CSE 156/L

Programming Assignment 2
Due: [ ]

In this project you will ~~to~~ develop a client and server for a file echo application, using UDP. The client of the application reads a local file and sends it to the server. The server simply echos back the received packets' application data. The client listens for the echoed packets from the server and saves the content of the packets to reassemble a new copy of the original file and stores the output to a new file. The packets sent by the client may get re-ordered in transit to the server and back.

Description

1. After the **server** is started, it waits to receive packets from clients. Once a packet arrives, it reads the packet and sends the payload back to the client IP address and port number (from which it received the packet). The server is, in effect, echoing the payload back to the client.

   The server is started by the following command giving the port number for receiving packets.

   `./myserver` *`port_number`*

2. The **client** reads a file from disk and sends it to the server over a UDP socket. In order to send the file, the client breaks the file up into *mtu*-byte sized packets.

   The client also reads the echoed packets from the server. It will assemble the received responses to re-construct the original file. The client will save the bytes it has received in an output file, specified by the command argument *out_file_path*. The resulting new output file must be identical to the initial read file for correctness and success. In particular, the client must save the bytes in the correct original order, even if the packets have been re-ordered in transit to the server and back.

   The client can exit successfully, i.e., with exit(0), when the same exact file has been reconstructed from the server responses via a UDP socket. If there is no packet loss, the client should be able to successfully reconstruct the original packet. Assume there are no packet losses in transit.

   If there are any losses, the client should detect it and print an error message "Packet loss detected" to **stderr**, and exit with failure, i.e., exit() with a non-zero value.

   The client is started by the following command.

   `./myclient` *`server_ip server_port mtu in_file_path out_file_path`*

In your document, describe your mechanism to split the file into packets, and the protocol you define to provide ordering of the data to assemble the original data file.

For example, the following to commands start a server and a client, respectively. Please note that the input or output filename arguments may be paths.

```
./myserver 9090

./myclient 10.10.0.1 9090 512 dir/testfile outfile
```

Requirements

- The protocol between the client and server cannot specify or use any transport (TCP or UDP) port number. That is, the application messages between the client and the server **must not** carry any transport port number.

- The client **must** not send packets with payload data larger than *mtu* bytes.
- The client **must** handle input or output filename arguments that include directory paths.
- The client **must** write the final re-assembled file named as given by the command argument `out_file_path`.
- The client **must not** block indefinitely on reading from or writing to any sockets.
- The client **must not** hang if the server is down. It should print an error message "Cannot detect server" to `stderr` and exit with failure, if it cannot contact the server or has not received the file within some maximum interval (e.g., 60 seconds).
- The client must be robust and not crash if there are any socket API errors. It should instead close itself properly. A message about the error should be displayed to the `stderr`.
- The client must be able to work with input files of any size, including zero.
- The client **must** exit with zero, if successful, and non-zero otherwise.
- The server **must** handle receiving packets from one or more clients.
- The server may block to receive packets from clients.
- You are free to output any needed debug messages on `stderr`.
- The documentation should describe the method to split the file into packets, and the defined protocol to provide ordering of the received echoed data to assemble the original data file.

What to submit?

You must submit all files in a single compressed tar file (with `tar.gz` extension). The files should include

1. A README file including your name, student ID, and a list of files in the submission with a brief description. It should be in the top directory.
2. Organize the files into sub-directories (`src`, `bin`, and `doc`)
3. The source files should be in the `src` sub-directory.
4. A `Makefile` that can be used to build the client program from the required source files. It should be in the top directory.
5. You should not include the compiled program. The `Makefile` will be used to generated the executable program. The make step should put the executable program in the `bin` sub-directory.
6. Documentation of your application in plain text or pdf. Do not include any Microsoft Word files or other formats. The documentation should describe how to use your application and the internal design, as well as any shortcomings it might have. `The documentation should list 5 test cases done by you to validate the functionality`. This file should be in the `doc` sub-directory.
7. Name your file lab2-CruzID.tar.gz where CruzID is your UCSC email username, and lab2 is the name of this assignment. Submit this file.

Grading

Each submission will be tested to make sure it works properly and can deal with errors. Grades are allocated using the guidelines below.

Basic Functionality:	50%
Dealing with Errors:	20%
Documentation:	20%
Style/Code Structure, etc.:	10%

Note that 20% of the grade will be based on how well your code deals with errors. Good practice includes checking all system calls for errors and avoiding unsafe situations such as a buffer overflow.

The files must be submitted before he due date and time. Please make sure to submit on time. Late policy commences immediately after the due date/time. Reminder: Late policy deducts 10% per day from your grade.

The basic functionality is that the received file by the client is identical to the original file the client sent to the server. Pay attention to how well your code deals with errors. Good practice includes checking all system calls for errors and avoiding unsafe situations such as a buffer overflow.

The files must be submitted before the due date and time. Please make sure to submit on time.

Honor Code

All the code must be developed independently. All the work must be your own. You can re-use code that you have written in your own assignments in the class. Also, please do not cut-and-paste code blocks off of the Internet. Any violation of the above will result in getting a zero for the test.