UNIVERSITY OF DELAWARE
DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

CISC 650 / CPEG 651 / ELEG 651: Computer Networks II

Fall Semester, 2017 Professor: Adarsh Sethi

PROGRAMMING PROJECT 2

This project is an individual assignment and must be done independently by each student.

In this project, you will write a client, a TCP server, and a UDP server. The client will use TCP to communicate with the TCP server and UDP to communicate with the UDP server. The main purpose of the client will be to discover servers that are being run by other students in the class.

While your client should be able to communicate with both servers written by you, your client will use a server discovery process to discover and communicate with servers written by other students of the class. Each client will collect information about the servers it has discovered and each server will collect information about the clients that have communicated with it. Part of your grade will depend on how many clients/servers you were successful in communicating with.

You may use either C or C++ for your programming. You may also use the example client-server programs supplied in the class as a starting point for your programs. All clients and servers must be run on the *cisc650* VM.

The accompanying document, *Helpful Hints*, contains useful and important information about this project, and you should read it before you begin working on the project.

Secret Information

Your client will have the following secret information that will be known only to the client. It is acceptable if you hard-code this information in the client program:

- Client's secret code: a 16-bit unsigned integer in the range 1-65535.
- Client's name: a character array containing your first name and last name separated by a hyphen ('-'). There should be no spaces in the name. Also there should be no null character at the end of the name. Maximum size for the name should be 80 bytes.

Both the TCP server and UDP server will have the following shared secret information that will be common to both servers and will be known only to them. It is acceptable if you hard-code this information in the server programs:

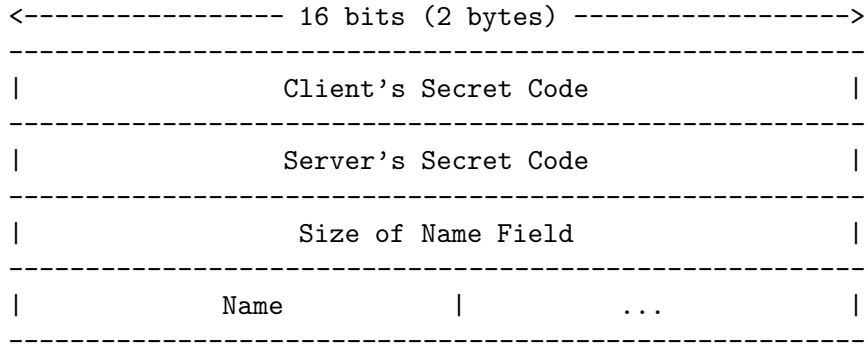
- Server's port number: a number chosen in the range 48000-48999. Both the UDP and TCP servers will use the same port number, although one will use a TCP port and the other will use a UDP port.
- Server's secret code: a 16-bit unsigned integer in the range 1-65535. This number must be different from the client's secret code.

- Server's name: a character string containing your first name and last name separated by a hyphen ('-'). There should be no spaces in the name. This name should be the same as the client's name and has the same constraints.

You should email to me (*sethi@udel.edu*) the Server's port number, Server's and Client's secret codes, and the Client/Server name chosen by you, latest by Monday November 6, 11 pm.

Message Format

All messages exchanged between the client and servers in both directions have the same format as shown in the diagram below. The first three fields of the message are 16-bit unsigned integers. All messages contain the Client's Secret Code. Some messages don't require the Server's Secret Code, in which case this field is set to 0. The Size of Name Field contains the number of bytes in the name field which follows. The name may be either the Client's Name or the Server's Name. Some messages don't require a name, in which case the Size field is 0 and there is no name field after that. The Name Field is a sequence of bytes that contains the Name; there is no terminating null character. The maximum number of bytes in the Name Field is 80.



Communication Steps

The communication between a specific client and a specific server pair will progress through a sequence of two steps:

1. First the client establishes a TCP connection with the TCP server. The client then sends a message to the server with the client's secret code in it. There is no server's secret code and no name in this message. The server sends a response message back to the client containing the client's secret code (which was just received by the server) and the server's own secret code. There is no name in this message. At this point, the TCP connection is terminated.

2. The client now sends a message to the UDP server. This must be a server that uses the same port number as the TCP server in the first step, but this is a UDP port instead of a TCP port. This message contains the client's secret code, the server's secret code (which the client received from the TCP server), and the client's name. The UDP server checks the server's secret code sent by the client to make sure it is the correct code (note that the TCP and UDP servers share the same secret code). If the server's secret code in the above message was correct, the UDP server sends a message back to the client containing the client's secret code,

the server's secret code, and the server's name. If the server's secret code was not correct, the UDP server sends a message back with the client's secret code, the (incorrect) server's secret code that was sent by the client, and no name.

Information Collected by Client and Server:

A client collects information from the UDP servers with whom it communicates and it stores this information in a file called *ClientInfo*. The UDP server collects information about the clients with whom it communicates and it stores this information in a file called *ServerInfo*.

Each line of the *ClientInfo* file should store information about one UDP server that the client has communicated with. This line should have the format:

```
Server Port,Server's Secret Code,Server Name
```

The fields above should be separated by commas. Two examples of lines in the *ClientInfo* file are:

```
48120,9801,Adarsh-Sethi
48772,23456,Bill-Smith
```

The client creates a line in this file after it has received a successful response from a UDP server. The lines should be created in chronological order, meaning that the latest line should be the last one in the file.

Each line of the *ServerInfo* file should store information about one client that the UDP server has communicated with. This line should have the format:

```
Client's Secret Code,Client Name
```

The fields above should be separated by commas. Two examples of lines in the *ServerInfo* file are:

```
9876,Adarsh-Sethi
45748,Bill-Smith
```

The UDP server creates a line in this file after it has received a valid message from a client (with the correct server's secret code). The lines should be created in chronological order, meaning that the latest line should be the last one in the file.

Server Discovery

Each client tries to connect to TCP servers on *cisc650* with port numbers in the range *48000* through *48999*. The client tries each port number in this range once, provided the port number has not been previously recorded in the *ClientInfo* file. If the port number appears in that file, then the client skips it so that it does not repeatedly communicate with the same server.

If the client to TCP server communication is successful, then the client communicates with the corresponding UDP server. It then proceeds to try the next port number. The client stops after it has looped through each of the 1000 possible port numbers.

You should wait for some time and then run the client again to go through the above process. This may be done multiple times over a period of a few days. If the client ever gets stuck (which may happen if it does not get a response back from a UDP server), then simply kill it with a Control-C and then run it again after some time.

Your servers should be designed so they can be run continuously for a period of a few days, during which various clients may try to communicate with them. For this requirement, it is important that the server not quit execution if an error happens once it has gone into the listening loop. Instead, it should simply deal with the error (perhaps aborting the current connection, closing the connection socket, etc.) and go back in the loop to wait for another connection request to come or another client message to be received. In particular, you should check the value returned by the recv() function; if this value is less than or equal to 0, it indicates that the connection has been broken or there is some other error. The same design methodology should be used for the client as well, as the client will have to search for many servers, and it will be problematic if the client quits each time there is an error from a server.

Output:

Each time the client communicates with a TCP/UDP server pair, it should print the messages sent to the servers and the response messages received from the servers. These messages should be printed only if a successful TCP connection was established with the TCP server. Below is a sample format for these messages for one TCP/UDP server pair:

```
-----------------
Port Number 48258
TCP Message sent:     45120
TCP Message received: 45120, 48337
UDP Message sent:     45120, 48337, Bill-Smith
UDP Message received: 45120, 48925, John-Davidson
----------------
```

Try to align the fields in the contents of the messages so they are easy to read, as in the above examples.

The servers should not print any output at all. **You should remove all printf() statements from the server code.** This is because the servers will have to be run in the background, allowing them to be run for long periods of time. Note: While you are debugging the servers, you should run them in the foreground (as you run normal programs), and in this phase, you can use printf()'s to help you debug. See the accompanying document (*Helpful Hints*) for information on how to run a server in the background.

Testing:

You should test your programs in three phases. In the first phase, disable the server discovery in the client. Instead, have the client simply communicate with your own servers. Use this phase to debug both your client and servers and make sure they run correctly with each other. Important: during this phase, use a different server port number which is not in the range 48000-48999. You should aim to complete the first phase by November 17.

In the second phase, which will run from November 14 through November 25, enable the server discovery in the client, and also use the selected server port number in the range 48000-48999. The client will now be able to discover and communicate with multiple servers. Your servers may also get requests from multiple clients. During this phase, I will also run some servers of my own for you to test against.

The third and final phase will run from November 27 through November 30. You will run your servers continuously during this phase for as long as possible. You will also run your client at various times during this period. Your aim should be to collect information from as many clients and servers as possible. The information files that you submit below and on which your project success will be judged must have been collected during this phase. When you run the client for the *first three times* during this final phase, run it in a script so you can make a record of its output. This output will be part of your submission (see below).

Submission:

Submit a zipped copy of your project directory. This directory should include all your original source files, the executables, the information files collected by the client and server, and the scripts generated by you during the client's execution in the final phase.

On the Assignments tab in Sakai, provide the following information as a cover page for your submission: The name of your project directory, the names of the source files, and the names of the files containing the client scripts.

Grading:

Your programs will be graded on the following criteria:

Program Code and Correctness: 40 %
Readability and Documentation: 20 %
Information Collected: 40 %

Part of the grade on "Information Collected" will be based on how many servers were reached by your client, and how many clients were able to reach your servers, as well as the amount of information collected on each. We will also be running our own clients and servers during Phase 3 to monitor your client and servers and we will use the information we collect as part of your grade as well.

Deadline for submission: 11 pm on Friday December 1.

Late submisssions will not be accepted because of the need to coordinate Phases 2 and 3. No extensions will be granted for *any* reason. It is to be understood that computers and computing facilities are subject to failure, overload, unavailability, etc., so it is your responsibility to plan and execute your project work sufficiently in advance so as to meet the due date.