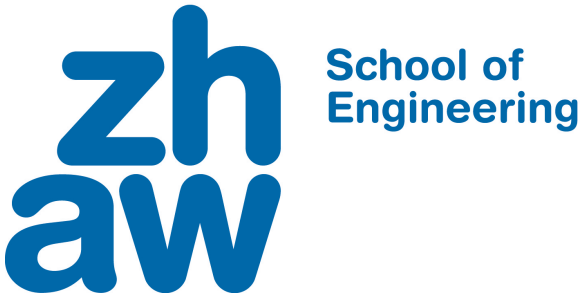
Zürcher Hochschule
für Angewandte Wissenschaften



Concurrent Programming in C

TCP-Fileserver

Seminararbeit FS 2014

Student: Micha Schönenberger

Dozent: Nico Schottelius

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Versionierung

Version	Datum	Beschreibung
V0.1	18.03.2014	Ersterstellung Dokument
V0.2	11.05.2014	Einarbeitung in Lucene, Installation Software, Lesen LiA
V0.3	13.05.2014	Buch LiA lesen / Coden
V0.4	14.05.2014	Doku schreiben, Query Optionen studieren
V0.5	15.05.2014	Doku LATEX, Query Optionen studieren
V0.6	17.05.2014	eingesetzte Hardware, Kapitel **??**
V0.7	18.05.2014	Kapitel **??**, Code Refactoring
V0.8	20.05.2014	Kapitel **??**, PDF und Office Indexing Probleme
V0.9	29.05.2014	Kapitel Dokumentation, Probleme Indexgrösse Kapitel **??**
V1.0	30.05.2014	Kapitel **??**, Dokumentation überarbeiten

Tabelle 1.1.: Versionierung Dokumentation

2. Aufwände

Datum	Zeit	Beschreibung
18.03.2014	1.25h	Ersterstellung Dokument -Download und Installation Lucene -Erste Versuche mit Lucene
26.03.2014	3.75h	Suche nach Thema für Eingabe EBS -Online Suche Funktionalitäten Lucene
11.05.2014	4.25h	Installation von Netbeans -Update Eclipse -Lucene in Action: Einlesen Kapitel 1 -1. Versuch Indexer und Searcher (gemäss Beispielen LiA)
12.05.2014	1.75h	siehe 11.05.2014
13.05.2014	7h	-Indexierung erweitern auf alle Subdirectories des Root-Folders -Einschränken von File-Extensions: momentan nur Textdateien (.txt, .h, .c, .java)
13.05.2014	2.75h	-Dokumentation Kapitel **??** bis **??** -Quellenverzeichnis erstellen -Kapitel **??** **??** -Kapitel **??** **??** -Einarbeiten in Query mit Lucene/Luke
15.05.2014	4.25h	-Dokumentation von Word in LATEX umschreiben -Kapitel **??** **??**
17.05.2014	2.25h	-Kapitel **??** mit teils Unterkapiteln -Kapitel **??** **??**
18.05.2014	5h	-Kapitel **??** mit teils Unterkapiteln -Code Refactoring and cleaning
28.05.2014	5h	-Code Refactoring and cleaning -Probleme mit Index-Grösse
29.05.2014	8.5h	-Probleme mit Index-Grösse -Kapitel **??**
30.05.2014	8.75h	-Kapitel **??** -Kapitel **??** fertig stellen -Probleme mit content Indizierung von PDF beheben

Tabelle 2.1.: Aufwände Seminararbeit

3. Einleitung

3.1. Rahmenbedingungen

Die Aufgabenstellung und die Rahmenbedingungen wurden über Github (https://github.com/telmich/zhaw_seminar_concurrent_c_programming) veröffentlicht.
Anbei ein Auszug aus den wichtigsten Eckdaten und Anforderungen:

3.1.1. Geplante Termine

Datum	Beschreibung
13.03.2014	Kick-Off Meeting
16.03.2016	Abgabe der schriftlichen Arbeit (1 Woche vor Präsentation)
22.06.2014	Präsentation der Arbeit
01.07.2014	Präsentation der Arbeit
02.07.2014	optionale Teilnahme an anderen Präsentationen
03.07.2014	optionale Teilnahme an anderen Präsentationen
21.07.2014	Notenabgabe

Tabelle 3.1.: geplante Termine

3.1.2. Administratives

- Abgabe Arbeit via git repository auf github.com
- Zum Zeitpunkt Äbgabe Arbeit"werden alle git repositories geklont, Änderungen danach werden *NICHT* für die Benotung beachtet.

