Concurrent C Programming

Capture the Flag with ØMQ

ZHAW – School of Engineering

Seminar: **Concurrent C Programming**

Dozent: **Nico Schottelius**

Autor: **Hauri David**

Datum: **20.06.2015**

I. Änderungsnachweis

Version	Datum	Autor	Änderungen
0.1	09.05.2015	HAD	Initial Draft
0.2a	10.05.2015	HAD	Fuss- und Kopfzeilen
0.2b	12.05.2015	HAD	Arabisch und Römische Aufteilung der Seiten- sowie Titelnummerierung
0.3	15.05.2015	HAD	EBS Seminar Eintrag
0.4	20.05.2015	HAD	Thema: Einleitung
0.5	24.05.2015	HAD	Thema: Installationsanleitung
0.6	26.05.2015	HAD	Thema: Anleitung zur Nutzung Draft
0.7	28.05.2015	HAD	Thema: Anleitung Make
0.8	02.06.2015	HAD	Thema: Anleitung Params
0.9	06.06.2015	HAD	Thema: Protokoll Draft
0.10	09.06.2015	HAD	Thema: Protokoll Issues
0.11	12.06.2015	HAD	Thema: Protokoll Requirements
0.12	14.06.2015	HAD	Thema: Herausforderungen Draft
0.13	16.06.2015	HAD	Thema: Loadbalancer
0.14	18.06.2015	HAD	Thema: Key Value Store
0.15	20.06.2015	HAD	Dokument überarbeitung
0.16	21.06.2015	HAD	Dokument: Glossar + Verzeichnisse
1.0	21.06.2015	HAD	Feinschliff

II. Inhaltsverzeichnis

1 EBS-Eintrag

1.1 Ausgangslage

Das Seminar setzt Kenntniss der Programmiersprache C voraus. Konzeption und Entwicklung eines Programms, das gleichzeitig auf einen Speicherbereich zugreift. Die Implementation erfolgt mithilfe von Threads oder Forks und Shared Memory (SHM).

1.2 Ziel der Arbeit

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.

1.3 Aufgabenstellunge

1.3.1 Ziel

Eroberung aller Felder des Spielfeldes durch einen Spieler.

1.3.2 Spielaufbau

Das Spielfeld ist ein Quadrat der Seitenlänge n, wobei n >= 4 ist. Die Koordinaten des Spielfeldes sind somit (0..(n-1), 0..(n-1)).

1.3.3 Spielablauf

- Der Server startet und wartet auf n/2 Spieler
- Sobald n/2 Spieler verbunden sind, kann jeder Spieler versuchen Felder zu erobern
- Es können während des Spiels neue Spieler hinzukommen oder Spieler das Spiel verlassen
- Wenn ein Spieler zu diesem Zeitpunkt alle Felder besitzt, hat er gewonnen und das Spiel wird beendet
- Der Server prüft alle y Sekunden den konsistenten Spielfeldstatus , wobei 1 <= y <= 30

1.3.4 Bedingungen für die Implementation

- Es gibt keinen globalen Lock (!)
- Der Server speichert den Namen des Feldbesitzers
- Kommunikation via TCP/IP
- fork + shm (empfohlen)
    - oder pthreads
    - für jede Verbindung einen prozess/thread
    - Hauptthread/prozess kann bind/listen/accept machen
    - Rating Prozess/Thread zusätzlich im Server
- Fokus liegt auf dem Serverteil
    - Client ist hauptsächlich zum Testen und "Spass haben" da
    - Server wird durch Skript vom Dozent getestet
- Locking, gleichzeitiger Zugriff im Server lösen
- Debug-Ausgaben von Client/Server auf stderr

1.4 Erwartete Resultate

- ✓ Git Repository auf Github vorhanden
- ✓ Applikation lauffähig auf Linux
- ✓ Nach "make" Eingabe existiert (somit auch ein Makefile!)
- ✓ Seminarberich
- ✓ "server": Binary des Servers
- ✓ Sollte nicht abstürzen / SEGV auftreten
- ✓ "client": Executable zum Testen des Servers
- ✓ "doc.pdf": Dokumentation

1.5 Geplante Termine

- 2015-03-11-1700 CET : Kickoff
- 2015-03-15-2359 CET : GitRepo
- 2015-03-20-2359 CET : EBS - Eintrag
- 2015-04-01-2359 CET : 1. Zwischenstandsbericht
- 2015-05-01-2359 CET : 2. Zwischenstandsbericht
- 2015-06-21-2359 CET : Abgabe Arbeiten
- 2015-07-01-1700 CET : Präsentation ZL O4.10

2 Einleitung

Es soll ein kleines Spiel programmiert werden welches mithilfe von C unter UNIX läuft. Das Spiel Prinzip basiert auf dem Capture The Flag Modus. Mehrere Spieler loggen sich auf dem Server ein und probieren alle Felder zu erobert. Der Spieler der zuerst alle Felder erobert, gewinnt. Die Kommunikation zwischen Spieler und Server basiert auf TCP/IP um das ganze Netzwerkfähig zu machen. Für die TCP/IP Kommunikation wird ØMQ eingesetzt.

2.1 Git Repository

Für die Sourcecode verwaltung wurde für diese Seminararbeit unter Github ein Repository erstellt[1]

2.2 ØMQ – ZeroMQ

> ØMQ[2] (also spelled ZeroMQ, 0MQ or ZMQ) is a high-performance asynchronous messaging library aimed at use in scalable distributed or concurrent applications. It provides a message queue, but unlike message-oriented middleware, a ØMQ system can run without a dedicated message broker. The library is designed to have a familiar socket-style API.
>
> ØMQ is developed by a large community of contributors, founded by iMatix, which holds the domain name and trademarks. There are third-party bindings for many popular programming languages.

Als Basis für die Semester und Bachelor Arbeit habe ich mich intensiv mit ØMQ auseinandergesetzt. Diese Seminararbeit, bietet ein ideales Einsatzgebiet.

[1] GitHub Repository: https://github.com/chubbson/sem_os_hpctf.git
[2] ØMQ Wikipedia: http://en.wikipedia.org/wiki/%C3%98MQ

3 Anleitung zu Nutzung

Dieses Kapitel Beschreibt die Nutzung und Installation der Notwendigen Komponenten.

3.1 Installation von ØMQ (Unix)

Um ØMQ unter Unix verwenden zu können habe ich im Rahmen dieses Seminars eine Installationsanleitung geschrieben[3]

Dieser Installguide basiert auf verschiedenen Links:

- ZeroMQ: get the software[4]
- Czmq: get the software[5]
- GitHub: zeromq/czmq[6]
- GitHub: zeromq/libzmq[7]
- GitHub: imatix/zguide[8]

3.1.1 Systemvorbereitungen

Ich arbeite unter fedora 17, somit verwende ich yum install anstell von aptget oder ähnliches um Packages zu installieren. Folgende Packages müssen insalliert werden:

```
yum install asciidoc
yum install xmlto
yum install pstoedit
yum install inkscape
yum install libuuid-devel
ldconfig
```

3.1.2 Clone Repositories

Folgende Repositories müssen geklont werden

```
git clone git@github.com:chubbson/sem_os_hpctf.git
git clone git@github.com:zeromq/libzmq.git
git clone git@github.com:zeromq/czmq.git
git clone git@github.com:imatix/zguide.git
```

[3] ØMQ installation guide by David Hauri:
https://github.com/chubbson/sem_os_hpctf/blob/master/INSTALLZMQ.md
[4] ØMQ get the software:
http://zeromq.org/intro:get-the-software
[5] Czmq get the software:
http://czmq.zeromq.org/page:get-the-software
[6] GitHub czmq:
https://github.com/zeromq/czmq
[7] GitHub libzmq:
https://github.com/zeromq/libzmq
[8] GitHub zguide:
https://github.com/imatix/zguide

3.1.3 Install lowlevel zmq libraries

Führen Sie dafür folgende Komandos aus

```
cd libzmq
./autogen.sh
./configure
```

Sollte folgender Fehler aufreten, basiert auf libzmq Issue 1274[9]:

```
configure: error: Package requirements (libsodium) were
not met:
```

gehen Sie wie folgt vor:

```
./configure --without-libsodium
make
make install
```

zmq man pages müssten nun Verfügbar sein:

```
man zmq
```

zmq version prüfen

```
./configure --version
```

Dies müsste diese oder eine neuere Version anzeigen:

```
zeromq configure 4.2.0
generated by GNU Autoconf 2.69
```

[9] Libzmq issue 1274: https://github.com/zeromq/libzmq/issues/1274

3.1.4 Install CZMQ – High-level C Bindings ZeroMQ

Führen Sie dafür folgende Komandos aus

```
cd ..
cd czmq
./autogen.sh
make
make install
sudo ldconfig
```

Die czmq manpages müssten nun verfügbar sein

```
man czmq
```

Prüfen Sie ob im ld.so.conf[10] folgender eintrag vorhanden ist:

```
/usr/local/lib
```

Falls nicht, fügen sie diese Line dem File hinzu und führen nochmals ldconfig aus:

```
sudo vi /etc/ld.so.conf
# adding line /usr/local/lib to it
sudo ldconfig
```

3.1.5 Testing ØMQ

Starten Sie ein zmq hello world beispiel: Start hwserver

```
cd ..
cd zguide
cd examples
cd C
./build all
./hwserver
```

Starten Sie ein neues Terminal und führen sie das pendant hwclient aus

```
./hwclient
```

Es werden nun 10 Hello Messages vom Server gesendet, der Client bekommt diese und gibt sie aus.

