Beispielsammlung mit Lösungsversuchen

Diese Beispielsammlung inkl. Lösungsversuche soll die VO-Prüfung Betriebssysteme erheblich erleichtern und beim Lernen Zeit sparen. Alle Lösungen sind nur Lösungsversuche, es besteht keine Garantie auf die Richtigkeit der Lösungen. Die Sammlung darf gerne ergänzt, sortiert, erweitert und verändert werden.

VO 182.711 4. Juni 2014

Prüfung Betriebssysteme

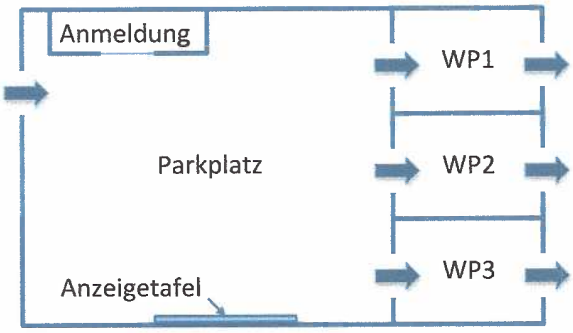
KNr. MNr. Zuname, Vorname

Ges.)(100) 1.)(35) 2.)(20) 3.)(45) Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation mit Semphoren (35)

Die Abläufe in einer Autoreparaturwerkstätte werden auf einem Computersystem durch eine Menge von Prozessen simuliert. Für die Simulation sind folgende Objekte von Bedeutung (siehe Skizze):



- Beim Anmeldeschalter (Anmeldung), den ein Auto beim Einfahren in den Werkstättenbereich passiert, melden Fahrer ihr Fahrzeug zur Reparatur an. Bei der Anmeldung bekommt der Fahrer eine Identifikationsnummer (ID), die er für die spätere Einfahrt auf einen Werkplatz benötigt. Beim Anmeldeschalter kann sich zu jedem Zeitpunkt maximal ein Fahrzeug aufhalten.
- Auf dem Parkplatz (gesamtes Areal zwischen linker Einfahrt und den Einfahrten zu WP1, WP2 und WP3) können sich maximal acht Fahrzeuge befinden. Fahrzeuge warten auf dem Parkplatz auf das Freiwerden eines Werkplatzes.
- Es gibt drei Werkplätze (WP1, WP2 und WP3), an denen Autos begutachtet bzw. repariert werden. Auf jedem dieser Plätze hat maximal ein Auto Platz.
- Auf dem Parkplatz befindet sich eine Anzeigetafel, die für jeden der drei Werkplätze die ID desjenigen Fahrzeugs anzeigt, das sich als nächstes für das Einfahren auf den Werkplatz bereithalten soll. Diese Reihenfolge kann von der Anmeldereihenfolge der Fahrzeuge abweichen. Die Aktualiserung der Anzeigetafel erfolgt mit folgendem Codestück:

```
P(display)
update_display()
V(display)
```

Schreiben Sie einen Prozess für einen Fahrer, der sein Fahrzeug unter Einhaltung der obigen Einschränkungen in der Werkstatt reparieren lässt. Es sollen beliebig viele solche Prozesse gestartet werden können. Verwenden Sie *Semaphore* zur Synchronisation der Prozesse und vermeiden Sie unnötige Einschränkungen der Parallelität. Ein Fahrer führt folgende Schritte bei einem Werkstattbesuch durch:

- Wenn Platz ist, meldet sich der Fahrer mit der Funktion `register()` beim Anmeldeschalter an und bekommt eine ID als Ergebniswert des Funktionsaufrufs.
- Der Fahrer parkt sein Auto (`park_car()`) auf dem Parkplatz und wartet auf die Zuweisung eines Werkplatzes: dabei ruft er abwechselnd die Funktionen `wait()` und `check_display()` auf, die das Warten bzw. das Ablesen der Anzeigetafel realisieren. An `check_display()` übergibt der Prozess die an der Anmeldung zugewiesene ID. Der Returnwert der Funktion gibt an, ob und für welchen Werkplatz die Nummer ID auf der Anzeigetafel steht – Returnwert 0: ID steht nicht auf der Anzeigetafel, Returnwert $x \in \{1, 2, 3\}$: Der Fahrer soll sich mit seinem Fahrzeug für das Einfahren auf Werkplatz WPx bereitmachen.
  Achten Sie darauf, dass Ihre Lösung das gleichzeitige Ablesen der Tafel durch mehrere Fahrer erlaubt.
- Der Fahrer wartet bis der ihm zugewiesene Werkplatz frei wird, fährt dann auf den Werkplatz (Aufruf von `enter_wp()` mit der Werkplatznummer als Parameter), lässt die Reparatur durchführen (Aufruf von `repair_car()`), und verlässt den Werkplatz durch die Ausfahrt, auf der Skizze rechts (Aufruf von `exit_wp()`, mit der Werkplatznummer als Parameter).

Lösen Sie die Synchronisationsaufgabe mit *Semaphoren* und geben Sie Initialisierungen für die Semaphore an. Achten Sie bei der Implementierung des Fahrerprozesses darauf, dass die Lösung die Ausführung und Synchronisation einer "beliebigen" Anzahl von Fahrern/Fahrzeugen erlaubt.

Code für Prozesse und Initialisierungen

`Initialisierungen:`

```
init(sem_anmeldung,1);
init(sem_parkpl,8);
init(sem_w1,1);
init(sem_w2,1);
init(sem_w3,1);
init(display,1);
init(rw,1);
int rc=0;
```

Prozess Fahrer:

```
int id;

P(sem_anmeldung);
id = register();
V(sem_anmeldung);

P(sem_parkpl);
park_car();

int erg=0;
for (;;) {
    wait();
    P(rw); rc++;
    if(rc==1){P(display);}
    V(rw);
    erg=check_display(id);
    P(rw); rc--;
    if(rc==0){V(display);}
    V(rw);
    if (erg!=0){ break;}
}

if(erg==1)
{ P(sem_w1);
}
else if(erg==2)
{ P(sem_w2);
}
else
{ P(sem_w3);
}
V(sem_parkpl);
enter_wp(erg);
repair_car();
exit_wp(erg);

if(erg==1)
{ V(sem_w1);}
else if(erg==2){
  V(sem_w2);}
else{
  V(sem_w3);
}
```

1 Synchronisation mit Semaphoren (30)

Der Betrieb eines kleinen Skigebiets soll in einer Simulation mit drei Arten von Prozessen, *Gondel*, *Skifahrer* und *Snowboardfahrer*, simuliert werden.

- *Gondel*: Diesen Prozess gibt es genau einmal im System. Er simuliert die Gondelbahn mit einer Gondel, die immer wieder Skifahrer und Snowboarder von der Talstation zur Bergstation befördert und dann leer zur Talstation zurückkehrt.
  Der Prozess simuliert das Öffnen und Schließen der Gondeltür durch Aufrufe der Funktionen *tuer_oeffnen()* bzw. *tuer_schliessen()*. Die Berg- bzw. Talfahrt der Gondel erfolgen durch Aufrufe der Funktionen *bergfahrt()* bzw. *talfahrt()*.

