PHILIPPS-UNIVERSITÄT MARBURG D-35032 Marburg
Fachbereich Mathematik und Informatik Hans Meerwein Straße
Prof. Dr. R. Loogen
Dipl.Inf. J.Berthold

28.April 2005

Übungen zu „Parallele Programmierung“, SS 2005

Nr. 3, Abgabe der Aufgaben: 4.Mai 2005 vor der Vorlesung

Hinweise:
Wegen des Feiertags ist dieses Blatt bereits am **4.Mai 2005 (Mittwoch)** abzugeben.

Aufgaben

3.1 **Erzeuger/Verbraucher mit beschränktem Puffer** 3 Punkte

Das in der Vorlesung vorgestellte *Erzeuger/Verbraucher-System* basiert auf einem in der Größe unbeschränkten Puffer.

Schreiben Sie ein MPD -Programm zur Erzeuger-Verbraucher-Synchronisation mit einem *beschränkten* Puffer.

3.2 **Dateidifferenz mit drei Prozessen** 4 Punkte

Schreiben Sie ein MPD -Programm, das zwei als Parameter anzugebende Textdateien zeilenweise auf Unterschiede prüft und diese ausgibt (d.h. implementieren Sie ein einfaches `diff`-Kommando). Ihr Programm soll alle Zeilenpaare, die sich voneinander unterscheiden, mit Dateiname und Zeilennummer wie folgt ausgeben (Beispiel):

```
#>myDiff datei1.txt datei2.txt
datei1.txt:n: <Zeile n aus Datei 1>
datei2.txt:n: <Zeile n aus Datei 2>
#>
```

wobei eine der Ausgabezeilen wegfällt, falls die entsprechende Datei weniger Zeilen als die andere enthält.

Verwenden Sie für Datei-I/O die folgenden Funktionen (vgl. auch MPD -Anleitung):

- `open(Pfad,Modus)` zum Öffnen einer Datei (liefert einen Dateideskriptor). Gültige Modi sind `READ,WRITE, READWRITE`.
- `read(Dateidescr,Stringvar)` zum Einlesen einer Zeile in einen (ausreichend großen!) String. Sie können eine maximale Zeilenlänge festlegen.
- `close(Dateidescr)` zum Schließen einer Datei.

Erzeugen Sie zwei konkurrierende Prozesse, um die Dateien in globale Puffer einzulesen, während ein dritter Prozess den Vergleich durchführt. Wie erreichen Sie, dass die konkurrierenden Prozesse synchronisiert werden? (Welches klassische Synchronisationsproblem liegt zugrunde?)

Testen Sie, mit welcher Puffergröße das Programm für identische Dateien am schnellsten arbeitet. Die MPD -Funktion `age()` liefert (als `int`) die Zeit seit dem Programmstart in Millisekunden.

Bitte wenden!

3.3 **Bestimmung von Präfixsummen**

5 Punkte

Das folgende Programm soll Präfixsummen eines Arrays `sum` von n Zahlen bestimmen. Das zweite Array `old` dient dabei zur Zwischenspeicherung alter Werte.

```
resource partial_sums()
  op saveAndUpdate(int i)
  ... # Deklarationen und Initialisierungen
  d = 1
  while (d < n) { # Berechnung
    co[i = 1 to n]  saveAndUpdate(i)  oc
    d *= 2
  }
  for[i = 1 to n] { # Ausgabe
    write(i, sum[i])
  }

  proc saveAndUpdate(i) {
    old[i] = sum[i]      # save
    if (i-d >= 1) {
      sum[i] += old[i-d] # update
    }
  }
end partial_sums
```

(a) Beschreiben Sie, welche Probleme in diesem Programm auftreten können. Korrigieren Sie das Programm, so dass das berechnete Ergebnis stets korrekt ist.

(b) Programmieren Sie eine weitere Version des Algorithmus, welche kein `co...oc` benutzt. Vergleichen Sie die Laufzeiten beider Programme mit Hilfe der MPD - Funktion `age()`.