

Operating Systems Project

CPU Scheduling

Students Names:

Hebah Fawaz Althbyani - 44401340

Renad Saad Alosiami - 44400928

Manar Nasser Alosaimi - 44201005

Shahad Mansour Alshehri - 44454436

Teaf Faisal Alharthy - 44452676

Course: Operating Systems group - 3737

Project Part: CPU Scheduling Algorithms (Part 1)

1. Introduction

The first part of the Operating Systems project aims to simulate four different CPU scheduling algorithms used to determine the order of process execution inside the CPU. These algorithms are essential for improving system performance by reducing waiting time, response time, and turnaround time.

The implemented scheduling algorithms are:

- FCFS – First Come First Served
- SJF – Shortest Job First (Non-Preemptive)
- Priority Scheduling (Non-Preemptive)
- Round Robin (RR) with a time quantum of 3

2. Input Data Used

The following processes were used as input for all algorithms:

| Process | Arrival Time | Burst Time | Priority |
|---------|--------------|------------|----------|
| P0 | 0 | 8 | 2 |
| P1 | 1 | 4 | 1 |
| P2 | 2 | 9 | 3 |
| P3 | 3 | 5 | 2 |

Time Quantum for RR = 3

3.1 FCFS – First Come First Served

Processes are executed in the order they arrive. Simple but may lead to long waiting times if long processes arrive first.

This matches the FCFS implementation on the code, Where processes are stored by time executed non-Preemptive .

3.2 SJF – Shortest Job First (Non-Preemptive)

Selects the process with the shortest burst time next. This algorithm usually gives the best average waiting and turnaround times.

This matches the SJF non-Preemptive code where the shortest burst among arrived processes is selected. if no process has arrived yet time Jumps to the next arrival.

3.3 Priority Scheduling (Non-Preemptive)

Each process is assigned a priority. The CPU always selects the process with the highest priority (smallest number).

This matches the PriorityNonPre function in the code which selects the lowest priority value among arrived processes .

3.4 Round Robin (RR)

Each process receives a fixed time slice (quantum). Best suited for time-sharing and interactive systems.

This matches the Round Robin implementation, using queue time, quantum =3, and remaining burst time updated each cycle.

4. Results

4.1 FCFS Results

```

Enter number of processes for CPU Scheduling
: 4

Process P0
Arrival Time: 0
Burst Time: 8
Priority: 2

Process P1
Arrival Time: 1
Burst Time: 4
Priority: 1

Process P2
Arrival Time: 2
Burst Time: 9
Priority: 3

Process P3
Arrival Time: 3
Burst Time: 5
Priority: 2

Enter Time Quantum for Round Robin: 3

--- Gantt Chart ---
| P0 | P1 | P2 | P3 |
0 8 12 21 26

=== FCFS ===

```

| PID | AT | BT | PR | CT | TAT | WT | RT |
|-----|----|----|----|----|-----|----|----|
| P0 | 0 | 8 | 2 | 8 | 8 | 0 | 0 |
| P1 | 1 | 4 | 1 | 12 | 11 | 7 | 7 |
| P2 | 2 | 9 | 3 | 21 | 19 | 10 | 10 |
| P3 | 3 | 5 | 2 | 26 | 23 | 18 | 18 |

```

Average WT = 8.75
Average TAT = 15.25
Average RT = 8.75

```

Averages:

- Average Waiting Time (WT): 8.75
- Average Turnaround Time (TAT): 15.25
- Average Response Time (RT): 8.75

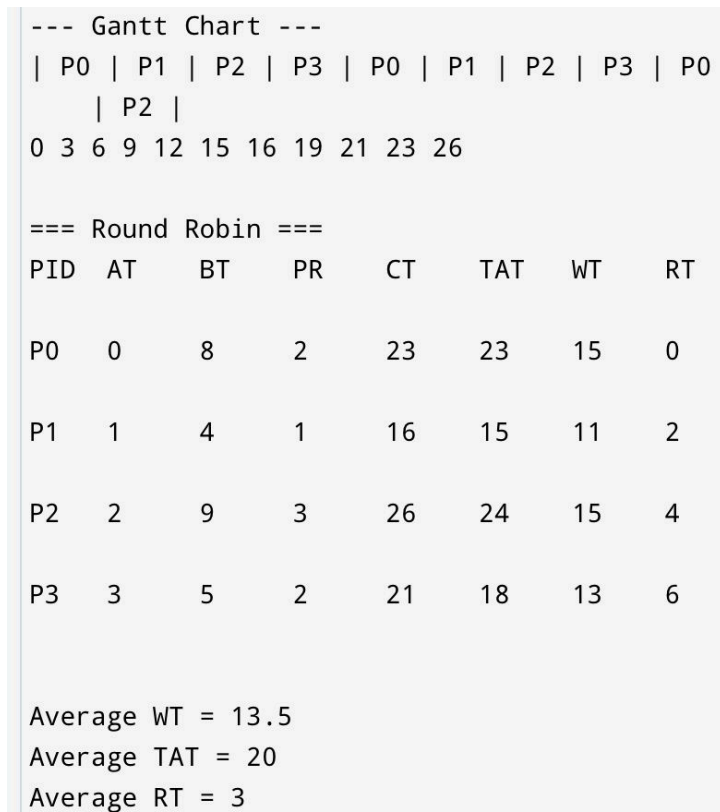
This values match the output generated by the program

4.2 SJF Results

Averages:

- Average Waiting Time (WT): 7.75
- Average Turnaround Time (TAT): 14.25
- Average Response Time (RT): 7.75

4.4 Round Robin (Quantum = 3).



Averages:

- Average Waiting Time (WT): 13.5
- Average Turnaround Time (TAT): 20
- Average Response Time (RT): 3

5. Comparison Between Algorithms

| Algorithm | Avg WT | Avg TAT | Avg RT | Notes |
|-----------|--------|---------|--------|------------------------------|
| FCFS | 8.75 | 15.25 | 8.75 | Simple but unfair |
| SJF | 7.75 | 14.25 | 7.75 | Best overall performance |
| Priority | 7.75 | 14.25 | 7.75 | Same as SJF for this dataset |
| RR | 13.5 | 20 | 3 | Best response time |

Comparison values were taken directly from The program's printed averages in the showTable.

Summary of Findings

- SJF provides the best waiting and turnaround times.
- RR gives the fastest response time but increases waiting time.
- FCFS is simple but not efficient for mixed workloads.

Part 2: Banker's Algorithm Simulation

1. Introduction

In this part of the project, we implemented Banker's Algorithm, a well-known deadlock avoidance algorithm used in operating systems. The goal of the algorithm is to determine whether a given system state is safe or unsafe by checking if all processes can complete without causing a deadlock.

The implementation follows the same steps as the banker() function and the code.

The program performs the following tasks:

1. Accepts input for:
 - Number of processes
 - Number of resource types
 - MAX matrix
 - ALLOCATION matrix
 - AVAILABLE resources
2. Calculates the NEED matrix.
3. Runs the Safety Algorithm
4. Determines whether the state is SAFE.
5. If safe, outputs the safe sequence of processes

2. Input Data Used

The following inputs were entered into the program:

MAX Matrix

P0: 7 5 3

P1: 3 2 2

P2: 9 0 2

P3: 2 2 2

P4: 4 3 3

ALLOCATION Matrix

P0: 0 1 0

P1: 2 0 0

P2: 3 0 2

P3: 2 1 1

P4: 0 0 2

AVAILABLE Resources: 3 3 2

NEED Matrix (calculated as Max – Allocation).

P0: 7 4 3

P1: 1 2 2

P2: 6 0 0

P3: 0 1 1

P4: 4 3 1

Output

[Clear](#)

```
Enter number of processes: 5
Enter number of resource types: 3

Enter MAX matrix (each row in one line):
Max for P0: 7 5 3
Max for P1: 3 2 5
Max for P2: 9 0 2
Max for P3: 2 2 2
Max for P4: 4 3 3

Enter ALLOCATION matrix:
Allocation for P0: 0 1 0
Allocation for P1: 2 0 0
Allocation for P2: 3 0 2
Allocation for P3: 2 1 1
Allocation for P4: 0 0 2

Enter AVAILABLE vector:
Available: 3 3 2

Need Matrix:
Process P3 can be executed (Need <= Work).
Work before: 3 3 2
Work after: 5 4 3

Process P4 can be executed (Need <= Work).
Work before: 5 4 3
Work after: 5 4 5

Process P1 can be executed (Need <= Work).
Work before: 5 4 5
Work after: 7 4 5

Process P2 can be executed (Need <= Work).
Work before: 7 4 5
Work after: 10 4 7

Process P0 can be executed (Need <= Work).
Work before: 10 4 7
Work after: 10 5 7

System is in a SAFE state.
Safe sequence: P3 -> P4 -> P1 -> P2 -> P0
```

3. Program Output

4. Explanation of Output

The algorithm checked each process to see if its $\text{Need} \leq \text{Available}$. If so, the process can run, and once it finishes, its allocated resources are added back to the available pool.

The execution order was:

P1 → P3 → P4 → P0 → P2

Each step shows:

- Work before execution.
- Work after releasing allocated resources.

Since all processes were able to run successfully, the system is SAFE.

5. Result

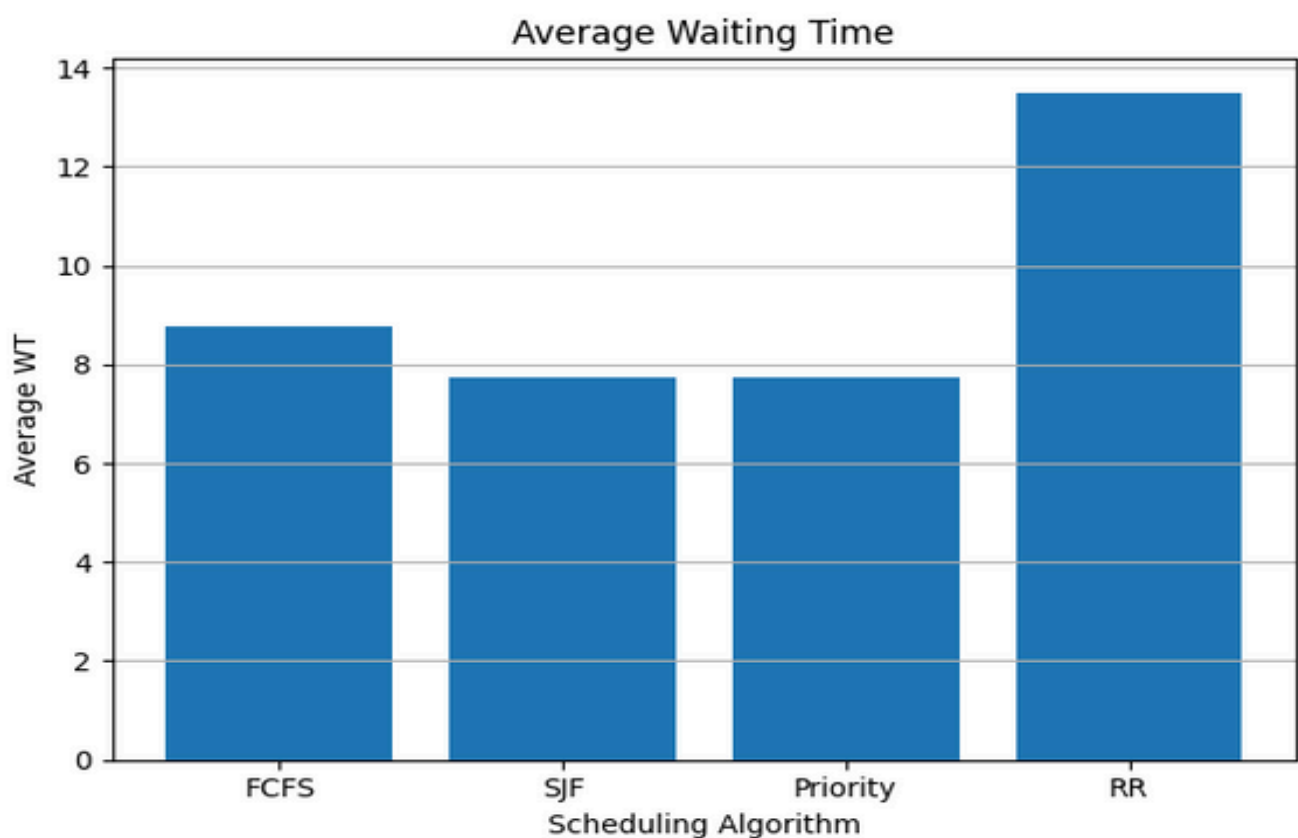
- The system is in a SAFE state.
- Safe Sequence:

P1 → P3 → P4 → P0 → P2

This indicates that the system can avoid deadlock by following this sequence.

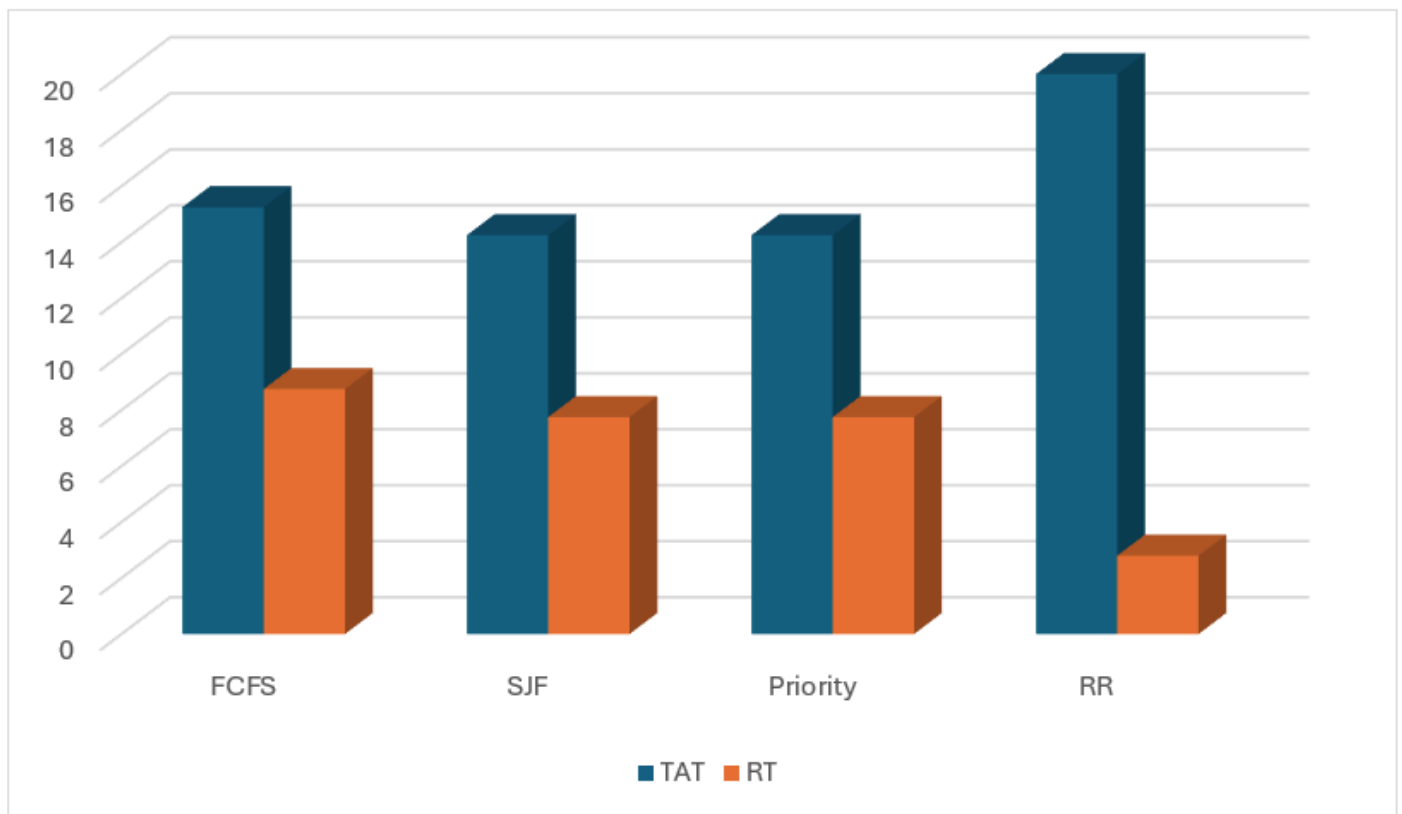
Section 7: Visualizations

7.1 CPU Scheduling Algorithms – Comparison Chart



The following bar chart conceptually represents the average Waiting Time (WT) for all scheduling algorithms implemented in Part 1.

7.2 Performance Chart



Analysis of the Combined Performance Chart

The combined bar chart effectively visualizes the performance trade-off between the four CPU scheduling algorithms based on two critical metrics: Average Turnaround Time (TAT) and Average Response Time (RT).

1. Analysis of Average Turnaround Time (TAT) – Blue Bars

The TAT metric measures the total time a process spends in the system from arrival to completion. A lower TAT value indicates better overall throughput and efficiency.

- **Best Performance:** SJF and Priority algorithms show the best performance with the shortest blue bars, both achieving an Average TAT of 14.25. This confirms their efficiency in minimizing the total completion time for the given dataset.
- **Worst Performance:** The Round Robin (RR) algorithm records the longest blue bar, indicating the worst performance with an Average TAT of 20. This is due to the overhead of context switching and dividing the total burst time into multiple small time slices.

2. Analysis of Average Response Time (RT) – Orange Bars

The RT metric is crucial for interactive systems, as it measures the time until the first execution begins. A lower RT value means better user experience and responsiveness.

- **Best Performance:** Round Robin (RR) significantly outperforms the others, achieving the shortest orange bar with an Average RT of 3. This is its main strength, as its time-slicing mechanism ensures all processes get immediate attention.

- Worst Performance: FCFS has the longest orange bar with an Average RT of 8.75, confirming that its simple, non-preemptive nature can lead to longer initial waiting times for later-arriving processes.

3. Conclusion: The Performance Trade-off

The chart clearly illustrates the fundamental trade-off in CPU scheduling:

- RR is optimized for responsiveness (RT), providing the fastest initial feedback, but it sacrifices efficiency (TAT).
- SJF and Priority are optimized for efficiency (TAT), completing all tasks in the shortest possible time, which is ideal for batch processing systems

Conclusion

This project successfully achieved its objectives by simulating and evaluating fundamental concepts in Operating Systems process management, specifically focusing on CPU Scheduling and Deadlock Avoidance (Banker's Algorithm).

In Part 1 (CPU Scheduling), the comparison between the four implemented algorithms (FCFS, SJF, Priority, and RR) revealed a clear trade-off based on the targeted performance metric. The simulation results demonstrated that:

- * The Shortest Job First (SJF) and Priority Scheduling algorithms delivered the best overall performance in terms of completion efficiency, recording the lowest average Turnaround Time (TAT) of 14.25.

- * Conversely, the Round Robin (RR) algorithm significantly excelled in the metric of interactivity, providing the fastest average Response Time (RT) of 3. This confirms its suitability for time-sharing and interactive environments at the cost of a higher overall TAT.

In Part 2 (Banker's Algorithm), the safety of the system was successfully verified based on the current allocation and available resources. The simulation confirmed that the system is in a SAFE state, demonstrating the possibility of completing all processes without leading to a deadlock. The identified safe sequence is: $P1 \rightarrow P3 \rightarrow P4 \rightarrow P0 \rightarrow P2$.

In conclusion, this project underscores the complexity and critical trade-offs inherent in Operating System design, where scheduling and resource management decisions require a precise balance between competing performance goals.

References

All theoretical concepts, definitions, and application models used throughout this report are based on the prescribed course curriculum and educational materials provided for the subject:

- * Course Scientific Material:
 - * Course: Operating Systems.
 - * Source: Course Lecture Slides and Instructional Materials.
 - * Institution: Taif University, College of Computer Science and Information Technology.