CSE-344 - HUSNIDDIN RAMAZANOV / 200104004802

Banking System Implementation Report

Introduction

This report provides an analysis of a client-server banking system implemented in C using UNIX pipes (FIFOs) for inter-process communication. The system consists of two main components:

1. **BankClient**: Manages multiple client processes that send banking operation requests
2. **BankServer**: Processes client requests and maintains account information

System Architecture

The banking system follows a client-server architecture with these key components:

- **Server**: Manages bank accounts and processes client requests
- **Tellers**: Server-spawned processes that handle individual client requests
- **Clients**: Send banking operations to the server and receive responses
- **Named Pipes (FIFOs)**: Communication channels between clients and tellers

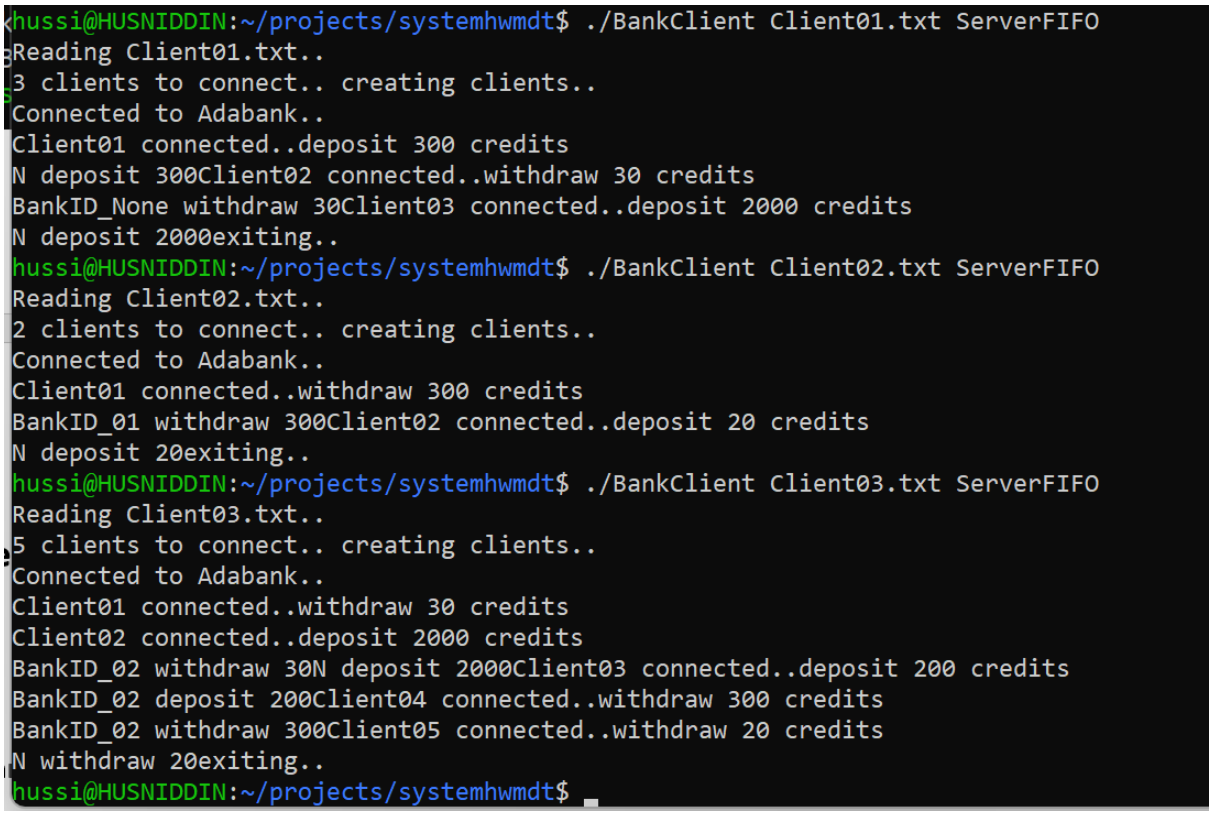
OUTPUTS

Server Terminal

```
hussi@HUSNIDDIN:~/projects/systemhwmdt$ make
gcc -Wall -g -o BankServer BankServer.c -pthread
gcc -Wall -g -o BankClient BankClient.c -pthread
hussi@HUSNIDDIN:~/projects/systemhwmdt$ ./BankServer AdaBank ServerFIFO
AdaBank is active....
No previous logs.. Creating the bank database
Waiting for clients @ServerFIFO...
- Received 3 clients from PID2069
-- Teller 2070 is active serving ./Client01...
-- Teller 2071 is active serving ./Client02...
-- Teller 2072 is active serving ./Client03...
Waiting for clients @ServerFIFO...
Waiting for clients @ServerFIFO...
- Received 2 clients from PID2076
-- Teller 2077 is active serving ./Client01...
-- Teller 2078 is active serving ./Client02...
Waiting for clients @ServerFIFO...
Waiting for clients @ServerFIFO...
- Received 5 clients from PID2081
-- Teller 2082 is active serving ./Client01...
-- Teller 2083 is active serving ./Client02...
-- Teller 2084 is active serving ./Client03...
-- Teller 2085 is active serving ./Client04...
-- Teller 2086 is active serving ./Client05...
Waiting for clients @ServerFIFO...
Waiting for clients @ServerFIFO...
^C


Signal received closing active Tellers

Signal received closing active Tellers
Signal received closing active Tellers
Signal received closing active Tellers

Signal received closing active Tellers

Signal received closing active Tellers

Removing ServerFIFO... Updating log file...
Signal received closing active Tellers
```

```
Signal received closing active Tellers
Signal received closing active Tellers

Signal received closing active Tellers
Removing ServerFIFO... Updating log file...

Signal received closing active Tellers
