SmartCare: Hospital Patient Scheduling Simulator

Final Project Report – CMPSC Operating Systems

Boston Karymshakov & Victor Liu

Date: Dec 10, 2025

1. Description of the Project

For our final project, we built a small system called **SmartCare: Hospital Patient Scheduling Simulator**. The main idea was to take what we learned about CPU scheduling and apply it to something that happens in real life how hospitals decide the order in which patients get treated.

In our simulator, every patient is treated like an OS process with attributes such as arrival time, burst (treatment) time, and priority (how severe their condition is). The backend handles all the scheduling calculations, and the GUI displays the results in a clean format.

We implemented four scheduling algorithms (FCFS, SJF, Priority, and Round Robin), and the program shows the execution order, waiting times, turnaround times, and averages. The whole goal was to see how concepts from operating systems can help improve medical workflow and fairness.

2. Significance of the Project

Efficient patient scheduling is one of the biggest challenges in modern healthcare systems. Hospitals deal with limited staff, emergency cases, unpredictable arrival times, and resource shortages. Long patient waits times directly impact patient safety, satisfaction, and health outcomes.

Our simulator is significant because:

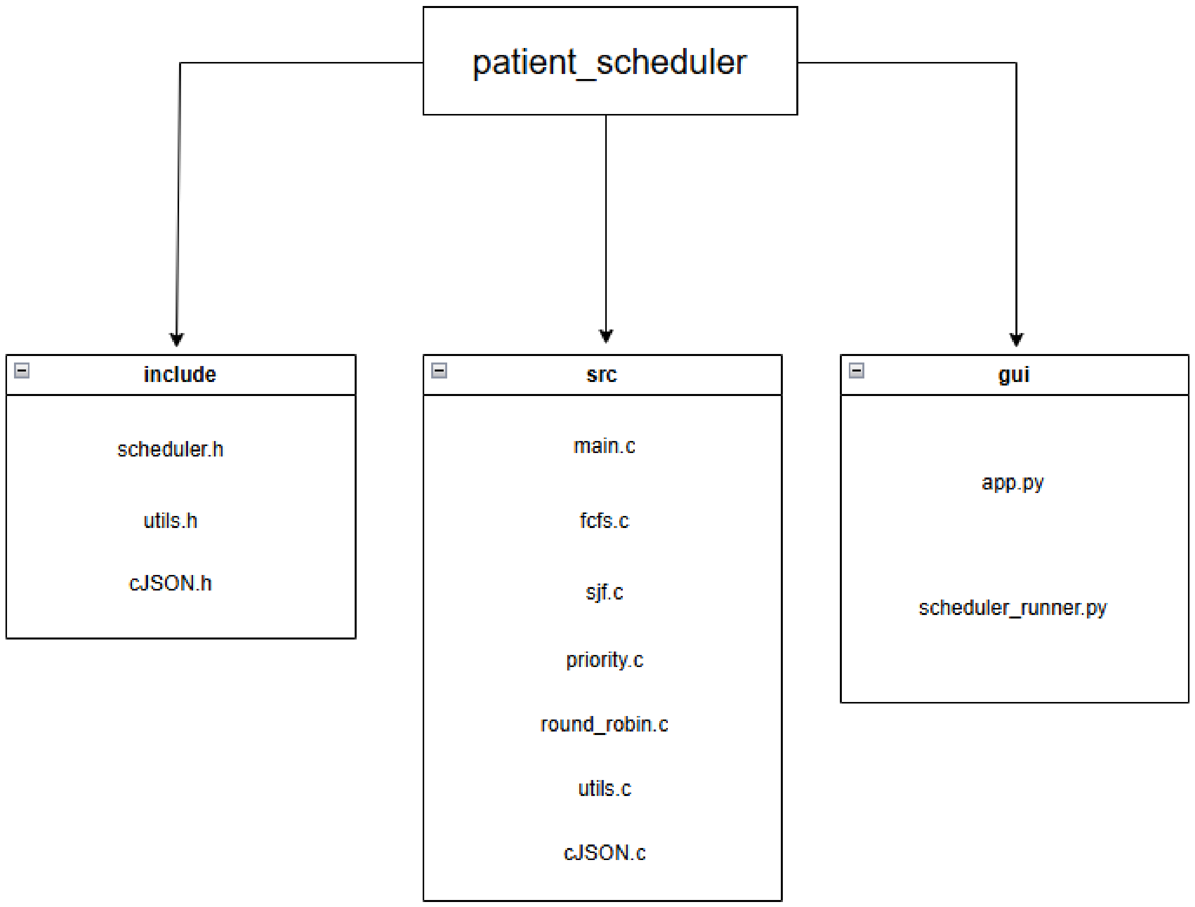
- It demonstrates how OS scheduling algorithms can be adapted to hospital workflows.
- It provides a simple but effective model for optimizing patient flow.
- It compares different scheduling approaches and highlights their strengths and weaknesses.
- It serves as a learning tool for exploring real-world scheduling, triage, and resource allocation problems.
- It reinforces core OS concepts in a meaningful, applied context relevant to public health.

Although simplified, the project mimics real hospital scheduling challenges and shows how computational approaches can improve fairness and efficiency in treatment delivery.

3. Code Structure

We organized our project into multiple modules to keep things clean and easy to understand. We separated scheduling logic, data structures, utility functions, and GUI backend communication.

3.1 Code Structure Diagram:



3.2 Code Structure Explanation

include/

This folder holds all the header files we use across the project.

- **scheduler.h**

  Defines the patient struct, scheduling function prototypes, and shared types used by the algorithms.

- **utils.h**

  Contains helper function prototypes (sorting, calculations, safe copying, etc).

- **cJSON.h**

  Header for the cJSON library we used to generate JSON output in C.

src/

- **main.c**

  Reads input provided by the GUI, calls the selected scheduling algorithm, and prints the results back as JSON.

- **fcfs.c**

  Implements First-Come First-Serve scheduling.

- **sjf.c**

  Implements the Shortest Job First scheduling.

- **priority.c**

  Handles the priority scheduling logic.

- **round_robin.c**

  Implements Round Robin using a time quantum and remaining times.

- **utils.c**

  Shared helper code for copying arrays, calculating waiting and turnaround times, sorting by arrival or priority, etc.

- **cJSON.c**

  The actual implementation of the cJSON library so our C program can output valid JSON for the GUI.

gui/

- **app.py**

  The front-end interface that collects user input. It displays the results visually after calling the C backend.

- **scheduler_runner.py**

  Responsible for taking the GUI input, formatting it, running the C executable and reading the JSON output so Streamlit can display it.

4. Description of Algorithms

Our project implements four CPU scheduling algorithms commonly taught in operating systems:

4.1 First Come First Serve (FCFS)

- Non-preemptive
- Patients are treated in the order they arrive.
- Simple but may cause long waiting times for later patients.

4.2 Shortest Job First (SJF)

- Non-preemptive
- Patients with the shortest treatment time goes first.
- Reduces average waiting time but may starve long-duration patients.

4.3 Priority Scheduling

- Non-preemptive
- Patients with highest medical severity are treated first.
- Models’ emergency room triage realistically.

4.4 Round Robin (RR)

- Preemptive
- Each patient receives a fixed time slice.
- Fair for time-sharing but may take longer for emergencies.

5. Verification of Algorithms with Toy Examples:

For each scheduling algorithm, we tested small sample inputs by hand and compared them with the simulator's output. This helped confirm:

- Correct execution order
- Accurate waiting/turnaround time computation
- Proper handling of equal arrival times
- Priority tie-breaking rules
- Round Robin remaining time bookkeeping

These toy examples helped catch edge cases early, especially for Round Robin.

