Musterlösung zur Zusatzübung

Lösung 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
P_1	ne	ru	ru	ru	ru	ru	wa	wa	wa	wa	re	ru	te	-	-	-	-	-	-	-	-	-
P_2	-	-	ne	re	re	re	re	ru	ru	ru	ru	bl	bl	bl	wa	wa	wa	wa	ru	ru	ru	te
P_3	-	ne	re	re	re	re	re	re	re	re	re	re	ru	ru	ki	-	-	-	-	-	-	-
P_4	-	-	-	-	ne	re	ru	bl	bl	bl	wa	wa	wa	wa	ru	ru	te	-	-	-	-	-

Lösung 2

Teil 2.a)

```
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  // Groesse des zu verwendenden Ringpuffers
8  #define CONST 50
9
10 // Zeiger auf Shared Memory, der als Ringpuffer verwendet wird
11 char *p_char;
12 // Zeiger auf Indizes, die Schreib-/Leseposition des Ringpuffers enthalten
13 int *p_int1, *p_int2;
14
15 // Erzeugerprozedur, die Zeichen aus einer Datei in den Ringpuffer schreibt
16 void proc1( char *n )
17 {
18         FILE *f;
19         int c = EOF;
20
21         // Oeffnen der durch n bezeichneten Datei zum Lesen
22         // Bei Erfolg mit Auslesen beginnen
23         if ( (f = fopen( n, "r" )) != NULL )
24                 // Solange noch Zeichen gelesen werden koennen
25                 while( ( c = fgetc( f )) != EOF )
26                 {
27                         // Warten solange Ringpuffer voll ist
28                         while ( ((*p_int1 + 1) % CONST) == *p_int2 );
29                         // Schreiben des aktuellen Elements in den Ringpuffer
30                         p_char[ *p_int1 ] = (char)c;
31                         // Weitersetzen des Schreibindex
32                         *p_int1 = (*p_int1 + 1) % CONST;
33                 }
34
35         // 2 maliges Schreiben des EOF-Zeichens in den Puffer,
36         // um beiden Verbrauchern das Ende der Datei mitzuteilen
37         while ( ((*p_int1 + 1) % CONST) == *p_int2 );
38         p_char[ *p_int1 ] = (char)c;
39         *p_int1 = (*p_int1 + 1) % CONST;
40
41         while ( ((*p_int1 + 1) % CONST) == *p_int2 );
```

```
42          p_char[ *p_int1 ] = (char)c;
43          *p_int1 = (*p_int1 + 1) % CONST;
44
45          // Datei schliessen
46          fclose(f);
47  }
48
49  // Verbraucherprozedur, die die aus dem Ringpuffer ausgelesenen Daten
50  // in eine Datei schreibt
51  void proc2( char *n )
52  {
53          FILE *f;
54          int c;
55
56          // Datei namens n wird zum Schreiben geoeffnet.
57          // Bei Misserfolg wird die Prozedur verlassen.
58          if ( ( f = fopen( n, "w" )) == NULL )
59                  return;
60
61          // Endlosschleife, die nach Auslesen des EOF-Zeichens per
62          // break-Anweisung verlassen wird
63          do
64          {
65                  // Warte solange Ringpuffer leer ist
66                  while( *p_int1 == *p_int2 );
67                  // Lese naechstes Zeichen
68                  c = p_char[*p_int2];
69                  // Weitersetzen des Leseindex
70                  *p_int2 = (*p_int2 + 1) % CONST;
71                  // Wenn gelesenes Zeichen nicht EOF ist, schreibe es in
72                  // die geoeffnete Datei; sonst verlasse die Schleife
73                  if ( c != EOF )
74                          fputc( c, f );
75                  else
76                          break;
77          } while( 1 );
78          // Datei schliessen
79          fclose( f );
80  }
81
82  int main( int argc, char *argv[] )
83  {
84          int c1, c2; // Hilfsvariable fuer Rueckgabewerte von fork()
85          int ID;     // ID des shared memory
86
87          // shared memory der Groesse des Ringpuffers plus der beiden Indizes
88          // vom Betriebssystem anfordern
89          ID = shmget( IPC_PRIVATE, CONST + 2*sizeof( int ), IPC_CREAT|0x1ff );
90          // p_char auf Anfang des shared memory setzen
91          p_char = (char *)shmat( ID, NULL, 0 );
92          // Zeiger der Indizes auf die letzten beiden int-Speicherplaetze
93          // im shared memory setzen
94          p_int1 = (int *)(p_char + CONST);
95          p_int2 = p_int1 + 1;
96
97          // Indizes mit 0 initialisieren
98          *p_int1 = 0;
```

```
 99          *p_int2 = 0;
100
101          // Kindprozess erzeugen
102          if ( (c1 = fork()) == 0 )
103          {
104                  // Dieser Prozess ist Kind
105                  // Noch einen Kindprozess erzeugen
106                  if ( (c2 = fork()) == 0 )
107                  {
108                          // Diesen Prozess als Erzeuger laufen lassen,
109                          // der den ersten Kommandozeilenparameter
110                          // als Dateiname uebergeben bekommt
111                          proc1( argv[1] );
112                  }
113                  else
114                  {
115                          // Diesen Prozess als Verbraucher laufen lassen,
116                          // der den zweiten Kommandozeilenparameter
117                          // als Dateiname uebergeben bekommt
118                          proc2( argv[2] );
119                  }
120          }
121          else
122          {
123                  // Diesen Prozess als Verbraucher laufen lassen,
124                  // der den dritten Kommandozeilenparameter
125                  // als Dateiname uebergeben bekommt
126                  proc2( argv[3] );
127          }
128  }
```

