Note: Extension has been implemented in my program

Program Design:

My programs contains two files, server.py and client.py.

Server.py, namely is the server for this program. The IP address of server is set to localhost and the port number is provided as a command line argument. (This can be changed on line 739 in server.py) Server will try to setup the server socket and listens for any incoming connections. For any incoming connection, the server sets a new thread for this connection, therefore, many connections can send messages to the server at the same time. The server will try to receive any messages sent from the clients and interpret these messages according to the rules. Many error checking has been implemented to make sure the server is stable. Global variables are used in server to record the connection data.

Client.py, has two sockets, one socket (client_socket in my program) is used to connect and communicate with server, the other socket (p2p_socket) is used to accept any private connections. Select.select() function is used to deal with the blocking issue of the sockets, otherwise one socket may be blocked forever until some message is received. Client.py tries to deal with the sockets returned from Select.select() individually and do different actions with different sockets. For example, if socket is the socket connects to server, it will try to receive message from server, else if socket is system.stdin, it will try to read in user input and sends these input to the server. Basically Select.select() is a way to deal with blocking sockets without setting socket to non-blocking.

Throughout the whole program, lots of error checking is implemented to make sure an error will not kill the server or client (not as bad as kill the server)

Extension:

Extension has been implemented in my program. For a file to be available, one has to register the file with the server by specifying file name and number of chunks to break this file. Before sending the command to the server, client will check if this file exists and the total number of bytes in the file to determine the chunk size. These information is combined together to sent to the server. To make it easier, say if a file has ten chunks, these chunks are named 0-9, after registering, server will show user has the file, chunk [0,1,2, ... ,9]. SearchFile and searchChunk behaves quite similar, searchFile shows any user has one or more chunk of a specified file, while searchChunk provides some specified chunk to search for, it will display any user has one or more chunks of the chunks specified. Download command will download specified file name, chunk number. Download manually should work perfectly while there are some problem with downloading file automatically (not specified in assignment), automatic downloading works but it fails to register each chunk with the server. When a file name and chunk number is given in the command, server will check if any user online and has not blocked has this chunk. It will return a list of all users available. No algorithm has been implemented, the program simply picks a random user and requests for the chunk. The luckily chosen client will then send the specified chunk through the private connection. Reading file and writing file are done by using fseek to move the pointer to the correct position. It will automatically combine the chunk in the correct position using seek. If one tries to manually download a file in order, i.e. download text 0, download text 1… , file will be fine. Otherwise if one tries to download a random unordered chunk, it will leave some binary non ascii characters in the file, but after all chunks have been received, it should contain none of those non ascii characters.

After receiving a chunk, the client requested this chunk will register this chunk with the server. Server can then consider this client as a source as well in the next round of sending.

Improvement:

How to identify who send the message is particularly hard on the client side. I implement a name tag to send together with the message, so client can check this name tag and tell who sends the message. Server uses <server> tag when sending, so an obvious issue would be if a user's name is 'server', client has no way to differentiate between server and this client. For my program, **please avoid using username 'server', 'server-p2p', 'server-p2p-file' when testing as these names is used to identify server**

It could be better if a receiving buffer is implemented in both server and client side. I sometimes notice I am getting several messages received at the same time. It will be great if server can know when a message is complete and ignore the rest of the message.

Even with many exceptions checking implemented, I believe there are still some assumptions about what command to send. My program can handle CTRL + c logout by client and many other exceptions, but the more error checking implemented, the more stable server and client program will be

Message format: (for all assume sender is A)

- **message <user> <message>**

  This command will send <message> to <user>, an error message will be displayed if <user> is invalid or has blocked A, an error message will be displayed in <user> is A, an error message will be displayed in message is None. If <user> is offline, <message> will be stored in offline messages and sent to the <user> when <user> logs on. <message> sentences are separated by space

  Example: message B how are you

- **broadcast <message>**

  This command will broadcast <message> to all the users online, if any user has blocked A, a warning message will be displayed, but it will still broadcast other users that has not blocked A. <message> sentences are separated by space

  Example: broadcast how are you

- **whoelse**

  This command display the names of all users that are currently online excluding A.

- **whoelsesince <time>**

  This command will display the names of all users who were logged in at any time within the past <time> seconds excluding A. An error message will be displayed if <time> is not a number

- **block <user>**

  This command will block <user>, an error message will be displayed if A has already blocked <user> or <user> is invalid. <user> must not be notified for this. This will block any message <user> sends to A

- **unblock <user>**

  This command will unblock <user>, an error message will be displayed if <user> has never been blocked or <user> is invalid. This will remove the effect of blocking

- **logout**

  This command will logout A

- **startprivate <user>**

  This command will setup a private direct connection from A to user, an error message will be displayed if <user> is invalid, not online or self. <user> will be notified for the setup of this connection

- **private <user> <message>**

  This command will send <message> to <user> through the private connection. An error message will be displayed if <user> is invalid, not online, self or A has not setup a private connection with <user>. <message> sentences are separated by space

- **stopprivate <user>**

  This command close the private direct connection between A and user, an error message will be displayed if <user> is invalid, not online or self. An error message will be displayed if A has never setup a private connection with <user>. <user> will be notified for the closure of the connection

- **register <file> <number_of_chunks>**

  This command will register <file> to the server and separate the file in <number_of_chunks> chunks. An error message will be displayed if this file has already been registered at server or does not exist in client's directory. The system will read the file and calculate the chunk size automatically.

- **searchFile <file>**

  This command will display the availability of <file> in the system. An error message will be displayed if <file> does not exist in server. The server will reply back with either 'Not available' or 'Available' along with list of online users that have one or more chunks of the requested.

- **searchChunk <file> <chunk_num>**

  This command will display the availability of <file>, chunk <chunk_num> (can be many chunks separated by space) in the system. An error message will be displayed if <file> does not exist in server or chunk does not exist in server. The server will reply back with either 'Not available' or 'Available' along with list of online users that have one or more chunks of the requested.

  E.g. searchChunk text 0 2 3

- **download <file> <chunk_num> (optional)**

  This command will download <file> from server, if <chunk_num> is not provided, the server will download the whole file if possible. Otherwise, it will download the specific <chunk_num> of the <file> required. An error message will be displayed if <file> has not been registered in server or <chunk_num> of <file> does not exist in server (automatically downloading does not work at the moment, and it is not required in the assignment spec, please include a chunk number when testing)