Beschreibung Kommunikationsprotokoll

1. Kurzbeschreibung

Die Kommunikation zwischen Mailserver und Mailclient findet über Sockets statt. Dabei wird jeweils von Server und Client abwechselnd gesendet und empfangen, um eine synchronisierte Kommunikation zu gewährleisten.

2. Verfügbare Operationen

Der Austausch von Informationen basiert auf mehreren verfügbaren Operationen. Diese Operationen beinhalten folgende Befehle:

- **Send:** sendet eine Nachricht mit Empfänger, Betreff und Inhalt an den Server, welcher diese in einem personenbezogenen Ordner im Mailspool-Verzeichnis abspeichert
- **List:** fordert vom Server eine Liste an Nachrichten (nur Betreff-Zeile) für den eigenen Benutzer an und zeigt diese beim Client an
- **Read:** fordert eine bestimmte Nachricht mit Empfänger, Betreff und Inhalt vom Server an und zeigt diese beim Client an
- **Del:** Löscht eine bestimmte Nachricht aus dem Mailspool-Verzeichnis beim Server
- **Quit:** Beendet die Kommunikation zwischen Client und Server, schließt den Socket und beendet das Client-Programm

3. Ablauf einer Operation

Der grundlegende Ablauf einer dieser Operationen ist wie folgt:

1. Der Client gibt einen der obigen Befehle in der Konsole ein
2. Der Befehl wird an den Server gesendet
3. Der Server empfängt diesen Befehl und sucht die dazugehörige Operation
4. Der Server fordert alle benötigten zusätzlichen Informationen für den jeweiligen Befehl vom Client an
    a. Der Server sendet pro benötigter Information eine Aufforderung an den Client, diese einzugeben
    b. Der Client gibt die geforderte Information ein und sendet diese zurück an den Server
    c. Der Server empfängt die Information vom Client, überprüft sie auf Korrektheit und speichert sie lokal zur späteren Verwendung ab
5. Sobald alle benötigten Informationen für den jeweiligen Befehl erhalten, überprüft und abgespeichert wurden, führt der Server die jeweilige Operation durch
6. Der Server sendet eine Status-Nachricht an den Client, um diesen darüber zu informieren, ob die Operation erfolgreich war oder fehlgeschlagen ist
7. Der Client empfängt die Status-Nachricht vom Server und zeigt diese an

4. Socket Implementierung

Das Erstellen eines neuen Sockets am **Server** wurde folgendermaßen umgesetzt:

```
int create_socket = socket (AF_INET, SOCK_STREAM, 0);
```

Erstellt einen neuen Socket mit IPv4 Adressen und TCP Protokoll. Dabei wird ein Integer-Wert zurückgegeben, der als Socket-Descriptor dient.

```
memset(&address, 0, sizeof(address));
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons (port);
if (bind (create_socket, (struct sockaddr *) &address, sizeof (address)) != 0) {
    perror("bind error");
    return EXIT_FAILURE;
}
```

Hier wird zunächst ein struct mit Informationen befüllt (IPv4 Adressen, alle erlauben und Port für Server-Socket festlegen).
Anschließend wird bind aufgerufen, um den vorher erstellten Socket nun auch an einen Port zu binden, um ihn später verwenden zu können.

```
listen (create_socket, 5);
```

Hier wird nun definiert, dass der Socket maximal 5 gleichzeitige Requests einreihen soll.

```
new_socket = accept ( create_socket, (struct sockaddr *) &cliaddress, &addrlen );
```

Anschließend kann über einen accept-call solange gewartet werden, bis ein Client versucht sich zum Server zu verbinden. In diesem Fall wird ein neuer Socket-Descriptor für die weitere Kommunikation mit diesem Client zurückgegeben.

```
close (create_socket);
```

Am Ende des Programms wird mit einem Aufruf von close der Server-Socket geschlossen.

Am **Client** wird ebenfalls mit einem Aufruf zu socket ein neuer Socket erstellt. Anschließend wird allerdings nicht bind aufgerufen, sondern connect (mit Server-IP und Server-Port als Parameter), um sich mit einem Remote-Server zu verbinden.

```
connect (create_socket, (struct sockaddr *) &address, sizeof (address))
```

Sobald connect aufgerufen wurde, kann nun über den Socket mit dem Server kommuniziert werden.