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

24.Oktober 2008

Philipps Universität Marburg

Übungen zu „Parallele Programmierung“, WS 2008/09

Nr. 2, Abgabe der Aufgaben: 31.Oktober 2008 in der Übung

Prozesse in MPD

In den Beispielen und der ersten Übung kamen bereits Prozeduren und (neben dem `co`-Konstrukt) die Anweisung `fork` vor. MPD stellt ein weiteres Konstrukt zur Verfügung: mit `process` können Prozesse definiert und gleich gestartet werden, ohne explizit eine Prozedur dafür zu definieren.

```
resource meinProgramm()
  int i,j

  # starte 25 konkurrierende Prozesse
  process meinProzess[i = 1 to 5,j = 8 downto 0 by -2] {
    # Anweisungen können i und j als Parameter benutzen
    write("Prozess (",i,",",j,") ...")
  }

  # Rest konkurrierend zu erzeugten Prozessen ausgeführt
  write("Hauptprozess ...")

  final { # nach Ende aller gestarteten Prozesse der resource:
      write("Programm beendet.")
  }
end meinProgramm
```

Der Rumpf des Prozesses wird als konkurrierender Thread abgespalten, der Hauptprozess läuft weiter (nicht synchronisiert). Den gleichen Effekt kann man mit `fork` und einer passenden Prozedur erzielen.
Durch die Quantifizierung im Beispiel wird gleich eine ganze Serie von Prozessen gestartet. Dabei sind `i` und `j` die Aktualparameter der implizit definierten Prozedur.

Aufgaben

2.1 **PCAM-Entwurf** — 4 Punkte

Entwerfen und diskutieren Sie *nach der PCAM-Methode* ein paralleles Sortierverfahren, das auf dem „Teile und herrsche“-Prinzip basiert.

Bitte wenden!

2.2 **Leser/Schreiber-Problem** 4 Punkte

Betrachten Sie die folgende Lösung des Leser/Schreiber-Problems.

```
int nr = 0, nw = 0;
sem e = 1;
sem r = 0, w = 0;
int dr = 0, dw = 0;

proc reader(){
 while (true) {
  P(e);
   if (nw == 0)
       { nr += 1; V(r);}
     else { dr += 1 }
  V(e);
  P(r);
   # lese Daten
  P(e);
   nr -= 1;
   if (nr == 0 and dw > 0)
     {dw -= 1; nw = 1; V(w); }
  V(e);
 }
}
```

```
proc writer(){
 while (true) {
  P(e);
   if (nr == 0 and nw == 0)
        { nw = 1; V(w);}
    else { dw += 1 }
  V(e);
  P(w);
   # schreibe Daten
  P(e);
   nw = 0;
   if (dw > 0)
     { dw -= 1; nw = 1; V(w); }
    else
     { while (dr > 0) {
        dr -= 1; nr += 1; V(r)}
     }
  V(e);
 }
}
```

(a) Beschreiben Sie die Funktionsweise dieser Synchronisation, insbesondere die Rolle der verwendeten Semaphore. / 2

(b) Begründen Sie, dass Schreiber exklusiven Zugriff auf die Daten haben. Ist die Synchronisation *fair*, oder werden Schreiber oder Leser bevorzugt? / 2

2.3 **Optimierte Barrieren** 4 Punkte

Wie in der Vorlesung angesprochen wurde, ermöglicht die Technik des *rekursiven Doppelns* effiziente Realisierungen von Barrieren mit Aufwand $O(\log_2 n)$.

Programmieren Sie eine Barriere für $n = 2^k (k \geq 1)$ Prozesse, welche ein sog. *Butterfly-Schema* benutzt: