EE450 Socket Programming Project, Fall 2023

Due Date : Nov 26, 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:

Library Management Systems are essential for efficient library operations. They organize resources, improve accessibility, automate tasks, and provide valuable data for better decision-making. Library Management Systems streamline library services, save time, and enhance the overall user experience. This can also empower library staff with valuable insights into the inventory, highlighting books that frequently run out of stock. This information aids in effective inventory management and allows for the timely ordering of high-demand books, thereby optimizing the library's response to user requests. Additionally, security is a critical aspect to address. Ensuring that our Library Management System incorporates proper authorization mechanisms, such as requiring usernames and passwords, are crucial. Without these safeguards, non-members could potentially borrow books without paying the membership fee. Therefore, the development of a secure, reliable, functional, and informative web registration system is paramount for our library's success.

In this project, you will implement a straightforward library registration system. To simplify our library's organization, we will divide it into three distinct departments, each dedicated to a specific genre of books: Science, Literature, and History.

This system will enable users to submit the book code of their desired book, check its availability within the relevant department, and proceed with borrowing if the book is available.
Specifically, a library member will use the client to access the central library 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 books offered in this department. Additionally, the main server will be used to verify the identity of the library member.

- **Client:** used by a member to access the registration system, encrypts the login info.
- **Main server (serverM):** Verifies the identity of the members and coordinates with the backend servers.

- **Department server(s) (Science (S), Literature (L), History (H)):** store the information of books offered by this department.

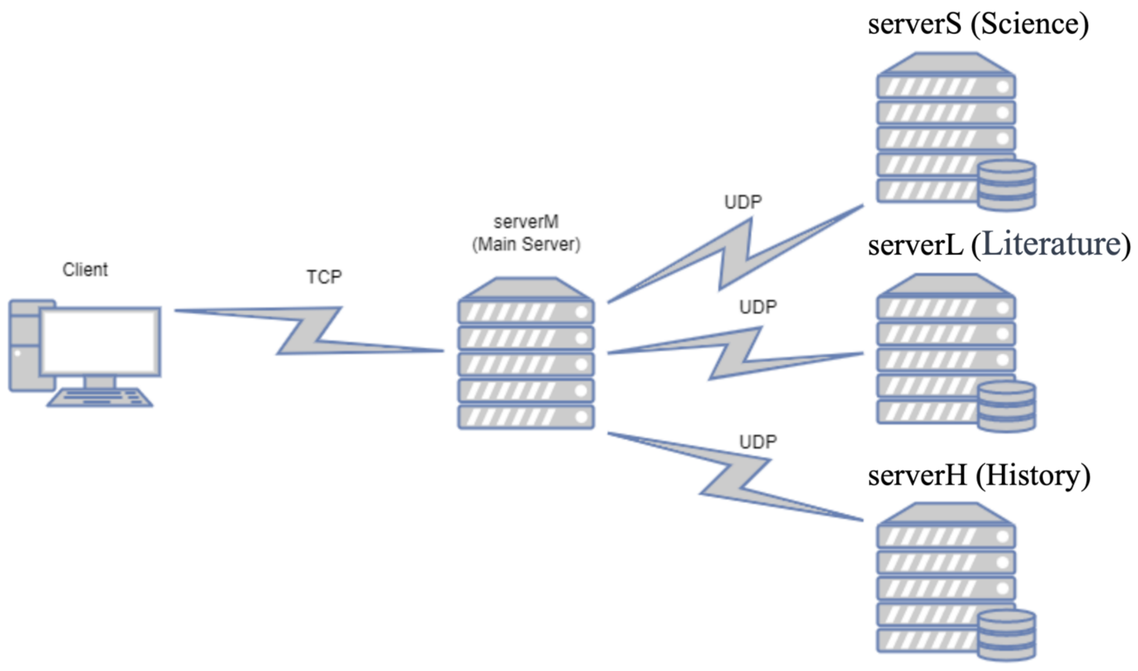

Figure 1: Illustration of the system

The backend 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 on 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.

