**Team:** {19, ["Björn Eberhardt", "Dieter Pisarewski"]}

**Aufgabenaufteilung:**

1. Björn Eberhardt,  
   \*.erl \ werkzeug.erl
2. \*.erl \ werkzeug.erl

**Quellenangaben:**

* <http://www.erlang.org/documentation/doc-5.1/doc/getting_started/getting_started.html>
* Folien aus der Vorlesung
* Beispieldateien aus der Vorlesung

**Begründung für Codeübernahme:** Es wurde kein Quellcode aus anderen Projekten übernommen.

**Bearbeitungszeitraum:**

* 27.03.2013, 02:00 – 14:00
* 28.03.2013, 07:50 – 15:50
* 29.03.2013, 04:50 – 23:50
* 30.03.2013, 03:30 – alle Zeiten sind falsch!

**Aktueller Stand:**

Fertig:

* Client
  + sammelt die versendeten Nummern und erkennt eigene Nachrichten.
  + beendet sich nach Ablauf der Lebenszeit.
* Server
  + liefert auf Anfrage vollständige msgids.
  + liefert auf Anfrage alle Nachrichten aus der DeliveryQueue an den Client.

Ungetestet:

* Client
  + sendet 5 Nachrichten, fordert eine 6. Nummer an, pollt dann alle Nachrichten vom Server und verändert die Wartezeit.
* Server
  + fügt auf Anfrage Nachrichten der HoldbackQueue hinzu.

Nicht implementiert:

* Client
  + Die Wartezeit wird nur alle 5 Nachrichten verändert und der neue Wert gespeichert.
* Server
  + Nachrichten, die an den Client gesendet wurden, werden nicht erneut gesendet.
  + Die Nachrichten werden nach dem Eintragen einer Nachricht in der HoldbackQueue und dem Kopieren in die DeliveryQueue aus der HoldBackQueue gelöscht.
  + Die Größe der HoldBackQueue wird überprüft. Außerdem wird bei einer zu großen HoldBackQueue eine Fehlermeldung erstellt.
  + Clients, die sich zu selten melden, werden vom Server „vergessen“.
  + Fehlermeldung, wenn HoldbackQueue überfüllt.

**Änderungen im Entwurf:** %

**Entwurf:** %

# Client

Der Client versendet Nachrichten an den Server und holt sich die neuesten Nachrichten wieder ab.

## Architektur

* Der Client wird logisch in den Redakteur- und in den Leseclient unterteilt.
* Die beiden logischen Clients wechseln sich sequentiell in der Ausführung ab.
* Mit kurzen Zeitabständen wird eine fortlaufende Nummer vom Server angefragt und eine entsprechende Nachricht versendet.
* Die Nachricht enthält den Rechnernamen, die Praktikumsgruppe, Teamnummer sowie eine Systemzeit und ggf. andere Informationen.
* Die Zeitabstände werden nach 5 versandten Nachrichten angepasst; zufällig um 50% vergrößert oder verkleinert, mindestens jedoch um 1 Sekunde.
* Die Zeitabstände dürfen nicht kleiner als 1 Sekunde sein.
* Eine 6. Nachrichtennummer wird angefordert und im Log protokolliert, ohne dass hierfür eine Nachricht versendet wird.
* Der Client merkt sich die für den Versand verwendete Nachrichtennummern.
* Der Leseclient empfängt nun alle neuen Nachrichten vom Server, bis dieser keine weitere Nachricht mehr ankündigt.
* Eigene Nachrichten werden zusätzlich mit einer Zeichenfolge (\*\*\*\*\*\*\*) gekennzeichnet.
* Die Nachrichten werden bei der Ausgabe zusätzlich mit einem Zeitstempel versehen.
* Nach Ablauf einer eingestellten Lebenszeit wird der Client beendet.

## Realisierung

Der Client soll sequentiell Aufgebaut werden d.h. das Senden von Nachrichten und das Abfragen von Nachrichten sollen nacheinander erfolgen.

Es sollen einzelne Punkte aus der Aufgabenstellung nacheinander umgesetzt werden. Der Redakteurclient und der Leseclient sollen nicht als eigene Prozesse sondern als Funktionen umgesetzt werden. Einzig die Funktion zum Sammeln von Nachrichtennummern, um diese später dem eigenem Redakteur zuordnen zu können soll als eigener Prozess erstellt und von dem Clientprozess benutzt werden.

Näher betrachtet soll der Client folgendes leisten

* Gestartet soll der Client mit den Parametern: Server und der Nummer des Clienten. Falls mehrere Clienten gestartet werden sollen soll das evtl. mit dem Shellskript realisiert werden.
* Es sollen die Konfigurationsdaten aus einer Datei gelesen werden. Diese Datei sollte:
  + das Verzeichnis mit dem Namen der Logdatei, die Lebenszeit des Clienten(im ms. oder s.),
  + den Sendeintervall der angibt wie viel Zeit zwischen dem Versenden der Nachrichten vergehen soll und
  + die Anzahl der Nachrichten die am Stück gesendet werden sollen beinhalten.
* Das Nachrichtensammelprozess soll mit Hilfe von zwei Listen die gesendete und die empfangene Nachrichtennummern verwalten und auf Anfrage prüfen, ob die empfangene Nachricht von eigenem Redakteur erstellt wurde.
* Es sollen mindestens Funktionen zum Senden und zum Empfangen von Nachrichten geben und evtl. noch weitere Hilfsfunktionen.

# Server

Der Server verwaltet die Nachrichten, die vom Redakteurclient gesendet werden und liefert diese in richtiger Reihenfolge den Leseclients aus.

