EE450 Socket Programming Project, Fall 2010
Due Date: Tuesday November 30th, 2010 11:59 AM (Noon)
(The deadline is the same for all on-campus and DEN off-campus students)
Hard deadline (Strictly enforced)

The objective of this assignment is to familiarize you with UNIX socket programming. This assignment is worth **10%** of your overall grade in this course.
It is an individual assignment and no collaborations are allowed. Any cheating will result in an automatic F in the course (not just in the assignment).
If you have any doubts/questions please feel free to contact the TAs and cc professor Zahid as usual.

The Problem:

In this project you will be simulating a query system for books in a number of libraries. The communication between the users, the libraries and the central database will be over TCP and UDP sockets in a network with client-server architecture. The project has 3 major phases: Submission of the available books in the libraries to the central database, Query to the Database from the users, Connection to the appropriate Library for the retrieval of the book description. In Phase 1 and 2 all communications are through UDP sockets. However, in phase 3 all communications are over TCP sockets.

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- Library
   You must create 3 concurrent Libraries
   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: **Library.c** or **Library.cc** or **Library.cpp.** Also you must call the corresponding header file (if any) **Library.h**. You must follow this naming convention.
   ii. Or by running 3 instances of the Library code. However in this case, you probably have 3 pieces of code for which you need to use one of these sets of names: (**Library1.c, Library2.c, Library3.c**) or (**Library1.cc, Library2.cc, Library3.cc)** or (**Library1.cpp, Library2.cpp and Library3.cpp)**. Also you must call the corresponding header file (if any) **Library.h** or **Library1.h, Library2.h** and **Library3.h**. You must follow this naming convention.

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.

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.

i. Or by running 3 instances of the User code. However in this case, you probably have 3 pieces of code for which you need to use one of these sets of names: (**User1, 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.

4- Input file: queries1.txt

This is the file that contains the queries to the central database that the first user wants to submit. The file consists of three lines and each line has the title of a book that the user wants to query the central database for. Thus, the first user sends queries for three books only. Here is the format of a sample query file:

Computer Networking
Operating Systems
VLSI Systems

5- Input file: queries2.txt

This is the file that contains the queries to the central database that the second user wants to submit. The file consists of three lines and each line has the title of a book that the user wants to query the central database for. Thus, the second user sends queries for three books only. It has the same format as input file queries1.txt.

6- Input file: library1.txt

This is the file that contains the books along with their descriptions that are available at library 1. The file consists of five lines in which, each line has a record for one book title. Every line consists of two different parts which are separated with the special character "#".Thus, it is easier for you when you want to pass 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 title of the book and the second part is the description of the book. It is assumed that each library has 5 books, that's why there are only 5 lines in this input file. Here is the format of a sample library1.txt file:

Computer Networking#It is designed for an advanced college-level course in networks.
Database Systems#The book describes the latest developments in relational databases.
VLSI Systems#The book presents in-depth coverage of the modern VLSI Design.
Operating Systems#The book helps you master the fundamental concepts of OS.
Algorithm Design#The book helps you learn and design algorithms.

7- Input file: library2.txt
   This is the file that contains the books of Library 2. It has the same format as library1.txt.

8- Input file: library3.txt
   This is the file that contains the books of Library 3. It has the same format as library1.txt.

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 libraries open their input file (library1.txt or library2.txt or library3.txt) and send the titles of the books to the central database. More specifically, each library opens a UDP connection with the central database to send the titles of the books that are present there. It sends one packet per book that is in the library over the same UDP connection. This means that the libraries should know the UDP port number of the central database in advance. In other words you must hardcode the UDP port number of the central database in the Library code. Table 1 shows how static UDP and UDP port numbers should be defined for each entity in this project. Each library then will use one dynamically-assigned UDP port number (different for each library) and establish one UDP connection to the central database. Thus, there will be three different UDP connections to the central database (one from each library). As soon as the central database receives the packets with the titles of the books from the three libraries, it stores locally the available books in the system along with the index of the library that has each book. It is up to you to decide how you are going to store this information. It may be stored in a file or in the memory (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 for book titles in the central database. By the end of phase 1, we expect that the central database knows which books are available in the system and in which library they are.

Phase2:
In phase 2 of this project, each of the users sends their queries to the central database through UDP connections. More specifically, opens a UDP connection to the central database to send the packets with the queries for book titles. 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. Thus, for this phase there are two UDP connections to the central database, one for each user in the system. Note that each user uses its UDP connection to the central database to send all of its packets. Each user sends one packet for each book title it wants to query the central database for. As soon as the central database receives a query for a book title, it searches for this title in the file or the array/linked list that stored the information regarding the books in phase 1. If the book exists, it replies to the corresponding user with the number of the library that has the book and

the title of the book. If the book is not in the system, the central database replies with the number 0, and the user realizes that the book is not present in the system and no further handling for this book is required.

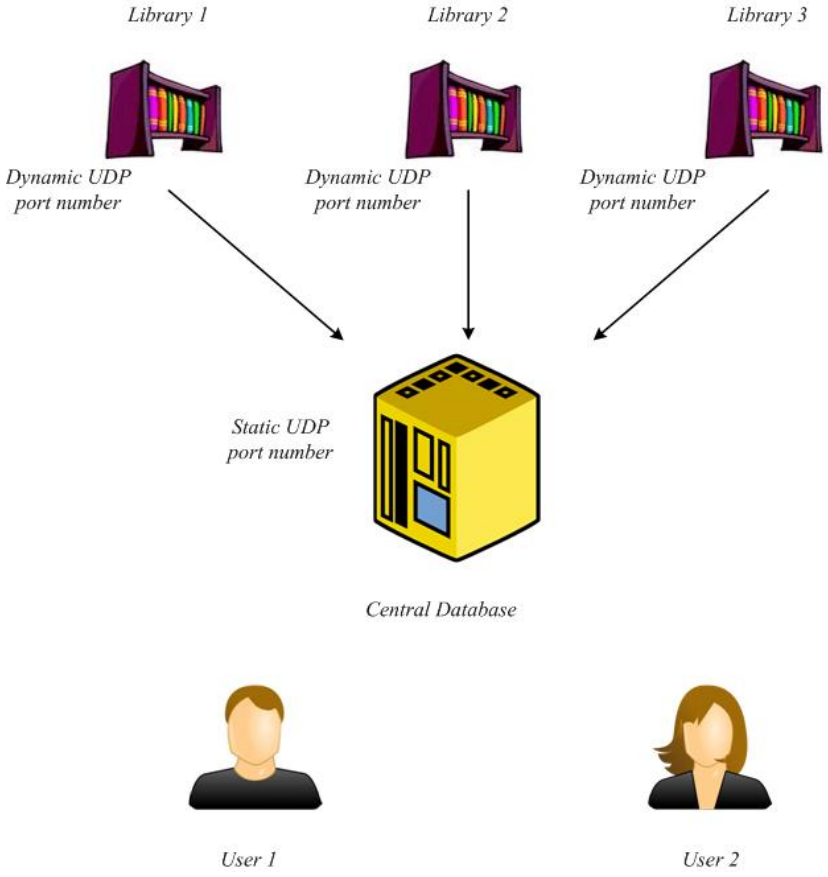

Figure 1. Port numbers needed in Phase 1

Phase3:

In this phase, each user reads the reply packets from the central database and opens a TCP connection to the corresponding library to ask for the description of the book. This means, that if one user is interested for three books that each one of them is in a different library, the user has to open three TCP connections, one for each library and send a packet to the corresponding library. If a user is interested for the descriptions of more than one book that exist in the same library, it has to use the same TCP connection and send the requests (one for each book) to the corresponding library. This means that the users should know in advance the static TCP port number of the three different libraries and you have to hardcode these values according to Table 1. As soon as a library receives a request from a user, it should search for the description of the book title and send back a reply to the user containing the book title and the book description. Then, when the user receives this packet should print an appropriate message in the screen.

