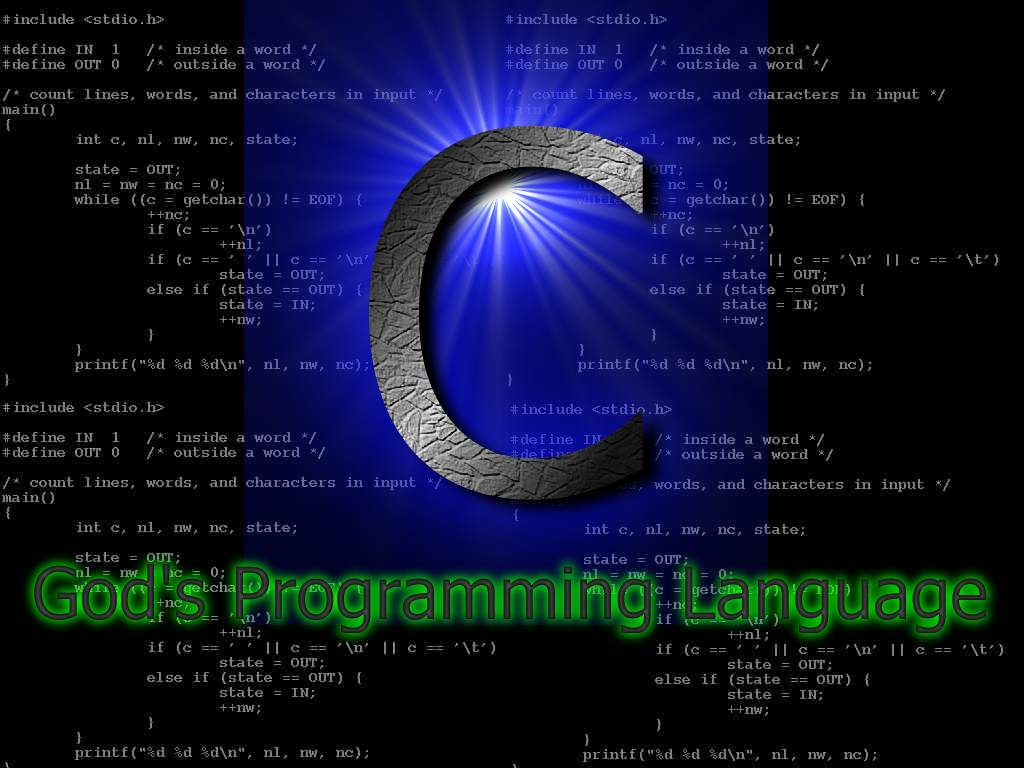
# Seminararbeit 2014

# Concurrent Programming in C

****

|  |  |
| --- | --- |
| Verfasser | Severin Müller |
| Dozent | Nico Schottelius |
| Modul | Betriebssysteme |
| Seminar | “Concurrent Programming in C” |
| Erscheinungsjahr | 2014 |

# 1. Themenwahl und Aufgabenstellung

## 1.1 Eigenmotivation

Als ich 2002 mit der Programmierung anfangen wollte, machte ich mir Gedanken darüber, welche Programmiersprache ich als erstes erlernen möchte. Da mein Vater für Studer Revox die Software für die elektronischen Geräte in C schrieb, entschied ich mich für eben diese Sprache, damit ich auch mal nach einem Tipp fragen konnte.

2008 begann ich ich mit einem Projekt, das meine Kenntnisse in C vertiefen sollte. Dieses sah einen Zusatzserver zu einem IRC Server vor. Dazu musste ich mir Kenntnisse in der Programmierung von Sockets und Datenbanken aneigenen.

In dieser Arbeit habe ich mich für das Thema „Fileserver“ entschieden, weil ich so die Konzepte, die ich in meinem früheren Projekt erarbeitet habe noch vertiefter erlernen konnte.

## 1.2 Abgrenzung der Aufgabenstellung

In der Aufgabenstellung wurde ein Fileserver verlangt, der Concurrency fähig ist. Das heisst, dass mehrere Clients gleichzeitig darauf zugreifen können müssen, ohne dass diese sich in die Quere kommen.

