EE450 Socket Programming Project, Fall 2022

Due Date : Monday, Nov 28, 11:59PM (Midnight)

(The deadline is the same for all on-campus and DEN off-campus students)

Hard Deadline (Strictly enforced)

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:

Web registration system has been a critical system for USC students. It is the key place for students to plan their future course of study and plays an important role in students' academic success. Imagine that one day the web registration system is gone and tens of thousands of students are left unknown about what to choose for the next semester. A course might be over-crowded with hundreds of students because students don't know how many students have already registered and the administrator might have to randomly select hundreds of people to drop from that course. Or you can imagine on the first day of the semester, the web registration system is suddenly down, all students are anxiously refreshing their webpage. And another thing to consider is the security, keep in mind that our web registration system should have a kind of authorization, for example username and password. Otherwise, it will be a popular hoax among students to drop courses for others. Thus a secure, reliable, functional and informative web registration system is vital for our school. As a networking course, we will try to approach this problem with some simplification. We will make the assumption that only one student will access the web registration system each time, and there are only two departments of courses to choose from. We will also introduce a very simple authorization schema.

In this project, you will implement a simple web registration system for USC. Specifically, a student will use the client to access the central web registration server, which will forward their requests to the department servers in each department. For each department, the department server will store the information of the courses offered in this department. Additionally, a credential server will be used to verify the identity of the student.

There are total 5 communication end-points:

- Client: used by a student to access the registration system.
- Main server (serverM): coordinate with the backend servers.
- Credential server (serverC): verify the identity of the student.
- Department server(s) (serverCS and serverEE)): store the information of courses offered by this department.

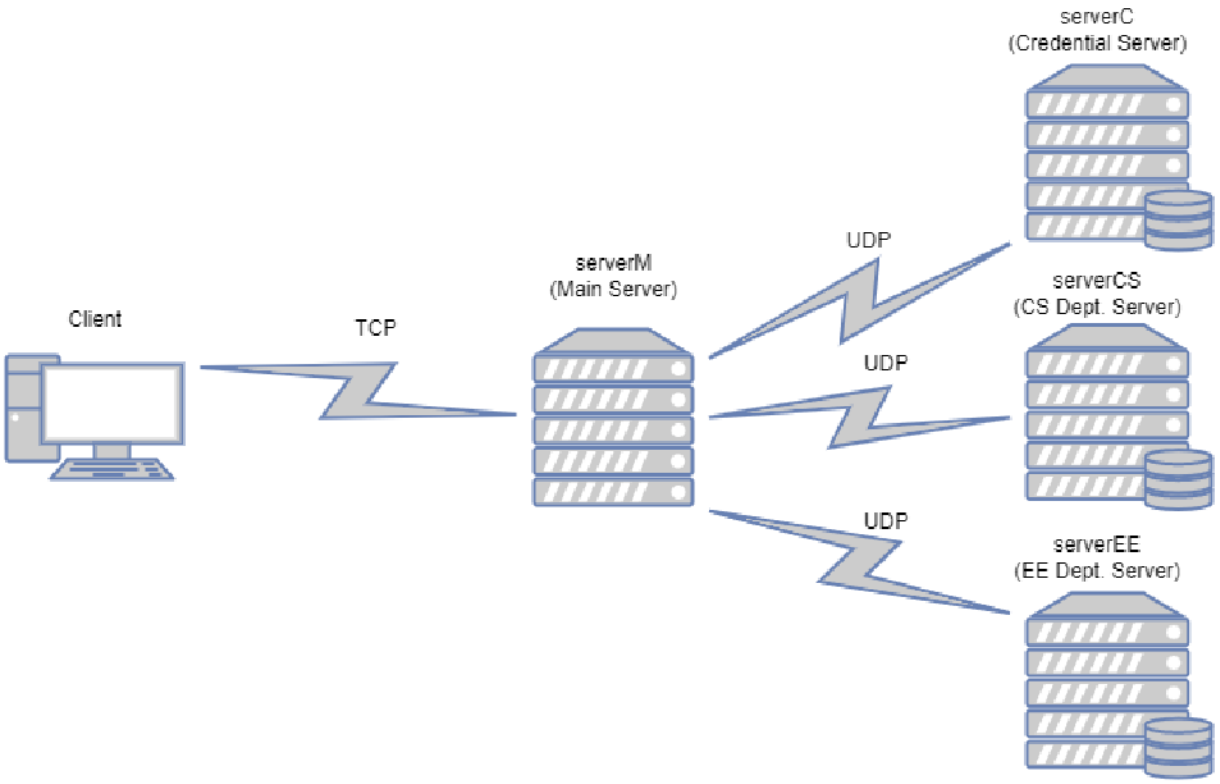

Figure 1. Illustration of the network

For the backend servers, Credential server and Department servers will access corresponding files on disk, and respond to the request from the main server based on the file content. It is important to note that only the corresponding server should access the file. It is prohibited to access a file in the main server or other servers. We will use both TCP and UDP connections. However, we assume that all the UDP packages will be received without any error.

