CSCI 1730 Breakout Lab 13

Client and Server Programs

Learning Outcomes

- Design and implement client and server programs that communicate via sockets in a Unix environment.

Problem / Exercise

The purpose of this lab is to get experience using the C programming language to make UNIX system calls for interprocess communication via sockets between a client program and a server program that run concurrently. You will write a C program that implements a client program that will send a short text message (less than 80 bytes) to a sever program. Your client program must use three command line arguments as shown below, where <ip address> is the IP version 4 address (using the dot notation discussed during lecture classes) of a server, <port> is a port number at <ip address> of a server process that is listening and accepting connections, and <message> is a short text message (less than 80 bytes) the client program will send to the server process listening and accepting connections at <ip address> and <port>.

```
./lab13.out <ip address> <port> <message>
```

You will also write a server program to test your client program to ensure that it is working properly. Your server program must receive a short text message (less than 80 bytes) from the client, and the server should print that message out to the server's standard output. The client and server program must run concurrently on vcf nodes.

Requirements

1. Read through the instructions and examples in this document and chapter 16 in your Advanced Programming in the UNIX Environment, Third Edition (Stevens & Rago) textbook.
2. Your client program must use system calls to socket, send, connect, and close. It may also use calls to other C library functions such as htonl, htons, ntohl, ntohs, and inet_addr. Your client program must use AF_INET and SOCK_STREAM. After your client program sends a message, it should close its sockets and terminate gracefully.
3. Your server program must use systems calls to socket, recv, listen, bind, accept, and close. It may also use calls to other C library functions such as htonl, htons, ntohl, ntohs, and inet_addr. Your server program must use AF_INET and SOCK_STREAM. Your server program should only use port numbers between 60,000 and 65,535 on a vcf node. Do NOT run your server program on odin. Only run your server program on vcf nodes. After your server program receives a short message from a client, it should print that message to the server's standard output (as shown in the examples), close its sockets, and terminate gracefully. You server program must only use one port number at a time. If a port number is in use, use networking commands discussed during lecture classes to find a port number that isn't being used.
4. Your program CANNOT use a call to `system`. If your program uses a call to `system`, then you will receive a grade of zero for this assignment.
5. Implement your client program in a C source file called `lab13.c`, and use our provided Makefile to create an executable called `lab13.out`. Do not modify our Makefile. Your source code must compile correctly (no warnings) without any modifications to our Makefile.
6. Your program's I/O must adhere to the instructions in the Examples section.
7. You may assume valid input for this lab.
8. If a function call fails in your program, then your program should print (to standard output) a reasonable error message and terminate gracefully.

Examples

After you finish implementing the client in `lab13.c`, the I/O from running `lab13.out` should match our examples exactly when run on a vcf node with a server processs running concurrently on a vcf node listing and accepting connections at <ip address> and <port>. You may assume all command line arguments are going to be passed to your program as valid inputs on a vcf node. The output shown for each example is printed by the server to the server's standard output. Before you run any example below, you need to run your server program with the appropriate IP address and port number on a vcf node. Use the network commands discussed during lecture classes to monitor and setup your client program and server program. Your client program must work for any valid sets of inputs provided to it.

```
./lab13.out 127.0.0.1 62010 'I am a client on vcf0 sending a message to vcf0'
The server received: I am a client on vcf0 sending a message to vcf0

./lab13.out 172.19.49.103 60000 'Hi world from vcf2'
The server received: Hi world from vcf2

./lab13.out 172.19.49.103 60111 'Hello from vcf1'
The server received: Hello from vcf1

./lab13.out 172.19.49.100 65001 'Good morning from vcf3'
The server received: Good morning from vcf3
```

1 C Program

1.1 General Requirements

1. Place the files for this assignment in a directory called `lab13`.
2. Your program's source file(s) must compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`).
3. Place your client C source code for this assignment in a file named `lab13.c`. Compile your C program using the provided Makefile for this assignment. Do NOT modify any commands, targets, or dependencies that are associated with compiling source code in the provided Makefile for this assignment. If there are any warnings or errors from the compiler, then you should fix them before you submit this assignment to us for grading.
4. Your program cannot contain a memory leak when executed. You are responsible for using valgrind to test your program to ensure it does not contain a memory leak, and valgrind should be run for every test case for your program. A program that contains one or more memory leaks (definitely, indirectly, or possibly lost) identified by valgrind will lose points.
5. The only include directives that you are authorized to use in your source code for this assignment are listed below.

   ```
   #include <arpa/inet.h>
   #include <netinet/in.h>
   #include <stdbool.h>
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <sys/socket.h>
   #include <sys/types.h>
   #include <unistd.h>
   ```

   You CANNOT use any other include directives anywhere in your source code.
6. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

1.2 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.
4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
5. Every line of source code must be indented properly and consistently.

1.3 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Lab 13. Submit only the following files.

1. All source files required for this lab: lab13.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own copy of the provided Makefile to compile your program. Do not submit your server code. We will use our own server program to grade your client program. We only need the files mentioned above with the file extensions aforementioned.

3 Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

4 Grading: 7 points

If your program does not compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`), then you'll receive a grade of zero on this part of the assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on vcf nodes using our server program	7 points
README.txt file missing or incorrect in some way	-1 point
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
One or more compiler warnings	-2 points
Not adhering to one or more coding style requirements	-2 points
Valgrind identifies a memory leak (definitely, indirectly, or possibly lost)	-2 points
Not submitting this assignment before its late period expires	-7 points
Penalty for using a call to `system`	-7 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin or a vcf node with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.