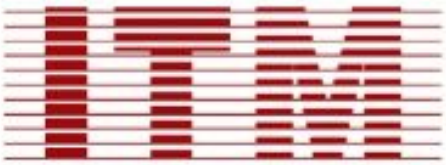


INSTITUTE OF TECHNOLOGY
& MANAGEMENT
GWALIOR • MP • INDIA
"THINK BIG... THINK BEYOND"

Practical File
CS-405
Operating System

SUBMITTED TO:-

NAME OF FACULTY:- Mrs. Aruna Bajpai

Designation:- Asst. Professor, Dept. of CSE

Department name:- CSE

SUBMITTED BY:-

NAME OF STUDENT:- TANYA GUPTA

Roll number:- 0905CS191184

Section (semester):-CSE(D)/ 4th Sem.

INDEX

S.No.	Experiment Name	Date of conduction	Date of submission
1.	Write a program to implement FCFS CPU scheduling algorithm.	26-03-21	21-06-21
2.	Write a program to implement SJF CPU scheduling Algorithm	20-04-21	21-06-21
3.	Write a program to implement SRTF CPU Scheduling algorithm	20-04-21	21-06-21
4.	Write a program to implement Priority CPU Scheduling algorithm.	10-05-21	21-06-21
5.	Write a program to implement preemptive priority CPU Scheduling algorithm	17-05-21	21-06-21
6.	Write a program to implement round robin CPU Scheduling algorithm.	29-05-21	21-06-21
7.	Write a program to implement producer consumer problem	03-05-21	21-06-21
8.	Write a program to implement classical inter process communication problem Reader Writers.	03-06-21	21-06-21
9.	Write a program to implement FIFO page replacement algorithm.	08-06-21	21-06-21
10.	Write a program to implement LRU page replacement algorithm	08-06-21	21-06-21
11.	Write a program to implement Banker's algorithms	17-06-21	21-06-21
12.	Write a program to implement FCFS disk scheduling algorithm.	19-06-21	21-06-21
13.	Write a program to implement SSTF disk scheduling algorithm.	19-06-21	21-06-21

Tanya Gupta
20905CS191184

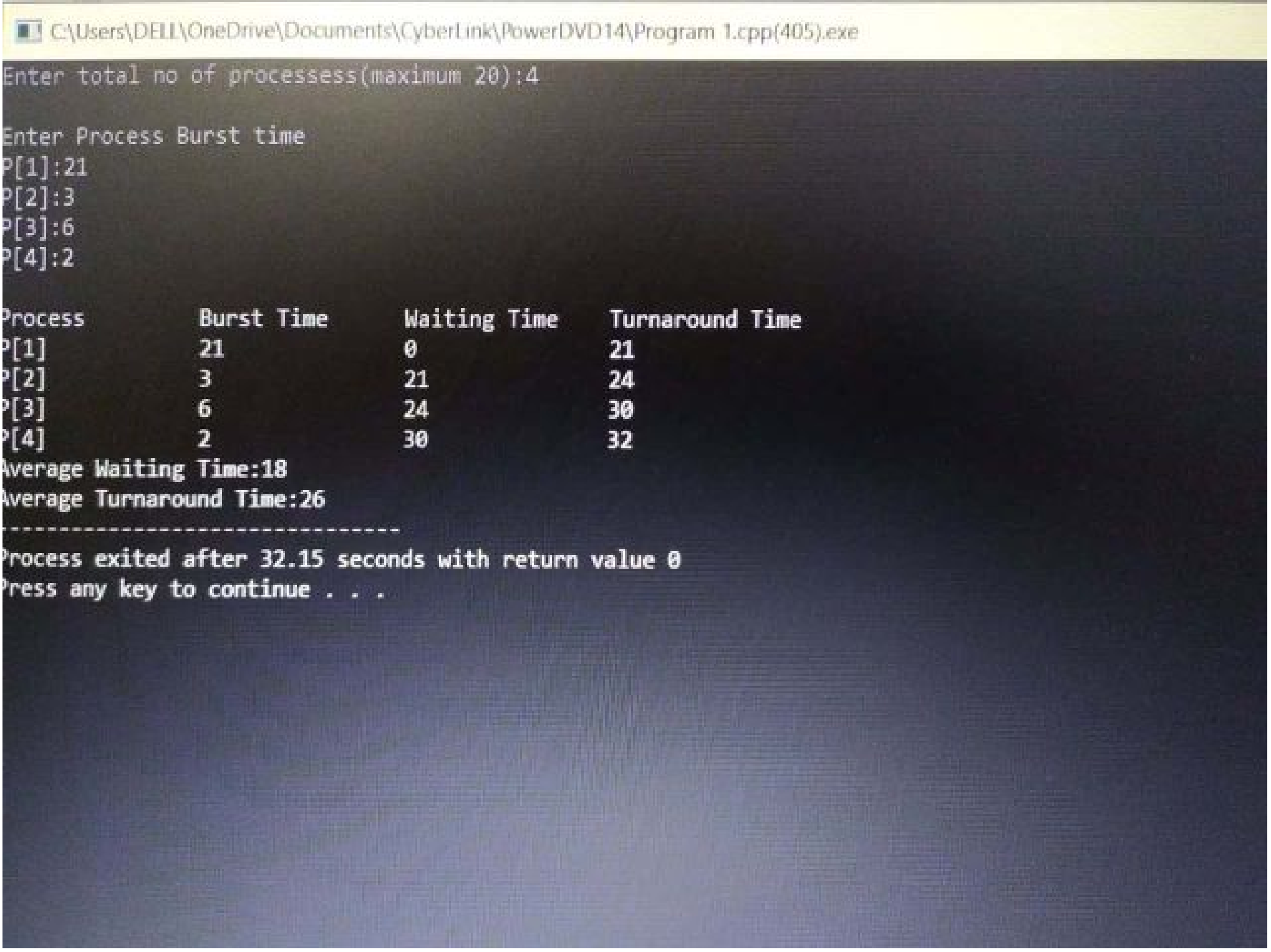
Experiment - 1

PAGE NO.
DATE

1. Write a program to implement FCFS CPU Scheduling algorithm

```
#include<iostream>
using namespace std;
int main()
{
int n, bt[20], wt[20], tat[20], avwt=0, avtat=0,
i, j;
cout<<"Enter total number of process(maximum 20):";
cin>>n;

cout<<"\n Enter Process Burst Time \n";
for(i=0; i<n; i++)
{
cout<<"P["<<i+1<<"]:";
cin>>bt[i];
}

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

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

