Vladimir Vorobyev, 19.05.2014
Report ze semestrální práce z BI-ATS

Popis aplikace

Testovaná aplikace je server který získává, kontroluje a ukládá informace od klientů. Komunikace servera a klienta je určená jednoduchým protokolem postaveným nad TCP. Pred nahráním dat klient musí se přehlásit. Data jsou dvou druhu: fotografii (ukládá se do souborů, server kontroluje checksum) a textové řetězce (ukládá se do logovacího souboru, žádná kontrola). Po otevření spojení server pracuje s klientem jenom určitou dobu (45 s.) pak se spojení uzavře. Server má slušné pracovat i s vícero klientů najednou. Podrobnější popis je tady https://edux.fit.cvut.cz/courses/BI-PSI/uloha1

N.B. protože programuji a komentuji v angličtině další část reportu si dovolim taky napsat anglicky.

Tests description

1. lpcontrol_positive_test()

   Positive test of the function lpcontrol(string login, string passwd). Test if the function recognizes valid username and password.

   Details: valid username (string of ASCII characters starting with “Robot”) and valid password (number that equals sum of the ASCII values of the usrname).

   Expected result: true.

2. lpcontrol_invalid_username_test()

   Negative test of the function lpcontrol(string login, string). Test if the function recognizes invalid username.

   Details: invalid username (string of ASCII characters not starting with “Robot”) and valid password

   Expected result: false

3. lpcontrol_passwd_invalid_val_test()

   Negative test of the function lpcontrol(string login, string). Test if the function recognizes invalid password value.

   Details: valid username and invalid password (number that does not equal sum of the ASCII values of the usrname).

   Expected result: false.

4. lpcontrol_passwd_invalid_fmt_test()

   Negative test of the function lpcontrol(string login, string). Test if the function recognizes invalid password format (contains not only numbers).

   Details: valid username and invalid password (number that does not equal sum of the ASCII values of the usrname).

   Expected result: false.

5. write_to_log_test()

   Test of write_to_log(string) that appends the argument to the end of logfile .

   Details: call function several times with different arguments

   Expected result: after each call the content of the file is not changed except the appending of a new line (which was an argument of the function) in the end

6. recv_message_test()

   Test if the recv_message() recognizes the messages correctly when all of them are sent at once.

   Details: the messages are separated with string “\r\n”. There is no restrictions on the content of messages. A message can be sent by parts . The client does not have to sen the whole message at once. The structure cli is used for data processing. Cli.data contains a parsed message, cli.buffer cotains data that were recived from client but have not been processed yet. In this test recv_message() is called three times after the client has sent three messages at once. The messages contain different symbols that must not affect the parsing.

   Expected result: after each recv_message() call a valid message occurs in cli.data

7. recv_message_one_by_one_test

   Test if the recv_message() recognizes the messages correctly when are sent one by one.

   Details: In this test recv_message() is called after each time the client has sent message. The client sends only one message each time. The messages contain different symbols that must not affect the parsing.

   Expected result: after each recv_message() call a valid message occurs in cli.data

8. recv_message_long_test()

   Test if the recv_message() is able to recieve a long message.

   Details: In this test recv_message() is called after the client has sent the whole message at once. The message 10 tiomes larger than the buffer.

   Expected result: valid data in cli.data

9. recv_message_split_test()

   Test if the recv_message() is working right when a part of msg is already in buffer and the rest is being sended by peer.

   Details: In this test cli.buffer is preventively filled with the first part of the message. recv_message() is called after the client has sent the rest.

   Expected result: valid data in cli.data (concatenation of two parts of the messages)

10. recv_message_buffer_test()

    Test if the recv_message_test() parses the buffer correctly.

