

Klausur zur Vorlesung "Grundlagen Betriebssysteme und Systemsoftware"

(Prof. Dr. J. Schlichter, Dr. G. Groh, WS 2007 / 2008)

- Datum: Freitag, 8.2.2008, 16-18 Uhr
- Orte: Hörsaal 1 (Gebäude Maschinenwesen, Garching) und Hörsaal 1 (Gebäude Mathematik-Informatik, Garching)
- Bearbeitungszeit: 90 Minuten
- Zugelassene Hilfsmittel: Keine
- Bitte beschriften Sie ZUERST dieses Deckblatt mit Vorname, Nachname, Studiengang und Matrikelnummer!
- DANN beschriften Sie bitte das karierte Papier!
- Bitte kennzeichnen Sie auf dem karierten Papier vor der Abgabe eindeutig, was bewertet werden soll und wo es sich um freie Noitzen handelt!
- Viel Erfolg!

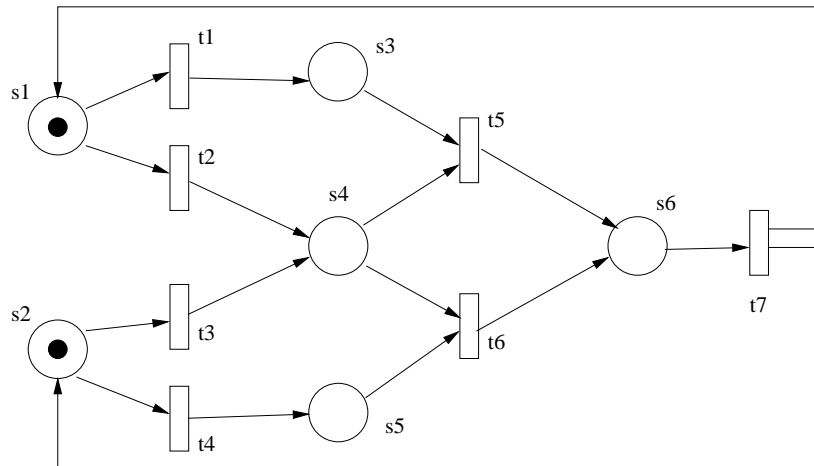
Vorname	Nachname
Matrikelnummer	Studiengang

Mitteilungen vom Studierenden an die Korrigierenden:

A1	A2	A3	A4	A5	Σ		Unterschrift Erstkorrektor
A1	A2	A3	A4	A5	Σ	Note	Unterschrift Zweitkorrektor

1 Aufgabe (Petri-Netze) (8 P)

Gegeben sei das boolsche Petri-Netz

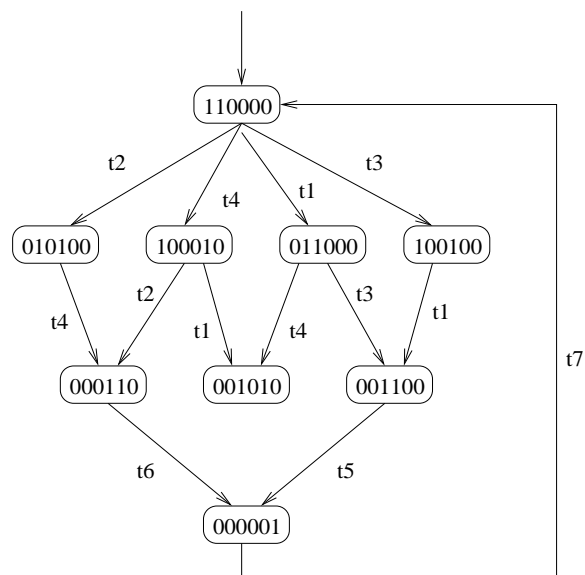


mit der gezeigten Anfangsbelegung.

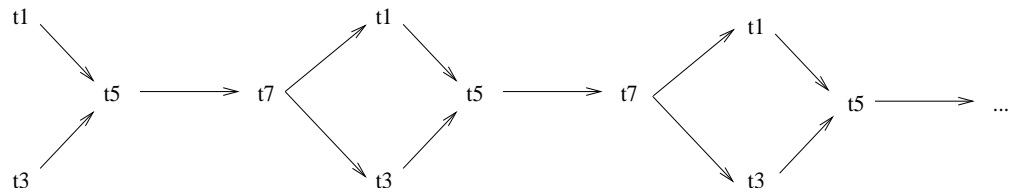
1. Geben Sie den Erreichbarkeitsgraphen an! (4 P)
2. Ist eine Verklemmung erreichbar? (Begründen Sie in einem Satz!) (2 P)
3. Es existieren unendliche Abläufe. Geben Sie ein Beispiel durch Angabe einer entsprechenden Folge von Transitionen! (2 P)

Lösungsvorschlag:

1. Erreichbarkeitsgraph:



2. Ja. Die Belegung 001010 entspricht einer Verklemmung, da keine Transition mehr schalten kann.
3. Folgender unendlicher Ablauf existiert:



2 Aufgabe (Deadlock-Verhinderung: Banker's Algorithmus) (8 P)

Wir gehen aus von 3 Prozessen, die um 3 Sorten von Ressourcen (Sorte 1: DVD-Brenner, Sorte 2: Grafikkarten, Sorte 3: TV-Karten) im System konkurrieren. Es seien die Zahlen der existierenden Instanzen der verschiedenen Ressourcen-Sorten durch das Tupel (e_1, e_2, e_3) gegeben und die Zahlen der im Moment noch freien Instanzen der Ressourcen-Sorten durch das Tupel (a_1, a_2, a_3) . Die aktuelle Belegung von Instanzen der verschiedenen Ressourcen-Sorten durch Prozesse sei durch die 3×3 Matrix c und die aktuelle maximale Rest-Anforderung von Instanzen der Ressourcen-Sorten durch Prozesse durch die 3×3 Matrix r gegeben:

$$c = \begin{pmatrix} 3 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 1 & 2 \end{pmatrix}$$

$$r = \begin{pmatrix} 3 & 2 & 0 \\ 0 & y & 2 \\ 0 & 0 & 2 \end{pmatrix}$$

$$e = (7, 4, 6)$$

$$a = (2, 1, 2)$$

In den Matrizen c und r seien die Zeilen den Prozessen und die Spalten den Ressourcen-Sorten zugeordnet. (Beispiel: Dem Prozess 3 entspricht in den Matrizen c und r jeweils die dritte Zeile. Das heißt, Prozess 3 hat im Moment 2 Instanzen der Ressourcen-Sorte 1 (also 2 DVD-Brenner), eine Instanz der Ressourcen-Sorte 2 (also eine Grafikkarte) und 0 (**Fehler: hier müsste es korrekterweise 2 heißen!**) Instanzen der Ressourcen-Sorte 3 (also 0 TV-Karten). Prozess 3 möchte in Zukunft bspw. maximal noch 0 DVD Brenner, 0 Grafikkarten und zwei TV-Karten haben.)

1. Abgesehen von safe / unsafe: Für welche Werte von y sind diese Angaben überhaupt sinnvoll? Nehmen Sie an, dass jeder Prozess "seine" Zeile in der Matrix c und das Tupel (e_1, e_2, e_3) kennt. Begründen Sie Ihre Antwort! (1 P)
2. Sollte im aktuellen Zustand für $y = 2$ eine Anforderung einer Instanz der Ressource r_1 durch den Prozess p_1 vom Betriebssystem erfüllt werden? Beantworten Sie die Frage durch Anwendung des Banker-Algorithmus! Stellen Sie alle Ihre Ausführungsschritte so dar, dass die Ausführung des Algorithmus nachzuvollziehen ist! (4 P)
3. Für welche der gemäß Teilaufgabe 1) sinnvollen y ist der oben angegebene Zustand selbst safe? Begründen Sie Ihre Antwort, indem Sie für jeden gemäß Teilaufgabe 1) sinnvollen Wert von y entweder eine mögliche Ausführungsreihenfolge der Prozesse angeben (safe Fall) oder angeben, dass der Zustand unsafe ist ! (3 P)

Lösungsvorschlag:

1. Da sinnvollerweise gelten muss

$$\forall i, j : c_{ij} + r_{ij} \leq e_j$$

(Kein Prozess sollte mehr Ressourcen einer Sorte anfordern als "aus seiner Sicht allein" noch übrig sind, d.h. nicht mehr Ressourcen dieser Sorte anfordern als sich aus der Differenz ihm bereits zugeteilter Ressourcen dieser Sorte und der insgesamt verfügbaren Zahl von Ressourcen dieser Sorte ergibt.)

Es ergibt sich für $i = 2$ und $j = 2$:

$$2 + y \leq 4 \quad (1)$$

also

$$y \leq 2 \quad (2)$$

also $y \in \{0, 1, 2\}$. (Werte: 0.5 P; Begründung: 0.5 P)

2. Um entscheiden zu können, ob im aktuellen Zustand für $y = 2$ eine Anforderung einer Instanz der Ressource r_1 durch den Prozess p_1 vom Betriebssystem erfüllt werden sollte, muss der Zustand der sich durch Erfüllung dieses Ressourcenwunsches ergibt

$$c = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 1 & 2 \end{pmatrix}$$

$$r = \begin{pmatrix} 2 & 2 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & 2 \end{pmatrix}$$

$$e = (7, 4, 6)$$

$$a = (1, 1, 2)$$

darauf hin untersucht werden, ob er safe ist. (1 P)

Ausführen des Banker Algorithmus auf diesem Zustand ergibt (3 P (pro Schritt 1 P)):

<p>Schritt 1:</p> $c = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ $r = \begin{pmatrix} 2 & 2 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & 0 \end{pmatrix}$ $e = (7, 4, 6)$ $a = (3, 2, 4)$ <p>Prozess p_3 ist terminiert.</p>	<p>Schritt 2:</p> $c = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ $r = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & 0 \end{pmatrix}$ $e = (7, 4, 6)$ $a = (7, 2, 6)$ <p>Prozess p_1 ist terminiert.</p>	<p>Schritt 3:</p> $c = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ $r = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ $e = (7, 4, 6)$ $a = (7, 4, 6)$ <p>Prozess p_2 ist terminiert.</p>
---	---	---

3. Für $y = 0$ ergibt das Ausführen des Banker Algorithmus die Reihenfolge (p_2, p_3, p_1) (auch möglich: (p_3, p_1, p_2) oder (p_3, p_2, p_1)) (der Zustand ist safe). (1 P)

Für $y = 1$ ergibt das Ausführen des Banker Algorithmus die Reihenfolge (p_2, p_3, p_1) (auch möglich: (p_3, p_1, p_2) oder (p_3, p_2, p_1)) (der Zustand ist safe). (1 P)

Für $y = 2$ ergibt das Ausführen des Banker Algorithmus die Reihenfolge (p_3, p_1, p_2) (auch möglich: (p_3, p_2, p_1)) (der Zustand ist safe). (1 P)

3 Aufgabe (Memory Management) (8 P)

1. Sei eine Menge von Seiten (pages) $N = \{0, 1, 2, 3, 4, 5, 6\}$ und eine Menge von Seitenrahmen (frames) $F = \{f_1, f_2, f_3\}$ gegeben. Nun wird in folgender Reihenfolge auf die Seiten zugegriffen:

$w = 6 \ 2 \ 5 \ 0 \ 4 \ 4 \ 6 \ 5 \ 1 \ 3$

Vollziehen Sie die Seitenersetzungs-Strategien LRU (Least Recently Used) und Second Chance schrittweise nach, indem Sie die entsprechenden folgenden leeren Tabellen ausfüllen!

Strategie LRU:

Anforderung	f_1	f_2	f_3	Page-Fault j/n
6				
2				
5				
0				
4				
0				
6				
5				
1				
3				

(2 P)

Strategie Second Chance:

Anforderung	f_1	f_2	f_3	Page-Fault j/n	FIFO		
					Pos 1	Pos 2	Pos 3
6					r	r	r
2					r	r	r
5					r	r	r
0					r	r	r
4					r	r	r
0					r	r	r
6					r	r	r
5					r	r	r
1					r	r	r
3					r	r	r

(3 P)

2. Gegeben sei ein virtueller Adressraum von 19 Bit und eine Seiten- und Kachelgröße von je 8192 Byte (8KB). Weiterhin sei eine Seiten-Kacheltabelle gegeben die u.a. folgende Einträge hat:

(- siehe umseitig -)

1	0101
2	0111
3	1000
4	0001
5	0100
6	0110
7	1010
8	1011
...	...

Ermitteln Sie zu der virtuellen Adresse

$$v = (0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1)$$

die physikalische Adresse! (3 P)

Lösungsvorschlag:

1. Strategie LRU:

Anforderung	f ₁	f ₂	f ₃	Page-Fault j/n
6	6			j
2	6	2		j
5	6	2	5	j
0	0	2	5	j
4	0	4	5	j
0	0	4	5	n
6	0	4	6	j
5	0	5	6	j
1	1	5	6	j
3	1	5	3	j

Da in der Aufgabenstellung eine Inkonsistenz zwischen der Angabe von w im Text und den vorgegebenen Werten von w in der Tabelle herrschte, wurden beide Varianten gleichermaßen gewertet. Hier die andere Variante:

Anforderung	f ₁	f ₂	f ₃	Page-Fault j/n
6	6			j
2	6	2		j
5	6	2	5	j
0	0	2	5	j
4	0	4	5	j
4	0	4	5	n
6	0	4	6	j
5	5	4	6	j
1	5	1	6	j
3	5	1	3	j

Strategie Second Chance:

Anforderung	f ₁	f ₂	f ₃	Page-Fault j/n	FIFO				
					Pos 1	Pos 2	Pos 3		
6	6			j	6				
2	6	2		j	2	6			
5	6	2	5	j	5	2	6		
0	0	2	5	j	0	5	2		
4	0	4	5	j	4	0	5		
0	0	4	5	n	4	0	1	5	
6	0	4	6	j	6	4	0	1	
5	0	5	6	j	5	0	6		
1	0	5	1	j	1	5	0		
3	3	5	1	j	3	1	5		

0	6	4	0
---	---	---	---

Da in der Aufgabenstellung eine Inkonsistenz zwischen der Angabe von w im Text und den vorgegebenen Werten von w in der Tabelle herrschte, wurden beide Varianten gleichermaßen gewertet. Hier die andere Variante:

Anforderung	f ₁	f ₂	f ₃	Page-Fault j/n	FIFO			
					Pos 1	Pos 2	Pos 3	
6	6			j	6			
2	6	2		j	2	6		
5	6	2	5	j	5	2	6	
0	0	2	5	j	0	5	2	
4	0	4	5	j	4	0	5	
4	0	4	5	n	4	0	1	5
6	0	4	6	j	6	4	0	1
5	5	4	6	j	5	6	4	1
1	5	4	1	j	1	4	5	0
3	3	4	1	j	3	1	4	0

4	5	6
---	---	---

2. $8K = 8192 = 2^{13}$. Daraus folgt: Wir benötigen $19 - 13 = 6$ Bit für die Seitenadressierung und 13 Bit für die Adressierung innerhalb der Seite. (1 P)

Die Seitenadresse (die ersten 6 Bit von v) lautet: $(000111) = 7$. Nachschauen in der Tabelle liefert für die Adresse 7 die Kacheladresse (1010) . (1 P)

Die restlichen 13 Bit von v lauten: (100000000111) . Damit lautet die physikalische Adresse: (1010100000000111) . (1 P)

4 Aufgabe (Quiz) (8 P)

1. Prozesskommunikation: Skizzieren Sie das Ablaufdiagramm für eine asynchrone Meldung und das Ablaufdiagramm für eine synchrone Meldung! Benennen Sie die grafischen Elemente aussagekräftig! (2 P)
2. Nennen und erläutern Sie kurz die 4 Deadlock-Bedingungen! (2 P)
3. Folgendes Code-Stück in C sei gegeben (mit Zeilennummern):

```
....
1  int arrayXYZ[10];
2  int i;
3  int *pi;
4  int horst;
5  int intVar;
6  for(i=0;i<10;i++)
7      arrayXYZ[i]=i;
8  horst = arrayXYZ[4];
9  //eins
10 printf("%i\n", horst);
11 pi = &arrayXYZ[7];
12 horst = *pi;
13 //zwei
14 printf("%i\n", horst);
15 pi = arrayXYZ;
16 horst = *pi;
17 //drei
18 printf("%i\n", horst);
19 pi = &horst;
20 *pi = *(arrayXYZ + 3);
21 //vier
22 printf("%i\n", horst);
....
```

Welche Werte hat die Variable `horst` an den durch die Kommentare `eins`, `zwei`, `drei` und `vier` bezeichneten Punkten im Programmablauf (bzw. was ist die Ausgabe des Programm-Stücks)? (Nur die vier Werte nennen!) (2 P)

