Progressive Education Society's

Modern College of Engineering, Pune

MCA Department

A.Y.2022-23

(310918) Operating System Laboratory

****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 1 **Date of Implementation:** 15. 5. 23

****

Q. b) To write C Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

Program:

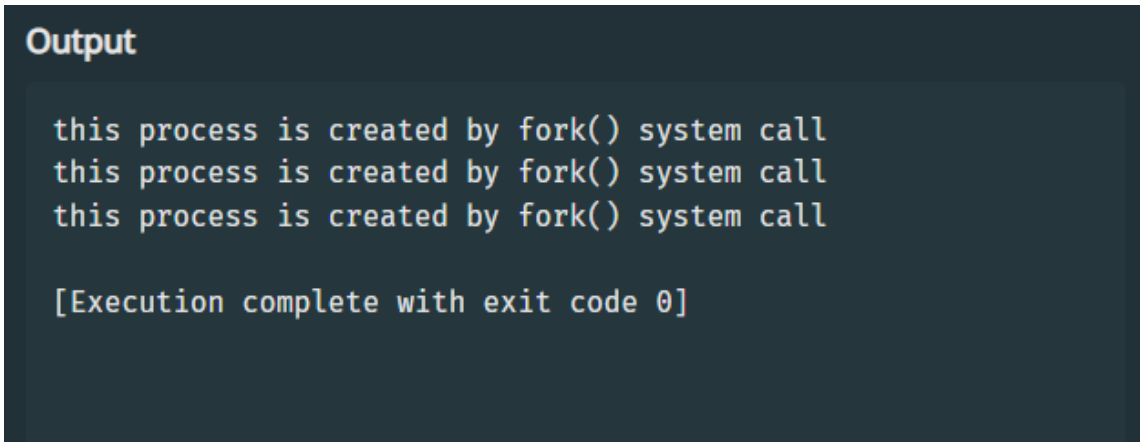
fork():

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

int main()
{
   fork();
   fork();
   fork();
   printf("this process is created by fork() system call\n");
   return 0;
}
```

Output:

```
this process is created by fork() system call
this process is created by fork() system call
this process is created by fork() system call

[Execution complete with exit code 0]
```

Screenshot:



Program:

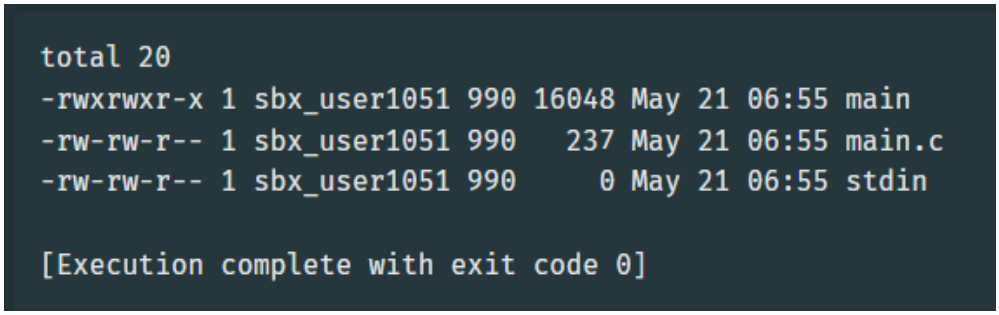
exec():

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  execlp("ls", "ls", "-l", NULL);
  perror("Return from execlp() not expected");
  exit(EXIT_FAILURE);
}
```

Output:

```
total 20
-rwxrwxr-x 1 sbx_user1051 990 16048 May 21 06:55 main
-rw-rw-r-- 1 sbx_user1051 990   237 May 21 06:55 main.c
-rw-rw-r-- 1 sbx_user1051 990     0 May 21 06:55 stdin

[Execution complete with exit code 0]
```

Screenshot:



Program:

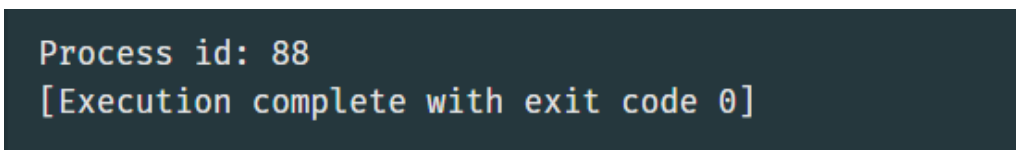
getpid():

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

int main() {
   int pid =  getpid();
   printf("Process id: %d", pid);
   return 0;
}
```

Output:

```
Process id: 88
[Execution complete with exit code 0]
```

Screenshot:



Program:

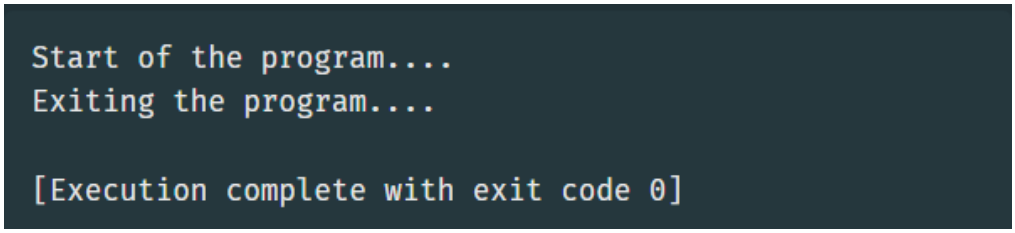
exit():

```
#include <stdio.h>
#include <stdlib.h>

int main () {
  printf("Start of the program....\n");

  printf("Exiting the program....\n");
  exit(0);

  printf("End of the program....\n");

  return(0);
}
```

Output:

```
Start of the program....
Exiting the program....

[Execution complete with exit code 0]
```

Screenshot:



Program:

wait():

```
#include<unistd.h>
#include<sys/types.h>
#include<stdio.h>
#include<sys/wait.h>
int main()
{
   pid_t p;
   p=fork();
if(p==0)//child
{
   printf("Child process\n");
   printf("Child Id %d\n",getpid());
   printf("Parent Id %d\n",getppid());
}
else//parent
{
   wait(NULL);
   printf("Parent process\n");
   printf("Child Id %d\n",p);
   printf("Parent Id %d\n",getpid());
}
   printf("\n");
}
```

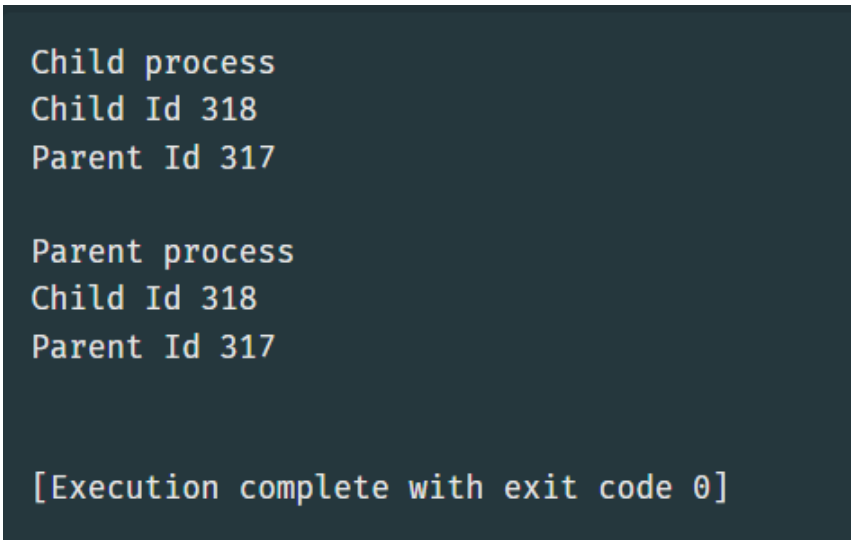
Output:

Child process
Child Id 318
Parent Id 317

Parent process
Child Id 318
Parent Id 317

[Execution complete with exit code 0]

Screenshot:



Program:

close():

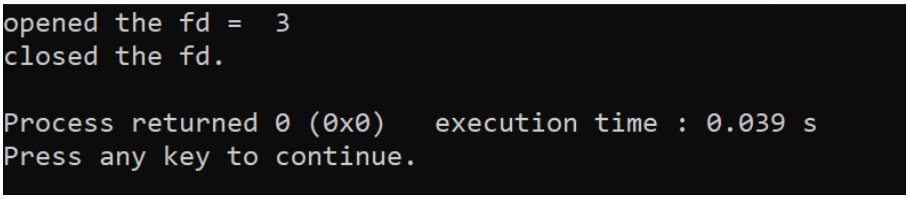
```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
   int fd1 = open("demo.txt", O_RDONLY);
   if (fd1 < 0) {
      perror("c1");
      exit(1);
   }
   printf("opened the fd = % d\n", fd1);

   if (close(fd1) < 0) {
      perror("c1");
     exit(1);
   }
   printf("closed the fd.\n");
}
```

Output:

opened the fd = 3
closed the fd.

Process returned 0 (0x0)   execution time : 0.023 s

Screenshot:



Program:

stat():

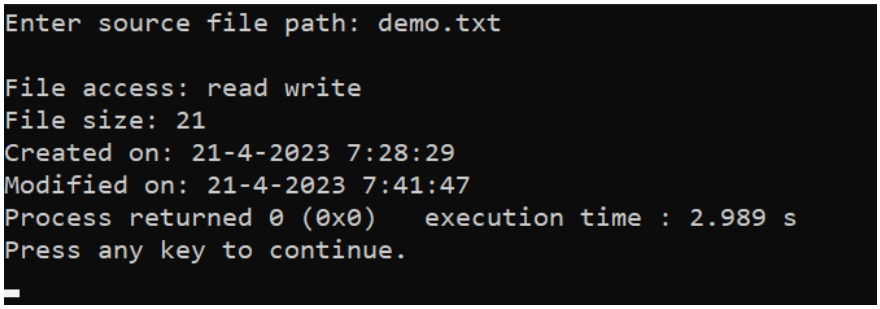
```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>


void printFileProperties(struct stat stats);


int main()
{
   char path[100];
   struct stat stats;

   printf("Enter source file path: ");
   scanf("%s", path);

   if (stat(path, &stats) == 0)
   {
      printFileProperties(stats);
   }
   else
   {
      printf("Unable to get file properties.\n");
      printf("Please check whether '%s' file exists.\n", path);
   }

   return 0;
}


void printFileProperties(struct stat stats)
{
   struct tm dt;

   printf("\nFile access: ");
   if (stats.st_mode & R_OK)
      printf("read ");
   if (stats.st_mode & W_OK)
      printf("write ");
   if (stats.st_mode & X_OK)
      printf("execute");


   printf("\nFile size: %d", stats.st_size);
```

```
    dt = *(gmtime(&stats.st_ctime));
    printf("\nCreated on: %d-%d-%d %d:%d:%d", dt.tm_mday, dt.tm_mon, dt.tm_year + 1900,
                                    dt.tm_hour, dt.tm_min, dt.tm_sec);
    dt = *(gmtime(&stats.st_mtime));
    printf("\nModified on: %d-%d-%d %d:%d:%d", dt.tm_mday, dt.tm_mon, dt.tm_year + 1900,
                                    dt.tm_hour, dt.tm_min, dt.tm_sec);

}
```

Output:

```
Enter source file path: demo.txt

File access: read write
File size: 21
Created on: 21-4-2023 7:28:29
Modified on: 21-4-2023 7:41:47
Process returned 0 (0x0)   execution time : 2.989 s
```

Screenshot:



Program:

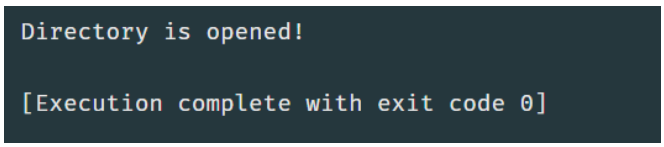
opendir():

```
#include<stdio.h>
#include<dirent.h>
int main(){
    DIR *folder;
    folder = opendir(".");
    if(folder == NULL){
        puts("Unable to read directory");
        return(1);
    }
    else{
        puts("Directory is opened!");
    }
    closedir(folder);
    return(0);
}
```

Output:

```
Directory is opened!
[Execution complete with exit code 0]
```

Screenshot:



Program:

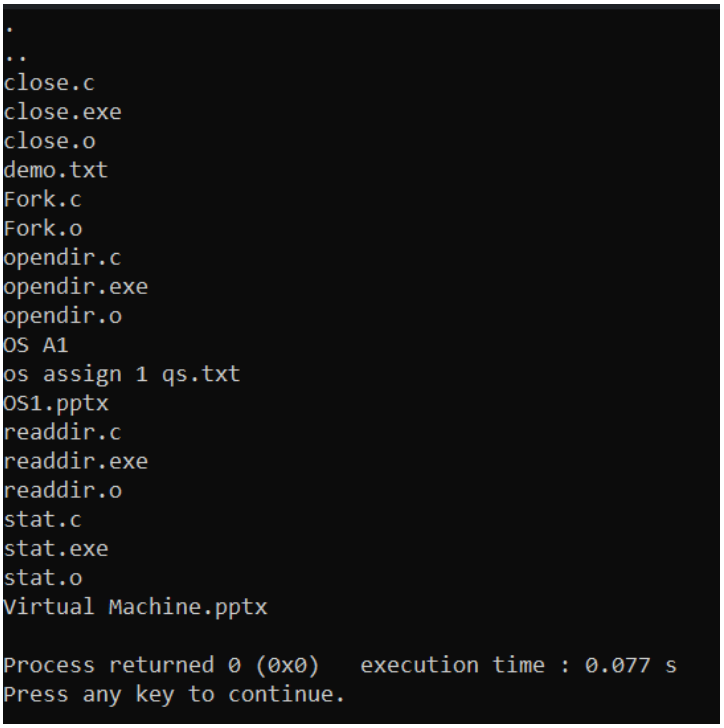
readdir():

```
#include <stdio.h>
#include <dirent.h>

int main(void)
{
        struct dirent *de;
        DIR *dr = opendir(".");

        if (dr == NULL)
        {
                printf("Could not open current directory" );
                return 0;
        }

        while ((de = readdir(dr)) != NULL)
                        printf("%s\n", de->d_name);

        closedir(dr);
        return 0;
}
```

Output:

```
.
..
close.c
close.exe
close.o
demo.txt
Fork.c
Fork.o
opendir.c
opendir.exe
opendir.o
OS A1
os assign 1 qs.txt
OS1.pptx
readdir.c
readdir.exe
readdir.o
stat.c
stat.exe
stat.o
Virtual Machine.pptx

Process returned 0 (0x0)   execution time : 0.077 s
```

Screenshot:


.
..
close.c
close.exe
close.o
demo.txt
Fork.c
Fork.o
opendir.c
opendir.exe
opendir.o
OS A1
os assign 1 qs.txt
OS1.pptx
readdir.c
readdir.exe
readdir.o
stat.c
stat.exe
stat.o
Virtual Machine.pptx
Process returned 0 (0x0)   execution time : 0.077 s
Press any key to continue.

Progressive Education Society's

Modern College of Engineering, Pune

MCA Department

A.Y.2022-23

(310918) Operator System Laboratory

****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 1 **Date of Implementation:** 15. 5. 23

****

Q. c) Write simple shell programs by using conditional, branching and looping statements.

i) Even and odd no.

ii) Find out Factorial

iii) Swapping of two integers.

Solution:

i) Even and odd no.

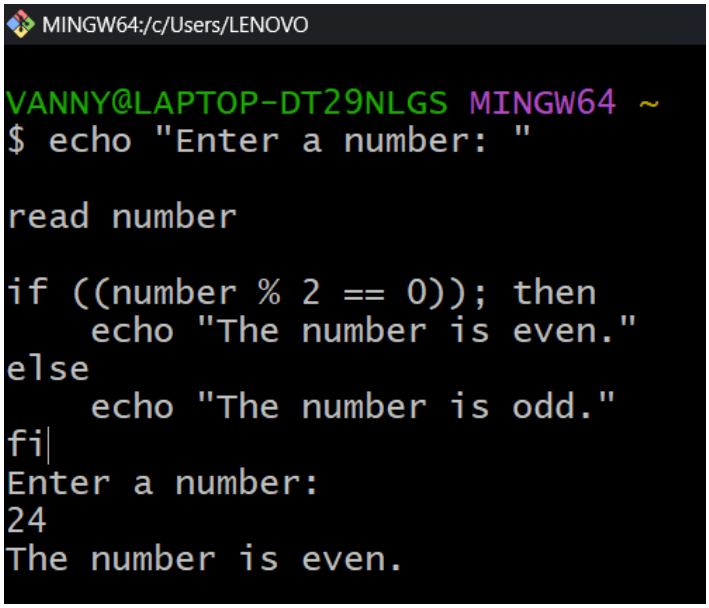
Program:

```
$ echo "Enter a number: "

read number

if ((number % 2 == 0)); then
   echo "The number is even."
else
   echo "The number is odd."
fi
```

Output:

```
Enter a number:
24
The number is even.
```

Screenshot:



ii) Find out Factorial

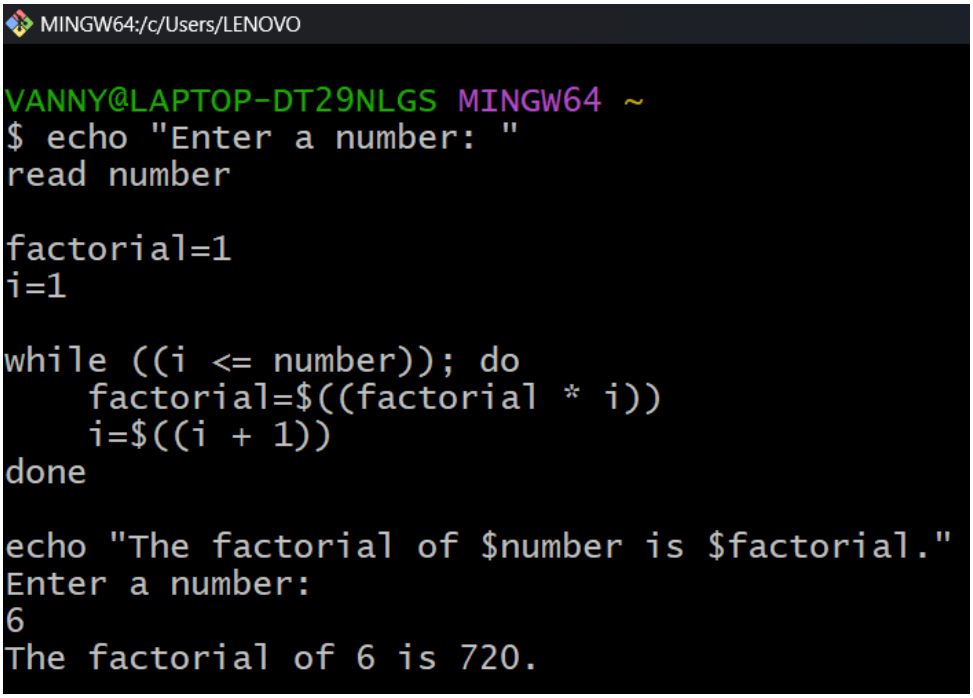
Program:

```
$ echo "Enter a number: "
read number

factorial=1
i=1

while ((i <= number)); do
   factorial=$((factorial * i))
   i=$((i + 1))
done

echo "The factorial of $number is $factorial."
```

Output:

```
Enter a number:
6
The factorial of 6 is 720.
```

Screenshot:



iii) Swapping of two integers.

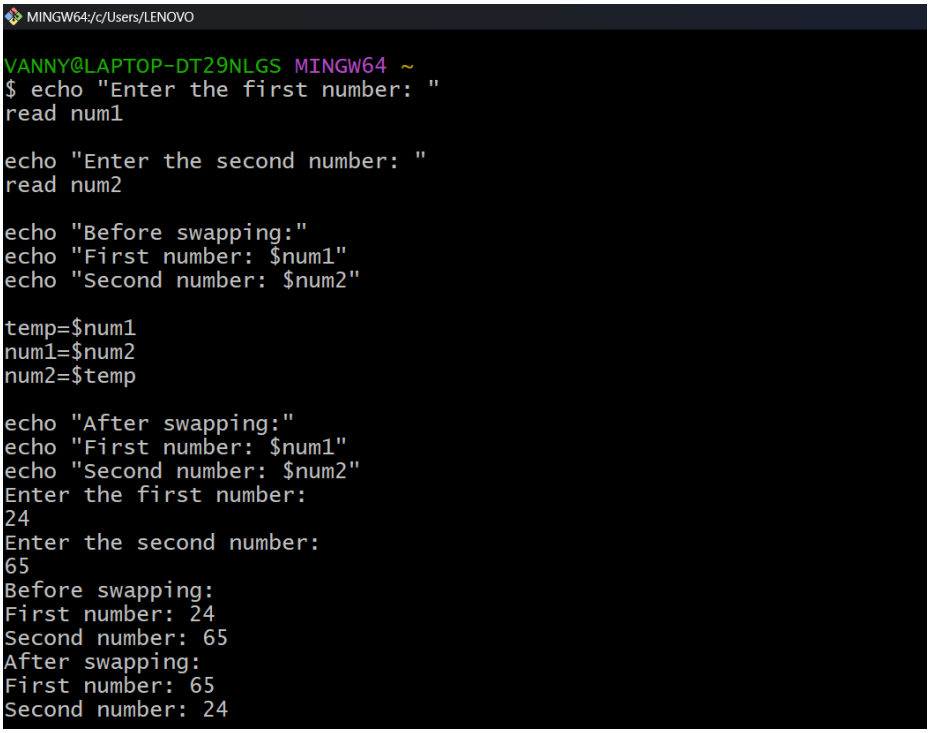
Program:

```
$ echo "Enter the first number: "
read num1
echo "Enter the second number: "
read num2

echo "Before swapping:"
echo "First number: $num1"
echo "Second number: $num2"

temp=$num1
num1=$num2
num2=$temp

echo "After swapping:"
echo "First number: $num1"
echo "Second number: $num2"
```

Output:

```
Enter the first number:
24
Enter the second number:
65
Before swapping:
First number: 24
Second number: 65
After swapping:
First number: 65
Second number: 24
```

Screenshot:



****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 2 **Date of Implementation:** 6. 6. 23

****

Q1. Write a program to implement FCFS, SJF, Priority and Round Robin algorithms

FCFS ALGORITHM

Program:

```
#include<stdio.h>
int main()
{
   int at[10],at2[10],bt[100],ex[100],seq[100],re[100],wt[100],tat[100];
   int n,i,j,start,pos,max=0,min,idle=0,k=0;
   float av1=0,av2=0;

   printf("*****INPUT*****\n");
   printf("Enter number of process\n");
   scanf("%d",&n);
   printf("Enter arrival time for processess\n");
   for(i=0;i<n;i++)
   {
    scanf("%d",&at[i]);
    at2[i]=at[i];
   }
   printf("Enter burst time for processess\n");
   for(i=0;i<n;i++)
   {
    scanf("%d",&bt[i]);
   }
   start=at[0];
   for(i=1;i<n;i++)
   {
     if(start>at[i])
      {
      start=at[i];
      }
    }
   printf("*****OUTPUT*****\n");
   printf("Sequence of execution is\n");
   for(i=0;i<n;i++)
   {
   if(max<at[i])
    {
     max=at[i];
    }
   }
```

```
   max=max+1;
  for(i=0;i<n;i++,k++)

    {  min=max;

