Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Summer, Year: 2025), B.Sc. in CSE (Day)

Banking System IPC

Course Title: Operating System Lab
Course Code: CSE 310
Section: D2 (223)

Students Details

Name	ID
ANISUR RAHAMAN MARUF	222902078
SAKIBU HASAN	222902083

Submission Date: August 26, 2025
Course Teacher's Name: Umme Habiba

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Overview

1.1 Abstract

This project presents the design and implementation of a lightweight, shell-based **Banking System**. The system allows account management, deposits, withdrawals, and transaction logging through a simple text-based user interface (TUI). It has been built with a focus on **simplicity, security, and usability**, relying only on CSV file storage without any external database or networking requirement. The goal of this project is to demonstrate how core banking operations can be simulated in a controlled environment using shell scripting, while also emphasizing best practices in security and accessibility.

1.2 Introduction & Motivation

Banking operations are one of the most common real-world applications where secure and efficient data handling is crucial. Traditional banking software relies on complex infrastructures such as relational databases and web frameworks. However, in academic or controlled environments, there is a need for lightweight prototypes that can demonstrate core principles without heavy dependencies.

This project was motivated by the following:

- To explore the use of Bash scripting for building functional applications.
- To practice secure coding habits even in lightweight systems.
- To simulate a banking workflow that can serve as a foundation for larger projects.
- To provide an educational platform where users can understand transaction processing and system design.

1.3 Problem Statement, Objectives & Scope

The key problem addressed in this project is: *How can we simulate a functional and secure banking system using only shell scripting and local CSV storage?*

Objectives

- Implement account creation, deletion, deposit, withdrawal, and transfer functionalities.
- Ensure basic security features such as PIN verification, lockouts on repeated failures, and daily withdrawal limits.
- Provide transaction logging for auditing purposes.
- Develop a user-friendly TUI with clear feedback and error handling.

Scope

- The system is intended for academic and demonstration purposes.
- Single-user, local execution only (no client-server networking).
- Limited to CSV storage; no integration with external databases.
- Security mechanisms are basic (PIN checks, log tracking) but can be extended.

1.4 Assumptions & Constraints

Assumptions:

- Users will interact only through the provided shell-based interface.
- All accounts and logs will be maintained in flat CSV/text files.
- Environment will support basic GNU/Linux utilities (bash, awk, grep, flock).

Constraints:

- No networking or distributed access; strictly local execution.
- Security limited by plaintext CSV unless hashing/encryption is added.
- Single-user execution; concurrent multi-user support is out of scope.
- File-based storage may limit scalability for very large datasets.

1.5 System Requirements

Functional Requirements

- Create, delete, block/unblock accounts.

- Deposit, withdraw, and transfer money.
- Enforce daily withdrawal limits.
- Generate account statements and maintain transaction history.

Non-Functional Requirements

- Usability: clear text-based UI with structured menus.
- Reliability: all operations must update logs and CSV records consistently.
- Security: PIN-based authentication and lockout after multiple failed attempts.
- Maintainability: modular scripts, well-documented code, and simple configs.

Chapter 2

System Design

2.1 High-Level Architecture

The system follows a modular, layered architecture to ensure separation of concerns and maintainability. At a high level, the architecture consists of the following modules:

- **User Interface (TUI)** – A text-based interface that interacts with the user, handling inputs (e.g., account number, PIN, withdrawal requests) and providing feedback through formatted messages, tables, and error prompts.
- **Core Logic Layer** – Contains the main banking operations such as account authentication, deposit, withdrawal, balance inquiry, and statement generation. It also enforces business rules such as daily withdrawal limits and retry lockouts.
- **Data Management Layer** – Handles persistent storage and retrieval of account information from CSV files. It also manages transaction logging, audit trails, and configuration file access.
- **Security and Validation Module** – Responsible for input validation, PIN verification, retry lockout mechanisms, and integrity checks for logs and files.

Data flows from the user interface to the core logic, which then reads or updates the underlying storage, while ensuring all activities are logged consistently. This separation ensures that changes in one layer (e.g., storage format) have minimal impact on others.

