

**Expected grade: BA**

**Program Report: AdaBank System**

File hierarchy :

```
project/
├── bin/                (not exist before compilation)
│   ├── client
│   └── server
├── src/
│   ├── bank.h
│   ├── server.c
│   ├── client.c
│   └── teller.c
├── test/
│   ├── client01.file
│   ├── client02.file
│   ├── client03.file
│   └── client04.file
├── Makefile
└── AdaBank.bankLog
```

**Commands:**

```
./bin/server AdaBank /tmp/bank_server.fifo
./bin/client test/client01.file /tmp/bank_server.fifo
./bin/client test/client02.file /tmp/bank_server.fifo
./bin/client test/client03.file /tmp/bank_server.fifo
```

## 1. Overview of IPC and Synchronization Mechanisms

The AdaBank system employs a combination of inter-process communication (IPC) and synchronization techniques to manage concurrent client requests and ensure data consistency. The design revolves around a client-server architecture where the server processes transactions via *Teller* processes, while clients communicate through FIFO (named pipe) channels. Below are the key mechanisms:

### Inter-Process Communication (IPC):

- **FIFOs (Named Pipes):**
  - Clients send requests to the server via the server FIFO (*/tmp/bank\_server.fifo*).
  - Each client creates a unique FIFO (e.g., */tmp/client\_<PID>*) to receive responses.
  - The server reads requests in batches from its FIFO and dispatches them to Teller processes.
  - Responses are written back to the client's FIFO atomically using a semaphore-guarded write operation.
- **Shared Memory:**
  - A shared memory segment (*/bank\_shm*) stores the bank's account data (SharedData structure), including account IDs, balances, and client counts.
  - This allows all Teller processes (child processes of the server) to access and modify the same data structure concurrently.

### Synchronization:

- **Semaphores:**
  - *sem (/bank\_sem)*: Ensures mutual exclusion when accessing the SharedData structure. Critical sections (e.g., account creation, balance updates) are guarded by this semaphore.
  - *fifo\_mutex (/bank\_fifo\_mutex)*: Protects write operations to client FIFOs to prevent interleaved messages.
  - *req\_sem (/bank\_req\_sem)*: Coordinates batch processing of client requests. The server waits on this semaphore until clients signal new requests.
- **Process Management:**
  - Each client request is handled by a dedicated *Teller* process (via *fork()*), isolating transaction logic and preventing blocking.
  - The server uses *waitpid()* to ensure Tellers complete processing before accepting new batches.

## 2. Core Function Logic

*handle\_client(Request \*req)*

**Purpose:** Validates client requests, assigns client/account IDs, and sends initial responses.

### Workflow:

1. **Semaphore Acquisition:** The function begins by acquiring *sem* to safely access *SharedData*.
2. **Account Validation:**
  - For new clients (*account\_id* = "NEW"), increments *client\_count* and assigns a sequential client ID.
  - For existing accounts, searches *SharedData.accounts* for a matching ID. If absent, treats it as a new client.
3. **Response Generation:** Constructs a confirmation message (e.g., "*Client01 connected..deposit 300 credits*").
4. **FIFO Write:** Uses *fifo\_mutex* to atomically write the response to the client's FIFO.

### Critical Sections:

- Modifications to *client\_count* and accounts are protected by *sem*.
- FIFO writes are guarded by *fifo\_mutex* to ensure atomicity.

*deposit(void \*arg)*

**Purpose:** Processes deposit requests, updates account balances, and logs transactions.

### Workflow:

1. **Semaphore Acquisition:** Locks *sem* to enter the critical section.
2. **New Account Handling:**
  - If *account\_id* is "NEW", creates a new account with a unique *BankID\_XX* and initializes its balance.
  - Rejects the request if *MAX\_ACCOUNTS* is reached.
3. **Existing Account Handling:**
  - Searches for the account ID in *SharedData.accounts*.
  - Updates the balance and logs the transaction via *write\_log()*.
4. **Response Dispatch:** Sends a success/failure message to the client's FIFO using *fifo\_mutex*.

### Edge Cases:

- Handles maximum account limits.
- Validates account existence before modifying balances.

*withdraw(void \*arg)*

**Purpose:** Processes withdrawals, validates balances, and removes empty accounts.

**Workflow:**

1. **Semaphore Acquisition:** Locks sem to enter the critical section.
2. **Account Validation:**
  - Searches for the account ID in SharedData.accounts.
  - Checks if the balance is sufficient for the withdrawal.
3. **Balance Adjustment:**
  - Deducts the amount and updates the log.
  - If the balance reaches zero, removes the account from SharedData.accounts using memmove().
4. **Response Dispatch:** Notifies the client of success/failure via FIFO.

**Edge Cases:**

- Rejects withdrawals from invalid or insufficient accounts.
- Handles account removal atomically to prevent data corruption.

### 3. Signal Handling and Cleanup

The server and clients gracefully handle termination signals (e.g., SIGINT):

- **Server Cleanup** (*cleanup()*):
    - Unmaps shared memory (*munmap()*) and unlinks IPC resources (*shm\_unlink*, *sem\_unlink*).
    - Removes the server FIFO and updates the log file.
  - **Client Cleanup:**
    - Closes FIFOs and releases semaphores to prevent resource leaks.
- 

### 4. Transaction Logging

The *write\_log()* function appends transaction details to *AdaBank.bankLog* with timestamps. Each entry includes:

- Account ID, transaction type (D/W), amount, balance, and timestamp.

## **Test Case Description: AdaBank Transaction Processing**

### **Server-Side Execution**

#### **1. Initialization:**

- The server (AdaBank) starts, initializes a new database (no prior logs), and listens on /tmp/bank\_server.fifo.
- Shared memory and semaphores are set up to manage concurrent access to account data.

#### **2. Batch Processing:**

- **Batch 1 (4 Clients):**
  - **Client01:** Deposits 300 credits, creating BankID\_01.
  - **Client02:** Withdrawal fails (invalid operation, e.g., insufficient balance or invalid account).
  - **Client03:** Deposits 1000 credits, creating BankID\_02.
  - **Client04:** Withdrawal fails.
- **Batch 2 (2 Clients):**
  - **Client01:** Withdraws 300 credits, closing BankID\_01.
  - **Client05:** Deposits 20 credits, creating BankID\_03.
- **Batch 3 (5 Clients):**
  - **Client03:** Withdraws 30 credits (balance: 970), deposits 200 credits (balance: 1170), withdraws 300 credits (balance: 870).
  - **Client06:** Deposits 2000 credits, creating BankID\_04.
  - **Client07:** Withdrawal fails.

#### **3. Termination:**

- On receiving SIGINT (^C), the server closes active tellers, removes FIFOs/semaphores, updates logs, and exits gracefully.

```

^[[Aanhelina@vbox:~/Desktop/banking$ ./bin/server AdaBank /tmp/bank_server.fi
Adabank is active...
No previous logs.. Creating the bank database

. . .
Waiting for clients @/tmp/bank_server.fifo..
-- Received 4 clients from PIDClientX..

-- Teller PID150137 is active serving Client01..
--Client01: Deposited 300 credits. New account: BankID_01... updating log
-- Teller PID150138 is active serving Client02..
Client02: Withdrawal failed. Invalid operation.
-- Teller PID150139 is active serving Client03..
--Client03: Deposited 1000 credits. New account: BankID_02... updating log
-- Teller PID150140 is active serving Client04..
Client04: Withdrawal failed. Invalid operation.
. . .
Waiting for clients @/tmp/bank_server.fifo..
-- Received 2 clients from PIDClientX..

-- Teller PID150226 is active serving Client01..
Client01: Withdrew 300 credits. Account closed... updating log
-- Teller PID150227 is active serving Client05..
--Client05: Deposited 20 credits. New account: BankID_03... updating log
. . .
Waiting for clients @/tmp/bank_server.fifo..
-- Received 5 clients from PIDClientX..

-- Teller PID150285 is active serving Client03..
Client03: Withdrew 30 credits. New balance: 970... updating log
-- Teller PID150286 is active serving Client06..
--Client06: Deposited 2000 credits. New account: BankID_04... updating log
-- Teller PID150287 is active serving Client03..
Client03: Deposited 200 credits. New balance: 1170... updating log
-- Teller PID150288 is active serving Client03..
Client03: Withdrew 300 credits. New balance: 870... updating log
-- Teller PID150289 is active serving Client07..
Client07: Withdrawal failed. Invalid operation.
. . .
Waiting for clients @/tmp/bank_server.fifo..
^C
Signal received closing active Tellers
Removing ServerFIFO.. Updating log file..
Adabank says "Bye"...

