CPU Scheduling Algorithms Simulator

Advanced Operating Systems Project

Team Members:

- Adedeji Olu (Project Lead)
- Almamy Sow (Testing & Documentation)
- Brionna Nunn (Performance Analysis)

Course: Advanced Operating Systems
Date: December 2025

Agenda

1. Project Overview
2. System Architecture
3. Implemented Algorithms
4. Live Demonstration
5. Testing Framework
6. Performance Analysis
7. Challenges & Solutions
8. Lessons Learned
9. Q&A

Project Overview

Objective

Develop a comprehensive CPU scheduling simulator implementing 5 classic algorithms with:

- Real-time visualization
- Performance metrics
- Professional testing framework
- Complete documentation

Technologies Used

- **Language:** C++17
- **Testing:** Google Test Framework
- **Build System:** GNU Make
- **Version Control:** Git/GitHub
- **Platform:** FreeBSD

System Architecture

Class Hierarchy

```
Scheduler (Abstract Base Class)
    ├── RoundRobinScheduler
    ├── PreemptivePriorityScheduler
    ├── NonPreemptivePriorityScheduler
    ├── MultilevelQueueScheduler
    └── MultilevelFeedbackQueueScheduler
```

Design Patterns

- **Template Method:** Common scheduling framework
- **Strategy Pattern:** Interchangeable algorithms
- **OOP Principles:** Inheritance, polymorphism, encapsulation

---

Implemented Algorithms

1. Round Robin (RR)

- **Mechanism:** Time-sharing with fixed quantum
- **Queue:** FIFO circular queue
- **Use Case:** Interactive systems
- **Pros:** Fair, no starvation
- **Cons:** High context switching overhead

2. Preemptive Priority

- **Mechanism:** Higher priority preempts lower
- **Queue:** Min-heap (priority queue)
- **Use Case:** Real-time systems
- **Pros:** Critical tasks execute first
- **Cons:** Potential starvation

Implemented Algorithms (cont.)

3. Non-Preemptive Priority

- **Mechanism:** Priority-based, no preemption
- **Queue:** Priority queue
- **Use Case:** Batch processing
- **Pros:** Simple, predictable
- **Cons:** Poor response time

4. Multilevel Queue (MLQ)

- **Mechanism:** Three fixed queues
  - System (0-1): Highest priority
  - Interactive (2-3): Round Robin
  - Batch (4-5): FCFS
- **Use Case:** Mixed workloads
- **Pros:** Queue separation
- **Cons:** Inflexible

Implemented Algorithms (cont.)

5. Multilevel Feedback Queue (MLFQ)

- **Mechanism:** Adaptive queue demotion
  - Q0: Quantum = 8 (highest)
  - Q1: Quantum = 16
  - Q2: FCFS (lowest)
- **Aging:** Prevents starvation
- **Use Case:** Unknown process behavior
- **Pros:** Adaptive, fair
- **Cons:** Complex implementation

Key Features

Process Management

▯ Complete Process Control Block (PCB) simulation
▯ Process states: NEW, READY, RUNNING, TERMINATED
▯ Dynamic process arrival
▯ Context switching overhead modeling

Performance Metrics

 Waiting Time
 Turnaround Time
 Response Time
 CPU Utilization
 Context Switch Count

Live Demonstration

Demo Flow

1. **Show Project Structure**

```
ls -la ~/cpu-scheduler
```

2. **Show Code with vi/vim**

```
vi src/scheduler.cpp
```

3. **Build from Scratch**

```
gmake clean && gmake build
```

4. **Run Tests**

```
gmake test
```

Live Demonstration (cont.)

Running the Simulator

```
./bin/scheduler
```

Algorithms to Demonstrate:

1. **Round Robin** (quantum = 4)
2. **Preemptive Priority**
3. **Run All** (comparison mode)

What to Show:

- Real-time visualization
- Process state transitions
- Performance metrics
- Context switching

Testing Framework

Google Test Integration

- **Unit Tests:** 4 tests covering components
- **System Tests:** 4 end-to-end tests
- **Coverage:** Process, Scheduler, Algorithms

Test Results

```
[==========] Running 8 tests from 4 test suites.
[  PASSED  ] 8 tests.
```

Test Categories:

Process creation & state transitions
Algorithm correctness
Metrics calculation
Edge cases (empty queue, single process)

Performance Analysis

Benchmark Results (10 processes)

Algorithm	Avg Wait	Avg Turnaround	Best Use Case
Round Robin (q=4)	6.50 ms	14.00 ms	Interactive
Preemptive Priority	2.50 ms	10.00 ms	Real-time
Non-Preemptive	2.50 ms	10.00 ms	Batch
Multilevel Queue	5.00 ms	12.50 ms	Mixed
MLFQ	11.8 ms	18.2 ms	Adaptive

