**VSP-1 – Anforderungen**

Zusatz von Michael // Zu klärende Fragen // Nochmal drüber schauen.

**Organisatorisches:**

* Praktikumsgruppe: 2, Teamnummer: 3
* 13.04.2017, bis 20 Uhr Entwurf an Klauck UND Schulz UND Teamkollegen senden.
* 18.04.2017, bis 20 Uhr finaler Stand als Zip an Klauck UND Schulz UND Teamkollegen senden.
* Klauck VS Page: users.informatik.haw-hamburg.de/~klauck/verteiltesysteme.html

**Server:**

1. Der Server verwaltet ID’s (beginnend bei 1), welche Textzeilen eindeutig zugeordnet werden können. Nach Außen bietet der Server eine Schnittstelle an welche die aktuelle ID liefert, welche von Redakteuren abgerufen werden kann.
2. Die dem Server zugestellten Textzeilen sollen bzgl. der Nummerierung in zusammenhängender Reihenfolge erscheinen. Da Nachrichten durchaus verloren gehen können oder in falscher Reihenfolge beim Server ankommen können, arbeitet der Server intern mit einer Holdback- und einer Deliveryqueue (Abk.: HBQ & DLQ). Eine Nachricht ist entweder in der DLQ oder HBQ!
3. In der **DLQ** stehen Nachrichten, welche an Leser ausgeliefert werden können. In der DLQ können nur eine bestimmte Anzahl an Textzeilen stehen, welche durch die Größe der DLQ festgelegt wird.
4. In der **HBQ** stehen alle Nachrichten, welche noch nicht ausgeliefert werden dürfen (noch nicht vollständig, falsche Reihenfolge, etc. ?).
5. **Bei Empfang einer neuen Nachricht** wird die Empfangszeit beim Eintrag in die HBQ und die Übertragungszeit beim Eintrag in die DLQ der Zeichenkette der neuen Nachricht hinzugefügt. Die Zeiten können anhand von **wergzeug:timeMilliSecond()** bestimmt werden.  
   Zusätzlich werden „dieser Nachrichten-Liste“ (@Aufgabenstellung Verwirrung) diese Zeitstempel mittels **erlang:now()** am Ende hinzugefügt.
6. Der Server bietet nach Außen eine Schnittstelle, an welcher Leser neue Nachrichten anfordern können und gemäß der Nummerierung eine Nachricht zugesandt bekommen, welche noch nicht an den Leser ausgeliefert wurde. In einem Flag wird dem Leser mitgeteilt, ob es noch weitere für ihn unbekannte Nachrichten gibt und die ID der nächsten Nachricht geliefert.  
   Falls keine weiteren Nachrichten vorhanden sind, oder noch keine Nachrichten beim Server eingegangen sind, dann wird eine (nicht-leere) Dummy-Nachricht an den Leser versendet.  
   (Wie kann die genau aussehen und wie wird damit umgegangen?) Keine Folgenachricht heißt aktuell keine oder insgesamt keine mehr?
7. Der Server verwaltet Client-Verbindungen in einem ADT (**CMEM**). Wenn ein Client eine bestimmte Zeit keine Anfragen mehr gesendet hat, so wird er vergessen und bei der nächsten Anfrage wie ein unbekannter Client behandelt.
8. Wenn in der HBQ mehr als 2/3el an Nachrichten enthalten sind, als durch die vorgegebene maximale Anzahl an Nachrichten in der DLQ stehen können, dann wird, sofern eine Lücke besteht, diese Lücke zwischen HBQ und DLQ mit genau einer Fehlernachricht geschlossen.  
   (Wie genau kann das funktionieren?)  
   etwa: \*\*\*Fehlernachricht fuer Nachrichtennummern 11 bis 17 um 16.05 18:01:30,580|. Es werden keine weiteren Lücken innerhalb der Holdbackqueue behandelt! Die Fehlernachricht soll nur anhand einer entsprechenden Zeichenkette innerhalb der verschickten Nachricht zu erkennen sein und sich sonst nicht von gewöhnlichen Nachrichten unterscheiden.
9. Der Server terminiert, wenn sein Timer ausläuft, welcher durch Client-Anfragen jeweils zurückgesetzt wird. Wie lang ist der Timer?
10. Der Server verwendet drei **ADTs**: **HBQ** (Datei hbq.erl), **DLQ** (Datei dlq.erl) und **CMEM** (Datei cmem.erl) als Gedächtnis für die Leser-Clients. Diese dürfen hauptsächlich nur als Erlang-Liste ([ ]) realisiert werden! Als Hilfsstrukturen dürfen Tupel eingesetzt werden. Dazu sind die weiter unten aufgeführten Vorgaben zu beachten!
11. Die HBQ ist als entfernte ADT zu implementieren. Ihre Schnittstelle ist daher durch Nachrichtenformate beschrieben. Die DLQ oder das CMEM sind als lokale ADT zu implementieren. Daher sind deren Schnittstellen durch Funktionen beschrieben. Intern kann die DLQ bzw. das CMEM jedoch als externer Prozess realisiert werden!
12. Die steuernden Werte sind in einer Datei server.cfg anzugeben. Der Server ist unter einem **Namen** <name> im lokalen Namensdienst von Erlang zu registrieren (register(<name>,ServerPid)).