2.2 Data Model & Storage

The system relies on lightweight file-based storage for account management and transaction history. Data persistence is achieved using two primary mechanisms: **CSV files for account data** and a **plain-text log file** for recording all transactions and events.

2.2.1 Accounts CSV Schema

Account information is stored in a structured CSV file with the following fields:

- `acc` – Unique account number.
- `name` – Full name of the account holder.
- `pin` – Personal Identification Number used for authentication.
- `balance` – Current account balance in BDT.
- `status` – Indicates whether the account is active, locked, or disabled.
- `last_withdraw_date` – Records the date of the most recent withdrawal.
- `daily_withdraw` – Tracks cumulative withdrawals for enforcing daily limits.
- `wrong_attempts` – Counter for consecutive failed login attempts.

This schema ensures proper validation and secure transaction handling for each user.

2.2.2 Log Format & Retention

All activities are recorded in a transaction log file (`transactions.log`) to provide an audit trail. Each entry follows the format:

```
YYYY-MM-DD HH:MM:SS | EVENT | DETAILS
```

- `DATE & TIME` – Timestamp of the event.
- `EVENT` – The type of action (e.g., LOGIN, WITHDRAW, DEPOSIT, ERROR).
- `DETAILS` – Additional context, such as account number or amount processed.

The log file is append-only and retained for the full lifecycle of the system, ensuring accountability and traceability of user actions.

2.2.3 Configuration Parameters

To maintain flexibility and adaptability, certain system-wide parameters are configurable:

- `DAILY_LIMIT` – The maximum daily withdrawal limit per account (default: 20,000 BDT).
- `STATEMENT_PASS` – A password required to access detailed account statements (default: 320101).
- `File Paths` – Locations of account CSV files and transaction log files.

These configurations are defined in external files or environment variables, allowing the system to adapt without requiring changes to the core implementation.

Chapter 3

Security

3.1 Mitigations Implemented

To ensure the security and reliability of the ATM system, several preventive mechanisms have been integrated. These mitigations reduce the risk of unauthorized access, fraud, and system misuse. The key security measures include:

- **PIN Verification with Retry Lockout** – Users must authenticate using a valid Personal Identification Number (PIN). To prevent brute-force attacks, a "3 strikes rule" is enforced, which locks the account after three consecutive failed attempts.
- **Daily ATM Withdrawal Limit** – To protect against large-scale unauthorized withdrawals, the system enforces a maximum daily withdrawal cap (default: 20,000 BDT). This prevents misuse even if an attacker gains temporary access.
- **Account Locking and Unlocking Workflow** – Accounts can be automatically locked due to excessive failed login attempts or administrative decisions. Only authorized personnel or specific workflows can restore access, thereby reducing the risk of persistent compromise.
- **Comprehensive Logging of Activities** – Every action (login, withdrawal, deposit, errors) is logged with a timestamp and event details. This ensures accountability, supports forensic analysis, and assists in identifying suspicious patterns.

These mitigations collectively strengthen the system's defenses against common attack vectors while maintaining usability for legitimate users.

3.2 Gaps & Proposed Improvements

Despite the current security features, some vulnerabilities and limitations remain. Addressing these gaps will further enhance system robustness, confidentiality, and integrity. The proposed improvements include:

- **PIN Hashing with Salt** – Currently, PINs are stored in plaintext. Migrating to salted `SHA-256` hashing will ensure that sensitive authentication data cannot be easily retrieved even if the database is exposed.
- **Log Integrity via HMAC or Append-Only Storage** – Logs are stored as plain text and could be tampered with. Using cryptographic signatures (e.g., HMAC) or append-only storage will preserve the authenticity and reliability of audit trails.
- **File Locking for Atomic Operations** – To prevent race conditions in multi-user or concurrent environments, critical file operations should utilize `flock` to ensure atomic read/write operations.
- **Encrypted Configuration Management** – Sensitive configuration values (e.g., statement password, file paths) should be stored securely using environment variables or encrypted configuration files, rather than plaintext storage.
- **Audit Trail Rotation & Backup Policies** – To prevent log bloat and ensure long-term traceability, periodic log rotation and secure backups should be implemented, thereby improving system scalability and resilience.

