EE450 Socket Programming Project, Fall 2024

Due Date : Sunday December 1st, 2024 11:59 PM (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 Piazza. **You must discuss all project related issues on Piazza**. We will give those who actively help others out by answering questions on Piazza up to 10 bonus points.

Problem Statement:

GitHub is a code management platform based on the Git version control system, primarily used for software development and version control. It facilitates collaboration, tracks code changes, manages projects, shares repositories and executes workflows. We will design a simplified version called the Git450 code management platform, which will effectively manage files. This system will assist with simple user authentication, repository management (with the assumption that each user has only one repository), and deployment execution. Additionally, since user security is critical and to ensure user safety, Git450 will encrypt login usernames and passwords, ensuring that it provides a secure, reliable, and user-friendly environment.

For the Git450 project, we need three backend servers: Server A, Server R, and Server D. Users will be able to perform various operations through the client interface, and these operations will be dispatched to the corresponding backend servers via the main server.

- **Client:** The Git450 interface allows users to perform actions such as authentication, push, deploy, lookup, and remove.
  - **Members:** Can log in, push files, deploy applications, look up files, and remove files.
  - **Guests:** Can only look up filenames of a specific member stored in Server R. All files in the repository are assumed to be public. In order to login, guests

must indicate “guest” as their username and password.

- **Main Server:** Handles all guest and member actions by dispatching requests to the appropriate backend servers.
- **Backend Servers:** Server A, Server R, and Server D are used for user authentication, repository management, and deployment execution, respectively.

When the system starts, Server A will read the member.txt file to authenticate members, and Server R will initialize the repository using the filenames.txt file.

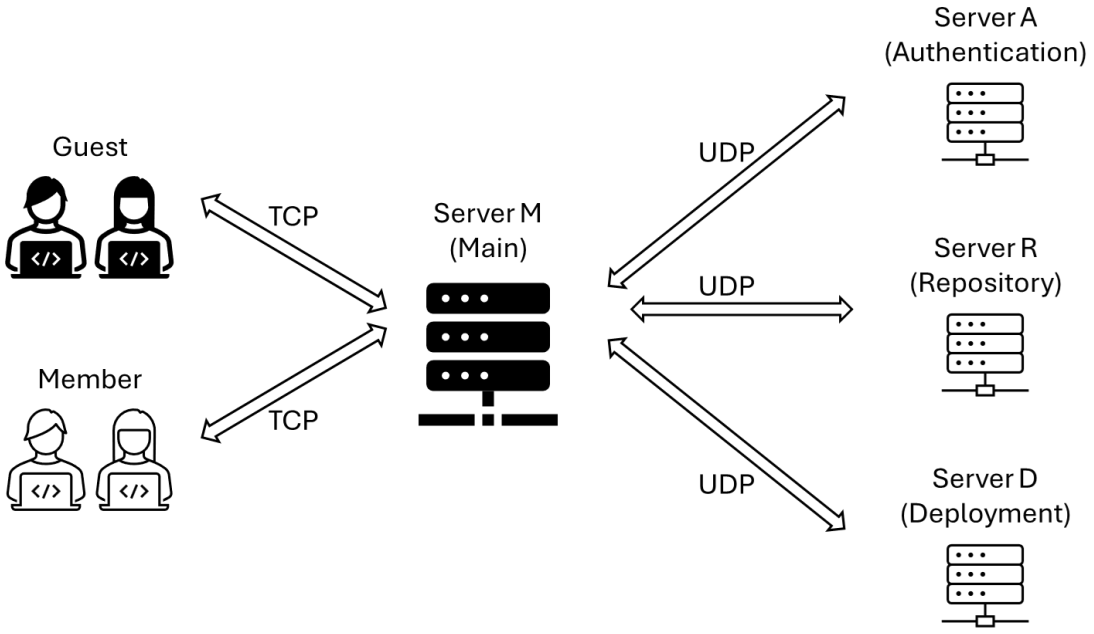

Figure 1. Illustration of the network

Source Code Files

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

1. 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).
2. 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’).
3. Backend Servers A, R, D: You are required to create three distinct files, choosing from the following naming conventions: **server#.c** or **server#cc** or **server#.cpp**. The filename must utilize one of these formats, substituting "#" with the specific

server identifier (either "A" or "R" or "D") to reflect the server it represents, resulting in filenames like serverA.c, serverA.cc, serverA.cpp, serverR.c, serverR.cc, or serverR.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

