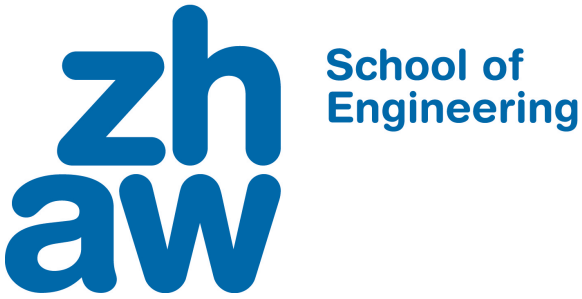
Zürcher Hochschule
für Angewandte Wissenschaften



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	15.03.2014	Ersterstellung Dokument
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	02.05.2014	TCP Connection Protocol, Loglevel
V0.9	02.05.2014	CREATE, LIST, Fehlebehebungen
V1.0	02.05.2014	One Thread per Client, Refactoring, Commenting Code
V1.1	02.05.2014	Client Connection to Server
V1.2	02.05.2014	Implementing RWLock for Creating File
V1.3	03.05.2014	Loglevel und eigenes TCP-Protokoll
V1.3b	04.05.2014	CREATE und Log Verbesserungen
V1.3c	05.05.2014	LIST shm / Fehlersuche CREATE
V1.4	06.05.2014	dynamisches TRACE_LOG
V1.5	26.05.2014	Erstellen PThread für Clients
V1.5b	27.05.2014	ReadWriteLock beim READ
V1.5c	28.05.2014	Dokumentation
V1.6	31.05.2014	Fehlerbehandung PThreadList. Client Commands
V1.6b	01.06.2014	Joining PThreads. Fehlersuche DELETE letzes File
V1.7	02.06.2014	Fehlerbehebung Übermittlung grössere Files
V1.8	10.06.2014	Optimierung Serverparameter (-l, -p)
V1.8b	11.06.2014	Fehlerbehandlung Code, testen Server
V1.8b	12.06.2014	Fehlerbehandlung Code, testen Server
V1.8b	14.06.2014	Fehlerbehandlung Code, testen Server
V1.8c	15.06.2014	Output LIST, Fehlerbehandlung Code
V1.8c	17.06.2014	__TIME__, Dokumentation überarbeiten
V1.8d	17.06.2014	Fehlerbehebungen, Dokumentation überarbeiten

Tabelle 1.1.: Versionierung Dokumentation

2. Aufwände

Datum	Zeit	Beschreibung
15.03.2014	3.75h	-Ersterstellung Dokumentation -Github Repo erstellen -Einlesen Buch Kapitel 15 (Semaphore, Shared Memory, . . .) -Erstellen Debian VM
17.03.2014	0.5h	-Dokumentation: Einleitung (Rahmenbedingungen, Projekt, Ausgangslage)
07.04.2014	1.75h	-Grundgerüst erstellen, LOG-LEVEL definieren
08.04.2014	3h	-Parsing Argumente bei Programmstart -Log-Level Implementierung
13.04.2014	1.75h	-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 Verbindungvon Client -Probleme: #define von LOG LEVELS in log-Level.h sind nicht sichtbar in “server.h“.
15.04.2014	1h	-Installieren von e-UML -> funktioniert nur mit Java ;-(
01.05.2014	9h	-Degub mit #define funktioniert nicht. -> Einlesen in andere Möglichkeiten für Log-Levels -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 **Fazit Arbeiten**: -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	6h	-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 nur beim Senden von Client zu Server)
04.05.2014	6h	-Überprüfung 1. Wort von Client als Command-Argument (Momentan nur Create File) -Verfeinern CREATE Command - LOG verbessern
05.05.2014	8h	-beim CREATE vom 2. File wurde der Name des ersten Files überschrieben. Stundenlange Suche nach Ursache (Problem war ein zuweisen eines Pointer zum einem zweiten filename = filename.new anstelle filename = strdup(filename.new) -Implementierung von LIST shm, was dem Client eine komplette Liste des Shared Memory mit Adresse und Dateinamen zurückliefert.
06.05.2014	5.5h	-Dokumentation letzte 2 Tage -Beheben von Warnings beim Kompilieren -für das TRACE_LOG können nur mehrere (dynamische) Variablen mitgeliefert werden. -Problem, dass Server teils beim Erstellen eines Files abstürzt. Recherche im Internet: 1 Fehler war das malloc vor einen strdup() -> Weniger Abstürze, aber nicht ganz weg -DELETE und READ fertig implementieren (ohne Lock)
26.05.2014	1.75h	-Erstellen von Pthreads für Clients
27.05.2014	4.25h	-Erstellen von PThreads für Client -Implementierung ReadWrite Lock mit pthread_rwlock_t (Momentan nur ReadLock beim Lesen) -Kommentieren von Code

		-löschen von altem, nicht mehr benutztem Code -Anpassen Log-Design (damit besser lesbar)
28.05.2014	2h	-Update Dokumentation -Anpassen Version Github/Dokumentation -Dokumentation ergänzen
31.05.2014	4.5h	-Fehlerbehebung PThreadList -Senden von EXIT bei Beenden von Client an Server -Joining PThread nach Client-EXIT bei Server
01.06.2014	7h	-Joining PThread nach Client-EXIT bei Server fertig -Probleme Segmentation Fault beim löschen des letzten Files (mehrere Stunden Fehlersuche) -> Problem war Test.txt (fix in Code als Testfile) -Implementierung von RW-Lock bei DELETE File
02.06.2014	3.75h	-Fehlerbehebung bei Übermittlung von grösseren Fileinhalten -Code kommentieren -Dokumentation erweitern
10.06.2014	3.75h	-Dokumentation anpassen und erweitern -Server Parameter optimieren (-p, -l) -Dokumentation Kapitel 4
11.06.2014	1h	-Ferhlebehandlung Server: letztes Wort beim Erstellen eines Files wurde nie gespeichert -Diverse kleinere Fehlerbehandlungen im Code -Dokumentation ergänzen
12.06.2014	4h	-Ferhlebehandlung Server: bei CREATE ohne weitere Angaben stürzte der Server mit segmentation fault ab -Dokumentation ergänzen 5.7
14.06.2014	8.5h	-Ferhlebehandlung Server: bei DELTET/READ/UPDATE ohne weitere Angaben stürzte der Server mit segmentation fault ab -Dokumentation ergänzen 5.7
15.06.2014	3.25h	-Ferhlebehandlung Server: bei DELTET/READ/UPDATE ohne weitere Angaben stürzte der Server mit segmentation fault ab -Dokumentation ergänzen 5.7, 3.1, 6
17.06.2014	4.75h	-Dokumentation auf Inhaltsfehler durchlesen -Probleme bei LOG: immer gleiche Zeitangabe -> Problem: __TIME__ ist statisch -Dokumentation: Inhaltfehler beheben
22.06.2014	xxh	-Dokumentation 4.4 -Absturz Server beheben bei einer leeren Eingabe beim Client -testen der Applikation auf mögliche Fehler -...

Tabelle 2.1.: Aufwände Seminararbeit

3. Einleitung

3.1. Quellenangaben

Der Quellcode der gesammten Arbeit wurde vom Verfasser selber geschrieben. Natürlich können sich einzelne Fragmente im Internet gefunden werden, da dies in der Programmierung immer der Fall sein kann. Es wurde jedoch kein Quellcode vom Internet übernommen, sondern Ideen gesammelt und standardisierte Funktionen benutzt. Hier soll als Beispiel ein memcpy erwähnt werden.
Der einzige Ort von dem Code kopiert wurde ist das github Repository des Dozenten Karl Brodowsky, welcher das Modul «Systemsoftware» durchführte (siehe Quelle [1]). Diese kopierten Dateien sind jedoch im Header als solche erkennbar.

Für die Umsetzung der Arbeit wurden sehr viele kleinere Probleme wie z.B.: wie kopieren ich einen String? im Internet gesucht. Es ist schlicht nicht möglich, diese Quellen alle anzugeben.
Die verwendeten Bücher sind im Quellenverzeichnis am Schluss dieser Arbeit zu finden.

3.2. 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:

3.2.1. Geplante Termine

Datum	Beschreibung
13.03.2014	Kick-Off Meeting
24.06.2016	Abgabe der schriftlichen Arbeit (1 Woche vor Präsentation)
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

3.2.2. 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.

3.2.3. 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

3.2.4. 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

3.2.5. 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.

3.2.6. 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).

3.3. 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

3.4. 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 die Anforderungen der Aufgabenstellung fast nicht zu erreichen wären. 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 für welche Teile der Arbeit aufgewendet wurden, sind im Kapitel 2 Aufwände die Zeiten erfasst und ausgewiesen.

4. Anleitung zur Nutzung des Servers und des Clients

Dieses Kapitel widmet sich mit dem Umgang des Servers und der dazugehörigen Clients. Angefügte Screenshots sollen einen Einblick geben, wie die Software funktioniert, auch wenn kein Computer zum Austesten der Applikation vorhanden ist.

4.1. Starten des Server

Der Server ist das Herzstück der Applikation. Er ist so ausgelegt, dass er mit gültigen Argumenten erweitert werden kann.
Momentan gibt es zwei implementierte Argumente, welche beim Start mitgegeben werden können:

- «-l» – Loglevel
  Für das Loglevel gültige Eingaben sind Integer mit Werten von 0-7.
  Wird ein ungültiger Wert grösser als 7 eingegeben, wird der Fehler abgefangen und das Loglevel wird auf den default-Wert = 5 gesetzt.
- «-p» – Serverport
  Für den Server-Port gültige Eingaben sind: 1024 - 65535. Die well-known Ports von 1 - 1023 wurden bewusst nicht erlaubt, da es Konflikte geben könnte mit anderen Applikationen.
  Wird ein ungültiger Wert eingegeben, wird der Server automatisch mit dem default-Port = 7000 starten.

Beispiele für gültiges Starten des Servers:
./Server -p 4637 -l 7
./Server -l 6 -p 5479
./Server -p 7788
./Server -l 8

Die Abbildung 4.1 zeigt einen gültigen Serverstart. Bei der Abbildung 4.2 ist zu sehen, wie das Loglevel 8 nicht gesetzt werden kann und wie die nachträglichen Argumente (Duplikate) des Serverports und des Loglevels ignoriert werden.

```
parallels@ubuntu:Program> ./Server -p 8877 -l 7
Setting up shared Memory ...Verify valid arguments ...

-------------------------------------------------------
Argument No. 1  Value = -p      -> Hit for Server-Port

Set up now Server Port to 8877 ...    ... Done

-------------------------------------------------------
Argument No. 3  Value = -l      -> Hit for Loglevel

Try to set Loglevel to 7
... Done
Set up TCP-Server settings ...
# Tue Jun 17 10:53:26 2014:     LOG_ALERT               Server is now going to Listening Mode for Clients.
# Tue Jun 17 10:53:26 2014:     LOG_ALERT               Client can connect to Server on Port 8877
# Tue Jun 17 10:53:26 2014:     LOG_INFORMATIONAL       Waiting for Client to connect...
```

Abbildung 4.1.: Starten des Servers
Quelle: eigener Screenshot

```
parallels@ubuntu:Program> ./Server -p 8877 -l 8 -p 4433 -l 4
Setting up shared Memory ...Verify valid arguments ...

-------------------------------------------------------
Argument No. 1  Value = -p      -> Hit for Server-Port

Set up now Server Port to 8877 ...    ... Done

-------------------------------------------------------
Argument No. 3  Value = -l      -> Hit for Loglevel

LogLevel not valid [1-8]. Will set Log-Level to 5   ... Done

-------------------------------------------------------
Argument No. 5  Value = -p      -> Hit for Server-Port

ServerPort was already set. New argument will be ignored.

-------------------------------------------------------
Argument No. 7  Value = -l      -> Hit for Loglevel

LogLevel was already set. New argument will be ignored.
Set up TCP-Server settings ...
# Jun 10 2014 15:13:35: LOG_ALERT               Server is now going to Listening Mode for Clients.
# Jun 10 2014 15:13:35: LOG_ALERT               Client can connect to Server on Port 8877
```

Abbildung 4.2.: Starten des Servers mit doppelten Argumenten
Quelle: eigener Screenshot

Anmerkung:

Die Reihenfolge der Argumente ist egal. Wird ein Argument zwei mal eingegeben (z.B. ./Server -p 7524 -l 6 -l 8), wird nur das erste Argument berücktsichtigt. Alle weiteren Argumente werden ignoriert.

