EE450 Socket Programming Project

Spring 2024

Due Date:

Friday, April 26, 2024, 11:59PM

(Hard Deadline, Strictly Enforced)

OBJECTIVE

The objective of this assignment is to familiarize you with UNIX socket programming. **It is an individual assignment and no collaborations are allowed. Any cheating will result in an automatic F in the course (not just in the assignment).** If you have any doubts/questions, post your questions on D2L. **You must discuss all project related issues on the Piazza discussion forum**. We will give those who actively help others out by answering questions on the Piazza discussion forum up to 10 bonus points.

PROBLEM STATEMENT

Dormitory Reservation Systems are key for managing student housing efficiently. They help organize room bookings, make finding and booking rooms easier, speed up the booking process, and give useful information for making decisions. These systems make it easier and quicker for students to find housing and for staff to understand which rooms are most wanted. This helps manage room assignments better and respond faster to student needs, making sure everyone finds a place to stay.

Security is very important too. It's essential to make sure the system is safe by requiring usernames and passwords. This stops people who are not students from booking rooms and keeps housing available for actual students. So, building a safe, reliable, easy-to-use online booking system is very important for our dormitory to work well.

For this project, you'll make a simple dormitory booking system. We'll divide the dormitory into three types of rooms: Single Rooms, Double Rooms, and Suites, to keep things organized.

Students can look up if the type of room they want is available and book it if it is. They use a client interface to reach the main dormitory server, which then connects with the specific section server corresponding to a specific room type. Each section server has information on the rooms it handles. The main server also checks the user's identity.

- **Client**: The system features include checking if rooms are available and booking them, with two types of clients: Guest and Member.
    - Members: can log in, search for room availability, and book a room.
        - Book a room, which will then decrease the room availability by 1 in the backend server's data structure (this change is only in the data structure and not written back to the input file).
        - Members' usernames and passwords are stored in a file named "members.txt."
    - Guests： can only search for room availability.

    - Users who skip the password input by pressing "Enter" will be treated as a Guest.
- **Main Server (ServerM)**: Verifies the identity of the students and coordinates with the backend servers.
- **Backend Servers (Single (S), Double (D), Suite (U))**: Store the information of a specific room type.

This dormitory reservation system aims to provide a seamless and efficient process for students seeking on-campus housing, ensuring a positive experience from search to reservation.

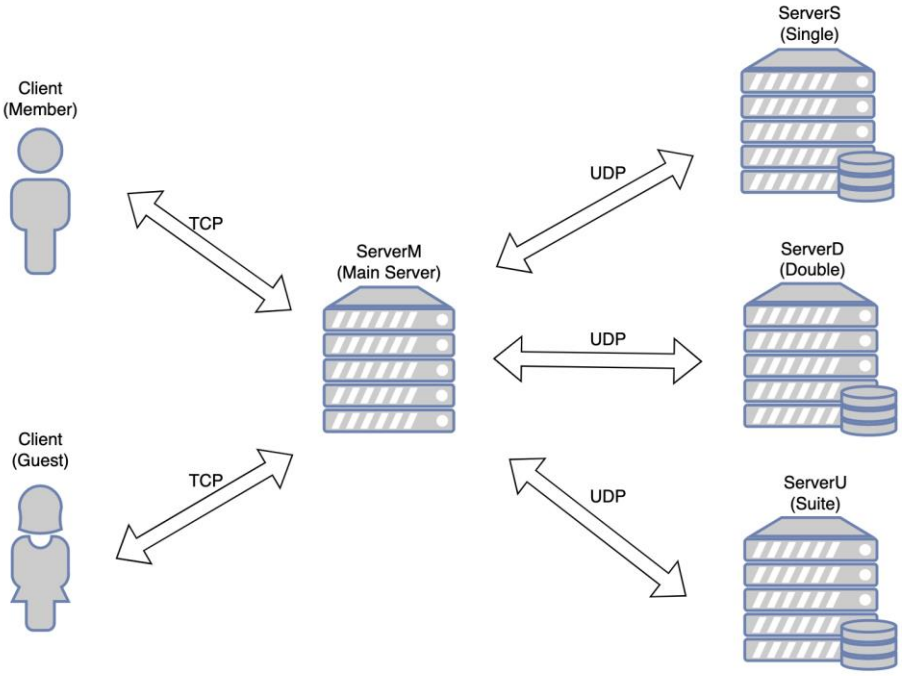

Figure 1: Illustration of the system

Source Code Files

Your implementation should include the source code files described below, for each component of the system.