3.1.3. Abgabebedingungen

- git repo auf github vorhanden
- Applikation lauffähig unter Linux
- Nach “make“ Eingabe existiert
  - “run“: Binary des Servers
    - Sollte nicht abstürzen / SEGV auftreten

  - "test": Executable zum Testen des Servers

- "doc.pdf": Dokumentation
- Einleitung
- Anleitung zur Nutzung
- Weg, Probleme, Lösungen
- Fazit
- Keine Prosa - sondern guter technischer Bericht
- Deutsch oder English möglich

3.1.4. Vortrag / Präsentation

- 10 - 15 Minuten + 5 Minuten Fragen
- Richtzeiten:
- Einleitung (2-3) min
- Weg, Probleme, Lösungen (4-10) min
- Implementation zeigen (2-5) min
- Fragen (2-5) min
- Vortrag ist nicht (nur) für den Dozenten

3.1.5. Lernziele

- Die Besucher des Seminars verstehen was Concurrency bedeutet und welche Probleme und Lösungssansätze es gibt.
- Sie sind in der Lage Programme in der Programmiersprache C zu schreiben, die auf gemeinsame Ressourcen gleichzeitig zugreifen.
- Das Seminar setzt Kenntnisse der Programmiersprache C voraus.

3.1.6. Lerninhalte

- Selbstständige Definition des Funktionsumfangs des Programmes unter Berücksichtigung der verfügbaren Ressourcen im Seminar.
- Konzeption und Entwicklung eines Programms, das gleichzeitig auf einen Speicherbereich zugreift.
- Die Implementation erfolgt mithilfe von Threads oder Forks und Shared Memory (SHM).

3.2. Das Projekt

- kein globaler Lock (!)
- Kommunikation via TCP/IP (empfohlen) - Wahlweise auch Unix Domain Socket
- fork + shm (empfohlen)
  - oder pthreads
  - für jede Verbindung einen prozess/thread
  - Hauptthread/prozess kann bind/listen/accept machen
- Fokus liegt auf dem Serverteil
  - Client ist hauptsächlich zum Testen da
  - Server wird durch Skript vom Dozent getestet
- Wenn die Eingabe valid ist, bekommt der Client ein OK
  - Locking, gleichzeitiger Zugriff im Server lösen
  - Client muss *nie* retry machen
- Protokolldefinitionen in protokoll/
- Alle Indeces beginnen bei 0
- Debug-Ausgaben von Client/Server auf stderr

Fileserver

- Dateien sind nur im Speicher vorhanden
- Das echte Dateisystem darf NICHT benutzt werden
- Mehrere gleichzeitige Clients
- Lock auf Dateiebene

3.3. Ausgangslage

Die Aufgabenstellung, wie sie oben beschrieben ist, ist für einen nicht Programmierer gemäss Dozent eine grosse Herausforderung. Mindestens vier Studenten, zu denen auch ich zähle, haben ihre Bedenken geäussert, dass diese Aufgabenstellung fast nicht zu erreichen ist. Ein Informatiker, dessen Zuhause ist das Programmieren ist geschweige denn die Sprache "C", wird für eine minimalistische Lösung bei weitem mehr Stunden benötigen als die 60 Stunden, welche für dieses Seminararbeit gedacht sind.
Damit für den Dozenten besser ersichtlich ist, wie viel Zeit aufgewendet wurde und für welche Teile der Arbeit, werden im Kapitel 2 (Aufwände) die Zeiten erfasst und ausgewiesen.

4. Anleitung zur Nutzung des Servers und des Clients

FEHLT NOCH

5. Umsetzung des Projektes

5.1. Voraussetzungen

Da der Student kein Programmierer ist und nur schulische Kenntnisse von der Programmiersprache Java besitzt, wird dieses Projekt eine grosse Herausforderung. Deshalb soll das Grundkonzept als Stütze dienen, so dass sich der Programmierer nicht in den Details verlieren soll.