./Server -p 5432 -l 5 -l 8 wird somit den TCP Port 5432 und das Loglevel 5 setzen.

./Server -p 2066 -p 5432 -l 4 -l 8 wird somit den TCP Port 2066 und das Loglevel 4 setzen.

4.2. Starten der Clients

Der Client besitzt im Gegensatz zum Server eine sehr eingeschänkte Logik. Seine Aufgabe besteht hauptsächlich darin, sich über einen TCP-Socket mit dem Server zu verbinden.
Beim Client ist es notwendig, die Argumente in der richtigen Reihenfolge einzugeben. So ist nur folgender Aufruf gültig:
./Client <IP Adresse von Server>
oder
./Client <IP Adresse von Server> <TCP Port Server>
Wird die Variante ohne den Port gewählt, versucht sich der Client über den default Port = 7000 mit dem Server zu verbinden.

Abbildung 4.3.: Starten des Clients
Quelle: eigener Screenshot

Beispiele für gültiges Starten des Clients:
./Client 10.211.55.12
./Client 10.211.55.12 5542

4.3. Benutzen der Clients

Um die Funktionalitäten des Servers nutzen zu können, müssen die Befehle vom Client zum Server gesendet werden. Direkte Eingaben im Server sind nicht zulässig.
Die gültigen Server-Befehle werden hier nicht erläutert, sondern nur wie die Befehle genutzt werden können. Einen Einblick in die Implementierung der CRUD-Befehle gibt das Kapitel 5.7.

- CREATE
- READ
- UPDATE
- DELETE
- LIST shm Der LIST zeigt das momentan vorhandene Shared-Memory an, wobei zu beachten gilt, dass hier kein Locking-Verfahren eingesetzt wurde. Das heisst, das Memory könnte bereits bei der Ausgabe auf der Konsole beim Client bereits wieder verändert worden sein.

4.4. Einschränkungen der Applikation

Die Benutzung dieser Applikation beinhaltet einigen Einschränkungen. Einige basieren auf der Implementation und deren Definition, andere sind bekannte Fehler, welche jedoch aufgrund von zeitlichen Problemen konnten nicht alle Fehler behoben werden, die erkannt wurden.
Aus diesem Grund sind folgende Einschränkungen vorhanden, welche den Server auch abstürzen lassen können. Es könnte auch sein, dass noch mehr Fehler vorhanden sind, welche jedoch noch nicht erkannt wurden.

4.4.1. Einschränkungen gemäss Implementationsdefinitionen

- Werden bei einem CREATE oder UPDATE beim content zwischen zwei Wörtern zwei oder mehr Leerzeichen platziert, werden diese (da der content in einzelne Array aufgeteilt wird) auf ein Leerzeichen reduziert.
  Beispiel:
  Bei der Eingabe von «CREATE test.txt Hallo Welt, dies ist ein Test» und anschliessendem READ der Datei test.txt ist der content: «Hallo Welt, dies ist ein Test»
- Die maximale Wortlänge, sprich aneinander gereihte Zeichen ohne Leerzeichen, darf maximal 256 sein.
  Diese Einschränkung in der Funktion void breakCharArrayInWords(...) definiert, welche die Eingabe des Clients auftrennt in ein Array, um so Serverbefehle vom Rest (z.B. content) unterscheiden zu können.
- CREATE und UPDATE
  Wird bei dem Erstellen oder Updaten eines Files ein content mitgegeben, welcher nicht mindestens 3 Zeichen hat, wird automatisch «NULL...» als content geschrieben.

  Beispiel:
  Bei einem «CREATE test.txt 12» und anschliessendes «READ test.txt» wird ausgegeben: «# Message from Server: NULL...».

4.4.2. Einschränkungen gemäss bekannter Fehler in der Applikation

- Wird bei dem Befehlt CREATE oder UPDATE ein Content mitgegeben, welcher > 900 ist, wird der Server mit grosser Wahrscheinlichkeit mit einem bufferoverflow abstürzen

```
# Sun Jun 22 14:45:21 2014:     LOG_INFORMATIONAL      Waiting for reveicing message from Client.
*** buffer overflow detected ***: ./Server terminated
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(__fortify_fail+0x37)[0x7f3697502f47]
/lib/x86_64-linux-gnu/libc.so.6(+0x109e40)[0x7f3697501e40]
/lib/x86_64-linux-gnu/libc.so.6(+0x1092a9)[0x7f36975012a9]
/lib/x86_64-linux-gnu/libc.so.6(_IO_default_xsputn+0xdd)[0x7f369747413d]
/lib/x86_64-linux-gnu/libc.so.6(_IO_vfprintf+0x25c2)[0x7f3697442f82]
/lib/x86_64-linux-gnu/libc.so.6(__vsprintf_chk+0x94)[0x7f3697501344]
/lib/x86_64-linux-gnu/libc.so.6(__sprintf_chk+0x7d)[0x7f369750128d]
./Server[0x401da7]
./Server[0x4037b3]
/lib/x86_64-linux-gnu/libpthread.so.0(+0x7e9a)[0x7f36979c7e9a]
/lib/x86_64-linux-gnu/libc.so.6(clone+0x6d)[0x7f36974ec3fd]
======= Memory map: ========
00400000-00407000 r-xp 00000000 00:13 17679                              /media/psf/Home/Dropbox_Zh
```

Abbildung 4.4.: Buffer Overflow bei zu grossen Dateien
Quelle: eigener Screenshot

- Stürzt der Server ab, wird das allozierte Shared Memory nicht gelöscht. Bei einem erneuten Start des Servers wird ein weiteres Shared Memory erzeugt. Bis zu einem Neustart der Linux-Umgebung oder manuelles löschen der nicht mehr benützten Shared-Memory Blöcken bleiben diese vorhanden.

```
parallels@ubuntu:Program> ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x011348d1 393216     parallels  600        65536      0
0x011383db 425985     parallels  600        65536      1
```

Abbildung 4.5.: Duplizierung SHM nach Server Crash
Quelle: eigener Screenshot

5. Umsetzung des Projektes

5.1. Voraussetzungen

Da der Student kein Programmierer ist und nur schulische Kenntnisse von der Programmiersprache Java besitzt, war dieses Projekt eine grosse Herausforderung.

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 das Listing 5.1 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 wurde auf einer Mac OS-X 10.9 (Mavericks) Umgebung. Die eingesetzte Software ist das Eclipse mit dem integrierten «Eclipse C/C++ Development Tools». Eclipse war bereits aus der Java-Programmierung im Grundstudium bekannt und eingerichtet. So mussten lediglich 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 wurde ein Ubuntu 12.04 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 wurde als eine der ersten Aufgaben in Angriff genommen. So sollte 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.

Die nächsten Unterkapitel zeigen einige aufgeworfene Fragen, welche gelöst werden mussten.

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. Der Grundgedanke beim Erstellen dieses Struktes war, eine verkettete Liste zu erstellen, die man von Anfang bis zum Ende durchlaufen kann. So gibt es keine Einschränkung wir bei einem Array, bei welchem von Begin her die Anzahl Elemente bekannt sein müssen. 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)
struct shm_ctr_struct *prev	Pointer auf den vorherigen Block (implementiert, aber nicht benutzt)
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 dies Speicherplatzverschwendnung. Zusätzlich muss der Block frei sein (isfree = TRUE).

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

Für die Aufteilung der Blöcke 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 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.

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

Das Vorgehen beim Aufteilen der Blöcke ist folgendermassen:

1. Durch den Aufruf der Funktion wird die gewünschte Grösse des Blockes mitgegeben. Da diese Funktion nur aufgerufen wird, wenn der Block noch nicht existiert, muss nicht mehr geprüft werden, ob es bereits einen optimalen Block gibt.
   Dessen Grösse ist in 2er Potenzen (in unserem Beispiel unten 16384)

2. Beginne beim 1. SHM-Block. Ist dieser grösser als 16384 und frei? Wenn ja, gehe zu Punkt 3, sonst wiederhole diesen Punkt bis Bedingung erfüllt ist.

3. Wurde ein Block zum Teilen gefunden, setzte die neuen Parameter (hier nur Auszugsweise dargestellt).
   Der Quellcode ist im Listing A.1 im Anhang zu finden.

   - erstelle eine neue Instanz des Struktes shm_ctr_struct
   - Halbiere die Grösse des momentanen Blockes mit
     shm_ctr->shm_size = (shm_ctr->shm_size) / 2;
   - setzte next des aktuellen Struktes auf das neu Erstellte
   - setze den Vorgänger des neuen Struktes auf das akutelle Strukt mit
     nextsshm->prev=shm_ctr
   - ... (für die komplette Implementierung siehe A.1 im Anhang)

Das folgende Beispiel des Buddy-System soll dessen Funktion beim Auftrennen von Blöcken aufzeigen:

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). Zuerst muss müssen nun die Blöcke aufgeteilt werden, so dass folgende Blöcke entstehen:

Im ersten Schritt wird nun der einzige Block (=65535) geteilt.

Block 1 SIZE = 32768	Block 2 SIZE = 32768

Tabelle 5.4.: Shared Memory - 2

Nun wird überprüft, ob ein optimaler Block vorhanden ist. Da dies hier nicht der Fall ist, werden die Blöcke weiter geteilt. Begonnen wird immer an Anfang des SHM. Das heisst, im nächsten Schritt wird der Block 1 (=32768) in zwei Blöcke mit je 16384 aufgeteilt, wie in der Tabelle 5.5 zu sehen ist.

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

Tabelle 5.5.: Shared Memory - 3

Nach der zweiten Aufteilung ist ein optimaler Block vorhanden. Die Funktion devide(...) gibt nun den Pointer auf den ersten optimalen Block zurück (es nach einer Aufteilung immer zwei Blöcke: hier Block 1 und Block 2).

5.6. Locks

Wie im Kapitel 3.3 erwähnt, ist ein global Lock nicht erlaubt.
Für die Umsetzung des Locks wurde schlussendlich kein mutex gewählt wie anfangs angedacht 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 Implementation 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.

Eine kurze Übersicht über die Impelementation des Locks gibt das Kapitel 5.7, welches die einzelnen CRUD Befehle aufführt.

5.7. Server-Befehle

Die gültigen Server-Befehle sind nach CRUD aufgebaut. CRUD ist ein Akronym in der Programmiersprache und soll die grundlegenden Befehle **C**reate, **R**ead, **U**pdate und **D**elete umfassen. Zusätzlich zu den vier Grundbefehlen kommt ein weiterer hinzu: «LIST». Dieser zeigt dem Client das momentan vorhandene Shared Memory an.

Grundätzlich werden alle Befehle in der Funktion runClientCommand(...) abgefangen. Auf den effektiven Programmcode wird hier nicht detailliert eingegangen.

5.8. CREATE

Das vereinfachte Vorgehen beim CREATE ist folgendermassen:

1. Überprüfe, ob Filename und Filecontent mitgegeben wurde. Fehlt das eine oder andere, sende dem Client direkt zurück, dass File nicht erstellt werden kann.
2. Wurde Filename und Fileconent mitgegeben, überprüfe ob das File bereits existiert. Wenn ja: return «File already exist». Wenn nein, Schritt 3
3. Suche einen guten Platz im Shared Memory. Aufgrund der Implementierung eines Buddy-Systems muss der SHM-Block grösser sein als das einzufügende File, aber kleiner als das doppelte. Diese Überprüfung wird von der Funktion «find_shm_place(...)» übernommen.
4. Wenn kein Platz gefunden: return -1, wenn Platz vorhanden: weiter mit Schritt 5
5. setze Flag für den gefunden SHM-Block = not free
6. initialisiere den RW Lock im Strukt shm_ctr_struct
7. setze den Write-Lock
8. schreibe Filename und Fileinhalt
9. gib den Write Lock wieder frei
10. return: «File xyz sucessfully created»

Beispiel für gültige Eingaben von CREATE:
–CREATE test.java Dies ist der Inhalt vom test.java file

Beispiele für ungültige Eingaben von CREATE:
–CREATE

Wenn kein Filename mitgegeben wird, sendet der Server folgende Nachricht an den Client: «Can not create a new file. Filename is missing.»

–CREATE filename.txt

Wenn ein Filename, aber kein content mitgegeben wurde, sendet der Server folgende Nachricht an den Client: «No file content set. File not created»

Der Programmcode für das Handling von CREATE ist im Listing A.2 in Anhang A.2.2 zu finden. Die Einschränkung zu CREATE ist im Kapitel 4.4.1 Einschränkungen gemäss Implementationsdefinitionen zu finden.

5.9. READ

Das Vorgehen bei einem READ Befehl ist folgendermassen:

1. Überprüfe ob ein Filename mitgegeben ist. Wenn nicht, sende an Client, dass nicht nach einem leeren Filenamen gesucht werden kann. Ist der Filename vorhanden, gehe zum nächsten Schritt.
2. Überprüfe im ersten SHM-Block, ob der Filename mit dem gesuchten Name übereinstimmt.
3. Gibt es einen Treffer:
   - setze den Read-Lock
   - kopieren den Fileinhalt in einen temporären Char Pointer
   - gib den Read-Lock wieder frei
   - Gib als return Wert den content des Files
4. Wenn der Filename nicht übereinstimmt, gehe zum nächsten Block im SHM.
5. Wiederhole Punkt 4 so lange, bis ein Treffer vorhanden ist oder bis das Ende des SHM erreicht ist. Ist ein Treffer vorhanden, gehe zu Punkt 3.
6. Wurde das Ende des SHM erreicht und keine Übereinstimmung gefunden, sende «File not found» an den Client

Beispiel für gültige Eingaben von READ:
–READ filename.txt

Als Antwort des Servers gibt es zwei Varianten. Entweder «File not found» oder der Inhalt des Files wird übermittelt.

Beispiele für ungültige Eingaben von READ:
–READ
Wenn kein Filename mitgegeben wird, antwortet der Server mit der Mitteilung: «Cannot read an empty filename».

Der Programmcode für das Handling von READ ist im Listing A.3 in Anhang A.2.3 zu finden.

5.10. UPDATE

Die Funktion für ein UPDATE eines Files wurde in zwei Teilschritten implementiert. Der erste Teil löscht das komplette File, der Zweite erstellt es mit dem neuen content wieder. Das Problem beim direkten Update eine Files könnte sein, dass der content grösser wird als der SHM-Block ist. Anstelle dies zu überprüfen, wurde global ein DELETE und CREATE implementiert.

1. Überprüfe ob ein Filename mitgegeben ist. Wenn nicht, sende an Client, dass ein File ohne Namen nicht geändert werden kann.
2. Überprüfe ob ein neuer Filecontent mitgegeben wurde. Wenn nicht, sende an den Client, dass der Content nicht leer sein darf.
3. Überprüfe, ob das File exisiert. Wenn nicht, sende an den Client, dass es nicht existiert.
4. Überprüfe im ersten SHM-Block, ob der Filename mit dem gesuchten Name übereinstimmt.
5. Gibt es einen Treffer:
   - rufe Funktion deleteFile(...) auf - siehe Kapitel 5.11
   - rufe Funktion createNewfile(...) auf - siehe Kapitel 5.8
   - sende dem Client die Nachricht, dass das File aktualisiert wurde.
6. Gibt es keinen Treffer, gehe zum nächsten SHM-Block und überprüfe, ob der Namen übereinstimmt. Wiederhole so lange, bis der Filename gefunden wurde. Bei einem Treffer, gehe zu Schritt 5

Beispiel für gültige Eingaben von UPDATE:
–UPDATE test.txt dies ist der neue content

Als Antwort des Servers gibt es zwei Varianten. Entweder «File with the name xyz does not exist!» oder «File with name xyz was successfully udpated.»

Beispiele für ungültige Eingaben von UPDATE:
–UPDATE
–UPDATE test.txt

Wenn kein Filename mitgegeben wird, antwortet der Server mit der Mitteilung: «Cannot update a file without a name».

Der Programmcode für das Handling von UPDATE ist im Listing A.4 in Anhang A.2.4 zu finden. Die Einschränkung zu UPDATE ist im Kapitel 4.4.1 Einschränkungen gemäss Implementationsdefinitionen zu finden.

5.11. DELETE

Das vereinfachte Vorgehen beim DELETE ist folgendermassen:

1. Überprüfe ob ein Filename mitgegeben ist. Wenn nicht, sende an Client, dass ein File ohne Namen nicht gelöscht werden kann.
2. Überprüfe im ersten SHM-Block, ob der Filename mit dem gesuchten Name übereinstimmt. Wenn ein Treffer vorhanden ist:
   - setze den Write-Lock
   - setze den Filenamen = NULL
   - setzte den Filecontent = "\0"
   - setzte Flag im SHM-Block is free = TRUE
   - gib den Write-Lock wieder frei
   - destroy den rwlock mit pthread_rwlock_destroy(...)
   - sende dem Client die Nachricht, dass die Datei erfolreich gelöscht wurde.
3. Ist kein Treffer vorhanden, gehe zum nächsten Block und wiederhole dies so lange, bis ein Treffer vorhanden ist. Bei einem Treffer gehe zu Punkt 2
4. Ist bis zum letzten SHM-Block kein Treffer vorhanden, sende dem Client, dass der Filename nicht gefunden wurde und somit nicht gelöscht werden kann.

Beispiel für gültige Eingaben von DELETE:
–DELETE filename.txt

Als Antwort des Servers gibt es zwei Varianten. Entweder «File with the name xyz does not exist» oder «File with name xyz was successfully deleted.».

Beispiele für ungültige Eingaben von READ:
–DELETE

Wenn kein Filename mitgegeben wird, antwortet der Server mit der Mitteilung: «Cannot delete a file without a name».

Der Programmcode für das Handling von DELETE ist im Listing A.5 in Anhang A.2.5 zu finden.

5.12. LIST

Der Befehl für LIST, welcher gemäss Dozent für den Server vorhanden sein sollte, wurde nicht so implementiert wie vorgeschlagen.
Gemäss Vorschlag soll der Client ein «LIST» schicken, worauf der Server mit der Anzahl Files und den einzelnen Filenamen anwortet.
Die implementierte Antwort von LIST des Server ist gemäss Abbildung 5.1 zu sehen.
In der Antwort enthalten ist die Block Nummer des SHM-Blockes, dessen Adresse auf dem Speicher, Grösse und ob es der letzte Block im SHM ist. Zusätzlich ist zu sehen, ob der Block mit einem File belegt ist oder nicht. Ist ein File vorhanden, steht der entsprechende Filename, sonst NULL. Ist ein File vorhanden, ist dessen Grösse zu sehen. Ganz am Ende werden die ersten 16 Zeichen des entsprechenden Fileinhaltes angezeigt.

```
# Message from Server:
================================================ ALL BLOCKS OF SHARED MEMORY ================================================
Block No        Block Address       Block-Size      isFree  Filename            Filesize        is Last         first 16 char of Filedata
=============================================================================================================================
1               a2d010              128             0       Test.txt            70              0               Dies ist ein Bei...
2               9c0036c0            128             1       NULL                0               0
3               9c0035c0            256             0       Main.java           190             0               Die Klasse Main....
4               9c0034c0            512             1       NULL                0               0
5               9c0033c0            1024            1       NULL                0               0
6               9c0032c0            2048            1       NULL                0               0
7               9c0031c0            4096            1       NULL                0               0
8               9c0030c0            8192            1       NULL                0               0
9               9c002fc0            16384           1       NULL                0               0
10              9c002ec0            32768           1       NULL                0               1
================================================ END OF SHARED MEMORY ================================================
```

Abbildung 5.1.: Server-Befehl LIST
Quelle: eigener Screenshot

Beispiel für gültige Eingaben von LIST:
Es gibt nur einen einzigen gültigen Befehl für LIST:
–LIST

Werden nach LIST noch weitere Argumente mitgegeben, werden diese ignoriert.

6. Fazit

Beim Kickoff Meeting und der Bekanntgabe der Eckdaten für die Programmieraufgabe kamen einigen der Teilnehmern, mich eingeschlossen, die Aufgabenstellung als nicht lösbar vor, wie auch im Kapitel 3.4 vermerkt ist. Als nicht Programmierer und ohne Kenntnisse in der C-Programmiersprache schien es als absolut nicht machbar.
Nichts desto trotz gab es keine andere Möglickkeit, mindestens einen Teil der Mission Impossible zu lösen.

Da ich wusste, dass diese Arbeit ein harter Brocken werden würde, habe ich mich frühzeitig hingesetzt und mit dem Grundkonzept angefangen. Während des Semesters und dem Modul «Systemsoftware» bekam ich jedoch einen Einblick in die C-Programmiersprache und der Server nahm von Woche zu Woche mehr an Gestalt an.
Da ich nicht wusste, ob ich die Aufgabe meistern konnte, habe ich mich entschieden, meine aktuellen Stände immer zu dokumentieren (siehe Kapitel 2) und auf github zu aktualisieren. So wäre immerhin ein roter Faden in der Arbeit zu sehen.
Zu meiner eigenen Verwunderung wuchs der Server- und der Clientteil der Aufgabe zu einem Konstrukt, das mir selber Freude bereitete, da ich sehen konnte, wie Fortschritte erzielt wurden.

In den letzten Wochen vor der Abgabe wurde die Dokumentation detaillierter und der Server auf seine Funktionalitäten besser geprüft. So konnten noch einige «segmentation faults» behoben werden. Diese traten vor allem auf, wenn ein Befehl wie CREATE ohne zustäzliche Argumente an den Server gestellt wurden.

Durch den eigenen Ehrgeiz, die während dem Kickoff gestellten Anforderungen zu erfüllen, enstand schlussendlich die finale Version des Server. Sie hat zwar keinen Anspruch auf Fehlerfreiheit, funktioniert jedoch sehr gut. Nicht zu vergessen sind jedoch die zusätzlich aufgewendenten Stunden, um der C-Programmiersprache mächtig zu werden.
Die Einarbeitungszeit, die Programmierung und die Dokumentation haben die doppelte Zeit in Anspruch genommen als für dieses Seminar veranschlagt war.

Trotz des viel grösseren Zeitaufwandes hat die Arbeit grundsätzlich Spass gemacht. Durch diese Arbeit konnte sehr viel Wissen angeeignet werden. Einzig das Debugging war teils sehr mühsam.
Müsste diese Arbeit mit dem angeeigneten Wissen (durch Unterricht Systemsoftware und eigenes Erarbeiten für diese Semiararbeit) nochmals wiederholt werden, denke ich, dass die Aufgabe im Zeitrahmen der vorgeschlagenen 50-60 Stunden gemeistert werden kann.

A. Anhang

A.1. Screenshots zum Server

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

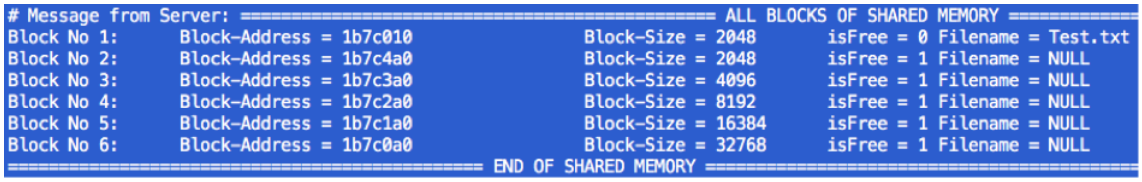


Abbildung A.1.: SHM vor dem Einfügen eines neuen Files
Quelle: eigener Screenshot

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

A.2. Programm-Code

A.2.1. Aufteilen des Shared Memory in der Funktion devide()

```
1  /* Will devide the shm control into pieces. a place for a file
2   is bigger than the file length, but less than 2 times the file length
3   *  e.g. 32KB free block is requested
4   *
5   *  64KB    |       64KB   | 16KB    | ...
6   *  not free |       free     |Â not free | ...
7   *
8   *  the two 32KB can be combined
9   *
10  *  result :
11  *  64KB    | 32KB | 32 KB | 16KB    | ...
12  *  not free | free | free    |Â not free | ...
13  */
14 int devide(struct shm_ctr_struct *shm_ctr, int untilSize) {
15   int retrcode = FALSE;
16   /* check if place can be devided */
17   if (shm_ctr->isfree == TRUE) {
18
19     /* new setting for devided, next place */
20     struct shm_ctr_struct *nextshm;
21     nextshm = malloc(sizeof(struct shm_ctr_struct));
22
23     struct shm_ctr_struct *tmpnext = shm_ctr->next;
24
25     /* new setting for this place */
26     int newsize = (shm_ctr->shm_size) / 2;
27     shm_ctr->shm_size = newsize; // setting size
28     shm_ctr->next = nextshm; //setting next
29
30     /* new setting for next place */
31     nextshm->prev = shm_ctr; // setting prev = this
32     nextshm->shm_size = newsize; // setting size
33     nextshm->isfree = TRUE; //setting isFree
34     nextshm->next = tmpnext; //setting next
35     nextshm->isLast = FALSE; // set is Last = 0
36     nextshm->filename = malloc(sizeof(char) * 128);
37     nextshm->filename = "NULL";
38     nextshm->filedata = (shm_ctr->filedata) + (shm_ctr->shm_size);
39
```

```
40      /* check if this was last one */
41      if (shm_ctr->isLast == TRUE) {
42        shm_ctr->isLast = 0; //FALSE
43        nextshm->isLast = 1; //TRUE
44        nextshm->next = nextshm;
45      }
46      /* if the size is the size which should be, return TRUE */
47      if (newsize == untilSize) {
48        LOG_TRACE(LOG_INFORMATIONAL, "PT: After deviding I have a good
              block size.");
49        retrcode = TRUE;
50        return retrcode;
51        /* if the size is not good, call recursive the function and split it, until the
              size is good */
52      } else {
53        if (newsize <= 2)
54          return FALSE;
55
56        LOG_TRACE(LOG_NOTICE, "PT: Recursive call in deviding because block size
              is to big (at moment = %i / should be: %i) ...",
57            newsize, untilSize);
58        retrcode = devide(shm_ctr, untilSize);
59      }
60    }
61
62    /* if block is not free */
63    else {
64      shm_ctr = (shm_ctr->next)->next;
65      retrcode = devide(shm_ctr, untilSize);
66    }
67    return retrcode;
68  }
```

Listing A.1: devide() - Aufteilen der Blöcke des SHM

A.2.2. CREATE

```
1  /* write a new file with its content to the shared memory.
2   * @return: Pointer to the shm control struct where the pointer of the filename and
        filecontent is
3   */
```

```
4  char * writeNewFile(struct shm_ctr_struct *shm_ctr, char *filename, char *filecontent,
       int filesize ) {
5    LOG_TRACE(LOG_DEBUG, "Now in Function writeNewFile()");
6    int retcode;
7    char *returnchar = malloc(sizeof(char) * 64);
8    memset(returnchar, '\0', sizeof(returnchar)); //clear return String
9
10   /* check if file already exists */
11   retcode = checkifexists(shm_ctr, filename);
12   /* if extist, return "File already exist */
13   if (retcode == TRUE) {
14     LOG_TRACE(LOG_INFORMATIONAL, "File \"%s\" already exists", filename);
15     return "File already exist\n";
16   }
17   /* if file does not exists, create a new file */
18   LOG_TRACE(LOG_INFORMATIONAL, "File with name %s does not exists. I will
         create it.", filename);
19
20   LOG_TRACE(LOG_DEBUG, "Address of shm Place to check is %p", shm_ctr);
21   struct shm_ctr_struct *place = find_shm_place(shm_ctr, filesize);
22   LOG_TRACE(LOG_DEBUG, "Checked a good address is: %p", place);
23
24   if (place == FALSE) {
25     LOG_TRACE(LOG_DEBUG,
26         "0 is not valid. So there is no good place to write the file into ... Trying no
               to devide the Shared Memory...");
27
28     int block_size_needed = round_up_int(filesize);
29     retcode = devide(shm_ctr, block_size_needed);
30     if (!retcode) {
31       return "There was no place in the shared memory for creating the file!";
32     }
33     place = find_shm_place(shm_ctr, filesize);
34   }
35
36   /* if a good place was found */
37   if (place != FALSE) {
38     place->isfree = FALSE;
39     /* init the read write lock for this file */
40     pthread_rwlock_init(&(place->rwlockFile), NULL);/* Default initialization */
41     /* lock the file */
```

```
42      retcode = pthread_rwlock_wrlock(&(shm_ctr->rwlockFile));
43      if (retcode == 0)
44        LOG_TRACE(LOG_NOTICE, "Locked RWLock for Writing new filename \"%s
              \"", filename);
45      place->filename = strdup(filename);
46      place->filedata = strdup(filecontent);
47      LOG_TRACE(LOG_NOTICE, "File \"%s\" successfully created in RWLock",
            place->filename);
48      /*unlock the file */
49      pthread_rwlock_unlock(&(shm_ctr->rwlockFile));
50      if (retcode == 0)
51        LOG_TRACE(LOG_NOTICE, "Unlocked RWLock for Writing filename \"%s\""
              , shm_ctr->filename);
52
53      return getSingleString("File \"%s\" successfully created.", (place->filename));
54    }
55    return -1;
56  }
```

Listing A.2: CREATE - Funktionsweise

A.2.3. READ

```
1  /* return the content of a file as a char Pointer
2   * Before reading the file content, a read lock will be made.
3   */
4  char * readFile(struct shm_ctr_struct *shm_ctr, char *filename) {
5    int retcode;
6    LOG_TRACE(LOG_DEBUG, "Now in function readFile(). Filename to find: \"%s
         \"\t current filename: \"%s\"", filename, shm_ctr->filename);
7    char * retchar = "File not found";
8    /* if file found */
9    if (strcmp(filename, (shm_ctr->filename)) == 0) {
10     /* lock for Reading*/
11     retcode = pthread_rwlock_rdlock(&(shm_ctr->rwlockFile));
12     if (retcode == 0)
13       LOG_TRACE(LOG_NOTICE, "Locked RWLock for Reading filename \"%s\"",
             shm_ctr->filename);
14     char * filedata = (char*) malloc(sizeof(char) * MAX_FILE_LENGTH);
15     filedata = strdup(shm_ctr->filedata);
16     pthread_rwlock_unlock(&(shm_ctr->rwlockFile));
17     if (retcode == 0)
18       LOG_TRACE(LOG_NOTICE, "Unlocked RWLock for Reading filename \"%s\"
             ", shm_ctr->filename);
19     return filedata;
20
21   }
22
23   /* if last = return not found */
24   else if (shm_ctr->isLast == TRUE) {
25     return "File not found. No content to display.";
26   }
27
28   else {
29     shm_ctr = shm_ctr->next;
30     LOG_TRACE(LOG_DEBUG, "Recursive call of readFile()");
31     retchar = readFile(shm_ctr, filename);
32   }
33
34   return retchar;
35 }
```

