Aufgaben zum Thema „Threads (Prozesse)“

T1. Erläutern Sie zwei Methoden, mit denen ein Prozeß seinen kritischen Abschnitt schützen kann! Geben Sie jeweils eine Implementationsmöglichkeit (in Pseudocode) an, und bewerten Sie die gewählten Methoden! 94/7

T2. Was versteht man unter dem Begriff „Prozeß“ im Rahmen von Betriebssystemen? Warum wird er eingeführt? Erläutern Sie den Zustandsbegriff für Prozesse einschließlich der möglichen bzw. sinnvollen Zustandsübergänge! 95/5

T3. Geben Sie einen Überblick über die Mittel, die in Betriebssystemen zur Realisierung kritischer Abschnitte bereitgestellt werden können, und bewerten Sie sie! 95/5

T4. Erläutern Sie das Problem „Race Condition“ an folgendem Beispiel!

```
VAR kontostand: INTEGER;
PROCEDURE abbuchen (betrag: INTEGER);
        BEGIN   kontostand := kontostand - betrag   END;

kontostand := 100;
Prozeß 0:   abbuchen(10);          Prozeß 1:   abbuchen(80);
```

95/5

T5. Der Prozeß A fülle zyklisch einen zweielementigen Puffer P_1. P_1 wird (gemäß FIFO) durch einen Prozeß B geleert, der ein gelesenes Element unmittelbar in einen Puffer P_2 schreibt; dieser Puffer wird durch einen Prozeß C geleert. Erläutern und lösen Sie das genannte Problem! 95/5

T6. Lösen Sie das folgende Erzeuger-Verbraucher-Problem mittels Semaphoren! Ein Prozeß E fülle einen linearen Puffer unbegrenzter Kapazität mit Datenelementen. Diese Datenelemente werden durch einen Prozeß V aus dem Puffer gemäß der Strategie LIFO entnommen.

Bewerten Sie Ihre Lösung! Welche Modifikationen sind erforderlich, wenn der Puffer durch mehrere Prozesse gefüllt wird? 96/2

T7. Drei parallele Prozesse nutzen einen Puffer der Größe 1 KByte zum byteweisen Senden von Daten (unterschiedlichen Umfangs) an einen Drucker; dabei soll das Senden durch einen Prozeß jeweils erst abgeschlossen sein, bevor der nächste Prozeß den Puffer benutzen darf. Ein weiterer Prozeß liest die Daten byteweise aus dem Puffer und übergibt sie an den Drucker.

Entwerfen Sie in Pseudocode unter Verwendung von Semaphoren eine Lösung des Problems! 96/8

T8. In einem System paralleler Prozesse existiere ein Druckerspooler, der aus einem 4 KByte großen Puffer Daten liest und byteweise zum Drucker sendet. Andere Prozesse übergeben ihre zu druk-kenden Daten (unterschiedlichen Umfangs), indem sie diese byteweise in den Puffer schreiben; dabei sollen sich die Daten eines Prozesses mit denen eines anderen nicht vermischen.

Beschreiben Sie in Pseudocode unter Verwendung von Semaphoren die Struktur des Druckerspoolers und der sendenden Prozesse, so daß eine maximale Parallelität möglich ist! (Beachten Sie die genaue Beschreibung der verwendeten Semaphore und deren Initialisierung!) 97/7

T9*. Beschreiben Sie unter Verwendung von Semaphoren die Implementation eines Wechselpuffers folgender Struktur und Strategie. Der Puffer bestehe aus den beiden Teilen `linkeHälfte` und `rechteHälfte` (und kann damit als zweielementig betrachtet werden). Er werde durch den Prozeß A gefüllt und durch den Prozeß B geleert; beide Prozesse arbeiten zyklisch. Ist der gesamte Puffer leer, so füllt A stets zuerst `linkeHälfte`, genauso leert B bei vollem Puffer zuerst `linkeHälfte`. Beide Prozesse sollen weitestgehend parallel arbeiten können. Zur Beschreibung des Pufferzustands ist neben den ggf. erforderlichen Semaphorvariablen maximal eine weitere Variable `zustand` zu verwenden. (Hinweis: Ermitteln Sie zunächst die möglichen Pufferzustände und deren Änderungen!).

Demonstrieren Sie die Korrektheit Ihrer Lösung!

