Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik

Prof. Dr. W.-P. de Roever

Dipl. Inf. K. Baukus

Informatik III

Wintersemester 2002/03 — **Serie 1** — 21. Oktober 2002

Thema : Gegenseitiger Ausschluß

Ausgabetermin: 21. Oktober 2002

Abgabe: 28. Oktober 2002

Bei nebenläufig, beziehungsweise auf einem Monoprozessor quasi-nebenläufig, laufenden Prozessen besteht die Gefahr, daß der gleichzeitige Zugriff auf gemeinsame Systemresourcen zu Fehlern führt. Beispiele sind schreibender Zugriff auf Variablen oder allgemein Speicherbereiche, Zugriff auf den Systembus, Benutzung des Modems, und vieles anderes mehr. Das Betriebssystem (evtl. mit Unterstützung der Hardware) muß Mechanismen zur Verfügung stellen, die die *exklusive* Ausführung bestimmter Kodesegmente garantieren, also ohne Interferenz anderer Prozesse. Derartige Programmteile heißen *kritische Abschnitte* und die Aufgabe ist, sicherzustellen, daß sich zu jedem Zeitpunkt höchstens ein Prozeß in seinem kritischen Abschnitt befindet.[1] Dies nennt man das Problem des *gegenseitigen Ausschlusses* (*mutual exclusion*). Es handelt sich um eine klassische Aufgabe eines jeden Betriebssystems und zu seiner Lösung gibt es eine Vielzahl unterschiedlicher Vorschläge.

Dieser Übungszettel beschäftigt sich mit verschiedenen Lösungen allein mit *Software.*

Aufgabe 1:

Die Aufgabe bestehe darin, eine einfache Lösung für gegenseitigen Ausschluß zu versuchen. Der Einfachheit halber werden nur zwei Prozesse betrachtet, gegeben in einem einfachen Pseudocode (s.u.). Die beiden Prozesse `P0` und `P1` sollen dabei (quasi)-nebenläufig ausgeführt werden. Wir zeigen dabei nur einen Prozeß, `P1` ist analog.

Der gegenseitige Ausschluß der kritischen Abschnitte (`critical_section`) soll mittels *einer globalen Variable* `turn` gewährleistet werden, auf die beide Prozesse lesend und schreibend zugreifen können.

Verhindern Sie den gleichzeitigen Zugriff auf die kritischen Abschnitte der beiden Prozesse mittels `turn`, und zwar in nicht-trivialer Weise.[2] Füllen sie also die Auslassungen

[1] Um Mißverständnisse zu vermeiden: mit „gleichzeitig“ und „nebenläufig“ ist hier stets Quasigleichzeitigkeit gemeint, also die Illusion der Gleichzeitigkeit, die das Betriebssystem durch Prozeßscheduling aufrecht erhält. Daß auf einem Monoprozessor nicht wirklich zwei Aktionen zur identischen physikalischen Zeit ausgeführt können, ändert am Problem nichts.

[2] Eine triviale „Lösung“ des Problems besteht darin, daß einer der beiden Prozesse niemals seinen kritischen Abschnitt ausführen darf, oder, noch radikaler, kein Prozeß darf jemals etwas Kritisches tun.

„...“ geeignet aus. Diskutieren Sie, ob Ihre Lösung im Rahmen eines Betriebssystems akzeptabel ist.

```
 ...                          /* Deklaration von turn */

process P0 =
begin
  while true                  /* Endlosschleife      */
  do
    some_actions;             /* unkritischer  Teil  */
     ...
     critical_section_0  ;    /* kritischer Abschnitt */
     ...
  od;
end;
```

Aufgabe 2:

Diese Aufgabe macht einen weiteren Versuch, das Problem des gegenseitigen Ausschlusses anzugehen, etwas komplizierter als in der Aufgabe zuvor. Die Idee ist es, diesmal *zwei* globale Variablen zur Synchronisation beider Prozesse zu verwenden, anstelle nur einer. Wir zeigen wieder nur `P0`, Prozeß `P1` ist analog. Die boolesche Variable `C0` soll dazu dienen, festzuhalten, ob sich der Prozeß `P0` in seinem kritischen Abschnitt befindet oder nicht; symmetrisch die Variable `C1` für den Prozeß `P1`. Da sich beide Prozesse zu Beginn außerhalb ihres jeweiligen kritischen Abschnittes befinden, werden beide Variablen mit `false` vorbesetzt.

```
process P0 =
begin

  while true                        /* Endlosschleife      */
  do
    some_actions;                   /* unkritischer  Code  */
    {while  ...     do skip od;}    /* Warten              */
     ...
     critical_section_0  ;          /* kritischer Abschnitt */
     ...
  od;
end;
```

In dem Programmfragment wartet `P0`. Versuchen Sie eine Lösung, indem sie die drei offenen Stellen „...“ einfüllen, wobei sie die Variablen `C0` und `C1` geeignet — lesend und schreibend — verwenden. Verhindert ihre Lösung das gleichzeitige Betreten der beiden Abschnitte? Wenn ja, warum? Wenn nein, warum nicht?

Aufgabe 3:

Die nächste Versuch zur Lösung des Problems verwendet wie die vorangegangene Aufgabe zwei Variablen zur Synchronisation. Sie dienen dazu, nicht den Aufenthalt eines Prozesses im kritischen Abschnitt, sondern seine *Absicht,* ihn zu betreten, anzuzeigen:

```
boolean  P0_wants_to_enter, P1_wants_to_enter = false;

process P0 =
begin

  while true                                  /* Endlosschleife     */
  do
    some_actions;                             /* unkritischer Code  */

    P0_wants_to_enter := true;
    {while P1_wants_to_enter do skip od;}     /* Warten             */
     critical_section_0 ;
    P0_wants_to_enter := false;
  od;
end;
```

Garantiert diese Implementierung die exklusive Ausführung der kritischen Abschnitte? Begründen Sie Ihre Antwort. Unabhängig davon: das Programm kann sich in einer anderen Weise unerwünscht verhalten. Was ist falsch mit dem Programm?