Teil 2.b)

Das Programm erwartet drei Dateinamen als Kommandozeilenparameter und erzeugt drei nebenläufige Prozesse: einen Erzeuger und zwei Verbraucher. Der Erzeuger öffnet die Datei mit dem Namen des ersten Kommandozeilenparameters, liest deren Daten Byte für Byte aus und schreibt sie in einen Ringpuffer, auf den alle Prozesse zugreifen können. Die beiden Verbraucher lesen jeweils Daten aus dem Ringpuffer aus und schreiben sie in die durch die zweiten und dritten Kommandozeilenparameter bezeichneten Dateien.

Teil 2.c)

Das Programm arbeitet nicht korrekt, da beide Verbraucher ungeschützt auf die Indexvariable p_int2 zugreifen, die die aktuelle Position des als nächstes zu lesenden Elements im Ringpuffer angibt. Dadurch kann es beispielsweise geschehen, dass dasselbe Element von beiden Verbrauchern verbraucht wird.

Lösung 3

Das Verfahren arbeitet nicht korrekt:

a) Mutual Exclusion wird nicht eingehalten:

Angabe eines möglichen Schedules, der eine Möglichkeit für beide Prozesse zeigt, gleichzeitig in den kritischen Bereich zu kommen.

```
1 // initial: turn=i, flag[i]=flag[j]=false;
2 // P_i                       P_j
3
```

```
 4                              flag[j]=true;
 5                              while (turn!=j) {
 6                                while (flag[j]);
 7      flag[i]=true;
 8      while (turn!=i)
 9      criticalSection(i);
10                                turn=j;
11                              }
12                              criticalSection(j)
13                              ...
14      ...
```

Dann sind beide Prozesse in ihrem kritischen Bereich.

b) Bounded Waiting wird nicht eingehalten:

```
 1  // initial: turn=j, flag[i]=flag[j]=false;
 2  // P_i                      P_j
 3
 4                              flag[j]=true;
 5                              while( turn!=j )
 6                              criticalSection(j)
 7     flag[i]=true;
 8     while (turn!=i) {
 9       while (flag[j])
10                              flag[j]=false;
11                              remainderSection(j)
12                              flag[j]=true;
13       // Pi bekommt CPU
14       // und scheitert
15       // nach wie vor
16       // an while-Bed.
17                              while( turn!=j )
18                              criticalSection(j)
19                              ...
```

P_i kann im obigen Szenario beliebig oft überholt werden.

c) Progress Requirement wird eingehalten:

Sobald ein Prozess sein Flag auf `true` gesetzt hat, ist es für den anderen nicht mehr möglich, die `turn` Variable zu ändern. Gilt oBdA initial `turn == i`, und P_i betritt als erster die Schleife, so gelangt P_i *immer* in seinen KB, da P_j `turn` nicht mehr ändern kann. Gilt initial `turn = j`, und P_i betritt als erster die Schleife, so hat er sein Flag auf `true` gesetzt. Entweder er kann die `turn` Variable auf `i` setzen, und gelangt danach in den KB, oder er wird von P_j überholt (siehe Teil b)), und P_j gelangt in den KB. Eine Blockade ist aber nicht möglich.

Lösung 4

Teil 4.a)

Implementierung mittels Bedingter Kritischer Regionen:

1. Initialisierung:

```
1  const int MAX_SHUTTLES = 12;
2  const int MAX_SEATS = 6;
```

```
3  const int MAX_SHUTTLE_WAIT = 60;
4
5  shared int free_bookings[MAX_SHUTTLES];
6  shared int free_seats[MAX_SHUTTLES];
7  shared bool flying[MAX_SHUTTLES];
8
9  //Initialisierung des Flugverkehrs
10 void init()
11 {
12     int i;
13     for(i=0; i<MAX_SHUTTLES; i++)
14     {
15         free_bookings[i] = MAX_SEATS;
16         free_seats[i] = MAX_SEATS;
17         flying[i] = FALSE;
18         shuttle(i);
19     }
20 }
```

