

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 4

3.1 Rahmenbedingungen 4

3.1.1 Termine 4

3.1.2 Administratives 4

3.1.3 Abgabebedingungen 4

3.1.4 Vortrag / Präsentation 4

3.1.5 Lernziele 4

3.1.6 Lerninhalte 5

3.2 Das Projekt 5

3.3 Ausgangslage 5

4 eigentliche Dokumentation 6

5 Literaturverzeichnis 7

# Versionierung

|  |  |  |
| --- | --- | --- |
| **Version** | **Datum** | **Bemerkung** |
| V0.1 | 15.03.2014 | Ersterstellung |
| V0.2 | 17.03.2014 | Einleitung, Ausgangslage |
| V0.3 | 07.04.2014 | Grundgerüst, Konzept |
|  |  |  |
|  |  |  |

# 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 | ??? h |  |

# 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

### DEBUG

Die Impelementierung des DEBUG soll als erstes geschehen. So soll sichergestellt werden, dass während der Programmierung das DEBUG-Level geändert werden kann und allfällige Fehler schneller gesehen werden können.

Die Definition der DEBUG-Levels wird anlog zu den syslog DEBUG-Level erstellt:

|  |  |
| --- | --- |
| **LEVEL** | **Bezeichnung** |
| 0 | Emergency |
| 1 | Alert |
| 2 | Critical |
| 3 | Error |
| 4 | Warning |
| 5 | Notice |
| 6 | Informational |
| 7 | Debug |

# Literaturverzeichnis