5.2. Libraries

Im Unterricht des Modules «Systemsoftware» wurden verschiedene Libraries durch den Dozenten zur Verfügung gestellt.
Diese sollen, da sie einige Grundfunktionen wir das Error-Handling bereits beinhalten, in diesem Projekt ebenfalls genutzt werden. Die so genutzten Dateien werden nicht explizit als Quelle erwähnt. Sie besitzen jedoch im Kopf die Daten des Dozenten und sind als externe Datei erkennbar. Als Beispiel zeigt die Abbildung **??** die Anbindung einer externen Datei.

```
1 /* (C) IT Sky Consulting GmbH 2014\
2  * http://www.it-sky-consulting.com/\
3  * Author: Karl Brodowsky\
4  * Date: 2014-02-27\
5  * License: GPL v2 (See https://de.wikipedia.org/wiki/GNU_General_Public_License
       )\
6  *\
7  * This file is inspired by\
8  * http://cs.baylor.edu/~donahoo/practical/CSockets/code/HandleTCPClient.c\
9  */
```

Listing 5.1: itskylib.c

5.3. Programmierumgebung

Programmiert wird auf einem MAC OS-X 10.9 (Mavericks). Die eingesetzte Software ist das Eclipse mit dem integrierten «Eclipse C/C++ Development Tools»Ėclipse ist bereits aus der Java-Programmierung im Grundstudium bekannt und eingerichtet. So musste nur noch die «Eclipse C/C++ Development Tools» installiert werden. Der grosse Vorteil gegenüber eines Texteditors ist das Auto-Complete und die automatische Formatierung des Codes.
Für das Kompilieren und Ausführen des Codes wird eine Ubuntu genutzt. Dieses ist als virtuelle Maschine über Parallels installiert. Zugegriffen auf das Ubuntu wird mittels SSH von MAC OS-X. Der Grund, Ubuntu zu nutzen liegt in den anderen Bibliotheken, welche teils in MAC OS-X nicht genutzt werden können oder anders implementiert sind. Ebenfalls aufgefallen im Unterricht war, dass Ubuntu 32-bit und Ubuntu 64-bit nicht immer gleich implementiert sind.

Eckdaten Ubuntu:

- OS: ubuntu 12.04 LTS
- Memory: 900 MB
- CPU: Intel Core i7-2677M CPU @ 1.80 GHz
- OS-Type: 64bit

5.4. LOG/DEBUG

Die Implementierung des LOG soll als Erstes geschehen. So soll sichergestellt werden, dass während der Programmierung das LOG-Level geändert werden kann und allfällige Fehler schneller gesehen werden können.
Die Definition der LOG-Levels wird anlog zu den syslog LOG-Level erstellt:

LEVEL	**Bezeichnung**
0	EMERGENCY
1	ALERT
2	CRITICAL
3	ERROR
4	WARNING
5	NOTICE
6	INFORMATIONAL
7	DEBUG

Tabelle 5.1.: Loglevels

5.5. Speicherverwaltung – Buddy System

Für die Verwaltung des Shared Memory (shm) bedarf es einer Logik, um die verschiedenen Adressen im Shared Memory richtig ansprechen zu können. Zusätzlich muss sichergestellt werden, dass kein File in das shm geschrieben wird, dass länger ist als der freie Speicherplatz, bevor das nächste File kommt.
Es gibt viele Dokumentierte Speicherverwaltungen. Nach längerer Recherche wurde entschieden, dass der Speicher mit dem Buddy-System verwaltet werden soll. Die Suche im Internet nach einer vorhandenen Library für die Speicherverwaltung mit dem Buddy-System blieb leider erfolglos. Also blieb nichts anderes übrig, als das Buddy-System von grund auf selber zu gestalten und zu implementieren.
Dabei wurden sehr viele Fragen aufgeworfen, welche Schrittweise erarbeitet wurden

