EE450 Term Programming Part 1

Fall 2019

Due Date: Sunday October 13th, 2019 11:59 PM (Midnight)

Hard Deadline

(Strictly Enforced: the submission website will automatically close on the deadline)

The objective of this assignment is to familiarize you with UNIX socket programming. This assignment is worth 3% of your overall grade in this course. 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, email TA your questions or come by TA's office hours. **You can ask TAs any** question **about the content of the project, but TAs has the right to reject your request for** debugging**.**

Problem Statement:

In this part of term project, you will implement TCP communication between client and server, where a single client send a greeting with the information of name to the server. This could help you get familiar with the socket programming for TCP communication.

To simplify the problem, the <NAME1> and <NAME2> will only be **a-zA-Z** will less than 10 characters.

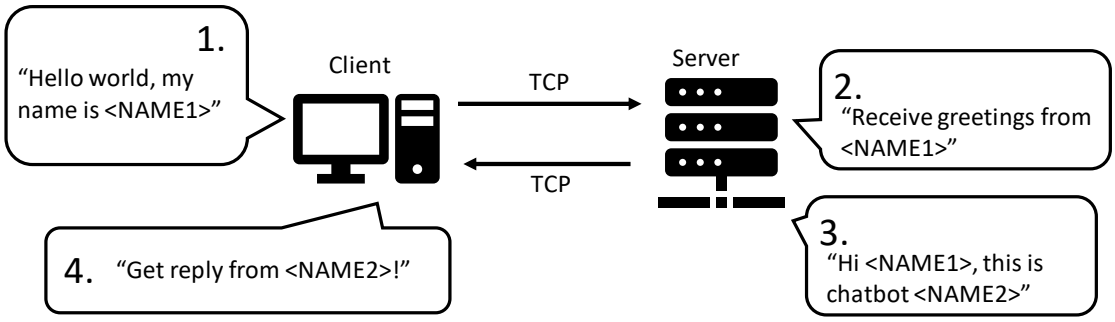

Figure 1. Illustration of the network

Source Code Files

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

1. Server: You must name your code file: **`server.c`** or aws.cc or **`server.cpp`** (all small letters). Also you must call the corresponding header file (if you have one; it is not mandatory) **`server.h`** (all small letters).

2. 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).

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 3. Static and Dynamic assignments for TCP ports		
Process	**Dynamic Ports**	**Static Ports**
Server	-	1 TCP with client, 21000+xxx
Client	1 TCP	<Dynamic Port assignment>

NOTE: xxx is the last 3 digits of your USC ID. For example, if the last 3 digits of your USC ID are "319", you should use the port: **21000+319 = 21319** for the server. **It is NOT going to be 21000319.**

ON SCREEN MESSAGES: Table 4. Server on screen messages	
Event	**On Screen Message (inside quotes)**
Booting up (Only while starting):	"The Server is up and running."
Upon receiving the input:	"Receive greetings from <NAME1>"
Upon sending the input to client:	"Send greetings to <NAME1>"

ON SCREEN MESSAGES: Table 8. Client on screen messages	
Event	**On Screen Message (inside quotes)**
Booting Up:	"The client is up and running"
Upon sending the input to serv	"The client sent greetings to the server"
After receiving the result from se	"Get reply from <NAME2>"

Example Output to Illustrate Output Formatting:

Server Terminal:

The Server is up and running
Receive greetings from ALICE
Send greetings to ALICE

Client Terminal:
The client is up and running
The client send greetings to the server
Get reply from BOB

Assumptions:

1. You have to start the processes in this order: **server, client.**

2. 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.

3. 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.

4. 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 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.

5. 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 port numbers that are to be obtained dynamically. Refer to Table 3 to see which ports are statically defined and which ones are dynamically assigned. Use `getsockname()` function to retrieve the locally-bound port number wherever ports are assigned dynamically as shown below:

```
/*Retrieve the locally-bound name of the specified socket
and store it in the sockaddr structure*/
```

```
Getsock_check=getsockname(TCP_Connect_Sock,(struct sockaddr
*)&my_addr, (socklen_t *)&addrlen);
//Error checking
if (getsock_check== -1) { perror("getsockname"); exit(1);
}
```

2. The host name must be hardcoded as **localhost (127.0.0.1)** in all codes.

3. Your client should terminate itself after all done. And the client can run multiple times to send requests. However, the backend servers and the AWS should keep running and be waiting for another request until the TAs terminate them by Ctrl+C. It 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/

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.
   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 BE SUBJECT TO A SERIOUS PENALTY

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

About the Makefile: makefile here is only for compiling your codes. Then two commands with one corresponding input will be run on their order.

`make all`	Compiles **all** your files and creates executables
`./server <NAME2>`	**Run**s server
`./client <NAME1>`	Starts the client

2. Compress all your files including the README file into a single "tar ball" and call it: **ee450_TermPart1_yourUSCusername_session#.tar.gz** e.g. my filename would be **ee450_TermPart1_songhao_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_TermPart1_yourUSCusername.tar *
   >> gzip ee450_ TermPart1_yourUSCusername.tar
   ```

   Now, you will find a file named "ee450_TermPart1_yourUSCusername.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. 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!

4. 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.

5. **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 AWS. But result is not received by the client because the AWS 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 5 out of 100 for the project.

6. If your submitted codes compile using make but when executed, produce runtime errors without performing any tasks of the project, you will receive 10 out of 100.

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

8. If your code does not correctly assign the TCP 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.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 5 out of 100.

10. Your code will not be altered in any ways for grading purposes and however it will be tested with different inputs. Your \ TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not. If your README is not consistent with the description, we will follow the description.

Cautionary Words:

1. In view of what is a recurring complaint near the end of a project, we want to make it clear that the target platform on which the project is supposed to run is *the provided **Ubuntu (16.04)***. It is strongly recommended that students develop their code on this virtual machine. In case students wish to develop their programs on their personal machines, possibly running other operating systems, they are expected to deal with technical and incompatibility issues (on their own) to ensure that the final project compiles and runs on the requested virtual machine. If you do development on your own machine, please leave at least three days to make it work on Ubuntu. It might take much longer than you expect because of some incompatibility issues.

2. 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.