Sockets
VSYS

Arber Bajraktari
Mariel Hajdari

FH University of Applied Sciences
TECHNIKUM WIEN

Aufgabe: Teil 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:
    a. SEND: Senden einer Nachricht vom Client zum Server.
    b. LIST: Auflisten der Nachrichten eines Users. Es soll die Anzahl der Nachrichten und pro Nachricht die Betreff Zeile angezeigt werden.
    c. READ: Anzeigen einer bestimmten Nachricht für einen User.
    d. DEL: Löschen einer Nachricht eines Users.
    e. QUIT: Logout des Clients

Aufgabe: Teil 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.
- Aus Sicherheitsgründen beendet der Server jedoch die Socket Verbindung zum Client nach 3 fehlerhaften LOGIN Versuchen 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.

Lösung

1.

Wenn es weniger als 2 Parameter sind, so nur die Server Name geschrieben ist, wird die IP-Adresse selbst geschrieben (die Localhost). Wenn die IP-Adresse geschrieben ist, dann wird sie anstatt der Localhost Adresse genommen.

```
if (argc < 2)
    {
         inet_aton("127.0.0.1", &address.sin_addr);
    }
    else if( argc == 2)
    {
         inet_aton(argv[1], &address.sin_addr);
    }else if( argc == 3){
         inet_aton(argv[1], &address.sin_addr);
         char *end;
         long prt = strtol( argv[2], &end, 10);
         address.sin_port = htons((uint16_t)prt);
    }
```

2.

Die Server erwartet 3 Parameters: Name, Mailsverzeichniss und Port. Die Mailsverzeichniss muss inbox sein, sonst geht es nicht. Der Port kann irgendeine Zahl sein. Aber der Client muss der Port wissen, sonst kann er den Client Programm nicht starten.

```
if( argc != 3){
     printf("Falsche Parametereingabe.\nBitte schreiben Sie die Mailsverzsichniss und port\
n");
         return EXIT_FAILURE;
    }else{
     if( strcmp(argv[1], "inbox") != 0){
         printf("Falsche Verzeichnisseingabe.\n");
         return EXIT_FAILURE;
     }
     char *end;
     long prt = strtol( argv[2], &end, 10);
     address.sin_port = htons((uint16_t)prt);

    }
```

3.

Die Verbindung mit dem Server ist festgelegt. Es wird die Adresse, die von dem Client geschrieben ist, genommen. Wenn es eine Fehler in dem Verbindung passiert, wird es gezeigt.
Ob es richtig verbunden ist, wird gezeigt was die Server zu dem Client geschickt hat.

```
// CREATE A CONNECTION
    if (connect(create_socket,
                (struct sockaddr *)&address,
                sizeof(address)) == 0)
    {
       printf("Connection with server (%s) established\n",
              inet_ntoa(address.sin_addr));


       ////////////////////////////////////////////////////////////////////////
       // RECEIVE DATA
       size = recv(create_socket, buffer, BUF - 1, 0);
       if (size > 0)
       {
          buffer[size] = '\0';
          printf("%s", buffer);
       }
    }
    else
    {
```

```
24.       perror("Connect error - no server available");
25.       return EXIT_FAILURE;
26.    }
```

4.

a)

Es wird gelesen in Client, was der Client mit dem Tastatur geschrieben hat. Dann, wann er „send“ geschrieben hat, geht das Programm im Send Modus. Diese Modus wird auch am Server geschickt, wo auch der Server im Send Modus geht.

```
2)  if (fgets(buffer, BUF, stdin) != NULL){
3)      ////////////////////////////////////////////////////////////////////
4)      // SEND DATA
5)      send(create_socket, buffer, strlen(buffer), 0);
6)          if( strcmp( buffer, "send\n") == 0){
7)              ...
8)          }else if( strcmp( buffer, "list\n") == 0{
9)              ...
10)         }else ...
11)
```

Am anfang werden die Sender, Reciever, Subject und Message gelesen und im Strings gespeichert. Danach, diese Strings werden in eine String zusammengefasst. Diese String heißt sich „to_send“. Die Send Kommand zu dem server wird nur mit eine Message geschickt und nicht alle allein. Das macht es schneller. Das Protokoll um Daten zu schicken ist das:

“sender;receiver;subject;msg”

Dann wird diese Message vom Server gelesen.

```
1.  if( strcmp( buffer, "send\n") == 0){
2.              check_username("Sender", sender);
3.              clean_stdin();
4.              check_username("Reciever", reciever);
5.              clean_stdin();
6.              check_subject( subject);
7.              strtok( subject, "\n");
8.              check_msg( msg);
9.
10.             strcat( to_send, sender);
11.             strcat( to_send, ";");
12.             strcat( to_send, reciever);
13.             strcat( to_send, ";");
14.             strcat( to_send, subject);
15.             strcat( to_send, ";");
16.             strcat( to_send, msg);
17.             send(create_socket, to_send, strlen(to_send), 0);
18.             memset(buffer, 0, sizeof(buffer));
19.             size = recv(create_socket, buffer, BUF - 1, 0);
20.             if (size > 0)
21.             {
22.                 printf("%s", buffer);
23.             }else{
24.                 printf("ERR\n");
25.             }
26.
```

```
            // list ------------------------------------------
        }
```

Es wird die Modus gelesen. Dann wird die Message von dem Client zerteilt und im Strings gespeichert. Dann wird es verwendet um das Message in einem File zu speichern. Die Speicherungsformat ist das:

New Email;
Sender:if19b022;
Subject:Subject1;
Msg:Hello;

Wenn es alles funktioniert, wird “OK” zurückgeschickt, sonst „ERR“.

```
size = recv(new_socket, buffer, BUF - 1, 0);
          if (size > 0)
          {
         // remove ugly debug message, because of the sent newline
         if(buffer[size-1] == '\n') {
            --size;
         }

         buffer[size] = '\0';
         printf("User selected %s option.\n", buffer);
         //if send option is selected
         if( strcmp( buffer, "send") == 0){
            size = recv(new_socket, buffer, BUF - 1, 0);
            printf("Recieved message from user!\n");
            sender = strtok( buffer, ";");
            reciever = strtok( NULL, ";");
            subject = strtok( NULL, ";");
            msg = strtok( NULL, ".");

            //lock the file, other threads are not allowed to open it
            pthread_mutex_lock( &lock);
            snprintf(filename, sizeof(filename), "inbox/%s.txt", reciever);
            fp = fopen(filename,"a");
            pthread_mutex_unlock(&lock);
            if( fp == NULL){
                send(new_socket, "ERR\n", 4, 0);
            }else{
                fputs("New Email", fp);
                fputs(";\nSender:", fp);
                fputs( sender, fp);
                fputs(";\nSubject:", fp);
                fputs( subject, fp);
                fputs(";\nMsg:", fp);
                fputs( msg, fp);
                fputs(";\n", fp);
                fflush(fp);
                fclose(fp);
                send(new_socket, "OK\n", 3, 0);
            }

         }
```

b)

Die gleiche passiert auch bei List. Username nur wird gelesen, geschickt und die Antwort kommt zurück. Wenn 0 zurückkommt, endet die Modus hier, aber wenn es mehr kommt, werden auch die Subjects gezeigt.

```
}else if( strcmp( buffer, "list\n") == 0){
            memset( buffer, 0, sizeof( buffer));
            check_username("Username", reciever);
            clean_stdin();
            send(create_socket, reciever, strlen(reciever), 0);
            size = recv(create_socket, buffer, BUF - 1, 0);
            if (size > 0)
            {
                if( strcmp( buffer, "0") == 0){
                    printf("%s has %s Emails.\n", reciever, buffer);
                }else{
                    printf("%s has %s Emails.\n", reciever, buffer);
                    size = recv(create_socket, buffer, BUF - 1, 0);
                    // print subjects
                    printf( "%s", buffer);
                }

            }else{
                printf("ERR\n");
            }

        }
```

Im Server: wird die File geöffnet, Subjects anzahl genommen, und wann es gibt, die Subjects zum Client geschickt.

```
if(fp == NULL){
                send(new_socket, "0", 1, 0);
            }else{
                cnt = 0;
                //file is being read, other threads should not change it
                pthread_mutex_lock( &lock);
                while( fgets( buffer, BUF - 1, fp)){
                    if( strcmp( buffer, "New Email;\n") == 0){
                        cnt++;
                    }else if( strstr( buffer, "Subject:") != NULL){
                        sender = strtok( buffer, ":");
                        subject = strtok( NULL, ";");
                        strcat( to_send, subject);

                        strcat( to_send, "\n");

                    }
                }
                pthread_mutex_unlock( &lock);
                if( cnt == 0){
                    send(new_socket, "0", 1, 0);
                }else{
                    snprintf( count, 5, "%d", cnt);
                    send( new_socket, count, BUF - 1, 0);
                    send( new_socket, to_send, BUF - 1, 0);
                    printf("Emails are shown!\n");
                }

            }
```

c) Read ist die gleiche wie Send, aber nur im Server gibt es eine kleines Unterschied. Und das ist beim lesen des Nachricht. Im Client werden die Username und SubjectID genommen und zum Server geschickt. Dann Server schickt zurück die Massage.

```
while( fgets( buffer, BUF - 1, fp)){
                            if( strstr( buffer, "Msg:") != NULL || msg_cnt){
                                if( in_email == e_nr){
                                    //clear what is not needed
                                    if( strstr( buffer, ":") != NULL){
                                        sender = strtok( buffer, ":");
                                        strncat( to_send, strtok( NULL, ";"), sizeof(strtok( NULL, ";")));
                                        msg_cnt = true;
                                    }else{
                                        if( strstr( buffer, ";") != NULL){
                                            strncat( to_send, strtok( buffer, ";"), sizeof(strtok( buffer, ";")));
                                            msg_cnt = false;
                                            break;
                                        }else{
                                            strncat( to_send, buffer, strlen(buffer));
                                            msg_cnt = true;
                                        }
                                    }

                                    in_email--;
                                }
                                in_email++;
                            }
                        }
```

Die File wird mit fgets gelesen. Weil für uns die Speicherungs format klar ist, war es einfach die Nachricht allein zu lesen. Was am schwierigsten war, war nicht nur eine Zeile zu lesen, sonst alle. Dafür müssten wir alle möglichen Speicherungsmöglichkeiten beachten.

d)

Im Client ist nur die Username und SubjectID gelesen, sowie bei Read. Die Arbeit ist im Server gemacht.

Um eine Nachricht zu löschen haben wir am Anfang alle andere Nachrichten von einem User zu eine TempFile geschpeichert, die UserFile mit alle Nachrichten gelöscht, und danach den TempFile zu unsere ClientFile umgewandelt.

```
if( e_nr > cnt){
                        send(new_socket, "ERR\n", 4, 0);
                    }else{
                        in_email = 0;
                        fseek( fp, 0, SEEK_SET);
                        //file being read
                        pthread_mutex_lock( &lock);
                        while( fgets( buffer, BUF - 1, fp)){
                            if( strstr( buffer, "New Email;") != NULL){
                                in_email++;
                            }
                            if( in_email != e_nr){
                                fputs( buffer, fp_temp);
```

```
                            printf("%s\n", buffer);
                        }
                    }
                    fflush(fp_temp);
                    fclose(fp_temp);
                    remove( filename);
                    //deleting an message from a user
                    rename( "inbox/del.txt", filename);
                    pthread_mutex_unlock( &lock);
                    send(new_socket, "OK\n", BUF - 1, 0);
                }
```

Threads:

Die Threads sind nur im Server verwenden. Weil wenn wir eine Verbindung von dem Server mit einem Client machen, die Kommunikation nicht mehr mit Main zu tun hat, haben wir „detached Threads verwenden“. Mit detached Threads, wird nicht bis join gewartet um die andere Thread starten zu können.

```
pthread_t t1;
pthread_create(&t1, NULL, handle_client, (void*) &new_socket);
pthread_detach( t1);
```

Weil das drinnen in der While-Schleife steht, werden mehr als eine Threads erstellt.
Was noch wichtig war, war die mutex für die Files zu schreiben. Die Threads haben keine „shared-Ressources“ gehabt, deshalb war keine mutex für die Variablen gebraucht. Aber was alle Threads verwenden hat, sind die Files wo die Emails gespeichert sind.

```
pthread_mutex_lock( &lock);
            snprintf(filename, sizeof(filename), "inbox/%s.txt", reciever);
            fp = fopen(filename,"a");
            if( fp == NULL){
                send(new_socket, "ERR\n", 4, 0);
            }else{
                fputs("New Email", fp);
                fputs(";\nSender:", fp);
                fputs( sender, fp);
                fputs(";\nSubject:", fp);
                fputs( subject, fp);
                fputs(";\nMsg:", fp);
                fputs( msg, fp);
                fputs(";\n", fp);
                fflush(fp);
                fclose(fp);
                send(new_socket, "OK\n", 3, 0);
            }
pthread_mutex_unlock(&lock);
```