5.5.1. Wie wird das komplette System gemanagt?

Für das Management des shared Memory wurde ein Struct erstellt (siehe Listing 5.2, welches das Shared Memory kontrollieren soll.

```
1  struct shm_ctr_struct {
2    int shm_size; //size of shm-block
3    int  isfree ; // indicates if block is free or not
4    int isLast; //indicates the end of shared memory
5    struct shm_ctr_struct *next;
6    struct shm_ctr_struct *prev;
7    char *filename;
8    char * filedata ; // just this pointer is a pointer to Shared memory
9    pthread_rwlock_t rwlockFile; //Read-write lock for file
10 };
```

Listing 5.2: shm_ctr_struct

Folgende Tabelle soll aufzeigen, welches Attribut im Struct welche Funktion übernehmen soll. Das Ziel des Structs ist eine verkette Liste. Das erste Struct ist im Main global bekannt, das letzte wird gefunden, da isLast auf TRUE gesetzt ist.

Struct Attribut	**Bezeichnung**
int shm_size	Grösse des Blockes des Shared Memory Bereiches
int isLast	TRUE wenn es der letzte Block ist, sonst FALSE
int isfree	TRUE wenn Block frei ist, FALSE wenn Block besetzt ist
struct shm_ctr_struct *next	Pointer auf den nächsten Block (zeigt auf sich selber, wenn es der letzte Block ist)
char *filename	Pointer auf den Dateinamen, der im Block gespeichert ist (NULL wenn kein File gespeichert ist)
char *filedata	Dies ist der einzige Pointer auf das Shared-Memory. Hier liegen die effektiven Daten des Files.
pthread_rwlock_t rwlockFile	Für jede Instanz des Structs und somit für jedes unique File wird ein ReadWrite-Lock erstellt.

Tabelle 5.2.: Shared Memory control Struct

5.5.2. Wie wird der optimale Block für ein neues File im Shared Memory gefunden?

Hierzu wurde die Funktion find_shm_place(...) erstellt.
Diese Funktion beginnt beim ersten Eintrag des Structs shm_ctr_struct (siehe Listing 5.2) und sucht über alle vorhanden Blöcke (über den next-Pointer) einen optimalen Block. Optimal bedeutet, dass er grösser oder gleich der Grösse der neu zu erstellenden Dokumentes sein muss, aber nicht grösser als das doppelte . Wäre er grösser als das Doppelte, wäre das Speicherplatzverschwendnung. Zusätzlich muss er frei sein (isfree = TRUE).

5.5.3. Gibt es nur zu grosse Blöcke, wie werden die geteilt?

Das Buddy-System gibt vor, dass die Blockgrössen aus 2er Potenzen gebildet werden. Also 2, 4, 8, 16, 32 ...
Beispiel Buddy-System:

Shared Memory – SIZE = 65535

Tabelle 5.3.: Shared Memory - 1

Ist die Dateigrösse = 14547, gibt es keinen optimalen Block. Der optimale Block wäre hier $2^{14}(= 16384)$. Der kleiner Block $2^{13}(= 8192)$ wäre hier zu klein. Zuerst muss müssen nun die Blöcke aufgeteilt werden, so dass folgende Blöcke entstehen:

Block 1 SIZE = 16384	Block 2 SIZE = 16384	Block 3 SIZE = 32768	

Tabelle 5.4.: Shared Memory - 2

Für die Aufteilung wurde die Funktion devide(...) implementiert.
Diese beginnt beim ersten Block und arbeitet sich (über den next-Pointer) nach hinten. Beim ersten gefundenen freien Block, wird nun die Block-Size halbiert. Es wird ein neuer Block erzeugt und die Verlinkungen (next, previous, Pointer auf Filename und Filedata sowie isFree und size) werden dem bestehenden und neuen Block gesetzt, so dass die Linked-List wieder komplett vorhanden ist.
Ist die Blockgrösse die gewünschte Grösse, findet ein return = TRUE statt. Ansonsten wird die Funktion selber rekursiv aufgerufen, bis die Blockgrösse genügend klein ist. Dann erfolgt der return = TRUE. Ein Screenshot der Funktion devide() ist im Anhang A.2 zu finden. Ebenfalls im Anhang A.1 und A.3 ist das Shared Memory vor und nach dem Einfügen eines neuen Files zu sehen.

