Programming Assignment #1
CS 3251, Fall 2022
Due: November 1st

Instructions:

- Feel free to talk to each other and ask and answer questions on Piazza about the assignment. However, please write your own code.
- Please use office hours for help. If you are unable to make it to office hours, please email the TAs.
- Refer to the code of existing UDT applications for help designing your own code and to seek answers for questions on how to use UDT.
- All answers will be graded on a Linux-based autograder on Gradescope. This should mean that those developing their code on Linux and MacOS machines should face no issues. However, if you are using a Windows machine to develop your code, please make sure to follow the Windows-specific instructions below.

Early Release Disclaimer

This is an early release version of the programming assignment. We expect to enable Gradescope for evaluation. That might lead to small changes to how the output is produced and some of the expectations listed below. We are releasing this PA early so you can get started with the logic of the programs. More information will be provided about Gradescope in the next few days.

Objective

- Understand creation of sockets using UDT
- Understand that application protocols often are simple plain text messages with special meanings
- Understand how to parse simple text commands

Introduction

In this assignment, you will create a chat room on a single computer where you and your (imaginary) friends can chat with each other. You will create the chat room application using UDT sockets. The following steps will be required for a functional chat room application:

1. Create a server program that is always running on a specific port (for example, 5001).
2. Create a client program that can join this server.
3. The client needs a display name and a passcode to enter the chat room. (Assume all clients use the same passcode but different display name).
4. The job of the server is to accept connections from clients, get their display name and passcode (in plaintext), verify that the passcode is correct and then allow clients into the chat room.
5. When any client sends a message, the display name is shown before the message, and the message is delivered to all other current clients.

6. Clients can type any text message, or can type one of the following shortcut codes to display specific text to everyone:

```
- Type :Exit to close your connection and terminate the client
- Type :) to display [feeling happy]
- Type :( to display [feeling sad]
- Type :mytime to display the current time
- Type :+1hr to display the current time + 1 hour
```

For `:mytime` and `:+1hr`, please use the [C standard ctime()] (https://www.cplusplus.com/reference/ctime/) function to format the date like this:

```
Mon Aug 13 08:23:14 2022
```

Learning Outcomes

- Basic socket programming to create a client-server application
- How do multiple clients connect to a single server?
- How to keep multiple persistent connections alive?
- Text parsing to develop a custom application layer protocol. (This is, in spirit, very similar to how HTTP works, for example)

Assignment Details

Connection Establishment and Password Checking - Single Client (20 points)

You will create two programs: a client and a server. Each program takes the following CLI parameters: the client takes the server's IP and listening port, the username, and the password (all clients should use the same password). The server takes its listening port and the password.

If the password is not correct, the client should receive a failure message "Incorrect passcode". Whenever a new client joins the chatroom, all other clients should receive a message indicating the username of the new user that has just joined the room.

Example:

1. Start the server (note the binaries on windows will have a .exe extension)

```
~:$ ./chatserver -port 5001 -passcode <passcode>
Server started on port 5001. Accepting connections…
```

2. Start the client

```
~:$ ./chatclient -join -host <hostname> -username<username> \
                       -passcode <passcode> -port 5001
Connected to <hostname> on port 5001
```

Producing this output on the server the server

```
<username> joined the chatroom
```

Resource: Sample code for providing command line arguments to a C application
https://linux.die.net/man/3/getopt_long

Connection Establishment and Password Checking - Multiple Clients (40 points)

The server should be able to handle multiple clients connecting to it. This means that by running the above client command again (with a different username), the server should perform similarly. The server should also inform the already-connected clients that a new client has joined.

Example:

1. One the new client client

   ```
   ~:$ ./chatclient -join -host <hostname> -port 5001 \
                  -username<username> -passcode <passcode>
   Connected to <hostname> on port 5001
   ```

2. On the server

   ```
   <username> joined the chatroom
   ```

3. On all already-connected clients

   ```
   <username> joined the chatroom
   ```

You don't have to check for unique usernames, we will only test the code with unique usernames.

Chat Functionality (20 points)

After successfully connecting to the server, clients should be able to type messages that get sent to the server when the user enters a newline. All text before the newline should be sent to the server, displayed on the server's screen, and broadcasted and displayed on the screens of all clients.

