EE450 Socket Programming Project, Fall 2015

Due Date: Thursday Nov 19th, 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 system similar to Dijkstra to build a map of the network topology. A set of neighbor information will be distributed among four servers. There will also be a client who is going to initiate the process and also do some computation to give the final topology output. The client and servers are going to communicate with each other to get the neighbor informations from the others and combine all this information together to build the entire map of the network. The original map is a connected graph, which may contain cycles (or loops). So later, based on the map, the client will calculate the minimum spanning tree (MST) of the network and print it out on the terminal.

The servers will communicate with the client using TCP sockets. The client will communicate with the servers using 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. **serverA.txt:** This input file contains the neighbors' names and corresponding link costs from the server A to each neighbor. It will contain several rows, where each row has the neighbor's name and the corresponding link cost (during grading, we will change the contents, but the format will remain the same):

serverB 20
serverC 10

2. **serverB.txt:** This input file contains the neighbors' names and corresponding link costs from the server B to each neighbor. It will contain several rows, where each row has the neighbor's name and the corresponding link cost (during grading, we will change the contents, but the format will remain the same):

   serverA 20
   serverC 30
   serverD 15

3. **serverC.txt:** This input file contains the neighbors' names and corresponding link costs from the server C to each neighbor. It will contain several rows, where each row has the neighbor's name and the corresponding link cost (during grading, we will change the contents, but the format will remain the same):

   serverA 10
   serverB 30

4. **serverD.txt:** This input file contains the neighbors' names and corresponding link costs from the server D to each neighbor. It will contain several rows, where each row has the neighbor's name and the corresponding link cost (during grading, we will change the contents, but the format will remain the same):

   serverB 15

Source Code Files

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

1. Server A, B, C and D: You must use one of these names for this piece of code: **server#.c** or **server#.cc** or **server#.cpp** (all small letters). Also you must call the corresponding header file (if you have one; it is not mandatory) **server#.h** (all small letters). The "**#**" character must be replaced by the server identifier (i.e. A or B or C or D), depending on which server it corresponds to. In case you are using

one executable for all four servers (i.e. if you choose to make a "fork" based implementation), you should use the same naming convention without adding the server's ID at the end of the file name (e.g. **server**.c). *In order to create four servers in your system using one executable, you can use the fork() function inside your server's code to create 4 child processes.* You must follow this naming convention! This piece of code basically handles the server functionalities.

2. Client **:** The name of this piece of code must be **client.c** or **client.cc** or **client.cpp** (all small letters) and the header file (if you have one; it is not mandatory) must be called **client.h** (all small letters). .

More Detailed Explanations:

Phase1: (20 points)

In this phase, each server creates a UDP server socket where it listens for incoming messages from the client (see Table 1 for the static port numbers to be assigned to the servers). The client will also create a TCP server socket to listen for incoming connections from servers (see Table 1 for the static port number to be assigned to the client).

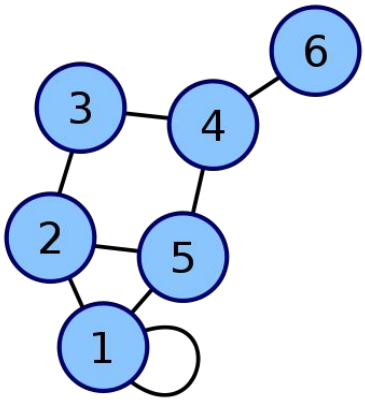
When each server first loads, it will read the content of a file containing its neighbor's information. The filename for Server A will be "**serverA.txt**" and the filename for Server B will be "**serverB.txt**", so on. Then it saves the information in some kind of data structure such as array or matrix or link-list etc. The sample of the file content are provided in the **Input Files** section. When the client first boots up, it **DOESN'T** need to read from any file.

Phase 2: (60 points)

In Phase 1, you read the neighbor information of each server. In other words, each server now understands which other servers it can connect to and the 'cost' associated with each link connection. The goal of Phase 2 is for each node to obtain the entire topology of the network, using the connectivity information of each node. The principle is similar to the link state advertisement used in protocols such as OSPF.

The topology of the network is represented as an undirected graph. There are various ways in which you may represent a graph while coding. For this project, we shall use an adjacency matrix. For a graph containing n nodes, an adjacency matrix is an n x n matrix consisting of 0's

and 1's that indicates if a node is connected to another. For example, consider the adjacency matrix of the 6 node graph given below.



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

You can find similar examples of adjacency matrices at the Wikipedia page.

For the output of phase 2, **each** server and the client should print the adjacency matrix of the network that they are part of. Since, they are all part of the same network they should ideally print the same adjacency matrix. Note that initially each server and client only knows the server/client(s) that they are connected to. Each server, now has to convey its connectivity information to all the other nodes and client. Use TCP or UDP connections (as applicable) to transmit connectivity information. At the end of phase 2 each node should have an adjacency matrix corresponding to its view of the network based on the information that it has received from the other nodes in the network. The adjacency matrix must have the following format:

Server	A	B	C	D
A	0	‘0’ or ‘1’	‘0’ or ‘1’	‘0’ or ‘1’
B	‘0’ or ‘1’	0	‘0’ or ‘1’	‘0’ or ‘1’
C	‘0’ or ‘1’	‘0’ or ‘1’	0	‘0’ or ‘1’
D	‘0’ or ‘1’	‘0’ or ‘1’	‘0’ or ‘1’	0

In our project, based on the example input files given above, the network topology is illustrated in figure 1. Since in this network topology, each link cost is also given. So to incorporate link cost information into the adjacency matrix as well, we can replace value ‘1’s as the corresponding link cost (Note, for the diagonal entries in the adjacency matrix, they still remain as ‘0’s). The corresponding adjacency matrix is as follows:

As an exercise, we suggest you write down the adjacency matrix for the illustrative network given below:

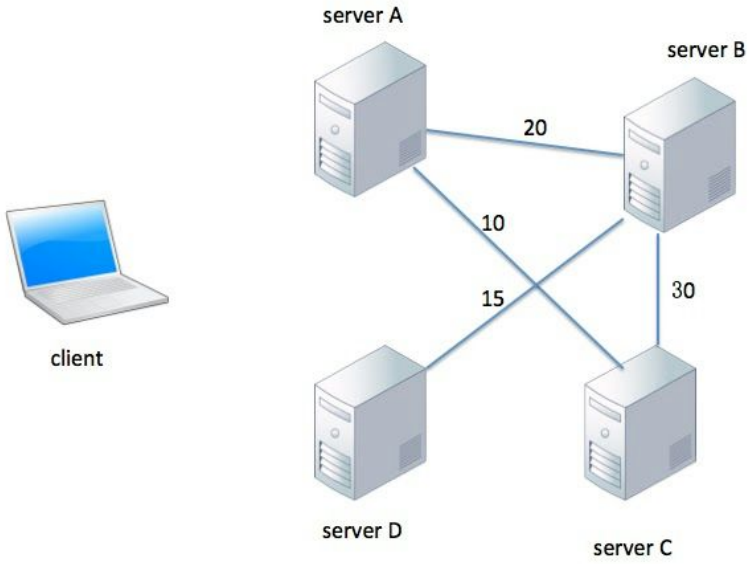


figure 1. Illustration of the network

There are multiple ways by which you can ensure that all servers agree on their view of the network. One way is for the servers to communicate among themselves, each server disseminating its view of the network till everyone settles down on the complete view. **For the purposes of this assignment, however, we shall follow a simpler strategy**. Each server shall communicate its view of the network (i.e, its connectivity information) to the client . Now that the client has the connectivity information of all the servers, it can construct their network topology. The client then broadcasts this topology to the servers. Specifically, the communication among the client and servers is structured as follows:

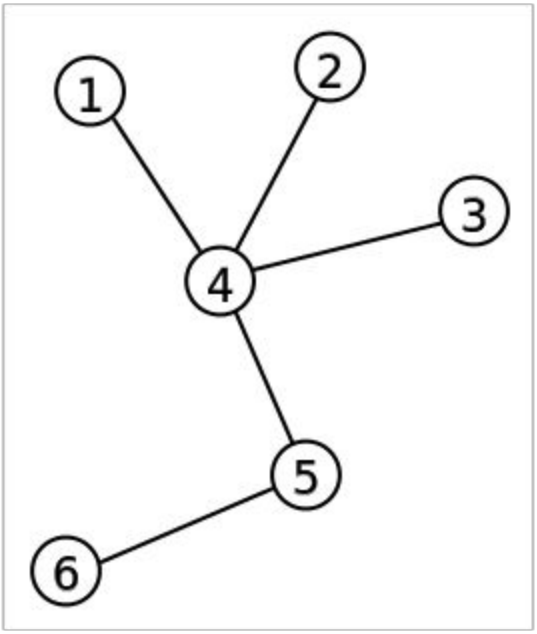
- The servers share their neighbor information to the client through TCP. Namely, the following connections need to be via TCP:
    - Server A to Client
    - Server B to Client
    - Server C to Client
    - Server D to Client
- The client 'broadcasts' the network topology to the servers though UDP. **While we use the term 'broadcast' here, for implementation purposes make the client open unicast UDP connections with each of the servers and transmit the same data individually.** Thus, we have the following UDP connections:
    - Client to Server A
    - Client to Server B
    - Client to Server C
    - Client to Server D

Phase 3: (20 points)

At the end of Phase 2, all client and servers have the entire topology of the network with link costs. In Phase 3, this topology is used to calculate a "tree" or a "minimum spanning tree" that contain all servers.

Tree:

A tree is an undirected graph in which any two nodes are connected by exactly one path. This means that there is no loops in a tree. An example of a tree is the following graph.

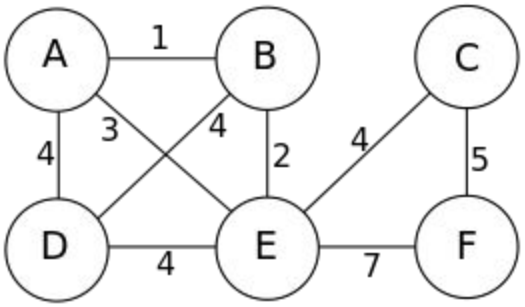


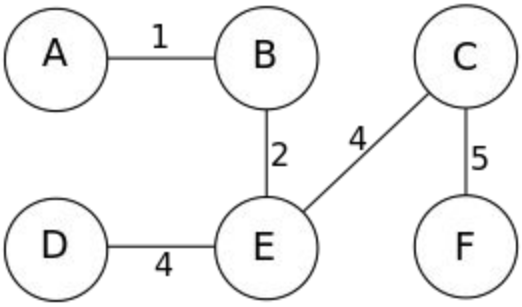
Source: Wikipedia|Tree

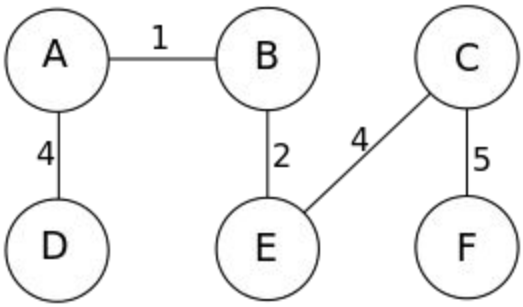
It is easy to see that from a connected graph, we can construct a tree from it **without using any link costs**. In fact, many trees might be constructed from a single graph.

Minimum spanning tree:

Given a connected graph containing nodes and links with link costs, **we say that the cost of a tree, constructed from a given graph, is the sum of all link costs of the tree.** For example, given a graph below, two trees have the same cost, which is 16.







Source: [Wikipedia|Minimum spanning tree]

A minimum spanning tree is a tree with the smallest cost. In fact, the trees in the above example are minimum spanning trees. (You can try to find other trees and their cost will be at least 16.)

Now, you have the knowledge of trees and minimum spanning trees. In Phase 3, the client constructs **a tree containing all servers**. Below is the grading criteria of this phase. The term "tree" below represents a tree that contains all servers.

- If the output is a tree, 10 points are obtained in this phase.
- If the output is a minimum spanning tree, 20 points are obtained in this phase.

Hint: A tree that contains all nodes in a given graph is called a “connected tree”. A connected tree can be verified by several methods. An efficient method is to run either depth-first search or breadth-first search algorithm and checks that an output set of nodes contains all nodes in the graph.

Hint: Do you need to verify that a tree is connected if it is a minimum spanning tree?

As a side note, the minimum spanning tree is used in a layer-2 switch network to avoid loops in the network where the calculation is done distributively. In this project, the process is simplified by letting the client calculate the minimum spanning tree.

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

Process	**Dynamic Ports**	**Static Ports**
Server A	1 TCP	1 UDP, 21000+xxx (last three digits of your USC ID)
Server B	1 TCP	1 UDP, 22000+xxx (last three digits of your USC ID)
Server C	1 TCP	1 UDP, 23000+xxx (last three digits of your USC ID)
Server D	1 TCP	1 UDP, 24000+xxx (last three digits of your USC ID)
Client	4 UDPs (one for each server)	1 TCP, 25000+xxx (last three digits of your USC ID)

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. **It is NOT going to be 21000319.**

ON SCREEN MESSAGES:
Table 2. ServerA on screen messages

Event	**On Screen Message**
Booting Up	The Server A has UDP port number _____ and IP address _____.
Upon Reading the file	The Server A has the following neighbor information: Neighbor------Cost

	_____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverB 20** **serverC 10**)
After sending its neighbor information to the Client	The Server A finishes sending its neighbor information to the Client with TCP port number ___ and IP address ___ (Client's TCP port number and IP address). For this connection with the Client, the Server A has TCP port number _____ and IP address _____.
After receiving the network topology from the Client	The server A has received the network topology from the Client with UDP port number ___ and IP address _____ (Client's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BC 30** **BD 15)** For this connection with Client, The Server A has UDP port number _____ and IP address _____.

Table 3. Server B on screen messages	
Event	**On Screen Message**
Booting Up	The Server B has UDP port number _____ and IP address _____.
Upon Reading the file	The Server B has the following neighbor information: Neighbor------Cost

	_____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverA 20** **serverC 30** **serverD 15**)
After sending its neighbor information to the Client	The Server B finishes sending its neighbor information to the Client with TCP port number ___ and IP address ___ (Client's TCP port number and IP address). For this connection with the Client, the Server B has TCP port number _____ and IP address _____.
After receiving the network topology from the Client	The server B has received the network topology from the Client with UDP port number ___ and IP address _____ (Client's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BC 30** **BD 15)** For this connection with Client, The Server B has UDP port number _____ and IP address _____.

Table 4. Server C on screen messages	
Event	**On Screen Message**
Booting Up	The Server C has UDP port number _____ and IP address _____.

Upon Reading the file	The Server C has the following neighbor information: Neighbor------Cost _____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverA 10** **serverB 30**)
After sending its neighbor information to the Client	The Server C finishes sending its neighbor information to the Client with TCP port number ___ and IP address ___ (Client's TCP port number and IP address). For this connection with the Client, the Server C has TCP port number _____ and IP address _____.
After receiving the network topology from the Client	The server C has received the network topology from the Client with UDP port number ___ and IP address _____ (Client's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BC 30** **BD 15)** For this connection with Client, The Server C has UDP port number _____ and IP address _____.

