SOCKET ÜBUNG

Arbeiten Sie in max. 2er Gruppen auf der lokalen Linux Installation in den EDV Sälen oder auf Ihrem Laptop (Linux VM oder Ubuntu Subsystem unter Windows 10).

Die untenstehende Beschreibung teilt die Aufgabenstellung in zwei Blöcke, um die initiale Komplexität überschaubar zu halten. Das vereinfacht den Einstieg, macht es jedoch auch notwendig, gewisse Teile der Implementierung zu ändern, sobald Sie Teil 2 umsetzen.

Es steht Ihnen allerdings frei, die erhöhte Komplexität in Kauf zu nehmen und die Elemente aus Teil 2 von Beginn an in die Umsetzung zu integrieren. Was zählt, ist das Endergebnis ;)

TWMailer Schritt 1

Erstellen Sie eine Client-Server Anwendung in C/C++ unter Linux zum Senden und Empfangen von internen Mails mithilfe von Socket Kommunikation.

1. Der Client wird mit einer IP Adresse und einem Port als Parameter gestartet
2. Der Server wird mit einem Port und einem Verzeichnispfad (Mailspoolverzeichnis) als Parameter gestartet und soll als iterativer Server ausgelegt werden (keine gleichzeitigen Requests)
3. Der Client connected mittels Stream Sockets über die angegebene IP-Adresse / Port Kombination zum Server und schickt Requests an den Server.
4. Der Server erkennt und reagiert auf folgende Requests des Clients:
   - **`SEND`**: Senden einer Nachricht vom Client zum Server.
   - **`LIST`**: Auflisten der Nachrichten eines Users. Es soll die Anzahl der Nachrichten und pro Nachricht die Betreff Zeile angezeigt werden.
   - **`READ`**: Anzeigen einer bestimmten Nachricht für einen User.
   - **`DEL`**: Löschen einer Nachricht eines Users.
   - **`QUIT`**: Logout des Clients

Hinweise

Die gesendeten Nachrichten sollen pro User im Mailspoolverzeichnis permanent gespeichert werden. Die Struktur des Verzeichnisses und der Speicherung bleibt ihnen überlassen (z.B. pro User eine Datei mit allen Nachrichten, oder pro User ein Unterverzeichnis mit einer Datei pro Nachricht).

Als Absender und Empfängeradressen werden nur Usernamen (ohne @domain) verwendet, es handelt sich also um Strings mit max. 8 Zeichen, z.B. if19b001.

Der Protokollaufbau des **`SEND`** Befehls ist wie folgt definiert:

```
SEND\n
<Sender max. 8 Zeichen>\n
<Empfänger max. 8 Zeichen>\n
<Betreff max. 80 Zeichen>\n
<Nachricht, beliebige Anzahl an Zeilen\n>
.\n
```

D.h. die Ende Kennung der Nachricht ist ein Newline, gefolgt von einem Punkt und noch einem Newline.
Der Server antwortet mit `OK\n` oder `ERR\n` im Fehlerfall.

Der Protokollaufbau des **`LIST`** Befehls ist wie folgt definiert:

```
LIST\n
<Username max. 8 Zeichen>\n
```

Der Server antwortet mit:

```
<Anzahl der Nachrichten für den User, 0 wenn keine Nachrichten
vorhanden sind>\n
<Betreff 1>\n
<Betreff 2>\n
…
<Betreff N>\n
```

Der Protokollaufbau des **`READ`** Befehls ist wie folgt definiert:

```
READ\n
<Username max. 8 Zeichen>\n
<Nachrichten-Nummer>\n
```

Der Server antwortet bei korrekten Parametern mit:

```
OK\n
<kompletter Inhalt der Nachricht wie beim SEND Befehl>
```

Der Server antwortet im Fehlerfall (Nachricht nicht vorhanden) mit:

```
ERR\n
```