- **“**members.txt” : Located in Server A (Authentication server) which maintains the user credentials (usernames and passwords, in this corresponding order). The format of this file consists of two columns, usernames and passwords. A space separates these columns “ ‘’. Each user’s data is stored on a separate, new line. The membes.txt file is used for authentication purposes before allowing the deploy, push, and remove rights to the users.
- **“**filenames.txt” : Server R (Repository server) maintains the filenames.txt file, which keeps the collection of the user’s filenames. This file consists of two columns, usernames and filenames. Whenever the user pushes a new file, its metadata (the username and filename) are pushed into this filenames.txt file on a new line.

More Detailed Explanations:

Phase 1A: (20 points)

Please refer to the “Process Flow” section to start your programs in the order of the main serverM, server A, server R, server D, and at least two Clients (one Member and one Guest client). 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 A, server R and server D) are up and running. You can choose any data structure that accommodates your needs. You should print correct on-screen messages onto the screen for the main server and the backend servers indicating the success of these operations that will be mentioned on phase 2 as described in the “ON-SCREEN

messages" section. After the servers are booted up, 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.

Phase 1B: (20 points)

Authentication:

Command format: ./client <username> <password>

In this phase, the client will be asked to enter the username and password on the terminal. In order to be authenticated the user will run the command as described at the beginning of this section.

There are two types of users for this system:

- Guest: This type of user will need to write down “guest” for both username and password. After providing this information they will have access to lookup operations, as explained in the problem statement.
- Member: After providing credentials to the main server, it will request information to server A (Authentication server). Server A has the member.txt and will check if there is an instance in this file that has the <username> and <password> provided at the beginning of this phase. If a line which includes the given user and password is found then we consider that the authentication has been completed and the user can proceed with future operation.

  If the credentials provided do not fall under any of the previous cases both server A, server M and client must provide an error message (detailed in ON SCREEN messages section) and the process must end. If a client wants to start over they need to run the command again.

  For members, this system will provide an encryption scheme for the authentication on their passwords. The scheme 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:

Table 1. Encryption Example

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

Constraints:

- The password will be case sensitive (5~50 chars)

Observation: The password information indicated on the members.txt corresponds to the encrypted version of the passwords. For members, when asked for credentials they are required to provide their username and original password (non-encrypted). A short list of usernames and original passwords will be provided as an additional file for reference ("original.txt").

Phase 2: (60 points)

Lookup:

Command format: lookup <username>

This command retrieves a list of documents from the repository for a specific username, routed through Server M to Server R. Guest clients can use "lookup <username>" to access any user's repository. Member clients, on the other hand, can access their own files (with or without specifying their username) or view another user's repository by providing the corresponding <username>.

Push:

Command format: push <filename>

This command must be followed by a filename; otherwise, an error message will be displayed. The request will be routed through Server M to Server R to verify if the

filename is associated with a current member’s repository in Server R.

Here, we will encounter two scenarios:

- If the filename is not part of the system, the file will be stored directly in the member's repository (filename and username will be added to the “filenames.txt”) at Server R.
- If the file already exists, then no change in filenames.txt is required. The response from Server R will be routed to Server M and then to Client to check if the member wants to overwrite an existing file with the same name. The user should input Y or N (case-insensitive) to decide whether to overwrite the file. The client response message is routed to Server R through M, Server R receives the response.

You do not need to store the entire file content on Server R, as this project is a simplified version. Only the filename (not the full directory) should be stored on Server R. Server R only needs to compare the filenames within the member's repository to determine if the file exists.

Deploy:

Command format: deploy

The command will first check the repository server R, retrieve all the file names from that specific user from the file filenames.txt, and deploy on the deployed.txt file maintained on Server D. Note that deployed.txt file is not provided, and server D creates this file when the first deployment occurs.

- If no filename is found in the repository at server R, an error message will be displayed; otherwise, files will be deployed and added to the deployed.txt file, and the corresponding “ON-Screen messages” will be displayed.
- Please note that for the deploy command, we do not specify the filename; therefore, all the files belonging to a specific user will be retrieved from server R and deployed at server D.