Willkommen in der Welt von ØMQ

[10] ld.so.conf based on skydb issue 63: https://github.com/skydb/sky/issues/63

4 Anleitung zur Nutzung

Das vorherige Kapitel beschrieb die Installation von ØMQ. Folgendes Kapitel befasst sich mit dem Programm selber.

Sollte das Programm nicht bereits geklont sein, hier wäre nochmals das Repository dazu:

```
git clone git@github.com:chubbson/sem_os_hpctf.git
cd sem_os_hpctf
cd 0_hpctf_sem
```

Das Programm ist in 3 Komponenten aufgeteilt.

- Server – server.c
- Client – client.c
- Viewer – fldviewer.c

4.1 Repository Ordnerstruktur

Das Repository ist in folgende Struktur aufgeteilt:

```
\sem_os_hpctf     (Main Repo Folder)
+--\0_hpctf_sem   (Sourcecode folder)
|  +->\lib        (c files)
|  +->\includde   (header files)
+->\sripts        (make file scirts)
```

4.2 Building Sourcecode

Der Sourcecode kann mithilfe von Make gebuildet werden. Nachfolgend wird jeweils folgender Ordner als Basis Verzeichnis verwendet.

```
cd sem_os_hpctf/0_hpctf_sem
```

4.2.1 Erstellen des makefiles

Das Make File wird mit Anhand folgendem script generiert.

```
[ 0_hpctf_sem]$ ../scripts/create-make > makefile
```

4.2.2 Make Clean

Mithilfe vom make clean können bestehende builds gecleaned werden

```
[ 0_hpctf_sem]$ make clean
```

4.2.3 Make All

Mithilfe vom make oder make all werden die ganzen sourcen gebuildet

```
[ 0_hpctf_sem]$ make
```

Make all -> equivalent zu make

```
[ 0_hpctf_sem]$ make all
```

4.2.4 Make specific file

Im makefile stehen sind alle möglichen spezifischen build möglichkeiten aufgefürtuhrt, unten einige beispiele:

Builden des clients

```
[ 0_hpctf_sem]$ make client
```

Builden des server

```
[ 0_hpctf_sem]$ make server
```

Builden des fldviewer

```
[ 0_hpctf_sem]$ make fldviewer
```

4.2.5 Makefile build options

Das Makefile verwendet folgende Build optionen:

```
CFLAGS=-Wall -g -O2 -std=gnu99 -I. -I./include -
L./lib

LIBS=-lcitsky -lm -lpthread -lzmq -lczmq
```

4.3 Parameter

Dieses Kapitel befasst sich mit der Parametrisierung der Excecutables. Wenn die Programme jeweils gestartet werden, erscheint als erstes ein USAGE Part der die verschiedenen Optionen beschreibt.

4.3.1 Server Parameter

```
USAGE:

./server
  4            parsing int as size, at least 4
  -v           Verbose
```

Server param –v:

Startet das Programm im ‚Verbose' Mode, welcher zusätzliche Debug Messages ausgibt.

Server param 4 (Size n):

Definiert die Feldlänge wobei n eine Zahl sein muss. Es wird ein Quadratisches Feld der grösse n*n generiert welches mit den Koordinaten von (0..(n-1)), (0..(n-1)) angesprochen werden kann.

4.3.2 Client Parameter

```
USAGE:

./client
  -ms=1000    Update in ms, default rand 1-1000ms
  -s=1         Strategy 1-6 will calc with mod 6
```

Server param –ms=1000:

Startet den Client der nach jedem Take ein Sleep mit den angegeben Millisekunden ausführt. Wird dieser Parameter nicht explizit angegeben, wird ein Zufallszahl von 1..1000 ms generiert, der nach jedem Take ausgeführt wird.

Server param –s=1 (Strategie):

Dem Client kann angegeben werden, mitwelcher Strategie er gestartet wird. Aktuell existieren 6 Strategien. Auf den angegeben Parameter wird ein Modulo 6 gerechnet um automatisch eine gültige Strategie zu wählen.