Der Protokollaufbau des **`DEL`** Befehls ist wie folgt definiert:

```
DEL\n
<Username max. 8 Zeichen>\n
<Nachrichten-Nummer>\n
```

Der Server antwortet bei korrekten Parametern und erfolgreichem Löschen mit:

```
OK\n
```

Der Server antwortet im Fehlerfall (Nachricht nicht vorhanden, Fehler beim Löschen, etc.) mit:

```
ERR\n
```

Wichtig: Achten Sie auf korrekte Fehlerabfragen und korrektes Speichermanagement (keine Speicherlecks, keine Buffer Overflows). Beachten Sie die Richtlinien der C-Programmierung unter Linux. Es dürfen keine Zombie-Prozesse erzeugt werden!

Tutorials

Verwenden Sie zum Lesen aus dem Socket bis zum nächsten Newline z.B. die Funktion `readline()` aus dem Tutorial tcpip_linux-prog-details.pdf

TWMailer Schritt 2

Erweitern Sie Ihre Client-Server Anwendung zum Senden und Empfangen von Mails wie folgt:

- Der Server soll nun als nebenläufiger Server ausgelegt werden (parallele Requests von mehreren Clients). Sie können entweder `fork()` oder Threads verwenden.
- Achten Sie dabei auf etwaige Synchronisationsprobleme im spool Verzeichnis, falls parallel mehrere Mails an denselben User geschickt werden.
- Erweitern Sie das Protokoll um folgenden Befehl:
  - **`LOGIN`**: Der Client schickt Username und Passwort an den Server. Der Server verbindet sich mit dem FH LDAP Server und versucht den User zu authentifizieren. Bei ungültiger Anmeldung verweigert der Server die Befehle **`SEND`**, **`LIST`**, **`READ`** und **`DEL`** und wartet auf einen erneuten **`LOGIN`** Request. Ist der **`LOGIN`** erfolgreich, wird der Username für die weiteren Befehle wiederverwendet, d.h. es können Mails nur mehr mit dem authentifizierten Benutzer verschickt, empfangen und gelöscht werden.

Protokoll Hinweise

Der Protokollaufbau des **`LOGIN`** Befehls ist wie folgt definiert:

```
LOGIN\n
<LDAP Username max. 8 Zeichen>\n
<Passwort>\n
```

Der Server antwortet bei korrekten Parametern und erfolgreichem Login mit:

```
OK\n
```

Der Server antwortet im Fehlerfall (User nicht vorhanden, Fehler beim Authentifizieren, etc.) mit:

```
ERR\n
```

Im **`SEND`** Protokoll entfällt die Angabe des Senders (ist automatisch der Username des Logins), auch bei **`LIST`**, **`READ`** und **`DEL`** entfällt die Angabe des Usernamens.
`LIST`, **`READ`** und **`DEL`** antworten mit `ERR\n`, falls zuvor kein erfolgreicher **`LOGIN`** durchgeführt wurde.

Übersicht über alle Low-Level I/O Funktionen, wie z.B. `open, close, read, write, fcntl` findet sich im C von A bis Z Open Book
http://openbook.galileocomputing.de/c_von_a_bis_z/
Kapitel 16 und Kapitel 17
Die Filegröße kann mittels `stat()` realisiert werden, Auslesen von Directories mittels `opendir(), readdir(), closedir()`

LDAP Hinweise

Verwenden Sie für die LDAP Anbindung die OpenLDAP C API (z.B. Ubuntu Paket libldap2-dev, Include Datei <ldap.h>, gcc Option -lldap -llber)

Ein Beispiel LDAP-C Gerüst finden Sie im Download-Verzeichnis.
z.B. Aufruf für das LDAP Grundgerüst:

```
g++ -std=c++11 -Wall -o myldap myldap.c -lldap -llber
```

Unser LDAP Server kann wie folgt angesprochen werden:
Host: ldap.technikum.wien.at
Port: 389
Search Base: dc=technikum-wien,dc=at

Die Authentifizierung am LDAP Server soll in 3 Schritten realisiert werden. Zuerst muss eine anonyme Anmeldung durchgeführt werden. Danach wird im LDAP Verzeichnis nach dem User (uid) gesucht. Falls der User gefunden wurde, kann im 3. Schritt durch eine erneute Anmeldung mit dem vollständigen DN des Users (Resultat der Suche) überprüft werden, ob das Passwort korrekt ist, da nicht direkt auf das Passwort Attribut zugegriffen werden kann.