4. Die Funktionen `P` und `V` auf Semaphoren `s` (vereinfacht durch `int *` dargestellt) seien durch folgenden C-artigen Pseudocode informell charakterisiert (Es wird als gegeben angenommen, dass die Operationen `P` und `V` atomar sind und wechselseitig ausgeschlossen ausgeführt werden.):

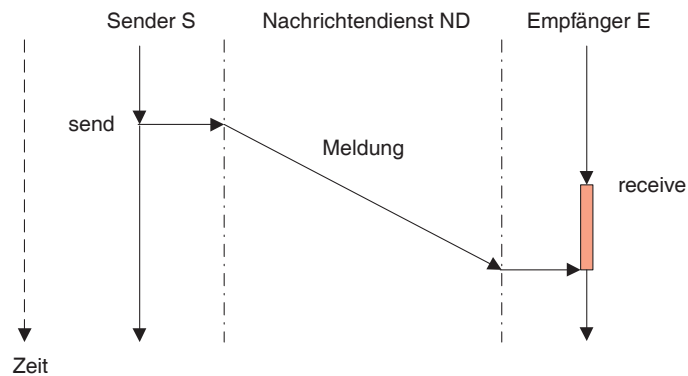
```
public void P (int *s) {
    *s = expr1 ;
    if ( expr2 ) { Prozess in die Menge der bezüglich *s
        wartenden Prozesse einreihen }
}

public void V (int *s) {
    *s = expr3 ;
    if ( expr4 ) { Führe genau einen der bezüglich *s wartenden
        Prozesse in den Zustand Rechenwillig über }
}
```

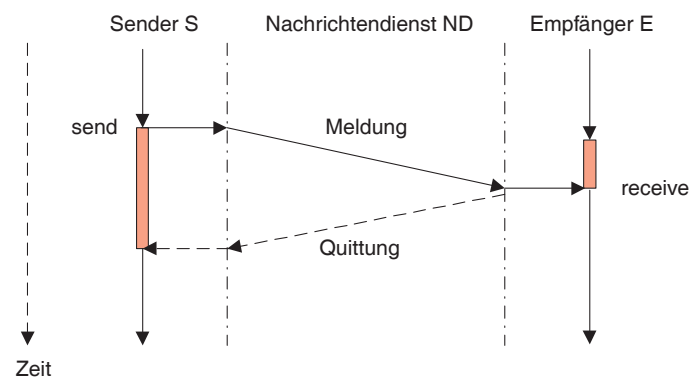
Ersetzen Sie die Platzhalter `expr1`, `expr2`, `expr3` und `expr4` durch geeignete Ausdrücke! (2 P)

Lösungsvorschlag:

1. asynchrone Meldung:



synchrone Meldung:



2. • **Exclusiveness Condition:** Die gemeinsam benutzbaren Ressourcen können nicht parallel genutzt werden, d.h. sie sind nur exklusiv benutzbar.
- **No Preemption Condition:** Die zugeteilten/belegten Ressourcen können nicht entzogen werden, d.h. die Nutzung ist nicht unterbrechbar.
- **Hold and Wait Condition:** Prozesse belegen die schon zugeteilten Ressourcen auch dann, wenn sie auf die Zuteilung weiterer Ressourcen warten, d.h. wenn sie weitere Ressourcen anfordern.
- **Circular Waiting Condition:** Es gibt eine zyklische Kette von Prozessen, von denen jeder mindestens eine Ressource belegt, die der nächste Prozess in der Kette benötigt, d.h. zirkuläre Wartebedingung.

3. Die Werte sind:

- eins: 4
- zwei: 7

- drei: 0
- vier: 3

4. Ersetze:

expr1 durch $*s - 1$

expr2 durch $*s < 0$

expr3 durch $*s + 1$

expr4 durch $*s \leq 0$

5 Aufgabe (Modellierung mit Semaphoren) (8 P)

Ein Möbelhändler hat eine Lagerhalle für Schränke und Couches. Beliefert wird er von einem Fabrikanten F, der bei jeder Lieferung einen Schrank und eine Couch bringt. Ein Ausfahrer S bringt je Fahrt einen Schrank zum Kunden, ein Ausfahrer C je Fahrt eine Couch. Zum Be- und Entladen muss eine Laderampe benutzt werden. Für sich alleine betrachtet laufen die beteiligten Prozesse folgendermaßen ab:

```
Process S          Process C          Process F
{
  while (TRUE)
  {
    <zur Rampe fahren>;
    <1 Schrank aufladen>;
    <Rampe verlassen>;
  }
}
{
  while (TRUE)
  {
    <zur Rampe fahren>;
    <1 Couch aufladen>;
    <Rampe verlassen>;
  }
}
{
  while (TRUE)
  {
    <zur Rampe fahren>;
    <1 Schrank entladen>;
    <1 Couch entladen>;
    <Rampe verlassen>;
  }
}
```

Synchronisieren Sie diese drei Prozesse, indem Sie in Pseudocode-Notation in geeigneter Weise Semaphore deklarieren und Semaphore-Operationen einfügen.

