EE450 Socket Programming Project

Part 3

Fall 2024

Due Date:

Wednesday, December 11, 2024, 11:59PM

(Hard Deadline, Strictly Enforced)

OBJECTIVE

The objective of project 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, please email TA, or come by TA's office hours. **You can ask TAs any question about the content of the project, but TAs have the right to reject your request for debugging.**

PROBLEM STATEMENT

Consider a Student Dormitory Reservation System for managing student housing efficiently. It helps organize room bookings, makes finding and booking rooms easier, and gives useful information for making decisions. Due to the university's multiple campuses, student reservations are processed by several Campus Servers, with each Campus server in charge of multiple departments.

For this project, you'll make a simple dormitory booking system. We'll divide the dormitory into three types of rooms: Single Rooms (S), Double Rooms (D), and Triple Rooms (T), to keep things organized. Students can look up if the type of room they want is available and book it if it is. They use a client interface to reach the main server, which then connects with the specific Campus server corresponding to a specific campus based on the department. Each Campus server has information on the rooms it handles.

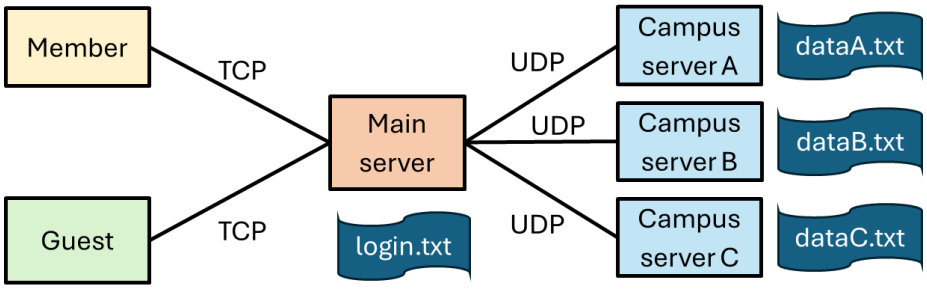

Figure 1: Overview of the Student Dormitory Reservation System

The detailed operations to be performed by all the parties are described with the help of Figure 1. There are in total 6 communication endpoints:

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

    - Guests： can only search for room availability.
        - Users who skip the password input by pressing "Enter" will be treated as a Guest.
- **Main Server**: Verifies the identity of the students (either member or guest) and coordinates with the Campus servers.
- **Campus server A/B/C**: Loading data from dataA/B/C.txt, storing dormitory information in a campus.

The full process can be roughly divided into four phases (see also "DETAILED EXPLANATION" section), the communication and computation steps are as follows:

Phase1: Bootup

1. [Computation]: Campus server A/B/C bootup first and read the files dataA.txt, dataB.txt and dataC.txt respectively and store the information in data structures.
    - Campus servers only need to read the text files once. When Campus servers are handling user queries, they will refer to the data structures, not the text files.
    - For simplicity, there is no overlap of departments or students among data files.
2. [Communication]: After step 1, Main server bootups and read the login.txt file. Then Main server asks each of Campus servers which departments they are responsible for. Campus servers reply with a list of departments to the Main server.
3. [Computation]: Main server will construct a data structure to book-keep such information from step 2. When the client queries come, the Main server can check the membership of the client and send a request to the correct Campus server.

Phase2: Login

1. [Computation]: Each client enter the username and password on the terminal. The client will encrypt this information. A *guest* can input the username while skipping the password input.
2. [Communication]: The client sends the encrypted login information to the Main server.
3. [Computation]: Main server receives the encrypted username and password, compares with the stored login information from login.txt, and reply the login results (success or fail) to the client.

Phase 3: Request

1. [Computation]: The client will input the department name, the room type (S, D, or T) and the action (search for the availability, rank by price, or make a reservation).
2. [Communication]: The client will send the request to the Main server using TCP.
3. [Computation]: The Main server parses the received information and identify the corresponding Campus server.

4. [Communication]: The Main server forwards the request to the corresponding Campus server.

Phase 4: Reply

1. [Computation]: The Campus server receives requests from Main server, calculating the available rooms for the query type of room, updating the room availability for a reservation.
2. [Communication]: The Campus server sends back the results to the Main server, Main server reply the message to the requested client.
3. [Communication]: Client receives the results from Main server and display it. Clients should keep active for further inputted queries, until the program is manually killed.

