EE 450
Spring 2013 Nazarian
Socket Programming Project

Assigned: Friday , Feb. 15
Due: Phase1- Friday, March 8, at 11:59pm
Phase2- Friday, April 5, at 11:59pm
Phase3- Friday, April 19, at 11:59pm
Late submissions will be accepted only during the first two days after deadline with a maximum of 15% penalty for each day. For each day, submissions between 12am and 1am: 2%, 1 and 2am: 4%, 2 and 3am: 8% and after 3am: 15%.
Maximum points: 100

The objective of this assignment is to familiarize you with UNIX socket programming. This assignment is worth 10% of your overall grade in this course.

Notes:

- It is an individual assignment and no collaborations are allowed. You may refer to the syllabus to review the academic honesty policies, including the penalties. Any questions or doubts about what is cheating and what is not, should be referred to the instructor or the TAs.
- Any references (pieces of code you find online, etc.) used, should be clearly cited in the readme.txt file that you will submit with your code.
- We may pick some students in random to demonstrate their design and simulations.
- Post your questions on project discussion forum. You are encouraged to participate in the discussions. It may be helpful to review all the questions posted by other students and the answers.
- If you need to email your TAs, please follow the syllabus guidelines mentioned under the assignments.

A. Problem Definition:

In this project you will be simulating an intelligence coordination system. The system consists of two Captains, a Major and a General. The communication between them will be over TCP and UDP sockets in a network with client-server architecture. The project has 3 major phases: Submission of the log files by captains to the Major, reporting to the General by the Major, and the General, ordering the Captain to execute the mission. In Phase 1 and 2 all communications are through UDP sockets. However, in phase 3 the communication is through TCP sockets.

B. Code and Input files:

You must write your program either in C or C++ on UNIX. In fact, you will be writing (at least) 3 different pieces of code:

1- Captains :
You must create 2 concurrent Captains

i. Either by using fork() or a similar Unix system call. In this case, you probably have only one piece of code for which you need to use one of these names: **Captain.c** or **Captain.cc** or **Captain.cpp.** Also you must call the corresponding header file (if any) **Captain.h**. You must follow this naming convention. Make sure the first letter of the word 'Captain' is capital.

ii. Or by running 2 instances of the Captain code. However in this case, you probably have 2 pieces of code for which you need to use one of these sets of names: (**Captain1.c, Captain2.c**) or (**Captain1.cc, Captain2.cc)** or (**Captain1.cpp, Captain2.cpp)**. Also you must call the corresponding header file (if any) **Captain1.h, Captain2.h**. You must follow this naming convention. Make sure the first letter of the word 'Captain' is capital.

2- General:
You must use one of these names for this piece of code: **General.c** or **General.cc** or **General.cpp**. Also you must call the corresponding header file (if any) **General.h**. You must follow this naming convention. Make sure the first letter of the word 'General' is capital.

3- Major:
You must use one of these names for this piece of code: **Major.c** or **Major.cc** or **Major.cpp**. Also you must call the corresponding header file (if any) **Major.h**. You must follow this naming convention. Make sure the first letter of the word 'Major' is capital.

4- Captain Input file: Captain1.txt, Captain2.txt,
These are the files that contain the mission logs that the Captains are supposed to send to the Major. The file consists of two lines and each line has encoded data about the mission. The Captains send this encoded data to the Major via UDP connection. Every line consists of two different parts which are separated with the special character "$". Thus, it is easier for you when you want to parse the input file and extract the information from there (check out the function strtok() when you are dealing with strings in C/C++). Here is the format of a sample log file:

Resources$4
Confidence$6

5- Input file: passMj.txt
This is the file that contains the password for the Major to communicate with the General. The file consists of the port number and password of the Major. The Major communicates with the General through a secure UDP connection in which it sends the password and the port number. Then, the General will match the received

password with his own database (obtained from another text file) and reply to the Major by saying whether the password is accepted or denied. Finally, if the password is correct, the Major sends the probability of success for each of the captains calculated from the data received from the Captains. The format of the passMj.txt is:

6010 indranil

6- Input file: passGn.txt
This is the file that contains the password for General to verify the password received from the Major. The file consists of the port number and password for the General to match the port number and the password sent by the Major. The connection between the General and the Major takes place through UDP, with both having static port numbers as mentioned in the table. After acknowledging and confirming received password, the General waits to receive the data from the Major. The format of the passGn.txt is:

6010 indranil

C. A more detailed explanation of the problem:

This project is divided into 3 phases. In each phase you will have multiple concurrent processes that will communicate either over TCP or UDP sockets. It is not possible to proceed to one phase without completing the previous phase and

Phase1:

In this phase, the Captains open their log file (Captain1.txt, or Captain2.txt) and send the encoded data to the Major. More specifically, each Captain opens a UDP connection with the Major to send the mission data. This means that the Captains should know the UDP port number of the Major in advance. In other words you must hardcode the UDP port number of the Major in the corresponding Captain code. Table 1 shows how static UDP and TCP port numbers should be defined for each entity in this project. Each Captain will use one dynamically-assigned UDP port number (different for each Captain) and establish one UDP connection to the Major. Thus, there will be two different UDP connections to the Major. The Major will print the data on the screen.

