Programming Assignment 1

Data Communication Networks – Spring 2022
A Basic File Transfer Protocol

Due Date: Feb. 16, 2021 by 11:59pm CST (local time in Starkville)

This assignment can be done in pairs for undergraduate students. Graduate students must work individually.

We are dealing with Unix sockets (NOT Windows sockets!). The two are similar, but differ in syntax in places.

Your code must compile on Pluto. Please refer to the syllabus for penalties if your code fails to compile.

Carefully follow the instructions for submitting your solution.

1. Assignment Objective

The goal of this assignment is to gain experience with **UDP socket** programming in a client-server environment (see Figure 1) by implementing a file transfer protocol. You will use `C++` or `Java` (your choice) to design and implement a client program (`client`) and server program (`server`) to communicate between themselves.

Figure 1

2. Assignment Specifications

2.1 Summary

In this assignment, the client will transfer a file <`filename`> (specified in the command line) to the server over the network using internet sockets. The **file formatting must be preserved** (line breaks, white spaces, punctuation, etc.). The "network" will actually be non-existent since you be running your code locally on Pluto, but everything you create here is applicable to a real network.

This assignment uses a two-stage communication process. In the *negotiation stage*, the client and the server agree on a random port <r_port> for later use through a fixed negotiation port <n_port> of the server. Each port is allowed to be between 1024 and 65535 (inclusive). Later, in the *transaction stage*, the client connects to the server through the selected random port for actual data transfer.

2.2 Client-Server Communication

Communication between client and server is done in two main stages as shown in Figure 2.

Stage 1: Handshake. In this stage, the client creates a UDP socket to the server using <server_address> as the server address and <n_port> as the handshake port on the server (where the server is known *a priori* to be listening). The client sends a request to get the random port number on the server where the client will send the actual data. For simplicity, **the client will send the characters 1248 (as a single message) to initiate a handshake with the server;** this does not need to be written to the screen.

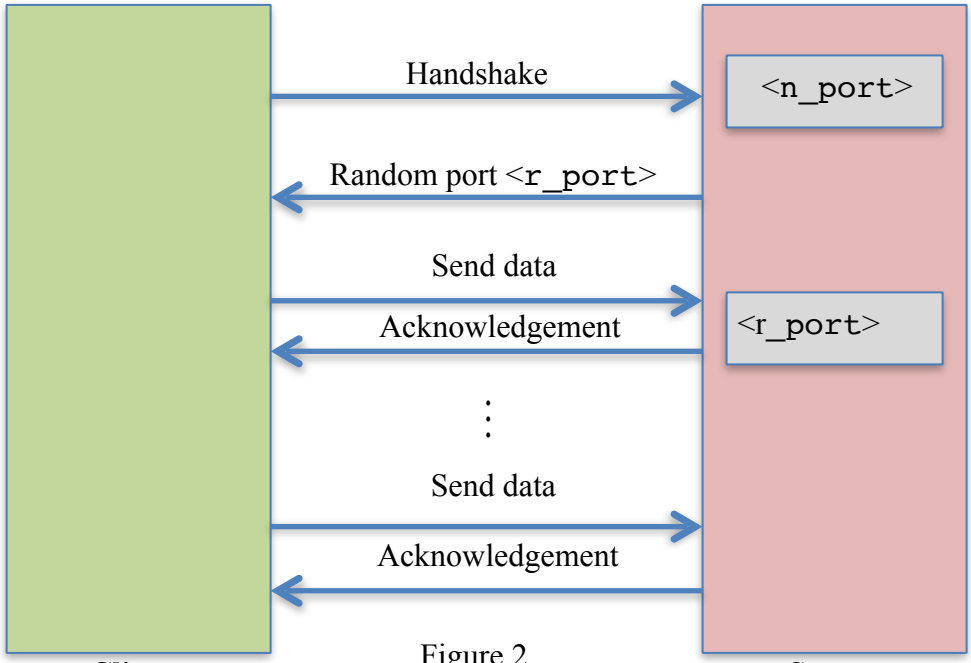

Figure 2

Once the server receives this request in the handshake port, the server will reply back with a random port number <r_port> *between 1024 and 65535* (inclusive) where it will be listening for the expected data.

The server will then write to screen **"Random port chosen: <r_port>"**. It must **write exactly that**, where <r_port> is replaced by a port number, no brackets; or you will lose points.

Both the client and server must close their sockets once the handshake stage has completed.