Table 5. Server D on-screen messages

Event	**On Screen Message**

Booting Up	The Server D has UDP port number _____ and IP address _____.
Upon Reading the file	The Server D has the following neighbor information: Neighbor------Cost _____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverB 15**)
After sending its neighbor information to the Client	The Server D finishes sending its neighbor information to the Client with TCP port number ___ and IP address ___ (Client's TCP port number and IP address). For this connection with the Client, the Server D has TCP port number _____ and IP address _____.
After receiving the network topology from the Client	The server D has received the network topology from the Client with UDP port number ___ and IP address _____ (Client's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BC 30** **BD 15)** For this connection with Client, The Server D has UDP port number _____ and IP address _____.

Table 6. Client on-screen messages

Event	**On Screen Message**

Booting Up	The Client has TCP port number _____ and IP address _____.
After receiving the neighbor info of the Server A	The Client receivers neighbor information from the Server A with TCP port number ___ and IP address ___ (The Server A's TCP port number and IP address). The Server A has the following neighbor information: Neighbor------Cost _____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverB 20** **serverC 10**) For this connection with Server A, The Client has TCP port number _____ and IP address _____.
After receiving the neighbor info of the Server B	The Client receivers neighbor information from the Server B with TCP port number ___ and IP address ___ (The Server B's TCP port number and IP address). The Server B has the following neighbor information: Neighbor------Cost _____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverA20** **serverC30** **serverD15)** For this connection with Server B, The Client has TCP port number _____ and IP address _____.
After receiving the neighbor info of the Server C	The Client receivers neighbor information from the Server C with TCP port number ___ and IP address ___ (The Server C's TCP port number and IP address).

	The Server C has the following neighbor information: Neighbor------Cost _____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverA10** **serverB30**) For this connection with Server C, The Client has TCP port number _____ and IP address _____.
After receiving the neighbor info of the Server D	The Client receivers neighbor information from the Server D with TCP port number ___ and IP address ___ (The Server D's TCP port number and IP address). The Server D has the following neighbor information: Neighbor------Cost _____ ____. (Repeat this line based on the number of entries. e.g. if you have three entries, you should print this line thrice. For the example scenario it will be. **Neighbor------Cost** **serverB15**) For this connection with Server D, The Client has TCP port number _____ and IP address _____.
After sending the network topology to the Server A	The Client has sent the network topology to the network topology to the Server A with UDP port number ___ and IP address _____ (Server A's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10**

	BC 30 **BD 15)** For this connection with Server A, The Client has UDP port number _____ and IP address _____.
After sending the network topology to the Server B	The Client has sent the network topology to the network topology to the Server B with UDP port number ___ and IP address _____ (Server B's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BC 30** **BD 15)** For this connection with Server B, The Client has UDP port number _____ and IP address _____.
After sending the network topology to the Server C	The Client has sent the network topology to the network topology to the Server C with UDP port number ___ and IP address _____ (Server C's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BC 30** **BD 15)** For this connection with Server C, The Client has UDP port number _____ and IP address _____.

After sending the network topology to the Server D	The Client has sent the network topology to the network topology to the Server D with UDP port number ___ and IP address _____ (Server D's UDP port number and IP address) as follows: Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BC 30** **BD 15)** For this connection with Server D, The Client has UDP port number _____ and IP address _____.
After a tree is calculated	The Client has calculated a tree. The tree cost is ____ (cost of the tree): Edge------Cost ____ ____ (Repeat this line based on the number of edges in the network topology. e.g. For the example scenario it will be. **Edge------Cost** **AB 20** **AC 10** **BD 15)**

Assumptions:

1. It is recommended to start the processes in this order: serverA, serverB, serverC, ServerD, and client.

2. If you need to have more code files than the ones that are mentioned here, please use meaningful names and all small letters and mention them all in your README file.

3. You are allowed to use blocks of code from Beej's socket programming tutorial (Beej's guide to network programming) in your project. However, you need to mark the copied part in your code.

4. When you run your code, if you get the message "port already in use" or "address already in use", please first check to see if you have a zombie process (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 80 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. We also encourage you to discuss homework and project problems on Piazza. We will give those who actively help others out by answering questions on Piazza up to 5 bonus points. (If you want to earn the extra credits, do remember to leave your names visible to instructors when answering questions on Piazza.)

18. 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.

19. 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.