REPORT

As a project work for Course

OPERATING SYSTEMS (CSE 316)

Name : Rahul Debnath

Reg. Number : 11802734

Program : CSE B.Tech

Semester : 4th

School : School of Computer Science

Name of the University: Lovely Professional University

1.Introduction

One of the most important function of the operating system is to manipulate the ordering of the processes which arrives for execution i.e. process scheduling. Operating System has to select a process from the waiting queue and put it in the ready queue. It has also to decide whether to pre-empt the process or let the process finish (non-pre-emptive). There are various methods to achieve this scheduling of process. This project is dedicated to solving one of those methods that is pre-emptive shortest job first.

In Shortest Job First scheduling, that process is selected to be moved into ready queue which has the lowest burst time among all the processes in the waiting queue. Since, the mode we are observing here is pre-emptive, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.

Algorithm with Time Complexity

```
1. Lmt ← Total no. of processes                                    O(1)

2. for(i = 0; i < lmt; i++)                                            O(n)

   {

        p[i]=i+1;

        prio[i]=0;

        printf("\nEnter Entire Details of Process[%d]\n", i + 1);

        printf("Arrival Time:\t");

        scanf("%d", &a_time[i]);

        printf("Burst Time:\t");
```

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

             temp[i] = b_time[i];

        }

3.      printf("\nEnter the Time Quantum:");                                                        O(1)

       scanf("%d", &time_quantum);                                                                  O(1)

       printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\t Priority\n");          O(1)

4.    for(total = 0, i = 0; x != 0;)                                                                O(n^3)

        {

            for(z=0;z<lmt;z++)

            {

                    int temp1;

                    pos=z;

                    for(j=z+1;j<lmt;j++)

                            {

                                if(prio[j]<prio[pos])

                                    pos=j;

                            }


                    temp1=prio[z];


                    prio[z]=prio[pos];


                    prio[pos]=temp1;


                            temp1=b_time[z];
```

```
                b_time[z]=b_time[pos];

                b_time[pos]=temp1;

                temp1=a_time[z];

                a_time[z]=a_time[pos];

                a_time[pos]=temp1;

                temp1=p[z];

                p[z]=p[pos];

                p[pos]=temp1;

                temp1=temp[z];

                temp[z]=temp[pos];

                temp[pos]=temp1;

            }
```

```
5.  if(temp[i] <= time_quantum && temp[i] > 0)                                    O(1)

       {

           total = total + temp[i];

           temp[i] = 0;

           counter = 1;

       }


   else if(temp[i] > 0)                                                           O(1)

       {

           temp[i] = temp[i] - time_quantum;
```

```
     total = total + time_quantum;

}


  for(b=0;b<limit;b++)                                                   O(n)

            {

                    if(b==i)                                             O(1)

                    prio[b]+=1;

                    else                                                 O(1)

                    prio[b]+=2;

            }


if(temp[i] == 0 && counter == 1)                                         O(1)

{

     x--;

     printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\t\t%d", p[i], b_time[i], total - a_time[i], total -
  a_time[i] - b_time[i],prio[i]);

     wait_time = wait_time + total - a_time[i] - b_time[i];

     turnaround_time = turnaround_time + total - a_time[i];

     counter = 0;

}

if(i == lmt - 1)                                                     O(1)

{

     i = 0;

}

else if(a_time[i + 1] <= total)                                      O(1)

{
```

```
            i++;

        }

        else                                                                              O(1)

        {

            i = 0;


          }

    }

    return 0;

}
```

Total complexity:

O(1)+O(n)+O(1)+O(1)+O(1)+O(n^3)+O(1)+O(1)+O(n)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1) = O(n^3)

Code for the program(Using C Lang)

```
#include<stdio.h>

int main()

