ENTWURF

ENTWURF ZUR AUFGABE 1 AUS DER VORLESUNGSREIHE “VERTEILTE SYSTEME”

TEAM 1: MICHAEL STRUTZKE, IGOR ARKHIPOV

Aufgabenaufteilung

Alle Aufgaben wurden gemeinsam entworfen und bearbeitet.

Quellenangaben

Aufgabenstellung, Vorstellung der Aufgabe in der Vorlesung, http://learnyousomeerlang.com/

Bearbeitungszeitraum

Datum	Dauer in Stunden
29.09.16	4
01.10.16	5
05.10.16	5
08.10.16	5
Gesamt	**19**

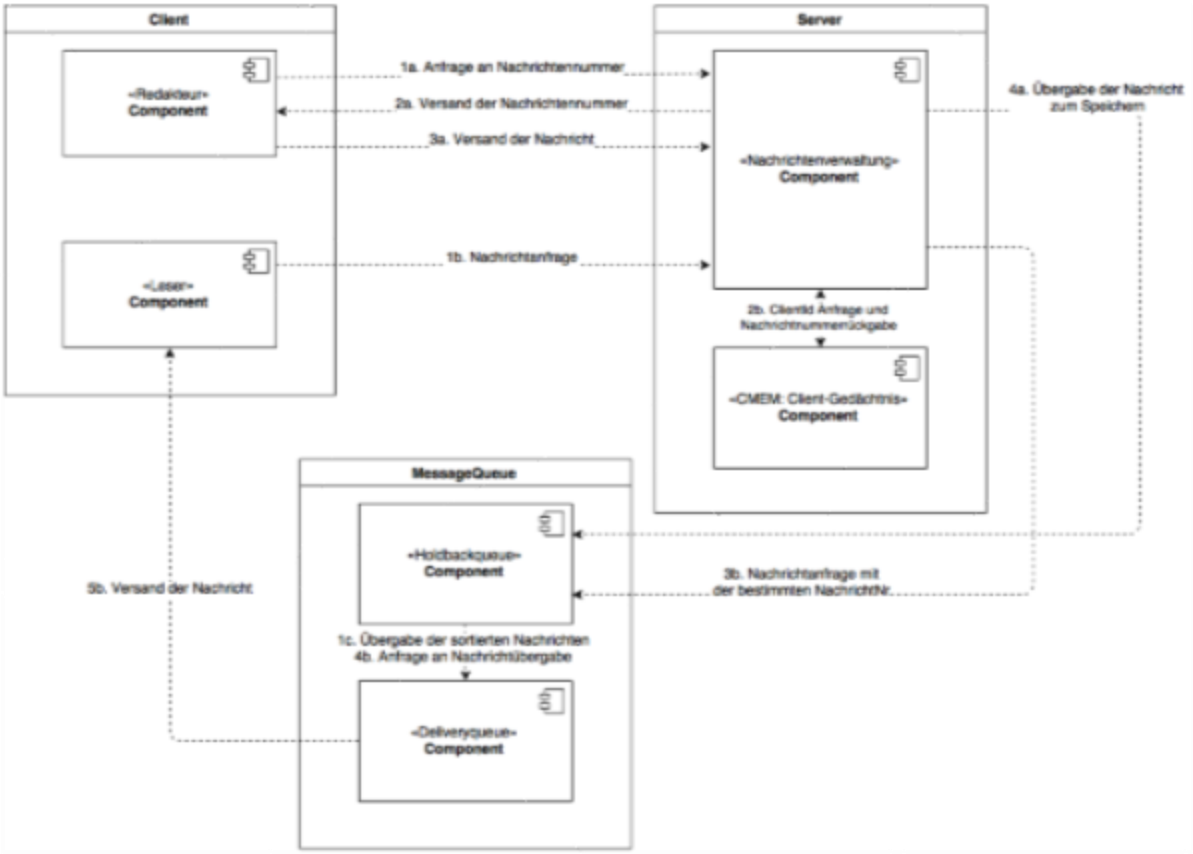
Aktueller Stand

Der Entwurf ist fertig. Die Implementation wird eingesetzt.

Änderungen im Entwurf

Noch keine.

Details zum Entwurf:



Es soll eine Client-Server Anwendung entwickelt werden, bei der ein Server die Nachrichten verwaltet, die ihm von unterschiedlichen Clienten (als Redakteur) zugesendet werden. Die Nachrichten werden fortlaufend nummeriert und in zufälligen Abständen von unterschiedlichen Clienten (als Leser) abgefragt. Dabei soll sich der Server an die Leser eine vordefinierte Zeit lang erinnern und ihnen nur noch ungelesene neue Nachrichten schicken, die sie noch nicht erhalten haben (1 Nachricht pro Anfrage). Wenn ein Client sich einige Zeit nicht mehr meldet, vergisst der Server ihn und sendet ihm beim erneuten Anfragen die 1. Nachricht (seiner Queue) – er behandelt ihn also wie einen neuen Clienten. Darüber hinaus achtet der Server auch darauf, dass die Nachrichten chronologisch sortiert werden. Wenn eine Nachricht durch Kommunikationsprobleme verloren geht, muss diese Situation korrekt behandelt werden, d.h. eine Fehlernachricht muss eine (oder mehrere) solcher verschwundenen Nachrichten ersetzen (dadurch können Lücken entstehen – die Nachrichtennummern sind dann nicht mehr zusammenhängend).

Komponentenbeschreibung

1. Client

wird weiter in Leser und Redakteur aufgeteilt. Beide Rollen dürfen nur als ein gesamter Prozess dargestellt werden, sollen aber unabhängig voneinander und laufen (nicht gleichzeitig).

Die Komponente Client besteht aus dem Modul „`client.erl`“, worin sowohl der Redakteur- als auch der Leser-Client implementiert wird.

Dazu müssen bestimmte Startparameter in einer Config Datei `client.cfg` definiert werden:

“`clients`” – die Anzahl der Clienten, die beim Start gestartet werden;

“`lifetime`” – Die Lebenszeit eines Clients in Sekunden;

“`sendeintervall`” – initiale Zeit, die zwischen den Nachrichten gewartet wird;

“`servername`” und “`servernode`” – dem Namen entsprechend.

1.1 Leser

Der Leser fragt den Server an, ob es neue, von ihm ungelesen Nachrichten für ihn gibt und schreibt diese in eine Log-Datei. Solange es noch neue Nachrichten gibt (wird mit Hilfe eines Flags mit jeder Nachricht zusammen übermittelt), fragt der Client weiter nach; sonst wird der Leser terminiert und wechselt zur Rolle des Redakteurs. Zudem wird dem Leser explizit die Nummer jeder Nachricht übermittelt.

1.2 Redakteur

Der Redakteur fragt eine eindeutige Nachrichtennummer vom Server ab und sendet ihm (anschließend oder später) eine mit dieser Nummer markierten Nachricht an ihn zurück. Nach fünf solchen Operationen holt sich der Redakteur die letzte Nummer, aber gibt darauf keine Antwort mehr (zu Test-Zwecken - um zu prüfen, ob mit fehlenden Nachrichten richtig umgegangen wird).

Nach dem Empfang der letzten Nummer wird das Warteintervall zwischen dem Versand der Nachrichten zufällig geändert (um ca. 50% per Zufall vergrößert oder verkleinert, aber darf nicht kürzer als 2 Sekunden sein) und zum Leser gewechselt.

2. Server

Der Server ist für die Nachrichtenvermittlung zuständig. Zu seinen wichtigen Komponenten gehören ein Queueverwaltungsdienst (verantwortlich für Empfang der Nachrichten von Redakteuren und für Versand an Lesern) und ein CMEM-Dienst (verantwortlich für zeitlich begrenzte Erinnerung an aktiven Clienten und der Nummer der zuletzt erhaltenen Nachricht des jeweiligen Clienten). Der Queueverwaltungsdienst ist ein eigenständiges Programm, welches auf einem anderen Server als der Nachrichtenvermittlungsdienst ausgeführt werden kann.

Der Server nimmt die Nachrichten von Redakteuren an, fügt ihnen jeweils die Empfangszeit an und schiebt sie zunächst in die `Holdbackqueue`. Danach werden sie vor der Übergabe an die `Deliveryqueue` den Nummern nach sortiert.

