Message Broker

Seminar Concurrent Programming in C

an der ZHAW am Standort Zürich

Reto Hablützel

April 30, 2014

Contents

1 Einleitung

Ein Message Broker ist eine zentrale Kommunikatinsschnittstelle für mehrere Programme. Sie ermöglicht es, dass verschiedenste Programme miteinander kommunizieren können, indem Nachrichten aneinander versendet werden. Ein Message Broker bietet typischerweise verschiedene Protokolle zur Kommunikation an.

stomp erwähnen

2 Funktionalität

Dieses Kapitel erläutert die Funktionalitäten, die im Message Broker im Rahmen dieses Seminars umgesetzt wurden.

2.1 Verbindungsauf- und abbau

Jede Verbindung zwischen einem Client und dem Message Broker beginnt mit einem kurzen Handshake. Der Client teilt dem Message Broker seinen Namen mit und der Message Broker bestätigt den Empfang. Sobald der Client die Kommunikation beenden will, teilt er dies dem Message Broker mit einer Disconnect Nachricht mit und kriegt wiederum eine Bestätigung.

2.2 Nachrichten empfangen

Ein verbundener Client kann einen Topic abonnieren und kriegt so vom Message Broker sämtliche Nachrichten zu diesem Topic zugestellt. Topics werden dynamisch erstellt, sobald sicher der erste Client dafür interessiert. Ein Client, der Nachrichten von einem Topic abonniert hat, wird Subscriber genannt.

2.3 Nachrichten senden

Ein verbundener Client kann Nachrichten zu einem bestimmten Topic an den Message Broker senden. Ein solcher Client wird als Publisher bezeichnet.

3 Konzept

3.1 Protokoll

Für die Umsetzung wurde eine abgespeckte Version vom STOMP[1] Protokoll implementiert. Die implementierten Kommandos werden unter aufgelistet. Auf sämtliche Kommandos vom Client kann der Server mit ERROR antworten, falls etwas schief gelaufen ist. Jedes Kommando hat die folgende Struktur:

Auf der ersten Zeile steht der Name des Kommandos in Grossbuchstaben. Auf folgenden Zeilen stehen durch Doppelpunkt separierte Key-Value Paare,

[1] https://stomp.github.io/

welche Header Werte darstellen. Nach den Header Werten folgt der Content, welcher durch eine Leerzeile vom Header getrennt ist. Das Ende eines Kommandos markiert wiederum eine Leerzeile gefolgt vom Null-Byte (hier ^@):

Listing 1: Struktur eines Kommandos

```
COMMAND
key: value
key: value
...

Content

^@
```

3.1.1 CONNECT

Wird vom Client an den Message Broker zum Beginn der Verbindung gesendet. Hat einen zwingenden Header 'login', welcher den Client identifiziert. Hat keinen Content. Sobald der Message Broker das Kommando den Client aufgenommen hat, sendet er zur Bestätigung CONNECTED.

3.1.2 CONNECTED

Wird vom Message Broker als Antwort auf das CONNECT Kommando versendet falls die Verbindung erfolgreich erstellt werden konnte. Hat weder Header noch Content.

3.1.3 ERROR

Wird vom Message Broker in verschiedenen Szenarien an den Client gesendet um einen Fehler mitzuteilen. Hat einen zwingenden Header 'message', der die Fehlermeldung enthält und keinen Content.

3.1.4 SEND

Wird von einem Client an den Message Broker gesendet um einen Nachricht zuzustellen. Hat einen zwingenden Header 'topic', der den Topic identifiziert, an den die Nachricht gesendet wird. Im Content steht der eigentliche Inhalt der Nachricht.

3.1.5 SUBSCRIBE

Wird von einem Client an den Message Broker gesendet um einen Topic zu abonnieren. Hat einen zwingenden Header 'destination', welcher den Topic identifiziert. Hat keinen Content.

3.1.6 MESSAGE

Wird vom Message Broker an den Client gesendet um eine Nachricht zuzustellen. Hat einen zwingenden Header 'destination', der den Topic identifiziert. Die Nachricht ist im Content.

3.1.7 DISCONNECT

Wird vom Client an den Message Broker gesendet um die Verbindung zu beenden. Hat weder Header noch Content.

3.1.8 RECEIPT

Wird vom Message Broker an den Client gesendet, sobald der Message Broker den DISCONNECT akzeptiert hat. Hat weder Header noch Content.

3.2 Speicherstruktur

Auf dem Server existieren zwei globale Listen: Die eine enthält alle Topics verknüpft mit einer Liste von Subscribern zum jeweiligen Topic. Die zweite Liste enthält alle Messages, die vom Message Broker empfangen wurden. Zudem enthält jede Nachricht eine Liste von Statistiken. Eine Statstik ist eine Tripel aus Subscriber, Anzahl versuchter Zustellungen und einen Timestamp der letzten erfolglosen Zustellung. Diese wird verwendet um zu bestimmen, welche Subscriber die Nachricht erhalten haben und ob ein erneuter Versuch der Zustellung gemacht werden soll.

Listing 2: Speicherstruktur

```
Topics:
  - topic1
      - subscriber1
      - subscriber2
  - topic2
      - subscriber1
      - subscriber3
      ..
  ..

Messages:
  - message1
      - topic
      - content
      - statistic1
          - last fail
          - nattempts
          - subscriber1
      - statistic2
```

```
        - last fail
        - nattempts
        - subscriber2
- message2
    ..
```

3.2.1 Einfügen einer neuen Nachricht

Wenn ein Client eine Nachricht an den Message Broker sendet, kopiert er die Liste von Subscribern von dem Topic, an den die Nachricht versendet wurde. Dann wird ein neuer Eintrag in der Liste der Nachrichten erstellt und die Subscriber werden in eine Liste von Statistiken überführt.

3.3 Komponenten

3.3.1 Handler

Der Handler erstellt pro Socket - Verbindung einen neuen Thread. Dies ist der Broker.

3.3.2 Broker

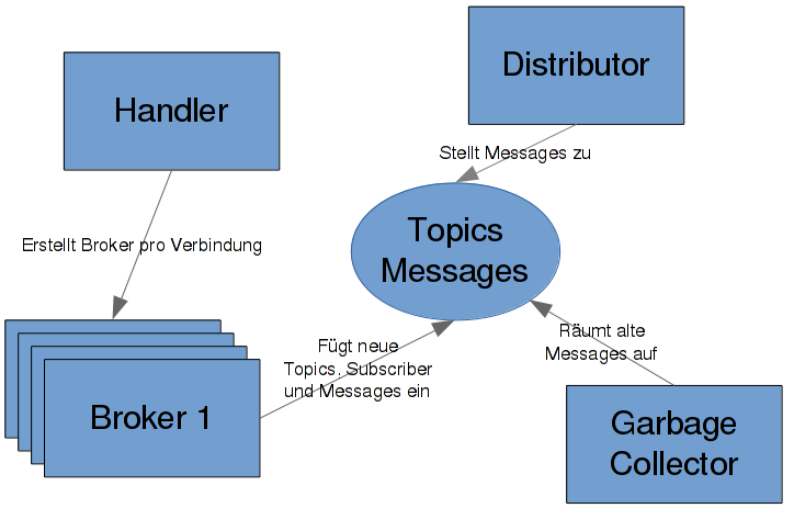
Pro verbundenem Client existiert ein Thread, der den Broker repräsentiert. Der Broker hält eine TCP Verbindung aufrecht und ist stets bereit, Kommandos zu lesen. Der Broker fügt den Client zur Liste der Subscriber, wenn er sich für einen Topic interessiert und nimmt eine Nachricht entgegen, wenn er eine sendet, und reiht diese in die Liste der Nachrichten.

3.3.3 Distributor

Der Distributor prüft ständig die Liste von Messages nach Nachrichten, die zugestellt werden sollen. Jedes Mal, wenn eine Nachricht erfolgreich zugestellt wurde, wird den Zähler 'nattempts' um eins erhöht. Falls die Zustellung fehlschlägt, wird der Timestamp 'last fail' gesetzt. Mit diesem Timestamp kann der Distributor auch feststellen, ob er bereits einen erneuten Versuch zur unternehmen soll.