{

    int i, lmt, total = 0, x, counter = 0, time_quantum,j;

    int wait_time = 0, turnaround_time = 0,pos,z,p[10],prio[10], a_time[10], b_time[10],     temp[10],b;


           float average_wait_time, average_turnaround_time

         printf("\nEnter Total Number of Processes:");

        scanf("%d", &lmt);

        x = lmt;

       for(i = 0; i < lmt; i++)
    {
           p[i]=i+1;
          prio[i]=0;
         printf("\nEnter Entire Details of Process[%d]\n", i + 1);
        printf("Arrival Time:\t");
        scanf("%d", &a_time[i]);
       printf("Burst Time:\t");
       scanf("%d", &b_time[i]);
       temp[i] = b_time[i];
}

printf("\nEnter the Time Quantum:");
scanf("%d", &time_quantum);
printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\t Priority\n");
for(total = 0, i = 0; x != 0;)
{
                  for(z=0;z<lmt;z++)
                     {
                              int temp1;
                              pos=z;
                              for(j=z+1;j<lmt;j++)
                              {
                                        if(prio[j]<prio[pos])
                                         pos=j;
                              }
```

```
                        temp1=prio[z];
                         prio[z]=prio[pos];
                         prio[pos]=temp1;
                         temp1=b_time[z];
                         b_time[z]=b_time[pos];
                         b_time[pos]=temp1;
                         temp1=a_time[z];
                         a_time[z]=a_time[pos];
                         a_time[pos]=temp1;
                         temp1=p[z];
                         p[z]=p[pos];
                         p[pos]=temp1;
                         temp1=temp[z];
                         temp[z]=temp[pos];
                         temp[pos]=temp1;
                  }
 if(temp[i] <= time_quantum && temp[i] > 0)
{
    total = total + temp[i];
    temp[i] = 0;
    counter = 1;
}
  else if(temp[i] > 0)
{
    temp[i] = temp[i] - time_quantum;
```

```
          total = total + time_quantum;

      }


        for(b=0;b<lmt;b++)

                {if(b==i)

                    prio[b]+=1;

                 else

                     prio[b]+=2;

                }


      if(temp[i] == 0 && counter == 1)

      {    x--;

          printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\t\t%d", p[i], b_time[i], total - a_time[i], total -
a_time[i] - b_time[i],prio[i]);

          wait_time = wait_time + total - a_time[i] - b_time[i];

          turnaround_time = turnaround_time + total - a_time[i];

          counter = 0;

      }

      if(i == lmt - 1)

      {

          i = 0;


        }

      else if(a_time[i + 1] <= total)

