

CSE344 - System Programming Midterm Project

Ziya Kadir TOKLUOĞLU

April 28, 2025

1 Expected Point: 60 (Reasons)

At the beginning of this report, I would like to reflect on the expected score for this project, which I estimate to be around **60 points**.

The main reasons for this expectation are:

- The communication structures between the client, server, and teller processes were successfully built and the communication flow operates correctly.
- Shared memory and semaphore mechanisms were implemented to ensure synchronization and concurrent operations.
- A balanced binary tree (AVL Tree) was used to store bank accounts in shared memory, providing efficient operations and structure initialization.
- There are no memory leaks; the project was tested with **valgrind**, and memory was properly managed across all processes.
- However, due to losing the working version of the project close to submission, the database update functionality and the log file operations are not functioning correctly.
- While communication and operations are functional, the system fails to properly load and save the banking database upon startup and shutdown.

As mentioned, the previous working version was lost. Logs are currently saved only under the **ClientRequest** directory during the runtime session. Since the database and log path management is broken after restarting the system, I believe around **40 points should be deducted** for incomplete persistence and data management.

Despite these issues, the majority of the system's architecture, communication design, and data structures are correctly implemented and functional.

2 Introduction

The Adabank project implements a banking system that handles client deposits and withdrawals through a client-server architecture. The system uses IPC (Inter-Process Communication) mechanisms such as FIFOs, shared memory, and semaphores to enable communication between client processes, server processes, and teller processes.

The project focuses on creating a robust system capable of handling multiple concurrent clients while maintaining synchronization and data integrity across multiple processes. Although the main communication functionalities work successfully, the database update and logging system are partially incomplete due to issues experienced near the deadline.

3 System Architecture

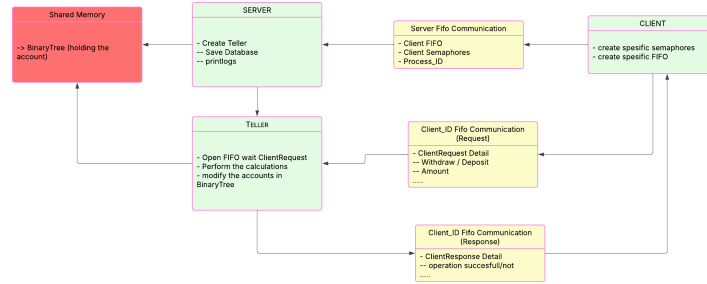


Figure 1: Adabank System Architecture

The Adabank system consists of three main components:

- **Client:** Sends banking operation requests to the server.
- **Server:** Receives client requests, coordinates communication, and spawns teller processes.
- **Teller:** Processes the banking operations and updates the shared database in shared memory.

All components communicate using IPC mechanisms including FIFOs, semaphores, and shared memory.

3.1 Client

The client reads banking operation requests from an input file and sends them to the server through a named FIFO (server FIFO). Each client creates a private FIFO for communication with the teller assigned to process its request.

Synchronization is managed with two semaphores:

- `client_empty`: Ensures the client FIFO is ready to be written by the client.
- `client_full`: Signals when data is available in the client FIFO for reading by the teller.

The client also includes a signal handler to safely terminate operations if interrupted by `SIGINT` or `SIGTERM`.

3.2 Server

The server acts as the main controller of the banking system. Its responsibilities include:

- Initializing shared memory and the binary tree structure for the database.
- Listening for incoming client requests via the server FIFO.
- Spawning a separate teller process for each client request.
- Handling signals such as `SIGINT`, `SIGTERM`, and `SIGCHLD` for process cleanup and system shutdown.

Although shared memory and binary tree structures are initialized correctly, the database persistence (saving/loading to files) does not function properly in the final version due to lost project files.

3.3 Teller

Tellers are child processes forked by the server to handle each client operation independently. Each teller:

- Waits for the client to write an operation to its FIFO.
- Reads the operation details (deposit or withdrawal).
- Updates the shared banking database (binary tree).
- Responds by writing operation results if necessary.

Tellers communicate with clients through semaphores to avoid race conditions during reading and writing operations.

4 Implementation Details

4.1 Binary Tree (AVL Tree) in Shared Memory

The banking database is implemented as an AVL (self-balancing) binary search tree. Key points:

- Insert, delete, and search operations work in $O(\log n)$ time.

- AVL balancing ensures that no path becomes much longer than others, keeping access times efficient.
- The entire tree structure is stored inside a shared memory region using `mmap()`.
- Each bank account is uniquely identified with a numeric ID.