Input Files

- *login.txt*: contains encrypted usernames and passwords. This file should only be accessed by the Main server.
- *dataA.txt, dataB.txt, dataC.txt:* data files for each Campus server, storing information of dormitory and rooms.
- Note: *login_unencrypted.txt* is the unencrypted version of *login.txt*, which is provided for your reference to enter a valid username and password. It should NOT be touched by any servers!

The login.txt stores the encrypded usernames and passwords, generated using rules in DETAILED EXPLANATION Phase 2. The format of *login.txt* is:

```
<Encrypted Username>, <Encrypted Password>
```

Part of the example *login.txt*:

```
Mdphv,VRGlgv625
Pdub,lh2vngmz@
```

The format of the data file (*dataA.txt, dataB.txt, dataC.txt*) is defined as follows.

```
<Department Name 1>,<Department Name 2>,<Department Name 3>,...
<Type>,<Building ID>,<Availability>,<Price>
<Type>,<Building ID>,<Availability>,<Price>
<Type>,<Building ID>,<Availability>,<Price>
...
```

Part of the example *dataA.txt*:

```
ECE,CS,Physics,Accounting,Business
S,101,5,1200
S,202,10,1500
D,101,2,1000
```

```
D,321,1,900
T,101,4,800
T,202,0,750
```

Assumptions on the data file:

1. The first line is a list of department names that this Campus server is responsible for. The following lines are the information of dormitories.
2. Department names are letters. The length of a department name can vary from 1 letter to at most 20 letters. It may contain both capital and lowercase letters but does not contain any white spaces.
3. There are three types of dormitory rooms: Single (S), Double (D), and Triple (T).
4. There are multiple buildings denoted in Building IDs. The Building IDs are non-negative integer numbers less than 20 digits.
5. A certain building may contain various types of dormitory rooms.
6. The availability means a personal slot. For instance, availability=1 in a double-room dorm means 1 slot rather than 2.
7. There is no additional empty line(s) at the beginning or the end of the file. The whole data file does not contain any empty lines. A data file is not empty.
8. There is no overlap of department names among all Campus servers.

DETAILED EXPLANATION

Phase1: Bootup

All programs boot up in this phase in the order of 1) Campus servers A/B/C, 2) Main server, 3) two clients. Your programs must start in this order. Each of the servers and the clients have boot-up messages that must be printed on the screen. Please refer to the on-screen messages section for further information.

Campus servers should boot up first and display a bootup message. A Campus server then needs to read the text file, store the department names and dormitory information. There are many ways to store the dormitory information, such as dictionary, array, vector, etc. You need to decide which data structure to use based on the requirement of the problem. You can use **any** data structure if it can give you correct results. Note that the data structure should allow you to count total availability of a type of room, rank the rooms by prices, and update the availability for future steps.

Main server then boots up after Campus servers are running. Once the Main server programs have booted up, it should display a bootup message as indicated in the onscreen messages table. Note that the Main server code takes no input argument from the command line.

Once all four servers are booting up, Main server will ask Campus servers so that Main server knows which Campus server is responsible for which departments. Campus servers will reply with a list of departments to Main server. The communication between Main server and Campus servers is using **UDP**. Main server may store the received information using an unordered map:

Phase2: Login and Confirmation

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

The encryption scheme for member authentication would be as follows:

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

A few examples of encryption are given below:

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

Constraints:

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

Phase 2A:

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

```
./client
Please enter username: <unencrypted_username>
Please enter password ("Enter" to skip for guests): <unencrypted_password>
Please enter department name: <department name>
```

Phase 2B:

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

```
./client
Please enter username: <unencrypted_username>
Please enter password ("Enter" to skip for guests): <unencrypted_password>
Please enter department name: <department name>
Failed login. Invalid username/password
```

After the successful login, at the client side, show:

```
Welcome <member/guest> < unencrypted_username> from <department name>!
Please enter the dormitory type: <S/D/T>
```

Phase 3: Query

After entering the department name and the dormitory type, a client can continue performing three types of query actions: 1) check availability (availability), 2) check price (price), and 3) make a reservation (reserve).

Phase 3A: Check Availability

Both member and guest can check availability of a specific type of room. A campus server counts the total number of available rooms for a specific type. Taking the above example data file as an example: the availability of single-room dormitories is 15; double-room dormitories is 3, triple-room dormitories is 4. **Return and show the building IDs that have available rooms (room availability > 0) of the requested type.**

Phase 3B: Check Price

Only the member client can query the price of rooms and **rank them from low to high**. Print the building ID and prices on screen and send back this information to Main server. **Return and show all the building IDs that have the requested room type (room availability ≥ 0).** If a guest is requesting checking price, a "permission denied" prompt will show on the client screen.

