Zusatzübung

Bei Fragen und Problemen können Sie uns per E-mail unter den folgenden Adressen erreichen:

Mesut Güneş	guenes@informatik.rwth-aachen.de
Ralf Wienzek	wienzek@informatik.rwth-aachen.de
Übungsgruppen	{canan,carsten,lukas,min,oezguer,ulrich}@i4.informatik.rwth-aachen.de

Aufgabe 1 (8 Minuten): Prozesszustände

In der Vorlesung haben Sie gelernt, dass sich Prozesse während ihrer Lebensdauer in verschiedenen Zuständen befinden können:

Zustand	Bedeutung
new	Prozess wird erzeugt (kreiert)
running	Anweisungen des Prozesses werden gerade ausgeführt
ready	Prozess ist bereit zur Ausführung, wartet aber noch auf freien Prozessor
waiting	Prozess wartet auf das Eintreten eines Ereignisses, z.B. darauf, dass ein von ihm belegtes Betriebsmittel fertig wird
blocked	Prozess wartet auf ein fremdbelegtes Betriebsmittel
killed	Prozess wird (vorzeitig) abgebrochen
terminated	Prozess ist beendet, d.h. alle Instruktionen sind abgearbeitet

Gegeben seien 4 Prozesse, die zu unterschiedlichen Zeitpunkten gestartet werden und sich in die ready-Warteschlange einreihen. Der Scheduler arbeitet nach dem nicht-preemtiven LIFO-Verfahren.

Einige Prozesse benötigen Zugriff auf den Drucker. Die zweite Zeile der nachfolgenden Tabelle gibt an, nach wie vielen Zeiteinheiten im Zustand *running* ein Prozess auf den Drucker zugreifen möchte. Während der 4 Zeiteinheiten dauernden Druckphase wird der Prozessor für andere Prozesse freigegeben. Nach Beendigung des Druckvorgangs reiht sich der Prozess wieder in die ready-Warteschlange ein. Der Prozess P3 wird abgebrochen, nachdem er 2 Zeiteinheiten gerechnet hat.

Prozess	P1	P2	P3	P4
Startzeitpunkt	0	2	1	4
Drucken nach Zeiteinheit	5	4	-	1
Benötigte Zeiteinheiten im Zustand running	6	7	10	3

Geben Sie zu den ersten 22 Zeiteinheiten die aktuellen Prozesszustände der Prozesse P1 bis P4 an. Beachten Sie, dass ein ankommender Prozess bei seiner Ankunft den Zustand *new* annimmt und erst nach einer Zeiteinheit in einen anderen Zustand wechselt, wobei keine CPU-Zeit verbraucht wird. Des Weiteren können Prozesse, die ihren Druckvorgang beendet haben, direkt in den Zustand *running* wechseln, ohne vorher eine Zeiteinheit im Zustand *ready* verbringen zu müssen.

Aufgabe 2 (15 Minuten): Programmieren in C

Betrachten Sie das folgende C-Programm:

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/ipc.h>
```

```
4  #include <sys/shm.h>
5  #include <unistd.h>
6
7  #define CONST 50
8
9  char *p_char;
10 int *p_int1, *p_int2;
11
12 void proc1( char *n )
13 {
14         FILE *f;
15         int c = EOF;
16
17         if ( (f = fopen( n, "r" )) != NULL )
18                 while( ( c = fgetc( f )) != EOF )
19                 {
20                         while ( ((*p_int1 + 1) % CONST) == *p_int2 );
21                         p_char[ *p_int1 ] = (char)c;
22                         *p_int1 = (*p_int1 + 1) % CONST;
23                 }
24
25         while ( ((*p_int1 + 1) % CONST) == *p_int2 );
26         p_char[ *p_int1 ] = (char)c;
27         *p_int1 = (*p_int1 + 1) % CONST;
28
29         while ( ((*p_int1 + 1) % CONST) == *p_int2 );
30         p_char[ *p_int1 ] = (char)c;
31         *p_int1 = (*p_int1 + 1) % CONST;
32
33         fclose(f);
34 }
35
36 void proc2( char *n )
37 {
38         FILE *f;
39         int c;
40
41         if ( ( f = fopen( n, "w" )) == NULL )
42                 return;
43
44         do
45         {
46                 while( *p_int1 == *p_int2 );
47                 c = p_char[*p_int2];
48                 *p_int2 = (*p_int2 + 1) % CONST;
49                 if ( c != EOF )
50                         fputc( c, f );
51                 else
52                         break;
53         } while( 1 );
54         fclose( f );
55 }
56
57 int main( int argc, char *argv[] )
58 {
59         int c1, c2;
60         int ID;
```