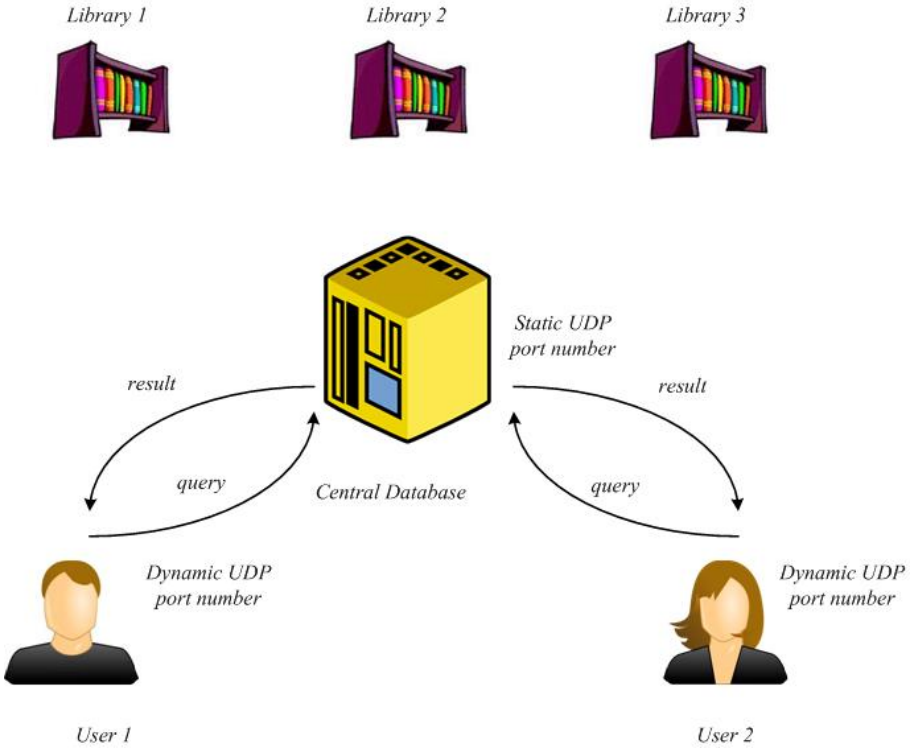

Figure 2. Port numbers needed in phase 2

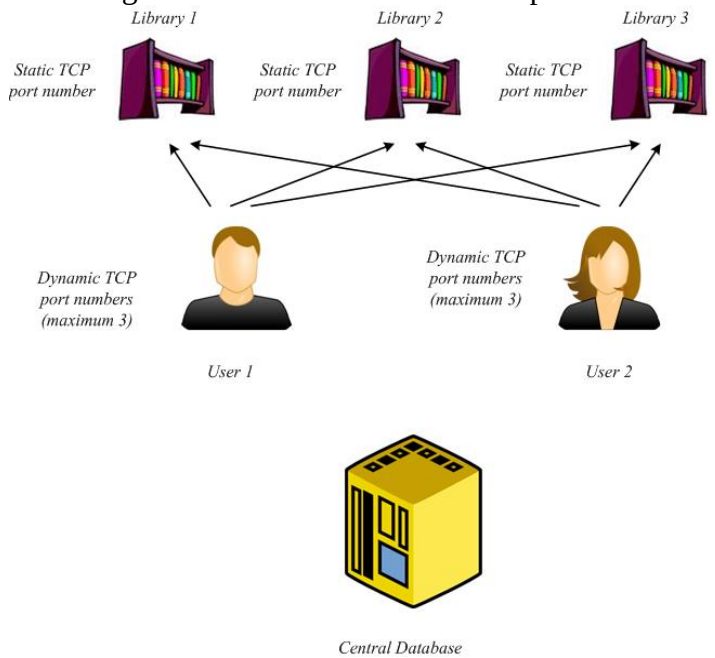

Figure 3. Port numbers needed in phase 3

A more detailed description of the project is shown in the following figures.