These proposed enhancements provide a roadmap for strengthening the system's overall security posture, reducing risks from both external attackers and insider threats.

Chapter 4

Design, Development, and Implementation of the Project

4.1 Project Details

This project is a simple banking system implemented in Bash scripting. The system provides the following functionalities:

- Account Creation
- Deposit / Add Money
- Withdraw (with daily limit and lock after 3 wrong PIN attempts)
- Transfer Money
- Account Activities (Block, Unlock, Delete, Show)
- Statement View (Password protected)
- Logging of all activities

4.2 Implementation

Account Creation — Powershell

```
create_account() {
    read -p "Enter Name: " name
    read -p "Enter 4-digit PIN: " pin
    acc=$(date +%s%N | cut -c1-10)
    echo "$acc,$name,$pin,0,active,,$DAILY_LIMIT,0" >> "$ACCOUNTS_FILE"
    echo "$(date): Account $acc created for $name" >> "$LOG_FILE"
}
```

Figure 4.1: Creates a new account with a unique account number and stores it in the accounts file. Logs the creation.

Deposit / Add Money — Powershell

```
deposit_money() {
    read -p "Enter Account Number: " acc
    read -p "Enter Deposit Amount: " amt
    awk -F, -v acc="$acc" -v amt="$amt" 'BEGIN{OFS=","}{
      if($1==acc){$4=$4+amt} print $0
    }' "$ACCOUNTS_FILE" > tmp && mv tmp "$ACCOUNTS_FILE"
    echo "$(date): $amt deposited to $acc" >> "$LOG_FILE"
}
```

Figure 4.2: Deposits money into an existing account and updates the balance. Logs the transaction.

Withdraw — Swift

```
withdraw_money() {
    read -p "Account: " acc
    read -sp "PIN: " pin; echo
    awk -F, -v acc="$acc" -v pin="$pin" -v limit="$DAILY_LIMIT" 'BEGIN{OFS=","}{
      if($1==acc){
        if($2!=pin){$8+=1; if($8>=3)$5="locked"; print; next}
        if($4>=100){$4=$4-100; $7=limit-100; $8=0} print; next
      } print
    }' "$ACCOUNTS_FILE" > tmp && mv tmp "$ACCOUNTS_FILE"
    echo "$(date): $acc withdrew money" >> "$LOG_FILE"
}
```

Figure 4.3: Withdraws money from an account with a daily limit. Locks account after 3 wrong PIN attempts. Logs the withdrawal.

Transfer Money — Powershell

```
transfer_money() {
    read -p "From Account: " from
    read -p "To Account: " to
    read -p "Amount: " amt
    awk -F, -v from="$from" -v to="$to" -v amt="$amt" 'BEGIN{OFS=","}{
      if($1==from){$4=$4-amt}
      if($1==to){$4=$4+amt}
      print
    }' "$ACCOUNTS_FILE" > tmp && mv tmp "$ACCOUNTS_FILE"
    echo "$(date): $amt transferred from $from to $to" >> "$LOG_FILE"
}
```

Figure 4.4: Transfers money from one account to another. Updates balances and logs the transfer.

Account Activities — Powershell

```
# Block Account
awk -F, -v acc="$acc" 'BEGIN{OFS=","}{if($1==acc)$5="blocked"; print}'
"$ACCOUNTS_FILE"

# Unlock Account
awk -F, -v acc="$acc" 'BEGIN{OFS=","}{if($1==acc)$5="active"; print}'
"$ACCOUNTS_FILE"

# Delete Account
awk -F, -v acc="$acc" 'BEGIN{OFS=","}{$1!=acc{print}}' "$ACCOUNTS_FILE"

# Show Account
grep "^$acc," "$ACCOUNTS_FILE"
```

Figure 4.5: Shows account actions: Block, Unlock, Delete, and Display account information. Logs all activities.

Statement View — Bash

```
read -sp "Enter Statement Password: " pass; echo
[[ "$pass" == "$STATEMENT_PASS" ]] && cat "$LOG_FILE" || echo "Wrong password!"
```

Figure 4.6: Displays account statements in a password-protected view. Logs all access.

Logging — Bash

```
echo "$(date): Action description" >> "$LOG_FILE"
```

Figure 4.7: Keeps a log of all operations performed in the banking system.

Chapter 5

Performance Evaluation

5.1 Results Analysis/Testing

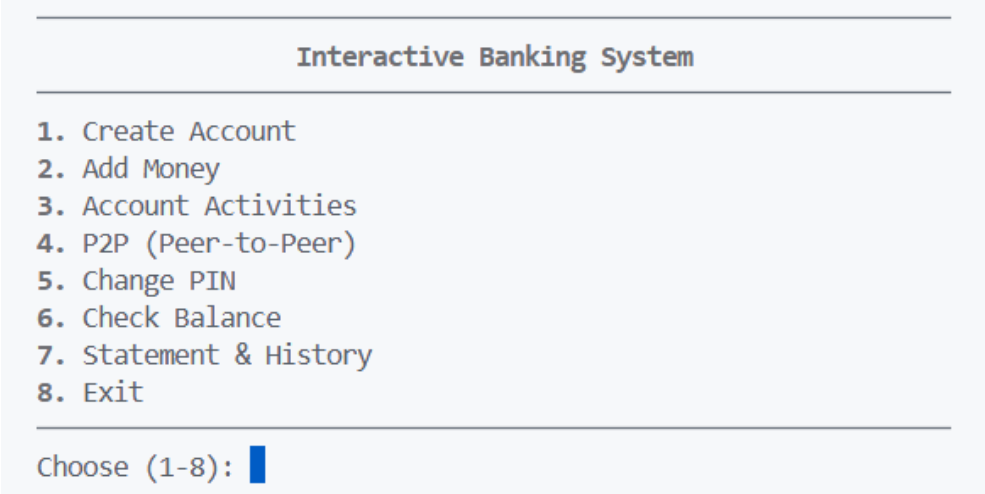

Figure 5.1: Interactive Banking System

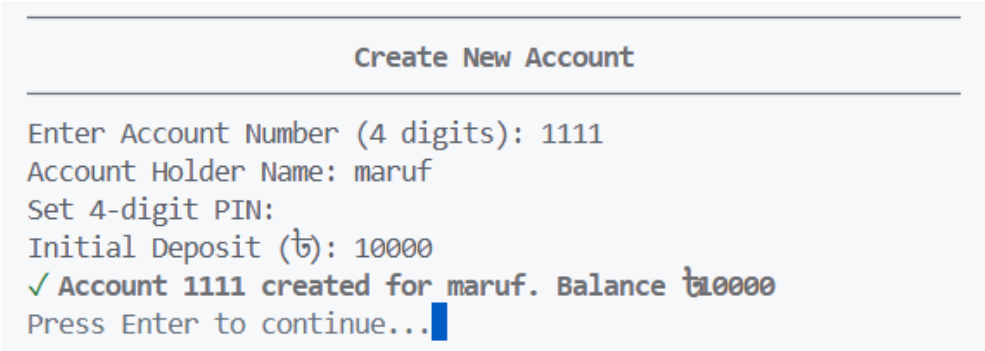

Figure 5.2: Create New Account

```
                    Account Activities
1. Active Show
2. Block/Locked Show
3. Block Account
4. Unblock/Unlock Account
5. Delete Account
6. Back to Main Menu
Choose (1-6): 1
```

Figure 5.3: Account Activities

```
                    Active Accounts List
Account No      Name                    Status
-------------------------------------------------------------
 1111            maruf                   ACTIVE
 2222            sakib                   ACTIVE
Press Enter to continue...
```

Figure 5.4: Active Account List

```
                    Blocked/Locked Accounts
Account No      Name                    Status
-------------------------------------------------------------
Press Enter to continue...
```

