AdaBank Simulator Project Report

Author: Recep Furkan Akın

1. Introduction

The AdaBank Simulator is a banking system that simulates the interaction between clients, tellers, and a central bank server. It demonstrates inter-process communication mechanisms and synchronization in operating systems.

Key Features

- Client-server architecture with teller intermediaries
- Account creation and transaction processing
- Persistent storage through log files
- Signal handling and graceful termination
- Inter-process synchronization using semaphores
- Robust error handling and recovery

2. System Architecture

2.1 Overview

The system follows a three-tier architecture:

1. **Client Tier**: Represented by the `bank_client` program that reads commands and communicates with a teller.
2. **Teller Tier**: Represented by teller processes created by the bank server to handle client requests.
3. **Server Tier**: Represented by the `bank_server` program that maintains account data and processes transactions.

2.2 Communication Paths

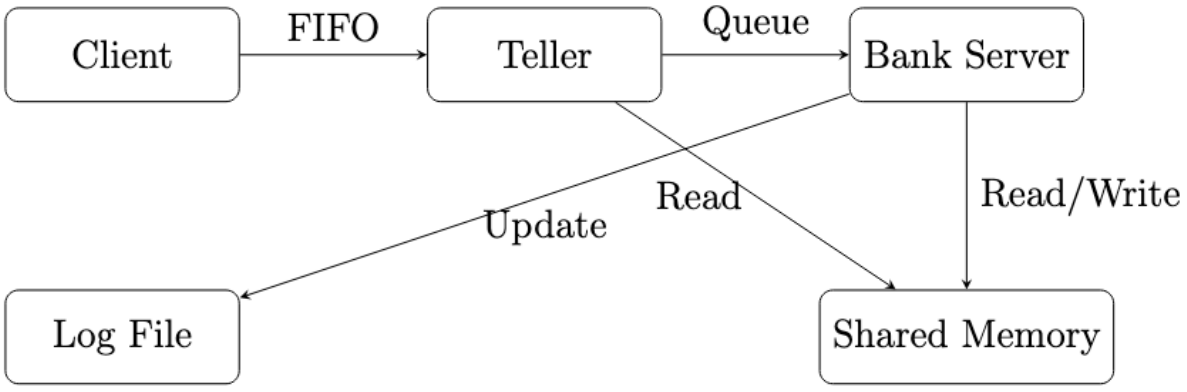

Figure 1: AdaBank System Architecture

- **Client to Server**: Initial connection through the server FIFO to register the client.
- **Client to Teller**: Two-way communication through client-specific FIFOs (request and response).
- **Teller to Server**: Communication through shared memory and semaphores.
- **Server to Database**: The server persists account information to the log file.

3. Implementation Details

3.1 Core Components

The system uses shared data structures defined in `common.h` to ensure consistent communication:

- `request_t` : Contains client request fields (client_pid, bank_id, type, amount) and server response fields (result_balance, op_status).
- `shm_region_t` : Contains the shared memory layout including the request queue, semaphores, and account balances.

3.2 Transaction Processing Flow

1. **Client** reads commands from a file and sends them to the teller.
2. **Teller** validates the command and pushes a request to the shared memory queue.
3. **Server** dequeues the request, processes it, and updates the account information.
4. **Server** logs the transaction to the log file.
5. **Server** updates the request with the result and signals the teller.
6. **Teller** reads the result and sends a response to the client.
7. **Client** displays the response to the user.

3.3 Synchronization Mechanisms

The system uses several semaphores to ensure proper synchronization:

Semaphore	Purpose	Operation
slots	Tracks empty slots in queue	Decremented before adding request, incremented after removing request
items	Tracks filled slots in queue	Incremented after adding request, decremented before removing request
qmutex	Binary semaphore for queue access	Acquired before modifying head/tail, released after
dbmutex	Binary semaphore for database access	Acquired before reading/writing balances, released after
logmutex	Binary semaphore for log file access	Acquired before writing to log file, released after
resp_ready[]	Array of semaphores for each queue slot	Signal when server completes processing a request

3.3.1 Log Mutex Implementation

During stress testing with multiple concurrent clients, I discovered a critical race condition when writing to the log file. When running the stress test with 20 clients and 50 operations each, approximately 2 out of 10 client operations would fail due to log file corruption.