```
61
62          ID = shmget( IPC_PRIVATE, CONST + 2*sizeof( int ), IPC_CREAT|0x1ff );
63          p_char = (char *)shmat( ID, NULL, 0 );
64          p_int1 = (int *)(p_char + CONST);
65          p_int2 = p_int1 + 1;
66
67          *p_int1 = 0;
68          *p_int2 = 0;
69
70          if ( (c1 = fork()) == 0 )
71          {
72                  if ( (c2 = fork()) == 0 )
73                  {
74                          proc1( argv[1] );
75                  }
76                  else
77                  {
78                          proc2( argv[2] );
79                  }
80          }
81          else
82          {
83                  proc2( argv[3] );
84          }
85  }
```

a) Kommentieren Sie das Programm ausführlich.

b) Beschreiben Sie, was das Programm macht.

c) Arbeitet das Programm korrekt? Begründen Sie Ihre Meinung.

Aufgabe 3 (12 Minuten): Wechselseitiger Ausschluss

Eine Lösung des Problems des wechselseitigen Ausschlusses von zwei Prozessen wurde 1966 von L. Hyman vorgestellt. Der Algorithmus stellt den Prozessen P_0 und P_1 folgende gemeinsame Variablen zur Verfügung:

```
bool flag[2];   // mit false initialisiert
int turn;       // mit den Werten 0 bis 1, Initialisierung beliebig
```

Der Prozess P_i arbeitet nach folgender Programmsequenz, wobei P_j den jeweils konkurrierenden Prozess bezeichnet:

```
 1  while (true)
 2  {
 3      flag[i] = true;
 4      while (turn != i)
 5      {
 6          while (flag[j])
 7          {
 8              noop;
 9          }
10          turn=i;
11      }
12      criticalSection(Pi);
13      flag[i] = false;
```

