Part 2 report

Environment Setup

Prerequisites:

1. **Clang++ (C++ Compiler)**: The program uses the Clang compiler to compile C++ code. Ensure it's installed.
2. **C++14 Support**: The program is written in C++14, so the compiler needs to support C++14 (Clang++ does by default).

Compiling the Program

The program is written in C++ and consists of a single file (server.cpp). To compile it with Clang++, run the following command in your terminal:

```
clang++ -std=c++14 -o server server.cpp
```

or

```
make
```

3. Executing the Program

Once compiled, you can run the executable by executing the following command:

```
./server <portnum>
```

Program Flow

1. **Server Initialization**

- The server initializes a socket to listen for incoming client connections.
- A thread pool is created with a fixed number of worker threads, which remain idle until assigned a task.

2. **Client Connection**

- When a client connects, the server accepts the connection and enqueues the client socket into the thread pool's task queue.
- A worker thread dequeues the task and begins processing the client request.

3. **Command Handling**

- The server processes the following commands:

Registration: REGISTER#<username>

- Registers a new user. The server verifies username uniqueness and stores the user in memory.

Login: <username>#<port_number>

- Authenticates the user and marks them as online.
- **Balance Inquiry:** Sends the current account balance of the user.
- **Money Transfer:** <sender>#<amount>#<receiver>

Transfers funds between users, updating balances atomically.

- **List Online Users:** Lists all currently logged-in users.

4. **Response and Persistence**

- The server responds to each request with a success message or descriptive error. User data is managed in memory, with appropriate synchronization to ensure thread safety.

5. **Client Disconnection**

- When a client disconnects, the server ensures that their status is updated (e.g., marking them offline).

Error Handling

To ensure robustness, the server handles various error scenarios gracefully. Input syntax errors, such as malformed commands, are detected early. If a client sends an invalid command like REGISTERuser, the server responds with a clear message: ERROR: Invalid command syntax.

Synchronization issues are also carefully managed. Mutex locks guard shared resources like the user database, preventing race conditions. If a thread cannot acquire a lock, the server logs the issue and retries briefly to avoid deadlocks,

ensuring uninterrupted service. Runtime exceptions, such as sudden client disconnections, are caught and handled to maintain stability.

Concurrency Management

The thread pool efficiently manages tasks by distributing client requests among a fixed number of threads, avoiding excessive resource consumption. Critical sections, such as updating account balances during money transfers, are executed atomically using mutex locks. This guarantees consistency even under heavy load, such as when multiple users attempt simultaneous operations.

GUI

- **Green:** Successful operations or connections.
- **Yellow:** Warnings or ongoing transactions.
- **Red:** Errors or client disconnections.