Hacettepe University

Computer Science and Engineering Department

Name and Surname	:	İrem Dereli
Identity Number	:	21727135
Course	:	BBM 203
Experiment	:	Assignment 2
Subject	:	Stack, Queue and Dynamic Memory
	:	Allocation
Due Date	:	25.11.2018
e-mail	:	iremm.dereli@windowslive.com

Problem Definition

In this assignment, we accomplish design a computer network. Problem is, we must design it with using queues and stacks. There will be clients and one message to be sent, we will create a frame as a stack to hold data. Frames will hold smaller message than the original one, mac addresses, IP addresses and port numbers with layers. These frames will be pushed into queues according to rules of queues and stacks. Frame to be sent to the client to be determined by the routing file and a path will be created. We must create that path, create queues, logs, routes for all clients. Then, message will be parsed according to maximum message size(character). Message will be received or not according to path.

Solution

Firstly, started from reading files. I put all the data from files into client and routing array. Then started to read commands file. According to commands, I called some function that created for these commands.

1. For message command, called commandMessage function to parse message, create frames as a stack and push the stack into queue.

2. For show_frame_info command, called commandShowFrameInfo function to show all data in these frames. Function gets parameter of client id and frame number. According to these knowledges, function gets that frame from client's queue and prints all data.

3. For show_q_info command, called commandShowQInfo function to print how many frames in that queue.

4. For send command, called sendCommand function to send these frames that in queues to receiver client according to path. This function calls messageControl function to control and create that path.

5. For print_log command, called printLog function to print all information that holds in log array according to client id.

6. Else, program prints invalid command.

For send command, there are a function called messageControl that controls that receiver mac and receiver id is same or not. If they are not same, function calls routeControl function to get location of client that be used to reach receiver client. But, because of receiver id and receiver mac still don't match, messageControl calls itself again and again until they are same. When they are same, that means message reached the received client. While these functions are processing, queues continue to be processed (frames are sending or deleting).

routeControl function checks that which client receive the frames first to reach receiver client from sender client. In other words, checks path. If there is not such path (could not reach receiver client) function returns -1.

Lastly, if message received to receiver client successfully, all arrays that memory allocated will be freed.

And in my program, I used some global variables like queue, route, clients arrays because of they are used in many functions.
Also, frames array is global because of stack.
Client size, hop number, and top(for stack) is used in all functions, so I declare it global too.

Functions Implemented

- **void readClientsRouting(FILE *clientFile, FILE *routingFile):**
  Function gets two parameters that client file and routing file. First, it gets client size from client file. According to size, it allocates the client array. And puts data into it.
  After this, function allocates route array too depending client size again. Route array is two dimensional, and it will be allocated again but this time it will be allocated client size – 1 times because of the form of routing file.
  Function will allocate queue and logs too due to client size.
  And function finally initialize some variables depending on queue basic.

Like front, rear, size.
and logs entry size will be 0 at first.

- **void readCommands(FILE *commandFile, int msg_size, char* incoming_port, char* outgoing_port):**
  Read commands function gets command file, message size and port numbers as a parameter. It reads command file and calls function according to these commands.

- **void printLog(char* client_id):**
  Print log function prints all the data of log of chosen client id.

- **void sendCommand(char* client_id, int msg_size):**
  Send command function determine the location of sender client and receiver id. It creates log according to sending message and calls message control function.

- **void messageControl(int location_sender, int location_receiver, int msg_size):**
  This function controls that receiver mac and receiver id is same or not. If they are same, that means message reached the receiver client successfully. If they are not same, function must declare a location of client to be used to reach receiver client. This client gets all frames from sender client without opening into its incoming queue. After declaring and initializing, client to be used must be sender. Now queues are ready but receiver mac and receiver id are not paired with. So, message control function must call itself again until they are same(paired).

- **void createLog(int location, char* activity_type, char* success_status):**
  Create log function creates log according to activity type like forwarded, received or sent and also success status, yes or no.
  If a message forwarded, this client will have 2 logs as received and forwarded.
  But if a message sent or only received, this client will have only 1 log.
  According to these knowledges, function creates log.

- **void commandShowQInfo(char *client_id, char* whichqueue):**
  This function prints that how many frames in selected queue.

- **void commandShowFrameInfo(char *client_id, char* whichqueue, int framenumber):**
  This function prints the data in selected client's frame. Frame number is also selected. Before printing, it controls that there is exist such a frame or not. If not, function prints that "no such frame".

- **void commandMessage(char* sender_id, char* receiver_id, char* message, int msg_size, char* incoming_port, char* outgoing_port):**
  Function first initialize the frame size depending on message length and maximum size of character. Then it starts to generate frame. It gets data from client array and put these data into frame. Frame is created as a stack. When frame is ready to insert into queue, function allocates memory on sender's outgoing queue and puts that stack into it. And finally, it pops all data from frame and frame is ready to get new data.

- **int routeControl(int location_receiver, int location_sender):**
  Route control function checks that if a client wants to reach another client, which client it should go first to reach. This function returns the location of client that will be used. If there is not a way to reach, it returns -1.

- **void remove_outcome(int j):**
  Removes the first data in outgoing queue.

- **void remove_income(int j):**
  Removes the first data in incoming queue.

- **Frame* peek_income(int location):**
  Returns the first data from incoming queue.

- **Frame* peek_outcome(int location):**
  Returns the first data from outgoing queue.

- **void insert_outcome(int j, Frame fr[]):**
  Inserting data into outgoing queue of location j. J means location of client to be inserted into that client's.

- **void insert_income(int j, Frame fr[]):**
  Inserting data into incoming queue of location j. J means location of client to be inserted into that client's.

- **void pop():**
  It pops the last item in frame stack, and prints what popped.

- **void push(Frame fr[]):**
  Puts item into frame stack.

Algorithm

i. Gets file names from command line and opens them
ii. Gets message size, outgoing port and incoming port from command line
iii. Calls readClientsRouting function
    a. Initialize client size from client file
    b. Allocate memory of client array due to client size
    c. Allocate memory of route two dimensional array due to client size
    d. Allocate memory of queue due to client size
    e. Allocate memory of log array due to client size
    f. Initialize queue's front, rear, size for all clients depending on the rules of queue.
iv. Calls readCommands function
    a. Reads file and initialize size of commands
    b. For loop until reach size of command
        i. If command is "message"
            1. Gets sender id
            2. Gets receiver id
            3. Gets message
            4. Calls commandMessage function
                a. Initializes frame size
                b. Gets location of sender client
                c. Gets location of receiver client
                d. For loop until frame size
                    i. Puts all data into stack layer by layer

ii. Allocate memory on outgoing queue of sender client due to frame size times
iii. Insert frame into queue
iv. Pop frame

ii. If command is “show frame info”
    1. Gets client id
    2. Gets which queue is (in or out)
    3. Gets frame number
    4. Calls commandShowFrame function
        a. Gets location of client
        b. If queue is out
            i. If frame number is greater than frame size or less than 0
                1. Prints “no such frame”
            ii. Else
                1. Prints all layers and data in that layers
        c. If queue is in
            i. If frame number is greater than frame size or less than 0
                1. Prints “no such frame”
            ii. Else
                1. Prints all layers and data in that layers

iii. If command is “show queue info”
    1. Gets client id
    2. Gets which queue is (in or out)
    3. Calls commandShowQInfo function
        a. Gets location of client
        b. If queue is out
            i. Prints the number of frames in outgoing queue
        c. If queue is in
            i. Prints the number of frames in incoming queue

iv. If command is “send”
    1. Gets client id
    2. Calls sendCommand function
        a. Gets location of sender client
        b. Gets location of receiver client
        c. Calls createLog function
            i. If message is forwarded
                1. Creates 2 log
                2. Puts all data in log
            ii. Else if message is received or sent
                1. Creates 1 log
                2. Puts all data in log
        d. Calls messageControl function

i. If receiver mac and receiver id pairs
    1. Increase hop number
    2. Get size of the frames
    3. Allocate memory of receiver's incoming queue
    4. Insert frames into receiver's incoming queue from sender's outgoing queue
    5. Remove frames from sender's outgoing queue
    6. createLog
    7. Print the message
    8. Free sender's outgoing queue
ii. Else if receiver mac and receiver id do not pair
    1. Get location by using to reach receiver client from routeControl
    2. If location by using is not equal -1
        a. Allocate memory of using client incoming queue
        b. Increase hop number
        c. Insert frames into using client's incoming queue from sender's outgoing queue
        d. Remove frames from sender's outgoing queue
        e. Free sender's outgoing queue
        f. Allocate memory of using client outgoing queue
        g. Initialize location sender as location by using
        h. Get new using client from routeControl function
        i. If location by using is not equal -1
            i. Insert sender's outgoing queue from incoming queue
            ii. Remove frames from incoming queue
            iii. createLog
            iv. Free sender's incoming queue
            v. Call message control function
        j. Else
            i. Increase hop

ii. Print Error message
iii. createLog
iv. Free sender's outgoing queue

3. Else
    a. Increase hop
    b. Print error message
    c. createLog
    d. Free sender's outgoing queue

v. If command is "print log"
    1. Get client id
    2. Call printLog function
        a. Gets location of client
        b. If there is not log
            i. Does not print anything
        c. Else
            i. Prints all data in that log

vi. Else
    1. Prints the command
    2. Print "Invalid command"

v. Gets location of receiver client
vi. Free receiver's incoming queue
vii. Close all files
viii. Free client array
ix. Free queue array
x. Free logs array
xi. Free route array