Aufgabe 2: Verteilter Algorithmus

In dieser Aufgabe ist ein einfacher **verteilter Algorithmus** und **dessen Verwaltung/Koordination** zu implementieren. Jeder "Arbeits"-Prozess (ggT-Prozess) durchläuft den gleichen Algorithmus! Mit diesem Algorithmus kann man z.B. mit 42 ggT-Prozessen den ggT von 42 Zahlen "gleichzeitig" (nebenläufig) bestimmen.

Hintergrundwissen

Verwendet wird einer der ältesten, bekannten Algorithmen: **Euklids Algorithmus** zur Bestimmung des ggT, beschrieben im 7. Buch der Elemente um ca. 300 v.Chr. Dies ermöglicht uns, den Fokus auf die Koordinierung eines verteilten Algorithmus zu lenken, weil der Algorithmus selbst recht leicht zu verstehen ist.

> ***Satz von Euklid***:
> Der größte gemeinsame Teiler (ggT) zweier positiver ganzer Zahlen x, y (mit x >=y > 0) ist gleich dem ggT von y und dem Rest, der bei ganzzahliger Division von x durch y entsteht.

Bemerkungen

- Offenbar ist ggT(x,x) = x für alle x
- Man setzt nun noch ggT(x,0) := x für alle x
- Rekursive Realisierung: ggT(x,y) := ggT(y,mod(x,y))
- Erweiterung: mod*(x,y) := mod(x-1,y)+1
- Die Rekursion wird durch das Versenden von Nachrichten realisiert: es wird der gleiche Code aufgerufen!

Weitere Infos finden Sie z. B. hier.

Verteilter Algorithmus

1. Es werden n Prozesse verwendet. Die Prozesse werden in einem Ring organisiert, so dass der Prozess P_i die zwei Nachbarn P_{i+1} und P_{i-1} hat.
2. Jeder Prozess P_i hat seine eigene Variable M_i mit Wert $\#M_i$
3. ggT aller am Anfang bestehender M_i wird berechnet:

```
{Eine Nachricht mit der Zahl #y ist eingetroffen}
if y < Mi
   then Mi := mod(Mi-1,y) + 1;
        send #Mi to all neighbours;
fi
```

Aufgabenstellung

Das System für den verteilten Algorithmus ist so ausgelegt, dass es für eine längere Zeit installiert wird und dann für mehrere Aufgaben (also ggT-Berechnungen) zur Verfügung steht. Für die Implementierung werden im Wesentlichen fünf Module benötigt:

1. Der **Koordinator**, der den verteilten Algorithmus verwaltet. Dazu gehören das "Hochfahren" des Systems, die Koordinierung einer ggT-Berechnung, die Feststellung der Terminierung einer ggT-Berechnung und das "Herunterfahren" des Systems.
2. Den **Starter**, der im Wesentlichen das Starten von Prozessen auf für den Koordinator entfernten Rechnern ermöglicht.
3. Der **ggT-Prozess**, der die eigentliche Arbeit leistet, also die Berechnung des ggT.

4. Der **Client**, der die benötigten steuernden Werte setzt, und die Berechnung des ggT startet. Ausserdem kann er Koordinator und Starter terminieren.
5. Der **Monitor**, den Ablauf beobachtet, protokolliert und das Ergebnis anzeigt.

Am Anfang wird das System hochgefahren. Dazu ist als erstes ein Namensdienst zu starten. Dann wird der Koordinator gestartet und anschließend auf verschiedenen Rechnern die Starter. Die Starter melden sich beim Koordinator an und warten nun auf Kommandos vom Koordinator. Der Koordinator selber wartet auf Befehle vom Client.

Das System geht in die **Arbeitsphase**: Der Client wird gestartet und übergibt dem Koordinator die Anzahl der gewünschten ggT-Parameter. Daraufhin informiert der Koordinator die Starter über die jeweilige Anzahl der zu startenden ggT-Prozesse. Die Prozesse werden vom Starter gestartet. Während der Initialisierung der Prozesse melden sie sich beim Koordinator an. Dieser baut einen Ring auf und informiert die ggT-Prozesse über ihre Nachbarn und ihre ihre Werte `#Mi`. Damit wechseln die ggT-Prozesse in den Zustand "bereit". Die Anordnung der einzelnen Prozesse in dem Ring soll **zufallsgesteuert** erfolgen.

Die drei ggt-Prozesse, die vom Koordinator die kleinsten Startwerte erhalten haben, werden vom Koordinator beauftragt, mit der Berechnung zu beginnen. Erhält ein ggT-Prozess im Zustand "bereit" eine Zahl, so versetzt er sich in den Zustand "rechnen". Meldet ein ggT-Prozess die Terminierung der aktuellen Berechnung, so erhält der Koordinator gleichzeitig von ihm das Endergebnis der Berechnung. Der ggT-Prozess versetzt sich in den Zustand "beendet".

