Documentation for the semester work called **Server for TCP/IP communication**

Created by:
Karolina Zegeryte
2023

Description:

The goal of the task is to create a multi-threaded server for TCP/IP communication and implement the communication protocol according to the given specification.

There I have created a server to automatically control remote bots. The robots themselves log in to the server and it guides them to the center of the coordinate system. For testing purposes, each robot starts at random coordinates and tries to reach the [0,0] coordinate. At the target coordinate, the robot must pick up the secret message. On the way to the destination, the robots may encounter obstacles that they must avoid. The server can navigate multiple robots at the same time and implements a flawless communication protocol.

Communication between the server and the robots is realized by a fully text protocol. Each command is terminated by a pair of special symbols "\a\b". (These are the two characters '\a' and '\b'.) The server must follow the communication protocol in detail, but must allow for imperfect bot firmwares.

Here are the examples of server messages:

Name	Message	Description
SERVER_CONFIRMATION	<decimal number of 16 bits>\a\b	A message with a confirmation code. It can contain a maximum of 5 numbers and the ending \a\b.
SERVER_MOVE	102 MOVE\a\b	Order to move 1 cell forward
SERVER_TURN_LEFT	103 TURN LEFT\a\b	Order to turn left
SERVER_TURN_RIGHT	104 TURN RIGHT\a\b	Order to turn right
SERVER_PICK_UP	105 GET MESSAGE\a\b	Order to pick up the secret message
SERVER_LOGOUT	106 LOGOUT\a\b	Order to end up the connection
SERVER_KEY_REQUEST	107 KEY REQUEST\a\b	Request of the key from the server for further communication
SERVER_OK	200 OK\a\b	Positive confirmation
SERVER_LOGIN_FAILED	300 LOGIN FAILED\a\b	Failed attempt to log in
SERVER_SYNTAX_ERROR	301 SYNTAX ERROR\a\b	Wrong syntax of the message
SERVER_LOGIC_ERROR	302 LOGIC ERROR\a\b	The message to be sent when an error occurs
SERVER_KEY_OUT_OF_RANGE_ERROR	303 KEY OUT OF RANGE\a\b	Wrong key ID

Messages with the similar purposes will appear from client server, mainly username details, key ID, confirmation details and so on.

Server and client have 5 pairs of authentication keys, all of them are in the source code written.
Here are steps of communication between client and server:
1. Robot sends its username to start the communication.
2. Server requires the client to send its key ID.
3. Server calculates the hash to find out whether the key ID is valid or not.
4. Server sends the hash to the client and client checks, whether the hash is the same as he/she has.
5. In case the hash is the same, client sends confirmation message to the server, otherwise sends SERVER_LOGIN_FAILED and the communication is over.

There will be similar communication between client and server in order to navigate the robots including times of robots' recharging process. While recharging robot does not react on any of the messages and the time of recharging should be 5 or less seconds, otherwise TIMEOUT_RECHARGING will be sent to stop the communication.

The timeout for sending/receiving messages is 1 second. If the client or the server will not respond withing 1 second, the communication will be stopped.

The server is patient enough to to receive one request from the client in a few messages. It always waits till the end of the request except one situation: if the received message is too long, the server will send SERVER_SYNTAX_ERROR to the client and will stop the communication.

Functions and classes description:

1. **void getPairForKey (int key, int & value, bool first)**
This function is used to find the pair key (value) for the given key. Depending on whether we need the key from the server of from the client, boolean variable indicates the receiver.

2. **void checkSending (int c, const string & src)**
This function is responsible for checking whether the sending has been done properly or not. It uses c variable as the identification of the process and src as the message.

3. **void serverLoginFailed/ serverSyntaxError/ serverLogicError/ serverKeyOutOfRangeError (int c)**
All this function are responsible for informing the client about errors occurred during communication process and stopping it, where c is the identification of the process.

4. ***class CInput***
This class is responsible for receiving and processing data from the input (client).

Variables:
1. deque < string > m_input_queue – queue of requests received from the client
2. string m_resting_input – string for storing the message which was not ended with \a\b ending
3. fd_set m_sockets – all the sockets for communication
4. struct timeval m_timer – timer for timeouts

Methods:

a. **string readDataAndReturnNextStr (int c, const string & stage)**
This method is used for reading requests from the client and for the returning of the message from the top of the query to work with. It uses c as identification of the process and stage as the identification of the current stage of communication (it can be username stage, confirmation, message receiving or recharging).

b. **void receiveDataFromSocket (int c, const string & stage)**
This method is used for receiving requests and messages from the socket c. It reads a message from the client, adds it to the queue and also checks the recharging status and handles the timeouts for receiving messages. The variable stage is used to identify if the server should wait up to 5 seconds due to recharging or just 1 for continuation of communication.

c. **void addNewDataToInputQueue ()**
This method adds the requests from the client to the queue of requests – m_input_queue. It adds the whole request without ending \a\b.

d. **void fulfillInputDeque (int c, const string & stage)**
This method is used for fulfilling an empty queue of client requests when the queue is empty. It also checks for syntax errors for the cases when m_resting_input is too long to be a valid message. Stage shows which type of the message the server is awaiting to receive.

f. **void initializeSockets (int c)**
This method initializes the m_sockets socket set for use in further work with network connections, where c is a socket.

g. **void initializeTimeout (bool recharging)**
This method is used to initialize the timer for the right timeout – recharging or default one.

h. **string recharge (int c)**
This method handles recharging process.

