Side Project Documentation

David Matute Jimenez
david.matutejimenez1.email@marist.edu

October 18, 2025

1 Introduction

This project evaluates whether preemptive shortest job first (SJF) scheduling is the most efficient CPU scheduling algorithm. Using a C++ scheduling simulator, we test SJF across three different scenarios and analyze the results in terms of average wait time and average turnaround time. Preemptive SJF keeps wait times and turnaround times as low as possible. It also fixes the "convoy effect" problem that happens in non-preemptive scheduling, where short jobs get stuck waiting behind a long one.

2 Experiment 1: Varied Job Lengths with Staggered Arrivals

```
Process pA{};
pA.id          = 'A';
pA.cycles      = 8;
pA.arrivalTime = 0;

Process pB{};
pB.id          = 'B';
pB.cycles      = 2;
pB.arrivalTime = 1;

Process pC{};
pC.id          = 'C';
pC.cycles      = 5;
pC.arrivalTime = 2;

Process pD{};
pD.id          = 'D';
pD.cycles      = 1;
pD.arrivalTime = 3;
```

In Experiment 1, we tested preemptive SJF with a mix of short and long jobs (8, 2, 5, and 1 cycles) arriving at different times. The results show that letting shorter jobs interrupt longer ones gives the best average wait times. For example, running short jobs right when they arrive (like B at time 1 and D at time 3) performs much better than waiting to run them later. This shows that preemptive SJF keeps short jobs from being stuck behind long ones. In real systems, this means that quick tasks get done quickly without being delayed by larger jobs, making the system fairer and more responsive.

3 Experiment 2: Many Short Jobs vs. Few Long Jobs

```
Process pA{};
pA.id          = 'A';
pA.cycles      = 1;
pA.arrivalTime = 0;

Process pB{};
pB.id          = 'B';
pB.cycles      = 1;
pB.arrivalTime = 0;

Process pC{};
pC.id          = 'C';
pC.cycles      = 1;
pC.arrivalTime = 0;

Process pD{};
pD.id          = 'D';
pD.cycles      = 6;
pD.arrivalTime = 1;

Process pE{};
pE.id          = 'E';
pE.cycles      = 6;
pE.arrivalTime = 1;
```

In this experiment preemptive SJF still works efficiently by making sure long jobs don't slow down the short ones. In this test, three 1-cycle jobs start at time 0, and two 6-cycle jobs arrive at time 1. The simulator shows that SJF lets the short jobs finish first when the longer ones show up, avoiding any backlog. Although the performance gap is smaller than in Experiment 1 (since short jobs already finish fast), this proves that SJF is reliable and it stays efficient even when job lengths aren't very different and handles all kinds of workloads smoothly.

4 Experiment 3: Late-Arriving Long Job (Convoy Effect Test)

```
Process pA{};
pA.id          = 'A';
pA.cycles      = 7;
pA.arrivalTime = 0;

Process pB{};
pB.id          = 'B';
pB.cycles      = 1;
pB.arrivalTime = 3;

Process pC{};
pC.id          = 'C';
pC.cycles      = 1;
pC.arrivalTime = 4;

Process pD{};
pD.id          = 'D';
pD.cycles      = 1;
pD.arrivalTime = 5;
```

SJF is very effective and fixes the “convoy effect,” where short jobs get stuck waiting for a long one to finish. In this test, without preemption, short jobs like B, C, and D would have to wait for A to finish, even though they only take 1 cycle each. With preemptive SJF, the CPU pauses A when shorter jobs arrive, runs them first, and then goes back to A. This greatly cuts down wait times for the short jobs, and the small cost of switching tasks is worth it. In real life, this means quick, interactive tasks can respond fast even when big background jobs are running which makes preemptive SJF the better, fairer choice.

5 Conclusion

After looking at all three experiments, I believe that preemptive shortest job first (SJF) is the most efficient scheduling method. Experiment 1 shows it works best when job lengths vary, Experiment 2 also shows that it stays fast even when most jobs are short, and Experiment 3 shows it fixes the convoy effect by keeping short jobs from waiting behind long ones. Overall, the simulator proves that preemptive SJF gives the lowest average wait and turnaround times across different situations. The small cost of switching between jobs is worth it for the big performance boost.If I had to pick one algorithm for an operating system, preemptive SJF would be one of the better choices because it is fast, fair, and keeps the system responsive for users.