EE 450
Spring 2011 Nazarian
Socket Programming Project

Assigned: Friday , February 11
Due: Friday , April 22 (by 11:59am - noon)
Maximum points: 100

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

Notes:

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

A. The Problem

In this project you will be simulating a query system for Car Rental Company. The system consists of the car rental branches, the central database system that keeps track of the available cars and the users that are interested to rent a car. The communication between the car rental branches, the central database and the users will be over TCP and UDP sockets in a network with client-server architecture. The project has 3 major phases: Submission of the available cars in the branches to the central database system, Query to the Database from the users, Connection to the appropriate car rental branches for the request of the price of the cars that the user is interested in. In Phase 1 and 3 all communications are through TCP sockets. However, in phase 2 all communications are over UDP sockets.

B. Code and Input files

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

1- Branch

You must create 3 concurrent Branches

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

ii. Or by running 3 instances of the Branch code. However in this case, you probably have 3 pieces of code for which you need to use one of these sets of names: (**Branch1.c, Branch2.c, Branch3.c**) or (**Branch1.cc, Branch2.cc, Branch3.cc)** or (**Branch1.cpp, Branch2.cpp and Branch3.cpp)**. Also you must call the corresponding header file (if any) **Branch.h** or **Branch1.h, Branch2.h** and **Branch3.h**. You must follow this naming convention. Make sure the first letter of the word 'Branch' is capital.

2- Database

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

3- User

You must create 2 concurrent Users

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

ii. Or by running 2 instances of the User code. However in this case, you probably have 2 pieces of code for which you need to use one of these sets of names: (**User1, User2.c**) or (**User1.cc, User2.cc)** or (**User1.cpp, User2.cpp)**. Also you must call the corresponding header file (if any) **User.h** or **User1.h** and **User2.h**. You must follow this naming convention. Make sure the first letter of the word 'User' is capital.

4- Input file: user1.txt

This is the file that contains the cars for which User1 is interested in learning the prices. The file consists of three lines and each line has the make and the model of a car that the user wants to know the price. Initially, the user queries the central database about the car rental branch in which each of these cars exists and then the user is responsible to ask the corresponding branches for the prices of the cars. Here is the format of a sample query file:

Honda Accord
Toyota Corolla
Kia Rio

5- Input file: user2.txt
This is the file that contains the cars for which User2 is interested in learning the prices. The file consists of three lines and each line has the make and the model of a car that the user wants to know the price. Initially, the user queries the central database about the car rental branch in which each of these cars exists and then the user is responsible to ask the corresponding branches for the prices of the cars. It has the same format as input file user2.txt.

6- Input file: branch1.txt
This is the file that contains the cars along with their price that car rental branch 1 has. The file consists of five lines in which, each line has a record for one car. Every line consists of two different parts which are separated with the special character "#".Thus, it is easier for you when you want to parse the input file and extract the information from there (check out the function strtok() when you are dealing with strings in C/C++). The first part of each line is the make and the model of the car and the second part is the price per day of the car in branch 1. It is assumed that each branch has 5 available cars, that is why there are only 5 lines in this input file. Here is the format of a sample branch1.txt file:

Honda Accord#90
Tata Nano#40
Chevy Aveo#50
Jeep Grand Cherokee#130
Ford Focus#80

7- Input file: branch2.txt
This is the file that contains the cars of Branch 2. It has the same format as branch1.txt.

8- Input file: branch3.txt
This is the file that contains the cars of Branch 3. It has the same format as branch1.txt.

C. A more detailed explanation of the problem

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

Phase1:
In this phase, the three car rental branches open their input file (branch1.txt, or branch2.txt or branch3.txt) and send the make and the model of the cars to the central database. More specifically, each car rental branch opens a TCP connection with the central database to send the make and the model of the cars that exist in the branch. It sends one packet per car that is in the branch. This means that the car rental branches should know the TCP port number of the central database in advance. In other words you must hardcode the TCP port number of the central database in the Branch code. Table 1 shows how static UDP and TCP port numbers should be defined for each entity in this project. Each car rental branch then will use one dynamically-assigned TCP port number (different for each car rental branch)

and establish one TCP connection to the central database (see Requirement 1 of the project description). Thus, there will be three different TCP connections to the central database (one from each branch).

As soon as the central database receives the packets with the make and the model of the cars from the three branches, it stores locally the available cars in the system along with the index of the branch(es) in which the cars can be found. Note that multiple car branches may have the same car and the central database should store all the branches in which one specific car can be found. If later on, one user is interested in a car that exists in more than one branches, the central database should give him/her the index of the branch with the smallest index among the ones that the specific car can be found. It is up to you to decide how you are going to store in which branches the different cars exist. It may be stored in a file or in the memory (e.g. through an array or a linked list of structs that you can define). Before you make this decision, you have to think of how you are going to query later the central database. By the end of phase 1, we expect that the central database knows which cars are available in the system and in which car rental branch(es) they exist.

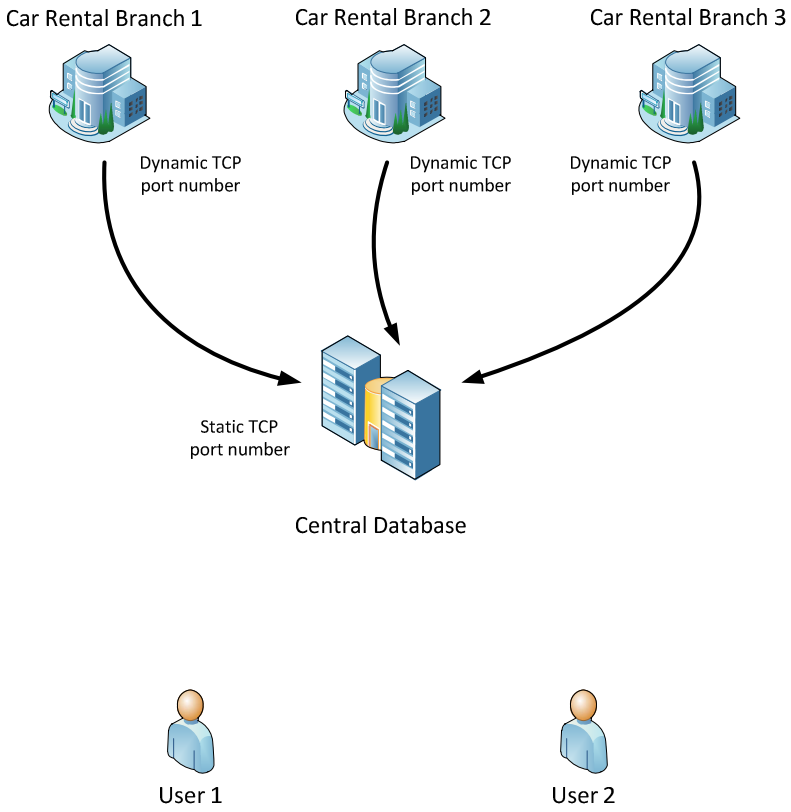

Figure 1. Communication in Phase 1

Phase2:

In phase 2 of this project, each of the users sends his/her queries to the central database through UDP connections. More specifically, each user opens a UDP connection to the central database to send the packets with the queries for cars. This means that each user should know in advance the static UDP port of the central database. You can hardcode this static UDP port setting the value according to Table 1. Then, it opens a UDP connection to this static UDP port of the central database. The UDP port number on the user side of the

UDP connection is dynamically assigned (from the operating system). Thus, for this phase there are two UDP connections to the central database, one for each user in the system. Each user sends one packet for each car it wants to query the central database for.

As soon as the central database receives a query for a car, it searches for it in the file or the array/linked list that stored the information regarding the cars in phase 1. If the car exists, it replies to the corresponding user with the index of the car rental branch that has the car. If the specific car is not in the system, the central database replies with the number 0, and the user realizes that the car is not present in the system and no further handling for this car is required. If the same make and model of a car that the user queried is present in more than one branch, the central database should return the number of the branch with the smallest index.

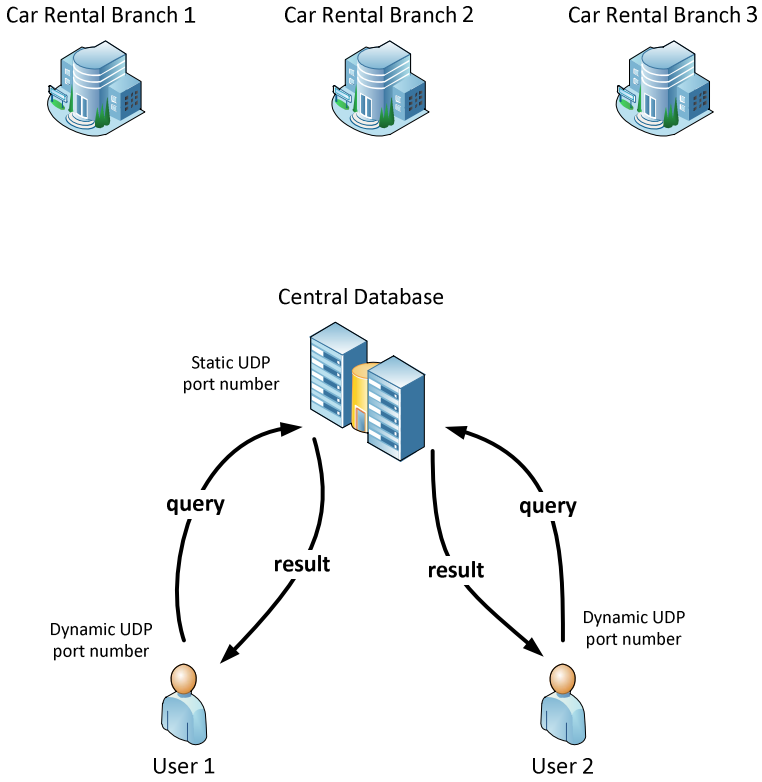

Figure 2. Communication in Phase 2

Phase3:

In this phase, each user reads the reply packets from the central database and opens a TCP connection to the corresponding branches to ask for the price of the cars. This means, that if one user is interested in the price of three cars that each one of them is in a different car rental branch, the user has to open three different TCP connections, one for each branch and send a packet to the corresponding branch. If the user is interested in the price of more than one car that exist in the same branch, it has to use the same TCP connection and send the requests (one for each car) to the corresponding branch. This means that the users should know in advance the static TCP port numbers of the three different branches and you have to hardcode these values according to Table 1. As soon as a branch receives a request from a user, it should search for the make and the model of the car and send back a

reply to the user containing the make and the model of the car along with the price. Then, when the user receives this packet should print an appropriate message in the screen.

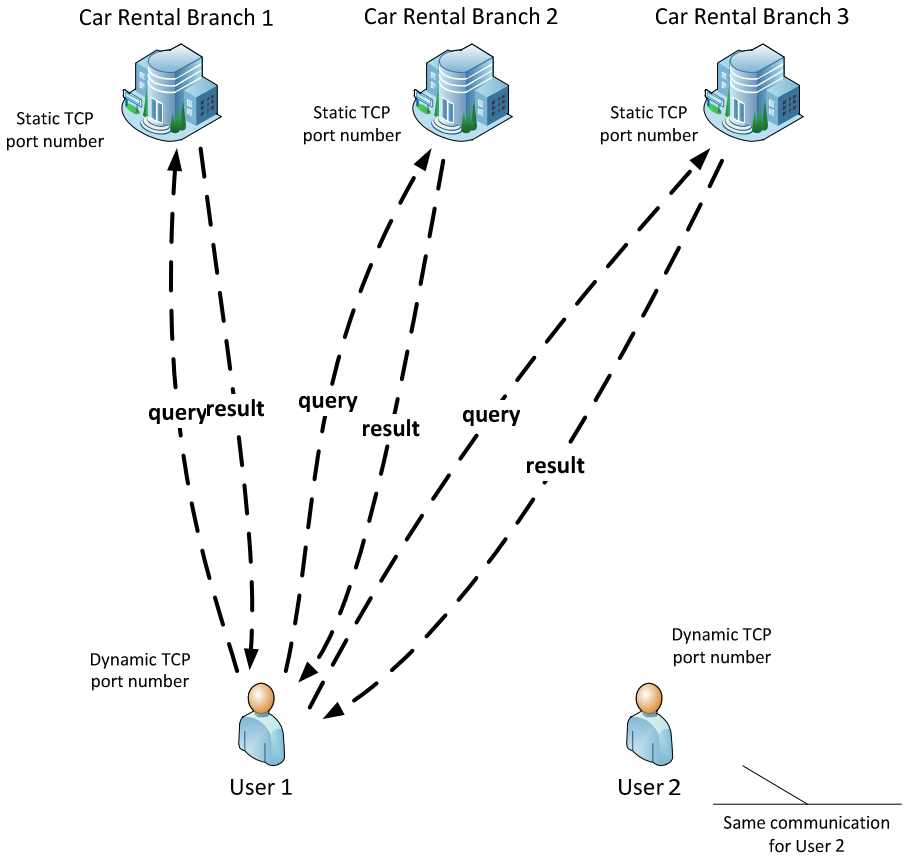

Figure 3. Communication in Phase 3

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

Process	Dynamic Ports	Static Ports
Branch1	1 TCP (phase 1)	1 TCP, 21200 + xxx (last digits of your ID) (phase 3)
Branch2	1 TCP (phase 1)	1 TCP, 21300 + xxx (last digits of your ID) (phase 3)
Branch3	1 TCP (phase 1)	1 TCP, 21400 + xxx (last digits of your ID) (phase 3)
Database	0	1 TCP, 3300 + xxx (last digits of your ID) (phase 1), 1 UDP, 3400 + xxx (last digits of your ID) (phase 2)
User1	1 UDP (phase 2), max 3 TCP (phase 3)	0
User2	1 UDP (phase 2), max 3 TCP (phase 3)	0

For example, if your USC ID number is 0123-4567-89, the static TCP port number of your Database in the first phase will be 3300+789 = 4089.

D. On-screen Messages

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

Table 2. User's on-screen messages

Event	On Screen Message
Upon Startup of Phase 2	<User#> has UDP port … and IP address …
Sending a packet to the central database	Checking <car> in the database
Upon sending all the cars to the central database	Completed car queries to the database from <User#>.
Receiving a response packet from the database	Received location info of <car> from the database
End of Phase 2	End of Phase 2 for <User#>
Upon Startup of Phase 3	<User#> has TCP port … and IP address …
Upon establishing a TCP connection to each branch	<User#> is now connected to <Branch#>
Sending a car query to a branch	Sent a query for <car> to <Branch#>
Receiving the price of a car	<car> in <Branch#> with price <price>
End of Phase 3	End of Phase 3 for <User#>

Table 3. Branch's on-screen messages

Event	On Screen Message
Upon startup of Phase 1	<Branch#> has TCP port … and IP address … for Phase 1
Upon establishing a TCP connection to the database	<Branch#> is now connected to the database
Sending a car's make and model to the central database	<Branch#> has sent <car> to the database
Upon sending all the cars' make and model to the central database	Updating the database is done for <Branch#>
End of Phase 1	End of Phase 1 for <Branch#>
Upon startup of Phase 3	<Branch#> has TCP port … and IP address … for Phase 3
Upon receiving a car's make and model from a user	<Branch#> received query for <car> from <User#>
Sending the price of a car to a user	<Branch#> sent the price of <car> to <User#>

Table 4. Database's on-screen messages

Event	On screen message
Upon startup of Phase 1	The central database has TCP port … and IP address …
Upon receiving all the cars' make and model from a branch	Received the car list from <Branch#>

End of Phase 1	End of Phase 1 for the database
Upon startup of Phase 2	The central database has UDP port … and IP address …
Sending a response to a car query	Sent branch info about <car> to <User#>
End of Phase 2	End of Phase 2 for the database

E. Assumptions

1. The Processes are started in this order: Database, Branch and then we have some delay (e.g 5 seconds) to guarantee that the central database stores the information related to the cars and then user.
2. If you need to have more code files than the ones that are mentioned here, please use meaningful names and all small letters and mention them all in your README file.
3. You are allowed to use blocks of code from Beej's socket programming tutorial (Beej's guide to network programming) in your project.
4. When you run your code, if you get the message "port already in use" or "address already in use", please first check to see if you have a zombie process (from past logins or previous runs of code that are still not terminated and hold the port busy). If you do not have such zombie processes or if you still get this message after terminating all zombie processes, try changing the static UDP or TCP port number corresponding to this error message (all port numbers below 1024 are reserved and must not be used). If you have to change the port number, please do mention it in your README file.

F. Requirements

1. Do not hardcode the TCP or UDP port numbers that must be obtained dynamically. Use getsockname() function to retrieve the locally-bound port number wherever ports are assigned dynamically. Refer to Table 1 to see which ports are statically defined and which ones are dynamically assigned. Sometimes, if you call getsockname() before the first call of sendto()/send(), you will get a port number 0. If that is the case, make sure you call getsockname() after the first call of sendto()/send() in your code. It is okay to postpone the on-screen messages till you have a valid port number.
2. You can use gethostbyname() or getaddrinfo() to obtain the IP address of nunki.usc.edu or the local host (127.0.0.0).
3. You can either terminate all processes after completion of phase3 or assume that the user will terminate them at the end by pressing ctrl-C.
4. All the naming conventions and the on-screen messages must conform to the previously mentioned rules.
5. You are not allowed to pass any parameter or value or string or character as a command-line argument. No user interaction must be required (except for when the user runs the code obviously). Everything is either hardcoded or dynamically generated as described before. By hardcoded, we mean that there should be #define statements in the beginning of the source code files with the corresponding port numbers. If you do not follow this requirement and use the port numbers directly inside your code, we will deduct points.
6. All the on-screen messages must conform exactly to the project description. You must not add any more on-screen messages or modify them. If you need to do so for

debugging purposes, you must comment out all of the extra messages before you submit your project.

7. Using fork() or similar system calls to create concurrent processes is not mandatory if you do not feel comfortable using them. However, the use of fork() for the creation of a new process in phases 1 and 3 when a new TCP connection is accepted is mandatory and everyone should support it. This is useful when three different clients are trying to connect to the same server simultaneously. If you don't use fork in the server when a new connection is accepted, the server won't be able to handle the concurrent connections.
8. Please do remember to close the socket and tear down the connection once you are done using that socket.

G. Programming platform and environment

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

H. Programming languages and compilers

You must use only C/C++ on UNIX as well as UNIX Socket programming commands and functions. Here are the pointers for Beej's Guide to C Programming and Network Programming (socket programming):

http://www.beej.us/guide/bgnet/

(If you are new to socket programming please do study this tutorial carefully as soon as possible and before starting the project)

http://www.beej.us/guide/bgc/

Once you run xwin and open an ssh connection to nunki.usc.edu, you can use a unix text editor like emacs or vi to type your code and then use compilers such as g++ (for C++) and gcc (for C) that are already installed on nunki to compile your code. You must use the following commands and switches to compile yourfile.c or yourfile.cpp. It will make an executable by the name of "yourfileoutput".

```
gcc -o yourfileoutput yourfile.c -lsocket -lnsl -lresolv
g++ -o yourfileoutput yourfile.cpp -lsocket -lnsl -lresolv
```

Do NOT forget the mandatory naming conventions mentioned before!

