CSE30264
Fall 2014
Project 3c (p2p talk)
P2P Server / Client

Total Points	**120**
Goal	**Program a TCP client / server**
	Program multi-threaded TCP client/server
Assigned	**November 18, 2014**
Due	**December 11, 2014**
Grouping	**To be done in a group**

Task: P2PChat Application Programming

The idea of this project is to implement an online chat program. This program will allow users to carry on a chat session with other users in a group. The chat service as a peer-to-peer service. Thus, your application will be both a client and a server. When you want to join a chat group you will do so by connecting to some other member of the group. Once you are a member of a group you must allow connections from other members.

In order to discover other group members, you will use a rendezvous server that your client contacts in order to receive a list of current groups, and members for the group you are interested in. As a peer-to-peer architecture, you will be responsible for forwarding messages you receive from one peer to other peers in your network.

Rendezvous server

The first thing that you need to implement is a rendezvous server. This server maintains a list of active chat groups, and the names, IP addresses and ports of the members connected to each chat group. When a user wants to join a chat group, they "register" with the rendezvous server, and obtain a list of current chat sessions (chat group name, active users, user address/port information). The user then joins a group (which adds their information to the table).

User Interface (UI)

The UI will allow users to connect to the chat service, identify themselves as chat users, join a group and then send messages. The actual UI design is completely up to you. The command line should accept the hostname and port number of the rendezvous server as it's arguments. For instance, the invocation:

p2pchat student01.cse.nd.edu:9421

This invokes the chat program, contacting to the rendevous server on student01 using UDP port 9421, This should be your normal invocation for using and testing this assignment. Once the application is running, it prompts the user for input. You should support the following features:

- **list** - get a list of groups from the rendezvous server.

- **join groupname username** - join the group "groupname" as the user "username". Restrict both of these to single word names with printable characters.
- **sending and receiving data** Text that is typed by the user should be echoed on the screen and sent to all connected chat programs. Text that is received should be displayed for the user.
- **leave** - leave the group, closing all group connections. The user could optionally join another group.
- **quit** - exit the program, closing all group connections.

The Protocols

You need to define two protocols in order to make this work. In many ways, the protocol proposed here is just a start with many possible enhancements left to be made. As part of this assignment you will be evolving the details. In particular, you may decide that some additional messages are necessary to get this to work as you like. At a minimum, you will most certainly need to further clarify the semantics and behaviors of a correct implementation of your service.

First, what is the interface between your client and the rendezvous server? When the user joins a group, before you indicate the successful join, you must send a UDP query to the rendezvous server and request a list of group members.

Join message from client to rendezvous server:

- **Message Type** - 1 byte - value is ASCII character "J" for Join
- **Group Name** - Variable length character sequence terminated by ":"
  - ASCII name of group to join
- **User Name** - Variable length character sequence terminated by ":"
  - ASCII name of user joining the group

Response message from rendezvous server to client:

- **Message Type** - 1 byte - value is ASCII character "S" for Success or "F" for failure
- **List of group members** and their IP addresses
  represented as a sequence of Username:Address: pairs
- **User Name** - Variable length character sequence terminated by ":"
  - ASCII name of user in group
- **IP Address** - A 32-bit integer in network byte order
  Note that the ":" character appears after the address integer.
- The message terminates with "::" to mark the last group member.
  You should see yourself in this list.

The join message will cause a group to be created if you specify a group that does not exist. In this case you will be listed as the only member of this group.

What about getting the list of current groups?

List request message from client to rendezvous server:

- Message Type - 1 byte - value is ASCII character "L" for List

List response message from rendezvous server to client:

- Message Type - 1 byte - value is ASCII character "G" for Groups.

- List of Group Names, separated by ":"
  Group Name - Variable length character sequence terminated by ":"
  - ASCII name of group
- The message terminates with "::" to mark the last group name.

Once you have a list, you must try to join the group network by connecting to one of these group members. You will connect to them using the same port, 9421. You will simply go through this list of IP addresses, trying to connect to each one. Once you find a successful connection, you don't have to try any more nodes. This node will be your link to the chat group.

