OPERATING SYSTEM

CSE-316

PROJECT TITLE:

SHORTEST JOB FIRST SCHEDULING (SJF)

NAME: PRINCE KUMAR
SECTION: K21WY-G2
REGI NO: 12112403
ROLL NO: RK21WYA02

SUBMITTED TO:
MS.RICHA SHARMA

S.NO	CONTENT
1	Introduction
2	Objective
3	Scope of the project
4	Advantages and disadvantage
4	Source code
5	Test cases
6	Actual Outputs
7	Dfd
8	Flow diagram
9	Conclusion

INTRODUCTION

Shortest Job First (SJF) is a CPU scheduling algorithm that is designed to reduce the average waiting time and turnaround time of processes in a system. In this algorithm, the process with the smallest execution time is selected for execution first. SJF is a non-preemptive algorithm, which means that once a process starts executing, it will not be interrupted until it finishes, even if another process with a shorter execution time becomes available.

SJF is an effective algorithm for reducing waiting time and turnaround time, as it prioritizes shorter processes, allowing them to complete their execution quickly and efficiently. This results in a more efficient use of system resources and a faster completion time for all processes.

One important consideration when using SJF is accurately estimating the execution time of processes. If the execution times are not estimated correctly, the algorithm may prioritize processes that are actually longer, which can lead to longer waiting times and reduced efficiency.

In summary, SJF is a popular CPU scheduling algorithm that prioritizes shorter processes to reduce waiting time and turnaround time. It is effective in batch processing systems and real-time systems where the response time of processes is critical. Accurate estimation of process execution times is crucial for the success of this algorithm.

OBJECTIVE

- The objective of the SJF algorithm is to improve system performance.
- This is achieved by prioritizing shorter jobs to reduce the average waiting time and turnaround time of processes.
- By selecting the shortest job first, SJF ensures that shorter processes are executed first, while longer processes are executed later.
- The algorithm aims to achieve optimal performance by minimizing the amount of time that each process spends waiting in the ready queue.
- This can increase the system's throughput and reduce the overall response time of the system.

Overall, the objective of the SJF algorithm is to prioritize shorter jobs and reduce waiting times and turnaround times, resulting in improved system performance and efficiency.

SCOPE OF THE PROJECT

The scope of the Shortest Job First (SJF) algorithm is to optimize CPU scheduling in a system to reduce the average waiting time and turnaround time of processes. This algorithm can be used in various operating systems and applications, including batch processing systems, real-time systems, and multi-user systems.

- Batch processing systems: In batch processing systems, where a large number of similar jobs need to be executed, SJF can be used to prioritize shorter jobs and reduce the overall execution time.

- Real-time systems: In real-time systems, where the response time of processes is critical, SJF can be used to prioritize shorter jobs and ensure that they are executed quickly.

- Multi-user systems: In multi-user systems, where multiple users are competing for system resources, SJF can be used to optimize CPU scheduling and ensure that all users have fair access to system resources.

Advantages& disadvantage of SJF

Advantages:

- Shortest jobs are favored having less execution time.
- It is probably optimal, in that it gives the minimum average waiting time and turnaround time for a given set of processes.

Disadvantages:

- SJF may cause starvation if shorter processes keep coming. This problem is solved by aging.
- It cannot be implemented at the level of short-term CPU scheduling

SOURCE CODE

```c
#include<stdio.h>
int main()
{
    //n    number of processes
    //bt   burst time
    //p    process
    //at   arrays to store the arrival time
    //wt   waiting time
    //tat  turnaround time
    //rt   remaining time
    //ct   completion time
    // variables for loop counters i and j and a temporary variable temp
for sorting

    int n, bt[20],P[20], at[20], wt[20], tat[20], rt[20], ct[20], i, j,
temp;
    float avg_wt, avg_tat;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("\nEnter the arrival time & burst time for each process:\n");
    for(i=0; i<n; i++)
    {
        printf("P%d: ", i+1);
        P[i]=i+1;
        scanf("%d %d", &at[i], &bt[i]);
        rt[i]=bt[i];
    }
```

```c
  // sorting processes based on arrival time using selection sort
    for(i=0; i<n; i++)
    {
        for(j=i+1; j<n; j++)
        {
            if(at[i]>=at[j])
            {   //sorting arival time
                temp=at[i];
                at[i]=at[j];
                at[j]=temp;
               //sorting bt
                temp=bt[i];
                bt[i]=bt[j];
```

```
                bt[j]=temp;
               //sorting rt remaining time
                temp=rt[i];
                rt[i]=rt[j];
                rt[j]=temp;
              //sorting process
                temp=P[i];
                P[i]=P[j];
                P[j]=temp;
            }
        }
    }
```

```
  //algorithm for sjf
    int time=at[0], done=0, min_bt, k;
    while(done!=n)
    {
        min_bt=9999; // assuming maximum burst time to be 9999
        k=-1;
        for(i=0; i<n; i++)
        {
            if(rt[i]>0 && at[i]<=time && rt[i]<min_bt)
            {
                min_bt=rt[i];
                k=i;
            }
        }
        if(k==-1) // no process available to execute
        {
            time++;
            continue;
        }
        // executing the process
        rt[k]--;
        time++;
        if(rt[k]==0) // process execution completed
        {
            done++;
            ct[k]=time;
            tat[k]=ct[k]-at[k];
            wt[k]=tat[k]-bt[k];
            if(wt[k]<0) wt[k]=0;
        }
    }
```

```
 // calculating average waiting time and
    //average turnaround time and printing it
    avg_wt=0;
    avg_tat=0;
    printf("\nProcess\tArrival Time\tBurst Time\tCompletion
Time\tWaiting Time\tTurnaround Time\n");
    for(i=0; i<n; i++)
    {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", P[i], at[i], bt[i],
ct[i], wt[i], tat[i]);
        avg_wt+=wt[i];
        avg_tat+=tat[i];
    }
    avg_wt/=n;
    avg_tat/=n;
    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);
    return 0;
}
```