Source Code Files

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

1. 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').

2. Backend-Servers C, CS and EE: You must use one of these names for this piece of code: **server#.c** or **server#.cc** or **server#.cpp** (all small letters except for #). Also you must include the corresponding header file (if you have one; it is not mandatory). **server#.h** (all small letters, except for #). The "#" character must be replaced by the server identifier (i.e. C or CS or EE), depending on the server it corresponds to. (e.g., serverC.cpp, serverEE.cpp & serverCS.cpp)

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

3. 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).

Input Files:

There are three input files that are given to the credential Server and two department servers and should be read by the server when it is up and running.

- **cred.txt**: contains encrypted usernames and passwords. This file should only be accessed by the Credential server.
- **ee.txt**: contains course information categorized in course code, credit, professor, days and course name. Different categories are separated by a comma. There could be space(s) or semicolons, except commas, in a category. One example is given below. This file should only be accessed by the EE Department server.

  ```
  EE450,4,Ali Zahid,Tue;Thu,Introduction to Computer Networks
  ```

- **cs.txt**: Same format as ee.txt. This file should only be accessed by the CS Department server.

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

Application Workflow Phase Description:

Phase 0:

Please refer to the Process Flow section to start the main server, Credential server, EE Department server, CS Department server and Client in order. Upon three backend servers (Credential server, CS Department server and EE Department server) are up and running, each backend server should read the corresponding file and store the information in a certain data structure. You can choose any data structure that accommodates the needs.

Phase 1: (30 points)

In this phase, you will be authenticating the credentials of the client. The client will be asked to enter the username and password on the terminal. The client will forward the request to the main server, and the main server will encrypt this information and again forward this request to the credential server. The credentials server would have all the encrypted credentials (both username and password would be encrypted) of the registered users, but it would not have any information about the encryption scheme. The information about the encryption scheme would only be present at the main server. The encryption scheme would be as follows:

- Offset each character and/or digit by 4.
- The scheme is case sensitive.
- Special characters (including spaces and/or the decimal point) will not be encrypted or changed.

Few examples of encryption are given below:

Example	Original Text	Cipher Text
#1	Welcome to EE450!	Aipgsqi xs II894!
#2	199@$	533@$
#3	0.27#&	4.61#&

Phase 1A: Client sends the authentication request to the main server over TCP connection.

Constraints:

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

Upon running the client using the following command, the user will be prompted to enter the username and password:

./client
Please enter the username: <unencrypted_username>
Please enter the password: <unencrypted_password>

This unencrypted information will be sent to the main server over TCP, the main server will encrypt this information and send it to the credential server, which takes us to phase-1B.

Phase 1B: Main server forwards the authentication request to the credentials server over UDP.

Phase 2: (20 points)

Phase 2A: serverC sends the result of the authentication request to serverM over a UDP connection.

In this phase we check the result of the authentication request sent to the serverC and communicate the result back to serverM. Once we receive the authentication request at serverC, The authentication request will contain the encrypted form of the username and the password. At serverC, once the authentication request is received, the serverC should first check if the username in the authentication request matches with any of the usernames present in the cred.txt file. If the username exists, it secondly checks if the password in the authentication request is the same as the password corresponding to the same username in the cred.txt file (It should be a case-sensitive match). If both the checks are passed then serverC sends an authentication pass message to the serverM using UDP. If either of the conditions fail then the serverC sends an authentication fail message to the serverM (along with the reason for failure - username does not exist or password does not match). You can use any type of encoding to notify the main server that the authentication has passed or failed. For example: Send 0 to serverM to indicate that the authentication request has failed (no username), send 1 to serverM to indicate that the authentication request has failed (password does not match) and 2 to serverM to indicate that the authentication request has passed. Or PASS indicates that the authentication request has passed, FAIL_NO_USER indicates that the authentication request has failed (no username), and FAIL_PASS_NO_MATCH indicates that the authentication request has failed (password does not match).

Phase 2B: serverM sends the result of the authentication request to the client over a TCP connection.