      for(j=0;j<n;j++){
         if(at[j]!=-1)
           {
            if(at[j]<min)
              {
               min=at[j];
               pos=j;
              }
           }
        }
      printf("[P%d]  ",pos);
      seq[k]=pos;
      if(start<at[pos]){
        re[pos]=start;
        idle+=at[pos]-start;
        start=at[pos];
        start+=bt[pos];
        at[pos]=-1;
        ex[pos]=start;
      }
      else{
        re[pos]=start;
        start+=bt[pos];
        at[pos]=-1;
        ex[pos]=start;
       }
     }
    printf("\n");
    for(i=0;i<n;i++)
    {
      tat[i]=ex[i]-at2[i];
      wt[i]=tat[i]-bt[i];
    }
 printf("Process  Arrival-time(s)  Burst-time(s)  Waiting-time(s)  Turnaround-time(s)\n");
   for(i=0;i<n;i++)
    {
      printf("P%d          %d          %d          %d          %d\n",i,at2[i],bt[i],wt[i],tat[i]);
    }
   for(i=0;i<n;i++)
   {
    av1+=tat[i];
    av2+=wt[i];
   }
  printf("Average waiting time(s) %f\nAverage turnaroundtime(s) %f\nCPU idle
time(s)%d\n",av2/n,av1/n,idle);
}
```

Output:

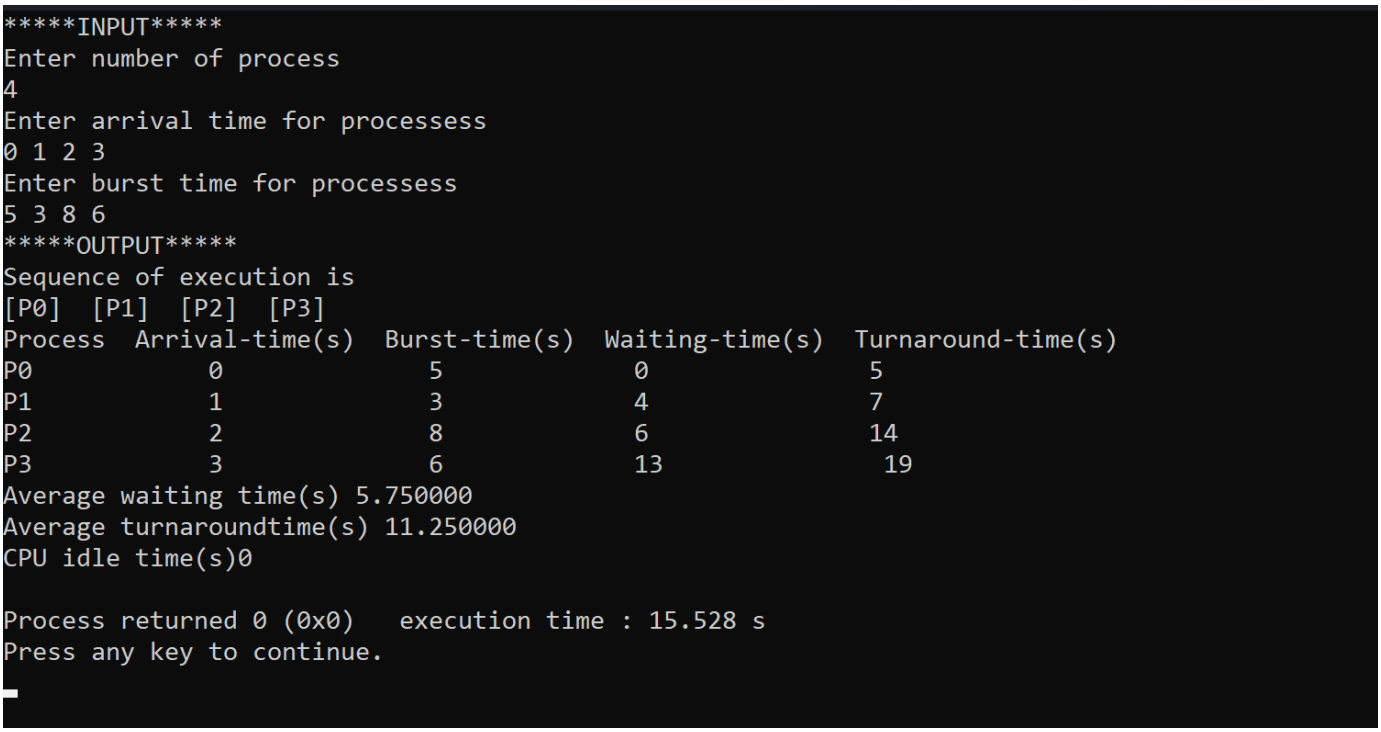
*****INPUT*****
Enter number of process
4
Enter arrival time for processess
0 1 2 3
Enter burst time for processess
5 3 8 6
*****OUTPUT*****
Sequence of execution is
[P0] [P1] [P2] [P3]

Process	Arrival-time(s)	Burst-time(s)	Waiting-time(s)	Turnaround-time(s)
P0	0	5	0	5
P1	1	3	4	7
P2	2	8	6	14
P3	3	6	13	19

Average waiting time(s) 5.750000
Average turnaroundtime(s) 11.250000
CPU idle time(s)0

Process returned 0 (0x0) execution time : 15.528 s
Press any key to continue.

Screenshot:



SJF ALGORITHM

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_PROCESS 100
struct process{
   int pid;
   int burst_time;
   int arrival_time;
   int turn_around_time;
   int wait_time;
   int processed;
};

typedef struct process Process;
void sort_process_by_arrival_time(Process p[], int n)
   {
      int i, j;
      Process temp;
      for( i=0; i<n-1; i++)
      {
         for ( j=0; j<n-i-1; j++)
         {
            if( p[j].arrival_time > p[j+1].arrival_time)
            {
               temp = p[j];
               p[j] = p[j+1];
               p[j+1] = temp;
            }
         }
      }
   }
void print_gantt_chart (Process p[], int n)
   {
      int i, j, timet = 0, end_time[10],sum_bt=0, smallest, current=100;
         for(i=0; i<n; i++)
            {
            end_time[i] = p[i].arrival_time;
            }
         for(i=0; i<n; i++)
         {
            sum_bt += p[i].burst_time;
         }

      for(timet=0; timet<=sum_bt; timet++)
      {
         smallest = 100;
         p[smallest].burst_time = 10;
            for(i=0; i<n; i++)              {
               if(p[i].arrival_time<=timet && p[i].processed!=1 && [i].burst_time<p[smallest].burst_time)
               {
               smallest = i;
               }
            }
```

```
            p[smallest].burst_time -= 1;
            if(current != smallest)
            {
               printf("|%d|", p[smallest].pid);
               p[smallest].wait_time += (timet - end_time[smallest]);
               end_time[smallest] = timet+1;
               current = smallest;
            }

            if(p[smallest].burst_time == 0)
            {
               p[smallest].turn_around_time = (timet+1)-p[smallest].arrival_time;
               p[smallest].processed = 1;
            }
         }
      }
   void print_tat_wait_time(Process p[], int n)
      {
      int i = 0, total_TAT = 0, total_wait_time = 0 ;
      double avg_TAT = 0, avg_wait_time =0;
      printf(" \nJob | TAT | Wait time\n");
         for (i=0 ; i < n; i++)        {
            printf("%d | %d | %d \n", p[i].pid, p[i].turn_around_time,p[i].wait_time);
            total_TAT = total_TAT + p[i].turn_around_time;
            total_wait_time = total_wait_time + p[i].wait_time;
         }
      avg_TAT = (double)total_TAT / (double)n;
      avg_wait_time = (double)total_wait_time /(double) n;
      printf("Average Turn aroud time = %f\n", avg_TAT);
      printf("Average waiting time = %f\n", avg_wait_time);
      }

   int main()
   {
      Process p[MAX_PROCESS];
      int n, i, j;
         printf("Enter total process\n");
         scanf("%d",&n);
         printf("Enter burst time and Arrival time for each process\n");
         for( i=0; i<n; i++)
         {
            printf("p[%d]: Burst time :", i+1);
            scanf("%d",&p[i].burst_time);
            printf("p[%d]: Arrival time :", i+1);
            p[i].pid = i+1;
            scanf("%d",&p[i].arrival_time);
            p[i].wait_time = 0;
         }
      sort_process_by_arrival_time(p,n);
      printf("\nGantt chart:\n");
      print_gantt_chart(p,n);
      print_tat_wait_time( p, n);
   }
```

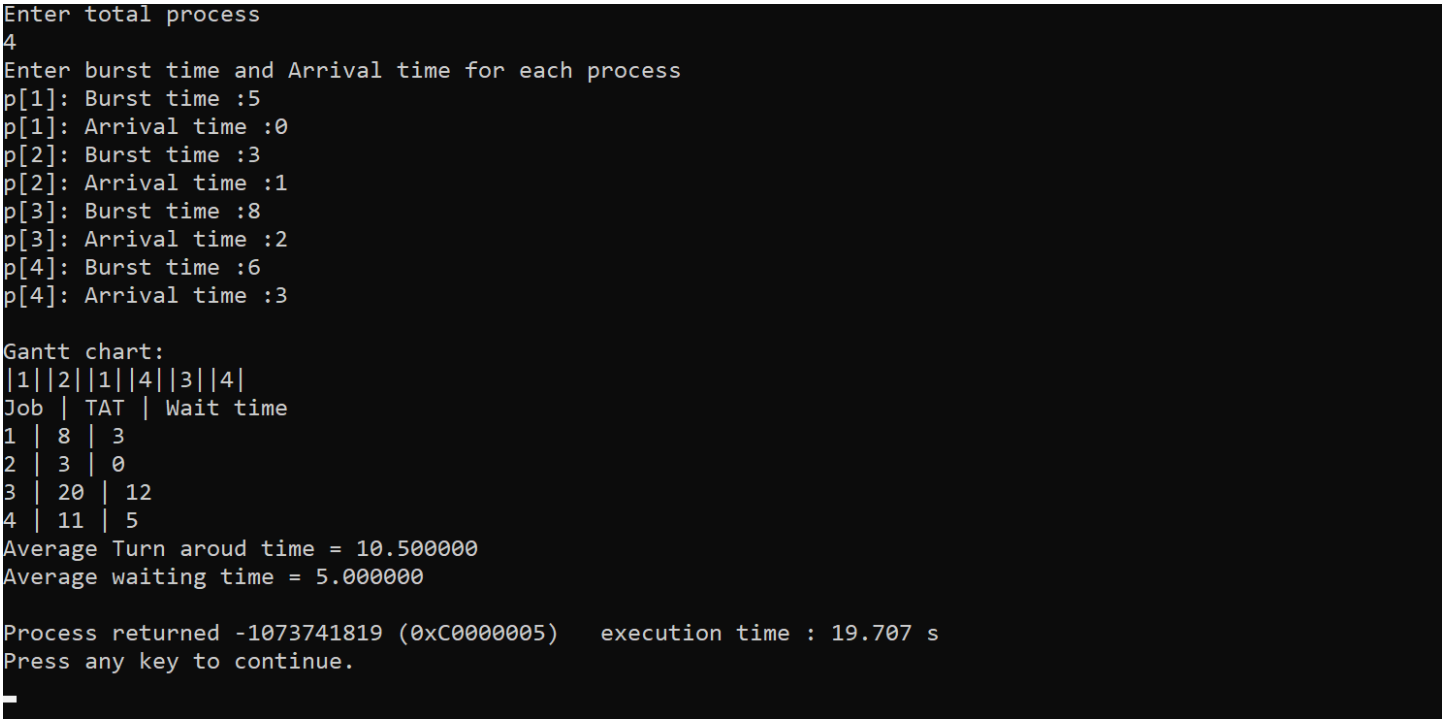
Output:

Enter total process
4
Enter burst time and Arrival time for each process
p[1]: Burst time :5
p[1]: Arrival time :0
p[2]: Burst time :3
p[2]: Arrival time :1
p[3]: Burst time :8
p[3]: Arrival time :2
p[4]: Burst time :6
p[4]: Arrival time :3

Gantt chart:
|1||2||1||4||3||4|
Job | TAT | Wait time
1 | 8 | 3
2 | 3 | 0
3 | 20 | 12
4 | 11 | 5
Average Turn around time = 10.500000
Average waiting time = 5.000000

Process returned -1073741819 (0xC0000005) execution time : 19.707 s
Press any key to continue.

Screenshot:



PRIORITY ALGORITHM

Program:

```
#include<stdio.h>
#include<string.h>
struct process
{
        char pid[5];
        int at,bt,rt,wt,tat,strt_time,finish_time,tbt,prior;
}p[30];
int tottime=0;

void accept(int n)
{
        int i;
        for(i=0;i<n;i++)
        {
        printf("Enter the process name, arrival time,burst time and priority:");
        scanf("%s%d%d%d",p[i].pid,&p[i].at,&p[i].bt,&p[i].prior);
        tottime=tottime+p[i].bt;
   p[i].tbt=p[i].bt;
        }

}



void display(int n)
{
        printf("|PID|\t|AT|\t|BT|\t|ST|\t|FT|\t|RT|\t|WT|\n\n");
   int i;
        for( i=0;i<n;i++)
   {

printf("|%s|\t|%d|\t|%d|\t|%d|\t|%d|\t|%d|\t|%d|\n\n",p[i].pid,p[i].at,p[i].bt,p[i].strt_time,p[i].finish_time,p[i
].rt,p[i].wt);
        }
}

int getprocess(int time1,int n)
{
        int i,min=99,tpid=0,min_at=99;
        for(i=0;i<n;i++)
        {
                if(p[i].at<=time1 && p[i].prior<=min && p[i].bt!=0)
                {
                        if(min==p[i].prior)
                        {
                                if(p[i].at<min_at)
                                {
                                        min=p[i].prior;
                   min_at=p[i].at;
```

```
                        printf("\t\t\tMIN AT %d\n",min_at);
                                tpid=i;
                        }
                }
                else
                {
                        min=p[i].prior;
        tpid=i;
                }
            }
        }
   return tpid;
}


void gantt_chart(int n)
{
        int time1=0,tempid;
   while(time1<tottime)
   {
      tempid=getprocess(time1,n);
            p[tempid].strt_time=time1;
     {
                p[tempid].bt--;
                printf("%d|%s|",time1,p[tempid].pid);
                time1++;
            printf("%d\t",time1);
     }
   p[tempid].finish_time=time1;
   p[tempid].wt=p[tempid].finish_time-p[tempid].tbt-p[tempid].at;
   p[tempid].tat=p[tempid].wt+p[tempid].tbt;
  }
printf("\n\n");
}

void calcuate_avg(int n)
{
        float  tot_wt=0,tot_tat=0;
        int i;
        for(i=0;i<n;i++)
        {
      tot_wt=tot_wt+p[i].wt;
      tot_tat=tot_tat+p[i].tat;
        }
      printf("\nAVERAGE WAIT  TIME: %f\nAVERAGE TURN AROUND TIME
:%f\n",tot_wt/n,tot_tat/n);
}

void main()
{
```

```
        int n;
        do
        {
        printf("Enter the number of process:\n");
    scanf("%d",&n);
          if(n<=20)
              {
                    accept(n);
                    printf("\nGantt chart\n\n");
                    gantt_chart(n);
                    printf("\n");
                    display(n);
                    calcuate_avg(n);
    }
              else
              printf("Please enter process number less than 20!\n" );
              }while(n>20 || n<=0);
}
```

