EE450 Socket Programming Project, Fall 2011
Due Date: Sunday November 27th, 2011 11:59 AM (Noon)
(The deadline is the same for all on-campus and DEN off-campus students)
Hard deadline (Strictly enforced)

The objective of this assignment is to familiarize you with UNIX socket programming. This assignment is worth **10%** 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 please feel free to contact the TAs and cc professor Zahid as usual.

The Problem:
In this project you will be simulating an instant messaging system. After authentication with a LogIn server, the users can exchange instant text messages through a SuperNode. All communications take place over TCP and UDP sockets in client-server architecture. The project has 3 major phases: User authentication, activation of SuperNode, instant messaging between users.

LogIn Server:
You must create one LogIn server and use one of the following names for the code: "Login.c", "Login.cc" or "Login.cpp". Also you must call the corresponding header file (if any) "Login.h". You must strictly follow this naming convention.

SuperNode:
You must create one SuperNode and use one of the following names for the code: "Supernode.c", "Supernode.cc" or "Supernode.cpp". Also you must call the corresponding header file (if any) "Supernode.h". You must follow this naming convention.

User:
You must create 3 concurrent users.
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: User.c or User.cc or User.cpp (all small letters). Also you must call the corresponding header file (if any) User.h (all small letters). You must follow this naming convention.

ii. Or by running 3 instances of student code. However in this case, you probably have 3 pieces of code for which you need to use one of these sets of names: (user1.c, user2.c, user3.c) or (user1.cc, user2.cc, user3.cc) or (user1.cpp, user2.cpp, user3.cpp) (all small letters). Also you must call the corresponding header file (if any) User.h (all small letters) or user1.h, user2.h, user3.h (all small letters). You must follow this naming convention.

Input Files for LogIn server (Phase 1): UserPassMatch.txt

This is the file that contains the registered valid username/password matches for each user. Each line of this file is a username followed by a space followed by a password. This is the information the login server will check each time a user attempt's a login to the server.

Example file:
Bob 123456
Jain pass123

In this example the file contains two sets of usernames and passwords (just two lines): the first with username "Bob" and password: "123456" and the second one with username "Jain" and password "pass123".

Input Files for the User (Phase 1): UserPass1.txt, UserPass2.txt, UserPass3.txt
These files contain the username/password for the specific user. The file has just one line specifying the username/password pair as shown. The username/password pair can be a valid or an invalid pair.

Examples file for User#1:
Bob 123456

Input Files for the User (Phase 3): UserText1.txt, UserText2.txt, UserText3.txt
UserText1.txt, UserText2.txt, UserText3.txt contain the outgoing text of users 1 to 3, respectively. A sample input file for user1 is provided as follows:

User#2-User#1:Hello Bob! How are you?
User#1-User#2:Hello Alice
User#2-User#1:Did you finish your project?
User#1-User#2:Yes, I did.
User#2-User#3:How was the food?
User#3-User#2:It was delicious.
User#1-User#3:Hello Sir! How can I help you today?
User#3-User#1:I need to pay my bill.

In this file each line starts with the identifier string that refers to the receiver followed by the sender of the message. For instance, in the given example, User#1 sends "Hello Bob! How are you?" to User#2.

