

**The Superior University**

**📝 Operating Systems Lab – Project Documentation Template**

**📌 Project Title**

*Simulation of FCFS-Based Parking Lot Management System in Python (e.g., “Simulation of Round Robin Scheduling in Python”)*

**👥 Group Members**

💡 *Project can be completed individually or in a group of up to 4 members.*

*List all team members with roll numbers:*

* **Faiza Mushtaq Ahmad** (Roll No. SU92-BSSEM-F23-056)

**📂 GitHub Repository**

✅ **Both the Python code file and this documentation must be uploaded to a public GitHub repository.**

**GitHub Repository Link:**

[https://github.com/faizamushtaqahmad/Parking-Lot-Management-System]

**🔧 Scheduling Algorithm Implemented**

✅ Tick the scheduling algorithm your group implemented:

* FCFS (First Come First Serve)
* SJF (Shortest Job First – Non-Preemptive)
* SJF (Preemptive)
* Round Robin

**📄 Project Description**

**• What problem your project solves:**  
The project solves the problem of managing a parking lot efficiently using the First-Come-First-Served (FCFS) scheduling algorithm. It handles vehicle arrivals, departures, waiting queues, billing, and provides performance metrics like average waiting and turnaround time.

**• What inputs are required (arrival time, burst time, time quantum):**  
The required inputs are:

* Vehicle number
* Arrival time (in hours)
* Departure time (in hours)
* Total parking slots
* Parking rate per hour

**• What outputs are generated:**  
The program generates:

* Real-time status of parking slots
* Notifications for parking and removals
* Waiting queue updates
* Billing information for each vehicle
* A table of vehicle parking details
* A Gantt chart for visual representation
* Average waiting time and turnaround time

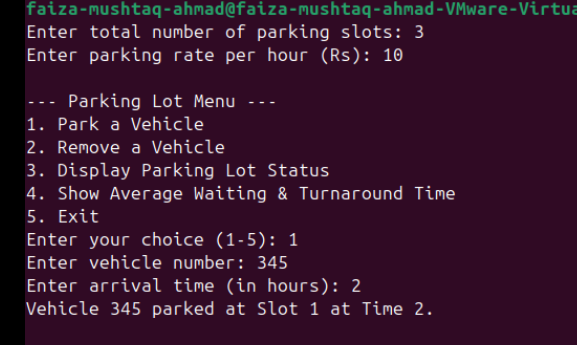
**How the algorithm is implemented**

* The FCFS algorithm is implemented by allocating parking slots to vehicles in the order of their arrival.
* If all slots are full, vehicles wait in a queue and are assigned the next available slot in order.
* Vehicle records are stored and analyzed to compute performance metrics and generate visual reports.

