EE 450 Socket Programming Project

Summer 2017 Nazarian

Assigned: Tues, July 11th

Maximum points: 100

Due:

Phase1- Tuesday, July 25th, at 11:59 PM

Phase2- Friday, August 4th, at 11:59 PM

Late submissions will be accepted only during the first two days after a deadline with a maximum of 15% penalty per day. For each day, submissions between 12am and 1am: 2%, 1 and 2am: 4%, 2 and 3am: 8% and after 3am: 15%.

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:

- This assignment is based on individual effort and no collaborations are allowed. You may refer to the syllabus to review the academic honesty policies, including the penalties of violating them. Any questions or doubts about what is cheating and what is not, should be referred to the instructor or the TA.
- 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 Followers at 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 Followers and the answers.
- If you need to email your TAs, please follow the syllabus guidelines mentioned under the assignments.

A. Problem Definition:

In this project you will simulate a twitter process. This process involves the people who send twitters, the server that manages the twitters, and the followers that are reading twitters. The communication among the people sending twitters, server and the followers are performed over TCP and UDP sockets in a network with client-server architecture. The project has 2 major phases: 1) People tweet to server, 2) The server sends twitters to followers, the followers feedback their "like" to server and then the server informs the tweeting people who like their tweets. In phase 1 all

communication is through TCP sockets. In phase 2 the communication is through both TCP and UDP sockets.

B. Code and Input Files:

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

1- Tweet people:
You must create 3 concurrent tweet people

   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: **Tweet.c** or **Tweet.cc** or **Tweet.cpp.** Also you must call the corresponding header file (if any) **Tweet.h**. Note that it is mandatory to follow all the naming convention mentioned in this project definition. Make sure the first letter of the word 'Tweet' is capital.
   ii. Or by running 3 instances of the Tweet code. However in this case, you probably have 3 pieces of code for which you need to use one of these sets of names: (**TweetA.c, TweetB.c, TweetC.c**) or (**TweetA.cc, TweetB.cc, TweetC.cc**) or (**TweetA.cpp, TweetB.cpp, TweetC.cpp**). Also you must call the corresponding header file (if any) **TweetA.h** and **TweetB.h, TweetC.h**. Make sure the first letter of the word 'Tweet' is capital.

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

3- Followers
You must create 5 concurrent followers

   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: **Follower.c** or **Follower.cc** or **Follower.cpp**. Also you must call the corresponding header file (if any) **Follower.h**. Make sure the first letter of the word 'Follower' is capital.
   ii. Or by running 5 instances of the Follower code. However in this case, you probably have 5 pieces of code for which you need to use one of these sets of names: (**Follower1.c, Follower2.c, Follower3.c, Follower4.c, Follower5.c**) or (**Follower1.cc, Follower2.cc, Follower3.cc, Follower4.cc, Follower5.cc**) or (**Follower1.cpp, Follower2.cpp, Follower3.cpp, Follower4.cpp, Follower5.cpp**). Also you must call the corresponding header file (if any) **Follower1.h, Follower2.h, Follower3.h, Follower4.h** and **Follower5.h**. Make sure the first letter of the word 'Follower' is capital.

4- Input file: Follower1.txt
This is the file that contains the information of Follower1. The file consists of at most four lines. 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 line is the following of Follower1. The next lines indicate whether they clik "like" for their followings:

Following:TweetA, TweetB, TweetC
TweetA:like
TweetB:
TweetC:

5- Input file: Follower2.txt
This is the file that contains the information of Follower2. It has the same format as Follower1.txt.

6- Input file: Follower3.txt
This is the file that contains the information of Follower3. It has the same format as Follower1.txt.

7- Input file: Follower4.txt
This is the file that contains the information of Follower4. It has the same format as Follower1.txt.

8- Input file: Follower5.txt
This is the file that contains the information of Follower5. It has the same format as Follower1.txt.

9- Input file: TweetA.txt
This is the file that contains the tweets:
The distance from UCLA to best University is less than 15 miles.

10- Input file: TweetB.txt
This is the file that contains the program information of Tweet B. It has the same format as TweetA.txt.