Erhält während eines Laufs des Algorithmus (Zustand "rechnen") ein ggT-Prozess $t_{Timeout}$ Sekunden lang keine Zahl y, startet er eine **Terminierungsabstimmung**: Dazu stößt der Prozess die Übertragung einer Terminierungsanfrage an, die von den Prozessen entlang des Ringes solange weitergereicht wird, bis sie wieder bei dem ursprünglichen Absender angekommt. Die Terminierungsnachricht wird mit dem Inhalt *"terminiere"* gestartet. Ist ein Empfänger bereit zu terminieren, reicht dieser die Terminierungsnachricht weiter ebenfalls mit dem Inhalt *"terminiere"*. Falls er nicht bereit ist, reicht er die Terminierungsnachricht auch weiter, aber mit dem Inhalt *"terminiere nicht"*. Ist der Inhalt *"terminiere nicht"* so muss ein Empfänger an seinen rechten Nachbarn ebenfalls *"terminiere nicht"* senden. Ein Empfänger ist bereit zu terminieren, wenn seit der letzten Zusendung einer Zahl mehr als $t_{Timeout}/2$ Sekunden vergangen sind. Erhält ein Prozess von seinem linken Nachbarn die eigene Terminierungsanfrage mit dem Inhalt *"terminiere"* so terminiert der Prozess. Er meldet sein Ergebnis an den Monitor und teilt dem Koordinator mit, dass die Berechnung beendet ist. Der Koordinator veranlasst dann die Beendigung aller ggT-Prozesse. Das System ist jetzt bereit, eine neue Berechnung durchzuführen.

Der Client kann über den Koordinator die Beendigung des Systems anstossen. Die Starter werden vom Koordinator über die Beendigung des Systems informiert. Falls noch ggT-Prozesse laufen, sollen diese unabhängig von ihrem momentanen Zustand möglichst unverzüglich beendet werden. Anschließend beenden sich die Starter und zum Schluss der Koordinator.

Funktionalität

Client

1. Der Client kann beim Koordinator drei Aktionen ausführen: Abfrage der bekannten Starter, Starten der ggT-Berechnung und Beendigung des Systems
2. Die für die ggT-Berechnung benötigten steuernden Werte sollen über die Kommandozeile eingegeben werden. Die Werte sind:
   - der Name des Koordinators
   - der Name des Monitors
   - ein Intervall für die Anzahl der ggT-Prozesse auf einem Rechner,
   - ein Intervall für die Verzögerungszeit der ggT-Prozesse,
   - ein Wert für $t_{Timeout}$ und
   - der gewünschte ggT.
3. Der Client ermittelt den Koordinator über den Corba-Namensdienst unter dem per Kommandozeile vorgegebenen Namen.
4. Der Client ermittelt den Monitor über den Corba-Namensdienst unter dem per Kommandozeile vorgegebenen Namen.

Koordinator

1. Der Koordinator ist unter einem über die Kommandozeile vorgebbaren Namen beim CORBA-Namensdienst registriert.
2. Die benötigten steuernden Werte erhält der Koordinator mit dem Aufruf durch den Client.
3. Nach dem Starten des Koordinators wartet er auf Registrierungen von Starter und auf Kommandos vom Client.
4. Verlangt der Client eine ggT-Berechnung, so beauftragt der Koordinator seine Starter mit dem Anlegen der ggT-Prozesse. Die Anzahl der Prozesse wird für jeden Starter mit Hilfe einer Zufallsfunktion aus dem vom Client vorgegebenen Intervall bestimmt.
5. Die Prozesse melden sich beim Koordinator an. Sie identifizieren sich mit dem Namen des Starters und durch eine fortlaufende Nummer, die durch den Starter vergeben wird. Der Koordinator erstellt daraus eine Liste.
6. Nachdem die Prozesse angelegt worden sind, baut der Koordinator den Ring auf, indem er per Zufall die ggT-Prozesse auswählt. Jedem ggT-Prozess wird übermittelt, welche Nachbarn er besitzt. Gleichzeitig bekommen sie die für die Berechnungen notwendigen Informationen:
   - Startwert für die ggT-Berechnung:
     `gewünschter_ggT * Zufallszahl_1_bis_100 * Zufallszahl_1_bis_100.`
   - Verzögerungszeit des ggT-Prozesses, ermittelt als Zufallszahl aus dem vom Client übergebenen Intervall.
   - $t_{Timeout}$
   - Referenz des Monitors zum Protokollieren des Ablaufs.
7. Der Koordinator teilt dem Monitor den Aufbau des Ringes und die zu berechnenden Zahlen mit.
8. Anschließend startet der Koordinator die Berechnung. Dazu wählt er die drei Prozesse aus, denen er die kleinste Zahl zugewiesen hat.
9. Nach Terminierung der Berechnung beauftragt der Koordinator die Starter mit der Beendigung der Prozesse.
10. Per Aufruf vom Client kann der Koordinator beauftragt werden, dass System zu beenden. Es werden zunächst die Starter gestoppt, anschließend beendet sich der Koordinator selber.

