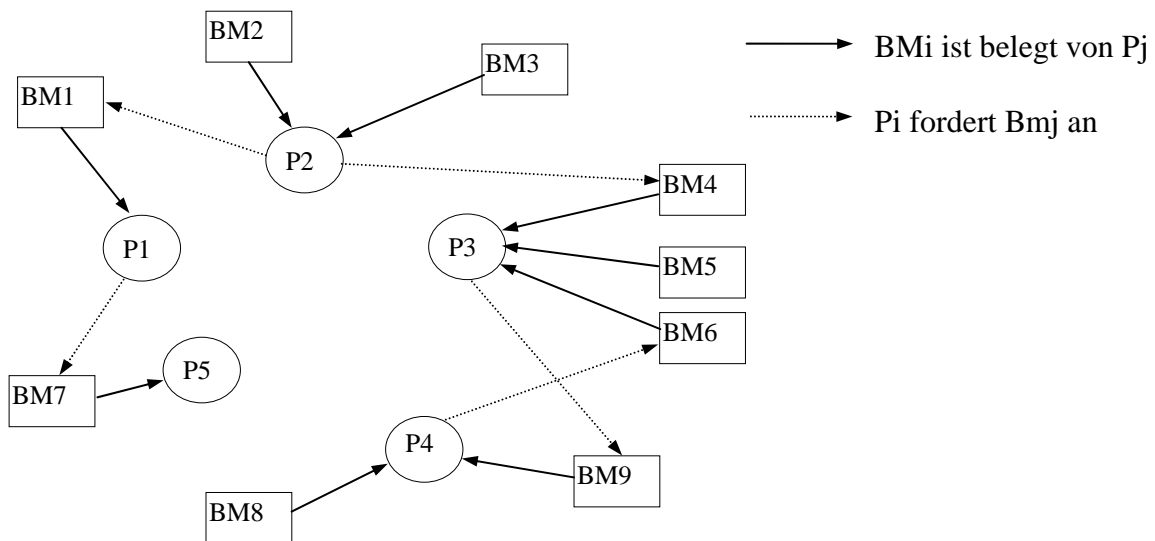


Aufgaben zum Thema „Verklemmungen“

V1. Untersuchen Sie das folgende Prozeßsystem auf das Auftreten von Deadlocks (s1, s2, s3: binäre Semaphore, mit true initialisiert): 95/5

<i>Prozeß 1</i>	<i>Prozeß 2</i>	<i>Prozeß 3</i>
P(s1);	P(s2);	P(s3);
P(s3);	/*working*/	P(s2);
/*working*/	V(s2);	P(s1);
V(s3);		/*working*/
V(s1);		V(s3);
		V(s2);
		V(s1);

V2. Gegeben sei folgender Prozeß-Betriebsmittel-Graph:



Gibt es in diesem Fall eine Menge von Prozessen, die sich im Deadlock befinden können? Wenn ja, wie lautet diese Menge, und unter welchen weiteren Voraussetzungen tritt ein Deadlock tatsächlich auf?

Erläutern Sie eine Möglichkeit, Deadlocks entgegenzuwirken! 96/8

V3. Kann sich ein System in einem Zustand befinden, der weder ein Deadlock-Zustand noch ein sicherer Zustand ist? Wenn ja, geben Sie ein Beispiel an; wenn nein, bewiesen Sie dies! 97/2

V4. Konstruieren Sie (in Pseudocode) unter Verwendung von Semaphoren ein Prozeßsystem, in dem es zum Auftreten eines Deadlocks kommen kann, aber nicht muß! Begründen Sie Ihre Lösung! Erläutern Sie anhand Ihres Beispiels die Bedingungen, die für das Auftreten von Deadlocks erfüllt sein müssen! 98/7

V5*. Was versteht man im Zusammenhang mit Verklemmungen unter einem sicheren Zustand? Geben Sie ein Beispiel mit maximal drei (möglichst nur zwei) Prozessen dafür an, daß sich ein System in einem Zustand befinden kann, der weder ein Verklemmungszustand noch ein sicherer Zustand ist! Wie kann man vorgehen, um dieser Situation vorzubeugen, d.h. nur sichere Zustände zuzulassen? Worin bestehen die Nachteile dieses Vorgehens? 99/2

V6.

- Unter welchen Bedingungen kommt es in einem System paralleler Prozesse allgemein zu einer Verklemmung (Deadlock)?
- Gegeben sei folgendes System paralleler Prozesse (s1, s2, s3: binäre Semaphore, mit 1 bzw. true initialisiert):

(Prozeß 1)	(Prozeß 2)	(Prozeß 3)
P(s2);	P(s1);	P(s3);
P(s3);	P(s2);	P(s2);
{working}	{working}	{working}
V(s3);	V(s2);	V(s2);
V(s2);	V(s1);	V(s3);

Zeigen Sie, daß in diesem Beispiel eine Verklemmung eintreten kann, aber nicht muß! Verwenden Sie zur Begründung auch einen Prozeß-Betriebsmittel-Graphen! 00/8

V7. Die Software eines Bankhauses stelle einen Dienst

```
umbuchen (int betrag, int kto_A, int kto_B)
```

bereit, der das Umbuchen eines Betrags von einem Konto A auf ein Konto B dieses Bankhauses gestattet (als Kontobezeichnungen werden nur natürliche Zahlen verwendet) und der von verschiedenen parallelen Prozessen gleichzeitig genutzt werden kann. Die Implementation habe folgende Struktur (sem_A, sem_B: dem jeweiligen Konto zugeordnete binäre Semaphore):

```
P(sem_A);
P(sem_B);
/*Aktualisieren der beiden Kontostände*/
V(sem_B);
V(sem_A);
```

- Welches Problem wird durch die Verwendung der Semaphore gelöst? (Es werde angenommen, daß eine Umbuchung stets ausgeführt werden kann.)
- Welches andere Problem kann durch die Verwendung der Semaphore in der angegebenen Weise auftreten? Wie lauten die Bedingungen, unter denen es eintritt, und inwieweit sind diese Bedingungen erfüllt bzw. erfüllbar?
- Geben Sie – unter Beibehaltung der Verwendung von Semaphoren – eine Implementation des Dienstes an, die dieses Problem löst! Begründen Sie kurz Ihre Lösung! 01/2

V8. In einem System paralleler Prozesse gebe es vier Prozesse P1,...,P4, die vier Dateien A,...,D in folgender Weise benutzen:

P1 liest von A und B und schreibt in C

P2 liest von A und schreibt in D

P3 liest von C und schreibt in D

P4 liest von C und schreibt in B und D.

Die Prozesse müssen die von ihnen benötigten Dateien einzeln öffnen, wobei zuerst die zu lesenden Dateien geöffnet werden. Versucht ein Prozeß eine bereits geöffnete Datei zu öffnen, so wird er blockiert. Ein Prozeß schließt alle von ihm benutzten Dateien erst, nachdem er seine gesamte Schreibaktivität beendet hat.

- Zu welchem Problem kann die angegebene Prozeßstruktur führen? Verwenden Sie zur Begründung auch einen Prozeß-Betriebsmittel-Graphen!
- Welche Konsequenzen hat es, wenn P4 von C liest, aber nur in D schreibt?
- Welches Problem ergibt sich, wenn eine Datei von mehreren Prozessen gleichzeitig geöffnet werden kann? 01/8

V9. Ein Prozeßsystem bestehe aus vier Prozessen P_1, \dots, P_4 und vier Betriebsmitteln A, B, C, D. Dabei gebe es von A, B und D je ein Exemplar und von C mehrere Exemplare. Das Betriebssystem teilt ein freies Betriebsmittel-Exemplar nur *einem* Prozeß zu. Außerdem erfolgt die Zuteilung eines Exemplars nur, nachdem es ein Prozeß freigegeben hat (abgesehen vom Anfangszustand). Ein Prozeß gibt erst dann seine Betriebsmittel-Exemplare frei, wenn er alle angeforderten Exemplare auch erhalten hat; sofern möglich, erhält er alle angeforderten Exemplare auf einmal, sonst gar keine. Betrachtet werde folgender Zustand:

- P_1 besitzt nichts
- P_2 besitzt 1 Exemplar von C
- P_3 besitzt je 1 Exemplar von A und B
- P_4 besitzt je 1 Exemplar von C und D.