For simplicity, we will define just one message type for the peer-to-peer protocol. This message contains the text that some user has typed. The format is as follows:

- Message Type - 1 byte - value is ASCII character "T" for Text
- Group Name - Variable length character sequence terminated by ":"
  - Group name for which this message is intended.
- User Name - Variable length character sequence terminated by ":"
  - Original user that sent this message
- Message ID - A 32-bit integer in network byte order
  - This is a unique identifier for messages from this user.
- Message Length - A 32-bit integer in network byte order
  - The number of bytes of data in the user's message.
- User Message - The message data - Note that there is no termination character.
  You must read only the indicated number of bytes or you will run into the next message!

So, what do you do when you receive such a message?

First, you check the group name to see if this is the group you are in. If not, drop the message and close this socket. Apparently someone connected to you with outdated group information.

If this is your group, you should check the username/messageid values to make sure this is a new message you haven't seen before. If you've already seen this message, do nothing.

If this is a new message for your group, you have two jobs. First, display the message for the user by printing it with the username. Next, you must forward this message to any other users that are currently connected to you.

The rendezvous server will maintain the current list of group members but we did not define a "leave group" message. Instead, the rendezvous server will drop any members of the group it has not heard from in the last 2 minutes. You will need to periodically resend your join message to the server while you are actively participating in the group.

Sample Chat Session

Here is a sample successful chat session, with a simple command line interface, starting and ending with the shell prompt. You should come up with a nicer interface, this is just an example.

[curt@dopey ~]$ p2pchat student01.cse.nd.edu:9421
P2PChat>list
NDfootball

NDBasketball
CSE30264
P2PChat>join CSE30264 spongebob
...connecting
CSE30264>send
Send>Hello!
CSE30264>
From: Wayne> Welcome!
From: Garth> Arrrrgh.
CSE30264>leave
...closing group
P2PChat>quit
[curt@dopey ~]$

Notes

Your client/server will be handling several different sockets. First, to the rendezvous server. When the user issues a join request, you must send a message to the server to get the group list. Second, you will have a connection you open to at least one other peer. (Unless you are the first group member.) And finally, you must be prepared for other peers to contact you! You must have a socket bound to port 9421 where you periodically check for new clients and prepare to receive messages from them.

You will need to maintain a list of other peers you are currently connected to via open sockets. You will periodically cycle through this list of sockets to see if there are any new messages for you to process. For new messages, you will forward them to all of the other open sockets except of course for the one on which it arrived.

You should be able to test your program against itself. Simply run the same program, connecting to a group as a different user.

Define your own features

A final component of this project (10%) will be the definition of your own extended features. These features could range from a client implementation in a different language (C#, Java, Python, Perl) to features such as the ability to send multimedia (video/audio), use of images as user identifiers, a GUI (maybe using the ncurses library), causing the window to flash, or a bell to ring when a new message arrives, etc.

Submission

Submit a tar / gzip of your entire project package to one of the drop boxes for your group in the Project3c directory. The archive must include the following:

* Working client code in a client subdirectory
    - o Makefile for the client
    - o Source file for the client
    - o Appropriate comments with team member names
* Working server code in a server subdirectory
    - o Makefile for the server
    - o Source file for the server

    o Appropriate comments with team member names
  * README file
    o Team member names
    o Listing of the included file names in the archive
    o Instructions on how to launch you r application.

Grading Rubric
Total: 120 points

- Integrated client/server code 60 points
  - Included Makefile and code compiles without errors: 10
  - Code follows argument convention: 10
  - Code executes the scenarios as mentioned in the evaluation: 30
  - Code provides reasonable error checking: 5
  - Reasonable code performance: 3
  - Code with appropriate comments: 2

- Rendezvous Server code subtotal: 40
  - Included Makefile and code compiles without errors: 5
  - Code follows argument convention: 5
  - Rendezvous server operates as described above: 20
  - Code provides reasonable error checking: 5
  - Reasonable code performance: 3
  - Code with appropriate comments: 2

- Extended features subtotal: 12
  - Define your own extended features: 12 pts

- Overall subtotal: 8
  - Code is submitted in requested form: 5