**CPSC 3780**

# Course Project
# Due: December 8, before class.
# Total marks: 50

This project will give you an introduction on socket programming, which is the foundation of the *client-server computing model*. Due to the limited class time in one semester, we cannot cover many topics at the application layer. It is thus hoped that this project will compensate for this.

Your task is to transfer one big ASCII flat file from the server (A) to the client (B). Each time only one line in the file is transferred. However, each transfer can only handle at most a certain number, e.g., 20, of characters. If one line contains too many characters, you need to chop it down into several transfer tasks. You also need to simulate when an error happens, under which condition you ask for retransmission.

There are two simplex channels you simulate between A and B. While A's task is to send the ASCII file to B using one simplex channel, B uses the other channel to tell A whether a transfer is correct or not.

**Project background**

Sockets are a mechanism for exchanging data between processes. The processes can be on the same host or on different hosts via a network connection. Once a socket connection is established, data can be exchanged in both directions until one of the two parties closes the connection.

Usually the process which provides a service (to be made concrete later on) is called the *server* while the process which requests the service is called the *client*. Both the server and client are implemented as processes.

A typical client-server connection consists of the following steps. First, the server creates a listening socket and waits for connection attempts from clients. The client creates a socket on its side, and attempts to connect with the server. The server then accepts the connection. After that, the data exchange begins. Once all data has been passed through the socket connection, either party can close the connection. This is summarized in the table as follows.

| Server | Client |
|---|---|
| 1. Establish a listening socket and wait for connections from the client. | |
| | 2. Create a client socket and attempt to connect to server. |
| 3. Accept the client's | |

| | |
|---|---|
| connection attempt. | |
| 4. Send and receive data. | 4. Send and receive data. |
| 5. Close the connection. | 5. Close the connection. |

Read the tutorial posted on the course website for socketing.

**Project description**

You can form a group of at most four (4) members. It is your responsibility to distribute the workload among group members. All members in one group will be awarded the same mark.

In the following, A refers to the server while B refers to one client. Essentially, A needs to set up two sockets, one for transferring data while the other for accepting ack/nak. So you need to change the general procedure in the above table between server and client for this purpose.

Upon receiving data from A, B checks the data and simulates the following. For every 10 transfers, it flips a coin using a function. If the coin is head, it replies that the transfer is good. Otherwise, it replies that the transfer is bad (even though the received data is good) and asks for retransmission. Note that this could happen in the transfer task of a single short line or in one of the several transfer tasks of a single long line. But once A receives this information, that particular piece of data needs to be retransferred.

Once the client receives a correct entire line, it needs to print it out on its screen.

The data file in your program should be ASCII based and contain more than 1000 lines. Some lines can have only a few characters while others can have, say, more than 300 characters, to simulate the deassembly and reassembly operations. One possible way to do this is to download a document from the Internet and then reformat it. Note that the content of the file is not important. When you reformat it, please add a line number to each line such that the marker can trace it.

You are required to design the protocol between the server and the client. There are two scenarios for you to design and implement.

(S1) There is one server and one client. The client makes a request and the server serves the request. This process continues until the file is finished transferring.

(S2) There are one server and multiple clients (up to 5). A client makes a request and the server responds it by creating a new thread. The new thread is responsible for exchanging data with the client. The server then goes back and waits for requests from other clients.

**Project tasks and submission**

(1) Read the tutorial on the course website.

(2) Design a solution to the above problem and implement it using C++.
(3) For your program, write a small tutorial (hardcopy) as how to compile and use it. Your marker will check your program based on your tutorial.
(4) Write a summary (hardcopy and at most three pages) of your design and implementation, such as the protocol, e.g., data format you have designed, what difficulties you have encountered and how you have handled them, special data structures you have used, etc.
(5) Submit your program (all the source code files, make file, etc.) in a CD such that the marker can check it. Also contained in the CD should be the data file you have used.

**Project hints**

(1) Finish Part (S1) first. Part (S2) is just a generalization of Part (S1) by including threading in your program. For multithreading in C++, please see the link on the course website.
(2) Design the data format first, and then create a function as how to operate on it.