Example:

1. `<username>` client inputs “Hello Room” and its output should be
   ```
   <username>: Hello Room
   ```
2. On the server the output should be
   ```
   <username>: Hello Room
   ```
3. On all other clients, this should be the output
   ```
   <username>: Hello Room
   ```

If the chat functionality works with just one client, you will get only 2 points.

Chat Shortcuts (20 points)

As discussed earlier, clients should be able to send shortcuts that are translated to text. The shortcut should be displayed as full text on all screens.

Example:

1. <username> client input “:)” and its output should be
   ```
   <username>: [Feeling Happy]
   ```
2. On the server the output should be
   ```
   <username>: [Feeling Happy]
   ```
3. On all other clients, this should be the output
   ```
   <username>: [Feeling Happy]
   ```

Each shortcut is worth 4 points.

Notes and Suggestions

The Gradescope Autograder expects very specific outputs from your programs. Make sure that there are no extra lines or missing/extra spaces. Make sure your output looks like this:

```
Server started on port 5001. Accepting connections…
alex joined the chatroom
alex: Hello World!
```

Not

```
Server started on port 5001. Accepting connections
```

```
alex joined the chatroom
alex:Hello World!
```

Your submission will be graded only if it compiles. We will be using Ubuntu 18.04 for testing your code. But if your submission has some extraordinary requirements (OS other than Linux, some obscure library (pthread is not obscure), etc.) please let us know beforehand and we can confirm if it is ok for you to use.

For Windows Users:
The autograder will run on Linux. It's part of your task to ensure that your program compiles correctly in the autograder. This requires that you make your code portable. Windows and Unix-based systems (Linux and MacOS) have different system calls. Thus, you will notice that libraries like UDT, that aim for portability between operating systems, have to handle this by telling the compiler which system calls and configurations to use when the library is compiled on each operating system. To do so, the code includes preprocessor conditional directives to tell the compiler which code to use on which operating system. The directives used are `#ifndef` and `#ifdef`. The conditional directives check the following macro `WIN32` which is defined only when UDT is compiled on Windows.

For example, in the code of `sendfile.cpp`, you will notice that the headers included for socket programming depend on the operating system.

```
#ifndef WIN32
  #include <cstdlib>
  #include <netdb.h>
#else
  #include <winsock2.h>
  #include <ws2tcpip.h>
#endif
```

Further, the code for creating threads is different for Windows and Unix-based operating systems. Thus, the prototype of the thread function is defined based on the operating system.

```
#ifndef WIN32
void* sendfile(void*);
#else
DWORD WINAPI sendfile(LPVOID);
#endif
```

Thread creation also varies.

```
#ifndef WIN32
  pthread_t filethread;
  pthread_create(&filethread, NULL, sendfile, new UDTSOCKET(fhandle));
  pthread_detach(filethread);
#else
  CreateThread(NULL, 0, sendfile, new UDTSOCKET(fhandle), 0, NULL);
#endif
```

The thread function definition and its return values also vary.

```
#ifndef WIN32
void* sendfile(void* usocket)
#else
DWORD WINAPI sendfile(LPVOID usocket)
#endif
{
  // function code
  #ifndef WIN32
    return NULL;
  #else
     return 0;
  #endif
}
```

The remaining difference is ensuring that the maximum segment size is correctly configured in windows.

```
//after defining the socket serv, set its mss
#ifdef WIN32
  int mss = 1052;
  UDT::setsockopt(serv, 0, UDT_MSS, &mss, sizeof(int));
#endif
```

To ensure the portability of your code, please ensure to include conditional directives in your code whenever you are specifying which socket libraries to include and creating threads. Further, please make sure to set the mss correctly when you are running experiments locally on your Windows machine (but include the code in a conditional to avoid running it in the autograder). If for any reason you decide to use system calls other than those in the above examples, please check UDT's code for examples of how the system call is used and follow that example to ensure portability.

Additional note on resources

GT provides computing resources for CoC students which you can make use of to help you with this assignment. Students have used these computing resources in the past in programming assignment 1, although there can be challenges with them at times. Please use the support helpdesk if you run into any issues while working with them.

Note: Please protect your folders if you are working on these servers. By default your home directory is world readable on those servers. You can fix it with the command `chmod 600~` and then only you will be able to access your homedir. Link - https://support.cc.gatech.edu/facilities/general-access-servers

Questions?
Use piazza or office hours for project-related questions. Given the size of the class, we will find it difficult to manage emails.