- Client: The name of this piece of code must be **client.c** or **client.cc** or **client.cpp** (all small letters) and the header file (if you have one; it is not mandatory) must be called **client.h** (all small letters).
- serverM (Main Server): You must name your code file: **serverM.c** or **serverM.cc** or **serverM.cpp** (all small letters except 'M'). Also, you must include the corresponding header file (if you have one; it is not mandatory) **serverM.h** (all small letters except 'M').
- Backend Servers S, D, U: You are required to create three distinct files, choosing from the following naming conventions: **ServerS.c**, **ServerS.cc**,

ServerS.cpp or **ServerD.c**, **ServerD.cc**, **ServerD.cpp** or **ServerU.c**, **ServerU.cc**, **ServerU.cpp**. The filename must utilize one of these formats, substituting "#" with the specific server identifier (either "A" or "B") to reflect the server it represents, resulting in filenames like serverA.c, serverA.cc, serverA.cpp, serverB.c, serverB.cc, or serverB.cpp (note that the name should be entirely in lowercase except for the letter replacing "#"). If available, you should also include a corresponding header file named server#.h, adhering to the same naming rule for the "#" replacement. This ensures a clear, organized naming structure for your code and its associated header file, if any.

Note: You are not allowed to use one executable for all four servers (i.e. a "fork" based implementation).

Input Files

member.txt: contains encrypted usernames and passwords. This file should only be accessed by the Main server.

single.txt: contains single room information categorized in roomcode, and number of the available rooms. Different categories are separated by a comma. This file should only be accessed by the Backend Server S.

double.txt: contains double room information categorized in roomcode, and number of the available rooms. Different categories are separated by a comma. This file should only be accessed by the Backend Server D.

suite.txt: contains suite room information categorized in roomcode, and number of the available rooms. Different categories are separated by a comma. This file should only be accessed by the Backend Server U.

Note: member_unencrypted.txt is the unencrypted version of member.txt, which is provided for your reference to enter a valid username and password. It should NOT be touched by any servers!

DETAILED EXPLANATION

Phase 1: Bootup

Please refer to the "Process Flow" section to start your programs in the order of the main serverM, server S, server D, server U, and two Clients. Your programs must start in this order. Each of the servers and the clients have boot-up messages that must be printed on the screen. Please refer to the on-screen messages section for further information.

When three backend servers (server S, server D, and server U) are up and running, each backend server should read the corresponding input file (single.txt, double.txt, and suites.txt) and store the information in a certain data structure. You can choose any data structure that accommodates your needs. After storing all the data, server S, server D, and server U should then send all the room

statuses they have to the main server via UDP over the port mentioned in the PORT NUMBER ALLOCATION section. Since the room statuses are unique, the main server will maintain a list of room statuses corresponding to each backend server. In the following phases, you have to make sure that the correct backend server is being contacted by the main server for corresponding room statuses. You should print correct on-screen messages onto the screen for the main server and the backend servers indicating the success of these operations as described in the “ON-SCREEN MESSAGES" section.

After the servers are booted up and the required room statuses are transferred from the backend servers to the main server, the client will be started. Once the client boots up and the initial boot-up messages are printed, the client waits for the user to check the authentication, login, and enter the room layout code.

Please check Table 3 and 4 Client on-screen messages for the on-screen message of different events You should store the above room statuses. Once you have the book statuses list stored in your backend server and send the book code list of each backend server to the main server, you can consider phase 1 of the project to be completed. You can proceed to phase 2.

Phase 2: Log in and confirmation

In this phase, the client will be asked to enter the username and password on the terminal. There are two types of clients: *guest* and *member*. A *member* can be authenticated by the Dormitory Reservation System by inputting the member’s username and password. The client will encrypt this information and forward this request to the Main server. The Main server would have all the encrypted credentials (both username and password would be encrypted) of the registered users. Still, it would not have any information about the encryption scheme. The information about the encryption scheme would only be present on the client side. A *guest* can input the username while skipping the password input, but a guest can only query the room status but cannot reserve a room.

The encryption scheme for member authentication would be as follows:

- Offset each character and/or digit by 3.
    - character: cyclically alphabetic (A-Z, a-z) update for overflow
    - digit: cyclically 0-9 update for overflow
- The scheme is case-sensitive.
- Special characters (including spaces and/or the decimal point) will not be encrypted or changed.

A few examples of encryption are given below:

Example	Original Text	Encrypted Text
#1	Welcome to EE450!	Zhofrph wr HH783!
#2	199xyz@$	422abc@$
#3	0.27#&	3.50#&

Constraints:

- The username will be of lower case characters (5~50 chars).
- The password will be case sensitive (5~50 chars)

Phase 2A:

A member client sends the authentication request to the main server over TCP connection. Upon running the client using the following command, the user will be prompted to enter the username and password. This unencrypted information will be encrypted at client side and then sent to the main server over TCP. A guest client can skip the password authentication and directly login.

```
./client
(Please refer to the on-screen messages)
Please enter the username: <unencrypted_username>
Please enter the password ("Enter" to skip for guests): <unencrypted_password>
```

Phase 2B:

Main server receives the encrypted username and password from the client. ServerM sends the result of the authentication request to the client over a TCP connection. If the login information was not correct/found:

```
./client
(Please refer to the on-screen messages)
Please enter the username: <unencrypted_username>
Please enter the password: <unencrypted_password>
Failed login. Invalid username/password
```

After the successful login:

```
Welcome member/guest <username>!
Please enter the room layout code: <roomcode>
```

Multiple clients

In phase 2, Main server will have to receive requests from both the clients. For a server to receive requests from several clients at the same time, the function **fork()** should be used for the creation of a new process. Fork() function is used for creating a new process, which is called ***child process***, which runs concurrently with the process that makes the fork() call (***parent process***).

For a TCP server, when an application is listening for stream-oriented connections from other hosts, it is notified of such events and must initialize the connection using accept(). After the connection with the client is successfully established, the accept() function returns a non-zero descriptor for a socket called the child socket. The server can then fork off a process using fork() function to handle connection on the new socket and go back to waiting on the original socket. Note that the socket that was originally created, that is the parent socket, is going to be used only to listen to the client requests, and it is not going to be used for communication between client and Main server. Child sockets that are created for a parent socket have the identical well-known port number IP address at the server side, but each child socket is created for a specific client. Through using the child socket with the help of fork(), the server can handle the two clients without closing any one of the connections.

Phase 3: Forwarding request to Backend Servers

Phase 3A: Query

Both a member client and a guest client can query the current statues of a specific room layout. Upon user input of a room code for a type and layout, the client is responsible for transmitting the request to the server M (the Main server) via a TCP connection. The server M parses the received roomcode to determine the appropriate destination server for request forwarding. Specifically, when the roomcode commences with "S," the request must be routed to Server S. Similarly, if the roomcode initiates with "D," the request is directed to Server D. In the event that the roomcode originates with "U," the request must be forwarded to Server U. All the valid book codes are eligible for forwarding to their respective servers from the Server M via a UDP connection.

RoomCode from Client	Source Server	Destination Server
S146	Server M	Server S
D111	Server M	Server D
U211	Server M	Server U
A111	Server M	None

Phase 3B: Reservation

Only the member client can reserve a room. Each server will have a dedicated database file. This file should be read only once at server startup to ensure that if a user reserves a room, the corresponding book's inventory count is updated accurately in the respective data structure and must not be overwritten by reading the database file over and over. The updated room number will be printed on the member client screen.

If a guest is requesting a reservation, a “permission denied” prompt will show on screen.

Phase 4: Reply

The corresponding room type server will check its input file and find the count of the requested roomcode. If the count is greater than 0, then the respective server will reply to the main server using UDP - "The requested room is available". And if the count of the room is 0, then the server will reply to the main server using UDP - "The requested room is not available". It is also possible that the roomcode entered by the client is not there in the system, in that case, the server will respond with a message - "Not able to find the room layout".

For a reservation request, after sending the reply to the main server the room type server will decrement the count of the corresponding roomcode by 1 so that when a client requests a book a second time, the availability is updated and correct. And at last, the main server will print the on-screen message and will forward the reply from the room type server to the client using TCP. And the client will print the on screen message which gives the availability of the requested roomcode.

See ON-SCREEN MESSAGES table for more details.

Extra credit:

In this section, we will enhance the security of this system by applying more powerful encryption algorithms. You are encouraged to search the Internet for any existing security encryption/decryption algorithms to substitute the simple character shifting method in Phase 2, or you can even create some complicated methods as long as the transmitted message is not in plaintext. If you use symmetric or asymmetric encryption with public/private keys, you can assume they are known by the host and servers, and the keys can be hardcoded in the code.

To get full credit for this part, you are required to explain the algorithm with all details as well as show an example of the original text and encrypted text in the **Readme.txt** file (you can create an extra credit section), and you also need to provide clear instructions to tell graders how to compile and execute the system using upgraded encryption algorithms. (For example, you can add an argument to programs indicating which encryption protocol you are using, as shown in the sample code below. Then in Makefile, you can create another entry — "make extra" to compile and run the program using this more secure and complicated encryption/decryption algorithm.)

```
int main( int argc, char *argv[] )  {
   if( argc == 1 ) {
      printf("no arguments, use character shifting protocol\n");
   }
   else if( argc == 2 ) {
      printf("Using encryption protocol %s.\n", argv[1]);
   }
}
```

Process Flow/Sequence of Operations:

- Your project grader will start the servers in this sequence: serverM, serverS, serverD, serverU, and two Clients in 6 different terminals.
- Once all the ends are started, the servers and clients should be continuously running unless stopped manually by the grader or meet certain conditions as mentioned before.

