Mawlana Bhashani Science and Technology University

Lab-Report

Report No:

Course code: ICT-3110

Course title: Operating Systems Lab

Date of Performance:

Date of Submission:

Submitted by

Name: Hasibul Islam Imon

ID:IT-18047

3th year 1ndsemester

Session: 2017-18

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Lab report name:Implementation of SJF Scheduling Algorithm

Objectives:

1. The SJF scheduling is especially appropriate for batch jobs for which the run times are known in advance.
2. The SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal.

3. The obvious problem with SJF scheme is that it requires precise knowledge of how long a job or process will run, and this information is not usually available.

4. The best SJF algorithm can do is to rely on user estimates of run times.

Question No.1. What is SJF Scheduling Algorithm?

Answer: Shortest-Job-First (SJF) is a non-preemptive discipline in which waiting job (or process) with the smallest estimated run-time-to-completion is run next. In other words, when CPU is available, it is assigned to the process that has smallest next CPU burst.

Question No.2. How to implementation in C?

Answer:

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

   int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
```

```
int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10];
//clrscr();
printf("Enter the number of process:");
scanf("%d",&n);
for(i=0; i<n; i++)
{
   printf("Enter process name, arrival time& execution time:");
   //flushall();
   scanf("%s%d%d",pn[i],&at[i],&et[i]);
}
for(i=0; i<n; i++)
   for(j=0; j<n; j++)
   {
      if(et[i]<et[j])
      {
         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];
    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("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
for(i=0; i<n; i++)
```

```
    printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
  printf("\nAverage waiting time is:%f",awt);
  printf("\nAverage turnaroundtime is:%f",ata);
  getch();
}
```

Output:

```
Enter the number of process:3
Enter process name, arrival time& execution time:2 5 7
Enter process name, arrival time& execution time:3 6 14
Enter process name, arrival time& execution time:4 7 12

Pname   arrivaltime     executiontime   waitingtime     tatime
2           5               7               0               7
4           7              12               5              17
3           6              14              18              32
Average waiting time is:7.666667
Average turnaroundtime is:18.666666
```

Conclusion:

The SJF scheduling is especially appropriate for batch jobs for which the run times are known in advance. Since the SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal. The SJF algorithm favors short jobs (or processors) at the expense of longer ones.