CS 211: C and Systems Programming, Spring 2019
Programming Assignment 3: Multithreaded Online Shopping

1 Introduction

For this assignment, you will write programs for a multithreaded shopping cart simulation. This will give you an opportunity to exercise mutexes and thread coordination. You will write client/server programs with a single server that supports multiple clients communicating through TCP/IP network connections. Having to support multiple concurrent client-service threads in the server will require the use of mutexes to protect and manage shared data structures.

2 Server: Multithreaded Shopping Server Program

Your server process should have a single session-acceptor thread. The session-acceptor thread will accept incoming client connections from separate client processes. For each new connection, the session-acceptor thread should spawn a separate client-service thread that communicates exclusively with the connected client. You may have more than one client connecting to the server concurrently, so there may be multiple client-service threads running concurrently in the same server process.

The shopping server process will maintain a simple shopping service capable of supporting multiple clients. There will be a maximum of 20 shopping carts, one for each potential client. Initially your shopping service will have no clients, but clients may shop as needed. Information for each customer will consist of:

- `Customer name (a string up to 40 characters long)`
- `Current address line 1 (a string up to 40 characters long)`
- `Current address line 2 (a string up to 40 characters long)`
- `Current city (a string up to 40 characters long)`
- `Current state (a two-character string)`
- `Current zip code (a five-character string)`
- `Current balance (a floating-point number)`
- `In-session flag (a boolean flag indicating whether or not the customer is currently being serviced)`

The server will handle each client in a separate client-service thread. Keep in mind that any client can shop for any customer at any time, so adding customers to your shopping service must be a mutex-protected operation.

3 Client: Connecting to the server

The client program requires the name of the machine running the server process as a command-line argument. The machine running the server may or may not be the same machine running the client processes. On invocation, the client process must make repeated attempts to connect to the server. Once connected, the client process will prompt for commands. The syntax and meaning of each command is specified in the next section.

Your client implementation would have two threads: a command-input thread to read commands from the user and send them to the server, and a response-output thread to read messages from the server and send them to the user. Having two threads allows the server to proactively and asynchronously send messages to the client even while the client is waiting for commands from the user.

4 Command Syntax

The command syntax allows the user to register new customers, to start sessions to serve specific customers, and to exit the client process altogether. Here is the command syntax:

- **register** `customername`
- **shop** `customername`
- **add** `itemname`
- **remove** `itemname`
- **show**
- **empty**
- **placeorder**
- **exit**

The client process will send commands to the server, and the server will send responses back to the client. The server will send back error or confirmation messages for each command.

The **register** command registers a new customer with the server. It is an error if the server already has a full list of customers, or if a customer with the specified name already exists. A client in a customer session cannot register new customers, but another client who is not in a customer session can register new customers. The name specified uniquely identifies the customer. A customer name will be at most 40 characters. The initial shopping cart of a newly registered customer is empty. It is only possible to register one new customer at a time, no matter how many clients are currently connected to the server.

The **shop** command starts a session for a specific customer. The **add**, **remove**, **show**, **empty** and **placeorder** commands are only valid in a customer session. It is not possible to start more than one customer session for any single client, although there can be concurrent customer sessions for different customers for different clients. Under no circumstances can there be concurrent customer sessions for the same customer. It is possible to have any number of sequential client sessions.

The **add** and **remove** commands add and removes items from a client cart. Either command should complain if the client is not in a customer session. A **remove** is invalid if the customer does not have the specified item in their cart.

The **show** command lists the items in the cart.

The **finish** command removes all items from the cart and ends the customer session.

The **placeorder** command prints a receipt for all items in the cart, including a total price. This command also finishes the customer session. Once the customer session is ended, it is possible to register new customers or start a new customer session.

The **exit** command disconnects the client from the server and ends the client process. The server process should continue execution.

5 Deadlocks and Race Conditions

There should be NO DEADLOCKS and NO RACE CONDITIONS in your code.

6 Program Start-up

The client and server programs can be invoked in any order. Client processes that cannot find the server should repeatedly try to connect every 3 seconds. The client must specify the name of the machine where the client expects to find the server process as a command-line argument.

The server takes no command line arguments.

7 Implementation

Minimally, your code should produce the following messages:

- Client announces completion of connection to server.
- Server announces acceptance of connection from client.
- Client disconnects (or is disconnected) from the server.
- Server disconnects from a client.
- Client displays error messages generated by the server.
- Client displays informational messages from the server.
- Client displays successful command completion messages generated by the server.

8 Program Termination

The server can be shut down by SIGINT, no signal handler necessary. The client(s) should shut down when the server shuts down.

9 Extra Credit

It should not be possible to have concurrent sessions for the same customer, which requires a mutex lock for each customer. That's not the extra credit part. This is the extra credit part: Instead of silently blocking while trying to start a customer session, the server could try to lock the mutex for the customer every 2 seconds and if the locking attempt fails, the server could send a "waiting to start customer session for customer so-and-so" message to the appropriate client process.

10 What to turn in

A tarred file named `pa3.tar` that contains a directory called pa3 with the following files in it:

- A `readme.pdf` file documenting your design **paying particular attention to the thread synchronization requirements of your application**.
- A file called server-testcases.txt that contains a thorough set of test cases for your code, including inputs and expected outputs.
- All source code including both implementation (.c) and interface(.h) files.
- A makefile for producing the executable program files.

Your grade will be based on:

- Correctness (how well your code is working, including avoidance of deadlocks.
- Efficiency (avoidance of recomputation of the same results).
- Good design (how well written your design document and code are, including modularity and comments).