Lab report NO : 08

Lab Report Name : Implementation of SJF scheduling algorithm .

Theory

Program for Shortest Job First (or SJF) CPU Scheduling | Set 1 (Non- preemptive)

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.

Algorithm:

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.

How to compute below times in SJF using a program?

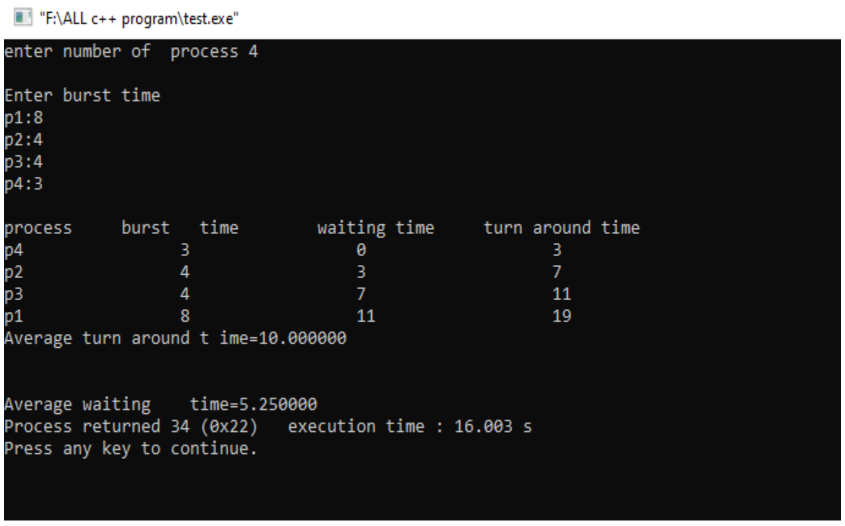
1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time.
   Waiting Time = Turn Around Time – Burst Time

C program for implementing for implementing shorted job first algorithms (SJF):

```
#include<stdio.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>
```

```
void main()
{
    float awt,atat;
    int k,l,bst[20],p[20],x,to=0,po,tem,wt[20],tat[20];
    printf("enter number of  process ");
    scanf("%d",&x);
    printf("\nEnter burst time \n");

    for(k=0; k<x; k++)
    {
        printf("p%d:",k+1);
        scanf("%d",&bst[k]);
        p[k]=k+1;
    }
        for(k=0; k<x; k++)
    {
        po=k;
        for(l=k+1; l<x; l++)
        {
            if(bst[l]<bst[po])
                po=l;
        }
        tem=bst[k];
        bst[k]=bst[po];
        bst[po]=tem;
        tem=p[k];
        p[k]=p[po];

        p[po]=tem;
    }

    wt[0]=0;
    for(k=1; k<x; k++)

    {

        wt[k]=0;

        for(l=0; l<k; l++)

            wt[k]+=bst[l];


        to+=wt[k];

    }


    awt=(float)to/x;          to=0;


    printf("\nprocess\t    burst   time    \twaiting time\t turn around time");

    for(k=0; k<x; k++)

    {

        tat[k]=bst[k]+wt[k];

        to+=tat[k];

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

    }


    atat=(float)to/x;


         printf("\nAverage turn around t ime=%f\n",atat);


    printf("\n\nAverage waiting    time=%f",awt);


}
```

Output:



```
enter number of  process 4

Enter burst time
p1:8
p2:4
p3:4
p4:3

process     burst   time        waiting time     turn around time
p4               3                 0                   3
p2               4                 3                   7
p3               4                 7                   11
p1               8                 11                  19
Average turn around t ime=10.000000


Average waiting    time=5.250000
Process returned 34 (0x22)   execution time : 16.003 s
Press any key to continue.
```