Angenommen, die Daten besitzen eine Gültigkeitsdauer. Welches Problem birgt dann die Strategie des Füllens und Leerens in sich? 98/2

T10. In einem System paralleler preemptiver Prozesse gebe es zwei gemeinsam genutzte Variablen

```
int  s = 1; z = 0;
```

sowie zwei Prozesse P_0, P_1 folgender Struktur:

```
P0:   { z++; }          P1:   { z--; }
```

a) Welche Werte kann z annehmen, welches ist das Endergebnis? Welches Problem tritt auf?

b) Die Prozesse werden in folgender Weise modifiziert:

```
P0:   { while (s ==1);          P1:   { while (s == 0);
        z++;                            z--;
        s = 1;                          s = 0;
      }                               }
```

Welche Werte kann z nun annehmen, welches ist das Endergebnis? Was wird durch die Verwendung der Variablen s bewirkt?

c) Verwenden Sie Semaphore statt der while-Konstruktion, um das gleiche Endergebnis von z wie in b) zu erreichen! Welche negative Eigenschaft der Implementation b) wird damit beseitigt? 99/8, 09/8

T11. Ein Prozeß A will mit `read(x)` einmalig eine Nachricht in Form einer Integer-Variablen `x` lesen, die ihm von einem anderen Prozeß B durch `write(x)` in einem gemeinsam genutzten Speicher übergeben werden soll.

a) Welches Problem tritt dabei auf? Lösen Sie das Problem mittels Semaphoren!

b) Kann das Problem auch ohne Semaphore und ohne spezielle Hardware-Unterstützung, also nur mit einfachen Lese-, Schreib- und Testoperationen auf einer Variablen gelöst werden? Wenn ja, geben Sie eine möglichst einfache Lösung an und begründen Sie deren Korrektheit, wenn nein, beweisen Sie es!

c) Angenommen, die Nachricht wird nicht als Integer-Wert übergeben, sondern als Zeichenkette, die byteweise in einem Puffer der Größe 10 Byte abgelegt wird; sie kann auch byteweise entnommen werden. Dabei sollen beide Prozesse möglichst parallel arbeiten. Die Länge der Nachricht ist nicht von vornherein bekannt. Beschreiben Sie die Struktur der Lösung in den Fällen a) und b) (mit bzw. ohne Semaphore), sofern dies möglich ist! 00/2

T12. Betrachtet werde folgendes Problem. Für ein System paralleler Prozesse existiere ein gemeinsam genutzter Puffer (der als Ganzes betrachtet wird). Dieser Puffer wird von einem Prozeß S beschrieben und von prinzipiell beliebig vielen Prozessen L gelesen, allerdings ist ein gleichzeitiges Lesen nur durch maximal fünf Prozesse zugelassen; außerdem ist kein gleichzeitiges Lesen und Schreiben möglich.

Dazu sei folgender Lösungsversuch gegeben:

```
                    int z = 0;
                    sem_t *S = new_sem(1);
                    sem_t *L = new_sem(5);
Leserprozeß: while(1) {              Schreiberprozeß:    while (1) {
               z = z+1;                                    P(S);
               if (z == 1)                                 schreiben();
                 P(S);                                     V(S);
               P(L);                                     }
               lesen();
               V(L);
               z = z-1;
               if (z == 0)
                 V(S);
             }
```

a) Welchem Zweck dienen die drei Variablen z, *S und *L?

b) Die Lösung ist nicht korrekt. Worin liegt die Ursache?

c) Geben Sie eine korrekte Lösung an, indem Sie die vorliegende Lösung geeignet modifizieren!

02/2

T13.

a) In einem Multiprozessorsystem gebe es eine gemeinsam genutzte Variable

```
int z = 7;
```

und ein Programm

```
{ z = z+1; }
```

das von zwei Prozessen A, B einmalig abgearbeitet wird.

a) Warum hat nach Beendigung von A und B die Variable `z` nicht notwendig den Wert 9? Welche Werte kann sie haben?

b) Modifizieren Sie das Programm so, daß der Endwert 9 garantiert ist! (Die Anweisung `z = 9` ist dabei ausgeschlossen.)

