Hospital Resource Scheduler

CMPSC 472 GitHub **Avik Bhuiyan & Gabriel Silva**

Project Description

Since modern hospitals need to be efficient and allocate their limited resources such as doctors nurses, operating rooms, etc. They also balance fairness and urgency, and poor scheduling can lead to long wait times, underutilized resources and reduced quality of care.

Hospital Resource Scheduler is a web application displaying features of scheduling for healthcare resources. The main goal of this project is to model real-world healthcare scheduling scenarios using operating system scheduling algorithms and evaluate their effectiveness in different contexts. The system allows the users to submit patient requests with priority levels and schedules them using three different types of scheduling algorithms which are FCFS, Priority, and Round Robin. We aimed to compare these three different approaches.

Significance

Healthcare environments are very dynamic and time sensitive. Emergency cases have to be prioritized while more routine procedures still require guaranteed access to resources over time. This project is significant because it Demonstrates the application of scheduling algorithms to healthcare systems, it highlights the tradeoffs of fairness, efficiency, and responsiveness. It Shows a visual and interactive way to compare scheduling strategies. By using the patients as processes and medical resources as limited system resources, it creates a realistic simulation of a real-life hospital workflow challenge.

Code Structure

The backend is organized into clean modular components so it is easy to understand and maintain.

```
HospitalResourceScheduler/        /* project directory */
    Backend/                      /* back-end folder */
        Models/                   /* Pydantic models for requests and resources */
    Schedulers/                   /* where FCFS, Priority, and Round Robin are located */
        Resource_manager.py       /* centralized resource allocation and tracking of
                                     schedulers */
```

Main.py /* main application entry point running FastAPI */

Scheduler-frontend/ /* React front-end folder */

Description of Algorithms

First-Come-First-Served (FCFS):

The FCFS schedules each patient request strictly in their order of arrival, and when each request begins, it runs until it's completed and each patient is seen. Some of the characteristics of FCFS are that it's simple and predictable, and there is no prioritization or fairness guarantees. In terms of health care applications, it's suitable for non-emergency clinics or appointment-based services.

Priority Scheduling:

Each request is ordered by its priority level, meaning the highest priority patient is scheduled first. Some of its characteristics are its fast response for critical cases and it risks of starvation for lower priority patients. In healthcare it's more ideal for emergency departments where the more serious or urgent cases must go above the routine procedures

Round Robin Scheduling:

Every request receives a fixed time slice, and if the task is not completed, it is placed back into the queue. Its characteristics are that it has fair resource sharing, it prevents starvation, and it has a higher overhead since it uses context switching. It's useful when a lot of patients are competing for shared equipment such as an MRI machine.