**Client (Redakteur):**

1. Vor dem Versenden einer Textzeile an den Server ruft ein Redakteur die neuste ID ab und stellt diese seiner Textzeile voran.
2. Ein Redakteur schickt in einem festgelegten Zeitintervall Textzeilen an den Server, welche seinen Rechnernamen, die Praktikumsgruppe, die Teamnummer und die aktuelle Systemzeit enthalten. Optional können weitere Informationen in den Nachrichten stehen.
3. Nachdem der Redakteur eine eindeutige ID für seine Nachricht angefordert hat, wartet er also so lange, bis das Zeitintervall abgelaufen ist und versendet anschließend seine Nachricht
4. Das Intervall wird nach dem Versenden von 5 Textzeilen jeweils um ca 50% per Zufall vergrößert oder verkleinert, darf hierbei aber nicht unter 2 Sekunden fallen.
5. Nach dem Versenden von 5 Textzeilen, vergisst der Redakteur nach Abruf einer ID die zugehörige Nachricht zu verschicken und vermerkt dies in seinem Log. (Nummer, Zeit, vergessen zu senden)
6. **Die steuernden Werte sind in einer Datei client.cfg anzugeben. Lediglich die aktuelle node des Servers kann als Parameter übergeben werden, d.h. beim Starten des Clients ist maximal ein Parameter erlaubt! Der Client darf (neben dem Prozess zum loggen) maximal aus einem Prozess bestehen.**

**Client (Leser):**

1. Ein Leser-Client kann dem Server Anfragen senden, woraufhin dieser eine gemäß der Nummerierung noch nicht an den Leser ausgelieferte Nachricht versendet.
2. **Ein Leser nimmt seine Rolle erst ein, nachdem er Redakteur war.** So ähnlich, der Client ist entweder Leser oder Redakteur doch wechselt ständig hin und her.
3. Ein Leser fragt solange aktuelle Textzeilen ab, bis er alle erhalten hat und gibt sie dann in seiner GUI aus. Dabei erhält er immer genau eine ihm unbekannte Textzeile pro Nachricht.
4. Ein Leser merkt sich die Nachrichtennummern der erhaltenen Nachrichten und fügt den Nachrichten die durch seinen Redakteur-Client erstellt wurden die Zeichenfolge \*\*\*\*\*\*\* an.  
   (Ist damit eine Variable Zeichenkette gemeint, welche wir festlegen können?)
5. Der Leser wertet die mitgelieferten Zeitstempel aus. Wenn eine Nachricht aus der Zukunft stammt, dann wird diese mit der entsprechenden Zeitdifferenz am Ende der Zeichenkette markiert.
6. Bei der Initialisierung des Clients wird eine Lebenszeit definiert, nach Ablauf welcher er sich selbst terminiert.
7. **Die steuernden Werte sind in einer Datei client.cfg anzugeben. Lediglich die aktuelle node des Servers kann als Parameter übergeben werden, d.h. beim Starten des Clients ist maximal ein Parameter erlaubt! Der Client darf (neben dem Prozess zum loggen) maximal aus einem Prozess bestehen.**