5.6. Locks

Wie im Kapitel 3.2 erwähnt, ist ein global Lock nicht erlaubt.
Für die Umsetzung des Locks wurde schlussendlich kein mutex gewählt, wie anfangs gedacht war. Das Problem beim mutex ist, dass ein lesender Client das ganze File ebenfalls sperrt für weitere Lesezugriffe. Dies soll jedoch nicht der Fall sein.
Aus diesem Grund wurde wurde auf «pthread_rwlock_t» zurückgegriffen.

Die Impelementation des Locks wurde gemäss Tabelle im Kapitel 5.5.1 vorgenommen.
Da das Kontroll-Strukt für das Shared-Memory bereits vorhanden war, konnte «pthread_rwlock_t» ohne Probleme eingefügt werden.
Soll nun ein ein File gelockt werden, kann das elegant gelöst werden:

- Wenn das File gelesen werden möchte, muss zwingend die Adresse des enstprechenden Strukt bereits vorhanden sein = struct shm_ctr_struct *shm_ctr
- Nun kann ein ReadLock über pthread_rwlock_rdlock(&(shm_ctr->rwlockFile)); gemacht werden.

5.7. Files und deren Funktionen

Um die auf den ersten Blick nicht ganz klare Strukturen aufzeigen zu können, soll sich dieses Kapitel mit den einzelnen Files geschäftigen, die für den Server und den Client notwendig sind. Jede Funktion jedes Files soll kurz und bündig erläutert werden.

myfunctions.c beinhaltet eigene definierte Funktionen
getFixCharLen(char *mychar, int mylength) füllt einen CharPointer bis zur gewählten Länge auf. Wird benötigt für schöne Darstellung im LOG
void print_all_shm_blocks(struct shm_ctr_struct *shm_ctr) Gibt auf der Konsole alle Blöcke des SHM aus. Wird zu DEBUG-Zwecken benötigt
char * get_all_shm_blocks(struct shm_ctr_struct *shm_ctr) Gibt alle Blöcke des SHM als char Pointer zurück. Wird benötigt, um Client das SHM zu übermitteln
void print_single_shm_blocks(struct shm_ctr_struct *shm_ctr) Gibt auf der Konsole einen Block des SHM aus. Wird zu DEBUG-Zwecken benötigt
char * getSingleString(char *msg, . . .) Gibt einen Char Pointer als Return Wert. Diese Funktion erlaubt es, einen ?String? mit Argumenten (z.B. %i, %s) zu übergeben. Diese werden zur Laufzeit interpretiert und als neuen Char Pointer zurückgegeben

Tabelle 5.5.: bar

Tabelle 5.6.: myfunctions.c

6. Fazit

FEHLT NOCH

A. Anhang

A.1. Screenshots zum Server

A.1.1. Shared Memory vor dem Einfügen eines neuen Files

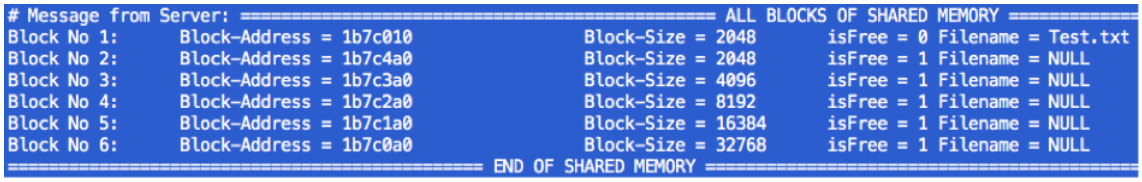


Abbildung A.1.: SHM vor dem Einfügen eines neuen Files
Quelle: eigener Screenshot