```
cout<<"\n Process \t\t Burst Time \t Waiting Time \t
Turnaround Time";

// calculating turnaround time
for(i=0; i<n; i++)
{
tat[i] = bt[i] +wt[i];
avwt += wt[i];
avtat += tat[i]
cout<<"\n P["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\t"
<<tat[i];
}

avwt /=i;
avtat /=i;
cout<<"\n\n Average Waiting Time:"<<avwt;
cout<<"\n Average Turnaround Time:"<<avtat;
return 0;
}
```



Tanya Gupta
0905CS191184

Experiment - 1

PAGE NO.
DATE

1. WAP to implement in c++ FCFS scheduling with arrival time and burst time to calculate TAT and WT

```
#include<iostream>
using namespace std;

//Function to find the waiting time for
//all process

void findWaitingTime( int processes[], int n, int bt[],
                      int wt[], int at[]
{ int service_time[n];
  service_time[0] = at[0];
  wt[0];

// calculating waiting time

for( int i=1; i<n; i++)

// Add burst time of previous processes
service_time[i] = service_time[i-1] + bt[i-1];

// find waiting time for current process

wt[i] = service_time[i] - at[i];
  if(wt[i]<0)
    wt[i]=0
}
```

PAGE NO.
DATE

```
}

// function to calculate turn around time

void findTurnAroundTime( int processes[], int n, int bt[],
                         int wt[], int tat[])
{
  // calculating turn around time
  for (int i=0; i<n; i++)
    tat[i] = bt[i] + wt[i];
}

// function to calculate average waiting and
// turn-around time

void findavgTime(int processes[], int n, int at[])
{
  int wt[n], tat[n];

// function to find waiting time of all processes
   findWaitingTime( processes, n, bt, wt, at);

// function to find turn around time for all
processes
   findTurnAroundTime(process, n, bt, wt, tat);

// Display process
```

PAGE NO.
DATE

```
Cout<< "Processes" << "\t" << "Burst Time" << "\t" << "Arrival
Time" << "\t" << "Waiting Time" << "\t" << "Turn-around Time"
<< "\t" << "Completion Time\n";

int total_wt = 0; total_tat = 0;
for(int i=0; i<n; i++)
{
 total_wt = total_wt + wt[i];
 total_tat = total_tat + tat[i];
 int compl_time = tat[i] + at[i];

Cout<< i+1 << "\t\t" << bt[i] << "\t\t" << at[i] << "\t\t" << wt[i] <<
"\t\t" << tat[i] << "\t\t\t" << compl_time << endl;
}

Cout << "Average waiting time =" << (float) total_wt/
                                        float n;

Cout<< "\n Average turn around time =" << (float) total_tat
                                        /(float) n;
}
int main()
{ int processes[] = {1, 2, 3};
 int n = sizeof processes / sizeof processes[0];
int burst_time[] = {5, 9, 6};
int arrival_time[] = {0, 3, 6};
findavgTime(processes, n, burst_time, arrival_time);
 return 0;
}
```

```
Processes       Burst Time      Arrival Time    Waiting Time    Turn-Around Time        Completion Time
1               5               0               0               5                       5
2               9               1               7               31                      14
3               6               6               8               24                      20
Average waiting time = 5.333333
Average turn around time = 18

--------------------------------

Process exited after 0.05019 seconds with return value 0
Press any key to continue . . .
```

Tanya Gupta
0905CS191184

Experiment-2

PAGE NO.
DATE

2. WAP to implement SJF algorithm without arrival time.

```
#include<iostream>
using namespace std;
int main()
{
int n, temp, i, j;
float atat=0, awt=0, stat=0, swt=0;

cout<<"Enter no. of Process:"<<endl;
cin>>n;

int b[n], tat[n], wt[n];

for(i=0; i<n; i++)
{
cout<<"Enter burst time:";
cin>>b[i];
}

for(i=0; i<n; j++)
{ for(j=i+1; j<n; j++)
{if(b[i]>b[j])
 {
   temp=b[i];
   b[i]=b[j];
   b[j]=temp;
```

```
        }
    }
}
// Calculating waiting time
wt[0] = 0;
for(i=1; i<n; i++)
{
  wt[i]=0;
  for(j=0; j<i; j++)
      wt[i] = wt[i] + b[j];
      swt  = swt + wt[i];

}
// calculating average waiting time
awt = swt/n;
// calculating turn around time
for(i = 0; i<n; i++)
{
tat[i] = b[i] + wt[i];
stat = stat + tat[i];
 }
// average turn aroung time
atat = stat/n;

// Display process
cout << " Pobcess \t Burst-time(s) \t Waiting-time(s) \t
  Turnaround-time(s) \n" ;
```

PAGE NO.
DATE

```
for(i=0; i<n; i++)
{
cout<<"p"<<i+1<<"\t\t"<<b[i]<<"\t\t"<<wt[i]<<"\t\t"<<tat[i]<<
endl;
}
cout<<" AWT="<<awt<<endl<<" ATAT="<<atat;

}
```

```
Enter no of Process:
5
Enter burst time:8
Enter burst time:6
Enter burst time:1
Enter burst time:9
Enter burst time:3
Process   Burst-time(s)  Waiting-time(s)  Turnaround-Time(s)
P1              1                0              1
P2              3                1              4
P3              6                4              10
P4              8                10              18
P5              9                18              27
AWT-6.6
ATAT-12
--------------------------------
Process exited after 7.905 seconds with return value 0
Press any key to continue . . .
```

Tonya Gupta
0905CS191184

Experiment-2

PAGE NO.
DATE

2. WAP to implement SJF algorithm with arrival time.