```

≡ AdaBank.bankLog U X

banking > ≡ AdaBank.bankLog

```

1  # BankID_01 D 300 300 @01:02 April 28
2  # BankID_02 D 1000 1000 @01:02 April 28
3  # BankID_01 W 300 0 @01:02 April 28
4  # BankID_03 D 20 20 @01:02 April 28
5  # BankID_02 W 30 970 @01:02 April 28
6  # BankID_04 D 2000 2000 @01:02 April 28
7  # BankID_02 D 200 1170 @01:02 April 28
8  # BankID_02 W 300 870 @01:02 April 28
9

```

---

## Client-Side Execution

### Client 01 (test/client01.file)

- **Transactions:**
  - **Deposit 300:** Success → Account BankID\_01 created.
  - **Withdraw 200:** Fails (invalid operation).
  - **Deposit 1000:** Success → Account BankID\_02 created.
  - **Withdraw 275:** Fails.
- **Outcome:** Two accounts created; two withdrawals rejected.

### Client 02 (test/client02.file)

- **Transactions:**
  - **Withdraw 300:** Success → BankID\_01 closed (balance zero).
  - **Deposit 20:** Success → Account BankID\_03 created.
- **Outcome:** One account closed, one new account created.

### Client 03 (test/client03.file)

- **Transactions:**
  - **Withdraw 30:** Success → Balance updated to 970.
  - **Deposit 200:** Success → Balance updated to 1170.
  - **Withdraw 300:** Success → Balance updated to 870.
  - **Deposit 2000:** Success → Account BankID\_04 created.
  - **Withdraw 20:** Fails (likely invalid account).
- **Outcome:** Multiple balance updates on BankID\_02; one new account created.



```

anhelina@vbox:~/Desktop/bankings$ ./bin/client test/client01.file /tmp/bank_server.fifo
Client fifo name: /tmp/client_150136
Reading test/client01.file..
4 clients to connect.. creating clients..
Connected to Adabank..
Client01 connected..deposit 300 credits
--Client01 served.. BankID_01
Client02 connected..withdraw 200 credits
--Client02: Withdrawal failed. Invalid operation.
Client03 connected..deposit 1000 credits
--Client03 served.. BankID_02
Client04 connected..withdraw 275 credits
--Client04: Withdrawal failed. Invalid operation.
^C
Signal received closing active Client
Removing Client FIFO..
EXIT...
anhelina@vbox:~/Desktop/bankings$ ./bin/client test/client02.file /tmp/bank_server.fifo
Client fifo name: /tmp/client_150225
Reading test/client02.file..
2 clients to connect.. creating clients..
Connected to Adabank..
Client01 connected..withdraw 300 credits
--Client01 served.. account closed
Client05 connected..deposit 20 credits
--Client05 served.. BankID_03
exiting..
anhelina@vbox:~/Desktop/bankings$ ./bin/client test/client03.file /tmp/bank_server.fifo
Client fifo name: /tmp/client_150284
Reading test/client03.file..
5 clients to connect.. creating clients..
Connected to Adabank..
Client03 connected..withdraw 30 credits
--Client03 served.. New balance: 970... updating log
Client06 connected..deposit 2000 credits
--Client06 served.. BankID_04
Client03 connected..deposit 200 credits
--Client03 served.. New balance: 1170
Client03 connected..withdraw 300 credits
--Client03 served.. New balance: 870... updating log
Client07 connected..withdraw 20 credits

```

## Key Observations

1. **Concurrency & Synchronization:**
  - Teller processes (e.g., PIDI\$0137) handle requests in parallel.
  - Semaphores ensure atomic access to shared data (e.g., BankID\_01 modifications).
2. **FIFO Communication:**
  - Clients use unique FIFOs to receive responses.
  - Server writes responses atomically using fifo\_mutex to prevent interleaving.
3. **Error Handling:**
  - Invalid withdrawals are rejected with clear error messages (e.g., "Invalid operation").
  - Closed accounts are removed from the shared database.
4. **Logging:**

- All transactions are logged with timestamps and balances in server, client and AdaBank.bankLog

## **Conclusion**

The AdaBank system effectively leverages FIFOs, shared memory, and semaphores to achieve concurrency and data consistency. Critical functions like `handle_client`, `deposit`, and `withdraw` ensure thread-safe operations through meticulous semaphore management. While the design addresses core banking operations, future enhancements could focus on optimizing IPC performance and error recovery.