1) Protocols

1.1) Message exchange order:

- Client connect to server
- Each time a client sends a request, the server replies with an appropriate response. By exchanging messages in alternative turn, we don't have to worry about message boundary in TCP.

1.2) Packet format

Packet header:

The packet header consists of:

- Protocol version (1 byte): self-explanatory
- Packet type (1 byte): Specify the type of packet. This is discussed more later.
- Packet length (2 bytes): specify the total length of the packet in bytes, including header. This number follows network-order endianness.
- Session token (4 bytes): this is a unique, random token generated for the current session and for the current user.

Packet types:

- **SIGNUP_REQUEST**: request to create a new user.
  The data for this packet is a null-terminated username follows by a null terminated password. E.g. *username\0password\0*
  The session token included in the header must be 0.

- **LOGON_REQUEST**: request to logon to server as an existing user
  The data is similar to that of SIGNUP_REQUEST.

- **LIST_REQUEST**: request the server to return the list of files belong to the current user. This packet has no data (only header)

- **FILE_REQUEST**: request the server to transfer a file that the server has. The data is a null-terminated file name. E.g. *my_music.mp3\0*

- **LEAVE_REQUEST**: client requests to leave. This packet has no data (only header).

- **TOKEN_RESPONSE**: this is a server response to SIGNUP_REQUEST or LOGON_REQUEST, if authentication is successful. The session token in header is set to a random token unique to the current session and user. There is no data for this packet.

- **LIST_RESPONSE**: this is a server response to LIST_REQUEST. The data is a variable length list of file name and the corresponding file checksum. File name is a null-terminated string, padded to a length of 64. File checksum is a 32-bit integer, stored in network byte order.

- **FILE_TRANSFER**: transfer a file. This packet can be sent by both server and client.
  + If sent from client, the data consist of the file name (which is a null-terminated string, padded to 64 bytes), followed by the binary content of the file.
  + If sent from server (as a response to FILE_REQUEST), the data is just the content of the file.

- **FILE_RECEIVED**: server confirms that a file is successfully received. There is no data.

- **ERROR**: server indicates that an error happen when handling a client's request. The data is an 8-bit integer specifying the type of error (names are self-explanatory):
  + EROR_UNKNOWN
  + ERROR_MALFORMED_REQUEST
  + ERROR_SERVER_BUSY
  + ERROR_USERNAME_TAKEN
  + ERROR_INVALID_PASSWORD
  + ERROR_FILE_NOT_EXIST
  + ERROR_FILE_UPLOAD_FAILED

2) Server implementation

2.1) Multithreading/multiplexing

The server repeatedly uses *select()* in order to select client sockets with pending request. It then handle those requests one by one. After all pending requests have been handled, the server then repeats *select()* to get new pending requests.

2.2) Authentication

Passwords are hashed using a library implementation of MD5, which produces 128-bit hashes. The server stores all username/password hash pairs in a file. The format of the file is a list of username and password hash pair. Usernames are null-terminated string, padded to length of 128 bytes. Password hash is simply a 128-bit integer (in binary form).
When authenticating or creating a user, the AuthenticationService crawls through the file to check if the username exist/password hash is correct.

2.3) Users' files storage

For each user, the StorageService creates a directory, which stores all files belong to that user.

3) Client implementation

3.1) Authentication

- The client prompts for username and password. Password is prompted through the function *get_pass()* which is a built-in secure function to input password.
- The username and password are passed to server using the LOGON/SIGNUP request.
- If authenticated, the server sends back a session token, which will be included in further requests.

3.2) LIST

- The client simply requests the list of files from server, then display them.

3.3) DIFF

- Using the checksum received from LIST, the client compare them pair-wise with the checksums of the client-side files. It then marks the files missing from server and missing from client.
- The checksum algorithm used by both server and client is CRC32.

3.4) SYNC

- Using the list of missing files from DIFF, the client first upload to server all files that are missing from server (using FILE_TRANSFER packet)
- The client then download all files that are missing from client, using FILE_REQUEST request.s