COE 4DN4 Lab2 - v1

A Multi-threaded Song/Photo Sharing Application "McNapster"

Lab 2 Posted: Sat., Jan. 25, 2014
Labs: M,T,W,Th - JHE-234 EOW

TA Supervision & Demo Times (for Labs 1 and 2):
(These labs will be supervised if 2+ groups reserve time on each day)
Wed, Thu - Jan 29, 30
Wed, Thu - Feb 5, 6
Wed, Thu - Feb 12,13, Feb 26,27
Wed, Thu - March 5, 6, March 12,13
(or by appointment)

Lab 3 Report Due: Mon, March 17, 2014 on Avenue-to-Learn (by 10 am)
Early submissions are welcome
(check email for updates on this lab)

Goal

Develop an FTP client-server system for sharing music and photos, based upon the now defunct *Napster* file-sharing system, using a TCP/IP socket connection.

Napster had an immense impact on society: It alone probably accounted for most of the immense growth of Internet traffic in the years 1999-2001. It allowed for the (illegal) sharing of music regardless of copyright issues, and it was eventually shut by the US government and forced into bankruptcy, and its assets were purchased by other companies. Napster prompted others to write a new generation of Internet file-sharing systems that did not have a centralized file server, called 'peer-to-peer' (P2P) systems. These systems include Gnutella and Bit-Torrent. P2P systems are now used to deliver many commercial digital television services over the Internet.

Assumptions

Implement the ftp client and server using TCP sockets. TCP automatically takes care of the reliability, thus enabling you to focus on application details.

You don't need to have to implement any authentication mechanism; i.e., the client does not need to provide a username/password.

You can use binary mode message transfers for all files. You don't have to consider ASCII mode.

Specification

The server process will spawn (clone) a child process (thread) for every client. You can use the sample code for the multi-threaded server discussed in class and in the reference textbook (TCP/IP Sockets in C or Java) as a starting point for this lab.

Server Specification

The ftp server is invoked on the server machine as :

% ftpserver <*port_number*>

<port_number> specifies the port at which the ftp server accepts connection requests.

Optional: You may optionally specify a file-system directory for your searches:

<root_directory> is the directory you specify, whose contents will be made accessible to connecting clients. For instance, the root directory could be a directory in your server's file-system named ***pub*** under which you place all the files to be available for others to see. The directory pub can have 2 subdirectories, such as MUSIC and PHOTOS, and potentially more directories.

You should implement the following commands in your application:

Command	Initiated by	Description
LIST-ALL	Client	List the files and their sizes from the server's current music directory.
READ <*filename*>	Client	The client requests the file specified by *filename* and stores it in it's current working directory.
WRITE <*filename*>	Client	The client writes the file specified by *filename* into the server's appropriate directory.
BYE	Client	Exit from the client after closing the connection server, if one exists.
QUIT	Server	Exit from the server after closing the connection(s) with the clients, if there are any.

We may announce a specific format for these command-messages, so that every team's software should be able to communicate with each other. Please check the web-site for updates.

Client Specification

The client initiates all transactions and sends commands to the server. The client is invoked as follows:

% ftpclient <server_machine> <server_port>

<***server_machine*>** stands for the IP address of the machine hosting the ftp server.

<***server_port*>** refers to the port number on which server is listening.

Error Handling

Identify errors in the middle of a file transfer, ie broken connections or timeouts, and cleanup any intermediate state (ie partly transferred files) before exiting.

Design

You can use one or two connections (i.e., sockets) for each ftp session. If you use 2 sockets, you can separate the control and data exchanges. You should close all sockets created by your application before you exit the program.

Development Guidelines

Create a directory called 4DN4_lab2 with two sub-directories: client and server.
The client file should be called FTPclient (in c or Java) and placed in the client directory
The server file should be called FTPserver (in c or Java) and placed in the server directory

If you use a UNIX/LINUX command-line-interface for gcc, you should have a separate makefile in each subdirectory.

Hints

- Make sure that you do not leave any "zombie" processes in the machine after you run your program. In UNIX, use "**ps -aux | grep <username>**" to check if there is any process whose status is "Z" (indicating a zombie). You can use "**kill -9 <process id>"** to kill them.
- Cleanly terminate all connections in your code before exiting. In UNIX, one way to check for unterminated connections is "**netstat -ap**". If your connections have being left hanging, you should try to fix the problem.

- For a good solution, you may create in your server a data structure that stores your music files and pictures to be shared, of size say 50 elements, where each element includes a string file-name (up to 64 bytes), and the size of the file in bytes. This data structure can store the current songs, and another data structure can store the current photos. When a client requests one of these items, the server process should search the database, and if the desired file is found, then it should open the file through a call to the operating system, and send the file to the client.
- For an alternative but more complex solution, you may call the Operating System to return a data structure containing the list of files in the current directory that your server is working in. You can then parse this directory data structure in your HandleTCPClient code to extract the information you want.
- You may use any C or Java compiler, but preferably you will use NetBeans, since the TAs will have some expertise in the NetBeans IDE.
- We have posted videos on how to use NetBeans on Avenue-to-Learn.
- Please contact the prof for permission to work in C# or C++ or Ruby.

What to hand in:

You may work in teams of 2 students. The working system should be demonstrated to a TA during the laboratory hours, who will record a presentation mark. The TA(s) will ask questions during the demonstration to assess your software.

The lab report should be brief and have these sections:
(1) Objective and brief introduction
(2) Server flow-chart and brief discussion
(3) Client flow-chart and brief discussion
(4) professionally documented C or Java code, referring back to flow-chart when necessary
(5) Experimental results (include brief discussion and screen snap-shots to demonstrate working software)
(6) If you wish to customize your solution, please proceed. Innovation will be rewarded.
(7) Issues / Problems that were encountered & needed to be solved
- enumerate any issues.
(8) Conclusion - brief summary

Please submit an electronic copy of the lab report along with professionally documented software on Avenue-to-Learn. Please organize your work in a folder called 4DN4-Lab2-WX,YZ, where WX,YZ are the initials of your team members. ZIP-compress the folder). Do not compress a folder with a non-conforming file name, and then change the name of the compressed file, since it will expand into the original folder with the non-conforming name, which complicates our managing of numerous electronic submissions. Marks will be awarded for using the proper folder names and the proper submission process.

A Good First Step:

To start off, we have supplied the threaded TCPEchoserver and TCPechoclient that we looked at in class in Lecture 8 (Thurs. Jan 23). This server can receive a GET command from the client, and send the client a pre-selceted file. You should be able to modify this code to implement all the desired functionality.