CSC 427/527 LAB: SOCKET PROGRAMMING IN C/C++

GODFREY MUGANDA

1. Example TCP Server

The following is an example of a networked TCP server and associated client. The server opens a passive socket and listens for connection requests from clients. This is a very simple server: it accepts at most one connection request. Once a client connects, the server sends the message `"Hello there!"` to the client and shuts down.

The client is not expected to send anything to the server. It simply connects, reads the server's "reply," and prints it out.

```
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream>
#include <stdio.h>
using namespace std;

int main(int argc, char** argv)
{
   int server_sock;
   int client_sock;
   int error;

   //create the server socket
   server_sock = socket(PF_INET, SOCK_STREAM, 0);

   // Need an address to bind the serversocket to
   struct sockaddr_in server_address;
   memset(&server_address, 0, sizeof(server_address));
   server_address.sin_addr.s_addr = htonl(INADDR_ANY);
   server_address.sin_family = PF_INET;
   server_address.sin_port = htons(2222);

   // bind the server socket to a local address
   error = bind(server_sock,
               (struct sockaddr *)&server_address,
               sizeof(server_address));
   if (error == -1)
   {
       perror("bind failed.");
       return 1;
   }

   // listen for a connection
   error = listen(server_sock, 10);
   if (error == -1)
   {
       perror("listen failed.");
       return 3;
   }

   cerr << "Now listening for connections..." << endl;

   // accept a connection
   struct sockaddr_in client_address;
   socklen_t client_addr_len;
   error = client_sock = accept(server_sock,
                               (struct sockaddr *) &client_address,
                                &client_addr_len);
   if (error == -1)
   {
       perror("accept failed.");
       return 0;
   }
   cerr << "Connection request received." << endl;

   write(client_sock, "hello there!", 12);
   close(client_sock);
   close(server_sock);

   return 0;
}
```

2. Example TCP Client

This is the client to go along with the server just shown.

```
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream>
#include <stdio.h>
using namespace std;

int main(int argc, char** argv)
{
   int client_sock;
   int error;

   //create the client socket
   client_sock = socket(PF_INET, SOCK_STREAM, 0);
   // Need the address of the server to connect to
   struct sockaddr_in server_address;
   memset(&server_address, 0, sizeof(server_address));
   server_address.sin_addr.s_addr = inet_addr("192.168.1.10");
   server_address.sin_family = PF_INET;
   server_address.sin_port = htons(2222);

   error = connect(client_sock, (struct sockaddr *)&server_address,
                                         sizeof(server_address));
   if (error == -1)
   {
       perror("Failed to connect.");
       return 1;
   }
   cout << "Here is the message from the server:\n ";
   char ch;
   while (read(client_sock, &ch, 1) > 0)
   {
       write(1, &ch, 1);
   }

   close(client_sock);

   return 0;
}
```

This server and client use 192.168.1.10 for the IP address of the server, and both assume that the server is running at port 2222. You may have to change these values for your system.

3. Lab Assignment 1

Modify the server and client above so that server computes and sends back to the client the square of any integer that the client sends. The protocol is binary, and is as follows:

(1) Client connects and sends a single integer in binary form.
(2) Server reads the integer and sends back a single integer, the square of the received integer, also in binary form.
(3) The server closes the connection to the client.

The server will be a single-threaded server that serializes service to its clients. The server will stay in a loop in which it accepts a connection, reads the integer sent through the connection, and sends back the integer square through the same connection. After that, the server closes the connection and goes back to wait for another connection.

The client should ask the user for an integer, read the integer, open a connection to the server, and send the integer. Then it should read the reply and print it (the square) to the screen.

Test your server and client on your own machine by using two different terminals. Also, test your client against another student's server, and then test your server against another student's client.

4. Lab Assignment 2

Modify the server and client of part 1 so that the client can ask the server to compute either a square or a cube.

Use a binary protocol as follows. To request computation of the square of x, the client sends

```
<integer 6><string square><integer x>
```

That is, it sends an integer in binary form for the length of a string, then sends the characters of the string, and then sends the integer argument in binary form.

Requesting computation of a cube is similar: the client sends

```
<integer 4><string cube><integer x>
```

These are the length of the string `"cube"`, the string `"cube"` itself, and then the argument in binary form.

5. Lab Assignment 3

The protocol of lab assignment 2 can be summarized as follows (omitting the implementation details)

(1) Client sends the request `cube x`; server returns x^3.
(2) Client sends the request `square x`; server returns x^2.

The implementation details are that to send a request, the client sends the name of a function in the form of length of the name followed by the name itself, and then follows this up with the argument. The length of a string is sent before the string to allow the server to know where the string ends.

Devise an appropriate implementation of the following protocol

(1) Client sends the request `sum x y`; server returns $x + y$.
(2) Client sends the request `product x y`; server returns $x \times y$.
(3) Client sends the request `negate x`; server returns $-x$.