Phase 3C: Make Reservation

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

Phase 4: Reply

At the end of Phase 3, the responsible Campus server should have the corresponding results ready. The result should be sent back to the Main server using **UDP**. When the Main server receives the result, it needs to forward all the result to the corresponding Client using **TCP**. The clients will print out academic performance statistics and then print out the messages for a new request.

For example:

```
Please enter request action (availability, price, reserve): availability
Campus A found 15 available rooms in S type dormitories.
Their Building IDs are: 101, 202.
```

```
Please enter request action (availability, price, reserve): price
Campus A found S type dormitories with prices:
        Building ID 101, Price $1200
        Building ID 202, Price $1500
```

```
Please enter request action (availability, price, reserve): reserve
Please enter Building ID for reservation: 101
Reservation is successful for Campus A Building ID 101!
```

Above are simplified examples. See ON-SCREEN MESSAGES table for more details.

Process Flow/Sequence of Operations:

- Your project grader will start the servers in this sequence: serverA, serverB, serverC, Main server, and two Clients in 6 different terminals.

- Once all the ends are started, the servers and clients should be continuously running unless stopped manually by the grader or meet certain conditions as mentioned before.

Required Port Number Allocation

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

Static and Dynamic assignments for TCP and UDP ports.		
Process	**Dynamic Ports**	**Static Ports**
ServerA	-	UDP, 31xxx
ServerB	-	UDP, 32xxx
ServerC	-	UDP, 33xxx
Main server	-	UDP (with servers), 34xxx TCP (with clients), 35xxx
clients	TCP	

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: **31319** for the Campus-Server (A), etc. Port number of all processes print port number of their own.

ON SCREEN MESSAGES

Table 2. Campus Server A/B/C on-screen messages

Event	**On-screen Messages**
Booting up (Only while starting):	The Server <A/B/C> is up and running using UDP on port <server A/B/C port number>.
Sending the department list that contains in "dataA/B/C.txt" to Main Server:	Server A/B/C has sent a department list to Main Server
(a) Query: Availability (member client and guest client)	
Upon receiving the input query:	Server <A/B/C> has received a query of Availability for room type <S/D/T>
If this room type cannot be found:	Room type <input room type> does not show up in Server <A/B/C>
Upon finding the room type, analyze the results:	Server A/B/C found totally <num> available rooms for < S/D/T> type dormitory in Building: <Building ID1>, < Building ID2>...
Sending result(s) back to Main Server:	Server A/B/C has sent the results to Main Server
(b) Query: Price (member client only)	
Upon receiving the input query:	Server <A/B/C> has received a query of Price for room type <S/D/T>
If this room type cannot be found:	Room type <input room type> does not show up in Server <A/B/C>
Upon finding the room type, rank the entries by price from low to high:	Server A/B/C found room type < S/D/T> with prices: Building ID <Building ID1>, Price $<Price 1> Building ID <Building ID2>, Price $<Price 2> … For example: Server A found room type S with prices: Building ID 101, Price $1200 Building ID 202, Price $1500
Sending result(s) back to Main Server:	Server A/B/C has sent the results to Main Server

(c) Query: Reserve (member client only)	
Upon receiving the input query:	Server <A/B/C> has received a query of Reserve for room type <S/D/T> at Building ID <Building ID>
If this room type cannot be found:	Room type <input room type> does not show up in Server <A/B/C>
If this Building ID cannot be found:	Building ID <input Building ID> does not show up in Server <A/B/C>
Upon finding the room type and building ID, if the room is not available (availability=0):	Server A/B/C found room type < S/D/T> in Building ID <input Building ID>. This room is not available.
Upon finding the room type and building ID, if the room is available:	Server A/B/C found room type < S/D/T> in Building ID <input Building ID>. This room availability is < num >. This room is reserved, and availability is updated to <updated num>.
Sending result(s) back to Main Server:	Server A/B/C has sent the results to Main Server