c) Das folgende Programm initialisiert die Variable `i` mit 0, öffnet durch `fopen` eine Datei namens `i.txt` mit Schreibrecht und erzeugt danach einen weiteren Prozeß. Jeder der nunmehr zwei Prozesse erhöht die Variable `i` um 1 und schreibt sie mit `fprintf` als Dezimalzahl in die geöffnete Datei. Welchen Inhalt hat die Datei nach Ende beider Prozesse (unter der Annahme, daß es keine Fehler während der Programmausführung gibt)? (Begründung!)

```
#include <stdio.h>
#include <unistd.h>

int main(void)  {
  int i = 0;
  FILE *f = fopen("i.txt", "w");
  if (f) {
    if (fork() >= 0) {
      i = i+1;
      fprintf(f, "%d", i);  }
    fclose(f);  }
  return 0;  }
```

d) Das folgende Programm initialisiert die Variable `i` mit 0, erzeugt zwei Prozesse und jeder der nunmehr drei Prozesse erhöht die Variable um 1. Welche Ausgabe ist auf dem Bildschirm zu erwarten? (Fehler bei der Prozeßerzeugung wurden der Einfachheit halber ignoriert.)

```
#include <stdio.h>
#include <unistd.h>
int main(void)
{
  int i, j;
  i=0;
  for(j=0; j<2; j++) {
    if (fork() == 0) {
      i = i+1;
      printf("i = %d\n", i);
      exit(0);
    }
  }
  i = i+1;
  printf("i = %d\n", i);
  return 0;
}
```

02/2, 01/2

T14. Für ein System paralleler Prozesse existiere ein gemeinsam genutzter Puffer (der als Ganzes betrachtet wird). Dieser Puffer werde von einem Prozeß S beschrieben und von prinzipiell beliebig vielen Prozessen L gelesen. Dabei ist ein gleichzeitiges Lesen nur durch maximal fünf Prozesse zugelassen, und es ist kein gleichzeitiges Lesen und Schreiben möglich. Außerdem kann der Schreiberprozeß S auch nicht schreiben, solange es noch einen wartenden Leserprozeß gibt.

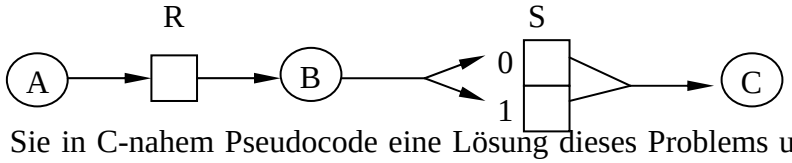
a) Geben Sie (in Pseudocode) eine Implementation für die Prozesse L und S unter Verwendung von Semaphoren an! (Hinweis: Es handelt sich ***nicht*** um ein Erzeuger-Verbraucher-Problem!)

b) Zu welchem Problem führt die in der Aufgabenstellung beschriebene Struktur des Prozeßsystems? Wie müßte die Aufgabenstellung geändert werden, um dieses Problem zu lösen, und welche Konsequenzen hätte dies?

c) Inwieweit ändert sich die Situation, wenn es mehrere Schreiberprozesse gibt? 02/8, 7/99

T15. Zwei Threads, die im gleichen Adreßraum laufen, negieren unabhängig voneinander im Laufe ihrer Existenz genau einmal dieselbe boolesche Variable `x` mit dem Ausgangswert `true`. Welchen Wert hat `x`, nachdem die beiden Threads ihre Operation ausgeführt haben? 03/3

T16. Betrachtet werde das folgende Erzeuger-Verbraucher-Problem. Ein Prozeß A füllt einen einelementigen Puffer R mit Datenelementen. Ein Prozeß B entnimmt ein Datenelement, gibt R frei, modifiziert das Datenelement (dies beansprucht eine gewisse Zeit) und trägt es anschließend in einen zweielementigen Puffer S (implementiert als zweielementiges Feld) ein. Das erste Datenelement wird in das Pufferfeld S[0] eingetragen, die weiteren Einträge erfolgen streng abwechselnd. Ein Prozeß C entnimmt die Datenelemente aus S gemäß FIFO. Jedes Pufferfeld kann genau ein Datenelement aufnehmen. Der Zugriff auf jedes der drei Pufferfelder ist exklusiv; allerdings sollen B und C gleichzeitig auf den Puffer S zugreifen können.



a) Geben Sie in C-nahem Pseudocode eine Lösung dieses Problems unter Verwendung von Semaphoren und ohne das Auftreten von busy waiting an! Das Füllen und Leeren eines Pufferfeldes `X` kann durch nicht näher spezifizierte Funktionen `fuellen(X)` und `leeren(X)` symbolisch beschrieben werden.

