Name: Adwait S Purao
UID: 2021300101
Batch: B2
Experiment: 5

Problem: With the increasing number of devotees visiting the temple, the wait time for "Dev Darshan" has become longer, causing inconvenience to the devotees. This has resulted in a need for
a more efficient system that can manage the waiting queue and reduce the wait time for "Dev Darshan".

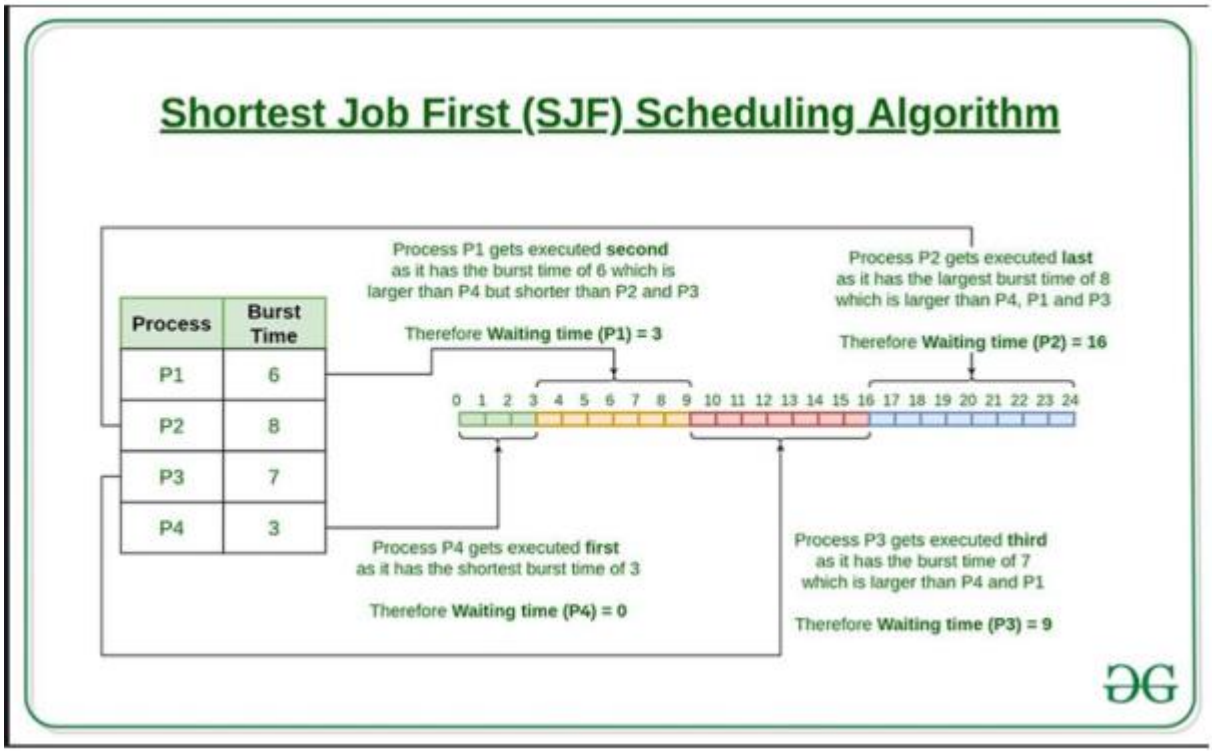
Goal: To design and implement a CPU scheduling algorithm for the "Dev Darshan" system in the temple that can effectively manage the waiting queue, minimize the wait time for devotees, and improve the overall experience of visiting the temple.

Scope: The CPU scheduling algorithm will be implemented in the existing "Dev Darshan" system, and
will consider the following factors:
The number of devotees waiting in the queue.
The priority of each devotee, such as senior citizens, children, and differently-abled individuals.
The time taken by each devotee to complete "Dev Darshan".
The availability of resources, such as the number of "Dev Darshan" counters and the capacity of the
temple.
The solution will be tested and evaluated based on its ability to reduce the wait time for devotees, improve the utilization of resources, and ensure fairness and equality in the allocation of resources.
Give the solution with example of Dev darshan using CPU scheduling algorithm.

