Seminar Concurrent C Programming

Einleitung

Autor: Ramon Gamma

ZHAW Zürcher Hochschule für Angewandte Wissenschaften
Dozent: Nico Schottelius
22.06.2014

Inhaltsverzeichnis

Seminar Concurrent C Programming

Einleitung

Diese Seminararbeit wird durch das Wahlpflichtmodul Betriebssysteme vorausgesetzt und hat das Ziel das angeeignete Wissen anzuwenden und durch diese Praxisarbeit weiter zu vertiefen. Die Arbeit ist in der Programmiersprache C zu verfassen und die Vorgaben sind durch den Dozenten auf github[1] festgelegt.
Diese Arbeit umfasst einen Multi-User-Editor TCP Server der mehreren Clients gleichzeitig Zugriff auf 1 Dokument ermöglicht.

Funktionsweise und Bemerkungen

Der Server arbeitet hauptsächlich mit einem Char Array, in welchem die Zeilen gespeichert sind, und mit einem Limit von 255 Zeilen arbeitet. Bei einem neuen Client wird versucht die Textdatei zu öffnen und einzulesen, dieser Inhalt bleibt bestehen bis alle Threads zu ende sind. Falls keine weiteren Threads vorhanden sind, führt der letzte Thread die Funktion aus, welche die Datei erstellt oder aktualisiert.

Die Befehle und Zeilen die der Client sendet, werden als ein String erwartet wie im Protokoll[2] definiert, also an einem Stück abgetrennt durch "\n".

Anleitung zur Nutzung

Das Programm wurde für Linux entwickelt und die Lauffähigkeit auf anderen Plattformen kann nicht garantiert werden. Die Befehle sind in einer Linux Shell auszuführen.

Programm herunterladen

```
$ git clone git@github.com:ramon-ga/ccp.git
$ cd ccp
```

[1] https://github.com/telmich/zhaw_seminar_concurrent_c_programming/
[2] https://github.com/telmich/zhaw_seminar_concurrent_c_programming/blob/master/protokoll/multi-user-editor

Programm compilieren

```
$ make run
$ make test
```

Server starten

Der Server akzeptiert nach dem starten Clients auf dem TCP Port 3000.
Es wird in unregelmässigen Abständen die Textdatei "textfile.txt" aktualisiert.

```
$ ./run
```

Test Client starten

Der Test Client sendet nach dem Start Befehle an den Server und zeigt die Antworten des Servers an.

```
$ ./test
```

Weg, Probleme, Lösungen

Weg

Die Erledigung der anstehenden Arbeiten und Umsetzungen wurde Schritt für Schritt erledigt:

1. TCP Server der für jeden Client einen Thread erstellt
2. TCP Client der auf den Server zugreifen kann und Daten senden kann
3. Implementierung im Server um auf gemeinsame Ressourcen zugreifen zu können
4. Text Kommandos erkennen und in separaten Funktionen ausführen
    a. INSERTLINES, REPLACELINES, READLINES, DELETELINES, NUMLINES
5. Pro zu bearbeitende Linie im Textdokument darf zur gleichen Zeit nur einem Client den Zugriff gewährt werden (Lock auf Zeilenebene)
6. Text der temporär gespeicherten Daten in eine Textdatei speichern / lesen
7. Testen mit mehreren Kommandos gleichzeitig

Probleme und Lösungen

Dadurch das Programmieren mit C nicht zu den alltäglichen Aufgaben gehört, gab es diverse Probleme, Schwierigkeiten und Unverständlichkeiten. Einige davon werden nachstehend aufgezeigt.