b) Wenn nur der Puffer R zur Verfügung steht (d.h. wenn B die modifizierten Datenelemente wieder in R einträgt, von wo sie dann durch C endgültig entnommen werden), so hat dies eine Konsequenz, die für das Gesamtverhalten des Prozeßsystems wesentlich ist. Welche Konsequenz ist das? Achten Sie auf eine korrekte Begründung! 03/3

T17. Drei Threads P, Q und R eines Prozesses laufen parallel auf mehreren Prozessoren eines Multiprozessor-Systems mit gemeinsamem Speicher. Jeder der Threads inkrementiert die globale Variable `count` um 1 (Anfangswert: 17).

a) Warum kann `count` verschiedene Endwerte haben, welche sind das?

b) Inwieweit ändert sich die Situation, wenn die Inkrement-Operation auf den einzelnen Prozessoren nicht unterbrechbar ist?

c) Inwieweit spielt die Verteilung der Prozesse auf die Prozessoren eine Rolle?

d) Geben Sie eine Lösung an, die zu einem deterministischen Ergebnis führt, bei der kein busy waiting auftritt! 03/8

T18. Im Adreßraum eines Prozesses gebe es die drei wie folgt definierten globalen Variablen:

```
int  i;
int  z = 0;
sem_t  *s = new_sem(1);
```

In diesem Adreßraum laufen zwei Threads A und B, die beide einmalig folgenden gleichen Programmcode abarbeiten:

```
for (i = 0;  i < 2;  i++)  {
   P(s);
   z++;
   V(s);
}
```

a) Welchen Endwert hat `z`, wenn A erst nach Beendigung von B gestartet wird?

b) Ist der Endwert von `z` bei paralleler Abarbeitung von A und B eindeutig? (Begründung!)

c) Wenn der Wert eindeutig ist: welcher Wert ist dies und warum? Wenn der Wert nicht eindeutig ist: welche Werte sind möglich, wie kommen sie zustande? 04/3

T19. Im Adreßraum eines Prozesses gebe es zwei globale Variable

```
int  x = 1, y =1;
```

In diesem Adreßraum laufen zwei Threads A und B, die jeweils einmalig folgenden Programmcode abarbeiten:

Thread A:

```
void A() {
  x = x+1;
  x = x*y;
}
```

Thread B:

```
void B() {
  y = y+1;
}
```

Die drei Anweisungen sind als atomar (unteilbar) zu betrachten.

a) Welchen Endwert hat `x`, wenn beide Threads rein sequentiell abgearbeitet werden?

b) Inwieweit ändern sich die möglichen Werte von `x` bei paralleler Abarbeitung?

c) Verändern Sie die Threads A und B so, daß ein eindeutiges Ergebnis gesichert ist (das Auftreten von busy waiting sowie Anweisungen der Form `x = 7` sind dabei nicht zugelassen)! 04/8

T20. Im Adreßraum eines Prozesses gebe es drei globale Variablen

```
int  x = 1;
sem_t  s1 = new_sem(1),  s2 = new_sem(0);
```

In diesem Adreßraum laufen drei Threads A, B und C, die jeweils einmalig folgenden Programmcode abarbeiten:

Thread A:

```
void A() {
  P(s1);
  x = x+1;
  V(s1);
}
```

Thread B:

```
void B() {
  P(s2);
  x = x+x;
  V(s1);
}
```

Thread C:

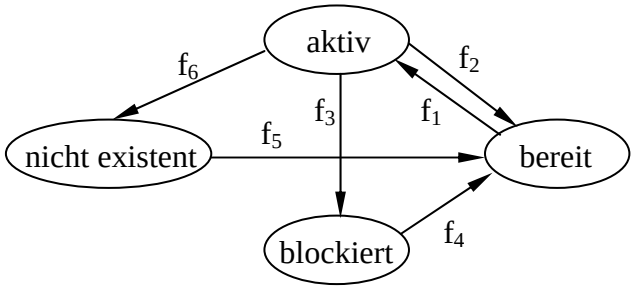
```
void C() {
  P(s1);
  x = x-1;
  V(s2);
}
```

a) Geben Sie alle möglichen Ablauffolgen und den jeweiligen Endwert von `x` an!

b) Was wird durch den Einsatz der Semaphore `s1` und `s2` erreicht?

c) Können Deadlocks zwischen diesen Threads auftreten? 05/2

T21. In der Prozeßverwaltung eines Betriebssystems seien die sechs Zustandsübergänge f_1,...f_6 des nebenstehenden Zustandsdiagramms implementiert.



a) Geben Sie für jeden dieser Übergänge eine Situation (eine Bedingung) an, in der er ausgeführt wird! Nennen Sie außerdem die wesentliche Systemaktivität bei f_5 und f_6!

b) Angenommen, es seien die folgenden Voraussetzungen erfüllt:
- besitzt ein Prozeß alle geforderten Betriebsmittel, so wird er nach endlicher Zeit fertig (es treten insbesondere keine unendlichen Zyklen auf);
- jeder Prozeß gibt spätestens am Ende alle von ihm belegten Betriebsmittel frei;
- das System führt die Übergänge f_1,...,f_6 stets fehlerfrei aus.

Trotz dieser Voraussetzungen ist es aus zwei unterschiedlichen Gründen möglich, daß im Multiprogrammbetrieb ein Prozeß nie den Zustand „nicht existent“ erreicht. Beschreiben Sie *einen* der beiden Gründe! Geben Sie eine zugehörige Modifikation des obigen Zustandsdiagramms an, die dem „ewigen“ Verbleiben eines Prozesses im System entgegenwirkt! 05/8

T22. Jemand stellt folgende Forderung auf: Wenn zwei Threads eines Prozesses eine Variable gemeinsam benutzen, so muß in jedem Fall der Zugriff auf diese Variable als kritischer Abschnitt behandelt und entsprechend geschützt werden, da sonst Wettlaufsituationen (race conditions) auftreten können.

a) Geben Sie in Pseudocode (ähnlich Aufg. b) ein möglichst einfaches Beispiel an, das diese Forderung rechtfertigt, und erläutern Sie damit den Begriff „Wettlaufsituation“!

b) Im Adreßraum eines Prozesses gebe es zwei globale Variablen

```
int  x = 1;
sem_t  s(0);
```

In diesem Adreßraum laufen einmalig zwei Threads A und B, die jeweils folgenden Programmcode abarbeiten:

Thread A:

```
void A() {
  s.P();
  x = x+1;
}
```

Thread B:

```
void B() {
  x = x+x;
  s.V();
}
```

Geben Sie alle möglichen Abläufe von A und B sowie alle dadurch möglichen Endwerte von `x` an!

c) Welchen Schluß ziehen Sie daraus für die Allgemeingültigkeit der eingangs genannten Forderung? 06/8

T23. a) Welche beiden grundlegenden Bedingungen muß jeder Mechanismus zum Schutz kritischer Abschnitte erfüllen (Begriffe und Erklärungen!)?

b) In WIKIPEDIA findet man die folgende Information:

Der Dekker-Algorithmus ist … eine vollständige Lösung des Problems des wechselseitigen Ausschlusses in der dezentralen Steuerung von Prozessen. Er vermeidet gegenseitiges Blockieren und gewährleistet, dass stets genau ein Prozess in einen kritischen Abschnitt gelangen kann. Der Algorithmus kann mit folgendem Pseudo-C-Code schematisch beschrieben werden:

```
// globale Variablendeklaration
boolean flag0 = false, flag1 = false;
int turn = 0;
```

```
    // Prozess #0
 1   // ...
 2   P:   flag0 = true;
 3        while (flag1) {
 4           if (turn == 1) {
 5              flag0 = false;
 6              goto P;
 7           }
 8        }
 9        // <krit. Abschnitt>
10        flag0 = false;
11        turn = 1;
12   // ...
```

```
// Prozess #1
 1   // ...
 2   P:   flag1 = true;
 3        while (flag0) {
 4           if (turn == 0) {
 5              flag1 = false;
 6              goto P;
 7           }
 8        }
 9        // <krit. Abschnitt>
10        flag1 = false;
11        turn = 0;
12   // ...
```

(Ende des Zitats.) Die jeweils links stehenden Zeilennummern gehören nicht zum Algorithmus; sie sollen lediglich zur Referenz für die folgende Aufgabenstellung dienen.