A more detailed explanation of the problem

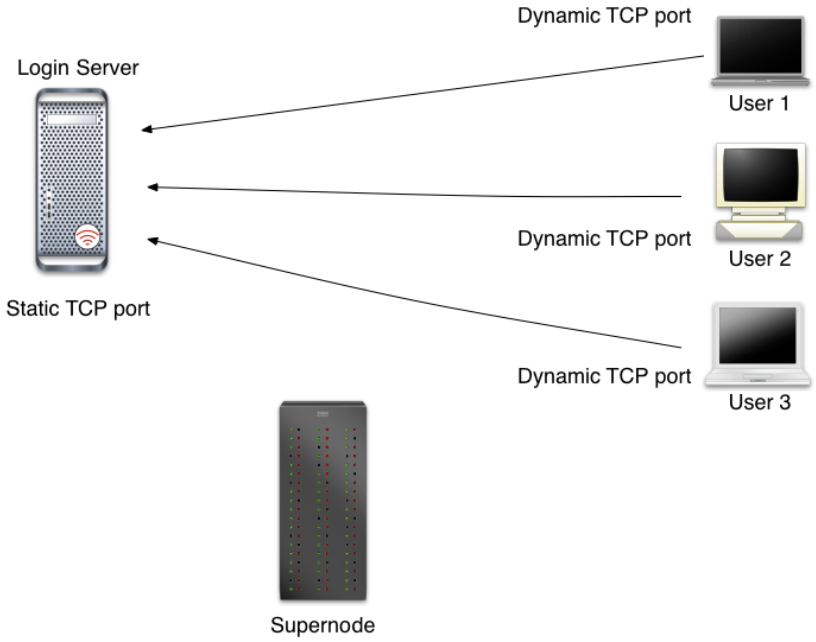
Phase1:

In this phase, the LogIn server and the users will open their .txt files. The LogIn server will load the matching usernames and passwords and the users will load their respective username and password. Now the LogIn server will open a TCP socket and start listening on it. Each user will try to establish a TCP connection to the server and try to login. For this purpose each user will have the TCP port of the LogIn server hardcoded so that it knows where to connect to (please

refer to table 1 for details). Please note that in total there will be three different TCP connections simultaneously, one for each user.

For the login process each user will send the following command: "Login#username password" where everything before the pound key (#) will be interpreted as a command by the server and everything after will be the argument. The server will check to see if this username/password combination is a valid match and will reply accordingly. If it was successful the reply message should be: "Accepted#" and if it was unsuccessful: "Rejected#". Upon acceptance the server will save the IP address of the accepted user and will bind it to its username for future reference. It will also send a reply packet to the user containing the IP address and the port number of the SuperNode. If the user is rejected then it will remain idle and disregarded from that point onwards. For simplicity, use one command and its arguments per packet. After the exchange of messages is completed the server will close the TCP connection to each user. The users will create UDP sockets to listen to.

Figure Phase1:

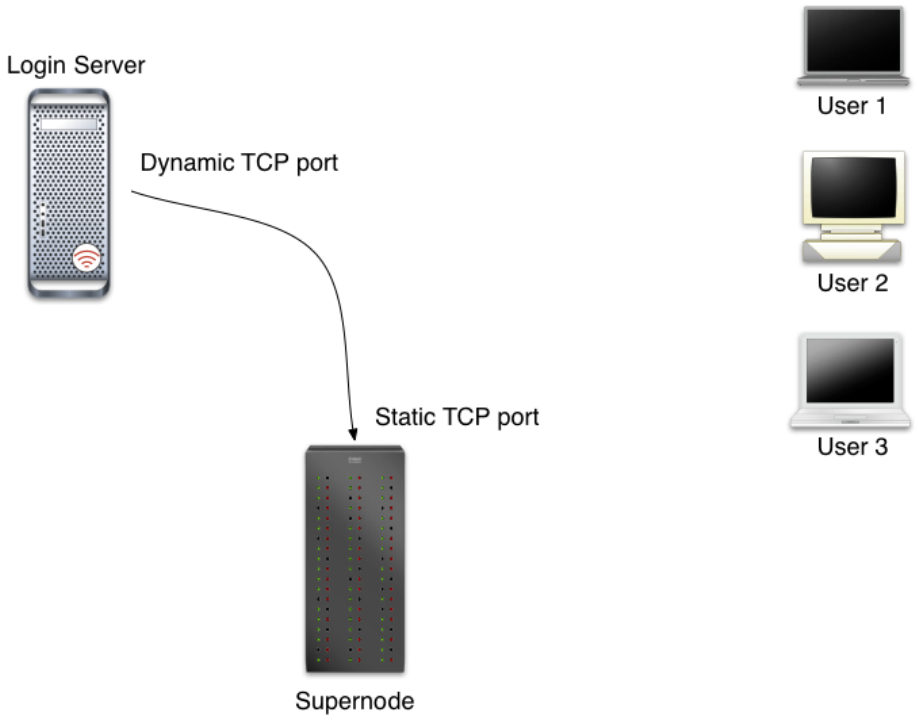


Phase2:

In this phase, the LogIn server will send the usernames and their corresponding IP addresses to the SuperNode. To do so, while the SuperNode is listening on a static TCP port, the LogIn server opens a TCP connection to the SuperNode. Once the connection is established, the LogIn server will send the "username/IP address" information of logged-in users over the TCP connection. When the SuperNode receives this information, it saves it and closes the TCP connection. Therefore, the LogIn server needs to know the TCP port of the SuperNode in advance. In other

words, you must hardcode the TCP port number of the SuperNode in the LogIn server code, but the TCP port number of the LogIn server is dynamically assigned. You can hardcode this static TCP port according to Table 1. When the SuperNode fully receives the information from the LogIn server, it displays an appropriate message according to Tables 2 and 3.

Figure Phase2:

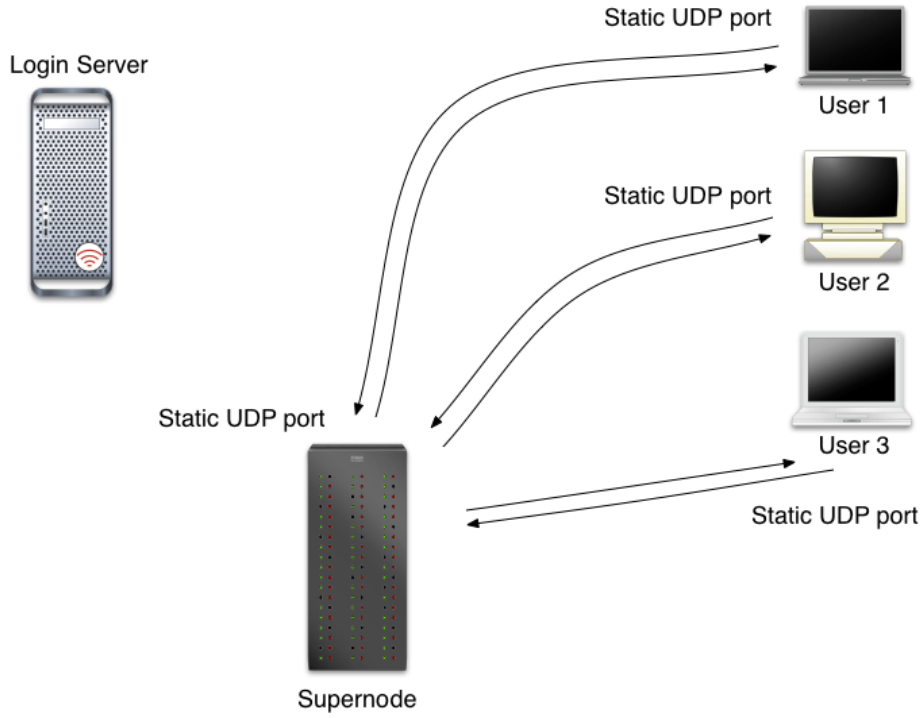


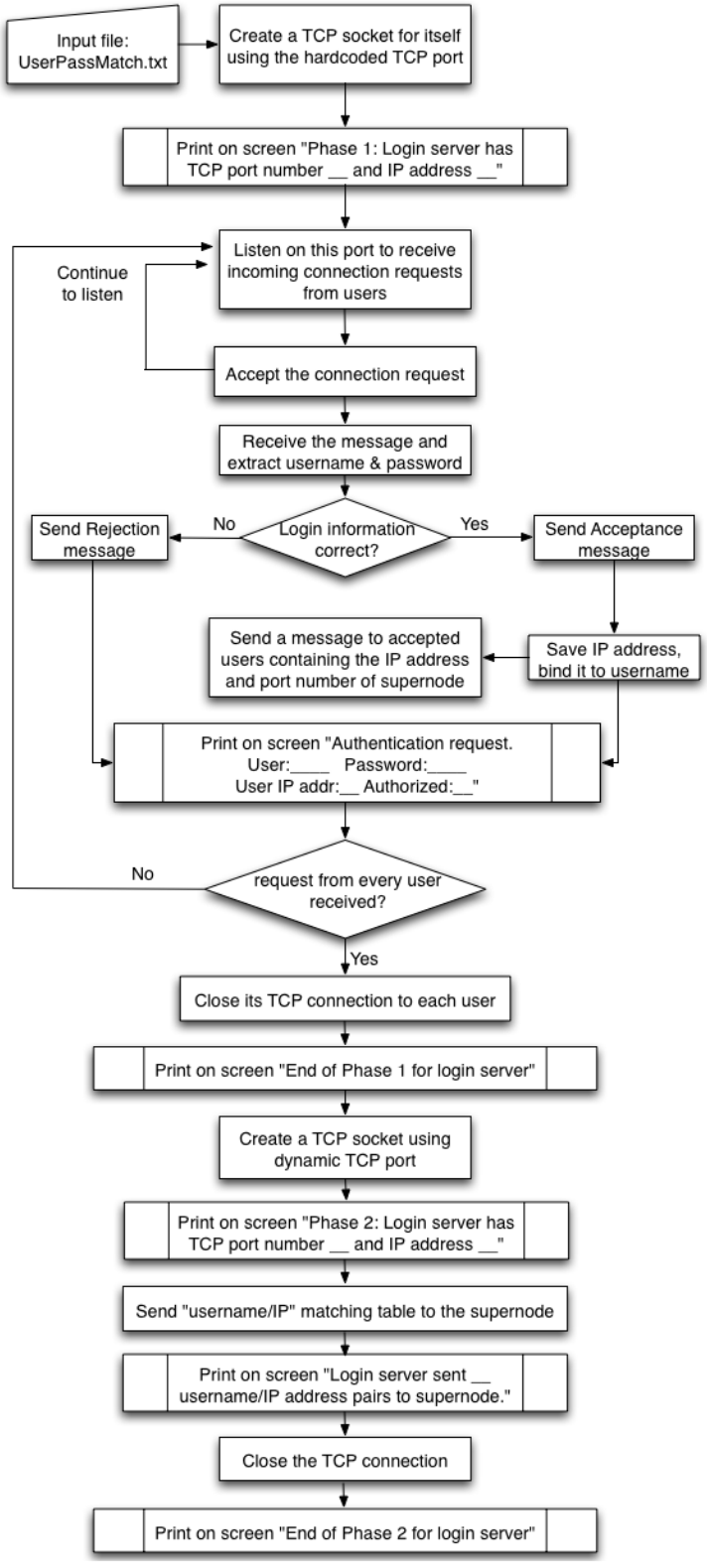
Phase 3:

In this phase, the users communicate with each other through UDP with the SuperNode acting as a relay. This means that, if the User#1 wants to chat with User#2, then User#1 will send a UDP packet containing the chat message (meant for User#2) to the SuperNode. This requires the user to know the IP address/port number of the SuperNode. It gets this information from the Login Server at the end of the Phase 1. The SuperNode will parse the message from User#1 to figure out which user this message is destined to. In this case, the SuperNode finds out that the message was meant for User#2. So the SuperNode sends out a UDP packet containing the chat message to User#2. Each user will send all its chat messages to the single UDP port on which the SuperNode is listening. The SuperNode will relay the received chat messages to the respective user ports. This means that, for sending the message the SuperNode should know in advance the static UDP port number of each user. (For port number details refer Table 1).