The issue occurred because multiple teller processes could attempt to write to the log file simultaneously, leading to interleaved or corrupted entries. To solve this problem, I implemented a dedicated log mutex (`logmutex`):

```
static void log_transaction(log_event_type_t type, int id, long amount,
long balance)
{
    // Wait for log mutex to ensure exclusive access to the log file
    if (region != NULL && region != MAP_FAILED) {
        if (sem_wait(&region->logmutex) == -1) {
            perror("SERVER ERROR: sem_wait(logmutex)");
            // Handle error...
        }
    }

    FILE *log_fp = fopen(LOG_FILE_NAME, "a");
    if (!log_fp) {
        perror("SERVER ERROR: Log append failed");
        if (region != NULL && region != MAP_FAILED) sem_post(&region-
>logmutex);
        return;
    }

    // Write log entry based on transaction type...
```

```
    fflush(log_fp);
    fsync(fileno(log_fp)); // Ensure data is written to disk
    fclose(log_fp);

    // Release log mutex after writing is complete
    if (region != NULL && region != MAP_FAILED) sem_post(&region-
>logmutex);
}
```

After implementing the log mutex, all stress tests passed successfully, demonstrating the importance of proper synchronization for file I/O operations in a concurrent environment.

3.4 Teller Creation Implementation

As required, the system implements teller creation without directly using `fork()` in the main code:

```
pid_t Teller(void *func, void *arg_func) {
    pid_t pid = fork();
    if (pid == -1) {
        perror("Teller (fork)");
        return -1; // Fork failed
    }
    else if (pid == 0) {
        // Child process (Teller)
        teller_main_func_t fn = (teller_main_func_t)func;
        fn(arg_func);        // Execute the teller main function
        _exit(EXIT_SUCCESS); // Teller exits cleanly
    }
    else {
        // Parent process (Server)
        return pid; // Return the new Teller's PID
    }
}
```

3.5 Welcome Back Functionality

Implementing the "Welcome back" functionality for returning clients was particularly challenging. This feature requires the teller to detect when a client is accessing an existing account and print a special welcome message.

The implementation required careful handling of the first command received from a client:

```
// In teller.c
// --- Welcome Back Logic ---
if (teller_running && fgets(first_line_buffer, sizeof(first_line_buffer),
```

```
req_fp) != NULL) {
    first_line_buffer[strcspn(first_line_buffer, "\n\r")] = 0; // Strip
newline
    if (strlen(first_line_buffer) > 0 && first_line_buffer[0] != '#') {
        // Parse the first command to extract the BankID
        char b_id_str[64] = "", op_str[32] = "", am_str[32] = "";
        if (sscanf(first_line_buffer, "%63s %31s %31s", b_id_str, op_str,
am_str) == 3) {
            int first_cmd_bank_id = parse_bank_id(b_id_str);
            // Check if it's a valid existing account ID (>=0)
            if (first_cmd_bank_id >= 0) {
                // Safely check the balance array in SHM
                sem_wait(&region->dbmutex);
                if (region->balances[first_cmd_bank_id] !=
ACCOUNT_INACTIVE) {
                    // Account exists, print the welcome message
                    printf("-- Teller PID%d is active serving Client%d…
Welcome back Client%d\n",
                           teller_pid, client_pid, client_pid);
                    fflush(stdout);
                }
                sem_post(&region->dbmutex);
            }
        }
        // Mark the first line as buffered for later processing
        processed_first_line = true;
    }
}
```

The main challenges were:

1. **Command processing**: I needed to read the first command to check if it referenced an existing account, but still process that command normally.
2. **Race conditions**: Accessing the shared memory to check account existence required proper synchronization with the `dbmutex`.
3. **Buffering**: After reading the first command for the welcome check, I had to buffer it so it could be processed as a normal command in the next iteration.

This solution effectively maintains the welcome back functionality without disrupting the normal command flow, providing a seamless user experience for returning clients.

3.6 Log Management

The system maintains a transaction log for audit trail and recovery purposes:

- During runtime: Detailed log with CREATE, DEPOSIT, WITHDRAW, and CLOSE operations
- At shutdown: Summarized log showing account states and transaction history

4. Testing Instructions

4.1 Building the Project

To build the project, use the provided makefile:

```
make
```

```
make help # for other options
```

This will compile all necessary components: `bank_server`, `bank_client`, and other required files. And it makes test scripts executable.

