CSCI 565: Distributed Systems (Fall 2024)

Programming Assignment 1: Developing a Simple Distributed Application

1 Overview

The purpose of this programming project is to learn how to design a simple distributed application using different types of sockets.

2 Project Details

For this project, you are required to implement a client/server message routing system. The system will have two distinct pieces: the client code (which sends and receives "messages" from the server) and the server code (which receives, stores, and forwards "messages" to the clients). You must implement this project using two different types of sockets: TCP and UDP. It is recommended that you read the attached SocketProgramming file.

You will write both the client and server using either C/C++, Java, or Python. You must submit your source code along with appropriate build and README files.

2.1 Clients

The client code will be a command line application that starts, sends or receives a message, and terminates. The client can be started in either send mode or receive mode. When started in send mode, the client will take the contents of a specified text file and forward it to the server. When started in receive mode, the client will contact the server, receive a message, and output it to the console.

From the command line, a client should take 3 or 4 arguments. When sending, the client needs to know the server's location (host name and port), the socket type (e.g. "TCP"), the mode ("send") and the text file containing the message to forward. When receiving, the client needs to know the server's location and the mode. For example:

```
$ p1client ctb60-01.mines.edu 55555 TCP send msgfile.txt
and
$ p1client ctb60-01.mines.edu 55555 TCP receive
```

You do not necessarily need to use the port argument. It is suggested for your convenience to help design and test.

2.2 Server

The server will receive, store, and send messages for a number of clients. This will be a continuously running application that buffers messages in a queue and deals them out in FIFO order. Whenever contacted by a client who is sending a message, the server will append the sent message to its queue. Whenever contacted by a client who is receiving a message, the server will remove the message at the start of the queue (i.e. the one that has been held the longest) and send it to the client.

Your server must be able to be contacted by multiple clients (possibly simultaneously). The message queue that the server maintains is a **single, shared queue**. Thus there may be race conditions between clients if they connect at approximately the same time. Depending on how you design your system, there may be a number of concurrency points where shared data structures are being used or threads interact. The only explicit concurrency requirement is that **any particular message is only forwarded to one client!** It should not be possible for two clients to receive a copy of one message (if only one such message is in the queue). As is generally expected, you should also guarantee that there is no deadlock or corruption of shared data structures.

The server should accept a port number and the socket type as a command line arguments. For example:

```
$ p1server 55555 TCP
```

2.3 Sample Output

Assume that you have a text file named message1.txt that contains a message "This is message 1" and message2.txt that contains a message "This is message 2", and that you have already started your server on ctb60-01.mines.edu. At the client side, your system should output at least the following:

```
$ p1client ctb60-01.mines.edu 55555 TCP send message1.txt
$ p1client ctb60-01.mines.edu 55555 TCP receive
Message received:
This is message 1
$ p1client ctb60-01.mines.edu 55555 TCP receive
Error: No messages
$
$ p1client ctb60-01.mines.edu 55555 TCP send message1.txt
$ p1client ctb60-01.mines.edu 55555 TCP send message2.txt
$ p1client ctb60-01.mines.edu 55555 TCP receive
Message received:
This is message 1
$ p1client ctb60-01.mines.edu 55555 TCP receive
Message received:
This is message 2
```

Note that this is the minimum requirement and the formatting does not need to be exact. Feel free to add more information when sending and receiving, as long as the message received are clear. There is no output requirement at the server-side, but it may be helpful for implementation, debugging, and grading to output information about the queue contents and incoming connections.

3 Project Group

All students should work on this project individually.

4 Test Cases

- We will only test your system with up to 3 simultaneous clients, though there is little in this project that prevents you from handling an arbitrary number of clients. We recommend that you design as though an arbitrary number will connect to your system at any given time.
- Make sure that your program can handle up to 3000 characters in a message. Depending on your design, it may be possible to test your client/server implementations on the same machine. Taking advantage of separate ports will help with this.
- Remember that you can always ask for help with design and clarifications on Ed Discussion!

5 Deliverables

You need to hand in all of your source files, a README file, and a Makefile if you use C/C++. Zip all your files into one file and submit it. Do not submit your object files or executables!

Your README should describe your design, list any assumptions you made, and describe how to test your project in detail. If we cannot figure out how to compile and run your project based on what you have written in the README, we will grade it as if it is broken. In addition to documenting your design in the README, you should also thoroughly comment your code such that with the background information given in the README and the comments written in the code, we can figure out exactly what your code is doing.

If you use C/C++, your Makefile should correctly compile the source files and produce any necessary files. We will be supplying our own text files to grade with, so do not submit your text files. Either before or after compilation (i.e. with the makefile's help), your code should be appropriately separated into separate server and client directories.

Ensure that your code work on Isengard. This is a Linux machine available to all Mines students (more details below).

6 Grading

- README (along with Makefile for C/C++ implementation): 20 points
- Implementation: 80 points

If you have any questions, feel free to post them to the forums on Ed Discussion. We prefer that you post questions to Ed Discussion rather than sending emails because others in the class might also benefit from the answer. However, please do not post any source code there.

7 Resources

The projects for this course will require code that can send messages on a network, and many of them will also require you to use multiple threads. If you have not had experience with multithreaded programming or network programming, you should spend some time to figure that out now to make it easier for you when doing future assignments.

Java To write these programs in Java, you should look into the following classes: Thread, ServerSocket, Socket, FileReader, BufferedReader, InputStream, OutputStream, ObjectInputStream, and ObjectOutputStream. Refer to the Java API for more information about these classes: http://www.oracle.com/technetwork/java/api-141528.html

Also see NIO APIs for nonblocking I/O (managing multiple sockets): http://java.sun.com/j2se/1.4.2/docs/guide/nio/, and examples.

A tutorial on using sockets in Java: http://java.sun.com/docs/books/tutorial/networking/sockets/

C/C++ To write these programs in C/C++,"Beej's Guide to Network Programming" is a great place to start:http://beej.us/guide/bgnet/

For some reference on basics in C++: http://www.cplusplus.com/doc/tutorial/

And in C:

A. D. Marshall's guide to C http://www.cs.cf.ac.uk/Dave/C/CE.html

Steve Holmes' guide to C Programming http://www.strath.ac.uk/IT/Docs/Ccourse/

For spawning new threads, you should look into the fork() system call (actually creates a new process) and the pthreads library.

A quick search will give you plenty of good resources for all of the above as well. Feel free to post questions about any of these topics on Ed Discussion to get help from other students.

8 Hints and Clarification

- We are generally assuming there are no failures and that the server is persistent for this project. You don't need to make the server queue persistent between executions of the server.
- We intended that the separate server and client directories were to be used primarily for the binaries/object files, though it says "code" above. As such, we'll accept any variant of this for the solution (code separated, binaries separated, both separated).
- You should test your clients and servers on Isengard.mines.edu. This can be done through remote access or from the ALAMODE lab located in CTLM B60. The lab computers are typically named as ctb60-01.mines.edu, ctb60-02.mines.edu, etc. You can remotely access them or Isengard directly. Isengard or individual ALAMODE machines can be accessed off campus using a VPN or by using jumpbox (without a VPN). To enter into jumpbox, run:

  ```
  $ ssh jumpbox.mines.edu -l <user-name>
  ```

  where <user-name> is your Mines Multipass user name. It will prompt you for your Multipass password and two-factor authentication. Once in jumpbox, you can access Isengard using:

```
$ ssh isengard -l <user-name>
```

If you are on campus or using a VPN, then you should be able to ssh into directly Isengard without going through jumpbox.