Beispiel:
Anonyme Anmeldung und Suche nach uid=nimm liefert als Ergebnis:
dn: uid=nimm,ou=People,dc=technikum-wien,dc=at
ACHTUNG: Anonyme Anmeldung ist nur FH-intern erlaubt!

Weitere Infos zur LDAP API: http://www.yolinux.com/TUTORIALS/LinuxTutorialLDAP-SoftwareDevelopment.html

Optionale Zusatzaufgaben (mögliche Bonuspunkte, um andere Fehler kompensieren zu können)

- Zeitlich begrenzte IP-Sperre

  Nach 3 fehlerhaften **`LOGIN`** Versuchen beendet der Server die Socket Verbindung zum Client und sperrt die IP-Adresse des Clients für eine gewisse Zeitspanne (Konstante mit #define definieren, z.B. 30 Minuten). Verwalten Sie daher am Server eine geeignete Datenstruktur für die gesperrten Clients.

- Erweiterung des **`SEND`** und **`READ`** Befehls um Attachments.

  Überlegen Sie sich für den **`SEND`** Befehl ein geeignetes Protokoll zur blockweisen binären Übertragung des optional vorhandenen Attachments (inkl. Dateiname, Filegröße etc.). Auch der **`READ`** Befehl muss dahingehend angepasst werden, dass ein optional vorhandenes Attachment als File lokal gespeichert wird.

- Mehrere Empfänger und Sent Folder.

  Überlegen Sie sich für den **`SEND`** Befehl ein geeignetes Protokoll zum Senden an mehrere Empfänger und überlegen Sie ein Format, wie eine redundante Speicherung in allen Empfängermailboxen vermieden werden kann. Weiters sollen gesendete Nachrichten in einer Sent Mailbox am Server archiviert werden.

- Verschlüsselung.

  Wie könnten Nachrichten verschlüsselt übertragen und/oder verschlüsselt am Server abgespeichert werden. Ist eine End-to-End Encryption möglich?

Abgabe

Allgemeine Hinweise:

- Der Quellcode sollte dokumentiert sein
- Geben Sie, wo nötig, sinnvolle Fehlermeldungen aus und fangen Sie Falscheingaben korrekt ab.

Im Abgabesystem ist ein .tgz File abzugeben. Dieses soll beinhalten:

- Alle Sourcen inkl. Code Kommentaren für Client und Server!
- Ein Makefile mit mind. Targets **all** und **clean**
- ausführbare Programme für Client und Server
- Beschreibung des verwendeten Kommunikationsprotokolls zwischen Client und Server.

Abgabeschluss siehe Moodle Kurs

Die Abgabe muss in einem **Code-Review** in der folgenden Übung präsentiert werden (max. 30 Punkte)! Beim Code-Review herrscht Anwesenheitspflicht.

Benotungskriterien

Es können max. 30 Punkte beim Code Review erreicht werden. Insgesamt sind >15 Punkte für eine positive Note notwendig!

- Vollständigkeit der Abgabe inkl. Makefile und Protokoll 2
- Implementierung und Korrektheit Server
  - SEND 3
  - LIST 3
  - READ 3
  - DEL 3
- Implementierung und Korrektheit Client 2
- Parallelität und Synchronisation 5
- LDAP Anbindung 3
- Error Handling 2
- Code Verständnis, Gesamteindruck 4

Max. 3 Bonuspunkte pro Zusatzfeature