Protokoll TW-MailerPro

Wir verwenden C/C++ und die wichtigsten libraries die wir verwenden sind: signal.h, thread, mutex, filesystem, fstream und diverse socket libraries.

Um bei unserem Projekt eine gewisse Struktur beizubehalten, verwenden wir ein selbst erstelltes Message Objekt. Dieses initialisiert sich selber, wenn man die SEND Funktion beim Client aufruft und befüllt alle notwendigen Felder außer den Sender an sich. Dieser wird später Serverseitig befüllt um zu verhindern, dass man sich Clientseitig irgendeinen User eintragen kann.

Das Senden an den Server funktioniert grundsätzlich so, dass das Message Objekt eine eigenständige Funktion aufruft, die dieses in einen zusammengesetzten String verwandelt. Auf der Server Seite gibt es natürlich die Möglichkeit des Message Objektes sich wieder in ein Objekt zusammen zu setzen. So ist der Umgang mit der Nachricht an sich einfach und ermöglicht uns die Daten gesammelt zu verwerten.

Unser Nachrichtenspeichersystem funktioniert so, dass im angegebenen Pfad(oder wenn nicht angegeben im Root Ordner) der beim Start des Servers mitgegeben wurde, ein Receiver Ordner erstellt wird – außer er existiert bereits. Dieser beinhaltet standardmäßig ein index.txt File, welches später zur leichteren Weiterverarbeitung genutzt wird und einen „inbox" Ordner, welcher ein Textfile pro User speichert der diesem Receiver eine oder mehrere Nachrichten geschrieben hat. In das Index File werden abwechselnd ID, Sender, und Subject festgehalten.

Ein zusammengesetzter String schaut je nach Funktion/Befehl grundsätzlich so aus: BEFEHL\n *gefragte Informationen*\n …

Durch die Vorschrift des Befehls kann der Server zwischen den einzelnen Funktionen unterscheiden und ruft separate Funktionen für jede dieser auf…

LOGIN & Blacklist

Clientseitig wird ein String aus Username und Passwort zusammengebastelt und dem Server übermittelt. Jeder Thread behält die Versuch im Auge wie oft man sich versucht einzuloggen. Bei 3 Versuchen erhält der User die Nachricht „Locked" und die IP wird auf ein Textfile notiert. Direkt darunter wird außerdem der aktuelle Timestamp des Servers festgehalten. Grundsätzlich wird bei jeder LOGIN anfrage geprüft → ist der Username bereits auf der Blacklist. Wenn ja schau dir den Timestamp an und vergleiche ihn mit der aktuellen Zeit. Beträgt die Differenz mehr als 60 Sekunden wird der User von der Blacklist gelöscht und kann sich wieder einloggen.

Serverseitig wird bei erfolgreichem Login eine boolsche „isLoggedIn" Variable festgehalten. Die essentiell dafür ist welche Befehle dem User angeboten werden und ob er diese benutzen darf.

SEND

Wie schon erwähnt wird der String Server-seitig wieder zu einem Objekt zusammengebaut. Dieses wird dazu verwendet ein neue Verzeichnisse anzulegen oder sich in bereits bestehendes rein zu navigieren. Seine Werte werden anschließend in ein Textfile gespeichert in dem alle bisher von der Person gespeicherten Nachrichten liegen. Die {MessageID} und das {end of message} begrenzen dabei die Nachricht und erleichtern uns das spätere auslesen.

LIST:

Das index.txt File, welches alle Nachrichten die nicht gelöscht wurden strukturiert ablegt, machen wir uns jetzt zunutze. Durch eine einfach Modulo(%) Konstruktion lesen wir aus der eben genannten Textdatei - Id und Subject. Das hält der Server in einem Counter fest und gibt alles laut Formatvorlage aus.

READ:

..ist nichts anderes als die Umkehrfunktion des SEND Befehls. Wir suchen in der entsprechenden User Textdatei, welche wir mithilfe einer separaten Funktion und der angegebenen MessageID erhalten, nach der gewünschten Nachricht. Diese wird genau wie vorher in ein Message Objekt gespeichert und durch die integrierte Funktion direkt in einen versendbaren String umgewandelt. Diesem wird noch ein OK\n drangehängt und dem Client gesendet.

DEL:

Mithilfe des index.txt Files des Users findet der Server heraus zu welcher Nachricht.txt die ID gehört, die der User Löschen will. Nachdem er sich in dieses navigiert hat, löscht er die Nachricht gezielt durch unserer {MessageID} und {end of Message} Marker restlos. Anschließend nummeriert er alle Nachrichten neu durch und ändert dies auch beim index.txt file um so entstandene Lücke zu schließen. Das Löschen funktioniert so, dass der Server ein temp.txt File erstellt und dort alles reinkopiert außer die zu Löschende Nachricht. Danach wird die Ursprüngliche Textdatei gelöscht und der Name der temp.txt Datei durch die Originale Bezeichnung geändert.

QUIT:

Wird dem Server der Befehl QUIT übermitteln - bricht das die Serverseitige While-Schleife die anschließend die Verbindung mit dem Client schließt.

Threading & Mutexes

Nachdem alle Servervorbereitungen abgeschlossen sind und der Server bereit ist auf Anfragen zu reagieren beginnen wir die clientCommunication Funktion zu threaden. Wir haben uns fürs threaden entschieden, weil es einfach zugänglich ist und das inkludieren in unsere Architektur schneller funktioniert als Forken. Ist der Teilprozess erstmal erstellt wird er detached und der Main thread ist wieder bereit auf Anfragen zu reagieren. Um zu verhindern, dass die Threads sich im Weg sind oder Textüberlagerungen entstehen, verwenden wir gezielte Mutexes (lock(), unlock()).

Anpassungen

Zu allererst mussten wir die Login Funktion inkludieren und die Blacklist zum Laufen bekommen. Erst bei erfolgreichen Login über LDAP sollte es möglich sein die anderen Befehle aufzurufen. LDAP war schnell aufgesetzt und funktionierte bis auf Kleinigkeiten einwandfrei. Die Befehle konnten gut adaptiert werden und es brauchte nur vereinzelnde Änderungen bei dem Message Objekt um den User des LOGIN Prozesses anstelle des ständigen Inputs zu verwenden. Threading bereitet auch nicht allzu viele Umstände, außer verschieben einzelner Variablen und kleine Änderungen wie wann und was übergeben wird.