Network Programming Assignment #1

1. Network Programming Assignment - 100 points

In this assignment you will implement the client and server for a simple chat service. The goal of the assignment is to introduce you to socket programming, reading specifications, and interoperability testing.

Assignments must be written in C or C++, and be compiled and tested in a Linux environment. Because the goal of the exercise is to understand the system calls to the socket layer you are prohibited from using any socket wrapper libraries; however, you may use libraries for simple data-structures. It is acceptable to use the Unix Network Programming, Vol. 1, 3rd Edition wrappers for the basic networking function calls.

Protocol Specification

1. Overview

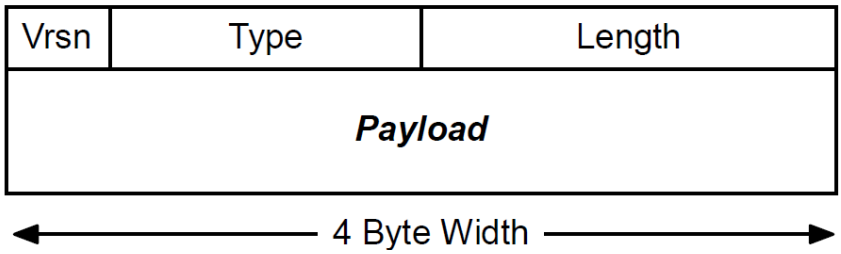
The Simple Broadcast Chat Protocol (SBCP) is a protocol that allows clients to join and leave a global chat session, view members of the session, and send and receive messages. An instance of the server provides a single “chat room”, which can only handle a finite number of clients. Clients must explicitly JOIN the session. A client will receive a list of the connected members of the chat session once they complete the JOIN transaction. Clients will use SEND messages to carry chat text, and will receive chat text from the server using the FWD message. Clients may exit unceremoniously at any time during the chat session. The server should detect this, cleanup the resources allocated to that client and notify the other clients. Additionally, the client program will be able to detect idle clients and notify the server.

Some of the above mentioned features are bonus features, covered in the last part of the homework.

2. Message Format

All SBCP messages will share a common header format. The format for an SBCP message is:

SBCP Message



SBCP Header Fields

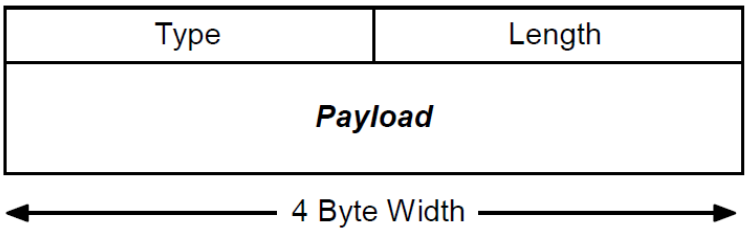
Field	Size	Description
Vrsn	9 bits	protocol version is 3
Type	7 bits	indicates the SBCP message type
Length	2 bytes	indicates the length of the SBCP message
Payload	0 or more bytes	contains zero or more SBCP attributes

SBCP Mandatory Header Types

NAME	Type	Description
JOIN	2	client sends to server to join the chat session
SEND	4	client sends to server, contains chat text
FWD	3	server sends to clients, contains chat text

The format for the SBCP attribute is:

SBCP Attribute



SBCP Attribute Fields

Field	Size	Description
Type	2 bytes	indicates SBCP Attribute type
Length	2 bytes	indicates length of the SBCP attribute
Payload	0 or more bytes	contains the attribute payload

