# Operating Systems Lab Report

## Intelligent CPU Scheduler - Web-Based Simulation

**Student Name:** Harshit Singh
**Roll Number:** 02
**Registration Number :** 12413807
**Subject:** Operating Systems Lab
**Date:** December 10, 2025

---

## Table of Contents



---

## 1. Introduction

This project is a comprehensive web-based CPU Scheduling Simulator that demonstrates various CPU scheduling algorithms used in operating systems. The application provides an interactive interface to add processes, simulate different scheduling algorithms, visualize results through Gantt charts, and compare algorithm performance to recommend the most efficient one.

The project is built using modern web technologies (HTML5, CSS3, JavaScript ES6+ modules) and implements five major CPU scheduling algorithms with proper visualization and metrics calculation.

---

## 2. Objective

The main objectives of this project are:

- To understand and implement various CPU scheduling algorithms
- To visualize process execution using Gantt charts

- To calculate important scheduling metrics (Completion Time, Turnaround Time, Waiting Time)
- To compare different algorithms and recommend the best one for given processes
- To create an interactive, user-friendly interface for learning scheduling concepts
  • To demonstrate proficiency in system calls and process management concepts

---

## 3. System Requirements

**Software Requirements:** - Web Browser (Chrome, Firefox, Safari, Edge latest versions) - Any HTTP server (Live Server, Python HTTP Server, Node.js http-server) - Text Editor/IDE (VS Code, Sublime Text, etc.)

**Hardware Requirements:** - Any modern computer with 2GB+ RAM - Display resolution: 1024x768 or higher

**Technologies Used:** - HTML5 for structure - CSS3 for styling and responsive design - JavaScript ES6+ for logic and algorithms - CSS Grid and Flexbox for layout - Google Fonts (Outfit family)

---

## 4. Algorithms Implemented

### 4.1 First Come First Serve (FCFS)

**Type:** Non-Preemptive
**Description:** Processes are executed in the order they arrive. The process that arrives first is executed first.

**Key Characteristics:** - Simple and easy to implement - No starvation - High average waiting time for short processes - Convoy effect possible

### 4.2 Shortest Job First (SJF)

**Type:** Non-Preemptive
**Description:** The process with the shortest burst time is executed first among all available processes.

**Key Characteristics:** - Optimal average waiting time - Risk of starvation for long processes - Requires knowledge of burst times - Better CPU utilization

### 4.3 Shortest Remaining Time First (SRTF)

**Type:** Preemptive
**Description:** Preemptive version of SJF. The process with the shortest remaining burst time is executed.

**Key Characteristics:** - Minimum average waiting time - Context switching overhead - Starvation possible for long processes - Most optimal for minimizing waiting time

#### 4.4 Round Robin (RR)

**Type:** Preemptive
**Description:** Each process gets a fixed time quantum in circular order.

**Key Characteristics:** - Fair allocation of CPU time - No starvation - Performance depends on time quantum - Better response time - Context switching overhead

#### 4.5 Priority Scheduling

**Type:** Non-Preemptive
**Description:** Processes are executed based on priority (lower number = higher priority).

**Key Characteristics:** - Flexible scheduling based on importance - Starvation possible for low-priority processes - Can be combined with aging to prevent starvation - Realworld applicability

---

## 5. Project Structure

```
CPU-Scheduler/

 index.html          # Main HTML structure
 style.css           # Styling and layout
 script.js           # Main application
                     logic
 algorithms.js      #                Algorithm
                     implementations
 README.md          # Project documentation
```

**File Descriptions:**

**index.html** (45 KB) - Contains the complete HTML structure - Input forms for adding processes - Algorithm selection dropdown - Process queue table - Results section with Gantt chart - Comparison section - Responsive layout structure **style.css** (6.5 KB) - Modern dark theme design - CSS custom properties for theming - Responsive grid layouts - Gantt chart styling - Interactive hover effects
- Mobile-responsive breakpoints **script.js** (7.2 KB) - DOM manipulation and event handling - Process management (add/remove) - Simulation execution - Results rendering - Comparison logic - Error handling overlay
**algorithms.js** (8.9 KB) - Implementation of all 5 algorithms - Process scheduling logic - Gantt chart data generation - Metrics calculation - Algorithm comparison function

