directory]

4. cd cse_batch

[change directory path into cse_batch]

5.i) ls

```
rec1
```

ii) ls -l

```
total 4
-rw-rw-r--, 1 student student 35 Jan 30 14:20 rec1
```

iii) ls -a

```
...... rec1
```

III . File Handling Commands

1. cat > ex_2.1   [Creates a new file]

```
"This is my file containing Ex.2.1"
Basic file handing function
```

2. Cat ex_2.1   [Views a file]

```
"This is my file containing Ex.2.1"
Basic file handling function
```

3. cp   [Copy command]

```
cp ex_2.1 sample_ex
```

4. rm   [Removes the file]

```
rm sample_ex
```

5. mv   [Moves the file, deleting the old file]

```
mv ex-2.1 sample_ex
```

6. i) file   [Used todetermine type of file]

```
file ex_2.1
ex_2.1 : ASCII text
```

ii) file -i scary.jpg   [Display MIME type]

```
image.jpg: image/jpeg
```

iii) file -b scary.jpg

```
JPEG image data, 640x480, 8-bit/color RGB.
```

7. wc [counts the number of words]

```
wc ex-2.1
2   5   32   ex-2.1
```

8. ls > sample [Directing output to a file]

```
Cat sample
rec 1
rec 2
ex-2.1
```

9. who | wc [output of one is the input of other]

```
2   10   88
```

10. who | tee sample | wc [tee stores the content in file]

```
2   10   88
```

11. Metacharacters

i) ls r*

```
rec1  rec2
```

ii) ls [a-m]*

```
ex-2.1
```

iii) ls r?

```
rec1  rec 2
```

iv) ls [! a-m]*

```
rec1  rec 2  sample
```

v) ls -l [a-z]*{1,2}.?

```
rec1.txt
rec2.txt
```

12. File Permissions

```
ls -l

-rw -r--r-- 1 cse195 32 Feb 1 13:40 sample
-rw -r--r-- 1 cse195 32 Feb 1 13:41 rec123
```

13. Permission granting:

chmod

i) `chmod u +rw rec1`
[read-write is granted to users]

ii) `chmod u -rw rec1`
[read-write is disable to users]

iii) `chmod g -w rec1`
[write is disabled to group]

iv) `chmod g +r rec1`
[read is granted to group]

v) `chmod o - w rec1`
[write is disabled to othere]

vi) `chmod a + rw rec1`
[read write is granted to all]

vii) `chmod a +x rec1`
[execute is granted to all]

14. Octal Notations

i) chmod 555 rec1

5 - (4+1) ⇒ +rx = users
5 - (4+1) ⇒ +rx = Group
5 - (4+1) ⇒ +rx = others

ii) chmod 111

1 - (1) ⇒ +x = users
1 - (1) ⇒ +x = Group
1 - (1) ⇒ +x = others

IV. Grouping commands:

1) semicolon :

```
date +%S ; date +%h
51
Feb
```

2) date && ls (And)

```
Saturday 01 February 2025 02:01:07 PM IST
rec1 rec2
```

3) date || ls (or)

```
Saturday 01 February 2025 02:04 :28 PM IST
```

V. Filters command

1) head-5song_1

```
I was broken from a young age
Taking my sulking to the masses
Writing my poems for the few
That look at me, took at me, shook to me, feeling me
Singing from heartache, from the pain
```

2) tail -3 song_1

```
Seeing the beauty through the...
Pain!
You made me a, you made me a believer, believer
```

3) less -l | more

```
total 60
-rw-r--r--, 1  cse 195  cse 195  13208 Feb 1 14:16 poem
-rw-r--r--, 1  cse 195  cse 195  419   Feb 1 14:20 rec3
```

4) grep "believer" song_1

```
You made me a , you made me a believer , believer
```

5) i) sort song-1 (sorts the entire text file)

ii) sort -c student

```
sort: Student : 3: disorder: Balaji cse
```

iii) sort -r student

```
Ram cse
Kani cse
Arun cse
```

iv) sort -n student

```
Arun cse
Kani cse
Ram cse
```

v) sort -m student

```
Arun cse
kani cse
Ram cse
```

6) nl student

```
1  Arun cse
2  Kani cse
3  Ram cse
```

7) cut

i) cut -d ',' -f 4,5 song_1

```
That look. at me,
Singing from heartache,
```

ii) cut -c 1-5 song_1

```
I was
Takin
writi
That
Singi
Takin
Speak
Seein

Pain!
You m
```

VI. other Essential commands

1) free

```
          total      used      free      shared    buff/cache    available
Mem:     1993952    447208    899988
Swap:    2129916       0      2129916     65980      646736        1390932
```

2) top

```
top - 13:48:00  up 5 min , 2 users, load average: 0.16, 0.59, 0.36
Tasks : 159 total, 1 running, 158 sleeping, 0 stopped, 0 zombie
....

PID    USER    PR    NI    VIRT   RES   SHR  S  %CPU  %MEM  TIME+    COMMA
671    archi   20    0     33436  6228  3916 S  1.0   0.8   0:07.72  archi-
                                                                      dae
...
```

3) ps

```
PID     TTY      TIME       CMD

1547    pts/1    00:00:00   bash
1576    pts/1    00:00:00   ps
```

4) vmstat

```
procs  -----memory-------swap--  ---io---system--- cpu---
r  b  swpd  free  buff  cache  si  so  bi   bo   in   cs  us sy id wa st
d  0   0   900504 51212 596160  0   0  544  125  382  661  5  2 75 14  0
```

5) df

```
Filesystem   1K-blocks   Used  Available  Use%  Mounted on
devtmpfs     985980       0     985980     0%   /dev
tmpfs        996964       0     996964     0%   /dev/shm
....
```

6) ping www.google.com

```
PING   www.google.com (142.250.195.132) 56(84) bytes of data.
64 bytes from maa03s40-in-f4.1e100.net (142.250.195.132):
                              icmp_seq=1 ttl=59 time = 4.37 ms
....
```

7) ifconfig

```
enp3s0: flags=4163 <UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
    inet 172.16.9.34 netmask 255.255.252.0 broadcast
                                                   172.16.11.255.
 ....
```

8) trace route

Useage :

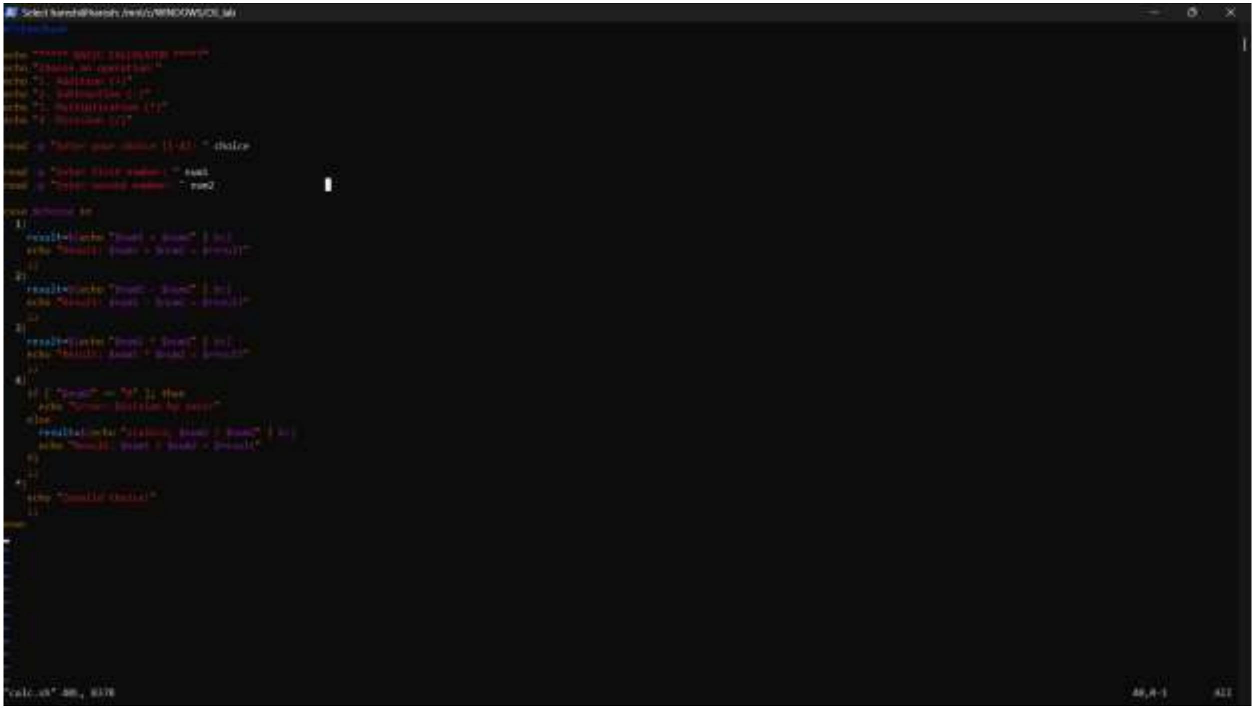
```
traceroute to www.google.com (142.250.195.132), 30 hops max,
                                         60 byte packets
1  gateway (172.16.8.1)  0.191 ms  0.134 ms  0.126 ms
2  * * *
```

3a)

i)

Aim: To write a shell script to build a basic calculator

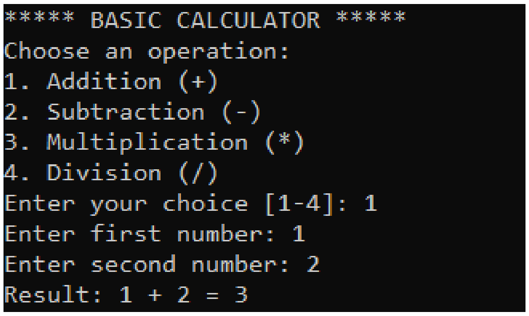
Program:



Select harish@harish: /mnt/c/WINDOWS/OS_lab

```
#!/bin/bash

echo "***** BASIC CALCULATOR *****"
echo "Choose an operation:"
echo "1. Addition (+)"
echo "2. Subtraction (-)"
echo "3. Multiplication (*)"
echo "4. Division (/)"

read -p "Enter your choice [1-4]: " choice

read -p "Enter first number: " num1
read -p "Enter second number: " num2

case $choice in
  1)
    result=$(echo "$num1 + $num2" | bc)
    echo "Result: $num1 + $num2 = $result"
    ;;
  2)
    result=$(echo "$num1 - $num2" | bc)
    echo "Result: $num1 - $num2 = $result"
    ;;
  3)
    result=$(echo "$num1 * $num2" | bc)
    echo "Result: $num1 * $num2 = $result"
    ;;
  4)
    if [ "$num2" == "0" ]; then
      echo "Error: Division by zero!"
    else
      result=$(echo "scale=2; $num1 / $num2" | bc)
      echo "Result: $num1 / $num2 = $result"
    fi
    ;;
  *)
    echo "Invalid choice!"
    ;;
esac
```

Output:

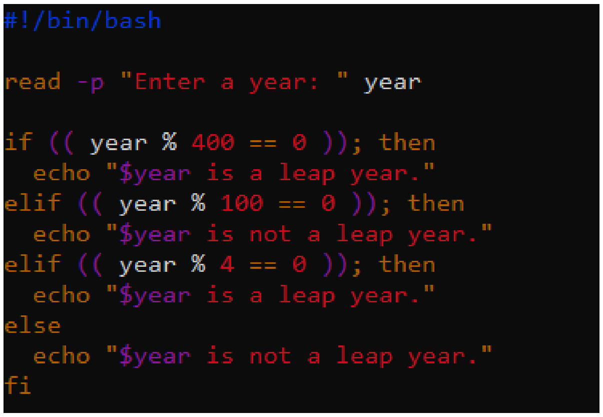


```
***** BASIC CALCULATOR *****
Choose an operation:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
Enter your choice [1-4]: 1
Enter first number: 1
Enter second number: 2
Result: 1 + 2 = 3
```

ii)

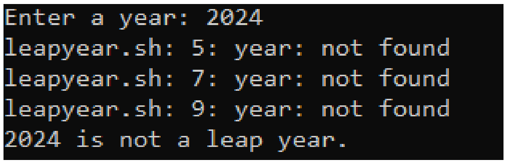
Aim: To write a shell script to test a given year is leap or not using conditional statement.

Program:



```
#!/bin/bash

read -p "Enter a year: " year

if (( year % 400 == 0 )); then
  echo "$year is a leap year."
elif (( year % 100 == 0 )); then
  echo "$year is not a leap year."
elif (( year % 4 == 0 )); then
  echo "$year is a leap year."
else
  echo "$year is not a leap year."
fi
```

Output:

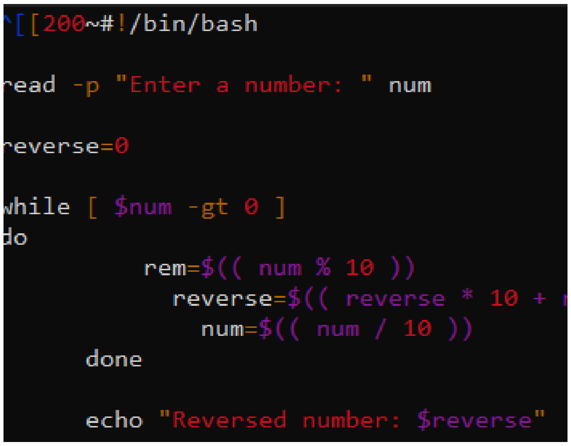


```
Enter a year: 2024
leapyear.sh: 5: year: not found
leapyear.sh: 7: year: not found
leapyear.sh: 9: year: not found
2024 is not a leap year.
```

3b)

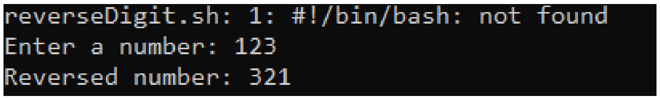
i) To write a shell script to reverse a digit

Program:



```
^[[200~#!/bin/bash

read -p "Enter a number: " num

reverse=0

while [ $num -gt 0 ]
do
          rem=$(( num % 10 ))
            reverse=$(( reverse * 10 +
              num=$(( num / 10 ))
      done

      echo "Reversed number: $reverse"
```

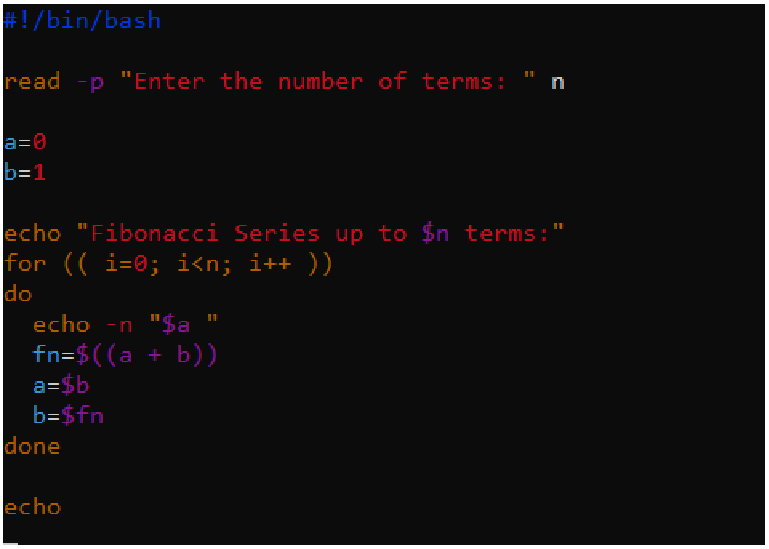
Output:



```
reverseDigit.sh: 1: #!/bin/bash: not found
Enter a number: 123
Reversed number: 321
```

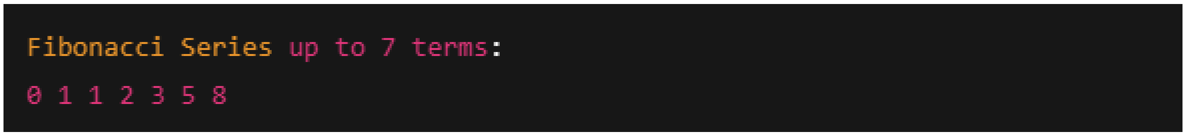
ii) To generate a Fibonacci series using a for loop

Program:



```
#!/bin/bash

read -p "Enter the number of terms: " n

a=0
b=1

echo "Fibonacci Series up to $n terms:"
for (( i=0; i<n; i++ ))
do
  echo -n "$a "
  fn=$((a + b))
  a=$b
  b=$fn
done

echo
```

Output:

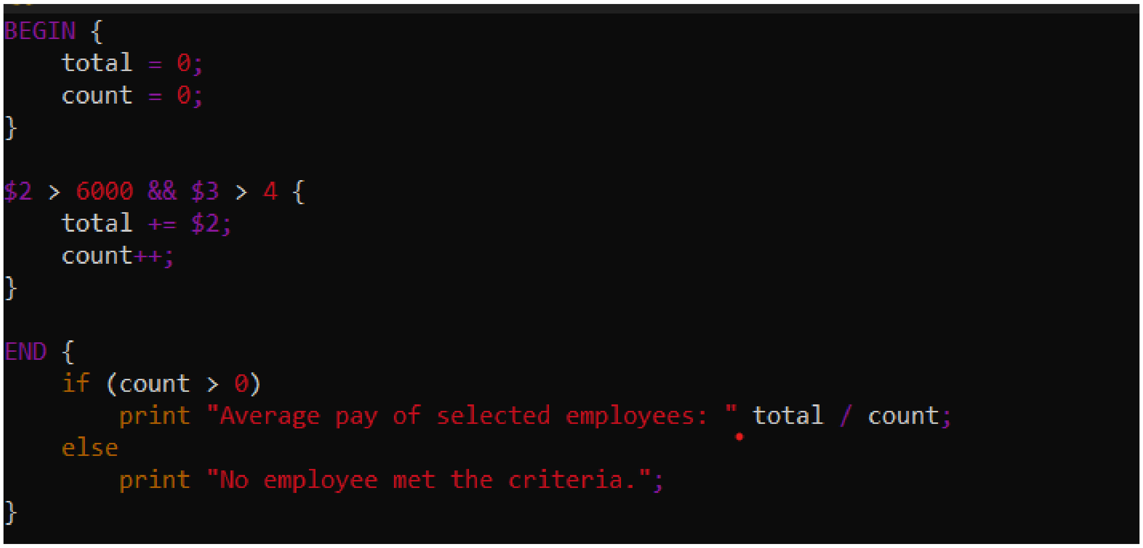


```
Fibonacci Series up to 7 terms:
0 1 1 2 3 5 8
```

4a)

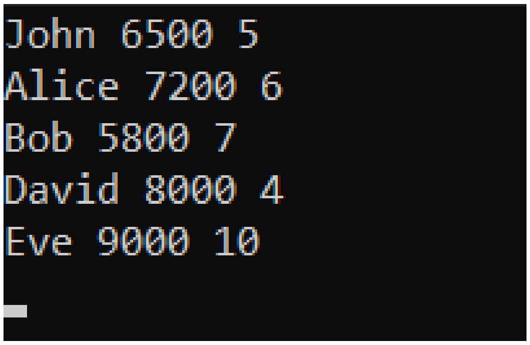
Aim: To find out the average pay of all employees whose salary is more than 6000 and no. of days worked is more than 4.

Program code:



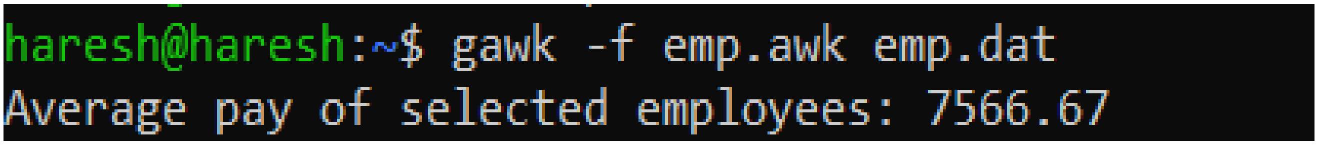
```
BEGIN {
    total = 0;
    count = 0;
}

$2 > 6000 && $3 > 4 {
    total += $2;
    count++;
}

END {
    if (count > 0)
        print "Average pay of selected employees: " total / count;
    else
        print "No employee met the criteria.";
}
```

Input:



```
John 6500 5
Alice 7200 6
Bob 5800 7
David 8000 4
Eve 9000 10
```

Output:

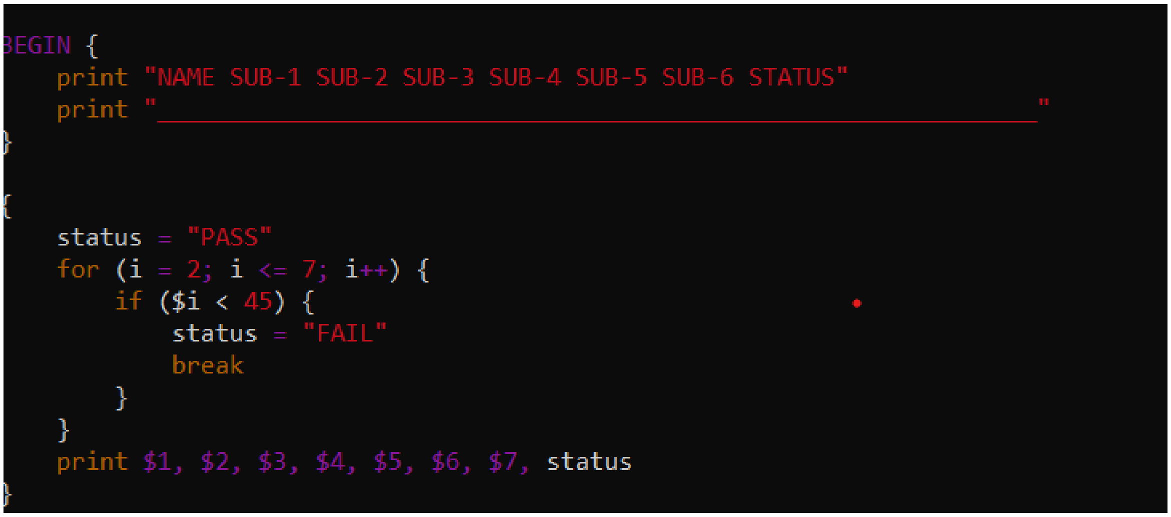


```
haresh@haresh:~$ gawk -f emp.awk emp.dat
Average pay of selected employees: 7566.67
```

4b)

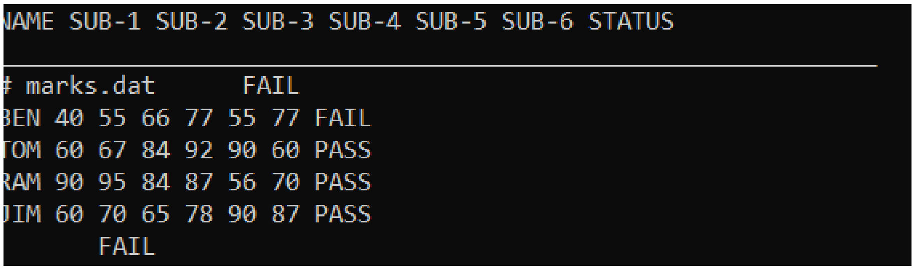
Aim: To print the pass/fail status of a student in a class.

Program code:



```
BEGIN {
    print "NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS"
    print "____________________________________________________________"
}

{
    status = "PASS"
    for (i = 2; i <= 7; i++) {
        if ($i < 45) {
            status = "FAIL"
            break
        }
    }
    print $1, $2, $3, $4, $5, $6, $7, status
}
```

Output

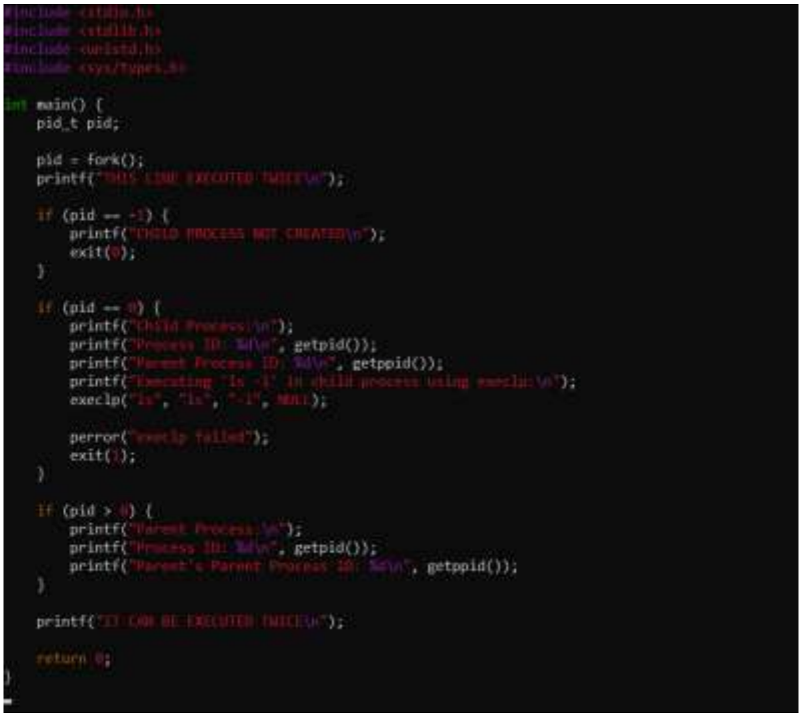


```
NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS
____________________________________________________________
# marks.dat      FAIL
BEN 40 55 66 77 55 77 FAIL
TOM 60 67 84 92 90 60 PASS
RAM 90 95 84 87 56 70 PASS
JIM 60 70 65 78 90 87 PASS
       FAIL
```

5)

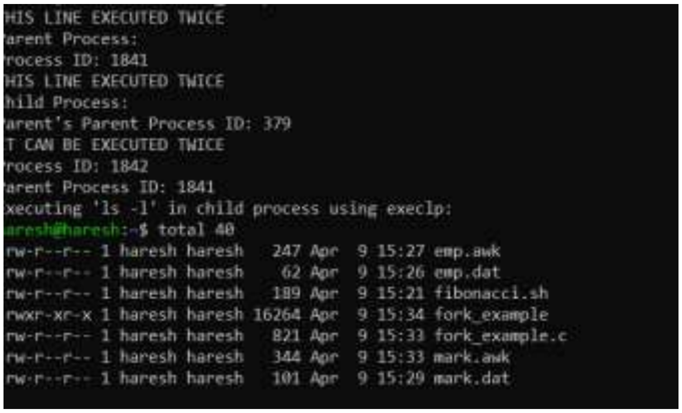
Aim: To experiment system calls using fork(), execlp() and pid() functions.