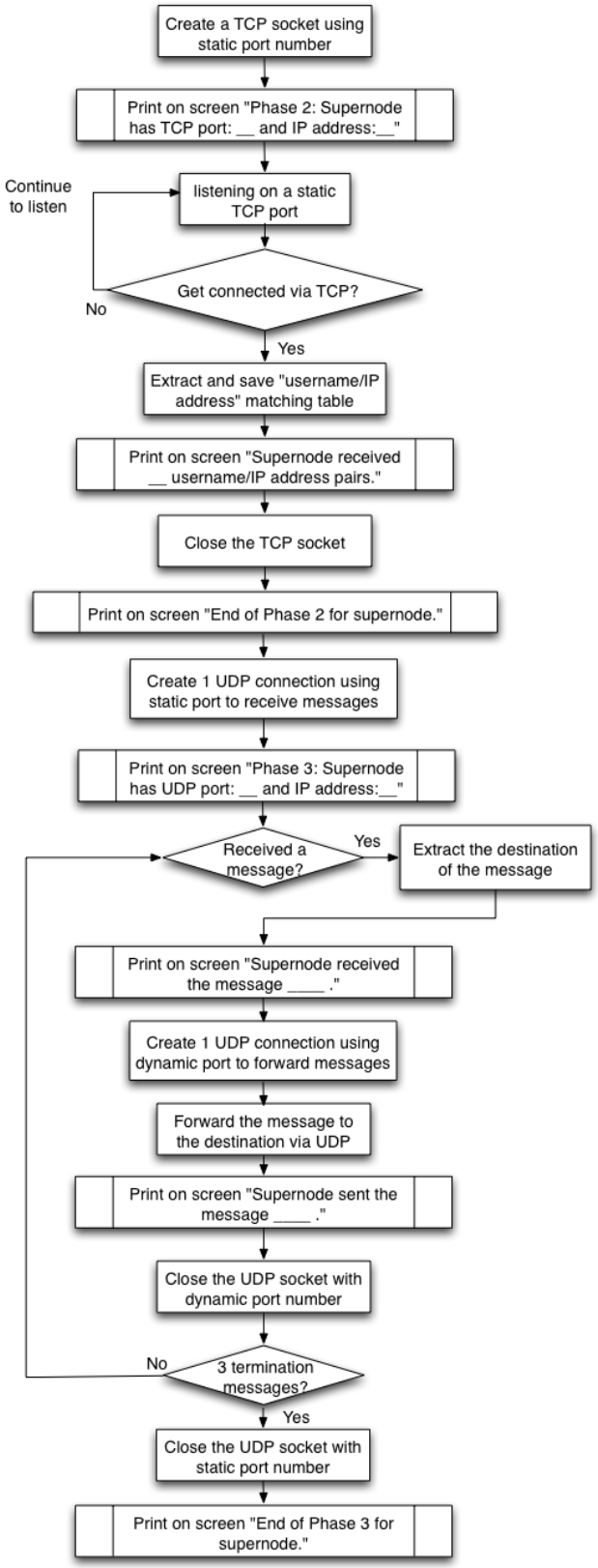
The chat messages for the users will be provided as an input text file. A sample message in this text file for User#1 will look like this "User#2-User#1: Hello Bob! How are you?" where everything before the hyphen (-) specifies the receiver and everything between the hyphen and the colon (:) specifies the sender of the chat message. The SuperNode will receive this entire message verbatim. Entire message means the sender name, receiver name and the chat-message. After parsing, it finds out that this message was meant for User#2. The SuperNode will forward that entire message to User#2 at its static UDP port. Remember that the User#2 is listening at its static UDP port which is hardcoded in the SuperNode code. The message received at User#2 should be displayed on the screen like this "User#1: Hello Bob! How are you?"

Figure Phase3:

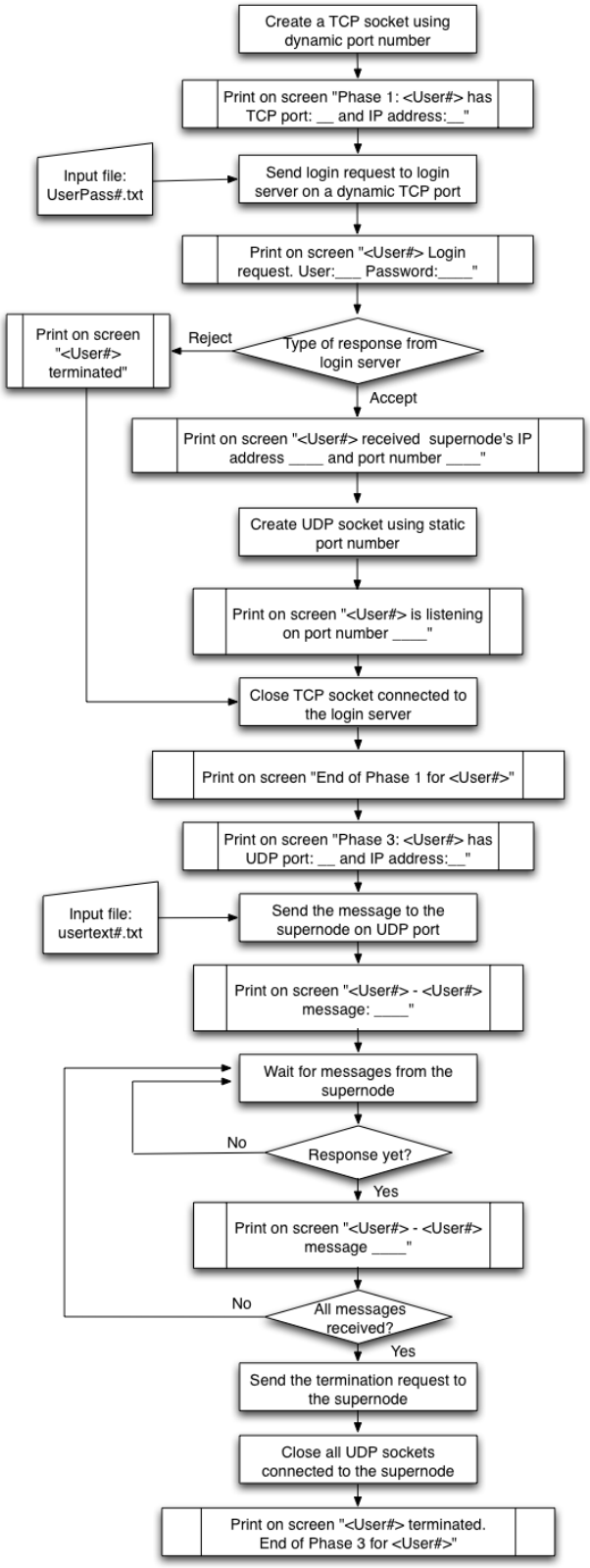




Flowchart: LogIn Server



Flowchart: SuperNode



Flowchart: Users

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

Process	Dynamic Ports	Static Ports
LogIn Server	1 TCP (phase 2)	1 TCP, 21100+xxx (last three digits of your ID) (phase 1)
SuperNode	UDP as many as required (phase 3)	1 TCP, 22100+xxx (last three digits of your ID) (phase 2) 1 UDP, 3100 + xxx (last digits of your ID) (phase 3)
User#1	1 TCP (phase 1) 1 UDP(phase 3)	1 UDP, 3200 + xxx (last digits of your ID) (phase 3)
User#2	1 TCP (phase 1) 1 UDP(phase 3)	1 UDP, 3300 + xxx (last digits of your ID) (phase 3)
User#3	1 TCP (phase 1) 1 UDP(phase 3)	1 UDP, 3400 + xxx (last digits of your ID) (phase 3)

Table 2. LogIn Server on screen messages

Event	On Screen Message
Upon startup of phase 1	Phase 1: LogIn server has TCP port number _____ and IP address _____
Upon receiving authentication request	Phase 1: Authentication request. User: _____ Password: _____ User IP Addr: ____. Authorized: ____
Upon acceptance send the IP and Port number of the SuperNode to the Users.	Phase 1: SuperNode IP Address: _____ Port Number: _____ sent to the <User#>
End of Phase 1	End of Phase 1 for Login Server
Upon startup of phase 2	Phase 2: LogIn server has TCP port number _____ and IP address _____
When "username/IP" matching table is sent.	Phase 2: LogIn server sent ___ (# of users logged in) username/IP address pairs to SuperNode.
End of Phase 2	End of Phase 2 for Login Server

Table 3. SuperNode on screen messages

Event	On Screen Message
Upon startup of phase 2	Phase 2: SuperNode has TCP port ___ and IP address: ____
When "username/IP" matching table is received.	Phase 2: SuperNode received ___ username/IP address pairs.
End of Phase 2	End of Phase 2 for SuperNode.
Upon startup of phase 3	Phase 3: SuperNode has static UDP port _____ IP address _____
Receiving a connection	Phase 3: SuperNode received the message _____ (print out message here)
Forwarding the message	Phase 3: SuperNode sent the message _____(print out the message here) on dynamic UDP port number _____

End of Phase 3	End of Phase 3 for SuperNode.