## 6. Implementation Details

### 6.1 Data Structures Used Process

**Object:**

```
{
    id: number,           //    Unique     process
                          identifier
    arrivalTime: number, // When process arrives
    burstTime: number,    // CPU time required
    priority: number,     // Process priority (1-
                          10)
    remainingTime:        //    For     preemptive
    number,               algorithms
    completionTime:       // When process
    number,               completes
    turnaroundTime:       // CT - AT
    number,
    waitingTime: number   // TAT - BT
}
```

**Schedule Block:**

```
{ type: 'process' |
    'idle', processId:
    number, start: number,
    end: number, duration:
    number
}
```

### 6.2 Key Functions FCFS Algorithm:

```
FCFS: (processes) => {
    // Sort processes by arrival time
    // Execute each process completely
    // Calculate metrics
    // Return schedule and solved processes
}
```

**SJF Algorithm:**

```
SJF: (processes) => {
    // At each decision point, select available process
    // with shortest burst time
```

```
    // Execute completely (non-preemptive)
    // Handle idle time if no process available
}
```

**SRTF Algorithm:**

```
SRTF: (processes) => {
    // Execute process with shortest remaining time
    // Preempt if shorter process arrives
    // Execute in 1-unit time slices
    // Update remaining time after each unit
}
```

**Round Robin Algorithm:**

```
RR: (processes, quantum) => {
    // Maintain a ready queue
    // Execute each process for time quantum
    // If not complete, move to end of queue
    // Handle new arrivals during execution
}
```

**Priority Scheduling:**

```
Priority: (processes) => {
    // Select process with highest priority (lowest number)
    // Execute completely (non-preemptive)
    // Handle equal priorities (FCFS order)
}
```

### 6.3 Metrics Calculation

**Completion Time (CT):** - Time at which process finishes execution

**Turnaround Time (TAT):** `TAT = Completion Time - Arrival Time`

**Waiting Time (WT):** `WT = Turnaround Time - Burst Time`

**Average Metrics:**

```
Avg TAT = Sum of all TAT / Number of processes Avg
WT = Sum of all WT / Number of processes
```

---

## 7. Features

### 7.1 Core Features

1. **Process Management**
    - Add processes with arrival time, burst time, and priority
    - Delete individual processes
    - View all processes in a table
    - Reset all data
2. **Algorithm Selection**
    - Choose from 5 different algorithms
    - Dynamic time quantum input for Round Robin
    - Real-time algorithm switching
3. **Simulation**
    - Execute selected algorithm
    - Generate complete schedule
    - Calculate all metrics
    - Handle edge cases (idle time, same arrival)
4. **Visualization**
    - Interactive Gantt chart
    - Color-coded process blocks
    - Hover tooltips with timing information
    - Time ruler for reference
    - Process-specific color generation
5. **Results Display**
    - Detailed metrics table
    - Average turnaround time
    - Average waiting time
    - CPU utilization
    - Individual process statistics
6. **Algorithm Comparison**
    - Run all algorithms simultaneously
    - Compare average metrics
    - Automatic recommendation
    - Highlighted best algorithm
    - Side-by-side comparison table

### 7.2 User Experience Features

1. **Modern UI/UX**
    - Dark theme with gradient accents
    - Smooth animations and transitions
    - Responsive design for all devices
    - Clean, professional interface
2. **Error Handling**
    - Input validation
    - Runtime error overlay

- User-friendly error messages
- Console logging for debugging

3. **Accessibility** • Semantic HTML
   - ARIA labels
   - Keyboard navigation
   - High contrast design

---

## 8. How to Run the Project

### Method 1: Using Live Server (VS Code)

1. Install "Live Server" extension in VS Code
2. Open project folder in VS Code 3. Right-click on `index.html`
4. Select "Open with Live Server"
5. Browser will open automatically at `http://127.0.0.1:5500/index.html`

### Method 2: Using Python HTTP Server

```
# Navigate to project
directory cd
path/to/CPUScheduler

# Python 3 python -m http.server
8000

# Open browser and go to:
# http://localhost:8000/index.html
```

### Method 3: Using Node.js http-server

```
# Install http-server globally (one
time) npm install -g http-server

# Navigate to project
directory cd
path/to/CPUScheduler

# Start server httpserver

# Open browser and go to displayed URL
```