```
#include<iostream>
using namespace std;
int main()
{
 int n, temp, tt=0, min, d, i, j;
 float atat=0, awt=0, stat=0, awt=0;

 Cout<<"Enter no. of Process:"<<endl;
 cin>>n;
 int a[n], b[n], e[n], tat[n], wt[n];

 for(i=0; i<n; j++)
 {
 Cout<<"Enter aerival time:";
 cin>> a[i];
 }


 for(i=0; i<n; i++)
 {
 cout<<"Enter burst time:";
 cin>> b[i];
 }


 for(i=0; i<n; j++)
 {
```

PAGE NO.
DATE

```
for (j=i+1; j<n; j++)
 {
  if (b[i]>b[j])
   {
     temp = a[i];
     a[i] = a[j];
      a[j] = temp;

     temp = b[i];
      b[i] = b[j];
       b[j] = temp;
    }
  }
}


min = a[0];
for (i=0; i<n; i++)
{
  if (min>a[i])
   {
     min= a[i];
      d=i;
   }
}

tt = min;
e[d] = tt + b[d];
tt = e[d];
```

PAGE NO.
DATE

```
for (i=0; i<n; i++)
{
if (a[i] == min)
{
e[i] = b[i] + tt;
tt = e[i];
}
}

for(i=0; i<n; i++)
{
tat[i] = e[i] - a[i];
stat = stat + tat[i];
wt[i] = tat[i] - b[i];
swt = swt + wt[i];
}
atat = stat/n;
awt = swt/n;

cout<<" Process\t Arrival-time(s)\t Burst-time(s)\t
Waiting-time(s)\t Turnaround-time(s)\n";
for(i=0; i<n; i++)
{
cout<<"p"<<i+1<<"\t\t"<<a[i]<<"\t\t"<<b[i]<<"\t\t"<<wt[i]<<"\t\t"
<<tat[i]<<endl;
}
cout<<" AWT="<<awt<<endl<<" ATAT="<<atat;
}
```

```
Enter no of Process:
5
Enter arrival time:3
Enter arrival time:1
Enter arrival time:4
Enter arrival time:0
Enter arrival time:2
Enter burst time:1
Enter burst time:4
Enter burst time:2
Enter burst time:6
Enter burst time:3
Process  Arrival-time(s)  Burst-time(s)  Waiting-time(s)  Turnaround-time(s)
P1              3               1               3               4
P2              4               2               3               5
P3              2               3               7               10
P4              1               4               11              15
P5              0               6               0               6
AWT=4.8
ATAT=8
--------------------------------

Process exited after 16.61 seconds with return value 0
Press any key to continue . . .
```

Tanya Gupta
09905CS191184

Experiment-3

Date / /
Page No.
Shivalal

3. Write a program to implement SRTF CPU Scheduling algorithm.

```
#include <stdio.h>
int main()
{
int a[10], b[10], x[10], i, j, smallest, count=0,
 time, n;
double avg=0, tt=0, end;

printf("enter the number of Processes: \n");
scanf("%d", &n);
printf("enter arrival time \n");
for(i=0; i<n; i++)
 scanf("%d", &a[i]);
printf("enter burst time \n");
for(i=0; i<n; i++)
scanf("%d", &b[i]);
for(i=0; i<n; i++)
  x[i] = b[i];
  b[9] = 9999;

for(time=0; count!=n; time++)
 {
  smallest = 9;
for(i=0; i<n; i++)
{
if(a[i]<=time && b[i]<b[smallest] && b[i]>0)
  smallest=i;
```

```
}

b[smallest]--;

if (b[smallest]==0)
{
count++;
end = time+1;
avg = avg + end - a[smallest] - x[smallest];
tt = tt + end - a[smallest];
}
}
printf("\n\n Average waiting time = %.lf\n", avg/n);
printf(" Average turnaround time = %.lf", tt/n);
  return 0;
}
```

C:\Users\hp\Documents\SRTF.exe

```
enter the number of Processes:
4
enter arrival time
0
1
2
3
enter burst time
8
4
9
5


Average waiting time = 6.500000
Average Turnaround time = 13.000000
--------------------------------
Process exited after 36.83 seconds with return value 0
Press any key to continue . . .
```

Tanya Gupta
0905CS191184

Experiment-4

PAGE NO.
DATE

4. WAP to implement priority scheduling algorithm with taking the different number of process with its burst time and priority for calculating WT, average waiting time, TAT and ATAT

```
#include <iostream>
using namespace std;
int main()
{
int bt[20], wt[20], p[20], tat[20], pr[20], i, j, n,
total_wt=0, total_tat=0, pos, temp, avg_wt,
avg_tat;

Cout<<" Enter number of Process:";
Cin>>n;
Cout<<" Enter Burst Time and Priority \n";
for (i=0; i<n; i++)
{
cout<<" \nP["<<i+1<<"]\n";
Cout<<" Burst Time :";
Cin>> bt[i];
Cout<<" Priority :";
Cin>> pr[i];
p[i]=i+1;          // process number
}
// Sorting Burst Time, Priority and Process Number
//in ascending order using selection sort
```

PAGE NO.
DATE

```
for(i=0; i<n; i++)
 { pos=i;
  for(j=i+1; j<n; j++)
   {
   if(pr[j] < pr[pos]) // check Process Priority
    pos=j;
   }

 temp=pr[i];
 pr[i]=pr[pos];
 pr[pos]=temp;

 temp = bt[i];
 bt[i]=bt[pos];
 bt[pos]=temp;

 temp= p[i];
 p[i] = p[pos];
 p[pos] = temp;
 }

 wt[i]=0;  // Waiting time of first Process
 // calculate Waiting Time
 for(i=1; i<n; i++)
 {
  wt[i]=0;
  for(j=0; j<i; j++)
   wt[i]+=bt[j];
```

```
    total_wt += wt[i]; // Total Waiting Time
}
avg_wt = total_wt/n; // Average Waiting Time

