Protocol:

Handling applications:

1. Server

* Should be started with the following command:   
  file\_server users\_file dir\_path [port]

file\_server – name of the executable.

users\_file – path to an existing file with tab-delimited info as mentioned in the guidelines. File\_server is required to run with the permissions to read this file.

dir\_path – path to an existing empty directory. File\_server is required to run with the permissions to read and write in that directory.

* To close the server, you will need to create a file called "exit.txt" in dir\_path. **After** every request-handling, the server will search for this file and if one found – terminates.   
  Note: server will search a maximum of MAX\_FILES\_TO\_CHECK files in the directory (as defined in server.h).  
  Note2: to properly use this feature, add exit.txt after all clients except for 1 quitted, then quit the last one (otherwise server will exit while other clients still waiting for it to respone ☹ )

1. Client

* Should be started with the following command:   
  file\_client [hostname [port]]

file\_ client – name of the executable.

Hostname – optional parameter, can be provided as:

* + Dotted Decimal IP (e.g. 127.0.0.1).
  + Hexadecimal IP (e.g. 0x7F000001).
  + Integer IP (e.g. 2130706433).
  + Host Name (e.g. localhost).

Port – integer representing port number.

* Client closes itself automatically after ending the communication.

Actions

1. list\_of\_files – prints a list of all regular file in the user's directory.
2. delete\_file [name of file to delete] – removes specific file from the user's directory (on server).
3. add\_file [path to file] [file’s new name] – coping a specific file from a local directory to the user's directory (on server).
4. get\_file [file name] [path to save]- coping a specific file from the user's directory to a local directory.
5. Quit- close communication.

\* Note: an action will be considered valid only if it is followed by the arguments specified above.

protocol for establishing a connection

1. The server sends a "SERVER\_WELCOME\_MSG" message.
2. The user inserts username and password by the given format. Inputs that does not match the format are being ignored. Legal inputs are sent to the server in a "CLIENT\_LOGIN\_MSG" message.
3. The Server will respond with a "SERVER\_LOGIN\_PASS\_MSG" message or a "SERVER\_LOGIN\_FAIL\_MSG" message, according to the message and its content.
4. The client will have up to ALLOWED\_TRIALS attempts (3, as defined in “utilities.h”) to provide correct username and password. If the client fails to do so, the connection will be terminated.
5. After a "SERVER\_LOGIN\_PASS\_MSG" the client can perform any action he desires.
6. If in any point sending something from server to client failed – current connection with the client will be considered invalid: server will disconnect that client.

Friendly message

1. Our server allows user to communicate with each other using a messaging mechanize.
2. Message that was received by the server will be sent to the user or stored in the server until it will be required by the user.
3. Stored messages will be kept in a file until the client will ask for them (read\_msgs command). User is not allowed to use command “get\_file” on this file.
4. We do not guaranty the privacy of those messages. They are stored as plain text, and we might sell them to a third party or read them just for fun.
5. It is prohibited to use this mechanism for illegal activities. We are required by law to report any suspicion activity to the authorities.
6. Failed attempt to deliver such message will result in disconnection of the user from the system (as the client was probably terminated without quit command).
7. A feedback from the server will be given for any attempt to send friendly message (for example: Friendly message sent and received successfully, User was off-line: message was successfully written to his file etc.).

Structure of messages

Every message starts with a prefix of:

* + 4 bytes representing the length (in bytes) of the message not including (the prefix)
  + 2 bytes representing the type of message:

Types of message

Server->Client

* + - SERVER\_WELCOME\_MSG 0
    - SERVER\_LOGIN\_PASS\_MSG 1
    - SERVER\_LOGIN\_FAIL\_MSG 2
    - SERVER\_FILES\_LIST\_MSG 3
    - SERVER\_FILE\_REMOVE\_MSG 4
    - SERVER\_FILE\_ADD\_MSG 5
    - SERVER\_FILE\_DOWNLOAD\_MSG 6
    - SERVER\_FILE\_DOWNLOAD\_FAILED\_MSG 7
    - SERVER\_ALL\_CONNECTED\_USERS\_MSG 8
    - SERVER\_STATUS\_FRIENDLY\_MSG 9
    - SERVER\_ACTUAL\_FRIENDLY\_MSG

Client->Server

* + - CLIENT\_LOGIN\_MSG 0
    - CLIENT\_FILES\_LIST\_MSG 1
    - CLIENT\_FILE\_DELETE\_MSG 2
    - CLIENT\_FILE\_ADD\_MSG 3
    - CLIENT\_FILE\_DOWNLOAD\_MSG 4
    - CLIENT\_CLOSE\_MSG 5
    - CLIENT\_FRIENDLY\_MSG 6
    - CLIENT\_GET\_USERS\_MSG 7

Server side:

SERVER\_WELCOME\_MSG

Used to announce starting of new communication

SERVER \_LOGIN\_PASS\_MSG

Used to announce successful attempt to log in.

