EE450 Socket Programming Project

Part3

Spring 2023

Due Date:

Sunday, April 30, 2023 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, post your questions on Blackboard -> Discussion Board 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

In this project, you will implement a Student Performance Analysis system, a software application that allows students to check their GPA, percentage ranks, and other relevant data. This information can be used by students to identify areas where they need to focus their efforts and improve their performance. It can also be used by universities to identify areas where they need to improve their teaching methods or provide additional resources to help students succeed.

This system generates customized academic statistics based on user queries. Specifically, consider a student in a specific department wants to request an analysis of academic performance. The student would send a request to a main server and receive results replied from the main server. Now since there are many departments, the university use a distributed system design where the main server is further connected to many (in our case, two) backend servers. Each backend server stores the academic records for a list of departments. For example, backend server A may store the student data of ECE and CS department, and backend server B may store the student data of Art department. Therefore, once the main server receives a user (student) query, it decodes the query and further sends a request to the corresponding backend server. The backend server will search through its local data, analyze the academic statistics, and reply to the main server. Finally, the main server will reply to the user to conclude the process.

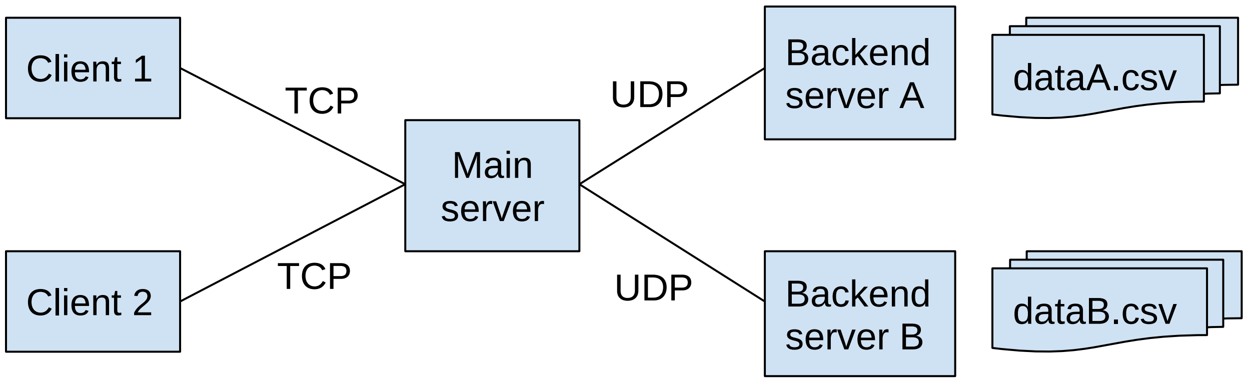


Figure 1: Overview of the Student Performance Analysis system.

The detailed operations to be performed by all the parties are described with the help of Figure 1. There are in total 5 communication endpoints:

- Client 1 and Client 2: representing two different students, possibly in different department
- Main server: responsible for interacting with the clients and the backend servers
- Backend server A and Backend server B: responsible for generating the statistics of the students' academic performance
    - Student data is stored in plain text files using CSV (comma-separated values) format. Backend server A stores dataA.csv and Backend server B stores dataB.csv in their local file system.

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

Bootup

1. [Computation]: Backend server A and B read the files dataA.csv and dataB.csv respectively and store the information in data structures.
    - Assume a "static" system where contents in dataA.csv and dataB.csv do not change throughout the entire process.
    - Backend servers only need to read the text files once. When Backend servers are handling user queries, they will refer to the data structures, not the text files.
    - For simplicity, there is no overlap of departments between dataA.csv and dataB.csv.
2. [Communication]: after step 1, Main server will ask each of Backend servers which departments they are responsible for. Backend 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 send a request to the correct Backend server.

Query

1. [Communication]: Each client sends a query (a department name and a student ID) to the Main server.
    - A client can terminate itself only after it receives a reply from the server (in the Reply phase).
    - Main server may be connected to both clients at the same time.
2. [Computation]: Once the Main server receives the queries, it decodes the queries and decides which backend server(s) should handle the queries.

Analysis

1. [Communication]: Main server sends a message to the corresponding backend server so that the Backend server can perform computation.