Wenn ein Leser nach neuen Nachrichten fragt, holt sich der Server die Nummer der nächsten Nachricht, die dieser Client erhalten soll von CMEM ab und übergibt sie (zusammen mit dem Client-Objekt) an die `Holdbackqueue`. Die `Holdbackqueue` vergleicht die Nummer ihrer ersten Nachricht mit der höchsten Nummer der `Deliveryqueue`. Wenn es keine Lücke zwischen diesen Nummern gibt, wird die `Deliveryqueue` aufgefüllt (hier wird die Übertragungszeit jeder Nachricht angefügt – wobei nur zusammenhängende Nachrichtennummern zur DQ übertragen werden).

Falls Nachrichten aus irgendwelchen Gründen nicht angekommen sind, wird die Lücke zwischen `Deliveryqueue` und `Holdbackqueue` mit genau einer Fehlernachricht abgeschlossen. Die Fehlernachricht bekommt die Nummer, die um 1 niedriger ist als die niedrigste der HBQ. Nach solchem Abschluss darf keine Nachricht mit einer niedrigeren Nummer (im Vergleich zu der Nachricht, die eine Lücke ersetzt) vom Server angenommen werden. Es wird solange darauf gewartet, bis sich die Lücke sich von alleine schließt oder bis die `Holdbackqueue` mehr als 2/3 der möglichen Länge der `Deliveryqueue` erreicht hat. Im Fall der Längenüberschreitung wird zuerst die Lücke geschlossen und danach die `Deliveryqueue` aufgefüllt. Die „Notfall-Lückenschließung“ soll nur bei einer zu vollen HBQ erfolgen (nicht bei der Client-Nachrichten-Abfrage).

Darüber hinaus merkt sich der Server zu jedem einzelnen Client (Leser) eine vorkonfigurierte Zeit lang dessen zuletzt erhaltene Nachricht, damit er nur die neuen (noch nicht abgeschickten) Nachrichten nach Anfrage von demselben Client sendet. Wenn der Leser keine Abfrage mehr gemacht hat, wird er vom Server nach vorkonfigurierter Zeit vergessen. Fragt ein „vergessener" Client anschließend wieder nach einer Nachricht, kann es passieren, dass er eine Nachricht bekommt, die er schon gelesen hatte. Er bekommt dann nämlich die Nachricht mit der niedrigsten Nummer der DQ.

Wenn die vorkonfigurierte Lebenszeit des Servers überschritten ist (d.h. keine Clientanfragen mehr angekommen), terminiert sich der Server. Die „Inaktiv-Zeit" des Servers wird mit jeder Anfrage wieder auf 0 gesetzt.

Die Server-Komponente besteht aus den zwei Modulen „`server.erl`" und „`cmem.erl`". Das Modul „`cmem.erl`" ist speziell für das Behalten der bei ihm registrierten Clients gedacht.

Dazu müssen bestimmte Startparameter in einer Config Datei `server.cfg` definiert werden:

"`latency`" – Die Lebenszeit (maximale Inaktiv-Zeit) des Servers in Sekunden;

"`clientlifetime`" – wie lange darf sich der Client nicht melden lassen, bevor ihn der Server vergisst und somit seine zuletzt erhaltene Nachricht nicht mehr kennt;

"`servername`" – der Name des Servers;

"`hbqname`" und "`hbqnode`" – Parameter zur Holdbackqueue ADT: Name und Node;

"`dlqlimit`" – Die maximale Anzahl an Nachrichten in der Deliveryqueue.

3. Eigene ADTs für Nachrichtenqueues

Die Komponente Queue besteht aus den zwei Modulen. Die erste „`hbq.erl`" (Holdbackqueue) ist für die Speicherung der ankommenden Nachrichten zuständig und das Modul „`dlq.erl`" verwaltet die auszuliefernden Nachrichten, die nach Anfrage an Clients gesendet werden.

3.1 Holdbackqueue

Die Holdbackqueue enthält die Nachrichten, die von den Redakteur-Clienten abgeschickt werden.

Die Holdbackqueue ist durch eine Liste von zweistelligen Tupeln repräsentiert, wobei das erste Element die Nachricht und das zweite Element die Nummer der Nachricht enthält.

