**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**

**A Report on**

**“Thread Programming”**

**[Code No.: COMP 307]**

**(CE-III/I)**

**Submitted by**

**Puspa Hari Ghimire (21)**

**Group members**

**Suyog Dhungana (18)**

**Swochhanda Jangam (25)**

**Submitted to**

**Mr. Santosh Khanal**

**Assistant Professor**

**Department of Computer Science and Engineering**

**Submission Date:**

**June 11, 2024**

**Simulation of banking model using the concept of thread programming in python.**

**Source code:** [**https://github.com/puspah-ghimire/Thread-programming**](https://github.com/puspah-ghimire/Thread-programming)

**Code Description:**

This banking model simulation emulates customer service processes in a bank with three tellers, implementing various CPU scheduling algorithms—FCFS, Non-preemptive SJF, Preemptive SJF, and Round Robin to manage customer queues. Customers arrive at random intervals, each assigned a random service time, and are added to either a regular queue or a priority queue based on the selected algorithm.

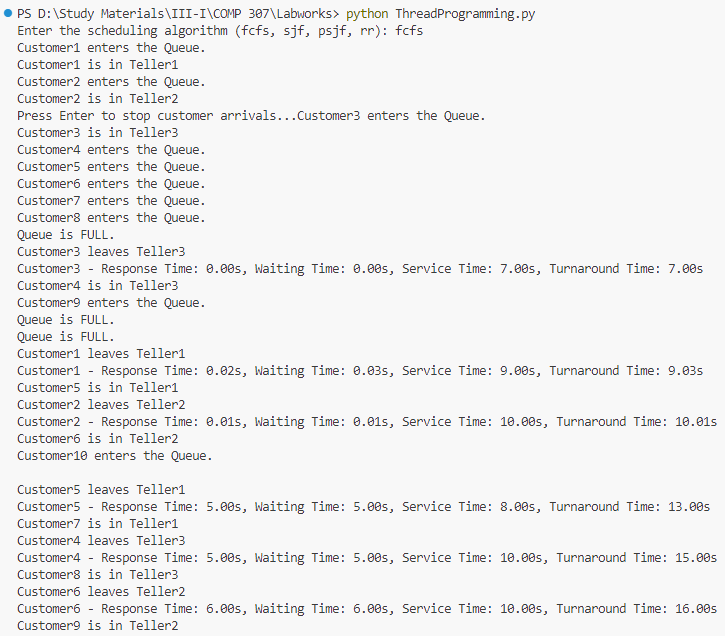
Tellers process customers according to the specified scheduling method, recording service and completion times, and calculating key performance metrics such as response time, waiting time, and turnaround time. The simulation tracks and computes total and average statistics for these metrics. User interaction allows for stopping customer arrivals, and the simulation provides visualizations of teller service times and queue sizes over time.

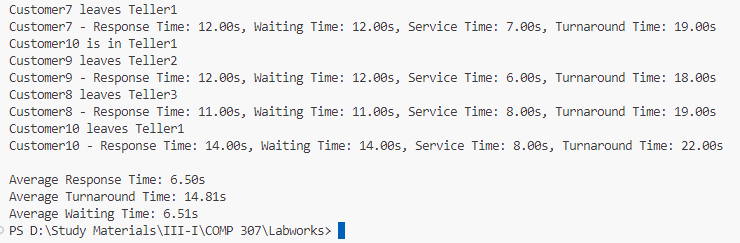
It offers insights into the performance and behavior of different scheduling algorithms in a multi-teller banking environment.

**First-Come, First-Served (FCFS):**

FCFS is the simplest scheduling algorithm where customers are served in the order they arrive. The teller processes the first customer in the queue until their service is complete before moving on to the next customer. This simple, non-preemptive approach can lead to longer waiting times, especially if a lengthy service precedes shorter ones, causing a "convoy effect" where shorter tasks wait behind longer ones.

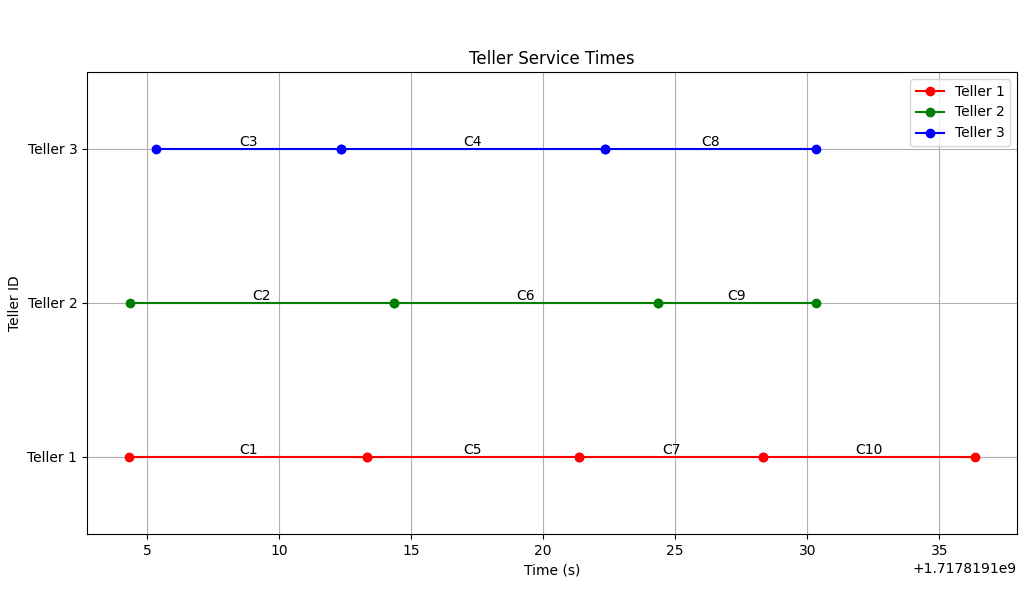
**Output for FCFS:**

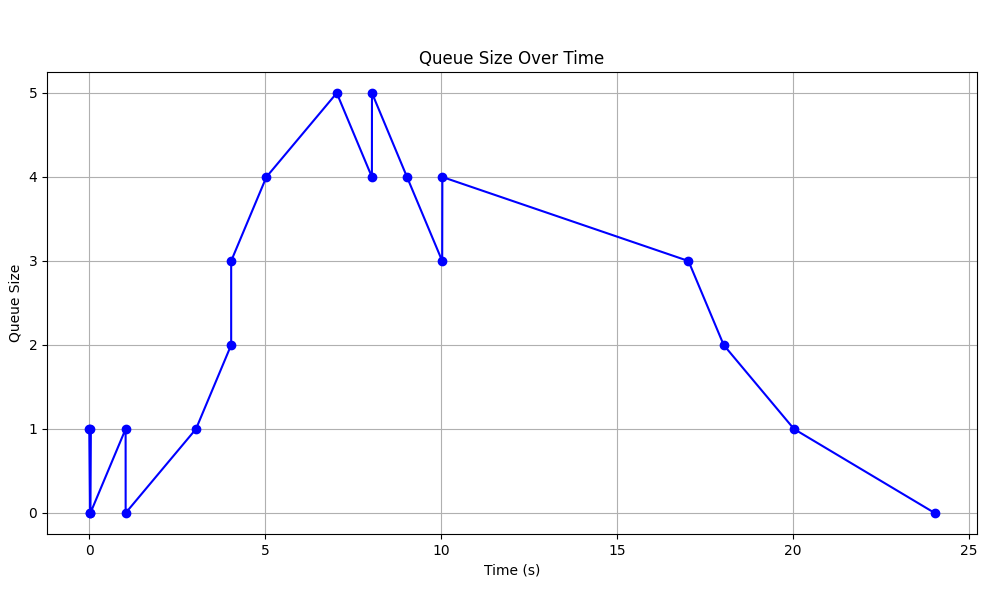




The average response time and average waiting time are the same for FCFS scheduling.

**Graph for FCFS:**



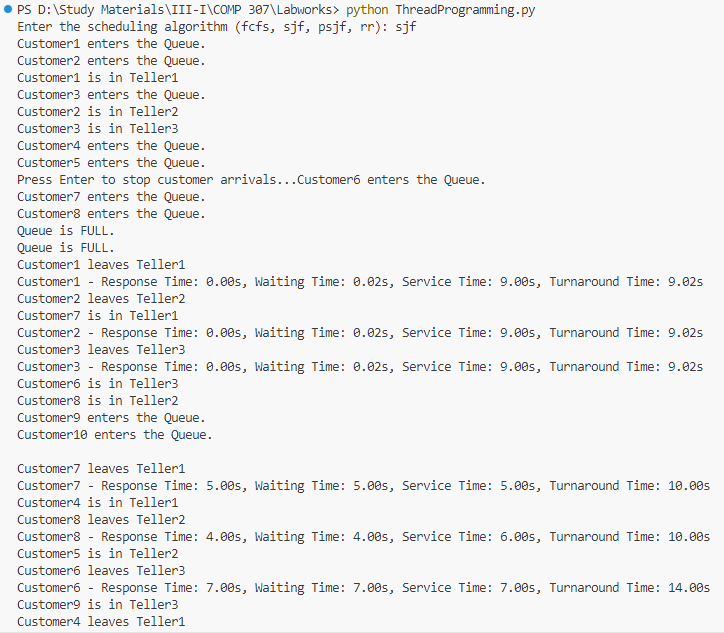


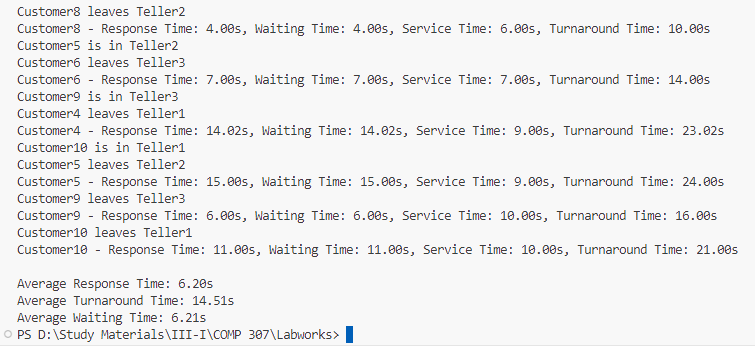
The first graph shows the service times of each customer in their respective teller. C1, C2, C3… are customers and they arrive according to their id in the regular queue. The customers are served sequentially according to their arrival time. The second graph represents the size of the regular queue with respect to time.

**Non-preemptive Shortest Job First (SJF):**

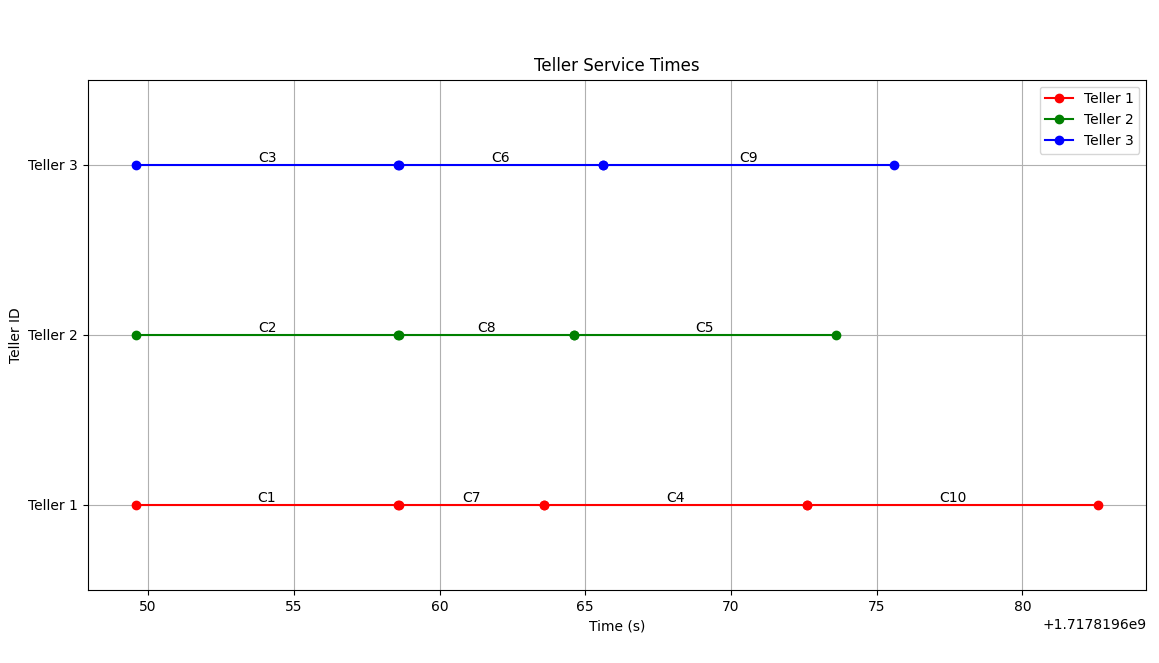
Here, customers are prioritized based on their service time, with the shortest jobs being served first. Customers are added to a priority queue sorted by service time. Since it is a non-preemptive method, once the customer starts being served, it leaves the teller only after the completion of its service time. This approach minimizes average waiting time but can cause longer jobs to starve if shorter ones keep arriving.

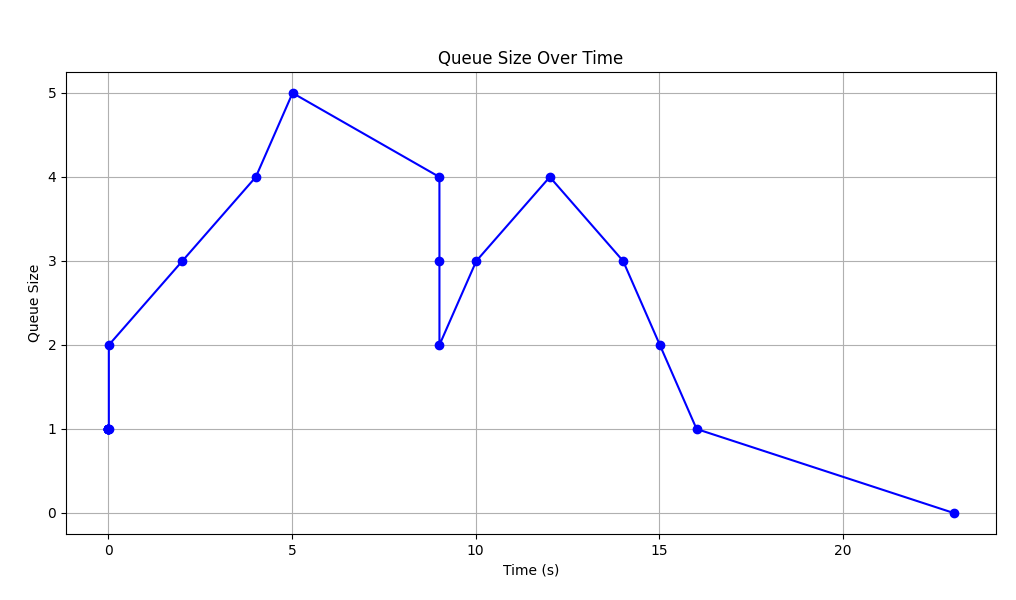
**Output for SJF:**





**Graph for SJF:**



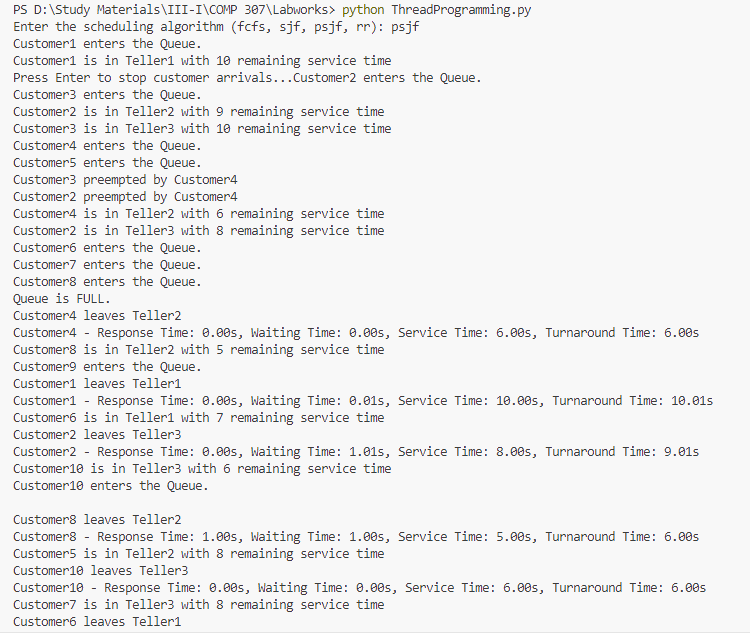


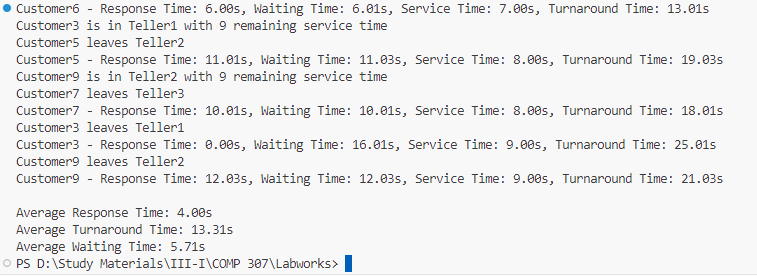
The first graph shows the service times of each customer in their respective teller. C1, C2, C3… are customers arriving in the priority queue where customer having shortest service time is placed at the front of the queue. We observe that although C7 is arrived later than C4, it is being served first due to shorter service time. The second graph represents the size of the priority queue with respect to time.

**Preemptive Shortest Job First (PSJF) (aka Shortest Remaining Time First):**

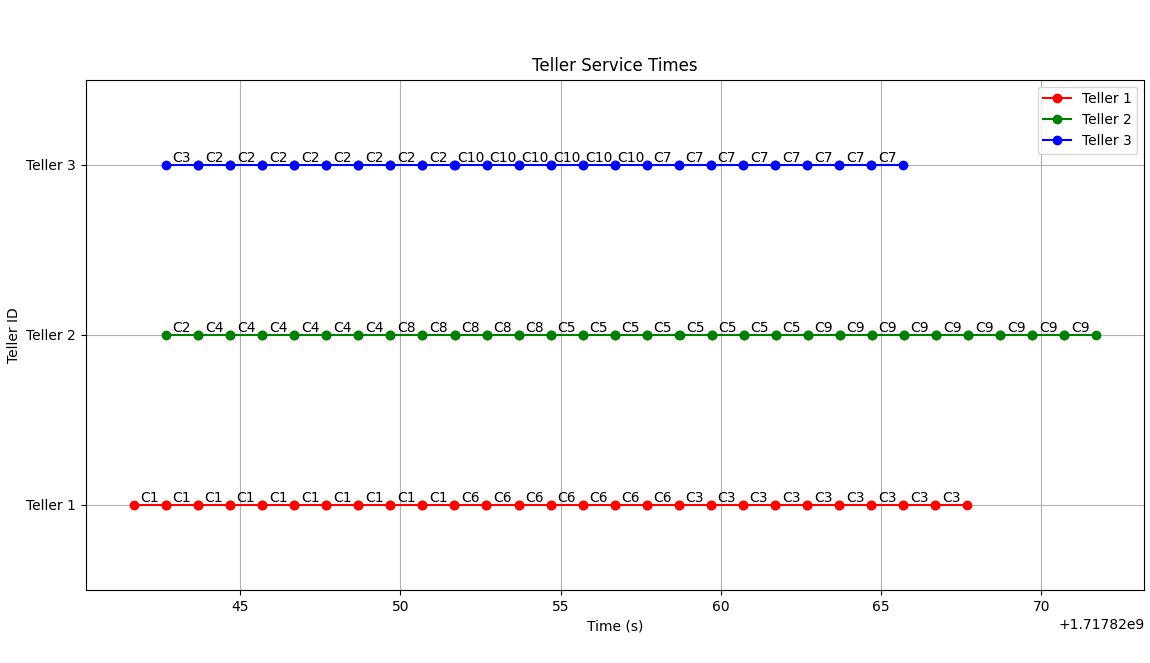
This algorithm is a preemptive version of SJF. If a new customer arrives with a shorter remaining service time than the current customer being served, the current service is interrupted, and the new customer is served. This ensures the shortest jobs are always prioritized, further minimizing waiting and turnaround times. However, it introduces complexity due to frequent context switching.

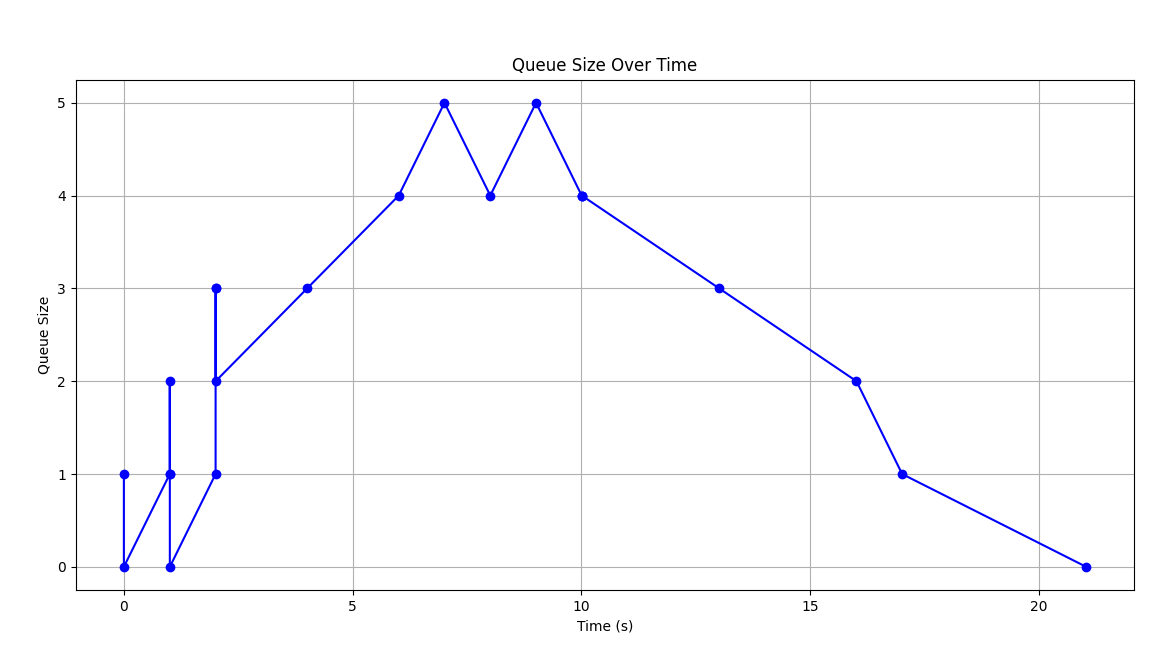
**Output for PSJF:**





**Graph for PSJF:**





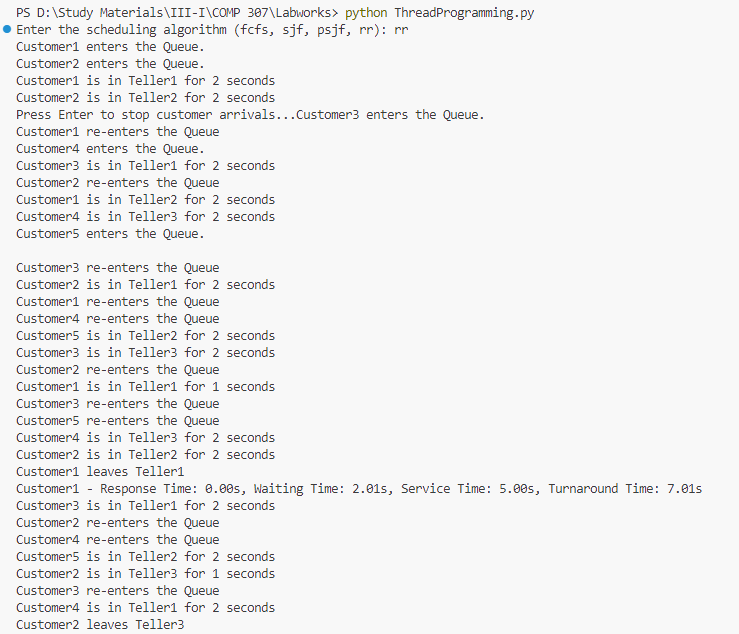
The first graph shows the service times of each customer in their respective teller. C1, C2, C3… are customers and they arrive according to their id in the priority queue where customer having shortest remaining service time is placed at the front of the queue. The algorithm checks for the shortest remaining time in the queue in every 1 second. In the graph, we see that C2 with 7s of remaining service time is preempted by C4 having 6s of service time.

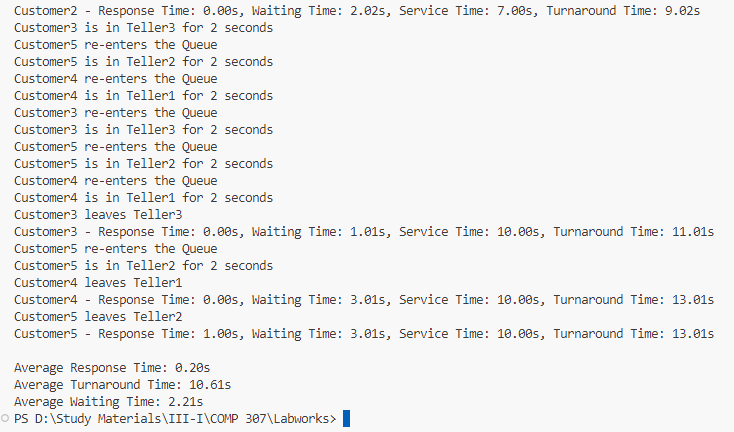
The second graph represents the size of the priority queue with respect to time. The sudden spikes represent the change in the size of queue instantaneously.

**Round Robin (RR):**

Round Robin scheduling assigns a fixed time quantum (here 2 seconds) for each customer in the queue. Customers are served in cyclic order for a set time slice. If a customer's service isn't complete within the time quantum, they are placed back in the queue to wait for another turn. This method ensures fair and equitable service distribution, reducing response time but potentially increasing context switching overhead if the time quantum is too small.

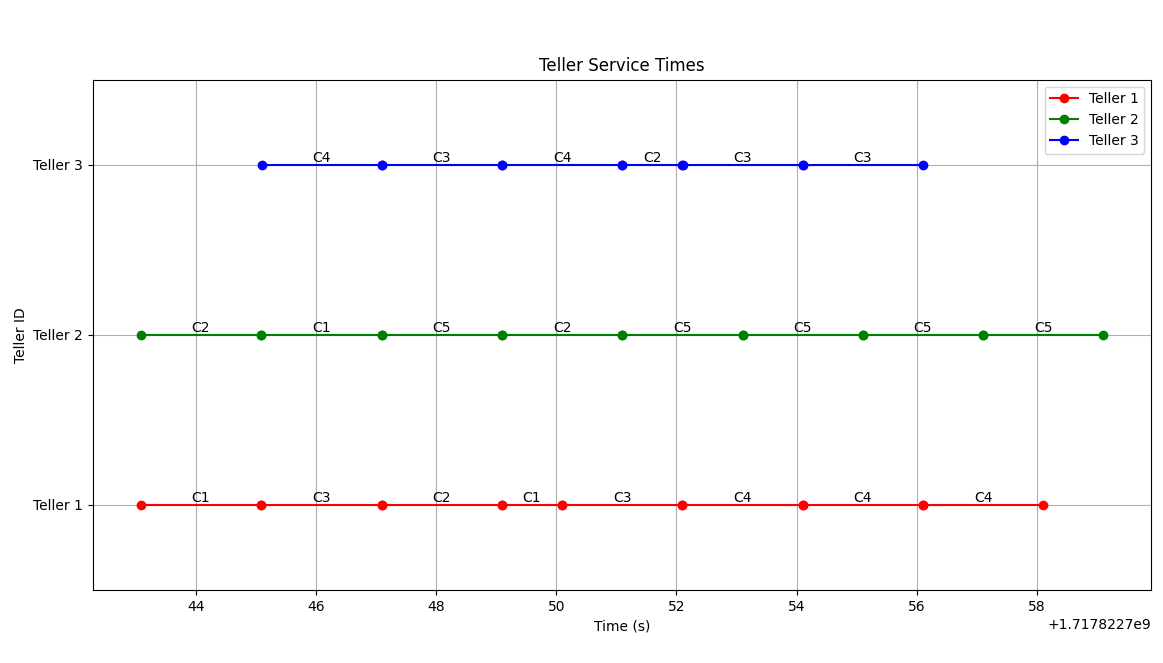
**Output for RR:**

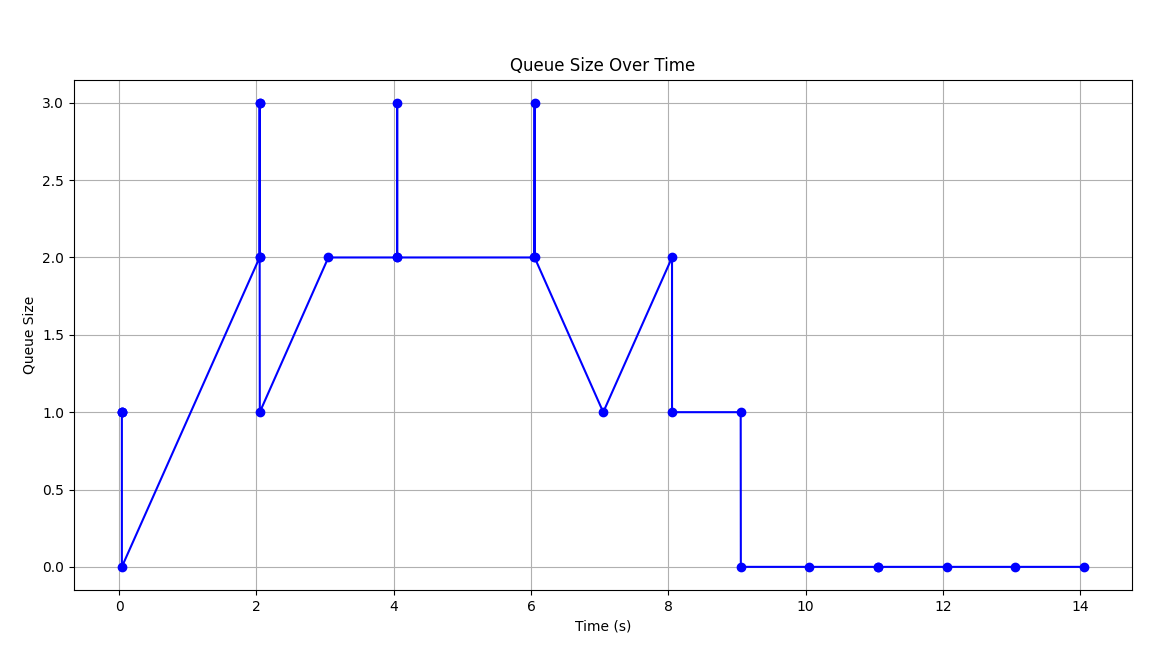




It has the lowest average response time of all algorithms.

**Graph for RR:**





The first graph shows the service times of each customer in their respective teller. C1, C2, C3… are customers and they arrive according to their id in the regular queue. We clearly observe that after every 2 seconds, the customers are placed back into the queue and served in a cyclic manner.

The second graph represents the size of the regular queue with respect to time. The sudden spikes represent the change in the size of queue instantaneously.

**Comparison Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | FCFS | Non-preemptive SJF | Preemptive SJF | RR |
| Average Response Time | Generally high, can be high for longer jobs | Typically, lower than FCFS, depends on job length distribution | Lower than non-preemptive SJF for varying job lengths | Can be low, depends on time quantum |
| Average Waiting Time | Can be high, especially if long jobs arrive first | Lower than FCFS, better with shorter jobs first | Typically, the lowest, as jobs are processed more fairly | Can be higher than SJF, but better with smaller time quantum |
| Average Turnaround Time | High for longer jobs arriving first | Lower than FCFS, better with shorter jobs | Generally lower, due to preemption | Depends on time quantum, balances between waiting and turnaround |

**Conclusion:**

Each algorithm has its advantages and disadvantages, and the choice of an appropriate scheduling algorithm should be guided by the specific requirements of the application, such as the need for fairness, efficiency, response time, and the ability to handle varied job lengths effectively.