Output:

Enter the number of process:
4
Enter the process name, arrival time,burst time and priority:p1 0 2 8
Enter the process name, arrival time,burst time and priority:p2 1 3 3
Enter the process name, arrival time,burst time and priority:p3 2 1 11
Enter the process name, arrival time,burst time and priority:p4 4 4 6

Gantt chart

0|p1|1 1|p2|2 2|p2|3 3|p2|4 4|p4|5 5|p4|6 6|p4|7 7|p4|8 8|p1|9 9|p3|10

|PID| |AT| |BT| |ST| |FT| |RT| |WT|

|p1| |0| |0| |8| |9| |0| |7|

|p2| |1| |0| |3| |4| |0| |0|

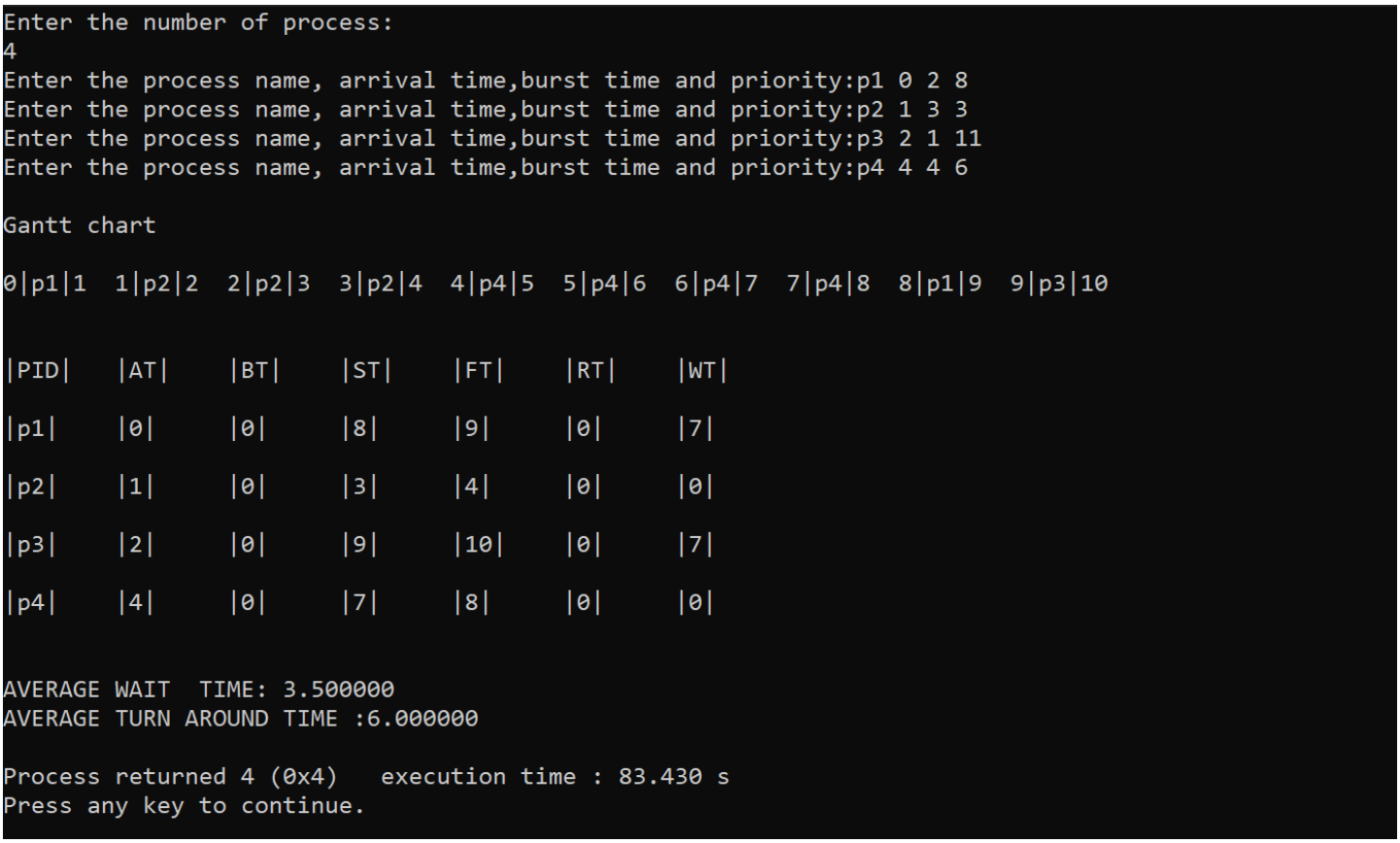
|p3| |2| |0| |9| |10| |0| |7|

|p4| |4| |0| |7| |8| |0| |0|

AVERAGE WAIT TIME: 3.500000
AVERAGE TURN AROUND TIME :6.000000

Process returned 4 (0x4) execution time : 83.430 s
Press any key to continue.

Screenshot:



```
Enter the number of process:
4
Enter the process name, arrival time,burst time and priority:p1 0 2 8
Enter the process name, arrival time,burst time and priority:p2 1 3 3
Enter the process name, arrival time,burst time and priority:p3 2 1 11
Enter the process name, arrival time,burst time and priority:p4 4 4 6

Gantt chart

0|p1|1  1|p2|2  2|p2|3  3|p2|4  4|p4|5  5|p4|6  6|p4|7  7|p4|8  8|p1|9  9|p3|10


|PID|   |AT|    |BT|    |ST|    |FT|    |RT|    |WT|

|p1|    |0|     |0|     |8|     |9|     |0|     |7|

|p2|    |1|     |0|     |3|     |4|     |0|     |0|

|p3|    |2|     |0|     |9|     |10|    |0|     |7|

|p4|    |4|     |0|     |7|     |8|     |0|     |0|


AVERAGE WAIT  TIME: 3.500000
AVERAGE TURN AROUND TIME :6.000000

Process returned 4 (0x4)   execution time : 83.430 s
Press any key to continue.
```

ROUND ROBIN ALGORITHM

Program:

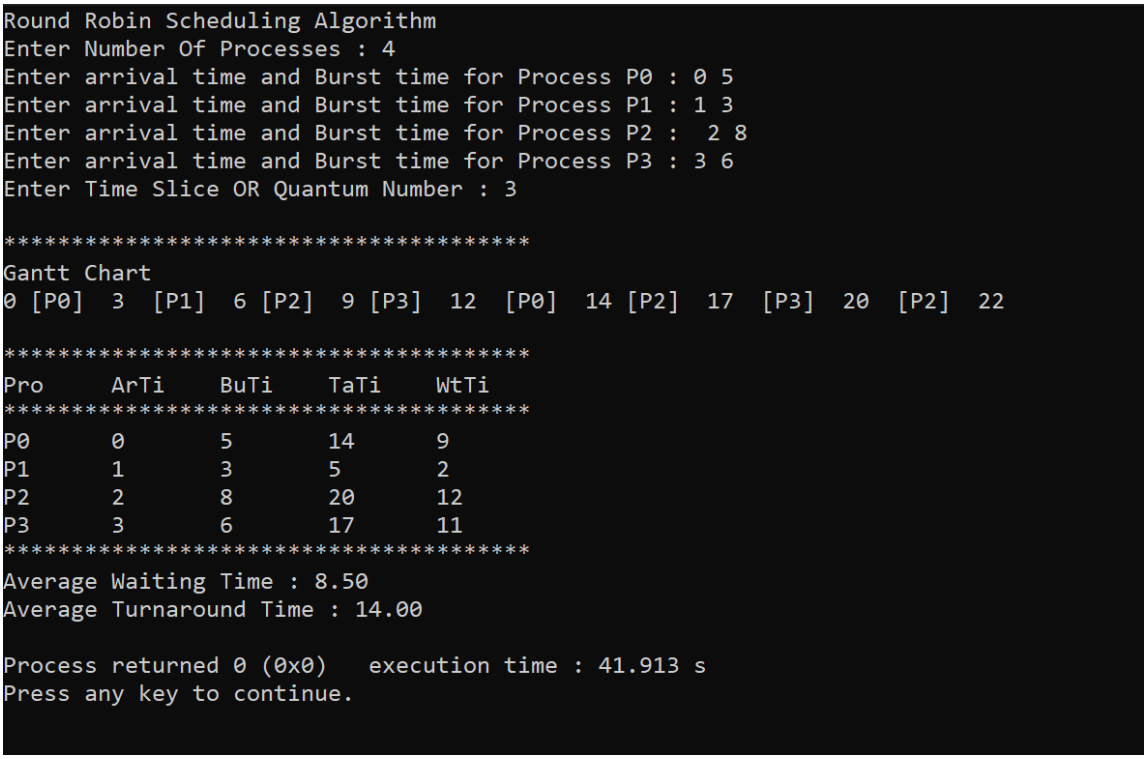
```
#include<stdio.h>

struct times
{
    int p,art,but,wtt,tat,rnt;
};


void sort(struct times a[],int pro)
{
    int i,j;
    struct times temp;
    for(i=0;i<pro;i++)
    {
        for(j=i+1;j<pro;j++)
        {
            if(a[i].art > a[j].art)
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    return;
}

int main()
{
    int i,j,pro,time,remain,flag=0,ts;
    struct times a[100];
    float avgwt=0,avgtt=0;
    printf("Round Robin Scheduling Algorithm\n");
    printf("Enter Number Of Processes : ");
    scanf("%d",&pro);
    remain=pro;
    for(i=0;i<pro;i++)
    {
        printf("Enter arrival time and Burst time for Process P%d : ",i);
        scanf("%d%d",&a[i].art,&a[i].but);
        a[i].p = i;
        a[i].rnt = a[i].but;
    }
    sort(a,pro);
    printf("Enter Time Slice OR Quantum Number : ");
    scanf("%d",&ts);
    printf("\n****************************************\n");
    printf("Gantt Chart\n");
```

```
    printf("0");
    for(time=0,i=0;remain!=0;)
    {
        if(a[i].rnt<=ts && a[i].rnt>0)
        {
            time = time + a[i].rnt;
            printf("  [P%d]  %d",a[i].p,time);
            a[i].rnt=0;
            flag=1;
        }
        else if(a[i].rnt > 0)
        {
            a[i].rnt = a[i].rnt - ts;
            time = time + ts;
            printf(" [P%d]  %d",a[i].p,time);
        }
        if(a[i].rnt==0 && flag==1)
        {
            remain--;
            a[i].tat = time-a[i].art;
            a[i].wtt = time-a[i].art-a[i].but;
            avgwt = avgwt + time-a[i].art-a[i].but;
            avgtt = avgtt + time-a[i].art;
            flag=0;
        }
        if(i==pro-1)
            i=0;
        else if(a[i+1].art <= time)
            i++;
        else
            i=0;
    }
    printf("\n\n");
    printf("*****************************************\n");
    printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n");
    printf("*****************************************\n");
    for(i=0;i<pro;i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\n",a[i].p,a[i].art,a[i].but,a[i].tat,a[i].wtt);
    }
    printf("*****************************************\n");
    avgwt = avgwt/pro;
    avgtt = avgtt/pro;
    printf("Average Waiting Time : %.2f\n",avgwt);
    printf("Average Turnaround Time : %.2f\n",avgtt);
    return 0;
}
```

Output:

```
Round Robin Scheduling Algorithm
Enter Number Of Processes : 4
Enter arrival time and Burst time for Process P0 : 0 5
Enter arrival time and Burst time for Process P1 : 1 3
Enter arrival time and Burst time for Process P2 :  2 8
Enter arrival time and Burst time for Process P3 : 3 6
Enter Time Slice OR Quantum Number : 3

**************************************
Gantt Chart
0 [P0]  3  [P1]  6 [P2]  9 [P3]  12  [P0]  14 [P2]  17  [P3]  20  [P2]  22

**************************************
Pro    ArTi   BuTi   TaTi   WtTi
**************************************
P0     0      5      14     9
P1     1      3      5      2
P2     2      8      20     12
P3     3      6      17     11
**************************************
Average Waiting Time : 8.50
Average Turnaround Time : 14.00

Process returned 0 (0x0)   execution time : 41.913 s
Press any key to continue.
```

Screenshot:



****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 2 **Date of Implementation:** 6. 6. 23

****

Q2. Implement Bankers algorithm and find safe sequence of processes.