- *Skifahrer*: Von diesem Prozess kann es beliebig viele Instanzen im System geben. Er simuliert eine Folge von Berg- und Talfahrten eines Skifahrers.
  Die Funktionen *einsteigen()*, *aussteigen()* und *abfahren()* simulieren das Einsteigen in bzw. Aussteigen aus der Gondel sowie das Abfahren des Skifahrers auf der Piste.

- *Snowboardfahrer*: wie *Skifahrer*, allerdings mit Einschränkungen bezüglich der Gondelfahrt (siehe unten).

Ergänzen Sie die auf den Folgeseiten gegebenen Prozesstemplates mit *Semaphoroperationen*, um die Prozesse zu synchronisieren. Dabei sind folgende Punkte zu beachten:

- Die Gondel kann N Fahrgäste aufnehmen. Die Gondel fährt von der Talstation ab, wenn sie voll ist und beginnt ihre Talfahrt erst dann, wenn alle Leute am Gipfel ausgestiegen sind.

- Die Gondel verfügt über ausreichend Halterungen für Skier, aber nur über K Halterungen für Snowboards. Daher dürfen bei jeder Bergfahrt maximal K Snowboardfahrer mit der Gondel fahren. Der Rest sind Skifahrer. Der Betrieb der Gondel darf nicht durch das Warten auf Snowboardfahrer verzögert werden.

- Beim Ein- und Aussteigen kann die Türe der Gondel von maximal zwei Personen gleichzeitig passiert werden.

`Initialisierungen`

```
init(sem_cap,0);
init(sem_cap_b,K);
init(sem_enter,2);
init(sem_full,0);
init(sem_cap2,0);
```

```
/** Code Gondel **/

for(;;) {

  tuer_oeffnen();
  for(int i=0; i<N; i++){
    V(sem_cap); }
  for(int i=0; i<N; i++){
    P(sem_full); }
  tuer_schliessen();

  bergfahrt();

  tuer_oeffnen();
  for(int i=0; i<N; i++){
    V(sem_cap2);
  }
  for(int i=0; i<N; i++){
    P(sem_full); }
  tuer_schliessen();

  talfahrt();

}
```

(Wait for N exits)

```
/** Code Skifahrer **/

do {

  P(sem_cap);

  P(sem_enter);
  einsteigen();
  V(sem_enter);

  V(sem_full);

  P(sem_cap2);
  P(sem_enter);
  aussteigen();
  V(sem_enter);

  V(sem_full);

  abfahren();

} while (not_tired);
```

```
/** Code Snowboardfahrer **/

do {

  P(sem_cap_b);
  P(sem_cap);

  P(sem_enter);
  einsteigen();
  V(sem_enter);

  V(sem_full);

  P(sem_cap2);
  P(sem_enter);
  aussteigen();
  V(sem_enter);

  V(sem_full);
  V(sem_cap_b); //reset board counter

  abfahren();

} while (not_tired);
```

2 Deadlock (25)

In einem Computersystem gibt es vier Arten von Ressourcen, $R1 \ldots R4$, wobei folgende Stückzahlen von Ressourcen vorhanden sind: $R1 : 6$, $R2 : 4$, $R3 : 6$, $R4 : 5$. Für dieses Computersystem soll eine Deadlockanalyse durchgeführt werden. Zum Zeitpunkt der Analyse sind 5 Prozesse, $P1 \ldots P5$, aktiv. Die Prozesse belegen folgende Ressourcen:

$P1$: keine
$P2$: $R1 : 1$, $R2 : 2$, $R3 : 1$, $R4 : 1$
$P3$: $R1 : 2$, $R3 : 1$, $R4 : 1$
$P4$: $R1 : 1$, $R2 : 1$, $R4 : 3$
$P5$: $R2 : 1$, $R3 : 3$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 \\ 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 3 \\ 0 & 1 & 3 & 0 \\ \hline 4 & 4 & 5 & 5 \end{pmatrix} \quad R = (6, 4, 6, 5)$$

Die Prozesse fordern zum betrachteten Zeitpunkt folgende zusätzliche Ressourcen an:

$P1$: $R1 : 2$, $R3 : 1$
$P2$: $R1 : 1$, $R3 : 3$
$P3$: $R1 : 1$
$P4$: $R1 : 1$, $R4 : 1$
$P5$: $R1 : 1$, $R2 : 2$, $R3 : 1$, $R4 : 1$

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

$A = (0, 0, 0, 0)$
$\Rightarrow$ marked
$\rightarrow Q_1 = (0, 0, 0, 0)$

Stellen Sie für das gegebene Szenario fest, ob ein Deadlock vorliegt. Wenden Sie dabei einen geeigneten Algorithmus zur Deadlockerkennung an. Im Falle eines Deadlocks geben Sie bitte die am Deadlock beteiligten Prozesse an.

$V = (2, 0, 1, 0)$

P3:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 3 \\ 0 & 1 & 3 & 0 \\ \hline 2 & 4 & 4 & 4 \end{pmatrix} \quad Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix} \quad V = (4, 0, 2, 1)$$

P4:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ \hline 1 & 3 & 4 & 1 \end{pmatrix} \quad Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 \end{pmatrix} \quad V = (5, 1, 2, 4)$$

Deadlock!

P2: Terminieren

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ \hline 0 & 1 & 3 & 0 \end{pmatrix} \quad Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 \end{pmatrix} \quad V = (6, 3, 3, 5)$$

PS:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$V = (6, 4, 6, 5)$$

4 Memory Management (25)

Das betrachtete Speicherverwaltungssystem verwendet zur Adressierung 16-Bit Adressen. Für die angegebenen virtuellen Speicheradressen sind, in Abhängigkeit von der Adressierungsart, die entsprechenden physikalischen Adressen zu ermitteln. Von den angebenen Adressen sind die niederwertigen 8 Bit der Offset der Adresse und die höherwertigen 8 Bit die Segmentnummer bzw. die Seitennummer. Bei Paging sind alle Seiten 256 Bytes (Hexadezimal 0x100) lang. Alle Werte mit führendem **0x** sind als Hexadezimalzahl angegeben, Werte mit abschließendem **b** sind als Binärzahl zu interpretieren. Ergibt sich bei der Umwandlung eine ungültige Adresse, so schreiben Sie bitte **ungültig** in das entsprechende Feld.

Es werden folgende Begriffe (englische Notation) verwendet:

Base	Basisadresse des Segmentes
Entry	Eintrag in der Adressübersetzungstabelle
Frame#	Seitenrahmennummer (im physischen Speicher)
Length	Länge des Segmentes
Page#	Seitennummer (im virtuellen Speicher)

8 4 2 1
0 0 0 0

a) Paging — Assoziativer Zugriff (associative mapping)

A 10
B 11
C 12
D 13
E 14

Adressübersetzungstabelle:

Page#	Frame#
00010110b	0x41
00001000b	0x5A
01011110b	0x13
00100100b	0xAF
00010001b	0x20

0x16
0x08
0x5E
0x11
0x24

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x5E41	0x2041
0x24AB	0xAFAB
0x108A	ungültig
0x16B2	0x41B2

b) Paging — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:

Entry	Frame#
0	0x24
1	0xB1
2	0x77
3	0x86
4	0xF4
5	0xCC
6	0x01

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x0311	0x8611
0x14AC	ungültig
0x0512	0xCC12
0x08A8	ungültig

c) Segmentierung — Direkter Zugriff (direct mapping)

Adressübersetzungstabelle:

Entry	Base	Length
0	0x2840	0x004
1	0x0563	0x0FF
2	0x72AB	0x100
3	0x0300	0x010
4	0x11FD	0x0F0
5	0x4444	0x030
6	0x3112	0x060
7	0x4220	0x010

Zu berechnende Adressen:

Virtuelle Adresse	Physikalische Adresse
0x01F2	0xF2 + 0x0563
0x0305	0x03 + 0x0300
0x0010	0x10 + 0x2840
0x04A0	0xA0 + 0x11FD
0x4222	ungültig

Bewertung: 1 Pluspunkt pro richtiger Lösung, 1 Punkt Abzug pro falscher Lösung.

d) Verständnisfragen (12)

Kreuzen Sie bitte die richtigen Antworten an. Achtung! Falsche Antworten werden negativ gewertet. (Bewertung: 2 Pluspunkte pro richtiger Antwort, 2 Punkte Abzug pro falscher Antwort.)