Key Finding

Preemptive Priority has 61.5% lower wait time than Round Robin

---

Performance Insights

Context Switching Impact

- **Overhead:** 1-2 time units per switch
- **Total Impact:** 5-17% of execution time
- **Trade-off:** Fairness vs. efficiency

Quantum Selection (Round Robin)

- **Too small (q=1):** Excessive context switching
- **Too large (q=100):** Becomes FCFS
- **Optimal:** q=4-8 for interactive systems

Scalability

All algorithms scale linearly: O(n)
Tested with up to 100 processes
Maintains performance characteristics

GitHub Repository

Professional Development Practices

Repository: https://github.com/Adedeji-Olu/cpu-scheduling-algorithms

Features:

Clean directory structure
Comprehensive README with team members
CI/CD with GitHub Actions
Professional commit history
Complete documentation (532 lines)
Contributing guidelines
Pull request templates

Project Structure

```
cpu-scheduling-algorithms/
├── src/              # Source files (C++)
├── include/          # Header files
├── tests/            # Google Test suite
│   ├── unit/         # Unit tests
│   └── system/       # System tests
├── doc/              # Documentation
│   ├── API.md
│   ├── DESIGN.md
│   ├── USER_MANUAL.md
│   └── PERFORMANCE_ANALYSIS.md
├── .github/          # CI/CD workflows
├── Makefile          # Build system
└── README.md         # Project overview
```

Build System

Makefile Targets

```
 gmake build        # Optimized compilation (-O2)
gmake debug        # Debug symbols (-g)
gmake test         # Run all tests
gmake unit-test    # Unit tests only
gmake system-test  # System tests only
gmake clean        # Remove artifacts
gmake install      # System installation
```

Compilation

- **Standard:** C++17
- **Compiler:** g++/clang++
- **Flags:** -Wall -Wextra -Wpedantic
- **Platform:** FreeBSD, Linux, macOS

Challenges & Solutions

Challenge 1: Context Switching Overhead

Problem: Realistic modeling difficult
Solution: Configurable overhead parameter (0-5 time units)

Challenge 2: Starvation in Priority Scheduling

Problem: Low priority processes starve
Solution: Aging mechanism (priority boost every 10 time units)

Challenge 3: MLFQ Complexity

Problem: Multiple queues, demotion logic
Solution: Clear state machine, extensive testing

Challenge 4: Real-time Visualization

Problem: Balance detail vs. readability
Solution: Periodic state snapshots + final metrics

Code Quality

Best Practices

- **Object-Oriented Design:** Clear class hierarchy
- **STL Containers:** Efficient data structures
- **Const Correctness:** Immutable where possible
- **Error Handling:** Edge case coverage
- **Documentation:** Inline comments + external docs

Testing Strategy

- **Unit Tests:** Component validation
- **System Tests:** End-to-end scenarios
- **Benchmark Tests:** Performance validation
- **Edge Cases:** Empty queues, single process, zero burst

Lessons Learned

Technical Skills

- Deep understanding of CPU scheduling
- Process state management
- Performance metrics analysis
- Professional testing practices
- Git/GitHub collaboration

Software Engineering

- Importance of modular design
- Value of comprehensive testing
- Documentation as first-class artifact
- CI/CD for quality assurance

Teamwork

- Clear role division
- Version control workflows
- Code review processes

Future Enhancements

Planned Features

1. **I/O Burst Modeling**
    - Add WAITING state
    - Device queues
    - I/O completion events
2. **Real-Time Scheduling**
    - Earliest Deadline First (EDF)

    - Rate Monotonic Scheduling (RMS)

3. **Multiprocessor Support**
    - Load balancing
    - Processor affinity
    - Gang scheduling

4. **GUI Interface**
    - Interactive Gantt charts
    - Real-time metrics dashboard

Demonstration Time!

Live Demo Checklist

Show project structure
Display code with vi/vim
Clean build from scratch
Run all tests
Execute scheduler with 3 algorithms
Show documentation
Present GitHub repository

Ready for questions!

Key Takeaways

Project Achievements

5 algorithms implemented and tested
8 tests - 100% passing
532 lines of documentation
Professional GitHub repository
Comprehensive performance analysis

Academic Goals Met

Understanding of CPU scheduling
Process management expertise
Performance analysis skills
Professional development practices

Questions & Answers

Contact Information

GitHub Repository:
https://github.com/Adedeji-Olu/cpu-scheduling-algorithms

Team Members:

- Adedeji Olu
- Almamy Sow
- Brionna Nunn

Thank you for your attention!

Appendix: References

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
2. Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Pearson.
3. FreeBSD Scheduler Documentation: https://docs.freebsd.org/

4. Google Test Documentation: https://google.github.io/googletest/
5. GitHub Actions Documentation: https://docs.github.com/en/actions