11- Input file: TweetC.txt
This is the file that contains the program information of Tweet C. It has the same format as TweetA.txt.

C. Detailed Explanation of the Problem:

This project is divided into 2 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.

Phase 1:

In this phase, each tweet person opens its input file (TweetA.txt, TweetB.txt or TweetC.txt) and opens a TCP connection with the server to send tweets. It sends one packet per program that it has. This means that the Tweet should know the static TCP port number of the server in advance. In other words you must hardcode the TCP port number of the server in the Tweet code. Table 1 shows how static UDP and TCP port numbers should be defined for each entity in this project. Each Tweet then uses one dynamically-assigned TCP port number (different for each Tweet) and establishes one TCP connection to the server (see Requirement 1 of the project description). Thus, there are three different TCP connections to the server (one from each tweet).

As soon as the server receives the tweets, it stores locally the available tweets in the system along with the person names who offer the tweets. 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 (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 a Follower sends their feedback later to the server.

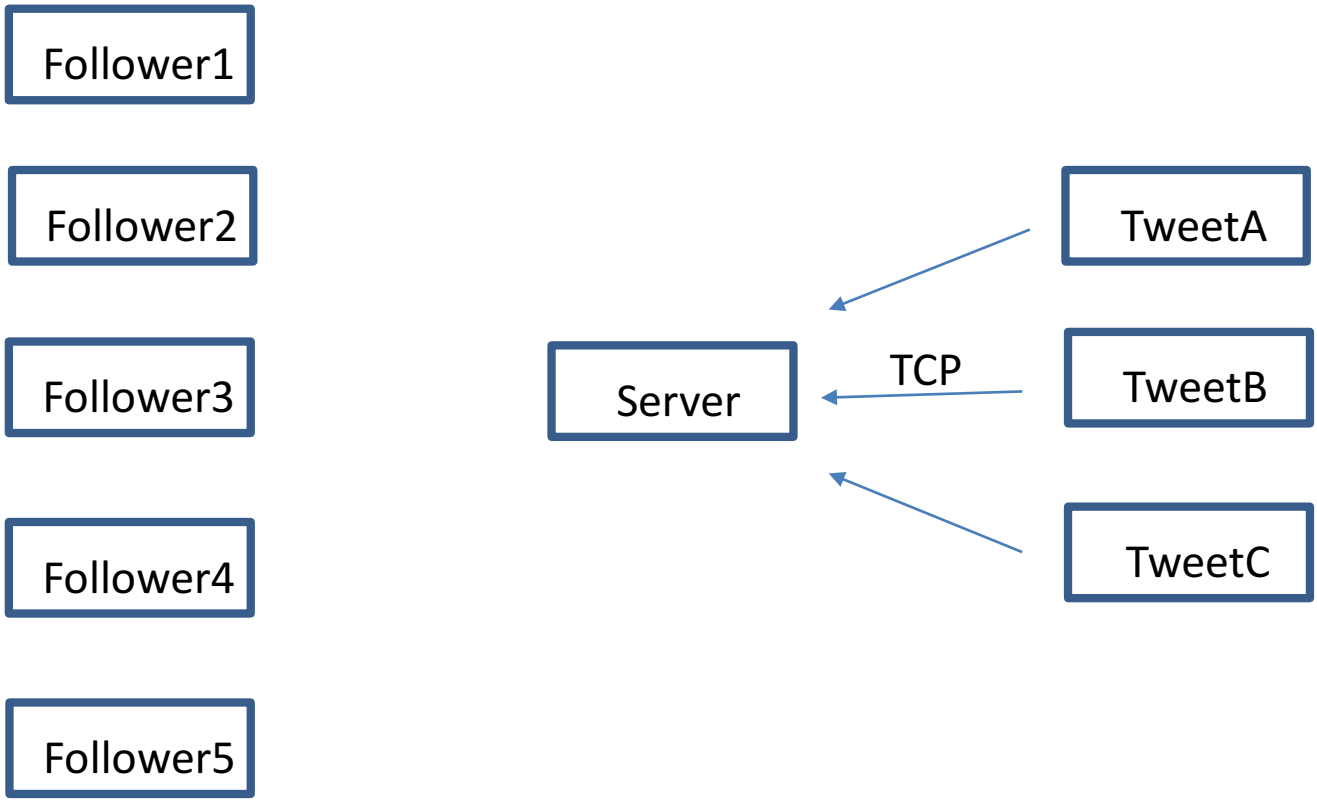


Figure 1. Communication in Phase 1

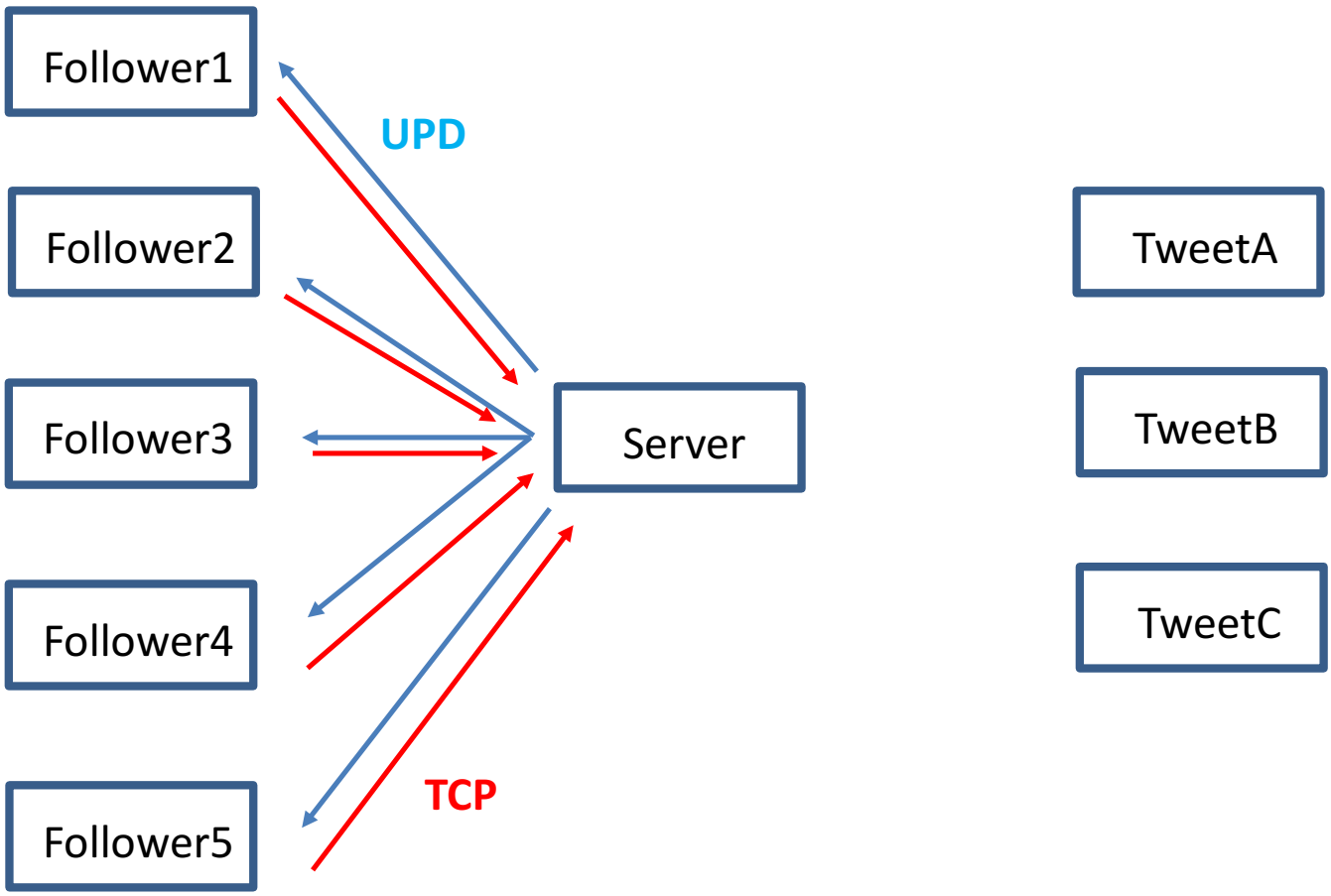
Phase 2:

The first part uses TCP connections and the second part uses UDP connections.