- Bei folgender Speichverwaltungstechnik tritt sowohl *interne* als auch *externe* Fragmentierung auf.
  - ◯ fixed partitioning
  - ◯ simple segmentation
  - ◯ virtual memory with combination of paging and segmentation
  - ◯ virtual memory paging
  - ◯ dynamic partitioning
  - ◯ simple paging
- Zu welchen Effekten kann es bei *Segmentierung* kommen?
  - ◯ Internal Fragmentation
  - ◯ External Fragmentation
- In einem fehlerfreien System können zwei oder mehrere virtuelle Adressen auf eine physikalische Adresse abgebildet sein.
  - ◯ richtig
  - ◯ falsch
- Eine virtuelle Adresse verweist immer auf eine Seite auf dem Sekundärspeicher.
  - ◯ richtig
  - ◯ falsch
- Um die externe Fragmentierung zu reduzieren, muss man die *Page Size* vergrößern.
  - ◯ richtig
  - ◯ falsch
- Ein virtueller Speicher kann sowohl mit Paging als auch mit Segmentierung implementiert werden.
  - ◯ richtig
  - ◯ falsch

2 Memory Management (20)

Was ist eine Inverted Page Table?

Die Page Table eines kleinen Computersystems ist als Inverted Page Table (IPT) mit acht Einträgen realisiert. Beim Auftreten von Page Faults wird in diesem Computersystem der Clock Algorithmus als globale Page Replacement Strategie eingesetzt.

Im zu lösenden Beispiel ist eine Ausgangsbelegung der Page Table (links), sowie eine Folge von Zugriffen auf den virtuellen Speicher, charakterisiert durch Prozessnummer (PID), adressierte Seite (page) und Offset, gegeben. Simulieren Sie die Ausführung der Zugriffsfolge von links nach rechts und tragen Sie für jeden Zugriff auf den virtuellen Speicher (a) die entsprechende physikalische Adresse, (b) den Zustand der IPT und (c) den Wert des Replacement Pointers (rep. pointer) für den Clock Algorithmus nach dem Speicherzugriff an (Sie brauchen bei jedem Schritt nur die Werte in der IPT anzugeben, die sich beim aktuellen Zugriff geändert haben).

PID	page	offset
1	0	00a0

phys. address: miss

i	PID	page	use
0	1	5	1
1	3	1	0
2	2	3	0
3	2	1	0
4	1	2	1
5	1	3	1
6	1	4	1
7	3	3	1

rep. pointer: 5

PID	page	offset
1	0	00a2

phys. address: 1 0092

i	PID	page	use
0	1	5	0
1	1	0	1
2			
3			
4			
5	1	3	0
6	1	4	0
7	3	3	0

rep. pointer: 2

PID	page	offset
1	4	0002

phys. address: 6 0002

i	PID	page	use
0			
1			
2			
3			
4			
5			
6			
7			

rep. pointer: 2

PID	page	offset
2	3	fffe

phys. address: 2fffe

i	PID	page	use
0	1	4	1
1			
2			
3			
4			
5			
6			
7			

rep. pointer: 2

PID	page	offset
2	4	0000

phys. address: miss

i	PID	page	use
0	2	3	1
1			
2	2	3	0
3	2	4	1
4			
5			
6			
7			

rep. pointer: 2

PID	page	offset
2	4	0000

phys. address: miss

i	PID	page	use
0			
1			
2	2	3	0
3	2	4	1
4			
5			
6			
7			

rep. pointer: 4

2 Memory Management (20)

Gegeben ist eine Folge von Speicherreferenzen eines Prozesses. Die Entwicklung des *Working Sets* des Prozesses während der Ausführung soll für unterschiedliche *Window Sizes* (Δ) beobachtet werden. Tragen Sie dazu in der unten stehenden Tabelle für jeden Zeitpunkt das Working Set nach der Referenzierung der angegebenen Speicherseite an.

	Referenzierte	Window Size, Δ		
log. Zeit	Seite	2	3	4
1	16	16	16	16
2	17	16,17	16,17	16,17
3	18	17,18	16,17,18	16,17,18
4	20	18,20	17,18,20	16,17,18,20
5	22	20,22	18,20,22	17,18,20,22
6	20	20,22	20,22	18,20,22
7	18	18,20	18,20,22	18,20,22
8	22	18,22	18,20,22	18,20,22
9	20	20,22	18,20,22	18,20,22
10	18	18,20	18,20,22	18,20,22
11	23	18,23	18,20,23	18,20,22,23
12	23	23	18,23	18,20,23
13	17	17,23	17,23	17,23,18
14	24	17,24	24,17,23	24,17,23
15	23	24,23	24,17,23	24,17,23
16	24	24,23	24,23	24,17,23

Nehmen Sie an, dass das Lokalitätsverhalten des im Beispiel verwendeten Prozesses typisch für die Prozesse auf einem Rechner sind. Diskutieren Sie anhand der Beobachtungen aus der obigen Tabelle, welchen Wert für Δ Sie bei der Implementierung der Speicherverwaltung für diesen Rechner verwenden würden.

2 Deadlock Avoidance – Banker's Algorithm (20)

In einem Computersystem gibt es drei Prozesse (P_1, P_2, P_3) und drei Arten von Ressourcen (R_1, R_2, R_3). Der *Resource Vector* R=(7, 8, 7) beschreibt die Anzahl der vorhandenen Ressourcen. Prozesse verwenden die Operationen *get*(r_1, r_2, r_3) bzw. *free*(r_1, r_2, r_3), um r_i Ressourcen der Ressourcenart R_i ($i = 1 \ldots 3$) anzufordern bzw. freizugeben.

Die folgende Abbildung zeigt für jeden der drei Prozesse die Folge von *get* und *free*-Operationen, die bei jeder Prozessabarbeitung durchgeführt werden.

P_1	P_2	P_3
get (4, 2, 0)	get (0, 0, 1)	get (2, 1, 0)
free (2, 1, 0)	get (2, 0, 0)	free (1, 0, 0)
get (0, 1, 3)	free (1, 0, 0)	get (0, 1, 3)
get (1, 1, 0)	get (0, 3, 0)	free (0, 0, 1)
free (2, 1, 1)	get (1, 0, 0)	get (3, 0, 1)
get (1, 0, 2)	free (1, 2, 0)	get (1, 1, 0)
free (2, 2, 1)	get (1, 0, 0)	free (2, 2, 2)
free (0, 0, 3)	free (2, 1, 1)	free (3, 1, 1)

Nehmen Sie an, dass die drei ersten Operationen jedes Prozesses abgearbeitet wurden (d.h., die Abarbeitung jedes Prozesses befindet sich an der strichlierten Linie). Als nächster Schritt soll die vierte Operation von P_1 (grau hinterlegte Operation) durchgeführt werden.