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

Cout<<" \n Process \t Burst-time \t waiting-time \t
Turnaround-time ";
for(i=0; i<n; i++)
{
Cout<<" \n P[" << p[i] << "]" << " \t\t " << bt[i] << " \t\t " << wt[i] << " \t\t "
<< tat[i];
}
Cout << endl;

Cout<<" \n Waiting Time =" << total_wt << endl;
Cout<<" Turnaround Time = " << total_tat << endl;
Cout<<" AWT=" << avg_wt << endl << " ATAT=" << avg_tat;
}
```

```
Enter number of Process:3
Enter Burst Time and Priority

p[1]
Burst Time:10
Priority:2

p[2]
Burst Time:5
Priority:0

p[3]
Burst Time:8
Priority:1

Process     Burst-time          Waiting-time      Turnaround-time
P[2]            5                  0                 5
P[3]            8                  5                 13
P[1]            10                 13                23

Waiting Time=18
Turnaround Time=41
AWT=6
ATAT=13
--------------------------------
Process exited after 19.16 seconds with return value 0
Press any key to continue . . .
```

Tanya Gupta
0905CS191184

Experiment-5

PAGE NO.
DATE

5. Write a program to implement preemptive priority CPU scheduling algorithm

```
# include < stdio.h>
# include < conio.h>
# include < string.h>
void main()
{
int et[20], at[10], n, i, j, temp, p[10], st[10],
ft[10], wt[10], ta[10];
 int totwt = 0, totta = 0;
 float awt, ata;
 char pn[10][10], t[10];

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

for(i=0; i<n; i++)
{ printf(" Enter process name, arrival time,
execution time & priority :");
scanf("%s%d %d %d", pn[i], &at[i], & et[i], & p[i]);
 }

for(i=0; i<n; i++)
 for(j=0; j<n; j++)
  {
    if( p[i] < p[j])
     {
```

```
        temp = p[i];
        p[i] = p[j];
        p[j] = temp;

        temp = at[i];
        at[i] = at[j];
        at[j] = temp;

        temp = et[i];
        et[i] = et[j];
        et[j] = temp;

        strcpy(t, pn[i]);
        strcpy(pn[i], pn[j]);
        strcpy(pn[j], t);
        }
    }

for(i=0; i<n; i++)
{
  if(i==0)
  {
    st[i] = at[i]
    wt[i] = st[i] - at[i];
    ft[i] = st[i] + et[i];
    ta[i] = ft[i] - at[i];
  }
```

```
else
{
  st[i]= ft[i-1];
  wt[i] = st[i] -at[i];
  ft[i]= st[i] +et[i];
  ta[i] = ft[i] - at[i];
}

totwt += wt[i];
totta += ta[i];
}

awt = (float) totwt/n;
ata = (float) totta/n;

printf("\n Pname\t arrival time\t execution time\t
      priority\t waiting time\t atime");

for(i=0; i<n; i++)
printf("\n %s\t %5d\t\t %5d\t\t %5d\t\t %5d\t\t %5d
      \t\t %5d", pn[i], at[i], et[i], p[i], wt[i], ta[i]);

printf("\n Average Waiting time is %f", awt);
printf("\n Average turnaroundtime is %f", ata);
}
```

C:\Users\hp\Documents\Priority scheduling with arrival time.exe

```
Enter the number of process:5
Enter process name,arrivaltime,execution time & priority:p1
0
4
2
Enter process name,arrivaltime,execution time & priority:p2
1
3
3
Enter process name,arrivaltime,execution time & priority:p3
2
1
4
Enter process name,arrivaltime,execution time & priority:p4
3
5
5
Enter process name,arrivaltime,execution time & priority:p5
4
2
5

Pname    arrivaltime      executiontime    priority         waitingtime      tatime
p1           0                4                2                0                4
p2           1                3                3                3                6
p3           2                1                4                5                6
p4           3                5                5                5               10
p5           4                2                5                9               11
Average waiting time is:4.400000
Average turnaroundtime is:7.400000

Process exited after 36.76 seconds with return value 45
Press any key to continue . . .
```

Tanya Gupta
0905CS191184

Experiment-6

PAGE NO.
DATE

6. WAP to implement round robin scheduling algorithm for calculating average waiting and average turn around time.

```
#include<iostream>
#include<iomanip>    // manipulators
using namespace std;
class roundrobin     // class
{
 int bt[10], n, q;
public:
        void input();        // member function
        void display();      // declaration
        void calculate();

};

void roundrobin :: input()  // function definition
{                           // outside the class
Cout<<" How many process to be entered";
cin>>n;
 for( int i=0; i<n; i++)
{
Cout<<"Enter execution or burst time of Process"<<
 i+1<<":";
 Cin>> bt[i];
}
Cout<<" Enter Quantum Slice:";
```

```
cin >> q;
}
void roundrobin :: display()
{
cout << endl << " Process ID\t Burst Time " << endl;
for( int i=0; i<n; i++)
cout << setw(5) << i+1 << setw(15) << bt[i] << setw(15) << endl;
}

void roundrobin :: calculate() // function definition
{                              outside the class
int tbt=0, bt2[10], flag=1;
int wt=0, tat=0;
float avg=0, avtat=0;
for( int i=0; i<n; i++)
{
  tbt += bt[i];
  bt2[i] = bt[i];
}

Cout << " In Process ID \t Waiting Time \t Turn Around Time" <<
endl;
for(int y=0; y< tbt ; y++)
{
 for( int i=0; i<n; i++)
 {
    if(bt[i]<=0)
      continue;
```

```
        tat += q;
        bt[i] -= q;
        if (bt[i] <= 0)
        {
          tat += bt[i];
          wt = tat - bt[i];
cout << setw(5) << i+1 << setw(15) << wt << setw(15) << tat <<
 endl;
 avg += wt;
 avtat += tat;
    }
  }
}

 avg = avg / n;
 avtat /= n;
cout << "\n Average Waiting Time:" << avg;
cout << "\n Average Turn Around Time :" << avtat << endl;
}

int main()
{
roundrobin rr;
rr.input();
rr.display();
rr.calculate();
}
```

```
How many process to be entered5
Enter execution or burst time of Process1:8
Enter execution or burst time of Process2:6
Enter execution or burst time of Process3:1
Enter execution or burst time of Process4:9
Enter execution or burst time of Process5:3
Enter Quantum slice:1

process ID      Burst Time
    1              8
    2              6
    3              1
    4              9
    5              3

 Process ID     Waiting Time     Turn Around Time
    3              2               3
    5             10              13
    2             15              21
    1             17              25
    4             18              27

 Average Waiting Time:12.4
 Average Turn Around Time:17.8

--------------------------------
Process exited after 22.32 seconds with return value 0
Press any key to continue . . .
```

Tonya Gupta
0905CS191184

Experiment -7

PAGE NO.
DATE

7. Write a program to implement classical problem of synchronization Producer Consumer problem.