Die Pseudocode-Operation zum Deklarieren eines Semaphors mit Namen `name` mit Startwert `n` lautet:

```
name(n);
```

Die Pseudocode-Operationen zum Aufrufen von P und V auf der Semaphore mit Namen `name` lauten:

```
P(name)
```

```
V(name)
```

Sperrphasen sind möglichst kurz zu halten und folgende Bedingungen müssen erfüllt sein:

- Zur Laderampe kann jeweils nur ein Prozess fahren.
- Der Abholer S darf nur zur Rampe fahren, wenn noch ein Schrank im Lager ist. Analoges gilt für den Abholer C.
- Die Lagerhalle hat beschränkte Kapazität, sie kann höchstens 20 Möbelstücke aufnehmen.
- Der Fabrikant darf nur zur Rampe fahren, wenn er seine Lieferung vollständig abladen kann.
- Zu Beginn sei das Lager leer und die Rampe frei.

1. Listen Sie alle benötigten Semaphoren auf und erklären Sie ihren Zweck bzw ihre Pragmatik (4 P)
2. Fügen Sie in den /zu dem oben aufgelisteten Pseudocode des Ablaufs der beteiligten Prozesse an geeigneter Stelle entsprechende Aufrufe zum Deklarieren und von P und V ein! (4 P)

Lösungsvorschlag:

Angenommen, in der Lagerhalle befinden sich s Schränke und c Couches. Wir verwenden dann je einen Semaphor für die drei Ungleichungen $20 \geq s + c$, $s \geq 0$, $c \geq 0$. Für die exklusive Benutzung der Rampe verwenden wir einen booleschen Semaphor (Mutex).

Wir brauchen also folgende Semaphore (4 P):

- Boolescher Semaphor `mutexRampe` mit Startwert 1; zu Beginn ist damit die Rampe frei
- Semaphor `lagerhalle` mit Startwert 20; verhindert die Anfahrt des Fabrikanten F, wenn die Lagerhalle nicht Platz für mindestens 2 Möbelstücke hat
- Semaphor `schrankZahl` mit Startwert 0; zählt die Anzahl der Schränke in der Lagerhalle; verhindert die Anfahrt des Ausfahrers S, falls sich kein Schrank in der Lagerhalle befindet
- Semaphor `couchZahl` mit Startwert 0; zählt die Anzahl der Couches in der Lagerhalle; verhindert die Anfahrt des Ausfahrers S, falls sich keine Couch in der Lagerhalle befindet.

Die P- und V-Operationen und die Deklarationen müssen folgendermaßen eingebaut werden (4 P):

```
mutexRampe(1);
lagerhalle(20);
schrankZahl(0);
couchZahl(0);
```

```
Process S          Process C          Process F
{                  {                  {
while (TRUE)      while (TRUE)      while (TRUE)
{                  {                  {
  P(schrankZahl);  P(couchZahl);    P(lagerhalle);
  P(mutexRampe);  P(mutexRampe);   P(lagerhalle);
  <zur Rampe fahren>;
  <1 Schrank aufladen>;
  V(lagerhalle);  V(lagerhalle);   P(mutexRampe);
  <Rampe verlassen>;
  V(mutexRampe);  V(mutexRampe);   <zur Rampe fahren>;
}                  }                  <1 Schrank entladen>;
}                  }                  V(schrankZahl);
}                  }                  <1 Couch entladen>;
}                  }                  V(couchZahl);
}                  }                  <Rampe verlassen>;
}                  }                  V(mutexRampe);
}                  }                  }
}                  }                  }
```

Die P-Operation für den Semaphor `lagerhalle` muss zweimal aufgerufen werden um sicherzustellen, dass zwei Plätze in der Lagerhalle frei sind.