Program code:



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

int main() {
    pid_t pid;

    pid = fork();
    printf("THIS LINE EXECUTED TWICE\n");

    if (pid == -1) {
        printf("CHILD PROCESS NOT CREATED\n");
        exit(0);
    }

    if (pid == 0) {
        printf("Child Process:\n");
        printf("Process ID: %d\n", getpid());
        printf("Parent Process ID: %d\n", getppid());
        printf("Executing 'ls -l' in child process using execlp:\n");
        execlp("ls", "ls", "-l", NULL);

        perror("execlp failed");
        exit(1);
    }

    if (pid > 0) {
        printf("Parent Process:\n");
        printf("Process ID: %d\n", getpid());
        printf("Parent's Parent Process ID: %d\n", getppid());
    }

    printf("IT CAN BE EXECUTED TWICE\n");

    return 0;
}
```

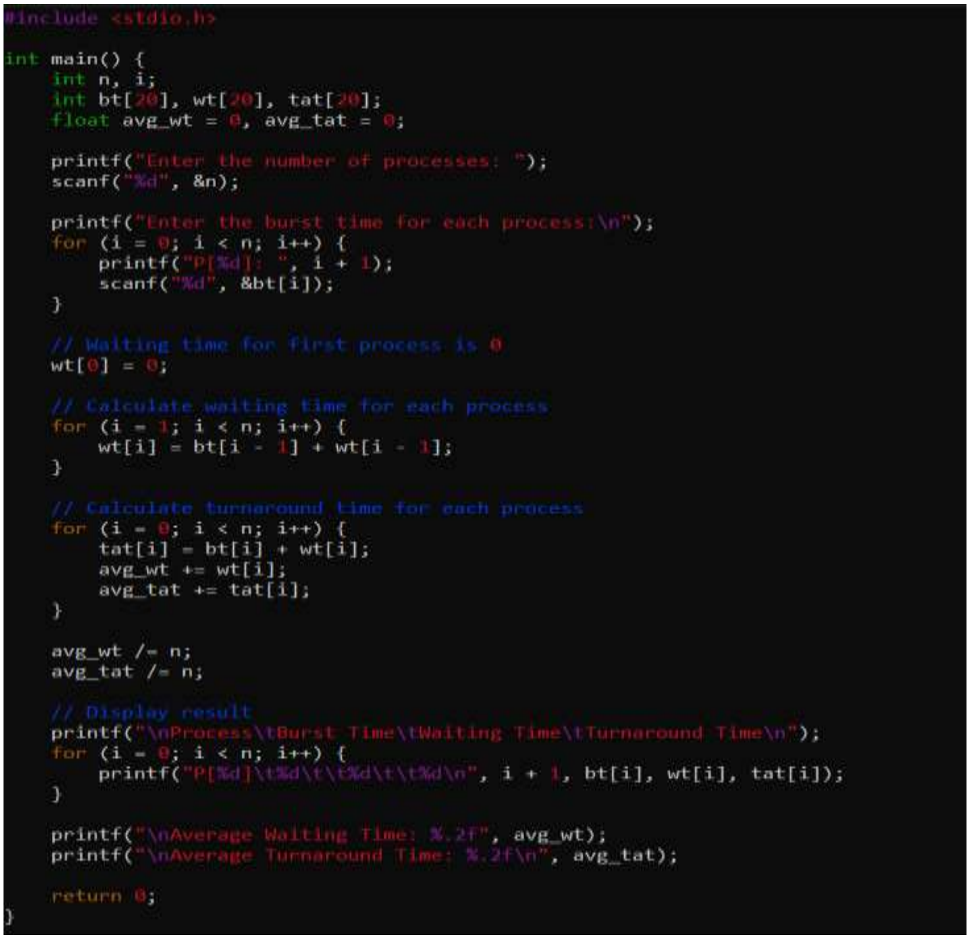
Output:



```
HIS LINE EXECUTED TWICE
arent Process:
rocess ID: 1841
HIS LINE EXECUTED TWICE
hild Process:
arent's Parent Process ID: 379
T CAN BE EXECUTED TWICE
rocess ID: 1842
arent Process ID: 1841
xecuting 'ls -l' in child process using execlp:
aresh@haresh:~$ total 40
rw-r--r-- 1 haresh haresh   247 Apr  9 15:27 emp.awk
rw-r--r-- 1 haresh haresh    62 Apr  9 15:26 emp.dat
rw-r--r-- 1 haresh haresh   189 Apr  9 15:21 fibonacci.sh
rwxr-xr-x 1 haresh haresh 16264 Apr  9 15:34 fork_example
rw-r--r-- 1 haresh haresh   821 Apr  9 15:33 fork_example.c
rw-r--r-- 1 haresh haresh   344 Apr  9 15:33 mark.awk
rw-r--r-- 1 haresh haresh   101 Apr  9 15:29 mark.dat
```

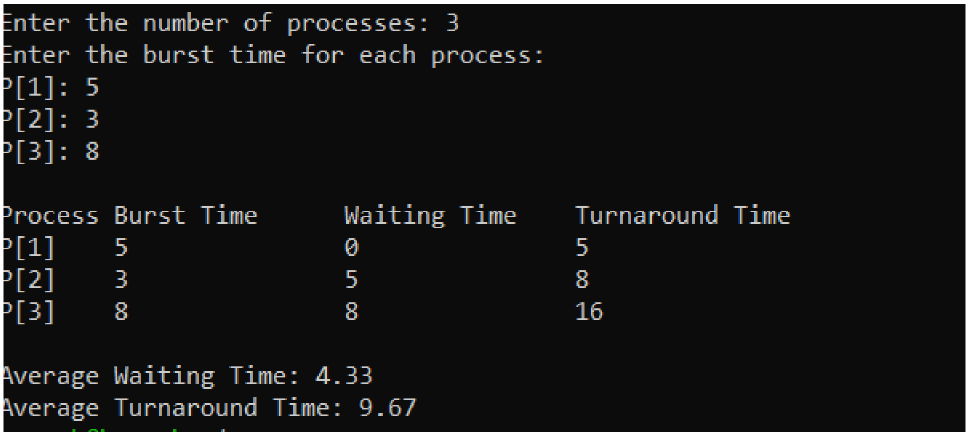
6a) FIRST COME FIRST SERVE (FCFS)

Program code:



```c
#include <stdio.h>

int main() {
    int n, i;
    int bt[20], wt[20], tat[20];
    float avg_wt = 0, avg_tat = 0;

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

    printf("Enter the burst time for each process:\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]: ", i + 1);
        scanf("%d", &bt[i]);
    }

    // Waiting time for first process is 0
    wt[0] = 0;

    // Calculate waiting time for each process
    for (i = 1; i < n; i++) {
        wt[i] = bt[i - 1] + wt[i - 1];
    }

    // Calculate turnaround time for each process
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= n;
    avg_tat /= n;

    // Display result
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

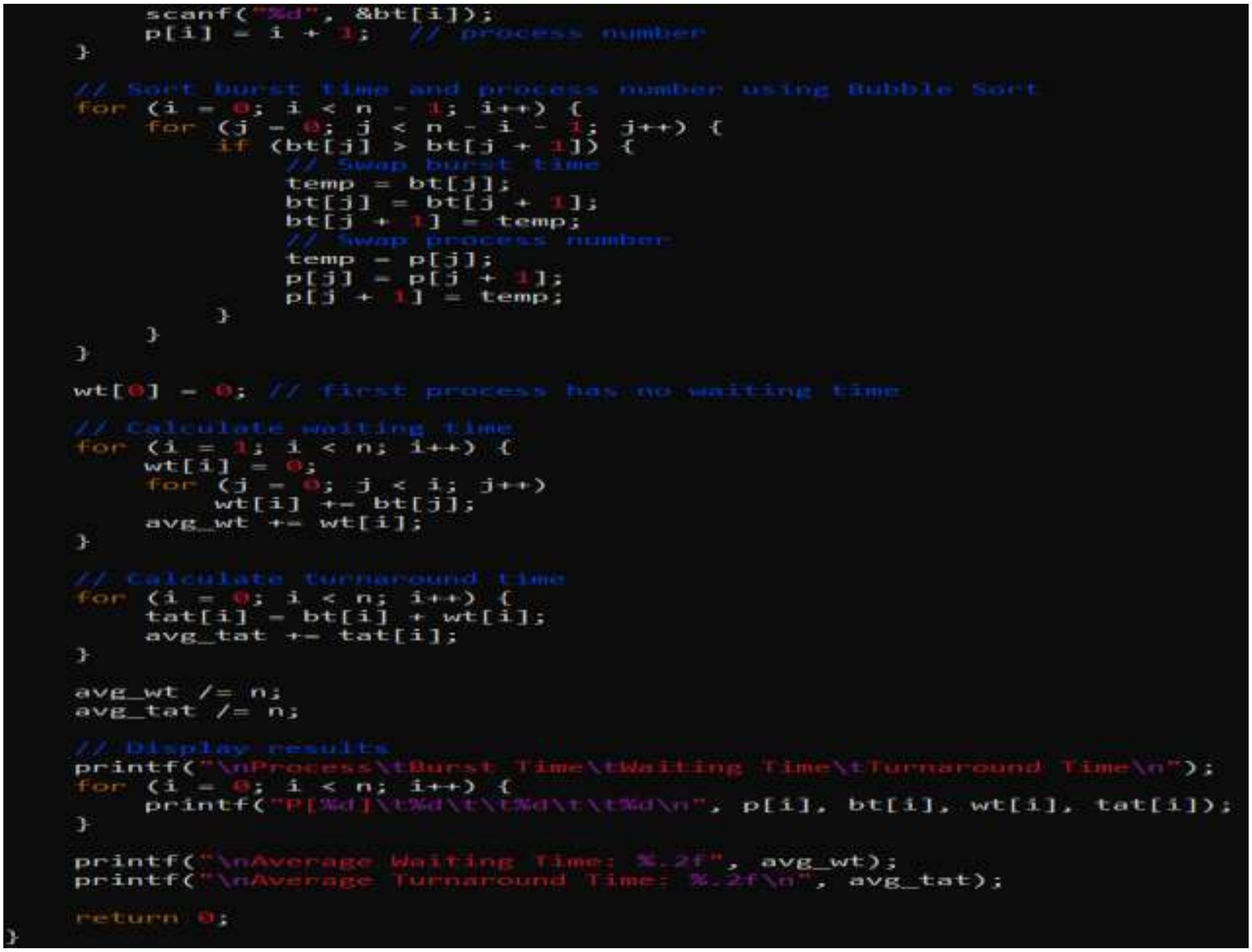
Output:



```
Enter the number of processes: 3
Enter the burst time for each process:
P[1]: 5
P[2]: 3
P[3]: 8

Process Burst Time      Waiting Time    Turnaround Time
P[1]    5               0               5
P[2]    3               5               8
P[3]    8               8               16

Average Waiting Time: 4.33
Average Turnaround Time: 9.67
```

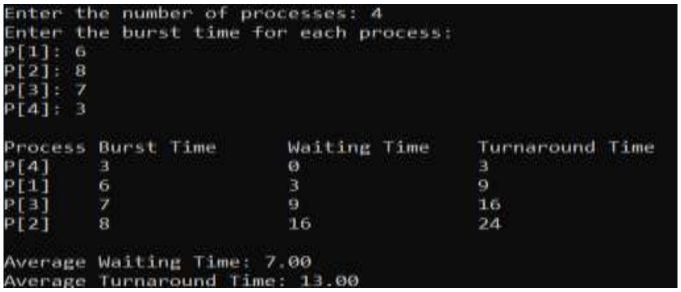
6b) Shortest Job First (SJF)

