p2p

To Execute

Supported-Commands

Working

1. At Least one tracker will always be online.
2. Client needs to create an account (userid and password) in order to be part of the network.
3. Client can create any number of groups(groupid should be different) and hence will be owner of those groups
4. Client needs to be part of the group from which it wants to download the file
5. Client will send join request to join a group
6. Owner Client Will Accept/Reject the request
7. After joining group ,client can see list of all the shareable files in the group
8. Client can share file in any group (note: file will not get uploaded to tracker but only the : of the client for that file)
9. Client can send the download command to tracker with the group name and filename and tracker will send the details of the group members which are currently sharing that particular file
10. After fetching the peer info from the tracker, client will communicate with peers about
    the portions of the file they contain and hence accordingly decide which part of data to take from which peer (You need to design your own Piece Selection Algorithm)
11. As soon as a piece of file gets downloaded it should be available for sharing
12. After logout, the client should temporarily stop sharing the currently shared files till the next login
13. All trackers need to be in sync with each other

System Calls Used:

Socket - Connection Calls:

1. connect : `connection_status = connect(client_socket, (struct sockaddr * ) &server _ address , sizeof(server_address));`
2. htons : `server_address.sin_port = htons((PORT));`
3. socket : `client_socket = socket(AF_INET, SOCK_STREAM, 0);`
4. close : `close(client_socket);`
5. bind : `bind(server_sd, (struct sockaddr *)&server_addr, sizeof(server_addr));`
6. listen : `listen(server_sd, 5);`
7. accept : `accept(server_sd, (sockaddr *)&client_addr, &client_addr_size);`

Multi - Threading

1. pthread : To create and execute c threads (`pthread_create`)
2. pread : To read parallelly from threads
3. pwrite : To write parallelly from threads
4. pthread_join: To join threads

Sys/Stat Calls:

1. stat st ⇒ To obtain file size `(using st.size( ) )` while uploading the file , this information is sent to tracker so that it can maintain a bit-map of chunks a particular peer has.

Message Passing:

From Socket.h library following were used to implement functionality of ***sending and receive***

1. *send*() : function shall initiate transmission of a message from the specified socket to its peer. The *send*() function shall send a message only when the socket is connected (including when the peer of a connection-less socket has been set via *connect*()).
2. The *recv*() : The recv() function receives data on a socket with descriptor socket and stores it in a buffer. The recv() call applies only to connected sockets.

Hashing Calls:

To obtain SHA1 (hash) of a file so that intgerity of each and individual chunks can be verified. following system calls are used

1. SHA1 : from sha.h library
2. sprintf : To Write formatted output to S

Concepts Used:

- **Clarity of Distributed - Architecture Program Building**
- **BIT Mapping / File Intergration - Division**