EE450 Socket Programming Project, Fall 2014

Due Date: Sunday November 30th, 2014 11:59 PM (Midnight)

(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. Before that, make sure you have **read the whole project description carefully**.

Problem Statement:

In this project you will be simulating an online medical appointment system using a hybrid architecture of TCP and UDP sockets. The project has three major phases: 1) authenticating into the health center server, 2) requesting available appointments and reserving one of them, and 3) sending insurance information to the corresponding doctor to get an estimated cost for the visit.

In Phases 1 and 2, all communications are through TCP sockets. In phase 3, however, all the communications are over UDP sockets. The main components of this network architecture are: 1) one health center server which keeps track of available appointments and it is also in charge of giving out appointments to the requesting patients, 2) two patients who are going to ask the health center server for available appointments and reserve one of them (the health center server will coordinate the process to avoid any conflicts of the appointments), and 3) two doctors that can get requests from the patients and provide estimated costs of the visits

Input Files Used:

The files specified below will be used as **inputs** in your programs in order to **dynamically** configure the state of the system. The contents of the files should **NOT** be "**hardcoded**" in your source code, because during grading, the input files will be different, but the formats of the files will remain the same.

1. **patient1.txt:** This input file contains the username and password of the first patient. It will contain only one row with exactly the following contents (during grading, we will only change the patient's password, but not the patient's name):

patient1 password111

2. **patient2.txt:** This input file contains the username and password of the second patient. It will contain only one row with exactly the following contents (during grading, we will only change the patient's password, but not the patient's name):

   patient2 password222

3. **users.txt:** This input file contains information about the patients that have registered already with the health center server. No other patient is allowed to use the health center server. Each row corresponds to the username and password of each patient. The contents of the file will be exactly as follows (during grading, we will only change the patients' passwords, but not the patients' name):

   patient1 password111
   patient2 password222

4. **availabilities.txt**: This input file contains information about the available appointments of the two doctors. Each row in the file corresponds to one available appointment. There are 6 rows in total in the file, i.e., there are only 6 available appointments. The first column in each row is the unique ID of the appointment called *appointment time index*. The second column is the day of the appointment, and the third column is the time of the appointment. The fourth and fifth columns are the corresponding doctor's name and port number to access them. We only care about the port numbers, since all the programs will run on the same machine (i.e. Nunki) and the IP will be the same for every program (i.e. 127.0.0.1 or 0.0.0.0). The format of the file will be:

   ```
   TimeIndex# Day Time DoctorId Port#
   ```

   It will contain 6 rows with exactly the following contents (during grading, we will test it with another file with 6 rows as well):

   ```
   1 Tue 01pm doc1 41000
   2 Mon 03pm doc2 42000
   3 Thu 02pm doc1 41000
   4 Wed 10am doc1 41000
   5 Sat 12pm doc2 42000
   ```

```
6 Fri 04pm doc2 42000
```

5. **doc1.txt:** This file contains the amount that the patient will have to pay from his/her pocket if he/she is insured under some given insurance and gets an appointment for doctor 1. There are three insurances that the doctor accepts: insurance1, insurance2 and insurance3. Here, we assume that both patients only use one of those three insurances and both doctors support all three insurances (only the price varies).
   The content of the file will be exactly as follows (during grading, we will only change the cost of each insurance, but not the insurance names):

   insurance1 30
   insurance2 20
   insurance3 50

6. **doc2.txt:** This file contains the amount that the patient will have to pay from his/her pocket if he/she is insured under some given insurance and gets an appointment for doctor 2. There are three insurances that the doctor accepts: insurance1, insurance2 and insurance3. Here, we assume that both patients only use one of those three insurances and both doctors support all the insurances (only the price varies).
   The content of the file will be exactly as follows (during grading, we will only change the cost of each insurance, but not the insurance names):

   insurance1 40
   insurance2 60
   insurance3 10

7. **patient1insurance.txt:** This file contains the insurance information for patient 1. There is going to be only one line in the file containing the insurance name that this patient is registered with. The contents of this file will be as follows (during grading, we will test a different insurance type from the three types supported).

   insurance1

8. **patient2insurance.txt:** This file contains the insurance information for patient 2. There is going to be only one line in the file containing the insurance name that this patient is registered with. The contents of this file will be as follows (during grading, we will test a different insurance type from the three types supported).

insurance2

Note: both patients can be registered with the same insurance.

Source Code Files

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

1. Health Center Server: You must use one of these names for this piece of code: **healthcenterserver.c** or **healthcenterserver.cc** or **healthcenterserver.cpp** (all small letters). Also you must call the corresponding header file (if you have one; it is not mandatory) **healthcenterserver.h** (all small letters). You must follow this naming convention! This piece of code basically handles the health center server duties.
2. Patient 1 and 2**:** The name of this piece of code must be **patient#.c** or **patient#.cc** or **patient#.cpp** (all small letters) and the header file (if you have one; it is not mandatory) must be called **patient#.h** (all small letters). The "#" character must be replaced by the patient number (i.e. 1 or 2), depending on which patient it corresponds to. *For people that are familiar with multithreading, for this part of the code, you should NOT use fork().*

3. Doctor 1 and 2**:** The name of this piece of code must be **doctor#.c** or **doctor#.cc** or **doctor#.cpp** (all small letters) and the header file (if you have one; it is not mandatory) must be called **doctor#.h** (all small letters). The "#" character must be replaced by the doctor number (i.e. 1 or 2), depending on which doctor it corresponds to. In case you are using one executable for both doctors (i.e. if you choose to make a multithreaded implementation), you should use the same naming convention without adding the doctor's number at the end of the file name (e.g. doctor.c). *In order to create two doctors in your system using one executable, you can use the fork() function inside your doctor's code to create 2 child processes.*

Phase 1: (Authentication)

In the first phase of the project, you will implement the authentication of the patients with the health center server. Specifically, each patient connects to the health center server through a TCP connection and informs the heath center server of its credentials (i.e. username and password) in order to authenticate. The health center server will match the provided credentials against its user database and respond back with a message indicating "**success**" or "**failure"** of the authentication process.

Phase 2: (Appointment Booking)

The patients would enter this phase only if they pass the first phase **successfully**, i.e. they provide valid usernames and passwords. Upon entering the second phase, they should send a

request to the health center server asking for a list of available appointments. The request will be sent over the same TCP port from phase 1 and the health center server will send the available appointments in a string format to the requesting patient. The requesting patient will choose 1 row from the received appointment information as its preferred appointment time, by entering the respective time index from the keyboard. Then, the patient will send the time *index* of the chosen appointment to the health center server through the same TCP connection.

The health center server will look into the time *index* that has been received from the patient, and if it is available, it will **reserve** the time index for the patient and send back the corresponding doctor's port number, over the same TCP connection. Before the response is sent to the patient, the heath center server will mark this time slot as **reserved** in order to prevent it from being reserved by the other patient.

If the requested appointment time index is already reserved (i.e., not available), the health center server will respond with a message "**notavailable**" and upon receiving this message the patient will exit.

Shangxing Wang 11/3/2014 10:03 PM
Deleted: connection

A A 10/29/2014 9:43 PM
Deleted: *Note: we assume that there are not going to be concurrent requests from both patients that need complicated transaction processing.*

Phase 3: (Appointment Confirmation)

In this phase, each patient contacts his designated doctor through a UDP socket by using the doctor's UDP port number that was obtained from the health center server in phase 2 in order to request information about the cost of the visit. After the doctor receives the request from the patient containing its insurance name, it will send a message containing the estimated price for the visit through UDP socket.

More Detailed Explanations:

Phase1:

In this phase, the heath center server creates a TCP socket where it listens for connections from the patients (see Table 1 for the static TCP port number to be assigned to the health center server). Patients now can connect to the health center server using the health center's TCP static port number.

When each patient first loads, it will read the content of a file containing its username and password. The filename for patient 1 will be "**patient1.txt**" and the filename for patient 2 will be "**patient2.txt**". The contents of each file will be:

patient1.txt:

patient1 password111

patient2.txt

patient2 password222

After the patient has loaded the contents of its own file and set up the TCP connection, the patient immediately sends a request to the TCP port of the health center server, containing its username and password that were read from its credentials file. The message sent through the TCP connection should have the following format: "**authenticate username password**", where "**authenticate**" is a hint to the server that what follows is a username and password.

For example, patient 1 will send the following message to the health center server over TCP:

"authenticate patient1 password111" (without quotes)

The health center server when it first boots, it reads the contents of the user database contained in the file "**users.txt**" and stores the information in its memory. The format of the "users.txt" file will be the same as the individual patient credential files, but now it will contain the information for both patients. In our case it will be:

patient1 password111

patient2 password222

The health center server is expected to load the user database before accepting any connections from the patients. For this, never run a patient before the health center server has been started.

