Ibrahim Abou Zahr

ID: 202300976

**OS Project**

1. **Introduction**:

In the following OS project, we explore the concept of process scheduling (something we have practiced in class) within OS using a scheduling simulator which was made to replicate how different algorithms manage CPU time among processes.

The simulator lets you create your own dataset of processes and to pick the different scheduling strategy that you desire. This includes: First-Come, First-Served (**FCFS**), Shortest Job First (**SJF**), Round Robin (**RR**), and **Priority Scheduling**.

The **primary objective** of this project is to extend our knowledge of CPU scheduling and to compare the performance of these strategies with different workloads. Also, by looking at the output, we would be able to understand what happened in each step. This builds critical thinking in OS design.

The app’s name is**: OS Algorithm Simulator by Rafael Lopez.**

**Part A, Project 3:**

1. **Problem Statement and Experimentation:**

In section A, I chose **Project Number 3**. The project provides the following given:

• Number of Processes = 12

• DOM = 20

• Number of Bursts = 15

• CPU Bursts Range: Minimum = 5, Maximum = 20

• I/O Bursts Range: Minimum = 20, Maximum = 100

• Priority: Minimum = 1, Maximum = 7

• Arrival time: Minimum = 0, Maximum = 25

• Context Switch = 2

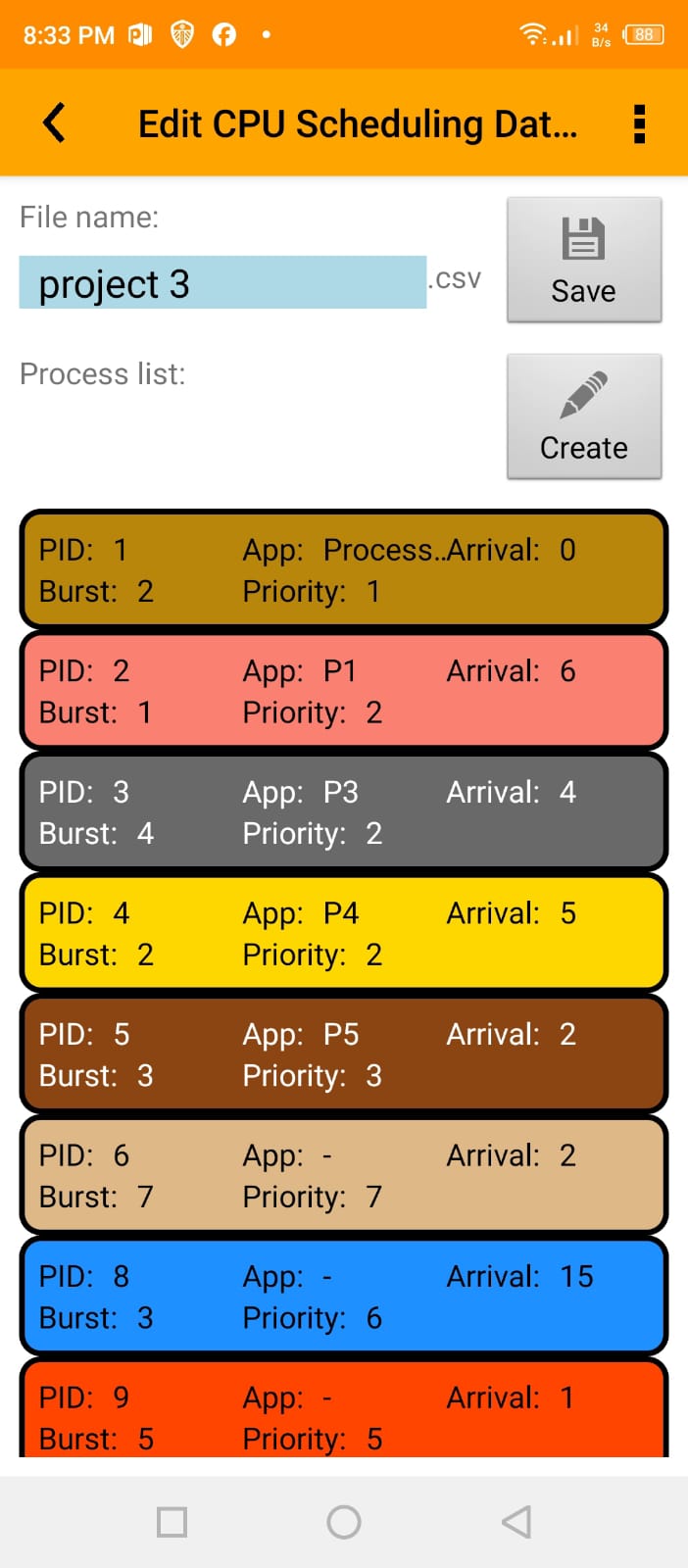
• Alpha = 0.5, Tau = 50

Use the “**FCFS**” Scheduling algorithm

To start off, before starting the experiment, I took a moment to review the app’s **guide/documentation** before trying anything. The creator of the app made navigating the simulator more user friendly when providing a comprehensive guide.

**Creating the Data:**

I started my experiment by creating the suitable csv file based on the provided given. I made my way to the **Create** data button and started entering every process value based on the limitations and quantity, below is a screenshot of this task:

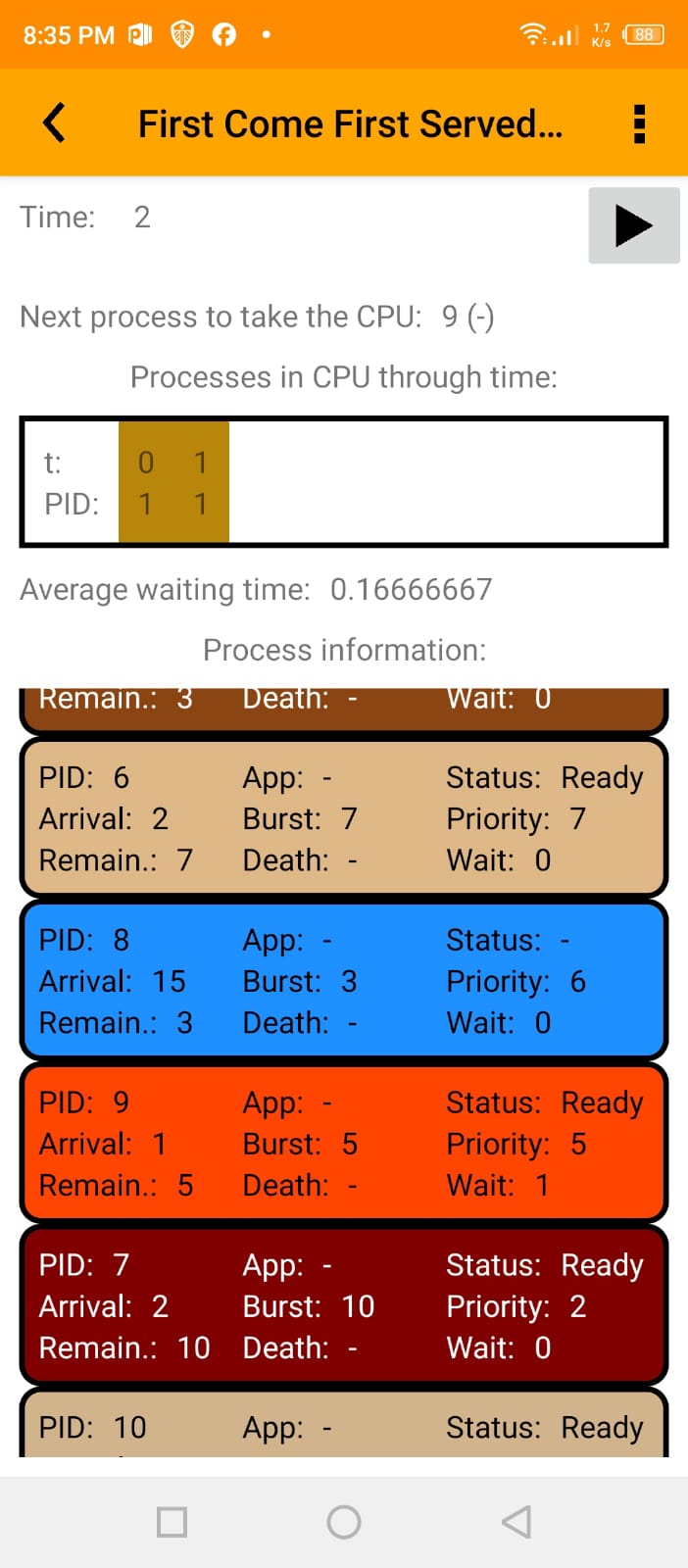
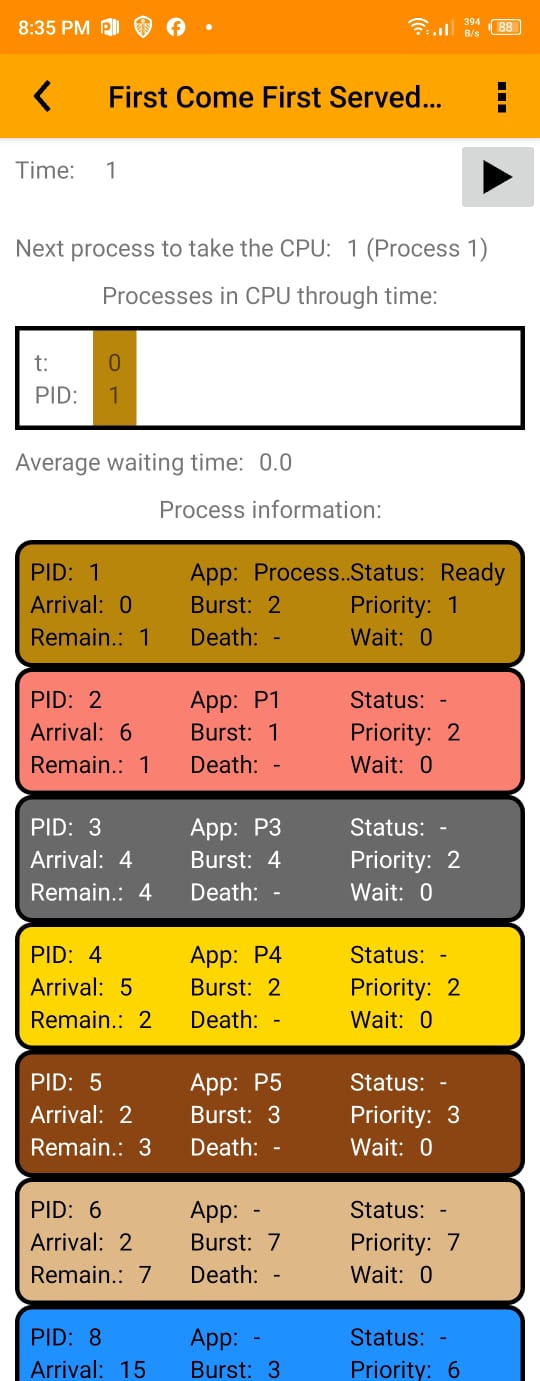
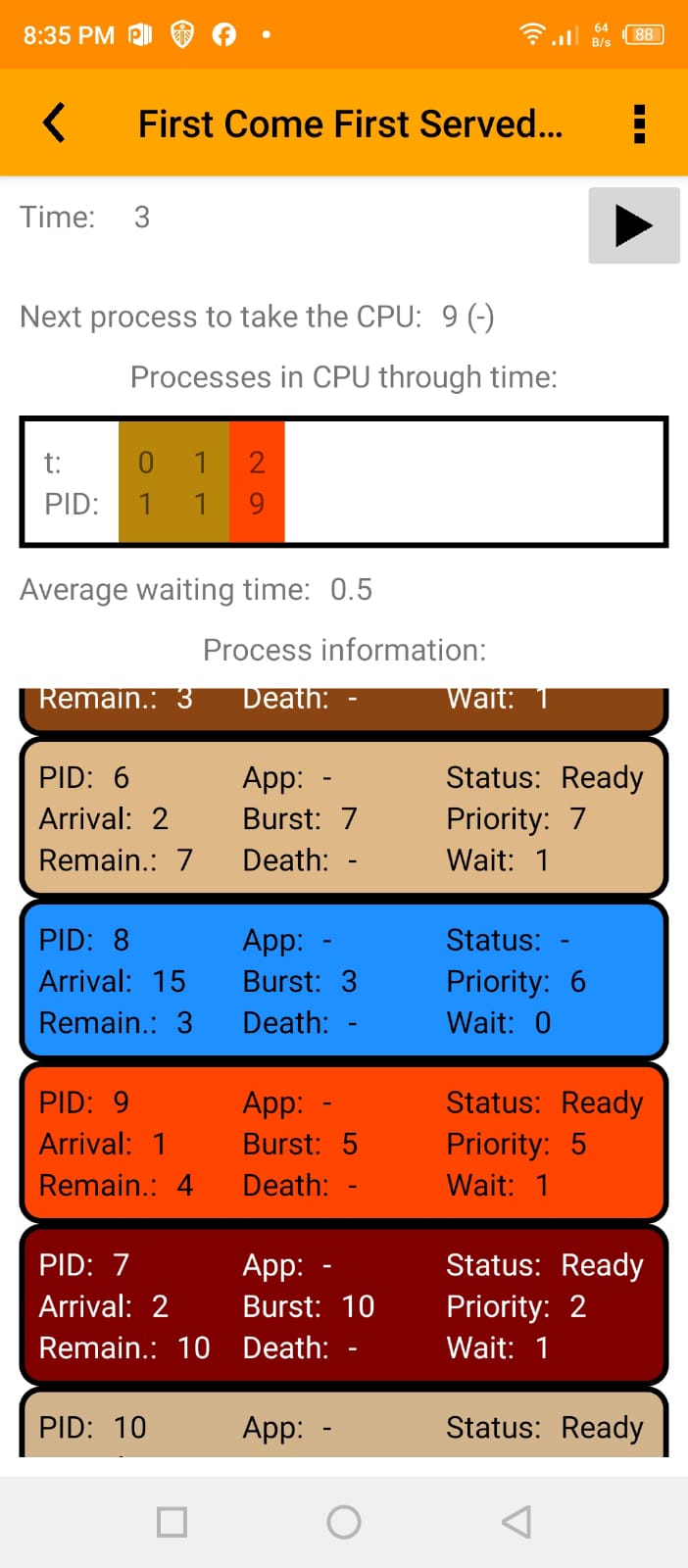


I created process 1 which had **arrival time** of 0, **burst** of 2 and **priority** 1. Then I did the rest of the processes all the way to **process number 12** which was the required quantity.

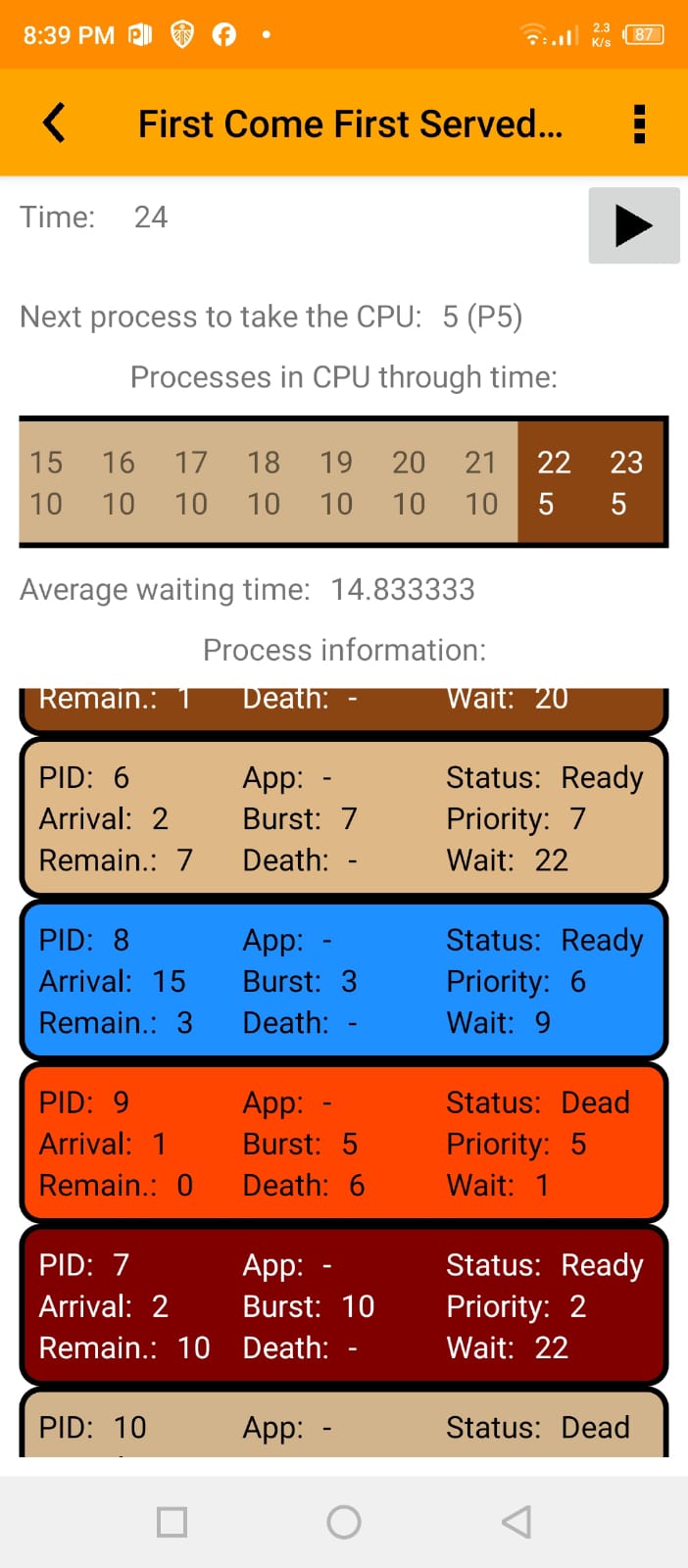
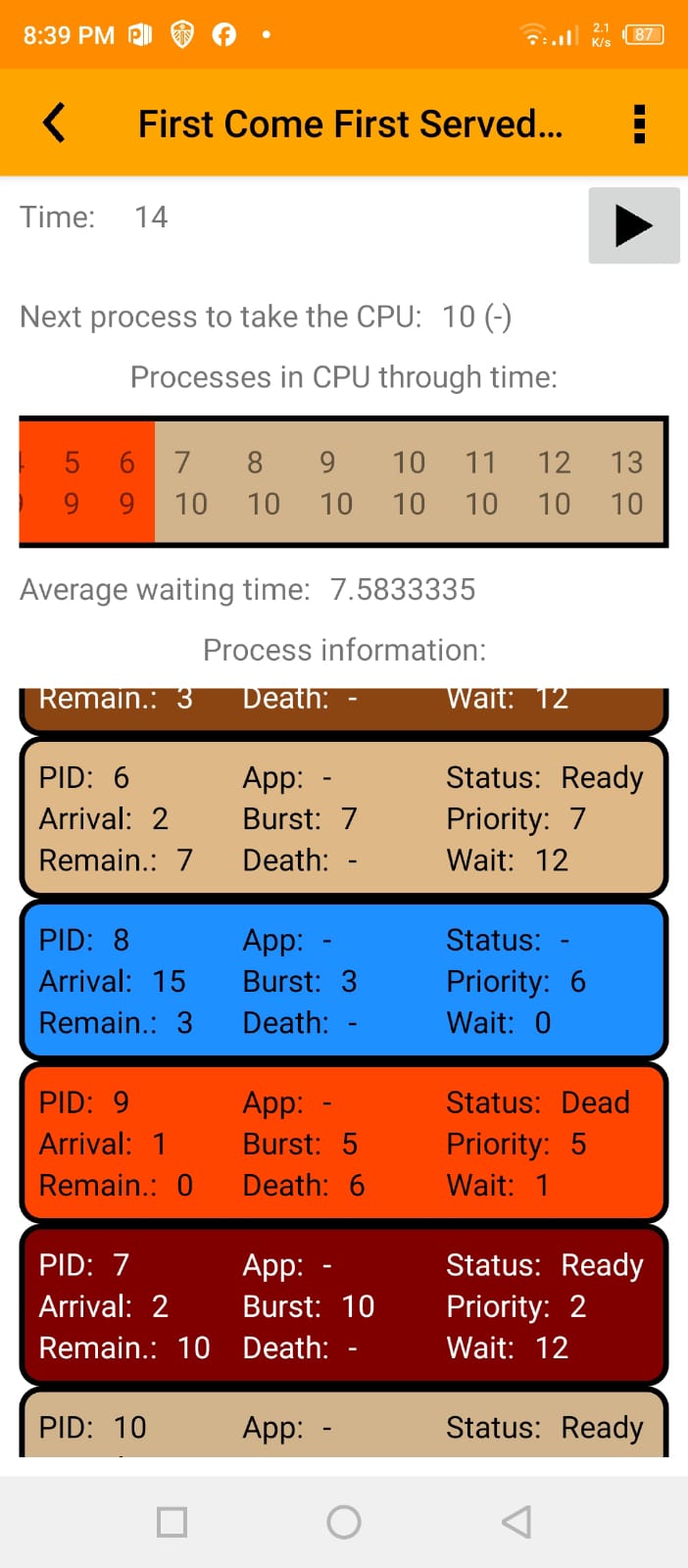
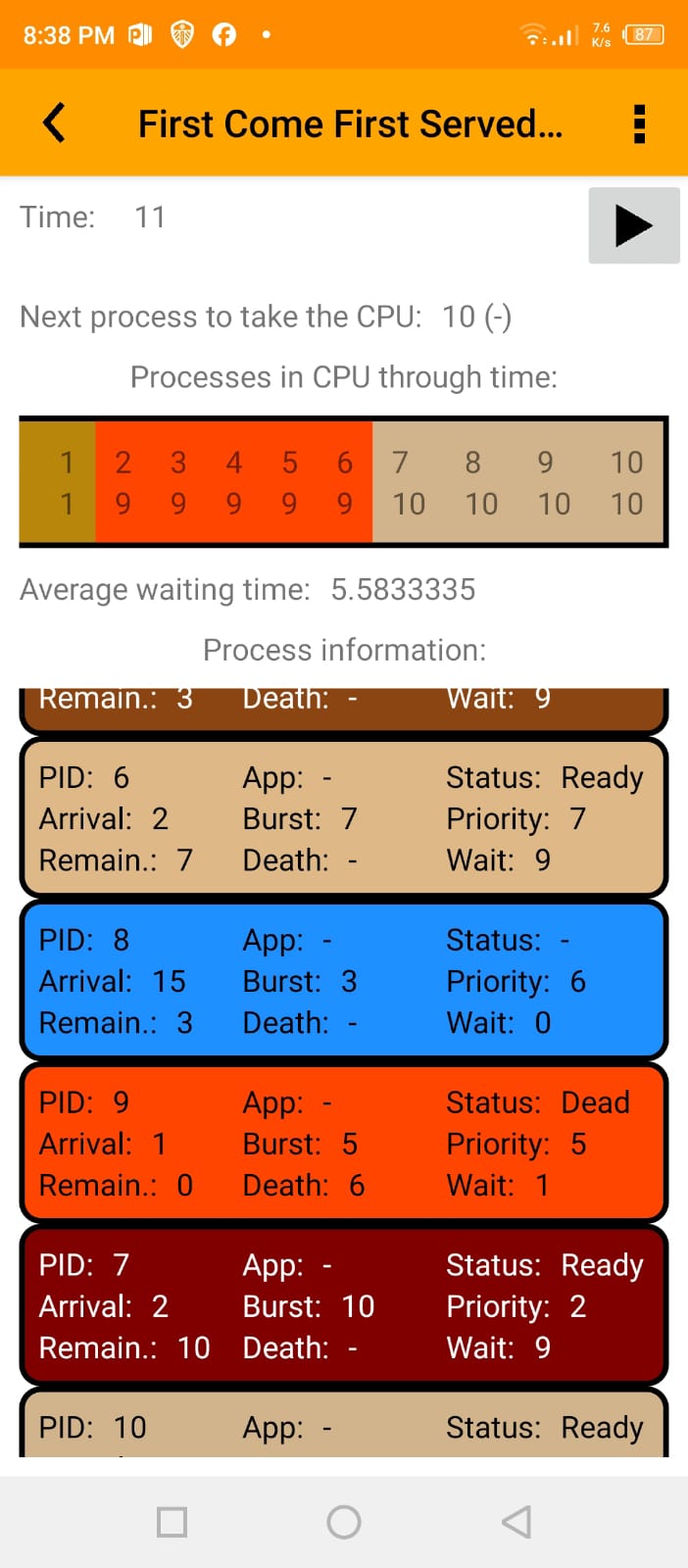
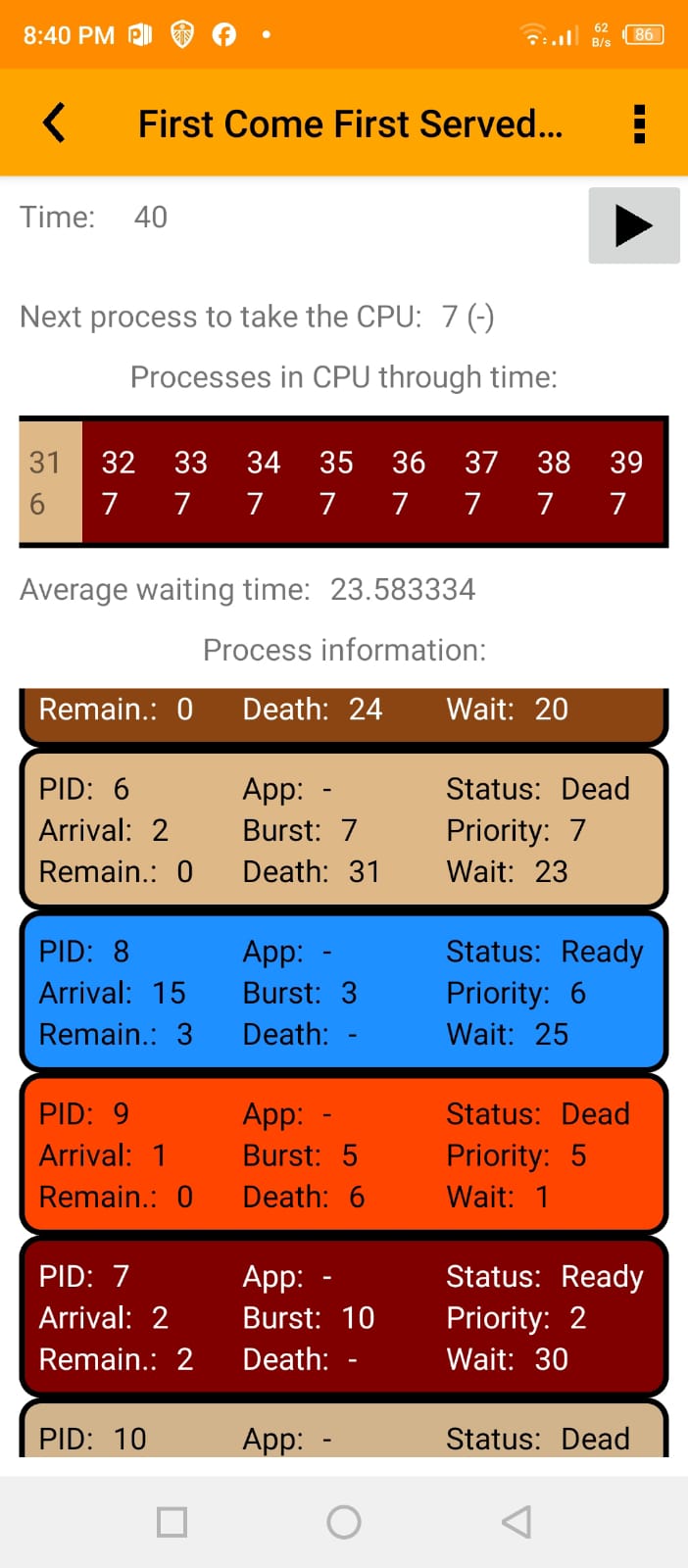
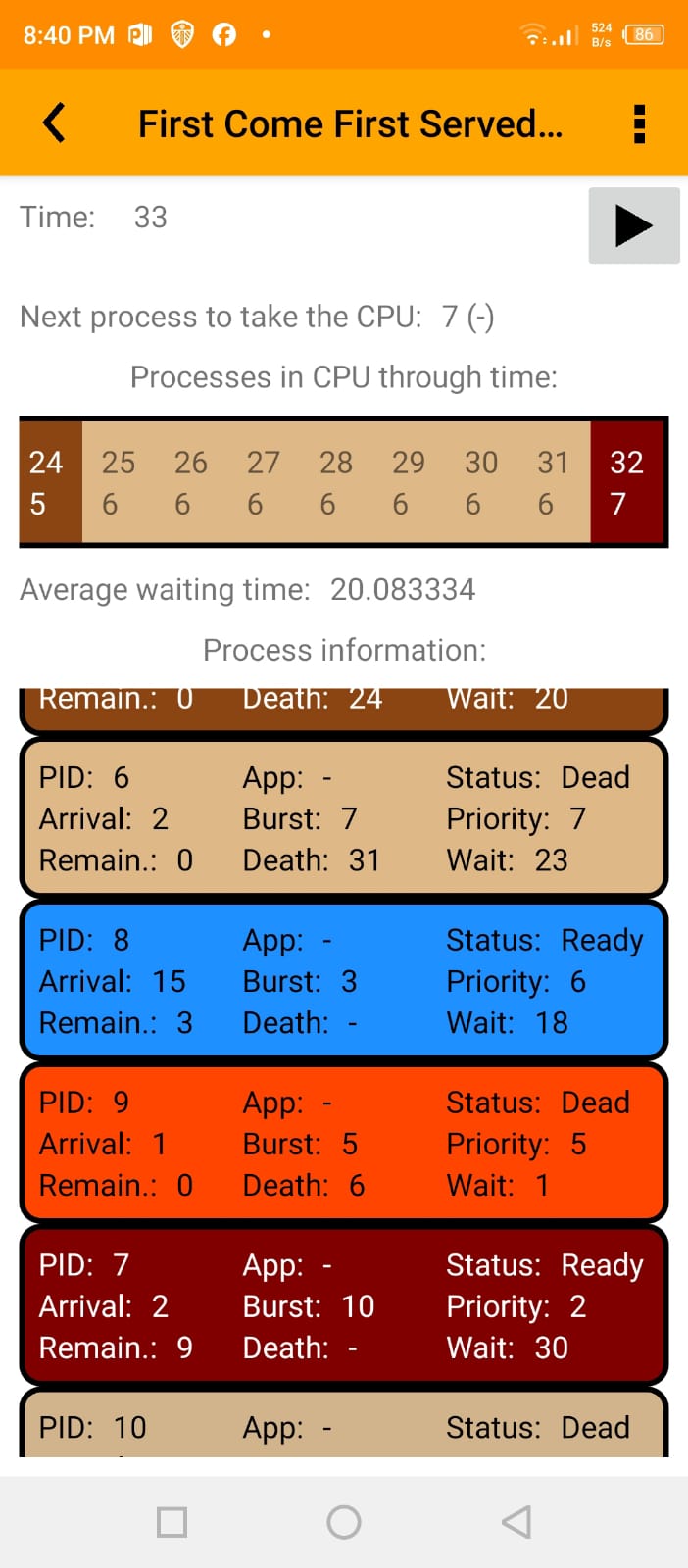
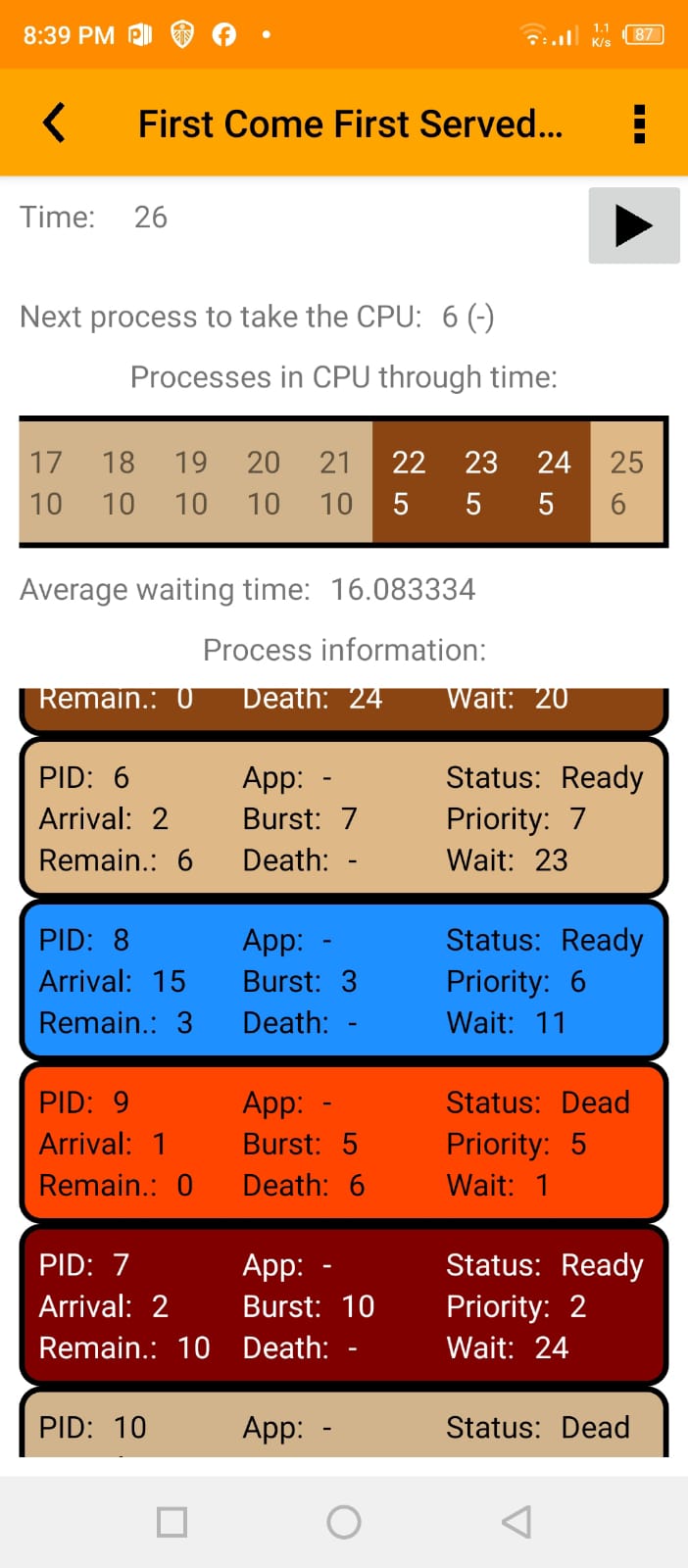
**The Simulation:**

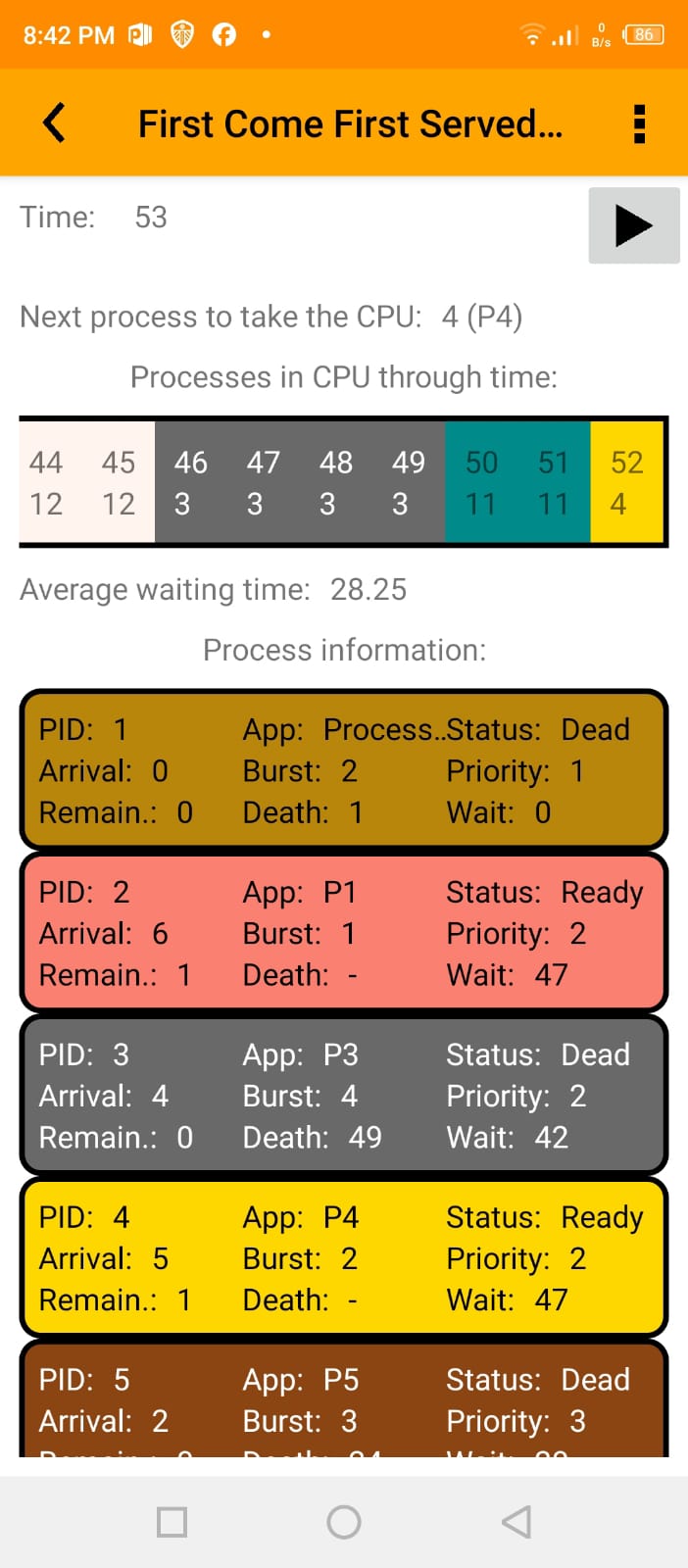
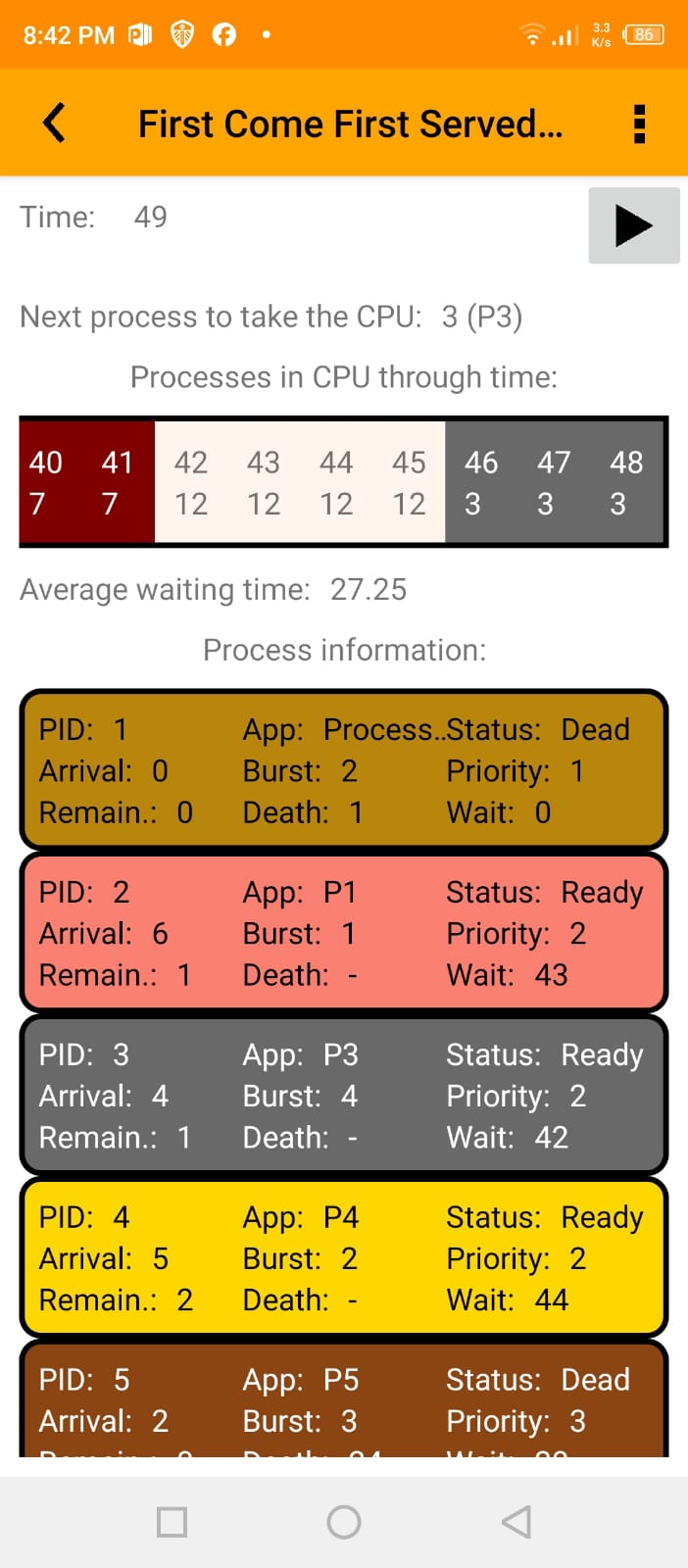
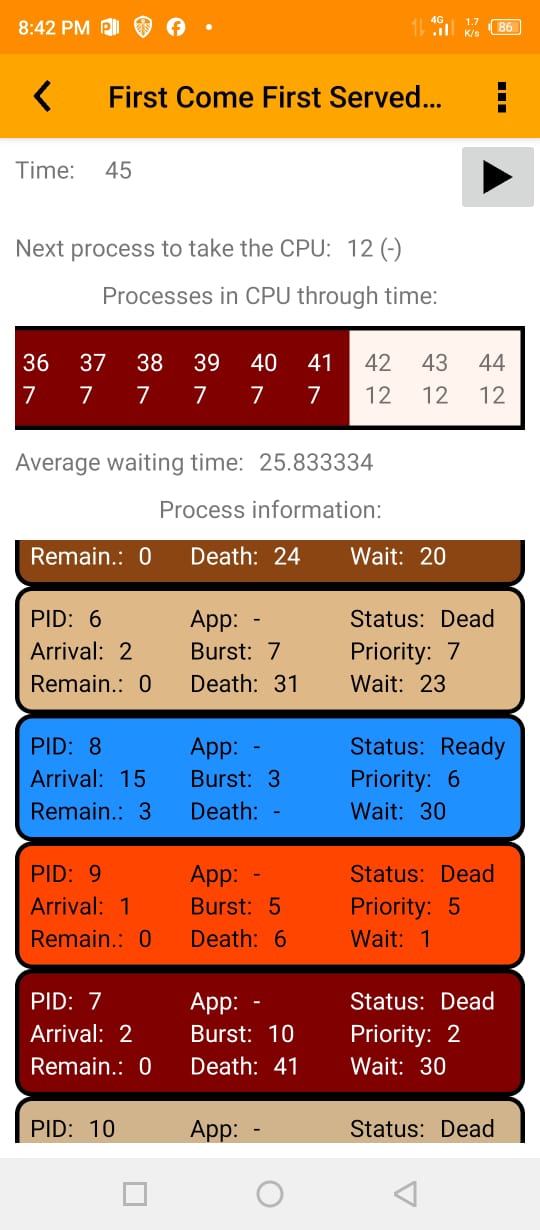
I chose the FCFS scheduling algorithm and the corresponding dataset (that I created) and began scheduling!

Throughout the scheduling process, I made sure to document **every step** to see exactly what happened and analyze my results. The steps are shown in the corresponding screenshots.





1. **Discussion:**

The simulation was conducted using the **FCFS** (First Come, First Serve) Scheduling algorithm as required, where I created **12 processes** and scheduled them based on their arrival times. Each process had CPU burst limitations ranging from **5 to 20 units**. The arrival times varied between **0 and 25**, while their priorities ranged from 1 to 7 (in FCFS priority is not considered). Some parameters given only apply to other relevant algorithms like (SJF or SRTF).

**Simulation Explanation:**

Throughout the simulation, each process was scheduled strictly based on their arrival times, with no consideration for burst lengths or any priority levels. This is evident when scheduling long processes which take up most time in the CPU all the way until there were dead.

Since **FCFS** is non preemptive, once a process began execution, it ran all the way to its completion.

The context switch time added an extra delay between process transitions, adding to the overall waiting time. Add to this, the random distribution of arrival times and burst times, made some processes experience significant waiting before getting any CPU time.

During this project, one thing was noticed well. This was the drawback of **FCFS** which was the **convoy effect** we learned in class. Where short processes wait behind longer ones. This was definitely seen when conduction **process 7** which had a whopping burst of 10.

**Linking What We Learned in Class:**

Compared to what we learned in our OS class, the simulated behavior and output aligns well with what we learned. In lectures, FCFS was taught as simple but inefficient in optimizing the best average waiting time possible.

This was obviously evident when processes like **process 11** only had a small burst only was executed at the end.

This practical simulation reinforced the limitations of FCFS and highlighted the importance of selecting a scheduling algorithm based on workload parameters and system goals.

1. **Conclusion**

As a wrap up, we implemented a scheduling simulation using FCFS (First-Come, First-Serve) in order to manage 12 processes with varying arrival times and burst lengths.

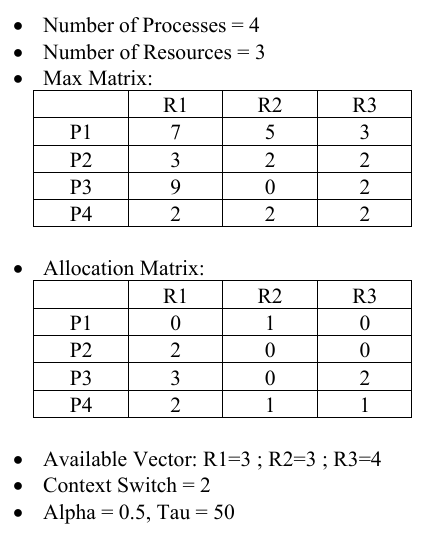
This simulation’s results highlighted the simplicity and disadvantages of FCFS, since each process was served based on the order of arrival. Particularly the **convoy effect** drawback.

In summary, this project enabled us to connect theoretical ideas with practical simulations, highlighting the advantages and disadvantages of **FCFS** and underlining the necessity of selecting the appropriate scheduling approach according to the objectives of the system and the nature of the workload.

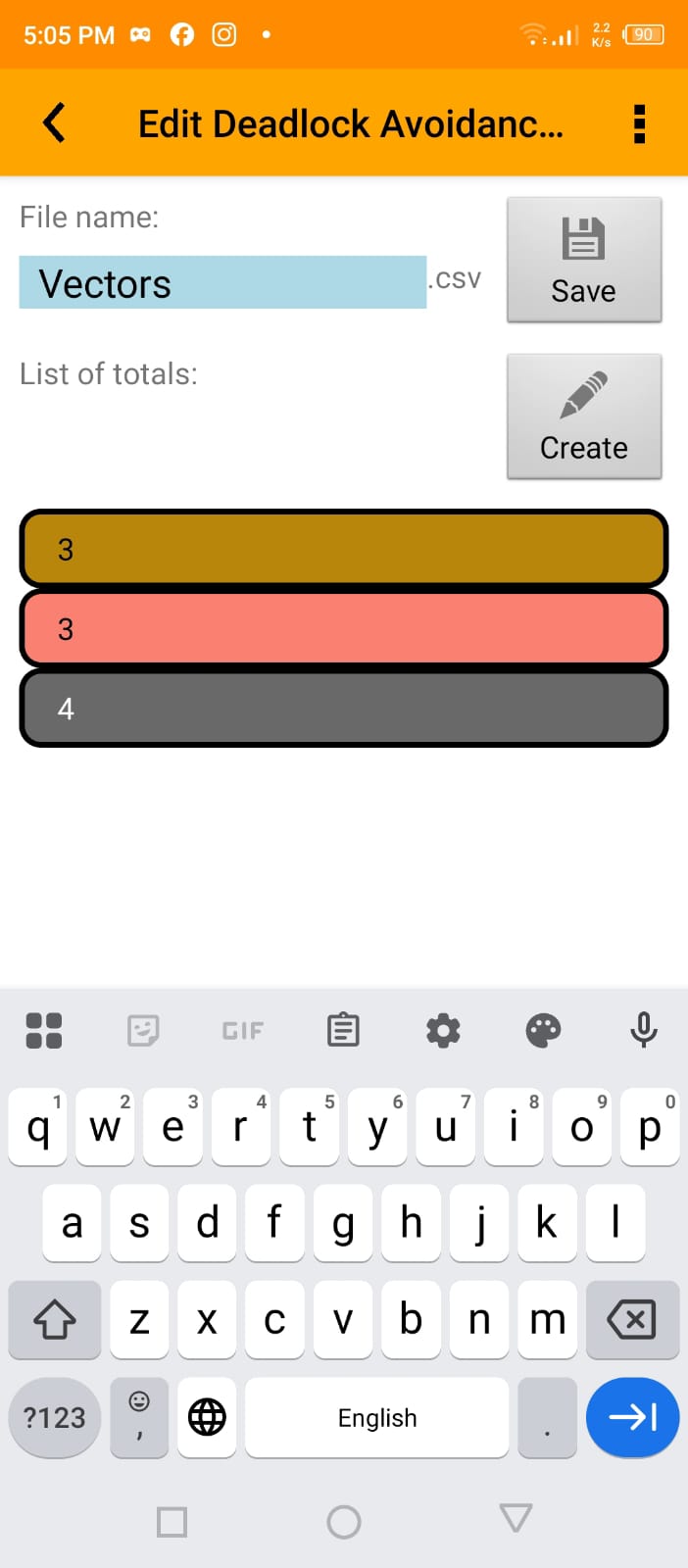
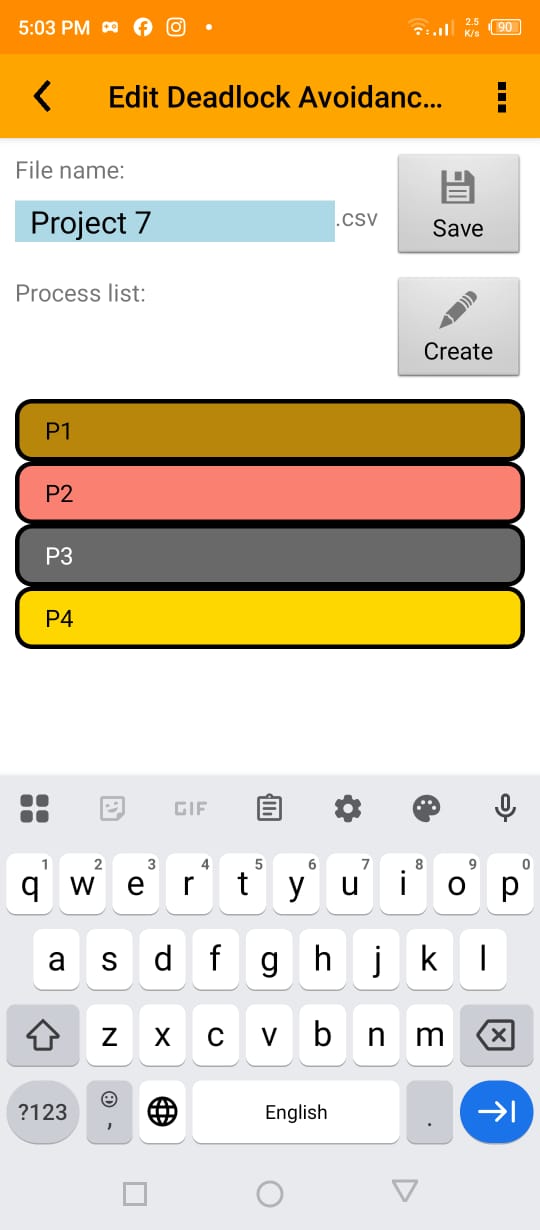
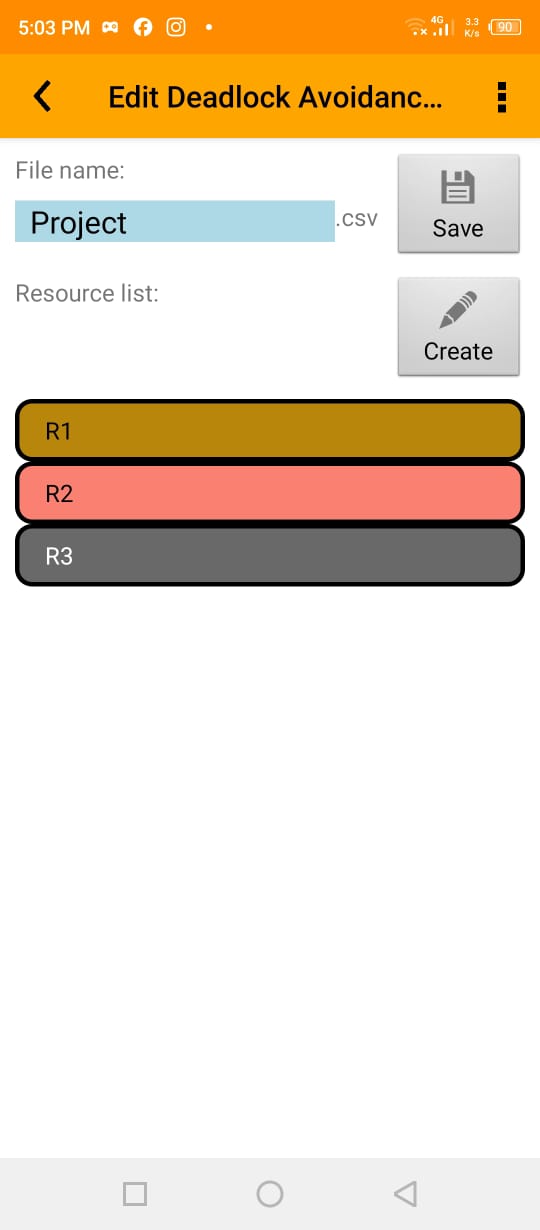
**Project Number 7:**

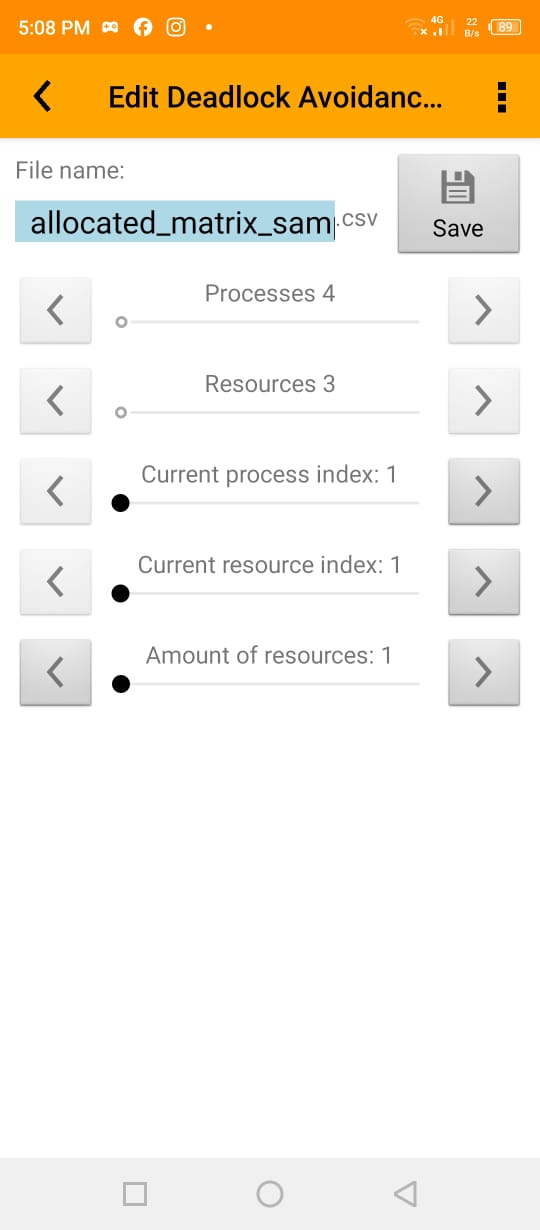
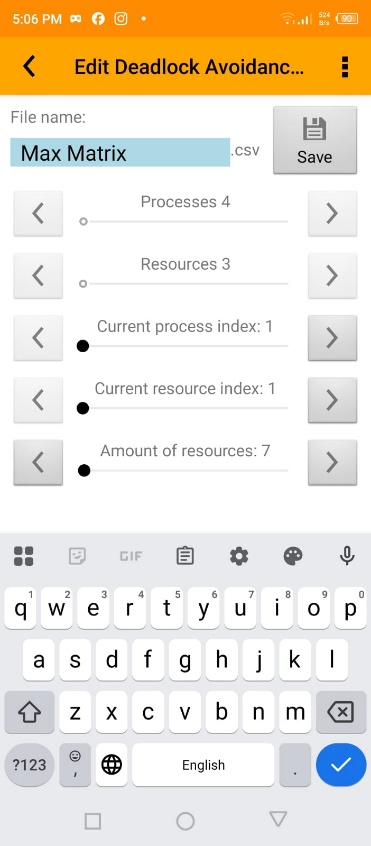
In the second project, the **Banker’s Algorithm** comes into play. It’s a recourse allocation deadlock avoidance algorithm. It makes sure to avoid the possibility of any deadlock.

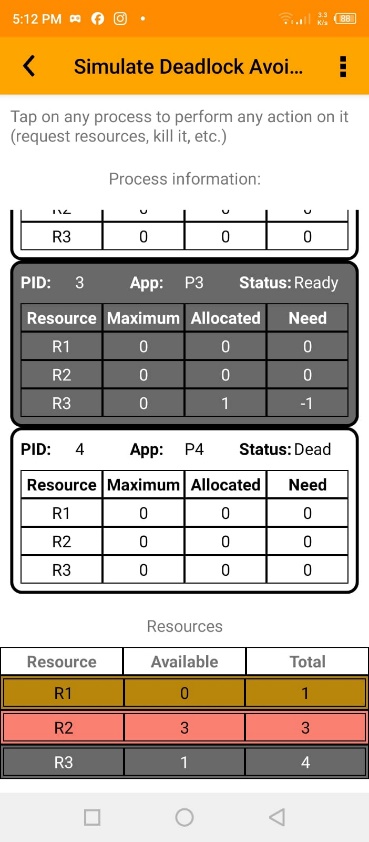
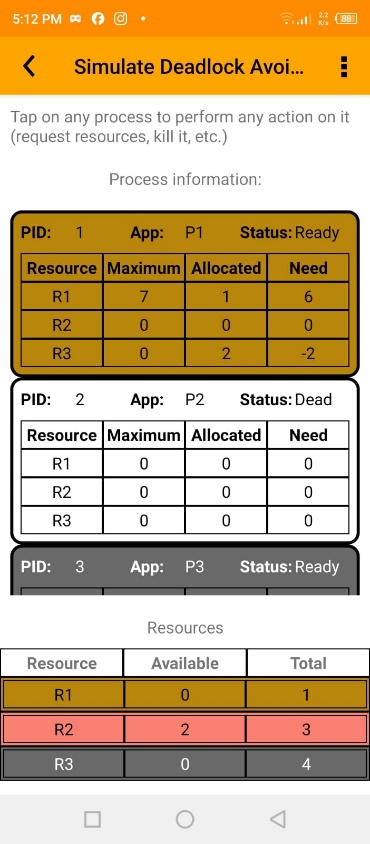
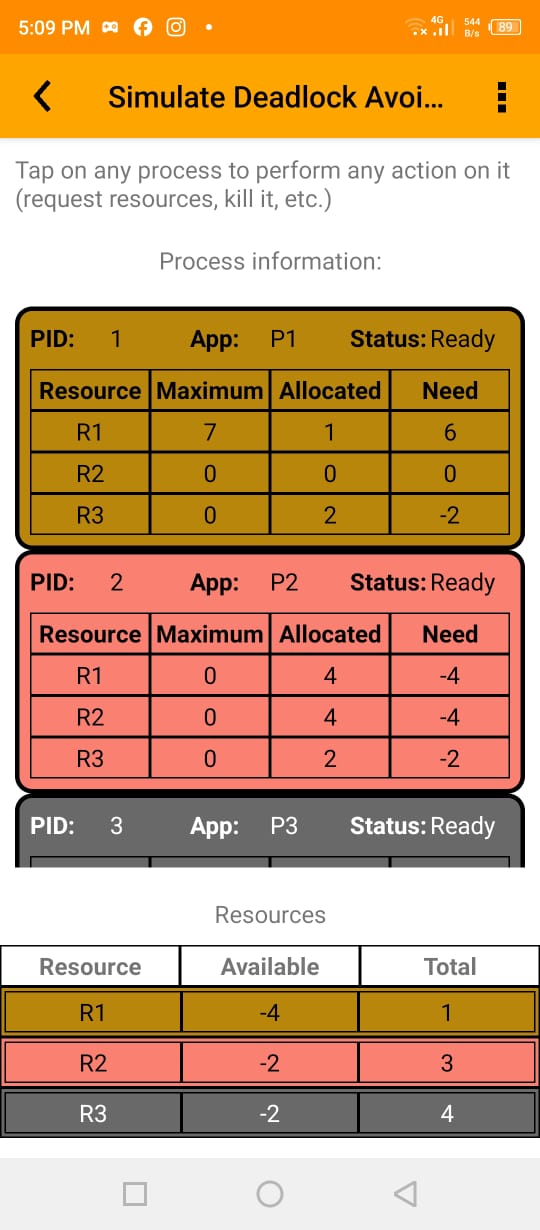
The project focuses on simulating **Banker’s Algorithm** with processes and recourses. My goal is to observe and verify system safety at each step, such steps are listed below, with the corresponding given:



Filling the info based on the given:







**Simulation Explanation:**

Whilst undergoing the simulation, we actively tested how the system responded to changes particularly under scenarios involving process terminations.

When a process was terminated manually, the simulator reclaimed their allocated resources accordingly.

I observed that the system kept checking for safe states after every request, to make sure no deadlock happens.

Also, every time I terminated a recourse, recourses that were held were freed.

In cases where multiple processes were competing for limited resources, the algorithm prioritized maintaining a safe execution sequence rather than granting immediate access.

The biggest insight from this simulation was how terminating just a single blocked process could shift the **entire system** into a safe state. To ensure execution continues with no deadlocks.

Seeing **Banker’s Algorithm** in simulation made me deepen my understanding of the importance of deadlock prevention and managing complex resource allocation scenarios.

**Conclusion:**

In conclusion, the project gave us a practical dive in to **Banker’s Algorithm** and its crucial role for deadlock avoidance. We were able to observe (on the app) how the system makes decisions to ensure a safe state.

Scenarios such as terminating processes, made us gain a deeper understanding of how recourse management really works. Not only did this reinforce the theoretical principles covered in class but also gave real world implementation.

Overall, this was a valuable experience to applying fundamental concepts of operating systems in real life.