Required Port Number Allocation

The ports to be used by the clients and the servers for the exercise are specified in the following table (Major points will be lost if the port allocation is not as per the below description):

Static and Dynamic assignments for TCP and UDP ports.		
Process	**Dynamic Ports**	**Static Ports**
serverS	-	UDP, 41000+xxx
serverD	-	UDP, 42000+xxx
serverU	-	UDP, 43000+xxx
serverM	-	UDP (with servers), 44000+xxx TCP (with clients), 45000+xxx
clients	2 TCPs	

NOTE: xxx is the last 3 digits of your USC ID. For example, if the last 3 digits of your USC ID are “319”, you should use the port: 41000+319 = 41319 for the Backend-Server (A). It is NOT going to be 41000319. Note that the serverM has only one UDP port. The same port is used to connect to all the backend servers.

ON-SCREEN MESSAGES

Table 1. Backend-Server S/D/U on-screen messages

Event	**On-screen Messages**
Booting up (Only while starting):	The Server <S/D/U> is up and running using UDP on port <server S/D/U port number>.
Sending the room status to Main Server:	The Server <S/D/U> has sent the room status to the main server.
(a) Availability query	
After receiving an availability request from Main Server	The Server <S/D/U> received an availability request from the main server.
If the count of the room is greater than 0	Room <roomcode> is available.
If the count of the room is 0	Room <roomcode> is not available.
If the room code is not in the system	Not able to find the room layout.
After sending the results to Main Server	The Server <S/D/U> finished sending the response to the main server.
(b) Reservation query	
After receiving a reservation request from Main Server	The Server <S/D/U> received a reservation request from the main server.
If the count of the room is greater than 0	Successful reservation. The count of Room <roomcode> is now <updated_count>.
If the count of the room is 0	Cannot make a reservation. Room <roomcode> is not available.
If the room code is not in the system	Cannot make a reservation. Not able to find the room layout.
After sending the results to Main Server (if the count of the room changes)	The Server <S/D/U> finished sending the response and the updated room status to the main server.
After sending the results to Main Server (if the count of the room does not change)	The Server <S/D/U> finished sending the response to the main server.

Table 2. Main Server on-screen messages

Event	**On-screen Messages**
Booting up:	The main server is up and running.
Upon receiving the room status from Server S/D/U	The main server has received the room status from Server <S/D/U> using UDP over port <Main server UDP port number>.
(a) Member	
After receiving the username and password from the ***member***	The main server received the authentication for <username> using TCP over port <main server TCP port number>.
Upon sending an authentication response to the client	The main server sent the authentication result to the client.
(b) Guest	
After receiving the username from the ***guest***	The main server received the guest request for <username> using TCP over port <main server TCP port number>. The main server accepts <username> as a guest.
Upon sending a guest response to the client	The main server sent the guest response to the client.
(a) Availability request	
Upon receiving the input from the client for availability	The main server has received the availability request on Room <roomcode> from <username> using TCP over port <Main server TCP port number>.
After forwarding the request to Server <S/D/U>	The main server sent a request to Server <S/D/U>.
After receiving the result from Server <S/D/U>	The main server received the response from Server <S/D/U> using UDP over port <Main server UDP port number>.
After forwarding the result to the client	The main server sent the availability information to the client.
(b) Reservation request	
Upon receiving the input from	The main server has received the reservation request on Room

the client for the reservation	<roomcode> from <username> using TCP over port <Main server TCP port number>.
Error message (if the client is a guest, not a member)	<username> cannot make a reservation.
After sending an error message to the client	The main server sent the error message to the client.
After forwarding the request to Server <S/D/U>	The main server sent a request to Server <S/D/U>.
After receiving the result and updated room status from Server <S/D/U> (if the count of the room has been updated)	The main server received the response and the updated room status from Server <S/D/U> using UDP over port <Main server UDP port number>.
After receiving the result from Server <S/D/U> (if the count of the room does not change)	The main server received the response from Server <S/D/U> using UDP over port <Main server UDP port number>.
After updating the room status	The room status of Room <roomcode> has been updated.
After forwarding the result to the client	The main server sent the reservation result to the client.

Table 3. Member Client on-screen messages

Event	**On-screen Messages**
Booting up(only while starting)	Client is up and running.
Asking the user to enter the username	Please enter the username: <unencrypted_username>
Asking the user to enter the password	Please enter the password: <unencrypted_password>
Upon sending an authentication request to Main Server	<username> sent an authentication request to the main server.
After receiving the result of the authentication request from Main Server (if the authentication passed)	Welcome member <username>!

