Mawlana Bhashani Science and Technology University

Lab-Report

Report No: 08

Course code: ICT-3110

Course title: Operating System Lab

Date of Performance: 19-09-2020

Date of Submission:

Submitted by

Name: Jannatul Ferdush Dhina

ID:IT-18012

3rd year 1st semester

Session: 2017-2018

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Experiment no:08

Experiment Name: Implementation of SJF Scheduling Algorithm.

Theory:

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

SJF can be used in specialized environments where accurate estimates of running time are available.

Algorithm:

- Sort all the process according to the arrival time.
- Then select that process which has minimum arrival time and minimum Burst time.
- After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

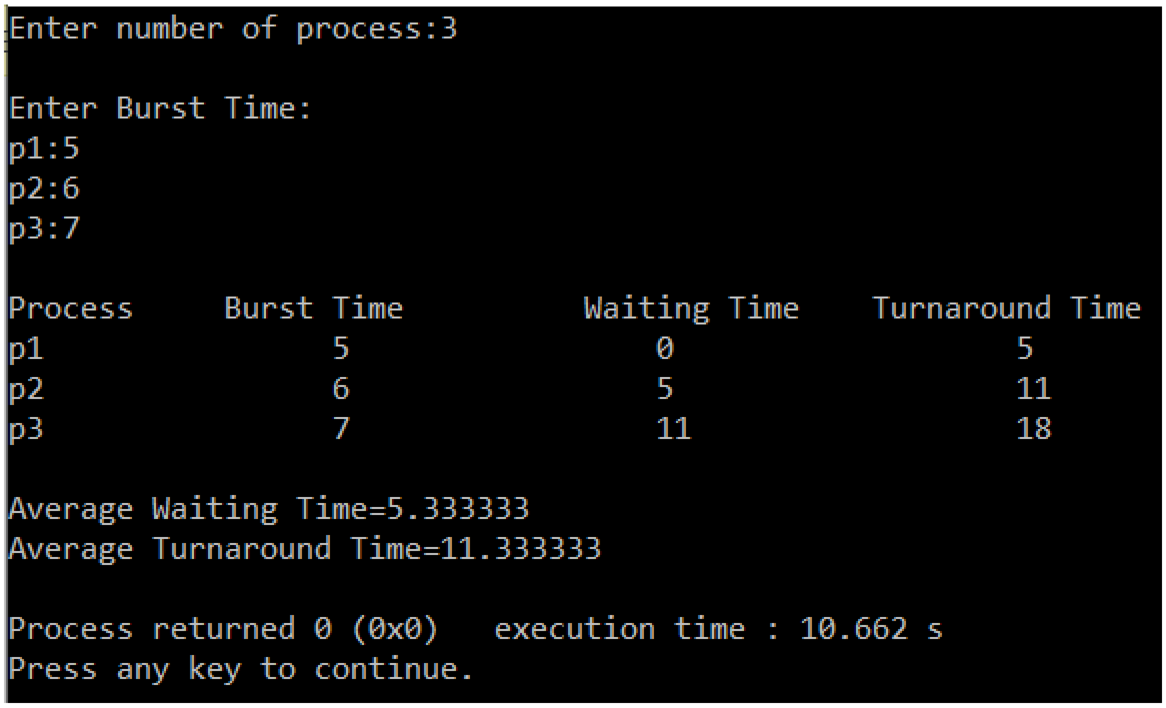
Working Process:

```
#include<stdio.h>


int main()

{

   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
```

```
float avg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);

printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
    printf("p%d:",i+1);
    scanf("%d",&bt[i]);
    p[i]=i+1;
}



for(i=0;i<n;i++)
{
    pos=i;
    for(j=i+1;j<n;j++)
    {
        if(bt[j]<bt[pos])
            pos=j;
    }

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

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

```
        p[pos]=temp;
    }
 wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output Sample:



```
Enter number of process:3

Enter Burst Time:
p1:5
p2:6
p3:7

Process     Burst Time          Waiting Time    Turnaround Time
p1                5                   0                   5
p2                6                   5                  11
p3                7                  11                  18

Average Waiting Time=5.333333
Average Turnaround Time=11.333333

Process returned 0 (0x0)   execution time : 10.662 s
Press any key to continue.
```

<u>Discussion:</u>

The lab helped to learn sjf algorithm. Now we could be able to solve these kind of problem.