Banking Management System

Overview

The **Banking Management System** is a C-based **client–server** application that simulates core banking operations such as **account management**, **transactions**, **loan processing**, and **administrative control**. It supports **multiple concurrent users** with **role-based access control** and ensures **data integrity** through efficient file handling and synchronization mechanisms.

Roles and Functionalities

Customer

- View balance, deposit/withdraw money, transfer funds
- Apply for loans and view loan status
- View transaction history and give feedback
- Change password and manage profile

Employee

- Add or modify customer accounts
- Approve/reject loans assigned by the manager
- View customer details and transaction records

Manager

- Activate or deactivate customer accounts
- Assign loan requests to employees
- Review customer feedback

Administrator

- Manage employee accounts and user roles
- Create or modify employees

How to Use

1. **Build the project**

   ```
   make
   ```

2. **Run the Server**

   ```
   ./server
   ```

3. **Run the Client**

```
./client
```

4. **Login Details**
    - **Initial Admin Credentials:**
        - **Username:** `E001`
        - **Password:** `admin`
    - The administrator can then create additional employee and customer accounts.
5. **Multiple Clients**
    - Multiple clients can connect to the same server concurrently.
    - Each user session is **isolated** and **independently managed**.

Project Structure

File	Description
`server.c`	Handles client connections and manages sessions
`client.c`	Provides the user interface for clients
`handler.c`	Routes user requests to appropriate functions
`customer.c`	Implements all customer operations
`employee.c`	Implements employee-related operations
`loan.c`	Manages loan requests and approvals
`utils.c`	Contains helper and utility functions
`Makefile`	Used to build both server and client binaries
`class_diagram.png`	UML Class Diagram representing the system structure

Concurrency, Synchronization & ACID Properties

- **File Locking** – Prevents concurrent write operations, ensuring consistency across account, transaction, and loan records.
- **Shared Memory** – Tracks active user sessions and prevents multiple logins for the same account.
- **Session Handling** – Each client session is handled in a separate process to maintain isolation.
- **ACID Compliance:**
    - **Atomicity:** Transactions (deposit, withdrawal, loan approval) complete fully or not at all.

- **Consistency:** Validation and locking maintain consistent system state.
- **Isolation:** Concurrent users operate independently without interference.
- **Durability:** Changes are written immediately to files using `fsync()`.

Class Diagram

The UML class diagram illustrating relationships among **Customer**, **Employee**, **Manager**, **Loan**, **Admin**, **Transactions**, and **Feedback** modules:

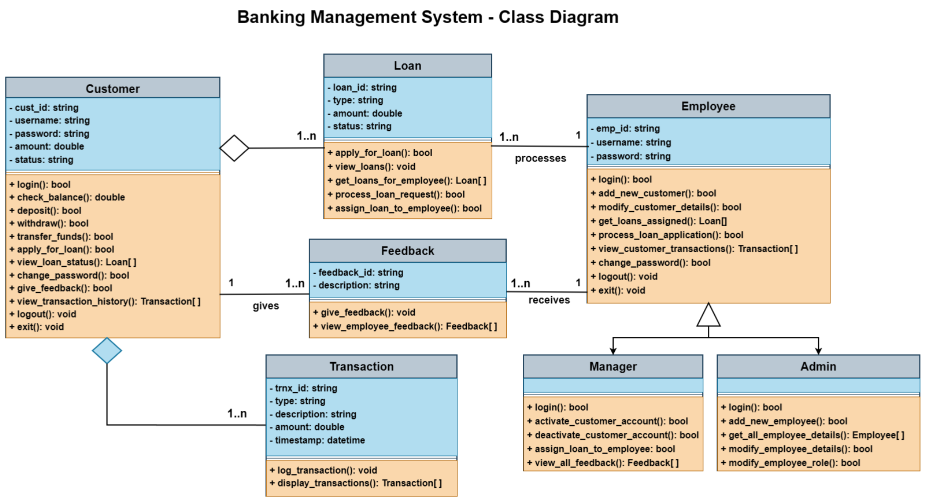
Banking Management System - Class Diagram
Customer
- cust_id: string
- username: string
- password: string
- amount: double
- status: string
+ login(): bool
+ check_balance(): double
+ deposit(): bool
+ withdraw(): bool
+ transfer_funds(): bool
+ apply_for_loan(): bool
+ view_loan_status(): Loan[ ]
+ change_password(): bool
+ give_feedback(): bool
+ view_transaction_history(): Transaction[ ]
+ logout(): void
+ exit(): void
Loan
- loan_id: string
- type: string
- amount: double
- status: string
+ apply_for_loan(): bool
+ view_loans(): void
+ get_loans_for_employee(): Loan[ ]
+ process_loan_request(): bool
+ assign_loan_to_employee(): bool
1..n
1..n
processes
1
Employee
- emp_id: string
- username: string
- password: string
+ login(): bool
+ add_new_customer(): bool
+ modify_customer_details(): bool
+ get_loans_assigned(): Loan[]
+ process_loan_application(): bool
+ view_customer_transactions(): Transaction[ ]
+ change_password(): bool
+ logout(): void
+ exit(): void
Feedback
- feedback_id: string
- description: string
+ give_feedback(): void
+ view_employee_feedback(): Feedback[ ]
1
1..n
gives
1..n
1
receives
Transaction
- trnx_id: string
- type: string
- description: string
- amount: double
- timestamp: datetime
+ log_transaction(): void
+ display_transactions(): Transaction[ ]
1..n
Manager
+ login(): bool
+ activate_customer_account(): bool
+ deactivate_customer_account(): bool
+ assign_loan_to_employee: bool
+ view_all_feedback(): Feedback[ ]
Admin
+ login(): bool
+ add_new_employee(): bool
+ get_all_employee_details(): Employee[ ]
+ modify_employee_details(): bool
+ modify_employee_role(): bool

Figure 1: Class Diagram