Prof. Dr. R. Loogen, M. Dieterle
Fachbereich Mathematik und Informatik
Hans-Meerwein-Straße
D-35032 Marburg

31.Oktober 2008

Philipps Universität Marburg

# Übungen zu „Parallele Programmierung“, WS 2008/09

**Nr. 3, Abgabe der Aufgaben: 7.November 2008 in der Übung**

---

3.1 **Sparkonto-Monitor** — 3 Punkte

Spezifizieren Sie einen Monitor zur Verwaltung eines Sparbuchs, dessen Kontostand stets $\geq 0$ ist. Einzahlungen beliebiger Beträge können zu jeder Zeit vorgenommen werden. Auszahlungen von Geldbeträgen werden sofort ausgeführt, falls ausreichend Geld auf dem Konto ist.

Auszahlungen, die nicht ausführbar sind, werden blockiert und vorgenommen, sobald das Konto wieder ausreichend gedeckt ist.

**Beispiel:**
Der Kontostand sei €1, und es warten die folgenden Auszahlungen: [€9,€5,€3,€3]. Nach einer Einzahlung von €6 wird die Auszahlung von €5 vorgenommen.

Setzen Sie eine FIFO-Queue für Bedingungsvariablen voraus. Geben Sie an, welche Signalisierungsmethoden Ihr Monitor verwenden kann.

3.2 **Odd-Even-Transposition Sort** — 5 Punkte

Das Verfahren *Odd-Even-Transposition Sort* ist eine parallele Variante von Bubble-Sort. Dem $i$. Prozessor wird das Element $a_i$ zugewiesen. Der Algorithmus besteht aus zwei sich wiederholenden Phasen:

- Beim Ungerade/Gerade-Austausch vergleicht jeder $k$. Prozessor ($k$ ungerade) sein Element mit demjenigen seines rechten Nachbarn (sofern vorhanden) und vertauscht die Elemente falls dies nötig ist.
- Beim Gerade/Ungerade-Austausch vergleicht jeder $k$. Prozessor ($k$ gerade) sein Element mit dem Element seines rechten Nachbarn und vertauscht diese falls nötig.

(a) Wie viele parallele Schritte benötigt der Algorithmus? / 1

(b) Programmieren Sie eine parallele Version von Odd-Even-Transposition Sort in MPD. Verwenden Sie zur Kommunikation ausschließlich Nachrichten und Aufrufparameter. / 4

Bitte wenden!

## 3.3 Kommunikation in MPD

4 Punkte

```
resource sync
  type item = int
  op get() returns item {call}
  op put(item) {call}
body sync()
  op try(cap(item))
  process manager {
    cap(item) go
    item x;
    while(true) {
#      in put(x) -> receive try(go); send go(x); ni
       receive put(x); receive try(go); send go(x);
    }
  }
  proc get() returns x {
    op take(item)
    send try(take)
    receive take(x)
  }
end sync
```

(a) Beschreiben Sie die Funktionsweise sowie den Zweck der Resource `sync`. Schreiben Sie ein kleines Beispielprogramm, das `sync` sinvoll benutzt. / 2

(b) Was ändert sich, wenn der `receive`- mit dem `in`- Befehl vertauscht wird (auskommentierte Zeile). / 1

(c) Welchen Effekt hat es, die Operation `put` mit `send` statt mit `call` zu benutzen? / 1