```
14      remainderSection(Pi);
15  }
```

Arbeitet das Verfahren korrekt? Prüfen Sie dies anhand der Anforderungen an eine korrekte Lösung des wechselseitigen Ausschlussproblems. Skizzieren Sie für *jede* der Bedingungen den Beweis oder geben Sie ein Gegenbeispiel an.

Aufgabe 4 (20+15+5+9+1 Minuten): Synchronisation

Das Raumschiff *Enterprise* hat wieder einmal eine äußerst gefährliche Mission abgeschlossen. Nun soll sich die Besatzung bei einem ausgedehnten Landurlaub auf dem friedlichen Planeten *Risa* erholen. Da die Transporter im Einsatz irreparabel beschädigt wurden, bekommt Chief O'Brien von Captain Picard den Auftrag, eine für die Besatzung zufriedenstellende Lösung für die Synchronisation der 12 zur Verfügung stehenden Shuttles, welche jeweils bis zu 6 Besatzungsmitglieder gleichzeitig nach *Risa* transportieren können, zu gestalten. Sie sollen ihm dabei helfen!

O'Brien hat bereits folgende Dinge festgestellt:

- Es ist davon auszugehen, dass jedes Shuttle einwandfrei funktioniert und jeder Transport erfolgreich abgeschlossen wird.
- Sollte ein teilweise besetztes Shuttle (> 0 Passagiere) länger als 60 Minuten auf Passagiere warten, bekommt der entsprechende Pilot die Anweisung, sofort loszufliegen.
- Entschließt sich ein Besatzungsmitglied für den Flug nach *Risa*, so soll die Person das nächste Computerterminal aufsuchen und dort eine Shuttle-Nummer anfordern (buchen).
- Ist ein Besatzungsmitglied einmal einem der Shuttles zugeteilt, so muss die betreffende Person seinen Sitzplatz innerhalb der Zeit, die das Shuttle insgesamt noch wartet, belegen. Ansonsten verfällt die Buchung.

a) Realisieren Sie O'Briens Ansätze mittels **Bedingter Kritischer Regionen** in einem **C-ähnlichen Pseudocode**. Insbesondere sollen folgende Programmteile implementiert werden:

   1. passende Variablen (`const`, `shared`) sowie eine Funktion `init()` zur Initialisierung der gemeinsam genutzten Variablen und der Shuttle-Prozesse `1`, ..., `6`,
   2. eine Funktion zur Ablauf-Steuerung eines Shuttles, die sowohl die Funktion `flyToRisaAndBack()` als auch die Funktion `stopwatch(int time, bool *signal)` verwendet, welche nicht blockierend ist und die Variable `signal` nach `time` Minuten auf `TRUE` setzt,
   3. eine Funktion, mit der ein reiselustiges Besatzungsmitglied eine Shuttle-Nummer buchen kann und
   4. eine Funktion, welche eine Buchung für ein bestimmtes Shuttle einlöst und dem Besatzungsmitglied einen Sitzplatz zuweist (Funktion `showPassengerHisSeat()`), falls das Shuttle noch nicht unterwegs ist.

b) Implementieren Sie nun in einem **C++-/Java-ähnlichen Pseudocode Monitore**, welche die Funktionalität aus Teil a) bieten. Es sei vereinbart, dass wenn Prozess *A* ein Signal setzt, auf das Prozess *B* wartet, *A* solange weiterarbeitet, bis er entweder den Monitor verlässt oder bis er mittels `wait` auf ein Signal warten muss und sich schlafen legt. *B* erhält unmittelbar im Anschluss daran die Kontrolle über den Monitor, auch dann, wenn weitere Prozesse in der Zwischenzeit eine Monitor-Funktion verwenden wollen. Das folgende Grundgerüst soll Ihnen als Vorlage dienen (die von Ihnen auszufüllenden Bereiche sind mit `//todo` gekennzeichnet).

1. Monitor für die Verwaltung der Shuttle-Flüge:

```
1  Monitor ShuttleManager
2  {
3      final int MAX_SHUTTLES = 12;
4      Shuttle[] shuttles = new Shuttle[MAX_SHUTTLES];
5
6      //Initialisierung des Flugverkehrs
7      void init()
8      {
9          //todo
10     }
11
12     /* Methode zur Buchung eines Sitzplatzes in einem Shuttle.
13      * Liefert -1, falls gerade alle Plaetze belegt sind,
14      * sonst die entsprechende Shuttle Nummer.
15      */
16     int bookSeat()
17     {
18         //todo
19     }
20 }
```

2. Monitor für das Shuttle:

```
1  Monitor Shuttle
2  {
3      final int MAX_SEATS = 6;
4      final int MAX_SHUTTLE_WAIT = 60;
5
6      //todo
7
8      Shuttle(int i)
9      {
10         //todo
11     }
12
13     /* Liefert -1, falls alle Plaetze belegt sind,
14      * sonst i und decrementiert die freien Buchungen.
15      */
16     int decFreeBookings()
17     {
18         //todo
19     }
20
21     void run()
22     {
23         while(true)
24         {
25             //MAX_SHUTTLE_WAIT Minuten Stoppuhr initialisieren
26             stopwatch(MAX_SHUTTLE_WAIT, start);
27
28             //todo
29         }
30     }
31
32     /* Methode zur Belegung eines Sitzplatzes.
33      * Liefert false, falls das Shuttle bereits unterwegs ist,
```

```
34        * sonst wird der Passagier auf seinen Platz geleitet und true
35        * zurueckgegeben.
36        */
37      boolean assignSeat()
38      {
39          //todo
40      }
41
42      void startFlight()
43      {
44          //todo
45      }
46
47      //vorgegeben
48      showPassengerHisSeat()
49      {
50          //...
51      }
52
53      //vorgegeben
54      flyToRisaAndBack()
55      {
56          //...
57      }
58  }
```

3. Hilfsfunktion:

```
1
2  // Nicht-blockierende Funktion, die den Ablauf von m Minuten signalisiert
3  void stopwatch( int m, condition c )
4  {
5      // ... m Minuten warten
6      signal( c );
7      // ...
8  }
```

Die *Enterprise* besitzt 3 Shuttle-Hangars mit jeweils 4 einsatzbereiten Shuttles. In jedem Hangar kann immer nur ein Shuttle gleichzeitig durch das Hangar-Kraftfeld fliegen. Da O'Brien sich den Nachmittag für seine Frau Keiko Zeit nehmen möchte, sollen Sie die Planung für die Synchronisation der Start- und Landevorgänge der Shuttles übernehmen.

c) Entwerfen Sie ein **Petri-Netz**, das das Starten und Landen von (der Einfachheit halber) zwei Shuttles bezüglich eines Hangars modelliert (z.B. mit 5 Stellen und 4 Transitionen). Kennzeichnen Sie die initiale Markierung (alle Shuttles im Hangar) durch Einzeichnen der entsprechenden Token.

d) Implementieren Sie Ihr Modell nun für 4 Shuttles mittels **Semaphoren** in einem **C-ähnlichen Pseudocode**. Definieren Sie dabei insbesondere die Funktionen `init()`, `launch(int i)` und `land(int i)`.

e) Als Sie gerade mit der Lösung von Teil d) fertig geworden sind, fällt Ihnen ein Technischer Bericht von Chefingenieur Geordi La Forge in die Hände, in dem es heißt, dass die Shuttle-Hangars nun auch mit Kraftfeldern ausgerüstet sind, die 5 Shuttles gleichzeitig durchfliegen können. Warum ärgern Sie sich nun?

Aufgabe 5 (8 Minuten): Lifetime Funktion

Geben Sie die Lifetime-Funktion LTF zu einem Programm an, das durch den folgenden Referenzstring gekennzeichnet ist:

$$\omega = 123212343212345432123456543212$$

Die Zeit, die zwischen zwei Seitenanfragen vergeht, soll jeweils eine Zeiteinheit betragen. Die Seitenersetzung erfolge mittels FIFO-Strategie.

a) Erstellen Sie eine Tabelle, die die Seitenfehler für die jeweilige Fenstergröße m aufzeigt. Beachten Sie, dass erstmals aufgerufene Seiten als Fehler gewertet werden. Geben Sie für $1 \leqslant m \leqslant 8$ die Werte von $LTF(m)$ in einer Wertetabelle an und zeichnen Sie den Graphen.

b) Ermitteln Sie die optimale Fenstergröße m_{opt} anhand des Knie-Kriteriums.

Aufgabe 6 (7 Minuten): Working-Set

Ein Multiprogramming-System verwende die Working-Set-Strategie mit einer Fenstergröße von $h = 6$ zur Speicherverwaltung. Es sei der folgende Referenzstring eines Prozesses gegeben:

$$\omega = 124223412344132132145621211142 13$$

a) Konstruieren Sie einen Graphen, auf dessen x-Achse die Zeit $0 \leqslant t \leqslant 32$ und auf dessen y-Achse die Anzahl der momentan dem Prozess zugeteilten Seiten aufgetragen ist. Zwischen zwei Seitenanfragen vergeht jeweils eine Zeiteinheit.

b) Geben Sie die Working-Sets der folgenden Zeitpunkte an: 5, 19, 25

Aufgabe 7 (7 Minuten): WS und VOPT

Das von der WS-Strategie verwendete Rückwärtsfenster sei mit $RF(t,h)$ bezeichnet und das von der VOPT-Strategie verwendete Vorwärtsfenster mit $VF(t,h)$. Gegeben sei der folgende Referenzstring

$$\omega = 02135463747335531117$$

a) Bestimmen Sie $RF(8,6)$, $RF(12,6)$ sowie $VF(8,6)$ und $VF(12,6)$.

b) Zeichnen Sie die Anzahl der von WS und VOPT zugeteilten Speicherseiten als eine Funktion der Zeit für den Zeitraum $0 \leqslant t \leqslant 20$ und $h = 6$ auf.