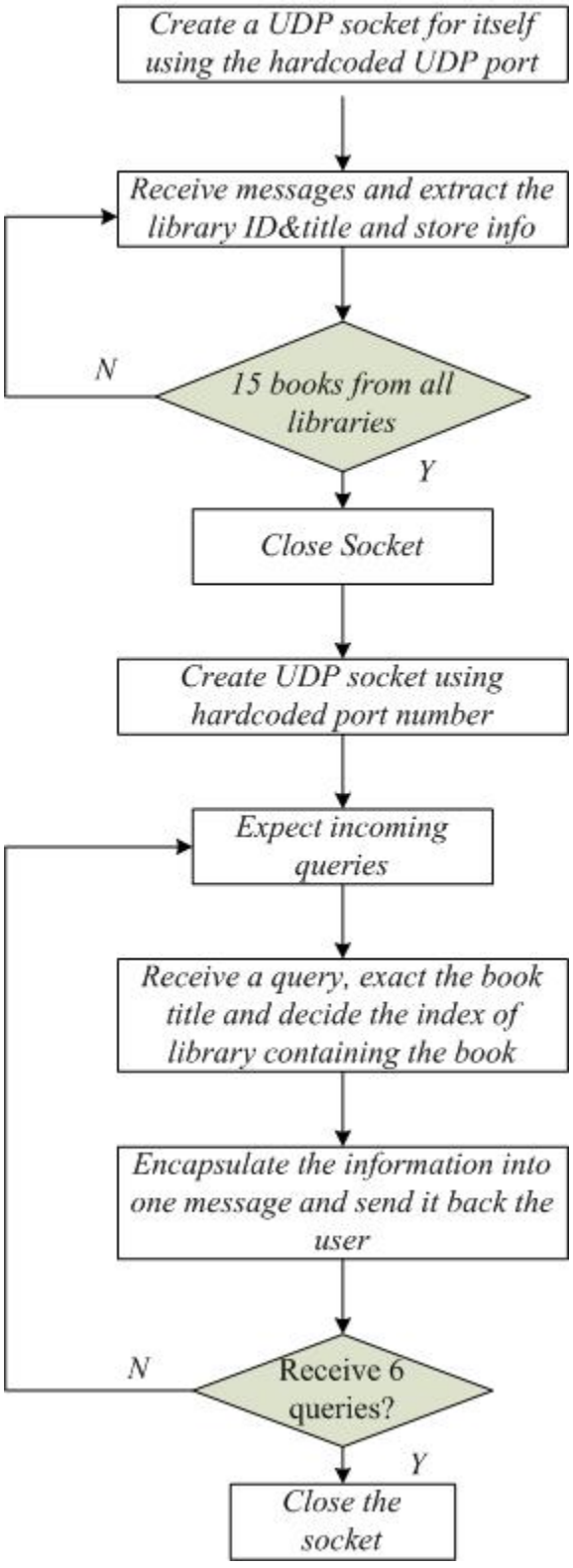

Figure 4. The flow chart of database

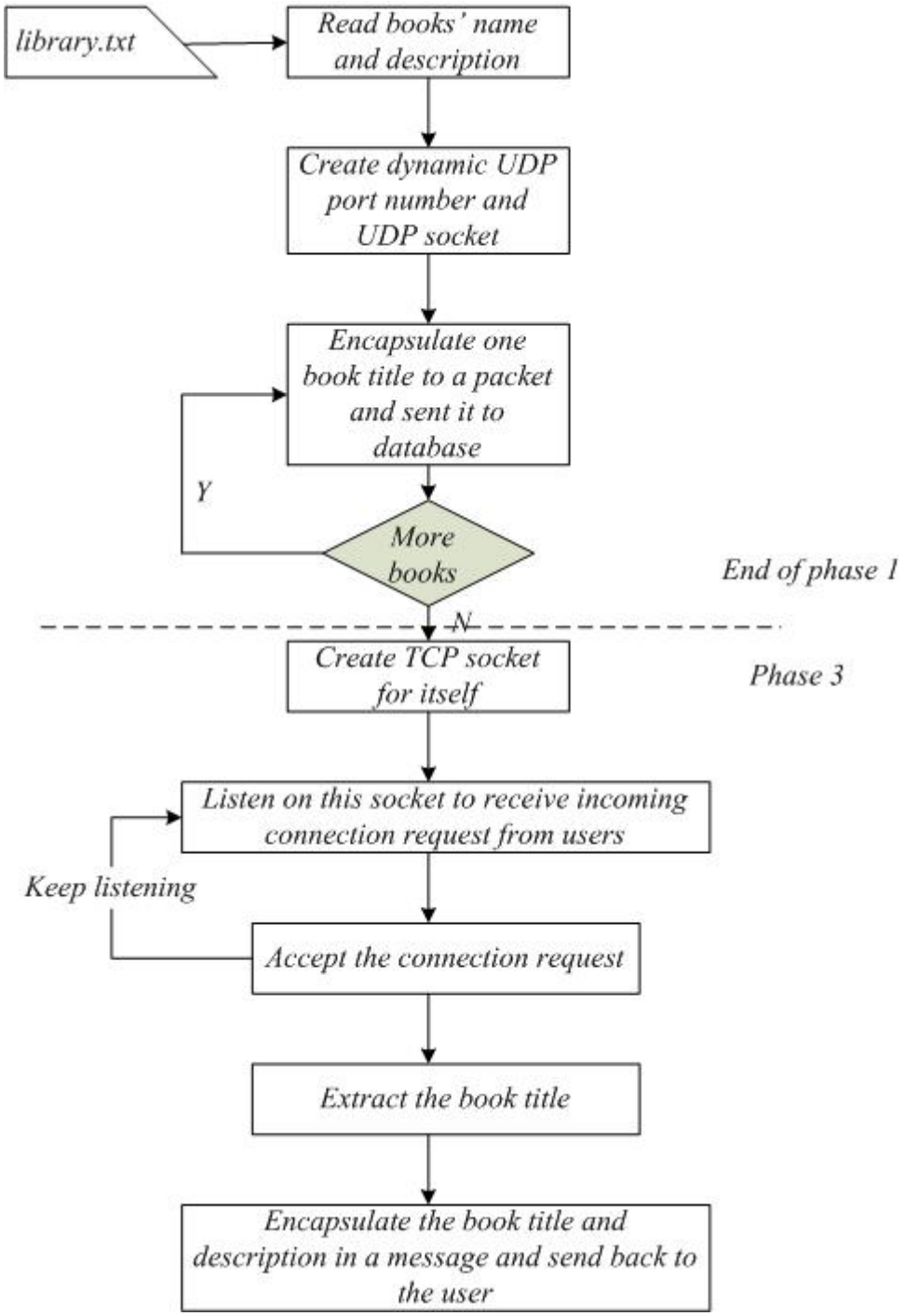

Figure 5. Flow chart of library

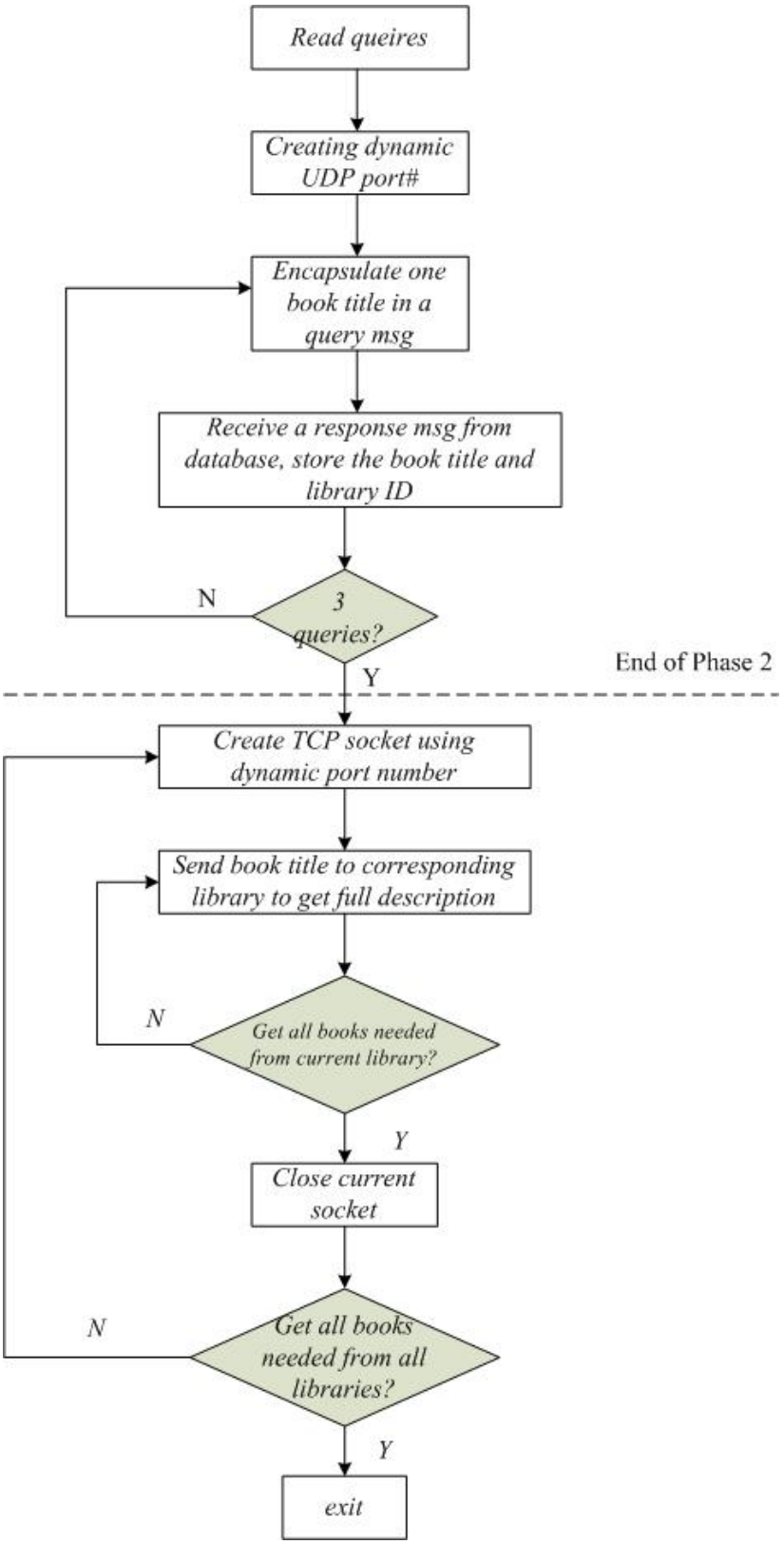

Figure 6. Flow chart of users

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

Process	Dynamic Ports	Static Ports
Library1	1 UDP	1 TCP, 21100 + xxx (last digits of your ID) (phase 3)
Library2	1 UDP	1 TCP, 21200 + xxx (last digits of your ID) (phase 3)
Library3	1 UDP	1 TCP, 21300 + xxx (last digits of your ID) (phase 3)
Database	0	1 UDP, 3200 + xxx (last digits of your ID) (phase 1), 1 UDP, 3100 + xxx (last digits of your ID) (phase 2)
User1	1 UDP (phase 2), 3 TCP (phase 3)	0
User2	1 UDP (phase 2), 3 TCP (phase 3)	0

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 <book title> in the database
Upon sending all the book queries to the central database	Completed book queries to the database from <User#>.
Receiving a response packet from the database	Received location info of <book title> 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 library	<User#> is now connected to <Library#>
Sending a book query to a library	Sent a query for <book title> to <Library#>
Receiving the description of a book	<book title> in <Library#> with description <description>
