Übungen zur Vorlesung
Betriebssysteme I
Studiengang Informatik
WS 05/06

AG Betriebssysteme FB3
Kirsten Berkenkötter, Jan Peleska

Ausgabe: 10.11.2005
Abgabe: 24.11.2005

Übungszettel 2

Hinweise

Die Abgabe erfolgt als Ausdruck am Ende der Vorlesung und als E-Mail an *kirsten@tzi.de*. **Auf jeden Fall** sollten alle CSP-Dateien auch in elektronischer Form (als E-Mail-Attachment) abgegeben werden. Verwendet für Aufgabe 1 und 2 jeweils eigene Dateien. Zur vollständigen Lösung der Aufgabe gehören Spezifikation, Verifikation und Dokumentation (in Latex). Der Betreff der E-Mail sollte folgendes Aussehen haben:
BS1 Abgabe x Gruppe y.

Bitte immer die Namen aller Gruppenmitglieder und die Gruppennummer angeben!

Aufgabe 1: Mutual Exclusion *N*=2

Folgender Algorithmus soll dazu verwendet werden, eine kritische Region zu schützen. Dabei sollen genau zwei Prozesse diese kritische Region verwenden.

```
#define N     2                                  //number of processes

int  turn;                                       //initialized with 0
bool blocked[N];                                 //initialized with {false,false}

void enter_region(int pid)
{
  blocked[pid] = true;

  while(turn != pid)
  {
    while(blocked[1-pid])
    {
      ;                                          //active wait
    }
    turn = pid;
  }
  //now comes the critical stuff
}

void leave_region(int process)
{
  blocked[pid] = false;
}
```

a) Erläutert die Funktionsweise dieses Algorithmus.

b) Gebt (in FDR-Syntax) eine CSP-Spezifikation dieses Algorithmus an, indem ihr zwei Prozesse spezifiziert, die möglichst direkte Umsetzungen der beiden Prozeduren *enter_region* und *leave_region* enthalten.

c) Verifiziert den Algorithmus sowohl über einen Watchdog als auch über Trace Refinement, so wie es in der Vorlesung und der Übung erklärt wurde. Interpretiert das Ergebnis. Warum funktioniert der Algorithmus nicht?

d) Verändert den Algorithmus so, dass *Mutual Exclusion* wirklich garantiert wird. Es sollen wiederum beide Verifikationsmethoden angewandt werden. Es soll lediglich **EINE** Änderung vorgenommen werden.

Aufgabe 2: Mutual Exclusion $N \geq 2$

Der folgende Algorithmus soll *Mutual Exclusion* für mehr als zwei Prozesse garantieren. Die PIDs der Prozesse sind im Bereich *{1..N}*.

```
#define N                                       //number of processes

int x;                                          //in {1..N}
int y;                                          //in {0..N}, initialized with 0
bool blocked[N+1]                               //initialized with false

void enter_region(int pid)
{
  int j;
  start: blocked[pid] = true;
         x = pid;
         if(y != 0)
         {
           blocked[pid] = false;
           while(y != 0)
             ;                                  //active wait
           goto start;
         }
         y = pid;
         if(x != pid)
         {
           blocked[pid] = false;
           for(j = 1; j <= N; j++)
             while(blocked[j])
               ;                                //active wait
           if(y != pid)
           {
             while(y != 0)
               ;                                //active wait
             goto start;
           }
         }
}

void leave_region(int pid)
{
  y = 0;
  blocked[pid] = false;
}
```

a) Erläutert die Funktionsweise dieses Algorithmus.

b) Schützt dieser Algorithmus vor

- Deadlocks,
- Livelocks,
- Starvation?

c) Warum bezeichnet man diesen Algorithmus als *schnell*? Vergleicht ihn mit der Lösung für Peterson's Algorithmus für $N \geq 2$, der in der Übung vorgestellt wurde.

d) Folgendes Codesegment wird als *Splitter* bezeichnet. Auf welches Verhalten dieses Segments bezieht sich diese Bezeichnung? Wieviele Prozesse werden maximal an den Verzweigungen scheitern?

```
x = pid;
if(y != 0)
 ...
...
y = pid;
if(x != pid)
  ...
```