CPU SCHEDULER

Name : Arun Kumar
Enroll. No. : 22116016
Dept. and Year : ECE III
DATE : 20th June, 2024

INTRODUCTION

The aim of this project, "CPU Scheduler," is to design and implement an efficient CPU scheduling algorithm to optimize the allocation of CPU resources among various processes in a computer system. The project seeks to achieve the following objectives:

1.**Efficiency**: Minimize the overall system processing time and maximize CPU utilization by implementing a scheduling algorithm that efficiently handles process queues.

2.**Fairness**: Ensure that all processes are treated fairly by preventing any single process from monopolizing the CPU, thus avoiding starvation and ensuring equitable distribution of CPU time.

3.**Responsiveness**: Improve the system's responsiveness, particularly for interactive and real-time applications, by reducing the waiting time and turnaround time for processes.

By achieving these goals, the project aims to contribute to the development of more efficient and robust CPU scheduling mechanisms that enhance the overall performance and user experience of modern computing systems.

About Algorithms and their implementation in C++ code :

There are total of 6 algorithms implemented in this project:

1. **FCFS** : First-Come, First-Served (FCFS) is a simple CPU scheduling algorithm where processes are executed in the order they arrive. It’s non-preemptive, easy to implement, but can lead to the convoy effect.

   It’s Implementaion : The First-Come, First-Served (FCFS) algorithm schedules processes in the order they arrive. It’s non-preemptive, meaning each process runs to completion before the next starts. C++ code sorts processes by arrival time and calculates waiting and turnaround times based on when each process starts and finishes execution.

2. SJF : Shortest Job First (SJF) non-preemptive scheduling selects and executes the shortest job fully before moving to the next, without interruption once the job starts.

It’s Implementaion : The Shortest Job First (SJF) algorithm selects processes with the shortest burst time next. C++ code sorts processes by arrival and burst times, then iteratively picks the shortest available process. It’s non- preemptive, so once a process starts, it runs to completion, with waiting and turnaround times calculated accordingly.

3.SJF (Preemptive) : Also knows as SRTF (Sortest Remaining Time first) , selects the process with the shortest remaining time to execute, preempting the current process if a shorter job arrives.

It’s Implementaion : The Shortest Job First (Preemptive) algorithm, also known as Shortest Remaining Time First (SRTF), selects the process with the shortest remaining burst time at every time unit. C++ code updates current time, remaining burst times, and checks for process completion, ensuring accurate waiting and turnaround time calculations.

4.Priority (Non Preemptive) :Priority non-preemptive scheduling selects and executes the highest priority process fully before starting the next one, ensuring tasks with higher priority are completed first without interruption.

It’s Implementaion : The Priority (Non-Preemptive) algorithm schedules processes based on priority, with lower numbers indicating higher priority. C++ code sorts processes by arrival time and priority, then selects the highest priority process available. Once a process starts, it runs to completion, calculating waiting and turnaround times accordingly.

5.Priority (Preemptive): Priority preemptive scheduling selects processes based on priority, preempting the current process if a higher priority process arrives, ensuring higher priority tasks execute sooner.

It’s Implementaion : The Priority (Preemptive) algorithm schedules processes based on priority, preempting the current process if a higher priority one arrives. C++ code updates current time, remaining burst times, and checks for process completion, recalculating priorities dynamically. Waiting and turnaround times are computed after each process completes execution.