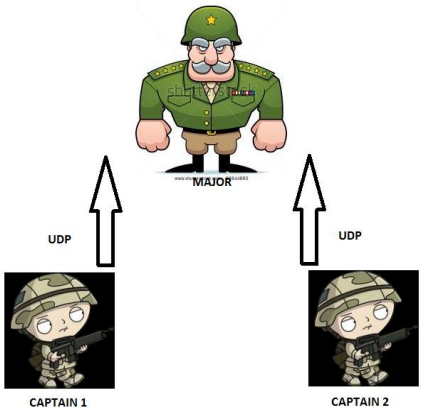

Figure 1. Communication in Phase 1

Phase2:

In phase 2 of this project, the Major calculates the probability of attack based on the log data received from the Captains. The confidence level of the Captain is already hardcoded in the code for Major. The formula for calculation of the success rate is as follows:

$$P(\text{success rate}) = (5 * \text{Resources} + 3 * \text{Confidence} + 2* \text{Record})$$

Where the values of the "Record" are as follows:
Captain1 = 8
Captain2 = 9

The Major sends the P(success rate) to the General through a secure UDP connection. More specifically, the Major opens up a UDP connection to the General to send the packets with the value of P(success rate). The Major first sends the data from the password file to the General. In this case you need to edit the first field in passMj.txt file which is the port number; according to the static Port Number you have assigned (as in the table below). The password will be provided to you. This means that the Major should know in advance the static UDP port of the General. You can hardcode this static UDP port setting the value according to Table 1. Then, it opens a UDP connection to this static UDP port of the General. Thus, for this phase there is one UDP connection to the General.

The General matches the password data in its own passGn.txt file, which contains the port number and the password for the Major. You will have to edit the passGn.txt file with the port numbers of the Major according to the static UDP port number you assign to the Major. The format is as follows:

port_no_Major password

The required password will be provided to you. If the password sent by the Major matches with the General's records, then the General would print the received data {P(success rate)} for each captain on the screen. If there is a password mismatch, then the General would print an error message. This phase will continue until the General receives the correct password data from the Major. This concludes phase 2.

Figure 2. Communication in Phase 2

Phase3:

In this phase, the General reads the packets from the Major, compares the values of P(success rate) and opens a TCP connection. The Major would ask the Captain with the higher success rate to launch the mission and the other one to stay as a backup. Each of the two Captains must open a TCP connection to listen to the command from the General prior to the commencement of this phase. The General would have the port numbers of each of the Captains hardcoded in the code. Then, when the Captain receives this packet it should print an appropriate message to the screen.

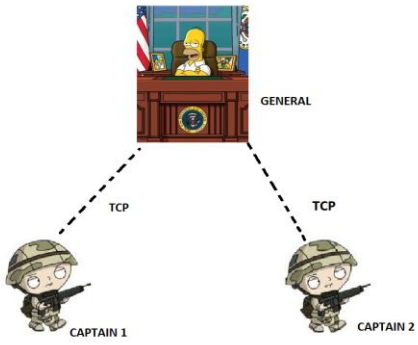

Figure 3. Communication in Phase 3

Table 1. A summary of Static and Dynamic assignment of TCP and UDP ports

Process	Dynamic Ports	Static Ports
Captain1	1 UDP (phase 1)	1 TCP, 21200 + xxx (last digits of your ID) (phase 3)
Captain2	1 UDP (phase 1)	1 TCP, 21300 + xxx (last digits of your ID) (phase 3)
Major		1 UDP, 3300 + xxx (last digits of your ID) (phase 1), 1 UDP, 3400 + xxx (last digits of your ID) (phase 2)
General		1 UDP, 3700 + xxx (last digits of your ID) (phase 2), 1 TCP, 3800 + xxx (last digits of your ID) (phase 3)

Note that if your USC ID number is 0123-4567-89, the static TCP port number of your Major 1 in the first phase will be 3300+789 = 4089.

D. On-screen Messages:

In order to clearly understand the flow of the project, your codes must print out the following messages on the screen as listed in the following Tables.

