COMP3331 Project Document

Program Design

The Client and Server interact via two sockets, with each socket consisting of a reader thread and a writer thread pair. The writer initiates an interaction, and the reader responds accordingly. The overall sequence is:

1) When first started, the Client begins a writing thread for processing user commands. During initialization, a corresponding thread on the server side as the “server-read” thread is created, which reads from the connection and processes user commands.
2) Using this Client-Write, Server-Read connection, the authentication process happens
3) Upon successful authentication, a Client-Read, Server-Write connection pair begins.
4) The Server-Write Thread sends to the user any jobs it receives (messages, presence notifications etc.)
5) The Client-Read Thread accepts the above jobs, and prompts the user as necessary
6) The Client also begins a non-blocking thread for reading the user input from the terminal, so that other threads can do process jobs. User input is appended to an ArrayList<String> of userInputs
7) General exchanges then happen as described in the threads’ functionality, as described above, starting at the Client’s “main-write” thread

Private messaging is initiated as follows:

1) The Client’s write thread sends a startPrivate command to the server, with the “user” field set
2) The server checks that “user” has not blocked the client, and other error conditions like invalid user, user already in a p2p exchange, and so on. If there are errors, the server blocks the request
3) Otherwise, it is passed on to the recipient, who either accepts or declines. If it is accepted, 1) a ServerSocket is created waiting for the original client’s connection request, and 2) its port number is added to the response to the server. If declined, this is set in the response to the server
4) If the original Client receives an “accepted” message, it tries to connect to the ServerSocket with the provided port Number
5) If successful connection, Private Message Exchanges follow

Data Structure Design

There are the following Main Additional Classes / Data Structures

- All jobs (messages, broadcasts, …) internally are stored as type HashMap<String, String>. For Application Layer Transport, the MessageMapParser class helps to convert a HashMap to an XML format described below, and vice versa. The Application Layer Format Table describes all jobs in detail
- A User Class has general User Attributes like username, password, however two notable structures are: 1) ArrayList<String> blockedUsers; a list of users blocked by this user, and 2) ArrayList <HashMap<String, String>> jobs; a list of jobs that have to be sent to this user by this user’s Server-Writer, for example a message that another user wants to send to this user will be appended to this list
- A UserLog class maintains Information of a user’s particular session instance with three variables: 1) username 2) log in Timestamp 3) log out Timestamp. The server maintains an ArrayList of userLogs, called onlineHistory, which is used for the whoelse, and whoelsesince commands

General Application Layer Message Format:

The message format follows the general XML format, with a single modification as follows:

If no attributes: <tag >body</tag>

If attributes: <tag field1="..." field2="..." >body</tag>

Modification: There is a single space before the ending '>' in the opening tag in both cases, for easier parsing

Tag	**Field**	**Value(s)** (one of the list elements)	**Meaning**
/* These are general fields and values */	status	recvd	tag recvd
		try-again	something went wrong – try again
	missing	field, or tag	Eg if a <msg> tag has no 'type' field, a response will have 'missing':'type'
	invalid	field value, or tag	similar to the above, except for an invalid field value
thread	client	[write, read]	the client sets-up 2 threads with the server: 1) the client writes first, the server waits and listens 2) the server writes first, the client waits and listens Described in more detail under Program Design.
	mainPort	port number	the port number of the main client-write socket, sent by the client-read thread
auth	status	[username, enter-password, already-online, new-user, password, matched, try-again, blocked, registered]	username -> client sends username password -> client sends password enter-password -> server accepts username, asks for password already-online -> user already online new-user -> username doesn't exist, begin registration process matched -> received password matched try-again -> received password didn't match blocked -> received password didn't match, thrice, now blocked for block_duration registered -> new user registered
	username	username value	
	password	password value	
login	user	username	user who logged in
logout	user	username	user who wants to logout
	status	[done, already-out]	done -> logged out, and other actions taken already-out <-
msg	type	[one, all]	one -> msg to single user all -> broadcast
	fromUser	username	the sender

	toUser	username	the recipient
	returnStatus	[no-such-user, self, blocked, sent, sent-all, sent-some]	type=one no-such-user -> toUser does not exist self -> toUser == fromUser blocked -> toUser has blocked fromUser sent -> message sent to toUser type=all sent-all -> msg sent to all users (except self) sent-some -> msg sent to all, except self and some users who have blocked fromUser
	body	the message	
online	type	[now, since]	now -> currently online since -> since timestamp
	time	x seconds	get all users online, since x seconds before now
	body	list of users, separated by " ", or "" if there are none	
block	type	[on, off]	on -> add to blocked users off -> remove from blocked users
	user	username	user to block / unblock
	blockStatus	[false-self, false-already, true]	false-self -> cannot (un)block self false-already -> user already (un)blocked true -> successfully (un)blocked
private	type	[start, stop]	start -> start new private session stop -> stop existing private session
	mode	[initiate, accept, decline]	initiate -> this user is initiating a new private session request accept -> this user is accepting a private session request decline -> this user is declining a private session request
	user	p2p session partner	
	privateStatus	[no-such-user, self, offline, blocked, accepted, declined, timed-out, sent-response]	no-such-user -> user does not exist self -> user is self, can't start p2p session offline -> user is currently offline blocked -> user has blocked the requestor accepted -> request accepted declined -> request declined timed-out -> the request has timed out sent-reponse -> response sent
	listeningPort	port number	if user accepts, this is the port number on which user's ServerSocket is listening