After receiving the result of the authentication request from Main Server (username does not exist)	Failed login: Username does not exist.
After receiving the result of the authentication request from Main Server (username does not exist)	Failed login: Password does not match.
Asking the user to input the room code	Please enter the room code: <roomcode>
Asking the user to choose the desired action	Would you like to search for the availability or make a reservation? (Enter "Availability" to search for the availability or Enter "Reservation" to make a reservation): <Availability or Reservation>
(a) Availability request	
Upon sending an availability request to Main Server	<username> sent an availability request to the main server.
After receiving the response from Main Server (the count is greater than 0)	The client received the response from the main server using TCP over port <client port number>. The requested room is available. <Leave one blank line> -----Start a new request-----
After receiving the response from Main Server (the count is 0)	The client received the response from the main server using TCP over port <client port number>. The requested room is not available. <Leave one blank line> -----Start a new request-----
After receiving the response from Main Server (room code is not in the system)	The client received the response from the main server using TCP over port <client port number>. Not able to find the room layout. <Leave one blank line> -----Start a new request-----
(b) Reservation request	
Upon sending a reservation request to Main Serves	<username> sent a reservation request to the main server.
After receiving the response from Main Server (the count is greater than 0)	The client received the response from the main server using TCP over port <client port number>. Congratulation! The reservation for Room <roomcode>

	has been made. <Leave one blank line> -----Start a new request-----
After receiving the response from Main Server (the count is 0)	The client received the response from the main server using TCP over port <client port number>. Sorry! The requested room is not available. <Leave one blank line> -----Start a new request-----
After receiving the response from Main Server (room code is not in the system)	The client received the response from the main server using TCP over port <client port number>. Oops! Not able to find the room. <Leave one blank line> -----Start a new request-----

Table 4. Guest Client on-screen messages

Event	**On-screen Messages**
Booting up(only while starting)	Client is up and running.
Asking the user to enter the username	Please enter the username: <unencrypted_username>
Asking the user to enter the password	Please enter the password: (Press "Enter" to skip)
Upon continuing as a guest	<username> sent a guest request to the main server using TCP over port <client port number>.
After receiving the response from Main Server	Welcome guest <username>!
Asking the user to input the room code	Please enter the room code: <roomcode>
Asking the user to choose the desired action	Would you like to search for the availability or make a reservation? (Enter "Availability" to search for the availability or Enter "Reservation" to make a reservation): <Availability or Reservation>
(a) Availability request	
Upon sending an availability	<username> sent an availability request to the main

request to Main Server	server.
After receiving the response from Main Server (the count is greater than 0)	The client received the response from the main server using TCP over port <client port number>. The requested room is available. <Leave one blank line> -----Start a new request-----
After receiving the response from Main Server (the count is 0)	The client received the response from the main server using TCP over port <client port number>. The requested room is not available. <Leave one blank line> -----Start a new request-----
After receiving the response from Main Server (room code is not in the system)	The client received the response from the main server using TCP over port <client port number>. Not able to find the room layout. <Leave one blank line> -----Start a new request-----
(b) Reservation request	
Upon sending a reservation request to Main Serves.	<username> sent a reservation request to the main server.
After receiving the response from Main Server.	Permission denied: Guest cannot make a reservation.

Example Output to Illustrate Output Formatting

Server S Terminal:

The Server S is up and running using UDP on port 41319.
The Server S has sent the room status to the main server.
The Server S received an availability request from the main server.
Room S146 is available.
The Server S finished sending the response to the main server.

Server D Terminal:

The Server D is up and running using UDP on port 42319.
The Server D has sent the room status to the main server.
The Server D received an availability request from the main server.
Room D111 is not available.
The Server D finished sending the response to the main server.

Server U Terminal:

The Server U is up and running using UDP on port 43319.
The Server U has sent the room status to the main server.
The Server U received a reservation request from the main server.
Successful reservation. The count of Room U211 is now 1.
The Server U finished sending the response and the updated room status to the main server.

Main Server Terminal:

The main server is up and running.
The main server has received the room status from Server S using UDP over port 44319.
The main server has received the room status from Server D using UDP over port 44319.
The main server has received the room status from Server U using UDP over port 44319.
The main server received the authentication for james using TCP over port 45319.
The main server sent the authentication result to the client.
The main server has received the availability request on Room S146 from james using TCP over port 45319.
The main server sent a request to Server S.
The main server received the response from Server D using UDP over port 44319.
The main server sent the availability information to the client.
The main server has received the availability request on Room D111 from james using TCP over port 45319.
The main server sent a request to Server D.
The main server received the response from Server D using UDP over port 44319.
The main server sent the availability information to the client.
The main server has received the reservation request on Room U211 from james using TCP over port 45319.
The main server sent a request to Server U.
The main server received the response and the updated room status from Server U using UDP over port 44319.
The room status of Room U211 has been updated.
The main server sent the reservation result to the client.

