

**Mawlana Bhashani Science and Technology University**

**Lab-Report**

Report No: 08

Course code: ICT-3110

Course title: Operating Systems Lab

Date of Performance:

Date of Submission:

**Submitted by Submitted To**

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Name: Md Azharul Islam

ID:IT-18052

3rd year 1st semester

Session: 2017-2018

Dept. of ICT

MBSTU.

**Experiment no :** 08

**Experiment Name : Implementation of SJF Scheduling Algorithm.**

**Theory :**

Shortest job first scheduling is the job or process scheduling algorithm that follows the nonpreemptive scheduling discipline. In this, scheduler selects the process from the waiting queue with the least completion time and allocate the CPU to that job or process. Shortest Job First is more desirable than FIFO algorithm because SJF is more optimal as it reduces average wait time which will increase the throughput.In shortest job first scheduling algorithm, the processor selects the waiting process with the smallest execution time to execute next.

**Implementation:**

**1.** Sort all the process according to the arrival time.

**2.** Then select that process which has minimum arrival time and minimum Burst

time.

**3.** 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 :**

Code for SJF Scheduling Algorithm –

#include<stdio.h>

void 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;           //contains process number

    }

    //sorting burst time in ascending order using selection sort

    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;            //waiting time for first process will be zero

    //calculate waiting time

    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;      //average waiting time

    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];     //calculate turnaround time

        total+=tat[i];

        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

    }

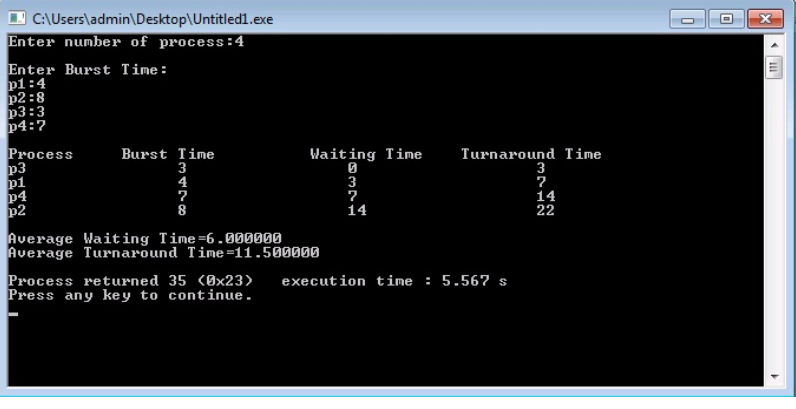
    avg\_tat=(float)total/n;     //average turnaround time

    printf("\n\nAverage Waiting Time=%f",avg\_wt);

    printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

**Output :**

****

**Discussion :**

In SJF process completion time needs to be known earlier. Although prediction is difficult. Sometimes the problem of starvation occurs in SJF.SJF needs the knowledge to know how long a process will run.It is not easy to know the upcoming CPU request length..In SJF, it is necessary to record elapsed time, resulting in more overhead the processor.