```
#include<stdio.h>
void main()
{
int buffer[10], buffersize, produce, consume;
int choice=0, in=0, out=0, bufsize=3; // initialization

while(choice!=3)
{
printf("\n 1. produce \n 2. Consume \n 3. Exit");
printf("\n Enter your choice:");
scanf("%d", &choice);

switch(choice)
{
 case 1:
        if((in+1)%bufsize==out)  // code of Producer
        printf("\n Buffer is full");  // Process
        else
        {
        printf("Enter the value:");
        scanf("%d", &produce);
         buffer[in]= produce;
         in = (in+1)%bufsize;
        }
```

```
break;

case 2:                    // code of Consumer Process
    if (in == out)
    printf("\n Buffer is Empty");
    else
    {
    consume = buffer[out];
    printf("\n The Consumed Value is %d", consume);
    out = (out+1) % bufsize;
    }
    break;
  }
 }
}
```

```
 1.produce
 2.Consume
 3.Exit
Enter your choice:1
Enter the value:10

 1.produce
 2.Consume
 3.Exit
Enter your choice:1
Enter the value:20

 1.produce
 2.Consume
 3.Exit
Enter your choice:1

 Buffer is Full
 1.produce
 2.Consume
 3.Exit
Enter your choice:2

 The Consumed Value is 10
 1.produce
 2.Consume
 3.Exit
Enter your choice:2

 The Consumed Value is 20
 1.produce
 2.Consume
 3.Exit
Enter your choice:2

Buffer is Empty
 1.produce
 2.Consume
 3.Exit
Enter your choice:3

--------------------------------
Process exited after 53.67 seconds with return value 3
Press any key to continue . . .
```

Tanya Gupta
0905CS191184

Experiment-8

PAGE NO.
DATE

8. Write a program to implement classical inter process communication problem Reader Writers.

```
# include < stdio.h>
# include < semaphore.h>
# include < pthread.h>

// here consider : 10 readers & 5 writers
sem_t wrt;                    // semaphore write
pthread_muter_t m;            // muter m
int cnt = 1;
int numreader = 1;            // count of reader access
                                 that resource
// writer method
void * writer (void *wno)
{
sem_wait (& wrt);             // wait operating
cnt = cnt * 2;
printf (" Writer %d modified cnt to %d \n",
            (*(int *) wno), cnt);   // printing modified cnt
sem_post (& wrt);
}
// reader method
void reader (void * rno)
{
// reader acquire lock before updating num_reader
pthread_mutex_lock (& m);
```

```
numreader++;

// if encounter first reader
if (numreader == 1)
{
   sem_wait (&wrt);     // block writer
}

pthread_muter_unlock (&m);
printf ("reader %d read cnt as %d \n", *((int *)arg), cnt);

pthread_muter_lock (&m);
numreader--;

// if last reader
if (numreader == 0)
{
 sem_post (&wrt);          // activate writer
}
pthread_muter_unlock (&m);
}

main()
{ pthread_t read[10], writer[5];   // reader writer thread
pthread_muter_init (&m, NULL); // initialise muter to null
 sem_init (&wrt, 0, 1);  // initialise writer semaphores
                            to 1 with no sharing(0)
                            with other process.
```

PAGE NO.
DATE

```
int a[10] = { 1,2,3,4,5,6,7,8,9,10 };
int i;
// calling function for each thread created
for(i=0; i<10; i++)
{
pthread_create(&read[i], NULL, (void*)reader, (void*)&a[i]);
}

for(i=0; i<5; i++)
{
pthread_create(&write[i], NULL, (void*)writer, (void*)&
                                                a[i]);
}

// joining the read
for(i=0; i<10; i++)
{
pthread_join(read[i], NULL);
}
for(i=0; i<5; i++)
{
pthread_join(write[i], NULL);
}

// destroy mutex and semaphore.
pthread_mutex_destroy(&m);
sem_destroy(&wrt);
}
```

C:\Users\hp\Documents\reader_writer_problem.exe

```
reader 1 read cnt as1
reader 2 read cnt as1
reader 3 read cnt as1
reader 4 read cnt as1
reader 5 read cnt as1
reader 6 read cnt as1
reader 7 read cnt as1
reader 8 read cnt as1
reader 9 read cnt as1
reader 10 read cnt as1
writer 1 modified cnt to 2
writer 2 modified cnt to 4
writer 3 modified cnt to 8
writer 4 modified cnt to 16
writer 5 modified cnt to 32


--------------------------------
Process exited after 0.09792 seconds with return value 0
Press any key to continue . . .
```

Tanya Gupta
0905CS191184

Experiment-9

PAGE NO.
DATE

9. Write a program to implement FIFO page replacement algorithm.

