Project 2

Due May 8, 2016 at 11:55 PM

This project specification is subject to change. You will be working alone for this project, though your software project is expected to work properly with other students' submissions. You must implement this project in C/C++. You will be implementing a peer-to-peer instant messaging client. The messaging client will use UDP to discover other clients, but will use TCP connections for transferring messages. The messaging client must be able to maintain multiple simultaneous connections with other clients, and maintain a list of the clients available on the network. Submit a gzipped tar file of your source code, makefile, and a readme file.

The application syntax is: p2pim [options]

-u username	Specifies the username to use for communications, if not specified it is assumed to be the username from the environmental variable USER.
-up port	Specifies the UDP port the peer-to-peer client receives its requests from, if not specified it is assumed to be 50550.
-tp port	Specifies the TCP port the peer-to-peer client receives its incoming connections from, if not specified it is assumed to be 50551.
-dt timeout	Specifies the UDP discovery initial timeout in seconds for the peer-to-peer client to receive replies to discovery broadcast, if not specified it is assumed to be 5 seconds.
-dm timeout	Specifies the UDP discovery maximum timeout in seconds for the peer-to-peer client to receive replies to discovery broadcast, if not specified it is assumed to be 60 seconds.
-pp host:port	Specifies a host/port in which to send a unicast UDP discovery message. This message is only sent if the host is not on the client's local subnet. More than one host may be specified on the command line.

UDP Client Discovery

The instant messaging client starts up without any knowledge of currently available clients, but may have hints of previous clients. In order to discover other clients that are available on the local subnet, the instant messaging client broadcasts discovery messages over UDP. Online clients that receive this discovery message, respond with a unicast message to the client that sent the discovery broadcast. Discovery messages may also be sent directly to another client (if specified with –pp) in order to connect to clients on another subnet; however the contents of the message will remain the same. Clients that do not discover any other clients within the current timeout will send a discovery message after the elapsed time. The current timeout will double each time no reply messages are received, until it reaches the maximum timeout. The current timeout is reset to the initial value when a client transitions from one to zero discovered clients. When a client application is closing normally, it should broadcast a closing message. See Figure 1 for a state diagram of the transitions.

Discovery Message:

Size	4				2	2	2	m	n
Description	Signature				Type	UDP Port	TCP Port	Hostname	Username
Value	P	2	P	I	0001	XXXX	XXXX	HOSTNAME\0	USER\0

Reply Message:

Size	4				2	2	2	m	n
Description	Signature				Type	UDP Port	TCP Port	Hostname	Username
Value	P	2	P	I	0002	XXXX	XXXX	HOSTNAME\0	USER\0

Closing Message:

Size	4				2	2	2	m	n
Description	Signature				Type	UDP Port	TCP Port	Hostname	Username
Value	P	2	P	I	0003	XXXX	XXXX	HOSTNAME\0	USER\0

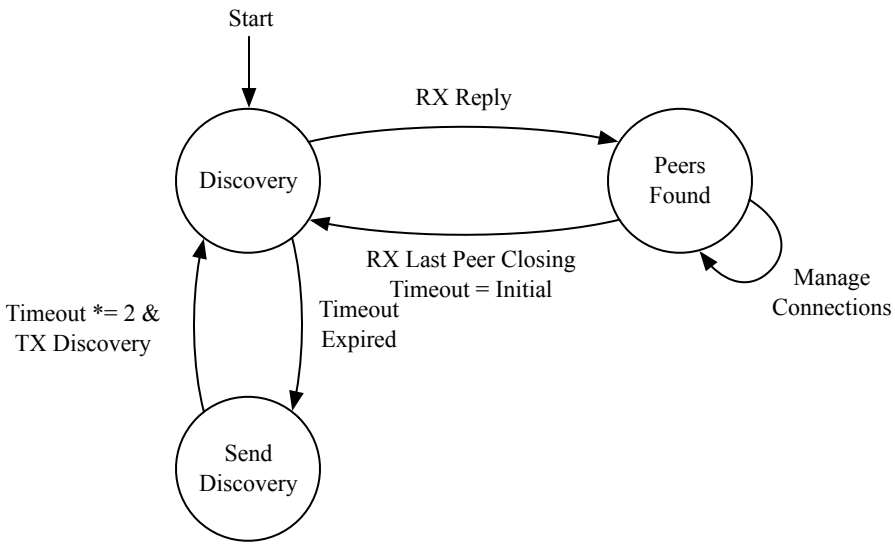

Figure 1. Discovery state machine

TCP Client Messaging

The messages between clients are sent via a TCP connection. A client will connect to another client on the port specified in the discovery or reply messages. Clients should accept incoming messages initially, and should reply with a User Unavailable Message if communication is not desired. The TCP messaging is also capable of requesting and sending user lists; this allows the discovery of clients in other subnets without prior knowledge. When a user wishes to disconnect from a peer with an accepted communication path, the peer should send a Discontinue Communication Message.

Establish Communication Message:

Size	4				2	n
Description	Signature				Type	Username
Value	P	2	P	I	0004	USER\0

Accept Communication Message:

Size	4				2
Description	Signature				Type
Value	P	2	P	I	0005

User Unavailable Message:

Size	4				2
Description	Signature				Type
Value	P	2	P	I	0006

Request User List Message:

Size	4				2
Description	Signature				Type
Value	P	2	P	I	0007

User List Reply Message:

Size	4				2	4	Entry Count
Description	Signature				Type	Entry Count	Entries
Value	P	2	P	I	0008	XXXXXXX	See below

List Entry:

Size	4	2	m	2	n
Description	Entry Number	UDP Port	Hostname	TCP Port	Username
Value	XXXXXXXX	XXXX	HOSTNAME\0	XXXX	USER\0

Data Message:

Size	4				2	n
Description	Signature				Type	Message
Value	P	2	P	I	0009	Message Contents\0

Discontinue Communication Message:

Size	4				2
Description	Signature				Type
Value	P	2	P	I	000A

User Interface

The user interface must maintain a list of the available clients and have a mechanism to display them (either upon request or inside a window). The user interface must also have a mechanism to establish connections and send messages to specific clients. There must be a way to distinguish messages that are received from other clients (whether it is as simple as "username: " before the message or an entirely separate window is entirely up to you). The user interface must have a mechanism to discontinue communication with an established client.

As an absolute minimum, the interface should be a text based program with stdin set to non-canonical mode. An example of non-canonical mode can be found in the resources under Source/noncanmode.c.

You can use the ncurses panel API to maintain your "windows". Documentation on ncurses is available at http://tldp.org with specific information on the panel API at http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/panels.html.

You can use the GTK+ graphics library if you choose that you want to make a graphical interface. A GTK+ tutorial can be found at https://developer.gnome.org/gtk-tutorial/stable/book1.html. An example GTK+ 2.0 based UDP server can be found in the resources under Source/gtkudpserver.c. The reference manuals for GTK+ 2.0 and 3.0 can be found at https://developer.gnome.org/gtk2/ and https://developer.gnome.org/gtk3/ respectively. Both versions appear to be installed on the CSIF, but the example provided is for GTK+ 2.0.

Useful Information

Linking Panels Library

In order to link in the ncurses and panel library correctly, you must add -lpanel –lncurses to your final linking. For example gcc <program files> -lpanel –lncurses.

Linking GTK+ Library

In order to compile and link in the GTK+ library correctly, you must add `pkg-config --cflags --libs gtk+-2.0` to your compiling and linking for GTK+ 2.0 and `pkg-config --cflags --libs gtk+-3.0` for GTK+ 3.0. For example:
gcc <program files> `pkg-config --cflags --libs gtk+-2.0`

Get Username

The username is typically specified in the as an environmental variable "USER". Calling getenv("USER") should return the string to the username.

Get Hostname and IP

An example of getting the hostname and IP of the local machine can be found in the resources under Source/gethostinfo.c.

Extra Credit

Extra credit will be given only to submissions that have fully completed the main project. Extra credit will be given to applications that can share the UDP port, and that can automatically find an open TCP port if default port is currently used by another application.