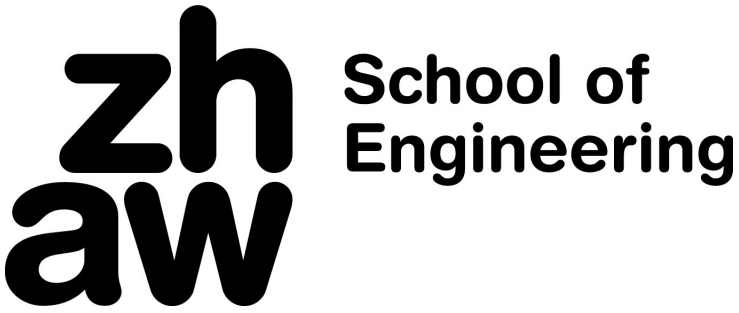
Zürcher Hochschule für Angewandte Wissenschaften

Seminararbeit

Concurrent C Programming



Dozent:
Nico Schottelius
???@zhaw.ch

Kandidat:
Benjamin Bütikofer
bbutik@microsoft.com
Matr.: 11-480-993

13. Juni 2015

Inhaltsverzeichnis

1 Projekt

1.1 Abstract

2 Einleitung

Im Seminar “Concurrent C Programming” welches als Vertiefung zum Kurs Systemsoftware und Betriebssystemprogrammierung gedacht ist, wurde die Aufgabe gestellt ein Multiuser und Netwerkfähiges Client/Server Spiel in C zu entwickeln.

Ziel des Spiels ist es alle Felder dies Spielfelds zu besitzen. Das Spielfeld ist ein Quadrat der Seitenlänge n, wobei

$$n \geq 4$$

ist. Die Koordinaten des Spielfeldes sind somit

$$(0..(n-1), 0..(n-1))$$

.

2.1 Spielablauf

1. Der Server startet und wartet auf $\frac{n}{2}$ Spieler
2. Sobald $\frac{n}{2}$ Spieler verbunden sind, kann jeder Spieler versuchen Felder zu erobern
3. Es können während des Spiels neue Spieler hinzukommen oder Spieler das Spiel verlassen
4. Der Server prüft alle y Sekunden den konsistenten Spielfeldstatus , wobei

$$1 \leq y \leq 30$$

5. Wenn ein Spieler zu diesem Zeitpunkt alle Felder besitzt, hat er gewonnen und das Spiel wird beendet

2.2 Spiel-Protokol

- Befehle werden mit \n abgeschlossen.
- Kein Befehl ist länger als 256 Zeichen inklusive dem \n.
- Jeder Spieler kann nur 1 Kommando senden und muss auf die Antwort warten.

2.2.1 Anmeldung

Listing 2.1: Erfolgreiche Anmeldung

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

Listing 2.2: Nicht erfolgreiche Anmeldung

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

2.2.2 Spielstart

Der Server wartet auf $\frac{n}{2}$ Verbindungen vor dem Start.

Listing 2.3: Spielstart

```
Server: START\n
Client: - (erwidert nichts, weiss das es gestartet hat)
```

2.2.3 Feld erobern

Wenn kein anderer Client gerade einen TAKE Befehl für das selbe Feld sendet, kann ein Client es nehmen.

Listing 2.4: Erfolgreiche Eroberung

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

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

Listing 2.5: Nichterfolgreiche Eroberung

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

2.2.4 Besitz anzeigen

Listing 2.6: Spielstart

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

2.2.5 Spielende

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

Listing 2.7: Spielstart

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

2.3 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 SSpass haben"da
- Server wird durch Skript vom Dozent getestet
- Locking, gleichzeitiger Zugriff im Server lösen
- Debug-Ausgaben von Client/Server auf stderr

3 Planung und Implementation

Bevor mit der Implementation begonnen wurde, wurden die Anforderungen genauer Analysiert und den Spielablauf mit einem simplen Blocks-and-Arrows Diagram zu Papier gebracht.

Mit dieser Visualisierung konnte im nächsten Schritt die Anforderungen an den Server geprüft werden. Danach wurde ein weiteres Diagram angefertigt, welches die Funktionalität des Servers und die Abhängigkeiten der einzelnen Komponenten des Servers aufzeigt, siehe Abbildung 3.1.

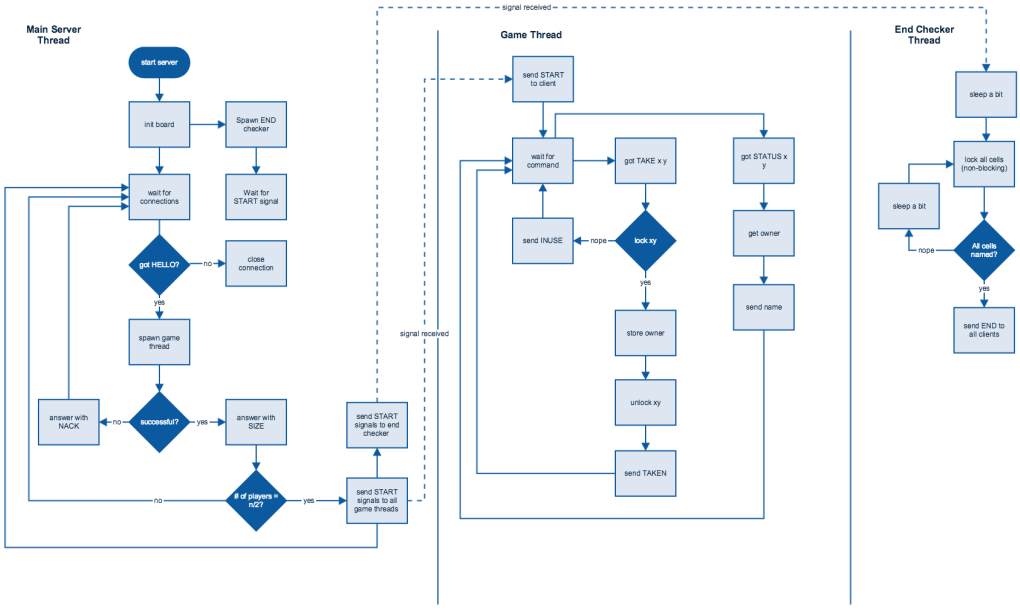


Abbildung 3.1: Server

3.1 Server

Wie Abbildung 3.1 zu entnehmen ist, besteht der Server aus drei Hauptkomponenten:

- Listener
- Game Thread
- End-Checker

3.1.1 Listener

Der Listener ist verantwortlich neue Verbindungen anzunehmen und speichern. Sobald genügend Clients verbunden sind, soll er auch die START Nachricht wie in 2.2.2 definiert an alle Clients versenden.

3.1.2 Game Thread

Für jeden Spieler wird ein neuer Thread gestartet. Der Thread behandelt alle Nachrichten des jeweiligen Clients und schickt auch die Antworten an ihn.

3.1.3 End-Checker

Der End-Checker Thread wird am Anfang einmal gestartet und läuft in einer Dauerschleife bis ein Gewinner ermittelt ist. Git es einen Gewinner, schickt der End-Checker die END Nachricht.

In der Aufgabenstellung wird gefordert, dass kein globaler Lock verwendet wird. Es wird aber auch verlangt, dass das Spielfeld konsistent geprüft werden muss. Da sich diese zwei Anforderungen jedoch nicht vereinbaren lassen, habe ich mich entschieden eine Lösung ohne globalen Lock zu entwickeln. Dies hat jedoch zur Folge, dass ein Spielfeld welches am Anfang getestet wird beim Abschluss der Prüfung aller Felder einen neuen Besitzer haben kann.

3.2 Client

Der Client ist äusserst simple implementiert und beherrscht nur einen Modus.

4 Fazit

Das Projekt war äusserst interessant und hat neben viel Ärger auch sehr viel Freude bereitet. Es war das erste Projekt welches ich in C geschrieben habe, demzufolge war die Lernkurve auch ziemlich Steil. Hinzukommt, dass ich beruflich nicht Vollzeit programmiere und es schon eine Weile her war seit meinem letzten Programmier-Projekt.

Glücklicherweise ist beinahe jedes Problem schon einmal gelöst worden und es gibt ausreichend Dokumentation im Internet. Auch der Tipp mit valgrind war äusserst nützlich.

Das Projekt zeigte mir auch deutlich warum Sprachen wie Java und C# soviel auftrieb erhalten haben. Muss man sich doch in C jede noch so kleine Funktion in der Regel selbst bauen. Ein TCP Server in Java kann inner Minuten geschrieben werden ohne das man sich gross Gedanken machen muss, wie man es im Hintergrund abläuft.

Auch der Umgang mit dem Speicher (Speicherreservierung) hat viel Nerven gekostet - umso schöner jedoch, wenn man dann endlich auch den letzten Fehler ausgeschliffen hat!

Abbildungsverzeichnis