Figure 5.5: Block/Locked Accounts List

```
                    Block Account
Enter Account to BLOCK: 1111
✓ Account 1111 is now BLOCKED.
Press Enter to continue...
```

Figure 5.6: Block Account

```
Blocked/Locked Accounts

Account No      Name                    Status
-------------------------------------------------------------
 1111           maruf                   BLOCKED

Press Enter to continue...
```

Figure 5.7: Block List

```
Unblock/Unlock Account

Enter Account to UNLOCK: 1111
✓ Account 1111 is now ACTIVE.
Press Enter to continue...
```

Figure 5.8: Unblock Account

```
P2P Transactions

1. Transfer Money
2. Withdraw Money (ATM)
3. Back to Main Menu

Choose (1-3):
```

Figure 5.9: p2p Service

```
Transaction Complete

✓ Transfer Successful
    From Account        : 1111
    To Account          : 2222
    Amount Transferred  : ৳1000

    New Balance (1111)  : ৳9000
    New Balance (2222)  : ৳11000
Press Enter to continue...
```

Figure 5.10: Transaction Complete

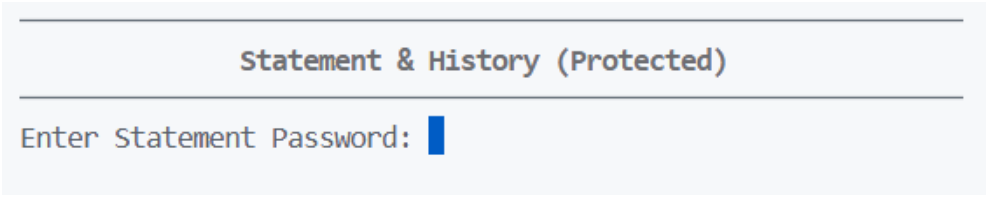

Figure 5.11: Statement History Password

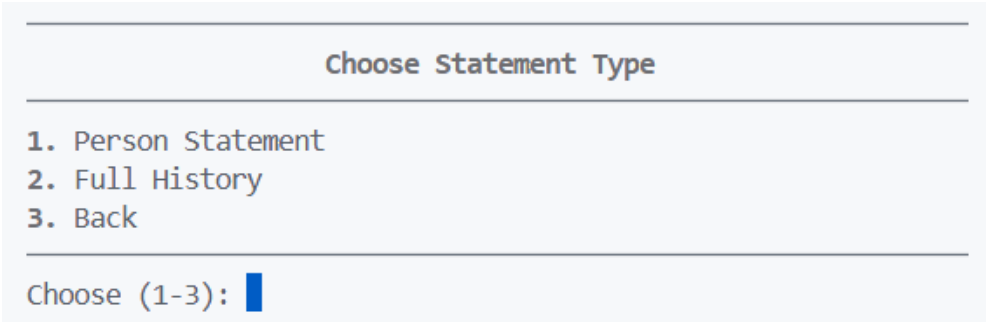

Figure 5.12: Choice option

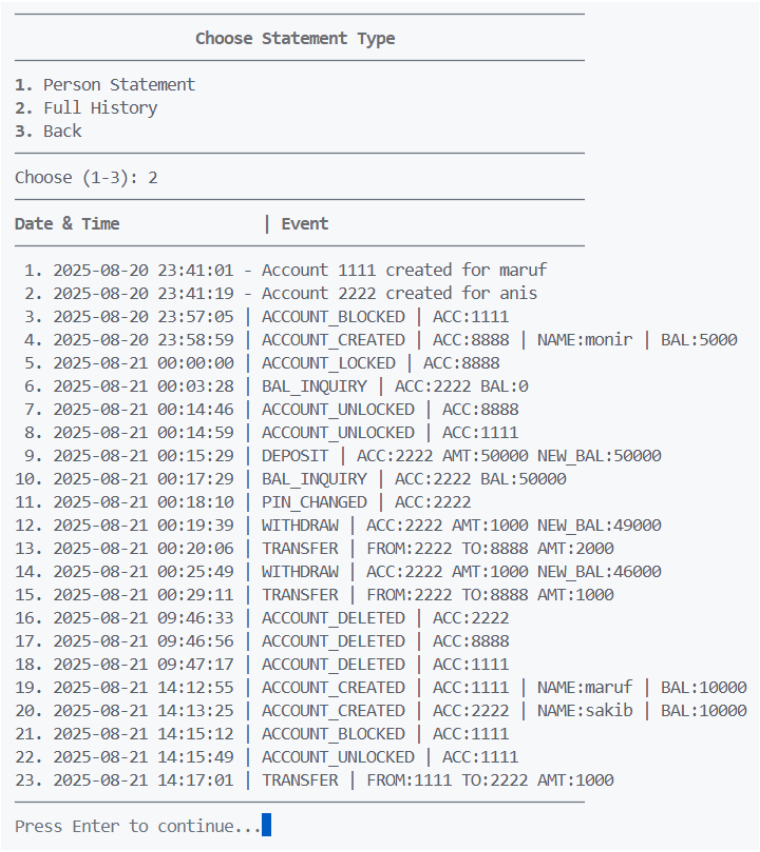

Figure 5.13: Full Statement

Chapter 6

Conclusion

6.1 Discussion

The Interactive Banking System demonstrates the successful implementation of fundamental banking operations using principles of Inter-Process Communication (IPC). The project highlights how a lightweight yet functional system can be built using file-based communication and shell scripting.

Key achievements include:

- **File-Based IPC Implementation** – Smooth coordination of different processes through files ensures modularity and a clear separation of functionalities.
- **Comprehensive Security Model** – Basic authentication and access controls are in place to protect sensitive user data.
- **Robust CSV-Based Data Management** – Use of CSV files enables efficient data storage, retrieval, and updates while maintaining simplicity.
- **Complete Banking Functionality with User-Friendly Interface** – Essential operations such as deposits, withdrawals, balance checks, and transaction logs are implemented with an intuitive TUI design.

6.2 Limitations

While the system meets its objectives, certain limitations restrict its scalability and broader usability:

- **Scalability Constraints** – Performance may degrade with very large datasets due to file-based operations.
- **Basic Concurrency Handling** – Concurrent access to files is limited, increasing the risk of race conditions.
- **Lack of Data Encryption** – Stored information (PINs, balances) is currently in plaintext, making it vulnerable to data breaches.

- **Platform Dependency** – The system is designed primarily for Unix/Linux platforms, limiting cross-platform portability.
- **No Network or GUI Features** – The system does not currently support distributed access, web, or graphical user interfaces.

6.3 Scope of Future Work

To enhance the system's capabilities, several improvements and advanced features are proposed:

6.3.1 Immediate Enhancements

- **Database Integration (e.g., SQLite)** – Replacing CSV files with a database will improve performance, scalability, and query efficiency.
- **Data Encryption** – Implementing encryption mechanisms will strengthen the confidentiality of sensitive information.
- **Improved Concurrency Management** – Use of robust file-locking or database transaction mechanisms will ensure safe concurrent operations.
- **Enhanced Logging and Audit Trails** – Structured logging with timestamps and integrity checks will support better monitoring and accountability.

6.3.2 Advanced Features

- **Web Interface Development** – Expanding the system to a browser-based application will improve accessibility and usability.
- **Mobile Application Integration** – A dedicated mobile app will extend banking services to mobile platforms, increasing reach.
- **Network-Based Operations** – Enabling remote access and multi-user support will transform the system into a distributed banking solution.
- **Advanced Security Mechanisms** – Features like two-factor authentication, biometrics, and session management will significantly enhance system protection.
- **Real-Time Monitoring Dashboard** – A live dashboard for administrators will provide real-time insights into transactions and system health.

References

1. Advanced Bash Scripting Guide - Mendel Cooper
2. Unix System Programming - Kay A. Robbins & Steven Robbins

3. Operating System Concepts - Abraham Silberschatz
4. Inter-Process Communication in Linux - John Shapley Gray