Aufgabe 8 (8 Minuten): Scheduling Strategien

Gegeben sei foglende Scheduling-Strategie mit vier Prioritätsklassen. Jeder Klasse ist eine eigene FIFO-Warteschlange und ein eigenes Quantum zugeordnet:

Klasse	Quantum	Priorität
0	1	höchste
1	4	
2	8	
3	∞	niedrigste

Es wird *Process Aging* verwendet, d.h. wenn das Zeitquantum eines Prozesses abgelaufen ist, wird er an das Ende der Warteschlange mit nächstniedriger Priorität gestellt. Neuen Prozessen ist eine bestimmte Priorität zugeordnet. Sie werden an das Ende der Warteschlange ihrer Prioritätsklasse angehängt. Es wird am Ende jeder Zeiteinheit überprüft, welcher Prozess am Anfang der nicht leeren Warteschlange mit der höchsten Priorität steht und dieser dann im nächsten Schritt bearbeitet.

Ein Prozess, der zum Zeitpunkt t ankommt, wird erst ab dem Zeitpunkt $(t+1)$ berücksichtigt, d.h. er kann frühestens eine Zeiteinheit nach seiner Ankunft die CPU verwenden.

Stellen Sie für die ersten 15 Zeiteinheiten tabellarisch die Warteschlangen der einzelnen Prioritätsklassen sowie die Prozesszuteilung für die folgenden Prozesse dar:

Prozess	Ankunftszeit	Priorität	Laufzeit
A	0	3	3
B	1	2	5
C	1	1	4
D	4	1	2
E	5	0	6
F	10	1	4
G	12	0	2

Aufgabe 9 (7 Minuten): CPU-Scheduling

Gegeben seien die Prozesse $P_1, \ldots, P_7$, die alle auf einer CPU bearbeitet werden müssen. Die Länge der Prozesse ist der folgenden Tabelle zu entnehmen:

Prozess	P_1	P_2	P_3	P_4	P_5	P_6	P_7
Länge	2	5	7	8	3	2	6

Zum Zeitpunkt $t = 0$ sind alle Prozesse bereits im System vorhanden und sind nach aufsteigendem Index angekommen (also P_1 zuerst).

a) Geben Sie die entstehenden Warteschlagen für die Strategien FIFO, LIFO und SJF an und bestimmen Sie zu jeder Strategie die mittlere Wartezeit.

b) Wenden Sie nun das Round-Robin Verfahren für die obigen Prozesse und mit einem Zeitquantum von 5 an. Geben Sie für jeden Prozess an, zu welchen Zeitpunkten er die CPU belegt. Dabei sollen die Umschaltzeiten vernachlässigt werden. Bestimmen Sie auch hier die mittlere Wartezeit.

Aufgabe 10 (15 Minuten): Paging

In dieser Aufgabe geht es um die Speicherverwaltung mit Paging. Der Hauptspeicher eines Systems habe 5 Frames, die alle zu Beginn leer seien. Gegeben sei der Referenzstring $\omega = 25345689708943589351$.

Geben Sie für jede der folgenden Strategien die Belegung der Speicherstellen zu jedem Zeitpunkt und die Anzahl der Seitenfehler an. Orientieren Sie sich an die Definitionen und Vorgaben aus den Folien der Vorlesungen.

a) FIFO

b) LRU

c) LFU

d) Second Chance

e) Climb

f) OBL als Demand-Prepaging

g) Optimal

Aufgabe 11 (12 Minuten): Buddy-Systeme

Der Hauptspeicher eines Systems sei 64 MB groß und werde mit dem Buddy-System verwaltet. Der Reihe nach werden folgende Speicherbereiche angefordert bzw. freigegeben.

Operation	Name	Größe
Anforderung	A	4 MB
Anforderung	B	8 MB
Freigabe	A	
Anforderung	C	16 MB
Anforderung	D	8 MB
Anforderung	E	16 MB
Freigabe	D	
Anforderung	F	8 MB
Freigabe	C	

a) Stellen Sie die einzelnen Schritte in einem Baum dar (bei freier Wahl wird linker Ast gewählt).

b) Stellen sie die einzelnen Schritte nun für gewichtete Buddies dar (bei durch 3 teilbarer Speichergröße Gewichtung 1:2, sonst Gewichtung 1:3).

