Vorlesungsprüfung aus Systemprogrammierung 2001-03-21

KNr. MNr. Zuname, Vorname

Ges.)(100)	1.)(25)	2.)(20)	3.)(30)	4.)(25)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Deadlock (25)

Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen A, B und C benötigen. Jede der drei Ressourcen ist nur einmal vorhanden und kann immer nur von einem Prozess belegt werden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Die Abbildung unten zeigt für jeden der beiden Prozesse, zu welchem Zeitpunkt ihrer Abarbeitung sie jeweils die einzelnen Ressourcen benötigen. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

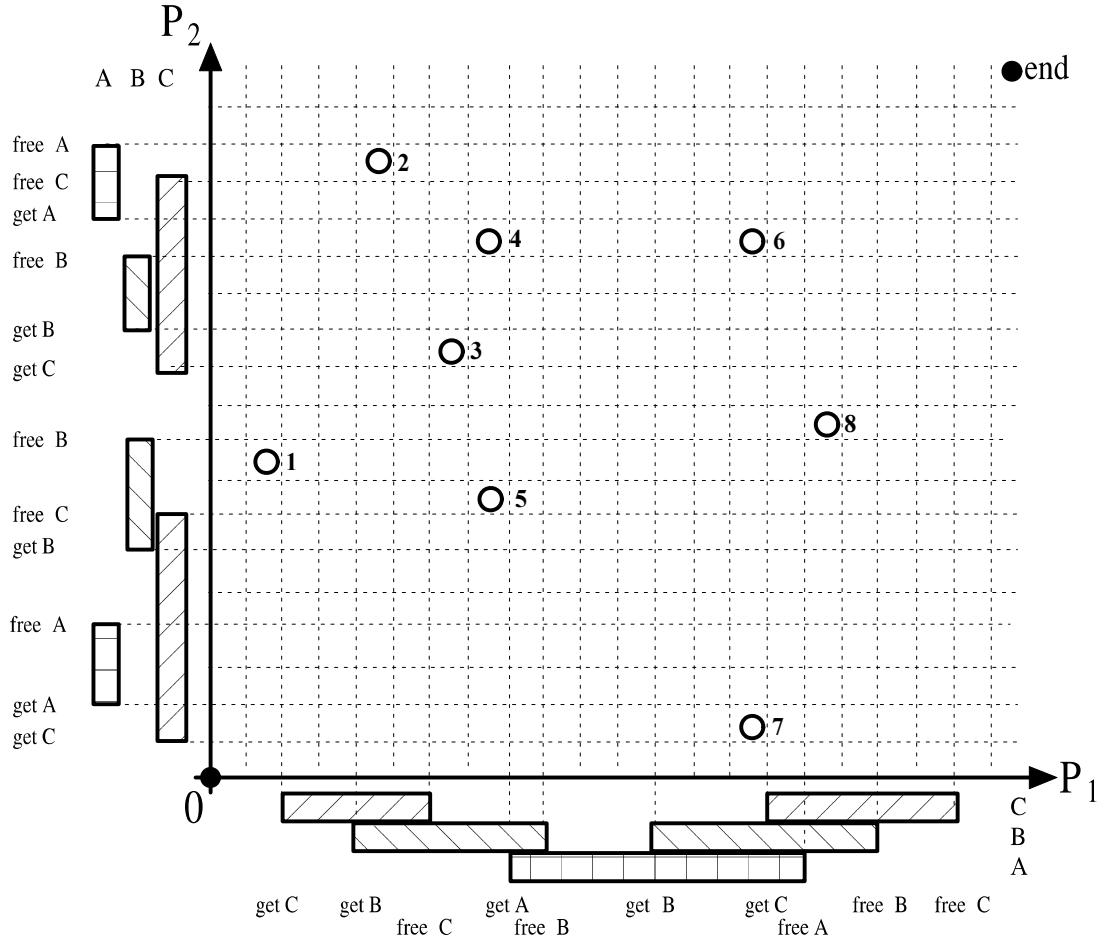
Der Fortschritt der Prozesse P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen dem Punkt *0* und dem Punkt *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

a) (8)

Umranden Sie in der Grafik jene Bereiche deutlich, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

b) (9)

Umranden und schraffieren Sie in der Grafik die Bereiche, die ein Kantenzug nicht passieren darf, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche zusätzlich deutlich mit einem "D".



c) (8)

In der Grafik sind acht Punkte durch einen Kreis gekennzeichnet und nummeriert. Tragen Sie in der folgenden Tabelle für jeden Punkt ein, ob eine durch diesen Punkt gehende Abarbeitung der beiden Prozesse ab dem Punkt (a) immer deadlockfrei erfolgt, (b) zu einem Deadlock führen kann aber nicht muss, (c) immer zu einem Deadlock führt, oder (d) gar nicht möglich ist.