<u>Source Code Files</u>

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

1. <u>serverM (Main Server)</u>: 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. <u>Backend-Servers S, L, H:</u> 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. A or B), depending on the server it corresponds to. (e.g., serverA.cpp & serverB.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:

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

science.txt: contains science book inventory information categorized in book code, and number of the available books. different categories are separated by a comma. This file should only be accessed by the Science Department server.

literature.txt: contains literature book inventory information categorized in book code, and number of the available books. different categories are separated by a comma. This file should only be accessed by the Literature Department server.

history.txt: contains history book inventory information categorized in book code and number of available books. different categories are separated by a comma. This file should only be accessed by the History Department server.

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!!!

Phase 1: Boot-up

Please refer to the "Process Flow" section to start your programs in order of the main server, server S, server L, server H, and Client. Your programs must start in this order. Each of the servers and the client have boot-up messages which have to be printed on screen, please refer to the on-screen messages section for further information.

When three backend servers (server S, server L, and server H) are up and running, each backend server should read the corresponding input file (*science.txt, literature.txt and history.txt*) and store the information in a certain data structure. You can choose any data structure that accommodates the needs. After storing all the data, server S, server L and server H should then send all the book statuses they have to the main server via UDP over the port mentioned in the PORT NUMBER ALLOCATION section. Since the book statuses are unique the main server will maintain a list of book 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 book 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 book 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, log in, and enter the book code.

Please check Table 8. Client on-screen messages for the on-screen message of different events

You should store the above book 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: Login and confirmation

In this phase, you will be authenticating the library member. The client will be asked to enter the username and password on the terminal. The client will encrypt this information and again 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, but it would not have any information about the encryption scheme. The information about the encryption scheme would only be present on the client side. The encryption scheme would be as follows:

- Offset each character and/or digit by 5.
- 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	Cipher Text
#1	Welcome to EE450!	Bjqhtrj yt JJ905!
#2	199@$	644@$
#3	0.27#&	5.72#&

Constraints:

● The username will be of lower case characters (5~50 chars).
● The password will be case sensitive (5~50 chars).
Phase 2A: 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:

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

This unencrypted information will be encrypted at client side and then sent to the main server over TCP.

Phase 2B: Main server receives encrypted username and password from the client. ServerM sends the result of the authentication request to client over a TCP connection.
If the login information was not correct/found:

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

After the successful login:

```
./client
Please enter the book-code: <bookcode>
```

Phase 3: Forwarding request to Backend Servers

Upon user input of a book code, the client is responsible for transmitting the request to the server M that is to the Main server via a TCP connection. The server M parses the received bookcode to determine the appropriate destination server for request forwarding.

Specifically, when the book code commences with "S," the request must be routed to Server S. Similarly, if the bookcode initiates with "L," the request is directed to Server L. In the event that the bookcode originates with "H," the request must be forwarded to Server H. All the valid book codes are eligible for forwarding to their respective servers from the Server M via a UDP connection.

BookCode from Client	Source Server	Destination Server
S146	Server M	Server S
L111	Server M	Server L
H211	Server M	Server H

A111	Server M	None of the Backend servers should receive this request

Note: Each server will have a dedicated database file. This file should be read only once at server startup to ensure that if a user checks out a book, 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.

Phase 4: Reply

The corresponding genre server will check its input file and find the count of the requested book-code. If the count is greater than 0, then the respective server will reply to the main server using UDP - "The requested book is available". And if the count of the book is 0, then the server will reply to the main server using UDP - "The requested book is not available". It is also possible that the book-code 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 book". Also after sending the reply to the main server the genre server will decrement the count of the corresponding book-code by 1 in the file so that when a client requests a book a second time, the availability is updated and correct in the file.

And at last, the main server will print the on-screen message and will forward the reply from the genre server to the client using TCP. And the client will print the on screen message which gives the availability of the requested book-code.