3.3.4 Garbage Collector

Der Garbage Collector räumt Statistiken, Nachrichten und Subscriber auf. Mittels bestimmter Regeln entscheidet der Garbage Collector, ob ein Eintrag in der Liste der Statistiken zu einer Nachricht gelöscht werden kann (z.B. Nachricht wurde erfolgreich zugestellt oder Subscriber hat die Verbindung beendet). Sobald eine Nachricht keine Statistiken mehr hat, kann die gesamte Nachricht abgeräumt werden und aus der Liste der Messages gelöscht werden. Wenn ein Subscriber in keiner der Liste der Statistiken mehr auftaucht, kann auch er gelöscht werden.



4 Implementation

4.1 Locking

Damit das Programm auch mit mehreren interagierenden Threads stets in einem konsistenten Zustand bleibt, wurden die beiden globalen Listen mit Locks geschützt. Die folgenden Tabellen zeigt links die Resourcen, die geschützt werden müssen und in den Spalten 2-n die Aktionen, die diese Resourcen verändern. Mit R, W und M wird dann angedeutet, welcher Lock für eine bestimmte Aktion gehaltet werden muss.

- R: Es wurde ein Read Write Lock verwendet und dessen Read Lock muss gehalten werden
- W: Es wurde ein Read Write Lock verwendet und dessen Write Lock muss gehalten werden
- M: Es wurde eine Mutex verwendet, welche gehalten werden muss

Dabei ist es wichtig, dass die Locks top-to-bottom geholt werden. Wenn also eine Aktion zwei oder mehrere Locks für eine Aktion benötigt, muss zuerst der Lock gehalten werden, der oben in der Liste ist. Würde nämlich ein Thread die Locks von oben halten wollen und ein anderer von unten, könnte ein Deadlock auftreten.

4.1.1 Broker

Resource	Topic erstellen	Subscriber hinzufügen	Nachricht hinzufügen
Topics Liste	W	R	R
Subscriber Liste		W	R
Messages Liste			W
Statistik Liste			
Statistik			

4.1.2 Garbage Collector

Resource	Topic erstellen	Subscriber hinzufügen	Subscriber entfernen	Statistik löschen prüfen	Statistik löschen	Nachricht löschen prüfen	Nachricht löschen
Topics Liste	W	R / W *	R				
Subscriber Liste		W	W				
Messages Liste				R	R	R	W
Statistik Liste				R	W	R	
Statistik				R			

* Wenn ein Subscriber hinzugefügt werden soll, wird zuerst mit einem Read Lock auf der Liste der Topics geprüft, ob der Topic bereits existiert. Wenn der Topic noch nicht existiert, wird der Write Lock auf der Liste der Topics angefordert und die Existenz wird nochmal geprüft. Falls er noch immer nicht existiert, wird der Topic angelegt.

4.1.3 Distributor

Resource	Nachricht zustellen prüfen	Nachricht zustellen
Messages Liste	R	R
Statistik Liste	R	R
Statstik	R	W

4.1.4 Subscriber / Client

Nebste den diversen Locks für die beiden Listen, existieren noch drei Mutexe pro Client. Je eine wird für das Lese- und Schreib-Ende vom Socket verwendet und die dritte für ein Dead-Flag, welches auf 1 gesetzt wird, falls der Client nicht mehr erreichbar ist. Dies ist für den Garbage Collector auch ein Indikator, dass der Subscriber abgeräumt werden kann, sobald er auch in keiner Statistik mehr auftaucht.

4.2 Dateien

Sämtlicher Produktivcode ist im Verzeichnis src. Es folgt eine Auflistung der Dateien und kurze Beschreibung von deren Inhalt.