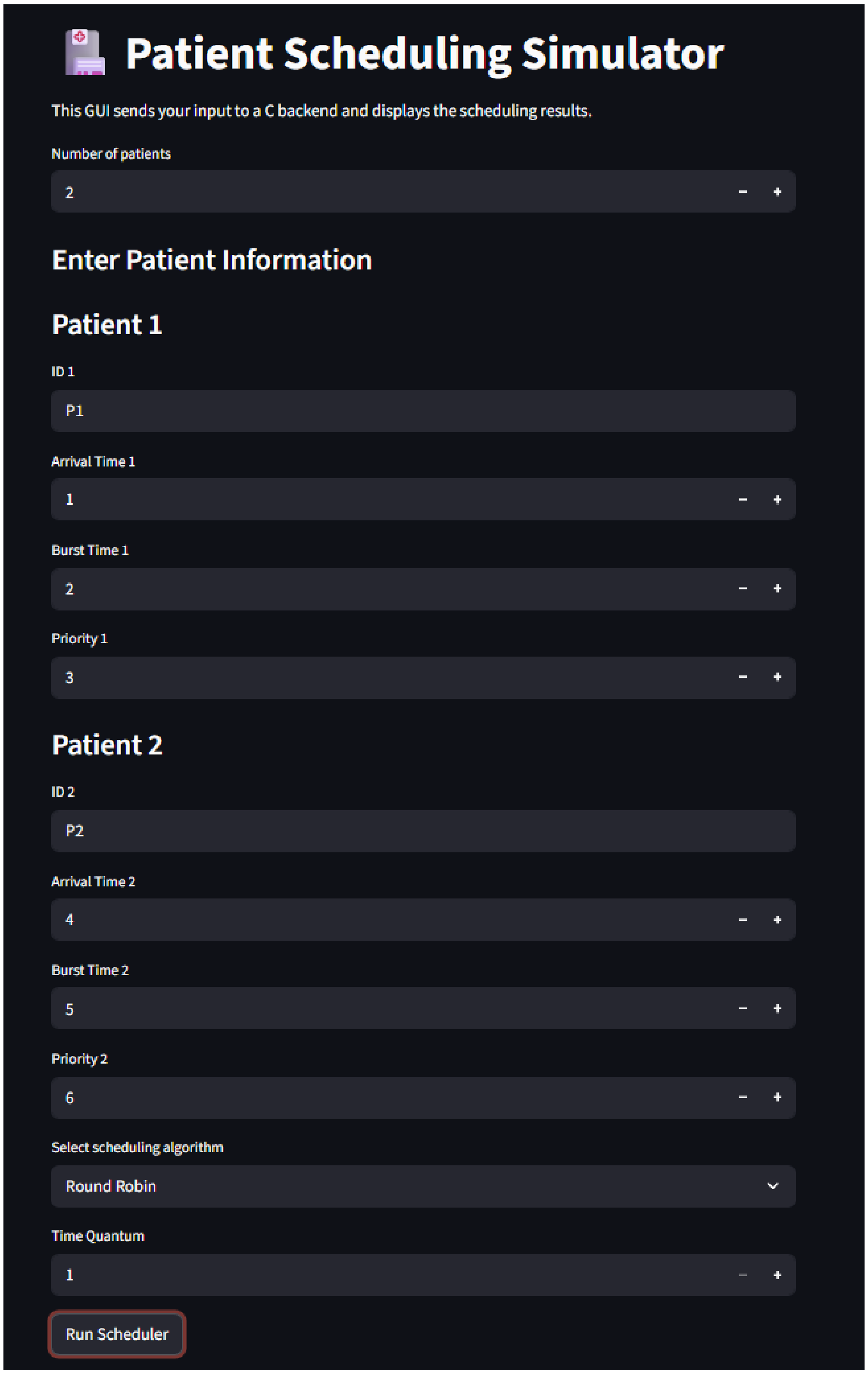
6. Functionalities of the System:

Our simulator supports:

- Adding patient records
    - ID
    - Arrival time
    - Burst/treatment time
    - Priority/severity
- Running four scheduling algorithms
- Displaying complete results:
    - Execution order
    - Waiting times
    - Turnaround times
    - Average values
- JSON-formatted output for GUI integration
- A clean and simple GUI for easier use

Overall, it behaves like a small real-world scheduling tool.

