Multithreaded Echo Client/Server

TJ Maynes and Chris Migut

Abstract—The purpose of this project is to build an echo client/server using multithreading, semaphores, and sockets.

Index Terms—Operating Systems, Shared Memory, Multithreading, pthreads, sockets, C, USF

1 Introduction

In this assignment we are building an echo client and server to demonstrate the power of multithreading, semaphores, and sockets. The following describes the overall setup of our echo client/server: When the client connects with the server, the server will create a new thread for every connection established; The new thread will send a message to the client asking to input a message (up to 10 characters longs); Next, the client will send back the message to the server. The server will read the received message and reverse the message. Finally, the server will send back to the client, the reversed version of the received message and the number of current connections on the server.

We will be writing the client and server in the C Programming Language. The client will be using using sockets to establish a connection to the server. The server will be using sockets to send data back and forth between the clients connected, pthreads to create a new thread for each connection, and semaphores to keep a running count of the number of connections.

1.1 Motivation

The motivation for this project was to better understand computer networks and multithreading. How can a server process several clients at once? Including sending requests, making computations, etc. How are resources from the server being assigned to each client? Also, how do you setup the server so that there are no conflicts with other clients connected to the server. Once a client has diconnected from the server, how are those resources are freed and recollected by the server. Before this project, most of these operations were not fully understood how this protection could be established in a real program.

1.2 Multithreading w/ pthreads

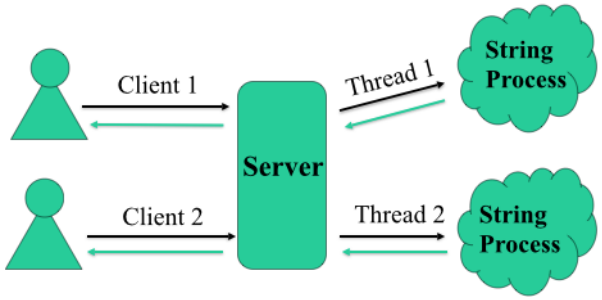

Fig. 1. Mulithreading sockets for Client/Server requests.

In specific situations, a programmer may want to write programs that do multiple tasks at once (as opposed to one at a time). For instance, an echo server needs to be set up as a multithreaded application since incoming requests from client (to server) needs have its own thread to service the request. The benefits of multithreaded programming being responsiveness, resource sharing, economy, and scalability. Pthreads refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation and synchronization, which are found in the pthread.h library. Figure 1 above shows how a thread is created for each client connection.

1.3 Semaphores

A semaphore is a integer variable that is used for controlling processes via two standard atomic operations: wait() and signal(). All modifications to the integer value of the semaphore in the wait() and signal() operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously modify the same semaphore value.

1.4 Client

Fig. 2. Client connecting to server.

The client is setup to hold a single connection with the server running on a specfic port number and internet protocol address (IP address). In order to run the client approriately, the server needs to already be running on the same port number and IP address. When running the client, we setup a Transmission Control Protocol, or TCP, socket and make a connection with that socket. On connection with the server, we will receive (using the recv function) a message requesting for an input of up to 10 characters. This message will be sent (using the send function) back to the server to be processed Soon after, we should receive (using the recv function) a two messages back from the server: the sent message returned in reversed and the current number of client connections on the server. Finally, we close the TCP socket to the server and client program exits.Figure 2 above shows the client connected and disconnected to the server.

1.5 Server

The server is setup to concurrently accept up to two connections at a time. This means we can have up to two clients running communications between the server at a time. The

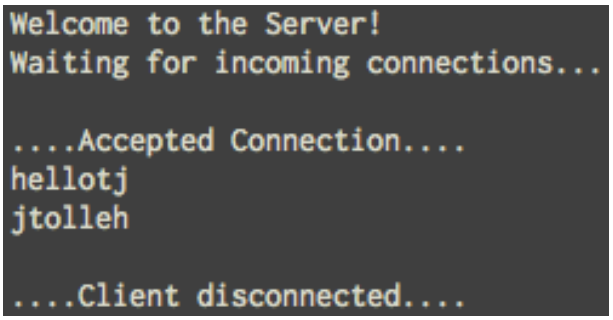

Fig. 3. Running the server.

following is how the server is setup to connect to multiple clients: First, we create a TCP socket; Second, we bind the socket to a local address so that other sockets can be connected to it; Third, we queue a limit to the number of incoming connections using the listen function; Lastly, we run a while-loop for accepting the incoming clients connecting to the server. In the while-loop, we increment the number of client connections in the shared memory buffer. We protect the shared memory buffer by placing (producer) semaphores around the critical section. Figure 3 above shows what a client has sent, “hellotj”, and the reverse, “jtolleh” it will send back in response.

For each incoming connection, we will spin up a new thread that will process the data being sent between client and server. When we are in the threaded function, called “handler”, we will send a message to the client requesting the client to send back a message (up to 10 characters). Once, the server has received the client’s message, the handler function will reverse the message and send the reversed message back to the client along with a message containing the amount of current connections on the server. Finally, the handler function will exit via pthread_exit.

1.6 Analysis

A majority of the time spent debugging was solving the following issues: We ran into issues with the server not accepting client connections, the client not receiving (and displaying) the right data from the server, and having issues getting the correct number of connected clients. Also, to clarify, the server’s final message to the client is the current number of

clients connected at the start of the client program. Once those issues were solved, the server was able to handle the different number of clients. With each of client assigned a thread for processing and collected once the clients request has finished by the server. To ensure that messages are passed correctly to the different parties(server-client), the Transmission Control Protocol (TCP) was used for this quality of service. This protocol provides error handling if a packet used to transmit a message happens to be corrupted. The protocol will ask the sender to retransmit that packet to the receiver. This will ensure that the message will be delivered correctly. This server-client infrastructure can serve as a blueprint to include extra features for a later date. For example, TJ wants to include a hashing function that could hash the client's message. This could be used to hash passwords and save those into a database of a user account into a program. This would add protection to those client accounts if that database happen to be broken into by an attacker. Overall, this has been a great learning experience for the both of us in learning networks with a practical use of multithreading and semaphores.

REFERENCES

[1] Silberschatz, Galvin, and Gagne *Operating System Concepts*, 8th ed. John Wiley and Sons.