End of Phase 3	End of Phase 3 for <User#>

Table 3. Library's on-screen messages

Event	On Screen Message
Upon startup of Phase 1	<Library#> has UDP port … and IP address … for Phase 1

Upon establishing a TCP connection to the database	<Library#> is now connected to the database
Sending a book name to the central database	<Library#> has sent <book title> to the database
Upon sending all the book titles to the central database	Updating the database is done for <Library#>
End of Phase 1	End of Phase 1 for <Library#>
Upon startup of Phase 3	<Library#> has TCP port … and IP address … for Phase 3
Upon receiving a book name from a user	<Library#> received query for <book title> from <User#>
Sending the book description to a user	<Library#> sent the description of <book title> to <User#>

Table 4. Central Database's on-screen messages

Event	**On screen message**
Upon startup of Phase 1	The central database has UDP port … and IP address …
Upon receiving all the book titles from a library	Received the book list from <Library#>
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 book query	Sent location info about <book title> to <User#>
End of Phase 2	End of Phase 2 for the database

Assumptions:

1. The Processes are started in this order: Database, Library and then we have some delay (e.g 5 seconds) to guarantee that the central database stores the information related to the books 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.

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.
2. Use gethostbyname() to obtain the IP address of nunki.usc.edu or the local host.
3. You can either terminate all processes after completion of phase3 or assume that the user will terminate them at the end by pressing ctrl-C.
4. All the naming conventions and the on-screen messages must conform to the previously mentioned rules.
5. You are not allowed to pass any parameter or value or string or character as a command-line argument. No user interaction must be required (except for when the user runs the code obviously). Everything is either hardcoded or dynamically generated as described before.
6. All the on-screen messages must conform exactly to the project description. You must not add anymore onscreen messages. If you need to do so for the debugging purposes, you must comment out all of the extra messages before you submit your project.
7. Using fork() or similar system calls are not mandatory if you do not feel comfortable using them to create concurrent processes.
8. Please do remember to close the socket and tear down the connection once you are done using that socket.