Listing A.3: READ - Funktionsweise

A.2.4. UPDATE

```
1  else if (strcmp(command, "UPDATE") == 0) {
2      char *returnvalue = malloc(sizeof(char) * MAX_FILE_LENGTH);
3      LOG_TRACE(LOG_INFORMATIONAL, "Will no try to UPDATE the file \"%i\"
           ", recMessage[2]);
4      int retcode;
5
6      /* check if filename was given */
7      if (recMessage[2] == NULL) {
8        LOG_TRACE(LOG_INFORMATIONAL, "Can not update a file without a name
             ");
9        returnvalue = "Can not update a file without a name";
10       send(clntSocket, returnvalue, strlen(returnvalue), 0);
11     }
12
13     /* check if new content was given*/
14     else if (recMessage[3] == NULL) {
15       LOG_TRACE(LOG_INFORMATIONAL, "Can not update a file without any
             content");
16       returnvalue = "Can not update a file without any content";
17       send(clntSocket, returnvalue, strlen(returnvalue), 0);
18     }
19
20     else {
21
22       retcode = checkifexists(shm_ctr, recMessage[2]);
23
24       /* if file does not exist, send message to Client */
25       if (!retcode) {
26         sendtoClient = getSingleString("File with the name \"%s\" does not exist!\n",
               recMessage[2]);
27         send(clntSocket, sendtoClient, strlen(sendtoClient), 0);
28       }
29
30       /* if file exist, delete it and create it new */
31       else {
32         retcode = deleteFile(shm_ctr, recMessage[2]);
33         /* if deleting was successful, call runClientCommand and */
34         if (retcode) {
35
```

```
36            /* combine now the free blocks */
37            retcode = combine(shm_ctr);
38            /* repeat until there is no more deviding option */
39            while (retcode == TRUE) {
40              retcode = combine(shm_ctr);
41            }
42
43            /* create the new file with the content */
44            char *tmpcontent = (char *) malloc(MAX_FILE_LENGTH);
45            tmpcontent = getFileContent(recMessage);
46            char * filecontent  = strdup(tmpcontent);
47            free(tmpcontent);
48
49            char *filename = strdup(recMessage[2]);
50            LOG_TRACE(LOG_INFORMATIONAL, "Filesize = %i \t Content = %s",
                  (int) strlen(filecontent), filecontent);
51
52            char *returnvalue = malloc(sizeof(char) * 256);
53            returnvalue = createNewFile(shm_ctr, filename, filecontent, strlen(
                  filecontent));
54            if (returnvalue > 0) {
55              returnvalue = getSingleString("File \"%s\" was successfully updated",
                    filename);
56              LOG_TRACE(LOG_INFORMATIONAL, "Sending message to Client: %s"
                    , returnvalue);
57              send(clntSocket, returnvalue, strlen(returnvalue), 0);
58            }
59            free(returnvalue);
60
61          }
62
63          /*if deleting was not successful */
64          else {
65            LOG_TRACE(LOG_INFORMATIONAL, "Updating file \"%s\" was not
                  successful", recMessage[2]);
66            sendtoClient = getSingleString("Updating file \"%s\" was not successful",
                  recMessage[2]);
67            send(clntSocket, sendtoClient, strlen(sendtoClient), 0);
68          }
69
70        }
```

```
71
72      }
73    }
```

Listing A.4: UPDATE - Funktionsweise

A.2.5. DELETE

```
1  /* if a file should be delete, the used shared memory will be free after.
2   * This function will set new pointer of the shm_ctr_struct which is handling the files
       */
3  int deleteFile(struct shm_ctr_struct *shm_ctr, char *filename) {
4    LOG_TRACE(LOG_DEBUG, "In function deleteFile()");
5    int retcode = FALSE;
6    /* if hit, make settings */
7    if (strcmp(filename, (shm_ctr->filename)) == 0) {
8      LOG_TRACE(LOG_DEBUG, "Filename \"%s\" found...", filename);
9
10     /* lock the file now */
11     retcode = pthread_rwlock_wrlock(&(shm_ctr->rwlockFile));
12     if (retcode == 0)
13       LOG_TRACE(LOG_NOTICE, "Locked RWLock for deleting filename \"%s\"",
             filename);
14
15     /* deleting file content, filename .... */
16     shm_ctr->filename = "NULL";
17     LOG_TRACE(LOG_DEBUG, "Filedata was until now: %s\n", shm_ctr->filedata
           );
18     memset((shm_ctr->filedata), 0, shm_ctr->shm_size);
19     shm_ctr->isfree = TRUE;
20
21     /*unlock the file */
22     pthread_rwlock_unlock(&(shm_ctr->rwlockFile));
23     if (retcode == 0)
24       LOG_TRACE(LOG_NOTICE, "Unlocked RWLock for deleting filename \"%s\""
             , shm_ctr->filename);
25
26     pthread_rwlock_destroy(&(shm_ctr->rwlockFile));
27     LOG_TRACE(LOG_NOTICE, "RWLock now destroyed after deleting file");
28
29     return TRUE;
30   }
31   /* if at end of SHM and no hit, return FALSE */
32   else if (shm_ctr->isLast == TRUE) {
33     LOG_TRACE(LOG_DEBUG, "At the end of SHM. No filename found.");
34     return FALSE;
35   }
```

```
36    /* if no hit and not at end of shm, go to next */
37    else {
38      LOG_TRACE(LOG_DEBUG, "Recursive call of function deleteFile()");
39      retcode = deleteFile((shm_ctr->next), filename);
40    }
41    return retcode;
42  }
```

Listing A.5: DELETE - Funktionsweise

Literaturverzeichnis

[1] *Github Repository - Dozent Systemsoftware - Karl Brodowsky.* https://github.com/bk1/sysprogramming-examples, june 2014. 3.1

[2] A.Rago, W. Richard Stevens; Stephen: *Advanced Programming in the UNIX Environment, 3rd Edition.* Pearson Education, Inc., 2010.

[3] Kaiser, Ulrich: *C/C++ - Von den Grundlagen zur professionellen Programmierung.* Galileo Computing, 2000.

[4] Wolf, Jürgen: *Grundkurs C.* Galileo Computing, 2010.