Table 4. Users on screen messages

Event	**On Screen Message**
Upon startup of phase 1	Phase 1: <User#>__ has TCP port ___ and IP address: ____
When sending login request	Phase 1: Login request. User: ____ password: _____
When receiving reply to login request	Phase 1: Login request reply: _____.
If the reply is Accept	Phase 1: Supernode has IP Address _____and Port Number_____
After creating UDP socket	Phase 1: <User#> is listening on Port Number _____
End of Phase 1	End of Phase 1 for <User#>.
Upon Startup of Phase 3	Phase 3: <User#> has static UDP port ____ IP address _____
Sending the message	Phase 3: <User#> is sending the message _____(print out the message here) on UDP dynamic port number _____
Receiving the message	Phase 3: <User#> received the message _____(print out the message here)
End of Phase 3	End of Phase 3 for <User#>

Assumptions:

1. The Processes are started in this order: Login, Supernode and User. You are allowed to use delays in your code to allow the LogIn server to send the username/IP matching table to the SuperNode.

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.

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.

Requirements:

1. Do not hardcode the TCP or UDP port numbers that are to be obtained dynamically. Refer to Table1 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. Use gethostbyname() to obtain the IP address of nunki.usc.edu or the local host however the host name must be hardcoded as nunki.usc.edu or localhost in all pieces of code.

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.

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. Using fork() or similar system calls are not mandatory if you do not feel comfortable using them to create concurrent processes.

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 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. 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 following 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/

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 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>
```

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.
   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 files WILL NOT BE GRADED.

2. Compress all your files including the README file into a single "tar ball" and call it: **ee450_yourUSCusername_session#.tar.gz** (all small letters) e.g. my file name would be **ee450_hkadu_session1.tar.gz**. Please make sure that your name matches the one in the class list. 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. **you@nunki>>** tar cvf **ee450_yourUSCusername_session#.tar** * - Now, you will find a file named "ee450_yourUSCusername_session#.tar" in the same directory.

   c. **you@nunki>>** 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 CoreFtp to do so. (The FTP programs are available at software.usc.edu and you can download and install them on your windows machine.)

3. 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.

4. Right after submitting the project, send a one-line email to your designated TA (NOT all TAs) informing him or her that you have submitted the project to the Digital Dropbox. Please do NOT forget to email the TA or your project submission will be considered late and will automatically receive a zero.

5. You will receive a confirmation email from the TA to inform you whether your project is received successfully, so please do check your emails well before the deadline to make sure your attempt at submission is successful.

6. You must allow at least 12 hours before the deadline to submit your project and receive the confirmation email from the TA.

7. By the announced deadline all Students must have already successfully submitted their projects and received a confirmation email from the TA.

8. 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.

9. Please do not wait till the last 5 minutes to upload and submit your project because you will not have enough time to email the TA and receive a confirmation email before the deadline.

10. Sometimes the first attempt at submission does not work and the TA will respond to your email and asks you to resubmit, so you must allow enough time (12 hours at least) before the deadline to resolve all such issues.

11. **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:

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 phase1 correctly, you will receive 40 out of 100.

8. If your codes compile but when executed perform only phase1 and phase 2 correctly, you will receive 70 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 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 0f 100.

15. Using fork() or similar system calls are not mandatory however 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.

16. If you submit a makefile or a script file along with your project that helps us compile your codes more easily, you will receive 5 bonus points.

17. 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.

18. 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.

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:

5. Kill -9 processnumber

6. 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.

7. 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.

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.