Design Report: OS Messenger App

Group Members
Ali Malik (250901628)
Ali Chouman (250883895)
Yazan Al-Lahham (250893700)
Andrei Popescu (250897291)

Abstract

The following chat application implements the concepts of multithreading, socket programing, user interface design, and semaphores. A user can locally launch the Python-based client application on their device and are then prompted to join a chat room in which they can communicate with other users. The server dictates how many chat rooms are available which can be modified by the administrator and the client responds by indicating which chat room they are in. The strategy utilized was to append the chat room number with each corresponding client message.

Server

The SocketServer uses one single thread as there is no need to separate execution paths. However, each time the server accepts a new connection, a socket thread is generated and stored by the server within a vector. This is because the dynamic memory can be safely deallocated upon program termination. The design uses the idea that each socket thread pertains to a specific client and this is important to the server because it allows it to distinguish between chat rooms from a high-level perspective. Since only one socket is used for all communication, all clients must be blocked when a Read() call is issued to the socket. The server thread contains the semaphore that is the owner of all socket thread semaphores. To issue a block on Wait(), the provided Semaphore class links all semaphores with the same name and so, the port number of the socket is used as a unanimous reference. In order for clients in the same chat room to see their messages and not to see messages from other rooms, the server only allows Read() calls from clients who correspond to the given socket thread's chat room number. The server uses FlexWait and Blockable classes to allow background thread processing to go on forever.

Client

The application synchronizes with the server responses so that the client can see which room they are currently in, when other clients leave the chat, and which user is messaging (based on username). The client threads each Receive() call so that a server response is obtained immediately. The client is required to inform the server of which room they are currently in and this is achieved by appending their chat room number to their message being sent so that the server can parse them separately. Python’s Thread and Socket libraries correspond to the server’s C++ libraries and there are no discrepancies with socket Read() and Write() calls.

Graceful Termination

In the server, each socket thread safely removes themselves from the vector of referenced client threads by issuing a Wait() through their own semaphore and freeing themselves from server memory. When the server abruptly stops or is asked to shut down, the server thread destructor is responsible for looping through all references of created threads, destroying those objects, and deleting the vector holder itself by a vector swap() with an empty vector.