	1	2	3	4	5	6	7	8
(a) immer deadlockfrei	○	○	○	○	○	○	○	○
(b) Deadlock möglich	○	○	○	○	○	○	○	○
(c) sicherer Deadlock	○	○	○	○	○	○	○	○
(d) Abarbeitung unmöglich	○	○	○	○	○	○	○	○

2 Deadlock (25)

Am Institut für technische Informatik werden über den Sommer Praktika von vier Studenten (**A**ndreas, **B**eatrice, **C**hristine und **D**aniela) absolviert. Zur Abwicklung ihre Praktika brauchen die Leute folgende Ressourcen:

	Andreas	**B**eatrice	**C**hristine	**D**aniela
Laserdrucker	1	1	1	0
Workstation	1	4	1	2
Knotenrechner	5	3	2	9

Abgesehen von diesen Ressourcen, die sich die Leute teilen müssen, hängen die einzelnen Arbeiten nicht voneinander ab, können also (bzw. sollen) parallel ausgeführt werden. Weiters werden Ressourcen von den Studenten erst dann wieder freigegeben, wenn ihr Praktikum beendet ist.

Insgesamt stehen am Institut 1 **L**aserdrucker, 4 **W**orkstations und 9 **K**notenrechner zur Verfügung.

Im Augenblick (Initialzustand) sieht die aktuelle Zuteilung der Resourcen zu den Praktikanten folgendermaßen aus:

- **A**ndreas hat bereits den **L**aserdrucker und 3 **K**notenrechner belegt.
- **B**eatrice arbeitet gerade auf 2 **W**orkstations und 1 **K**notenrechner.
- **C**hristine hat zur Zeit 2 **K**notenrechner in Verwendung.
- **D**aniela benützt 1 **W**orkstation und 1 **K**notenrechner.

a) Initialzustand (5)

Ermitteln Sie *Ressource-Vektor* (RV) und *Available-Vektor* (AV) sowie *Claim-Matrix* (CM) und *Allocation-Matrix* (AM) für den obig beschriebenen Initialzustand.

$$RV = \begin{pmatrix} L & W & K \\ & & \end{pmatrix} \qquad AV = \begin{pmatrix} L & W & K \\ & & \end{pmatrix}$$

$$CM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \qquad AM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix}$$

b) Resource Allocation Denial (20)

Benützen Sie zur Abwicklung der weiteren Ressourcenzuteilung die Strategie des *Resource Allocation Denial*. Wenden Sie ausgehend vom Initialzustand den *Banker's Algorithmus* solange an, bis entweder alle Praktika beendet wurden, oder ein Deadlock auftritt.

Notieren Sie hierbei, welcher Praktikant gerade sein Praktikum beenden konnte (also aktiv war) bzw. wenn ein Deadlock entstanden ist, kreuzen Sie an, welche Praktikanten daran beteiligt sind. Es müssen nur die tatsächlich verwendeten Felder ausgefüllt werden.

◯ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:
◯**A**ndreas ◯**B**eatrice ◯**C**hristine ◯**D**aniela

◯ _____ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} L & W & K \\ & & \end{pmatrix}$$

◯ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:
◯**A**ndreas ◯**B**eatrice ◯**C**hristine ◯**D**aniela

◯ _____ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} L & W & K \\ & & \end{pmatrix}$$

◯ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:
◯**A**ndreas ◯**B**eatrice ◯**C**hristine ◯**D**aniela

◯ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} L & W & K \\ & & \end{pmatrix}$$

◯ Es liegt ein Deadlock vor, an dem folgende Praktikanten beteiligt sind:
◯**A**ndreas ◯**B**eatrice ◯**C**hristine ◯**D**aniela

◯ konnte das Praktikum beenden. Der neue Zustand sieht wie folgt aus:

$$CM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AM = \begin{pmatrix} & L & W & K \\ A & & & \\ B & & & \\ C & & & \\ D & & & \end{pmatrix} \quad AV = \begin{pmatrix} L & W & K \\ & & \end{pmatrix}$$

c) Erklären Sie den Unterschied zwischen den Begriffen *Deadlock*, *Lifelock* und *Starvation* (5)

