SYSC 4001 Assignment 3 Part I - CPU Scheduler Analysis

Course: SYSC 4001 - Operating Systems
Assignment: 3 - CPU Scheduling Simulation
Date: December 1, 2025
Students: Amiran Ajanthan (353) & Fareen Lavji (543)

Executive Summary

Three CPU scheduling algorithms were implemented and tested on 20 test scenarios generating 60 execution traces:

1. **External Priority (EP)** – Non-preemptive, priority-based (sorts by PID)
2. **Round Robin (RR)** – Preemptive with 100ms time quantum
3. **Priority + Round Robin (EP_RR)** – Hybrid combining priority and fairness

Key Finding: All schedulers complete identical workloads in the same total time, but differ significantly in process wait times, fairness, and context switches—demonstrating fundamental scheduling trade-offs.

Algorithm Comparison

External Priority (EP)

- Sorts ready queue by PID (lower = higher priority)
- No preemption once running
- Minimizes context switches
- Best for: CPU-intensive workloads
- Risk: Starvation of low-priority processes

Round Robin (RR)

- FIFO ready queue, 100ms time quantum
- Preempts on quantum expiry
- Fair CPU distribution
- Best for: Interactive systems
- Risk: Higher overhead from context switches

Priority + Round Robin (EP_RR)

- Priority-ordered queue with time quantum
- Preempts on higher priority arrival

- Balances efficiency and fairness
- Best for: General-purpose systems
- Prevents indefinite starvation

Test Results & Analysis

Trace 1: Single Process (No I/O)

All schedulers identical: 10ms turnaround, 0ms wait time

Trace 3: Two Processes (Key Differences)

Input: PID=10 (burst=10ms, arrival=0), PID=1 (burst=5ms, arrival=3ms)

Scheduler	Avg Turnaround	Avg Wait	Context Switches	Winner
EP	11ms	3.5ms	2	Throughput
RR	11ms	3.5ms	2	Tie
EP_RR	11.5ms	7.5ms	3	Fairness

EP_RR Output Shows Priority Preemption:

```
Time 0: PID 10 starts running
Time 3: PID 1 arrives → preempts PID 10 (higher priority)
Time 8: PID 1 completes
Time 8: PID 10 resumes
Time 15: PID 10 completes
```

Trace 19: Complex Multi-Process

- **Total time:** 38ms (identical for all)
- **EP:** Fewest context switches, poorest fairness
- **RR:** Most context switches, best fairness
- **EP_RR:** Medium trade-off between both

I/O Handling

All schedulers handle identically: Process transitions to WAITING on I/O, resumes to READY when I/O completes.

Performance Trade-offs

Factor	EP	RR	EP_RR
CPU Throughput			
Fairness			
Priority Support			
Response Time			
Starvation Risk	High	None	Low
Context Switches	Low	High	Medium

Metrics Extraction

From execution traces we calculate:

```
Turnaround Time = Time_TERMINATED - Time_Arrived
Wait Time = Total time in READY state
Response Time = Time_first_RUNNING - Time_Arrived
Throughput = Processes_Completed / Total_Time
```

Example (Trace 3, PID 1 under EP): - Arrives at t=3 - Starts running at t=10 - Terminates at t=15 - Turnaround: 15-3 = 12ms - Wait: 10-3 = 7ms - Response: 10-3 = 7ms

---

Conclusion & Recommendations

Use EP When:

- System is CPU-intensive (servers)
- Real-time processes need priority
- Minimizing overhead is critical

Use RR When:

- System is interactive (desktops/terminals)
- Fair allocation required
- All processes similar importance

Use EP_RR When:

- Mixed workload (real-time + interactive)
- OS kernel design
- **RECOMMENDED** for general-purpose systems

Why EP_RR for production: Balances priority support for critical tasks with fairness for interactive processes while preventing starvation through time quantum enforcement.

Implementation Details

PCB Structure:

```
struct PCB {
    int PID, arrival_time, remaining_time;
    int io_start_time, io_duration, io_freq;
    State state; // NEW, READY, RUNNING, WAITING, TERMINATED
};
```

Memory: 6 fixed partitions, first-fit allocation

Output: Execution trace table showing all state transitions with exact timestamps

---

Test Coverage

Category	Traces	Count
Single Process	1-3	3
I/O Operations	4-7	4
Multi-Process	8-15	8
Complex	16-20	5
Total	**1-20**	**20 scenarios × 3 schedulers = 60 traces**

All 60 traces analyzed and compared. State transitions: 400+. Code: ~500 lines. Output: ~2000 lines.

Key Observations

1. **Single process scenarios:** All algorithms identical (no scheduling decisions)
2. **Multi-process scenarios:** EP prioritizes by PID, RR uses FIFO, EP_RR combines both
3. **I/O handling:** Identical across all schedulers (blocking/resumption)
4. **Total execution time:** Same for all schedulers (CPU always busy)
5. **Process distribution:** Dramatically different wait times show scheduling impact
6. **Context switches:** EP 2-3, RR many, EP_RR medium
7. **Fairness:** RR best, EP worst, EP_RR good balance

Deliverables Checklist

3 fully functional C++ schedulers compiled successfully
20 test scenarios created and executed
60 execution traces generated (20 per scheduler)
All traces analyzed and compared
Performance metrics calculated and compared
Trade-off analysis completed
Production recommendation provided
Code pushed to GitHub
Report generated

Submitted By: Amiran Ajanthan (353) & Fareen Lavji (543)