A.1.2. Aufteilen des Shared Memory in der Funktion devide()

```
Want to write filename with size=1056 to shm
Adress of shm Place to check is 215f010
Size of shm Place is 65536
At the end of all shared memory places... No hit found to enter the filename.
Checked a good address is:  0
0 is not valid. So there is no good place to write the file into... Trying no to
Now in round_up_int(). Filesize needed = 1056    until = 16
i = 11   Output now set to = 2048 and return it.
Now in devide(). I want to devide until I reach Block size of 2048
================================================================================
Address: d386b000        Block-Size = 32768      Filename = NULL
================================================================================
Recursive call in deviding because block size is to big (at moment = 32768) ...
Now in devide(). I want to devide until I reach Block size of 2048
================================================================================
Address: d386b000        Block-Size = 16384      Filename = NULL
================================================================================
Recursive call in deviding because block size is to big (at moment = 16384) ...
Now in devide(). I want to devide until I reach Block size of 2048
================================================================================
Address: d386b000        Block-Size = 8192       Filename = NULL
================================================================================
Recursive call in deviding because block size is to big (at moment = 8192) ...
Now in devide(). I want to devide until I reach Block size of 2048
================================================================================
Address: d386b000        Block-Size = 4096       Filename = NULL
================================================================================
Recursive call in deviding because block size is to big (at moment = 4096) ...
Now in devide(). I want to devide until I reach Block size of 2048
After deviding I have a good block size.

Will now output all shm-blocks...
================================================================================
Block No 1:      Block-Size = 2048       Filename = NULL         isLast = 0
Block No 2:      Block-Size = 2048       Filename = NULL         isLast = 0
Block No 3:      Block-Size = 4096       Filename = NULL         isLast = 0
Block No 4:      Block-Size = 8192       Filename = NULL         isLast = 0
Block No 5:      Block-Size = 16384      Filename = NULL         isLast = 0
Block No 6:      Block-Size = 32768      Filename = NULL         isLast = 1
================================================================================
```

Abbildung A.2.: Aufteilen des SHM in der Funktion devide()
Quelle: eigener Screenshot

A.1.3. Shared Memory nach dem Einfügen eines neuen Files

```
# Message from Server: ================================================ ALL BLOCKS OF SHARED MEMORY ==============
Block No 1:      Block-Address = 1b7c010              Block-Size = 2048       isFree = 0 Filename = Test.txt
Block No 2:      Block-Address = 1b7c4a0              Block-Size = 2048       isFree = 1 Filename = NULL
Block No 3:      Block-Address = 1b7c3a0              Block-Size = 32         isFree = 0 Filename = test2.txt
Block No 4:      Block-Address = bc003210             Block-Size = 32         isFree = 1 Filename = NULL
Block No 5:      Block-Address = bc003110             Block-Size = 64         isFree = 1 Filename = NULL
Block No 6:      Block-Address = bc003010             Block-Size = 128        isFree = 1 Filename = NULL
Block No 7:      Block-Address = bc002f10             Block-Size = 256        isFree = 1 Filename = NULL
Block No 8:      Block-Address = bc002e10             Block-Size = 512        isFree = 1 Filename = NULL
Block No 9:      Block-Address = bc002d10             Block-Size = 1024       isFree = 1 Filename = NULL
Block No 10:     Block-Address = bc002c10             Block-Size = 2048       isFree = 1 Filename = NULL
Block No 11:     Block-Address = 1b7c2a0              Block-Size = 8192       isFree = 1 Filename = NULL
Block No 12:     Block-Address = 1b7c1a0              Block-Size = 16384      isFree = 1 Filename = NULL
Block No 13:     Block-Address = 1b7c0a0              Block-Size = 32768      isFree = 1 Filename = NULL
```

Abbildung A.3.: SHM nach dem Einfügen eines neuen Files
Quelle: eigener Screenshot

Literaturverzeichnis

[1] *Apache Lucene - Index File Formats.* http://lucene.apache.org/core/3_5_0/fileformats.html, may 2014.

[2] *Lucene Query Parser Syntax.* http://lucene.apache.org/core/2_9_4/queryparsersyntax.html, may 2014.

[3] *Who is using Lucene/Solr.* http://searchhub.org/2012/01/21/who-uses-lucenesolr/, may 2014.

[4] McCandless, Michael; Hatcher, Erik; Gospondenetic Otis: *Lucene in Action - Second Edition.* Manning Publications Co, 2010.