      { i++;}

      else
```

```
            {i = 0;}
        }
        return 0;
}                          //end of program
```

Boundary conditions:

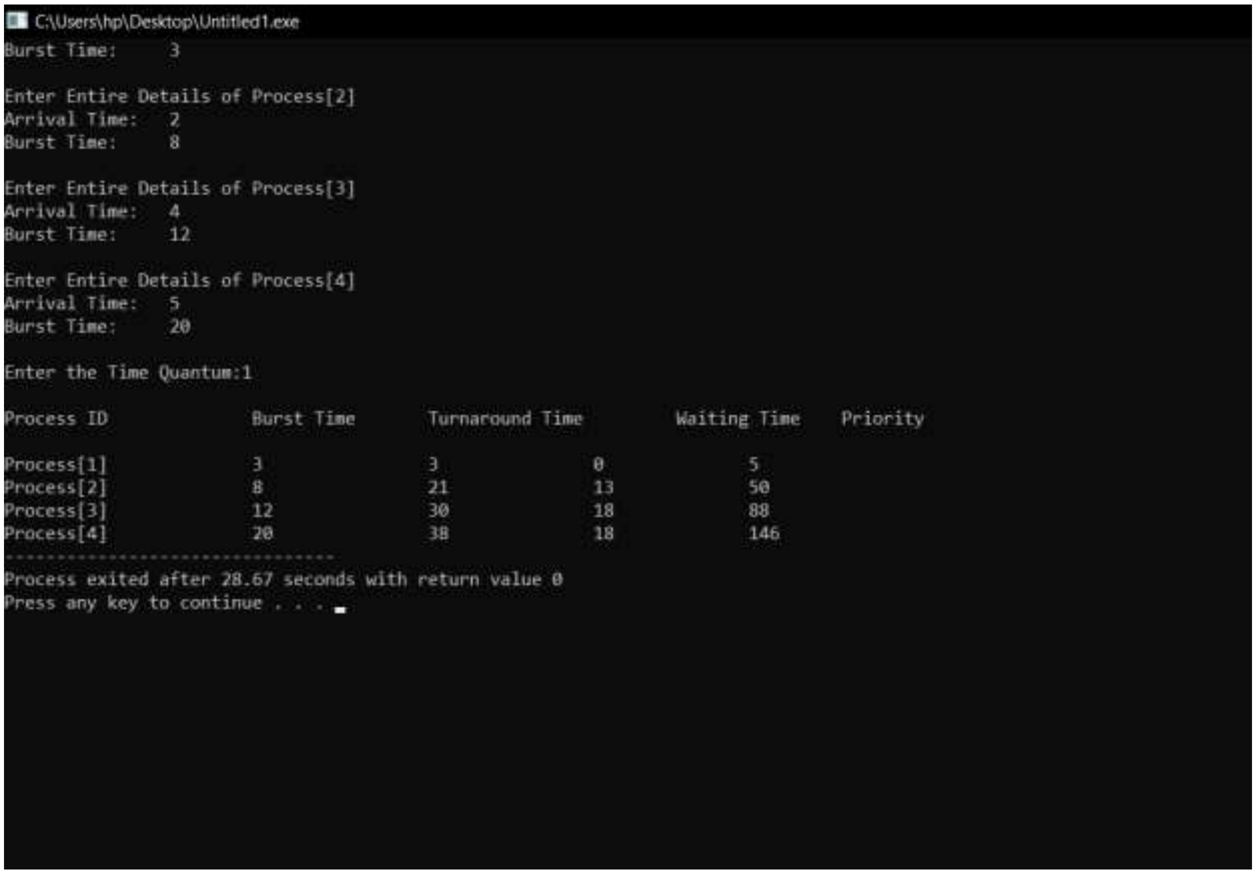
Note – While specifying the input, following points are to be strictly followed.

1. The maximum numbers of processes that can be checked for is 10.
2. The arrival time of all the processes should be greater than 0.
3. The burst time of all the processes should be greater than 0.
4. All the input should be integer (whole numbers wiz. 0, 1, 2, … n).
5. No two processes are to have same arrival time and same burst time

Sample test case 1:

Process	**Arrival time**	**Burst time**
P1	1	3
P2	2	8
P3	4	12
P4	5	20

Output:

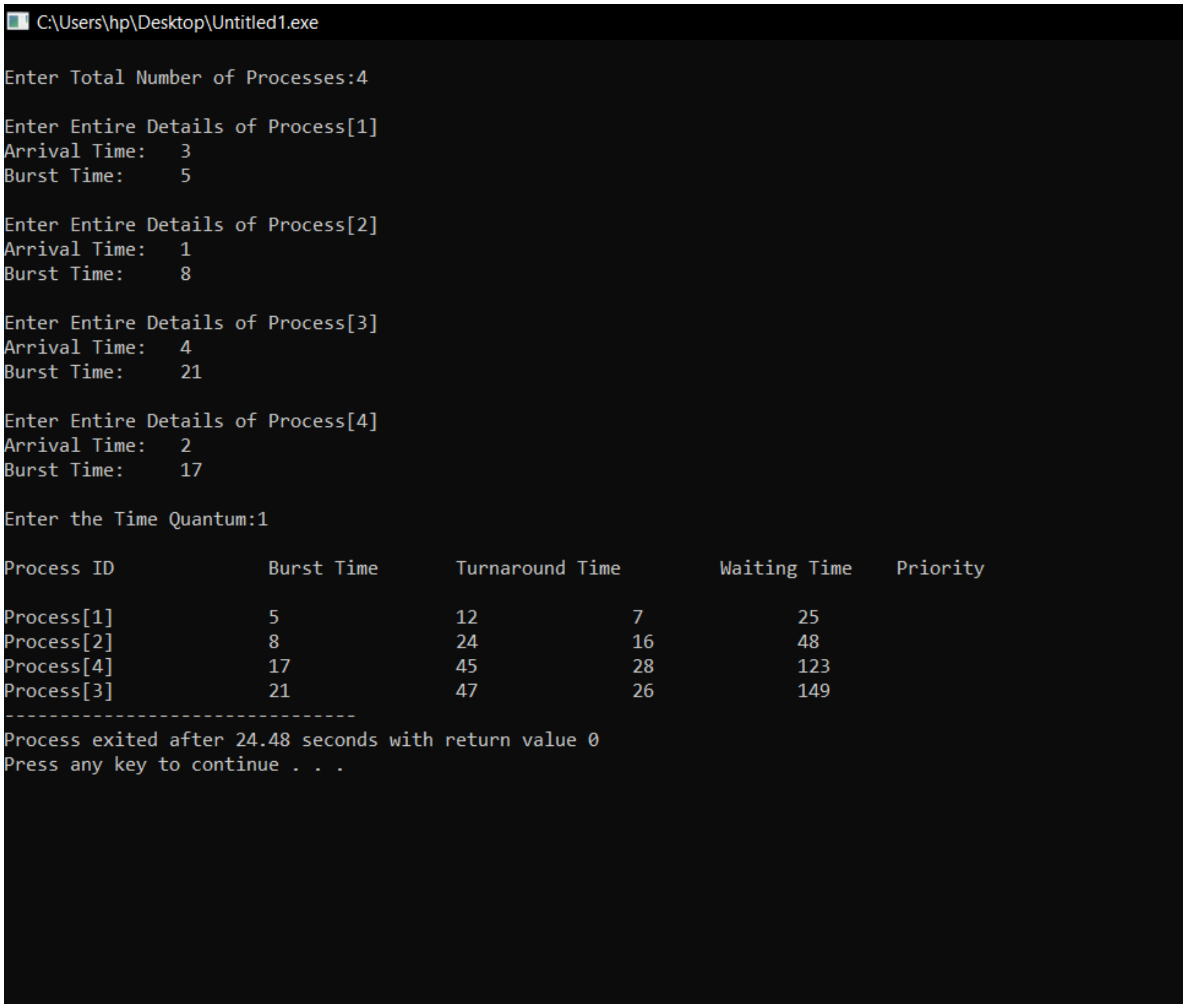