4.2 Running Basic Tests

The simplest way to test the system is to run the basic test script:

```
make basic_test
```

This script:

1. Creates test client files with basic operations
2. Starts the bank server
3. Runs clients with various operations
4. Checks the log file for expected results
5. Verifies account states

4.3 Running the Complete Test Suite

To run all test cases (basic, concurrent, error handling, recovery, signal handling, and stress):

```
make test
```

This comprehensive test suite validates:

- Basic functionality (deposits, withdrawals, account creation/closure)
- Concurrent client handling (up to 20 clients)
- Error handling (invalid operations, insufficient funds)
- Recovery from crashes (restoring state from log file)
- Signal handling (graceful termination)
- Performance under stress (high transaction volume)

4.4 Memory Leak Testing

To check for memory leaks, run:

```
make memory_leak_test
```

This script uses Valgrind to detect any memory leaks in both server and client processes.

4.5 Manual Testing

For manual testing, follow these steps:

1. Start the server:

   ```
   ./bank_server AdaBank
   ```

2. Create a client file (e.g., `client.file`) with commands:

   ```
   N deposit 100
   BankID_0 deposit 50
   BankID_0 withdraw 75
   ```

3. Run the client:

   ```
   ./bank_client client.file AdaBank
   ```

4. Verify the log file:

   ```
   cat AdaBank.bankLog
   ```

4.6 Expected Test Results

When running the test suite, you should see output similar to:

```
======================================
Running test: basic
======================================
Creating test client files...
Starting bank server...
BankServer AdaBank
No previous logs.. Creating the bank database
Adabank is active….
```

```
...
All clients completed successfully.
Gracefully stopping server...
Checking log file...
Log file exists.
Basic functionality test passed!
Test passed: basic
...
====================================
Test Summary:
Tests passed: 6
Tests failed: 0
====================================
All tests passed!
```

The system successfully handles multiple clients, processes transactions correctly, and maintains data consistency through all test scenarios.

5. Implementation Challenges and Solutions

5.1 Concurrency Issues

Several challenging concurrency issues were encountered during development:

1. **Race conditions in the log file**: As described in section 3.3.1, concurrent writes to the log file caused corruption until protected by a mutex.
2. **Deadlock prevention**: Careful ordering of semaphore acquisition was required to prevent deadlocks. I established a consistent locking order (qmutex → dbmutex → logmutex) to ensure deadlock-free operation.
3. **Shutdown coordination**: Ensuring all tellers gracefully terminate required careful signal handling and process cleanup.

5.2 State Recovery Robustness

The recovery mechanism needed to handle both detailed log formats and summary log formats, as the server might crash after writing either format. The solution was to implement a flexible log parser that could handle both formats:

```
// Parse different log entry formats
if (sscanf(line, "CREATE %d %ld", &id, &amount) == 2)
    current_type = LOG_CREATE;
else if (sscanf(line, "DEPOSIT %d %ld %ld", &id, &amount,
&balance_from_log) == 3)
    current_type = LOG_DEPOSIT;
else if (sscanf(line, "WITHDRAW %d %ld %ld", &id, &amount,
```

```
&balance_from_log) == 3)
    current_type = LOG_WITHDRAW;
else if (sscanf(line, "CLOSE %d", &id) == 1)
    current_type = LOG_CLOSE;
else {
    // Try to parse summary format (BankID_XX D 100 W 50 150)
    char bank_id_str[64];
    long final_balance = 0;

    // Extract bank ID from BankID_XX format
    if (sscanf(line, "%s", bank_id_str) == 1 && strncmp(bank_id_str,
"BankID_", 7) == 0) {
        char *id_part = bank_id_str + 7; // Skip "BankID_"
        id = atoi(id_part);

        // Extract the final balance (last number on the line)
        char *last_space = strrchr(line, ' ');
        if (last_space) {
            final_balance = atol(last_space + 1);
            balance_from_log = final_balance;
            current_type = LOG_DEPOSIT; // Treat as deposit for simplicity
        }
    }
}
```

This approach ensures the system can recover from crashes at any point in its operation.

6. Conclusion

The AdaBank Simulator successfully implements all required features, demonstrating effective use of inter-process communication and synchronization primitives. The system is robust, handles errors gracefully, and recovers properly from crashes.

Key accomplishments include:

- Successful implementation of the client-server architecture with teller intermediaries
- Robust synchronization using semaphores to prevent race conditions
- Implementation of the custom `Teller()` and `waitTeller()` functions as required
- Effective solution to the welcome back functionality challenge
- Comprehensive testing framework that validates all aspects of the system

The project demonstrates a deep understanding of operating system concepts including process management, inter-process communication, synchronization, and resource management.