You do not need to store the file content on Server D; only file names are enough. For testing purposes and grading, we will only run deploy at most one time per user.

Remove:

Command format: remove <filename>

For this command the objective is to remove a file that the username has located in his

repository. Using this command we will remove the filename from the list of stored files found at the Repository server (server R). The entry corresponding to the previously authenticated user and the filename to be removed will be taken out from the filenames.txt file. If the filename is not found on the server R consider writing an error message as indicated on the “ON-SCREEN messages” section.

Note: If the user inputs an invalid command, the system will prompt the user to input a command again after displaying the error message. For example, if the member enters push without specifying a filename, first print the message: "filename is not specified." Then, prompt the user by printing: "Please enter the command: <lookup <username>> , <push <filename> > , <remove <filename> > , <deploy> , <log>."

Extra credit (10 points):

Command format : log

In this section, you will implement another operation – log. The main server can receive and execute the log command to return the history of actions performed by a member. For example, user Yvette logs into her account and passes the authentication. She wants to view her operation history, so she runs the log command, and then the main server returns the result, displaying the history in ascending order of time (from old to new) on the client terminal. The main server should maintain a persistent record of all operations, ensuring that this data remains accessible even if a member terminates and then restarts the program. Detailed on-screen messages are shown below. To earn extra credit, please indicate that you have implemented this extra credit section in your readme.txt file.

Required Port Number Allocation

The ports to be used by the clients and the servers for the exercise are specified in the following table:

Table 2. Static and Dynamic assignments for TCP and UDP ports.		
Process	**Dynamic Ports**	**Static Ports**
serverA	-	1 UDP, 21000+xxx
serverR	-	1 UDP, 22000+xxx
serverD	-	1 UDP, 23000+xxx
serverM	-	1 UDP, 24000+xxx 1 TCP with client, 25000+xxx
client	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: **21000+319 = 21319** for the Backend-Server (A). **It is NOT going to be 21000319.**

On-Screen Messages

Table 3. Server A (Authentication Server) on-screen messages

Event	On Screen Message
Booting Up (Only while starting):	“Server A is up and running using UDP on port <port number>”.
Upon Receiving the auth request:	“ServerA received username <USERNAME> and password ****** ”
If auth request is member user:	“Member <USERNAME> has been authenticated”
If either username or password incorrect:	“The username <USERNAME> or password ****** is incorrect”

Table 4. Server R (Repository Server) on-screen messages

Event	On Screen Message
Booting Up (Only while starting):	“Server R is up and running using UDP on port <port number>.”
lookup request	
Upon receiving a lookup request from the main server	“Server R has received a lookup request from the main server.”
After returning the list of documents in the repository	“Server R has finished sending the response to the main server.”
push request	
Upon receiving a push request from the main server	“Server R has received a push request from the main server.”
After checking the filename and finding a duplicate within the member's repository	“<filename> exists in <username>'s repository; requesting overwrite confirmation."
Upon receiving a Y response to the overwrite confirmation	“User requested overwrite; overwrite successful.”
Upon receiving a N response to the overwrite confirmation	“Overwrite denied”
After checking the filename and finding no duplicates, the file is stored directly.	“<filename> uploaded successfully.”
remove request	
Upon receiving remove request	“Server R has received a remove request from the main server.”
deployment request	
Upon receiving a deploy request from the main server	“Server R has received a deploy request from the main server.”
After returning the list of documents in the repository	“Server R has finished sending the response to the main server.”

Table 5. Server D (Deployment Server) on-screen messages

Event	On Screen Message
Booting Up (Only while starting):	“Server D is up and running using UDP on port <port number>.”
deployment request	
Upon receiving a deploy request from the main server	“Server D has received a deploy request from the main server.”
Upon completing the deploy request from the main server	“Server D has deployed the user <username>’s repository.”