Verwenden Sie den Banker's Algorithmus, um festzustellen, ob die vierte Operation von P_1 durchgeführt werden soll. Geben Sie alle erforderlichen Vektoren und Matrizen an und führen Sie den Banker's Algorithmus schrittweise durch, d.h. geben Sie für jeden Schritt die Werte der Elemente aller relevanten Matrizen und Vektoren an.

$R = (7, 8, 7)$

$$C = \begin{pmatrix} 4 & 3 & 4 \\ 2 & 3 & 1 \\ 5 & 3 & 3 \end{pmatrix} \qquad A = \begin{pmatrix} 2 & 2 & 3 \\ 1 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

$$C - A = N = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 3 & 0 \\ 4 & 1 & 0 \end{pmatrix}$$

$Q_1 = (1, 1, 0)$

$$A_2 = A + Q_1 = \begin{pmatrix} 3 & 3 & 3 \\ 1 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$
$$\begin{matrix} 5 & 5 & 7 \end{matrix}$$

$$N_2 = C - A_2 = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 1 \\ 5 & 3 & 3 \end{pmatrix}$$

$V = (2, 3, 0)$

$\hookrightarrow$ P1 kleiner gleich!

$\Rightarrow$ P1:

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix} \qquad N = \begin{pmatrix} 0 & 0 & 0 \\ 2 & 3 & 1 \\ 5 & 3 & 3 \end{pmatrix}$$
$$\begin{matrix} 2 & 2 & 4 \end{matrix}$$

$V = R - (2, 2, 4) = (5, 6, 3) \Rightarrow$ ~~nicht kleiner~~

$\rightarrow$ P2 kleiner ~~oder UNSAT~~

$\Rightarrow$ P2 :

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 3 \end{pmatrix} \qquad N = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 5 & 3 & 3 \end{pmatrix}$$

$V = R - (1,2,3) = (6,6,4)$ P3 bzbner

$\Rightarrow$ P3

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad N = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$V = R = (7,8,7)$ Safe

2 Deadlock Avoidance – Banker's Algorithm (20)

In einem Computersystem gibt es drei Prozesse (P_1, P_2, P_3) und drei Arten von Ressourcen (R_1, R_2, R_3). Der *Resource Vector* R=(4, 5, 4) beschreibt die Anzahl der vorhandenen Ressourcen. Prozesse verwenden die Operationen *get*(r_1, r_2, r_3) bzw. *free*(r_1, r_2, r_3), um r_i Ressourcen der Ressourcenart R_i ($i = 1 \ldots 3$) anzufordern bzw. freizugeben.

Die folgende Abbildung zeigt für jeden der drei Prozesse die Folge von *get* und *free*-Operationen, die bei jeder Prozessabarbeitung durchgeführt werden.

R(4,5,4)

C=max(get u. free at one time)

P_1	P_2	P_3
get (1, 1, 2)	get (2, 1, 0)	get (0, 1, 0)
get (1, 0, 1)	free(2, 0, 0)	get (1, 0, 0)
free(1, 0, 2)	get (1, 0, 1)	get (0, 1, 0)
----	----	----
get (1, 0, 1)	free(0, 1, 0)	get (2, 2, 2)
free(0, 1, 2)	get (2, 2, 1)	free(2, 2, 2)
get (0, 0, 3)	get (1, 0, 0)	get (1, 0, 2)
free(2, 0, 3)	free(2, 2, 1)	free(2, 2, 2)
	free(2, 0, 1)	

Nehmen Sie an, dass die drei ersten Operationen jedes Prozesses abgearbeitet wurden (d.h., die Abarbeitung jedes Prozesses befindet sich an der strichlierten Linie). Als nächster Schritt soll die vierte Operation von P_1 (grau hinterlegte Operation) durchgeführt werden.

Verwenden Sie den Banker's Algorithmus, um festzustellen, ob die vierte Operation von P_1 durchgeführt werden soll. Geben Sie alle erforderlichen Vektoren und Matrizen an und führen Sie den Banker's Algorithmus schrittweise durch, d.h. geben Sie für jeden Schritt die Werte der Elemente aller relevanten Matrizen und Vektoren an.

$R=(4,5,4)$

$Q_1=(1,0,1)$

$$C=\begin{pmatrix}2&1&3\\4&2&2\\3&4&2\end{pmatrix} \quad A_1=\begin{pmatrix}1&1&1\\1&1&1\\1&2&0\end{pmatrix}$$

$$C-A=N_1=\begin{pmatrix}1&0&2\\3&1&1\\2&2&2\end{pmatrix} \quad N_{ki}=C_{ki}-A_{ki}$$

$$A_2=Q_1+A_1=\begin{pmatrix}2&1&2\\1&1&1\\1&2&0\end{pmatrix} \quad N_2=C-A_2=\begin{pmatrix}0&0&1\\3&1&1\\2&2&2\end{pmatrix}$$

$$\begin{matrix}4&4&3\end{matrix}$$

$$V=R-A_2=(0,1,1)=V$$

~~$Q_1 = (1,0,1)$~~ $R(4,5,4)$

$$C = \begin{pmatrix} 2 & 1 & 3 \\ 4 & 2 & 2 \\ 3 & 4 & 2 \end{pmatrix} \qquad A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \end{pmatrix}$$

$$\begin{matrix} 3 & 4 & 2 \end{matrix}$$

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

~~$V = R - (3,4,2) = (1,1,2)$~~

$Q_1 = (1,0,1)$

$$A_2 = A + Q_1 = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \end{pmatrix}$$

$$N_2 = C - A_2 = \begin{pmatrix} 0 & 0 & 1 \\ 3 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} \qquad + \begin{matrix} 4 & 4 & 3 \end{matrix}$$

$$V = R - (4,4,3) = (0,1,1)$$

$N_2(P1) = (0,0,1)$

↳ kleiner $(0,1,1)$ neu P1

$\Rightarrow$ P1

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

$$\begin{matrix} 2 & 3 & 1 \end{matrix}$$

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

$V = R - (2,3,1) = (2,2,3)$

$P_3 < (2,2,3)$

$\Rightarrow P_3$ neu

$\Rightarrow P3:$

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

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

$$V = R - (1,1,1) = \underline{\underline{(3,4,3)}}$$

$$\underline{\underline{P_2 < (3,4,3)}}$$

$\Rightarrow P2:$

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$N = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\underline{\underline{V = (4,5,4)}} \Rightarrow \underline{\underline{\text{Safe}}}$$

Deadlock-Erkennung (13)

In einem Computersystem laufen fünf Prozesse, die gemeinsame Ressourcen aus vier Ressourcenkategorien verwenden. Die vorhandenen Ressourcen sind durch den Vektor $R = (6, 5, 9, 6)$ gegeben. Die unten gegebenen Matrizen beschreiben die aktuellen Ressourcenanforderungen und -allokation der fünf Prozesse. Führen Sie den Algorithmus zur Deadlock-Erkennung durch, um festzustellen, ob im gegebenen System ein Deadlock vorliegt.

