Networks Lab - Assignment 8 Report

Group 27
18CS10057 - Yarlagadda Srihas
18CS30047 - Somnath Jena

Code Structure:

The application contains two source files:

- ➔ **chat_server.c:** Primarily contains functions that set up the listening server and accept new connections. The `main()` function is also a part of this file, which contains a loop running `select()` in order to multiplex the various file descriptors.
- ➔ **chat_client.c:** Contains functions that deal with sending and receiving messages.

Header file:

- ➔ **chat.h:** Includes function prototypes that are shared by the two files, as well as constant definitions. A `struct user_entry` is defined, which corresponds to an entry in the user_info table. The table itself is described by another `struct user_info`.
- ➔ `struct user_entry`: Contains the IP address, port, username, corresponding `sockaddr_in`, client socket (-1 if no connection present) and the next timeout value.
- ➔ `struct user_info`: Contains an array of `struct user_entry`, sized up to the maximum number of peers. It also stores the number of peers.

Bash Script:

- ➔ **test.sh**: In order to test the application, we have provided a script file that will create 4 network namespaces P1, P2, P3 and P4 and set up a network topology assigning IP addresses to 6 different veths. It is a star topology, with P4 being used only to forward packets between the peers.

Input text files:

- ➔ We have provided 3 user_info.txt files for each user of P1, P2 and P3 namespaces. It contains information about the port to run current instance of user and info about other 2 peers.

Overall Flow:

The main method first takes the filename to retrieve and populate the `user_info` table. The username, ip address and port are stored in each peers `struct user_entry`, and the `struct sockaddr_in` is also filled using these values. The socket is initialized to -1, which indicates no connection.
The server is then started using `start_server()`, binding itself to the port specified in the user_info.txt file. It continues to listen for new connections, but allows reuse of the port via a socket option `SO_REUSEPORT`.

After the server is started, the code enters an infinite loop. The `add_fds()` function is used to populate the `fd_set` to be passed to `select()`. It checks for connections that have timed out and closes their sockets, setting the value to -1. It also calculates and updates the value of the maximum file descriptor, as well as the minimum duration to the next timeout. These parameters are passed to the `select()` call. A timeout of the `select()` call indicates that a connection needs to be closed, which is done by the `add_fds()` function in the next iteration.
If the selected file descriptor is the server socket file descriptor, the `accept_connection()` method is used to accept a new peer connection. The IP address and port are verified with the user_info table before accepting the connection.
If the selected file descriptor is `STDIN`, the user input is read into a buffer using a function `read_stdinput()`. It is then tokenized into the username and the actual message, which are passed to the `send_message()` function. The peer corresponding to this username is found, and a connection is created if it does not already exist. The message is then sent over the corresponding connection. The same port as the server is reused for this connection.
If the selected file descriptor is one of the peer socket descriptors, the `read_messages()` function is called. The user_info table is checked to find the sockets that have messages waiting, and each of these messages are read and output to the terminal.
The timeout values are updated whenever a connection is created, a message is received or sent.

Compilation:

1. Copy the 2 source files, the header file, test.sh script and the Makefile into your directory from where you wish to run the application. Also copy the user_info<number>.txt files into this directory.
2. Run make from the same directory.

Running:

1. On running make, an executable file **chatApp** is created in pwd.
2. Since the file is to be run in separate namespaces simultaneously, open 3 terminal windows for each peer.
3. We need to run this file inside each namespace corresponding to each peer. Do the following steps to run the chatApp in a namespace:
    a. sudo ip netns exec <namespace> ./chatApp [Note: replace <namespace> with the namespace of that peer mentioned in the script]
    b. Input the filename to read input from for that user. For example if the peer is corresponding to the namespace P1, input user_info1.txt and so on.
    c. To start sending messages input peer<number>/msg, [Note: replace <number> with the peer number 1,2 or 3]

Sample Input:

[Terminal <i> corresponds to peer <i>]
This is a sample input in order:
Terminal 1:
peer2/Test message from peer 1
peer3/Test message from peer 1

Terminal 2:
peer1/Test message from peer2
peer3/Test message from peer2
Terminal 3:
peer1/Test message from peer3
peer2/Test message from peer3

Sample Output:
[Terminal <i> corresponds to peer <i>]
Terminal 1 Output:
**** Welcome to the P2P chat application ****
Enter the filename to retrieve user_info and server details: user_info1.txt
Peer 1
    Name: peer2
    IP Address: 10.0.20.57
    Port: 1200

Peer 2
    Name: peer3
    IP Address: 10.0.30.57
    Port: 1200

Receiving at port 1200.
peer2/Test message from peer 1
peer3/Test message from peer 1
peer2: Test message from peer2
peer3: Test message from peer3

Terminal 2 output:
**** Welcome to the P2P chat application ****
Enter the filename to retrieve user_info and server details: user_info2.txt
Peer 1
    Name: peer1
    IP Address: 10.0.10.57
    Port: 1200

Peer 2
    Name: peer3
    IP Address: 10.0.30.57
    Port: 1200

Receiving at port 1200.
peer1: Test message from peer 1
peer1/Test message from peer2
peer3/Test message from peer2
peer3: Test message from peer3

Terminal 3 output:

**** Welcome to the P2P chat application ****
Enter the filename to retrieve user_info and server details: user_info3.txt
Peer 1
	Name: peer1
	IP Address: 10.0.10.57
	Port: 1200

Peer 2
	Name: peer2
	IP Address: 10.0.20.57
	Port: 1200

Receiving at port 1200.
peer1: Test message from peer 1
peer2: Test message from peer2
peer1/Test message from peer3
peer2/Test message from peer3