3 Deadlock (25)

Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Die Abbildung unten zeigt für jeden der beiden Prozesse, zu welchem Zeitpunkt seiner Abarbeitung er jeweils die einzelnen Ressourcen benötigt. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *0* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

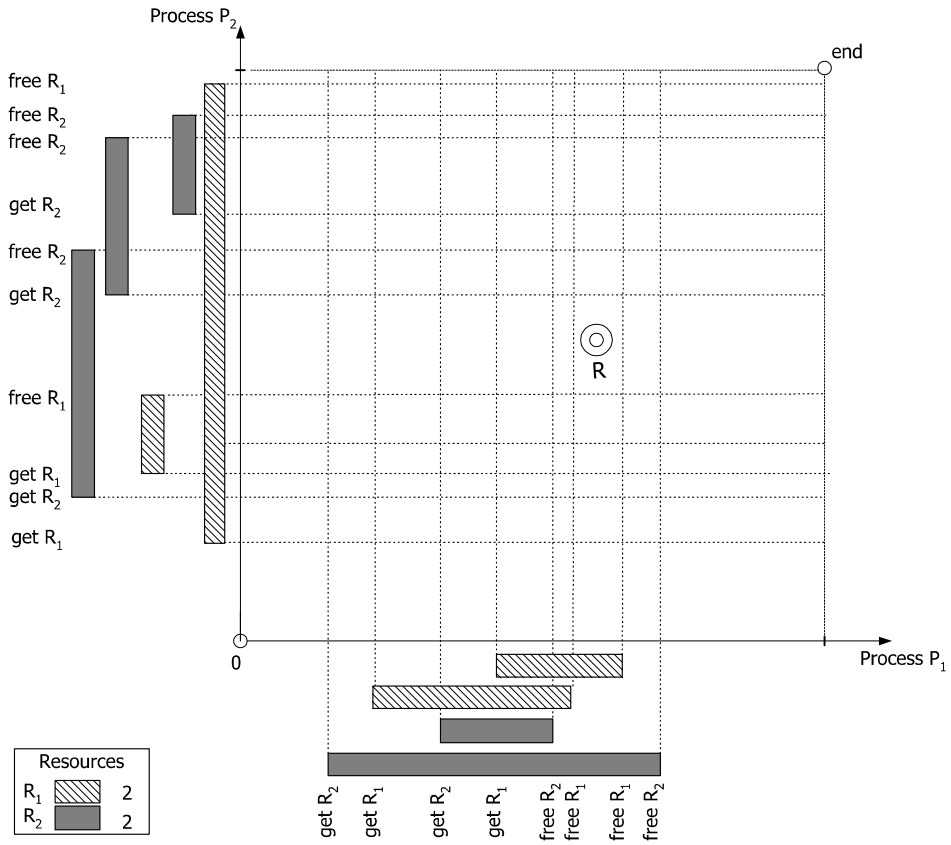
1. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

2. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

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

4. In der Grafik ist ein Punkt R mit einem Kreis markiert. Kann dieser Punkt bei der Abarbeitung der Prozesse erreicht werden? Begründen Sie Ihre Antwort!

Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 1 und 2 *deutlich unterscheidbar* sind.


Process P2
end
free R1
free R2
free R2
get R2
free R2
get R2
free R1
get R1
get R2
get R1
R
0
Process P1
Resources
R1 2
R2 2
get R2
get R1
get R2
get R1
free R2
free R1
free R1
free R2

3 Deadlock (25)

Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Die Abbildung unten zeigt für jeden der beiden Prozesse, zu welchem Zeitpunkt seiner Abarbeitung er jeweils die einzelnen Ressourcen benötigt. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *0* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