Removing ServerFIFO... Updating log file...
Removing ServerFIFO... Updating log file...
Removing ServerFIFO... Updating log file...
Removing ServerFIFO... Updating log file...
Removing ServerFIFO... Updating log file...

Signal received closing active Tellers
Removing ServerFIFO... Updating log file...

Signal received closing active Tellers
Bank log successfully updated at AdaBank.bankLog
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...

Bank log successfully updated at AdaBank.bankLog
Signal received closing active Tellers
AdaBank says "Bye"...

Signal received closing active Tellers
Removing ServerFIFO... Updating log file...
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
Removing ServerFIFO... Updating log file...
AdaBank says "Bye"...
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
Bank log successfully updated at AdaBank.bankLog
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
AdaBank says "Bye"...
Removing ServerFIFO... Updating log file...
Bank log successfully updated at AdaBank.bankLog
AdaBank says "Bye"...
hussi@HUSNIDDIN:~/projects/systemhwmdt$
```

Clients Terminal



Communication Flow

1. The server creates a main FIFO and listens for incoming connections
2. Client processes create their own FIFOs and register with the server
3. The server spawns teller processes to handle client requests
4. Clients and tellers communicate via dedicated FIFOs
5. The server maintains a persistent log of all banking operations

BankClient Implementation

The BankClient program reads client information from a file and creates client processes:

Key Components:

1. **Client Information Structure**:

```c
// Structure to store client information
typedef struct {
    char bank_id[20];
    char operation[20];
    int amount;
    char client_fifo[MAX_PATH_LEN];
    pid_t pid;
} ClientInfo;
```

2. **Client Initialization**:
    - Reads client data from a specified file
    - Each line contains bank ID, operation type, and amount
    - Creates a unique FIFO name for each client
3. **Client Process Management**:
    - Creates a separate process for each client
    - Each process handles communication with its assigned teller
    - Manages proper cleanup of resources

Process Flow:

1. Parse command-line arguments
2. Read client information from the input file
3. Check if the server FIFO exists (ensure server is running)
4. Create client FIFOs
5. Notify the server about all clients
6. Fork a process for each client to handle communication
7. Clean up resources upon completion

