

---

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

---

## Blatt 6

Abgabe: 20.6.2006

1. [10 Punkte] Programmiere einen Server, der eine Liste von Semaphoren verwaltet. Mit dem Funktionsaufruf (`get-semaphore n`) legt der Client eine neue Semaphore an, die fortan vom Server verwaltet wird. Das Argument  $n$  ist der initiale Zählerstand; Rückgabewert von `get-semaphore` ist ein Record, welches die Semaphore repräsentiert.

Die vom Server verwalteten Semaphoren haben eine ähnliche Funktionsweise wie die aus der Vorlesung bekannten Semaphoren: Sie beschreiben durch einen Zähler die Anzahl freier Ressourcen. Es gibt eine Funktion `semaphore-wait-rv`, die als Argument eine Semaphore nimmt und ein Rendezvous zurückliefert. Dieses Rendezvous kann nur dann synchronisiert werden, wenn der Zähler der Semaphore größer als Null ist (also noch mindestens eine Ressource frei ist). Wurde in einer Rendezvous-Auswahl das von `semaphore-wait-rv` erzeugte Rendezvous gewählt, so wird eine Ressource reserviert. Andernfalls steht die Ressource weiterhin zur Verfügung — achte darauf bei der Programmierung des Servers! Mit `semaphore-post` gibt der Client eine Ressource wieder frei.

*Hinweis:* Orientiere dich an der Implementierung des Lock-Servers aus der Vorlesung. Verwende keine Locks oder Semaphoren in deiner Implementierung.

2. [13 Punkte] Der Multicast-Timeout-Channel dient dazu eine Nachricht an eine beliebige Anzahl von Lesern zu verteilen. Ein Schreiber sendet die Nachricht in den Kanal gibt außerdem noch eine Zeitspanne an. Die Leser lesen die Nachricht und bestätigen den Empfang. Mit Hilfe der Bestätigungen ist es dem Schreiber möglich festzustellen, ob alle Leser diese Nachricht gesehen haben.

Ein Multicast-Timeout-Channel wird durch `make-multicast-timeout-channel` erzeugt und bekommt als Argument eine der Liste der Kanäle, auf denen diese Leser auf den Eingang von Nachrichten warten. Die Leser verwenden die Funktion `receive-multicast-message`, um die Nachricht zu lesen und den Empfang zu bestätigen. Der Schreiber sendet die Nachricht mit `broadcast-message` in den Multicast-Timeout-Channel. Diese Funktion gibt zwei Rendezvous zurück: Auf einem läßt sich genau dann synchronisieren wenn alle Leser innerhalb der Zeitspanne bestätigt haben. Auf dem anderen Rendezvous kann synchronisiert werden, wenn dies nicht der Fall ist.

3. [12 Punkte] Programmiere asynchrone Kanäle in Rendezvous-Form. Schreibe dazu eine Funktion `make-async-channel` (ohne Parameter), die einen asynchronen Kanal zurückliefert. Programmiere dann eine Funktion `send-async`, die als Argumente einen asynchronen Kanal sowie eine Nachricht akzeptiert, und die Nachricht asynchron auf den Kanal schreibt. Implementiere dann noch die Funktion `receive-async-rv`, die ein Rendezvous liefert, das sich synchronisieren läßt, wenn eine Nachricht auf dem Kanal verfügbar ist. Dabei soll die Synchronisation — wie gehabt — die Nachricht aus dem Kanal entfernen und zurückgeben.