2. [Computation]: Once the query student ID is received, Backend server generates academic statistics for this student.
3. [Communication]: Backend servers, after generating the statistics, will reply to Main server.

Reply

1. [Computation]: Main server decodes the messages from Backend servers and then decides which academic statistic result correspond to which client query.
2. [Communication]: Main server prepares a reply message and sends it to the appropriate Client.
3. [Communication]: Clients receive the academic statistic result from Main server and display it. Clients should keep active for further inputted queries, until the program is manually killed.

DETAILED EXPLANATION

Phase 1 (20 points) -- Bootup

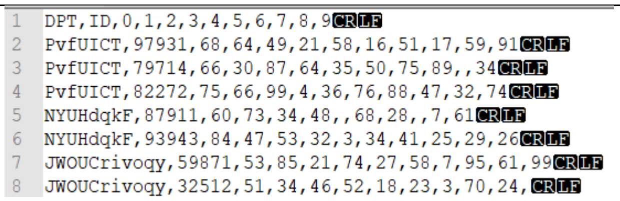
All three server programs (Main server, Backend servers A & B) boot up in this phase. While booting up, the servers must display a boot up message on the terminal. The format of the boot up message for each server is given in the onscreen message tables at the end of the document. As the boot up message indicates, each server must listen on the appropriate port for incoming packets/connections.

Backend servers should boot up first. A backend server needs to read the CSV file and store the department names and student academic records in appropriate data structures. There are many ways to store them, such as dictionary, array, vector, etc. You need to decide which format to use based on the requirement of the problem. You can use **any** format if you can generate the correct results.

Data File Formats

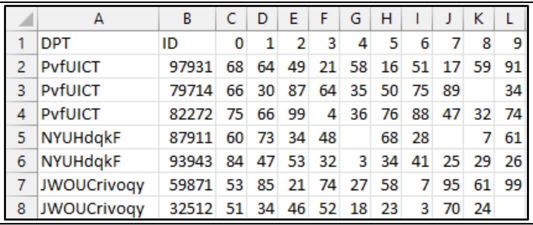
CSV stands for “comma-separated values” and is a simple file format used to store tabular data, such as a spreadsheet or database. CSV files are often used when data needs to be compatible with many different programs. They are plain text files that store data by delimiting data entries with commas. You can open CSV files in text editors, spreadsheet programs like Excel, or other specialized applications, as is shown in Figure 2-a and Figure 2-b.

The format of dataA.csv and dataB.csv is defined as follows. For simplicity, consider the university is using 100-point scale grades for all courses, and all courses are 1 unit. The first row is a header, showing the columns name [DPT, ID, 0, 1, …], where DPT means department name, ID means student ID, and following numerical values 0, 1, … are the index of courses. Below the first row, each entry represents the academic record (course scores) for a specific student. For example, row 2 means the student in department "PvfUICT" with student ID "97931" has taken course 0-9 with scores: 68,64,49,21,58,16,51,17,59,91.



```
DPT,ID,0,1,2,3,4,5,6,7,8,9CRLF
PvfUICT,97931,68,64,49,21,58,16,51,17,59,91CRLF
PvfUICT,79714,66,30,87,64,35,50,75,89,,34CRLF
PvfUICT,82272,75,66,99,4,36,76,88,47,32,74CRLF
NYUHdqkF,87911,60,73,34,48,,68,28,,7,61CRLF
NYUHdqkF,93943,84,47,53,32,3,34,41,25,29,26CRLF
JWOUCrivoqy,59871,53,85,21,74,27,58,7,95,61,99CRLF
JWOUCrivoqy,32512,51,34,46,52,18,23,3,70,24,CRLF
```

Figure 2-a: Example dataA.csv open in text editor



	A	B	C	D	E	F	G	H	I	J	K	L
1	DPT	ID	0	1	2	3	4	5	6	7	8	9
2	PvfUICT	97931	68	64	49	21	58	16	51	17	59	91
3	PvfUICT	79714	66	30	87	64	35	50	75	89		34
4	PvfUICT	82272	75	66	99	4	36	76	88	47	32	74
5	NYUHdqkF	87911	60	73	34	48		68	28		7	61
6	NYUHdqkF	93943	84	47	53	32	3	34	41	25	29	26
7	JWOUCrivoqy	59871	53	85	21	74	27	58	7	95	61	99
8	JWOUCrivoqy	32512	51	34	46	52	18	23	3	70	24	