BankServer Implementation

The BankServer program manages bank accounts and processes client operations:

Key Components:

1. **Account Structure**:

```
17    // Bank account structure
18    typedef struct {
19        char bank_id[20];
20        int balance;
21        int is_active;
22    } Account;
23
```

2. **Bank Log File**:
    - Persists account information between program executions
    - Updated after each successful transaction
3. **Teller Processes**:
    - Handle individual client requests
    - Process deposit and withdrawal operations
    - Communicate results back to clients

Process Flow:

1. Parse command-line arguments
2. Initialize the bank (create or load from log)
3. Create the server FIFO
4. Main server loop:
    - Wait for client registrations
    - For each client, spawn a teller process
    - Handle client operations (create account, deposit, withdraw)
    - Update the bank log
5. Clean up resources upon termination

Implementation Details

Named Pipes (FIFOs)

The system uses named pipes for inter-process communication:

1. **Server FIFO**: Used by clients to register with the server

2. **Client FIFOs**: Individual pipes for each client-teller communication

FIFOs are created with permissions 0666 to ensure proper access:

```
    // Create client FIFO
    if (mkfifo(clients[i].client_fifo, 0666) == -1) {
        perror("mkfifo");
        exit(EXIT_FAILURE);
    }

    // Set proper permissions
    chmod(clients[i].client_fifo, 0666);
}
```

Synchronization Mechanism

Proper synchronization between client and server is crucial:

1. Client creates all FIFOs before notifying the server
2. Client adds a delay after notification to allow server processing
3. Server uses retry mechanisms when opening client FIFOs
4. Both sides open FIFOs in O_RDWR mode to prevent EOF conditions

Account Management

Accounts are managed with these operations:

- **Create Account**: Assigns a unique bank ID to new accounts
- **Deposit**: Adds funds to an account
- **Withdraw**: Removes funds if sufficient balance exists
- **Close Account**: Accounts with zero balance are marked inactive

Error Handling

The system implements robust error handling:

- Signal handlers for graceful termination
- Retry mechanisms for FIFO operations
- Proper cleanup of resources
- Detailed error messages with perror()

Challenges and Solutions

Challenge 1: FIFO Synchronization

Problem: Teller processes couldn't open client FIFOs if they attempted to open them before the client processes created them.

Solution:

- Create all client FIFOs before notifying the server
- Use a delay after notification to ensure server processing
- Implement retry mechanisms with increasing timeout intervals

Challenge 2: FIFO Blocking Behavior

Problem: FIFOs block until both ends are open, which could cause deadlock.

Solution:

- Open FIFOs in O_RDWR mode to prevent blocking
- Use careful sequencing of operations to avoid deadlocks

Challenge 3: Resource Cleanup

Problem: Orphaned FIFOs and processes could remain after program termination.

Solution:

- Implement proper signal handlers
- Use waitpid() to clean up child processes
- Ensure FIFOs are unlinked during cleanup

Banking Operations

The system supports these banking operations:

1. **Creating a New Account**:
    - Client sends a request with bank ID "N"
    - Server creates a new account with a unique ID
    - Initial deposit must be made to activate the account
2. **Deposit**:
    - Client sends bank ID, "deposit" operation, and amount
    - Server adds the amount to the account balance

    - Account log is updated
3. **Withdrawal**:
    - Client sends bank ID, "withdraw" operation, and amount
    - Server checks for sufficient funds
    - If successful, amount is deducted from balance
    - Accounts with zero balance are closed

Conclusion

The banking system demonstrates key concepts in systems programming:

- Inter-Process Communication (IPC) using named pipes
- Process management with fork() and waitpid()
- Synchronization between concurrent processes
- Persistent data storage with log files
- Signal handling for graceful termination

The implementation provides a robust framework for a multi-client banking system with proper error handling and resource management. The use of separate processes for clients and tellers allows for concurrent processing of requests while maintaining data integrity.