Rene Rumetshofer Jakob Mayer

TW Mailer PRO

Server multithreading

To enable multithreading on the server whenever a new connection that is not blacklisted is made, a new thread is created that executes the clientHandler. This way multiple connections can be managed in concurrence. This also enables each thread to keep the user context as long as the connection persists.

Multhithreading synchronisation

Since the server now uses multiple threads that need to access the same resources (Mail Spooler and Blacklist), there could be problems when two threads try to access the same resources at the same time. To prevent this from happening, mutexes are used to lock the corresponding spooler subdirectory on a per user basis. This ensures no unnecessary locking of the whole spooler directory, massively slowing down the server, whilst still safeguarding from file corruption.

Server architecture

The Server is currently now multithreaded. In the top of the *server.cpp* file the ***main*** function which sets up the socket and waits for a connection. Once a client connects, the socket is passed on to the ***clientHandler*** function in a new thread which waits for a valid command to be sent.
When a valid command is sent from the client, the ***clientHandler*** passes the socket to another ***handler*** which is responsible for this command. The various handlers can be found in the *handlers.cpp* file. After completing the wanted command the called handler exits back to the ***clientHandler*** which then, again, waits for another incoming command or for the connection to close.

Client architecture

The client connects to an IP and port that is specified in the start arguments.
In the interactive mode (no –t flag set) the client prompts the user for the wanted commands and data strings like subject or message body. It then sends the command and collects the reply from the server which it later displays. The client structure is identical to the server, with a main loop waiting for a valid command to be prompted and then passing on the socket to the corresponding handler. There is a wrapper function for the interactive mode which prompts for the needed inputs before passing them on to the handler.

Used technologies / Development strategy

There were no special technologies used besides lldap. The entire program is written with standard libraries which would be too many to list here. They can be found in the declarations of the .cpp files.

Out development strategy was to split our workload into the server and client part and to have each of us working on one of them. There are also "shared" files that both programs use, namely the message formatter for the mails (message.cpp), and the utility functions for receiving and sending of lines and the verifying of usernames. They can be found in the "shared" directory.

After completing both programs, we reviewed each other's source code and corrected any problems there arose when they tried to communicate.

To work together efficiently we used GitHub for version control.

Needed adaptations (Handling of large Messages)

We did not encounter any major problems asides one.
The readline() function always read a whole tcp packet and since tcp is a streaming protocol there were situations where when too many sendLine() commands were sent too soon after each other, the packets got combined by the tcp stack buffering and sent together, ruining the line separation.

This was fixed by writing a new readLine() function which only reads one Byte at a time and concatinates them in a string until a "\n" is read in.

GitHub

Our sourcecode can also be found and cloned from our GitHub repository:

https://github.com/ReneRumetshofer/TWMailer