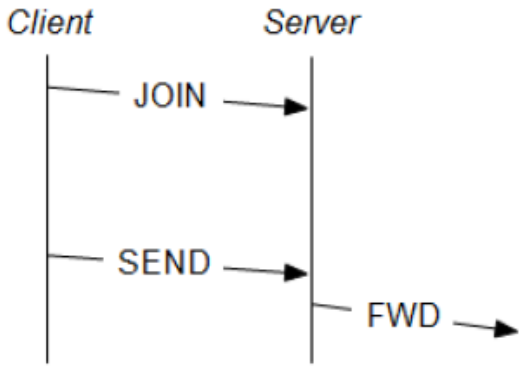
SBCP Attribute Payloads

Name	Attr. Type	Size	Description
Username	2	1-16 bytes	nickname client is using for chat
Message	4	1-512 bytes	actual chat text
Reason	1	1-32 bytes	text indicating reason of failure
Client Count	3	2 bytes	number of clients in chat session

3. Message Sequences

The following is a very brief sequence diagram showing a client JOIN a chat, SEND a message, which is broadcast (FWD) to any other client participants.

Mandatory Sequence



Message Attribute Table

SBCP	Sender	Receiver	Attributes
JOIN	Client	Server	username
SEND	Client	server	message
FWD	Server	client	message, username

4. Client Operations

#	Requirement
1	The client should connect to the server using the IP and port supplied on the command line. <./client username server_ip server_port>
2	The client should initiate a JOIN with the server using the username supplied on the command line.
3	The client should display a basic prompt to the operator for displaying received FWD messages, and typed message that are sent (SEND).
4	The client will need to use I/O multiplexing (select) to handle both sending and receiving of messages.
5	The client should discard any message that is not understood.

5. Server Operations

#	Requirement
1	The server should start on the IP and port supplied on the command line. <./server server_ip server_port max_clients>
2	A SEND received by the server will cause a copy of the MESSAGE text to be sent in FWD to all clients except the original sender.
3	A server may only accept JOIN SBCP messages from unknown clients. The server will not allow multiple clients to use the same username.
4	Clients may leave the chat session unceremoniously. The server should cleanup its resources (close socket, make the username available).
5	The server should discard any messages that are not understood.
6	The implementation can be an iterative server. A distinct TCP connection is used for each client-server transaction.

Bonus

For the assignment you must implement the Mandatory features; however, you may implement the IPv4/6 Bonus and one of the other two bonus features for extra credit. The bonus features that may be implemented include:

IPv4 and IPv6 (10 points)

Write your code so that it will work with either IPv4 or IPv6 networks. You will not be able to test the IPv6 functionality since the ECE Linux machines do not have IPv6 turned on. Nevertheless, we should all be writing network code today that works for both IPv6 and IPv4.

Feature 1 (20 points)

New message types: ACK, NAK, ONLINE, and OFFLINE. The server uses the

ACK message to an explicit confirmation of the client's JOIN. The ACK includes a single Client Count attribute and zero or more Username attributes (Alternatively, the ACK may include a message attribute containing the client count and the list of clients as a string to be displayed at the client side. In this case assume that all the usernames will fit into a single message.). The server may limit the number of active clients and refuse a JOIN with a NAK. The NAK includes a Reason attribute. Finally, the server can indicate when clients enter and leave the chat session through the ONLINE/OFFLINE messages. These messages contain a single username attribute indicating the client which is entering/leaving the chat.

Feature 2 (10 points)

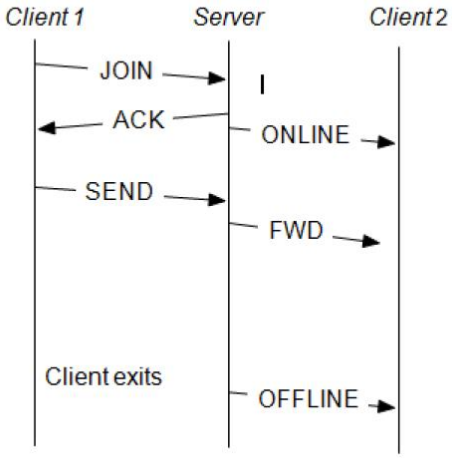
The client should keep track of the time for which a user has not used the chat session. If this interval is more than 10 seconds (fixed interval for this homework), the client should send an IDLE message to the server. The server should in turn send an IDLE message to all other clients with the single username attribute indicating the client which is idle.

