Project 1
Operating systems
Roya Hosseini
4320-6320
Spring 2026

Operating Systems Project: Process Scheduling Simulation

🔹 **What You Will Do**

You will **write a program** that simulates how an **Operating System schedules processes** using different algorithms.

✅ Step 1: Read Process Data from a File

1. **Create a text file** (e.g., `processes.txt`) with a list of processes.
2. Each process should have:
   - **Process ID (PID)**
   - **Arrival Time** (When the process arrives in the system)
   - **Burst Time** (How long the process needs the CPU)
   - **Priority** (If using Priority Scheduling)

   Example of `processes.txt`:

   ```
   PID  Arrival_Time  Burst_Time  Priority
   1    0             5           2
   2    2             3           1
   3    4             2           3
   ```

3. Your program should **read this file** and store the data in memory.

✅ Step 2: Implement Two Scheduling Algorithms

You must write code for **at least two** of these CPU scheduling algorithms:

✅ **First-Come, First-Served (FCFS)** → The first process that arrives runs first.
✅ **Shortest Job First (SJF)** → The process with the smallest burst time runs first.
✅ **Round Robin (RR)** → Each process gets a fixed time (time quantum), then the next process runs.
✅ **Priority Scheduling** → Processes with a higher priority run first.

Each algorithm should:
✔ Sort the processes based on the algorithm's rule.
✔ Simulate execution (decide which process runs at each step).
✔ Calculate **Waiting Time (WT) and Turnaround Time (TAT)**.

✅ Step 3: Display a Gantt Chart (Execution Order)

1. Your program should **show the order in which processes run**.
2. Display a **simple text-based Gantt chart** in the console.

   Example Output:

   ```
   | P1 | P2 | P3 | P1 | P4 |
   0    2    5    7    12   15
   ```

3. At the end, print:
   - **Waiting Time (WT) for each process**
   - **Turnaround Time (TAT) for each process**
   - **Average WT and TAT**

✅ Step 4: (Optional) Implement Memory Management

If you want to go further, you can add **memory allocation**:
✔ **First-Fit, Best-Fit, or Worst-Fit allocation**
✔ Simulate **paging and page replacement (FIFO, LRU)**

✅ Step 5: Submit Your Work

1️⃣ **Only C, C++, or Java source code is accepted. Submissions in any other language will not be graded and will receive a zero (0).**

2️⃣ **Your input file (`processes.txt`) and sample output** (Gantt chart + calculations).
3️⃣ **A short report (2-4 pages, PDF) explaining:**

- What scheduling algorithms you implemented
- Sample test cases and results
- Any challenges you faced

📁 Submission Requirements

Each group must submit:

1️⃣ Source Code

- Provide a well-documented **C, C++, or Java program**.
- Use **command-line arguments or a menu-driven approach** for user input.

2️⃣ Sample Input & Output

- Submit a test file and corresponding output.
- **Example Output (Gantt Chart Representation):**

```
| P1 | P2 | P3 | P1 | P4 |
0    2    5    7    12   15
```

- Display **waiting time, turnaround time, and CPU utilization** in the final output.

3️⃣ Report (2-4 Pages, PDF)

- **Overview**: What algorithms were implemented?
- **Implementation Details**: How did you handle process scheduling?
- **Results**: Show sample runs and performance comparison (e.g., FCFS vs. RR).
- **Challenges & Solutions**: What difficulties did you face?

📊 Grading Criteria

Category	Points	Description
Correct Implementation of Two Scheduling Algorithms	40	Must work correctly & produce expected results
Gantt Chart & Performance Metrics	20	Shows execution order, waiting time, and TAT
File Handling & Process Input	10	Reads input file correctly
Code Quality & Documentation	10	Well-structured, readable, and commented code
Report Quality	10	Clear explanation & sample outputs
Bonus (Memory Management)	10	Extra points for implementing memory management

---

📢 Final Notes

✔ **Collaboration is encouraged, but each project must be unique.**

✔ **You do NOT need to pay for anything** – Use free tools like C, C++, or Java.
✔ **No special software is required** – A basic text editor and compiler are enough.
✔ **Test your program with different process sets before submitting.**
✔ **Think practically—how does an OS efficiently schedule processes?**

Why This Project?

✅ **Real-World Application** – This project helps you understand **how real operating systems handle CPU scheduling**.
✅ **Hands-On Learning** – By writing scheduling algorithms, you gain practical **OS programming experience**.
✅ **Scalable** – Can be completed in **C, C++, or Java** based on your experience.
✅ **Engaging** – Instead of theoretical learning, you **simulate real process scheduling**.

✅ Key OS Concepts Covered

- **Process Scheduling** (Core topic in OS)
- **CPU Scheduling Algorithms** (FCFS, SJF, RR, Priority)
- **Memory Management** (Optional but beneficial)
- **File Handling** (Reading process data from a file)

✅ Real-World Applicability

- These concepts **directly translate** to real-world **OS scheduling mechanisms**.
- Helps you **prepare for exams, interviews, and OS-related jobs**.