EE 122 – Project #1

Spring 2015

Total:	100 Points
Due:	Friday, March 6, 2015
Submission:	Please upload your code and write-up to bcourses

Complete this project in groups of two, using two laptops.

This project asks you to write some code and generate a write-up of results for experiments performed with this code. You will need to submit both the source code and the experimental results write-up to the bcourses website.

Step 0: Set Up

Determine the IP addresses of your computers on campus.
On Linux, you can use the `ip addr` command, on windows use `ipconfig`, or go to *http://www.whatsmyip.org/*

Prepare arbitrary files of sizes 10 MB, 50 MB, and 100 MB. You can use any files you like. We have also provided a python script that can be used to generate text files of any size given. You can generate files with the following:

```
python makefileofsize.py -s <SIZE_IN_BYTES> <OUTPUT_FILENAME>
```

Step 1: Code Specification

Write two programs – a server and a client – such that:

Server Program:

The server program is to transfer a specified file to any connecting clients. It only needs to connect to one client at a time, but it should be able to serve multiple clients in sequence without needing to be restarted.

Usage:
```
./proj1_server <mode> <port> <filename> <packet_size>
        <packet_delay>
```

Where:

<mode>
Either 0, 1, or 2. This value indicates what type of connection to establish with the server.

0 : Connection-oriented sockets
1 : Connectionless sockets
2 : Connectionless sockets without checksum enabled

<port>
The port number to use for the server

<filename>
A path to a local file to host. When a client connects to the server, it will be served this file using the protocol specified by <mode>.

<packet_size>
The size of each packet to send, in bytes. You will need to break the input file into packets of this size, and send the packets separately through the network.

<packet_delay>
The server should sleep for this many seconds between sending packets. Note that the value can be expressed as a floating point value. For sleeping in fractions of a second, look at the man page for `usleep()`.

Example:

The following call will specify the server to use datagram sockets (UDP) on port 33122, hosting a file called `bar.txt` in a folder named `foo`. The program will send the file in packets of size 1000 bytes, with zero delay between packets.

```
./proj1_server 1 33122 foo/bar.txt 1000 0
```

Client Program:

The client program is to connect to the specified server on the specified port, receive a transferred file using the specified mode, and to save the given file to disk at the specified location.

When running the client, it should attempt to connect to the server, retrieve the file, and exit.

The client should also print out statistics information of how long the connection lasted (in seconds) and how large the received file is. In addition to the received file itself, the client is required to export a second file listing the time-delay (in seconds) between each packet received from the server.

Usage:
```
./proj1_client <mode> <server_address> <port>
        <received_filename> <stats_filename>
```

Where:

<mode>

Either 0, 1, or 2. This value indicates what type of connection to establish with the server. It is assumed that the same mode is specified for both the server and the client.

<address>
The IP address of the server to connect to.

<port>
The port number of the server to connect to. Use the same number here as was used to initialize the server.

<filename>
Where to store the file on the local machine after it has been transferred from the server.

<stats_filename>
Where to write the file containing statistics information about the received packets from the server. This should be a text file, where each line indicates the delay between successive packets, as a floating-point number in seconds.

Example:

This call open a client to connect to the server located at 128.32.47.83
using datagram sockets (UDP) on port 33122, and will retrieve the file `foo.txt` and write out statistics info to `stats.txt` :

```
./proj1_client 1 128.32.47.83 33122 bar.txt stats.txt
```

Both programs must be written in C/C++. All code must be submitted along with a README file that specifies:

1. Names of group members
2. Emails of group members
3. Platform (e.g. Linux, Windows, Mac)
4. Compilation instructions

Step 2: Experimenting with Distance

We now ask you to run a few experiments with your code. For each of the following experiments, document your results in a write-up, and submit that write-up along with your code to the bcourses website.

Use your code, as specified above, to compare the time needed to transmit each of your data files under the following conditions:

(a) Position both computers close to each other on campus (e.g. both computers connected to Wi-Fi in the same room).

(b) Position one of your computers on campus running the server program, and the other computer running the client program at a location off-campus (e.g. your apartment or an "open access point").

(c) Choose the placement of the receiving laptop to have weak wireless signal (e.g. AirBears on memorial glade).

Plot the total time required for the file transfer of each of your three test files, for each mode described in **Step 1**, for each of these scenarios. Feel free to use any packet size of your choice, but make sure to document what value you used in your results.

Also, plot the packet loss for each mode specified. Missing packets can be identified by differences in the original and received copies of the file. Out-of-order packets or corrupted packets can be observed via the `diff` command.

Step 3: Experimenting with Delay

Use your code to set up the server to send a large file as a stream of short packets – 200 bytes of payload per packet, with a delay between each packet of 50 milliseconds.

(a) Measure the delay differences of individual packets by analyzing the output statistics file from your client code. Compute the variance and generate a histogram of the delay times between packets. Perform this analysis for the scenarios described in **Step 2a** and **Step 2c**.

(b) Use the **packet pair technique** to determine the bottleneck capacity of the communication path for each of these scenarios.

Step 4: Experimenting with Buffering

Create another server/client pair of programs. The purpose of these programs is to send a stream of data with mean sending rate of 1 packet every 5 seconds. Use a random number generator to produce a uniform distribution of delays between packets of 0 to 10 seconds. The content of these packets is unimportant. We are interested in the rate at which they are being sent.

The client program will receive these packets and generate a signal (such as a "blink") every time it receives a packet. It will have an command-line option that will specify that the client will buffer the incoming packets, so that they are "consumed" at a uniform rate once every 5 seconds, regardless of whether the incoming packets are spaced more or less than 5 seconds apart.

NOTE: You can, and we expect you to, use blocking calls to recv() for the client.

Server4 Program:

The server program is to transfer a stream of packets to any clients until the client closes the socket. The server should be able to serve multiple clients in sequence without needing to be restarted.

Usage: `./proj1_server4 <port>`

Where:

<port> - The port on which to run the server.

Client4 Program:

The client program will connect to the specified server, receive all packets from the server, and produce a "blink" (or other signal) to the screen every time a packet is received. The client should run until the user sends a SIGINT (ctrl-c), at which point it should exit gracefully.

Usage: `./proj1_client4 <address> <port> [ -b ]`

Where:

<address> - The IP address of the server to connect to.

<port> - The port on which to connect to the server.

[- b] - An optional flag. If present, then the client will buffer the incoming packets, so that the "blinks" appear at a uniform rate of 5 seconds, regardless of whether the incoming packets are spaced more or less than 5 seconds apart.

Both programs must be written in C/C++. All code must be submitted along with a README file that specifies:

1. Names of group members
2. Emails of group members
3. Platform (e.g. Linux, Windows, Mac)
4. Compilation instructions

Submission Details

In total, each team must submit the following:

(1) Code for Step 1, including README file and compilation Makefile/scripts used. This should be in an archive named:
`EE122project1-step1-(last name)-(last name).(extension)`

(2) Code for Step 4, including README file and compilation Makefile/scripts used. This should be in an archive named:
`EE122project1-step4-(last name)-(last name).(extension)`

(3) Writeup for experimental results in **Steps 2 and 3.** This report should include both students' names, the answers to all questions. This should be a PDF file named:
`EE122project1-report-(last name)-(last name).pdf`