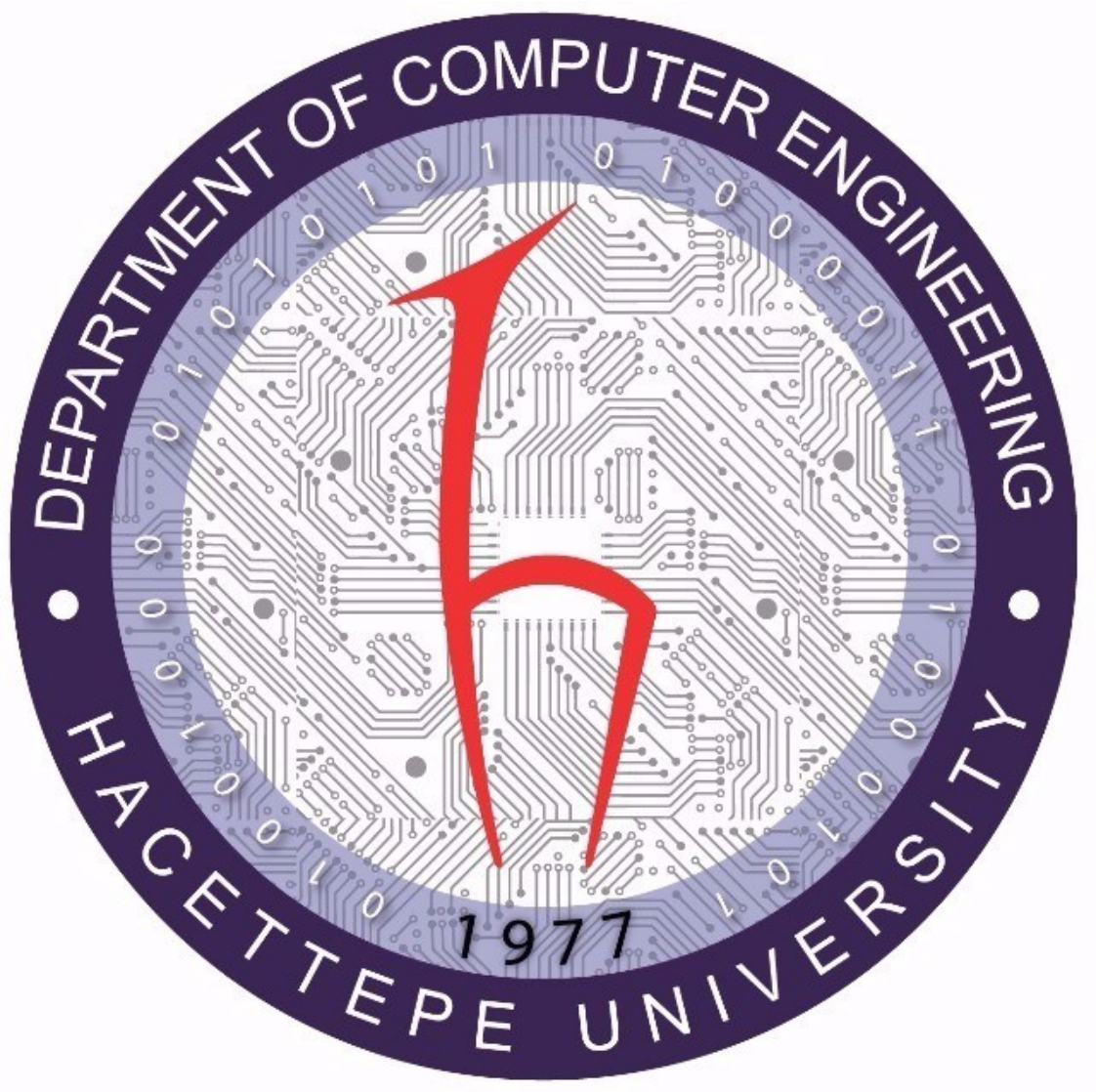

HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

GÖKHAN ÖZELOĞLU
b21627557@cs.hacettepe.edu.tr

21627557

BBM 203 - ASSIGNMENT – 2
COMPUTER NETWORKING

2.1. Software Usage

This program is compiled by using Makefile. There are some input files and arguments that will be used in program.

INPUTS

```
$ make
$ ./HUBBMNET clients.dat routing.dat commands.dat 21 0706 0607
```

Program compiles and runs above. **clients.dat**, **routing.dat** and **commands.dat** files name's can be changed. These are independent of the text format or name. The first 3 input files contain the networking data. 4th command represents the maximum message size and last 2 inputs are sender and receiver port numbers.

OUTPUT

The program gives different outputs on the screen. The output depends on the commands.

2.2. Error Messages

There are 2 different error messages in the program. The 1st error messages occurs if the input files could not open. The 2nd error messages occurs if the command say that show me information about not exists frames.