## Architektur

* Der Server liefert dem Redakteurclient auf Anfrage eine fortlaufende Nummer für die Nachricht.
* Die dem Server zugesendeten Nachrichten werden über eine HoldbackQueue und eine DeliveryQueue verwaltet, um die Reihenfolge einzuhalten.
* Der Server protokolliert am Ende der Nachricht den Eingangszeitstempel beim Eintritt in die Holdback- und DeliveryQueue.
* Die Größe der DeliveryQueue ist beschränkt, und alte Nachrichten werden bei Überschreitung der Größe entfernt.
* Die HoldbackQueue wird benötigt, sobald die Nachrichten nicht in Reihenfolge eintreffen und eine Lücke entsteht.
* Besteht zwischen der ersten Nachricht der HoldbackQueue und der letzten Nachricht der DeliveryQueue keine Lücke, wird die Nachricht aus der HoldbackQueue in die DeliveryQueue geschoben. Dies erfolgt, solange die Liste nicht leer ist und keine Lücke entsteht.
* Überschreitet die Größe der HoldbackQueue die Hälfte der Maximalgröße für die DeliveryQueue, besteht eine Lücke. Die wird durch eine Fehlernachricht, die direkt in die DeliveryQueue geschoben wird und die Nummer der ersten Nachricht der HoldbackQueue minus 1 trägt, geschlossen und die HoldbackQueue wird weiter abgearbeitet.
* Erfolgt in einer vorgegebenen Zeit keine weitere Nachrichtenabfrage von einem Client, wird der Server terminiert.

## Realisierung

Wir haben unseren Server in folgende Einheiten aufgeteilt: **QueueManager**, **ClientManager** und **Sender** sowie 2 **Queues** zum Speichern der Nachrichten.

Die zentrale Einheit (der Server selber) übernimmt die Kommunikation mit den Clients und die Verarbeitung der drei in der Schnittstelle beschriebenen Nachrichten.

* Auf getmsgid wird eine fortlaufende Zahl zurückgeliefert. Dies erledigt der Server selbst, ohne auf die anderen Einheiten zuzugreifen.
* Auf getmessages wird die älteste ungelesene Nachricht in der DeliveryQueue an den Client gesendet. Die Ausführung dieses Vorgangs wird vom **Sender** übernommen.
* Auf dropmessage wird die Nachricht in der HoldbackQueue gespeichert. Anschließend wird die HoldbackQueue und die DeliveryQueue verwaltet. Diese Aufgabe wird vom **QueueManager** übernommen.

**QueueManager** ist verantwortlich für das Verwalten der eingehenden Nachrichten in die HoldbackQueue und Übertragung dieser Nachrichten in die DeliveryQueue.

Jede erhaltene Nachricht kommt zuerst in die HoldbackQueue. Nach dem Speichern der Nachricht in die HoldbackQueue wird zunächst die auf die neueste Nachricht aus der DeliveryQueue folgende Nachricht gesucht. In diesem Fall wird diese von der HoldbackQueue in die DeliveryQueue geschoben. Ist die DeliveryQueue nun größer als die festgelegte Maximalgröße, wird außerdem die älteste Nachricht gelöscht. Diese Suche wird wiederholt, bis die HoldbackQueue leer ist, oder eine Lücke gefunden wird.

Anschließend prüft der QueueManager die Größe der HoldbackQueue. Überschreitet diese die Hälfte der Maximalgröße der DeliveryQueue, geht der QueueManager davon aus, dass in der HoldbackQueue eine Lücke entstanden ist. Dann füllt er diese Lücke mit einer Fehlernachricht und schiebt sie in die DeliveryQueue mit der größten Nachrichtennummer in der Lücke.

**Sender** dient dazu, Nachrichten aus der DeliveryQueue an den Lese-Client zu senden und die Informationen (letzte gesendete Nachrichtennummer, letzte Zugriffszeit) über den Lese-Client zu aktualisieren.

Er fragt die letzte Nachrichtennummer, die dem Client gesendet wurde, beim ClientManager ab.

Dabei überprüft der Sender, ob der Client dort schon existiert. In diesem Fall prüft er den gespeicherten Zeitstempel und setzt die letzte Nachrichtennummer des Clients zurück, wenn die vergangene Zeit vom Zeitstempel bis zur aktuellen Zeit ein in Konfiguration vorgegebenes Zeitintervall überschreitet. Danach wird der Zeitstempel des Clients aktualisiert und die Nachrichtennummer auf die kleinste Nachrichtennummer in der DeliveryQueue gesetzt.

Dann holt er sich die Nachrichten aus der DeliveryQueue, wählt die benötigte Nachricht aus und sendet sie an den Client.

**ClientManager** verwaltet eine Liste von registrierten Lese-Clients mit zugehörigen Informationen (ClientPid, letzte gesendete Nachrichtennummer, letzte Zugriffszeit).

Ein Client landet in dieser Liste, wenn er Nachrichten vom Server anfordert. Bevor er dort gespeichert wird, prüft der Sender, der die oben genannte Aktion initiiert, ob der Client dort schon existiert.

**HoldbackQueue** und **DeliveryQueue** haben eine gemeinsame generische Implementierung. Wir realisieren die Queues mit einfachen Lists, da sie für die Speicherung von geordneten Mengen ausreichen, und diese lassen sich einfach mit Funktionen aus der Standard Library verwalten. Sie unterstützen folgende Aktionen:

* Einfügen einer Nachricht ans Ende der Queue
* Ankleben von mehreren Nachrichten ans Ende der Queue
* Holen aller Nachrichten aus der Queue als eine Liste