Starter

1. Dem Starter wird per Kommandozeile einen Namen zugewiesen (z.B. Name des Rechners, möglichst kurze, aussagekräftige Namen verwenden).
2. Der Starter registriert sich beim Koordinator und übergibt ihn dabei seinen Namen.
3. Anschließend wartet der Starter auf Kommandos vom Koordinator. Für eine Berechnung wird der Starter beauftragt, eine bestimmte Anzahl von ggT-Prozessen zu starten. Die Prozesse bekommen eine laufende Nummer zugeordnet, unter der sie sich beim Koordinator bekannt machen können.
4. Bekommt der Starter den Auftrag, sich zu beenden, so muss er zunächst seine eventuell noch vorhandenen Prozesse stoppen und kann sich danach selber beenden.

ggT-Prozess

1. Der ggT-Prozess wird vom Starter gestartet und muss sich als erstes mit einer Identifikation (String), die sich aus dem Namen des Starters und der vom Starter vergebenen fortlaufenden Nummer ergibt, beim Koordinator registrieren.
2. Danach wartet der Prozess auf die Parametrisierung durch den Koordinator. Anschließend ist der Prozess "bereit" und kann auf Berechnungsanforderungen reagieren. Diese können initial vom Koordinator oder von den Nachbarn eintreffen.
3. Erhält der Prozess von einem seiner Nachbarn eine neue Zahl, so teilt er diese dem Monitor mit.
4. Muss er eine Berechnung durchführen (empfangene Zahl ist kleiner als die eigene) braucht er eine gewisse Zeit (die Verzögerungszeit), die ihm vom Koordinator bei der Parametrisierung mitgegeben wurde. Dies simuliert eine größere, zeitintensivere Aufgabe. Der ggT-Prozess kann z.B. in dieser Zeit einfach nichts tun.
5. Der ggT-Prozess beobachtet die Zeit seit dem letzten Empfang einer Zahl. Hat diese $t_{Timeout}$ Sekunden überschritten, startet er eine Terminierungsanfrage.
6. Die Abstimmung arbeitet wie folgt:
   1. Wird eine Terminierungsanfrage empfangen, so ist das dem Monitor mitzuteilen
   2. Ist der Inhalt der Anfrage *"terminiere"* und ist seit dem letzten Empfang einer Zahl mehr als $t_{Timeout}/2$ ($t_{Timeout}$ halbe) Sekunden vergangen, dann ist der Inhalt der an seinen rechten Nachbarn weitergeleiteten Anfrage ebenfalls *"terminiere"*. Ansonsten ist die Anfrage mit dem Inhalt *"terminiere*

nicht" weiterzuleiten.

7. Ist die Terminierungsanfrage erfolgreich durchgeführt, sendet der Prozess dem Koordinator eine Mitteilung über die Terminierung der aktuellen Berechnung. Zusätzlich ist der Monitor über die Terminierung zu informieren. Dabei ist die Identifizierung und der errechnete ggT.

Monitor

Der Monitor wird als fertiges Programm hier als jar-Datei zur Verfügung gestellt. Die Quelldateien befinden sich in diesem Zip-Archiv. Seine Schnittstelle ist mit dieser IDL-Datei realisiert. Gestartet werden kann der Monitor mit

```
java -jar GGTMonitor.jar <MonitorName> -ORBInitialHost <host> -ORBInitialPort <port>
```

Durchführung:

Folgende Schritte sind für eine erfolgreiche Durchführung der Aufgabe notwendig:

- Es ist eine schriftliche Ausarbeitung des Konzepts und des Lösungsansatzes zu erstellen. Der Entwurf muss als .pdf-Datei, verpackt in einer .zip-Datei, spätestens am **Abend vor dem Praktikumstermin** mit Hilfe des Uploaders auf der Webseite abgeliefert werden. Der Entwurf sollte grob beschreiben, wie Sie sich die Realisierung denken, z. B. wie Sie sich die Abläufe der Zugriffe vorstellen, welche Fehler auftreten können und wie darauf reagiert werden soll etc. Typische Abläufe sollen als Sequenquenzediagramm dargestellt werden, dazu kann man dieses Tool verwendet werden: Quick Sequence Diagram Editor. Ein kleines Beispiel zur Benutzung des Editors finden Sie hier, Der Entwurf muss den aktuellen Stand des **Klassendiagramms** enthalten.
- Sie müssen die erfolgreiche Versuchsdurchführung mittels einer Befragung abnehmen lassen. Diese soll nach Möglichkeit am gleichen Tag erfolgen, kann aber auch am darauffolgenden Labortermin stattfinden, also spätestens am:
  Gruppe 1: 16.05.2012
  Gruppe 2: 23.05.2012
  Gruppe 3: 06.06.2012
  Die Abgabe der Dokumentation einschließlich des Sourcecodes muss allerdings **spätestens einen Tag davor** mit Hilfe des Uploaders erfolgt sein!

Heitmann
Druckversion