3.2 Deliveryqueue

Die Deliveryqueue enthält die Nachrichten, die an die Leser-Clienten nach Anfrage gesendet werden. Die Nachrichten sind immer der Nummer nach aufsteigend geordnet.

Die Deliveryqueue ist durch eine Liste von zweistelligen Tupeln repräsentiert, wobei das erste Element die Nachricht und das zweite Element die Nummer der Nachricht enthält.

4. Werkzeug-Komponente

Diese Komponente stellt nützliche Funktionen für das Loggen und Auslesen der Konfigurationsdatei bereit, die jeweils von Clienten und Server verwendet werden.

Die Komponente besteht aus dem Erlang Modul "`werkzeug.erl`".

Schnittstellen

1. Entitäten

- `Nr` (Nummer): Integer Wert.
- `Msg` (Nachricht): String, in dem die eigentliche Nachricht enthalten ist
- `TSclientout`: Zeitstempel, wann die Nachricht den Redakteur-Client verlässt.
- `TShbqin`: Zeitstempel, wann die Nachricht die Holdbackqueue erreicht.
- `TSdlqin`: Zeitstempel, wann die Nachricht die Deliveryqueue erreicht.
- `TSdlqout`: Zeitstempel, wann die Nachricht die Deliveryqueue verlässt (an Client gesendet wird).
- `TSclientin`: Zeitstempel, wann die Nachricht den Leser-Client erreicht.
- `ToClient`: PID des jeweiligen Clients.

2. Server

```
/* Abfragen aller Nachrichten */
Server ! {self(), getmessages}
receive
{reply,[Nr,Msg,TSclientout,TShbqin,TSdlqin,TSdlqout],Terminated}

/* Senden einer Nachricht */
Server ! {dropmessage, [Nr, Msg, TSclientout]}
```

```
/* Abfragen der eindeutigen Nachrichtennummer */
Server ! {self(), getmsgid}
receive {nid,Nr}
```

getmessages

Fragt beim Server eine aktuelle Textzeile ab. `self()` stellt die Rückrufadresse des Leser-Clients dar. Als **Rückgabewert** erhält er eine für ihn aktuelle Textzeile (Zeichenkette) zugestellt (`Msg`) und deren eindeutige Nummer (`Nr`). Zudem erhält er die Zeitstempel explizit (erstellt durch `erlang:now()`, `TSclientout`, `TShbqin`, `TSdlqin`, `TSdlqout`). Mit der Variablen `Terminated` signalisiert der Server, ob es noch weitere Nachrichten gibt. `Terminated == false` bedeutet, es gibt noch weitere Nachrichten, `Terminated == true` bedeutet, dass es keine Nachrichten mehr gibt, d.h. weitere Aufrufe von `getmessages` sind nicht notwendig.

dropmessage

Sendet dem Server **eine Textzeile** (`Msg`), die den **Namen** des aufrufenden Clients und seine aktuelle **Systemzeit** sowie ggf. irgendeinen Text beinhaltet, zudem die zugeordnete (globale) Nummer der Textzeile (`Nr`) und seine Sendezeit (erstellt mit `erlang:now()`, `TSclientout`).

getmsgid

Fragt beim Server die aktuelle Nachrichtenummer ab. `self()` stellt die Rückrufadresse des Redakteur-Clients dar. Als **Rückgabewert** erhält er die höchste freie und somit eindeutige Nachrichtennummer (`Nr`).

3. HBQ

```
/* Initialisieren der HBQ */
HBQ ! {self(), {request,initHBQ}}
receive {reply,ok}
```

```
/* Speichern einer Nachricht in der HBQ */
HBQ ! {self(), {request,pushHBQ,[Nr,Msg,TSclientout]}}
receive {reply,ok}
```

```
/* Abfrage einer Nachricht */
HBQ ! {self(), {request,deliverMSG,Nr,ToClient}}
receive {reply,SendNr}
```

```
/* Terminierung der HBQ */
HBQ ! {self(), getmsgid}
receive {reply,ok}
```

{request,initHBQ }

initialisiert die HBQ und die DLQ. Bei Erfolg wird ein ok gesendet.