7. Execution Results and Analysis:



Patient Scheduling Simulator

This GUI sends your input to a C backend and displays the scheduling results.

Number of patients: 2

Enter Patient Information

Patient 1

ID 1: P1

Arrival Time 1: 1

Burst Time 1: 2

Priority 1: 3

Patient 2

ID 2: P2

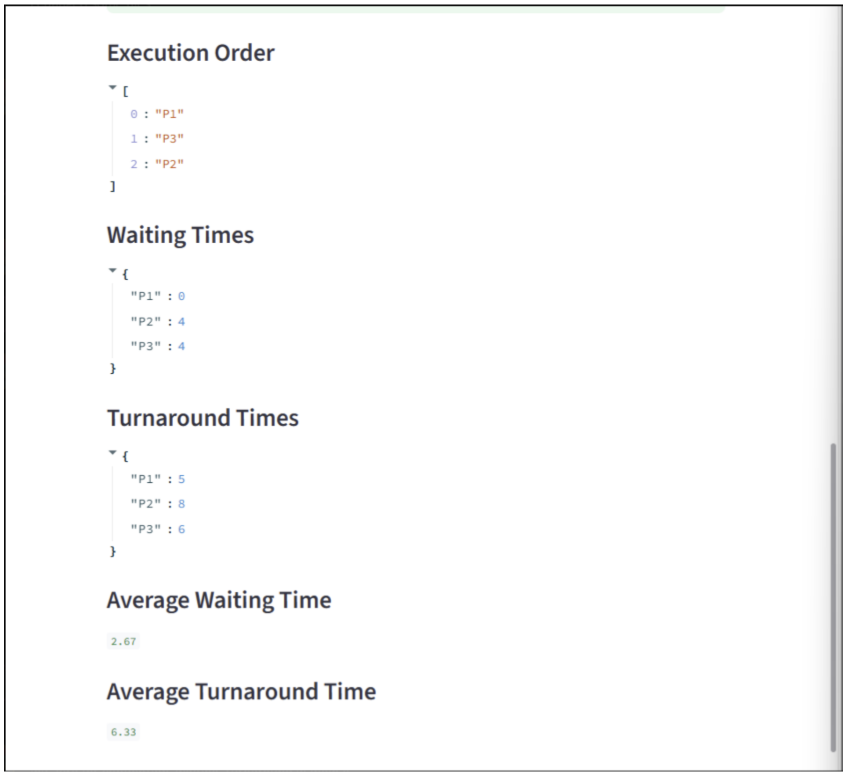
Arrival Time 2: 4

Burst Time 2: 5

Priority 2: 6

Select scheduling algorithm: Round Robin

Time Quantum: 1



Execution Order

```
[
  0 : "P1"
  1 : "P3"
  2 : "P2"
]
```

Waiting Times

```
{
  "P1" : 0
  "P2" : 4
  "P3" : 4
}
```

Turnaround Times

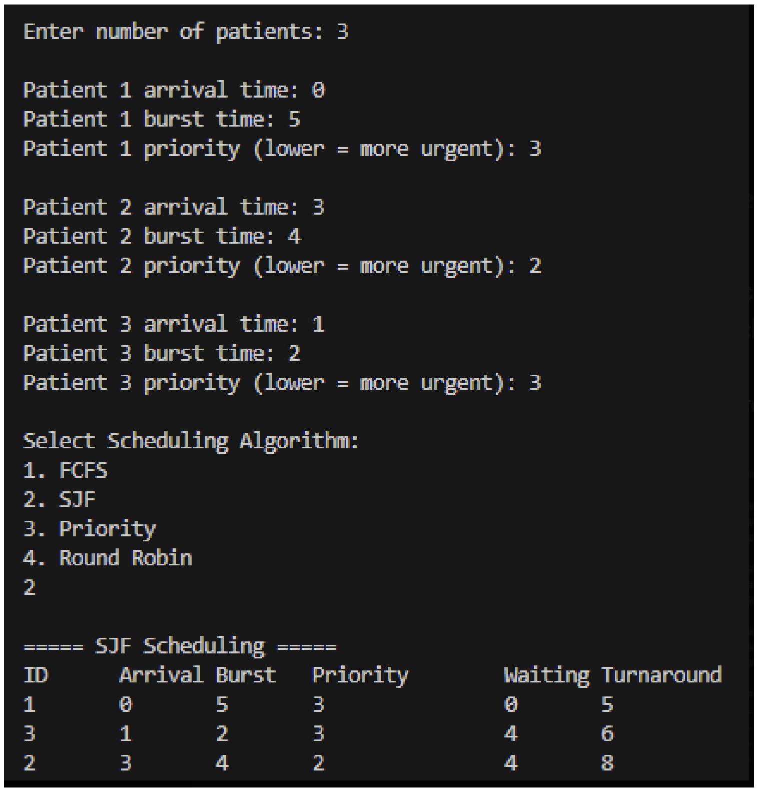
```
{
  "P1" : 5
  "P2" : 8
  "P3" : 6
}
```

Average Waiting Time

`2.67`

Average Turnaround Time

`6.33`



```
Enter number of patients: 3

Patient 1 arrival time: 0
Patient 1 burst time: 5
Patient 1 priority (lower = more urgent): 3

Patient 2 arrival time: 3
Patient 2 burst time: 4
Patient 2 priority (lower = more urgent): 2

Patient 3 arrival time: 1
Patient 3 burst time: 2
Patient 3 priority (lower = more urgent): 3

Select Scheduling Algorithm:
1. FCFS
2. SJF
3. Priority
4. Round Robin
2

===== SJF Scheduling =====
ID      Arrival Burst   Priority        Waiting Turnaround
1       0       5       3               0       5
3       1       2       3               4       6
2       3       4       2               4       8
```

The results showed how different algorithms change patient wait times:

- FCFS works well when arrivals are well-spaced but causes long delays if a long “job” comes early.
- SJF had the lowest average waiting time but wasn’t always fair.