3. SOFTWARE DESIGN

3.1. Problem

This assignment wants to creating a network system that transporting messages between the clients. According to input files and command line arguments, the program must transfer the messages by using client, command and routing information. The message dividing into chunks and each chunk stores in the layers. Layers stores in the frames and frames stores in the queues. There are 2 different queues for each clients that names are **incoming** and **outgoing.**

3.2. Solution

I used **struct** in this assignment to store the information. Also, I chose to **queue** and **stack** implementations. I created 4 different struct for each layer. Layer information holds in **char pointer** type. I used dynamic memory allocation for each layer information. I implemented for each frame as a stack. I had to use **void pointer** while storing 4 different type layer. So, I have just pointed to address of the layers. There is not any problem while pushing the layers into frames, but when I want to re-use them, I had to make type casting. Also, I defined

Queue struct to store frames. It has rear and front variables as classical queue implementation. Also, I defined a **Clients** struct to store client's information. Name, MAC, IP, log, Incoming and Outgoing Queue information stores here. Other struct is **Clients_Infos**. This struct just stores the inputs. I had some problem in **MESSAGE** command. So, I solved the problems with this struct. The last struct is **Log**. This represents the log information.

I read the commands line-by-line. I tokenized the lines. Then, I called the proper functions according to command. I tried to used dynamic memory allocation for each string operation. After using the these strings, I freed them.

I also had to write helper-functions like parsing the message or finding neighbor.

SOLUTION

```
typedef struct {    /** Holds t
    char *receiver_MAC;
    char *sender_MAC;
} PhysicalLayer;
```

* I defined the 4 different structs for each layer

* I stored them as a char pointer.

```
typedef struct {    /** Holds t
    char *receiver_IP;
    char *sender_IP;
} NetworkLayer;
```

* I used them for unknown size of the layers. (But the TA said that size of the strings will be fixed)

* I defined them by using **typedef**. So, actually, I defined 4 different data types.

```
typedef struct {    /** Holds t
    char *receiver_port_number;
    char *sender_port_number;
} TransportLayer;
```

```
typedef struct {    /** Holds t
    char *receiver_ID;
    char *sender_ID;
    char *message_chunk;
} ApplicationLayer;
```

```
typedef struct {
    int top;
    void *frame[4];
} Stack;
```

* I defined **Stacks** and **Queues**.
* Stack includes top and frame variables. Top is points to stacks's top and frame stores the 4 layers. I defined as a **void** because there are 4 different layers. Each of them has different type.

```
typedef struct {
    int front;
    int rear;
    int frame_num;
    Stack **queue_array;
} Queue;
```

* Queue includes the front, rear to point the queue's place. Frame_num represents the number of frames that includes inside of the queue. queue_array stores the each frames. So, I defined as double pointer array.

```
typedef struct {    /** Clie
    char *client_name;
    char *IP;
    char *MAC;
    Queue *incoming_queue;
    Queue *outgoing_queue;
    Log *log;
} Clients;
```

* Clients struct stores the client's name, **MAC**, **IP**, **incoming/outgoing** queues and **log** information. So, I was able to reach directly client's information if necessary.

void push(): Classic stack push function that adds new layers on the frame.

void pop(): Classic pop function pops the layers from stack.

bool isEmpty(): Controls the queue is empty or not. Returns **true** or **false**. I included **<stdbool.h>** library.

void enqueue(): Adds the new stack(frame) into queue.

Stack *dequeue(): Removes the frames from queue. It returns frame.

void print_frames(): Prints out the frame information on the screen in **MESSAGE** command.

Clients *read_client_data(): Reads and stores client data into the array

void parse_message(): Parse the message into small chunks by using max_msg_size parameter. Stores the each chunks into parsed_message array.

char *find_next_neighbor(): Searches the next neighbor according to routing data and sender client's MAC – receiver client's IP addresses. Returns the next client's MAC address.

void declare_client_queues(): Declare and allocates the memory for each client's incoming and outgoing queue's. Also, assigns some initial values to queues.

void getTime(): The local time is finds and assigns a string.

void send(): This is a recursive function that calls itself if the frames do not reaches the final client. If the frames reaches, then, prints out the message and terminates the running of the function. If messages drops somewhere, it prints out the error message. This function calls helper-functions like finding neighbor or enqueue, dequeue, pop. While transfering the frames, firstly, queues makes the dequeue and then, makes enqueue. The past client freed after transfer. While opening the frames, I used pop function. Also, while updating MAC addresses, firstly, I pop the **Physical** and **Network** layers. I updated and then push back to frames. I updated logs in this function.

void activate_logs(): Allocates the memory for each client's logs and assign some of initial values.

void read_command_file(): The all operations are running in this function according to commands. I read the command file line-by-line. I used **fgets** because we do not know the length of the lines. I tokenized the lines. I controlled the commands by using if-else statements. Each command calls the proper functions. I pushed the layers and added the

queues in **MESSAGE** command. End of the function, I freed the which allocated dynamically.

void read_routing_data(): Reads the routing data and stores them into double pointer char array.

int main(): Opening and closing files, taking command line arguments, free operations and calling read_command_file function handled in this function.