2. Shuttleprozess:

```
1  shuttle(int i)
2  {
3      //MAX_SHUTTLE_WAIT Minuten Stoppuhr initialisieren
4      bool sig = FALSE;
5      stopwatch(MAX_SHUTTLE_WAIT, &sig);
6
7      while(TRUE)
8      {
9          //Auf Startsignal warten
10         region(free_seats[i], flying[i]) when(sig)
11         {
12             //Nur starten, falls >0 Passagiere an Bord
13             if(free_seats[i] < MAX_SEATS) flying[i] = TRUE;
14         }
15
16         if(flying[i]) flyToRisaAndBack();
17
18         region(flying[i], free_bookings[i], free_seats[i])
19         {
20             //initalen Zustand fuer das Shuttle herstellen
21             flying[i] = FALSE;
22             sig = FALSE;
23             stopwatch(MAX_SHUTTLE_WAIT, &sig);
24             free_seats[i] = MAX_SEATS;
25             free_bookings[i] = MAX_SEATS;
26         }
27     }
28 }
```

3. Buchung von Sitzplätzen:

```
1  /* Funktion zur Buchung eines Sitzplatzes in einem Shuttle.
2   * Liefert -1, falls gerade alle Plaetze belegt sind,
3   * sonst die entsprechende Shuttle Nummer.
4   */
5  int bookSeat()
```

```
 6  {
 7      //atomares Testen ggf. aller Shuttles
 8      region(free_bookings)
 9      {
10          int i;
11          for(i=0; i<MAX_SHUTTLES; i++)
12          {
13              if(free_bookings[i] > 0)
14              {
15                  free_bookings[i]--;
16                  return i;
17              }
18          }
19      }
20      return -1;
21  }
```

4. Belegung von Sitzplätzen eines bestimmten Shuttles:

```
 1  /* Funktion zur Belegung eines Sitzplatzes in Shuttle i.
 2   * Liefert FALSE, falls das Shuttle bereits unterwegs ist,
 3   * sonst wird der Passagier in das Shuttle geleitet und TRUE
 4   * zurueckgegeben.
 5   */
 6  bool assignSeat(int i)
 7  {
 8      if(flying[i]) return FALSE;
 9      region(free_seats[i], flying[i])
10      {
11          //Sitzplatz einnehmen
12          showPassengerHisSeat();
13          free_seats[i]--;
14          //Shuttle kann fruehzeitig starten, falls alle Plaetze belegt sind
15          if(free_seats[i] == 0) sig = TRUE;
16          return TRUE;
17      }
18  }
```

Teil 4.b)

Implementierung mittels Monitoren:

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          for(int i=0; i<MAX_SHUTTLES; i++)
10              shuttles[i] = new Shuttle(i);
11      }
12
13      /* Methode zur Buchung eines Sitzplatzes in einem Shuttle.
```

```
14       * Liefert -1, falls gerade alle Plaetze belegt sind,
15       * sonst die entsprechende Shuttle Nummer.
16       */
17      int bookSeat()
18      {
19          for(int i=0; i<MAX_SHUTTLES; i++)
20          {
21              int index = shuttles[i].decFreeBookings();
22              if(index != -1) return index;
23          }
24          return -1;
25      }
26  }
```

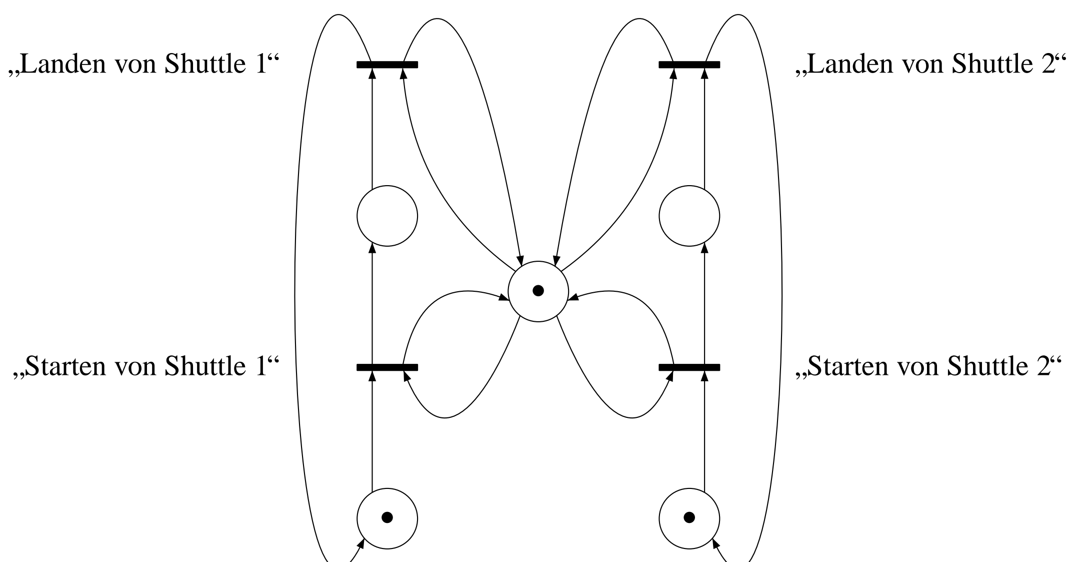
2. Monitor für das Shuttle:

```
 1  Monitor Shuttle
 2  {
 3      final int MAX_SEATS = 6;
 4      final int MAX_SHUTTLE_WAIT = 60;
 5
 6      int i,free_bookings, free_seats;
 7      boolean flying;
 8
 9      condition start;
10
11      Shuttle(int i)
12      {
13          this.i = i;
14          free_bookings = MAX_SEATS;
15          free_seats = MAX_SEATS;
16          flying = false;
17      }
18
19      /* Liefert -1, falls alle Plaetze belegt sind,
20       * sonst i und dekrementiert die freien Buchungen.
21       */
22      int decFreeBookings()
23      {
24          if(free_bookings <= 0) return -1;
25          free_bookings--;
26          return i;
27      }
28
29      void run()
30      {
31          while(true)
32          {
33              //MAX_SHUTTLE_WAIT Minuten Stoppuhr initialisieren
34              stopwatch(MAX_SHUTTLE_WAIT, start);
35              //Auf Startsignal warten
36              wait(start);
37
38              //Nur starten, falls >0 Passagiere an Bord
39              if(free_seats < MAX_SEATS) startFlight();
40          }
41      }
```

```
42
43
44      /* Methode zur Belegung eines Sitzplatzes.
45       * Liefert false, falls das Shuttle bereits unterwegs ist,
46       * sonst wird der Passagier auf seinen Platz geleitet und true
47       * zurueckgegeben.
48       */
49      boolean assignSeat()
50      {
51          if(flying) return false;
52
53          //Sitzplatz einnehmen
54          showPassengerHisSeat();
55          free_seats--;
56          //Shuttle kann fruehzeitig starten, falls alle Plaetze belegt sind
57          if(free_seats == 0) signal(start);
58          return true;
59      }
60
61      void startFlight()
62      {
63          flying = true;
64          if(flying) flyToRisaAndBack();
65
66          //initalen Zustand fuer das Shuttle wieder herstellen
67          flying = false;
68          free_seats = MAX_SEATS;
69          free_bookings = MAX_SEATS;
70      }
```

Teil 4.c)

Petri-Netz zur Synchronisation eines Shuttle-Hangars mit zwei Shuttles:



Teil 4.d)

Semaphor-Lösung für den Shuttle-Hangar mit vier Shuttles:

```
1  const int SHUTTLE_COUNT = 4;
2  bool flying[SHUTTLE_COUNT];
3  semaphor mutex;
4
5  //Initialisierung des Hangars
6  void init()
7  {
8      init(mutex, 1);
9      int i;
10     for(i=0; i<SHUTTLE_COUNT; i++)
11         flying[i] = FALSE;
12 }
13
14 //Startanfrage des i-ten Shuttles
15 void launch(int i)
16 {
17     if(!flying[i])
18     {
19         wait(mutex);
20         //starten
21         signal(mutex);
22         flying[i] = TRUE;
23     }
24 }
25
26 //Landeanfrage des i-ten Shuttles
27 void land(int i)
28 {
29     if(flying[i])
30     {
31         wait(mutex);
32         //landen
33         signal(mutex);
34         flying[i] = FALSE;
35     }
36 }
```

Teil 4.e)

Da sich nur 4 Shuttles in jedem Hangar befinden, können auch immer nur maximal $4 < 5$ Shuttles gleichzeitig das Hangar-Kraftfeld durchfliegen. Der Aufwand von Teil c) und d) war also umsonst!

Lösung 5

Teil 5.a)

m	1	2	3	2	1	2	3	4	3	2	1	2	3	4	5
1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	x	x	x		x	x	x	x		x	x		x	x	x
3	x	x	x					x			x	x	x	x	x
4	x	x	x					x							x
5	x	x	x					x							x
6	x	x	x					x							x
7	x	x	x					x							x
8	x	x	x					x							x

m	4	3	2	1	2	3	4	5	6	5	4	3	2	1	2
1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2		x	x	x		x	x	x	x		x	x	x	x	
3			x	x		x	x	x	x			x	x	x	
4				x	x	x	x	x	x				x	x	
5									x					x	x
6									x						
7									x						
8									x						

m	1	2	3	4	5	6	7	8
$LTF(m)$	1	1.3	1.67	2.3	3.75	5	5	5

Teil 5.b)

$m_{opt} = 6$

Lösung 6

Teil 6.a)

ω	1	2	4	2	2	3	4	1	2	3	4	4	1	3	2	1
6																
5																
4						x		x	x	x	x	x	x	x	x	x
3			x	x	x		x									
2		x														
1	x															
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

ω	3	2	1	4	5	6	2	1	2	1	1	1	4	2	1	3
6						x										
5					x		x	x	x							
4	x			x						x						x
3		x	x								x		x	x	x	
2												x				
1																
t	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Teil 6.b)

- $WS(5,6) = \{1,2,4\}$
- $WS(19,6) = \{1,2,3\}$
- $WS(25,6) = \{1,2,4,5,6\}$

Lösung 7

Teil 7.a)

- $RF(8,6) = \{1,3,4,5,6\}$
- $RF(12,6) = \{3,4,6,7\}$
- $VF(8,6) = \{3,4,5,7\}$
- $VF(12,6) = \{1,3,5\}$

Teil 7.b)

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
						4	6													
					5	5	4	6	7	7										
				3	3	3	5	4	6	6	7	7		5	5					7
			1	1	1	1	3	5	4	4	6	6	7	7	7	5	1	1	1	1
		2	2	2	2	2	1	3	5	5	4	4	4	4	4	7	5	5	5	5
WS	0	0	0	0	0	0	2	1	3	3	3	3	3	3	3	3	3	3	3	3
SF	x	x	x	x	x	x	x		x					x			x			x
ω	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7
SF	x	x	x	x	x	x	x		x					x			x			x
VOPT	0	2	1	3	3	3	3	3	3	3	3	3	3	3	3	3	1	1	1	7
					5	4	4	4	4	4	7			5	5					
							6		7	7										

Lösung 8

t	Kl. 0 RR(1)	Kl. 1 RR(4)	Kl. 2 RR(8)	Kl. 3 FIFO	Incoming	Running
0	-	-	-	-	A(3)→3	-
1	-	-	-	A(3)	B(5)→1,C(4)→2	A
2	-	C(4)	B(5)	A(2)	-	C
3	-	C(3)	B(5)	A(2)	-	C
4	-	C(2)	B(5)	A(2)	D(2)→1	C
5	-	C(1),D(2)	B(5)	A(2)	E(6)→0	C
6	E(6)	D(2)	B(5)	A(2)	-	E
7	-	D(2),E(5)	B(5)	A(2)	-	D
8	-	D(1),E(5)	B(5)	A(2)	-	D
9	-	E(5)	B(5)	A(2)	-	E
10	-	E(4)	B(5)	A(2)	F(4)→1	E
11	-	E(3),F(4)	B(5)	A(2)	-	E