- Priority behaved the most like a real hospital but could delay non-critical patients for a long time.
- Round Robin balanced fairness but wasn’t ideal for emergency-style scenarios.

Comparing the results helped us understand the trade-offs between fairness, efficiency, and response time.

8. Conclusion:

This project demonstrated how operating system scheduling strategies can be applied to real healthcare challenges such as patient management and triage. By modeling patients as processes and applying FCFS, SJF, Priority Scheduling, and Round Robin, we gained insight into how scheduling decisions impact patient waiting times and resource allocation.

We learned how important algorithm choice is for fairness and efficiency, especially in high-stakes environments like hospitals. The project helped reinforce OS concepts such as waiting time, turnaround time, preemption, and process prioritization.

Project Issues:

- Implementing Round Robin required careful handling of remaining times.
- Sorting by both arrival and priority introduced edge cases.
- JSON parsing and formatting
- Code compiling and GUI errors

Course Learning Applied:

- CPU scheduling algorithms
- Process lifecycle concepts
- Modularity, struct usage, and system-level coding in C
- Runtime computation and real-world modeling
- How to organize a larger C project with multiple files
- Parsing and producing JSON from C
- Building a GUI that communicates with a C backend

Overall, this project helped us strengthen our understanding of OS scheduling and gave us experience modeling real-world systems.

9. GitHub Repository:

https://github.com/Spektra29/Hospital-Patient-Scheduling-Simulator.git