Technische Universität München
Institut für Informatik
Prof. Dr. Ernst W. Mayr

WS 1993/94
Übungsblatt 9
12. Jan. 1994

Einführung in die Informatik III

(Abgabetermin: 19. Jan. 1994, 13:00Uhr)

Aufgabe 1

Beschreiben Sie in einer höheren Programmiersprache (C, Modula oder Pascal), die um den Datentyp **semaphor** mit den möglichen Werten 0 und 1 und den zugehörigen Operationen PP und VV (als Prozeduren verfügbar) erweitert worden sind, eine Lösung für das *Erzeuger-Verbraucher-Problem.* Bei diesem Problem gibt es zwei *nebenläufige, asynchrone* Prozesse, den `erzeuger` und den `verbraucher`, die mittels eines Puffers der Länge $N > 0$ kommunizieren. Der `erzeuger` erzeugt bei jedem Durchlauf einer Endlosschleife einen Gegenstand und legt ihn in den Puffer, der `verbraucher` holt sich bei jedem Durchlauf einer Endlosschleife einen Gegenstand aus dem Puffer und verbraucht ihn. Jeder Gegenstand nimmt *einen* Platz im Puffer ein. Zur Synchronisation der beiden Prozesse ist es nötig, dafür zu sorgen, daß der Puffer weder unter- noch überläuft.
Begründen Sie die Korrektheit Ihrer Lösung.

Aufgabe 2

Ereigniszähler sind Variablen vom Typ **nat**, auf denen nur die folgenden, jeweils unteilbaren Operationen ausgeführt werden können:

1. $read(e)$: diese Funktion liefert den aktuellen Wert von e,
2. $advance(e)$: diese Prozedur erhöht den Wert von e um 1 (unteilbar!), und
3. $await(e, v)$: diese Prozedur wartet, bis der Wert von $e \geq v$ ist.

Ereigniszähler können also insbesondere nur erhöht, aber nie erniedrigt werden. Sie sind mit 0 initialisiert.
Beschreiben Sie eine Lösung des in der vorhergehenden Aufgabe beschriebenen Erzeuger-Verbraucher-Problems mit Hilfe von Ereigniszählern als Synchronisationsmittel (die *read*-Operation wird hierfür nicht benötigt) und begründen Sie die Korrektheit.

Aufgabe 3

Zeigen Sie, wie binäre Semaphore allgemein mit Ereigniszählern simuliert werden können.

Aufgabe 4

Siehe Rückseite!

Betrachten Sie die folgenden, in C gegebenen Vereinbarungen:

```
#define N      2                            /* Anzahl der Prozesse */

int  turn;                                  /* Wer ist an der Reihe? */
int  interested[N];                         /* Werte sind mit 0 initialisiert */

void enter_critical_region (int process) /* process (0 oder 1) ist die Nummer
                                            des aufrufenden Prozesses */
{
     int other;                             /* Nummer des anderen Prozesses */
     other = 1-process;

     interested[process] = TRUE;            /* Prozess zeigt Absicht an */
     turn = process;                        /* Prozess setzt Zeiger */

     while (turn==process && interested[other]==TRUE)
      ;                                     /* leere Anweisung */
}

void leave_critical_region (int process) /* process (0 oder 1) ist die Nummer
                                            des aufrufenden Prozesses */
{
     interested[process] = FALSE
}
```

Zeigen Sie, daß diese Routinen das Problem des wechselseitigen Ausschlusses korrekt lösen.

Lese-Aufgabe 5

Informieren Sie sich über Pipes und Umleitung der Ein- und Ausgabe (stdin und stdout, etc.) in der Shell (csh, ksh, o.ä.).