Client Terminal:

Client is up and running.
Please enter the username: james
Please enter the password: 2kAnsa7s
james sent an authentication request to the main server.
Welcome member james!
Please enter the room code: S146
Would you like to search for the availability or make a reservation? (Enter “Availability” to search for the availability or Enter “Reservation” to make a reservation): Availability
james sent an availability request to the main server.
The client received the response from the main server using TCP over port <port number>.
The requested room is available.

-----Start a new request-----

Please enter the room code: D111
Would you like to search for the availability or make a reservation? (Enter “Availability” to search for the availability or Enter “Reservation” to make a reservation): Availability
james sent an availability request to the main server.
The client received the response from the main server using TCP over port <port number>.
The requested room is not available.

-----Start a new request-----
Please enter the room code: U211
Would you like to search for the availability or make a reservation? (Enter “Availability” to search for the availability or Enter “Reservation” to make a reservation): Reservation
james sent a reservation request to the main server.
The client received the response from the main server using TCP over port <port number>.
Congratulation! The reservation for Room U211 has been made.

-----Start a new request-----
Please enter the room code: <Ctrl C>

(***Note:*** You should replace <port number> with the TCP port number dynamically assigned by the system.)

SUBMISSION FILES

Submission File and Folder Structure:

Your submission should have the following folder structure and the files (the examples are of .cpp, but it can be .c files as well):

- ee450_lastname_firstname_uscusername.tar.gz
    - ee450_lastname_firstname_uscusername
        - client.cpp
        - serverM.cpp
        - serverS.cpp
        - serverD.cpp
        - serverU.cpp
        - Makefile
        - readme.txt (or) readme.md
        - <Any additional header files>

The grader will extract the tar.gz file, and will place all the input data files in the same directory as your source files. The executable files should also be generated in the same directory as your source files. So, after testing your code, the folder structure should look something like this:

- ee450_lastname_firstname_uscusername
    - client.cpp
    - serverM.cpp
    - serverS.cpp
    - serverD.cpp
    - serverU.cpp
    - Makefile
    - readme.txt (or) readme.md
    - client
    - serverM
    - serverS
    - serverD
    - serverU
    - member.txt
    - single.txt
    - double.txt
    - suites.txt
    - <Any additional header files>

Note that in the above example, the input data files (member.txt, single.txt, double.txt and suites.txt) will be manually placed by the grader, while the 'make all' command should generate the executable files.

Assumptions

1. You have to start the processes in this order: serverM, serverS, serverD, serverU, and client. If you need to have more code files than the ones that are mentioned here, please use meaningful names and all small letters and mention them all in your README file.

2. You are allowed to use blocks of code from Beej's socket programming tutorial (Beej's guide to network programming) in your project. However, you need to cite the copied part in your code. Any signs of academic dishonesty will be taken very seriously.

3. When you run your code, if you get the message "port already in use" or "address already in use", **please first check to see if you have a zombie process**. If you do not have such zombie processes or if you still get this message after terminating all zombie processes, try changing the static UDP or TCP port number corresponding to this error message (all port numbers below 1024 are reserved and must not be used). If you have to change the port number, please do mention it in your README file and provide reasons for it.

Requirements

1. Do not hardcode the TCP or UDP port numbers that are to be obtained dynamically. Refer to Table Port Number Allocation to see which ports are statically defined and which ones are dynamically assigned. Use getsockname() function to retrieve the locally-bound port number wherever ports are assigned dynamically as shown below:

   ```
   /*Retrieve the locally-bound name of the specified socket and store it in the sockaddr structure*/

   Getsock_check=getsockname(TCP_Connect_Sock, (struct sockaddr*)&my_addr, (socklen_t *)&addrlen);

   //Error checking
   if (getsock_check== -1) {
           perror("getsockname");
           exit(1);
   }
   ```

2. The host name must be hard coded as **"localhost" or "127.0.0.1"** in all codes.

3. Your client, the backend servers and the main server should keep running and be waiting for another request until the TAs terminate them by Ctrl+C. If they terminate before that, you will lose some points for it.

4. All the naming conventions and the on-screen messages must conform to the previously mentioned rules.

5. You are not allowed to pass any parameter or value or string or character as a command-line argument except what is already described in the project document.

6. All the on-screen messages must conform exactly to the project description. You should not add anymore on-screen messages. If you need to do so for the debugging purposes, you must comment out all of the extra messages before you submit your project.

