Custom CPU Scheduling Simulator

Kotipalli Venkata Sriram (202311043)
Dept. of Computer Science
IIIT Vadodara - ICD
Diu, India

***Abstract*—This paper is regarding custom cpu scheduling which simulates algorithms based on process arrival time, burst time and plots a gannt chart based on the input. Provides a gui interface for simulating algorithms and calculates completion time, turnaround time and response time.**

***Index Terms*—arrival time, burst time, gannt chart, completion time, turnaround time, response time**

I. Introduction

CPU scheduling is one of the core responsibilities of an operating system. As modern computing systems continue to support a large number of applications simultaneously, scheduling plays a critical role in determining how efficiently system resources are utilized. Efficient scheduling improves CPU utilization, throughput, response time, and overall system performance.

The design of a CPU scheduler is influenced heavily by the workload, type of system, and fairness policies. For example, interactive systems focus more on responsiveness, whereas batch systems aim to reduce turnaround time. Each algorithm follows a different strategy to determine which job to run next and how long it should run.

This paper presents a **Custom CPU Scheduling Simulator** developed to help students, researchers, and software developers observe and analyze the behavior of scheduling algorithms. The simulator is particularly useful in an academic environment where understanding internal OS behavior visually is extremely beneficial.

The simulator supports multiple scheduling algorithms, generates real-time Gantt charts, computes essential timing metrics, and provides a functional graphical interface to interact with the system.

A. Need for a Scheduling Simulator

Most operating system textbooks explain CPU scheduling theoretically. However, many learners find it difficult to understand how preemption happens, how queues are maintained, and how context switching affects performance. A simulator provides:

- A dynamic visualization of each scheduling step
- A platform to test custom inputs
- Better understanding through trial and error
- Clear comparison between algorithms

Visual and interactive learning leads to a significantly stronger understanding compared to reading theory alone.

B. Goals of the Project

The main goals of this project include:

- Implementing multiple CPU scheduling algorithms
- Providing visual Gantt charts for understanding execution flow
- Calculating Completion Time, Waiting Time, Response Time, and Turnaround Time
- Allowing comparison of algorithm performance
- Creating an educational tool with an easy-to-use GUI

This ensures that the simulator can be used in labs, universities, and research studies.

II. Background and Related Work

CPU scheduling has been studied extensively for decades and forms the foundation of multitasking operating systems. Earlier operating systems such as UNIX used simple schedulers, while modern systems such as Linux rely on much more complex algorithms like the Completely Fair Scheduler (CFS).

Several simulators exist online, but they often lack:

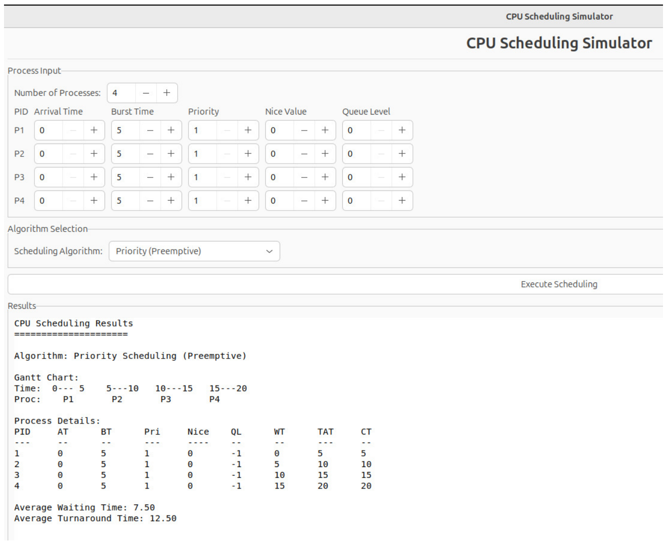
- Multi-algorithm comparison
- Real-time visualization
- Detailed process metrics
- Support for preemptive vs non-preemptive modes

The simulator created in this work addresses these limitations through a modular architecture that enables high flexibility and clarity.

III. Simulation Framework

A. System Architecture

The simulator is divided into three major components, shown in Fig. 1.


CPU Scheduling Simulator

CPU Scheduling Simulator

Process Input

Number of Processes: 4

PID	Arrival Time	Burst Time	Priority	Nice Value	Queue Level
P1	0	5	1	0	0
P2	0	5	1	0	0
P3	0	5	1	0	0
P4	0	5	1	0	0

Algorithm Selection

Scheduling Algorithm: Priority (Preemptive)

Results

```
CPU Scheduling Results
=====================

Algorithm: Priority Scheduling (Preemptive)

Gantt Chart:
Time:  0--- 5    5---10   10---15   15---20
Proc:    P1        P2        P3        P4

Process Details:
PID     AT      BT      Pri     Nice    QL      WT      TAT     CT
---     --      --      ---     ----    --      --      ---     --
1       0       5       1       0       -1      0       5       5
2       0       5       1       0       -1      5       10      10
3       0       5       1       0       -1      10      15      15
4       0       5       1       0       -1      15      20      20

Average Waiting Time: 7.50
Average Turnaround Time: 12.50
```

Fig. 1. Overall System Architecture of the Simulator

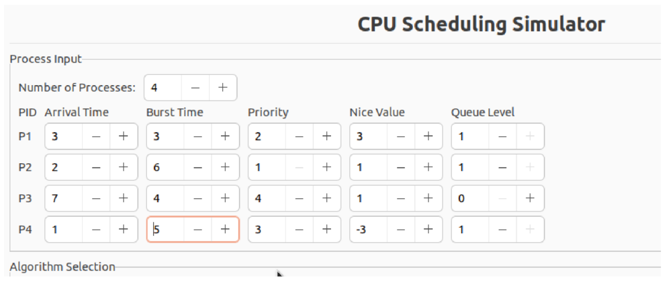
- **Input:** It accepts process parameters such as Process ID, arrival time, burst time, priority, time quantum and nice value.
- **Scheduling File:** It contains the logics of all scheduling algorithms.
- **Output:** Visually represent and plot Gantt charts, tables, and Completion time, Response time, Turnaround time.

B. Graphical User Interface

B. Graphical User Interface

- Process input table
- Drop down algorithm selection
- Buttons to generate charts
- Output table containing all timing results

An example of the GUI is shown in Fig. 2.


CPU Scheduling Simulator

Process Input

Number of Processes: 4

PID	Arrival Time	Burst Time	Priority	Nice Value	Queue Level
P1	3	3	2	3	1
P2	2	6	1	1	1
P3	7	4	4	1	0
P4	1	5	3	-3	1

Algorithm Selection

Fig. 2. Simulator Graphical User Interface

C. Process Representation

Each process is stored in a structure similar to a PCB.It contains :

- C. Process Representation
- Arrival Time
- Burst Time
- Remaining Time
- Priority
- Completion Time
- Turnaround Time
- Waiting Time
- Response Time

Table Igives an example of the date used for testing

TABLE I
SAMPLE INPUT PROCESS TABLE

PID	Arrival Time	Burst Time	Priority
P1	0	7	3
P2	2	4	1
P3	3	6	2
P4	5	5	4

IV. Implementation Details

A. Data Structures Used

The simulator uses different data structures for different parts of the scheduling logic :

- **Vectors/Arrays** to store all process information
- **Queues** for algorithms like FCFS and Round Robin
- **Min heaps** for SRTF and CFS
- **Multiple ready queues** for MLQ and MLFQ

This makes it easy to add new scheduling algorithms in the future.

B. General Scheduling Loop

All algorithms follow a similar simulation loop:

1) Initialize clock = 0
2) Insert available processes into ready queue
3) Select next process based on scheduling policy
4) Update Gantt chart timeline
5) Record completion and timing metrics

C. Pseudocode Example: Round Robin

A simplified version of the Round Robin scheduling algorithm is given below:

Algorithm 1 Round Robin Scheduling

```
1: Initialize ready queue
2: Set time = 0
3: while unfinished processes exist do
4:    for each process in queue do
5:       Run process for min(quantum, remaining time)
6:       Update Gantt chart
7:       if process not finished then
8:          Requeue process
9:       else
10:         Record completion metrics
11:      end if
12:   end for
13: end while
```

D. MLFQ Queue Movement

MLFQ queue movement rules:

- New processes start in Queue 1 (highest priority)
- If quantum expires $\rightarrow$ move to next lower queue
- If a process waits too long $\rightarrow$ boosted upward

MLFQ offers good balance between fairness and efficiency.

V. Methodology

This portion concludes and explains how each algorithm works in the simulator. Mainly the implementation part. All the Algorithms in this follow a simulation loop, But get differentiate in the policy used to choose the next process.

A. First-Come, First-Served (FCFS)

FCFS is a simple scheduling algorithm. Where the processes are executed linearly. Depends upon the order of arrival. And based on this, The gantt chart is developed to help us to visualize the time stamps of the execution of each process. And the Ready Queue is maintained as the FIFO queue.And there are no interruptions until the completion of the process once it starts .

B. Shortest Job First (SJF)

The shortest job first scheduling algorithm depends on the burst time of the Process. This reduces the waiting time of the process and for shorter tasks also. It comes out with the better throughput it is non- pre-emptive scheduling Algorithms once a job gets started it can't be interrupted until the completion.

C. Shortest Remaining Time First (SRTF)

SRTF is a pre-emptive scheduling algorithm where some of the processes may be interrupted. For suppose a newly arrived process has a lesser completion time. This algorithm gives us the better and best average waiting time. But may cause us to starve

D. Priority Scheduling

This scheduling algorithm is divided into the two types: 1.pre-emptive priority scheduling 2.Non pre-emptive priority scheduling The scheduler always chooses the process with the highest priority value. And each process is assigned a priority. Pre-emptive priority scheduling interrupts the executing process if a highest priority process arrives.

E. Round Robin (RR)

Round Robin scheduling algorithm improves the fairness by assigning a fixed time quantum. And it is ideal for time sharing. It causes more context switching when the smaller quantum increases its responsiveness.

F. Multilevel Queue Scheduling (MLQ)

In MLQ, processes are divided into queues based on a fixed classification such as:

- System processes (Top priority)
- Interactive processes
- Background processes (Lowest)

Each queue may use a different scheduling algorithm.

G. Multilevel Feedback Queue (MLFQ)

In MLFQ, processes migrate between queues. This gives more CPU time to short interactive tasks while pushing long-running tasks downward.

H. Completely Fair Scheduler (CFS)

Linux uses CFS as its main scheduler. CFS assigns each task a *virtual runtime* (vruntime) and always picks the task with the lowest vruntime. The vruntime is updated as:

$$vruntime = vruntime + \frac{actual\_runtime}{priority\_weight}$$

VI. Results and Discussion

To elevate the performance of the scheduler, some of the experiments are done using different types of processes. And the scheduler gave us the Exact results as the outputs.consistently performed the different algorithms . And Visualization of executed steps.

A. Gantt Chart Results

The simulator generates dynamic Gantt charts that visually show context switching, preemption, and execution flow. Fig. 3 shows an example.

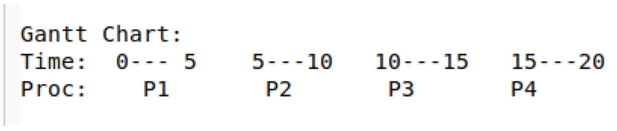

```
Gantt Chart:
Time:  0--- 5    5---10   10---15   15---20
Proc:    P1        P2       P3        P4
```

Fig. 3. Final Gantt Chart Representation from the Simulator

B. Performance Metrics

The following formulas are used:

- Turnaround Time (TAT):

$$TAT = Completion\ Time - Arrival\ Time$$

- Waiting Time (WT):

$$WT = TAT - Burst\ Time$$

- Response Time (RT):

$$RT = First\ Execution - Arrival\ Time$$

C. Results Comparison of Algorithms

A visual comparison graph is shown in Fig. 4.

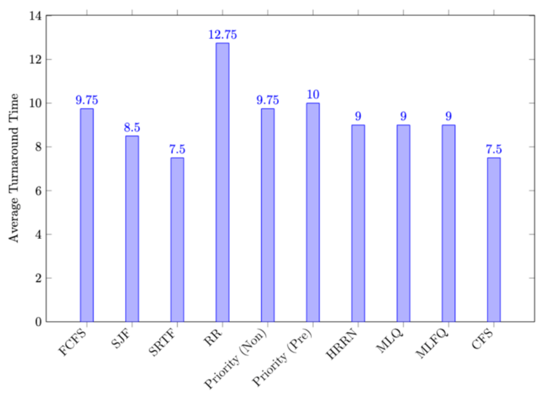


Fig. 4. Comparison of Average TAT

D. Analysis

From repeated experiments, several results are found:

– **SRTF provides the lowest waiting time** in most cases because it prioritises short tasks.
– **FCFS performs worst if a process is too large which has lot of burst time**. Process with large burst times delay shorter ones causing starvation lock i.e a process waiting for indefnite time for its execution.
– **RR performance depends on the quantum**. A very large time quantum makes RR similar to FCFS. So we need to choose time quantum cleverly.
– **MLFQ adapts to dynamic workloads**, making it ideal for modern systems.
– **CFS is the most balanced**, when dealing with large numbers of processes, also a real world linux based simulator.

E. Evaluation

Table II shows results from running 10 processes.

TABLE II
RESULTS FOR LARGE OF NUMBER PROCESSES (10 PROCESSES)

Algorithm	Avg TAT	Avg WT	Avg RT
FCFS	32.4	18.1	9.4
SRTF	22.9	11.0	4.7
RR	28.7	16.2	8.1
MLFQ	24.3	13.8	5.1
CFS	23.9	12.4	6.0

The experiments show that the simulator handles both small and large number of processes correlcty.

VII. LIMITATIONS

The simulator is right and efficient but a few limitations exist:

– Do not represent I/O bound processes.
– Do not model thread level scheduling.
– Does not simulate multi-core CPU execution.

VIII. FUTURE ENHANCEMENTS

The improvements for future are:

– Adding support for real time scheduling algorithms.
– I/O wait events and interrupts.
– Implementing multi core scheduling simulation.
– Web based deployment for online access.
– Export of Gantt charts as file.

These enhancements will significantly increase usability and expand research applications.

IX. CONCLUSION

This project gives a brief view on cpu scheduling algorithms and how processes utilise cpu and this project we implemented 9 scheduling algorithms in which cfs is a real world based linux algoerithm which has nice values ranging from -20 to +19 and schedules processes based on priority which convert nice values to weights and then according to vruntime (virtual runtime) process is excecuted and gant chart is plotted and we will find best values for waiting time, turnarouund time and visually represents the given graph.