Network Programming – Project 1
Multithreaded Chat Using TCP

1) Overview

In this lab assignment, we will learn to use the TCP/IP networking interface. Using TCP stream sockets, you will be implementing a client/server chat application. The overall architecture that you should have in mind for this is shown in **Figure 1**.

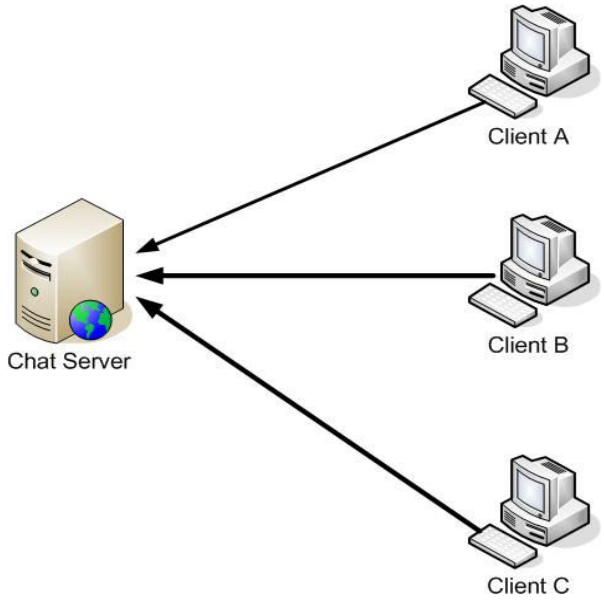

Figure 1

The server accepts connection requests from clients. When a client connects to the server, it is assigned an identification number (**starting from 1**). This identifier will be used to switch messages between the various clients. In addition, to help keep track of all the connections, clients can register a **name** (**31 characters**) with the server. Clients can send messages to other clients by sending a message to the appropriate identifier. The server is responsible for handling the various requests from the user including: passing messages to the appropriate destination, maintaining/closing connections, maintaining the ID Number/Name database, and distributing the ID Number/Name database (when requested). As part of this assignment you must determine what semantics will be used to deliver messages to the clients and when the clients actually see the message. Specifically, each client can send multiple messages to other clients. How these messages are delivered to the client is up to you.

2) Protocol Specification (see net_msg.h)

This section provides the specification of the messages that your server must be able to handle. The valid commands between client and server are shown in **Table 1**:

Command	to/from server	op-code	Description
sv_cnt	from	1	Returns a client's ID number on connection setup
sv_list	from	2	Returns list of ID number/Name pairs
sv_add	from	3	Add a single client to the buddy list.
sv_full	from	4	Server is Full
sv_remove	from	5	Someone disconnected so remove them from the combo box / buddy list.
sv_cl_msg	to/fom	6	Chat Message
cl_reg	to	7	Client Registers a name
cl_get	to	8	Request server to return all ID number/Name pairs

Table 1: Message Types

NOTE: The size of the packet **length (2 bytes)** should be included in the total packet size. For example, if we have a chat message like "Hello", "Hello" is 6 bytes (including the NULL). The total size of the packet would be 8 bytes. We add 2 bytes to account for the prefix (**len**). So, 8 bytes gets sent across the network.

Each client should be able to handle the messages shown in **Figure 2** from the server:

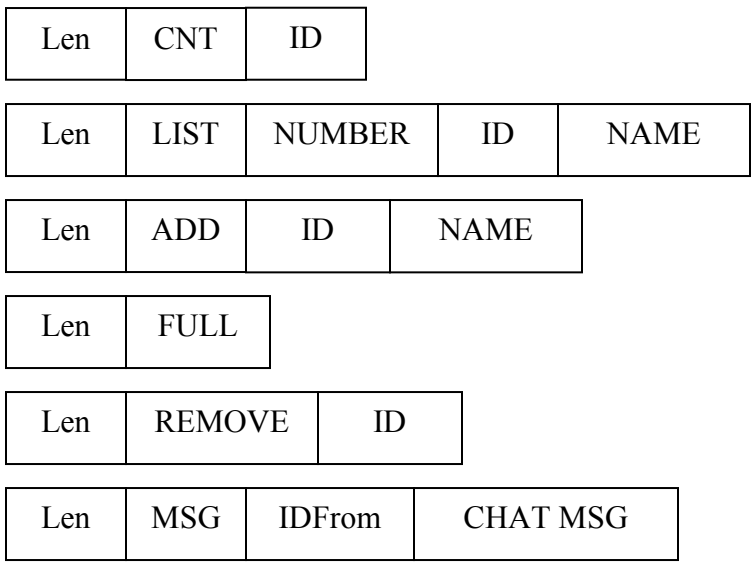

Figure 2: Messages from the server to the clients

CNT - (opcode - 1) Sent to the new client that has just connected to the server.
Len – 2 byte length of the entire data segment.
sv_cnt – 1 byte opcode
ID = 1 byte version of the ID assigned

LIST - (opcode - 2) A list of the ID/server pairs from the server.
Len – 2 byte length of the entire data segment.
sv_list - 1 byte opcode
NUMBER = 1 byte version of the total number of ID/NAME pairs
ID = 1 byte version of the ID of the sender/receiver
NAME = 30 byte (max) character name

ADD – (opcode 3) Adds a single client to the buddy list.
Len – 2 byte length of the entire data segment.
sv_add - 1 byte opcode
ID = 1 byte version of the ID of the buddy
NAME = 30 byte (max) character name

FULL – (opcode 4) Server is Full
Len – 2 byte length of the entire data segment.
sv_full - 1 byte opcode

REMOVE – (opcode 5) Remove someone from the combo box and buddylist.
Len – 2 byte length of the entire data segment.
sv_remove - 1 byte opcode
ID – 1 byte Id of the person to remove

MSG - (opcode - 6) A chat message.
Len – 2 byte length of the entire data segment.
sv_cl_msg - 1 byte opcode
IDFrom = 1 byte version of the ID of the sender
DATA = the message contents

The server should be able to handle the following messages shown in **Figure 3**:

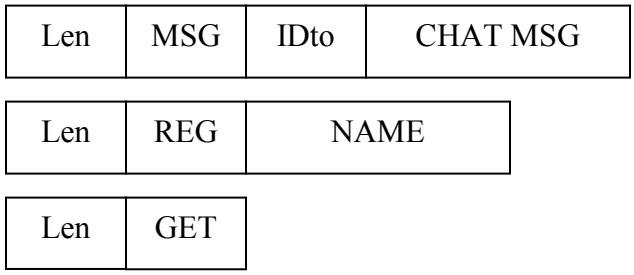

Figure 3: Messages from the client to the server

MSG - (opcode - 6) Sends a chat message to the server for delivery to another client
Len – 2 byte length of the entire data segment.
sv_cl_msg - 1 byte opcode
IDto - 1 byte version of the ID of the receiver
DATA – The chat message.

REG – (opcode 7) Registers a name with the server
Len – 2 byte length of the entire data segment.
cl_reg - 1 byte opcode
NAME – The user's name.

GET - (opcode - 8) Requests a list of the ID/name pairs to be returned to the client
Len – 2 byte length of the entire data segment.
cl_get - 1 byte opcode

Client/Server Interaction:

1. Client initiates connection (connect function).
2. The server should respond with the **sv_cnt** message containing the ID number for the client.
3. Once the ID is received, the client then registers their name on the server (**cl_reg**).
4. Client also sends the **cl_get** to the server.
5. Upon receipt of a **cl_get** command, the server should return a **sv_list** message that contains all the ID/NAME pairs
6. Once a client receives the initial **sv_list** message, if anyone new connects, the server should send an **sv_add** to all (excluding the new client) clients to add the newly connected client to the buddy list.
7. Upon the receipt of a **sv_cl_msg** command, the server should forward the message to the appropriate client based on the ID number.
8. Upon client disconnect, the server should close the socket and clear the ID/NAME pairs from its tables.
9. You should allow the client to send a message to any one of the clients connected to the server (private message).
10. If a client sends a private message, that message is not sent back to the sending client. The client should print that message locally. If a client sends a private message to himself / herself, the message will be sent back from the server but, the client should print the message locally as well.
11. If the server shuts down, all clients should be informed.
12. If a client disconnects, all other clients should be informed.
13. The chat messages should be displayed by the sender and receiver.
14. If the server has too many clients, the server should respond with **sv_full** and shutdown that connection.
15. You should be sensitive not to allow the client or server to arbitrarily delay any message.

3) Program Notes

1. Winsock is already initialized and shutdown for you.
2. The functions **SV_Shutdown**, **CL_Shutdown, CL_Init, SV_Init and CL_SendChatMessage** are called for you but, you must write the body of the functions. See the TODO's in the function definitions.
3. **Shared thread data should be protected with critical section objects.**
4. **The buffer class written in lab 1-3 should be used to read and write network data.**

4) What to Turn In

1. zip file (lastname.firstname.Chat.zip) containing all source **excluding** release, vtune, debug folders and .ncb files.
2. Well commented / formatted source code.

3. Code should be turned in with zero errors and zero warnings during compile time (better do a Rebuild Solution).

5) Check List

General Programming

Compiler Warnings (subtract 5 pts per warning) _____
Displays who Connected / Disconnected: _____
Handles partial receives: _____
Threads created for the client and server: _____
Global Data Protected with critical sections: _____
Threads are correctly shutdown (handles closed etc): _____
Handles Multiple Clients (select function): _____
Sends ID # to client upon connect: _____
Detects Max Clients: _____
Routs Messages to correct destination (public / private message): _____
Detects Client disconnect (graceful and abnormal shutdown): _____
Sends Updated client list to all connected clients: _____
Connects to server: _____
Receives ID # from server: _____
Registers client name with the server: _____
Detects Server disconnect (graceful and abnormal shutdown): _____
Displays client list from the server: _____
Displays the name of who sent a chat message: _____
Displays chat messages sent and received: _____
Sends / Receives private message: _____
Displays server full: _____