# Programming Assignment 1

## CSE 4153/6153 – Data Communication Networks – Spring 2017
## A File Transfer Protocol

**Due Date:** February 10, 2016 by 11:59pm CST (local time in Starkville)
**Assignments are to be done individually**.

**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 both TCP and UDP socket programming in a client-server environment (see Figure 1) by implementing a file transfer protocol. You will use `Java` or C++ (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.)

This assignment uses a two-stage communication process. In the *negotiation stage*, the client and the server negotiate 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

The communication between client and server in this project is done in two main stages as shown in Figure 2.

Stage 1: Negotiation using TCP sockets – In this stage, the client creates a TCP connection with the server using <server_address> as the server address and <n_port> as the negotiation 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 123 (as a single message) to initiate the negotiation with the server**.

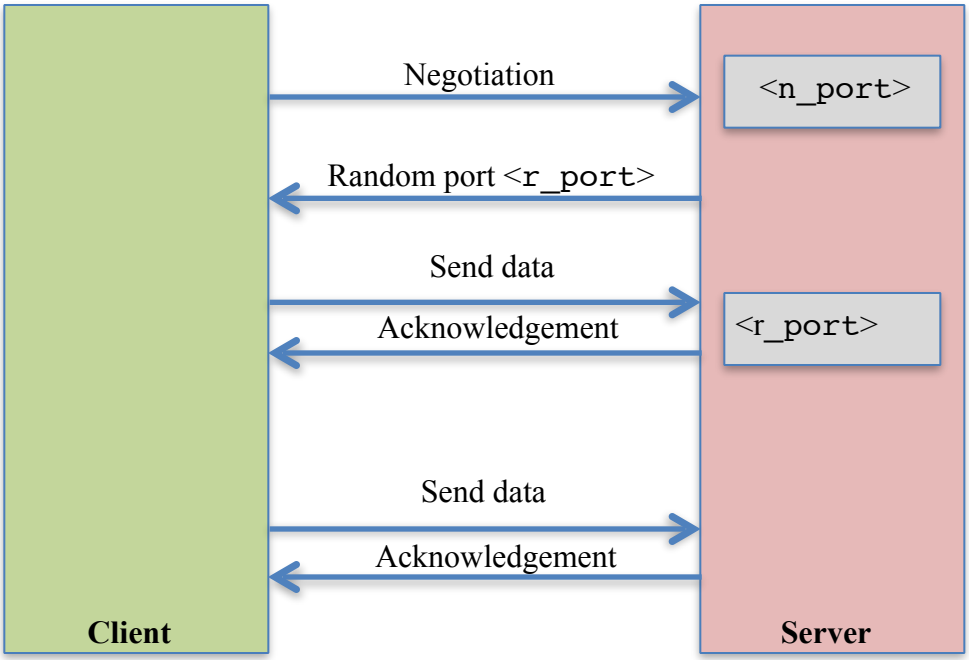


Figure 2

Once the server receives this request in the negotiation 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 **"Negotiation detected. Selected the following random port <r_port>"**. Both the **client and server must close their sockets once the negotiation stage has completed.**

Stage 2: Transaction using UDP sockets – In this stage, the client creates a UDP socket to the server in <r_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 can be sent in the next packet. 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 (does not append) to file using the filename "`output.txt`". 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. The server does **not** write the received data to 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 Your Submitted Solution

### 3.1 Due Date

This assignment is due on Feb 10, 2017 by 11:59pm CST (local time in Starkville).

### 3.2 Hand in Instructions

Submit your files in a single compressed file (either .zip or .tar) using **myCourses**. Do **not** email me (or the TA) 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 only. No other files!**
- **Makefile: 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 covered in class. There will be no exceptions to this rule.

A sample file (output.txt) 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 Online Resource 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. I believe the best online source for learning socket programming is:

http://beej.us/guide/bgnet/output/html/multipage/index.html

I included some man pages in my slides (lecture 2) 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.

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.