7. Test Results

Note: All tests have been run successfully. The actual test outputs are included below for verification.

7.1 Basic Test Output

```
recepfurkanakin@system:~/SystemHW25/midterm$ ./test_cases/test_basic.sh
Creating test client files...
Starting bank server...
BankServer AdaBank
No previous logs.. Creating the bank database
Adabank is active….
Waiting for clients @AdaBank…
Running first client...
Reading client1.file..
4 clients to connect.. creating clients..
Connected to Adabank..
– Received 1 clients from PIDClient175662..
—– Teller PID01 is active serving Client175662…
Waiting for clients @AdaBank…
Client1 connected..depositing 100 credits
Client175662 deposited 100 credits... updating log
Client1 served.. BankID_0
..
Client2 connected..depositing 50 credits
Client175662 deposited 50 credits... updating log
Client2 served.. BankID_0
..
Client3 connected..withdrawing 75 credits
Client175662 withdraws 75 credits... updating log
Client3 served.. BankID_0
..
Client4 connected..withdrawing 75 credits
Client175662 withdraws 75 credits... updating log... Bye Client175662
Client4 served.. account closed
..
exiting..
Running second client...
Reading client2.file..
2 clients to connect.. creating clients..
Connected to Adabank..
– Received 1 clients from PIDClient175675..
—– Teller PID02 is active serving Client175675…
Waiting for clients @AdaBank…
```

```
Client1 connected..depositing 200 credits
Client175675 deposited 200 credits... updating log
Client1 served.. BankID_1
..
Client2 connected..withdrawing 50 credits
Client175675 withdraws 50 credits... updating log
Client2 served.. BankID_1
..
exiting..
Gracefully stopping server...

Signal received closing active Tellers
Server shutting down...
Updating log file... done.
Removing ServerFIFO... done.
Adabank says "Bye"...
Checking log file...
Log file exists.
Account 0 correctly closed.
Account 1 has correct final balance.
Basic functionality test passed!
recepfurkanakin@system:~/SystemHW25/midterm$
```