Program:

```
#include<stdio.h>
#include<string.h>
struct process
{
   char pnm[10];
   int alloc[10],max[10],need[10];
}p[10];

   int i=0,j=0,s=0,m,n,avail[10];
   char finish[10],safe_seq[10][10];

void accept()
   {
      printf("Enter the available Resources \n");
      for(i=0;i<m;i++)
      {
         scanf("%d",&avail[i]);
      }

      for(i=0;i<n;i++)
      {
         printf("\nEnter process name\n");
         scanf("%s",p[i].pnm);
            for(j=0;j<m;j++)
             {
               printf("Enter allocation \n");
               scanf("%d",&p[i].alloc[j]);
            }
            for(j=0;j<m;j++)
            {
               printf("Enter Max \n");
               scanf("%d",&p[i].max[j]);
            }
      }
   }
void need()
   {
      for(i=0;i<n;i++)
      {
```

```
            for(j=0;j<m;j++)
            {
               p[i].need[j]=p[i].max[j]-p[i].alloc[j];
            }
         }
      }
void display()
   {
      printf("*************************\nAvailable Resources \n");
         for(i=0;i<m;i++)
         {
            printf("\n%d\n",avail[i]);
          }
      printf("\tAllocation Matrix\t\t\tMax Matrix\n");
      for(i=0;i<n;i++)
      {
         printf("\n%s",p[i].pnm);
            for(j=0;j<m;j++)
            {
               printf("\t%d",p[i].alloc[j]);
            }
               printf("\t\t");
            for(j=0;j<m;j++)
            {
               printf("\t%d",p[i].max[j]);
            }
      }
      printf("\n");
   }

void display_need()
   {
      printf("\n\tNeed Matrix\n");
      for(i=0;i<n;i++)
      {
         printf("\n%s",p[i].pnm);
         for(j=0;j<m;j++)
          {
            printf("\t%d",p[i].need[j]);
          }
      }
      printf("\n");
   }

void bankers()
   {
   int work[10],flag=0,fin_flag=0,x=1,i=0,j=0;
      for(j=0;j<m;j++)
      {
         work[j]=avail[j];
      }
   while(x<=2)
      {
         for(i=0;i<n;i++)
```

```
        {
            flag=0;
            if(finish[i]==0)
            {
                for(j=0;j<m;j++)
                {
                    if(work[j]>=p[i].need[j])
                    {
                        flag=1;
                    }
                    else
                    {
                        flag=0;
                        break;
                    }
                }
                if(flag==1)
                {
                    for(j=0;j<m;j++)
                    {
                        work[j]=work[j]+p[i].alloc[j];
                        finish[i]=1;
                    }
                    strcpy(safe_seq[s++],p[i].pnm);

                }
            }
        }//end of for
        x++;
    }//end of while
printf("\nFor process: %s\n",p[i].pnm);

for(i=0;i<n;i++)
{
    if(finish[i]==1)
    {
        fin_flag=1;
    }
    else
    {
        fin_flag=0;
        break;
    }
}

if(fin_flag==1)
{
    printf("System is in safe state \n");

    printf("Safe sequence \n");
        for(i=0;i<s;i++)
        {
            printf("%s\t",safe_seq[i]);
        }
```

```
            printf("\n work\n");
            for(i=0;i<m;i++)
            {
               printf("%d\t",work[i]);
            }
         printf("\n");
      }
      else
      {
         printf("System is in unsafe state \n");
      }
   }

   void main()
   {
      printf("Enter number of processes:\n");
         scanf("%d",&n);
      printf("Enter number of resources:\n");
         scanf("%d",&m);
         accept();
         display();
         need();
         display_need();
         bankers();
   }
```

Output:

```
Enter number of processes:
5
Enter number of resources:
3
Enter the available Resources
3 3 2

Enter process name
P0
Enter allocation
0
Enter allocation
1
Enter allocation
0
Enter Max
7
Enter Max
5
Enter Max
3

Enter process name
P1
Enter allocation
2
Enter allocation
```

0
Enter allocation
0
Enter Max
3
Enter Max
2
Enter Max
2

Enter process name
P2
Enter allocation
3
Enter allocation
0
Enter allocation
2
Enter Max
9
Enter Max
0
Enter Max
2

Enter process name
P3
Enter allocation
2
Enter allocation
1
Enter allocation
1
Enter Max
2
Enter Max
2
Enter Max
2

Enter process name
P4
Enter allocation
0
Enter allocation
0
Enter allocation
2
Enter Max
4
Enter Max
3
Enter Max
3

****
Available Resources

3

3

2

	Allocation Matrix			Max Matrix		
P0	0	1	0	7	5	3
P1	2	0	0	3	2	2
P2	3	0	2	9	0	2
P3	2	1	1	2	2	2
P4	0	0	2	4	3	3

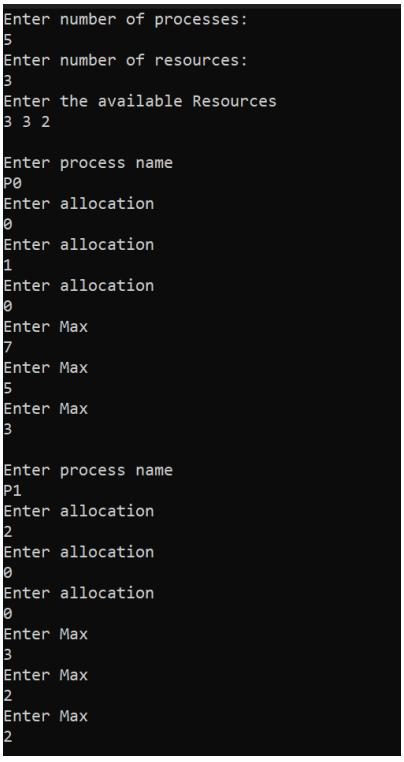
Need Matrix

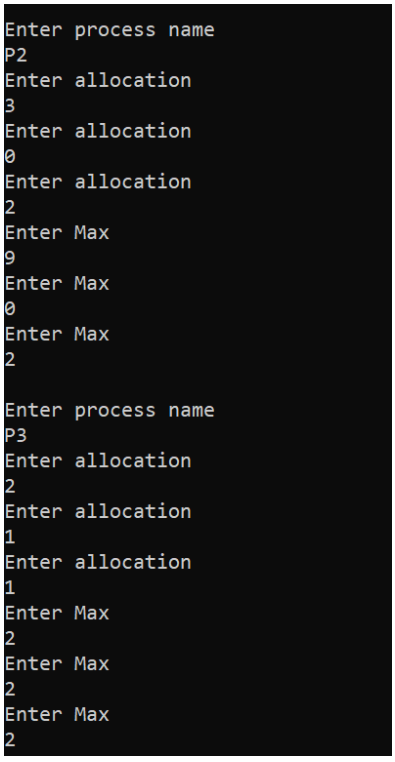
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

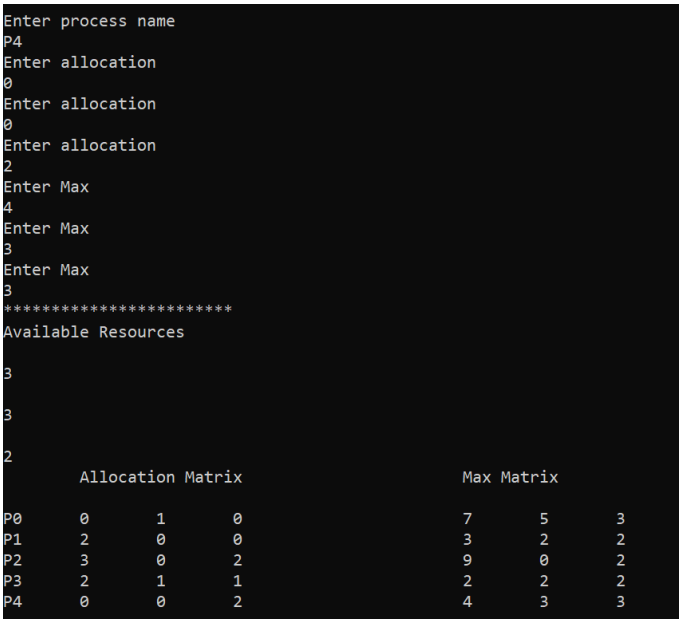
For process:
System is in safe state
Safe sequence
P1 P3 P4 P0 P2
work
10 5 7

Process returned 10 (0xA) execution time : 82.803 s
Press any key to continue.

Screenshot:







```
        Allocation Matrix                       Max Matrix

P0      0       1       0                       7       5       3
P1      2       0       0                       3       2       2
P2      3       0       2                       9       0       2
P3      2       1       1                       2       2       2
P4      0       0       2                       4       3       3

        Need Matrix

P0      7       4       3
P1      1       2       2
P2      6       0       0
P3      0       1       1
P4      4       3       1

For process:
System is in safe state
Safe sequence
P1      P3      P4      P0      P2
 work
10      5       7

Process returned 10 (0xA)   execution time : 82.803 s
Press any key to continue.
```

****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 2 **Date of Implementation:** 6. 6. 23

****

Q3. Write a C-program to implement the producer – consumer problem using semaphores.

Program:

```
#include<stdio.h>
#include<conio.h>
int S=1,full=0,empty=3,x=0;
int main()
  {
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
      printf("\nEnter your choice:");
      scanf("%d",&n);
      switch(n)
       {
         case 1:
           if((S==1)&&(empty!=0))
             producer();
           else
             printf("Buffer is full!!");
           break;

         case 2:
           if((S==1)&&(full!=0))
             consumer();
           else
             printf("Buffer is empty!!");
           break;

         case 3:
           exit(0);
           break;
       }
     }
    return 0;
  }
```

```
int wait(int s)
   {
      return (--s);
   }

int signal(int s)
   {
      return(++s);
   }

void producer()
   {
      S=wait(S);
      full=signal(full);
      empty=wait(empty);
      x++;
      printf("\nProducer produces item %d",x);
      S=signal(S);
   }

void consumer()
   {
      S=wait(S);
      full=wait(full);
      empty=signal(empty);
      printf("\nConsumer consumes item %d",x);
      x--;
      S=signal(S);
   }
```

Output:

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
```

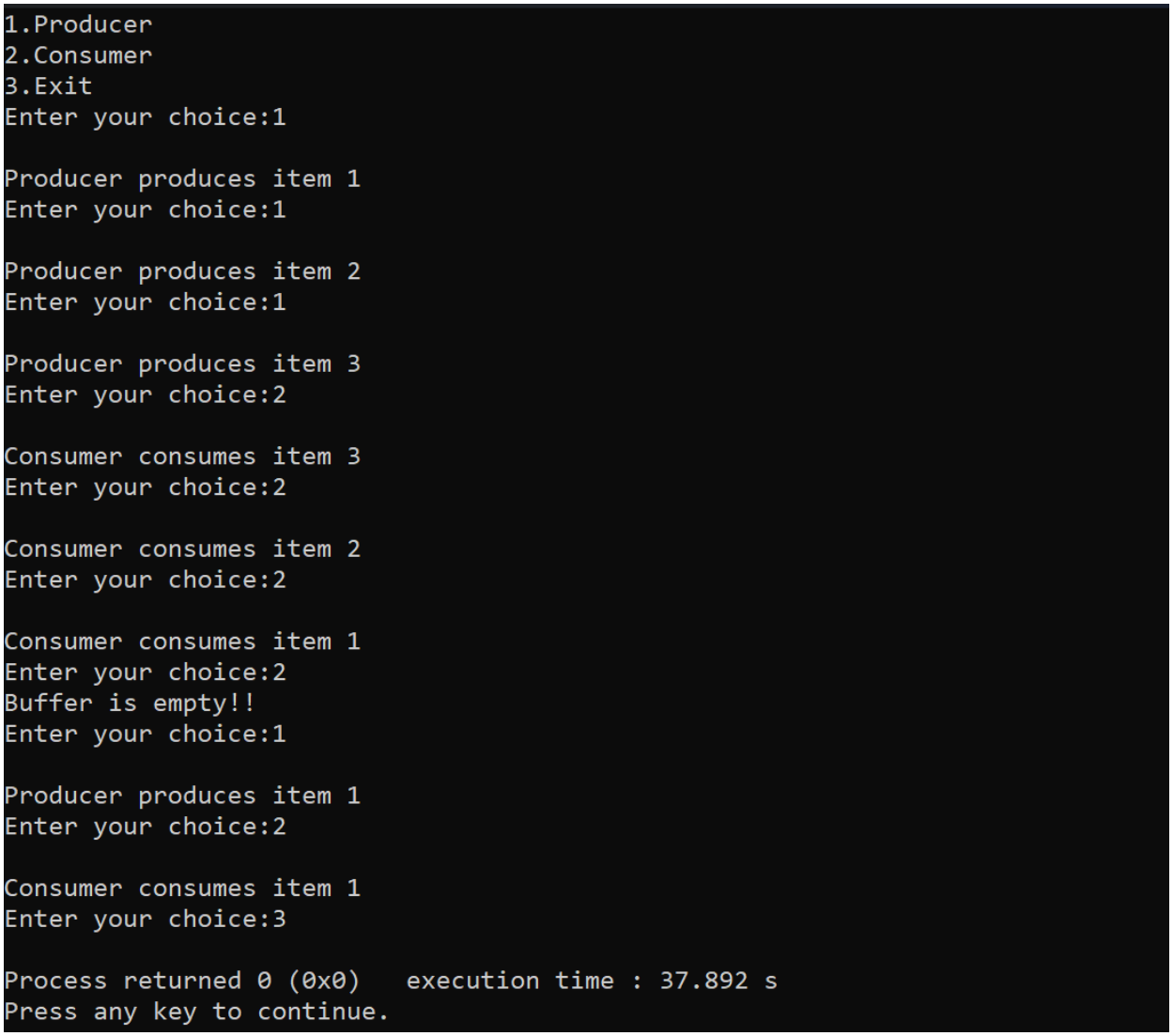
Buffer is empty!!
Enter your choice:1

Producer produces item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:3

Process returned 0 (0x0)   execution time : 37.892 s
Press any key to continue.

Screenshot:



****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 3 **Date of Implementation:** 22. 5. 23

****

Q. Implement multithreading for Matrix Multiplication using threads.

Program:

```
#include<stdio.h>
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
#define MAX 10


void *mult(void* arg){
   int *data = (int *)arg;
   int k = 0, i = 0;

   int x = data[0];
   for (i = 1; i <= x; i++)
      k += data[i]*data[i+x];

   int *p = (int*)malloc(sizeof(int));
      *p = k;
   pthread_exit(p);
}

int main(){

   int matA[MAX][MAX];
   int matB[MAX][MAX];


   int r1, c1, r2, c2, i, j, k, max;

      printf("Enter no.of rows: ");
      scanf("%d", &r1);

      printf("Enter no.of columns: ");
      scanf("%d", &c1);

   printf("Matrix A input\n");
      for (i = 0; i < r1; i++){
```

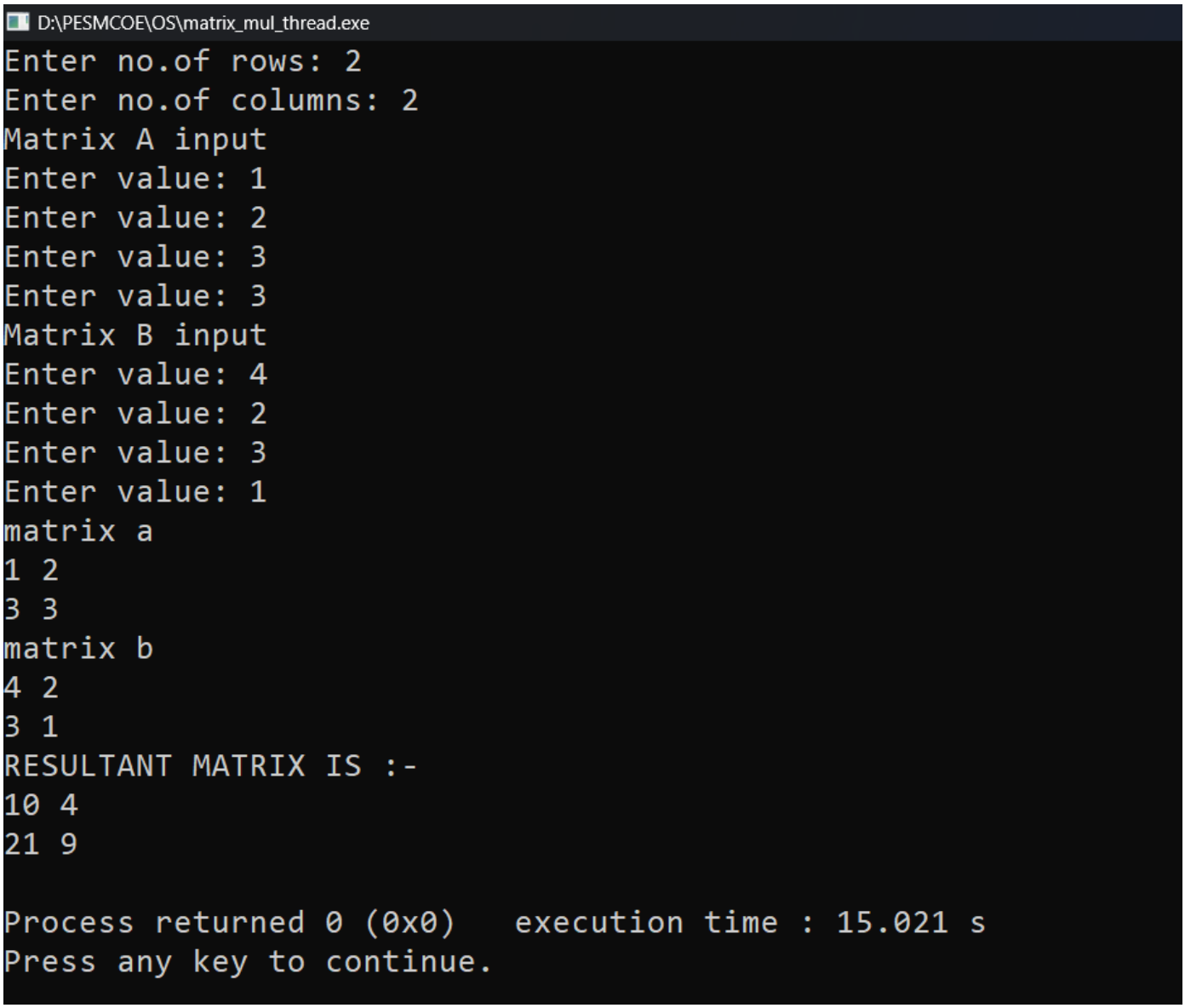
```
        for (j = 0; j < c1; j++){
            printf("Enter value: ");
            scanf("%d",&matA[i][j]);
        }
    }


printf("Matrix B input\n");
    for (i = 0; i < r1; i++){
        for (j = 0; j < c1; j++){
            printf("Enter value: ");
            scanf ("%d",&matB[i][j]);
        }
    }

printf("matrix a\n");
    for (i = 0; i < r1; i++){
        for(j = 0; j < c1; j++)
        printf("%d ",matA[i][j]);
        printf("\n");
    }

printf("matrix b\n");
    for (i = 0; i < r1; i++){
        for(j = 0; j < c1; j++)
        printf("%d ",matB[i][j]);
        printf("\n");
    }


max = r1*c1;

pthread_t *threads;
threads = (pthread_t*)malloc(max*sizeof(pthread_t));

int count = 0;
int* data = NULL;
for (i = 0; i < r1; i++){
    for (j = 0; j < c1; j++){

        data = (int *)malloc((20)*sizeof(int));
        data[0] = c1;

        for (k = 0; k < c1; k++)
            data[k+1] = matA[i][k];

        for (k = 0; k < r1; k++)
            data[k+c1+1] = matB[k][j];

        pthread_create(&threads[count++], NULL,
```

```
            mult, (void*)(data));

        }
    }

    printf("RESULTANT MATRIX IS :- \n");
    for (i = 0; i < max; i++){
        void *k;

        pthread_join(threads[i], &k);


        int *p = (int *)k;
            printf("%d ",*p);
        if ((i + 1) % c1 == 0)
            printf("\n");
    }

    return 0;
}
```

Output:

```
Enter no.of rows: 2
Enter no.of columns: 2
Matrix A input
Enter value: 1
Enter value: 2
Enter value: 3
Enter value: 3
Matrix B input
Enter value: 4
Enter value: 2
Enter value: 3
Enter value: 1
matrix a
1 2
3 3
matrix b
4 2
3 1
RESULTANT MATRIX IS :-
10 4
21 9

Process returned 0 (0x0)   execution time : 15.021 s
```

Screenshot:


D:\PESMCOE\OS\matrix_mul_thread.exe
Enter no.of rows: 2
Enter no.of columns: 2
Matrix A input
Enter value: 1
Enter value: 2
Enter value: 3
Enter value: 3
Matrix B input
Enter value: 4
Enter value: 2
Enter value: 3
Enter value: 1
matrix a
1 2
3 3
matrix b
4 2
3 1
RESULTANT MATRIX IS :-
10 4
21 9
Process returned 0 (0x0) execution time : 15.021 s
Press any key to continue.

****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 4 **Date of Implementation:** 12. 6. 23

****

Q1. Write a program to implement FIFO, LRU, Optimal Page.

FIFO

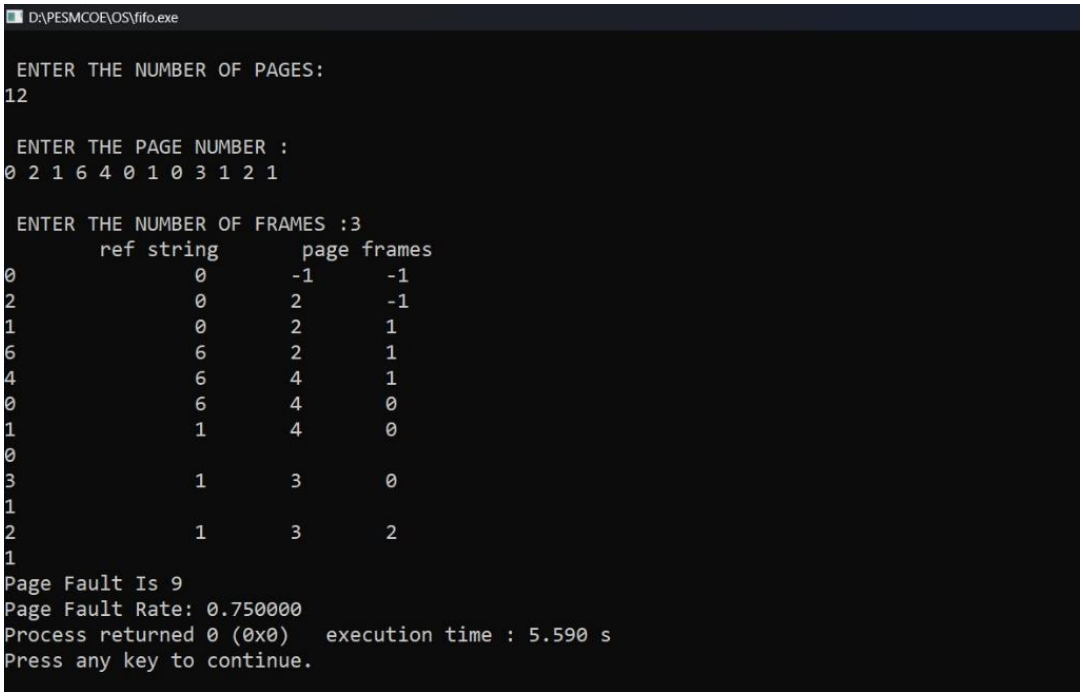
Program:

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
float rate;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++){
                        printf("%d\t\t",a[i]);
                        avail=0;
                        for(k=0;k<no;k++)

                if(frame[k]==a[i])
                                avail=1;

                if (avail==0)
                        {
                                frame[j]=a[i];
                                j=(j+1)%no;
                                count++;
                                for(k=0;k<no;k++)
                                printf("%d\t",frame[k]);
                        }
                        printf("\n");

}
                rate = (float)count/(float)n;
                printf("Page Fault Is %d",count);
                printf("\nPage Fault Rate: %f",rate);
                return 0;
}
```

Output:

```
 ENTER THE NUMBER OF PAGES:
12

 ENTER THE PAGE NUMBER :
0 2 1 6 4 0 1 0 3 1 2 1

 ENTER THE NUMBER OF FRAMES :3
       ref string      page frames
0              0       -1      -1
2              0       2       -1
1              0       2       1
6              6       2       1
4              6       4       1
0              6       4       0
1              1       4       0
0
3              1       3       0
1
2              1       3       2
1
Page Fault Is 9
Page Fault Rate: 0.750000
```

Screenshot:



****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 4 **Date of Implementation:** 12. 6. 23

****

LRU

Program:

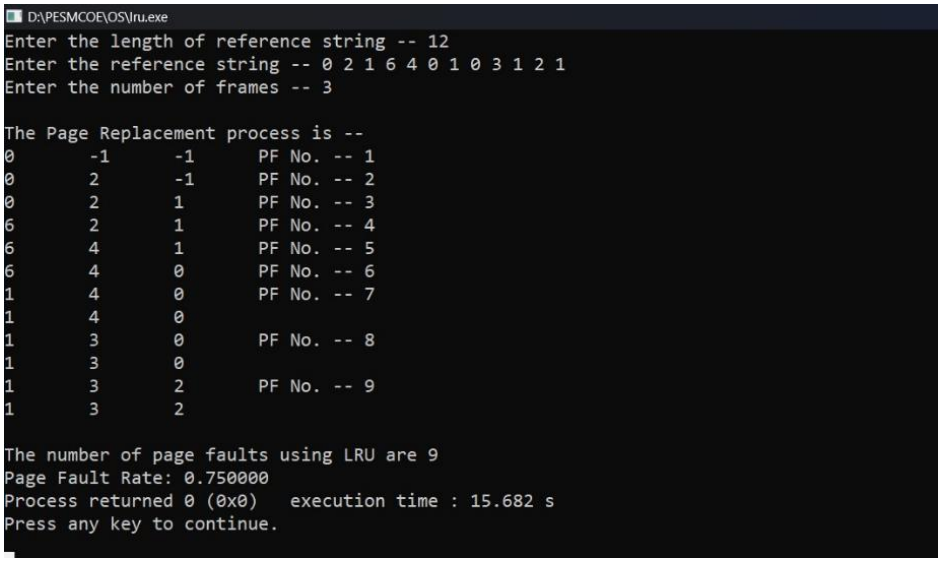
```
#include<stdio.h>
#include<conio.h>
int main(){
    int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
    float rate;
    printf("Enter the length of reference string -- ");
    scanf("%d",&n);
    printf("Enter the reference string -- ");
     for(i=0;i<n;i++){
          scanf("%d",&rs[i]);
          flag[i]=0;
       }
    printf("Enter the number of frames -- ");
    scanf("%d",&f);

    for(i=0;i<f;i++){
       count[i]=0;
       m[i]=-1;
    }
    printf("\nThe Page Replacement process is -- \n");
    for(i=0;i<n;i++){
       for(j=0;j<f;j++){
          if(m[j]==rs[i]){
             flag[i]=1;
             count[j]=next;
             next++;
          }
       }
       if(flag[i]==0){

          if(i<f){
             m[i]=rs[i];
             count[i]=next;
             next++;
          }
          else{
             min=0;
             for(j=1;j<f;j++)
             if(count[min] > count[j])
                min=j;

             m[min]=rs[i];
             count[min]=next;
             next++;
```

```
            }
               pf++;
            }

            for(j=0;j<f;j++)
               printf("%d\t", m[j]);
            if(flag[i]==0)
               printf("PF No. -- %d" , pf);
               printf("\n");
         }
         rate = (float)pf/(float)n;
         printf("\nThe number of page faults using LRU are %d",pf);
         printf("\nPage Fault Rate: %f",rate);
        return 0;
}
```

Output:

```
Enter the length of reference string -- 12
Enter the reference string -- 0 2 1 6 4 0 1 0 3 1 2 1
Enter the number of frames -- 3

The Page Replacement process is --
0       -1      -1      PF No. -- 1
0       2       -1      PF No. -- 2
0       2       1       PF No. -- 3
6       2       1       PF No. -- 4
6       4       1       PF No. -- 5
6       4       0       PF No. -- 6
1       4       0       PF No. -- 7
1       4       0
1       3       0       PF No. -- 8
1       3       0
1       3       2       PF No. -- 9
1       3       2

The number of page faults using LRU are 9
Page Fault Rate: 0.750000
```

Screenshot:



****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 4 **Date of Implementation:** 12. 6. 23

****

OPTIMAL PAGE

Program:

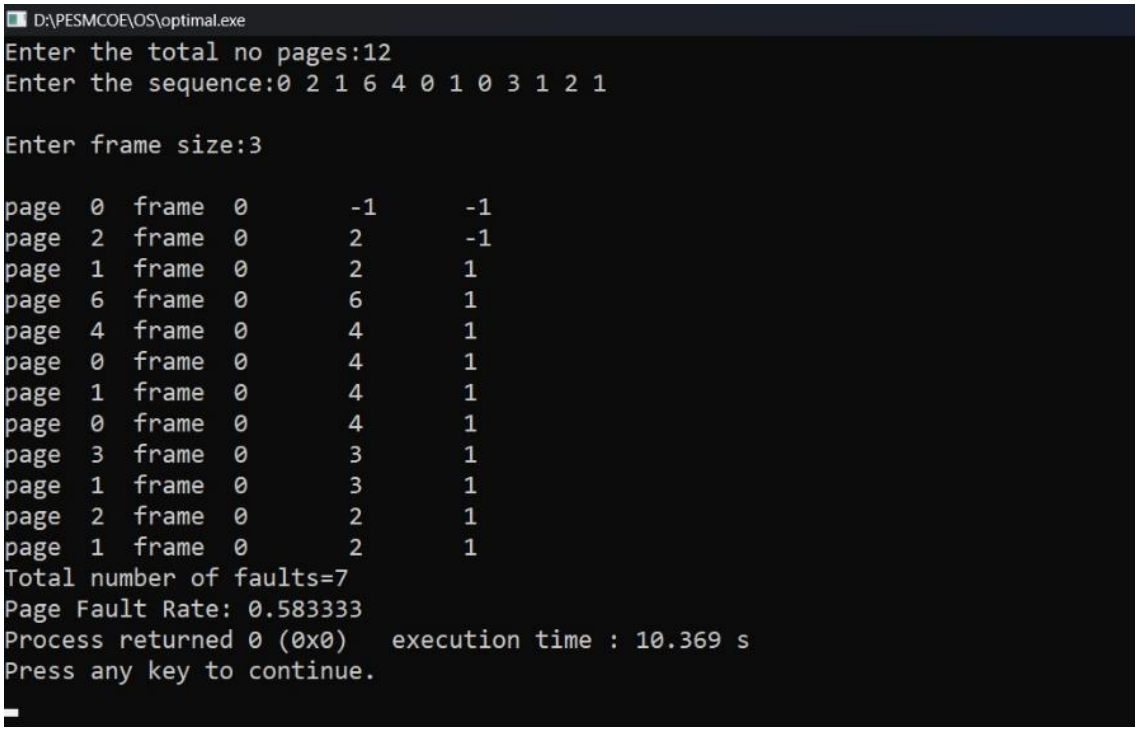
```
#include<stdio.h>
int main()
{
   int n,pg[30],fr[10];
   int count[10],i,j,k,fault,f,flag,temp,current,c,dist,max,m,cnt,p,x;
   float rate;
      fault=0;
      dist=0;
      k=0;

      printf("Enter the total no pages:");
      scanf("%d",&n);
      printf("Enter the sequence:");

         for(i=0;i<n;i++){
            scanf("%d",&pg[i]);
         }

      printf("\nEnter frame size:");
      scanf("%d",&f);

      for(i=0;i<f;i++){
         count[i]=0;
         fr[i]=-1;
      }
   for(i=0;i<n;i++){
      flag=0;
      temp=pg[i];
      for(j=0;j<f;j++)
      {
         if(temp==fr[j])
         {
            flag=1;
            break;
         }
      }
      if((flag==0)&&(k<f)){
         fault++;
         fr[k]=temp;
         k++;
      }
      else if((flag==0)&&(k==f)){
         fault++;
         for(cnt=0;cnt<f;cnt++){
```

```
                current=fr[cnt];
                for(c=i;c<n;c++){
                    if(current!=pg[c])
                        count[cnt]++;
                    else
                        break;
                }
            }
            max=0;

            for(m=0;m<f;m++){
                if(count[m]>max){
                    max=count[m];
                    p=m;
                }
            }
            fr[p]=temp;
        }

        printf("\npage  %d  frame\t",pg[i]);

        for(x=0;x<f;x++){
            printf("%d\t",fr[x]);
        }
    }
    rate = fault/(float)n;
    printf("\nTotal number of faults=%d",fault);
    printf("\nPage Fault Rate: %f",rate);

    return 0;
}
```

Output:

```
Enter the total no pages:12
Enter the sequence:0 2 1 6 4 0 1 0 3 1 2 1

Enter frame size:3

page  0  frame  0      -1     -1
page  2  frame  0      2      -1
page  1  frame  0      2      1
page  6  frame  0      6      1
page  4  frame  0      4      1
page  0  frame  0      4      1
page  1  frame  0      4      1
page  0  frame  0      4      1
page  3  frame  0      3      1
page  1  frame  0      3      1
page  2  frame  0      2      1
page  1  frame  0      2      1
Total number of faults=7
Page Fault Rate: 0.583333
```

Screenshot:


D:\PESMCOE\OS\optimal.exe
Enter the total no pages:12
Enter the sequence:0 2 1 6 4 0 1 0 3 1 2 1
Enter frame size:3
page 0 frame 0 -1 -1
page 2 frame 0 2 -1
page 1 frame 0 2 1
page 6 frame 0 6 1
page 4 frame 0 4 1
page 0 frame 0 4 1
page 1 frame 0 4 1
page 0 frame 0 4 1
page 3 frame 0 3 1
page 1 frame 0 3 1
page 2 frame 0 2 1
page 1 frame 0 2 1
Total number of faults=7
Page Fault Rate: 0.583333
Process returned 0 (0x0) execution time : 10.369 s
Press any key to continue.

****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 4 **Date of Implementation:** 12. 6. 23

****

Q2. First Fit, Worst Fit, Best Fit

FIRST FIT

Program:

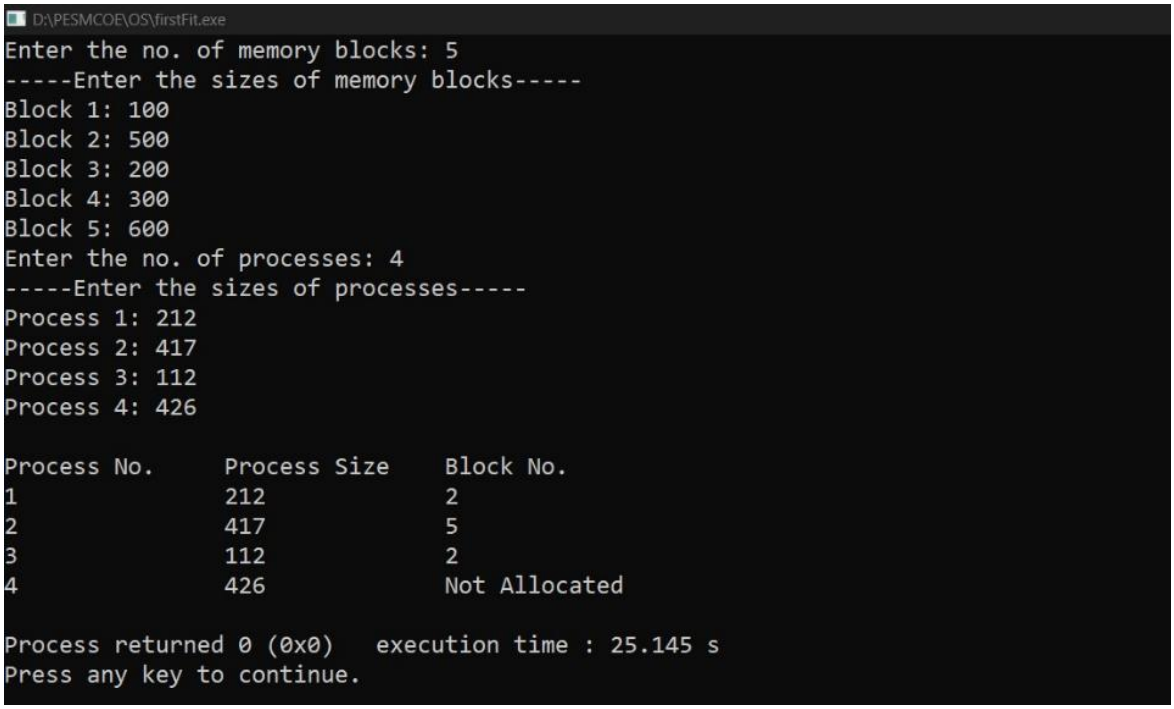
```
#include <stdio.h>
void firstFit(int blockSize[], int m, int processSize[], int n){
    int i,j,allocation[n];

    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;
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock No.\n");

    for (i = 0; i < n; 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");
    }
}
int main(){
    int m, n,i;
      printf("Enter the no. of memory blocks: ");
      scanf("%d", &m);
    int blockSize[m];
      printf("-----Enter the sizes of memory blocks-----\n");

       for (i = 0; i < m; i++){
          printf("Block %d: ", i + 1);
          scanf("%d", &blockSize[i]);
        }

      printf("Enter the no. of processes: ");
      scanf("%d", &n);
```

```
    int processSize[n];

      printf("-----Enter the sizes of processes-----\n");

      for (i = 0; i < n; i++){
        printf("Process %d: ", i + 1);
        scanf("%d", &processSize[i]);
      }

    firstFit(blockSize, m, processSize, n);
  return 0;
}
```