Jeder der Prozesse P_1, P_2 und P_4 fordert nun je ein Exemplar von A und C an (in einer gemeinsamen Anforderung), P_3 stellt keine weiteren Forderungen. Welches ist die Mindestanzahl n von Exemplaren des Betriebsmittels C, so daß keine Verklemmung auftritt? Begründen Sie, daß es bei $n - 1$ Exemplaren zwangsläufig zu einer Verklemmung kommt; verwenden Sie dazu auch einen Prozeß-Betriebsmittel-Graphen! 02/2

V10. Drei Philosophen sitzen an einem runden Tisch, jeder vor einem Teller mit Spaghetti. Zwischen den Tellern liegt jeweils eine Gabel (insgesamt also drei Gabeln). Zum Essen benötigt ein Philosoph die linke und die rechte Gabel. Werden die Gabeln durch binäre Semaphore `gabel[i]` und die Philosophen durch Prozesse `philosoph(int i)` ($i = 0,1,2$) modelliert, so läßt sich das Verhalten der Philosophen folgendermaßen beschreiben:

```
philosoph(int i) {
    denken();
    down(gabel[i]);
    down(gabel[(i+1)%3]);
    essen();
    up(gabel[(i+1)%3]);
    up(gabel[i]);
}
```

(`gabel[i]` ist dabei die rechte Gabel des Philosophen i ; `down` und `up` bezeichnen die entsprechenden Semaphor-Operationen P und V).

- a) Welches Problem steckt in diesem Verhalten?
- b) Welche Konsequenzen hat es, wenn unter den Philosophen ein oder mehrere Linkshänder sind, die also zuerst nach der linken Gabel greifen und entsprechend zuerst die rechte Gabel ablegen?
- c) Welche (positiven und negativen) Konsequenzen hat es, wenn in der Mitte des Tisches eine vierte Gabel liegt, die in der Ausgangssituation (nur Rechtshänder) folgendermaßen benutzt wird: Hat ein Philosoph die rechte Gabel erhalten, so prüft er zunächst, ob die linke Gabel frei ist; wenn ja, ergreift er sie, wenn nein, verzichtet er auf die linke Gabel und bemüht sich um die in der Mitte liegende Gabel. Diese Gabel legt er auch als erste wieder ab.
- d) Welche Konsequenzen hat es, wenn die nicht benutzten Gabeln nicht zwischen den Tellern liegen, sondern unsortiert in der Mitte des Tisches?

Alle zu den Semaphoren gehörenden Warteschlangen werden nach FIFO geleert. 02/11

V11. Gegeben sei folgendes System paralleler Prozesse (s_1, \dots, s_4 : binäre Semaphore, mit true bzw. 1 initialisiert):

<i>Prozeß 1</i>	<i>Prozeß 2</i>	<i>Prozeß 3</i>
P(s3);	P(s4);	P(s2);
P(s1);	P(s1);	P(s1);
P(s2);	P(s3);	<i>/*working*/</i>
<i>/*working*/</i>	<i>/*working*/</i>	V(s1);
V(s2);	V(s3);	V(s2);
V(s1);	V(s1);	
V(s3);	V(s4);	

Betrachtet werde der Zustand S, in dem jeder Prozeß seine zweite P-Operation begonnen hat.

- a) Geben Sie den dazu gehörigen Prozeß-Betriebsmittel-Graphen an!
- b) Zeigen Sie, daß es – jeweils vom Zustand S ausgehend – eine Fortsetzung des Prozeßablaufs gibt, bei der alle Prozesse beendet werden können, daß aber auch ein Deadlock auftreten kann!
- c) Modifizieren Sie die Aufgabe unter Beibehaltung aller Semaphor-Operationen so, daß Deadlocks ausgeschlossen sind!

03/8

V12. Bekanntlich ist die nachstehend angegebene Lösung des Philosophenproblems für fünf Philosophen nicht verklemmungsfrei.

```

void philospher(int i) {
    while (1) {
        think();
        take_fork(i);
        take_fork((i+1)%5);
        eat();
        put_fork((i+1)%5);
        put_fork(i);
    }
}

```

(Implementation von `take_fork(i)` mittels Semaphoren als $P(s[i])$, damit Zurücklegen von Gabeln erst nach Essen möglich).