Also inside your code you need to include these header files in addition to any other header file you think you may need:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/wait.h>
```

I. Submission Rules

1. Along with your code files, include a README file. In this file write
    a. Your **Full Name** as given in the class list
    b. Your **Student ID**
    c. What you have done in the assignment.
    d. What your code files are and what each one of them does. (Please do not repeat the project description, just name your code files and briefly mention what they do).
    e. What the TA should do to run your programs. (Any specific order of events should be mentioned.)
    f. The format of all the messages exchanged. For example if in the second phase the database sends a reply packet to the user in which it includes the make and the model of the car as well as the branch id, the format of the message could be <make and model of the car>#<branch_id>.
    g. Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
    h. Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.)
2. Include a makefile in order for us to be able to compile your source code. Make sure the executable files that you generate are named: Branch, Database, User (first letter is capital). If you don't use fork() to generate concurrent Branch and User processes your

executable files should be: Branch1, Branch2, Branch3, Database, User1, User2 (first letter is capital). If you haven't written a makefile before, you can search the web for sample makefiles in order to create yours.

3. Compress all your files including the README file into a single "tar ball" and call it: **ee450_yourUSCusername_session#.tar.gz** (all small letters) e.g. my file name would be ee450_dimitria_session1.tar.gz. Please make sure that your name matches the one in the class list. Also, do not include the special character # in the filename since we won't be able to download your project from the Digital Dropbox. Here are the instructions:
    a. On nunki.usc.edu, go to the directory which has all your project files. Remove all executable and other unnecessary files. Only include the required source code files and the README file. Now run the following commands:
    b. **tar cvf ee450_yourUSCusername_session#.tar *** - Now, you will find a file named "ee450_yourUSCusername_session#.tar" in the same directory.
    c. **gzip ee450_yourUSCusername_session#.tar** – Now, you will find a file named "ee450_yourUSCusername_session#.tar.gz" in the same directory.
    d. Transfer this file from your directory on nunki.usc.edu to your local machine. You need to use an FTP program such as FileZilla to do so. (The FTP programs are available at software.usc.edu and you can download and install them on your windows machine.)
4. Upload "ee450_yourUSCusername_session#.tar.gz" to the Digital Dropbox (available under Tools) on the DEN website. After the file is uploaded to the dropbox, you must click on the "send" button to actually submit it. If you do not click on "send", the file will not be submitted.
5. Right after submitting the project, send a one-line email to your designated TAs (NOT all TAs) informing him that you have submitted the project to the Digital Dropbox. Please do NOT forget to email the TAs or your project submission will be considered late and will automatically receive a zero.
6. 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.
7. You must allow at least 12 hours before the deadline to submit your project and receive the confirmation email from the TA.
8. By the announced deadline all Students must have already successfully submitted their projects and received a confirmation email from the TA.
9. 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.
10. 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.
11. 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.
12. **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 will simply receive a zero for the project.**

J. Grading Criteria

Your project grade will depend on the following:

1. Correct functionality, i.e. how well your programs fulfill the requirements of the assignment, specially the communications through UDP and TCP sockets.
2. Inline comments in your code. This is important as this will help in understanding what you have done.
3. Whether your programs work as you say they would in the README file.
4. Whether your programs print out the appropriate error messages and results.
5. If your submitted codes, do not even compile, you will receive 10 out of 100 for the project.
6. If your submitted codes, compile but when executed, produce runtime errors without performing any tasks of the project, you will receive 10 out of 100.
7. If your codes compile but when executed only perform phase 1 correctly, you will receive 40 out of 100.
8. If your codes compile but when executed perform only phase 1 and phase 2 correctly, you will receive 80 out of 100.
9. If your code compiles and performs all tasks in all 3 phases correctly and error-free, and your README file conforms to the requirements mentioned before, you will receive 100 out of 100.
10. If you forget to include any of the code files or the README file in the project tar-ball that you submitted, you will lose 5 points for each missing file (plus you need to send the file to the TA in order for your project to be graded.)
11. If your code does not correctly assign and print the TCP or UDP port numbers dynamically (in any phase), you will lose 20 points.
12. You will lose 5 points for each error or a task that is not done correctly.
13. The minimum grade for an on-time submitted project is 10 out of 100.
14. There are no points for the effort or the time you spend working on the project or reading the tutorial. If you spend about 2 months on this project and it doesn't even compile, you will receive only 10 out of 100.
15. Using fork() or similar system calls, for example in the creation of three different Branch processes is not mandatory. If you do use fork() or similar system files in your codes to create concurrent processes (or threads) and they function correctly you will receive 10 bonus points. Note that the use of fork() for the creation of a new process in phase 1 and 3 when a new connection is accepted is mandatory and everyone should support it. This is useful when two different clients are trying to connect to the same server simultaneously. If you don't use fork in the server when a new connection is accepted, the server won't be able to handle the concurrent connections. You won't receive extra credit for this case of fork(), only when you create the three Branches using the same source file Branch.c/Branch.cpp and when you create the two users using the same source file User.c/User.cpp.
16. The maximum points that you can receive for the project with the bonus points is 100. In other words the bonus points will only improve your grade if your grade is less than 100.
17. Your code will not be altered in any ways for grading purposes and however it will be tested with different input files. Your designated TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not.

K. Cautionary Words

1. Start on this project early!!! **It is strongly recommended that you complete the first phase of the project at the end of Spring Break (March 19th).**
2. In view of what is a recurring complaint near the end of a project, we want to make it clear that the target platform on which the project is supposed to run is nunki.usc.edu. It is strongly recommended that students develop their code on nunki. In case students wish to develop their programs on their personal machines, possibly running other operating systems, they are expected to deal with technical and incompatibility issues (on their own) to ensure that the final project compiles and runs on nunki.
3. You may create zombie processes while testing your codes, please make sure you kill them every time you want to run your code. To see a list of all zombie processes even from your past logins to nunki, try this command: ps –aux | grep <your_username>
4. Identify the zombie processes and their process number and kill them by typing at the command-line: kill -9 <processnumber>
5. You can kill a process if you know its name by typing the command line: killall -9 <processname>
6. If you want to run all or some of the processes in the same terminal you can use the special character '&' after the name of the process. For example, you can type the following in the same terminal:
   ./ Database &
   ./ Branch &
   ./ User
7. There is a cap on the number of concurrent processes that you are allowed to run on nunki. If you forget to terminate the zombie processes, they accumulate and exceed the cap and you will receive a warning email from ITS. Please make sure you terminate all such processes before you exit nunki.
8. Please do remember to terminate all zombie or background processes, otherwise they hold the assigned port numbers and sockets busy and we will not be able to run your code in our account on nunki when we grade your project.

Emails of TAs (in alphabetical order):

Dimitris Antonellis: dimitria@usc.edu

Mohammad Abdel-Majeed: abdelmaj@usc.edu

Mohammad Mirza-Aghatabar: mirzaagh@usc.edu

Yue (Sam) Gao: yuegao@usc.edu

Good luck.