Aufgabe 12 (15 Minuten): Segmentierung

In dieser Aufgabe geht es um die Speicherverwaltung mit Segmentierung. Gegeben sei folgende Belegung eines Speichers mit Lücken der Größe 200, 100 und 300 kB:

200	belegt	100	belegt	300

Weiterhin seien vier Belegungensmethoden für Speicherplatzanforderungen gegeben: First-Fit, Rotating-First-Fit, Best-Fit, Worst-Fit.

a) Wie sieht der obige Speicher aus, wenn nacheinander vier Anforderungen der Größe 50, 150,100 und 300 kB ankommen? Notieren Sie für jede Strategie die freien Speicherbereiche nach jeder Anforderung (z.B. (200, 100, 300) für die obige Belegung), und geben Sie an, für welche Methoden die Anforderungen erfüllt werden können. Beachten Sie, dass die Daten jeweils linksbündig in einer Lücke abgelegt und einmal belegte Speicherbereiche nicht wieder freigegeben werden.

b) Sie dürfen nun die Reihenfolge der freien Speicherbereiche und die der Anforderungen vertauschen (aber keine Speicherbereiche zusammenfassen). Geben Sie für jede Strategie eine Speicherbelegung und die zugehörigen Anforderungen an, die für jeweils nur diese Methode erfolgreich arbeitet und für die übrigen Methoden nicht alle Anforderungen erfüllen kann. Geben Sie auch die Speicherbelegungen nach Abarbeitung der einzelnen Anforderungen an.

Aufgabe 13 (7 Minuten): Segmentierung

Für die Speicherverwaltung nach dem Segmentierungsverfahren sei für ein Programm die folgende Segmenttabelle gegeben (Länge in Speicherworten):

Segment	Basis	Länge
0	1320	320
1	562	58
2	310	120
3	750	60
4	830	150
5	990	110

a) Wie viele Speicherworte stehen dem Programm im physikalischen Speicher zur Verfügung?

b) Welche ist die kleinste und welche die größte für das Programm verfügbare physikalische Adresse?

c) Berechnen Sie zu den folgenden physikalischen Adressen jeweils die logischen Adressen. Welche Anfrage löst einen *Sementation Fault* aus?

1. 780
2. 1436
3. 580
4. 984
5. 420

Aufgabe 14 (9 Minuten): Bankers Algorithmus

Gegeben seien drei Prozesse P1, P2 und P3, die drei exklusive Betriebsmittel B1, B2 und B3 benutzen wollen. Es existieren 5 Exemplare von B1, 7 von B2 und 3 von B3.

Weiterhin gilt:

- $H_1(0) = (2,1,0)$, $Q_1^{max}(0) = (2,4,2)$
- $H_2(0) = (1,1,1)$, $Q_2^{max}(0) = (3,2,3)$
- $H_2(0) = (0,2,0)$, $Q_3^{max}(0) = (2,3,1)$

a) Prüfen Sie mit dem *Banker's Algorithmus*, ob sich das System in einem sicheren Zustand befindet.

b) Welche der folgenden Zuteilungen führen vom obigen Zustand ausgehend in einen sicheren Zustand, d.h. welche Zuteilung darf erlaubt werden?

i) $Q_1(1) = (2,2,2)$
ii) $Q_2(1) = (1,1,1)$
iii) $Q_3(1) = (2,0,1)$

Aufgabe 15 (20 Minuten): Deadlocks

Gegeben seien 5 Prozesse $P_1, \ldots, P_5$ und 4 Betriebsmittel (BM) $R_1, \ldots, R_4$:

R_1 (zwei Einheiten), R_2 (eine Einheit), R_3 (drei Einheiten) und R_4 (drei Einheiten)

Die Prozesse $P_1, \ldots, P_5$ haben folgende (verbleibende) Maximalanforderungen:

- P_1: eine Einheit von R_1, eine Einheit von R_2
- P_2: zwei Einheiten von R_1, eine Einheit von R_3
- P_3: eine Einheit von R_4
- P_4: zwei Einheiten von R_3, eine Einheit von R_4