Verification of Algorithms

Algorithm correctness was verified through controlled tests in test_scheduling.py. We tested each scheduling algorithm and also compared them with each other. The outputs were:

```
================================================================================
 TEST 1: FIRST-COME-FIRST-SERVED (FCFS) SCHEDULER
================================================================================

 INPUT - Patient Requests (in arrival order):
  1. Alice      needs Doctor          Priority: LOW      Duration: 30 min
  2. Bob        needs Doctor          Priority: HIGH     Duration: 45 min
  3. Charlie    needs Bed             Priority: MEDIUM   Duration: 20 min
  4. Diana      needs Doctor          Priority: HIGH     Duration: 60 min

 EXPECTED OUTPUT:
  FCFS processes requests in arrival order (ignores priority):
  1. Alice   → gets Doctor (arrived first, despite LOW priority)
  2. Bob     → gets Doctor
  3. Charlie → gets Bed
  4. Diana   → gets Doctor

 ACTUAL OUTPUT:
  1. Alice      → Dr. A (Doctor)
  2. Bob        → Dr. B (Doctor)
  3. Charlie    → Bed 1 (Bed)
  4. Diana      → Dr. C (Doctor)

 KEY INSIGHT:
  - FCFS serves requests in First-In-First-Out order
  - Alice (LOW priority) is served before Bob (HIGH priority)
  - Simple and fair, but ignores urgency
```

```
================================================================================
 TEST 2: PRIORITY-BASED SCHEDULER
================================================================================

 INPUT - Patient Requests (same as Test 1):
  1. Alice      needs Doctor          Priority: LOW      Duration: 30 min
  2. Bob        needs Doctor          Priority: HIGH     Duration: 45 min
  3. Charlie    needs Bed             Priority: MEDIUM   Duration: 20 min
  4. Diana      needs Doctor          Priority: HIGH     Duration: 60 min
  5. Eve        needs Doctor          Priority: MEDIUM   Duration: 25 min

 EXPECTED OUTPUT:
  Priority scheduler processes HIGH priority first:
  1. Bob     → gets Doctor (HIGH priority)
  2. Diana   → gets Doctor (HIGH priority)
  3. Eve     → gets Doctor (MEDIUM priority)
  4. Charlie → gets Bed (MEDIUM priority)
  5. Alice   → gets Doctor (LOW priority, served last)

 ACTUAL OUTPUT:
  1. Bob        → Dr. A (Doctor) [Priority: HIGH]
  2. Diana      → Dr. B (Doctor) [Priority: HIGH]
  3. Charlie    → Bed 1 (Bed) [Priority: MEDIUM]
  4. Eve        → Dr. C (Doctor) [Priority: MEDIUM]
  5. Alice      → Dr. D (Doctor) [Priority: LOW]

 KEY INSIGHT:
  - Priority scheduler serves urgent cases first
  - Bob and Diana (HIGH) get resources before Alice (LOW)
  - Risk: LOW priority requests may experience starvation
```

```
 TEST 3: ROUND ROBIN SCHEDULER
================================================================================

 INPUT - Patient Requests (Time Quantum: 30 min):
  1. Patient-A needs Doctor          Duration: 90 min (3 time slices)
  2. Patient-B needs Doctor          Duration: 45 min (2 time slices)
  3. Patient-C needs Doctor          Duration: 20 min (1 time slices)
  4. Patient-D needs Bed             Duration: 60 min (2 time slices)

 EXPECTED OUTPUT:
  Round Robin gives each request 30 minutes at a time:
  Round 1:
    - Patient-A: 30 min (60 remaining)
    - Patient-B: 30 min (15 remaining)
    - Patient-C: 20 min (COMPLETE)
    - Patient-D: 30 min (30 remaining)
  Round 2:
    - Patient-A: 30 min (30 remaining)
    - Patient-B: 15 min (COMPLETE)
    - Patient-D: 30 min (COMPLETE)
  Round 3:
    - Patient-A: 30 min (COMPLETE)

 ACTUAL OUTPUT:
  1. Patient-A → Dr. A (90 min = 3 time slices)
  2. Patient-B → Dr. B (45 min = 2 time slices)
  3. Patient-C → Dr. C (20 min = 1 time slices)
  4. Patient-D → Bed 1 (60 min = 2 time slices)

 KEY INSIGHT:
  - Round Robin provides fair CPU time-sharing
  - Short jobs (Patient-C: 20 min) complete quickly
  - Long jobs (Patient-A: 90 min) don't monopolize resources
  - Prevents starvation, ensures fairness
```

```
================================================================================
 TEST 4: ALGORITHM COMPARISON
================================================================================

 Common Input (4 patients needing DOCTOR):
  1. Emergency-1 (HIGH priority,   45 min)
  2. Routine-1   (LOW priority,    30 min)
  3. Urgent-1    (MEDIUM priority, 60 min)
  4. Emergency-2 (HIGH priority,   20 min)

 RESULTS COMPARISON:

  FCFS:
    Order: Emergency-1 → Routine-1 → Urgent-1 → Emergency-2

  Priority:
    Order: Emergency-1 → Emergency-2 → Urgent-1 → Routine-1

  Round Robin:
    Order: Emergency-1 → Routine-1 → Urgent-1 → Emergency-2
```

```
================================================================================
 TEST 5: RESOURCE SHORTAGE SCENARIO
================================================================================

 INPUT:
  10 patients need DOCTOR, but only 8 doctors available

   1. Patient-1 (Priority: HIGH)
   2. Patient-2 (Priority: HIGH)
   3. Patient-3 (Priority: HIGH)
   4. Patient-4 (Priority: MEDIUM)
   5. Patient-5 (Priority: MEDIUM)
   6. Patient-6 (Priority: MEDIUM)
   7. Patient-7 (Priority: MEDIUM)
   8. Patient-8 (Priority: LOW)
   9. Patient-9 (Priority: LOW)
  10. Patient-10 (Priority: LOW)

 EXPECTED OUTPUT:
  Priority scheduler allocates to highest priority first:
  - Patient-1, Patient-2, Patient-3 (HIGH) get doctors
  - Patient-4 through Patient-8 (MEDIUM) get doctors
  - Patient-9, Patient-10 (LOW) remain in queue (insufficient resources)

 ACTUAL OUTPUT:
  Allocated: 8/10 patients
   1. Patient-1 → Dr. A [HIGH]
   2. Patient-2 → Dr. B [HIGH]
   3. Patient-3 → Dr. C [HIGH]
   4. Patient-4 → Dr. D [MEDIUM]
   5. Patient-5 → Dr. E [MEDIUM]
   6. Patient-6 → Dr. F [MEDIUM]
   7. Patient-7 → Dr. G [MEDIUM]
   8. Patient-8 → Dr. H [LOW]

  Waiting: 2 patients remain in queue

 KEY INSIGHT:
  - When resources are scarce, priority scheduling ensures critical
    patients are served first
  - Lower priority patients must wait for resources to free up

================================================================================
  ALL TESTS COMPLETED
================================================================================
```

The verifications of tests show that our core algoeithms for our system are running as expected.