4.3.3 FldViewer

```
USAGE:

./fldviewer
  -v              Verbose
  -ms=5000        Update in ms, default 5000
  -raw            Print field, Raw with x y coords
                  player if not, set print graph-
                  ical
  -col            Print field Colored, if this is
                  set explicitly, raw and col are
                  allowed (-raw -col)
```

FldViewer param –v:

Startet das Programm im ‚Verbose' Mode, welcher zusätzliche Debug Messages ausgibt.

FldViewer param –ms=1000:

Startet den FieldViewer werlche all 1000 ms das Spielfeld ausgibt. Der Default Wert beträgt 5000 ms.

FldViewer Param –raw:

Printet das Spielfeld im ‚raw' Modus. X,Y und Owner des Feldes.

FldViewer Param –col:

Printet das Spielfeld im Graphik Modus. Dies ist relevant, falls der raw sowie colored Modus aus gegeben werden soll.

5 Protokoll

Für die Kommunikation wurde ein Protokoll vorgegeben:

- Befehle werden mit \n abgeschlossen.
- Kein Befehl ist länger wie 265 Zeichen inklusiv \n
- Jeder Spieler kann nur 1 Kommando senden und muss auf die Antwort warten.

5.1 Protokoll Verhalten

In Folgendem Abschnitt wird das Protokoll verhalten beschrieben

5.1.1 Anmeldung

Erfolgreiche Anmeldung:

```
Client: HELLO\n
Server: SIZE n\n
```

Nicht erfolgreiche Anmeldung:

```
Client: HELLO\n
Server: NACK\n
    -> Trennt die Verbindung
```

5.1.2 Feld erobern: erfolgreich

Wenn kein anderer Spieler gerade ein Take für dasselbe Feld sendet, kann ein Client das Feld nehmen.

```
Client: TAKE X Y NAME\n
Server: TAKEN\n
```

5.1.3 Feld erobern: nicht erfolgreich

Wenn ein oder mehrere Clients gerade ein TAKE Befehl für dasselbe Feld sendet, sind alle bis auf der erste nicht erfolgreich.

```
Client: TAKE X Y NAME\n
Server: INUSE\n
```

5.1.4 Besitz anzeigen

Status Befehl, gibt den Namen eines Spielers zurück.

```
Client: STATUS X Y\n
Server: Name-des-Spielers\n
```

5.1.5 Spiel Ende

Sobald ein Client alle Felder besitzt wird der Gewinner bekanntgegeben. Diese Antwort kann auf jeden Client Befehl kommen, mit Ausnahme der Anmeldung kommen.

```
Server: END Name-des-Spielers\n
Client: - (beendet sich)
```

5.2 Protokoll Issues

PK-CMD-01	
Abschnitt:	Protokoll
Bezeichner:	PK-CMD-01
Name:	CMD Length 256
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: hoch Technologisches Risiko: leicht
Quelle:	Protokoll
Verantwortlich:	D. Hauri
Kurzbeschreibung:	Jedes Kommando besitzt eine Länge von maximal 256 Zeichen
Qualitäten:	REQ-F-01

Tabelle 1: PK-CMD-01

PK-CMD-02	
Abschnitt:	Protokoll
Bezeichner:	PK-CMD-02
Name:	CMD terminated by \n
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: mässig Technologisches Risiko: mässig
Quelle:	Protokoll
Verantwortlich:	D. Hauri
Kurzbeschreibung:	Jedes Kommando endet mit einer New Line ‚\n' Auch wenn es 256 Zeichen lang sein sollte, ist das letzte Zeichen davon ein \n
Qualitäten:	REQ-F-01

Tabelle 2: PK-CMD-02

PK-CL-01	
Abschnitt:	Protokoll
Bezeichner:	PK-CL-01
Name:	Logon succeed
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: hoch Technologisches Risiko: niedrig
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	HELLO -> SIZE N
Auslösendes Ereignis:	Logon
Akteure:	Client
Vorbedingung:	Server läuft
Nachbedingung:	Keine
Ergebnis:	SIZE N\n
Hauptszenario:	- Client sendet HELLO\n
Qualitäten:	REQ-F-06

Tabelle 3: PK-CL-01

PK-CL-02	
Abschnitt:	Protokoll
Bezeichner:	PK-CL-02
Name:	Logon failed
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: hoch Technologisches Risiko: niedrig
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	HELLO -> NACK
Auslösendes Ereignis:	Logon
Akteure:	Client
Vorbedingung:	Server läuft
Nachbedingung:	Keine
Ergebnis:	NACK\n
Hauptszenario:	- Client sendet HELLO\n - Max Players reached - Spiel hat noch nicht gestartet
Qualitäten:	REQ-F-03, REQ-F-07

Tabelle 4: PK-CL-02

PK-CL-03	
Abschnitt:	Protokoll
Bezeichner:	PK-CL-03
Name:	CTF Succeed
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: hoch Technologisches Risiko: mässig
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	TAKEN
Auslösendes Ereignis:	TAKE X Y NAME
Akteure:	Client
Vorbedingung:	Spiel läuft
Nachbedingung:	keine
Ergebnis:	TAKEN\n
Hauptszenario:	- Client sendet ein TAKE Befehl
Qualitäten:	REQ-F-05

Tabelle 5: PK-CL-03

PK-CL-04	
Abschnitt:	Protokoll
Bezeichner:	PK-CL-04
Name:	CTF Failed
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: niedrig Technologisches Risiko: mässig Performance Einfluss: moderat
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	INUSE
Auslösendes Ereignis:	TAKE X Y NAME
Akteure:	Client
Vorbedingung:	Spiel läuft
Nachbedingung:	Keine
Ergebnis:	INUSE\n
Hauptszenario:	- Client sendet ein TAKE Befehl - Feld wird bereits erobert
Qualitäten:	REQ-F-05

Tabelle 6: PK-CL-04

PK-CL-05	
Abschnitt:	Protokoll
Bezeichner:	PK-CL-05
Name:	Status
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: niedrig Technologisches Risiko: mässig
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	Status
Auslösendes Ereignis:	STATUS X Y
Akteure:	Client
Vorbedingung:	Keine
Nachbedingung:	Keine
Ergebnis:	Name des Spielers\n
Hauptszenario:	Client sendet ein STATUS Befehl
Qualitäten:	REQ-F-04

Tabelle 7: PK-CL-05

PK-CL-06	
Abschnitt:	Protokoll
Bezeichner:	PK-CL-06
Name:	Client, Send – Receive \n
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: mässig Technologisches Risiko: mässig
Quelle:	Protokoll
Verantwortlich:	D. Hauri
Akteure:	Client
Kurzbeschreibung:	Auf jedes Kommando welches vom Client gesendet wurde, wird eine Antwort vom Server erwartet.
Qualitäten:	REQ-F-09

Tabelle 8: PK-CL-06

PK-SV-01	
Abschnitt:	Protokoll
Bezeichner:	PK-SV-01
Name:	Start
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: niedrig Technologisches Risiko: hoch
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	Start
Auslösendes Ereignis:	START
Akteure:	Server
Vorbedingung:	n/2 Spieler haben sich im Spiel angemeldet.
Nachbedingung:	Spiel starten
Ergebnis:	Spieler dürfen TAKE Befehle senden
Hauptszenario:	- Es haben sich n/2 Spieler angemeldet
Qualitäten:	REQ-F-02, REQ-F-10

Tabelle 9: PK-SV-01

PK-SV-02	
Abschnitt:	Protokoll
Bezeichner:	PK-SV-02
Name:	End
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: niedrig Technologisches Risiko: moderat
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	END Name des Spielers
Auslösendes Ereignis:	Ein Spieler besitzt alle Felder.
Akteure:	Server
Vorbedingung:	Spiel läuft nicht
Nachbedingung:	Spiel beenden
Ergebnis:	Client beenden Sich
Qualitäten:	REQ-F-08

Tabelle 10: PK-SV-02

PK-SV-03	
Abschnitt:	Protokoll
Bezeichner:	PK-SV-03
Name:	Thread per Connection
Autor:	D. Hauri
Priorität:	Protokoll Stabilität: moderat Technologisches Risiko: hoch Skalierbarkeit: hock
Quelle:	Protokoll Verhalten
Verantwortlich:	D. Hauri
Kurzbeschreibung:	HELLO, startet einen neuen Thread pro Connection
Auslösendes Ereignis:	HELLO
Akteure:	Server
Vorbedingung:	Spiel läueft
Nachbedingung:	Keine
Ergebnis:	Neuer Worker Thread wird gestartet
Qualitäten:	REQ-NF-01

Tabelle 11: PK-SV-03

5.3 Anforderungen

Auf Basis der Protokoll und Spiel vorgaben wurden die Anforderungen in Funktionale und nicht Funktionale Anforderungen getrennt.

5.3.1Funktionale Anforderungen

ID	V.	Autor	Name	Quelle	Anforderungs-beschreibung
REQ-F-01	1.0	HAD	CMD256	PK-CMD-01 PK-CMD-02	Jedes Kommando besitzt eine maximale Länge von 256 und endet mit einem \n
REQ-F-02	1.0	HAD	Start	PK-SV-01	Haben sich n\2 Spieler im Spiel angemeldet, wird das Spiel gestartet.
REQ-F-03	1.0	HAD	Nack	PK-CL-02	Antwort des Servers, Client schickt einen ungültigen Befehl oder Logon konnte nicht durchgeführt werden.
REQ-F-04	1.0	HAD	Name	PK-CL-05	Server speichert die Namen des Spielers welcher ein Feld besetzt
REQ-F-05	1.0	HAD	Field lock	PK-CL-03 PK-CL-04	Wird ein Feld erobert, wird dieses Feld gelockt, bis es erobert und dessen neue Besitzer gespeichert wurde.
REQ-F-06	1.0	HAD	Size n	PK-CL-01	Das Spielfeld besitzt eine Länge von n, wobei es n*n Felder gibt.
REQ-F-07	1.0	HAD	Max Players	PK-CL-02	Maximal Anzahl an Spieler.
REQ-F-08	1.0	HAD	End	PK-SV-02	Spiel ist zu ende, sobald ein Spieler alle Felder besitzt.
REQ-F-09	1.0	HAD	Request Reply	PC-CL-06	Auf jede Anfrage des Clients, wird eine Antwort gesendet.
REQ-F-10	1.0	HAD	Subscribe	PK-SV-01	Server kann unabhängig vom Client, Meldungen verschicken.

Tabelle 12: Funktionale Anforderungen

5.3.2Nicht funktionale Anforderungen

ID	V.	Autor	Name	Quelle	Anforderungs-beschreibung
REQ-NF-01	1.0	HAD	Load Balancer	PK-CL-04, PK-SV-03	Es sollen mehrere worker Threads paralell, requests bearbeiten.
REQ-NF-02	1.0	HAD	Field Owner	PK-CL-05	Server speichert die Namen des Spielers welcher ein Feld besetzt

Tabelle 13: Nicht Funktionale Anforderungen

6 Herausforderungen

Im Laufe der Arbeit sind mehrere Herausforderungen aufgetaucht. Dieses Kapitel widmet sich den Wichtigsten. Eine der ersten Probleme die auftauchten, ist die Darstellung des Spielfeldes und die daraus resultierenden Maximale Spieler Anzahl[11] im Zusammenhang damit soll der Server jeweils den Namen des Besitzers speichert[12].

Durch die Anforderung Requets-Reply[13] und Subscribe[14] stellt sich natürlich das Problem der Port Verteilung. Auf welche Ports dürfen Anfrage gesendet werden und wo werden Antworten zurückgesendet und wo werden Statusupdates des Servers ohne Request Bezuges publiziert.

6.1 Maximal Player

Für die Darstellung des Spielfeldes, wurde gefragt wie viele verschiedene Stati ein Feld annehmen kann um die Eindeutigkeit der Spieler-Feld Zugehörigkeit klar zu definieren. Ein Feld soll grafisch auf der Konsole klar gekennzeichnet sein. Ein Feld sollte nicht mehr als ein Charakter Feld der Konsole in Anspruch nehmen. Der spielfeldrahmen sollte sich Farblich vom Rest abheben. Es wird das Alphabet in Gross und Klein Buchstaben verwendet [a-Z] sowie Space. Weiter werden Farbkombinationen von Vorder und –Hintergrund um eine grössere Spieleranzahl darzustellen. Aufgrund diesen Vorgaben wurde die Maximale Spieler ermittelt.

l = 26 [Letters a..z],

c = 6 [6 Different Colors]

$$p_{max} = c + (c^2 * l * 2)$$

$$p_{max} => 6 + 6 * 6 * 26 * 2 => 1878$$

Da in einem Feld jeweils dessen Spielername gespeichert wird muss noch ein Mapping geschaffen werden um immer denselben Spieler darstellen zu können. Es wird der Spielername des Besitzers eines Feldes gespeichert. Um eine eindeutige Spielerfarbe zufügen zu können, führt der Server noch eine PlayerId Liste.

Anhand der PlayerId lässt sich eine eindeutige Kombination aus Vorder- Hintergrund Farbe sowie Letter generieren welche es erlaubt 1878 Verschiedene Spielermöglichkeiten auf dem einem Charakterfelde in der Konsole, darzustellen.