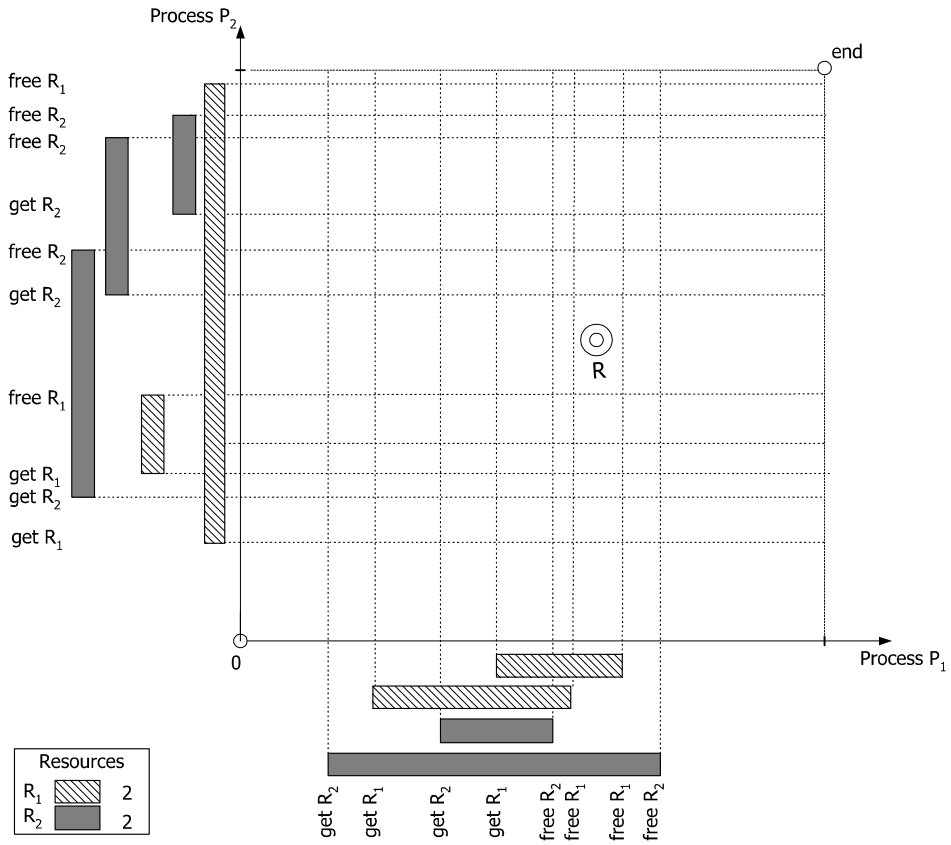
1. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

2. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

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

4. In der Grafik ist ein Punkt R mit einem Kreis markiert. Kann dieser Punkt bei der Abarbeitung der Prozesse erreicht werden? Begründen Sie Ihre Antwort!

Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 1 und 2 *deutlich unterscheidbar* sind.


Process P2
end
free R1
free R2
free R2
get R2
free R2
get R2
free R1
get R1
get R2
get R1
R
0
Process P1
Resources
R1 2
R2 2
get R2
get R1
get R2
get R1
free R2
free R1
free R1
free R2

4 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und die Ressourcen R_1, R_2 und R_3. Die Anzahl der vorhandenen Ressourcen im System können Sie aus dem Ressourcen-Vektor R ablesen. Aus der Claim-Matrix C ist die maximal notwendige Anzahl von Ressource pro Prozess ersichtlich. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Zu Beginn sind alle Ressourcen verfügbar.

$$\mathbf{R} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} \qquad \mathbf{C} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Das Diagramm in Abbildung 1 zeigt die Ressource-Anforderungen der beiden Prozesse. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *0* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

1. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

2. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

3. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein. Dieser Kantenzug soll durch möglichst viele Punkte (S_1-S_6) führen.

Entscheiden Sie für jeden der 6 Punkte (S_1-S_6) im Diagramm, ob die Punkte erreicht werden können, oder nicht und kreuzen Sie dementsprechend in der untenstehenden Tabelle an.

Punkt	erreichbar	nicht erreichbar
S_1	◯	◯
S_2	◯	◯
S_3	◯	◯
S_4	◯	◯
S_5	◯	◯
S_6	◯	◯