The result of the authentication request (Pass or Fail-with reason) is sent to the client from the main server over TCP (any encoding can be used similar to Phase 2A) and the result is displayed on the client screen. Please check the on-screen messages section for further information. If the result of the authentication request is a failure then the client will have two more attempts to enter the correct username and password (a total of 3 attempts). If the authentication request fails (in the first or second attempt), Phases 1A, 1B, 2A and 2B have to be repeated. If all of the attempts fail then the client shuts down after indicating that all 3 attempts failed (Please check the on-screen messages section for further information). If the result of the authentication request is Pass then the client can move to Phase 3. The client shuts down only if all the 3 authentication attempts have failed. If any of the authentication attempts pass then the client stays on until it is manually shut down.

Phase 3: (30 points)

In this phase, you are required to implement sending the request from client to main server and then forwarding the request from the central registration to the backend server.

Phase 3A:

In this part, you will implement the client sending a query to the central registration server. Your client should show a promote of

Please enter the course code to query:

Assuming the student entered:

EE450

And then the client program will promote:

Please enter the category (Credit / Professor / Days / CourseName):

The student will choose which category to search for, they may enter:

Professor

Your client should send this request to the main server via TCP connection. You are allowed to use any type of encoding. For example, you can use integer number 1 to represent EE and 0 to represent CS, or you can just use ascii *EE* and *CS* to represent the two departments. Similarly, you can use integer numbers to represent them. The main server will first extract the department information from the query and decide which department server has the corresponding information.

For the on-screen output, upon sending the request to the main server, your client should output an on-screen message. When receiving the information from the client, your main server should output an on-screen message. **See the ON SCREEN MESSAGES table for details.**

Phase 3B:
In this part, your main server will send the query information to the backend department server via UDP connection. Your main server should output an on-screen message upon sending the request to the backend server. After getting the query information, the department server would look through its stored local information to obtain the corresponding course information.

If the course was founded, print out:
The course information has been founded: The <category> of <course code> is <information>

If not, print out:
Didn't find the course: <course code>.

See the ON SCREEN MESSAGES table for details.

Phase 4: (20 points)

Phase 4B:
At the end of Phase 3, the responsible Department server should have the query information ready. The query information is first sent back to the Main server using UDP and print out an on-screen message.

Phase 4B:
When the Main server receives the result, it needs to print out an on-screen message, forward the result to the Client using TCP and print out an on-screen message.

When the client receives the result, it will print out the query information and the prompt messages for a new request as follows:

The <category> of <course code> is <information>.

-----Start a new request-----
Please enter the course code to query:

See the ON SCREEN MESSAGES table for details.

Extal Credits: (10 points extra, not mandatory)

If you want to earn 10 extra points, you can implement an extra functionality where a user can query N courses (N < 10) at once and receive the corresponding information of all categories from different back-end servers respectively. To be more specific, the input format on the client side is:

<CourseCode1> <CourseCode2> <CourseCode3>.....

Note: the maximum number of CourseCode is less than 10 and there is a whitespace between each CourseCode.

The request is sent to the main server using TCP. After receiving the packet, the main server will parse it and send one or two request(s) to the backend server(s). The one containing EE courses should be sent to the EE Department server, and the other containing CS courses should be sent to the CS Department server. The corresponding server will respond to the main server with all information of all categories. The main server will combine two responses from backend servers together and prepare one response message to the client. The final results shown at the client side should maintain the order of courses that the user input. For example, after the prompt shown on the client's terminal:

Please enter the course code to query:
The client inputs the following CouseCodes:

EE450 EE669 CS402

The following table will be shown on the client's terminal:

CourseCode: Credits, Professor, Days, Course Name
EE450: 4, Ali Zahid, Tue;Thu, Introduction to Computer Networks
EE669: 4, Jay Kuo, Mon;Wed, Multimedia Data Compression
CS402: 4, William Cheng, Mon;Wed, Operating Systems

(***Note***: there is no need to print those vertical and horizontal lines in the above table)

NOTE: The extra points will be added to the full 100 points. The maximum possible points for this socket programming project is 110 points.

Process Flow/ Sequence of Operations:

- Your project grader will start the servers in this sequence: serverM, serverC, serverEE, serverCS, Client in five 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:

Note: Major points will be lost if the port allocation is not as per the below description.

Table 3. Static and Dynamic assignments for TCP and UDP ports.