- server.c: Beinhaltet main-Funktion vom Message Broker. Hier werden alle Komponenten gestartet und es wird auf neue Clients gewartet.
- broker.c/broker.h: Funktionalität vom Broker. Die Funktion handle_client ist Einstiegspunkt für den Broker pro Client.
- distributor.c/distributor.h: Funktionalität vom Distributor. Die Funktion distributor_main_loop wird dem Distributor - Thread als Startfunktion übergeben und versucht mittels der anderen Funktionen in der Datei die Nachrichten zuzustellen.
- gc.c/gc.h: Funktionalität vom Garbage Collector. Die Funktion gc_main_loop wird dem Garbage Collector - Thread als Startfunktion übergeben und führt die Garbage Collector Funktion kontinuierlich aus.
- list.c/list.h: Eine Implementation einer verketteten Liste und einiger Operationen. Diese Liste wird überall verwendet (z.B. Liste von Topics).
- socket.c/socket.h: Ein High-Level Interface für Sockets. Im Gegensatz zu herkömmlichen Funktionen, die mit Byte-Array Buffer über einen Socket Daten versenden, kann mit diesen Funktionen komfortabel ein ganzes STOMP - Kommando über den Socket versendet bzw. empfangen werden.
- stomp.c/stomp.h: Beinhaltet Funktionen zum Parsen von Char - Arrays in ein STOMP - Kommando (struct) und wieder zurück. Diese Funktionen werden verwendet, um Kommandos über den Socket zu versenden.
- topic.c/topic.h: Beinhaltet die zentralen Strukturen und Funktionen zum operieren mit den beiden zentralen Listen von Messages und Topics. So gibt es zum Beispiel eine Funktion um einen Subscriber einem Topic hinzuzufügen.

4.3 Testing

Das ganze Programm wurde testgetrieben entwickelt und mit den Coverage Tools von GCC wurde kontinuierlich die Testabdeckung überwacht. Somit konnte schlussendlich eine hohe Abdeckung von über 97% erreicht werden. Als Test-Framework wurde cunit[2] verwendet.

Listing 3: Testabdeckung

```
Summary coverage rate:
  lines.....: 97.9% (967 of 988 lines)
  functions..: 97.2% (69 of 71 functions)
```

[2] http://www.cunit.sourceforge.net

5 Verwendung

Dieses Kapitel beschreibt die Anwendung des Programms. Der Hauptteil ist der Message Broker selbst, welcher grundsätzlich durch das oben beschriebene Protokoll verwendet werden kann. Im Rahmen dieser Seminararbeit wurde jedoch auch ein Testclient implementiert, der die Funktionalität demonstriert.

5.1 Kompilieren und Tests ausführen

Als Buildsystem wurde make verwendet und die Tests basieren auf cunit. Letzteres muss installiert sein, um die Tests auszuführen.

Um den Code für die Tests zu kompilieren muss folgender Befehl ausgeführt werden. Dieser wird die Tests auch direkt ausführen und einen Report ausgeben.

Listing 4: Tests kompilieren und ausführen

```
message-broker$ make test
```

Der Produktivecode wird mit folgendem Befehl erzeugt.

Listing 5: Produktivecode kompilieren

```
message-broker$ make
```

Dies erstellt die beiden Binaries run und test im Hauptverzeichnis.

5.2 Message Broker starten

Mit folgendem Befehl kann der Message Broker auf dem Standard Port gestartet werden. Welcher das ist, wird auf der Konsole ausgegeben.

Listing 6: Message Broker auf Standard Port

```
message-broker$ ./run
```

Falls ein anderer Port verwendet werden soll, kann dieser als Argument mitgegeben werden.

Listing 7: Message Broker mit bestimmten Port

```
message-broker$ ./run 44554
```

5.3 Testclient starten

Auch der Testclient kann ohne Argumente aufgerufen werden und wird sich dann mit dem Standard Port vom Message Broker verbinden.

Listing 8: Testclient mit Standard Argumenten

```
message-broker$ ./test
```

Zusätzlich bietet der Testclient einige Argumente um das Verhalten zu Steuern. Wichtig ist, dass alle Argumente in dieser Reihenfolge angegeben werden müssen.

- hostname: Der Hostname vom Message Broker
- port: Der Port vom Message Broker
- name: Der Name, der für das Login verwendet werden soll.
- nmsgs: Die Anzahl Nachrichten, die an den Broker versendet werden sollen.

Listing 9: Testclient mit bestimmten Argumenten

```
message-broker$ ./test localhost 44554 hmuster 33
```

6 Rückblick

7 Fazit

8 Anhang

8.1 Begriffe

Begriff	Erklärung
Topic	Nachrichten werden in Themen gruppiert. Clients abonnieren Themen und kriegen so die Nachrichten zugestellt, die an dieses Thema gesendet werden.
Publisher	Ein Client, der Nachrichten sendet
Subscriber	Ein Client, der an Nachrichten von einem Topic interessiert ist
STOMP	Simple Text Oriented Messaging Protocol