    Details: cli.buffer is preventively filled with data that must be parsed into three

expexted messages. The recv_message() is called three times

Expected result: valid data in cli.data after each call

11. check_buffer_command_positive_test()

    Test if the check_buffer_command() recognizes a valid command in the beginning of the buffer

    Details: command is defined in the beginning of the message and it determines the way how server must receive and process the followed data. in There are two commands: “info” and “foto”. The message with “info” command starts with a string “INFO ” and the “foto” message starts with “FOTO”. The function check_buffer_command() returns true if the data in cli.buffer can be a stat of command definition or false otherwise. In this test cli.buffer contains whole messages that start with valid command names.

    Expected result: check_buffer_command() return true for every content of cli.buffer

12. check_buffer_command_positive_start_test()

    Test if the check_buffer_command() recognizes a beginningof a valid command in the beginning of the buffer

    Details: In this test cli.buffer contains strings that can start valid command names.

    Expected result: check_buffer_command() return true for every content of cli.buffer

13. check_buffer_command_invalid_fmt_test()

    Test if the check_buffer_command recognizes invalid format of the commands that are in the buffer.

    Details: The name of the command must be separated from the rest of the message with space. The format ts invalid if this space absents or any other symbol after the command name presents. Here this situation is checked. Cli.buffer is filled with invalid messages and check_buffer_command() is called

    Expected result: check_buffer_command() return false for every content of cli.buffer

14. check_buffer_command_invalid_command_test()

    Test if the check_buffer_command() recognizes invalid command name in the beginning of the buffer.

    Details: Cli.buffer is filled with invalid comman named and check_buffer_command() is called.

    Expected result: check_buffer_command() return false for every content of cli.buffer

15. recv_command_online_test()

    Test if the recv_command() works right when getting data from client.

    Details: the function recv_command() determines which command is in the beginning of the buffer. If the command is valid it returns it's type and erases redundant prefix from the cli.buffer (command name and space). Otherwise it returns

"unknown" type of command and does not change the buffer. The function recieves extra data from client when it is needed. In this test cli.buffer is empty from the start of test and client sends messages with all the types of data. After each call of recv_command() data from cli.buffer is erased so that the next command coul be recognized properly.

Expected result: recv_command() returns the type of command defined in buffer. If the type is "info" or "foto" first 5 charaters are erased from the buffer

16. recv_command_buffer_test()

    Test if the recv_command() works right with valid and complete data in the buffer.

    Details: This test is like the test recv_command_online_test(). The only difference is that the data is inserted into cli.bufffer istead of being received from client

    Expected result: recv_command() returns the type of command defined in buffer. If the type is "info" or "foto" first 5 charaters are erased from the buffer

17. recv_foto_positive_test()

    Test if the recv_foto() works right when recieves a valid 'foto' message.

    Details: Processing photo message is different from processing a regular image. This message is not separated by "\r\n" and has the following format: "LENGTH <space> dataCHECKSUM". The LENGTH can contain only numbers determains the size of data in bytes. It must be > 0. Data can contain anything. CHECKSUM contains sum of ASCII values of the data represented in big-endian format. CHECKSUM is always 4 byes long. Recv_foto() must process this message in proper way. If the chechsum is correct the data has to be saved to a file with name "fileX.png" where X is a number between 0 and 999. In this test recv_foto() processed a valid message.

    Expected result: the sent data must be found in the file with greatest number (easy to get it prom the global structure)

18. recv_foto_bad_checksum_test()

    Negative test of the recv_foto() with wrong checksum.

    Details: recv_foto() receives a message with wrong checksum

    Expected result: exception 700 must be thrown. Data is not saved to file.

19. recv_foto_wrong_fmt_test()
    Negative test of the recv_foto() with wrong format. exception 600 expected.

    Details: recv_foto() receives a message with wrong format

    Expected result: exception 600 must be thrown. Data is not saved to file.

20. send_message_positive_test()

    test a method send_message() that sands a message to a client.

    Details: send_message() sends a c-string message to client. In this test

send_message() is callled with the proper argument and the testing client receives the message

Expected result: the message received is same as the argument of send_message()