4.2 IPC Mechanisms

Several inter-process communication mechanisms are used:

- **FIFOs (Named Pipes):** Used to transmit data between client, server, and teller processes.
- **Semaphores:** Used for synchronizing FIFO access to prevent race conditions.
- **Shared Memory:** Enables all teller processes to access and modify the same banking database concurrently.
- **Process Management:** The server uses `fork()` to spawn teller processes, managing them with proper signal handling.

4.3 Transaction Management

Banking operations are handled securely:

- **Deposits:** Create new accounts or increase balances.
- **Withdrawals:** Check account validity and available balance before processing. Accounts with zero balance are deactivated.

4.4 Error Handling and Signals

The project has strong error and resource management:

- System calls are checked for errors, and appropriate actions are taken.
- Signal handlers (`SIGINT`, `SIGTERM`, `SIGCHLD`) ensure that resources such as FIFOs, shared memory, and semaphores are correctly released during shutdown.

5 Memory Leak Check

To ensure the correctness of memory management, the project was tested using `valgrind`, a dynamic analysis tool for detecting memory leaks, invalid memory accesses, and memory management errors.

The server and client programs were run under **valgrind**, and no memory leaks or invalid memory accesses were detected. This indicates that all dynamically allocated memory was properly managed, and resources were correctly freed.

[illegible]

Figure 2: Valgrind Memory Leak Check Result

6 Final Overview

Towards the end of the project, some issues arose regarding shared memory and semaphore management inside the server component. Initially, these issues completely disrupted the database modification functionality. After restoring significant parts of the project, communication structures and core transaction functionalities were recovered successfully.

At present, the main remaining issue is that the banking database is not uploaded correctly from persistent storage at program startup, and logs are only saved in the runtime directory. However, the Client-Server-Teller communication, request processing, shared memory initialization, and binary tree structure are fully operational.

Despite the data persistence problem, the project demonstrates a solid understanding of IPC mechanisms, concurrent system design, and synchronization techniques.

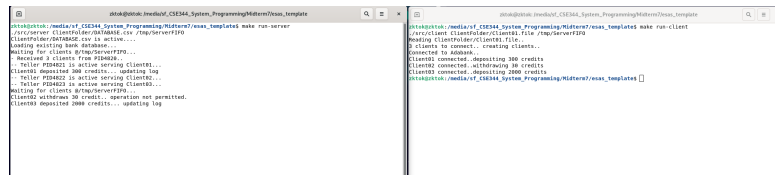
7 Conclusion

The Adabank project successfully implements a multi-process banking system that can handle concurrent client requests and synchronize teller operations through shared memory. Communication between clients, server, and tellers works reliably.

Although the final version suffers from incomplete database uploading and logging functionality, the system architecture, communication structures, and data models demonstrate a strong grasp of concurrent programming concepts and effective use of operating system mechanisms such as FIFOs, semaphores, and shared memory.

Project Execution Screenshot

The following screenshot shows the server and client programs being executed and interacting with each other successfully:



The screenshot displays two terminal windows side-by-side. The left window shows the server's execution, starting with 'make run-server' and displaying messages about database creation, client requests, and credit updates. The right window shows the client's execution, starting with 'make run-client' and displaying messages about file creation, connection to the server, and successful credit updates.

```
shashank@mediasoft:~/CS234_System_Programming/Networks/templates$ make run-server
rm -rf ClientServer/ClientDB.db
ClientServer/ClientDB.db: can't create: File exists
Creating existing bank database...
Waiting for clients to connect...
Received 1 client from 192.168.1.100
Client001 requested 300 credits... updating log
Teller P000001 is action serving Client001
... Teller P000010 is action serving Client001
Waiting for clients to connect...
Received 1 client from 192.168.1.100
Client001 requested 300 credits... updating log
Teller P000010 is action serving Client001
Client001 withdrawn 30 credits... operation not permitted.
Client001 requested 2000 credits... updating log

shashank@mediasoft:~/CS234_System_Programming/Networks/templates$ make run-client
rm -rf ClientServer/ClientDB.db
rm -rf ClientServer/Client001.txt
Waiting for clients to connect...
0 clients to connect... creating clients...
Connected to Address...
Client001 connected... depositing 300 credits
Client001 connected... withdrawing 30 credits
Client001 connected... depositing 2000 credits
shashank@mediasoft:~/CS234_System_Programming/Networks/templates$
```

Figure 3: Server and Client Execution