Process	Dynamic Ports	Static Ports
serverC	-	1 UDP, 21000+xxx
serverCS	-	1 UDP, 22000+xxx
serverEE	-	1 UDP, 23000+xxx
serverM	-	1 UDP, 24000+xxx 1 TCP, 25000+xxx
Client	1 TCP	<Dynamic Port assignment>

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: **21000+319 = 21319** for the Backend-Server (A). **It is NOT going to be 21000319.**

ON SCREEN MESSAGES: Table 4. Backend Department Server on screen messages (For both EE and CS Department Servers)	
Event	**On Screen Message (inside quotes)**
Booting Up (Only while starting):	The Server<EE or CS> is up and running using UDP on port <port number>.
After Receiving the request from main server:	The Server<EE or CS> received a request from the Main Server about the <category> of <course code>.
If the course is found:	The course information has been found: The <category> of <course code> is <information>.
If the course is not found:	Didn't find the course: <course code>.
After sending the results to the main server:	The Server<EE or CS> finished sending the response to the Main Server.

ON SCREEN MESSAGES: Table 6. Backend Credential Server on screen messages	
Event	**On Screen Message (inside quotes)**
Booting Up (Only while starting):	The ServerC is up and running using UDP on port <port number>.
Upon Receiving the request from main server:	The ServerC received an authentication request from the Main Server.
After sending the results to the main server:	The ServerC finished sending the response to the Main Server.

ON SCREEN MESSAGES: Table 7. Main Server on screen messages	
Event	**On Screen Message (inside quotes)**
Booting Up (only while starting):	The main server is up and running.
After receiving the username and password from the client:	The main server received the authentication for <username> using TCP over port <port number>.
Upon sending an authentication request to serverC:	The main server sent an authentication request to serverC.
After receiving result of the authentication request from serverC:	The main server received the result of the authentication request from ServerC using UDP over port <port number>.
After sending the authentication result to the client:	The main server sent the authentication result to the client.
After receiving the query information from the client:	The main server received from <username> to query course <course code> about <category> using TCP over port <port number>.
After querying EE or CS Department Server:	The main server sent a request to server<EE or CS>.
After receiving result from EE or CS Department server i for query information:	The main server received the response from server<EE or CS> using UDP over port <port number>.
After sending the query information to the client:	The main server sent the query information to the client.

ON SCREEN MESSAGES: Table 8. Client on screen messages	
Event	**On Screen Message (inside quotes)**
Booting Up:	The client is up and running.
Asking user to enter username and password:	Please enter the username: <unencrypted_username> Please enter the password: <unencrypted_password>
Upon sending 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):	<username> received the result of authentication using TCP over port <port number>. Authentication is successful
After receiving the result of the authentication request from Main server (username does not exist): n=2,1,0 after the First,second and third attempt respectively	<username> received the result of authentication using TCP over port <port number>. Authentication failed: Username Does not exist Attempts remaining:<n>
After receiving the result of the authentication request from Main server (Password does not match): n=2,1,0 after the First,second and third attempt respectively	<username> received the result of authentication using TCP over port <port number>. Authentication failed: Password does not match Attempts remaining:<n>
After receiving the result of the authentication request from Main server (Failure after 3 attempts):	Authentication Failed for 3 attempts. Client will shut down.
Asking user to input course to query:	Please enter the course code to query:
Asking user to input query category:	Please enter the category (Credit / Professor / Days / CourseName):
Upon sending the request to Main server:	<username> sent a request to the main server.
After receiving the query information from the Main server:	The client received the response from the Main server using TCP over port <port number>.

If the information is successfully found:	The <category> of <course code> is <information>. -----Start a new request----- Please enter the course code to query:
If the course is not found:	Didn't find the course: <course code>. -----Start a new request----- Please enter the course code to query:
(For Extra Credit Only) Upon sending the request to Main Server with multiple course codes:	<username> sent a request with multiple CourseCode to the main server.
(For Extra Credit Only) After receiving all corresponding information for multiple course codes from the Main Server:	CourseCode: Credits, Professor, Days, Course Name <course code>: <credits>, <professor>, <days>, <course name> <course code>: <credits>, <professor>, <days>, <course name> …

Submission files and folder structure:
(Additionally, refer #2 of submission rules for more details)

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**
    - **serverC.cpp**
    - **serverCS.cpp**
    - **serverEE.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**
  - **serverC.cpp**
  - **serverCS.cpp**
  - **serverEE.cpp**
  - **Makefile**
  - **readme.txt (or) readme.md**
  - **client**
  - **serverM**
  - **serverC**
  - **serverCS**
  - **serverEE**
  - **ee.txt**
  - **cs.txt**
  - **cred.txt**
  - **<Any additional header files>**

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

Example Output to Illustrate Output Formatting:

Backend-ServerC Terminal:
The ServerC is up and running using UDP on port 21319.
The ServerC received an authentication request from the Main Server.
The ServerC finished sending the response to the Main Server.

Backend-ServerCS Terminal:
The ServerCS is up and running using UDP on port 22319.
The ServerCS received a request from the Main Server about the credit of CS402.
The course information has been found: The credit of CS402 is 4.
The ServerCS finished sending the response to the Main Server.

Backend-ServerEE Terminal:
The ServerEE is up and running using UDP on port 23319.
The ServerEE received a request from the Main Server about the credit of EE450.
The course information has been found: The credit of EE450 is 4.
The ServerEE finished sending the response to the Main Server.

Main Server Terminal:
The main server is up and running.
The main server received the authentication for james using TCP over port 25319.
The main server sent an authentication request to serverC.
The main server received the result of the authentication request from ServerC using UDP over port 24319.
The main server sent the authentication result to the client.
The main server received from james to query course EE450 about credit.
The main server sent a request to serverEE.
The main server received the response from serverEE using UDP over port 24319.
The main server sent the query information to the client.
The main server received from james to query course CS402 about credit.
The main server sent a request to serverCS.
The main server received the response from serverCS using UDP over port 24319.
The main server sent the query information to the client.

Client Terminal:
The client is up and running.
Please enter the username: james
Please enter the password: 2kAnsa7s)
james sent an authentication request to the main server.
james received the result of authentication using TCP over port <port number>. Authentication is successful
Please enter the course code to query: EE450
Please enter the category (Credit / Professor / Days / CourseName): Credit

james sent a request to the main server.
The client received the response from the Main server using TCP over port <port number>.
The credit of EE450 is 4.

-----Start a new request-----
Please enter the course code to query: CS402
Please enter the category (Credit / Professor / Days / CourseName): Credit
james sent a request to the main server.
The client received the response from the Main server using TCP over port <port number>.
The credit of CS402 is 4.

(The following is only for extra credit part)
-----Start a new request-----
Please enter the course code to query: EE450 EE669 CS402
james sent a request with multiple CourseCode to the main server.
The client received the response from the Main server using TCP over port <port number>.
CourseCode: Credits, Professor, Days, Course Name
EE450: 4, Ali Zahid, Tue;Thu, Introduction to Computer Networks
EE669: 4, Jay Kuo, Mon;Wed, Multimedia Data Compression
CS402: 4, William Cheng, Mon;Wed, Operating Systems

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

Assumptions:

1. You have to start the processes in this order: **ServerM, ServerC, ServerEE, ServerCS, 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 3 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 (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:

1. Along with your code files, include a **README file and a Makefile**. In the README file write
   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 (suffix). 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.
   g. Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
   h. 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.)

Submissions WITHOUT README AND Makefile WILL NOT BE GRADED.

Makefile tutorial:

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
./serverM	**Run**s Main server
./serverC	**Run**s Credential server
./serverEE	**Run**s EE Department server
./serverCS	**Run**s CS Department server
./client	Starts the client

TAs will first compile all codes using **make all**. They will then open 5 different terminal windows. On 4 terminals they will start servers M, C, EE and CS. On the other terminal, they will start the client 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.

2. Compress all your files including the README file 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 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 star(*) at the end of the first command.

   Any compressed format other than .tar.gz will NOT be graded!

3. 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 "**send"** button to actually submit it. If you do not click on "**send**", the file will not be submitted.

4. 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.

5. 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 don't receive a confirmation email, try again later and contact your TA if it always fails.

6. 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.

7. 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!

8. After receiving the confirmation email, please confirm your submission by downloading and compiling it on your machine. 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.

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

6. 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.

7. If you forget to include the README file or Makefile in the project tar-ball that you submitted, you will lose 15 points for each missing file (plus you need to send the file to the TA in order for your project to be graded.)

8. If your code does not correctly assign the TCP or UDP port numbers (in any phase), you will lose 10 points each.

9. You will lose 5 points for each error or a task that is not done correctly.

10. The minimum grade for an on-time submitted and compiled project is 10 out of 100, assuming there are no compilation errors and the submission includes a working Makefile and a README.

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 10 bonus points 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 *the* ***provided Ubuntu (16.04)***. 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.

Copying code from friends is called **plagiarism** not **collaboration** and will result in an F for the entire course. Any libraries or pieces of code that you use and you did not write must be listed in your README file. 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.