[11] REQ-F-07: Maximal Player
[12] REQ-NF-02: Field Owner
[13] REQ-F-09: Request Reply
[14] REQ-F-10: Subscribe / REQ-F-02: Start

6.2 Load Balancer

Messages die Server seitig ankommen werden gequeuet. Diese werden der Reihe nach abgearbeitet. Eigentlich reicht ein Worker Thread um die ankommenden Messages zu verarbeiten. Durch die nun sehr hohe Anzahl an möglichen Spieler, ist es natürlich ratsam mehrere Worker Threads zur Verfügung zu stellen[15]. Die Anforderung dass nun Pro Connection ein Thread oder Prozess gestartet werden soll ist natürlich viel verlangt. Bei 1878 Potentielle Clients, wären dies 1878 Workerthreads die auf dem Server Parallel laufen.

Für das Prinzip der Skalierbarkeit wurde ein Loadbalancer, implementiert. Der Server hört und spricht über den Port 5555 auf seine Clients. Die Messages die dort ankommen werden über ipc an eine Backend.ipc Pipes weitergeleitet. Alle möglichen verfügbaren Workerthreads, hören auf diese Pipe. Ein verfügbarer Workerthread verarbeitet den Request und verarbeitet die Message und schreibt das Resultat zurück welche dann wieder dem Client gesendet wird, welcher den Request gesendet hat.

6.3 Request Reply

Der Client Kommuniziert über den Port 5555 mit dem Server. Für jede Anfrage die er sendet, erwartet er eine Antwort[16]. Bei der Prototyp Implementation wurde mit einem einzigen Workerthread gearbeitet. Dieser Leistet auch ohne Probleme die anfängliche Last. Als Faustregel empfiehlt ØMQ einen Workerthread pro 10000 Msg pro Sekunde. Da jeder Client auf eine Antwort wartet bevor neue Requests gesendet werden reicht dieser eine eigentlich aus.

[15] REQ-NF-01: Load Balancer
[16] REQ-F-09: Request Reply, REQ-F-01

6.4 Key Value Map

Der Server sendet über den Port 5556 Asynchrone Messages an den Client. Diese werden für Status Updates und sonstigen Meta Daten verwendet. Die Messages die über diesen Port publiziert werden erlauben einen gemeinsamen Key Value Store. Der Server entscheidet jeweils welche Messages publiziert und welche nur Lokal gehalten werden. Diese Messages können über Subscription angezapft werden um den Lokalen Store mit dem Game public store, abzugleichen. Somit können späte Spieler auch die Stati Abfragen, für ein Spiel, welches bereits gestartet wurde.

Folgende Messages werden als Key Value Pair gehalten:

```
Key               ValueDescription
[fldlen]          int: Size N
[%d][%d]          char[256]: Field Owner
[state]           int: Game State, FINISHED,
                  RUNNING, WAITING4PLAYERS
[plidxcnt]        int: Player index count
[winner]          char *: winner name
```

7 Review

Diese Seminar Arbeit gab mir die Gelegenheit mich vertieft mit ØMQ auseinander zu setzen. Ein erster Prototyp der Spiels funktionierte bereits im April. Dafür habe ich zwei Bücher durchgearbeitet. ZeroMQ: Code Connected Volume 1 [17], ZeroMQ von O'REILLY[18]. Diese eröffneten mir ganz neue Blinkwinkel. Mein Fokus wärend dieser Arbeit galt dem Vertiefen von ZeroMQ. Die Library ist auf c++ Basis und hat in jeglicher erdenklicher Sprache eine Portierung. Wir hatten eine kleine Prototypen Implementation in Core Bereich eines Produktes meines früheren Arbeitgebers. Leider kam ich nie dazu dieses Modul genauer zu analysieren und daran weiter zu entwickeln. Meines Wissens sollte dies der basisbaustaun einer Bus Architektur sein. Auf dessen Basis habe ich Meine Semester und Bachelor Arbeit geplant, die ebenfalls ZeroMQ einsetzen soll zur real time Workload Analyse. Im Hinblick darauf, bot dieses Seminar ein optimales Einarbeitungsszenario. Leider hat das studieren der Bücher wesentlich mehr Zeit in Anspruch genommen wie geplant. Von den vielen möglichen Lösungswegen konnte ich zum Schluss dann doch nur einer der Trivialsten umsetzen, da die Zeit doch schnell, recht eng wurde. Für die Zukunft werde ich es mir zu Herzen nehmen, den Prototyp Konstant weiter zu ziehen um früh genug Probleme lokalisieren und eliminieren zu können.