Stellt man sich vor, daß die Gabeln im Uhrzeigersinn numeriert sind, so handelt es sich bei den Philosophen ausschließlich um Rechtshänder. Angenommen, es sind auch Linkshänder unter den Philosophen (die die Gabeln also in umgekehrter Reihenfolge aufnehmen und auch wieder ablegen). Bei welcher Anzahl von Linkshändern ist das System verklemmungsfrei? Welche Auswirkung hat die Sitzordnung? Begründungen!

04/8

V13. An ein Mehrprozessorsystem mit vier Prozessoren sind acht (nur exklusiv nutzbare) Bandlaufwerke angeschlossen. Das System muß Prozesse mit folgender Struktur ausführen: Jeder Prozeß benötigt von Beginn an zwei Bandlaufwerke und gegen Ende für eine kurze Zeit zusätzlich ein drittes Laufwerk. Die Anzahl dieser Prozesse werde als unbegrenzt angesehen (d.h., sobald ein Prozeß beendet ist, kann ein nächster Prozeß zur Bearbeitung ausgewählt werden). Auf jedem Prozessor kann nur ein (1) derartiger Prozeß laufen. Der Scheduler beginnt erst dann einen Prozeß, wenn drei Bandlaufwerke verfügbar sind, und teilt diese drei Laufwerke dem Prozeß während seiner gesamten Laufzeit fest zu.

- a) Was ist der Grund für diese letzte Bedingung? Welchen erheblichen Nachteil hat sie?

- b) Wie viele Prozesse können maximal gleichzeitig laufen? Wie groß ist die minimale und die maximale Anzahl von Bandlaufwerken, die aufgrund dieser Vorgehensweise im Leerlauf sind?
- c) Welche bessere Zuteilungsstrategie der Bandlaufwerke könnte man wählen? Wie viele Prozesse können dann maximal laufen?
- d) Geben Sie (in C-Code) die Implementation eines Prozesses unter Verwendung von Semaphoren an, so daß die eingangs beschriebene Zuteilungsstrategie durchgesetzt wird! 05/2

V14. Gegeben sei ein System paralleler Prozesse (Threads) A,...,D, die drei binäre Semaphore s_1, \dots, s_3 in folgender Weise gemeinsam benutzen:

<i>Prozeß A</i>	<i>Prozeß B</i>	<i>Prozeß C</i>	<i>Prozeß D</i>
P(s1);	P(s3);	P(s3);	P(s2);
P(s2);	P(s1);	/*working*/	P(s3);
/*working*/	/*working*/	V(s3);	/*working*/
V(s2);	V(s3);		V(s2);
V(s1);	V(s1);		V(s3);

Dieses System ist nicht deadlockfrei.

- a) Geben Sie einen Systemzustand (Stand der Bearbeitung der einzelnen Prozesse) an, bei dem sich alle vier Prozesse in einem Deadlock befinden!
- b) Weisen Sie nach, daß in diesem Zustand tatsächlich ein Deadlock vorliegt! Verwenden Sie zur Begründung auch einen Prozeß-Betriebsmittel-Graphen! 05/8

V15. Ein Betriebssystem habe eine einzige Betriebsmittelart zu verwalten, die in mehreren identischen Exemplaren (BME) vorliegt; dabei kann ein einzelnes Exemplar nur exklusiv genutzt werden. Anforderung, Zuteilung und Freigabe der BME geschehen nach folgenden Regeln:

- Ein Prozeß muß bei seinem Start angeben, wie viele BME er im Laufe seiner Existenz maximal gleichzeitig benötigt.
- Ein Prozeß kann mehrere BME gleichzeitig anfordern; er kann jederzeit beliebig viele seiner ihm zugeteilten BME freigeben.
- Überschreitet ein Prozeß bei einer Anforderung mit der Summe der angeforderten und bereits zugeteilten BME den bei Programmstart vereinbarten Maximalwert, so wird er abgebrochen.
- Sind alle angeforderten BME verfügbar, so werden sie dem Prozeß zugeteilt. Andernfalls erhält der Prozeß momentan kein einziges BME und wird blockiert.
- Spätestens am Ende gibt ein Prozeß alle BME frei; beim Abbruch werden sie ihm entzogen.

