CSC 4320 Operating Systems

Project 01: CPU Scheduling & Memory Management

Deepak Srinivas Govindarajan

October 10, 2025

1 Overview

This project implements two fundamental CPU scheduling algorithms integrated with comprehensive memory management systems:

CPU Scheduling Algorithms:

- **Shortest Job First (SJF):** Non-preemptive algorithm that selects the process with the shortest burst time from the ready queue
- **Round Robin (RR):** Preemptive algorithm using time quantum = 3, cycling through processes in a circular queue

Repository: `https://github.com/1DeepakSrinivas/projects-4320`

2 Implementation Details

2.1 Process Scheduling Approach

Data Structures: Both algorithms use identical process structures containing process ID, arrival time, burst time, and calculated metrics (waiting time, turnaround time). Memory management fields track allocated addresses, page tables, and memory requirements.

SJF Implementation: The algorithm maintains a ready queue and selects the process with minimum burst time at each scheduling decision. Ties are broken by earliest arrival time. Once a process starts execution, it runs to completion without preemption.

Round Robin Implementation: Uses a circular queue with time quantum = 3. Processes are preempted when their quantum expires and re-queued if not completed. New arrivals are added to the queue during execution, ensuring fair CPU allocation.

2.2 Memory Management Integration

Contiguous Allocation: Each algorithm tests different allocation strategies:

- **First-Fit:** Allocates the first suitable memory block found
- **Best-Fit:** Allocates the smallest suitable block to minimize fragmentation

- **Worst-Fit:** Allocates the largest suitable block

Paging System: Processes are allocated pages dynamically with realistic memory access simulation. Page replacement occurs when frames are exhausted, using either FIFO or LRU algorithms.

2.3 Modular Architecture

The codebase follows clean separation with header files (`sjf.h`, `round_robin.h`) containing macros and function declarations, while implementation files handle algorithm logic. An interactive shell script (`main.sh`) provides menu-driven execution and comparison functionality.

3 Results

3.1 Test Dataset

PID	Arrival	Burst	Priority
1	0	5	2
2	2	3	1
3	4	2	3
4	5	4	2
5	6	6	1
6	8	3	4
7	10	5	2
8	12	2	3
9	13	4	1
10	15	3	2

3.2 Outputs

SJF:

```
Gantt Chart:
| P1 | P3 | P2 | P6 | P8 | P10 | P4 | P9 | P7 | P5 |
0    5    7    10   13   15    18   22   26   31   37

Process Statistics:
PID  Arrival  Burst  Waiting  Turnaround
1    0        5      0        5
2    2        3      5        8
3    4        2      1        3
4    5        4      13       17
5    6        6      25       31
6    8        3      2        5
7    10       5      16       21
8    12       2      1        3
9    13       4      9        13
10   15       3      0        3

Average Waiting Time:    7.20
Average Turnaround Time: 10.90
```

Round Robin:

```
Gantt Chart:
| P1 | P2 | P1 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P4 | P10 | P5 | P7 | P9 |
0    3    6    8    10   13   16   19   22   24   27   28    31   34   36   37

Process Statistics:
PID  Arrival  Burst  Waiting  Turnaround
1    0        5      3        8
2    2        3      1        4
3    4        2      4        6
4    5        4      19       23
5    6        6      22       28
6    8        3      8        11
7    10       5      21       26
8    12       2      10       12
9    13       4      20       24
10   15       3      13       16

Average Waiting Time:    12.10
Average Turnaround Time: 15.80
```

3.3 Performance Comparison

Algorithm	Avg Wait Time	Avg Turnaround Time	Performance
SJF	7.20	10.90	40% better wait time
Round Robin	12.10	15.80	Fair CPU allocation

Key Findings:

- SJF achieves optimal average waiting time but can cause starvation of longer processes
- Round Robin ensures fairness with bounded waiting time: maximum $(n-1) \times 3$ units
- Memory management shows different fragmentation patterns based on allocation strategy
- Paging eliminates external fragmentation but introduces page replacement overhead

4 Challenges

4.1 Algorithm Implementation

One of the hardest parts was building the circular queue for Round Robin scheduling. A small mistake in handling the indexes could cause processes to be lost or the schedule to break. Another challenge was handling processes that arrived while the system was already running. The timing had to be managed carefully so that new processes entered the queue at the right moment.

4.2 Memory Management

Coordinating memory allocation with scheduling was tricky. It wasn't always clear when memory should be assigned or released. Implementing Least Recently Used (LRU) page replacement was also a challenge, since it required tracking when each page was last accessed and identifying which one to replace.

4.3 System Integration

Designing the system in a modular way was difficult. The scheduling logic and memory management had to be separated, but still work together smoothly. Formatting the output consistently was another issue, especially since different parts of the system needed to present results in the same style.

4.4 Development Environment

Working only in the terminal with tools like WSL and Vim was a challenge at first, especially without the conveniences of modern IDEs as I opted to work in a Linux Environment. It required adjusting to a different workflow and set of tools.

5 Conclusion

This project demonstrates the key tradeoffs in operating system design. Shortest Job First (SJF) achieves the best average waiting time, but it is difficult to use in practice because it requires knowing burst times in advance and can lead to starvation. Round Robin is less efficient but fairer, making it better suited for interactive systems.

The memory management component shows how CPU scheduling and memory allocation are closely connected. Different strategies lead to different levels of fragmentation and performance, reflecting the complexity of real systems.

The modular design makes it possible to extend the system with features like priority scheduling, multilevel queues, or aging. Overall, the simulation environment provides useful insights into operating system principles and shows how theory translates into real implementation challenges and performance tradeoffs.