Hacettepe University

Computer Science and Engineering

BBM203 Programming Lab.

Assignment 2

Ahmet Faruk BAYRAK

21426716

25.11.2018

R.A. Selim YILMAZ

1. Software Using Documentation

1.1. Software Usage

The program runs on command prompt and the output will be printed out on the console. Program takes inputs as a command line argument. To run the program: firstly you go to the file path which contains main.c file. And write the “make” command. This command is provided by Makefile and compiles main.c file. After that program is ready to start. There are 6 arguments: first one is information of clients (clients.dat), second one is routing table of clients (routing.dat), third one is commands file (commands.dat), fourth one is maximum message size, fifth one is outgoing port number and the last argument is incoming port number. Example of command line argument is: ./HUBBMNET clients.dat routing.dat commands.dat 20 3030 6666

1.2. Error Messages

If there will be an another command except valid commands, program prints that this command is invalid.

And if the intended receiver is not reachable, program displays an error messages that the user is unreachable.

2. Software Design

2.1. Description of the Program

2.1.1. Problem

In this experiment, the problem that has been expected us to design a simple version of network communication between peers within a network. Program reads the information about clients such as their name, IP address, MAC address and routing table. After that program reads commands and operate them. Main task was sending message one peer to another one. For solving this problem, we were expected to use dynamic memory allocation, stack and queue.

2.1.2. Solution

For the solution, first of all I took the arguments and assign them. Since program knows the number of client from client.dat file, program creates clients dynamically. Program calls createClients() function and fill the clients’ informations. There is a struct which name is client. In this client, there are attributes of client. Also there is a multidimensional array for their routing table. They have 2 queue which are incoming queue and outgoing queue. And a stack structure for frame.

After creating of clients, program calls readCommands() for read commands and operate them. In this function I parse the commands and make changes according to command. There are 5 commands: message, show_frame_info, show_q_info, send and print_log.

- MESSAGE command: Format of message command is like: MESSAGE C E #A few small hops for frames, but a giant leap for this message.#. It means that peer C is a sender and C wants to send message between #s to peer E. I parse this line and send them to the messageCommand() function.

In this function, first of all program calculates the number of frame according to message size and maximum size of message. According to number of frame, program creates the queues dynamically. After that program creates layers, pushes them to stack(frame) and dequeue it to outgoing queue of sender peer until every frame placed in the peer's outgoing queue.

- SHOW_FRAME_INFO command: Format of show_frame_info command is like: SHOW_FRAME_INFO C out 3. It means that show the third frame of peer C's outgoing queue. If there is a frame like that, prints the information of this frame such as message chunk and layers. Command may demand the incoming queue information if writes "in" instead of "out".

First of all I parse this line and send them to the showFrameInfo() function. In this function, according to the "in" or "out" type of queue, program checks that whether this frame exists. If there is a frame like that, prints the information of frame, otherwise prints that "No such frame."

- SHOW_Q_INFO command: Format of show_q_info command is like: SHOW_Q_INFO C out. It means that prints the number of frames in the peer C's outgoing queue. Command may demand the incoming queue information if writes "in" instead of "out".

First of all I parse this line and send them to the showQInfo() function. In this function, according to the "in" or "out" type of queue, program prints the number of frame in this queue according to rear of queue.

- SEND command: Format of send command is like: SEND C. It means that C is going to send message. This command comes after message command so program already created the frames and placed them to C's outgoing queue. In this command, program calls the sendMessage() function. In this function, according to C's physical layer of frames, C sends their frames to receiver's incoming queue. Receiver checks the network layer of frames to check who the intended receiver is. If the intended receiver is. If the frame is addressed to the receiving client, it can proceed to unpack the message chunks. If the message intended to someone else, clients checks its routing table to determine next hop. This function call itself recursively until the message is received by the intended receiver.

One more important things here is forwarding message to its intended destination may not be always possible. For example message is intended to client E but there is no way to reach client E. In that case an error message stating that the user is unreachable should also be displayed and messages should be dropped.

- PRINT_LOG command: Format of print_log command is like: PRINT_LOG D. It means that show us all activity on the client D such as message received informations or message forwarded informations. Every client has their own log and program fills this log in sendCommand() function. When print_log command comes, program prints the log of peer which in the command.

Here is the functions that I used in the program:

- **void** createClients(FILE *file1, client *clients, **int** clientNumber, FILE *file2, **const char** *sender_port_number, **const char** *receiver_port_number)

First parameter is the file which informations of client (clients.dat). Second parameter is clients. Third parameter is number of client. Fourth parameter is routing table (routing.dat). Fifth and sixth parameters are sender and receiver port number.
Function fills the information of clients according to files.

- **int** findByID(client *clients, **char** clientName, **int** clientNumber)

First parameter is clients. Second parameter is name of client. And the last argument is number of client.
This function returns the index of clients which name is second parameter.

- **void** readCommands(FILE *commandFile, **int** commandNumber, client *clients, **int** max_message_size, **int** clientNumber, **const char** *sender_port_number, **const char** *receiver_port_number)

First parameter is the file which include commands (commands.dat). Second one is number of command. Third one is clients. Fourth one is maximum size of message. Fifth one is number of client. Sixth and seventh parameters are sender and receiver port number.
In this function, program reads the commands.dat file and parse the commands. According to commands, route them to functions.

- **char** findPath(client *clients, **char** source, **char** destination, **int** clientNumber)

First parameter is clients. Second parameter is name of source client. Third

parameter is name of receiver client. And the last argument is number of client.
This function returns name of next hop client.

- **void** messageCommand(**char** *senderID, **char** *receiverID, **char** *message, **int** max_message_size, client *clients, **int** clientNumber, **const char** *sender_port_number, **const char** *receiver_port_number)

First parameter is name of sender client. Second parameter is name of receiver client. Third parameter is message which intended to send. Fourth parameter is maximum size of message. Fifth parameter is clients. Sixth parameter is number of client. Seventh and the last parameter is sender and receiver port numbers.
This function places the message to outgoing queue of sender client. First of all calculates the number of frame according to maximum size of message and message size. According to number of frame, creates outgoing queue dynamically. After that creates the layers and push them to the stack (frame) in the last layer (physical layer) function calls the findPath() function to determine next hop client. According to this information, fills the physical layer and push them to stack as well. Finally enqueue this frame to outgoing queue. This loop will return size of frame.

- **void** showFrameInfo(**char** *clientName, **char** *queueType, **char** *frameNumber, client *clients, **int** clientNumber, **int** totalFrameNumber, **int** hops)

First parameter is name of client. Second parameter is type of queue, “in” or “out". Third parameter is number of frame. Fourth parameter is number of client. Fifth parameter is number of frame. And the last argument is number of hops.
This function firstly checks that whether that frame exist. If this frame exist, prints the information of frame. Otherwise prints “No such frame.”

- **void** showQInfo(**char** *clientName, **char** *queueType, client *clients, **int** clientNumber)

First parameter is name of client. Second parameter is type of queue, “in” or “out". And the last argument is number of client.
This function firstly checks that type of queue. According to them prints the number of frame in this queue.

- **void** sendCommand(client *clients, **int** clientNumber, **char** clientName, **int** hops)

First parameter is clients. Second parameter is number of client. Third parameter is name of sender client. And the last argument is number of hops.

This function firstly determine to sender client, receiver client and next client to reach receiver client. If receiver client and next client are the same then it means they are direct neighbor otherwise sender sends to next client. After determining next client, dequeue the outgoing queue of sender and enqueue them to incoming queue of receiver. If the next client is not the receiver which we want to intend, program determines the next client. According to the next address changes the physical layers (MAC address) and call itself recursively until reaches the receiver.

There is some case that receiver may be unreachable. In this situation program prints the error. This situation will be stated in the log such as “Success: No”

2.1.3. Algorithm

1) Program takes arguments and assign them.
2) Program reads the first line of client.dat and commands.dat to determine number of clients and commands.
3) Program creates clients according to number of clients dynamically.
4) Program calls createClients() function to fill the clients according to informations in the client.dat and routing.dat
5) After creating and filling the clients, program calls readCommands() function for reading the commands and rout them according to command.