PROJECT REPORT

Implementation and Analysis of CPU Scheduling Algorithms

Team Members

Name	Role	Responsibility
Teja Piepur Chakravarthi	Developer, Tester & Report	Core algorithm implementation, integration, result analysis, Test case generation & Report
Yashvikumari Bhagat	Documentation & Validation Engineer	Input file design, performance metrics computation, report preparation & validation of results.
Harshitha Arugonda	Tester & Documentation	Test case generation, documentation, report preparation, validation of results.

Introduction

The Project main goal is to design and implement 5 different ***CPU scheduling algorithms*** and analyses their performance using the following scheduling metrics: **1. Average waiting time, 2. turnaround time, and 3. response time**. The Schedule will read task data from external files and simulate the following algorithms: **1. First-Come, First-Served (FCFS)**, **2. Shortest Job First (SJF)**, **3. Priority Scheduling**, **4. Priority with Round Robin (PRR for tasks with equal priority) & 5. Round Robin (RR)** (Time Quantum is set to 10ms). All the processes information is randomly generated, and all the 5 algorithms will have the same randomly generated process information. The test cases are created to check different algorithms' performance effectively and we use the final result of the execution to pick the best performing algorithm.

- **Why is it Important?**

CPU scheduling is a fundamental function of an operating system that decides which process in the **ready queue** should be assigned the CPU. Since a **single-core CPU** can run only one process at a time, efficient scheduling ensures optimal resource utilization and fairness among processes. It i**mproves CPU utilization**, **throughput, response time**, and **overall performance**. Algorithms such as First-Come, First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR) handle different workloads and objectives. Choosing the right strategy helps minimize waiting and turnaround times, manage high-priority tasks efficiently, and maintain system responsiveness. Thus, CPU scheduling is crucial for achieving efficiency and balanced performance across single-core and multiprocessor systems.

Background

To understand the problem of CPU scheduling, it is essential to be familiar with the fundamental concepts, objectives, and types of scheduling algorithms used in operating systems. CPU scheduling is the process of determining which process from the **ready queue** should be assigned to the CPU next. The performance of the system depends greatly on how effectively this scheduling is managed.

∗ **Key Terminology**

- **Ready Queue:** A queue that holds all processes waiting to be executed by the CPU.

- **Arrival Time:** The time at which a process enters the ready queue and becomes ready for execution.
- **Burst Time:** The total time required by a process to complete its execution on the CPU.
- **Turnaround Time:** The total time from when a process arrives to when it completes execution.
- **Waiting Time:** The time a process spends waiting in the ready queue before being executed.
- **Response Time:** The time between a process's submission and the production of its first response.
- **Context Switch:** The act of saving the state of the currently running process and loading the state of the next process to be executed.

∗ **Scheduling Objectives**

The main goals of CPU scheduling are to improve system efficiency and responsiveness.

- **Maximize CPU Utilization:** Keep the CPU as busy as possible to avoid idle time.
- **Maximize Throughput:** Complete the maximum number of processes in a given time period.
- **Minimize Turnaround Time:** Reduce the total time taken from process submission to completion.
- **Minimize Waiting Time:** Limit the time a process waits in the ready queue.
- **Minimize Response Time:** Especially for interactive systems, reduce the time it takes to start responding.
- **Fairness:** Ensure each process receives a fair share of CPU time, preventing starvation of lower-priority processes.

Implementation

a. Solutions to the Problem

In this project, multiple CPU scheduling algorithms were implemented and compared to analyze their effectiveness under different conditions. Each algorithm provides a distinct approach to determining the order of process execution based on factors such as arrival order, burst time, priority, or time quantum. There is no single "best" solution to the CPU scheduling problem; instead, each algorithm optimizes specific performance metrics depending on system goals.

The implemented solutions include:

- **First-Come, First-Served (FCFS):** Executes processes in the order they arrive in the ready queue.
- **Shortest Job First (SJF):** Selects the process with the shortest CPU burst time, minimizing average waiting time.
- **Priority Scheduling:** Allocates CPU time based on process priority values, where a higher value represents higher importance.
- **Round Robin (RR):** Assigns each process a fixed time quantum (10 milliseconds) in a circular sequence, ensuring fairness.
- **Priority with Round Robin:** Combines both approaches by scheduling processes according to priority and applying round-robin within processes of equal priority.

Every algorithm has their own scheduling objectives such as minimizing waiting time, maximizing throughput, or maintaining fairness among processes. After testing on these algorithms across multiple test cases, the project evaluates which performs best under varying workloads and process characteristics

b. How It Was Implemented

i. Programming Language

The project was implemented using **C++**, chosen for its efficiency, speed, and low-level control over system resources, which makes it ideal for simulating process scheduling.

ii. Operating System

The implementation and testing were conducted on a **CSE machine environment** running both **Windows and macOS**

iii. Code Implementation

The Project is built with 6 Programing file, among these 6 files, 1 file is a header file – which is commonly used among all the other 5 files. The common header file is a time_reader file that is used to print the processing time of each process. The other 5 files are the CPU scheduling algorithm files coded in C++.

- **Header File – time_reader.h**: The main task of this header file is to calculate different metrics and display them in the execution terminal. This contains 2 functions.
    1) A ***parseScheduleFile* ()**: function that reads input from the schedule file and initializes task values.
    2) A **printMetrics ()**: function that calculates and displays **waiting time, turnaround time, and response time** for each task, along with their averages.
- **First Come First Serve (FCFS) – fcfs.cpp**
  The 'fcfs.cpp' is used to implement the **First Come First Serve** - CPU scheduling algorithm.
- **Shortest Job First (SJF) – sjf.cpp**
  The Shortest Job First – 'sjf.cpp' file is used to execute a non-preemptive scheduling algorithm.
- **Priority Scheduling – priority.cpp**
  The "priority.cpp" file is used to implement the **Non-Preemptive Priority Scheduling** algorithm.
  At the beginning of the algorithm the algorithm starts to sort the processes by considering their priority values, The task with the highest priority value is consider processing at the beginning of the algorithm. The process is sorted in descending order.
- **Round Robin (RR) Scheduling – rr.cpp**
  This file implements the **Round Robin** scheduling algorithm with a **time quantum of 10 ms**.
- **Priority Round Robin Scheduling – priority_rr.cpp**
  The 'pirority_rr.cpp' file as the name suggests is a mix of both priority scheduling algorithm and the round robin algorithm.

c. Test Cases

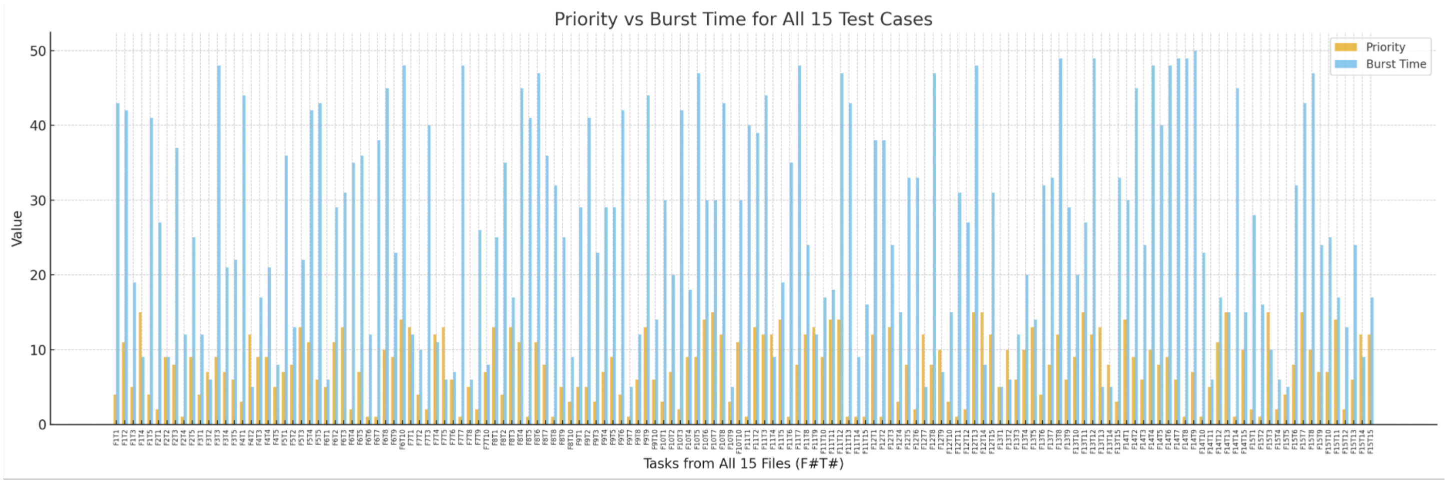
Multiple test cases were created for evaluate and compare the performance of the CPU Scheduling algorithms with given specification. every test case contained the different workload, containing process with different **priorities and CPU burst times**. The priorities ranges from 1 to 15, where higher number represents a higher relative priority. For the **Round Robin and Priority with RR algorithms** has a **time quantum of 10 milliseconds.**

Multiple Test files were created:

- **5 files** with **5 processes**
- **5 files** with **10 processes**
- **5 files** with **15 processes**

- **5 files** with **20 processes**

Test Case Graph:



Following is graph description:

- This graph shows the priority and burst time value for all 15 test files
- X - axis: Task name (T1, T2, …)
- Y - axis: Display the priority and Burst time values.

Experimental Results

- **Results 15 Test Cases using CPU Scheduling Algorithm:**
  Average turnaround time is chosen as the primary indicator of scheduling efficiency out of the three-performance metrics: **average waiting time, turnaround time, and response time**.

File Name	**FCFS WT (ms)**	**SJF WT (ms)**	**Priority WT (ms)**	**RR WT (ms)**	**PRR WT (ms)**	**FCFS TAT (ms)**	**SJF TAT (ms)**	**Priority TAT (ms)**	**RR TAT (ms)**	**PRR TAT (ms)**	**FCFS RT (ms)**	**SJF RT (ms)**	**Priority RT (ms)**	**RR RT (ms)**	**PRR RT (ms)**
File 1	69.00	43.40	48.60	84.20	56.60	99.80	**74.20**	79.40	115	87.40	69.00	43.40	48.60	19.80	42.00
File 2	44.20	29.80	42.40	56.20	42.40	66.20	**51.80**	64.40	78.20	64.40	44.20	29.80	42.40	19.40	42.40
File 3	36.60	24.80	54.80	48.80	54.80	58.40	**46.60**	76.60	70.60	76.60	36.60	24.80	54.80	17.60	54.80
File 4	49.20	19.80	24.20	39.80	26.20	68.20	**38.80**	43.20	58.80	45.20	49.20	19.80	24.20	17	22.80
File 5	53.80	46.40	55.20	90.40	55.20	85.00	**77.60**	86.40	121.60	86.40	53.80	46.40	55.20	20.00	55.20
File 6	116.80	100.9	151.20	190.50	152.20	147.10	**131.20**	181.50	220.80	182.50	116.80	100.90	151.20	41.40	151.00
File 7	77.40	42.10	47.60	76.60	49.60	94.80	**59.50**	65.00	94.00	67.00	77.40	42.10	47.60	41.00	44.40
File 8	148.40	107.7	140.20	213.80	148.50	179.60	**138.9**	171.40	245	179.70	148.40	107.7	140.20	45.00	132.10
File 9	132.00	84.90	131.90	170.50	134.90	158.80	**111.7**	158.70	197.30	161.70	132.00	84.9	131.90	43.50	129.80
File 10	136.80	99.60	139.40	181.20	138.90	166.30	**129.10**	168.90	210.70	168.40	136.80	99.60	139.40	44.50	136.60
File 11	214.40	136.27	196.73	260.73	198.53	242.40	**164.27**	224.73	288.73	226.53	214.40	136.27	196.73	69.20	171.13
File 12	193.40	129.40	180.40	255.80	185.13	220.07	**156.07**	207.07	282.47	211.80	193.40	129.40	180.40	66.00	168.87

File 13	139.40	97.07	160.20	177.07	162.40	162.00	**119.67**	182.80	199.67	185.00	139.40	97.07	160.20	60.87	155.67
File 14	263.53	172.67	220.40	339.13	227.80	297.13	**206.27**	254.00	372.73	261.40	263.53	172.67	220.40	68.9	208.53
File 15	148.67	96.67	161.20	182.00	162.07	169.73	**117.73**	182.27	203.07	183.13	148.67	96.67	161.20	63.67	157.87

Interpretation of the results:

- **SJF** consistently having the **lowest waiting and turnaround times**, demonstrating its efficiency when job lengths are known before execution.
- **FCFS** had moderate results, but when lengthy processes arrived early, waiting times increased due to the **convoy effect.**
- **Priority Scheduling** changes depending on priority distribution; it recommended high-priority tasks but increased waiting times for lower-priority process.
- **Round Robin (RR)** ensured fairness among processes but had higher turnaround times due to frequent context switches.
- **Priority with Round Robin (PRR)** achieved a **balance between fairness and responsiveness**, improving over RR in average response time while maintaining stable turnaround performance.

Conclusion

Based on the **results** obtained from the experiments, it was observed that **Shortest Job First (SJF)** always resulted in the **lowest average turnaround time** for all the test cases. The efficiency achieved because **SJF** always chooses the process with the shortest burst time, thereby **reducing the waiting time in the ready queue** and thus the total completion time taken by processes.

Nevertheless, the calculation of burst times must be known in advance in SJF, which may not always be known in real-world scenarios. Despite the limitation, it has remained one among the most efficient algorithms in handling batches and static workloads where process sizes are known in advance.

The **Round Robin (RR) and Priority with Round Robin (PRR)** algorithms give much better equality and responsiveness and **more applicable to interactive and time-sharing systems**.

On the other hand, the implementation of **FCFS and Priority Scheduling** was relatively simpler, yet **less efficient** because of **large average waiting times and convoy and starvation problems**.

References

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). John Wiley & Sons. ISBN 978-1-118-06333-0.

2. MIT OpenCourseWare. (2012). *6.828 – Operating System Engineering, Fall 2012: Projects and Labs.* Massachusetts Institute of Technology.

3. The FreeBSD Project. (n.d.). *FreeBSD Operating System.*