ECE 7428 (Computer Comm. Networks) Project 1

Instructor: Sarvesh Kulkarni (sarvesh.kulkarni@villanova.edu)
Department of Electrical and Computer Engineering
Villanova University

I. Introduction

The goal of this project is to learn how to use UDP sockets and to implement a client-server system for file-exchange.

The best way to get started is by learning about UDP sockets and the manner in which they are used. There are numerous textbooks as well as web sources to get you started. Chapter 2 of your course textbook has useful material on this topic as well. Spend a couple of days on reading and understanding the basics of UDP sockets. It will probably serve you better if you first *get your feet wet* by implementing a simple UDP socket between two machines in a client-server configuration and transfer a few bytes of data. You can add functionality and aesthetics later.

II. Project Specifications

Your programs must meet the following use-case requirements:

1. A server process (let us call it the *data server*) listens on UDP port 7777 on a machine of your choice, waiting for a client to send it UDP data segments.

2. Next, a client process starts execution and gets the IP address of the remote server, which you can supply manually from the terminal window, or from a stored file. The client process may run on the same machine as the server, or on a different machine on a different network.

3. The client then proceeds to transfer ten files to the server as follows:
   A random file (from ten available ones) is selected, and, from this file, a random number of lines (1 - 3), starting with line #1 are selected for transmission to the server. Thus, for example, lines #1 and #2 from file #5 may be transmitted first. After those lines are transmitted, another file is selected at random, and a random number of lines (1 - 3) are again transmitted. Thus, for instance, lines #1, #2 and #3 from file #7 may be transmitted, and so on.

   An example transmission schedule may look like this:
   1. From file #5: lines #1, #2.
   2. From file #7: lines #1, #2, #3.
   3. From file #1: line #1.
   4. From file #10: lines #1, #2, #3.
   5. From file #4: line #1.
   6. From file #7: lines #4, #5.
   7. From file #1: lines #2, #3, #4.
   8. From file #2: lines #1, #2.
   9. From file #7: line #6.
   10. From file #3: line #1.
   11. From file #4: lines #2, #3.

   etc.

   Each line is sent using a separate sendto() socket call until all of the ten files are completely transmitted. No line duplicates are sent unless the originals are lost. Ultimately, when all lines from all ten files are transmitted and received, the client-to-server transmissions are over.

   The server does not have any *a priori* information about the client's transmission schedule. Therefore, the client needs to somehow inform the server of the filename, the total number of lines in that file, and the line number associated with each transmitted line of the file. Doing so enables the server to correctly reconstruct the files from the transmissions. Segments may be lost, damaged, or reordered by UDP, so be sure to take these possibilities into account, and retransmit any lost or damaged segments, if needed.

4. Once the client finishes all transmissions, the server concatenates all files into a single large file and sends it back to the client, with all files and (lines within those files) in order. Again, this being UDP, lost/damaged/out-of-order deliveries are possible, so be sure to fix any problems before saving the concatenated file on the client machine.

III. Extra Credit

Extra credit proportional to the amount of effort that you put into your project will be awarded on a case-by-case evaluation. This extra credit will raise the grade on your project. Once your project grade hits the maximum, any leftover credit will be applied towards your course grade.

You can get extra credit for implementing the following additional features *only after the required features have been implemented.* You may implement any or all of these features.

1. When a data server process starts execution, it first registers the data server's IP address with a *registry server*; the registry server has a static IP address and is physically distinct from the data server. It listens on UDP port 8888. Hereafter, the client can query the registry server to find out the data server's IP address and port number. This information is then used to transfer files to the data server. A registry entry on the registry server is either terminated by the data server process when it exits, or is timed out after an hour, unless renewed by the data server process. ECE graduate students usually have an account on one of our department UNIX servers - run your server process under this account. If you are more comfortable with Windows or Mac environments, feel free to use your home machine (or something similar) for your server.

2. Implement "Automatic Service Discovery" as follows:
   The data server sends out a UDP broadcast *beacon* every three seconds advertising its IP address and port number. If the client and data server processes are located on machines in the same local area network, then the client will be able to receive this beacon and use it to transfer files to the server automatically, without needing manually supplied IP address and port number, or the use of the registry server.

3. Anything else that you can think of and implement in a reasonable amount of time.

IV. Languages And Platforms

You can use any of the following programming languages: C, C++. The use of any other language is strongly discouraged! Should you be completely unfamiliar with C or C++, you may use Java or Python, with special permission of the instructor. However, you will have to demonstrate **considerably** better progress than your peers, in your project implementation to get a good grade. There are no restrictions on your choice of platform. You may use either MS Windows or any of the UNIX/UNIX-like platforms (LINUX/Solaris/Apple OSX).

V. Deliverables

A. An electronic copy of your entire source code (well documented).
B. A typewritten and spell-checked report (approx. 3 - 5 pages) that explains the following.

B.1 Does your program meet all the specifications in Section II? Do all the required features work fully? What features have not been implemented? Finally, have you tested your program for stability?

B.2 Have you implemented any of the features marked for extra credit? If so, what are they? Are there any extra enhancements provided in your software, based on your own initiative? If so, describe them briefly.

B.3 **Clear and complete instructions on how to run the programs.** What is the programming language used, and the hardware platform and the operating system for which your program is compiled? This information is absolutely essential so that the instructor can independently verify the working of your program.

B.4 Documentation of Implemented Functionality - a very brief user guide.

Turn in your deliverables online, and make sure that your report and source code have your name on it.

Please retain the source code and the executable after turning in the project till the semester is over and the grades are mailed. You may be required to provide a running demonstration of your program.

VI. Deadline

See the Blackboard elearning site.