Output:

```
Enter the no. of memory blocks: 5
-----Enter the sizes of memory blocks-----
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the no. of processes: 4
-----Enter the sizes of processes-----
Process 1: 212
Process 2: 417
Process 3: 112
Process 4: 426

Process No.     Process Size    Block No.
1               212             2
2               417             5
3               112             2
4               426             Not Allocated
```

Screenshot:



****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 4 **Date of Implementation:** 12. 6. 23

****

WORST FIT

Program:

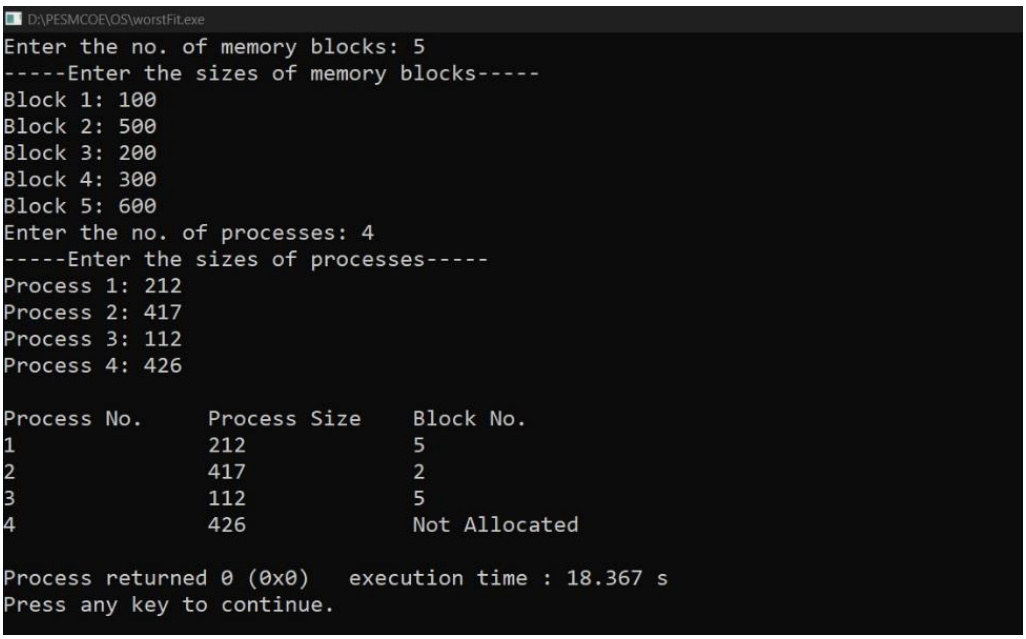
```
#include <stdio.h>
void worstFit(int blockSize[], int m, int processSize[], int n){
    int i,j,allocation[n];

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

    for (i = 0; i < n; i++){
        int worstIndex = -1;
        for (j = 0; j < m; j++)
       {
        if (blockSize[j] >= processSize[i])
       {
        if (worstIndex == -1 || blockSize[j] > blockSize[worstIndex])
        worstIndex = j;
        }
        }
        if (worstIndex != -1)
       {
        allocation[i] = worstIndex;
        blockSize[worstIndex] -= processSize[i];
        }
    }
    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for (i = 0; i < n; 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");
    }
}

int main(){
    int i,m, n;
      printf("Enter the no. of memory blocks: ");
      scanf("%d", &m);

    int blockSize[m];
      printf("-----Enter the sizes of memory blocks-----\n");

    for (i = 0; i < m; i++){
```

```
        printf("Block %d: ", i + 1);
        scanf("%d", &blockSize[i]);
    }
        printf("Enter the no. of processes: ");
        scanf("%d", &n);
    int processSize[n];
      printf("-----Enter the sizes of processes-----\n");

    for (i = 0; i < n; i++){
        printf("Process %d: ", i + 1);
        scanf("%d", &processSize[i]);
    }
    worstFit(blockSize, m, processSize, n);
    return 0;
}
```

Output:

```
Enter the no. of memory blocks: 5
-----Enter the sizes of memory blocks-----
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the no. of processes: 4
-----Enter the sizes of processes-----
Process 1: 212
Process 2: 417
Process 3: 112
Process 4: 426

Process No.     Process Size    Block No.
1               212             5
2               417             2
3               112             5
4               426             Not Allocated
```

Screenshot:



****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 4 **Date of Implementation:** 12. 6. 23

****

BEST FIT

Program:

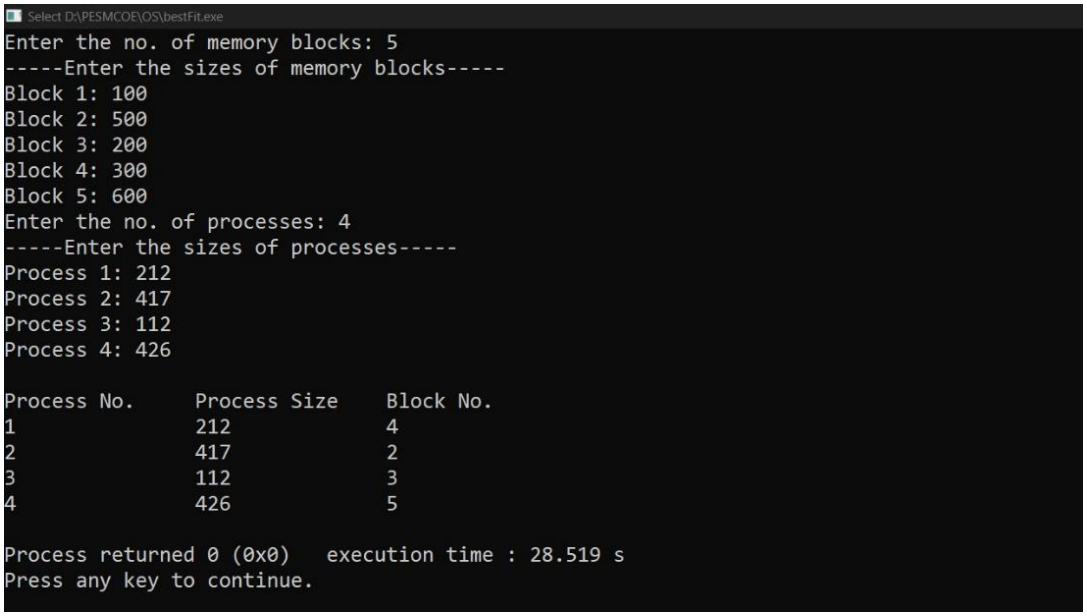
```
#include <stdio.h>
void bestFit(int blockSize[], int m, int processSize[], int n)
{
    int i,j,allocation[n];
    for (i = 0; i < n; i++){
      allocation[i] = -1;
    }

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

    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for (i = 0; i < n; 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");
    }
}
int main()
{
    int i,m, n;
        printf("Enter the no. of memory blocks: ");
        scanf("%d", &m);
    int blockSize[m];
      printf("-----Enter the sizes of memory blocks-----\n");

        for (i = 0; i < m; i++){
            printf("Block %d: ", i + 1);
```

```
            scanf("%d", &blockSize[i]);
        }
        printf("Enter the no. of processes: ");
        scanf("%d", &n);
    int processSize[n];
      printf("-----Enter the sizes of processes-----\n");

        for (i = 0; i < n; i++){
            printf("Process %d: ", i + 1);
            scanf("%d", &processSize[i]);
        }

    bestFit(blockSize, m, processSize, n);
    return 0;
}
```

Output:

```
Enter the no. of memory blocks: 5
-----Enter the sizes of memory blocks-----
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the no. of processes: 4
-----Enter the sizes of processes-----
Process 1: 212
Process 2: 417
Process 3: 112
Process 4: 426

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

Screenshot:



Progressive Education Society's

Modern College of Engineering, Pune

MCA Department

A.Y.2022-23

(310918) Operating System Laboratory

****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 5 **Date of Implementation:** 26. 6. 23

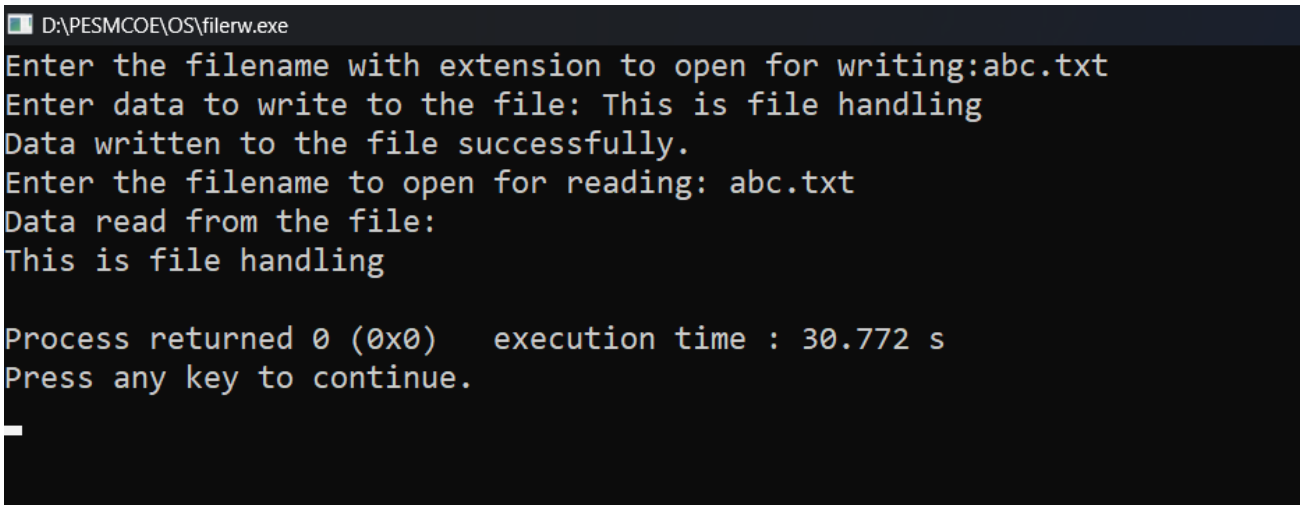
****

Q1. Implement File Handling System Calls to read write and open file.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *file;
    char filename[100],data[100];
      printf("Enter the filename with extension to open for writing:");
      scanf("%s", filename);
    file = fopen(filename, "w");
      if (file == NULL){
         printf("Error opening the file.\n");
         exit(1);
      }
      printf("Enter data to write to the file: ");
      scanf(" %[^\n]s", data);
      fprintf(file, "%s\n", data);
      printf("Data written to the file successfully.\n");
      fclose(file);
      printf("Enter the filename to open for reading: ");
      scanf("%s", filename);
      file = fopen(filename, "r");

      if (file == NULL{
      printf("Error opening the file.\n");
      exit(1);
      }
     printf("Data read from the file:\n");

    while (fgets(data, sizeof(data), file) != NULL){
      printf("%s", data);
    }
    fclose(file);
    return 0;
}
```

Screenshot:


D:\PESMCOE\OS\filerw.exe
Enter the filename with extension to open for writing:abc.txt
Enter data to write to the file: This is file handling
Data written to the file successfully.
Enter the filename to open for reading: abc.txt
Data read from the file:
This is file handling
Process returned 0 (0x0)   execution time : 30.772 s
Press any key to continue.

****

Class: FY-MCA **Shift / Div:** A **Batch:** F2 **Roll Number:** 51043

Name: Vanessa Reetu Prashant More **Assignment No:** 6 **Date of Implementation:** 03. 7. 23

****

Q1. Implement an assignment using File Handling System Calls (Low level system calls like open, read, write, etc).

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define BUFFER_SIZE 1024
int main()
{
    char sourceFile[100], destFile[100];
    int sourceFd, destFd;
    char buffer[BUFFER_SIZE];
    ssize_t bytesRead, bytesWritten;
    printf("Enter source file name: ");
    scanf("%s", sourceFile);
    sourceFd = open(sourceFile, O_RDONLY);

      if (sourceFd == -1){
        perror("Failed to open source file");
        exit(EXIT_FAILURE);
      }

    printf("Enter destination file name: ");
    scanf("%s", destFile);
    destFd = open(destFile, O_WRONLY | O_CREAT | O_TRUNC);

      if (destFd == -1){
        perror("Failed to create destination file");
        exit(EXIT_FAILURE);
      }
```

```
    while ((bytesRead = read(sourceFd, buffer, BUFFER_SIZE)) > 0){
        bytesWritten = write(destFd, buffer, bytesRead);
        if (bytesWritten != bytesRead){
            perror("Failed to write to destination file");
            exit(EXIT_FAILURE);
        }
    }

    if (bytesRead == -1){
        perror("Failed to read from source file");
        exit(EXIT_FAILURE);
    }

    if (close(sourceFd) == -1){
        perror("Failed to close source file");
        exit(EXIT_FAILURE);
    }

    if (close(destFd) == -1){
        perror("Failed to close destination file");
        exit(EXIT_FAILURE);
    }
      printf("File copied successfully!\n");
    return 0;
}
```

Screenshot: