Aufgabe 1

In dieser und der folgenden Aufgabe wird die Open Source Sprache Erlang/OTP verwendet (Plugin für Eclipse, auch für andere Editoren gibt es eine Unterstützung, etwa Notepad++, KWrite, XEmacs). Gemäß der Forderung nach Programmiersprachen für eine massive nebenläufige Programmierung in "G. Bengel, C. Baun, M. Kunze, K.-U. Stucky: *Masterkurs Parallele und Verteilte Systeme*", wird damit eine in der Praxis entwickelte Programmiersprache vorgestellt, bei der Konstrukte zur nebenläufigen und verteilten Programmierung direkter Bestandteil der Sprache sind. Insbesondere eignet sich diese Sprache auch für die Programmentwicklung im Hinblick auf die Generationen von Prozessoren, bei denen nicht nur vier oder acht Kerne, sondern 32, 64 oder sogar hunderte Prozessorkerne auf einem Chip vorhanden sind. Massiv nebenläufige Programme können einen Geschwindigkeitsvorteil erlangen und das Konzept der leichtgewichtigen Prozesse in Erlang/OTP unterstützt diesen Ansatz direkt.

Teil 1: Vorbereitung

Mit dieser Aufgabe sollen Sie die Sprache Erlang/OTP kennen lernen. Daher wird der Code ausschließlich in Erlang/OTP entwickelt. Neben Online-Materialien (z.B. einem Tutorial) und Büchern können Sie auch auf die interne Bachelorarbeit von Herrn S. Meiser (*.pdf) zu greifen. Die Programmiersprache Erlang/OTP ist im Raum 0765 installiert und sollte auch im 11-ten Stock installiert sein. Eine Installation auf Ihrem eigenen Rechner ist aber auch möglich.

Teil 2: Message of the Day

Implementieren Sie eine Client/Server-Anwendung; ein Server verwaltet Nachrichten (Textzeilen): **die Nachrichten des Tages**, die von unterschiedlichen Clients (den Redakteuren) ihm zugesendet werden. Diese Nachrichten sind eindeutig nummeriert. Clients (die Lesern) fragen in bestimmten Abständen die aktuellen Nachrichten ab. Damit ein Leser-Client nicht immer alle Nachrichten erhält, erinnert sich der Server an ihn und welche Nachricht er ihm zuletzt zugestellt hat. Meldet sich dieser Lese-Client jedoch eine gewisse Zeit lang nicht, so vergisst der Server diesen Lese-Client.

Im Nachfolgenden nun die detaillierte Beschreibung. Dabei werden die beiden Rollen Redakteur und Leser in einem Client vereinigt:

Funktionalität

Server

1. Die **Textzeilen** werden vom Server durchnummeriert (beginnend bei 1) und stellen eine eindeutige ID für jede Textzeile dar. Ein Redakteur-Client hat sich beim Server vor dem Versenden einer Textzeile diese Nummer zu besorgen und in der Zustellung seiner Nachricht an den Server diese Nummer der Textzeile voranzustellen. Der Server merkt sich nicht, von wem eine Textzeile gesendet wurde, insbesondere schaut er nicht in die Textzeile hinein!
2. Da die dem Server zugestellten Textzeilen bzgl. der Nummerierung in zusammenhängender Reihenfolge erscheinen sollen und Nachrichten mit Textzeilen verloren gehen können bzw. in nicht sortierter Reihenfolge eintreffen können, arbeitet der Server intern mit einer **Deliveryqueue** und einer **Holdbackqueue**.

   In der **Deliveryqueue** stehen die Nachrichten, die an Clients ausgeliefert werden können, maximal *** viele Textzeilen. Dies wird durch die Größe der Deliveryqueue vorgegeben.

   In der **Holdbackqueue** stehen alle Textzeilen, die nicht ausgeliefert werden dürfen, d.h. die größte Nummer in der Deliveryqueue und die kleinste Nummer in der Holdbackqueue haben mindestens die Differenz von eins!

3. Der Server fügt einer empfangenen Nachricht jeweils die Empfangszeit beim Eintrag in die Holdbackqueue und die Übertragungszeit beim Eintrag in die Deliveryqueue hinten/rechts an.
4. Ein **Lese-Client** bekommt auf Anfrage gemäß Nachrichtennummerierung eine noch nicht an ihn ausgelieferte und beim Server bekannte Textzeile geliefert. In einem Flag wird ihm mitgeteilt, ob es noch weitere, für ihn unbekannte Nachrichten gibt. Zudem wird ihm explizit die Nummer dieser Nachricht übermittelt. Wenn der Lese-Client nach neuen Nachrichten beim Server anfragt, dort jedoch keine neuen bzw. überhaupt noch keine Nachrichten vorhanden sind, sendet der Server eine nicht leere dummy-Nachricht.
5. Ein **Lese-Client**, der seit ** Sekunden keine Abfrage mehr gemacht hat, wird beim Server vergessen (unabhängig davon, wann er die letzte Textzeile als Redakteur-Client übertragen hat!). Bei einer erneuten Abfrage (nach dem Vergessen) wird er wie ein unbekannter Lese-Client behandelt.
6. Wenn in der Holdbackqueue von der Anzahl her mehr als die Hälfte an echten Nachrichten enthalten sind, als durch die vorgegebene maximale Anzahl an Nachrichten in der Deliveryqueue stehen können, dann wird, sofern eine Lücke besteht, diese Lücke zwischen Deliveryqueue und Holdbackqueue mit **genau einer Fehlernachricht** geschlossen: `***Fehlernachricht fuer Nachrichtennummern 11 bis 17 um 16.05 18:01:30,580|.` Es werden keine weiteren Lücken innerhalb der Holdbackqueue gefüllt, also Lücken die nach der kleinsten in der Holdbackqueue vorhandenen Textzeilennummer vorkommen! In dem

Sinne wird die Holdbackqueue in diesem Fall nicht zwingend geleert, sondern nur bis zur nächsten Lücke geleert.

7. Der **Server terminiert** sich, wenn die Differenz von aktueller Systemzeit und Zeit der letzten Abfrage eines Clients länger als seine Wartezeit beträgt, d.h. seine Wartezeit wird durch Abfragen der Clients erneut gestartet bzw. bestimmt die maximale Zeit, die der Server ungenutzt läuft.
8. Der Server ist in Erlang/OTP zu **implementieren** und muss auf jedem Rechner im Labor **startbar** sein! Bei der Verwendung von Eclipse kann das problematisch sein. Die steuernden Werte sind in einer Datei server.cfg anzugeben. Der Server ist unter einem **Namen** `<name>` im Namensdienst von Erlang zu registrieren `global:register_name(<name>,ServerPid)`. Der Server darf (neben dem Prozess zum Loggen) maximal aus zwei Prozessen bestehen. Als Datenstrukturen für die unterschiedlichen Queues sowie die Liste der Clients sind **ausschließlich Listen** erlaubt. Eine Sortierte Liste (SL) ist vorgegeben und kann genutzt werden.

Client

9. Ein **Redakteur-Client** sendet in bestimmten Abständen, d.h. alle **** Sekunden, eine Textzeile an den Server. Diese **** Sekunden Wartezeit sind zwischen der Anforderung einer eindeutigen Nachrichtennummer beim Server und vor dem Senden der Nachricht an den Server vorzunehmen. Die Textzeile enthält seinen Namen, seinen Rechnernamen (z.B. lab18), die Prozessnummer, die Praktikumsgruppe (z.B. 1), die Teamnummer (z.B. 03) beinhalten) und seine aktuelle Systemzeit (die der Sendezeit entsprechen soll) beinhaltet und ggf. anderen Text, zum Beispiel `0-client@lab18-<0.1313.0>-C-1-03: 22te_Nachricht. Sendezeit: 16.05 18:01:30,769|(22).`
10. Der **Abstand** von **** Sekunden wird nach dem Senden von 5 Textzeilen jeweils um ca. 50% (mindestens 1 Sekunde) per Zufall vergrößert oder verkleinert. Die Zeit **** darf nicht unter 2 Sekunde rutschen. Das neue Sendeintervall ist im Log zu vermerken.
11. Der **Redakteur-Client** fragt nach dem Senden von 5 Textzeilen eine eindeutige Nachrichtennummer beim Server ab, ohne ihm eine zugehörige Nachricht zu übermitteln. In seinem log ist dies zu vermerken, etwa:`121te_Nachricht um 16.06 09:55:43,525| vergessen zu senden ******`
12. Der **Lese-Client** fragt nach dem Senden von 5 Textzeilen (als Redakteur-Client) und dem reinen Abfragen einer eindeutigen Nachrichtennummer (Punkt 11.) solange aktuelle Textzeilen beim Server ab, bis er alle erhalten hat, d.h. alle bisher noch nicht erhaltene und beim Server noch vorhandene Textzeilen, und stellt sie in seiner GUI dar. Dabei gelten die vom Redakteur-Client gesendeten Textzeilen als bei diesem Client unbekannte Textzeilen! Alle unbekannten Textzeilen werden ihm also einzeln übermittelt bzw. pro

Anfrage erhält er nur genau eine unbekannte Textzeile. Der Client merkt sich die vom Server erhaltenen Nachrichtennummern und fügt einer als Lese-Client erhaltenen Nachricht, die durch sein Redakteur-Client erstellt wurde, die Zeichenfolge `*******` an, etwa `6te_Nachricht. C Out: 11.11 21:12:58,720|(6); HBQ In: 11.11 21:12:58,720| DLQ In:11.11 21:13:01,880|.*******; C In: 11.11 21:13:07,190|`

13. Bei der **Initialisierung** des Clients (z.B. beim Aufruf) wird seine Lebenszeit gesetzt. Ist diese Zeit erreicht, terminiert sich der Client. Verwenden Sie dazu Erlang Timer.
14. Der **Client ist in Erlang/OTP** zu **implementieren** und muss auf jedem Rechner im Labor **startbar** sein. 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 (das Paar Lese und Redakteur Client) darf (neben dem Prozess zum loggen) maximal aus einem Prozess bestehen. Der Client-Starter ist als eigener Prozess zu realisieren, der die Lebensdauer aller von ihm abhängigen Clients kontrolliert und diese terminiert. Verwenden Sie `spawn_link`.

GUI

13. **Server-GUI**: Die Ausgaben sind alle in eine Datei `NServer.log` zu schreiben: siehe: `NServer@Brummpa.log`
14. **Client-GUI**: Alle Ausgaben sind in eine Datei `client_<Nummer><Clienthost>.log`, z.B. `Client_2client@Brummpa.log`), zu schreiben: siehe: `Client_2client@Brummpa.log`

Fehlerbehandlung

15. Das Loggen der Ausgaben sollten mögliche Fehler leicht auffindbar machen.

Schnittstelle

Im Folgenden wird die **Schnittstelle des Servers** für einen Client beschrieben. Da eine gemeinsame Vorführung stattfindet, ist sie **unbedingt einzuhalten**!

```
/* Abfragen aller Nachrichten */
Server ! {query_messages, self()},
receive {message, Number,Nachricht,Terminated} ->

/* Senden einer Nachricht */
Server ! {new_message, {Nachricht, Number}},

/* Abfragen der eindeutigen Nachrichtennummer */
Server ! {query_msgid, self()}
receive {msgid, Number} ->
```

`query_messages`: Fragt beim Server eine aktuelle Textzeile ab. `self()` stellt die Rückrufadresse des Lese-Clients dar. Als **Rückgabewert** erhält er eine für ihn aktuelle Textzeile zugestellt (`Nachrichteninhalt`) und deren eindeutige Nummer (`Number`). Mit der Variablen `Terminated` signalisiert der Server, ob noch für ihn aktuelle Nachrichten vorhanden sind. `Terminated == false` bedeutet, es gibt noch weitere aktuelle Nachrichten, `Terminated == true` bedeutet, dass es keine aktuellen Nachrichten mehr gibt, d.h. weiter Aufrufe von `query_messages` sind nicht notwendig.

`new_message`: Sendet dem Server **eine Textzeile** (`Nachricht`), die den **Namen** des aufrufenden Clients und seine aktuelle **Systemzeit** sowie ggf. irgendeinen Text beinhaltet (z.B.), zudem die zugeordnete (globale) Nummer der Textzeile (`Number`).

`query_msgid`: Fragt beim Server die aktuelle Nachrichtenummer ab. `self()` stellt die Rückrufadresse des Lese-Clients dar. Als **Rückgabewert** erhält er die aktuelle und eindeutige Nachrichtennummer (`Number`).

Tipp

Damit Sie die Betreuung z.B. zur Klärung von Fragen nutzen können und damit Sie die Befragung am Anfang des Praktikums erfolgreich absolvieren können, ist die Aufgabe gut vorzubereiten! (Siehe hierzu die PVL-Bedingung). In der Datei `werkzeug.erl` gibt es für den Zeitstempel und loggen nützliche Funktionen sowie eine Implementierung einer sortierten Liste (SL). Das Arbeiten in größeren Teams, ggf. der gesamten Praktikumsgruppe, ist ausdrücklich gewünscht! Bei der Befragung haben dennoch die **beiden Partner in einem Team** alle Fragen zu beantworten! Ein Verweis zur Beantwortung der Frage auf Nicht-Teammitglieder ist **nicht zulässig**!

Halten Sie das System einfach und strukturiert! Wenn die Kernfunktionalität erstellt wurde, kann das System immer noch beliebig erweitert/verbessert werden.

Das Praktikum wird von Kollegen Hartmut Schulz mitbetreut, der auf seinen WWW-Seiten ggf. weitere Informationen zu den Aufgaben hat!

Abnahme (Ausnahme Montagstermin)

Bis Sonntag Abend vor Ihrem Praktikumstermin ist ein erster Entwurf für die Aufgabe **als *.pdf** Dokument (Dokumentationskopf `PFormal.doc` nicht vergessen!) mir per E-Mail über **den Verteiler abgabe_aivsp@informatik.haw-hamburg.de** zuzusenden. Das Format des Betreffs ist wie folgt **VSP-A<NR>-Design-PG<NR>-Team<NR>** und unbedingt einzuhalten. Ggf. können offene Fragen im Entwurf gestellt werden. Geben Sie bitte **auch Ihren Teampartner/-in** an **(inklusive E-Mail**

Adresse im cc!). **Der Entwurf hat** denen Ihnen **bekannten SE-Richtlinien** zu genügen (ggf. nacharbeiten, wie SE-Richtlinien zu einem Entwurf aussehen). Der Entwurf ist für die PVL wichtig: sollten die Abgabefrist und/oder die erforderlichen Inhalte nicht eingehalten bzw. erbracht worden sein, gilt die Aufgabe als nicht erfolgreich bearbeitet! Mehr noch ist der Entwurf für Sie wichtig: die Erfahrung zeigt, dass jede hier nicht investierte Minute meist zu zwei Minuten und mehr zwischen Entwurfsabgabe und Aufgabenabgabe mutiert. Als erfolgreich wird ein Entwurf bewertet, wenn Ihre Kenntnisse bzgl. der gestellten Aufgabe eine erfolgreiche Teilnahme an dem Praktikumstermin in Aussicht stellen.

Am Tag des Praktikums findet nach dem Eingangstest eine Befragung von Teams statt. Die **Befragung muss erfolgreich absolviert werden**, um weiter am Praktikum teilnehmen zu können. Ist die Befragung nicht erfolgreich, gilt die Aufgabe als nicht erfolgreich bearbeitet. Als erfolgreich wird die Befragung bewertet, wenn Ihre Kenntnisse bzgl. der gestellten Aufgabe zumindest ausreichend sind. Dazu gehört insbesondere eine ausreichende Kenntnis über Ihren Code (und nicht die Kenntnis über den Kommentar im Code).

Ab ca. **16:00** findet eine gemeinsame Vorführung statt. Bei der Vorführung wird per Zufall ausgewählt, welcher Server eines Teams zum Einsatz kommt. Alle Teams (auch das Team, das den Server gestartet hat) haben ihre Clients zu starten. Im Vorfeld sollten Sie Ihre Software unbedingt testen! Ggf. kann ein Beispielserver dazu gestartet werden bzw. ein Beispielclient Ihren Server ansprechen. Die Vorführung wird als erfolgreich bewertet, wenn Ihr Programm erfolgreich an der Vorführung teilnehmen kann.

Abgabe: **Unmittelbar am Ende des Praktikums** ist von **allen Teams** der Code abzugeben. Zu dem Code gehören die Sourcedateien, die *.log Dateien, die während der Vorführung erzeugt wurden, und eine `Readme.txt` Datei, in der ausführlich beschrieben wird, wie das System zu starten ist! Bitte beachten Sie: es muss klar sein, welche *.log Dateien zu welcher Vorführung bzw. zu welchem Test gehören! Zudem sind für die einzelnen Vorführungen einzelne Ordner zu erstellen. Die Abgabe gehört zu den PVL-Bedingungen und ist einzuhalten, terminlich wie auch inhaltlich! Verteiler **abgabe_aivsp@informatik.haw-hamburg.de**. Format des Betreffs ist wie folgt **VSP-A<NR>-Code-PG<NR>-Team<NR>** und unbedingt einzuhalten.

Wird eine Aufgabe nicht erfolgreich bearbeitet gilt die **PVL** als **nicht bestanden**. Damit eine Aufgabe als erfolgreich gewertet wird, müssen der Entwurf, die Befragung, die Vorführung sowie die Abgabe als erfolgreich gewertet werden. **Alle gesetzten Termine sind einzuhalten**.

AUSNAHME AUFGABE 1 FUER DIE MONTAGSGRUPPE

Für die Montagsgruppe gelten die gleichen Abgabebedingungen für den Entwurf, wie für die anderen Gruppen am Mittwoch/Donnerstag.

Der Test findet zu Beginn des ersten Termins statt.