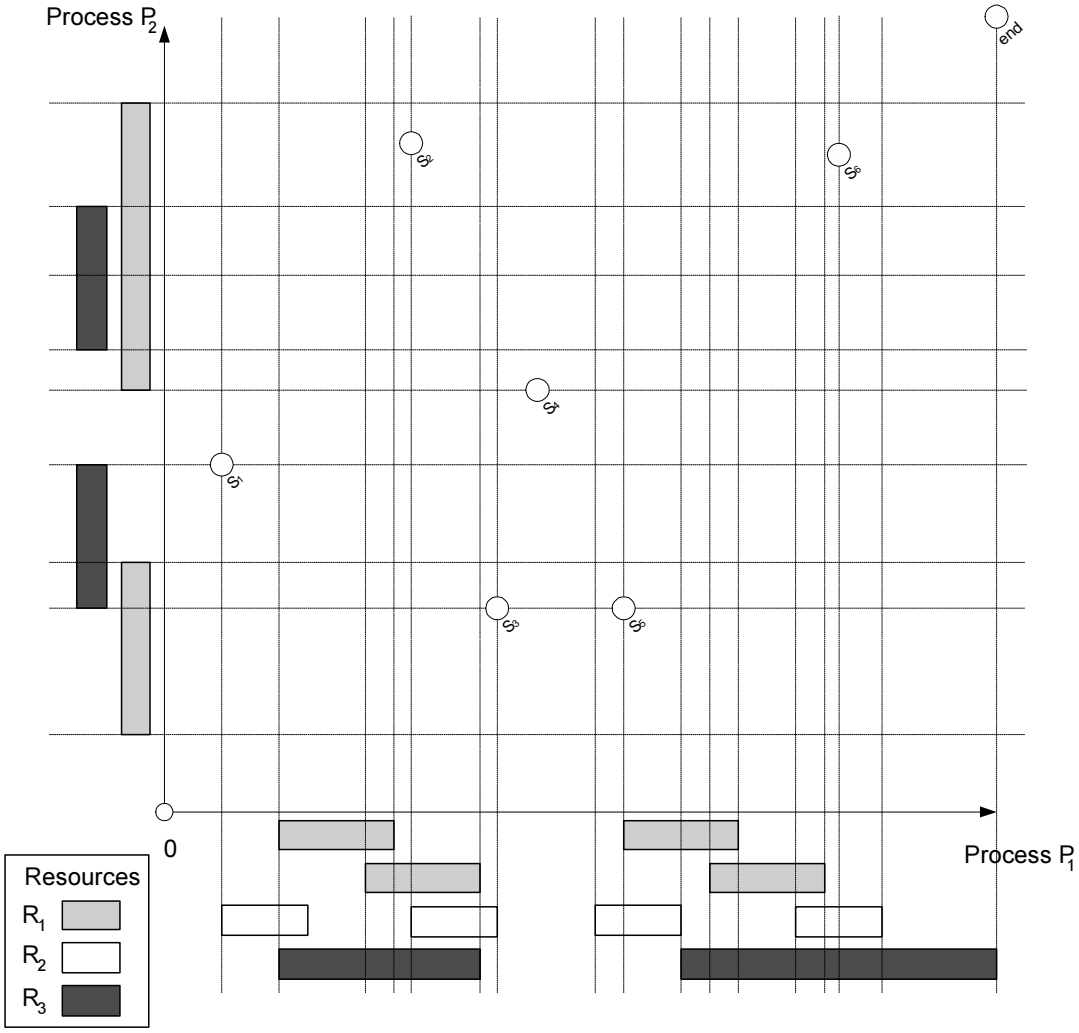


Abbildung 1: Abarbeitungs Diagramm

Vorlesungsprüfung aus ◯ Betriebssysteme ◯ Systemprogrammierung 2001-01-17

KNr. MNr. Zuname, Vorname

Ges.)(100) 1.)(25) 2.)(30) 3.)(25) 4.)(20) Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und die Ressourcen R_1, R_2 und R_3. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Zu Beginn sind alle Ressourcen verfügbar.

Das Diagramm in Abbildung 1 zeigt die Ressource-Anforderungen der beiden Prozesse. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *start* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

1. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

2. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

3. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein. Dieser Kantenzug soll durch möglichst viele Punkte (S_1-S_6) führen.

Entscheiden Sie für jeden der 6 Punkte (S_1-S_6) im Diagramm, ob der Punkt erreicht werden kann, oder nicht und kreuzen Sie dementsprechend in der untenstehenden Tabelle an.

