Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Prof. Dr. W.-P. de Roever
Jens Schönborn und Jan Waller

Nebenläufige und verteilte Programmierung

Wintersemester 2006/07

Serie 12 (EST)

22.1. 2007

Ausgabetermin: 22.1. 2007

Abgabe: 1.2. 2007 (14:00)

Diese Serie soll in Einzelarbeit gelöst werden! Auch Abschreiben lassen führt zur Disqualifikation.

Jede Aufgabe soll auf einem neuen Zettel begonnen werden. Verseht bitte alle Zettel die ihr abgebt mit eurem Namen.

Aufgabe 1 (2 Punkte) Show by means of an example that the following inference rule is not sound:

$$\frac{\{P_i\}\ S_i\ \{Q_i\}}{\{P_1 \wedge \ldots \wedge P_n\}\ \text{co}\ S_1; // \ldots // S_n;\ \text{oc}\ \{Q_1 \wedge \ldots \wedge Q_n\}}$$

Explain your example.

Aufgabe 2 (4 Punkte) Gegeben seien $n > 0$ Prozesse, wobei jeder Prozess eine individuelle Information hat. Beweise formal, dass mindestens $2 \cdot (n-1)$ one way message passing nötig sind, so dass jeder die Information des anderen weiß. D.h. zeige, dass es keine bessere Strategie als die zentralisierte- oder die Ring-Lösung (Andrews S. 317) gibt, um eine möglichst kleine Anzahl an gesendeten Nachrichten zu erhalten.
Tipp: Nehme zuerst ein kleinstes n von Nachrichten an, welches nötig ist damit ein Prozess alle Informationen erhält.

Aufgabe 3 (2 Punkte) [*Message passing using semaphores*]
Sei `synch_send` ein Kommando für synchrones Senden und `receive` das übliche (synchrone) Empfangen. Beide Kommandos sind also blockierend, d.h. jedes der Kommandos terminiert erst, nachdem das Gegenstück aufgerufen und die Nachricht ausgetauscht wurde. Der Kanal soll als Mailbox-Kanal aufgefasst werden, d.h. beliebig viele Prozesse dürfen auf dem Kanal senden und empfangen.

Benutze Semaphoren (für wechselseitigen Ausschluss und zum Signalisieren) und entwickle eine Implementierung (Pseudocode) von `synch_send(msg)` und `receive(msg)`. Nimm an,

dass es einen Pufferplatz der Größe 1 gibt, um die Nachricht vom Sender zum Empfänger zu kopieren. Minimiere die Anzahl der Semaphoren (ohne Busy Waiting zu benutzen). Erkläre die Lösung.

Aufgabe 4 (8 Punkte) [*One-Lane Bridge / Lindaunis*] Eine einspurige Brücke soll modelliert werden, die von Fahrzeugen von Norden nach Süden sowie in umgekehrter Richtung überquert werden soll. Fahrzeuge, die in der gleichen Richtung unterwegs sind, dürfen gemeinsam auf die Brücke fahren. Fahrzeuge in die andere Richtung dürfen dann nicht auf die Brücke.

Zusätzlich muss gelegentlich ein Zug über die Brücke, in diesem Fall darf also kein anderes Fahrzeug auf die Brücke. Es passt immer nur ein Zug zur Zeit auf die Brücke.

Von Zeit zu Zeit wird die Brücke hochgeklappt, um wartende Segelboote passieren zu lassen. Ist die Brücke hochgeklappt, so dürfen natürlich keine Züge und Kfz auf die Brücke. Es ist aber möglich, dass mehrere Boote gleichzeitig die Brücke passieren.

1. Entwickle einen Server-Prozess der den Verkehr über die Brücke regelt. Nimm an, dass die Fahrzeuge, Züge und Boote Client Prozesse sind. Benutze asynchrones Message passing für die Prozessinteraktion. Die Lösung soll fair sein.

2. Entwickle eine Simulation der Lösung zu (1). Benutze MPD. Die Verkehrsteilnehmer sollen immer wieder versuchen die Brücke zu überqueren. Jeder Teilnehmer soll eine zufällige Zeit lang auf der Brücke verweilen, und nach dem Verlassen eine zufällige Zeit lang warten, bis er erneut versucht, die Brücke zu passieren. Sende den Quellcode der Simulation an **jes@informatik.uni-kiel.de**.

Erkläre und kommentiere die Lösungen.

Aufgabe 5 (6 Punkte) Consider the following specification for a program to find the minimum of a set of integers. Given is an array of processes Min[1:n]. Initially, each process has one integer value. The processes repeatedly interact, with each one trying to give another the minimum of the set of values it has seen. If a process gives away its minimum value, it terminates. Eventually, one process will be left, and it will know the minimum of the original set.

1. Develop a Java program to solve this problem using only the rmi package explained in Section 8.5. Send the source code to **jes@informatik.uni-kiel.de**.

2. Develop a program to solve this problem using only the rendezvous primitives defined in Section 8.2. Pseudocode is sufficient.

Give comments and explanation for both answers.