```
#include<stdio.h>
int main()
{
int i, j, n, a[50], frame[10], no, k, avail, count=0;
printf("\n enter the length of the Reference string:");
scanf("%d", &n);

printf("\n enter the reference string:\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("\t.ref 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)
```

PAGE NO.
DATE

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

C:\Users\hp\Documents\FIFO.exe

```
 enter the number of Frames:3
        ref string        page frames
7                7        -1      -1

0                7        0       -1

1                7        0       1

2                2        0       1

0

3                2        3       1

0                2        3       0

4                4        3       0

2                4        2       0

3                4        2       3

0                0        2       3

3

Page Fault Is 10
--------------------------------
Process exited after 51.74 seconds with return value 0
Press any key to continue . . .
```

Tanya Gupta
0905CS191184

Experiment-10

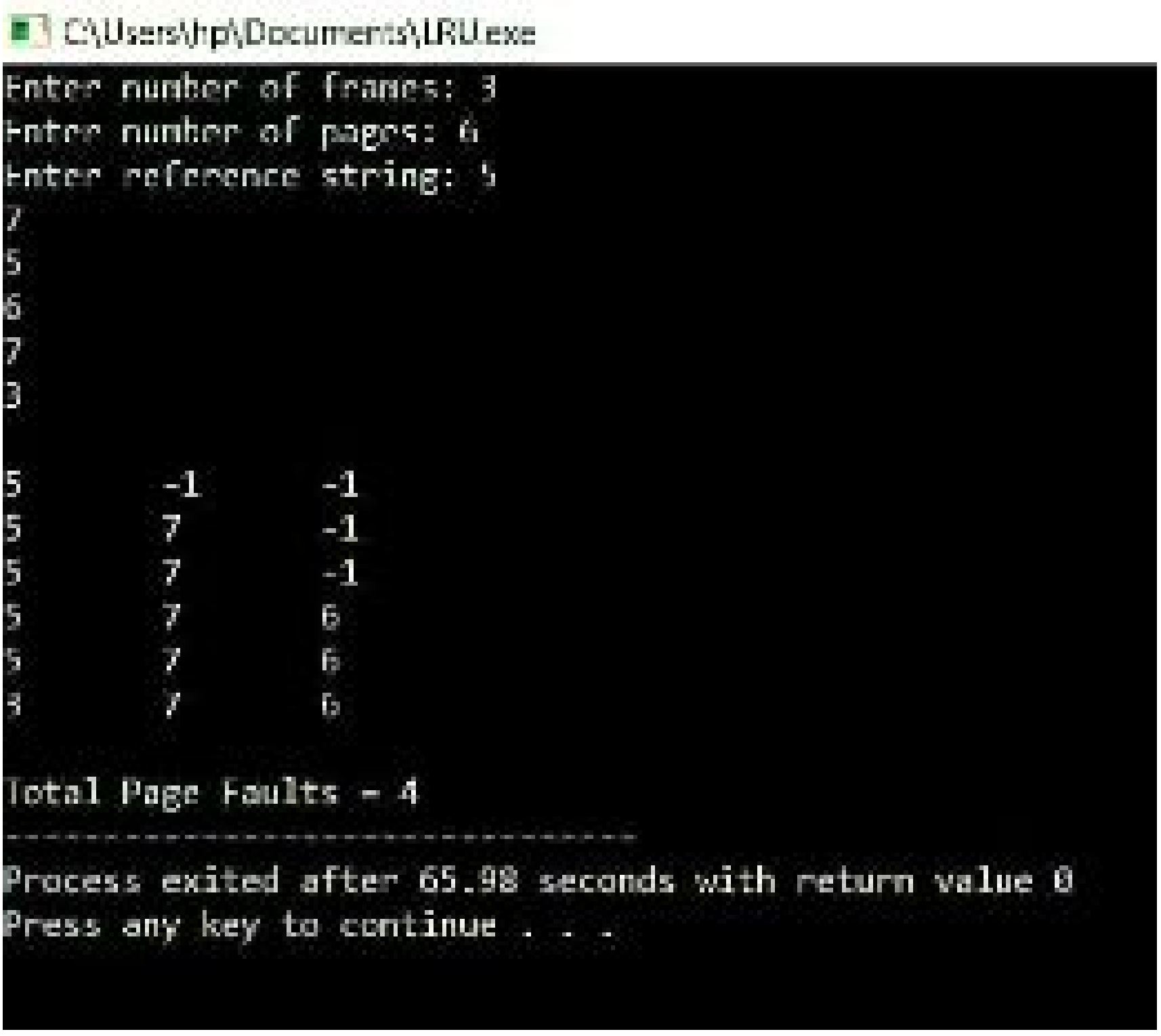
Date / /
Page No.
Shivalal

10. Write a program to implement LRU page replacement algorithm.

```
#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 no_of_frames, no_of_pages, frames[10], pages[30],
counter=0, time[10], flag1, flag2, i, j, pos, faults=0;
printf("Enter number of frames:");
scanf("%d", &no_of_frames);
printf("Enter number of pages:");
scanf("%d", &no_of_pages);
printf("Enter reference string:");
for(i=0; i<no_of_pages; ++i)
{
```

```
scanf("%d", &pages[i]);
}

for(i=0; i< no_of_frames; ++i)
{
frames[i] = -1;
}

for(i=0; i< no_of_pages; ++i)
{ flag1 = flag2 = 0;
for( j=0; j< no_of_frames; ++j)
{
if( frames[j] == pages[i])
{
counter++;
time[j] = counter;
flag1 = flag2 = 1;
break;
}
}

if (flag1 == 0)
{ for(j=0; j< no_of_frames; ++j)
{ if( frames[j] == -1)
{ counter++;
faults++;
frames[j] = pages[i];
time[j] = counter;
flag2 = 1;
```

