
Concurrent Programming<http://www-pu.informatik.uni-tuebingen.de/cp-2006>

Blatt 5

Abgabe: 13.6.2006

1. [5 Punkte] Manche parallelen Lisp-Dialekte bieten einen Mechanismus namens *Futures* an. Ein Future vereint Thread-Erzeugung, Kommunikation und Synchronisation. Auch Futures lassen sich durch Rendezvous implementieren: Die Funktion `future-rv` hat als Parameter einen Thunk (eine Funktion ohne Parameter). `future-rv` liefert ein Rendezvous zurück, auf dem sich synchronisieren läßt, wenn der Thunk nebenläufig abgelaufen ist; der Wert der Synchronisation ist dann der Rückgabewert des Thunks. Implementiere `future-rv`!

2. [5 Punkte] Die Programmiersprache ID-90 bietet sogenannte I-Strukturen zur Synchronisation zwischen Threads an. Bei einer solchen Struktur handelt es sich um einen Platzhalter für einen Wert, auf den synchronisiert mit `put`- und `get`-Operationen zugegriffen wird.

Die `put`-Operation füllt die Speicherzelle mit einem bestimmten Wert. Ist die Speicherzelle bereits gefüllt, signalisiert `put` einen Fehler. Die `get`-Operation blockiert solange, bis die Speicherzelle mit einem Wert gefüllt ist. `Get` darf mehrfach auf derselben I-Struktur aufgerufen werden.

Programmiere die I-Strukturen mit den zugehörigen Operationen in Scheme! Programmiere dazu zuerst die Prozedur `get-rv`, die ein Rendezvous zurückgibt, auf das synchronisiert werden kann, wenn die Speicherzelle gefüllt wurde. Der Rendezvous-Wert ist der Inhalt der I-Struktur. Implementiere `get` mit Hilfe von `get-rv`. Warum macht es Sinn `get-rv` bereitzustellen, aber nicht `put-rv`?

3. [20 Punkte] Das WSI expandiert. Das neue Gebäude hat 12 Stockwerke und drei Fahrstühle, die jedoch ohne eine Software zur Steuerung ausgeliefert wurde. Implementiere das fehlende Fahrstuhlkontrollsystem mit Hilfe von CML in Scheme!

Gehe dabei so vor: Repräsentiere jeden Fahrstuhl und jede Etage durch einen Thread. Auf jeder Etage gibt es nur einen Knopf, um den Fahrstuhl zu rufen. Drückt jemand diesen Knopf, werden alle Fahrstühle gefragt, wie lange sie für die Fahrt zu der entsprechenden Etage brauchen. Der Thread, der die Etage repräsentiert, ruft dann den Fahrstuhl mit der kürzesten Wartezeit. Im Fahrstuhl wählt der Benutzer die Etage, zu der er fahren möchte. Der Fahrstuhl hält dann auf jeden Fall in der gewünschten Etage. Etagen und Fahrstühle kommunizieren über CML-Kanäle.

Programmiere zuerst eine Teillösung, die aus den Etagen und vereinfachten Fahrstühlen besteht. Die vereinfachten Fahrstühle haben keine Knöpfe, um eine Zieletage zu wählen. Vervollständige deine Implementierung erst, wenn diese Teillösung funktioniert.

In einem zweiten Schritt ist also noch die Wahl einer Zieletage durch den Fahrgast zu implementieren. Gehe davon aus, dass der Fahrgast nach dem Betreten des Fahrstuhls eine zufällige Etage wählt. Es gilt also: Der Fahrstuhl hält in einer Etage, zu der er durch den Rufknopf gerufen wurde, es steigt ein Fahrgast ein und wählt ein zufällige Etage.

Erweitere dein Programm so, dass sich beobachten läßt, was passiert (Z. B. in dem Fahrstühle und Etagen Statusmeldungen ausdrucken).

Hinweis 1 Zufallszahlen lassen sich in scsh mit der Funktion `random-integer` aus dem Modul `srfi-27` erzeugen: (`random-integer n`) gibt eine zufällig Integer-Zahl zwischen 0 und $n - 1$ zurück.

Hinweis 2 Es ist sinnvoll für jeden Lift ein Rendezvous zu programmieren, welches immer dann synchronisiert werden kann, wenn der Fahrstuhl an einer Etage angelangt ist. Dadurch läßt sich die Fahrzeit zwischen zwei Etagen überbrücken.