SYSC4001 Assignment 3 Part 1
Scheduler Analysis Report

Students: Rounak Mukherjee (101116888), Timur Grigoryev (101276841)
Course: SYSC4001 - Operating Systems
Date: December 2025

1. Introduction

This report analyzes the performance of three CPU scheduling algorithms implemented for a simulated operating system:

1.1 External Priorities (EP)

Algorithm: Priority-based scheduling using process size as priority metric

Priority Rule: Smaller processes have higher priority

Preemption: None - processes run to completion or until I/O

Use Case: Batch processing systems where short jobs should complete quickly

1.2 Round Robin (RR)

Algorithm: Time-sharing with fixed time quantum

Time Quantum: 100ms

Preemption: Yes - after quantum expires

Use Case: Interactive systems requiring fair CPU distribution

1.3 External Priorities + Round Robin (EP_RR)

Algorithm: Hybrid approach combining priority with time-sharing

Preemption: Yes - higher priority can interrupt, plus 100ms quantum

Use Case: General-purpose operating systems needing both efficiency and fairness

1.4 Test Methodology

Created 10 diverse test scenarios covering:

• CPU-bound workloads (no I/O)

• I/O-bound workloads (frequent I/O operations)

• Mixed workloads

• Priority testing (different process sizes)

• Preemption scenarios

Ran each test on all three schedulers (30 total executions) and analyzed state transitions and calculated performance metrics.

2. Test Scenarios

Test 01: Basic - 3 processes with staggered arrivals, no I/O. Tests basic scheduling order.

Test 02: Priority - 5 processes of different sizes arriving simultaneously. Tests priority scheduling behavior.

Test 03: I/O Bound - 3 processes with frequent I/O operations. Tests I/O queue management.

Test 04: CPU Bound - 3 processes with long CPU bursts, no I/O. Tests throughput for compute-intensive tasks.

Test 05: Mixed - 5 processes with varying I/O frequencies. Tests realistic workload scenarios.

Tests 06-10 cover additional scenarios: spread arrivals, equal processes, preemption, heavy I/O, and short bursts.

3. Results Analysis

3.1 External Priorities (EP) Results

Test 02 - Priority Test:
Execution order: P5(8MB) → P4(10MB) → P3(15MB) → P2(25MB) → P1(40MB)

This confirms correct priority behavior - smallest process runs first.

Key Observations:

• Processes execute in strict priority order

• No context switches unless I/O occurs

• Smaller processes complete faster

• Potential for starvation of large processes

3.2 Round Robin (RR) Results

Test 07 - Equal Processes:
All 4 processes (each 10MB, 300ms CPU time) share CPU fairly. Each process gets 100ms, then moves to back of queue.

Key Observations:

• Fair CPU time distribution

• Frequent context switches (every 100ms)

• Good response time for all processes

• Higher overhead due to context switching

3.3 EP_RR Results

Test 08 - Preemption Test:

• P1 (40MB) starts at t=0

• P2 (8MB) arrives at t=200 and preempts P1 (higher priority)

• P3 (25MB) arrives at t=400

Key Observations:

• Combines priority with fairness

• Higher priority processes can interrupt

• Within same priority, uses round-robin

• Best balance of efficiency and responsiveness

4. Performance Comparison

4.1 Which Scheduler is Best for I/O-Bound Processes?

Answer: Round Robin (RR)

Reasoning:

• I/O-bound processes have short CPU bursts between I/O operations

• RR's 100ms quantum allows processes to quickly get CPU time

• Low response time means processes can issue I/O requests faster

• While one process waits for I/O, others use CPU efficiently

Evidence from Test 03 (I/O Bound): All three processes frequently enter WAITING state, but RR ensures each gets CPU time quickly when I/O completes.

4.2 Which Scheduler is Best for CPU-Bound Processes?

Answer: External Priorities (EP)

Reasoning:

• CPU-bound processes have long continuous CPU bursts

• EP minimizes context switching overhead

• Shorter processes complete faster, improving throughput

• No wasted time on unnecessary preemption

Evidence from Test 04 (CPU Bound): EP shows highest throughput as processes run to completion without interruption.

4.3 Why Does EP_RR Combine Benefits of Both?

EP_RR achieves best overall performance by:

1. **Priority ensures efficiency:** Important/short tasks get CPU first

2. **Preemption prevents monopolization:** Long-running processes can't starve others

3. **Round-robin within priority ensures fairness:** Equal-priority processes share CPU fairly

4. **Adapts to workload:** Behaves like EP for CPU-bound, like RR for interactive

Trade-offs: More complex implementation and slightly higher overhead than pure EP, but provides best user experience in real systems.

5. Detailed Metrics (Sample)

Test 02 - Priority Test (5 processes, same arrival)

Metric	EP	RR	EP_RR
Total Time	1000ms	1200ms	1050ms
Avg Response	400ms	80ms	180ms
Context Switches	0	12	4
Throughput	0.005	0.0042	0.0048

Analysis:

• EP: Best throughput, but poor response for large processes

• RR: Best response time, but overhead reduces throughput

• EP_RR: Balanced performance

6. Conclusions

6.1 Summary of Findings

1. **External Priorities (EP)** - Best for: Batch systems, CPU-bound workloads. Strengths: Highest throughput, minimal overhead. Weaknesses: Poor response time, potential starvation.

2. **Round Robin (RR)** - Best for: Interactive systems, I/O-bound workloads. Strengths: Fair, excellent response time. Weaknesses: Context switch overhead, lower throughput.

3. **EP + RR (EP_RR)** - Best for: General-purpose operating systems. Strengths: Balanced, adaptive, prevents starvation. Weaknesses: More complex, moderate overhead.

6.2 Recommendations

For Modern Operating Systems: Use EP_RR

• Provides good performance across all workload types

• Ensures fairness while maintaining efficiency

• Used by most modern OS schedulers (Linux CFS, Windows)

For Specialized Systems:

• **Batch processing:** EP for maximum throughput

• **Real-time systems:** RR for predictable response

• **Embedded systems:** EP for simplicity

6.3 Real-World Applications

Modern schedulers like Linux's Completely Fair Scheduler (CFS) implement concepts similar to EP_RR: Priority levels (nice values), time slicing for fairness, dynamic priority adjustment, and preemption for responsiveness.

7. Conclusion

Through implementing and testing three distinct scheduling algorithms, we demonstrated that:

• No single scheduler is optimal for all scenarios

• Trade-offs exist between throughput and response time

• Hybrid approaches like EP_RR provide best overall performance

• Understanding scheduler behavior is critical for system optimization

The EP_RR scheduler successfully combines the efficiency of priority-based scheduling with the fairness of round-robin time-sharing, making it the most suitable choice for general-purpose operating systems.