Table 2. Captains' on-screen messages

Event	On Screen Message
Upon start of Phase 1	<Captain#> has UDP port … and IP address …
Sending a packet to the Major	Sending log file to the <Major>
End of Phase 1	End of Phase 1 for <Captain#>
Upon Startup of Phase 3	<Captain#> has TCP port … and IP address …
Upon establishing a TCP connection to the General	<Captain#> is now connected to General
Upon receiving the command to launch a mission	<Captain#> has started the Mission!!!!!! <Captain#> is the back up for the mission!
End of Phase 3	End of Phase 3 for <Captain#>

Table 3. Major's on-screen messages

Event	On Screen Message
Upon startup of Phase 1	<Major> has UDP port … and IP address … for Phase 1
Upon receiving an UDP connection to the Captains	<Major> is now connected to the Captain#
Upon receiving the data from the Captains	<Major> has received the following: 1. <Captain#>: Resources - # , SuccessRate - #, Record- # 2. <Captain#>: Resources - # , SuccessRate - #, Record - #
End of Phase 1	End of Phase 1 for <Major>
Upon startup of Phase 2	<Major> has UDP port … and IP address … for Phase 2
Upon calculating average P(success rate)	< Captain 1> has P(success rate) = # < Captain 2> has P(success rate) = #
Sending the average P(success rate) to the General	<Major> sent the P(success rate) to the General

Table 4. General's on-screen messages

Event	On screen message
Upon startup of Phase 2	The General has UDP port … and IP address …
Upon receiving the password data from the Major	Received the password from <Major>
If password and port number are correct	Correct password from <Major>
If password and port number mismatch	Wrong password from <Major>, try again
End of Phase 2	End of Phase 2 for the General
Upon startup of Phase 3	The General has TCP port … and IP address …
Printing the Captain with the highest average P(success rate)	<Captain#> has the highest probability of success
Sending the mission/backup to the Captains	Command to start the mission sent to <Captain#> Command to be the backup for the mission sent to <Captain#>
End of Phase 3	End of Phase 3 for the General

E. Assumptions:

1. The Processes are started in this order: Major, Captain, General and then we have some delay (e.g 2 seconds) to guarantee that the data are received in the appropriate sequence.

2. If you need to have more code files than the ones 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.
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 (from past logins or previous runs of code that are still not terminated and hold the port busy). 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.

F. Requirements:

1. Do not hardcode the TCP or UDP port numbers that must be obtained dynamically. Use getsockname() function to retrieve the locally-bound port number wherever ports are assigned dynamically. Refer to Table 1 to see which ports are statically defined and which ones are dynamically assigned. Sometimes, if you call getsockname() before the first call of sendto()/send(), you will get a port number 0. If that is the case, make sure you call getsockname() after the first call of sendto()/send() in your code. It is okay to postpone the on-screen messages till you have a valid port number.
2. You can use gethostbyname() or getaddrinfo() to obtain the IP address of nunki.usc.edu or the local host (127.0.0.0).
3. You can either terminate all processes after completion of phase3 or assume that the user will terminate them at the end by pressing ctrl-C.
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. No user interaction must be required (except for when the user runs the code obviously). Everything is either hardcoded or dynamically generated as described before. By hardcoded, we mean that there should be #define statements in the beginning of the source code files with the corresponding port numbers. If you do not follow this requirement and use the port numbers directly inside your code, we will deduct points.
6. All the on-screen messages must conform exactly to the project description. You must not add any more on-screen messages or modify them. If you need to do so for debugging purposes, you must comment out all of the extra messages before you submit your project.
7. Using 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 new process in phases 1 and 3 when a new TCP connection is accepted is mandatory and everyone should support it. This is useful when three different clients are trying to connect to the same server simultaneously. If you don't use fork in the server when a new connection is accepted, the 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.

G. Programming platform and environment:

1. All your codes must run on nunki (nunki.usc.edu) and only nunki. It is a SunOS machine at USC. You should all have access to nunki, if you are a USC student.
2. You are not allowed to run and test your code on any other USC Sun machines (e.g. aludra.usc.edu). This is a policy strictly enforced by ITS and we must abide by that.
3. No MS-Windows programs will be accepted.
4. You can easily connect to nunki if you are using an on-campus network (all the user room computers have xwin already installed and even some ssh connections already configured).
5. If you are using your own computer at home or at the office, you must download, install and run xwin on your machine to be able to connect to nunki.usc.edu and here's how:
    a. Open software.usc.edu in your web browser.
    b. Log in using your username and password (the one you use to check your USC email).
    c. Select your operating system (e.g. click on windows XP) and download the latest xwin.
    d. Install it on your computer.
    e. Then check the webpage: http://www.usc.edu/its/connect/index.html for more information as to how to connect to USC machines.
6. Please also check this website for all the info regarding "getting started" or "getting connected to USC machines in various ways" if you are new to USC: http://www.usc.edu/its/

H. 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/

Once you run xwin and open an ssh connection to nunki.usc.edu, you can use a unix text editor like emacs or vi to type your code and then use compilers such as g++ (for C++) and gcc (for C) that are already installed on nunki 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 -lsocket -lnsl -lresolv
g++ -o yourfileoutput yourfile.cpp -lsocket -lnsl -lresolv
```

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>
```

