Mawlana Bhashani Science and Technology University

Lab-Report

Report No:08

Lab Report Name: Implementation of SJF Scheduling Algorithm

Course code: ICT-3110

Course title: Operating System Lab

Date of Performance: 16-09-2020

Date of Submission:

Submitted by

Name: Tazneen Akter

ID: IT-18056

3rd year 1st semester

Session: 2017-18

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) 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. 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.

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

Working Process:

```
#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);

  return 0;
}
```

Output:

```
Enter number of process:4

Enter Burst Time:
p1:4
p2:8
p3:3
p4:7

Process     Burst Time          Waiting Time    Turnaround Time
p3                3                   0                   3
p1                4                   3                   7
p4                7                   7                   14
p2                8                   14                  22

Average Waiting Time=6.000000
Average Turnaround Time=11.500000

Process returned 35 (0x23)   execution time : 21.311 s
Press any key to continue.
```

Discussion: In this lab we have implemented FIFO page replacement algorithm using C language. By solving this problem in future we can solve any problem of this algorithm.