Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Prof. Dr. W.-P. de Roever
Dipl. Inf. K. Baukus · Dipl. Inf. K. Stahl

Informatik III

Wintersemester 2001/02

Serie 4

12. November 2001

Thema : Scheduling und Dekker

Ausgabetermin: 12. November 2001

Abgabe: 19. November 2001, bis 11:15 Uhr im Schrein

Aufgabe 1: [Scheduling (4 Punkte)]

Unter *Scheduling* verteht man ganz allgemein die Festlegung der Ausführungsreihenfolge verschiedener Jobs/Tasks/Prozesse oder Ähnlichem. Unter *nicht-unterbrechendem* (*non-preemptive*) Scheduling versteht man Strategien, bei denen ein Job, einmal „gescheduled“, bis zu Ende arbeiten muß und nicht unterbrochen werden kann.[1]

Betrachten Sie folgendes Problem des *nicht-unterbrechenden* Scheduling. Gegeben seien Jobs mit bekannter Bearbeitungsdauer von

4, 7, 9, 4 und x

Zeiteinheiten. Welches Scheduling minimiert die mittlere Zeit, bis ein Job terminiert ist (Minimierung der mittleren *response time*)? Formulieren Sie eine allgemeine Strategie, die die mittlere *response time* minimiert, und zeigen Sie deren Korrektheit. Diskutieren Sie mögliche Anwendungen in Betriebssystemen.

[1] Das *Prozeß*scheduling in Timesharingsystemen wie in Xinu ist selbstverständlich *unterbrechend* (*preemptive*).

Aufgabe 2: [Dekker (6 Punkte)]

Untersuchen Sie den angegebenen Algorithmus auf folgende Eigenschaften:

1. Gewährleistung des gegenseitigen Ausschlusses.
2. Freiheit von Verklemmung (oder *deadlock*-Freiheit).
3. Kann die Terminierung („Absturz“) eines Prozesses in seinem nicht-kritischen Abschnitt den Partnerprozeß negativ beeinflussen?
4. Verhält sich die Lösung gutwillig, wenn ein Prozeß wesentlich öfter als der andere in seinen kritischen Abschnitt will?
5. Kann es passieren, daß ein Prozeß, obwohl er seinen kritischen Abschnitt betreten will, unendlich lange darauf wartet?

Beweisen Sie jeweils Ihre Antwort.

```
// Dekkers Algorithmus f"ur Gegenseitigen Ausschlu"s
// 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
       if    favored = 1
       then  P0_wants_to_enter := false;
             while favored = 1 do skip od;
             P0_wants_to_enter := true
       fi
     od;
      critical_section_0 ;
     favored := 1;
     P0_wants_to_enter := false;
   od;
 end;
```