EE450 Socket Programming Project, Spring 2015

Due Date: Sunday April 12th, 2015 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 a distributed hash table or database system. A set of (key, value) pairs will be distributed among three servers. The clients will contact a designated server with a key in order to retrieve its corresponding value. The designated server will directly reply with the value if it is stored in its memory. Otherwise, it will contact other servers in a manner similar to recursive DNS. Once the designated server will retrieve the value, it stores a copy of the pair (key, value) in its local memory and will send back the result to the client.

The servers communicate with each other using TCP sockets. The clients communicate with the designated server using Bidirectional UDP sockets.

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. **server1.txt (This is for the designated server):** This input file contains the key and value mapping of the first server. It will contain exactly four rows with exactly the following contents (during grading, we will change both the key and values, but their formats stay the same: each key has exactly 5 characters, and each value has exactly 7 characters):

   key01 value01

key02 value02

key03 value03

key04 value04

2. **server2.txt:** This input file contains the key and value mapping of the second server. It will contain exactly four rows with exactly the following contents (during grading, we will change both the key and values, but their formats stay the same: each key has exactly 5 characters, and each value has exactly 7 characters):

key05 value05

key06 value06

key07 value07

key08 value08

3. **server3.txt:** This input file contains the key and value mapping of the third server. It will contain exactly four rows with exactly the following contents (during grading, we will change both the keys and values, but their formats stay the same: each key has exactly 5 characters, and each value has exactly 7 characters):

key09 value09

key10 value10

key11 value11

key12 value12

4. **client1.txt**: This input file contains information about the mapping between search term and key value for client 1. It will contain the mapping between input search terms to a key value. It will contain exactly 12 rows with exactly the following contents (during grading, we will change only the key values and search values stay the same as below.

The key format remains the same: each key has exactly 5 characters.):

USC key01

UCLA key02

UCB key03

SFU key04

UCSD key05

UIUC key06

UCI key07

UCD key08

UMD key09

MIT key10

MSU key11

WUSL key12

5. **client2.txt**: This input file contains information about the mapping between search term and key value for client 2. It will contain the mapping of each input search terms to a key value. It will contain exactly 12 rows with exactly the following contents (during grading, we will change only the key values and search values stay the same as below. The key format remains the same: each key has exactly 5 characters):

   USC key01

   UCLA key02

   UCB key03

   SFU key04

   UCSD key05

   UIUC key06

UCI key07

UCD key08

UMD key09

MIT key10

MSU key11

WUSL key12

.

Source Code Files

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

1. Server 1, 2 and 3: You must use one of these names for this piece of code: **dhtserver#.c** or **dhtserver#.cc** or **dhtserver#.cpp** (all small letters). Also you must call the corresponding header file (if you have one; it is not mandatory) **dhtserver#.h** (all small letters). The "#" character must be replaced by the server number (i.e. 1 or 2 or 3), depending on which server it corresponds to. **Note that the server 1 is the designated server**. In case you are using one executable for all three servers (i.e. if you choose to make a "fork" based implementation), you should use the same naming convention without adding the server's number at the end of the file name (e.g. **dhtserver**.c). *In order to create three servers in your system using one executable, you can use the fork() function inside your server's code to create 3 child processes.* You must follow this naming convention! This piece of code basically handles the distributed database server functionalities.

2. Client 1 and 2**:** The name of this piece of code must be **client#.c** or **client#.cc** or **client#.cpp** (all small letters) and the header file (if you have one; it is not mandatory) must be called **client#.h** (all small letters). The "#" character must be replaced by the **client** number (i.e. 1 or 2), depending on which client it corresponds to. *You could also use multithreading to implement the client.*

Detailed Description:

Phase 1: Initialization:

This process imitates the Recursive DNS system to implement a distributed file system. Among three servers, Server 1 is the the designated server, whom clients should contact first. At the very beginning, all three server programs should load the respective key-value pairs from their input files **i.e., server#.txt.** They should store this information in the memory using some kind of structure such as array, matrix, link-list or etc. *You are free to choose any data structure.*

After loading files, Server 2 and Server 3 should create a TCP server socket and start listening on it to wait for request. Server 1, i.e., the designated server, opens up a UDP server socket to wait for request from clients*. For the port numbers of these servers, please refer to table 1.*

When two client programs boot up, each should read from its respective input files, **i.e, client#.txt,** and save the search term to key mapping information in its memory. The two client programs should run after servers are running. Note we only have exactly two client programs in the project.

Phase 2: Search term mapping at Client:

After the initialization stage is finished for both the servers and clients, a client process will ask the user (i.e. you) to enter a search term from the keyboard by typing the search term name. The search term has to be one of 12 search terms a client has (i.e., entries of the first column of cleint#.txt.) For example, one input could be "USC" (without quotation).

Some related information should be printed on the terminal regarding the input information entered by a user (i.e., you), for example,

Please Enter Your Search (USC, UCLA etc.): USC

After the client program takes the input, it will check the search term to key mapping that is already stored in the memory from Phase 1 and get the respective key value. *For example, if the search term is USC, the respective key will be key01.*

Phase 3: Client communication with Servers:

Once each client program resolves the "key" corresponding to the "search term" entered, it starts to communicate with the designated server i.e, Server 1 to retrieve the corresponding "value." We assume that the client's contact the designated server sequentially. Client 1 contacts the Server 1 (the designated server) first to retrieve a "value." The Client 2 will contact

Server 1 to retrieve a "value" only after the Client 1 finishes communication with the Server 1**. Therefore, there is no need to deal with concurrency, and we always assume Client 1 contact Server 1 first, then Client 2.**

First, Client 1 program opens up a UDP client socket and sends a request to Server 1 with the required "*key*." The request message should contain the keyword **"GET" (without quotation).** For example, in the case of "USC", the request message to be sent to Server 1 will be

"GET key01" (without quotation)

Upon receiving the request and *key* from the client, Server 1(the designated server) checks the *key to valu*e mapping stored in its memory. You need to deal with three different cases as follows.

Case I:

If the key-value mapping for the required key (e.g., "key01" (without the quotation) for the "USC" case) is already in Server 1 memory, the server will directly reply to the client with the *"value"* over a UDP connection and the UDP connection in this context has to be **bidirectional UDP**. The reply message should contain the keyword **"POST" (without quotation)**. For example, for the above-mentioned request, the reply message will be

"POST value01" (without quotation)

Case II:

If the key to value mapping for the required key is **NOT** found in the local mapping table, Server 1 will open a TCP connection with Server 2 and forward the request with the required "key" to Server 2. The request message should be in the same format as mentioned above. For example, in the case of "USC", the request message to be forwarded to Server 2 will be

"GET key01" (without quotation)

Upon receiving the request, Server 2 will check its local mapping table to find the required key-value pair. If there **EXISTS** a mapping for the required key, it will send back the value to Server 1, over the **same** TCP connection. The reply message format is the same as before. For example, for the above-mentioned request, the reply message will be

"POST value01" (without quotation)

Upon receiving the value from Server 2, Server 1 will close the TCP connection with Server 2, and reply to the client with the value over UDP connection with the same reply message format as before. For example, for the request with "key01", the reply message will be

"POST value01" (without quotation)

In addition, Server 1 will also store a copy of this mapping (required key-value pair) in its own table, so that if next time a client ask for same required key-value mapping, it can reply directly. Again, the UDP connection in this context has to be **bidirectional UDP**.

Case III:

Now, if the entry is also **NOT** in Server 2, Server 2 will open a TCP connection with Server 3, **while keeping the TCP connection with Server 1 open**. Then server 2 will forward the request with the required key to server 3 over the new TCP connection. The request message should be in the same format as mentioned above. For example, in the case of "USC", the request message to be forwarded to Server 3 will be

"GET key01" (without quotation)

Upon receiving the request, Server 3 will check its local mapping table to find the required key-value pair. If there **EXISTS** an entry with the key value, it will reply back the value to Server 2, over the **same** TCP connection. The reply message format is the same as before. For example, for the above-mentioned request, the reply message will be

"POST value01" (without quotation)

Upon receiving the value from server 3, server 2 will close the TCP connection with server 3 and reply to the server 1, over the existing TCP connection with server 1, with required key value. The reply message format is the same as before. For example, for the above-mentioned request, the reply message will be

"POST value01" (without quotation)

Server 2 also stores a copy of this mapping it its table, so that if server 1 asks for the same key-value mapping again, it can reply directly.

Upon receiving the value from Server 2, Server 1 will close the TCP connection and reply to the client with the value over UDP connection. Again, the reply message format is the same as before. For example, for the above-mentioned request, the reply message will be

"POST value01" (without quotation)

Server 1 will also store a copy of this mapping it its table, so that if next client ask for same key-value mapping, it can reply directly. Again, the UDP connection between the client and the designated server is **bidirectional**.

After receiving the value from the server, the client should print out the value as follows:

"The requested value is _____"(without quotation)

Then the client program terminates.

Once Client 1 terminates, Client 2 can start contacting the designated server and retrieve a value. The steps for Client 2 will be same as Client 1. The only difference will be that some of servers' tables have been modified from first client's search and retrieval. There should be NO open UDP and TCP connections after both clients finish their retrieval.

One thing need to be noted is that when a client uses a key to retrieve its associated value, the key-value pair can always be found in the mapping table of at least one server. Therefore in the project, you do not need to consider the edge case where a key-value pair cannot be found in any of the three servers.

Figure 1 provides an illustration of how client 1 can retrieve a value related to a search term. In this figure, we assume Server 1 and Server 2 doesn't have the key-value mapping, while Server 3 does.

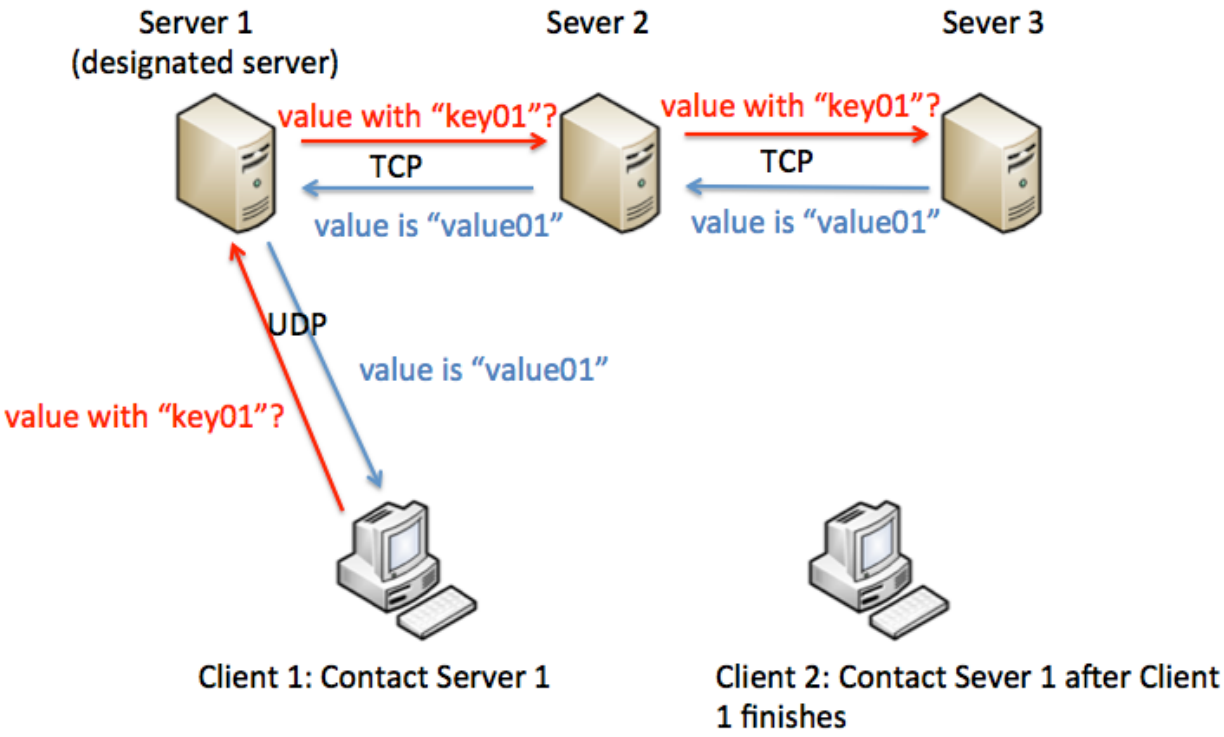

Fig. 1 Illustration

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

Process	Dynamic Ports	Static Ports
Server 1	2 TCP (For connection with server 2 in phase 3, case II; Each TCP corresponds to one client)	1 UDP, 21000+xxx (last three digits of your USC ID) (For communication with a client)
Server2	2 TCP (For connection with server 3 in phase 3, case III; Each TCP corresponds to one client)	1 TCP, 22000+xxx (last three digits of your USC ID) (For communication with Server 1)
Server 3	N/A	1 TCP, 23000+xxx (last three digits of your USC ID)

		(For communication with Server 2)
Client 1	1 UDP (For connection with server 1 in phase 3)	N/A
Client 2	1 UDP (For connection with server 1 in phase 3)	N/A

NOTE: For example, if the last 3 digits of your USC ID are "319", you should use the port: 21000+319 = 21319 for the server 1.

ON SCREEN MESSAGES:

Table 2. Server1 on screen messages

Event	**On Screen Message**
Booting Up	The Server 1 has UDP port number _____ and IP address _____.
Upon receiving the first request from Client 1	The Server 1 has received a request with key _____ from client 1 with port number _____ and IP address _____.
If the server have the entry in its table	The Server 1 sends the reply ____ to Client 1 with port number _____ and IP address ____. (Reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))
If the server doesn't have the entry in its table	The Server 1 sends the request ____ to the Server 2. The TCP port number is_____ and the IP address is ____. (Request message should follow the format as mentioned before, e.g., "GET key01" (without quotation))
Upon receiving then reply from Server 2.	The Server 1 received the value _____ from the Server 2 with port number _____ and IP address_____. The Server 1 closed the TCP connection with the Server 2.

After sending the reply to the Client 1	The Server 1, sent reply _____ to Client 1 with port number _____ and IP address_____. (Reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))
Upon receiving the second request from Client 2	The Server 1 has received a request with key _____ from Client 2 with port number _____ and IP address _____.
If the server have the entry in its table	The Server 1 sends the reply ____ to Client 2 with port number _____ and IP address ____. (Reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))
If the server doesn't have the entry in its table	The Server 1 sends the request ____ to the Server 2. The TCP port number is_____ and the IP address is ____. (Request message should follow the format as mentioned before, e.g., "GET key01" (without quotation))
Upon receiving then reply from Server 2.	The Server 1 received the value _____ from the Server 2 with port number _____ and IP address_____. The Server 1 closed the TCP connection with the Server 2.
After sending the reply to Client 2	The Server 1 sent reply _____ to the Client 2 with port number _____ and IP address_____. (reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))

Table 3. Server 2 on screen messages

Event	**On Screen Message**
Booting Up	The Server 2 has TCP port number _____ and IP address _____.
Upon receiving the first request from Server 1	The Server 2 has received a request with key _____ from the Server 1 with port number _____ and IP address _____.
If the server have the entry in its table	The Server 2 sends the reply ____ to the Server 1 with port number _____ and IP address ____. (Reply message should follow the format as mentioned

	before, e.g., “POST value01” (without quotation))
If the server doesn’t have the entry in its table	The Server 2 sends the request ____ to the Server 3. The TCP port number is_____ and the IP address is ____. (Request message should follow the format as mentioned before, e.g., “GET key01” (without quotation))
Upon receiving the reply from Server 3	The Server 2 received the value _____ from the Server 3 with port number _____ and IP address_____. The Server 2 closed the TCP connection with the Server 3.
After sending the reply to the Server 1	The Server 2, sent reply _____ to the Server 1 with port number _____ and IP address_____. (Reply message should follow the format as mentioned before, e.g., “POST value01” (without quotation))
Upon receiving the second request from Server 1	The Server 2 has received a request with key _____ from the Server 1 with port number _____ and IP address _____.
If the server have the entry in its table	The Server 2 sends the reply ____ to the Server 1 with port number _____ and IP address ____. (Reply message should follow the format as mentioned before, e.g., “POST value01” (without quotation))
If the server doesn’t have the entry in its table	The Server 2 sends the request ____ to the Server 3. The TCP port number is_____ and the IP address is ____. (Request message should follow the format as mentioned before, e.g., “GET key01” (without quotation))
Upon receiving then reply from Server 3.	The Server 2 received the value _____ from the Server 3 with port number _____ and IP address_____. The Server 2 closed the TCP connection with the Server 3.
After sending the reply to the Server 1	The Server 2, sent reply _____ to the Server 1 with port number _____ and IP address_____. (Reply message should follow the format as mentioned before, e.g., “POST value01” (without quotation))

Table 4. Server 3 on screen messages

Event	On Screen Message
Booting Up	The Server 3 has TCP port number _____ and IP address _____.
Upon receiving the first request from Server 2	The Server 3 has received a request with key _____ from the Server 2 with port number _____ and IP address _____.
After sending the reply to the Server 2	The Server 3 sends the reply ____ to the Server 2 with port number _____ and IP address ____. (Reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))
Upon receiving the second request from Server 2	The Server 3 has received a request with key _____ from the Server 2 with port number _____ and IP address _____.
After sending the reply to the Server 2	The Server 3, sent reply _____ to the Server 2 with port number _____ and IP address_____. (Reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))

Table 5. Client 1 on-screen messages

Event	On Screen Message
Booting Up	Please Enter Your Search (USC, UCLA etc.):
Upon receiving the a valid keyword	The Client 1 has received a request with search word _____, which maps to key _____.
After sending the request to the Server 1	The Client 1 sends the request ____ to the Server 1 with port number _____ and IP address ____. The Client1's port number is _____ and the IP address is _____. (Request message should follow the format as

	mentioned before, e.g., "GET key01" (without quotation))
After receiving the reply from the Server 1	The Client 1 received the value ____ from the Server 1 with port number _____ and IP address ____. The Client1's port number is _____ and IP address is _____. (Reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))

Table 6. Client 2 on-screen messages

Event	**On Screen Message**
Booting Up	Please Enter Your Search (USC, UCLA etc.):
Upon receiving the a valid keyword	The Client 2 has received a request with search word _____, which maps to key _____.
After sending the request to the Server 1	The Client 2 sends the request ____ to the Server 1 with port number _____ and IP address ____. The Client2's port number is _____ and IP address is _____. (Request message should follow the format as mentioned before, e.g., "GET key01" (without quotation))
After receiving the reply from the Server 1	The Client 2 received the value ____ from the Server 1 with port number _____ and IP address ____. The Client2's port number is _____ and IP address is _____.

	(Reply message should follow the format as mentioned before, e.g., "POST value01" (without quotation))

Assumptions:

1. It is recommended to start the processes in this order: server1, server2, server3, client1, and client2.

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 the input search term 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. a sample 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 and your README file conforms to the requirements mentioned before, you will receive 10 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 20 out of 100.

9. If your code compiles and can perform first two phases correctly and error-free, plus it works well ONLY for Case I in Phase 3 (i.e., Sever 1 has the required key-value pairs in its memory. It does not need to ask other servers for help) for both clients, and your README file conforms to the requirements mentioned before, you will receive 40 out of 100.

10. If your code compiles and can perform first two phases correctly and error-free, plus it works well for CASE I and CASE II in Phase 3 (i.e., Either Sever 1 or Server 2 have the required key-value pairs in its memory. They do not need to ask Server 3 for help) for both clients, and your README file conforms to the requirements mentioned before, you will receive 70 out of 100.

11. If your code compiles and performs all tasks of all 3 phases correctly and error-free (i.e., in Phase 3, your code works well for all three cases) for both clients, and your README file conforms to the requirements mentioned before, you will receive 100 out of 100.

12. 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.)

13. If your code does not correctly assign the TCP or UDP port numbers dynamically (in any phase), you will lose 20 points.

14. You will lose 5 points for each error or a task that is not done correctly.

15. The minimum grade for an on-time submitted project is 10 out of 100.

16. 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.

17. 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.

18. 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.

19. If you participate in discussions (answer someone's question) in **Piazza**, starting from the day the project is assigned, you will receive up to 5 bonus points. In order to get these points, you **shouldn't** post anonymous to the instructors.

20. 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.

21. 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.