
Concurrent Programming

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

Blatt 3

Abgabe: 23.5.2006

1. [10 Punkte] Erweitere die Lösung für das Producer/Consumer-Problem mit Condition-Locks für mehrere Threads aus der Vorlesung (`prodcons-cond-lock-broadcast.scm`), so dass der Puffer mehrelementig ist und die feste Länge n hat.

Schreibe ein Testprogramm, das die Grenzfälle deines Codes strapaziert. Dazu ist u. U. die Prozedur `sleep` praktisch, die als Argument eine Zeitdauerangabe in Millisekunden akzeptiert und den laufenden Thread für diese Dauer blockiert. Dokumentiere, welche Grenzfälle auftreten und abgetestet werden, und wie du aus dem Verhalten des Testprogramms auf die Korrektheit deiner Lösung (oder ihr Gegenteil) schließt.

Beantworte außerdem folgende Frage und begründe deine Antwort: Kommen bei mehreren Producern und einem Consumer die Daten in der gleichen Reihenfolge beim Consumer an, wie die `insert`-Aufrufe in den Producern auftreten?

2. [8 Punkte] Programmiere eine Lösung für das Producer/Consumer-Problem, bei der Producer und Consumer über einen Puffer unbeschränkter Länge kommunizieren. Dabei sollen, wie auch beim Programm aus der Vorlesung, beliebig viele Producer und Consumer auf den Puffer zugreifen können.
3. [7 Punkte] Die folgende Situation ist auch unter dem Namen Readers/Writers-Problem bekannt: In einem System gibt es zwei Sorten von Threads die auf einen gemeinsamen Puffer zugreifen. Die eine Sorte Threads (die *Leser*) liest aus diesem Puffer (dabei bleibt der Inhalt des Puffers erhalten), die andere Sorte (die *Schreiber*) füllt den Puffer mit neuen Werten. Es dürfen beliebig viele Threads gleichzeitig aus dem Puffer lesen, während des Schreibvorgangs hat allerdings nur der Schreiber Zugriff auf den Puffer. Insbesondere bedeutet das, dass Leser warten müssen wenn gerade geschrieben wird und Schreiber warten müssen wenn gerade gelesen wird. Gleichzeitiges Lesen zu erlauben ist in der Praxis wichtig, falls die Leseoperation länger dauert, zum Beispiel falls der Puffer durch eine Datenbank implementiert ist.

Implementiere mit Hilfe der SRFI-9-Records eine Datenstruktur `rw-buffer` mit synchronisierten Zugriffsfunktionen, die genau dieses Verhalten implementiert! Verwende zur Synchronisation ein Mutex-Lock und eine Condition-Variable. Schreibe außerdem ein kleines Test-Programm, um deine Implementierung zu testen.

Hinweis: Am einfachsten ist es, den Schreiber so lange warten zu lassen, bis keine Leser mehr auf den Puffer zugreifen.

4. [5 Punkte] Bei den Puffern die wir bis jetzt für das Producer/Consumer-Problem entwickelt haben, besteht der Nachteil, dass ein Consumer immer nur aus einem Puffer gleichzeitig lesen kann. Schreibe nun eine Prozedur `make-remove`, die als Argument eine Liste von Puffern bekommt und eine Prozedur ohne Argumente zurückliefert. Die Puffer sollen dabei durch eine der bisherigen Implementierungen erzeugt worden sein. Die Consumer rufen nun die zurückgegebene Prozedur anstelle von `remove` auf und sollen einen Wert erhalten, sobald in einen der Puffer aus der Liste ein Wert verfügbar ist. Damit der Consumer weiß, aus welchem Puffer der Wert stammt, ist es außerdem sinnvoll, ein Paar aus Puffer und Wert zurückzugeben.