Stage 2: File Transfer – In this stage, the client creates a UDP socket to the server using <`r_port`> as the port and sends the data. This data is assumed to be 8-bit ASCII (assuming standard 8-bit encoding for each character, so 1 byte per character) and may be of arbitrary finite length of at least 1 byte. The file is sent over the channel in chunks of 4 characters of the file per UDP packet (a character includes white space, exclamation points, commas, etc.). An exception to this rule occurs when the end of the file is reached and, in that case, less than 4 characters of the file may be sent. We call each such chunk of the file a *payload*.

The packet may contain other information in addition to the payload, if you deem it useful; for example, to indicate the end of the file. After each packet is sent, the client waits for an acknowledgement from the server that comes in the form of the most recent transmitted payload in **capital letters**. These acknowledgements are output to the screen on the client side as one line per packet. The client does not need to write these acknowledgements to any file.

Once the file has been sent and the last acknowledgement received, the client closes its ports and terminates automatically; that is, **it must determine the end of the file**.

On the other side, the server receives the data and writes it to file using the filename "`upload.txt`" and you must use this name or you will lose points. **By the end of execution, the entire file should be written on the server side. Note that if a file named "`upload.txt`" existed prior to executing your program, this file must be overwritten with these new contents.** The server does **not** write the received data to screen.

After each received packet, the server uses the UDP socket to send back an acknowledgement to the client that is the most recent received payload in capital letters. Recall that the client displays this data to the screen.

Once the last acknowledgement has been sent, the server closes all ports and terminates automatically; that is, it must determine that end of the transmission has occurred from the client.

2.3 Client Program (`client`)

You should implement a client program, named client. It will take the command line inputs in this order: <`server_address`>, <`n_port`>, and <`filename`>

2.4 Server Program (`server`)

You must also implement a server program, named `server`. It will take the command line input <n_port>.

2.5 Example Execution Commands in Java

Assume that host1 is the server and host2 is the client.

On host1: `java server <n_port>`
On host2: `java client <host1/server address> <n_port> <filename>`

So, for example, you will execute (assuming the use of negotiation port 6003):

```
java server 6003
java client localhost 6003 file.txt
```

2.6 Example Execution Commands in C++

```
./server 6003
./client localhost 6003 file.txt
```

An example execution and data file will be provided online for testing purposes. However, it is your responsibility to test your code thoroughly and ensure it conforms to the requirements.

3 Instructions for Submitting Your Solution

3.1 Due Date

This assignment is due on February 16, 2022 by 11:59pm CST (local time in Starkville).

3.2 Hand in Instructions

Put your full name(s) and student ID(s) at the top line of your code. I will deduct 2 points if you forget this.

If you are working as a pair, only ONE of the students in the pair should submit the assignment.

Submit your files in a single compressed file (either .zip or .tar) using **Canvas**. Do **not** email the TA or I your code; this does **not** count as a submission and you will incur the late penalty. You must hand in the following files:

- **Source code files: your .cpp or .java files (but not .o files or .class files).**
- **Makefile: recall that your code must compile and link cleanly by typing ``make".**

Your implementation will be tested on the Pluto system, and so it must run smoothly in this environment! If your code does compile and run correctly on this system, you will lose points as specified in the syllabus and discussed in class. There will be no exceptions to this rule.

A sample file has been provided online for you to use for testing your client/server programs. However, it is your responsibility to test your code thoroughly and ensure it conforms to the requirements.

3.3 Documentation

Since there is no external documentation required for this assignment, you are expected to have a reasonable amount of internal code documentation (to help the graders read your code).

3.4 Resources for Socket Programming

As specified in the syllabus, learning socket programming is primarily the responsibility of the student . This is the way it has been taught previously here at MSU, and also true of my undergraduate experience at the senior level.

To aid you, I've provided sample client and server code in C++ that uses datagram sockets to transfer several packets. I recommend starting by understanding this code and then expanding on it to fulfill the requirements of PA1.

In terms of references, I believe the best online source for learning socket programming is:

https://beej.us/guide/bgnet/

which presents things in C++. For Java, I recommend looking online for materials, the basic frame work is the same as with C++, but a bit streamlined. Here's a website that looks okay to me (but feel free to ignore):

https://www.geeksforgeeks.org/working-udp-datagramsockets-java/

I will include some manual pages in my slides (lecture slides) on socket programming that are useful. You may also find other online sources — Google will return *many* results — and you should feel free to read and learn from those sources. **Just remember to document whatever sources you use at the top of your code.**

Remember that what you should ***absolutely not do*** is simply copy code. The whole point of this first assignment is to "get your hands dirty" and learn the basics of socket programming. If you copy code, the second programming assignment will be very difficult to complete.