In phase 2 of this project, the server sends the tweets to the followers through UDP. The server should know the static UDP port of each Follower in advance. You can hardcode these static UDP ports and set the value according to Table 1. The UDP port number on the server side of the UDP

connection is dynamic for this part. There is one UDP connection from the server to each Follower. After received the tweets, each Follower sends his/her feedback to the server through TCP connections. More specifically, each Follower opens a TCP connection to the server to send the packets with the details of his/her feedback. This means that each Follower should know in advance the static TCP port of the server. You can hardcode this static TCP port and set the value according to Table 1. Then, it opens a TCP connection to this static TCP port of the server. The TCP port number on the Follower side of the TCP connection is dynamically assigned (from the operating system). Thus there are five TCP connections to the server, one for each Follower in the system. Each Follower sends one packet to the server for each line in the feedback file (at most 4 packets in total).

When the server receives all the packets from a Follower, it forwards the feedback to the pacific tweet person through TCP like in phase 1.



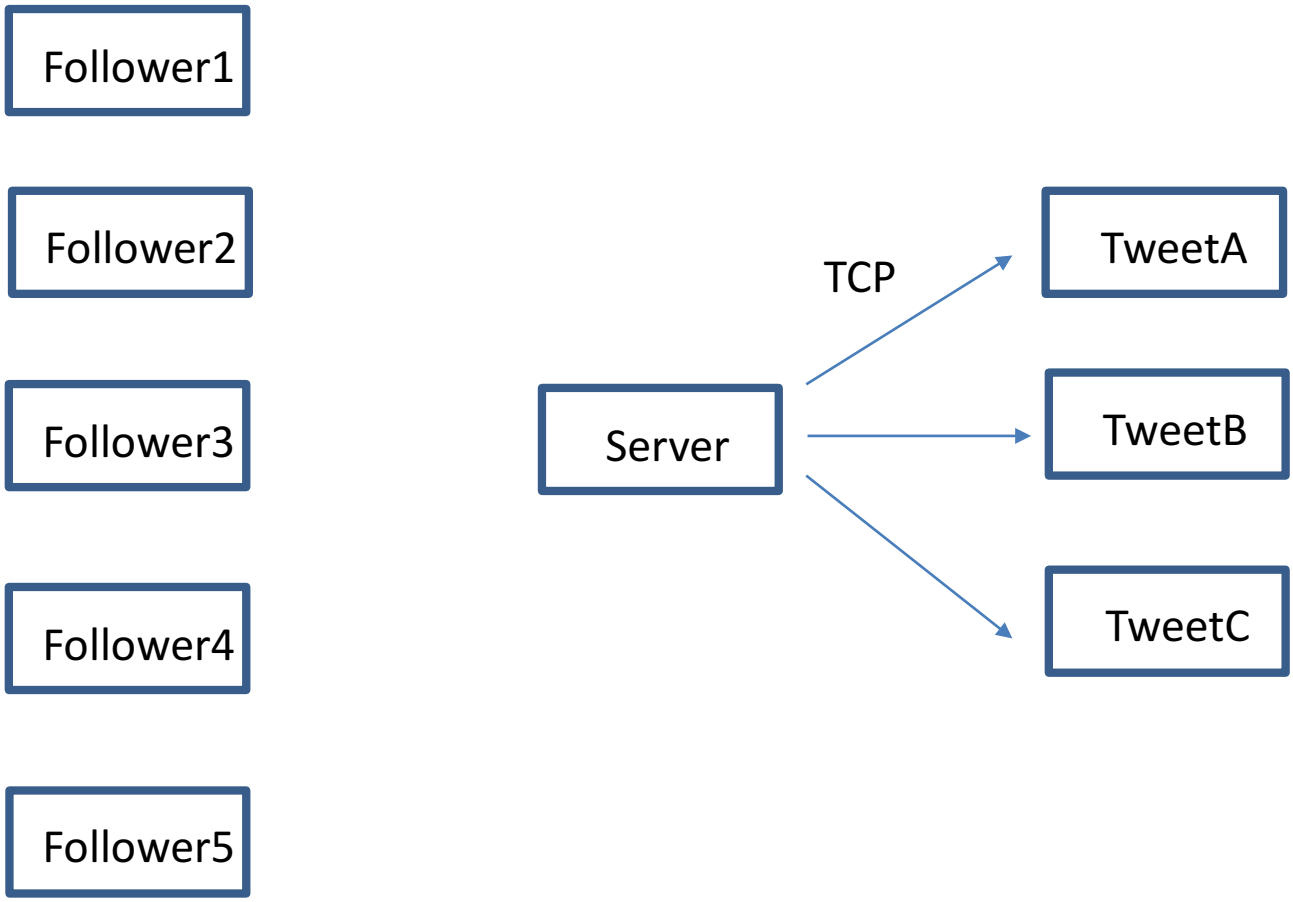


Figure 2. Communication in Phase 2

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

Process	Dynamic Ports	Static Ports
TweetA	1 TCP (phase 1&2)	
TweetB	1 TCP (phase 1&2)	
TweetC	1 TCP (phase 1&2)	
Server	max 5 UDP (phase 2)	1 TCP, 3300 + xxx (last digits of your ID) (phase 1 and phase 2)
Follower1	1 TCP (phase 2)	1 UDP, 21400 + xxx (last digits of your ID) (phase 2)
Follower2	1 TCP (phase 2)	1 UDP, 21500 + xxx (last digits of your ID) (phase 2)
Follower3	1 TCP (phase 2)	1 UDP, 21600 + xxx (last digits of your ID) (phase 2)

Follower4	1 TCP (phase 2)	1 UDP, 21700 + xxx (last digits of your ID) (phase 2)
Follower5	1 TCP (phase 2)	1 UDP, 21800 + xxx (last digits of your ID) (phase 2)

Note that 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. An Illustrative Example:

Consider an example with 5 Followers and 3 Tweets. Each Tweet has exactly 3 different programs. Necessary information is provided in 8 input files: Follower1.txt, Follower2.txt, Follower3.txt, Follower4.txt, Follower5.txt as follows:

Follower1.txt	Follower2.txt	Follower3.txt	Follower4.txt	Follower5.txt
Following:TweetA, TweetB,TweetC	Following:TweetA, TweetB	Following:TweetB, TweetC	Following:TweetB	Following:TweetA, TweetB,TweetC
TweetA:like	TweetA:	TweetA:	TweetA:	TweetA:like
TweetB:	TweetB:like	TweetB:like	TweetB:	TweetB:like
TweetC:	TweetC:	TweetC: like	TweetC:	TweetC:like

In the first phase, each Tweet creates TCP sockets to send their program information to the server. For example, TweetA reads TweetA.txt and sends the information in the three lines via three packets to the server. The server stores all information locally.

In the second phase, after receive tweets from server through UDP, Follower1 reads Follower1.txt input file, creates TCP sockets to send packets to the server, and receive an acknowledgement from the server. Follower2-Follower5 do similarly. The server forward followers' feedback to Tweets through TCP. The feedback from server to tweets should be as follows:

TweetA	Follower1#like (Note that each line is one packet!) Follower2 Follower5#like
TweetB	Follower1 Follower2#like Follower3#like Follower4 Follower5#like
TweetC	Follower1 Follower3#like Follower4 Follower5#like

E. 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. Tweet's on-screen messages

Event	On Screen Message
Upon startup of Phase 1	<Tweet#> has TCP port … and IP address … for Phase 1
Upon establishing a TCP connection to the server	<Tweet#> is now connected to the server
Sending tweet information to the server	<Tweet#> has sent <tweet> to the server
Upon sending all the tweets' information to the server	Updating the server is done for <Tweet#>
End of Phase 1	End of Phase 1 for <Tweet#>
Upon startup of Phase 2	<Tweet#> has TCP port … and IP address … for Phase 2
Receiving one Follower feedback information	<Follower#> is follwing <Tweet#> <Follower#> has liked <Tweet#>
End of Phase 2	End of Phase 2 for <Tweet#>