Folgende Randbedingungen galt es einzuhalten:

* keine Global Locks
* Das physische Dateisystem darf nicht benutzt werden
* Pthreads oder fork und shm
* Kommunikation über TCP/IP oder Unix Domain Sockets
* Indeces beginnen bei 0
* Client muss nie retry machen
* Das Protokoll muss eingehalten werden

Ich habe mich für eine Pthreads Lösung mit TCP/IP entschieden und möchte die Realisierung auf den folgenden Seiten darlegen.

## 1.3 Anleitung zur Nutzung

### 1.3.1 Aufsetzen

Um die Applikation zu nutzen muss das Programm mit make erstellt werden. Zunächst muss das Repository gezogen werden:

user@linux:~$ git clone https://github.com/fish-guts/concurrent.git

Danach können wir den Testclient erstellen. Dazu müssen wir uns im Rootverzeichnis des Projektes befinden.

user@linux:~/concurrent$ make test

Zu guter Letzt erstellen wir den Server:

user@linux:~/concurrent$ make run

### 1.3.2 Betrieb

Sobald wir die beiden binaries erstellt haben können wir den Server starten.

user@linux:~/concurrent$ ./run

###############################

Welcome to Severin's FileServer

###############################

Starting server...Bind Successful

Server started successfully, listening on port 8083

Wenn wir diesen Screen sehen, läuft der Server und wartet auf Verbindungen.

Wir können nun einen Client starten der auf den Server verbindet. Dazu öffnen wir ein zweites Terminal Fenster und geben folgendes ein.

user@linux:~/concurrent$ ./test

Konnte der Client erfolgreich verbinden ist folgendes zu sehen:

Connected to server

Nun können wir im Client Befehle an den Server schicken. Gültige Befehle sind:

* create <Filename> <Filesize>\n <Content>
* update <Filename> <Filesize>\n <Content>
* delete <Filename>
* read <filename>
* list
* exit

# 2. Realisierung

## 2.1 Lösungsansatz

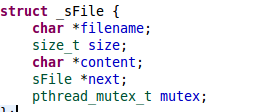
Wie bereits in Kapitel 1.2 erwähnt habe ich mich in dieser Arbeit für eine Pthreads Lösung entschieden.

Der Vorteil, den ich hier sehe ist die Tatsache, dass Threads untereinander die gleichen Resourcen teilen1, so also auf Shared Memory verzichtet werden kann.

Um einen Lock zu erzeugen werden Mutexe eingesetzt. Ein Mutex ist im Wesentlichen ein Lock den wir setzen, bevor auf eine Resource zugreifen, die auch von anderen verwendet werden darf2. Wenn mehrere Threads gleichzeitig auf eine Resource zugreifen möchten so setzt der erste Thread den Mutex und die anderen Threads werden in einen wartenden Zustanden versetzt bis der Lock wieder gelöst wird3.

Da eines der Kriterien war, dass keien Global Locks verwendet werdne dürfen, musste ich mir Gedanken darüber machen, wie ich den Lock auf Dateiebene setze.

Um dieses Problem zu lösen habe ich mich für eine globale Liste mit den Dateien entschieden die in Form einer verketteten Liste vorliegt.



Der Mutex wird dabei auf dem Listenknoten gesetzt.

Literaturverzeichnis

[1]: Stevens, W. Richard: Advanced Programming in the UNIX Environment. Hallbergmoos, Deutschland: Pearson Deutschland, 3. Auflage, 2013, Seite 383, ISBN 978-0-321-63773-4

[2]: Stevens, W. Richard: Advanced Programming in the UNIX Environment. Hallbergmoos, Deutschland: Pearson Deutschland, 3. Auflage, 2013, Seite 399, ISBN 978-0-321-63773-4

[3]: Stevens, W. Richard: Advanced Programming in the UNIX Environment. Hallbergmoos, Deutschland: Pearson Deutschland, 3. Auflage, 2013, Seite 400, ISBN 978-0-321-63773-4