Salma Morad, Andor Kurucz
BFI 3A

Protokoll TW_Mailer

Unser Projekt, TW_Mailer, ist eine Client-Server Applikation mit dem Fokus darauf, Nachrichten zwischen zwei Parteien auszutauschen.

Projektstruktur

Wir haben uns entschieden, die Lösung objektorientiert anzugehen, da wir es als übersichtlicher und sauberer empfinden. Dazu haben wir die wichtigsten Komponenten in mehrere Dateien unterteilt. Die beiden Hauptpunkte des Programmes, der Client und der Server, sind in zwei Klassen mit jeweils drei Dateien aufgeteilt:

- ***server.cpp/client.cpp:*** das sind die Einstiegspunkte des Programms, da sie die main-Funktion der jeweiligen Klassen enthalten.
- ***serverClass.h/clientClass.h:*** enthält die Definitionen der Variablen und Methoden der Klassen.
- ***serverClass.cpp/clientClass.cpp:*** hier ist der meiste Code vorhanden, da diese Dateien die Implementation der jeweiligen Klassen enthalten
- ***allHeaders.h:*** diese Header-Datei speichert alle erforderlichen Bibliotheken für das Programm.

Programmstart

Wir haben eine **Makefile** erstellt, die das Programm über die Kommandozeile automatisch durch die Eingabe von ***make*** in dem richtigen Verzeichnis kompiliert. Dazu muss Linux installiert sein. In unserem Fall haben wir alles über **WSL** (Windows-Subsystem für Linux) gelöst. Anschließend zur Kompilation muss man den Server starten, dies geschieht durch den command ***./server <port> <mail-spool-directoryname>***. Der letzte Parameter dient dazu, entweder einen neuen Ordner für die vom Server gespeicherten Nachrichten

Um sich nun als Client zum Server zu verbinden, wird in einer neuen Konsole der Befehl ***./client <ip> <port>*** eingegeben. Hierbei ist zu beachten, dass der gleiche Port eingegeben werden soll, auf dem der Server gestartet wurde. Anschließend muss man sich anmelden, um Zugriff auf die Programmfunktionen zu bekommen.

Username und Password des Clients wird an den Server übergeben, der wiederum die Anmeldedaten mit dem LDAP-Server der FH Technik. Nach erfolreicher Verifizierung übergibt der Server den Erfolg zurück an den Client, womit nun der Client zugriff auf sämtliche Befehle hat.

Benützte Technologien

C++ Code

Wir nützen objjektorientierte Programmierung für bessere lesbarkeit und code-maintenance.

LDAP Authentifizierung

Die LDAP Authentifizierung erfolgt exklusiv durch den Server, der mit dem LDAP-Server der FH Technikum kommuniziert.

Termios library

Um die Passworteingabe im Terminal zu verstecken, wird der canonical mode deaktiviert, sodass die Eingabemanipulation durch den code bereits während der Eingabe passieren kann. Für jeden einzelnen Character wird das Echo, die Ausgabe im Terminal, deaktiviert.

Algorithm und Ctype library

Damit die Befehle seitens des Clients unabhängig von Klein-und Großschreibung sind.

TCP/IP und Sockets

Wir benützen Sockets für low-level Kommunikation zwischen Client und Server

Mail-Spool and Message-Storage

Wir speichern die gesendeten Nachrichten in der „mail-spool“-directory als seperate CSV-Datein. Aus testing Gründen werden in unserem Code die CSV-Datein für den jeweiligen Sender erstellt, um sämtliche Kommandos einfacher zu überprüfen. In weiterer Folge würde man die Nachrichtenspeicherung natürlicherweise für den Empfänger der Nachrichten implementieren.

Multi-threading

Seitens des Servers wird Threading verwendet, damit der Server mit multiplen Clients umgehen kann. Gemeinsam verwendete Ressourcen werden durch Mutexes abgesichert, um gleichzeitige Zugriffe zu vermeinden.

Signal Handling

Der richtige Umgang mit SIGINT-Signalen, damit alles nach Bedarf korrekt terminiert wird.

Entwicklungsstrategie

Es wurde zuerst der Client und der Server ohne Login- und LDAP-Implementation erstellt, um die Befehls-Funktionalitäten auf der Server-Seite zu testen.
In dieser Phase war der Name des Versenders hardcoded, also war die Login-Funktion noch nicht notwendig.
In weiterer Folge wurde die Login-Funktion eingeführt und diese Mittels logging seitens des Servers getestet. Danach wurde die Kommunikation des Servers mit dem LDAP-Server bewerkstelligt und die Login-Authentifizierung implementiert, womit dann auch die Befehl-Beschränkungen, mit und ohne Login, seitens des Clients eingeführt wurden. Nach dem Testen des bestehenden Codes haben wir schließlich Signal-Handling und notwendige Error-Messages verwendet.

Synchronisations Methoden

Wie oben bereits erwähnt, benützen wir Threads und Mutexes um die Arbeit mit multiplen Clients zu bewerkstelligen.

Zusätzlich wird in der LDAP-Funktion ldap_sasl_bind_s() verwendet, welches synchron ist. Der Funktions-Aufruf blockiert den jeweiligen Thread bis die Bind-Operation ausgeführt wird, bevor es die Kontrolle zurück and die Applikation übergibt.
Das Warten auf das Resultat durch die asynchrone ldap_sasl_bind()-Funktion wäre in unserem vom Nachteil und potentiell Fehlererzeugend in unserer Server-Logik.