Table 6. Server M (Main Server) on-screen messages

Event	On Screen Message
Booting Up (Only while starting):	“Server M is up and running using UDP on port <port number>.”
Upon Receiving the authentication request from member	“Server M has received username <USERNAME> and password ****.”
Upon sending the authentication request to server A	“Server M has sent authentication request to Server A”
Upon receiving the authentication response from server A	“The main server has received the response from server A using UDP over <Main Server UDP port number>”
Upon sending the authentication response to client	“The main server has sent the response from server A to client using TCP over port <main server TCP port number>.”
lookup request	
Upon receiving a lookup request from a guest	The main server has received a lookup request from Guest to lookup <username>’s repository using TCP over port <main server TCP port number>.
Upon receiving a lookup request from a member	The main server has received a lookup request from <member’s username> to lookup <username>’s repository using TCP over port <main server TCP port number>.
After forwarding the lookup request to server R	The main server has sent the lookup request to server R.
After receiving the response from server R	The main server has received the response from server R using UDP over <Main Server UDP port number>
After forwarding the response to the client	The main server has sent the response to the client.
push request	

Upon receiving push request from a client	The main server has received a push request from <username>, using TCP over port <main server TCP port number>.
Upon receiving overwrite confirmation response from client	The main server has received the overwrite confirmation response from <username> using TCP over port <main server TCP port number>
Upon receiving a push request response from Server R (excluding overwrite confirmation requests)	The main server has received the response from server R using UDP over <Main Server UDP port number>
Upon receiving response from server R asking for overwrite confirmation	The main server has received the response from server R using UDP over <Main Server UDP port number>, asking for overwrite confirmation
After forwarding the response to the client (excluding overwrite confirmation requests)	The main server has sent the response to the client.
After forwarding the push request to server R (excluding overwrite confirmation response)	The main server has sent the push request to server R.
After forwarding the overwrite confirmation request to the client	The main server has sent the overwrite confirmation request to the client.
After forwarding the overwrite confirmation response to server R	The main server has sent the overwrite confirmation response to server R.
remove request	
Upon receiving a remove request from member user	“The main server has received a remove request from member <username> TCP over port <main server TCP port number>.”
Upon receiving a remove request done by server R	“The main server has received confirmation of the remove request done by the server R”
deployment request	
Upon receiving a deploy request from the Member User	The main server has received a deploy request from member <username> TCP over port <main server TCP port number>.
After forwarding the lookup request to server R	The main server has sent the lookup request to server R.
After receiving the response to lookup request to server R	The main server received the lookup response from server R.
After sending the server R response to request deploy to server D	The main server has sent the deploy request to server D.
After receiving a confirmation response from server D	The user <username>’s repository has been deployed at server D.
log request	
Upon receiving a log request	The main server has received a log request from member <username> TCP

from the Member User	over port <main server TCP port number>.
After sending the response to client	The main server has sent the log response to the client.

Table 7. Client (Member) on-screen messages

Event	On Screen Message
Booting Up	“The client is up and running.”
If client is a guest and wrote correct credentials	“You have been granted guest access.”
fail login	“The credentials are incorrect. Please try again.”
If client is a member and wrote correct credentials	“You have been granted member access”
Asking for commands (please note that the lookup command accepts <username>, and so on so forth)	“Please enter the command: <lookup <username>> <push <filename>> <remove <filename>> <deploy> <log>” *Menu can be printed in horizontal or vertical format.
Asking for commands (if you implement extra credit)	"Please enter the command: <lookup <username>> , <push <filename> > , <remove <filename> > , <deploy> , <log>."
lookup request	
No username is specified	“Username is not specified. Will lookup <member’s username>.”
Upon sending a lookup request	“<username> sent a lookup request to the main server.”
After receiving the response from the main server (the username exists)	“The client received the response from the main server using TCP over port <client port number>. <filename1.js> <filename2.html> … —--Start a new request—--”
After receiving the response from the main server (the username	“The client received the response from the main server using TCP over port <client port number>.

does not exist)	<username> does not exist. Please try again. —--Start a new request—--”
After receiving the response from the main server (the repository is empty)	“The client received the response from the main server using TCP over port <client port number>. Empty repository. —--Start a new request—--”
push request	
No filename is specified	“Error: Filename is required. Please specify a filename to push.”
Filename is invalid or unable to read	“Error: Invalid file: <filename> —--Start a new request—--”
Filename is exist in the member’s repository and server R request for confirmation	“<filename> exists in <username>’s repository, do you want to overwrite (Y/N)? “
Upon receiving a successful response from Server R	“<filename> pushed successfully”
Upon receiving a fail response from Server R	“<filename> was not pushed successfully.”
remove request	
Upon sending a remove request	“<username> sent a remove request to the main server.”
After receiving the confirmation from main server	“The remove request was successful.”
deployment request	
Upon sending a deploy request	<username> sent a lookup request to the main server.
After receiving the response from the main server (the username exists)	The client received the response from the main server using TCP over port <client port number>. The following files in his/her repository have been deployed. <filename1.js> <filename2.html> … —--Start a new request—--
(extra credit) log request	
Upon sending a log request	<username> sent a log request to the main server.

After receiving the response from the main server	The client received the response from the main server using TCP over port <client port number>. 1. push <filename> 2. push <filename> 3. lookup <username> 4. remove <filename> 5. deploy 6. log … —--Start a new request—--

Table 8. Client (Guest) on-screen messages

Event	On Screen Message
Booting Up	“The client is up and running.”
Asking for commands (please note that the lookup command needs <username>)	Please enter the command: <lookup <username>>
Invalid command (guest could only use the lookup command)	Guests can only use the lookup command
lookup request	
No username is specified	“Error: Username is required. Please specify a username to lookup. —--Start a new request—--”
Upon sending a lookup request	“Guest sent a lookup request to the main server.”
After receiving the response from the main server (the username exists)	“The client received the response from the main server using TCP over port <client port number>. <filename1.js> <filename2.html> … —--Start a new request—--”
After receiving the response from the main server (the username does not exist)	“The client received the response from the main server using TCP over port <client port number>. <username> does not exist. Please try again. —--Start a new request—--”

After receiving the response from the main server (the repository is empty)	"The client received the response from the main server using TCP over port <client port number>. Empty repository. —--Start a new request—--"

Assumptions:

1. You have to start the processes in this order: serverM, serverA, serverR, serverD, 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 mark the copied part in your code.

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** (see following). 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.

4. 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`

Requirements:

1. Do not hardcode the TCP or UDP port numbers that are to be obtained dynamically. Refer to Table 1 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 hardcoded as **localhost (127.0.0.1)** in all codes.
3. The clients and backend servers should keep running and 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 while running the client in Phase 1.
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 won't make any updates or changes to the virtual machine. It's your responsibility to make sure your code working well on the provided Ubuntu. "It works well on my machine" is not an excuse and we don't care.
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:

TABLE 9. Process flow for testing

`make all`	Compiles **all** your files and creates executables
`./serverM`	Runs Main server
`./serverA`	Runs backend server A
`./serverR`	Runs backend server R
`./serverD`	Runs backend server D
`./client <USERNAME> <PASSWORD>`	Runs the client

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, A, R and D using commands **`./serverM`**, **`./serverA`**, **`./serverR`**,**`./serverD`** and 2 terminals they will start 2 clients one as member one as guest using **`./client`**. **Remember that servers should always be on once started.** Clients can connect again and again with different commands and input values. TAs will check the outputs for multiple values of input. The terminals should display the messages shown in the onscreen messages section.

2. Compress all your files including the README file into a single "tar ball" and call it: **ee450_yourUSCusername_session#.tar.gz** (all small letters) e.g. my filename would be **ee450_nanantha_session1.tar.gz**. Please make sure that your name matches the one in the class list. Here are the instructions:

   a. 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:

   b.
   >> tar cvf **ee450_yourUSCusername_session#.tar** *
   >> gzip **ee450_yourUSCusername_session#.tar**
   Now, you will find a file named "ee450_yourUSCusername_session#.tar.gz" in the same directory. Please notice there is a star(*) at the end of first command.
   Any compressed format other than .tar.gz will NOT be graded!

3. Upload "ee450_yourUSCusername_session#.tar.gz" to the Digital Dropbox on the DEN website (DEN -> EE450 -> My Tools -> Assignments -> Socket 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. **You have plenty of time to work on this project and submit it in time hence 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.
For example, the TCP connection is established and data is sent to the main server. But the result is not received by the client because the main server got some errors.Then you will lose some points for phase 1 even though it might work well.

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

10. 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 months on this project and it doesn't even compile, you will receive only 10 out of 100.

11. **You must discuss all project related issues on Piazza**. 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 Piazza.)

12. Your code will not be altered in anyways for grading purposes and however it will be tested with different inputs. Your designated TA 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 (20.04)***. 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.