Gesendet Befehle nach "Space" trennen

Um ein Kommando von den Clients auf ihre Gültigkeit zu überprüfen muss der Befehl aufgetrennt werden.

```
InsertLines 2 2\n
HalloText\n
AndererText\n
```

Mit dem Befehl strtok() ist dies ein leichte Aufgabe, wenn dies nicht mehrfach zum Beispiel in einer Schlaufe gebraucht wird.
Nach diversen und gescheiterten Versuchen hat das Internet[3] die Lösung offenbart, wie man mit dem Befehl strtok_r() dieses Problem umgehen kann.

```
char *line = strtok_r(CommandBuffer, "\n", &saveptr1);
char *command_split = strtok_r(command_line, " ", &saveptr2);
```

Zeilennummern aus dem Kommando

Die Zeilennummern aus der Kommandozeile müssen für die Weiterverwendung in eine Zahl (int) umgewandelt werden. Die erste Annahme die einfach mit einem Cast umzuwandeln ergab falsche Nummern und Warnungen beim kompilieren.

Versuche:

```
command_start_num = (int) command_split;
command_start_num = (int) command_split - '0';
```

Der natürlich richtige Ansatz ist mit atoi()[4] zu Arbeiten:

```
command_start_num = atoi(command_split);
```

String (Char) Rückgabe von Funktion

Um die Übersichtlichkeit zu steigern und der Wiederverwendbarkeit halber, wurden die Funktionen in mehreren Dateien untergebracht. Die Rückgabe von einem Char und danach diesen weiter zu verwenden funktioniert anscheinend nur per Zufall bei den ersten Tests.

[3] http://linux.die.net/man/3/strtok_r
[4] http://stackoverflow.com/questions/628761/character-to-integer-in-c

Alle Vorschläge aus dem Internet schienen in keinem Fall verlässlich zu funktionieren, was schliesslich zu der Entscheidung führte die Funktionen in eine Datei zu verschieben um den Problemen mit den Resourcen und Rückgabewerten zu umgehen.

Zeilen Lock

Eine Resource zu schützen vor dem Zugriff weiterer Threads schien sehr einfach mit:

```
pthread_mutex_unlock(&line_locking_mutex);
pthread_mutex_lock(&line_locking_mutex);
```

Doch dadurch das nur eine eine Zeile und nicht der Inhalt gelockt werden soll, musste eine Version hin, mit der auf der Basis der Zeilen einen Lock durchgeführt werden kann.

```
pthread_mutex_lock(&line_locking_mutex[command_start_num])
```

Diese Version scheint zu funktionieren, obwohl das nicht eindeutig getestet werden könnte.

Umgang mit C-Array's

Der Umgang mit C-Array's hat immer wieder für Kopfschmerzen gesorgt, und schlussendlich mit den Beschreibungen[5] wurden die Problem aus dem Weg geräumt.

```
fileContents[lineNumber] = malloc(strlen(line) + addLineBreak);
strcpy(fileContents[lineNumber], line);
```

Warnungen beim Kompilieren

Die Fehler und Warnungen beim Kompilieren könnten bis auf wenige immer gelöst werden. Die untenstehenden Warnungen konnten leider nicht behoben werden.

> warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
> socket2 = (int) socket_pointer; ^ include/process.c: In function
>
> comparison between pointer and integer [enabled by default]
> if (strcasestr(commands_allowed2, command) != NULL) {
>
> comparison between pointer and integer [enabled by default]
> if (strcasestr(commands_allowed3, command) != NULL) { // single command

[5] http://www.thegeekstuff.com/2011/12/c-arrays/

Fazit

Die Erfahrung in der Programmiersprache C ist beschränkt und es kann eher von ersten Schritten die Rede sein. Von anderen Programmiersprachen ist man es gewohnt, das zum Beispiel mit String's umgegangen wird, was einem hier vor neue Herausforderungen stellt. Den Grossteil der Zeit in dieser Arbeit wurde dadurch auch in die Problembehebung gesteckt, die nur nach einem grossem Aufwand gelöst werden konnten.

Einiges ist nach der Arbeit um einiges Klarer als vorher, trotzdem kann nur von einer ersten Erfahrung gesprochen werden, und es bräuchte noch einige Zeit um das volle Verständnis zu erhalten.

Quellenverzeichnis

Bild Titelseite, http://www.gedan.net/wp-content/uploads/2009/03/GOV-carousel-code.jpg, 22.06.2014