7.1 Ausblick

Funktionell scheint ist diese Arbeit nicht sehr ausgereift. Sie erfüllt jedoch den Zweck. Es gibt viele Fragmente und Code Blöcke die ich gerne architektonisch anders bauen möchte. Doch Inhaltlich habe ich von dieser Arbeit sehr viel Profitiert und gelernt. Ich konnte eine 3rd Party Library C verwenden, diese Builden und einsetzen. Durch das Studium der dazugehörigen Fachliteratur habe ich viele neue Patterns kennen gelernt die ich auch ansatzweise mit in den Code einfliessen liess. Da wäre Pirate Pattern für die Verbindung zum Server vom Client aus. Load Balancer Server seitig um die Requests zu verarbeiten. Binary Star Pattern für einen Backup Server um nahtlose Failover Szenarien sicher zu stellen, um dies zu implementieren hat mir leider die Zeit nicht mehr gereicht. Diese Arbeit habe ich als Spielwiese für meine SA/BA geplant. Diese beschäftigen sich mit einer eigen entwickelten ETL Architektur (DataFlowNetwork) die nach dem pipelining Prinzip Funktioniert. ZeroMQ soll dort zum Einsatz kommen um eine Realtime Workload Analyse zu ermöglichen. Auf dessen Basis sollen sich Bottlenecks in der Konfiguration einfach lokalisieren lassen.

[17] ISBN: 978-1-449-33406-2 ZeroMQ: Code Connected Volume 1
[18] ISBN: 978-1-449-33406-2 ZeroMQ: Messaging for Many Applications:

Auf dessen Basis schwebt mir als BA vor einen Kalibrator zu schreiben. Dieser soll anhand dieser Messdaten sich automatisch optimal Konfigurieren lassen.

III. Glossar

ZeroMQ:

ZeroMQ ist eine hoch performante asyncrhone messaging Bibliothek, mit dem Fokus auf skalierbare verteilte parallele Applikationen. Es supportet message queues. Doch anders als message orientierte middleware, laufen zmq Systeme ohne dedizierte Message Broker. Die Bibliothek ist einer ähnelt einem socket-stype API.

Serialisierung:

Die Serialisierung ist in der Informatik eine Abbildung von strukturierten Daten auf eine sequenzielle Darstellungsform. Serialisierung wird hauptsächlich für die Persistierung von Objekten in Dateien und für die Übertragung von Objekten über das Netzwerk bei verteilten Softwaresystemen verwendet

DataFlowNetwork:

Als Kernstück einer Import-Architekur meines Früheren Arbeitgebers. Datafow-Network (DFN) baut auf dem TPL (Task Paralell Library .Net) Framewok auf. Das Dataflow-Network setzt auf paralelle Verarbeitung und bietet eine einfache und effektive Nutzung von Multi Core Prozessoren

Git:

Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien, die ursprünglich für die Quellcode-Verwaltung des Linux-Kernels entwickelt wurde.

TPL Task parallel Library:

Die Task Parallel Library (TPL) ist die Kernkomponente für die Parallelisierung innerhalb des .NET-Frameworks.[19]

[19] http://de.wikipedia.org/wiki/Task_Parallel_Library#Task_Parallel_Library

Bottleneck:

Als Flaschenhals (Bottleneck) bezeichnet man diejenige Anlage, Funktion, Abteilung oder Ressource, die in einer betrachteten Periode die höchste Auslastung in der gesamten Prozesskette hat, damit den Durchfluss begrenzt und somit eine Kapazitätsgrenze für ein Gesamtsystem darstellt.[20]

[20] http://de.wikipedia.org/wiki/Flaschenhals_%28Logistik%29

IV. Tabellenverzeichnis

V. Literaturverzeichnis

ZeroMQ:

ZeroMQ: Messaging for Many Applications:

ISBN: 978-1-449-33406-2

ZeroMQ: Code Connected Volume 1:

ISBN: 978-1-449-33406-2

VI. Zeitplan

Arbeiten: Jeweils Freitag Nachmitag + Wochenende

Priorität	Aufgabe	Zuständigkeit	Geplanter Aufwand (in Stunden)		Tatsächlicher Aufwand (in Stunden)
1	Dokumentaion	HAD	16	16	16
4	Anforderungen definieren (EBS)	HAD	5	5	3
1	Analyse Prototyp	HAD	4	4	8
3	Umsetzung Prototyp	HAD	20	20	24
2	Lesen Fachliteratur ZMQ	HAD	40	40	60
1	ZMQ installieren, lauffähig	HAD	4	4	6
2	Umsetzung Programm	HAD	40	40	80
4	Erstellen der Präsentation	HAD	3	3	4
5	Erstellen Handout	HAD	2	2	2
			134	134	203

Priorität: 1 Hoch - 5 Niedrig