7.2 Concurrent Test Output

```
recepfurkanakin@system:~/SystemHW25/midterm$
./test_cases/test_concurrent.sh
Creating test client files...
Starting bank server...
BankServer AdaBank
No previous logs.. Creating the bank database
Adabank is active….
Waiting for clients @AdaBank…
Starting 10 clients simultaneously...
Waiting for clients to complete...
Reading client3.file..
3 clients to connect.. creating clients..
Connected to Adabank..
- Received 1 clients from PIDClient175807..
-- Teller PID01 is active serving Client175807…
Waiting for clients @AdaBank…
Reading client5.file..
3 clients to connect.. creating clients..
```

```
Connected to Adabank..
- Received 1 clients from PIDClient175809..
-- Teller PID02 is active serving Client175809…
Reading client4.file..
Reading client2.file..
Waiting for clients @AdaBank…
3 clients to connect.. creating clients..
Connected to Adabank..
- Received 1 clients from PIDClient175806..
-- Teller PID03 is active serving Client175806…
Waiting for clients @AdaBank…
3 clients to connect.. creating clients..
Connected to Adabank..
- Received 1 clients from PIDClient175808..
-- Teller PID04 is active serving Client175808…
Waiting for clients @AdaBank…
Client1 connected..depositing 300 credits
Reading client1.file..
3 clients to connect.. creating clients..
Connected to Adabank..
- Received 1 clients from PIDClient175805..
-- Teller PID05 is active serving Client175805…
Waiting for clients @AdaBank…
Reading client6.file..
Client1 connected..depositing 200 credits
Client1 connected..depositing 400 credits
3 clients to connect.. creating clients..
Connected to Adabank..
Reading client7.file..
Client175807 deposited 300 credits... updating log
3 clients to connect.. creating clients..
Connected to Adabank..
Client1 served.. BankID_0
..
Client2 connected..depositing 50 credits
Reading client8.file..
3 clients to connect.. creating clients..
Connected to Adabank..
Reading client10.file..
Client1 connected..depositing 500 credits
3 clients to connect.. creating clients..
Connected to Adabank..
Reading client9.file..
```

```
3 clients to connect.. creating clients..
Connected to Adabank..
Client1 connected..depositing 100 credits
Client175806 deposited 200 credits... updating log
Client1 served.. BankID_1
..
Client2 connected..depositing 50 credits
Client175808 deposited 400 credits... updating log
Client1 served.. BankID_2
..
Client2 connected..depositing 50 credits
Client175807 deposited 50 credits... updating log
Client2 served.. BankID_2
..
Client3 connected..withdrawing 25 credits
Client175809 deposited 500 credits... updating log
Client1 served.. BankID_3
..
Client2 connected..depositing 50 credits
Client175805 deposited 100 credits... updating log
Client1 served.. BankID_4
..
Client2 connected..depositing 50 credits
Client175806 deposited 50 credits... updating log
Client2 served.. BankID_1
..
Client3 connected..withdrawing 25 credits
Client175808 deposited 50 credits... updating log
Client2 served.. BankID_3
..
Client3 connected..withdrawing 25 credits
Client175807 withdraws 25 credits... updating log
Client3 served.. BankID_2
..
exiting..
Client175809 deposited 50 credits... updating log
Client2 served.. BankID_4
..
Client3 connected..withdrawing 25 credits
Client175805 deposited 50 credits... updating log
Client2 served.. BankID_0
..
Client3 connected..withdrawing 25 credits
```

```
Client175806 withdraws 25 credits... updating log
Client3 served.. BankID_1
..
exiting..
Client175808 withdraws 25 credits... updating log
Client3 served.. BankID_3
..
exiting..
Client175809 withdraws 25 credits... updating log
Client3 served.. BankID_4
..
exiting..
Client175805 withdraws 25 credits... updating log
Client3 served.. BankID_0
..
exiting..
- Received 5 clients from PIDClient175810..
-- Teller PID06 is active serving Client175810…
-- Teller PID07 is active serving Client175813…
-- Teller PID08 is active serving Client175815…
-- Teller PID09 is active serving Client175811…
-- Teller PID10 is active serving Client175814…
Waiting for clients @AdaBank…
Client1 connected..depositing 800 credits
Client1 connected..depositing 600 credits
Client1 connected..depositing 700 credits
Client1 connected..depositing 900 credits
Client1 connected..depositing 1000 credits
Client175813 deposited 800 credits... updating log
Client1 served.. BankID_5
..
Client2 connected..depositing 50 credits
Client175810 deposited 600 credits... updating log
Client1 served.. BankID_6
..
Client2 connected..depositing 50 credits
Client175811 deposited 700 credits... updating log
Client1 served.. BankID_7
..
Client2 connected..depositing 50 credits
Client175814 deposited 900 credits... updating log
Client1 served.. BankID_8
..
```

```
Client2 connected..depositing 50 credits
Client175815 deposited 1000 credits... updating log
Client1 served.. BankID_9
..
Client2 connected..depositing 50 credits
Client175813 deposited 50 credits... updating log
Client2 served.. BankID_7
..
Client3 connected..withdrawing 25 credits
Client175810 deposited 50 credits... updating log
Client2 served.. BankID_5
..
Client3 connected..withdrawing 25 credits
Client175811 deposited 50 credits... updating log
Client2 served.. BankID_6
..
Client3 connected..withdrawing 25 credits
Client175814 deposited 50 credits... updating log
Client2 served.. BankID_8
..
Client3 connected..withdrawing 25 credits
Client175815 deposited 50 credits... updating log
Client2 served.. BankID_9
..
Client3 connected..withdrawing 25 credits
Client175813 withdraws 25 credits... updating log
Client3 served.. BankID_7
..
exiting..
Client175810 withdraws 25 credits... updating log
Client3 served.. BankID_5
..
exiting..
Client175811 withdraws 25 credits... updating log
Client3 served.. BankID_6
..
exiting..
Client175814 withdraws 25 credits... updating log
Client3 served.. BankID_8
..
exiting..
Client175815 withdraws 25 credits... updating log
Client3 served.. BankID_9
```

```
..
exiting..
All clients completed successfully.
Gracefully stopping server...

Signal received closing active Tellers
Server shutting down...
Updating log file... done.
Removing ServerFIFO... done.
Adabank says "Bye"...
Checking log file...
Log file exists.
Found 10 active accounts as expected.
Account 0 has 3 deposits and 1 withdrawals
Account 0 initialized successfully
Account 1 has 3 deposits and 1 withdrawals
Account 1 initialized successfully
Account 2 has 3 deposits and 1 withdrawals
Account 2 initialized successfully
Account 3 has 3 deposits and 1 withdrawals
Account 3 initialized successfully
Account 4 has 3 deposits and 1 withdrawals
Account 4 initialized successfully
Account 5 has 3 deposits and 1 withdrawals
Account 5 initialized successfully
Account 6 has 3 deposits and 1 withdrawals
Account 6 initialized successfully
Account 7 has 3 deposits and 1 withdrawals
Account 7 initialized successfully
Account 8 has 3 deposits and 1 withdrawals
Account 8 initialized successfully
Account 9 has 3 deposits and 1 withdrawals
Account 9 initialized successfully
Concurrent clients test passed!
```