5. **void makeSumOfBytes (const string & str, int & summary)**
This function calculates the summary of the bytes from str into summary.

6. **void hashCalculating (int key_id, int user, bool first, int & hash_key)**
This function calculates hash using given formula for authentication purposes.

7. **void checkForDigitsOnly (const string & str, int c)**
This function throws the syntax error in case str contains non-numeric characters.

8. **void usernameRequest (int c, const string & str, int & user, string & curr_state, string & stage, CInput & in)**
This function is used for getting the username from the client. It sends the request to get the key as well and sets the current stage to receiving key ID to continue communication.

9. **void keyIDRequest (int c, const string & str, int & key_id, int & user, string & curr_state, string & stage)**
This function is responsible for validation of the given key ID and setting current stage to confirmation stage in case the received message is correct, otherwise errors will be sent to the client to indicate failure of login process.

10. **void confirmationRequest (int c, const string & str, int key_id, int user, string & curr_state, string & stage)**
This function is responsible checking whether the confirmation has been successfully provided or not. In case of wrong message from the client (should be only numbers + ending as key ID) or hash is not valid the error of failed login will be sent to the client.

11. **bool beginServerWork(int c, string & curr_state, CInput & in)**
This function is responsible for the beginning of the client-server communication and for the checking of proper authentication. In case of failure, False will be returning, True in other cases.

12. **void receiveMessage (int c, const string & message_exp, CInput & in, string & message)**
This function is responsible for extracting the message from the input in.
message_exp shows the type of the message the server expects to receive from client.

13. **void getOKMessage (int c, const string & stage_exp, CInput & in, string & message)**
This function is responsible for handling the receiving of ok message from the client.

14. **void getOKMessageAndCoordinates (int c, const string & stage_exp, CInput & in, pair < int, int > & coordinates, string & message)**
This function is responsible for handling the receiving of ok message from the client and for the coordinates of the robot.

15. **void tryToMove (int c, const string & direction)**
This function is responsible for moving of the robot from the socket c. In case the given direction is invalid, it throws the error. It sends the possible direction to the client to confirm it. In case there is an obstacle, the client will send the same coordinates back to the server. In case of success the client sends updated coordinates of the robot to the server.

16. **void reportIfNoOkMessage (const string & receive, int c)**
Additional function to indicate an error during communication – ok message is missing.

17. **void getDirection (int c, pair < int, int > & coordinates_before, pair < int, int > & coordinates_future, string & direction)**
This function is responsible for calculating the direction based on the information received from the client and stored at the server. It shows the direction robot just moved in.

18. **void calculateNextDirection (string & dir, int c)**
This function is responsible for calculating further directions based on previous and navigates the robot to [0,0] coordinates.

19. **void turn (int c, string orig_direction, const string & goal_direction, CInput & in)**
This function is responsible for moving the robot till it reaches final coordinates and for the communication with the client during making moves.

20. **void makeMoveWithDirection (int c, const string & direction, CInput & in)**
This function is used for navigation of robots with the direction calculated from functions above.

21. **void solveObstacle (int c, pair < int, int > & coordinates, CInput & in)**
This function handles cases, when a robot move in the calculated direction due to the obstacle on its path, and moves robots to the forward coordinates followed the wanted (with obstacle).

22. **void moving (int c, CInput & in, pair < int, int > & coordinates, bool first)**
This function is responsible for moving the robots down to coordinate with 0 value. The algorithm of robots' navigation is the following: first the robot reaches x zero axis, then y zero axis.

23. **class CRobot**
This class is responsible for handling robots' moves. It navigates robots, reports errors and picks up the final (secret) message.

Variables:
1. pair < int, int > m_coordinates – current coordinates of the robot
2. string m_message – the direction, where the robot should go. It is called "message" due to the fact, that this direction should be sent to the client and approved by him/her as a message from the server.

Methods:

a. **void firstMoveInitial (int c, CInput & in)**
This method is responsible for the very first move of the robot to start moving and navigation.

b. **void passColumns (int c, CInput & in)**
This method is responsible for robot's moves till it reaches zero x axis coordinate – passing all the columns.

c. **void passRows (int c, CInput & in)**
This method is responsible for robot's moves till it reaches zero y axis coordinate – passing all the rows.

d. **void pickUpFinalMessage (int c, CInput & in)**
This method is used for picking up the final message. It is responsible for the correct logout as well.

24. **bool startMoving (int & c, string & curr_state, CInput & in)**
This function is responsible for initializing and moving the robots to the goal coordinates. Returns True in case of success and False in other cases.

25. **int main (int argc, char ** argv)**
The main function responsible for sockets initialization, connection, starting the communication and handling it.

Testing part:

Testing of this program has been provided with the tester given from elsewhere and all the tests have been passed successfully.