7. Please do remember to close the socket and tear down the connection once you are done using that socket.

Programming platform and environment

1. All your submitted code MUST work well on the provided virtual machine Ubuntu.

2. All submissions will only be graded on the provided Ubuntu. TAs/Graders won't make any updates or changes to the virtual machine. It's your responsibility to make sure your code works well on the provided Ubuntu. **"It works well on my machine" is not an excuse.**

3. Your submission MUST have a Makefile. Please follow the requirements in the following "Submission Rules" section

Programming languages and compilers

You must use only C/C++ on UNIX as well as UNIX Socket programming commands and functions. Here are the pointers for Beej's Guide to C Programming and Network Programming (socket programming):

http://www.beej.us/guide/bgnet/

(If you are new to socket programming please do study this tutorial carefully as soon as possible and before starting the project)

http://www.beej.us/guide/bgc/

You can use a unix text editor like emacs to type your code and then use compilers such as g++ (for C++) and gcc (for C) that are already installed on Ubuntu to compile your code. You must use the following commands and switches to compile yourfile.c or yourfile.cpp. It will make an executable by the name of "yourfileoutput".

```
gcc -o yourfileoutput yourfile.c
g++ -o yourfileoutput yourfile.cpp
```

Do NOT forget the mandatory naming conventions mentioned before!