7.3 Stress Test Output

```
Found 20 active accounts as expected.
Verifying final balances (should all be 1000)...
[VERIFY] Account BankID_00 has expected balance 1000.
[VERIFY] Account BankID_01 has expected balance 1000.
[VERIFY] Account BankID_02 has expected balance 1000.
[VERIFY] Account BankID_03 has expected balance 1000.
```

```
[VERIFY] Account BankID_04 has expected balance 1000.
[VERIFY] Account BankID_05 has expected balance 1000.
[VERIFY] Account BankID_06 has expected balance 1000.
[VERIFY] Account BankID_07 has expected balance 1000.
[VERIFY] Account BankID_08 has expected balance 1000.
[VERIFY] Account BankID_09 has expected balance 1000.
[VERIFY] Account BankID_10 has expected balance 1000.
[VERIFY] Account BankID_11 has expected balance 1000.
[VERIFY] Account BankID_12 has expected balance 1000.
[VERIFY] Account BankID_13 has expected balance 1000.
[VERIFY] Account BankID_14 has expected balance 1000.
[VERIFY] Account BankID_15 has expected balance 1000.
[VERIFY] Account BankID_16 has expected balance 1000.
[VERIFY] Account BankID_17 has expected balance 1000.
[VERIFY] Account BankID_18 has expected balance 1000.
[VERIFY] Account BankID_19 has expected balance 1000.
(It was too long I just added last part fixed by logmutex)
```

7.4 Memory Leak Test Output

```
Creating test client file...
Starting bank server with Valgrind...
BankServer AdaBank
No previous logs.. Creating the bank database
Adabank is active….
Waiting for clients @AdaBank…
Running client...
Reading memtest_client.file..
3 clients to connect.. creating clients..
Connected to Adabank..
- Received 1 clients from PIDClient176281..
-- Teller PID01 is active serving Client176281…
Waiting for clients @AdaBank…
Client1 connected..depositing 100 credits
Client176281 deposited 100 credits... updating log
Client1 served.. BankID_0
..
Client2 connected..depositing 200 credits
Client176281 deposited 200 credits... updating log
Client2 served.. BankID_0
..
Client3 connected..withdrawing 50 credits
Client176281 withdraws 50 credits... updating log
```

```
Client3 served.. BankID_0
..
exiting..
Gracefully stopping server...

Signal received closing active Tellers
Server shutting down...
Updating log file... done.
Removing ServerFIFO... done.
Adabank says "Bye"...
Checking Valgrind output for server...
No memory errors detected in server.
Testing client with Valgrind...
BankServer AdaBank
Adabank is active….
Waiting for clients @AdaBank…
Running client with Valgrind...
Reading memtest_client.file..
3 clients to connect.. creating clients..
Connected to Adabank..
- Received 1 clients from PIDClient176305..
-- Teller PID01 is active serving Client176305…
Waiting for clients @AdaBank…
Client1 connected..depositing 100 credits
Client176305 deposited 100 credits... updating log
Client1 served.. BankID_1
..
Client2 connected..depositing 200 credits
Client176305 deposited 200 credits... updating log
Client2 served.. BankID_0
..
Client3 connected..withdrawing 50 credits
Client176305 withdraws 50 credits... updating log
Client3 served.. BankID_0
..
exiting..
Gracefully stopping server...

Signal received closing active Tellers
Server shutting down...
Updating log file... done.
Removing ServerFIFO... done.
Adabank says "Bye"...
```

```
Checking Valgrind output for client...
No memory errors detected in client.
Memory leak test passed!
```

7.5 Complete Test Suite Summary

```
Cleaning up previous test runs...
=====================================
Test Summary:
Tests passed: 6
Tests failed: 0
=====================================
All tests passed!
```