SERVER\_LOGIN\_FAIL\_MSG

Used to announce unsuccessful attempt to log in.

SERVER\_FILES\_LIST\_MSG

Used to deliver list of files in directory.

SERVER\_FILE\_REMOVE\_MSG

Used to deliver textual feedback regrading file deletion attempt.

SERVER\_FILE\_ADD\_MSG

Used to deliver textual feedback regrading file adding attempt.

SERVER\_FILE\_DOWNLOAD\_MSG

Used to content of file.

SERVER\_FILE\_DOWNLOAD\_FAILED\_MSG

Used to announce unsuccessful attempt to download file.

SERVER\_ALL\_CONNECTED\_USERS\_MSG

Used to deliver a list of all connected users.

SERVER\_STATUS\_FRIENDLY\_MSG

Used to announce status of the friendly message (user not exist / message sent, etc.).

SERVER\_ACTUAL\_FRIENDLY\_MSG

Used to deliver a message from one user to another.

Client side:

CLIENT\_LOGIN\_MSG

Used to provide log in details.

CLIENT\_FILES\_LIST\_MSG

Used to ask for list of files.

CLIENT\_FILE\_DELETE\_MSG

Used to ask for file deletion.

CLIENT\_FILE\_ADD\_MSG

Used to ask for adding new file.

CLIENT\_FILE\_DOWNLOAD

Used to ask for downloading a file.

CLIENT\_CLOSE\_MSG

Used to announce termination of communication.

CLIENT\_FRIENDLY\_MSG

Used to send a friendly message to gal, its content is “hello!”

CLIENT\_GET\_USERS\_MSG

Used to ask a list of users online.

CLIENT\_FRIENDLY\_MSG\_WAS\_HANDLED

Used to announce that a friendly message was handled,

Further conditions:

Those conditions must be fulfilled at the time of exception. The behavior of the protocol and apps are not guaranteed when those requirements aren't held. An unexpected behavior might occur.

1. Both client and server must have privileges to access and create the files involved in their actions attempts.
2. Every command should be written exactly as described in the guideline, including having the same number of parameters.
3. The server's directory should not be modified during the time of execution.
4. The server's directory should not contain any existing directories or files prior to the execution of the server.
5. The user is not allowed to have more than (MAX\_FILES\_FOR\_USER) 15 files at the same time, server enforces it.
6. Number of users is limited to (MAX\_USERS) 15, and one connection per user, hence no more than 15 connection (active\_fds) at given point of time.
7. Username and password will not have more than (MAX\_PASSWORD\_LEN/ MAX\_USERNAME\_LEN) 25 characters.
8. The user will not attempt to upload file larger than (MAX\_FILE\_SIZE) 512 bytes.
9. Any path that will be used in the system (in its relative or absolute form) will not contain more than (MAX\_FILE\_PATH\_LEN) 255 characters (reasoning can be found [here](https://en.wikipedia.org/wiki/Comparison_of_file_systems)) .
10. Any given command will not exceed (MAX\_COMMAND\_LEN) 265 characters (that is enough for most cases).
11. A User will not attempt to manipulate the system and exceed his directory limitation by trying to upload files to a different directory (for example: sub directory, parent directory, sibling directory, etc.). In other worlds - every user has a single directory to store all his files, and path cannot be used in a download\delete\add attempts.
12. A user will not attempt to download files that will cause him to exceed his local disk quota or limitation.
13. Endpoint user should read and understand those instruction. Informative prints will be provided when needed, but we are not committed to specific scenario or outputs.
14. The system requires a file called "Messages\_received\_offline.txt" for its proper functioning. The user will not attempt to modify or delete this file.

Project files

1. server.h and server.c contains the server application.
2. client.h and client.c contains the client application.
3. utilities.h and utilities.s contains common function & constants that are been used by both parties.