**Milestones:**

1. (Lokal & Verteilt) Einfacher Client / Server mit cfg Dateien die sich anpingen können.
2. Server kann an Client Beispiel Nachricht schicken (schon mit Id & Flag & richtigem Handeln).
3. Server kann Zeilen Ids verschicken.
4. Client kann Ids empfangen und Nachricht schreiben die er zurückschickt.
5. Server kann Zeilen annehmen.
6. Client verliert nach 5 Zeilen die aktuelle Zeile.
7. Server kann geschriebene Zeilen an Client verschicken.
8. Server weiß welche Zeilen er wem geschickt hat und reagiert dementsprechend.
9. Server hält Zeilen in HBQ und DLQ.
10. Alles läuft.

**Frage:**

* Gestrichelte Linie von HBQ bzw. DLQ zu Server, was wird dort übergeben, Rückgabewert und was konkret?
* Gestrichelte Linie heißt ja Lokal also nur Funktionsaufrufe, dennoch Beschreibung der Schnittstelle mit „PID ! …“, also kein Funktionsaufruf, was nun?

% ------------------------ P1 Erklärungen

% Datenstruktur soll Daten zurückhalten die User nicht sehen darf.

% In der HOldbackQ (HBQ), in DeliveryQ (DLQ) alle Daten die User sehen darf.

% Gefahr von lückenhafter übertragun, Fehlernachricht generieren, gehen davon aus, dass es nicht mehr kommt.

% System erkennt solche Lücken, neue Nachricht erstellen mit erster Numme der fehlenden Range. DLQ enthält diese neue Nachricht mit besagter Nummer.

% Lücken nicht zu schnell schließen. Wenn die fehlenden Nachricht doch noch kommen, egal. Größte was in DLQ (= shcon ausgeliefert) immer im Vergleich zur Übertraung von HBQ zu DLQ, kleinere Nummern als diese Größte werden vergessen, sind uninteressant.

% von: HBQ 60,61,70,71 // DLQ 40,39 -> Lücke zwischen einschließlich 41 und 59.

% zu: HBQ 70,71 // DLQ 61,60,59,40,39 -> Lücke wird geschlossen in dem Fehlernachricht mit 59 erstellt wird. Wäre sie 41 würde von 41 bis 59 je Fehlernachrichten erzeugt werden, also 9 statt eine! Also Lieber eine.

% 3 Prozesse, Client / Server / Queues (Entweder gerade Redakteur oder Leser aktiv, deswegen zusammen in einem Prozess)

% Cmem vergisst leser, wenn der sich wieder meldet dann wieder bei 1 starten.

% DLQ direkt zu Leser um effizienz zu steigern, architektonisch bedenklich.

% Zeitstempel (vom Server geschickt, beim Redakteur angekommen, vom Leser gelesen, .....)

% solang Termi(nierung) auf false, gibt es noch Nachrichten für Leser die er dann auch anfrägt. Gibt es für Leser noch keine neuen Nachrichten, Dumm Message mit Termi = true. Wenn True, Leser hört auf.

% Datei bei DQL für Logging.

% Checkliste machen.

% Ablauf: Server -> Redakteur (Nachrichtennummer vergegen)

% Redakteur -> Server (Msg)

% Server -> HBQ (Msg)

% HBQ -> DLQ (Msg)

% DLQ -> Server/CMEM (Nummer geschickt)

% DLQ -> Leser (Msg)

%start() ->{ok,ConfigListe} = file:consult("server.cfg"),

% {ok, LIfetime] = werkzeug.}