Lab Report No. **08**

Lab Report Name: **Implementation of SJF Scheduling algorithm .**

Objectives:

i What is SJF Scheduling algorithm.
ii How to implementation in C

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.

Characteristics of SJF scheduling :

1. It is associated with each job as a unit of time to complete.
2. This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical.
3. It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time.
4. It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Advantages of SJF

1.Maximum throughput

2.Minimum average waiting and turnaround time

Disadvantages of SJF

1.May suffer with the problem of starvation

2.It is not implementable because the exact Burst time for a process can't be known in advance.

Corresponding Code:

```
#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;    //average turnaround time
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
    printf("\n");
    return 0;
}
```

Output: