CMPE 663 Project 3

Bank Teller Simulation

Author: Gloria Mbaka

Email: gm6629@rit.edu

Submitted: 10/28/24

Instructor: Prof. Wolfe

TA: Bertola Thomas

Analysis/Design

Project Overview

This project involved designing a bank teller simulation using FreeRTOS and an STM32L476RG microcontroller. The system simulates the operation of a bank with three tellers serving customers who arrive randomly throughout the day. The primary objective was to handle customers efficiently while managing teller breaks (both scheduled and forced via buttons) and collecting various metrics throughout the simulation. The system operates on a simulated time scale where 1 simulated minute equals 100 real-time milliseconds. The bank is open from 9:00 am to 4:00 pm, and all customers must be served even after the bank closes. Each teller operates based on a task scheduled by FreeRTOS. Breaks are managed either through a predefined schedule or by forcing a teller on break using buttons. The simulation tracks various performance metrics, including customer wait times, teller service times, break statistics, and queue depth.

Hardware Design

STM32 Nucleo Board

The STM32 Nucleo L476RG board was used to implement this project. The following peripherals were configured:

GPIO Pins:

- PA1 (Button for Teller 1), PA4 (Button for Teller 2), PB0 (Button for Teller 3): These pins trigger forced teller breaks.
- PA5 (LED D1), PA6 (LED D2), PA7 (LED D3): Indicate the status of each teller (on break or idle).
- The seven-segment display was configured on the PA8 (Clock), PA9 (Data) and PB5 (Latch) for displaying the number of customer on queue.

USART2 (UART): Handles communication with the host computer via Putty for displaying real-time metrics and end-of-day reports.

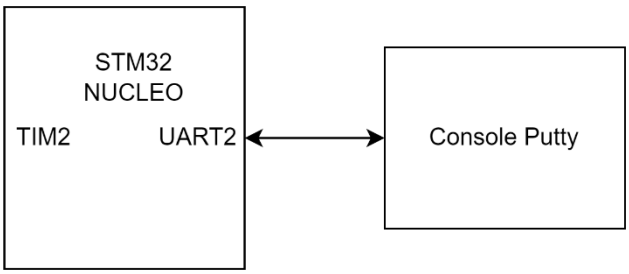

Figure 1: Hardware Block Diagram

Software Design Approach

The system includes the following key components:

1. **Customer Queue:** Simulated customers arrive at random intervals and are placed in a queue by bank manager. The queue depth, average wait times, and the number of customers in the queue are tracked and displayed.
2. **Teller Tasks:** Each teller is a FreeRTOS task that waits for customers in the queue. If there are no customers, the teller waits until a customer arrives. Tellers can go on scheduled or forced breaks, during which they stop processing customers.
3. **Break Management:** Each teller takes scheduled breaks at random intervals but can also be forced on break using a button. Buttons are debounced, and while a button is held down, the associated teller remains on break.
4. **Metrics Collection:** The system tracks various metrics throughout the simulation, including customer wait times, teller service times, break times, and queue depth. These metrics are printed both in real-time and as part of an end-of-day report.

Flow Diagram

The bank simulator flow chart as shown in Figure 2 begins with the initialization and setup of the necessary components, which includes creating threads for customers. This initial step ensures that the system is fully prepared for operation. Once the setup is complete, the first critical check involves confirming whether the bank is open. This check is essential because it regulates customer arrivals, allowing them to enter the bank only during designated operating hours. As customers arrive at the bank, they are added to a queue for service. Concurrently, the system assesses the status of the tellers to determine their availability.

In instances where no tellers are free, customers will wait patiently in the queue until a teller is available to assist them. When a teller becomes available, the next customer in line is assigned to them. This process involves not only the assignment but also tracking the time spent on the transaction. Once the transaction is successfully completed, the system updates various metrics to reflect the wait times and the duration of the service provided. After a customer leaves, the simulator checks to see if any tellers are on break.

If a teller is indeed on break, the system will notify the start of the break and update the teller's status accordingly. Before concluding the simulation, a final check is performed to determine if the bank remains open. If there are still customers waiting in the queue, the system loops back to reassess teller availability, ensuring that all customers are served before the simulation ends. The flow diagram is in Figure 2.

Software Flow:

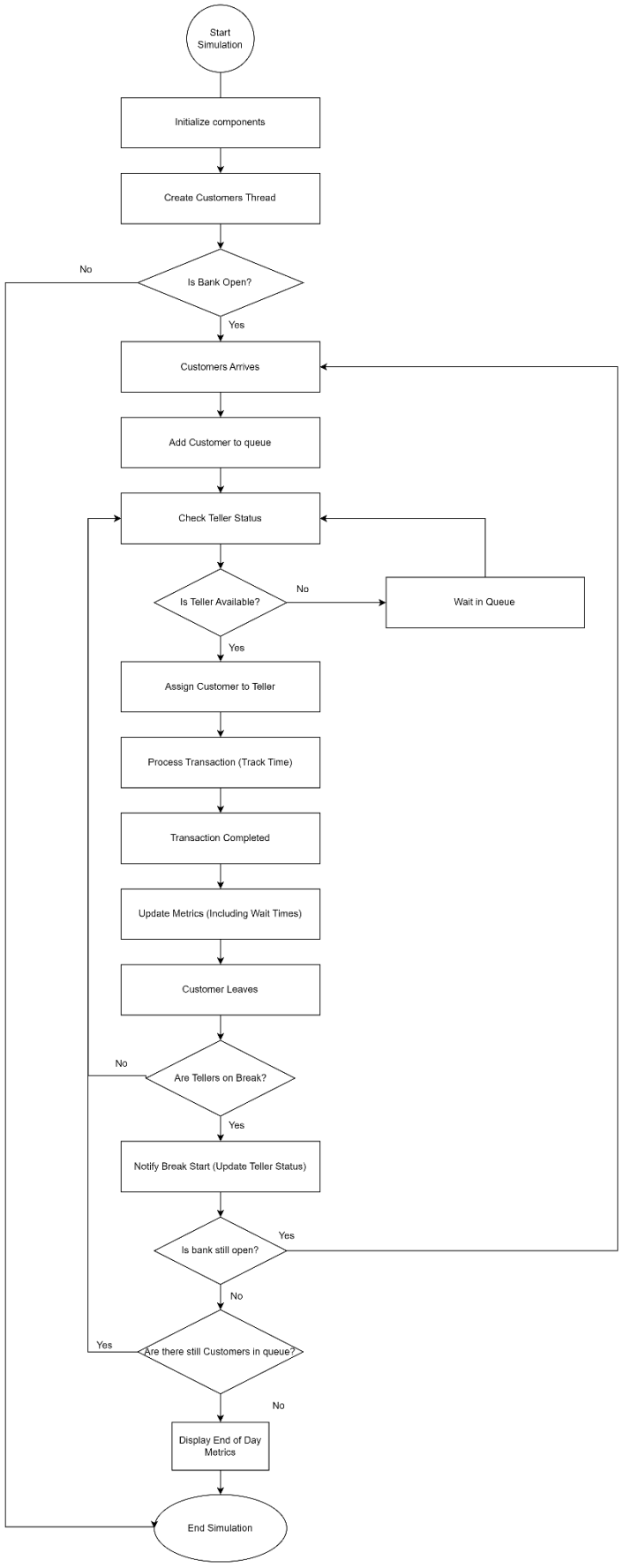

Figure 2: Software Design Flow Chart

Test Plan

- **Customer Arrival and Queue Processing:**
  Customers were simulated to arrive randomly between 1-4 simulated minutes. It was ensured that customers were processed in a first-come, first-served manner and that all customers were served even if the bank had closed.
- **Forced Break Testing:**
  Each teller was tested to ensure they could be forced into a break using the designated buttons. While on forced break, the teller did not process customers, and the break was held as long as the button was pressed.
- **Scheduled Break Testing:**
  The tellers were programmed to take breaks after random intervals, during which they would not serve customers.
- **Metrics Validation:**
  Various metrics were calculated and printed in real-time, including the number of customers served, average wait time, and queue depth. The final end-of-day report summarized all relevant statistics.

Results

The following were observed during testing:

- **Customer Queue and Processing:** The system correctly handled customer arrivals and queue processing. All customers were served even after the bank closed, ensuring the queue was empty by the end of the day.
- **Break Management:** The system correctly handled both scheduled and forced breaks. The forced breaks remained active while the buttons were pressed and resumed work once released. There was a minor challenge where forced breaks did not hold initially, which was resolved by checking button states continuously during breaks.
- **Metrics Collection:** The metrics were accurately tracked and reported at the end of the day. The system captured and displayed key statistics, including the total number of customers served, average wait times, and break durations.

The result from the putty during testing is as shown in Figure 3.

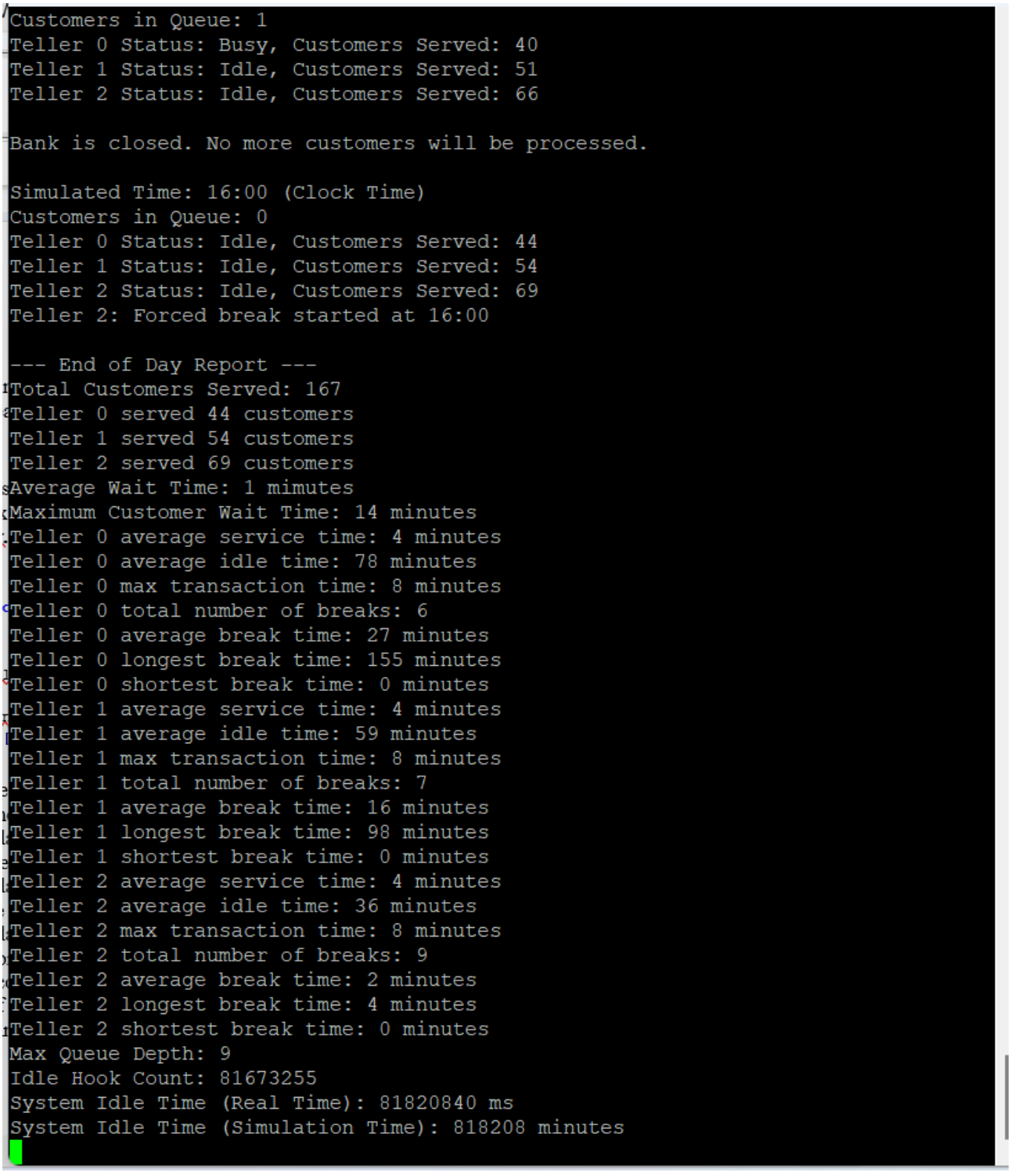

Figure 3: Screenshot of the End of the Day Metrics

Lessons Learnt

1. Using counting semaphores to manage teller availability was essential for tracking idle times and coordinating shared resources. This experience highlighted how synchronization is crucial in real-time systems to ensure smooth operation and accurate tracking.

2. Managing both scheduled and forced breaks was complex, requiring clear logic to handle priorities. By consolidating break logic as a task or thread, it helped to manage breaks.

3. Working with CMSIS (Cortex Microcontroller Software Interface Standard) functions highlighted the differences compared to FreeRTOS basic functions. This experience improved the understanding of the CMSIS abstraction layer.

Challenges Encountered

One major challenge was managing teller breaks. Initially, forced breaks triggered by button presses did not hold while the button was pressed down. This issue was resolved by continuously checking the button state in a loop, ensuring that the break remained active as long as the button was pressed. Additionally, accurately tracking and updating each teller's idle time presented some difficulty. The idle time for tellers initially returned zero, leading to incorrect metrics. To address this, a counting semaphore was introduced to track and synchronize teller availability effectively. The semaphore helped manage access to shared resources, ensuring that idle time was recorded only when a teller was genuinely free to serve a customer. This synchronization allowed for accurate and consistent idle time reporting in the final system.

Conclusion

This project successfully simulated a bank teller system using FreeRTOS on an STM32 microcontroller. Key objectives, including handling teller breaks, customer service, and tracking performance metrics, were achieved. The system provided real-time feedback via UART and successfully produced an end-of-day report summarizing key metrics. While there were some challenges in managing breaks and processing button inputs, these were overcome, and the final system met all functional requirements.