In summary, a client can be in one of three states: Online, Offline or Idle. The capacity of the chat-room is the total of Online and Idle clients. When a client goes offline, the server is responsible for cleaning up the resources.

SBCP Bonus Header Types

NAME	Type	Description
ACK	7	server sends to client to confirm the JOIN request
NAK	5	server sends to client to reject a request (JOIN, etc)
ONLINE	8	server sends to client indicating arrival of a chat participant
OFFLINE	6	server sends to client indicating departure of a chat participant
IDLE	9	Client sends to server indicating that it has been idle. Server sends to clients indicating the username which is idle.

Bonus Message Sequence



SBCP	Sender	Receiver	Attributes
ACK	server	Client	client count, username(s)
NAK	server	Client	Reason
ONLINE	*server*	*Client*	*username*
OFFLINE	*server*	*Client*	*username*
IDLE	*server*	*Client*	*username*
IDLE	*client*	*Serve*	*<none>*

1. The client should display the client count and username list of an ACK to its operator.
2. The server should acknowledge all well-formed JOIN requests with an ACK. If the request would cause the server to exceed its client limit then it should send a NAK with an appropriate reason.
3. The ACK should contain a client count attribute, and a list of username attributes. The client count should be inclusive of the requestor, while the username list should be exclusive.
4. Upon successful JOIN the server should send an ONLINE to its other clients and when a client gets disconnected, the server should send OFFLINE to its remaining clients.
5. When a client has not sent a chat message for 10 seconds, it is said to be idle. The Client should send an IDLE message without any attributes.
6. When the server receives the IDLE message, it should form another IDLE message with a username attribute. The username should be that of the idle client.
7. When a Client receives an IDLE message, it should display the username in the message attribute to stdout stating that “<username> is idle”.

Notes

1. The best way of learning new system calls is to build very small programs that demonstrate simple things. Unless you are familiar with the subject matter, you should experiment at first.
2. Once you are ready to begin you should start with a conceptual model/ architecture of your client and server? What is your basic data model? What are the major decision points in the code? You should sketch these thoughts out on paper before you begin (it is also required with submission of the project).
3. Test your code as you write it (unit testing is a great thing).
4. Coding style is very important. People, including yourself, are more likely to understand your code if it follows a style that is simple and self consistent. We will not enforce any specific style; however, you should spend a little time on deciding on a style and sticking with it. Two good references are: http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml, and http://lxr.linux.no/#linux+v2.6.35/Documentation/CodingStyle. What you pick is not as important as being consistent through your code.
5. A makefile is a standard tool for any software project. You will need to create one to build and clean your project. There are many online resources that can get you started.
6. A great way to do random testing is to partner with another student and perform interoperability testing. Your client should work with their server, and their server should work with your client. This is guaranteed to reveal implementation bugs, as well as possible design defects.
7. Come to one of the Recitation sections and/or Office hours to make sure you understand the assignment or are having trouble making progress.
8. You must comment your code.
9. Be sure to check all the error returns from the network functions and to check all buffer writes to avoid buffer overruns. You don't want to be the network programmer responsible for the next "Heartbleed" bug.
10. Servers should not allow multiple clients with the same username to join a chat session.
11. Once a client exits, its username must be reusable by any new client.
12. All submissions will go through Stanford Moss to detect plagiarism.

Submission Guidelines

1. The network programming assignments must be turned in by 5:00 pm on the date due.
2. All programming assignments must include: makefile, README, and the code.
3. The README should contain a description of your code: architecture, usage, and errata, etc.
4. Make sure all binaries and object code have been cleaned from your project before you submit.
5. Your project must compile on a standard linux development system. Your

code will be graded on a linux testbed.

6. Explanation on the submission tool and procedure will be supplied shortly.