- P_5: eine Einheit von R_2, drei Einheiten von R_3, eine Einheit von R_4

Prozess P_2 benutzt gerade eine Einheit von R_3, Prozess P_3 eine Einheit von R_2 und eine Einheit von R_4, Prozess P_4 benutzt eine Einheit von R_1 und zwei von R_3, Prozess P_5 benutzt eine Einheit von R_4.

Die Teilaufgaben b)-d) gehen vom Zustand nach Teilaufgabe a) aus. Belegen Sie für b)-d) im Falle eines unsicheren Zustands, welches Minimum an Forderungen zum Deadlock führt und stellen Sie dies mit einer geschlossenen Kette dar.

a) Stellen Sie den obigen Zustand durch einen Request-Allocation-Graphen dar. Löschen Sie den das Problem verursachenden Prozess und geben Sie dessen Ressourcen frei.

   Bearbeiten Sie jeweils einzeln die folgenden Teilaufgaben mit dem nun entstandenen Zustand.

b) Was passiert, wenn Prozess P_3 eine weitere Einheit von R_4 anfordert?

c) Was passiert, wenn Prozess P_5 eine weitere Einheit von R_4 anfordert?

d) Prozess P_5 verzichtet darauf, in Zukunft eine Einheit von R_2 anzufordern. Was passiert, wenn P_1 eine weitere Einheit von R_1 anfordert und Prozess P_5 eine weitere Einheit von R_4.

Aufgabe 16 (6 Minuten): Prozessfortschrittsdiagramme

Gegeben seien zwei Prozesse A und B. A benötigt zu seiner Ausführung zehn Zeiteinheiten, B neun Zeiteinheiten. Ferner stehen vier verschiedene Betriebsmittel (BM) zur Verfügung, die von den Prozessen während ihrer Ausführung benötigt werden. Die folgende Tabelle zeigt, in welchen Intervallen die beiden Prozesse die Betriebsmittel belegen:

Prozess	BM1	BM2	BM3	BM4
A	8-9	2-4	3-5	4-6
B	5-8	5-7	3-5	2-3

a) Skizzieren Sie das zugehörige Prozessfortschrittsdiagramm. Prozess A auf der x-Achse und Prozess B auf der y-Achse.

b) Kennzeichnen Sie unerreichbare, unmögliche und unsichere Bereiche.

c) Schreiben Sie einen sicheren Schedule für die beiden Prozesse auf und zeichnen Sie ihn in das Diagramm ein.

Aufgabe 17 (20 Minuten): Datenbanken

Gegeben sei folgender Schedule S:

Schedule S

T_1	T_2	T_3
$R_1(A)$		
	$R_2(A)$	
	$W_2(C)$	
		$R_3(B)$
	$R_2(C)$	
		$W_3(B)$
$R_1(C)$		
$R_1(B)$		

a) Kann der Schedule S von einem Zwei-Phasen-Sperrprotokoll erzeugt werden? Geben Sie den damit erzeugten Schedule zu S (in welcher Reihenfolge werden Sperren gesetzt und wieder freigegeben) (falls ja) oder eine Begründung (falls nein) an.

b) Geben Sie an, ob folgende Aussagen falsch oder richtig sind.

   i) Jeder vom Zwei-Phasen Sperrprotokoll erzeugte Schedule ist serialisierbar.

   ii) Beim Zwei-Phasen Sperrprotokoll treten Deadlocks nie auf.

c) Angenommen wir verwenden das Zeitstempel-Verfahren für den obigen Schedule S. Welche Transaktionen werden abgebrochen, wenn wir folgende Voraussetzung für die Zeitstempel der Transaktionen T_1, T_2, T_3 treffen?

   i) $TS(T_1) = 70, TS(T_2) = 40, TS(T_3) = 60$

   ii) $TS(T_1) = 50, TS(T_2) = 60, TS(T_3) = 70$

Aufgabe 18 (4 Minuten): Datenbanken

Gegeben sei folgender Schedule S:

Schedule S

T_1	T_2	T_3	T_4	T_5
$R_1(A)$				
	$R_2(B)$			
$W_1(B)$				
		$W_3(C)$		
	$W_2(C)$			
	$W_2(D)$			
			$R_4(A)$	
			$W_4(C)$	
				$W_5(D)$
			$W_4(A)$	

Falls S serialisierbar ist, dann geben Sie einen zu S äquivalenten seriellen Schedule an.