Bootcamp Week 2 : Simple echo client/server with multi-threading

This week, we will build a simple echo client-server application, with multi-threading at the server. Please work through the following tasks in order.

Single-threaded echo client and server

Your first task is to write simple C/C++ client-server programs using the socket API. In this exercise, the client should send a message to the server, and the server should simply echo it back to the client. That is, the server must send the same message back to the client. The server should keep doing this as long as the client is connected to it. It is enough if your programs work for a single client connected to a server for now.

Below are two video lectures to understand the concept of socket programming required to solve this exercise (you can watch either one):

Network I/O using sockets video lecture 1

Network I/O using sockets video lecture 2

Now, consider the simple client and server programs provided to you in this repository. You can use them as a guide to write your own code. The server program given to you takes one command line argument: the port number on which to listen. The client program takes two arguments: the server hostname and port number. You can give `localhost` as the hostname if you are running the client and server on the same machine. Once the client and server are connected to each other, the client sends a message to the server and gets a reply back.

Here is the output from these simple programs given to you (the client and server programs must be run in separate terminals):

```
$ gcc server.c -o server
$ ./server 5000
Here is the message: Hello
```

```
$ gcc client.c -o client
$ ./client localhost 5000
Please enter the message: Hello
I got your message
```

Please understand these sample programs and all the socket-related system calls completely. Also note the use of functions like `htons` to convert from host order to network order when communicating data over the network. Once you understand these examples, use this code as a template to write your own echo client and server in C or C++. Unlike this sample server, your server must echo back exactly the message received from the client. Also, it must repeatedly read data from the client and echo it back, and not just quit after one message exchange.

Complete writing this simple echo client and server, and test it thoroughly, before you proceed to the next exercise.

Other additional references to understand socket programming:

Beej’s Guide to Network Programming

Examples of socket programs in the textbook Peterson and Davie, Sec 1.4

Learning Socket Programming in cpp

Socket Programming

Multi-threaded server

In the server program written by you so far, the main server process is accepting the client connection and also serving it (by reading data and writing back a reply). Next, you will modify your server to handle the client processing in a separate thread. Once the server accepts a new connection, it will create a new worker thread, and it will pass the client's socket file descriptor (returned by accept) to this new thread. This thread will then read and write messages from/to the client.

To get started, here are some video lectures on threads: Threads video lecture 1, Threads video lecture 2

We will use the pthread library available in C/C++ to create threads. This document has a detailed explanation of the pthreads library and its functions: introduction to Pthread API. Sections 27.1 and 27.2 explain thread creation.

You can include the pthreads library in your programs by including this header file:

`#include <pthread.h>`

When writing code using this library, you must use the `-lpthread` flag to compile your code.

The pthread library has several useful functions. You can create a thread using the pthread_create() function present in this library. When you create a worker thread to handle a client request at your server, you must pass the accepted client file descriptor as an argument to the thread function, so that it can read and write from the assigned client. Understand how arguments are passed to threads, and be careful with pointers and casting. Here is sample code that creates a thread and passes it an argument:

```
void *start_function(void *arg) {
  int my_arg = *((int *) arg);
  // ...thread processing...
}

int main(int argc, char *argv[]) {
  int thread_arg = 0;
  pthread_t thread_id;

  pthread_create(&thread_id, NULL, start_function, &thread_arg);

  // ...more code...
}
```

Change your echo server to handle the accepted client in a separate thread as described above. The created thread will focus on communicating with the client given to it as an argument at creation time, while the main server thread can go back to accepting new connections. In this way, your server can perform the echo service with multiple clients at the same time.

You can check that your server is handling a small number of clients (say 5 or 10) at the same time by opening separate terminals, and connecting multiple clients to the server from the different terminals. You should find that the server is correctly able to echo back the messages received from the multiple clients.

Submission

Push your echo client and multi-threaded echo server client developed this week into your GitHub repository for the bootcamp.