**Method 4: Direct File Opening (Limited Functionality)**

1. Double-click `index.html`
2. Opens in default browser
3. **Note:** ES6 modules may not work with file:// protocol

---

## 9. Usage Instructions

**Step 1: Add Processes**

1. Enter **Arrival Time** (when process arrives in ready queue)
2. Enter **Burst Time** (CPU time required)
3. Enter **Priority** (for Priority Scheduling, lower = higher priority)
4. Click "Add Process"
5. Repeat to add multiple processes

**Example:** - Process 1: AT=0, BT=4, Priority=2 - Process 2: AT=1, BT=3, Priority=1 - Process 3: AT=2, BT=1, Priority=3

**Step 2: Configure Algorithm**

1. Select algorithm from dropdown:
   - FCFS • SJF
   - SRTF
   - Round Robin (specify time quantum)
   - Priority
2. If Round Robin selected, enter Time Quantum (e.g., 2)

**Step 3: Run Simulation**

1. Click **"Simulate"** button
2. View Gantt chart showing process execution timeline
3. Check metrics table for individual process statistics
4. Review average TAT and WT

**Step 4: Compare Algorithms (Optional)** 1. Click **"Compare & Recommend"** button

2. System runs all algorithms automatically
3. View comparison table
4. Best algorithm is highlighted with recommendation banner

**Step 5: Reset (if needed)**

1. Click **"Reset"** button to clear all data
2. Start fresh with new processes

---

## 10. Sample Test Cases

**Test Case 1: Basic FCFS**

**Input:** | Process | Arrival Time | Burst Time | |———|————|————| | P1 | 0 | 5 | | P2 | 1 | 3 | | P3 | 2 | 8 |
**Expected Output (FCFS):** - Gantt Chart: P1(0-5) → P2(5-8) → P3(8-16) Avg TAT: 10.67 - Avg WT: 5.33

**Test Case 2: SJF Optimization**

**Input:** | Process | Arrival Time | Burst Time | |———|————|————| | P1 | 0 | 7 | | P2 | 2 | 4 | | P3 | 4 | 1 | | P4 | 5 | 4 |

**Expected Output (SJF):** - Shortest jobs execute first when available - Lower average waiting time than FCFS

**Test Case 3: Round Robin**

**Input:** | Process | Arrival Time | Burst Time | |———|————|————| | P1 | 0 | 5 | | P2 | 1 | 4 | | P3 | 2 | 2 |

**Time Quantum:** 2

**Expected Output:** - Circular execution with preemption - Fair CPU allocation - Multiple context switches visible in Gantt chart

---

## 11. Learning Outcomes

Through this project, I have learned:

1. **Algorithm Understanding**
   - Deep understanding of CPU scheduling concepts
   - Differences between preemptive and non-preemptive scheduling
   - Trade-offs between different algorithms
2. **Programming Skills**
   - JavaScript ES6+ features (modules, arrow functions, destructuring)
   - DOM manipulation and event handling
   - Modular code organization
   - Deep vs shallow copying issues
3. **Problem Solving**
   - Implementing complex algorithms
   - Handling edge cases (idle time, simultaneous arrivals)
   - Queue management for Round Robin
   - State tracking for preemptive algorithms

4. **Web Development**
    - Responsive design principles
    - Modern CSS techniques (Grid, Flexbox, Custom Properties)
    - User interface design
    - Data visualization
5. **Operating System Concepts**
    - Process lifecycle and states
    - Context switching overhead
    - CPU utilization and efficiency
    - Starvation and aging concepts

---

## 12. Challenges Faced

### Challenge 1: SRTF Context Switching

**Problem:** Tracking when to preempt and creating proper Gantt chart blocks. **Solution:** Implemented time-slice execution (1 unit at a time) and merged consecutive blocks of same process.

### Challenge 2: Round Robin Queue Management

**Problem:** Managing ready queue with new arrivals during execution. **Solution:** Added `inQueue` flag and checked for new arrivals after each quantum execution.

### Challenge 3: Deep Copy Issues

**Problem:** Algorithms were modifying original process array.
**Solution:** Used `JSON.parse(JSON.stringify())` for deep copying before each algorithm run.