Table 3. Main Server on-screen messages

Event	**On-screen Messages**
Booting up:	Main server is up and running.
Receiving the department lists from Server A/B/C:	Main server has received the department list from server A/B/C using UDP over port <Main server UDP port>
List the results of which department Campus server A/B/C is responsible for:	Server A <Department Name 1> , <Department Name 2>… Server B <Department Name 3>,… Server C <Department Name 4>,…
Member login	
After receiving the username and password from the ***member***	The main server received the authentication for <encrypted username> using TCP over port <main server TCP port number>.
Upon sending an authentication response to the client	The authentication <passed/failed>. The main server sent the authentication result to the client.
Guest login	
After receiving the username from the ***guest***	The main server received the guest request for <encrypted username> using TCP over port <main server TCP port number>. The main server accepts < encrypted username> as a guest.
Upon sending a guest response to the client	The main server sent the guest response to the client.
Query	
Upon receiving the input from a member/guest for a request:	Main server has received the query from <member/guest> <encrytped username> in <department name> for the request of <Availability, Price, Reserve>.
Forwarding the request to Server <A/B/C> based on department name:	<Department Name> does not show up in Campus servers
	The main server forwarded a request of <Availability, Price, Reserve> to Server <A/B/C> using UDP over port <Main server UDP port number>.
Reply	

Main Server receives the results from Campus server A/B/C	The Main server has received result for the request of <Availability, Price, Reserve> from Campus server <A/B/C> using UDP over port <Main server UDP port>.
	The Main server has sent back the result for the request of <Availability, Price, Reserve> to the client <member/guest> <encrypted username> using TCP over port <Main Server TCP port >.

Table 4. Client on-screen messages

Event	**On-screen Messages**
Booting up(only while starting)	Client is up and running.
Asking the user to enter the username	Please enter username: <unencrypted_username>
Asking the user to enter the password	Please enter password: <unencrypted_password>
Asking the user to enter the department name	Please enter department name: <department name>
Upon sending an authentication request to Main Server	<username> sent an authentication request to the main server.
After receiving the result of the authentication request from Main Server (if the authentication passed)	Welcome <member/guest> < unencrypted_username> from <department name>!
After receiving the result of the authentication request from Main Server	Failed login. Invalid username or password.
Asking the user to input the room type	Please enter the room type S/D/T: <S/D/T>
Asking the user to choose the desired action	Please enter request action (availability, price, reserve):
(a) Query: Availability (member client and guest client)	
Upon sending an availability request to Main Server	<username> sent a request of Availability for type < S/D/T > to the main server.
After receiving the response from Main Server (the count is greater than 0)	The client received the response from the main server using TCP over port <client port number>. Campus <A/B/C> found <num> available rooms in < S/D/T> type dormitory. Their Building IDs are: <Building ID1>, < Building ID2>...

After receiving the response from Main Server (the count is 0)	The client received the response from the main server using TCP over port <client port number>. The requested room is not available.
After receiving the response from Main Server (room type is not in the system)	The client received the response from the main server using TCP over port <client port number>. Not able to find the room type.
(b) Query: Price (member client and guest client)	
Upon sending a checking price request to Main Serves	<username> sent a request of Price for type < S/D/T > to the main server.
After receiving the response from Main Server	The client received the response from the main server using TCP over port <client port number>. Campus A/B/C found room type < S/D/T> with prices: Building ID <Building ID1>, Price $<Price 1> Building ID <Building ID2>, Price $<Price 2> … For example: Campus A found room type S with prices: Building ID 101, Price $1200 Building ID 202, Price $1500
After receiving the response from Main Server (room type is not in the system)	The client received the response from the main server using TCP over port <client port number>. Not able to find the room type.
(c) Query: Reserve (member client and guest client)	
Enter building ID for reservation	Please enter Building ID for reservation:
Upon sending a checking price request to Main Serves	<username> sent a request of Reserve for type < S/D/T > and Building ID < Building ID > to the main server.

After receiving the response from Main Server (Building ID does not exist)	The client received the response from the main server using TCP over port <client port number>. Reservation failed: Building ID < Building ID > does not exist.
After receiving the response from Main Server (room type is not in the system)	The client received the response from the main server using TCP over port <client port number>. Reservation failed: Not able to find the room type.
After receiving the response from Main Server (the room is not available)	The client received the response from the main server using TCP over port <client port number>. Reservation failed: Building ID < Building ID > room type <S/D/T> is not available.
After receiving the response from Main Server (success)	The client received the response from the main server using TCP over port <client port number>. Reservation is successful for Campus <A/B/C> Building ID <Building ID>!
After printing the results	-----Start a new request-----

ASSUMPTIONS

1. You must start the processes in this order: **Campus-server (A), Campus -server (B), Campus -server (C), Main-server, Member Client, Guest Client.**

2. The dataA/B/C.txt files are created before your program starts.

3. 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.