12	-	E(2),F(4)	B(5)	A(2)	G(2)→0	E
13	G(2)	F(4)	B(5),E(1)	A(2)	-	G
14	-	F(4),G(1)	B(5),E(1)	A(2)	-	F
15	-	F(3),G(1)	B(5),E(1)	A(2)	-	F

Lösung 9

Teil 9.a)

<table>
<tr><th>Strat.</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th><th>32</th><th>33</th><th>WZ</th></tr>
<tr><td>FIFO</td><td colspan="2">P_1</td><td colspan="5">P_2</td><td colspan="7">P_3</td><td colspan="8">P_4</td><td colspan="3">P_5</td><td colspan="2">P_6</td><td colspan="6">P_7</td><td>13.9</td></tr>
<tr><td>LIFO</td><td colspan="6">P_7</td><td colspan="2">P_6</td><td colspan="3">P_5</td><td colspan="8">P_4</td><td colspan="7">P_3</td><td colspan="5">P_2</td><td colspan="2">P_1</td><td>14.4</td></tr>
<tr><td>SJF</td><td colspan="2">P_1</td><td colspan="2">P_6</td><td colspan="3">P_5</td><td colspan="5">P_2</td><td colspan="6">P_7</td><td colspan="7">P_3</td><td colspan="8">P_4</td><td>9.7</td></tr>
<tr><td>RR</td><td colspan="2">P_1</td><td colspan="5">P_2</td><td colspan="5">P_3</td><td colspan="5">P_4</td><td colspan="3">P_5</td><td colspan="2">P_6</td><td colspan="5">P_7</td><td colspan="2">P_3</td><td colspan="3">P_4</td><td>P_7</td><td>16</td></tr>
</table>

Teil 9.b)

Prozess	CPU-Einheiten	Wartezeit
P_1	1-2	0
P_2	3-7	2
P_3	8-12, 28-29	7+15
P_4	13-17, 30-32	12+12
P_5	18-20	17
P_6	21-22	20
P_7	23-27, 33	22+5

Mittlere Wartezeit: 16

Lösung 10

Teil 10.a)

First In First Out (FIFO)

ω	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
	2	5	3	4	4	6	8	9	7	0	0	0	4	3	5	8	9	9	9	1
		2	5	3	3	4	6	8	9	7	7	7	0	4	3	5	8	8	8	9
			2	5	5	3	4	6	8	9	9	9	7	0	4	3	5	5	5	8
				2	2	5	3	4	6	8	8	8	9	7	0	4	3	3	3	5
						2	5	3	4	6	6	6	8	9	7	0	4	4	4	3
							X	X	X	X			X	X	X	X	X			X
																				10PF

Teil 10.b)

LRU

ω	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
		2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5
			2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3
				2	2	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9
						2	3	4	5	6	6	6	7	0	8	9	4	4	4	8
							X	X	X	X			X	X	X					X
																				8PF

Teil 10.c)

Als Auswahlkriterium wird das Alter seit dem Beginn des Referenzstrings angewandt. In der Lösung wird die Anzahl der Zugriffe auf die Seiten in der jeweiligen zweiten Spalte angezeigt.

LFU

<table>
<tr><th>ω</th><th>2</th><th>5</th><th>3</th><th>4</th><th>5</th><th>6</th><th>8</th><th>9</th><th>7</th><th>0</th><th>8</th><th>9</th><th>4</th><th>3</th><th>5</th><th>8</th><th>9</th><th>3</th><th>5</th><th>1</th></tr>
<tr><td></td><td>2|0</td><td>5|0</td><td>3|0</td><td>4|0</td><td>4|0</td><td>6|0</td><td>8|0</td><td>9|0</td><td>7|0</td><td>0|0</td><td>0|0</td><td>0|0</td><td>4|0</td><td>3|0</td><td>3|0</td><td>3|0</td><td>3|0</td><td>3|1</td><td>3|1</td><td>1|0</td></tr>
<tr><td></td><td></td><td>2|0</td><td>5|0</td><td>3|0</td><td>3|0</td><td>4|0</td><td>6|0</td><td>8|0</td><td>9|0</td><td>7|0</td><td>7|0</td><td>7|0</td><td>0|0</td><td>4|0</td><td>4|0</td><td>4|0</td><td>4|0</td><td>4|0</td><td>4|0</td><td>3|1</td></tr>