See ON SCREEN messages table for details.

Extra credit: Inventory Management
In this section, we will empower library staff to access information about the total availability of books based on their respective codes. To do this, a staff member can log in using the following credentials: Username: Admin, Password: Admin. Once authenticated, the staff can enter a book code, and the system will display the total number of copies available in the library's inventory for that specific code.

Process Flow/ Sequence of Operations:

- Your project grader will start the servers in this sequence: ServerM, ServerS, ServerL and client in 4 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**
serverS	-	1 UDP, 41000+xxx
serverL	-	1 UDP, 42000+xxx
serverH		1 UDP, 43000+xxx
serverM	-	1 UDP, 44000+xxx 1 TCP, 45000+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: **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 of the backend servers.

ON SCREEN MESSAGES: **Table 4. Backend Server on screen messages** **(For S, L, H Backend Servers)**	
Event	**On Screen Message (inside quotes)**
Booting Up (Only while starting):	Server <S or L or H> is up and running using UDP on port <port number>.
After receiving the book code from the Main server:	Server <S or L or H> received <book-code > code from the Main Server.
After sending the availability status to the main server:	Server <S or L or H> finished sending the availability status of code <book-code> to the Main Server using UDP on port <port number>.
After receiving the book code from the Main server with Admin access: (Extra Credit)	Server <S or L or H> received an inventory status request for code <book-code>.
After sending the inventory status to the main server:(Extra Credit)	Server <S or L or H> finished sending the inventory status to the Main server using UDP on port <port number>.

ON SCREEN MESSAGES:
Table 7. Main Server on screen messages

Event	On Screen Message (inside quotes)
Booting Up (only while starting):	Main Server is up and running.
After receiving the book code list from server S, L or H:	Main Server received the book code list from server<S, L, or H> using UDP over port <port number>.
After loading the *member.txt*:	Main Server loaded the member list.
After receiving the username and password from the client:	Main Server received the username and password from the client using TCP over port <port number>.
If the username do not exist in the *member list*:	<username> is not registered. Send a reply to the client.
Checking the password (matched)	Password <password> matches the username. Send a reply to the client.
Checking the password (does not match)	Password <password> does not match the username. Send a reply to the client.
After receiving the book code from client:	Main Server received the book request from client using TCP over port <port number>.
If the book code exist in the book code list, the main server sends the corresponding book code to the responsible backend server:	Found <book code> located at Server <S, L or H>. Send to Server <S, L or H>.
If the book code does not exist in the book code list, the main server send a response to the client:	Did not find <book code> in the book code list.
After receiving the book status result from the backend server:	Main Server received from server <S, L or H> the book status result using UDP over port <port number>: Number of books <book code> available is: <number of books avaliable>.
After sending the book status to the client:	Main Server sent the book status to the client.

ON SCREEN MESSAGES:

Table 8. Client on screen messages

Event	On Screen Message (inside quotes)
Booting Up:	Client is up and running.
Asking user to enter username and password:	Please enter the username: <unencrypted_username> Please enter the password: <unencrypted_password>
After sending the authentication request to the main server:	<username> sent an authentication request to the Main Server.
After receiving the result of authentication from main server (if authentication is successful)	<username> received the result of authentication from Main Server using TCP over port <port number>. Authentication is successful.
After receiving the result of authentication from main server (if username not found)	<username> received the result of authentication from Main Server using TCP over port <port number>. Authentication failed: Username not found.
After receiving the result of authentication from main server (if password doesn't match)	<username> received the result of authentication from Main Server using TCP over port <port number>. Authentication failed: Password does not match.
Asking the user to input Book-code to query.	Please enter book code to query:
Upon sending the request to the Main Server.	<username> sent the request to the Main Server.
After receiving the reply from the main server.	Response received from the Main Server on TCP port: <port number>.
If the requested book-code information was found successfully and the book count is greater than 0.	The requested book <book-code> is available in the library. —- Start a new query —- Please enter book code to query:
If the requested book-code information was found successfully and the book count is equal to 0.	The requested book <book-code> is NOT available in the library. —- Start a new query —- Please enter book code to query:

If the requested book-code was not found.	Not able to find the book-code <book-code> in the system. —- Start a new query —- Please enter book code to query:
(For Extra credit ONLY) Asking the user to input Book-code to query.	Please enter book code to query:
After sending a request to the main server.	Request sent to the Main Server with Admin rights.
After receiving reply from the Main Server	Response received from the Main Server on TCP port: <port number>.
If the requested book-code information was found successfully.	Total number of book <book-code> available = <number of books returned by main server> —- Start a new query —- Please enter book code to query:
If the requested book-code information was NOT found successfully	Not able to find the book-code <book-code> in the system. —- Start a new query —- Please enter book code to query:

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**
        - **serverS.cpp**
        - **serverL.cpp**
        - **serverH.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**
    - **serverL.cpp**
    - **serverH.cpp**
    - **Makefile**
    - **readme.txt (or) readme.md**
    - **client**
    - **serverM**
    - **serverS**
    - **serverL**
    - **serverH**
    - **member.txt**
    - **science.txt**
    - **literature.txt**
    - **history.txt**
    - **<Any additional header files>**

Note that in the above example, the input data files (member.txt, science.txt, literature.txt and history.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-ServerS Terminal:

```
ServerS is up and running using UDP on port 41319.
ServerS  received S145 code from the Main Server.
ServerS finished sending the availability status of code <book-code> to the Main Server using
UDP on port 41319.
```

Main Server Terminal:

```
Main Server is up and running.
Main Server received the book code list from server S using UDP over port 23319.
Main Server received the book code list from server L using UDP over port 23319.
Main Server received the book code list from server H using UDP over port 23319.
Main Server loaded the member list.
Main Server received the username and password from the client using TCP over port 24319.
Jone is not registered. Send a reply to the client.
Password Pa@$w0rd matches the username. Send a reply to the client
Main Server received the book request from the client using TCP over port 24319.
Found S101 located at Server S. Send to Server S.
Main Server received from server <S, L or H> the book status result using UDP over port 23319:
Number of books S101 available is: 3.
Main Server sent the book status to the client.
```

Client Terminal:

```
Client is up and running.
Please enter the username: John
Please enter the password: Pa@$w0rd
John sent an authentication request to the Main Server.
John received the result of authentication from Main Server using TCP over port 31145.
Authentication is successful.
Please enter book code to query: S101
John sent the request to the Main Server.
Response received from the Main Server on TCP port: <port number>.
The requested book S101 is available in the library.
—- Start a new query —-
Please enter book code to query:
```

The following is only for the **extra credits** part. Only key messages are given here, for other messages you can follow a similar format as in previous phases.

Client Terminal:

```
Please enter book-code to query: H104
Request sent to the Main Server with Admin rights.
Response received from the Main Server on TCP port: 42963
Total number of book H104 available = 9
```

Main Terminal:

```
Received book code request from Client.
Send the book code to backend server H to update the status.
Main server finished sending the update information to Server H.
```

Backend-ServerS Terminal:

```
ServerS received an inventory status request for code S101.
ServerS finished sending the inventory status to the Main server using UDP on port 41319.
```

Assumptions:

1. You have to start the processes in this order: **ServerM, ServerS, ServerL, ServerH, 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" 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 (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, e.g., usernames are concatenated and delimited by a comma, etc.
    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). 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 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
make clean	Removes all the executable files
./serverM	**Run**s Main server
./serverS	**Run**s Backend server S
./serverL	**Run**s Backend server L
./serverH	**Run**s Backend server H
./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, S, L and H. 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.

- 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 "**send"** button to actually submit it. If you do not click on "**send**", 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 the same 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 <u>public</u> repository before the deadline!!!
Double check the setting and do some testing before posting in a <u>private</u> repository!!!
(You can post your code after the deadline.)

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.