Also inside your code you need to include these header files in addition to any other header file you think you may need:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/wait.h>
```

Submission Rules

Along with your code files, include a README file and a Makefile. The Makefile requirements are mentioned in the section below. The README file can be in any format (Markdown or txt). The only requirement is that the TAs should be able to open the file and read it in the studentVM / the VM your project will be graded in without installing any additional software. In the README file please include:

a. Your **Full Name** as given in the class list
b. Your Student ID
c. What you have done in the assignment, if you have completed the optional part (extra credit). If it's not mentioned, it will not be considered.
d. What your code files are and what each one of them does. (Please do not repeat the project description, just name your code files and briefly mention what they do).
e. The format of all the messages exchanged, e.g., username and password are concatenated and delimited by a comma, etc.
f. Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
g. Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code). Reusing functions which are directly obtained from a source on the internet without or with few modifications is considered plagiarism (Except code from the Beej's Guide). Whenever you are referring to an online resource, make sure to only look at the source, understand it, close it and then write the code by yourself. The TAs will perform plagiarism checks on your code so make sure to follow this step rigorously for every piece of code which will be submitted.

Submissions WITHOUT README and Makefile WILL NOT BE GRADED.

Makefile

https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html

About the Makefile: makefile should support following functions:

make all	Compiles all your files and creates executables
make clean	Removes all the executable files
./serverM	Runs Main server
./serverS	Runs Backend server S
./serverD	Runs Backend server D
./serverU	Runs Backend server U
./client	Runs the client
(extra credit, optional) make extra	(optional, you should show clearly how to run the extra credit code in every details in Readme.txt file) Compiles all your files again and creates executables, and run all exes using your implementation of encryption/decryption algorithm.

TAs will first compile all codes using **make all**. They will then open 6 different terminal windows. On 4 terminals they will start servers M, S, D and U. On the two other terminals, they will start the clients using ./client. Remember that all programs should always be on once started. TAs will check the outputs for multiple values of input. The terminals should display the messages shown in On-screen Messages tables in this project writeup.

Compress all your files including the README file and the Makefile into a single "tar ball" and call it: ee450_YourLastName_YourFirstName_yourUSCusername.tar.gz (all small letters) e.g. an example filename would be ee450_trojan_tommy_tommyt.tar.gz. Please make sure that your name matches the one in the class list.

Here are the instructions:
On your VM, go to the directory which has all your project files. Remove all executable and other unnecessary files. **Only include the required source code files, Makefile and the README file**. Now run the following two commands:
*tar cvf ee450_YourLastName_YourFirstName_yourUSCusername.tar **
gzip ee450_YourLastName_YourFirstName_yourUSCusername.tar

Now, you will find a file named
"ee450_YourLastName_YourFirstName_yourUSCusername.tar.gz" in the same directory.
Please notice there is a space and a star(*) at the end of the first command.

An example submission would be:
First Name: John
Last Name: Doe

USC username: jdoe (This can be found with your email address, In this case the email address would be jdoe@usc.edu)
So the submission would be:
ee450_Doe_John_jdoe.tar.gz
Any compressed format other than .tar.gz will NOT be graded!

Upload "ee450_YourLastName_YourFirstName_yourUSCusername.tar.gz" to the Digital Dropbox on the DEN website (DEN -> EE450 -> My Tools -> Assignments -> Project). After the file is uploaded to the dropbox, you must click on the "Submit" button to actually submit it. If you do not click on "Submit", the file will not be submitted.

D2L will keep a history of all your submissions. If you make multiple submissions, we will grade your latest valid submission. Submission after the deadline is considered as invalid.

D2L will send you a "Dropbox submission receipt" to confirm your submission. So please do check your emails to make sure your submission is successfully received. If you have not received a confirmation mail, contact your TA if it always fails.

After receiving the confirmation email, please confirm your submission by downloading and compiling it on your machine. This is exactly what your designated TA would do, So please grade your own project from the perspective of the TA. If the outcome is not what you expected, try to resubmit and confirm again. We will only grade what you submitted even though it's corrupted.

Please take into account all kinds of possible technical issues and do expect a huge traffic on the DEN website very close to the deadline which may render your submission or even access to DEN unsuccessful.

Please DO NOT wait till the last 5 minutes to upload and submit because some technical issues might happen and you will miss the deadline. And a kind suggestion, if you still get some bugs one hour before the deadline, please make a submission first to make sure you will get some points for your hard work!

There is absolutely zero tolerance for late submissions! Do NOT assume that there will be a late submission penalty or a grace period. If you submit your project late (no matter for what reason or excuse or even technical issues), you simply receive a ZERO for the project.

Grading Criteria

Notice: We will only grade what is already done by the program instead of what will be done. The grading criteria are subject to change.

Your project grade will depend on the following:

1. Correct functionality, i.e. how well your programs fulfill the requirements of the assignment, specially the communications through UDP and TCP sockets.

2. Inline comments in your code. This is important as this will help in understanding what you have done.

3. Whether your programs work as you say they would in the README file.

4. Whether your programs print out the appropriate error messages and results.

5. **Your code will only be tested on a fresh copy of the provided Virtual Machine (either studentVM (64-bit) or Ubuntu 22.04 ARM64 for M1/M2 Mac users). If your programs are not compiled or executed on these VM, you will receive only minimum points as described below. Be careful if you are going to use other environments!!! Do not update or upgrade the provided VM as well!!!**

6. If your submitted codes do not even compile, you will receive 5 out of 100 for the project.

7. If your submitted codes compile using make but when executed, produce runtime errors without performing any tasks of the project, you will receive 10 out of 100.

8. The minimum points for compiled and executable codes is 15 out of 100.

9. If your code does not correctly assign the TCP or UDP port numbers (in any phase), you will lose points each.

10. We will use similar test cases to test all the programs. These test cases cover all situations including edge cases.

11. There are no points for the effort or the time you spend working on the project or reading the tutorial. If you spend about 2 weeks on this project and it doesn't even compile, you will receive only 5 out of 100.

12. You must discuss all project related issues on the Piazza Discussion Forum. We will give those who actively help others out by answering questions on Piazza up to 10 bonus points. (If you want to earn the extra credits, do remember to leave your names visible to instructors when answering questions on D2L.)

13. The maximum points that you can receive for the project with bonus points and extra credits is 110.

14. Your code will not be altered in any way for grading purposes and however it will be tested with different inputs. Your TA/Grader runs your project as is, according to the project description and your README file and then checks whether it works correctly or not. If your README is not consistent with the description, we will follow the description.

Cautionary Words

1. Start on this project early!!!

2. In view of what is a recurring complaint near the end of a project, we want to make it clear that the target platform on which the project is supposed to run is studentVM (64-bit) or Ubuntu 22.04 ARM64 for M1/M2 Mac users. It is strongly recommended that students develop their code on this virtual machine. In case students wish to develop their programs on their personal machines, possibly running other operating systems, they are expected to deal with technical and incompatibility issues (on their own) to ensure that the final project compiles and runs on the requested virtual machine. If you do development on your own machine, please leave at least three days to make it work on Ubuntu. It might take much longer than you expect because of some incompatibility issues.

3. You may create zombie processes while testing your codes, please make sure you kill them every time you want to run your code. To see a list of all zombie processes, try this command:
   >>ps –aux | grep ee450
   Identify the zombie processes and their process number and kill them by typing at the command-line: **>>kill -9 processnumber**

Academic Integrity

All students are expected to write all their code on their own!!!

Do not post your code on Github, especially in a public repository before the deadline!!!

Double check the setting and do some testing before posting in a private repository!!!

Copying code from friends or from any unauthorized resources (webpages, github, etc.) is called **plagiarism** not **collaboration** and will result in an F for the entire course. Any libraries or pieces of code that you use or refer and you did not write must be listed in your README file. Students are only allowed to use the code from Beej's socket programming tutorial. Copying the code from any other resources may be considered as plagiarism. Please be careful!!! All programs will be compared with automated tools to detect similarities; examples of code copying will get an F for the course. **IF YOU HAVE ANY QUESTIONS ABOUT WHAT IS OR ISN'T ALLOWED ABOUT PLAGIARISM, TALK TO THE TA**. "I didn't know" is not an excuse.