$$Q = \begin{pmatrix} 2 & 0 & 6 & 4 \\ 0 & 1 & 3 & 0 \\ 3 & 0 & 1 & 1 \\ 1 & 4 & 3 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 4 & 2 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 5 & 3 & 5 & 6 \end{pmatrix} \quad V = (1, 2, 4, 0)$$

P3

P2:

$$Q = \begin{pmatrix} 2 & 0 & 6 & 4 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 1 & 4 & 3 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 4 & 3 & 1 & 4 \end{pmatrix} \quad V = (2, 2, 8, 2)$$

R=

P1:

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 1 & 4 & 3 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 3 & 3 & 0 & 3 \end{pmatrix} \quad V = (3, 2, 9, 3)$$

R=

P3:

$$Q \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 4 & 3 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 1 \end{pmatrix} \quad V = (6, 4, 9, 5)$$

$$Q = \begin{pmatrix} 2 & 0 & 6 & 4 \\ 0 & 1 & 3 & 0 \\ 3 & 0 & 1 & 1 \\ 1 & 4 & 3 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \qquad A = \begin{array}{c} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 4 & 2 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \\ \hline 5\ 3\ 5\ 6 \end{array}$$

$$V = (1, 2, 4, 0)$$

P2:

$$Q = \begin{pmatrix} 2 & 0 & 6 & 4 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix} \qquad A = \begin{array}{c} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \\ \hline 4\ 3\ 1\ 4 \end{array}$$

$$V = (2, 2, 8, 2)$$

P5:

$$Q = \begin{pmatrix} 2 & 0 & 6 & 4 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad A = \begin{array}{c} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ \hline 4\ 2\ 1\ 3 \end{array}$$

$$V = (2, 3, 8, 3) \Rightarrow \underline{\text{Deadlock}}$$

P1 Terminieren!

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{array}{cccc} 3 & 2 & 0 & 2 \end{array}$$

$$V = (3, 3, 9, 4)$$

P3:

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{array}{cccc} 0 & 0 & 0 & 0 \end{array}$$

$$V = (6, 5, 9, 6)$$

P5 im Deadlock

$\Rightarrow$ Lösung P1 terminieren

2 Deadlock (20)

Gegeben sind zwei Prozesse, P_1 und P_2, die die beiden Ressourcen R_1 und R_2 unter Mutual Exclusion verwenden.

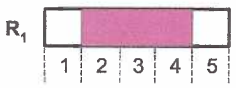
Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann im Prozessfortschrittsdiagramm als Kantenzug zwischen den Punkten *start* und *end* eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehls.

Unterhalb bzw. links der Diagrammachsen sind Balken abgedruckt, in denen die Reservierugen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

Teilaufgabe A

1. Tragen Sie die Reservierungen von Ressourcen für P_1 bzw. P_2 in die entsprechenden Balken ein. Dabei gilt eine Ressource ab Start der Anweisung `get()` als belegt und nach Beendigung der Anweisung `free()` als wieder freigegeben:

```
2: get(R1)
3: ...
4: free(R1)
```



2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung wegen Ressourcenkonflikten nicht möglich ist.

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug bei einer deadlockfreien Abarbeitung von P_1 und P_2 nicht passiert werden dürfen.

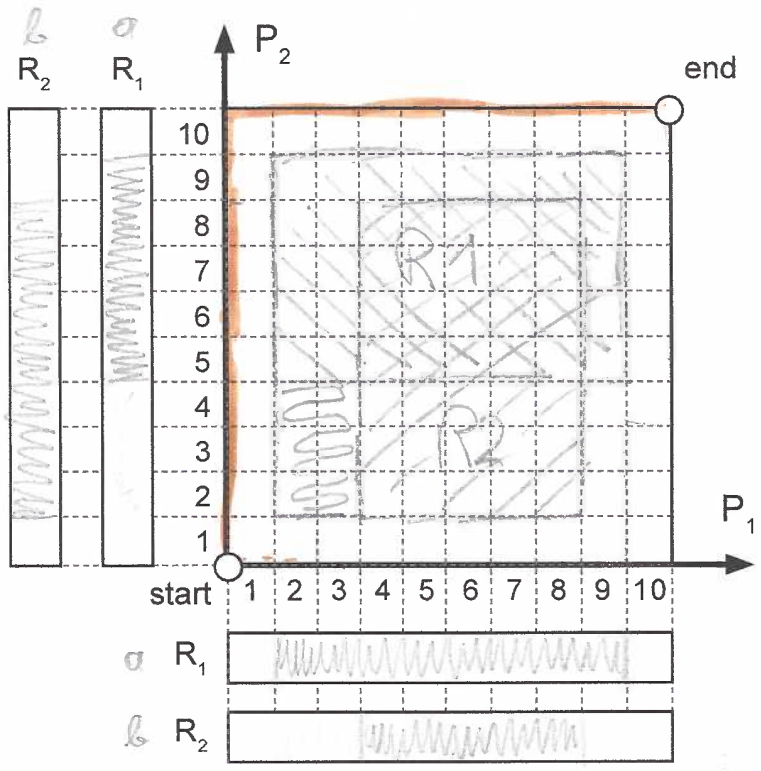
4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.

Program P_1:

```
 1: read(buf);
 2: get(R1);
 3: copy(R1, buf);
 4: get(R2);
 5: use(R1, R2);
 6: copy(buf2, R2);
 7: use2(R2, R1);
 8: free(R2);
 9: free(R1);
10: return;
```

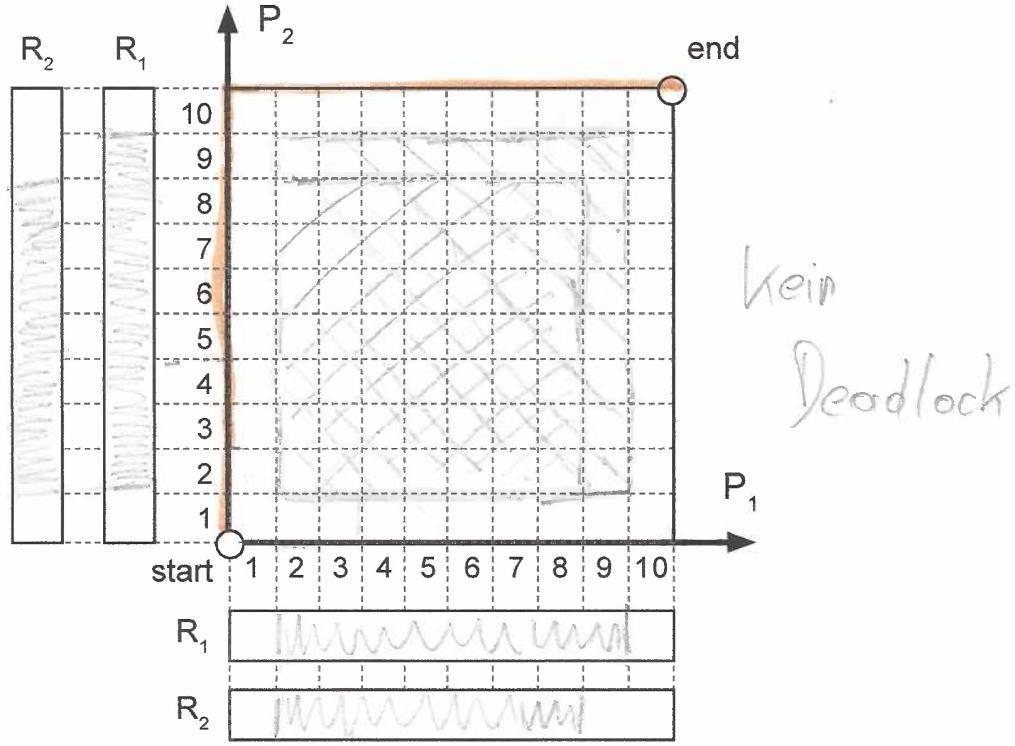
Program P_2:

```
 1: read(buf3);
 2: get(R2);
 3: copy(R2, buf3);
 4: clear(buf4);
 5: get(R1);
 6: copy(buf4, R1);
 7: use(R1, R2);
 8: free(R2);
 9: free(R1);
10: return;
```