```
        break;
    }
    }
    }

    if (flag2==0)
    { pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
    }
    printf("\n");

    for(j=0; j<no_of_frames; ++j)
    {
    printf("%d\t", frames[j]);
    }
}
printf("\n\n Total Page Faults = %d", faults);
    return 0;
}
```



Tanya Gupta
0905CS191184

Experiment - 11

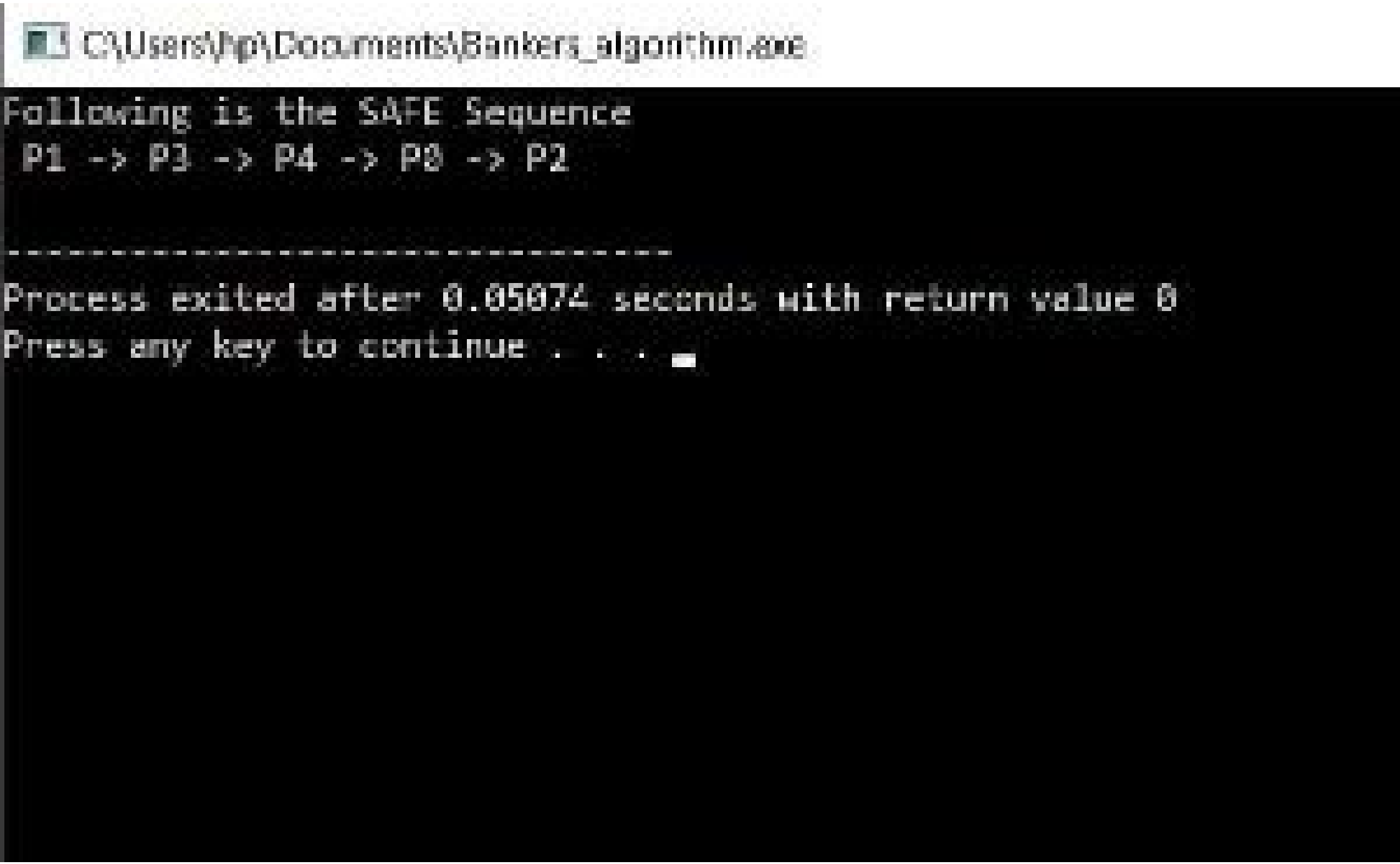
Date / /
Page No.
Shivalal

11. Write a program to implement Banker's algorithms.

```
#include<iostream>
using namespace std;
int main()
{
// P0, P1, P2, P3, P4 are the process names here
int n, m, i, j, k;
n=5 ; // Number of processes
m=3 ; // Number of resources

int alloc[5][3] = { {0,1,0}, // P0 // Allocation matrix
                    {2,0,0}, // P1
                    {3,0,2}, // P2
                    {2,1,1}, // P3
                    {0,0,2} }; // P4

int max[5][3] = { {7,5,3}, // P0 // Max Matrix
                  {3,2,2}, // P1
                  {9,0,2}, // P2
                  {2,2,2}, // P3
                  {4,3,3} }; // P4

int avail[3] = {3,3,2}; // Available Resources
int f[n], ans[n], ind=0;
for (k=0; k<n; k++)
{
    f[k] = 0;
}
```

```
int need[n][m];
for(i=0; i<n; i++)
{
 for(j=0; j<m; j++)
    need[i][j] = max[i][j] - alloc[i][j];
 }
 int y = 0;
for(k = 0; k<5; k++)
{
for(i=0; i<n; i++)
{
 if(f[i] == 0)
 {
  int flag = 0;
for(j=0; j<m; j++)
{
if(need[i][j] > avail[j])
  { flag = 1;
    break;
  }
 }


 if(flag == 0)
{ ans[ind++] = i;
for(y=0; y<m; y++)
   avail[y] += alloc[i][y];
    f[i] = 1;
  }
 }
```

```
    }
}

cout << " Following is the SAFE Sequence" << endl;
for (i=0; i<n-1 ; i++)
    cout << " P " << ans[i] << " -> ";
cout << " P " << ans[n-1] << endl;