Figure 2-b: Example dataA.csv open in MS Excel

Assumptions on the data file:

1. There can be empty items in the course columns, which means the student does not take the course, e.g., Figure 2-b item (K,3). In this case, there is no content between two commas in the plain text (see Figure 2-a).
2. A student takes at least one course, there is no entry with all empty scores.
3. There are at most 100 students and at least 1 student per department.
4. There are at most 20 departments and at least 1 department per file.
5. The student IDs are unique, there is no repeated student ID.
6. Department names are letters. The length of a department name can vary from 1 letter to at most 20 letters. It may contain only capital and lowercase letters but does not contain any white spaces or other characters. Department names "Abc" and "abc" are different.
7. Student IDs are non-negative integer numbers. The maximum possible student ID is (2^31 - 1). The minimum possible student ID is 0.
    - This ensures that you can always use int32 to store the student ID.
8. There is no additional empty line(s) at the beginning or the end of the file. That is, the whole dataA.csv and dataB.csv do not contain any empty lines.
9. For simplicity, there is no overlap of departments between dataA.csv and dataB.csv.
10. The student IDs in the text are not sorted.
11. dataA.csv and dataB.csv will not be empty.

An example dataA.csv and dataB.csv is provided for you as a reference. Other dataA.csv and dataB.csv will be used for grading.

Main server then boots up after Backend servers are running and finish processing the files of dataA.csv and dataB.csv. Main server will request Backend servers for department lists so that Main server knows which Backend server is responsible for which departments. The communication between Main server and Backend servers is using **UDP**.

Once the server programs have booted up, two client programs run. Each client displays a boot up message as indicated in the onscreen messages table. Note that the client code takes no input argument from the command line. The format for running the client code is:

```
./client
```

After running it, it should display messages to ask the user to enter a query department name and a query student ID (e.g., implement using std::cin):

```
./client
…
Enter department name:
Enter student ID:
```

For example, if the client 1 is booted up and asks for the academic performance analysis for student ID 97931 in Department "PvfUICT", then the terminal displays like this:

```
./client
…
Enter department name: PvfUICT
Enter student ID: 97931
```

After booting up, Clients establish **TCP** connections with Main server. After successfully establishing the connection, Clients send the input department name and student ID to Main server. Once this is sent, Clients should print a message in a specific format. Repeat the same steps for Client 2.

Each of these servers and the main server have its unique port number specified in "PORT NUMBER ALLOCATION" section with the source and destination IP address as localhost/127.0.0.1.

Clients, Main server, Backend server A and Backend server B are required to print out on screen messages after executing each action as described in the “ON SCREEN MESSAGES” section. These messages will help with grading if the process did not execute successfully. Missing some of the on-screen messages might result in misinterpretation that your process failed to complete. Please follow the exact format when printing the on-screen messages.

Phase 2 (30 points) -- Query

In the previous phase, Client 1 and Client 2 receive the query parameters from the two users and send them to Main server over TCP socket connection. In phase 2, Main server will have to receive requests from two Clients. If the department name or student ID are not found, the main server will print out a message (see the “On Screen Messages” section) and return to standby.

For a server to receive requests from several clients at the same time, the function **fork()** should be used for the creation of a new process. Fork() function is used for creating a new process, which is called ***child process***, which runs concurrently with the process that makes the fork() call (***parent process***). This is the same as in Project Part 1.

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

Phase 3 (40 points) -- Analysis

In this phase, each Backend server should have received a request from Main server. The request should contain a department name and a student ID. A backend server will generate academic statistics per request based on the academic records of the student and records of all students in the department.

Expected Results

You should calculate the following statistics from the records: Student GPA, Percentage Rank in Department, Department GPA Mean, Department GPA Variance, Department Max GPA, Department Min GPA. In detail:

- Student GPA: Grade-Point Average. Since we assume all courses are at the same unit, the GPA for a student is the average grades of all the courses the student has taken. Note that empty items should not be counted.
- Percentage Rank in Department: The GPA of the request student is higher than how many percent of students. For example, if the student is rank 4 in 10 students, meaning the student is higher than 6 students in GPA, the percentage rank should be (10-4)/10=60%, meaning the student is higher than 60% of the students. We define that if only one student in the department, then the rank is (1-1)/1=0%.
- Department GPA Mean: Sum(GPA of all students in this department)/(number of students in this department).
- Department GPA Variance: Assuming in a department, the student number is N, the mean GPA is μ, then the variance is defined as: $\sum(x-\mu)^2/N$, where x is the GPA of one student in the department.
- Department Max GPA, Department Min GPA: Sort all the students' GPA in the department and get the max and min values.

For example, in the "PvfUICT" department as shown in Figure 2, the GPA for the three students are: 49.4, 58.9, 59.7. Thus, if the request student ID is "79714", the returned results should be:

Student GPA: 58.9; Percentage Rank: 33.3%; Department GPA Mean: 56; Department GPA Variance: 21.9; Department Max GPA: 59.7; Department Min GPA: 49.4.

Phase 4 (10 points) -- Reply

At the end of Phase 3, the responsible Backend server should have the result ready. The result is the academic performance statistics. 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 as follows:

```
...

The performance statistics for Student 79714 in Department
PvfUICT is:
```

```
Student GPA: 58.9
Percentage Rank: 33.3%
Department GPA Mean: 56
Department GPA Variance: 21.9
Department Max GPA: 59.7
Department Min GPA: 49.4

-----Start a new request-----
Enter department name:
Enter student ID:
```

See the ON SCREEN MESSAGES table for an example output table.

PORT NUMBER ALLOCATION

The ports to be used by the client and the servers are specified in the following table:

Table 1. Static and Dynamic assignments for TCP and UDP ports

Process	**Dynamic Ports**	**Static Ports**
Backend-Server A		UDP: 30xxx
Backend-Server B		UDP: 31xxx
Main Server		UDP(with server): 32xxx TCP(with client): 33xxx
Client 1	TCP	
Client 2	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: **30319** for the Backend-Server (A), etc. Port number of all processes print port number of their own.

ON SCREEN MESSAGES

Table 2. Backend-Server A on-screen messages

Event	**On-screen Messages**
Booting up (Only while starting):	Server A is up and running using UDP on port <server A port number>
Sending the department list that contains in “dataA.csv” to Main Server:	Server A has sent a department list to Main Server
For student performance analysis, upon receiving the input query:	Server A has received a request for Student <Student ID> in <Department Name>
If this student ID cannot be found in this department, send “not found” back to Main Server:	Student < Student ID> does not show up in < Department Name>
	Server A has sent “Student < Student ID> not found” to Main Server
If this student ID is found in this department, calculate academic statistics of this student result back to Main Server:	Server A calculated following academic statistics for Student < Student ID> in < Department Name>: Student GPA:XXX Percentage Rank: XXX Department GPA Mean: XXX Department GPA Variance: XXX Department Max GPA: XXX Department Min GPA: XXX *(Replace XXX with your results)*
	Server A has sent the result to Main Server

Table 2. Backend-Server B on-screen messages

Event	**On-screen Messages**
Booting up (Only while starting):	Server B is up and running using UDP on port <server B port number>
Sending the department list that contains in “dataB.csv” to Main Server:	Server B has sent a department list to Main Server
For student performance analysis, upon receiving the input query:	Server B has received a request for Student <Student ID> in <Department Name>
If this student ID cannot be found in this department, send “not found” back to Main Server:	Student < Student ID> does not show up in < Department Name>
	Server B has sent “Student < Student ID> not found” to Main Server
If this student ID is found in this department, calculate academic statistics of this student result back to Main Server:	Server B calculated following academic statistics for Student < Student ID> in < Department Name>: Student GPA: XXX Percentage Rank: XXX Department GPA Mean: XXX Department GPA Variance: XXX Department Max GPA: XXX Department Min GPA: XXX *(Replace XXX with your results)*
	Server B has sent the result to Main Server

Table 4. Main Server on-screen messages

Event	**On-screen Messages**
Booting up(only while starting):	Main server is up and running.
Upon receiving the department lists from Server A:	Main server has received the department list from server A using UDP over port <Main server UDP port number>
Upon receiving the department lists from Server B:	Main server has received the department list from server B using UDP over port <Main server UDP port number>
List the results of which department server A/B is responsible for:	Server A <Department Name 1> <Department Name 2> Server B <Department Name 3>
Upon receiving the input from the client:	Main server has received the request on Student <student ID> in <Department Name> from client <client ID> using TCP over port <Main server TCP port number>
If the input department name could not be found, send the error message to the client:	<Department Name> does not show up in server A&B
	Main Server has sent "<Department Name>: Not found" to client <client ID> using TCP over port <Main server TCP port number>
If the input department name could be found, decide which server contains related information about the input department and send a request to server A/B	<Department Name> shows up in server <A or B>
	Main Server has sent request of Student <student ID> to server A/B using UDP over port <Main server UDP port number>
If this student ID is found in a backend server, Main Server will receive the searching results from server A/B and send them to client1/2	Main server has received searching result of Student <student ID> from server<A or B>
	Main Server has sent searching result(s) to client <client ID> using TCP over port <Main Server TCP port number>
If this student ID cannot be found in a backend server, send the error message back to client	Main server has received "Student <student ID>: Not found" from server <A or B>
	Main Server has sent message to client <client ID> using TCP over <Main Server TCP port number>

Table 5. Client 1 or Client 2 on-screen messages

Event	**On-screen Messages**
Booting up(only while starting)	Client is up and running
	Enter department name: Enter student ID:
After sending Student ID to Main Server:	Client has sent <Department Name> and Student<student ID> to Main Server using TCP over port <**dynamic TCP port**>
If input department not found	<Department Name>: Not found
If received message from main server saying that input Student ID not found	Student <student ID>: Not found
If input Student ID and department can be found and the result is received:	The performance statistics for Student < Student ID> in < Department Name> is: Student GPA: XXX Percentage Rank: XXX Department GPA Mean: XXX Department GPA Variance: XXX Department Max GPA: XXX Department Min GPA: XXX *(Replace XXX with your results)*
After the last query ends:	-----Start a new request----- Enter department name: Enter student ID:

ASSUMPTIONS

1. You must start the processes in this order: **Backend-server (A), Backend-server (B), Main-server, and Client 1, Client 2.**

2. The dataA.csv and dataB.csv 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/

(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 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 in addition to any other header file you used:

```
#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
- Your platform
- 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 Main Server	make servermain
Compile client	make client
Run Server A	./serverA
Run Server B	./serverB
Run Main Server	./servermain
Run client 1	./client
Run client 2	./client

TAs will first compile all codes using **make all**. They will then open 5 different terminal windows. On three terminals they will start servers A, B and Main Server using commands **./serverA**, **./serverB**, and **./servermain**. **Remember that servers should always be on once started.** On another two terminal they will start the client as **./client**. 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!

1. Upload "ee450_yourUSCusername.tar.gz" to Blackboard -> Assignments. After the file is submitted, you must click on the "submit" button to submit it. If you do not click on "submit", the file will not be submitted.

2. 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.

3. 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.

4. 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!

5. 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.

6. 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. 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 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, 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 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 (Announcements and Discussion Board) regularly 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.

EXTRA CREDIT (20% Extra to Project Part 3)

Can you recommend a student that may share the same academic interest with the input student?

Assumptions:

1. Recommendation can be across departments, but within the same backend server.
2. The recommendation calculation is processed in the same backend server as the student.
3. Design an algorithm to calculate the **similarity** between students **based on the academic records**, then recommend the student with the highest similarity (except the requesting student).
4. If there is a tie, pick the student with the smallest student ID.

Expected results:

Print the recommended student ID for each request, as below.

Table 6. Extra credit Client 1 or Client 2 on-screen messages

Event	**On-screen Messages**
Booting up(only while starting)	Client is up and running
	Enter department name: Enter student ID:
After sending Student ID to Main Server:	Client has sent <Department Name> and Student<student ID> to Main Server using TCP over port <**dynamic TCP port**>
If input department not found	<Department Name>: Not found
If received message from main server saying that input Student ID not found	Student <student ID>: Not found
If input Student ID and department can be found and the result is received:	The performance statistics for Student < Student ID> in < Department Name> is: Student GPA: XXX Percentage Rank: XXX Department GPA Mean: XXX Department GPA Variance: XXX Department Max GPA: XXX Department Min GPA: XXX **Friend Recommendation: <Student ID>** *(Replace XXX with your results)*