

Concurrent Programming in C

**Dozent**: Nico Schottelius  
xsou@zhaw.ch

**Student:** Micha Schönenberger  
schoenm1@students.zhaw.ch

**Inhaltsverzeichnis**

1 Versionierung 3

2 Aufwände 3

3 Einleitung 5

3.1 Rahmenbedingungen 5

3.1.1 Termine 5

3.1.2 Administratives 5

3.1.3 Abgabebedingungen 5

3.1.4 Vortrag / Präsentation 5

3.1.5 Lernziele 5

3.1.6 Lerninhalte 6

3.2 Das Projekt 6

3.3 Ausgangslage 6

4 eigentliche Dokumentation **Fehler! Textmarke nicht definiert.**

5 Literaturverzeichnis 11

# Versionierung

|  |  |  |
| --- | --- | --- |
| **Version** | **Datum** | **Bemerkung** |
| V0.1 | 15.03.2014 | Ersterstellung |
| V0.2 | 17.03.2014 | Einleitung, Ausgangslage |
| V0.3 | 07.04.2014 | Grundgerüst, Konzept |
| V0.4 | 08.04.2014 | Implementierung Argument-Überprüfung (LogLevel) |
| V0.5 | 13.04.2014 | Erweitern Server (Argument-Überprüfung) |
| V0.6 | 01.05.2014 | Speicherverwaltung mit Buddy |
| V0.7 | 02.05.2014 | Client Connection to Server |
| V0.8 | 03.05.2014 | TCP Connection Protocol, Loglevel |

# Aufwände

|  |  |  |
| --- | --- | --- |
| **Datum** | **Zeitaufwand** | **Bezeichnung** |
| 15.03.2014 | lesen = 2.5h  Arbeit = 1.25h | Ersterstellung Dokumentation  Github Repo erstellen  Einlesen Buch Kapitel 15 (Semaphore, Shared Memory…)  Erstellen Debian VM  SSH Key-gen for Github (Mac and Debian) |
| 17.03.2014 | 0.5h | Dokumentation: Einleitung (Rahmenbedingungen, Projekt, Ausgangslage) |
| 07.04.2014 | 1.75 h | Grundgerüst erstellen, LOG-LEVEL definieren |
| 08.04.2014 | 3 h | Parsing Argumente bei Programmstart Log-Level Implementierung |
| 13.04.2014 | 3.5 h | * Auslagern Funktionen in externe .h Dateien * Anpassen Argument-Validierung: wenn Argument mehr als 1mal vorkommt, wird es ignoriert * bei nicht setzen des LogLevel wird default LogLevel initialisiert * Erstinitialisierung TCP-Server: wartet auf Verbindung von Client * Probleme: #define von LOG LEVELS in ‚Log-Level.h‘ sind nicht sichtbar in „server.h“ . ->Mail an Nico Schottelius |
| 15.04.2014 | 1h | * Installing eUML -> not working, just with .java |
| 01.05.2014 | 9 h | * Degub mit #define funktioniert nicht. Einlesen in andere Möglichkeiten * gemäss Rücksprache mit anderen Studenten sollte nicht ein File wirklich eingelesen werden (von HDD geöffnet und Stream übermittelt), sondern lediglich mit dem Filenamen und Grösse angelegt werden im Shared Memory * Versuch, Control Shared Memory zu lösen mit einem Buddy System. Nach etlichen Stunden Aufgabe   Fazit Arbeiten Heute:   * Server startet ohne Fehler * Loglevel gelöscht (da nicht funktionstüchtig) * Port kann mit Argument „-p“ mitgegeben werden * bei starten des Servers ohne Argumente kommt die Hilfeseite * Das Kontroll-Strukt für das Shared Memory ist implementiert. * Die Speicherverwaltung mit Buddy-System wurde beschlossen. Das aufteilen der Blöcke funktioniert einwandfrei (wieder vereinen ist noch nicht implementiert) |
| 02.05.2014 | 2.25h | * Client TCP Connection zu Server aufbauen * Client kann Verbindung aufbauen, Message senden und Message erhalten. Es fehlt jedoch ein Protokoll, dass die Übertragung sicherstellt. * Teils werden noch zusätzlich Zeichen angezeigt (z.B. 25$?d anstelle von 25) * es gibt noch keine Validierung der Argumente (z.B. CREATE, DELETE…) |
| 03.05.2014 | 9h | * Log-Level implementiert mit verschiedenen Stufen. Output momentan nur möglich auf CLI, jedoch mit Datum (z.B. May 3 2014 15:37:15: WARNING Test Log Warning) * Implementierung von kleinem TCP Protokoll (funktioniert erst beim Senden Client): Die Länge der zu sendenden Zeichen werden gezählt und diese Zahl mitgesendet. Der Server splittet den Stream in Arrays und überprüft, ob die empfangenen Daten gleich viele Zeichen aufweisen wie die zu erwartende. |
|  |  |  |

# Einleitung

## Rahmenbedingungen

Die Aufgabenstellung und die Rahmenbedingungen wurden über Github (<https://github.com/spitzbueb/conc_programming>) veröffentlicht.

Anbei ein Auszug aus den wichtisten Eckdaten und Anforderungen:

### Termine

|  |  |
| --- | --- |
| **Datum** | **Bezeichnung** |
| 13.03.2014 | Kick-off Meeting |
| 16.03.2016 | Angabe des Git-Repository |
| 22.06.2014 | Abgabe 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 |

### Administratives

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

### 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

### Vortrag / Präsentation

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

### Lernziele

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

### 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).

## 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

## 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.

# Umsetzung des Projektes

## Ideen, Aufbau, Grundkonzept

### 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.

### 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 besitzten jedoch im Kopf die Daten des Dozenten und sind als exterene Datei erkennbar.

Beispiel:

/\* (C) IT Sky Consulting GmbH 2014

\* http://www.it-sky-consulting.com/

\* Author: Karl Brodowsky

\* Date: 2014-02-27

\* License: GPL v2 (See https://de.wikipedia.org/wiki/GNU\_General\_Public\_License )

\*

\* This file is inspired by

\* http://cs.baylor.edu/~donahoo/practical/CSockets/code/HandleTCPClient.c

\*/

### 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“. Eclipse 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 Untericht 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

### LOG/DEBUG

Die Impelementierung 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 |

### Speicherverwaltung

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.

#### Buddy-Speicherverwaltung

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

1. **Wie wird das komplette System gemanaged?**

Hierzu wurde ein Struct erstellt, welches das Shared Memory kontrollieren soll.

struct shm\_ctr\_struct {

int shm\_size; //size of shm-block

int isfree; // indicates if block is free or not

int isLast; //indicates the end of shared memory

struct shm\_ctr\_struct \*next;

struct shm\_ctr\_struct \*prev;

char \*filename;

char \*filedata; // just this pointer is a pointer to Shared memory

};

|  |  |
| --- | --- |
| **Struct Attribut** | **Bezeichnung** |
| int shm\_size | Grösse des Blockes des Shared Memory Bereiches |
| int isfree | TRUE wenn Block frei ist, FALSE wenn Block besetzt ist |
| int isLast | TRUE wenn es der letzte Block ist, sonst FALSE |
| struct shm\_ctr\_struct \*next | Pointer auf den nächsten Block (zeigt auf sich selber, wenn es der letzte Block ist) |
| struct shm\_ctr\_struct \*next | Pointer auf den vorhergehenden Block (zeigt auf sich selber, wenn es der erste Block ist) |
| char \*filename | Pointer auf den Filenamen, der im Block gespeichert ist (NULL wenn kein File gespeichert ist) |
| char \*filedate | Dies ist der einzige Pointer auf das Shared-Memory. Hier liegen die effektiven Daten des Files. |

1. **Wie finde ich für das File den optimalen Block im Shared Memory?**

Hierzu wurde die Funktion **find\_shm\_place(…)** erstellt.

Diese Funktion beginnt beim ersten Eintrag des Structes **shm\_ctr\_struct** und sucht über alle vorhanden Blöcke (über den next-Pointer) einen optimalen Block.

Optimal bedeutet, dass er grösser oder gleich der Grösse des Files sein muss, aber nicht grösser als das doppelte (da Speicherplatzverschwendung)

Zusätzlich muss er frei sein (isfree = TRUE).

1. **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 = 65536 |

Ist die Filegrösse = 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 |

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, welcher frei ist, 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.

Screehshot für Aufteilung der Blöcke (2014-05-01)



# Literaturverzeichnis