CMSC-621, PROJECT 1 (GRINGOTTS BANK)
SHREYA DATE (PQ56297)

<u>STEPS TO RUN server and client</u>

Server:
1. Open terminal 1
2. Traverse to src dir
3. Create server and client using make (make has 2 targets)
4. Run server using ./server <port number>

Client:
1. Open terminal 2
2. Traverse to src dir
3. Create client using make
4. For Experiment 1:
   Run client using ./client <port number>
   For Experiment 2:
   Run multiple clients (10) using the shell script provided.
   Command: ./multiple_clients.sh

Experiments:

1. Single process, multiple thread scenario:
   The client process internally spawns thread per transaction.
   Multiple client requests would be sent to the server.
   Here, there's a single process having multiple client threads

2. Multiple processes, multiple threads scenario:
   Using the multiple_clients.sh file, multiple client processes are created.
   Each process would spawn n threads if there are n transactions to be done.
   Though the bash script has process creation in a for loop, multiple threads are created per process and some threads from every process would run simultaneously.

<u>CODE DESIGN</u>

The project GringottsBank contains the following files:

1. GringottsServer: contains the server class having methods for handling server logic and accepting multiple client requests on separate threads

2. GringottsClient : contains the Client class having logic for client to create a client thread per transaction

3. Socket: contains wrappers over the socket system calls

4. SocketDef: contains structure for socket data

5. GringottsDef: contains structure for maintaining client account balance

6. ServerMain.cpp : This is the process which kicks off server creation by using methods exposed by the GringottsServer class

7. ClientMain.cpp : This is the process which kicks off client creation by using methods exposed by the GringottsClient class

**Note that because of heavy modularization, code workflow is pretty clear and hence not commented much.

CODE WORKFLOW

1. Server creates a listener socket
2. It keeps accepting connections from clients in a loop
3. for every connection accepted, server reads the string (transaction data) sent by the client.
4. A thread is spawned for each new client
5. In the thread handler, the actual transaction takes place to modify client account data which is a single shared memory between the multiple client threads
6. Synchronization is achieved by putting pthread_mutex_locks
7. A global waiting queue is maintained as a vector where a new client request is added
8. Once the transaction is complete, the client transaction is removed from the queue (FIFO manner)
9. Logs and updated account balance is printed after every transaction

In the Client,
1. Transactions.txt is read and a client thread is created for every transaction
2. every transaction is handled by a single client
3. Every client thread sends transaction data to the server