Programming platform and environment:

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

Programming languages and compilers:

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

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

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

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

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

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

Do NOT forget the mandatory naming conventions mentioned before!

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

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

Submission Rules:

1. Along with your code files, include a README file. In this file write
    a. Your **Full Name** as given in the class list
    b. Your **Student ID**
    c. What you have done in the assignment

d. What your code files are and what each one of them does. (Please do not repeat the project description, just name your code files and briefly mention what they do).
e. What the TA should do to run your programs. (Any specific order of events should be mentioned.)
f. The format of all the messages exchanged.
g. Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
h. Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.)

Submissions WITHOUT README files WILL NOT BE GRADED.

2. Compress all your files including the README file into a single "tar ball" and call it: **ee450_yourUSCusername_session#.tar.gz** (all small letters) e.g. my file name would be **ee450_dimitria_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 FileZilla to do so. (The FTP programs are available at software.usc.edu and you can download and install them on your windows machine.)
3. Upload "ee450_yourUSCusername_session#.tar.gz" to the Digital Dropbox (available under Tools) on the DEN website. After the file is uploaded to the dropbox, you must click on the "send" button to actually submit it. If you do not click on "send", the file will not be submitted.
4. Right after submitting the project, send a one-line email to your designated TA (NOT all TAs) informing him or her that you have submitted the project to the Digital Dropbox. Please do NOT forget to email the TA or your project submission will be considered late and will automatically receive a zero.
5. You will receive a confirmation email from the TA to inform you whether your project is received successfully, so please do check your emails well before the deadline to make sure your attempt at submission is successful.
6. You must allow at least 12 hours before the deadline to submit your project and receive the confirmation email from the TA.
7. By the announced deadline all Students must have already successfully submitted their projects and received a confirmation email from the TA.
8. Please take into account all kinds of possible technical issues and do expect a huge traffic on the DEN website very close to the deadline which may render your submission or even access to DEN unsuccessful.

9. Please do not wait till the last 5 minutes to upload and submit your project because you will not have enough time to email the TA and receive a confirmation email before the deadline.
10. Sometimes the first attempt at submission does not work and the TA will respond to your email and asks you to resubmit, so you must allow enough time (12 hours at least) before the deadline to resolve all such issues.
11. You have plenty of time to work on this project and submit it in time hence there is absolutely zero tolerance for late submissions! Do NOT assume that there will be a late submission penalty or a grace period. If you submit your project late (no matter for what reason or excuse or even technical issues), you simply receive a zero for the project.

Grading Criteria:

Your project grade will depend on the following:

1. Correct functionality, i.e. how well your programs fulfill the requirements of the assignment, specially the communications through UDP and TCP sockets.
2. Inline comments in your code. This is important as this will help in understanding what you have done.
3. Whether your programs work as you say they would in the README file.
4. Whether your programs print out the appropriate error messages and results.
5. If your submitted codes, do not even compile, you will receive 10 out of 100 for the project.
6. If your submitted codes, compile but when executed, produce runtime errors without performing any tasks of the project, you will receive 10 out of 100.
7. If your codes compile but when executed only perform 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 are not mandatory however if you do use fork() or similar system files in your codes to create concurrent processes (or threads) and they function correctly you will receive 10 bonus points.
16. If you submit a makefile or a script file along with your project that helps us compile your codes more easily, you will receive 5 bonus points.
17. The maximum points that you can receive for the project with the bonus points is 100. In other words the bonus points will only improve your grade if your grade is less than 100.

18. Your code will not be altered in any ways for grading purposes and however it will be tested with different input files. Your designated TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not.

Cautionary Words:

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

Academic Integrity:

All students are expected to write all their code on their own.

Copying code from friends is called plagiarism not collaboration and will result in an F for the entire course. Any libraries or pieces of code that you use and you did not write, must be listed in your README file. All programs will be compared with automated tools to detect similarities; examples of code copying will get an F for the course. **IF YOU HAVE ANY QUESTIONS ABOUT WHAT IS OR ISN'T ALLOWED ABOUT PLAGIARISM, TALK TO THE TA**. "I didn't know" is not an excuse.