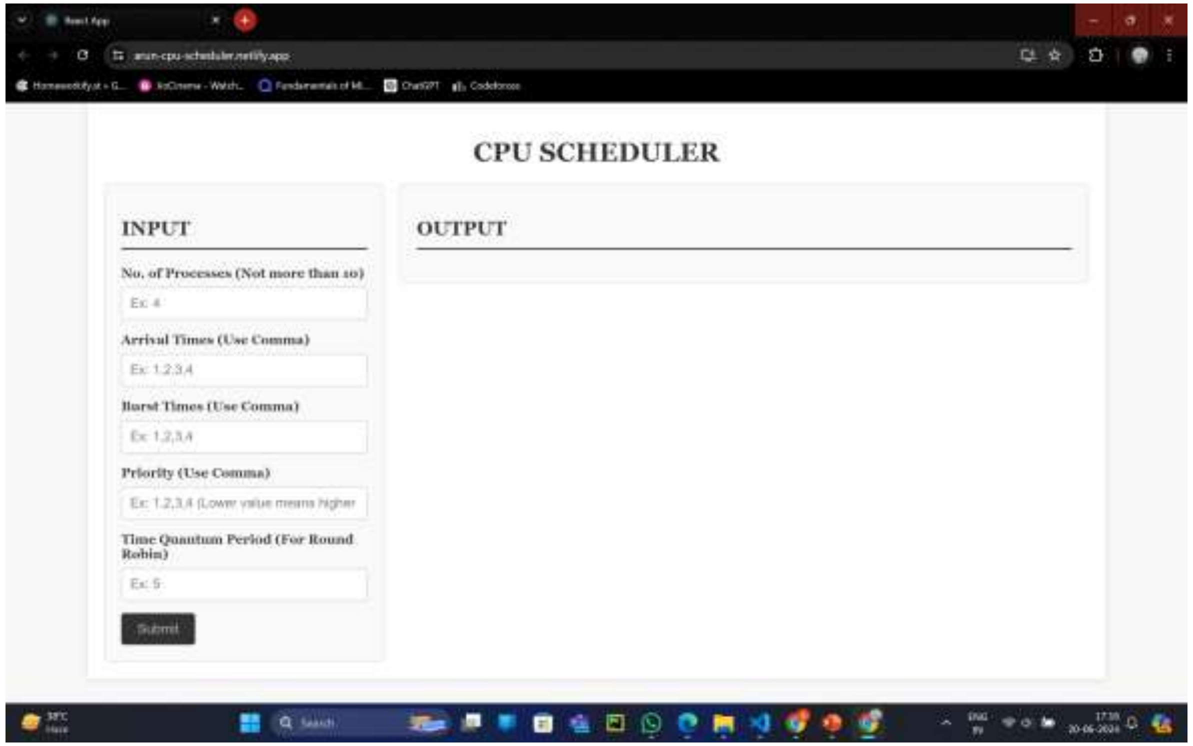
6.Round Robin: Round Robin CPU scheduling allocates a fixed time slice (quantum) to each process in a cyclic order, ensuring fair sharing of CPU time and reducing waiting time for all processes.

It’s Implementaion : The Round Robin algorithm assigns a fixed time quantum for process execution, cycling through processes in a circular queue. C++ code iterates over processes, executing each for the time quantum or until completion. Remaining times are updated, and waiting and turnaround times are calculated once all processes finish.

Note : I was able to implement all algorithm fine but in Round Robin i initially tried to maintain a ready queue but I do not know why that not worked then I had done with 2D vector.

FRONTEND

INITIALLY :



arun-cpu-scheduler.netlify.app

CPU SCHEDULER

INPUT

No. of Processes (Not more than 10)

Ex: 4

Arrival Times (Use Comma)

Ex: 1,2,3,4

Burst Times (Use Comma)

Ex: 1,2,3,4

Priority (Use Comma)

Ex: 1,2,3,4 (Lower value means higher

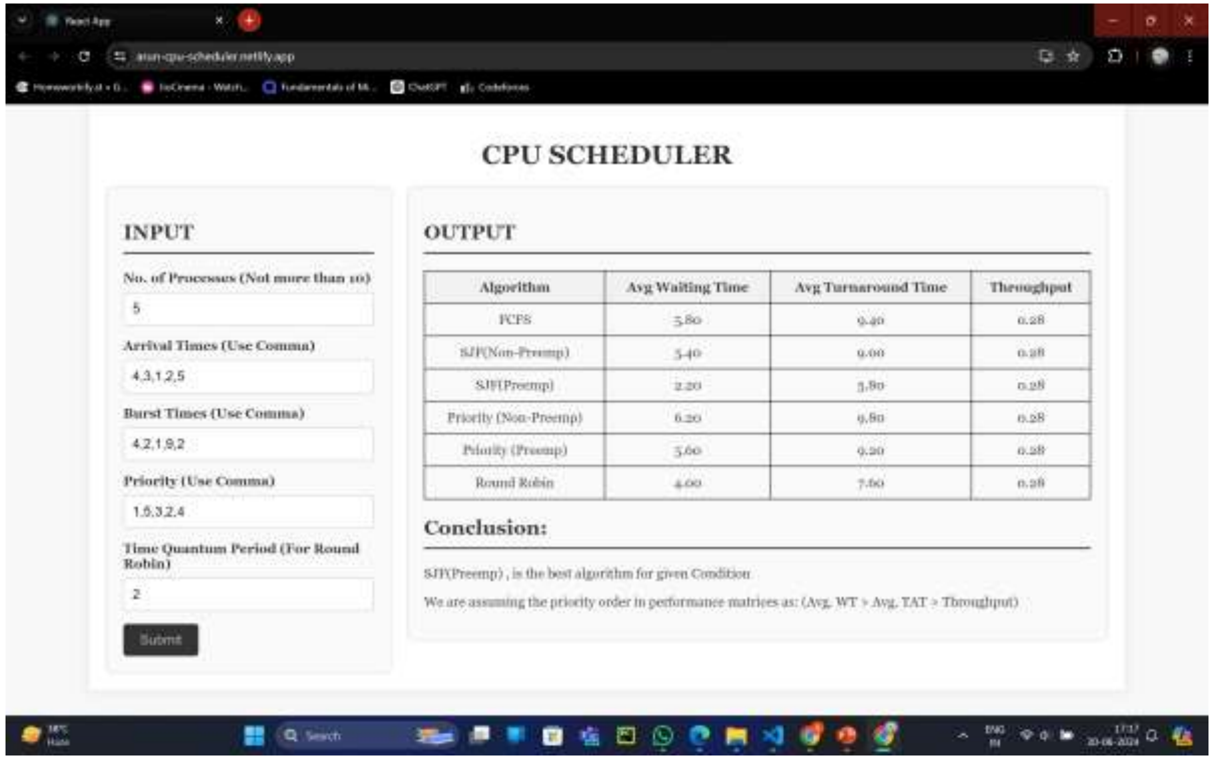
Time Quantum Period (For Round Robin)

Ex: 5

Submit

OUTPUT

After Submit:



arun-cpu-scheduler.netlify.app

CPU SCHEDULER

INPUT

No. of Processes (Not more than 10)

5

Arrival Times (Use Comma)

4,3,1,2,5

Burst Times (Use Comma)

4,2,1,9,2

Priority (Use Comma)

1,5,3,2,4

Time Quantum Period (For Round Robin)

2

Submit

OUTPUT

Algorithm	Avg Waiting Time	Avg Turnaround Time	Throughput
FCFS	5.80	9.40	0.28
SJF(Non-Preemp)	5.40	9.00	0.28
SJF(Preemp)	2.20	5.80	0.28
Priority (Non-Preemp)	6.20	9.80	0.28
Priority (Preemp)	5.60	9.20	0.28
Round Robin	4.00	7.60	0.28

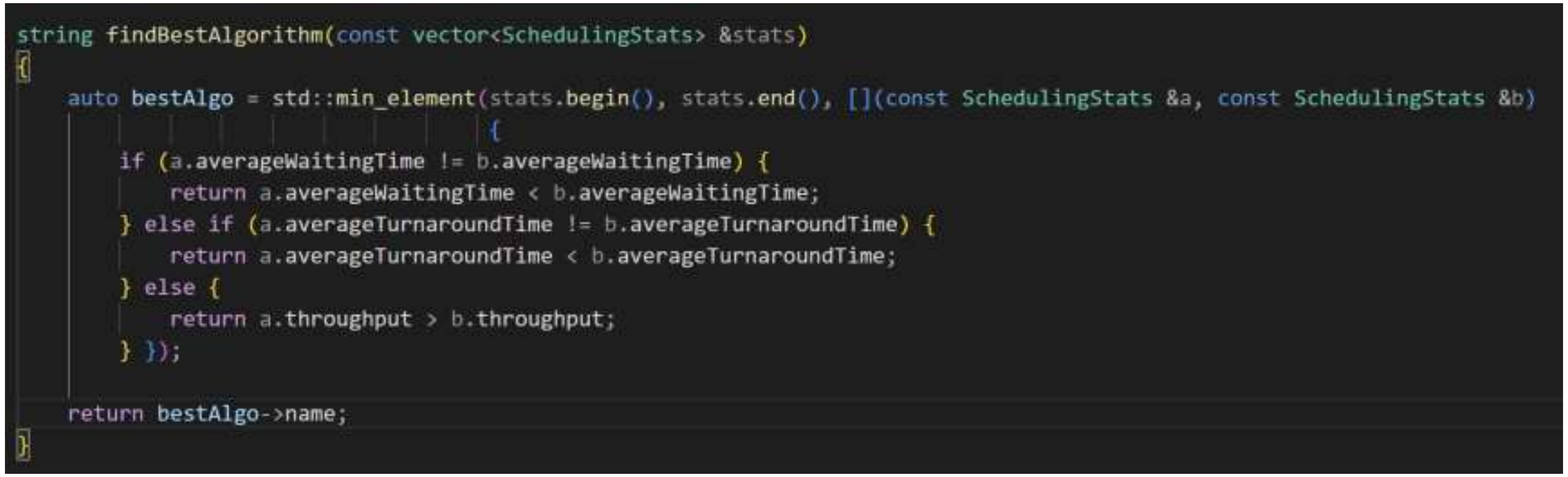
Conclusion:

SJF(Preemp) , is the best algorithm for given Condition

We are assuming the priority order in performance matrices as: (Avg. WT > Avg. TAT > Throughput)

BACKEND (means C++ file)

1. The C++ code starts with a Struct which store all the attributes of any process
2. In main function we are taking input through a input.txt file which takes required input for all algorithms
3. Then all algorithms run and give their respective Performance Matrices which in store in another vector of struct .
4. **Then we for our compare function which compare the performances of the matrices got for every algorithm.**



```cpp
string findBestAlgorithm(const vector<SchedulingStats> &stats)
{
    auto bestAlgo = std::min_element(stats.begin(), stats.end(), [](const SchedulingStats &a, const SchedulingStats &b)
                                                      {
        if (a.averageWaitingTime != b.averageWaitingTime) {
            return a.averageWaitingTime < b.averageWaitingTime;
        } else if (a.averageTurnaroundTime != b.averageTurnaroundTime) {
            return a.averageTurnaroundTime < b.averageTurnaroundTime;
        } else {
            return a.throughput > b.throughput;
        } });

    return bestAlgo->name;
}
```

Using this our code show which algorithm is best for our given constraints and conditions. (like arrival times, burst times,prioroties,time quantum)

Limitations :

Currently this program runs for not more than 10 operations due to fronted issue as the input section will not be able to fit large data set.

Challenges Faced :

1. I knew nothing about development.
2. Getting confused as cpu scheduler simulator available on google show different results for same input.
3. I think it’s very hard to learn html Css and javascript in 2 or 3 days and make a full stack web app
4. Not much resource on internet available about dev using C++ in backend.

RESOURCES :

1. https://www.youtube.com/playlist?list=PLBlnK6fEyqRitWSE_AyyySWfhRgyA-rHk
2. https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/
3. Chapter-4 of Linux Kernel Development by Robert Love (For implementation Details)
4. https://www.youtube.com/watch?v=I2UBjN5ER4s