Theory:
The 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, also known as Shortest Job Next (SJN), can be preemptive or non-preemptive.
Characteristics of SJF Scheduling:
• 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 times 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:
• Sort all the processes according to the arrival time.
• Then select that process that has minimum arrival time and minimum Burst time.
• After completion of the process make a pool of processes that arrives afterward till the completion of the previous process and select that process among the pool which is having minimum Burst time.



Advantages of Shortest Job First (SJF) Scheduling Algorithm:
1. It results in minimum average waiting time, as it selects the shortest job first for execution.
2. It is optimal in nature as it minimizes the average waiting time, which leads to high system throughput.
3. It is efficient in terms of minimizing the turnaround time for each process.
4. It gives a fair distribution of CPU time to each process, as it prioritizes shorter processes over longer ones.
5. It can be preemptive or non-preemptive, making it flexible for different types of systems.
6. It can prioritize important processes with shorter execution times, ensuring better response times for critical tasks.
7. It can be combined with other scheduling algorithms, such as round-robin, to achieve a better balance between performance and fairness.

Disadvantages of Shortest Job First (SJF) Scheduling Algorithm:

1. It requires accurate knowledge of the execution time of each process, which is not always possible to obtain.
2. It can cause starvation for long processes if a large number of short processes keep arriving, leading to a longer waiting time for the long processes.
3. It can be unfair for longer processes, as they have to wait for shorter processes to complete before they get a chance to execute.
4. It is not suitable for real-time systems, as the execution time of processes cannot be predicted with certainty.
5. It may cause a delay in the response time of the system as the scheduler has to wait for the shortest job to arrive before starting its execution.
6. It can be complex to implement in a multi-core environment.
7. It can suffer from priority inversion, where a higher priority process is blocked by a lower priority process that holds a shared resource.

Code:

```
#include <stdio.h>
#include <stdlib.h>

//Function to swap two values
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

//Function to sort People according to their time taken for completion of puja
void SelectionSort(int arr[100][5], int n)
{
    int most_app, i, j;
    for (i = 0; i < n - 1; i++)
    {
        most_app = i;
        for (j = i + 1; j < n; j++)
        {
            //Checks the burst time first
            if (arr[j][1] < arr[most_app][1])
            {
                most_app = j;
            }

            //If time taken for puja(burst times) are equal , checks the prior-
ity(e.g. Differently-abled etc.)
            //Higher the number higher is the priority
```

```
            else if (arr[j][1] == arr[most_app][1])
            {
                if (arr[j][4] > arr[most_app][4])
                {
                    most_app = j;
                }
            }
        }
        //Swaps The Person id , priority and the time taken for puja(burst times)
        if (most_app != i)
        {
            swap(&arr[i][1], &arr[most_app][1]);
            swap(&arr[i][4], &arr[most_app][4]);
            swap(&arr[i][0], &arr[most_app][0]);
        }
    }
}

// Waiting Time = Turn Around Time(Time at which person leaves) – Burst Time(Time
taken for puja)
int WaitingTimeCalculation(int arr[100][5], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
    {
        arr[i][2] = 0;
        for (int j = 0; j < i; j++)
        {
            arr[i][2] += arr[j][1];
        }
        total += arr[i][2];
    }
    return total;
}

int main()
{

    int arr[100][5], n, total = 0;
    printf("Enter the number of processes: \n");
    scanf("%d", &n);

    printf("Enter the priority according to this scheme\n1 - Normal People\n2 -
Children\n3 - Senior Citizens\n4 - Specially Abled\n");

    printf("Enter the burst time and priority of each process \n");
    for (int i = 0; i < n; i++)
```

```
    {
        scanf("%d %d", &arr[i][1], &arr[i][4]);
        arr[i][0] = i + 1; // Allocating ID's to people
    }

    SelectionSort(arr, n);
    total = WaitingTimeCalculation(arr, n);
    printf("Total Waiting time:%d\n", total);

    float avg_wt = (float)total / n;

    int tot_tat = 0;
    //Printing the
    printf("PID\tBT\tWT\tTAT\tPriority\n");
    for (int i = 0; i < n; i++)
    {
        arr[i][3] = arr[i][1] + arr[i][2];
        tot_tat += arr[i][3];
        printf("%d\t%2d\t%2d\t%2d\t%2d\n", arr[i][0], arr[i][1], arr[i][2],
arr[i][3], arr[i][4]);
    }

    float avg_tat = (float)tot_tat / n;
    printf("Average Waiting Time= %f", avg_wt);
    printf("\nAverage Turnaround Time= %f\n", avg_tat);

    //Here as we have considered that all processes arrive at the same time , so
Turn around time is same as completion time

    // Gannt Chart

    printf(" ------------------------------------------------------------------------
---------\n");
    printf("|\t");
    for (int i = 0; i < n; i++)
    {
        //Printing the ID's of people
        printf("P%d\t|\t", arr[i][0]);
    }
    printf("\n");
    printf(" ------------------------------------------------------------------------
---------\n");
    printf("0\t\t");
    //Printing the completion times
    for (int i = 0; i < n; i++)
    {
        printf("%d\t\t", arr[i][3]);
```

```
    }
    printf("\n\n");

    return 0;
}
```

Output:

```
PS C:\Users\aspur\OneDrive\C PROGRAMS> cd "c:\Users\aspur\OneDrive\C PROGRAMS\" ; if ($?) { gcc OS5.c -o OS5 } ; if ($?)
 { .\OS5 }
Enter the number of processes:
5
Enter the priority according to this scheme
1 - Normal People
7 3
5 4
Total Waiting time:31
PID     BT      WT      TAT     Priority
2       1       0       1       4
1       4       1       5       4
5       5       5       10      4
3       5       10      15      2
4       7       15      22      3
Average Waiting Time= 6.200000
Average Turnaround Time= 10.600000
 ---------------------------------------------------------------------------
|      P2     |      P1     |      P5     |      P3     |      P4     |
 ---------------------------------------------------------------------------
0             1             5             10            15            22

PS C:\Users\aspur\OneDrive\C PROGRAMS>
```

Post – Lab Question:

Discuss how the following pairs of scheduling criteria conflict in certain settings
a. **CPU utilization** and **response time**
b. **Average turnaround time** and **maximum waiting time**
c. **I/O device utilization** and **CPU utilization**

a. CPU utilization can be maximized by reducing the extra overhead time required by context switching
So, if context switching is more ,lesser will be the CPU uitilization. Also, if the context switching is minimized the response time will be increased because of process executing afterwards needs to wait for completion of previous process.
E.g. SRTF – Pre-emptive algorithm
Shortest job first scheduling algorithm favours CPU utilization by selecting the job with the shortest burst time. However, this can result in longer response times for longer jobs, which can lead to lower response time efficiency. In this case, the criteria of CPU utilization and response time are in conflict.

B)

Average turnaround time is the average time for a process to complete, while maximum waiting time is the longest time a process waits in the ready queue . In some cases , optimizing for one of these criteria can result in a trade-off with other.
For example, if the system prioritizes reducing the average TAT , it may lead to longer maximum waiting times for processes. Conversely, if the system prioritizes reducing the maximum waiting time , it may lead to longer longer average TAT.
Thus they can be considered to be inversely proportional to each other.
E.g. SJF.
SJF is used to reduce average TAT however this may lead to increase in maximum waiting time .Lets take a case where we have 5 processes with the burst time of 1 process being 60 and others of 30 each. In this case , in SJF < to reduce the avg. TAT the processes with BT 30 will be executed first and the process with BT 60 last. This leads to reduction in avg TAT Time however max waiting time becomes 4 * 30 = 120 which is a significant increase.
This shows that they are inversely proportional

C)
Since the CPU waits for the process to finish its CPU and I/O time before moving on to the next one in
any non-preemptive approach, such as FCFS, the CPU's utilisation is reduced for the entire duration of a
process's I/O time. Therefore, for non-preemptive processes, I/O time and CPU utilization are inversely
proportional.