- a) Welchen Vorteil hat diese Vorgehensweise im Vergleich zu einer bekannten ähnlichen Strategie? Worin liegt ihr entscheidender Nachteil?
- b) Das System verfüge über 12 BME. Vier Prozesse A,...,D wollen maximal 4, 5, 5 bzw. 2 BME gleichzeitig nutzen. Gegenwärtig sind den Prozessen 2, 4, 2 bzw. 1 BME zugeteilt. Betrachten Sie in dieser Situation den Fall, daß zwei BME von Prozeß A angefordert werden bzw. alternativ von Prozeß B bzw. von Prozeß C. Entscheiden Sie in jedem der drei Fälle, ob die BME dem jeweiligen Prozeß zugeteilt werden, und beschreiben Sie die Konsequenzen für das Prozeßsystem! 06/3

V16. Gegeben sei ein System mit vier Prozessen P_1, \dots, P_4 , die acht Betriebsmittel A,...,H nutzen können. Fordert ein Prozeß ein verfügbares Betriebsmittel an, so wird es ihm zugeteilt, an-

denfalls wird der Prozeß blockiert. Einzeln stellen die Prozesse nacheinander die folgenden Forderungen, ohne zwischendurch Betriebsmittel freizugeben:

P_1 : A B C P_2 : E B D P_3 : F B P_4 : C G H F.

In diesem System kann es zu einer Verklemmung kommen.

- Begründen Sie dies! Verwenden Sie dazu auch einen Prozeß-Betriebsmittel-Graphen!
- Eine der Ursachen für Verklemmungen ist die Unmöglichkeit, mehrere Betriebsmittel atomar anzufordern. Im vorliegenden Fall kann dies jedoch durch eine einfache Modifikation erreicht werden, indem mehrere Anforderungen unter Verwendung von Semaphoren „geklammert“ werden. Beschreiben Sie diese Modifikation! Dabei soll die aus der „Klammerung“ folgende Einschränkung der Parallelität so gering wie möglich sein (die vollständige Sequentialisierung aller vier Prozesse ist beispielsweise keine zugelassene Lösung).
- Mit welchem Fachbegriff des Gebiets Betriebssysteme bzw. der Theorie paralleler Prozesse wird die „Klammerung“ bezeichnet? 07/2

V17. In einem System paralleler Prozesse nutzen vier Prozesse P_1, \dots, P_4 mehrere verschiedenartige Betriebsmittel A, ..., H. Ausgangspunkt sei folgender Systemzustand:

P_1 hat B, C und fordert A, H an
 P_2 hat D, H und fordert E an
 P_3 hat A, F und fordert E an
 P_4 hat E, F, G.

- Zeichnen Sie den zugehörigen Prozeß-Betriebsmittel-Graphen (ohne sich kreuzende Kanten!).
- Gibt es Prozesse, die sich in einer Verklemmung befinden? (Begründung!)
- Betrachten Sie – in beiden Fällen von diesem Zustand ausgehend – die Prozeßaktionen
 - einerseits fordert P_2 das Betriebsmittel E an
 - andererseits fordert P_4 das Betriebsmittel C an.

Diskutieren Sie die jeweils entstehende Situation unter dem Gesichtswinkel „Verklemmungen“! 08/2, 07/8

V18. Ein Betriebssystem stelle eine Funktion `anfordern(n)` bereit, mit der ein Prozeß atomar n Exemplare eines in mehreren Exemplaren nutzbaren Betriebsmittels anfordern kann; ein einzelnes Exemplar kann dabei nur exklusiv benutzt werden. Sind alle angeforderten Exemplare verfügbar, so werden sie zugeteilt, andernfalls wird der Prozeß blockiert. Entsprechend kann ein Prozeß mit `freigeben(n)` mehrere Betriebsmittel-Exemplare gleichzeitig freigeben.

In dem System sei das Betriebsmittel in drei Exemplaren vorhanden. Weiterhin mögen in dem System die folgenden zwei parallelen Prozesse existieren:

```

P1:  { anfordern(1);
      /* working */
      anfordern(1);
      /* working */
      freigeben(2);
      }
P2:  { anfordern(3);
      /* working */
      freigeben(3);
      }
  
```

- Ist das System verklemmungsfrei?
- Welche Konsequenzen hat es, wenn es statt der Funktion `anfordern(n)` nur eine Funktion `anfordern` zum Anfordern eines einzelnen Betriebsmittel-Exemplars gibt? 09/8