Program code:



```c
        scanf("%d", &bt[i]);
        p[i] = i + 1;  // process number
    }

    // Sort burst time and process number using Bubble Sort
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (bt[j] > bt[j + 1]) {
                // Swap burst time
                temp = bt[j];
                bt[j] = bt[j + 1];
                bt[j + 1] = temp;
                // Swap process number
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }

    wt[0] = 0; // first process has no waiting time

    // Calculate waiting time
    for (i = 1; i < n; i++) {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        avg_wt += wt[i];
    }

    // Calculate turnaround time
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= n;
    avg_tat /= n;

    // Display results
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

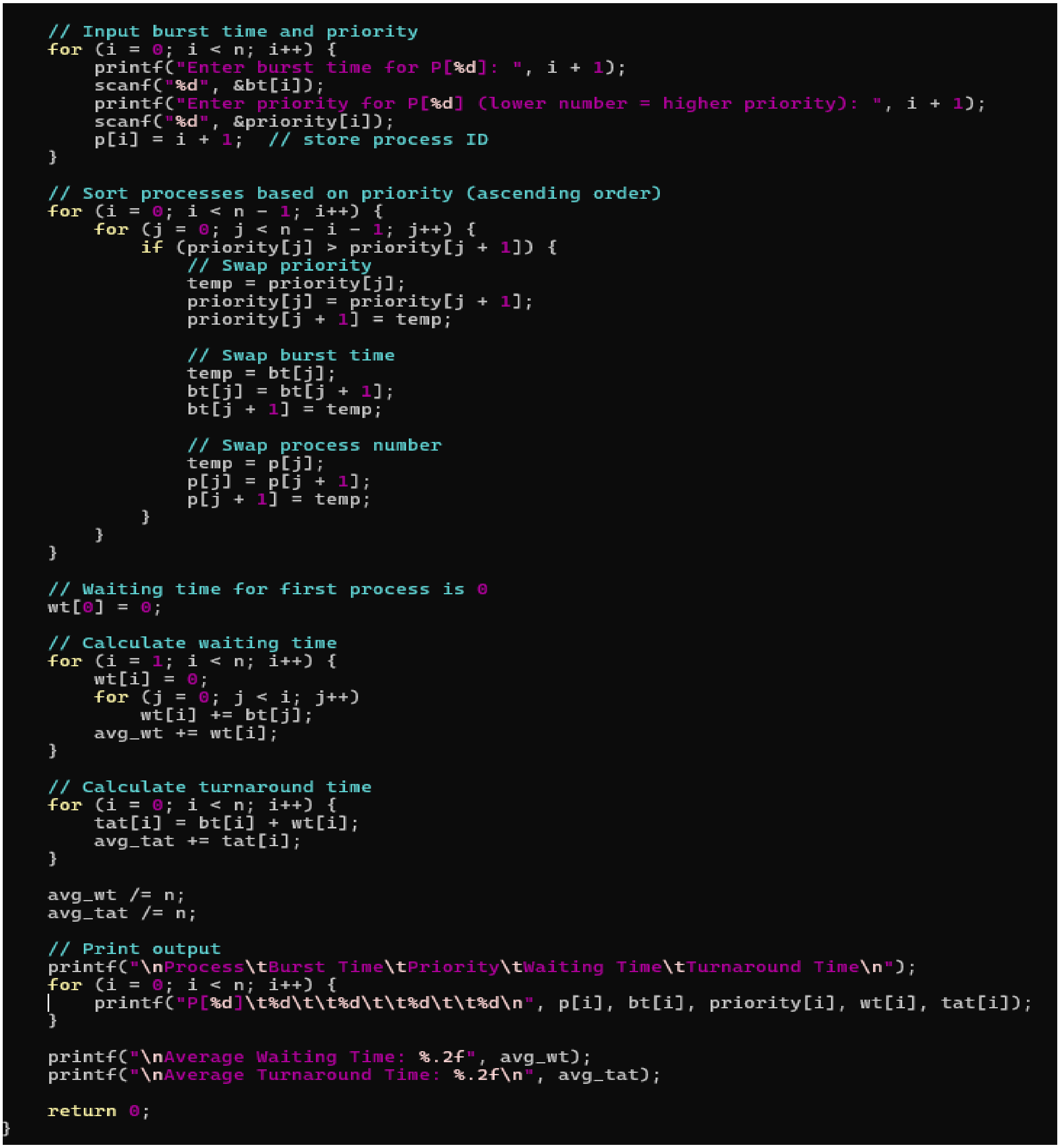
Output:



```
Enter the number of processes: 4
Enter the burst time for each process:
P[1]: 6
P[2]: 8
P[3]: 7
P[4]: 3

Process Burst Time      Waiting Time    Turnaround Time
P[4]    3               0               3
P[1]    6               3               9
P[3]    7               9               16
P[2]    8               16              24

Average Waiting Time: 7.00
Average Turnaround Time: 13.00
```

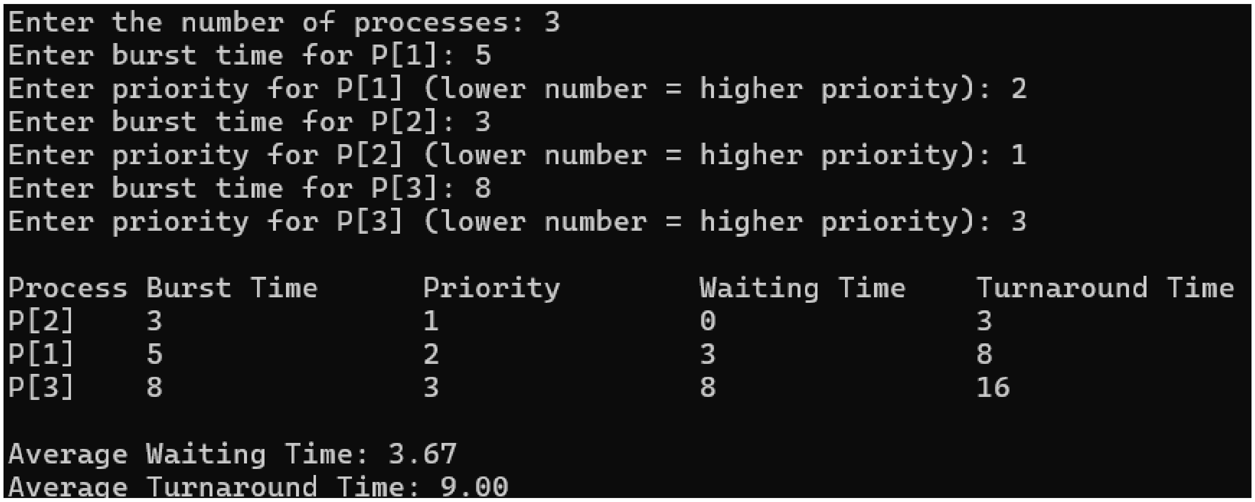
6c) PRIORITY SCHEDULING

Program Code:



```
    // Input burst time and priority
    for (i = 0; i < n; i++) {
        printf("Enter burst time for P[%d]: ", i + 1);
        scanf("%d", &bt[i]);
        printf("Enter priority for P[%d] (lower number = higher priority): ", i + 1);
        scanf("%d", &priority[i]);
        p[i] = i + 1;  // store process ID
    }

    // Sort processes based on priority (ascending order)
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (priority[j] > priority[j + 1]) {
                // Swap priority
                temp = priority[j];
                priority[j] = priority[j + 1];
                priority[j + 1] = temp;

                // Swap burst time
                temp = bt[j];
                bt[j] = bt[j + 1];
                bt[j + 1] = temp;

                // Swap process number
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }

    // Waiting time for first process is 0
    wt[0] = 0;

    // Calculate waiting time
    for (i = 1; i < n; i++) {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        avg_wt += wt[i];
    }

    // Calculate turnaround time
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= n;
    avg_tat /= n;

    // Print output
    printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]\t%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], priority[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

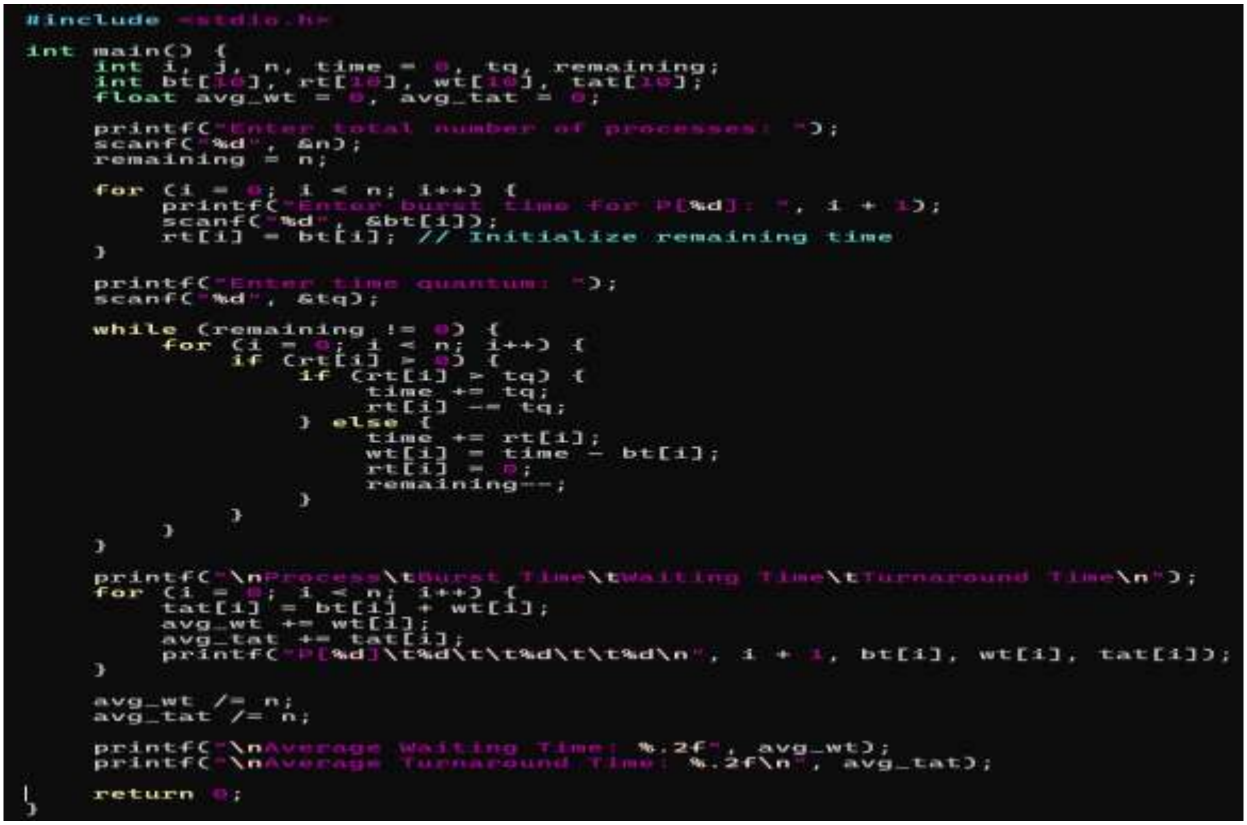
Output:



```
Enter the number of processes: 3
Enter burst time for P[1]: 5
Enter priority for P[1] (lower number = higher priority): 2
Enter burst time for P[2]: 3
Enter priority for P[2] (lower number = higher priority): 1
Enter burst time for P[3]: 8
Enter priority for P[3] (lower number = higher priority): 3

Process Burst Time      Priority        Waiting Time    Turnaround Time
P[2]    3               1               0               3
P[1]    5               2               3               8
P[3]    8               3               8               16

Average Waiting Time: 3.67
Average Turnaround Time: 9.00
```

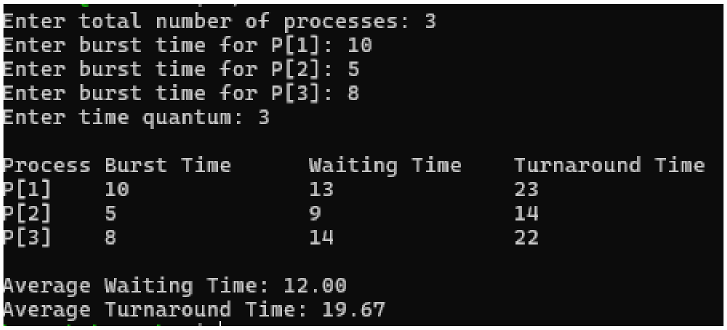
6d) ROUND ROBIN SCHEDULING (RR)