Teilaufgabe B (Deadlock Prevention)

Nehmen Sie nun ein System an, das *Hold and Wait* unterbindet. Tragen Sie die modifizierten Lösungen für die Punkte 1 bis 4 aus Teilaufgabe A im folgenden Diagramm ein.



2 Page Replacement (25)

Gebeben ist ein Arbeitsspeicher mit vier Frames, dessen Seiten mit unterschiedlichen Ersetzungsstrategien (OPT, FIFO, und LRU) ersetzt werden sollen. Die Seitenzugriffsfolge ist für alle Algorithmen gleich. Sie ist jeweils in der Kopfzeile der Tabelle gegeben. Geben Sie in den Spalten der Tabellen die Speicherinhalte für jeden Frame nach dem in der Kopfzeile angegebenen Seitenzugriff an und kennzeichnen Sie in der letzten, mit *PF* markierten Zeile das Auftreten von Page Faults. Der Arbeitsspeicher ist am Beginn leer und wird zunächst mit Frame 0 beginnend befüllt.

OPT-Strategie:

	A	B	C	D	B	E	D	F	C	E	B	F	A	F	E	D	F	A
0	A	A	A	A	A	F	F	F	F	F	F	F	F	F	F	D	D	D
1		B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
2			C	C	C	C	C	C	C	C	C	C	A	A	A	A	A	A
3				D	D	D	D	F	F	F	F	F	F	F	F	F	F	F
PF	1	1	1	1	0	1	0	1	0	0	0	0	1	0	0	1	0	0

= 7

FIFO-Strategie:

	A	B	C	D	B	E	D	F	C	E	B	F	A	F	E	D	F	A
0	A	A	A	A	A	F	F	F	F	F	F	F	F	F	F	D	D	D
1		B	B	B	B	B	B	F	F	F	F	F	F	F	F	F	F	F
2			C	C	C	C	C	C	C	C	B	B	B	B	B	B	B	B
3				D	D	D	D	D	D	D	D	D	A	A	A	A	A	A
PF	1	1	1	1	0	1	0	1	0	0	1	0	1	0	0	1	0	0

= 9

LRU-Strategie:

	A	B	C	D	B	E	D	F	C	E	B	F	A	F	E	D	F	A
0	A	A	A	A	A	F	F	F	F	F	F	F	F	F	F	F	F	F
1		B	B	B	B	B	B	B	C	C	C	C	A	A	A	A	A	A
2			C	C	C	C	C	F	F	F	F	F	F	F	F	F	F	F
3				D	D	D	D	D	D	D	B	B	B	B	B	D	D	D
PF	1	1	1	1	0	1	0	1	1	0	1	0	1	0	0	1	0	0

= 10

Vergleichen Sie die Anzahl der bei den Seitenersetzungsstrategien beobachteten Page Faults. Entspricht das Ergebnis Ihren Erwartungen? Warum bzw. warum nicht?

Kein großer Unterschied, opt natürlich sehr gut, da optimal. LRU besser erwartet, dem geschuldet, dass FIFO gut zu dieser Abfolge passt => Zufall

2 Deadlock (20)

Gegeben sind zwei Prozesse, P_1 und P_2, die die beiden Ressourcen R_1 und R_2 unter Mutual Exclusion verwenden.

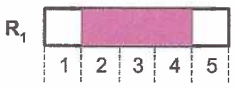
Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann im Prozessfortschrittsdiagramm als Kantenzug zwischen den Punkten *start* und *end* eingetragen werden. Die Achsenbeschriftung entspricht dabei der Zeilennummer des gerade auszuführenden Befehls.

Unterhalb bzw. links der Diagrammachsen sind Balken abgedruckt, in denen die Reservierugen von Ressourcen für P_1 bzw. P_2 eingetragen werden.

Teilaufgabe A

1. Tragen Sie die Reservierungen von Ressourcen für P_1 bzw. P_2 in die entsprechenden Balken ein. Dabei gilt eine Ressource ab Start der Anweisung `get()` als belegt und nach Beendigung der Anweisung `free()` als wieder freigegeben:

```
2: get(R1)
3: ...
4: free(R1)
```



2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung wegen Ressourcenkonflikten nicht möglich ist.

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug bei einer deadlockfreien Abarbeitung von P_1 und P_2 nicht passiert werden dürfen.

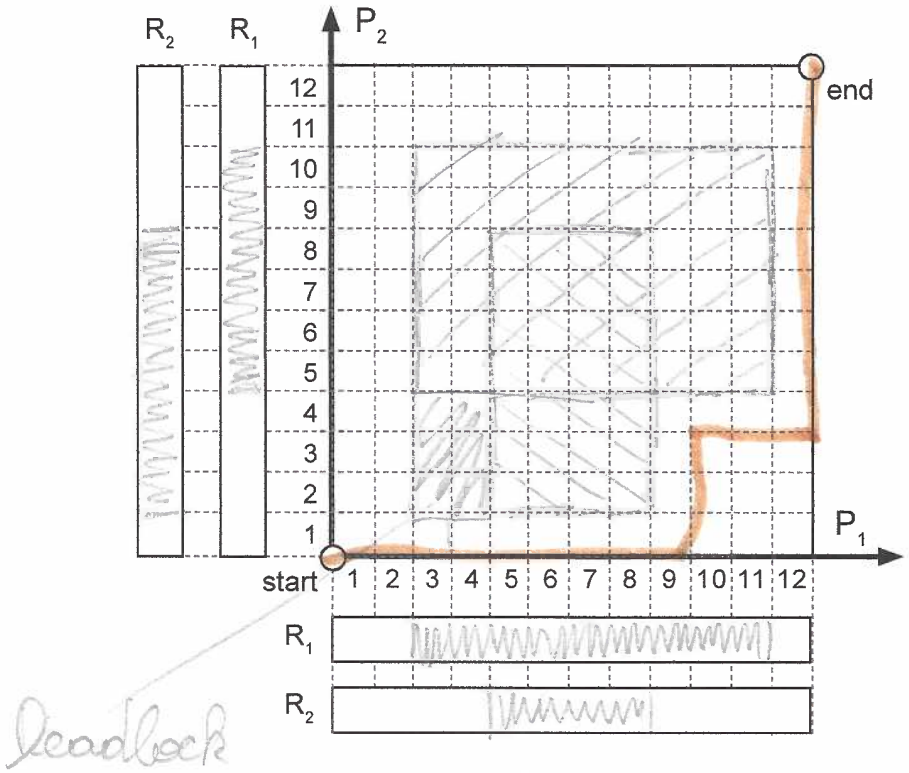
4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein.