TEST CASES

TEST CASE 1:

INPUT:
Number of processes: 4

PID	AT	BT
P1	1	3
P2	2	4
P3	1	2
P4	4	4

EXPECTED OUTPUT:

PID	AT	BT	CT	WT	TAT
P3	1	2	3	0	2
P1	1	3	6	2	5
P2	2	4	10	4	8
P4	4	4	14	6	10

Average waiting time: 3.00
Average turnaround time: 6.25

TEST CASE 2:

INPUT:
Number of processes: 4

PID	AT	BT
P1	5	8
P2	0	5
P3	4	9
P4	1	2

EXPECTED OUTPUT:

PID	AT	BT	CT	WT	TAT
P2	0	5	7	2	7
P4	1	2	3	0	2
P3	4	9	24	11	20
P1	5	8	15	2	10

Average waiting time: 3.75
Average turnaround time: 9.75

ACTUAL OUTPUTS:

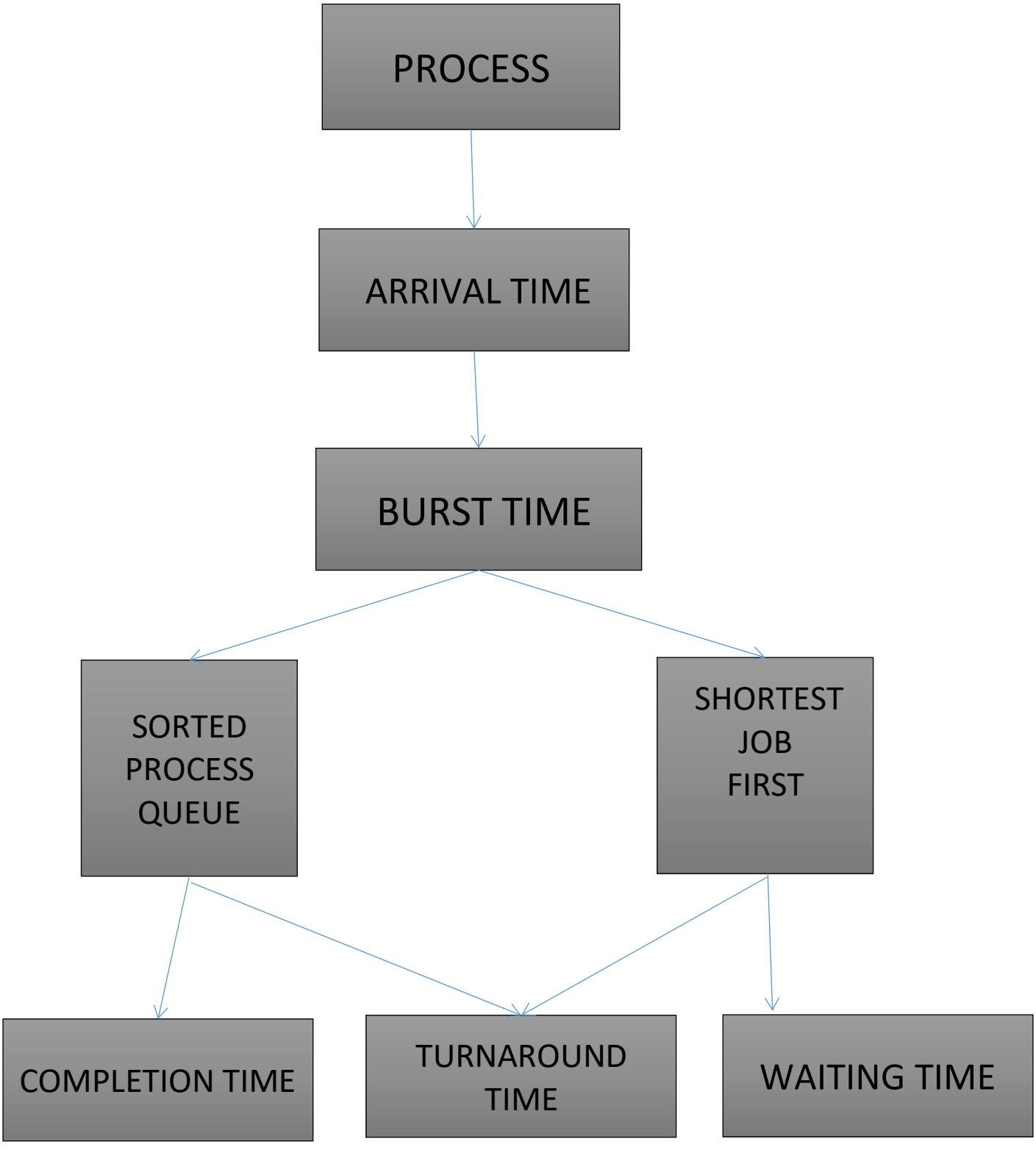
TEST-CASE1:

```
Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
P3      1               2               3               0               2
P1      1               3               6               2               5
P2      2               4               10              4               8
P4      4               4               14              6               10

Average Waiting Time: 3.000000
Average Turnaround Time: 6.250000
PS C:\Users\DELL\OneDrive\Desktop\OS PROJ\SJF>
```