<tr><td></td><td></td><td></td><td>2|0</td><td>5|0</td><td>5|1</td><td>3|0</td><td>4|0</td><td>6|0</td><td>8|0</td><td>9|0</td><td>9|0</td><td>9|1</td><td>9|1</td><td>9|1</td><td>9|1</td><td>9|1</td><td>9|2</td><td>9|2</td><td>9|2</td><td>9|2</td></tr>
<tr><td></td><td></td><td></td><td></td><td>2|0</td><td>2|0</td><td>5|1</td><td>3|0</td><td>4|0</td><td>6|0</td><td>8|0</td><td>8|1</td><td>8|1</td><td>8|1</td><td>8|1</td><td>8|1</td><td>8|2</td><td>8|2</td><td>8|2</td><td>8|2</td><td>8|2</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>2|0</td><td>5|1</td><td>5|1</td><td>5|1</td><td>5|1</td><td>5|1</td><td>5|1</td><td>5|1</td><td>5|1</td><td>5|2</td><td>5|2</td><td>5|2</td><td>5|2</td><td>5|3</td><td>5|3</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td><td>X</td><td>X</td><td>X</td><td></td><td></td><td>X</td><td>X</td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td colspan="20"></td><td>7PF</td></tr>
</table>

Teil 10.d)

Second Chance

ω	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
	2|0	5|0	3|0	4|0	4|0	6|0	8|0	9|0	7|0	0|0	0|0	0|0	4|0	3|0	3|0	3|0	3|0	3|1	3|1	1|0
		2|0	5|0	3|0	3|0	4|0	6|0	8|0	9|0	7|0	7|0	7|0	0|0	4|0	4|0	4|0	4|0	4|0	4|0	3|1
			2|0	5|0	5|1	3|0	4|0	6|0	8|0	9|0	9|0	9|1	9|1	9|1	9|1	9|1	9|1	9|1	9|1	9|1
				2|0	2|0	5|1	3|0	4|0	6|0	8|0	8|1	8|1	8|1	8|1	8|1	8|1	8|1	8|1	8|1	8|1
						2|0	5|1	5|1	5|1	5|1	5|1	5|1	5|1	5|1	5|1	5|1	5|1	5|1	5|1	5|1
							X	X	X	X			X	X						X

7PF

Teil 10.e)

Climb

ω	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
	2	5	3	4	4	6	8	9	7	0	0	0	4	3	5	8	9	9	9	1
		2	5	3	5	4	6	8	9	7	7	7	0	4	3	5	8	8	8	9
			2	5	3	5	4	6	8	9	8	9	7	0	4	3	5	3	5	8
				2	2	3	5	4	6	8	9	8	9	7	0	4	3	5	3	5
						2	3	5	4	6	6	6	8	9	7	0	4	4	4	3
							X	X	X	X			X	X	X	X	X			X

10PF

Teil 10.f)

Optimal

ω	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
	2	5	3	4	4	6	8	9	7	0	0	0	0	0	5	5	5	5	5	1
		2	5	3	3	4	6	8	9	9	9	9	9	9	0	0	0	0	0	5
			2	5	5	3	4	4	8	8	8	8	8	8	9	9	9	9	9	0
				2	2	5	3	3	4	4	4	4	4	4	8	8	8	8	8	9
						2	5	5	3	3	3	3	3	3	3	3	3	3	3	8
							X	X	X	X					X					X

6PF

Teil 10.g)

OBL als Demand-Prepaging-Variante

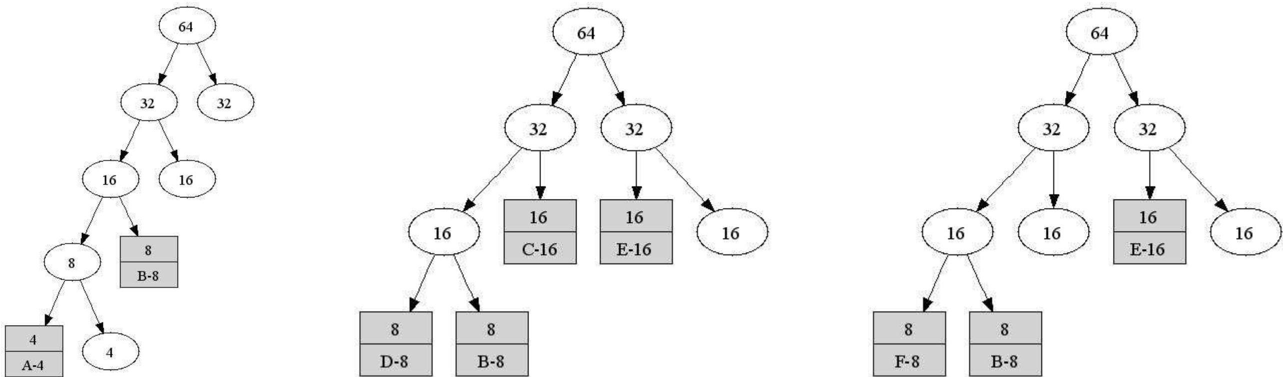
ω	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
	2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5	1
		2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3	5
			2	5	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9	3
				2	2	3	4	5	6	8	9	7	0	8	9	4	3	5	8	9
						2	9	4	0	6	6	6	3	0	8	9	4	4	4	8
							X		X				X		X					X

5PF

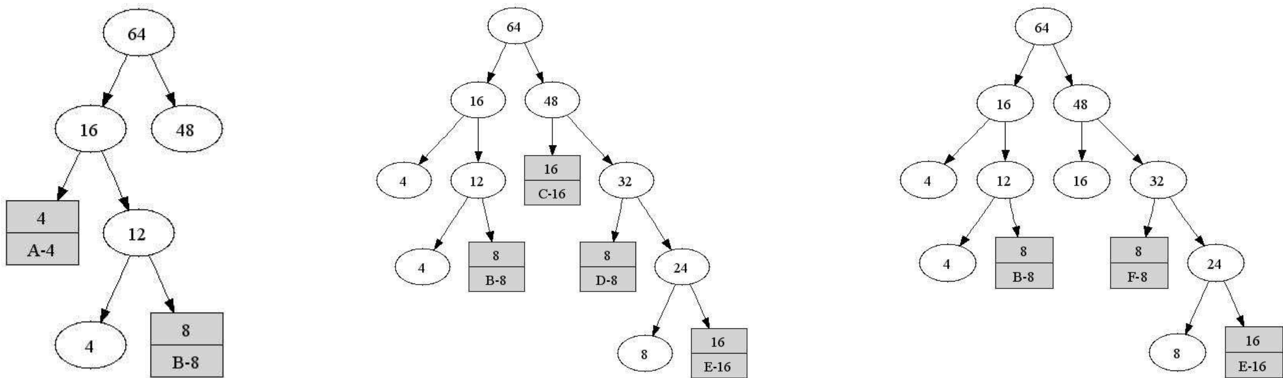
Lösung 11

In den Lösungen sind die wichtigsten Veränderungen in den Buddysystemen eingetragen.

Teil 11.a)



Teil 11.b)



Lösung 12

Teil 12.a)

FF

50		belegt		belegt		
50	150	belegt		belegt		
50	150	belegt	100	belegt		
50	150	belegt	100	belegt	300	

RFF

50			belegt		belegt	
50			belegt		belegt	150
50	100		belegt		belegt	150
50	100		belegt		belegt	150

BF

		belegt	50	belegt		
150		belegt	50	belegt		
150		belegt	50	belegt	100	
150		belegt	50	belegt	100	

WF

		belegt		belegt	50	
		belegt		belegt	50	150
100		belegt		belegt	50	150