Functionalities

Our System has the following features:

FastAPI Back-end:

- Add patient requests as processes with configurable priority levels
- Maintain beds, doctors, and operating rooms
- Track resource status
- REST API endpoints for integration and testing

React Front-end:

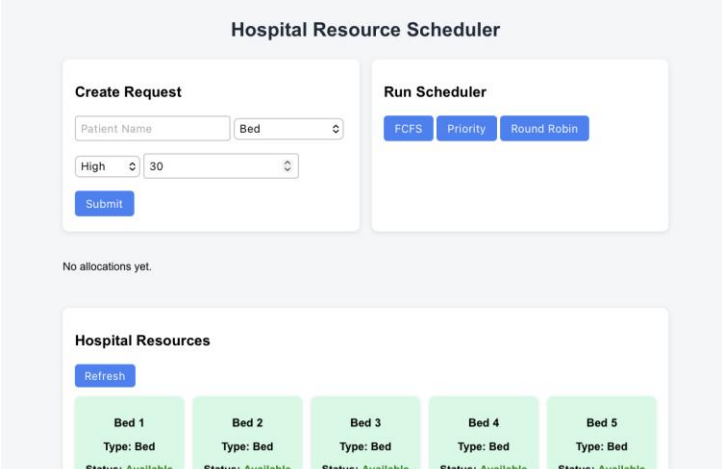
- View real-time availability of healthcare resources
- Display scheduling results and allocation order
- Select and execute different scheduling algorithms

Results and Analysis

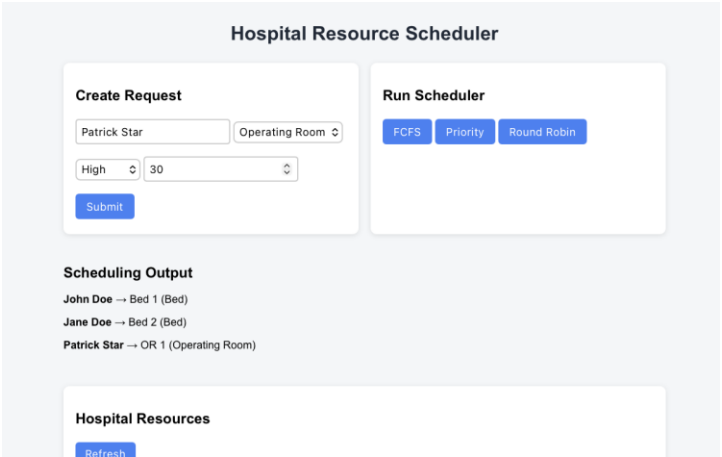
The front-end features 3 main panel sections. These are:

1. **Create Request**
   This panel allows users to add a patient to the schedule for hospital resources. They can input the patient's name, what they need, such as a bed or operating room, and also the duration of how long.
2. **Run Scheduler**
   This panel allows users to select one of the scheduling algorithms, including FCFS, Priority, and Round Robin. After selected an algorithm, the results are displayed as seen below in the "Scheduling Output."
3. **Hospital Resources**
   The most significant panel is the visualization of hospital resources at the bottom. This displays all of the beds, doctors and operating rooms in a color-coded format; green being open and red being occupied. There is also a refreshing button to update the grid.

Before Scheduling



After Scheduling



After testing different scheduling algorithms, we found that *Priority Scheduling* was ideal for emergencies. The algorithms behaved as expected, as the FCFS did not perform well under stress, and round robin was able to deliver fair resources to each request.

The front-end made it easy for the user to digest all of the information and quickly be able to determine what is occupied and vacant.

Conclusion

Overall, this project demonstrates operating system concepts learned throughout the course including FCFS, Round Robin, and Priority scheduling. We learned how these OS scheduling concepts can be applied to real life and implemented it into a hospital resource scheduling tool.

Our implementation demonstrates the operating concepts of learning by having the processes symbolize patients and hospital medical resources and multiple priorities showing different ways of responding to different responses and fairness affect the patient and resources.