**Design Document**

I created a common header file for providing some general functions and basic data structures.

The msg struct is used to be transferred among server and peers. The struct have a int enumerable flag from 1 to 8. Flag = 1 shows that this message comes from specific peer to server for registration. Flag = 2 shows this message is used for logging in. Flag = 3 means this is a chat message; server will transfer the content to the destination peer. Flag = 4, this message will help server to register the published file from a peer. Flag = 5 shows this message is used to search the required file using keywords. Flag = 6 will let server query peer information and send back to request peer, as well as Flag = 7 and flag = 8 shows this message is created for file transformation. Peer struct will store the peer id and peer name. File struct have 4 elements, they are file id, file owner and file size and its name which contains the file absolute path plus name.

Besides the above structs, I already packaged some common functions.Function i_init() help server side and client sides initialize the UDP style server socket, meanwhile I set the socket option SO_REUSEADDR so the sever can be restarted using the same sever port(8081). I prefer to UDP socket rather than TCP because the previous method is very useful to do with short connections and send message more efficiently and effectively. Function i_socket(), i_bind(), i_recvfrom(), i_sendto() are designed for assembling the system level functions. So I need not to handle the invoking exceptions every time.

I created both i_lseek() and i_fseek() functions, the reason is only consider about convenience and save time to exchange file descriptor with file stream. Functions i_open(), i_read() and i_write() provide the services to process with files for storing file and peers information. i_listFiles() and i_listPeers can show all the registered files and peers. i_get_time() could print the cuurent time and make the server and peers get more reasonable and readable letters.

Every time Client peer need to be launched with the server IP address. Although I defined a start port(8089) that would not be occupied by other applications frequently, I have to create a get_max_port () and find a free and available port for our peer client applications. In the initialize() function I assign the server's IP address using main function's parameter. Originally, I have no better method and must invoke the ifreq struct to get my and server IP address and hard code them for testing in the same system, at last I researched system APIS and then found the hostent struct can help me evaluate my address and server. After logging in, since we set the peer's sin_addr to 0.0.0.0(INADDR_ANY), so now we should enable it with detailed IP.

When I created exit_system(), I just use kill(0, SIGABRT) and result in core dumped problem, I feel that is not a good method and update to invoke linux commands to kill our processes. The menu() should be sleep 1 second to avoid the output mixed with the child process for clearer layout. In the opposite, without the sleep(), the outputs have some unreasonable stream.

For the message control components, I spitted the communication methods into senders and receivers functions, that means every action is more weak coupling. For example, based on workflow, first step, we can invoke the register_peer() function to get my name, build the reg_msg to send to server and wait for server reply. If registered successfully, I got my id and continue to invoke my_logon() method. Second step, in the function my_logon(), I can choose to login, continue to register or exit. As same as register_peer(), login_peer() also gather login id and build the log_msg which be sent to server latter. If login failed, we would loop in the my_logon().

Function send_message() and chat_msg_recv() can be used to send and receive the peer chat message and these functions are my first testing for p2p connection in UDP style.As the following, I tried creating two functions share_file() and share_file_recv(). In such codes, I already tested the performance with 500 files loop with 0.1ms gap, the result is I am very satisfied with it. The share_file_recv() shows all the 500 insertion feedbacks. search_file() scan the query keywords as well as build the query_msg which then be parsed by server. file_list_recv() should handle the message from the server. I define end symbol(& $) as the parameter of strtok() function.

Sscanf() can help us parse the received message content and get the file sharer IP address, which can be used in put file and get file functions, we dynamically assemble the destination peer address and communicate to each other. For the fread() return value we use offset(err) to store and send to destination peer before fclose current file stream.

To make sure the peer.cfg is filled with fixed line length. We should use fprint() with each element in width of 30 chars. search_file() created "$" symbol to gather all matched results in msg_recv content, if and only if the content have available space.

I'm afraid that this small application need to enhance the server response performance. Although I use multi processes to handle the clients' request and write the corresponding files , the server also will missed some requests if I disable the sleep(). Maybe next time I could upgrade I/O multiplexing within multi threads, such as select(), epoll() and thread pool.