Table 3. Server's on-screen messages

Event	On screen message
Upon startup of Phase 1	The server has TCP port … and IP address …
Upon receiving all the tweet information from a Tweet	Received the tweet list from <Tweet#>
End of Phase 1	End of Phase 1 for the server

Upon startup of Phase 2	The server has UDP port … and IP address …
Sending tweet information to the followers	The server has sent <tweet#> to the <Follower#>
Listening to Follower's feedback	The server has TCP port … and IP address
Receiving all the packets from a Follower	Server receive the feedback from <Follower#>
Upon startup of server-tweet TCP connection	The server has TCP port … and IP address …
Sending feedback result to a Tweet	The server has send the feedback result to <Tweet#>
End of Phase 2	End of Phase 2 for the server

Table 4. Follower's on-screen messages

Event	On Screen Message
Upon Startup of Phase 2	< Follower#> has UDP port … and IP address …
Receiving all the packets from Server	< Follower#> has received < Tweet#>
Upon sending all the packets to the server	< Follower#> has TCP port … and IP address …
Sending feedback information to the server	Completed sending feedback for <Follower#>.
End of Phase 2	End of phase 2 for <Follower#>

F. Assumptions:

1. The Processes are started in this order: Server, Tweet and Follower. You are allowed to use delays in your code if you think necessary.
2. If you need to have more code files than the ones 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.

G. 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 3 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 Phase2 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 child process in phase 1 and 2 when a new TCP connection is accepted is mandatory and everyone should support it. This is useful when 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.

H. 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 Follower.
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 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 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/

I. 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>
```

J. 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 **Follower 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 should follow the ones as given in the table.
    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 are 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. If you haven't written a makefile before, you can search the web for sample makefiles in order to create yours.
3. Do not include the sample input '.txt' files (provided by us) in your submission.
4. Compress all your files including the README file into a single "tar ball" and call it: **ee450_yourUSCusername_phase#.tar.gz** (all small letters) e.g. my file name would beee450_hkadu_phase1.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 DEN. 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 (don't include the input .txt files). Now run the following commands:
    b. **tar cvf ee450_yourUSCusername_phase#.tar *** - Now, you will find a file named "ee450_yourUSCusername_phase#.tar" in the same directory.
    c. **gzip ee450_yourUSCusername_phase#.tar** – Now, you will find a file named "ee450_yourUSCusername_phase#.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.)
5. Upload "ee450_yourUSCusername_phase#.tar.gz" to the DEN Submission link on the DEN website.
6. 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.
7. Please do not wait till the last 5 minutes to upload and submit your project.
8. **You have plenty of time to work on this project and submit it in time -- Please refer to the header information of this project description file for the late submission policy. Note that any submission which is late by more than two days will receive a zero.**

K. 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 20 out of 100.
7. If your codes compile but when executed only perform phase 1 correctly, you will receive 50 out of 100.
8. If your code compiles and performs all tasks in both 2 phases correctly and error-free, and your README file conforms to the requirements mentioned before, you will receive 100 out of 100.
9. 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.)
10. If your code does not correctly assign and print the TCP or UDP port numbers dynamically (in any phase), you will lose maximum of 20 points.
11. You will lose 5 points for each error or a task that is not done correctly.
12. The minimum grade for an on-time submitted project is 10 out of 100.
13. 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.
14. Using fork() or similar system calls in the creation of five concurrent Follower processes is not mandatory. If you implement concurrent Follower processes and Tweet processes you will get 5 bonus points. Note that the use of fork() for the creation of a new process in phase 1 and 2 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 two Tweets using the same source file Tweet.c/Tweet.cpp and when you create the five Followers using the same source file Studnet.c/Follower.cpp.
15. 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.
16. 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.

L. 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 Followers develop their code on nunki. In case Followers 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. 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.
7. 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.

Good luck.