SYS Lab

Socket Programming in C

December 2017

In today's lab session, we are going to acquire further knowledge of socket programming in C.

1 Experimenting with the code

First, download and save the sample programs `serverTime.c` and `clientTime.c` on your machine.

> ***Note:*** *If you are running the code on the Linux server, then you should first change the port numbers from 5000 to the last 5 digits of your student ID number (making sure the number is not below 1024 and not larger than 65535), both in the client and server programs.*

If you have not yet experimented with these two programs, then first compile and run them, and experiment with them a bit.

2 The `bind()` function

As you may already know, the server program should run first, and wait passively for connections from clients. Now, follow these steps:

1) Run the server;

2) Run the client program;

3) Shutdown the server (e. g., by issuing `Ctrl+C` on the Linux terminal);

4) Immediately start the server again;

5) Run the client.

Most probably you should receive an error on the client side, similar to the output shown in Fig. 1 below.

Figure 1 The error on the client side due to shutting down and immediately restarting the server.

```
$ ./clientTime 127.0.0.1

 Error : Connect Failed
```

2.1 The reason for the error

This error appears as a result of a subtlety of closing a TCP connection, which revolves around the need for the *Time-Wait* state.

When a TCP connection terminates, at least one of the sockets involved is supposed to persist in the Time-Wait state for a period of anywhere from 30 seconds to 2 minutes after both closing handshakes complete. This requirement is motivated by the possibility of messages being delayed in the network. If both endpoints' data structures go away as soon as both closing handshakes complete, and a new connection is immediately established between the same pair of socket addresses, a message from the previous connection, which happened to be delayed in the network, could arrive just after the new instance is established. Because it would contain the same source and destination addresses, it would be mistaken for a message belonging to the new instance, and its data might (incorrectly) be delivered as though it were part of the new connection.

2.2 How to deal with Time-Wait

Like many other functions that we use in Socket programming, `bind()` returns a negative value when an error occurs. Check the line in the source file `serverTime.c` where the `bind()` function is called.

Task

Surround the `bind()` call with an `if`-statement, and check whether the value that is returned is negative. If not, the server program should be allowed to proceed, otherwise, an error similar to the one in Fig. 2 below should be shown on `stderr`, and then the program must exit with failure, e. g., through issuing:

```
exit( EXIT_FAILURE);
```

Figure 2 The error message from server program when `bind()` returns with failure.

```
bind() error : Address already in use
```

Hint: Although the error message can be produced with the simple use of the `printf()` function, a more systematic mechanism for dealing with error messages of this kind is provided through the `perror()` function. See, e. g.:

- http://man7.org/linux/man-pages/man3/perror.3.html
- http://pubs.opengroup.org/onlinepubs/009695399/functions/perror.html

3 Obtaining clients details

In our server program, the server communicates with the client exclusively through the file handle `connfd` obtained via the following statement:

```
connfd = accept( socketFD, (struct sockaddr*)NULL, NULL);
```

Task

Modify the server program, so that whenever a client connects to the server, the IP address followed by the port number of the client is printed on the standard output, as in Fig. 3 on the following page.

Note that the port number of your client is (almost surely) different from the example shown (i. e., `43843`, and depending on where the client is relative to your server, the IP address might also not be the loopback address of `127.0.0.1`.

For this task, you need to replace the `NULL` arguments in the call to `accept()` on the server side by arguments of the appropriate types.

Figure 3 The client's details as printed on the standard output by the server. Note that these numbers might be different for your program, especially the port number.

```
Accepted a connection from 127.0.0.1: 43843
```

4 Optional

Write a client-server program in C for playing a simple game of guessing numbers. On receiving a connection from the client, the server sends a positive integer N to the client, meaning that the server has randomly chosen a non-negative integer between 0 and N. The job of the client is to guess what the number is.

At each iteration, the client guesses a number k:

- If $k > N$, then the server returns 1.
- If $k < N$, then the server sends -1 to the client.
- If $k = N$, then the server returns 0, and the game finishes.

The entire interaction should be reflected on the client side, with informative messages on the standard output indicating what the client's guess was, and what the server's corresponding reply was (i. e., `too high`, or `too low`, or `correct`).