{request,pushHBQ,[Nr,Msg,TSclientout]}

fügt eine Nachricht Msg (Textzeile) mit Nummer Nr und dem Sende-Zeitstempel TSclientout (mit erlang:now() erstellt) in die HBQ ein. Bei Erfolg wird ein ok gesendet.

{request,deliverMSG,Nr,ToClient}

beauftragt die HBQ über die DLQ die Nachricht mit der Nummer Nr (falls nicht verfügbar die nächst höhere Nachrichtennummer) an den Client ToClient (als PID) auszuliefern. Bei Erfolg wird die tatsächlich gesendete Nachrichtennummer SendNr gesendet.

{request,dellHBQ}

terminiert den Prozess der HBQ. Bei Erfolg wird ein ok gesendet.

4. DLQ

```
/* Initialisieren der DLQ */
initDLQ(Size,Datei)


/* Löschen der DLQ */
delDLQ(Queue)


/* Abfrage, welche Nachrichtennummer in der DLQ gespeichert
werden kann */
expectedNr(Queue)
delDLQ(Queue)


/* Speichern einer Nachricht in der DLQ */
push2DLQ([Nr,Msg,TSclientout,TShbqin],Queue,Datei)
```

```
/* Ausliefern einer Nachricht an einen Leser-Client */
deliverMSG(MSGNr,ClientPID,Queue,Datei)
```

Die DLQ darf nur von der HBQ aus angesprochen werden!

initDLQ(Size,Datei)

initialisiert die DLQ mit Kapazität Size. Bei Erfolg wird eine leere DLQ zurück geliefert. Datei kann für ein Logging genutzt werden.

delDLQ(Queue)

löscht die DLQ. Bei Erfolg wird ok zurück geliefert.

expectedNr(Queue)

liefert die Nachrichtennummer, die als nächstes in der DLQ gespeichert werden kann. Bei leerer DLQ ist dies 1.

push2DLQ([Nr,Msg,TSclientout,TShbqin],Queue,Datei)

speichert die Nachricht [Nr,Msg,TSclientout,TShbqin] in der DLQ Queue und fügt ihr einen Eingangszeitstempel an (einmal an die Nachricht Msg und als expliziten Zeitstempel **TSdlqin** mit erlang:now() an die Liste an. Bei Erfolg wird die modifizierte DLQ zurück geliefert. Datei kann für ein Logging genutzt werden.

deliverMSG(MSGNr,ClientPID,Queue,Datei)

sendet die Nachricht MSGNr an den Leser-Client ClientPID. Dabei wird ein Ausgangszeitstempel **TSdlqout** mit erlang:now() an das Ende der Nachrichtenliste angefügt. Sollte die Nachrichtennummer nicht mehr vorhanden sein, wird die nächst größere in der DLQ vorhandene Nachricht gesendet. Bei Erfolg wird die tatsächlich gesendete Nachrichtennummer zurück geliefert. Datei kann für ein Logging genutzt werden.

5. CMEM

```
/* Initialisieren des CMEM */
initCMEM(RemTime,Datei)


/* Löschen des CMEM */
delCMEM(CMEM)
```

```
/* Speichern/Aktualisieren eines Clients in dem CMEM */
updateClient(CMEM,ClientID,Nr,Datei)

/* Abfrage, welche Nachrichtennummer der Client als nächstes
erhalten darf */
getClientNNr(CMEM,ClientID)
```

Das CMEM darf nur vom Server aus angesprochen werden!

initCMEM(RemTime,Datei)

initialisiert den `CMEM`. `RemTime` gibt dabei die Zeit an, nach der die Clients vergessen werden. Bei Erfolg wird ein leeres `CMEM` zurück geliefert. `Datei` kann für ein Logging genutzt werden.

delCMEM(CMEM)

löscht den `CMEM`. Bei Erfolg wird `ok` zurück geliefert.

updateClient(CMEM,ClientID,Nr,Datei)

speichert bzw. aktualisiert im `CMEM` den Client `ClientID` und die an ihn gesendete Nachrichtenummer `Nr`. `Datei` kann für ein Logging genutzt werden.

getClientNNr(CMEM,ClientID)

gibt die als nächstes vom Client erwartete Nachrichtennummer des Clients `ClientID` aus `CMEM` zurück. Ist der Client unbekannt, wird 1 zurück gegeben.