Program P_1:

```
 1: read(buf);
 2: clear(buf2);
 3: get(R1);
 4: copy(R1, buf);
 5: get(R2);
 6: use(R1, R2);
 7: copy(buf2, R2);
 8: free(R2);
 9: print(buf2);
10: clear(buf);
11: free(R1);
12: return;
```

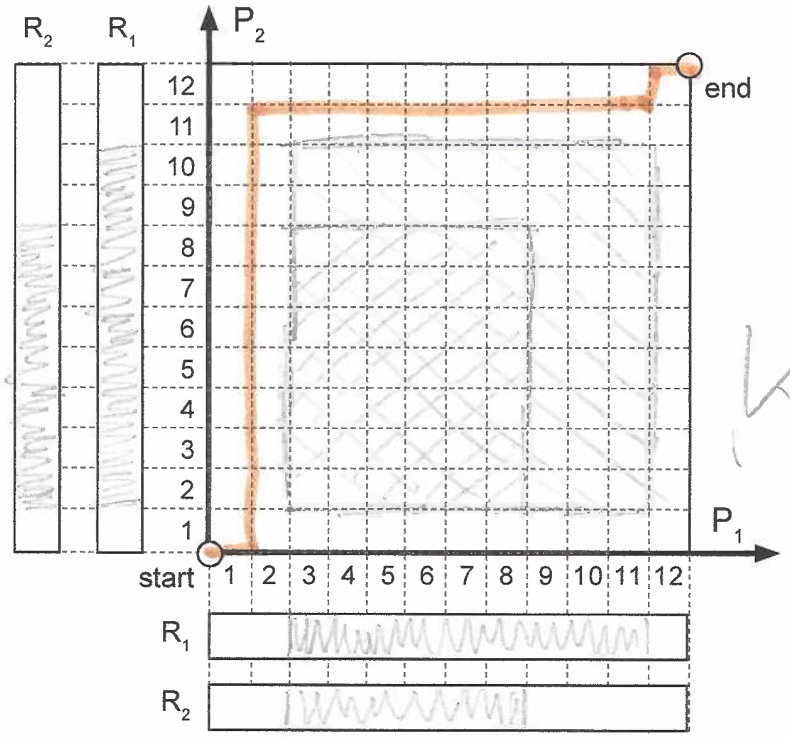
Program P_2:

```
 1: read(buf3);
 2: get(R2);
 3: copy(R2, buf3);
 4: clear(buf4);
 5: get(R1);
 6: copy(buf4, R1);
 7: use(R1, R2);
 8: free(R2);
 9: print(buf4);
10: free(R1);
11: clear(buf3);
12: return;
```



Teilaufgabe B (Deadlock Prevention)

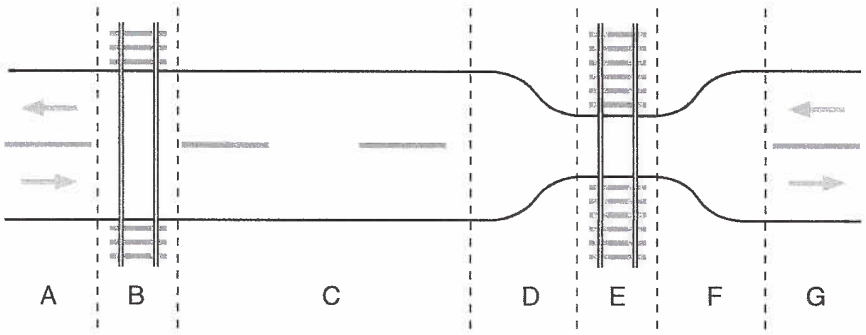
Nehmen Sie nun ein System an, das *Hold and Wait* unterbindet. Tragen Sie die modifizierten Lösungen für die Punkte 1 bis 4 aus Teilaufgabe A im folgenden Diagramm ein.



Kein deadlock

1 Synchronisation mit Semaphoren (30)

Ein System aus parallelen Prozessen soll den Verkehr auf einer Straße mit zwei Eisenbahnübergängen (siehe Skizze) simulieren. Dabei sollen die Fahrten von Autos und Zügen durch Prozesse realisiert werden, die durch Semaphoroperationen synchronisiert werden.



Ergänzen Sie die gegebenen Prozess-Codestücke mit *Semaphoroperationen*. Es gilt:

- Die Straße ist in die Abschnitte A bis G unterteilt. Autos werden durch die Prozesse *Auto_AG* (fährt von links nach rechts) bzw. *Auto_GA* (fährt von rechts nach links) simuliert. Von beiden Prozessen können beliebig viele Kopien gleichzeitig laufen.
- Die Prozesse *Auto_AG* bzw. *Auto_GA* rufen die Funktionen *A()*, *B()*, ..., *G()* auf, um die entsprechenden Straßenabschnitte zu durchfahren.
- Züge werden durch die Prozesse *Zug_B* (linkes Gleis) bzw. *Zug_E* (rechtes Gleis) simuliert. Züge verwenden die Funktionen *Anfahrt()* (Anfahrt zur Kreuzung mit der Straße), *Kreuzung()* (Durchfahren der Straßenkreuzung) und *Weiterfahrt()* (Weiterfahrt nach Passieren der Straßenkreuzung).
- Zweispurige Bereiche der Straße sollen möglichst parallel von Autos, die in unterschiedliche Richtungen fahren, genützt werden können. Auf dem Straßenabschnitt C dürfen sich zu jedem Zeitpunkt maximal k Fahrzeuge auf jeder Fahrspur befinden.
- Autos und Züge sind so zu synchronisieren, dass es an den Bahnübergängen zu keinen Kollisionen kommt. Züge sollen beim Passieren der Übergänge gegenüber den Autos bevorzugt behandelt werden. Wartende Autos dürfen die Gleise nicht verstellen.
- Die Engstelle mit dem Bahnübergang im rechten Straßenbereich kann zu jedem Zeitpunkt von maximal einem Auto passiert werden. Das korrekte Passieren der Engestelle ist durch Synchronisationskonstrukte sicherzustellen.

`Initialisierungen`

```
init(sem_b1, 1);
init(sem_b2, 1);
init(sem_c1, k);
init(sem_c2, k);
init(sem_e, 1);
init(sem_z1, 1);
init(sem_z2, 1);
init(sem_z11, 1);
```

```
/*** Code Auto_AG ***/

A();
P(sem_c1);
P(sem_b1);
P(sem_z1);
B();
V(sem_z1);
V(sem_b1);
C();
V(sem_c1)
D();
P(sem_e);
P(sem_z2);
E();
V(sem_z2);
V(sem_e);
F();
G();
```

```
/*** Code Auto_GA ***/

G();
F();
P(sem_c2);
P(sem_e);
P(sem_z2);
E();
V(sem_z2);
V(sem_e);
D();
C();
P(sem_b2);
P(sem_z11);
V(sem_c2);
B();
V(sem_z11);
V(sem_b2);
A();
```

```
/*** Code Zug_B ***/

Anfahrt();
P(sem_z1);
P(sem_z11);
Kreuzung();
V(sem_z1);
V(sem_z11);
Weiterfahrt();
```

```
/*** Code Zug_E ***/

Anfahrt();
P(sem_z2);
Kreuzung();
V(sem_z2);
Weiterfahrt();
```

1 Synchronisation mit Semaphoren (30)

Gegeben sind die Templates von drei Prozessen, A, B und C, die jeweils eine Folge von Funktionen ($a1()$, $a2()$, ..., $b1()$, $b2()$, ..., $c1()$, $c2()$, ...) aufrufen. Fügen Sie in die Codefragmente für die drei Prozesse *Semaphoreoperationen* ein, sodass die Abarbeitung der Prozesse die folgenden Anforderungen erfüllt.