```
C:\Users\hp\Desktop\Untitled1.exe
Burst Time:     3

Enter Entire Details of Process[2]
Arrival Time:   2
Burst Time:     8

Enter Entire Details of Process[3]
Arrival Time:   4
Burst Time:     12

Enter Entire Details of Process[4]
Arrival Time:   5
Burst Time:     20

Enter the Time Quantum:1

Process ID            Burst Time       Turnaround Time        Waiting Time    Priority

Process[1]            3                3              0            5
Process[2]            8                21             13           50
Process[3]            12               30             18           88
Process[4]            20               38             18           146
--------------------------------
Process exited after 28.67 seconds with return value 0
Press any key to continue . . .
```

Sample test case 2 :

Process	**Arrival time**	Burst time
P1	3	5
P2	1	8
P3	4	21
P4	2	17

Output:



```
Enter Total Number of Processes:4

Enter Entire Details of Process[1]
Arrival Time:   3
Burst Time:     5

Enter Entire Details of Process[2]
Arrival Time:   1
Burst Time:     8

Enter Entire Details of Process[3]
Arrival Time:   4
Burst Time:     21

Enter Entire Details of Process[4]
Arrival Time:   2
Burst Time:     17

Enter the Time Quantum:1

Process ID              Burst Time       Turnaround Time         Waiting Time    Priority

Process[1]              5                12              7              25
Process[2]              8                24              16             48
Process[4]              17               45              28             123
Process[3]              21               47              26             149
--------------------------------
Process exited after 24.48 seconds with return value 0
Press any key to continue . . .
```

Github repository link-https://github.com/rahul28092000/CSE-316-CA-Project