### Challenge 4: Gantt Chart Proportional Width

**Problem:** Making blocks proportional to execution time.
**Solution:** Calculated percentage based on total execution time and used flexbox for layout.

---

## 13. Future Enhancements

1. **Additional Algorithms**
    - Preemptive Priority Scheduling
    - Multilevel Queue Scheduling
    - Multilevel Feedback Queue Scheduling

2. **Advanced Features**
    - Process state diagram visualization
    - Step-by-step execution mode
    - Export results to PDF/CSV
    - Save/load process sets
    - Context switch time consideration
3. **Educational Features**
    - Algorithm explanation tooltips
    - Interactive tutorial mode
    - Quiz/assessment section
    - Animation speed control
4. **Performance Metrics**
    - Throughput calculation
    - Response time tracking
    - CPU utilization percentage
    - Context switch count

---

## 14. Conclusion

This CPU Scheduler project successfully demonstrates the implementation and comparison of major CPU scheduling algorithms. The web-based interface makes it an excellent educational tool for understanding operating system concepts visually and interactively.

The project achieves all stated objectives: - Implements 5 different scheduling algorithms correctly - Provides accurate metric calculations - Offers clear visualization through Gantt charts - Compares algorithms and recommends optimal choice - Delivers professional, user-friendly interface

This project has significantly enhanced my understanding of operating systems, particularly process scheduling, and has improved my skills in web development and algorithm implementation.

---

## 15. References


1. **Operating System Concepts** by Abraham Silberschatz, Peter B. Galvin, Greg Gagne (10th Edition)
2. **Modern Operating Systems** by Andrew S. Tanenbaum, Herbert Bos (4th Edition)
3. **Operating Systems Lab Manual** - Prepared by Pushpendra Kumar

Pateriya
4. **MDN Web Docs** - JavaScript Reference https://developer.mozilla.org/enUS/docs/Web/JavaScript
5. **W3Schools** - HTML, CSS, JavaScript Tutorials https://www.w3schools.com/
6. **GeeksforGeeks** - CPU Scheduling Algorithms https://www.geeksforgeeks.org/cpuscheduling-in-operating-systems/ 7. **YouTube Tutorials** - Operating Systems by Gate Smashers

---

## Appendix A: Code Snippets

### Key Algorithm: FCFS Implementation

```
FCFS: (processes) => { let
    currentTime = 0; const
    schedule = []; const
    solvedProcesses = [];

    const sortedProcesses = [...processes]
        .sort((a, b) => a.arrivalTime - b.arrivalTime);

    sortedProcesses.forEach(process => {
        if (currentTime <
        process.arrivalTime) {
        schedule.push({ type: 'idle',
        start: currentTime, end:
        process.arrivalTime
            }); currentTime =
            process.arrivalTime;
        }

        const startTime = currentTime; const completionTime =
        startTime + process.burstTime;

        schedule.push({ type:
            'process', processId:
            process.id, start:
            startTime, end:
            completionTime,
            duration:
            process.burstTime
        });
```

```
        solvedProcesses.push({
            ...process,
            completionTime,
            turnaroundTime:
            completionTime -
            process.arrivalTime,
            waitingTime:
            completionTime -
            process.arrivalTime -
            process.burstTime
        });

        currentTime = completionTime;
    });

    return { schedule,
solvedProcesses }; }
```

---

## Appendix B: Project Screenshots

*[Note: Include screenshots of the following when submitting]*

1. **Homepage** - Initial interface with empty process queue
2. **Process Addition** - Form filled with sample process
3. **Process Queue** - Table showing multiple added processes
4. **FCFS Simulation** - Gantt chart and results for FCFS
5. **Round Robin Simulation** - With time quantum configuration
6. **Comparison View** - All algorithms compared with recommendation 7. **Responsive Mobile View** - Interface on mobile device

---

## Declaration

I hereby declare that this project work titled **"Intelligent CPU Scheduler - Web-Based Simulation"** is my original work and has been completed by me under the guidance of my instructor. All sources of information have been properly acknowledged.

---

**Signature:** _____

**Date:** December 10, 2025

---

**ProjectRepository:** https://github.com/harhsitsingh123/Intelligent-CPU-scheduler
**Live Demo:** http://127.0.0.1:5500/index.html

---

*End of Report*