- Die Funktionen $a1()$, $b1()$ und $c1()$ sind in genau dieser Reihenfolge auszuführen.
- Die Funktionen $a2()$, $b2()$ und $c2()$ sollen parallel ausführbar sein, d.h. die Synchronisationskonstrukte dürfen die Parallelität der entsprechenden Programmteile nicht einschränken.
- Für die Ausführung von $a3()$, $b3()$ und $c3()$ muss gelten, dass maximale Parallelität möglich ist, wobei sichergestellt werden muss, dass jede der drei Funktionen erst dann ausgeführt wird, wenn die Ausführung der drei Funktionen $a2()$, $b2()$ und $c2()$ bereits abgeschlossen ist.
- Die Ausführung von $a4()$ darf sich nicht mit der Ausführung von $b4()$ oder $c4()$ überlappen, während die Parallelausführung von $b4()$ und $c4()$ möglich sein muss.

Vervollständigen Sie die Initialisierungen und ergänzen Sie die fehlenden Semaphoroperationen in die folgenden Programmfragmenten. Gehen Sie davon aus, dass jeder der drei Prozesse nur ein Mal gestartet wird.

Initialisierungen

init(b1,0);
init(c1,0);
init(sem_a2b,0)
init(sem_a2c,0)
init(sem_b2a,0)
init(sem_c2a,0)
init(sem_b2c,0)
init(sem_c2b,0)
init(sem_b4,1)
init(sem_c4,1)

/** Code Prozess A **/	/** Code Prozess B **/	/** Code Prozess C **/
	P(sem_b1);	P(sem_c1);
a1();	b1();	c1();
V(sem_b1);	V(sem_c1);	
a2();	b2();	c2();
V(sem_a2b); V(sem_a2c);	V(sem_b2a); V(sem_b2c);	V(sem_c2a) V(sem_c2b);
P(sem_b2a) P(sem_c2a)	P(sem_a2b) P(sem_c2b)	P(sem_b2c) P(sem_a2c)
a3();	b3();	c3();
P(sem_b4); P(sem_c4);	P(sem_b4);	P(sem_c4);
a4();	b4();	c4();
V(sem_b4); V(sem_c4);	V(sem_b4);	V(sem_c4);

VO 182.711 17. Januar 2014

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

Ges.)(100) 1.)(35) 2.)(20) 3.)(45) Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation mit Semphoren (35)

In einem entlegenen Skigebiet gibt es zwei Berggipfel, von denen Skifahrer abfahren können. Zu jedem Gipfel gibt es eine Talstation, von wo aus die Skifahrer mit Pistentaxis zum eben diesem Gipfel gebracht werden. Von jedem der beiden Gipfel gibt es zwei unterschiedliche Pisten, die zu den beiden Talstationen führen, d.h., Skifahrer müssen sich am Gipfel entscheiden, zu welcher Talstation sie fahren. Um die Pistentaxis gleichmäßig auszulasten, gibt es auf den Gipfeln Anzeigetafeln, von denen Skifahrer eine Empfehlung ablesen können, zu welcher der beiden Talstationen sie abfahren sollen.

Der Betrieb im Skigebiet wird in einem Softwarepaket durch 3 Arten von Prozessen simuliert.

- *Skifahrer* simuliert einen Skifahrer, der beliebig oft mit einem Pistentaxi zu einem Gipfel fährt, dann einen Blick auf die Anzeigetafel (realisiert durch ein Shared Memory) wirft und anschließend zur entsprechenden Talstation abfährt. Es kann beliebig viele Prozesse dieses Typs geben.
- *Taxi(nr)* simuliert das Pistentaxi mit der Nummer nr ($nr \in \{1, 2\}$), das Skifahrer von der Talstation nr zum Gipfel nr bringt.
- *Update* wird immer wieder gestartet, um die Information der Anzeigetafeln zu aktualisieren (schreibt auf das Shared Memory).

Ergänzen Sie die angegebenen Prozesstemplates, um die Prozesse durch *Semaphore* zu synchronisieren. Beachten Sie folgende Punkte:

- Alle Anzeigetafeln werden durch ein einziges Shared Memory simuliert. Von diesem Shared Memory sollen die Skifahrer gleichzeitig lesen können.
- Aktualisierungen des Shared Memory durch den *Update* Prozess sollen möglichst wenig verzögert werden.
- In ein Taxi passen K Leute. Das Taxi fährt von der Talstation ab, wenn es voll ist und lässt am Gipfel alle Leute aussteigen, bevor es wieder zu Tal fährt. Der Code für diesen Prozess ist gegeben. Zur Synchronisation mit dem Taxi NR werden Semaphore mit Namen *<Sem_Name>_NR* verwendet.

Code für Prozesse und Initialisierungen

```
int rc=0; int wc=0;
void Skifahrer(void)

{

  int nr = 1;

  for(;;) {

    /* Warten auf Taxi */

    P(in_NR);
    einsteigen_Taxi(nr);
    V(entered_NR);


    P(out_NR);
    aussteigen_Taxi(nr);
    V(exited_NR);
    P(z);
    P(shmlockread)
    P(x);
    rc++;
    if(rc==1){
      P(shmlock);
    }
    V(x)
    V(shmlockread)
    V(z);

    nr = ShM_Anzeige;

    P(x)
     rc--;
     if(rc==0){
       V(shmlock);
     }
    V(x)
    ~~V(z)~~


    fahre_Piste(nr);

  }

}
```

```
void Update(void)

{

  P(y);
  wc++
  if(wc==1){ P(shmlockread);}
  V(Y)

  P(shmlock);

  /* Anzeige ShM aktualisieren */

  ShM_Anzeige = eval_sensors();

  V(shmlock);

  P(Y);
    wc--;
    if(wc==0){ V(shmlockread);}
  V(Y);


}
```

```
void Taxi(int NR)
{
  int cnt;
  for(;;) {
    /* Talstation NR */
    /* freie Plaetze signalisieren */
    for(cnt=0; cnt<K; cnt++) {
        V(in_NR);
    }
    /* Warten auf Einsteigen */
    for(cnt=0; cnt<K; cnt++) {
        P(entered_NR);
    }

    bergfahrt_NR();

    /* Bergstation NR */
    /* Freigabe zum Aussteigen */
    for(cnt=0; cnt<K; cnt++) {
        V(out_NR);
    }
    /* Warten Aussteigen */
    for(cnt=0; cnt<K; cnt++) {
        P(exited_NR);
    }

    talfahrt_NR();
  }
}
```

```
/** Semaphorinitialisierungen **/
init(in_NR, 0);
init(entered_NR, 0);
init(out_NR, 0);
init(exited_NR, 0);
init(x, 1);
init(shmlock, 1);
init(shmlockread, 1);
init(y, 1);
init(z, 1);
```