After the patient has sent its username and password to the health center server, the health center server will match the provided credentials against the records in the "users.txt" file. If there is a match for both username and password, then the health center server will respond to the patient with the message "**success**" indicating that the provided credentials are valid. If the provided credentials do not have a matching record in the file "users.txt", the health center server will send back the message "**failure**" to the patient. Upon reception of the "**failure**" message from the health center server, the patient will terminate the connection and exit.

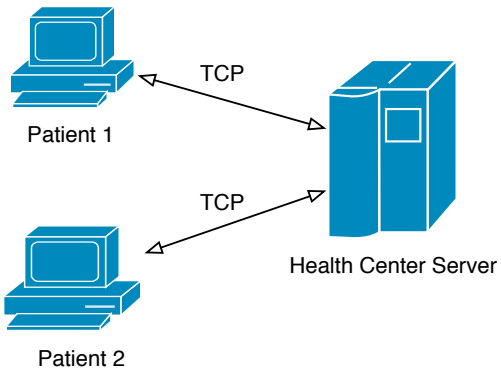

Figure1: Phase 1, Authentication of patients with the health center server.

Phase: 2

Stage1: (asking for availabilities and communicating preferences)

First of all, when the heath center main server program first starts, it will read the contents of the file **availabilities.txt** and store its contents in some data structure in the memory.

In this phase, patients use the same TCP port that has been setup on Phase 1 to schedule an appointment. First, a patient asks the server to send a list of available appointment slots. To ask for a slot, the patient will send the string **"available"** (without quotes) to the heath center main server. Upon receiving the request **"available"** (without quotes), the health center server will send the list of available (at that moment) appointments over the same TCP connection. The information sent in this step **contains only** the information from the 3 first columns of each line, e.g., **"**`2 Mon 03pm`**"** (without quotes). Therefore, the last two columns must not be included.

*The list sent to different patients may be different if one patient **completes** the reservation process before the other one.*

Upon receiving the list of available slots, the patient program should print them in the screen. Then the patient (you) will enter the preferred appointment time index from the keyboard by typing the respective time index number. The patient program will take that information and check whether it matches with any one of the time indices displayed. If not, the patient will be prompted to re-enter a correct time index. If it is a valid choice, the patient program will send back the **selected time index** to the health center server with the hint "**selection**" followed by the time index number. One example of reply message is **"selection 2"** (without quotes).

Shangxing Wang 11/3/2014 10:02 PM
Deleted: connection

Stage2: (Booking Appointment and receiving doctor's info)

Once the health center server receives the message from a patient with the selected timeslot, the health center server will go through the available slots information to check whether the requested slot is free. If yes, the health center server will mark that slot to be **"reserved"** in its data structure in the memory. The health center server will not have to update the file availabilities.txt, since it is only used once at the beginning when the server boots. After the slot is reserved the health center server will check the doctor's details associated with it, which is nothing but the information in the column, 4 and 5 of "**availabilities.txt"**. Next, the health center server will reply back to the patient with the doctor's name and port number.

For example, assume that time index "2" is chosen. The health center server has the following information from the file availabilities.txt:

"`2 Mon 03pm doc2 42000`**"**

So **'doc2'** is associated with that slot and the port number for **'doc2'** is 42000. So the health center server will reply "**doc2 42000"** to the respective patient (without quotes). Once the patient receives the port number, the TCP connection **must be closed**.

If the time slot requested by the patient is not available, the health center server will reply **"notavailable"** (without quotes)**.** When the patient receives **"notavailable"** (without quotes) it will realize that the requested slot is not available. Then, the patient will **close the TCP connection** and stop the process **immediately**.

Figure 2 illustrates the entire communication model of Phase 2.

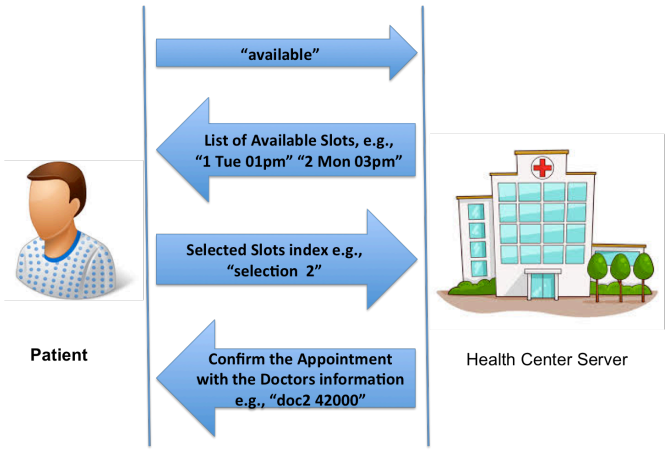

Figure 2: Communication Steps in Phase 2

Phase 3:

In this phase, each doctor will be already listening on its UDP port (as mentioned in the Table 1) for requests from the patients. In other words, the doctors will be already listening for incoming requests while phase 1 and 2 are in progress. Then, each patient should establish a UDP socket to communicate with its designated doctor using the UDP port number of the doctor that he got from the health center server in phase 2. During this phase, each patient will be assigned with a dynamic UDP port number that will be used for **BIDIRECTIONAL** communication with the doctor (only 1 UDP link should be used for that). Each patient sends an outgoing message containing its insurance name, which it has received from its insurance file. For example, if patient 1 is assigned to doctor 1, then it should send to doctor 1's UDP port number the following message (this is just an example, the insurance name might be different):

"**insurance1**" (without quotes)

Then the doctor will check his copay policy file **doc1.txt** and send back a string containing the estimated price of the appointment. The file doc1.txt can be loaded once when the doctor program boots. Here, we assume that one of the three valid insurance types will be requested and nothing else. After the doctor has found the amount of the requested insurance copay, it will respond back to the patient with the cost of the visit using the same UDP socket (for achieving this bidirectional communication, more hints will be provided in the discussion session). The format of the response will be just a number (it is assumed to be dollars), as follows: "30" (without quotes).

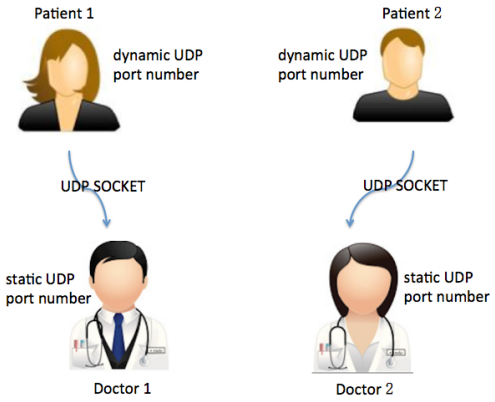

Figure 3: Illustration of Phase 3

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

Process	Dynamic Ports	Static Ports
Health Center Server	N/A	1 TCP, 21000+xxx (last three digits of your USC ID) (Phase 1)
Patient 1	1 TCP (Phase 1) 1 UDP (Phase 3)	N/A
Patient 2	1 TCP (Phase 1) 1 UDP (Phase 3)	N/A
Doctor 1	N/A	1 UDP, 41000 + xxx (last three digits of your USC ID) (Phase 3)
Doctor 2	N/A	1 UDP, 42000 + xxx (last three digits of your USC ID) (Phase 3)

NOTE: For example, if the last 3 digits of your USC ID are "319", you should use the port: 21000+319 = 21319 for the health center server.

ON SCREEN MESSAGES:

Table 2. Health Center Server on screen messages

Event	On Screen Message
Upon startup of phase 1	Phase 1: The Health Center Server has port number _____ and IP address _____.
Upon receiving the authentication request information from each of the patient users	Phase 1: The Health Center Server has received request from a patient with username _____ and password _____. (Note: you should print this twice, one for each patient)
After the authentication response has been sent to the patient	Phase 1: The Health Center Server sends the response ____ to patient with username _____.
Upon receiving the request from patients for available time slots	Phase 2: The Health Center Server, receives a request for available time slots from patients with port number _____

	and IP address _____. (Note: you should print this twice, one for each patient)
Upon sending the available time slots to each patient	Phase 2: The Health Center Server sends available time slots to patient with username _____. (Note: you should print this twice, one for each patient)
After the Health Center Server receives appointment preference from each patient	Phase 2: The Health Center Server receives a request for appointment _____ from patient with port number _____ and username _____. (Note: you should print this twice, one for each patient)
After checking the availabilities of the selected time slot by each patient	Phase 2: The Health Center Server confirms/rejects the following appointment _____ to patient with username _____. (Note: you should print this twice, one for each patient)

A A 10/29/2014 9:49 PM
Deleted: sends

A A 10/29/2014 9:49 PM
Deleted: s

Table 3. Patient 1 on screen messages

Event	**On Screen Message**
After TCP socket is created	Phase 1: Patient 1 has TCP port number _____ and IP address _____.
When the authentication request is sent to the health center server	Phase 1: Authentication request from Patient 1 with username _____ and password _____ has been sent to the Health Center Server.
Authentication Result	Phase 1: Patient 1 authentication result: _____.
End of Phase 1	Phase 1: End of Phase 1 for Patient1.
Upon receiving of list of appointments	Phase 2: The following appointments are available for Patient 1: ___ Please enter the preferred appointment index and press enter: ___
Upon receiving of a successful answer from the Health Center Server	Phase 2: The requested appointment is available and reserved to Patient1. The assigned doctor port number is _____.

Upon receiving of a negative answer from the Health Center Server	Phase 2: The requested appointment from Patient 1 is not available. Exiting…
Upon creating the UDP socket of phase 3	Phase 3: Patient 1 has a dynamic UDP port number _____ and IP address _____.
When the patient request is sent to the doctor	Phase 3: The cost estimation request from Patient 1 with insurance plan _____ has been sent to the doctor with port number _____ and IP address _____.
When the patient receives the estimated price of the appointment	Phase 3: Patient 1 receives _____$ estimation cost from doctor with port number _____ and name _____.
End of phase 3	Phase 3: End of Phase 3 for Patient 1.

Table 4. Patient 2 on screen messages

Event	**On Screen Message**
After TCP socket is created	Phase 1: Patient 2 has TCP port number _____ and IP address _____.
When the authentication request is sent to the health center server	Phase 1: Authentication request from Patient 2 with username _____ and password _____ has been sent to the Health Center Server.
Authentication Result	Phase 1: Patient 2 authentication result: _____.
End of Phase 1	Phase 1: End of Phase 1 for Patient 2.
Upon receiving of list of appointments	Phase 2: The following appointments are available for Patient 2: ___ Please enter the preferred appointment index and press enter: ___
Upon receiving of a successful answer from the Health Center Server	Phase 2: The requested appointment is available and reserved to Patient 2. The assigned doctor port number is _____.

Upon receiving of a negative answer from the Health Center Server	Phase 2: The requested appointment from Patient 2 is not available. Exiting…
Upon creating the UDP socket of phase 3	Phase 3: Patient 2 has a dynamic UDP port number _____ and IP address _____.
When the patient request is sent to the doctor	Phase 3: The cost estimation request from Patient 2 with insurance plan _____ has been sent to the doctor with port number _____ and IP address _____.
When the patient receives the estimated price of the appointment	Phase 3: Patient 2 receives _____$ estimation cost from doctor with port number _____ and name _____.
End of phase 3	Phase 3: End of Phase 3 for Patient 2.

Table 5. Doctor 1 on-screen messages

Event	**On Screen Message**
Upon creation of UDP socket	Phase 3: Doctor 1 has a static UDP port _____ and IP address _____.
Upon receiving the request from the patient	Phase 3: Doctor 1 receives the request from the patient with port number _____ and the insurance plan _____. (Note: You should print it for each request you get from each patient.)
Upon transferring the estimated price of the first appointment to the patient (i.e. sending the string containing the estimated price)	Phase 3: Doctor 1 has sent estimated price _____$ to patient with port number _____. (Note: You should print it for each request you get from each patient.)

Shangxing Wang 11/3/2014 10:00 PM
Deleted: name _____ with

A A 10/29/2014 9:50 PM
Deleted: named _____

Table 6. Doctor 2 on-screen messages

Event	On Screen Message
Upon creation of UDP socket	Phase 3: Doctor 2 has a static UDP port _____ and IP address _____.
Upon receiving the request from the patient	Phase 3: Doctor 2 receives the request from the patient with port number _____ and the insurance plan _____. (Note: You should print it for each request you get from each patient.)
Upon transferring the estimated price of the first appointment to the patient (i.e. sending the string containing the estimated price)	Phase 3: Doctor 2 has sent estimated price _____$ to patient with port number _____. (Note: You should print it for each request you get from each patient.)

Shangxing Wang 11/3/2014 10:00 PM
Deleted: name _____ with

A A 10/29/2014 9:50 PM
Deleted: named _____

Assumptions:

1. It is recommended to start the processes in this order: healthcenterserver, doctor1, doctor2, patient1, and patient2.

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 (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 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. 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 except for choosing the timing slots in phase 2.

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 http://itservices.usc.edu/software/ 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 8) and download the latest xwin.
    d. Install it on your computer.
    e. Then check the following webpage:
    http://itservices.usc.edu/unix/xservers/xwin32/ 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 http://itservices.usc.edu/software/ 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 20 out of 100.

8. If your code compiles and performs all tasks up to the end of 2 phases correctly and error-free, and your README file conforms to the requirements mentioned before, you will receive 70 out of 100.

9. If your code compiles and performs all tasks of 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 of 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:

Kill -9 processnumber

5. 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.

6. 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.