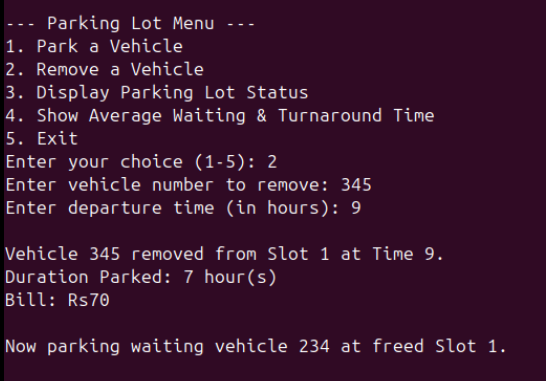
**📸 Output Screenshots**

**1st Output (On Linux):**

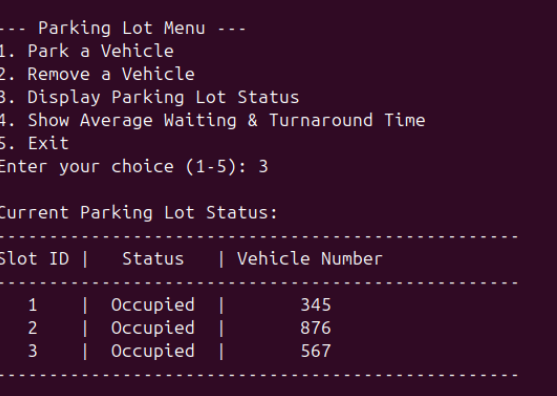
**Park A Vehicle:**

****

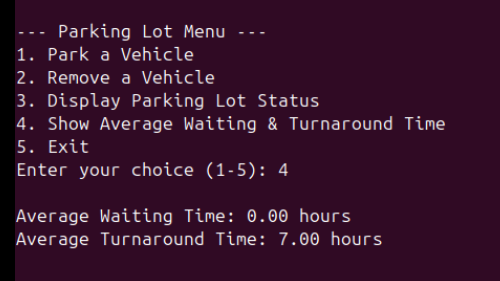
**Remove A Vehicle:**

****

**Display Parking Lot Status:**

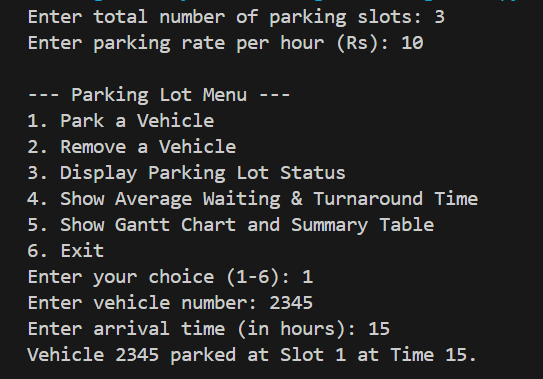
****

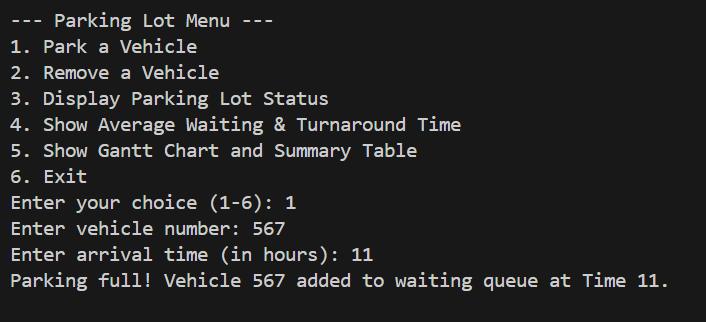
**Average waiting and turnaround time:**

