Project #2: Simple File Sharing

Due: May 1, 2006 11:59pm

Overview

For this assignment you will implement a simple peer-to-peer file sharing program. This will require that you implement both an index server and a peer ("client") program.

Part 1: `FileIndexServer`

The `FileIndexServer` program serves as a central contact point for the peers in the file sharing system. A peer will contact the index server to register its list of files, to search for a file, and to remove its list of files from the system.

This program takes a port as its command-line argument. It must then create a socket on that port and wait for incoming connections. Each connection will be serviced by a separate thread.

The server must support the following requests.

join A peer, when first starting, will join the file sharing network by sending the index server a join request. This request will include the list of files that the peer is sharing. The server will need to update its index appropriately.

leave When a peer is terminating, it will send a leave request to the server. The server must remove the peer's files from its index.

search A peer will send a search request to the index server to get a listing of other peers offering a file. For simplicity, we will require exact matches in file name.

add A peer will send the index server an add request when it has updated its set of shared files. The server must add the new file to its index.

Message Formats

For simplicity, and to ease testing and debugging, the messages between the index server and the peers will be in plain text.

join A join message takes the form

```
JOIN\n
client name\n
port number\n
file name 1\n
...
file name N\n
```

The "client name" is simply a string used to label the client. The server does not respond to this request. Duplicate client names are allowed as clients will be identified by their ip address/port pair internally. The server must maintain a mapping from each of the client's files to its ip address and port (this is the port on which the client will listen for file requests, not the port to which this socket is connected).

leave A leave message takes the form

```
LEAVE\n
port number\n
```

The client's files are removed from the index. The server does not reply to this request.

search A search message takes the form

```
SEARCH\n
file name\n
```

The server searches the index for peers providing the requested file. The server replies with the list of peers in a message formatted as follows

```
client name 1\n
ip address 1\n
port 1\n
...
client name N\n
ip address N\n
port N\n
```

If there are no peers sharing the requested file, then the reply will be empty.

add An add message takes the form

```
ADD\n
port number\n
file name\n
```

The server updates the index to include this file, but does not respond to the client.

For simplicity, there is no means to remove a file from the index, except when a peer leaves the system.

Part 2: FileSharePeer: Index Server Communication

The `FileSharePeer` program implements the peers in the file sharing network. There are two aspects to this program. The first aspect is communication with the index server (joining the system, leaving the system, and issuing a search). The second aspect is communication between peers. This part of the assignment requires that you implement the first aspect (communication with the index server).

The `FileSharePeer` program takes four command-line arguments: the client name (simply an identification string), the port on which the client will listen for file download requests, the index server name (ip address), and the port on which the index server is listening. For instance,

```
% java FileSharePeer "Bob's Box" 3000 falcon.csc.calpoly.edu 3010
```

The peer program contacts the server to join the file sharing system. It does so by sending a `JOIN` request (detailed above) with a list of the files that this peer will share. This list of files comes from the contents of the `SHARE` subdirectory relative to the directory in which the program is running (only normal files are shared, not subdirectories).

The peer then repeatedly prompts the user for a file name for which to search the system. Once given, the peer issues a `SEARCH` request, prints the results, and allows the user to select from which peer to copy the file (the communication with a peer is discussed in the next part).

If the user opts to leave the system (to cause the peer to quit), then the peer will send a `LEAVE` request to the index server and exit.

As sample session might work as follows.

```
% java FileSharePeer "Bob's Box" 3000 falcon.csc.calpoly.edu 3010
Enter the name of the file for which to search (EOF to quit): FinalExam.pdf

File found on --
1) Professor Plum's Palmtop
2) Dr. Dork's Desktop

Download from which: 2
Complete
Enter the name of the file for which to search (EOF to quit): ^D
```

Part 3: FileSharePeer: Peer-to-Peer Communication

Once a user selects from which peer to download the file, it will contact that peer directly by sending a message that contains only the name of the file to download. The peer, if available, creates a new thread to handle the request. If the file exists, then the serving peer replies by sending a message that consists of `OK\n` followed by the file contents. If the file does not exist, then the serving peer replies with message containing only `BAD\n`.

The receiving peer, on a successful download, writes the file into its `SHARE` directory and sends an `ADD` message to the server. If the file is not successfully downloaded, then the peer notifies the user. In either case, it is sufficient for the program to prompt the user for another file for which to search.

Note

You can test your server without a client. To do so, you can connect to the server using telnet. This program will send whatever you type to the server and print whatever the server's response is. Use this tool to test your server before developing your client. You'll want to make sure that one side of the communication channel is functioning properly before you attempt to implement the other (debugging both at once is less enjoyable than it may sound).

Handin

Submit your source files using handin on hornet/falcon. To do this, execute

handin akeen proj2 <your files here>

Grading will be divided as follows.

Part	Percentage
1	40
2	30
3	30