100		belegt		belegt	50	150

Nur FF kann alle Anforderungen erfüllen.

Teil 12.b)

Zunächst einige Vorüberlegungen. Beispielsweise wird BF immer schiefgehen, sobald die 50 zuerst angefordert wird. WF geht immer schief, wenn die 300 nicht als erstes angefordert wird. Man erhält als eine Lösung:

- FF: Wir haben oben gesehen, dass die ursprünglichen Reihenfolgen nur für FF funktionieren.
- BF: originale Speicherreihenfolge, erste Anforderung 100, Rest beliebig.
- WF: Anforderungen in der Abfolge $300, 50, 150, 100$; Speicher beginnt mit 100, danach beliebig.
- RFF: Anforderungen in der Abfolge $50, 100, 300, 150$; Speicher wie im Original.

Lösung 13

Teil 13.a)

818 Speicherworte

Teil 13.b)

kleinste Adresse: 310
größte Adresse: 1639

Teil 13.c)

Physikalische Adr.	Logische Adr.
780	(3, 330)
1436	(0, 116)
580	(1, 18)
984	segfault
420	(2, 110)

Lösung 14

Teil 14.a)

V(0) = (2, 3, 2)

Sicherheitsprüfung:

- P1: $Q_1^{max} \nleq V(0)$
- P2: $Q_2^{max} \nleq V(0)$
- P3: $Q_3^{max} \leqslant V(0) \rightarrow$ V(0) = (2,5,2) $\rightarrow$ Markiere P3.
- P1: $Q_1^{max} \leqslant V(0) \rightarrow$ V(0) = (4,6,2) $\rightarrow$ Markiere P1.
- P2: $Q_2^{max} \leqslant V(0) \rightarrow$ V(0) = (5,7,3) $\rightarrow$ Markiere P2.

Das System ist in einem sicheren Zustand, da alle Prozesse markiert sind.

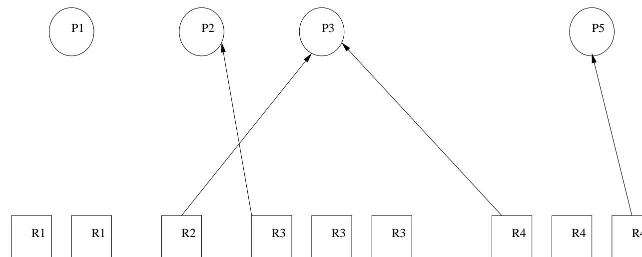
Teil 14.b)

Alle Zuteilungen führen in einen unsicheren Zustand.

Lösung 15

Teil 15.a)

Prozess P_4 fordert 4 Exemplare von R_3 an, da jedoch nur 3 Exemplare von R_3 existieren, kann diese Anforderung nie efüllt werden. Deshalb wird P_4 gelöscht. Daraus ergibt sich dann der folgende Request-Allocation-Graph:

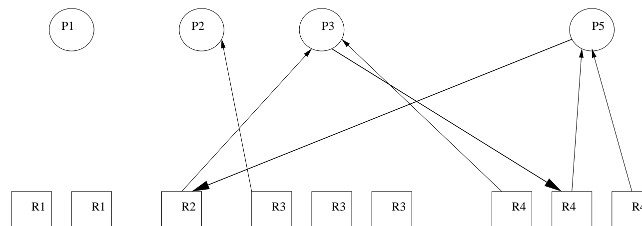


Teil 15.b)

Zustand sicher: P_2 und P_3 können zunächst terminieren. Danach können auch P_1 und P_5 terminieren:

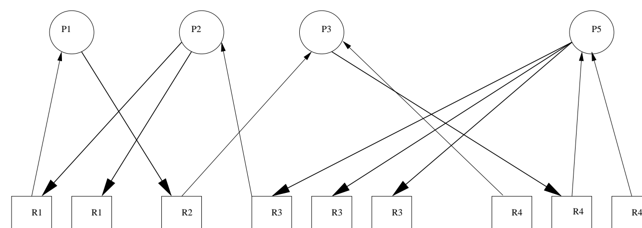
Teil 15.c)

Prozess P_2 kann terminieren, P_3 aber nicht. Deshalb P_1 und P_5 auch nicht. Deadlock entsteht, wenn P_5 R_2 anfordert und P_3 weitere R_4 fordert.



Teil 15.d)

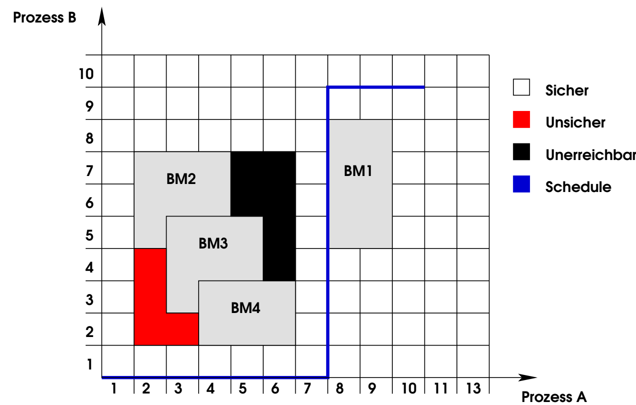
Deadlock entsteht, wenn P_1 R_2, $P3$ eine Einheit von R_4, P_5 3 Einheiten von R_3 und P_2 beide Einheiten von R_1 anfordert.



Lösung 16

Teil 16.a/b)

Prozessfortschrittsdiagramm:



Teil 16.c)

Ein möglicher zulässiger Schedule:

Zeit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Schedule 1	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	A	A	A

Lösung 17

Teil 17.a)

Der folgende Schedule zeigt, wie der Schedule S von einem Zwei-Phasen-Sperrprotokoll erzeugt werden kann.

S

T_1	T_2	T_3
RLock(A)		
R_1(A)	RLock(A)	
	R_2(A)	
	WLock(C)	
	W_2(C)	RLock(B)
		R_3(B)
	R_2(C)	
	Release	WLock(B)
		W_3(B)
RLock(C)		Release
R_1(C)		
RLock(B)		
R_1(B)		
Release		

Teil 17.b) Geben Sie an, ob folgende Aussagen falsch oder richtig sind.

1. Wahr
2. Falsch

Teil 17.c)

1. keine
2. T_1

Lösung 18

Ein möglicher äquivalenter serieller Schedule zu S ist der folgende Schedule:

Schedule S

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