****

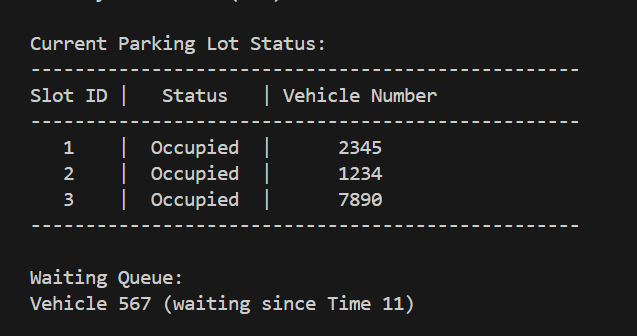
**2nd Output (VS code):**

**Park A Vehicle:**

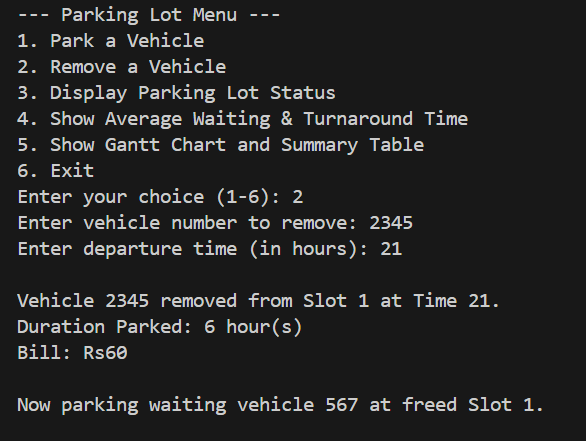
****

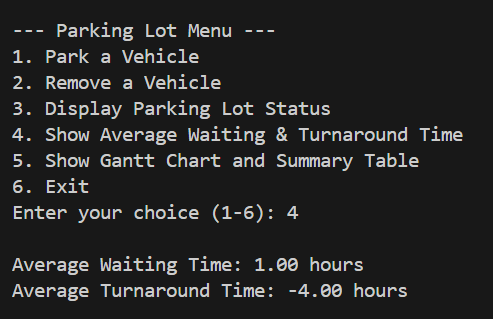
**Parking Full:  
**

**Display Status:**

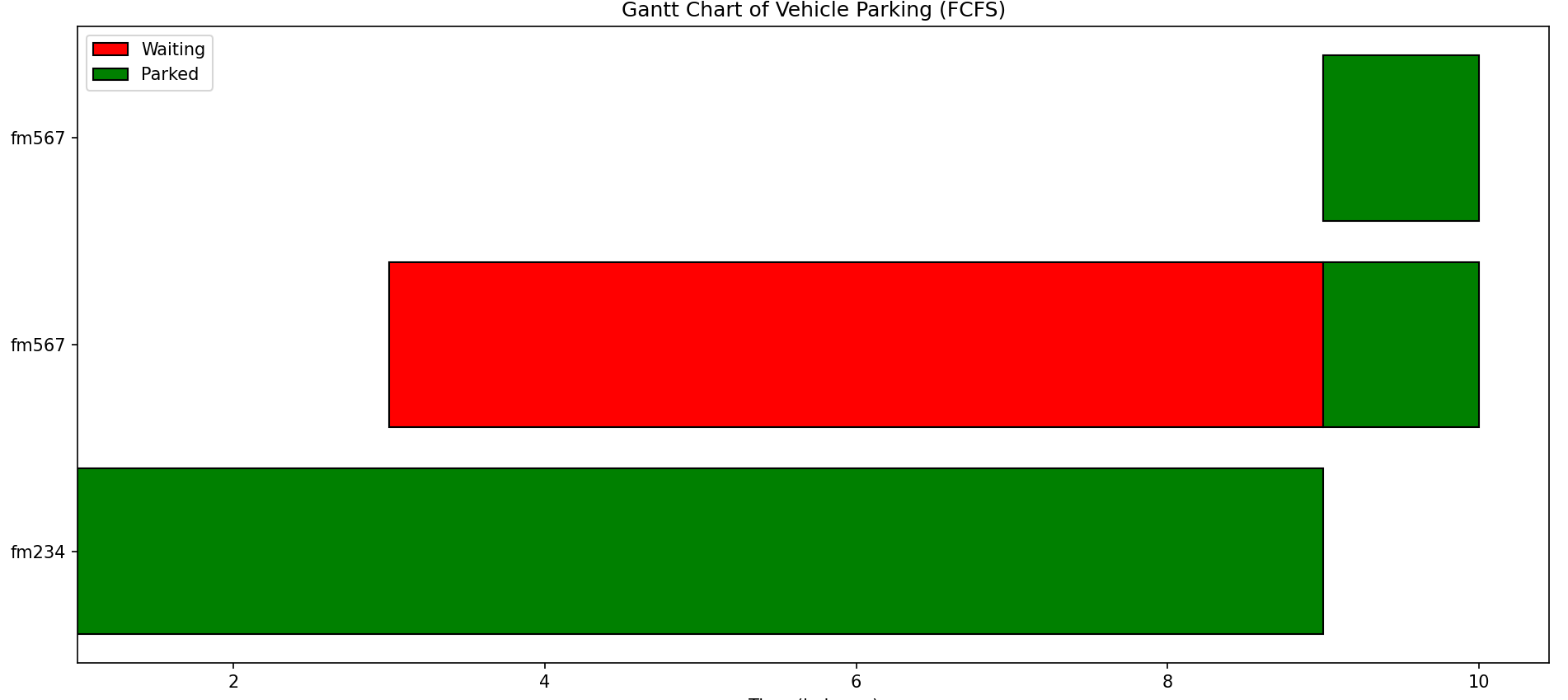
****

**Remove A Vehicle:**

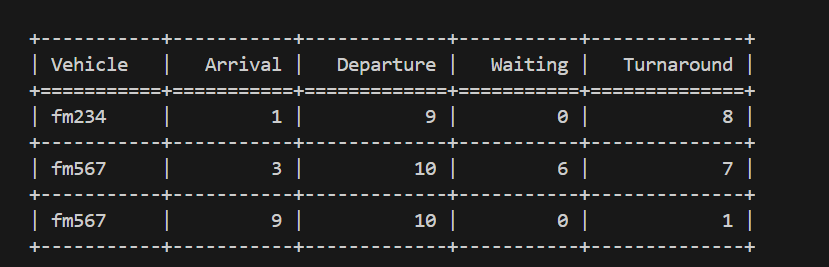
****

**Average Waiting Time:  
**

**Gantt Chart:**

****

**Summary Table:**

****

**🧠 Code Structure & Explanation**

**• Functions or classes used:**  
The project uses two main classes:

1. **ParkingSlot** – Represents an individual parking slot. It keeps track of whether the slot is free, the vehicle parked, and its arrival time.
2. **ParkingLot** – Manages the entire parking system. It handles vehicle parking, removal, waiting queue management, billing, statistics (average times), and output generation (status, tables, and Gantt chart). It contains key methods like:
   * park\_vehicle()
   * remove\_vehicle()
   * update\_departure()
   * display\_status()
   * calculate\_averages()
   * generate\_gantt\_and\_table()

The **main()** function serves as the user interface, allowing interaction via a menu-driven loop.

**• Core logic of the scheduling algorithm:**  
The project implements the **First-Come-First-Served (FCFS)** scheduling algorithm. Vehicles are parked in the order they arrive:

* If a slot is available, the vehicle is parked immediately.
* If all slots are full, the vehicle is placed in a **waiting queue**.
* When a parked vehicle leaves, the next vehicle from the queue (the one that arrived earliest) is assigned the freed slot.
* This simulates FCFS behavior, where tasks (vehicles) are handled strictly by arrival order.

**• Any external libraries used:**  
Two external libraries are used for formatting and visualization:

1. **matplotlib.pyplot** – Used to generate a **Gantt chart**, visually representing each vehicle’s waiting and parking duration on a timeline.
2. **tabulate** – Used to display a **neat table** of vehicle parking details (arrival time, departure time, waiting time, turnaround time) in a structured format.

**📊 Performance Metrics**

| **Metric** | **Value** |
| --- | --- |
| Average Waiting Time | (Auto calculated in output) |
| Average Turnaround Time | (Auto calculated in output) |
| Time Quantum (if RR) | N/A |

**🛠️ Challenges Faced**

### **Problem 1: Managing Full Parking Capacity and Waiting Queue**

**Solution:**  
Implemented a dynamic queue system to handle waiting vehicles. The system automatically assigns available parking slots to waiting vehicles as soon as a slot is freed, ensuring efficient parking lot management.

### **Problem 2: Accurate Calculation of Parking Duration and Billing**

**Solution:**  
Developed a mechanism to precisely track both arrival and departure times. This ensured accurate billing calculations, even for vehicles parked for less than an hour, and handled edge cases where the departure time was delayed.

### **Problem 3: Visualization of Parking Data Using Gantt Charts**

**Solution:**  
Utilized the **matplotlib** library to create a Gantt chart that visualized vehicle parking and waiting times. Refined the chart’s layout and labels to ensure clarity and readability, even with a larger number of vehicles.

**Summary:**

* **Parking Queue Management:**  
  We resolved the issue of handling full parking capacity by implementing a dynamic waiting queue. This ensured that vehicles waiting for a slot were assigned available spaces as soon as they became free, maintaining efficient lot management.
* **Accurate Billing and Duration Calculation:**  
  The problem of inaccurate billing and parking duration calculations was fixed by accurately tracking both arrival and departure times. This allowed us to ensure correct billing even for short parking durations and handle edge cases where departure times were delayed.
* **Gantt Chart Visualization:**  
  We overcame challenges in visualizing parking and waiting durations by using **matplotlib** to create a clear and accurate Gantt chart. Adjustments were made to improve the chart's readability, with proper alignment and labels to handle a larger number of vehicles effectively.