4. You can use code snippets 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.

5. 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.

6. 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 developer
   ```

   Identify the zombie processes and their process number and kill them by typing at the command-line:

   ```
   kill -9 <process number>
   ```

7. You may use the following command to double check the assigned TCP and UDP port numbers:

   ```
   sudo lsof -i -P -n
   ```

REQUIREMENTS

1. Do not hardcode the TCP 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 hard coded as **localhost (127.0.0.1)** in all codes.

3. Your client should keep running and ask to enter a new request after displaying the previous result, until the TA manually terminate it by Ctrl+C. 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.

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 the extra messages before you submit your project.

7. Please use fork() or similar system calls to create concurrent processes is not mandatory if you do not feel comfortable using them. However, the use of fork() for the creation of a child process when a new TCP connection is accepted is mandatory and everyone should support it. This is useful when different clients are trying to connect to the same server simultaneously. If you don't use fork() in the Main server when a new connection is accepted, the Main Server won't be able to handle the concurrent connections.

8. 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 works 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/

You can use a Unix text editor like emacs or gedit 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 may need to include these header files:

```
#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**.
Submissions without README and Makefile will be subject to a serious penalty.

In the README file write:

- Your **Full Name** as given in the class list
- Your Student ID
- Briefly summarize what you have done in the assignment. (Please do not repeat the project description).
- List all your code files and briefly summarize their fulfilled functionalities. (Please do not repeat the project description).
- The format of all the messages exchanged between servers.
- Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
- 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.)

About the Makefile

Makefile Tutorial:

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

Makefile should support following functions:

Compile **all** your files and creates executables	make all
Compile server A	make serverA
Compile server B	make serverB
Compile server C	make serverC
Compile Main Server	make servermain
Compile Client	make client
Run Server A	./serverA
Run Server B	./serverB

Run Server C	./serverC
Run Main Server	./servermain
Run Client	./client

TAs will first compile all codes using **make all**. They will then open 7 different terminal windows: three terminals for Backend Servers A, B, C using commands **./serverA**, **./serverB, ./serverC**; one terminal for and Main Server using command **./servermain**; two terminals for clients using **./client**. **Remember that servers should always be on once started.** TAs will check the outputs for multiple queries. The terminals should display the messages specified above.

1. Compress all your files including the README file into a single "tar ball" and call it: ee450_yourUSCusername.tar.gz (all small letters) e.g. an example filename would be ee450_nanantha.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 such as data files!!! Only include the required source code files, Makefile and the README file. Now run the following commands:

   ```
   tar cvf ee450_yourUSCusername.tar *
   gzip ee450_yourUSCusername.tar
   ```

   Now, you will find a file named "ee450_yourUSCusername.tar.gz" in the same directory. Please notice there is a star (*) at the end of the first command.

2. Do NOT include anything not required in your tar.gz file. Do NOT use subfolders. Any compressed format other than .tar.gz will NOT be graded!
3. Upload "ee450_yourUSCusername.tar.gz" to Blackboard -> Assignments. After the file is submitted, you must click on the "submit" button to actually submit it. If you do not click on "submit", the file will not be submitted.

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

6. 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!

7. After submitting, 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.

8. You have sufficient 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.

Your project grade will depend on the following:

1. Correct functionality, i.e. how well your programs fulfill the requirements of the assignment, especially the communications through 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 15 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 you add subfolders or compress files in the wrong way, you will lose 2 points each.

9. If your data file path is not the same as the code files, you will lose 5 points.

10. Do not submit datafile (three .txt files) used for test, otherwise, you will lose 10 points.

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

12. Detailed points assignments for each functionality will be posted after finishing grading.

13. The minimum grade for an on-time submitted project is 10 out of 100, the submission includes a working Makefile and a README.

14. There are no points for the effort or the time you spend working on the project or reading the tutorial. If you spend plenty of time on this project and it doesn't even compile, you will receive only 10 out of 100.

15. Your code will not be altered in any way 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 project description, we will follow the project description.

FINAL WORDS

1. Start on this project early. Hard deadline is strictly enforced. No grace periods. No grace days. No exceptions.

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. 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. Check Blackboard for additional requirements and latest updates about the project guidelines. Any project changes announced on Blackboard are final and overwrites the respective description mentioned in this document.

4. Plagiarism will not be tolerated and will result in an “F” in the course.