**Team**: 6, Mert Siginc, Michael Müller

**Aufgabenaufteilung**:

**Quellenangaben**:

* <http://erlang.org/doc/apps/stdlib/index.html>
* <http://users.informatik.haw-hamburg.de/~klauck/verteiltesysteme.html>

**Bearbeitungszeitraum**: <Datum und Dauer der Bearbeitung an der Aufgabe von allen Teammitgliedern>

**Aktueller Stand**:

Alles fertig implementiert und getestet.

Bestimmte Kombinationen (mit unserem und Professors Code) erstellt und keine Probleme mehr bekannt.

**Änderungen des Entwurfs**: <Vor dem Praktikum auszufüllen: Welche Änderungen sind bzgl. des Vorentwurfs vorgenommen worden.>

**Entwurf**:

**Allgemeine Funktion der Anwendung:**

Die Anwendung realisiert ein verteiltes System. In diesem sollen von beliebig vielen Prozessen, die jeweils ein #Mi-Wert haben, der ggT(größte gemeinsamer Teiler) berechnet werden. Für die Berechnung wird der Satz des Euklid verwendet.

Zur Realisierung dieser Aufgabe sollen die Prozesse vom Koordinator zufällig zu einem Ring aufgestellt werden, wobei jeder einzelne Prozess jeweils seinen linken und seinen rechten Nachbarn kennt. Dies ist wichtig, da der #Mi-Wert sich nach einer Berechnung verändern kann. Die Prozesse teilen dies ihren Nachbarn und dem Koordinator mit, so dass der ggT gefunden wird. Finden keine Berechnungen mehr statt, wird nach einem positiven Vote, alle Prozesse beendet.

**Starter:**

**Allgemein:**

Der Starter hat die Aufgabe die ggt-Prozesse zu starten. Um die Prozese mit den richtigen Konfigurationen zu starten, liest der Starter aus einer Config-Datei und fragt den Koordinator nach steuernden Werten. Diese werden dann beim Start eines Prozesses mitgegeben.

**Realisierung:**

Zum erhalten der steuernden Werte des Koordinators muss man die Schnittstelle des Koordinators ansprechen.

{From:,getsteeringval} -> {steeringval,ArbeitsZeit,TermZeit,Quota,GGTProzessnummer}

Zum lesen aus der Datei benötigt man IO-Methoden, die das lesen aus einer Datei ermöglichen. Hiernach werden beliebig viele ggt-Prozesse in einer Schleife erzeugt.

Dabei wird die Anzahl der Prozesse, durch die steuernden Werte vom Koordinator bestimmt.

Zudem benötigt jeder Starter vor dem starten eine eindeutige Starternummer. Dieser wird benötigt, da diese im Namen des ggT-Prozesses genutzt wird.

**ggt-Prozess:**  
Allgemein: Die Hauptaufgabe des ggt-Prozesses ist es, den Algorithmus, zur Berechnung des ggT, auszuführen. Zudem kann der ggt-Prozess ein Voting starten, wenn er eine gewisse Zeit nichts mehr berechnet hat.

**Realisirung:** Der ggt-Prozess wird vom Starter erzeugt. Nach der Erzeugung durch den Starter, welches jedem Prozess einen eindeutigen Namen zuweist, meldet sich der ggt-Prozess beim Koordinator und beim Namendienst und teilt ihm seinen Namen mit. Zuzüglich registriert er sich lokal mit seinem Namen als Erlang-Node.

Hiernach erwartet er vom Koordinator seine Nachbarn und danach seine Mi-Zahl und ist ab dann bereit eine Berechnung zu starten.

Die Nachbarn werden über die Schnittstelle {setneighbors,LeftN,RightN} übergeben.   
Die Mi-Zahl über die Schnittstelle {setpm,MiNeu}.

Die Berechnung wird über den Koordinator oder über andere Prozesse angestoßen.   
Über die Schnittstelle {sendy,Y} wird die Berechnung angestoßen.

Der Algorithmus zur Berechnung des ggT hat eine Bedingung, wird diese erfüllt kommt es zu einer Berechnung, falls nicht wird nichts Berechnet.

Kommt es zu einer Berechnung ändert sich der Mi-Wert des Prozesses, welches der Prozess dann an den Koordinator und an seine Nachbarn über deren Schnittstellen mitteilt.

Bei jeder Berechnung wird eine gewisse Zeit gewartet um eine längere Berechnung zu Simulieren. Wird eine gewisse Zeit keine Berechnung mehr angestoßen, so wird vom Prozess ein Voting gestartet indem er die Schnittstelle des Namendienstes anspricht. Wird eine gewissen Quote an Zustimmungen erhalten, wird der Prozess terminiert. TODO:

Der ggt-Prozess hat unteranderem auch eine Schnittstelle die sein aktuelles Mi zurück gibt:

{From,tellmi} -> gibt das aktuelle Mi zurück

Und auch eine Schnittstelle worüber man überprüfen kann, ob der Prozess noch lebt:

{From,pingGGT} -> gibt ein {pongGGT,GGTname} zurück.

**Koordinator:**

**Allgemein:**

Der Koordinator hat die Aufgabe den Startern die steuernden Werte zu übergeben und den Prozessen deren M-Wert zu übergeben.

Hiernach startet der Koordinator die Berechnung.

Der Koordinator kann zudem alle Prozesse beenden.

Der Koordinator wird über manuelle Eingaben gesteuert.

Realisierung:

Zu Beginn liest der Koordinator seine spezifische Konfigurationsdatei ein. Hiernach öffnet der Koordinator seine Schnittstellen, damit z.B. der Starter die steuernden Werte erfragen kann oder die Prozesse sich registrieren können. Wird der Koordinator Manuel mit der Eingabe von „step“ auf bereit gesetzt, schließen sich die offenen Schnittstellen und alle vorhandenen Prozesse werden im Zufall zu einem Ring angeordnet. Um die registrierten Prozesse zu verwalten würde sich eine Liste anbieten, die sich für jedes registriere Prozess füllt. Der Ring wäre auch eine Liste, welche sich zufällig aus der Prozessliste generiert werden würde. Wird dann der Manuele Befehl „calc“ eingeben, werden allen Prozessen ein spezifischen Mi-Wert übermittelt. Dieser Mi-Wert wird über die gegebene vsutil erzeugt. Zudem werden zu allen Prozessen die jeweiligen Nachbarn zugeordnet und diese über die Schnittstellen der Prozesse übermittelt. Dabei wird in der Ring-Liste nach den jeweiligen Nachbarn geschaut. Dabei ist zu beachten, dass das erste Element unteranderem das letzte Element in der Liste als Nachbarn hat. Hiernach werden bei 20% jedoch min. 2 Prozesse, über die Schnittstelle(sendy), die Berechnung angestoßen.

Jederzeit könnte ein ggt-Prozess den Koordinator über seine Terminierung informieren. Ist dies der Fall, so wird überprüft, ob die erhaltenen Information, welches den Mi-Wert des Prozesses beinhaltet, größer ist als die bereits bekannte Zahl. Ist dies der Fall und in den Informationen ist ein spezielles Flag gesetzt, so wird die kleinste Zahl an diesen Prozess per „sendy“ übergeben.

Durch die Manuele Eingabe von „kill“ wechselt der Koordinator in den Zustand beenden, wobei er dann alle Prozesse hierüber informiert.