Untersuchen Sie, ob der Algorithmus in der angegeben Form tatsächlich die erforderlichen Eigenschaften hat! Wenn ja, zeigen Sie dies! Wenn nein, so begründen Sie dies durch die Angabe eines entsprechenden Ablaufbeispiels mit kurzer Erläuterung! 06/3

P.S. Obiger Code wurde 1 Tag nach der Klausur geändert, nicht allerdings der – zu den Änderungen nicht mehr passende! – ursprüngliche Korrektheits-„Beweis“.

T24. Im Adreßraum eines Prozesses laufen einmalig zwei Threads A und B, die jeweils folgenden Programmcode mit der globalen Variablen

```
int  x = 1;
```

abarbeiten:

Thread A:

```
void A() {
  x = x+2;
}
```

Thread B:

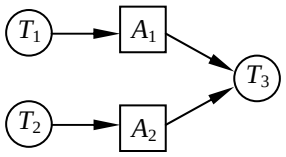
```
void B() {
  x = 4*x;
}
```

a) Erläutern Sie den Begriff „Wettlaufsituation (race condition)“ und zeigen Sie, daß diese Situation hier vorliegt! Geben Sie dazu alle Endwerte von `x` an, die im vorliegenden Fall möglich sind (Begründung!)!

b) Lösen Sie unter Verwendung von Semaphoren obiges Problem, indem Sie die Zugriffe auf die Variable `x` als kritischen Abschnitt behandeln! Welche der Endwerte von `x` sind nunmehr ausgeschlossen?

c) Geben Sie eine Lösung mittels Semaphoren an, bei der keine Wettlaufsituation auftritt und bei der zusätzlich der Endwert von `x` eindeutig bestimmt ist! 07/8

T25. Betrachtet werde das folgende Erzeuger-Verbraucher-Problem. Zwei Threads T_1, T_2 eines Prozesses füllen wiederholt jeweils einen einelementigen Puffer A_1, A_2; diese beiden Puffer werden durch einen dritten Thread T_3 geleert (s. Abb.), wobei die Reihenfolge der Entnahme gleichgültig ist. Geben Sie unter Verwendung von Semaphoren eine Lösung dieses Problems in Pseudocode an, die folgende Eigenschaften hat:



- es darf kein busy waiting auftreten;
- es muß maximale Parallelität gewährleistet sein (insbes. soll T_3 arbeiten können, sobald einer der beiden Puffer gefüllt ist);
- es sollen so wenig Variablen wie möglich verwendet werden. 07/2

T26. Betrachtet werde ein Threadsystem mit dem folgenden Verhalten. Zwei Threads A und B dürfen nicht gleichzeitig ausgeführt werden; ein dritter Thread C kann erst abgearbeitet werden, wenn die Ausführung von A und B beendet ist.

Geben Sie in Pseudocode eine Implementation dieser Threads an (die eigentliche Ausführung der Threads werde durch //working// symbolisiert) mit folgenden Eigenschaften:

- es ist größtmögliche Unabhängigkeit bei der Thread-Abarbeitung gewährleistet;
- die Anzahl der Variablen ist so gering wie möglich. 08/2

T27. Betrachtet werde ein System von drei Threads, die in einem gemeinsamen Adreßraum laufen und folgende globalen Variablen nutzen:

```
sem_T s(1), t(0);
```

Die Threads haben folgende Struktur:

Thread A:
```
void A() {
  s.P();
  //Code-A//
  s.V();
}
```

Thread B:
```
void B() {
  s.P();
  //Code-B//
  s.V();
  t.V():
}
```

Thread C:
```
void C {
  t.P();
  s.P();
  //Code-C//
  s.V();
}
```

a) Geben Sie alle dadurch möglichen Abläufe an! Was bewirken die beiden Semaphore `s` und `t`?

b) Das Vertauschen der beiden P-Operationen in Thread C kann dazu führen, daß sich alle drei Threads in einem Deadlock befinden.

- Beschreiben Sie einen Ablauf, der zu dieser Situation führt!
- Diskutieren Sie die Deadlock-Situation unter dem Gesichtswinkel aller Bedingungen, die für das Auftreten von Deadlocks notwendig sind! Verwenden Sie dazu auch einen Prozeß-Betriebsmittel-Graphen! Welcher bemerkenswerte Schluß muß damit gezogen werden, was ist die Ursache? 08/8, 09/2