I. Submission Rules:

1. Along with your code files, include a README file. In this 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. What the TA should do to run your programs. (Any specific order of events should be mentioned.)
    f. The format of all the messages exchanged should follow the ones as given in the table.
    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.)
2. Include a makefile in order for us to be able to compile your source code. Make sure the executable files that you generate are named: Captain, Major, and General (first letter is capital). If you don't use fork() to generate concurrent Captain and Major processes your executable files should be: Captain1, Captain2, Major, and General (first letter is capital). If you haven't written a makefile before, you can search the web for sample makefiles in order to create yours.
3. 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 file name would be ee450_dimitria_session1.tar.gz. Please make sure that your name matches the one in the class list. Also, do not include the special character # in the filename since we won't be able to download your project from the Digital Dropbox. Here are the instructions:
    a. On nunki.usc.edu, go to the directory which has all your project files. Remove all executable and other unnecessary files. Only include the required source code files and the README file. Now run the following commands:
    b. **tar cvf ee450_yourUSCusername_session#.tar *** - Now, you will find a file named "ee450_yourUSCusername_session#.tar" in the same directory.
    c. **gzip ee450_yourUSCusername_session#.tar** – Now, you will find a file named "ee450_yourUSCusername_session#.tar.gz" in the same directory.

d. Transfer this file from your directory on nunki.usc.edu to your local machine. You need to use an FTP program such as FileZilla to do so. (The FTP programs are available at software.usc.edu and you can download and install them on your windows machine.)

4. Upload "ee450_yourUSCusername_session#.tar.gz" to the Digital Dropbox (available under Tools) on the DEN website. 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.
5. 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.
6. Please do not wait till the last 5 minutes to upload and submit your project.
7. **You have plenty of time to work on this project and submit it in time -- Please refer to the header information of this project description file for the late submission policy. Note that any submission which is late by more than two days will receive a zero.**

J. Grading Criteria:

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 but when executed, produce runtime errors without performing any tasks of the project, you will receive 10 out of 100.
7. If your codes compile but when executed only perform phase 1 correctly, you will receive 40 out of 100.
8. If your codes compile but when executed perform only phase 1 and phase 2 correctly, you will receive 80 out of 100.
9. If your code compiles and performs all tasks in all 3 phases correctly and error-free, and your README file conforms to the requirements mentioned before, you will receive 100 out of 100.
10. If you forget to include any of the code files or the README file in the project tar-ball that you submitted, you will lose 5 points for each missing file (plus you need to send the file to the TA in order for your project to be graded.)
11. If your code does not correctly assign and print the TCP or UDP port numbers dynamically (in any phase), you will lose 20 points.
12. You will lose 5 points for each error or a task that is not done correctly.
13. The minimum grade for an on-time submitted project is 10 out of 100.
14. 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.
15. Using fork() or similar system calls, for example in the creation of three different Captain processes is not mandatory. If you do use fork() or similar system files in your codes to create concurrent processes (or threads) and they function correctly you will

receive 10 bonus points. Note that the use of fork() for the creation of a new process in phase 1 and 3 when a new connection is accepted is mandatory and everyone should support it. This is useful when two different clients are trying to connect to the same server simultaneously. If you don't use fork in the server when a new connection is accepted, the server won't be able to handle the concurrent connections. You won't receive extra credit for this case of fork(), only when you create the four Captains using the same source file Captain.c/Captaincpp and when you create the two Majors using the same source file Major.c/Major.cpp.

16. The maximum points that you can receive for the project with the bonus points is 100. In other words the bonus points will only improve your grade if your grade is less than 100.
17. Your code will not be altered in any ways for grading purposes and however it will be tested with different input files. Your designated TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not.

K. 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 nunki.usc.edu. It is strongly recommended that students develop their code on nunki. 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 nunki.
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 even from your past logins to nunki, try this command: ps –aux | grep <your_username>
4. Identify the zombie processes and their process number and kill them by typing at the command-line: kill -9 <processnumber>
5. You can kill a process if you know its name by typing the command line: killall -9 <processname>
6. If you want to run all or some of the processes in the same terminal you can use the special character '&' after the name of the process. For example, you can type the following in the same terminal:
   ./ Captain &
   ./ Major &
   ./ General
7. There is a cap on the number of concurrent processes that you are allowed to run on nunki. If you forget to terminate the zombie processes, they accumulate and exceed the cap and you will receive a warning email from ITS. Please make sure you terminate all such processes before you exit nunki.
8. Please do remember to terminate all zombie or background processes, otherwise they hold the assigned port numbers and sockets busy and we will not be able to run your code in our account on nunki when we grade your project.

Good luck.