return 0;
}
```



Tanya Gupta
0905CS191184

Experiment -12

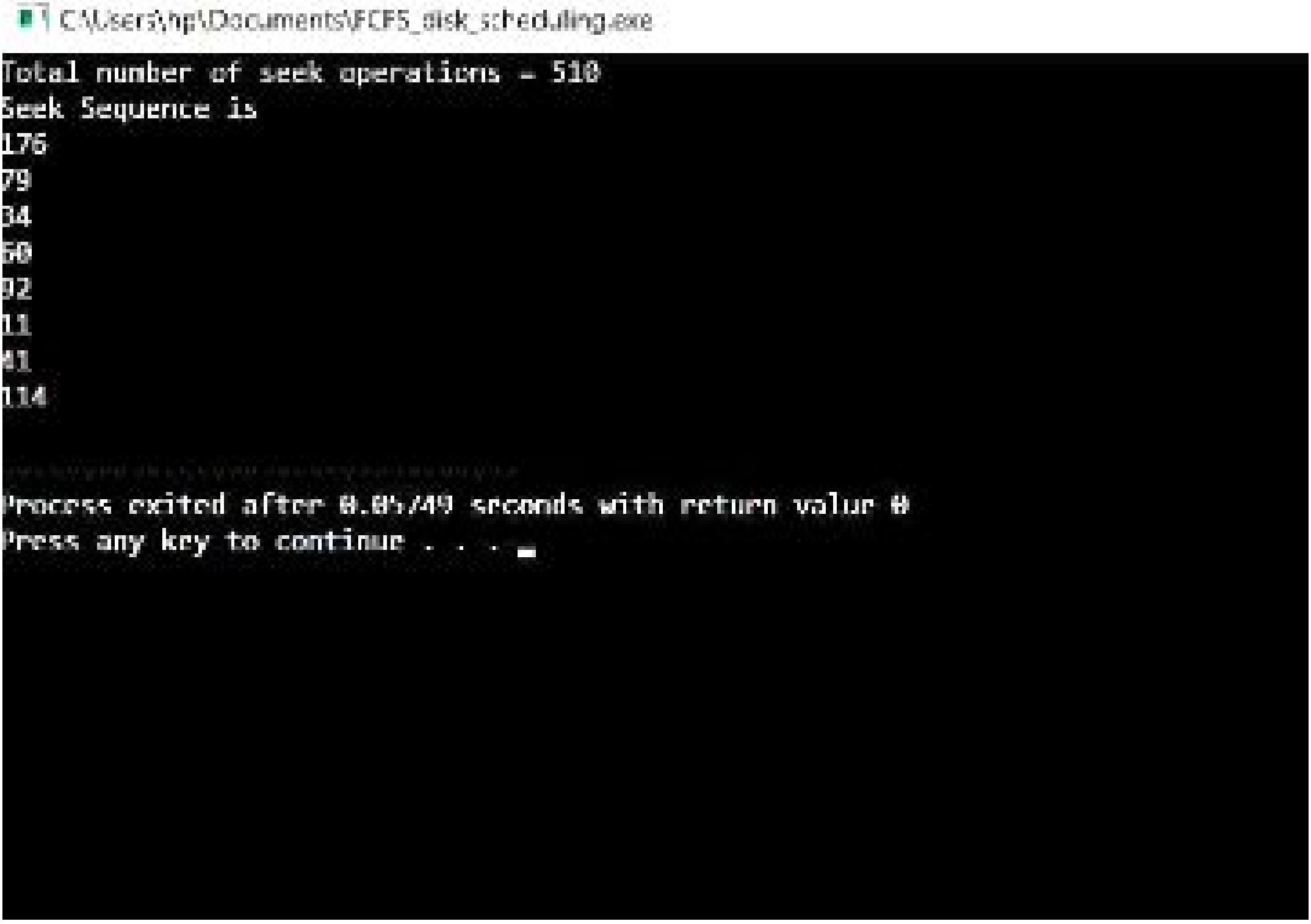
Date / /
Page No.
Shivalal

12. Write a program to implement FCFS disk scheduling algorithm.

```
#include <bits/stdc++.h>
using namespace std;
int size = 8;
void FCFS( int arr[], int head)
{
    int seek_count = 0;
    int distance, cur_track;

for( int i=0; i< size; i++)
 {  cur_track = arr[i];

// calculate absolute distance
 distance = abs( cur_track - head);

// increase the total count
 seek_count += distance;

// accessed track is now new head
  head = cur_track;
}
cout<<" Total number of seek operation ="<< seek_count
        << endl;
}
// seek sequence would be the same as array
// sequence
```

13. cout<<" Seek Sequence is" << endl;

```
for( int i=0; i< size; i++)
 {
 cout << arr[i] << endl;
 }
}

int main()
{
 // request array
int arr[size] = {176, 79, 34, 60, 92, 11, 41, 114};
 int head = 50;
 FCFS (arr, head);
 return 0;
 }
```



Tonya Gupta
0905CS191184

Experiment-13

Date / /
Page No.
Shivalal

13. Write a program to implement SSTF disk scheduling algorithm.

```
#include<bits/stdc++.h>
using namespace std;

// calculate difference of each track number with
// the head position

void calculateDifference( int request[], int head, int n,
                          int diff[][2])
{
for( int i=0; i<n; i++)
 {
  diff[i][0]= abs( head- request[i]);
 }
}

// find maccessed track which is at minimum
// distance from head

int findMINC int diff[][2], int n)
{
 int index =-1;
 int minimum =1e9;

for( int i=0; i<n; i++)
 {
  if( !diff[i][1] && minimum > diff[i][0])
```

```
        { minimum = diff[i][0];
          index = i;
        }
    }
    return index;
}

void shortestSeekTimeFirst (int request[], int head, int n)
{
    if (n==0)
    { return;
    }

// Create array of objects of class node
int diff[n][2] = {{0,0}};

// Count total number of seek operation
int seekcount = 0

// stores sequence in ~~order~~ which disk access is done
int seeksequence[n+1] = {0};

for (int i=0; i<n; i++)
{
seeksequence[i] = head;
calculatedifference (request, head, diff, n);
int index = findMIN ( diff, n);
diff[index][1] = 1;
```

```
// Increase the total count
seekcount += diff[index][0];

// Access track is now new head
head = request[index];
}
seeksequence[n] = head;

cout<<" Total number of seek operations =" << seekcount<<endl;
cout<<" Seek Sequence is:" << "\n";

// Print the sequence
for (int i=0; i<=n; i++)
{
cout<< seeksequence[i] <<"\n";
}
}

int main
{ int n=8;

int proc[n] = { 176, 79, 34, 60, 92, 11, 41, 114 };

shortestSeekTimeFirst (proc, 50, n);

return 0;
}
```