Program code:



```
#include <stdio.h>

int main() {
    int i, j, n, time = 0, tq, remaining;
    int bt[10], rt[10], wt[10], tat[10];
    float avg_wt = 0, avg_tat = 0;

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

    for (i = 0; i < n; i++) {
        printf("Enter burst time for P[%d]: ", i + 1);
        scanf("%d", &bt[i]);
        rt[i] = bt[i]; // Initialize remaining time
    }

    printf("Enter time quantum: ");
    scanf("%d", &tq);

    while (remaining != 0) {
        for (i = 0; i < n; i++) {
            if (rt[i] > 0) {
                if (rt[i] > tq) {
                    time += tq;
                    rt[i] -= tq;
                } else {
                    time += rt[i];
                    wt[i] = time - bt[i];
                    rt[i] = 0;
                    remaining--;
                }
            }
        }
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
        printf("P[%d]\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    avg_wt /= n;
    avg_tat /= n;

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

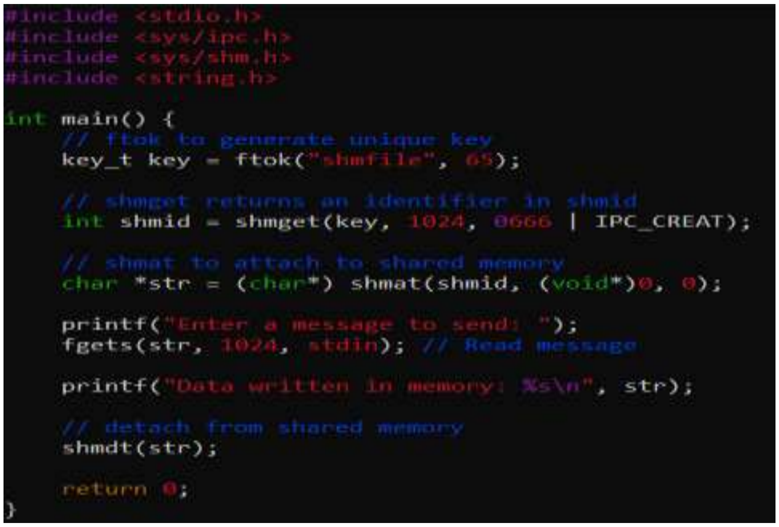
Output:



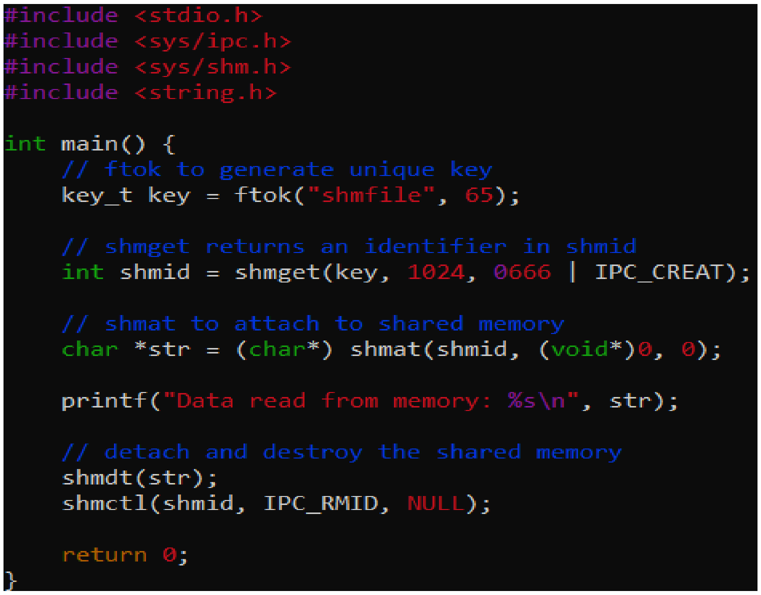
```
Enter total number of processes: 3
Enter burst time for P[1]: 10
Enter burst time for P[2]: 5
Enter burst time for P[3]: 8
Enter time quantum: 3

Process Burst Time      Waiting Time    Turnaround Time
P[1]    10              13              23
P[2]    5               9               14
P[3]    8               14              22

Average Waiting Time: 12.00
Average Turnaround Time: 19.67
```

7) IPC USING SHARED MEMORY

Program Code:

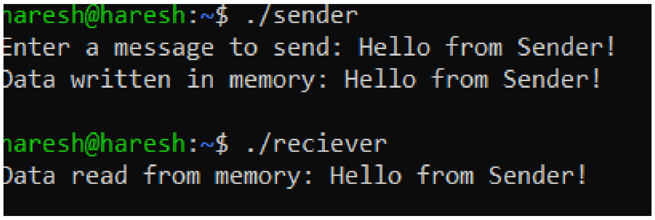


```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>

int main() {
    // ftok to generate unique key
    key_t key = ftok("shmfile", 65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid, (void*)0, 0);

    printf("Enter a message to send: ");
    fgets(str, 1024, stdin); // Read message

    printf("Data written in memory: %s\n", str);

    // detach from shared memory
    shmdt(str);

    return 0;
}
```



```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>

int main() {
    // ftok to generate unique key
    key_t key = ftok("shmfile", 65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid, (void*)0, 0);

    printf("Data read from memory: %s\n", str);

    // detach and destroy the shared memory
    shmdt(str);
    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}
```

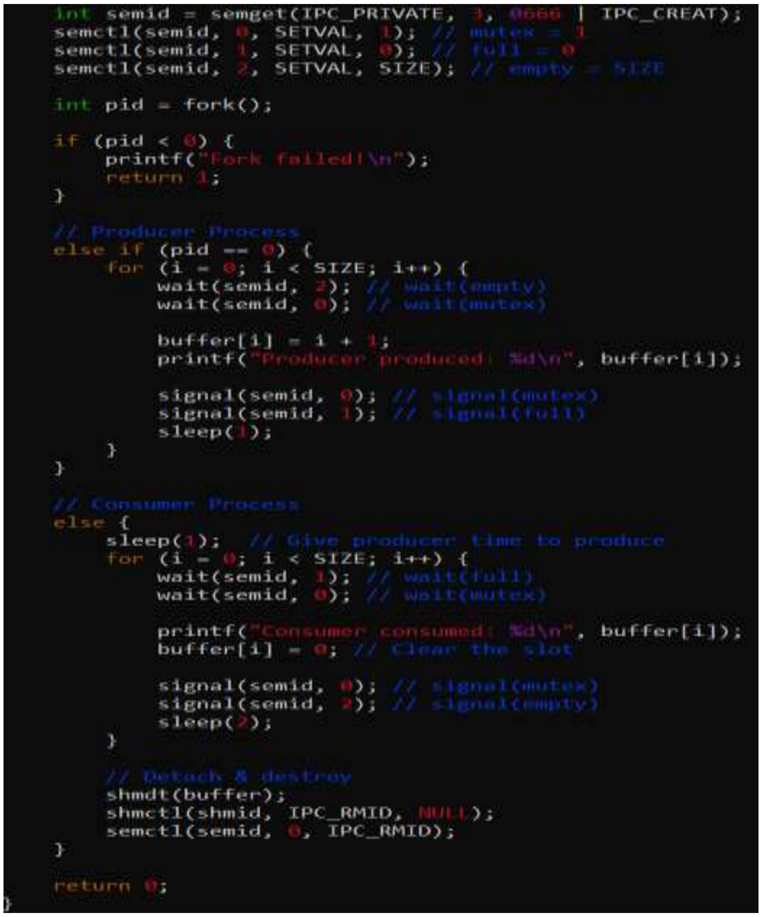
Output:



```
haresh@haresh:~$ ./sender
Enter a message to send: Hello from Sender!
Data written in memory: Hello from Sender!

haresh@haresh:~$ ./reciever
Data read from memory: Hello from Sender!
```

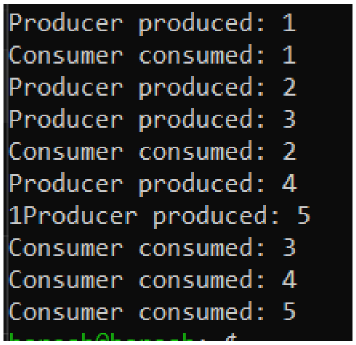
8) Producer consumer problem using semaphores

Program code:



```
    int semid = semget(IPC_PRIVATE, 3, 0666 | IPC_CREAT);
    semctl(semid, 0, SETVAL, 1); // mutex = 1
    semctl(semid, 1, SETVAL, 0); // full = 0
    semctl(semid, 2, SETVAL, SIZE); // empty = SIZE

    int pid = fork();

    if (pid < 0) {
        printf("Fork failed!\n");
        return 1;
    }

    // Producer Process
    else if (pid == 0) {
        for (i = 0; i < SIZE; i++) {
            wait(semid, 2); // wait(empty)
            wait(semid, 0); // wait(mutex)

            buffer[i] = i + 1;
            printf("Producer produced: %d\n", buffer[i]);

            signal(semid, 0); // signal(mutex)
            signal(semid, 1); // signal(full)
            sleep(1);
        }
    }

    // Consumer Process
    else {
        sleep(1);  // Give producer time to produce
        for (i = 0; i < SIZE; i++) {
            wait(semid, 1); // wait(full)
            wait(semid, 0); // wait(mutex)

            printf("Consumer consumed: %d\n", buffer[i]);
            buffer[i] = 0; // Clear the slot

            signal(semid, 0); // signal(mutex)
            signal(semid, 2); // signal(empty)
            sleep(2);
        }

        // Detach & destroy
        shmdt(buffer);
        shmctl(shmid, IPC_RMID, NULL);
        semctl(semid, 0, IPC_RMID);
    }

    return 0;
}
```

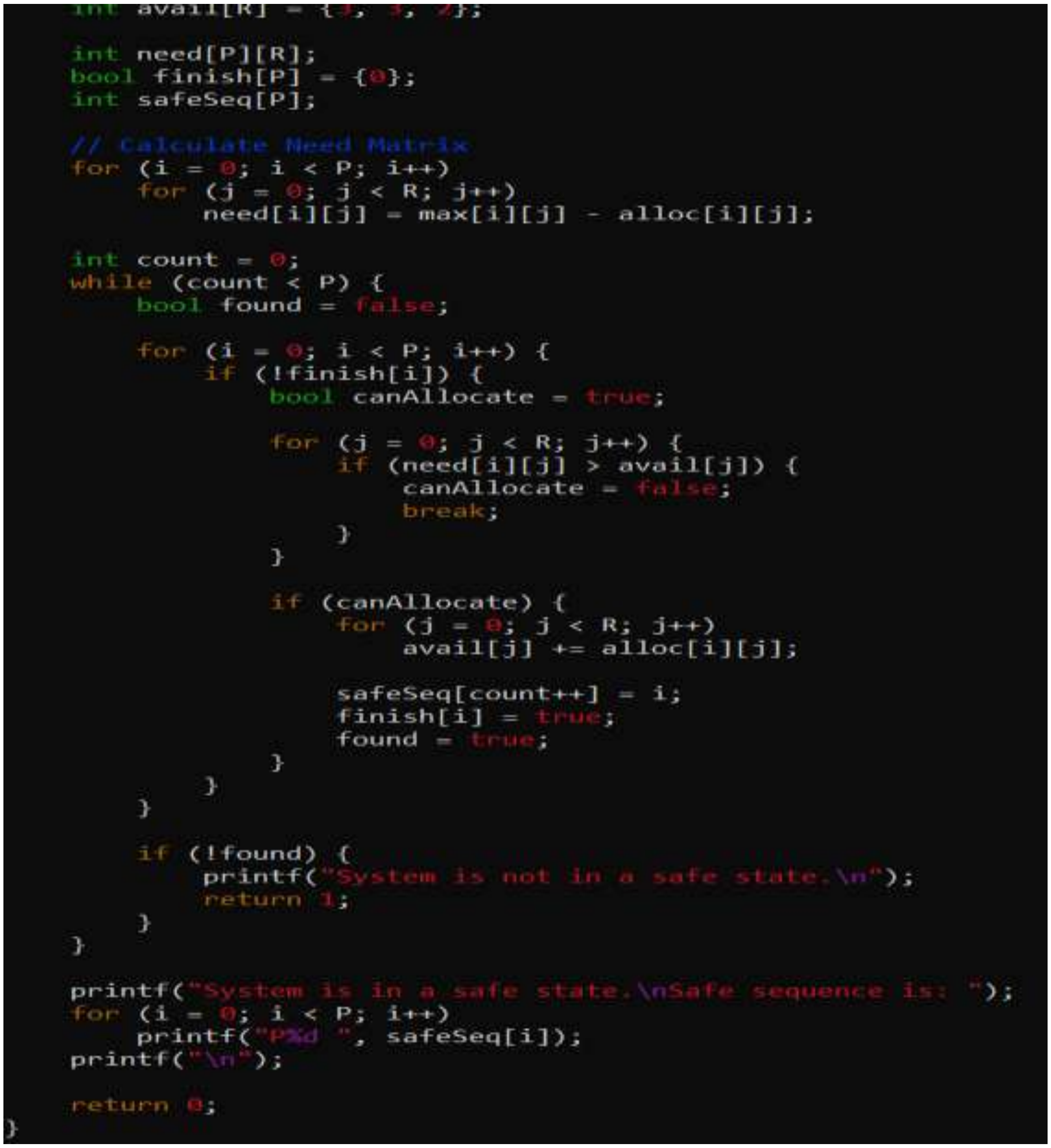
Output:



```
Producer produced: 1
Consumer consumed: 1
Producer produced: 2
Producer produced: 3
Consumer consumed: 2
Producer produced: 4
1Producer produced: 5
Consumer consumed: 3
Consumer consumed: 4
Consumer consumed: 5
```

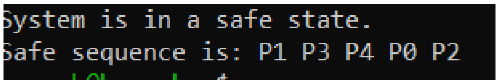
9) Banker's algorithm for deadlock avoidance

Program Code:



```
    int avail[R] = {3, 3, 2};

    int need[P][R];
    bool finish[P] = {0};
    int safeSeq[P];

    // Calculate Need Matrix
    for (i = 0; i < P; i++)
        for (j = 0; j < R; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    int count = 0;
    while (count < P) {
        bool found = false;

        for (i = 0; i < P; i++) {
            if (!finish[i]) {
                bool canAllocate = true;

                for (j = 0; j < R; j++) {
                    if (need[i][j] > avail[j]) {
                        canAllocate = false;
                        break;
                    }
                }

                if (canAllocate) {
                    for (j = 0; j < R; j++)
                        avail[j] += alloc[i][j];

                    safeSeq[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }

        if (!found) {
            printf("System is not in a safe state.\n");
            return 1;
        }
    }

    printf("System is in a safe state.\nSafe sequence is: ");
    for (i = 0; i < P; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");

    return 0;
}
```

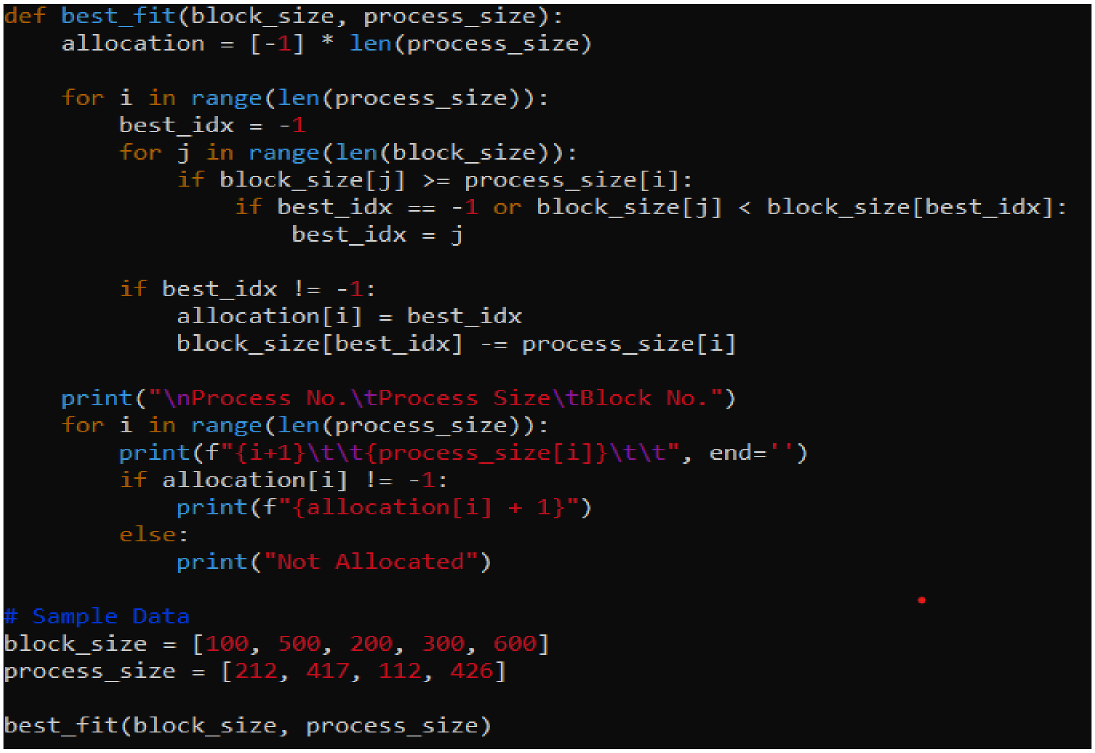
Output:



```
System is in a safe state.
Safe sequence is: P1 P3 P4 P0 P2
```

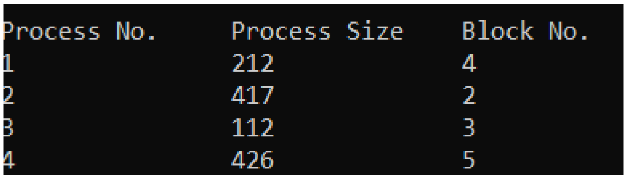
10a) Best Fit memory allocation technique

Program Code:



```python
def best_fit(block_size, process_size):
    allocation = [-1] * len(process_size)

    for i in range(len(process_size)):
        best_idx = -1
        for j in range(len(block_size)):
            if block_size[j] >= process_size[i]:
                if best_idx == -1 or block_size[j] < block_size[best_idx]:
                    best_idx = j

        if best_idx != -1:
            allocation[i] = best_idx
            block_size[best_idx] -= process_size[i]

    print("\nProcess No.\tProcess Size\tBlock No.")
    for i in range(len(process_size)):
        print(f"{i+1}\t\t{process_size[i]}\t\t", end='')
        if allocation[i] != -1:
            print(f"{allocation[i] + 1}")
        else:
            print("Not Allocated")

# Sample Data
block_size = [100, 500, 200, 300, 600]
process_size = [212, 417, 112, 426]

best_fit(block_size, process_size)
```

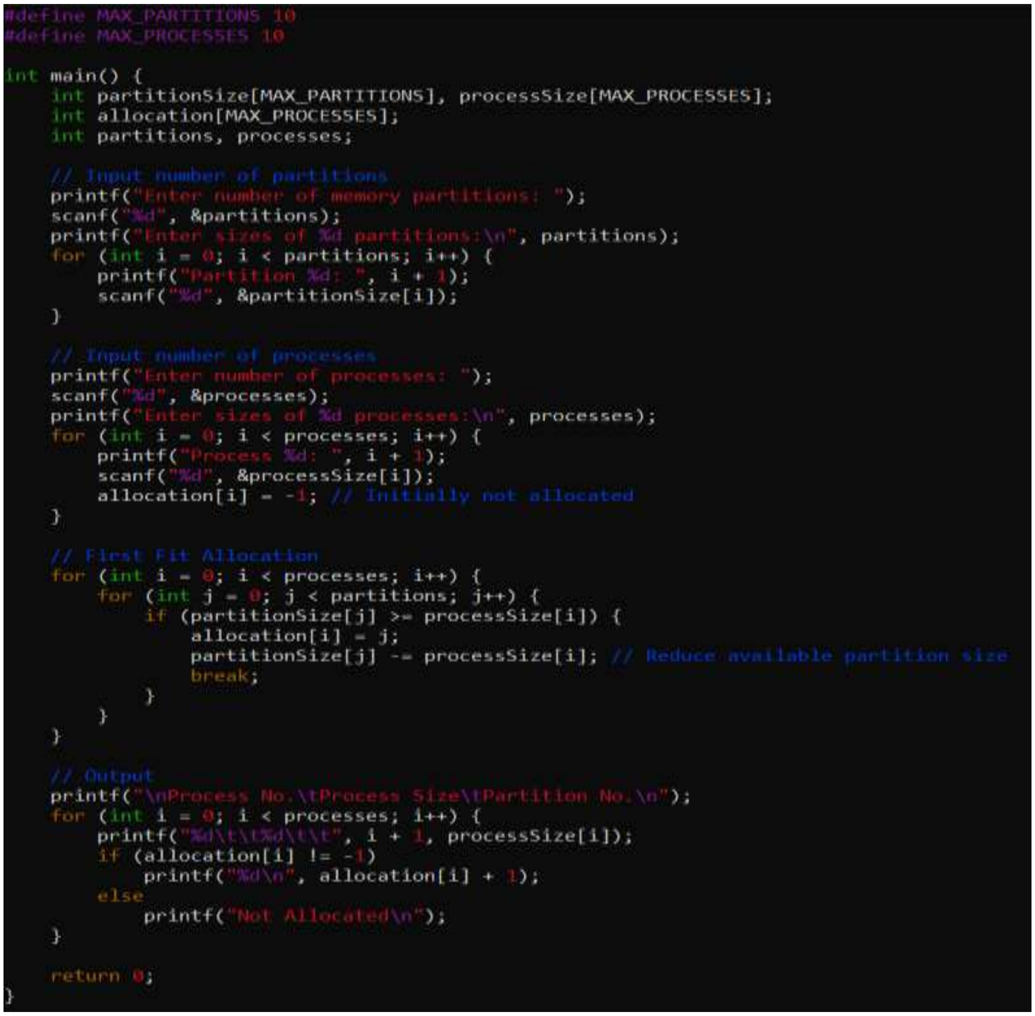
Output:



```
Process No.     Process Size    Block No.
1               212             4
2               417             2
3               112             3
4               426             5
```

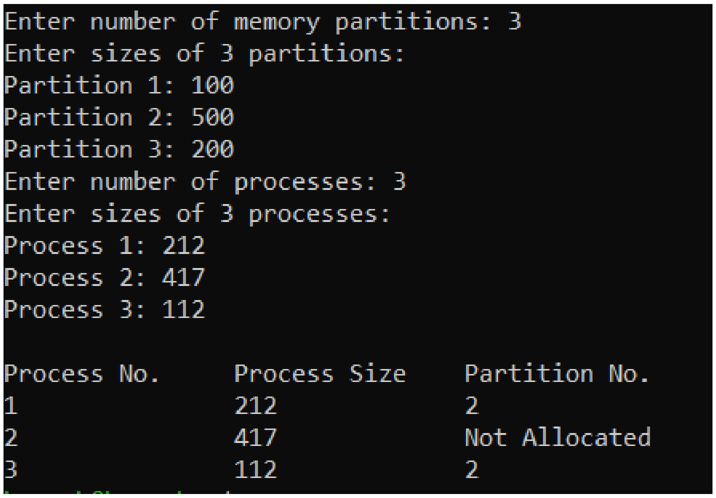
10b) memory allocation methods for fixed partition using first fit

Program Code:



```c
#define MAX_PARTITIONS 10
#define MAX_PROCESSES 10

int main() {
    int partitionSize[MAX_PARTITIONS], processSize[MAX_PROCESSES];
    int allocation[MAX_PROCESSES];
    int partitions, processes;

    // Input number of partitions
    printf("Enter number of memory partitions: ");
    scanf("%d", &partitions);
    printf("Enter sizes of %d partitions:\n", partitions);
    for (int i = 0; i < partitions; i++) {
        printf("Partition %d: ", i + 1);
        scanf("%d", &partitionSize[i]);
    }

    // Input number of processes
    printf("Enter number of processes: ");
    scanf("%d", &processes);
    printf("Enter sizes of %d processes:\n", processes);
    for (int i = 0; i < processes; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d", &processSize[i]);
        allocation[i] = -1; // Initially not allocated
    }

    // First Fit Allocation
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < partitions; j++) {
            if (partitionSize[j] >= processSize[i]) {
                allocation[i] = j;
                partitionSize[j] -= processSize[i]; // Reduce available partition size
                break;
            }
        }
    }

    // Output
    printf("\nProcess No.\tProcess Size\tPartition No.\n");
    for (int i = 0; i < processes; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }

    return 0;
}
```

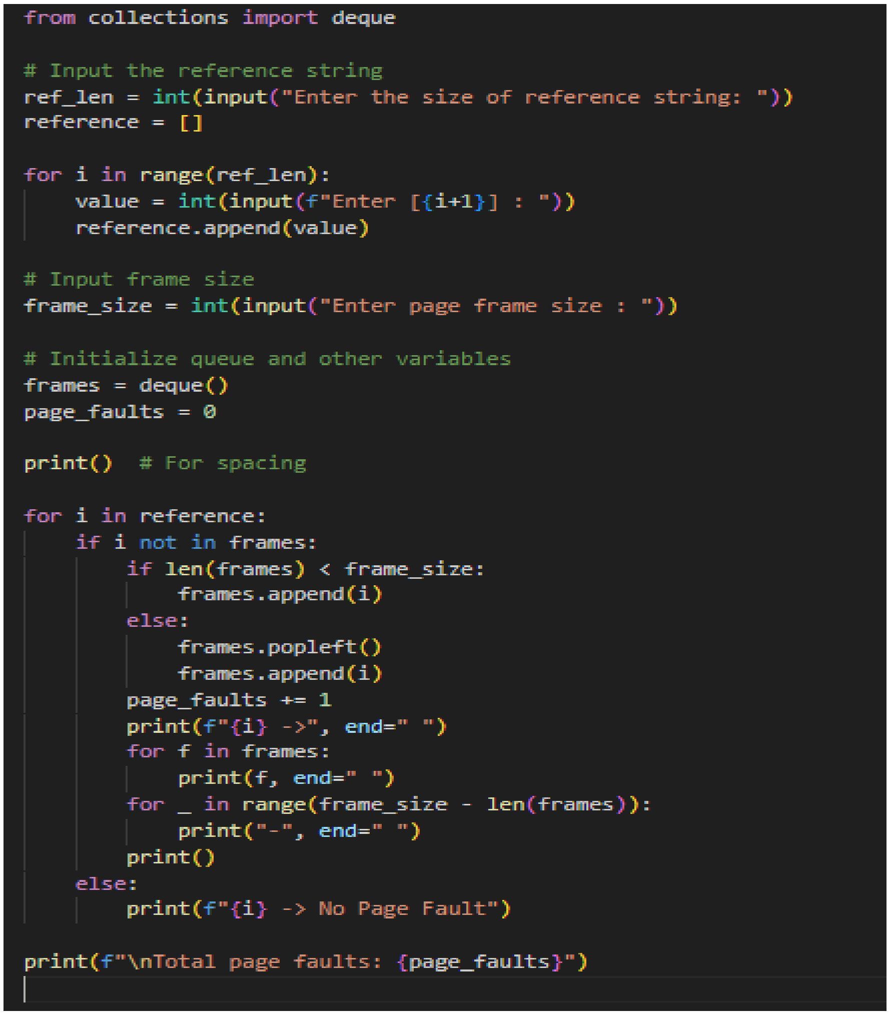
Output:



```
Enter number of memory partitions: 3
Enter sizes of 3 partitions:
Partition 1: 100
Partition 2: 500
Partition 3: 200
Enter number of processes: 3
Enter sizes of 3 processes:
Process 1: 212
Process 2: 417
Process 3: 112

Process No.     Process Size    Partition No.
1               212             2
2               417             Not Allocated
3               112             2
```

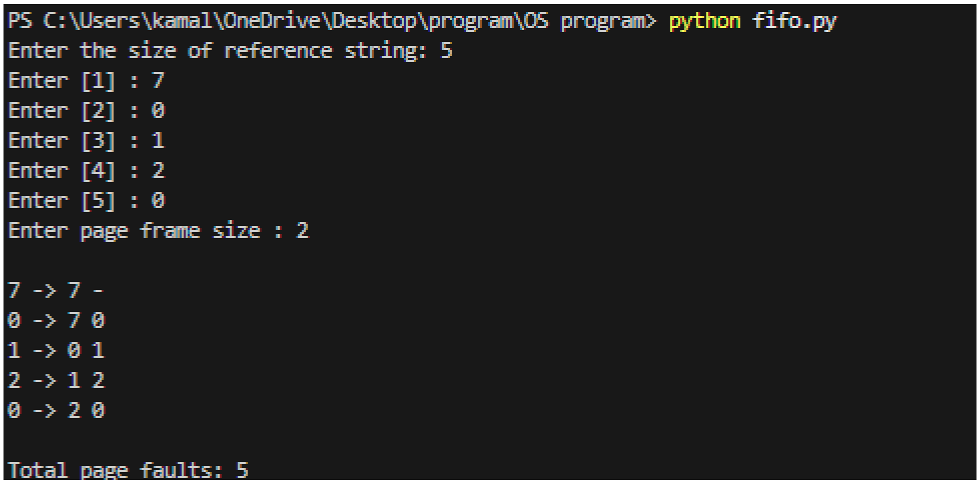
11a) FIFO

Program Code:



```python
from collections import deque

# Input the reference string
ref_len = int(input("Enter the size of reference string: "))
reference = []

for i in range(ref_len):
    value = int(input(f"Enter [{i+1}] : "))
    reference.append(value)

# Input frame size
frame_size = int(input("Enter page frame size : "))

# Initialize queue and other variables
frames = deque()
page_faults = 0

print()  # For spacing

for i in reference:
    if i not in frames:
        if len(frames) < frame_size:
            frames.append(i)
        else:
            frames.popleft()
            frames.append(i)
        page_faults += 1
        print(f"{i} ->", end=" ")
        for f in frames:
            print(f, end=" ")
        for _ in range(frame_size - len(frames)):
            print("-", end=" ")
        print()
    else:
        print(f"{i} -> No Page Fault")

print(f"\nTotal page faults: {page_faults}")
```

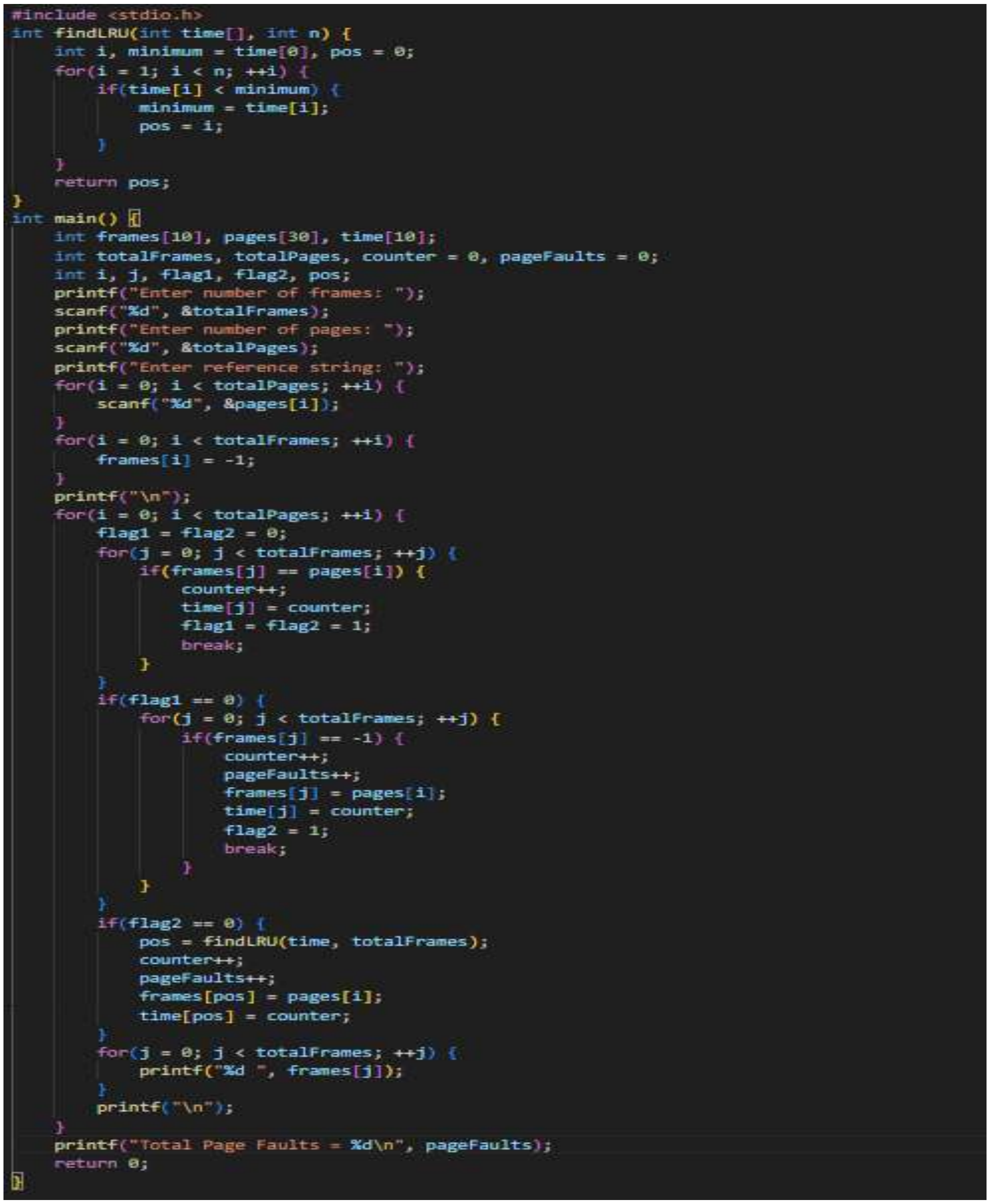
Output:



```
PS C:\Users\kamal\OneDrive\Desktop\program\OS program> python fifo.py
Enter the size of reference string: 5
Enter [1] : 7
Enter [2] : 0
Enter [3] : 1
Enter [4] : 2
Enter [5] : 0
Enter page frame size : 2

7 -> 7 -
0 -> 7 0
1 -> 0 1
2 -> 1 2
0 -> 2 0

Total page faults: 5
```

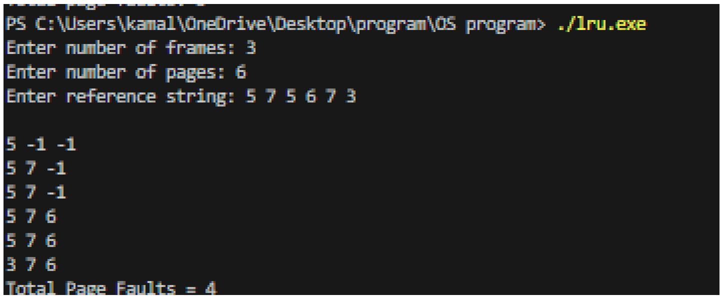
11b) LRU

Program Code:



```c
#include <stdio.h>
int findLRU(int time[], int n) {
    int i, minimum = time[0], pos = 0;
    for(i = 1; i < n; ++i) {
        if(time[i] < minimum) {
            minimum = time[i];
            pos = i;
        }
    }
    return pos;
}
int main() {
    int frames[10], pages[30], time[10];
    int totalFrames, totalPages, counter = 0, pageFaults = 0;
    int i, j, flag1, flag2, pos;
    printf("Enter number of frames: ");
    scanf("%d", &totalFrames);
    printf("Enter number of pages: ");
    scanf("%d", &totalPages);
    printf("Enter reference string: ");
    for(i = 0; i < totalPages; ++i) {
        scanf("%d", &pages[i]);
    }
    for(i = 0; i < totalFrames; ++i) {
        frames[i] = -1;
    }
    printf("\n");
    for(i = 0; i < totalPages; ++i) {
        flag1 = flag2 = 0;
        for(j = 0; j < totalFrames; ++j) {
            if(frames[j] == pages[i]) {
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                break;
            }
        }
        if(flag1 == 0) {
            for(j = 0; j < totalFrames; ++j) {
                if(frames[j] == -1) {
                    counter++;
                    pageFaults++;
                    frames[j] = pages[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
                }
            }
        }
        if(flag2 == 0) {
            pos = findLRU(time, totalFrames);
            counter++;
            pageFaults++;
            frames[pos] = pages[i];
            time[pos] = counter;
        }
        for(j = 0; j < totalFrames; ++j) {
            printf("%d ", frames[j]);
        }
        printf("\n");
    }
    printf("Total Page Faults = %d\n", pageFaults);
    return 0;
}
```

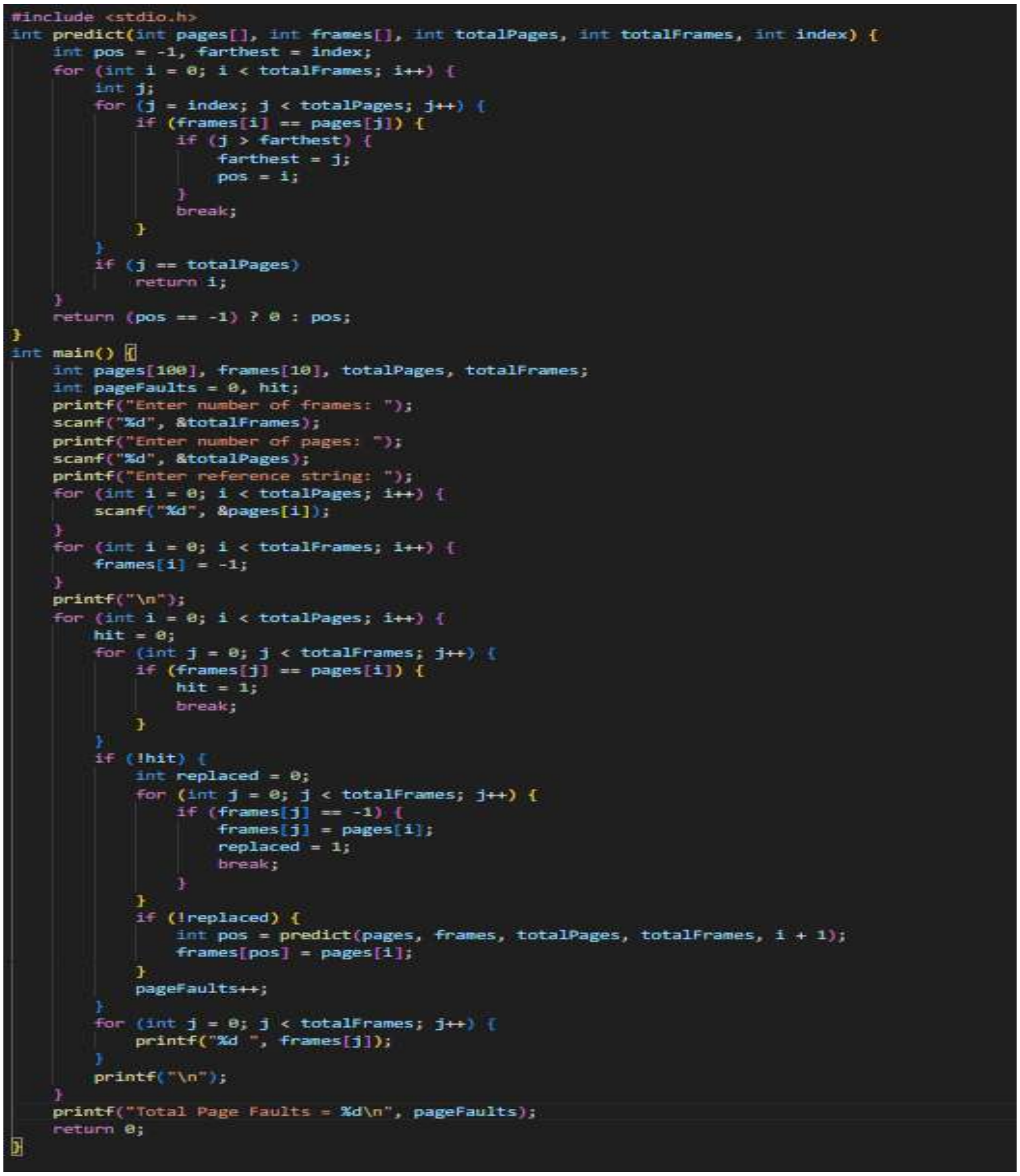
Output:



```
PS C:\Users\kamal\OneDrive\Desktop\program\OS program> ./lru.exe
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3

5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6
Total Page Faults = 4
```

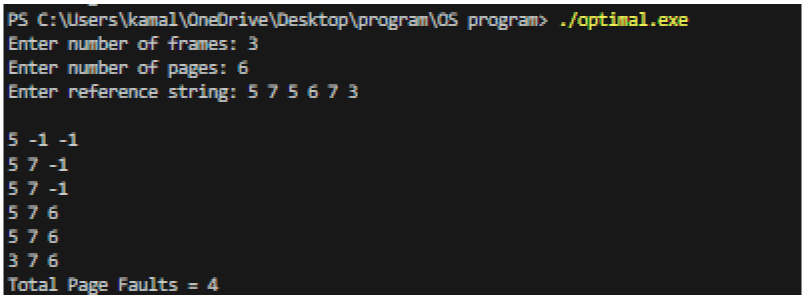
11c) Optimal

Program Code:



```
#include <stdio.h>
int predict(int pages[], int frames[], int totalPages, int totalFrames, int index) {
    int pos = -1, farthest = index;
    for (int i = 0; i < totalFrames; i++) {
        int j;
        for (j = index; j < totalPages; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    pos = i;
                }
                break;
            }
        }
        if (j == totalPages)
            return i;
    }
    return (pos == -1) ? 0 : pos;
}
int main() {
    int pages[100], frames[10], totalPages, totalFrames;
    int pageFaults = 0, hit;
    printf("Enter number of frames: ");
    scanf("%d", &totalFrames);
    printf("Enter number of pages: ");
    scanf("%d", &totalPages);
    printf("Enter reference string: ");
    for (int i = 0; i < totalPages; i++) {
        scanf("%d", &pages[i]);
    }
    for (int i = 0; i < totalFrames; i++) {
        frames[i] = -1;
    }
    printf("\n");
    for (int i = 0; i < totalPages; i++) {
        hit = 0;
        for (int j = 0; j < totalFrames; j++) {
            if (frames[j] == pages[i]) {
                hit = 1;
                break;
            }
        }
        if (!hit) {
            int replaced = 0;
            for (int j = 0; j < totalFrames; j++) {
                if (frames[j] == -1) {
                    frames[j] = pages[i];
                    replaced = 1;
                    break;
                }
            }
            if (!replaced) {
                int pos = predict(pages, frames, totalPages, totalFrames, i + 1);
                frames[pos] = pages[i];
            }
            pageFaults++;
        }
        for (int j = 0; j < totalFrames; j++) {
            printf("%d ", frames[j]);
        }
        printf("\n");
    }
    printf("Total Page Faults = %d\n", pageFaults);
    return 0;
}
```

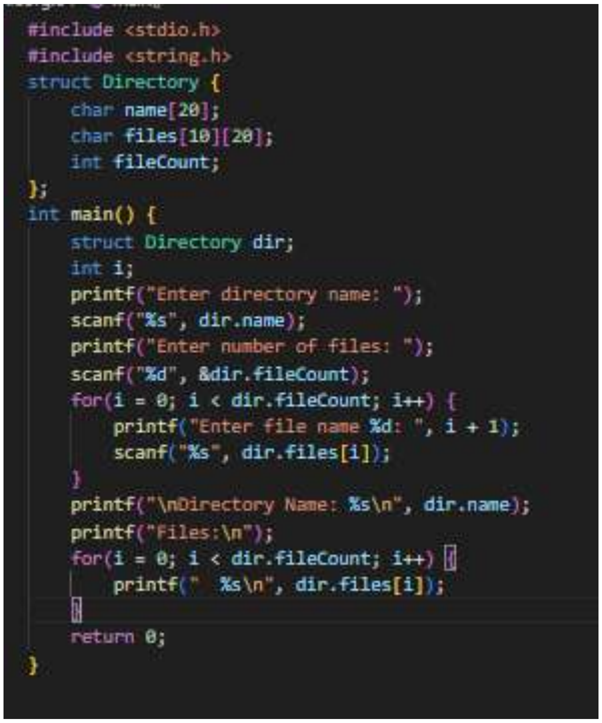
Output:



```
PS C:\Users\kamal\OneDrive\Desktop\program\OS program> ./optimal.exe
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3

5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6
Total Page Faults = 4
```

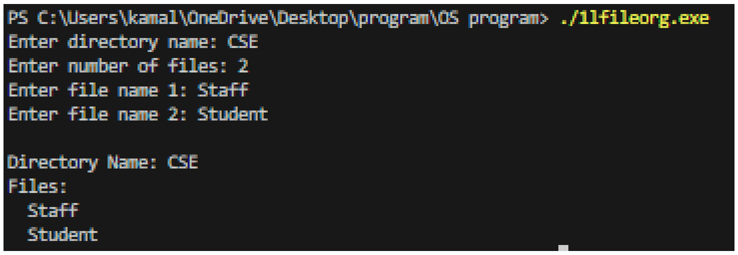
12a) Single Level File Organization Technique

Program Code:



```
#include <stdio.h>
#include <string.h>
struct Directory {
    char name[20];
    char files[10][20];
    int fileCount;
};
int main() {
    struct Directory dir;
    int i;
    printf("Enter directory name: ");
    scanf("%s", dir.name);
    printf("Enter number of files: ");
    scanf("%d", &dir.fileCount);
    for(i = 0; i < dir.fileCount; i++) {
        printf("Enter file name %d: ", i + 1);
        scanf("%s", dir.files[i]);
    }
    printf("\nDirectory Name: %s\n", dir.name);
    printf("Files:\n");
    for(i = 0; i < dir.fileCount; i++) {
        printf("  %s\n", dir.files[i]);
    }
    return 0;
}
```

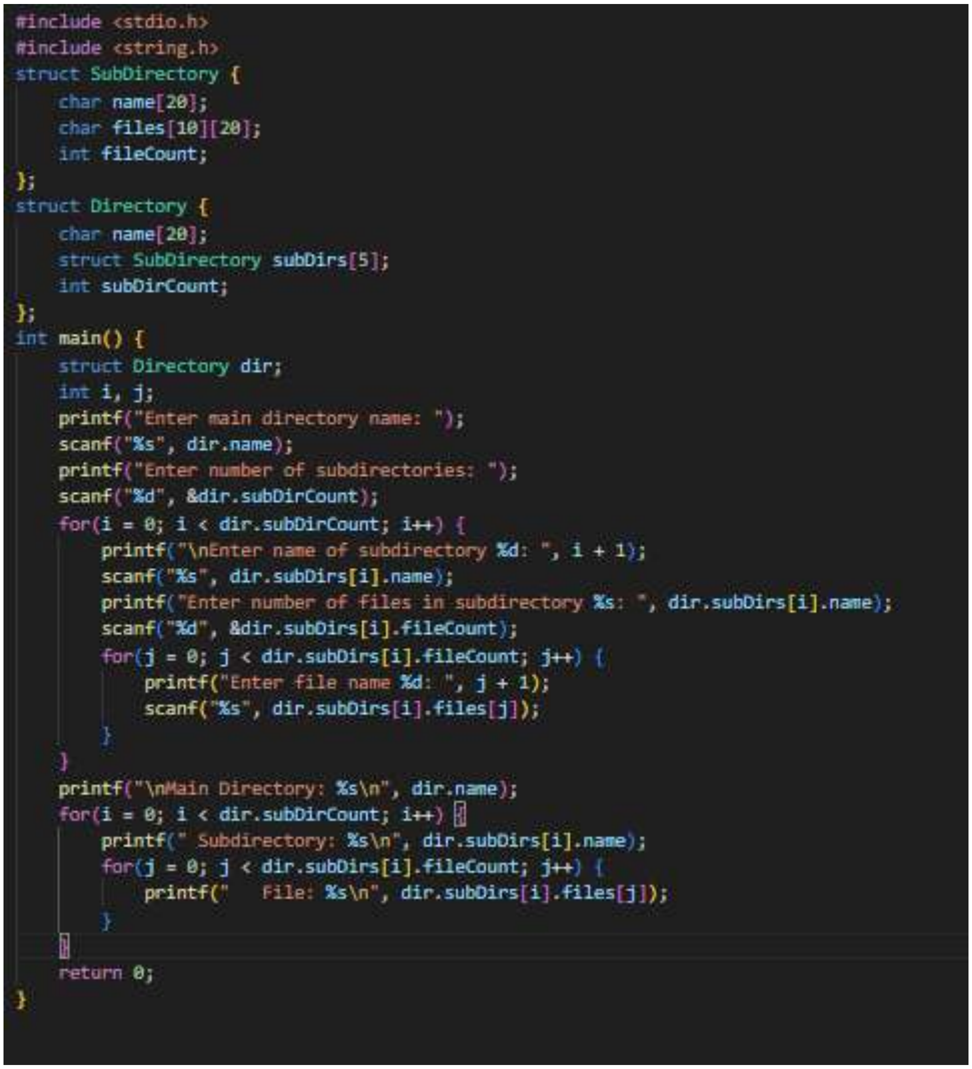
Output:



```
PS C:\Users\kamal\OneDrive\Desktop\program\OS program> ./11fileorg.exe
Enter directory name: CSE
Enter number of files: 2
Enter file name 1: Staff
Enter file name 2: Student

Directory Name: CSE
Files:
  Staff
  Student
```

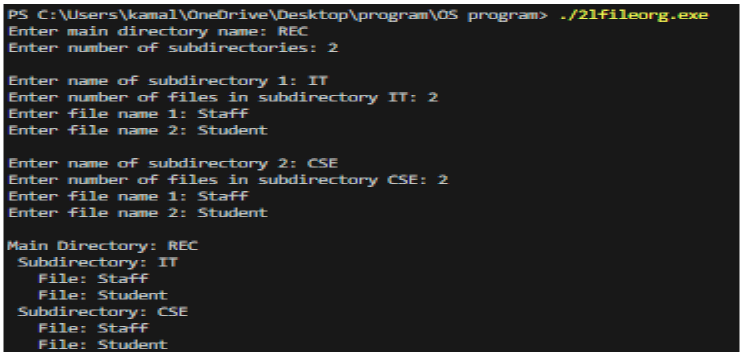
12b) Two-level File Organization Technique

Program Code:



```c
#include <stdio.h>
#include <string.h>
struct SubDirectory {
    char name[20];
    char files[10][20];
    int fileCount;
};
struct Directory {
    char name[20];
    struct SubDirectory subDirs[5];
    int subDirCount;
};
int main() {
    struct Directory dir;
    int i, j;
    printf("Enter main directory name: ");
    scanf("%s", dir.name);
    printf("Enter number of subdirectories: ");
    scanf("%d", &dir.subDirCount);
    for(i = 0; i < dir.subDirCount; i++) {
        printf("\nEnter name of subdirectory %d: ", i + 1);
        scanf("%s", dir.subDirs[i].name);
        printf("Enter number of files in subdirectory %s: ", dir.subDirs[i].name);
        scanf("%d", &dir.subDirs[i].fileCount);
        for(j = 0; j < dir.subDirs[i].fileCount; j++) {
            printf("Enter file name %d: ", j + 1);
            scanf("%s", dir.subDirs[i].files[j]);
        }
    }
    printf("\nMain Directory: %s\n", dir.name);
    for(i = 0; i < dir.subDirCount; i++) {
        printf(" Subdirectory: %s\n", dir.subDirs[i].name);
        for(j = 0; j < dir.subDirs[i].fileCount; j++) {
            printf("   File: %s\n", dir.subDirs[i].files[j]);
        }
    }
    return 0;
}
```

Output:



```
PS C:\Users\kamal\OneDrive\Desktop\program\OS program> ./21fileorg.exe
Enter main directory name: REC
Enter number of subdirectories: 2

Enter name of subdirectory 1: IT
Enter number of files in subdirectory IT: 2
Enter file name 1: Staff
Enter file name 2: Student

Enter name of subdirectory 2: CSE
Enter number of files in subdirectory CSE: 2
Enter file name 1: Staff
Enter file name 2: Student

Main Directory: REC
 Subdirectory: IT
   File: Staff
   File: Student
 Subdirectory: CSE
   File: Staff
   File: Student
```