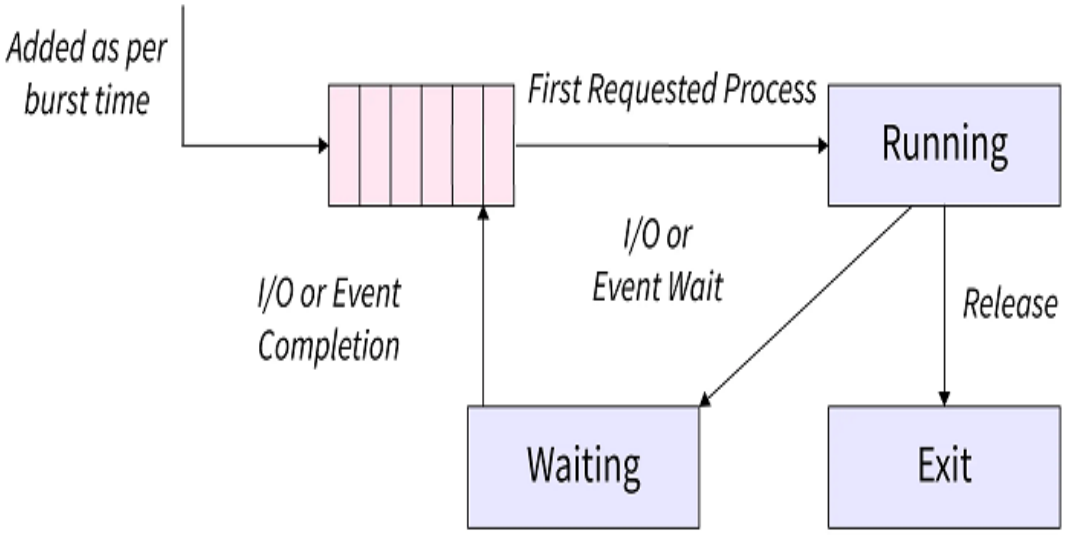
TEST-CASE2:

```
Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
P2      0               5               7               2               7
P4      1               2               3               0               2
P3      4               9               24              11              20
P1      5               8               15              2               10

Average Waiting Time: 3.750000
Average Turnaround Time: 9.750000
PS C:\Users\DELL\OneDrive\Desktop\OS PROJ\SJF>
```

DFD DIAGRAM


PROCESS
ARRIVAL TIME
BURST TIME
SORTED PROCESS QUEUE
SHORTEST JOB FIRST
COMPLETION TIME
TURNAROUND TIME
WAITING TIME

FLOW DIAGRAM



CONCLUSION

In conclusion, SJF (Shortest Job First) is a CPU scheduling algorithm that prioritizes the process with the shortest burst time for execution. This algorithm can minimize the average waiting time of processes and increase CPU utilization. However, it can also cause long processes to suffer from starvation if there are always shorter processes in the queue.

In practice, there are various modifications of the SJF algorithm that address some of its limitations, such as the SRTF (Shortest Remaining Time First) algorithm that dynamically re-evaluates the remaining burst time of the processes. Additionally, SJF is one of many CPU scheduling algorithms that are used in modern operating systems, with each having their own strengths and weaknesses that need to be considered in different scenarios.