Punkt	erreichbar	nicht erreichbar
S_1	◯	◯
S_2	◯	◯
S_3	◯	◯
S_4	◯	◯
S_5	◯	◯
S_6	◯	◯

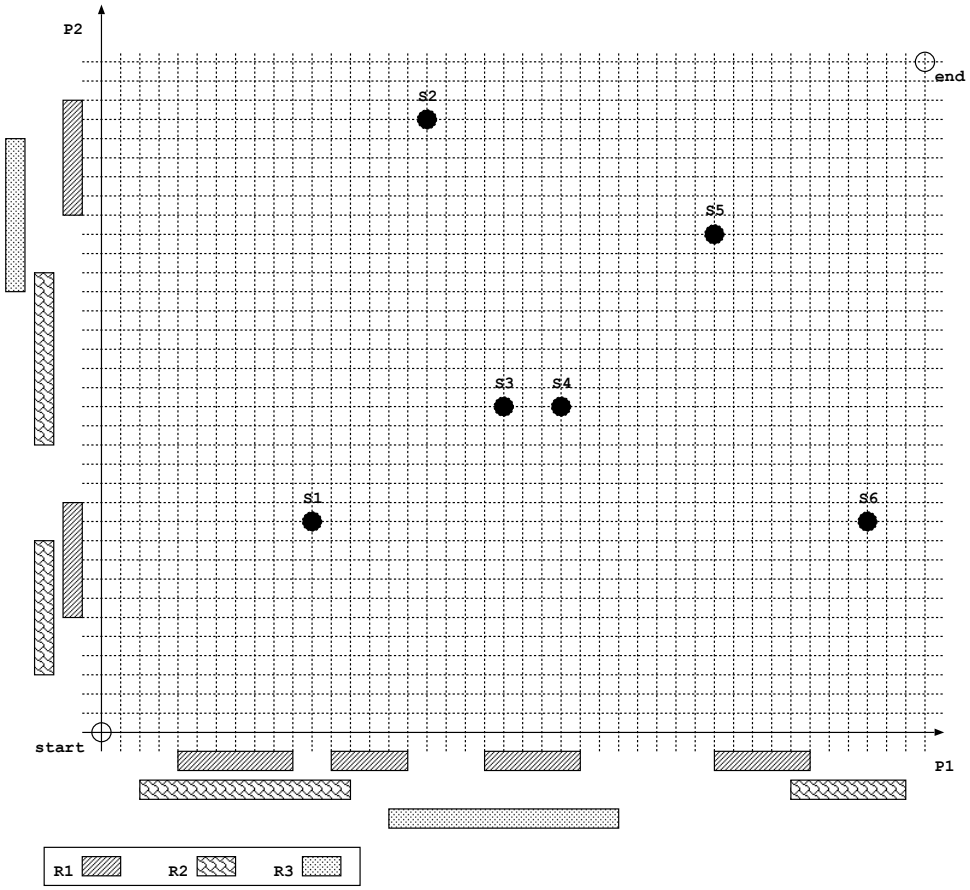


Abbildung 1: Abarbeitungs Diagramm

- Beschreiben Sie den Unterschied zwischen Process Initiation Denial und Resource Allocation Denial:

- Was ist Starvation:

Vorlesungsprüfung aus Betriebssysteme 2003-01-22

KNr. MNr. Zuname, Vorname

Ges.)(100) 1.)(25) 2.)(30) 3.)(23) 4.)(22) Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und zwei Ressourcen R_1, R_2. Die Ressourceallokationen der Prozesse in Abhängigkeit des jeweiligen Prozessfortschritts sind in Tabelle 1 eingetra-

Zeit	P1	P2
t=0		
t=1	P(R1)	P(R1)
t=2	P(R1)	P(R1)
t=3	P(R1)	
t=4		P(R2)
t=5		
t=6	V(R1)	
t=7	V(R1)	V(R1)
t=8		V(R1)
t=9	V(R1)	
t=10		
t=11	P(R2)	P(R1)
t=12	P(R2)	P(R1)
t=13		P(R1)
t=14	V(R2)	V(R1)
t=15	V(R2)	V(R1)
t=16		V(R1)
t=17		
t=18		V(R2)
t=19		
t=20	Termination	Termination

Tabelle 1: Prozess P1 und P2

gen. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Zu Beginn sind alle Ressourcen verfügbar. Von Ressource R_1 sind 3 Einheiten vorhanden, von R_2 sind 2 Einheiten vorhanden.

1.1 Abarbeitungs Diagramm

Abbildung 1 stellt ein Abarbeitungs Diagramm für die Prozesse P_1 und P_2 dar. Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *start* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

1. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.(3P. pro Flaeche, 1P Abzug pro falsches Kästchen)

2. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D". (4P. f. oben, 2P. f. unten, 1P Abzug pro falsches Kästchen, alles falsch - keine Punkte)

3. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein. (1P)

4. Stellt der Punkt P einen Deadlock dar? Begründen Sie Ihre Antwort! (2P, 1P f. Ja/Nein, 1P f. Begruendung)

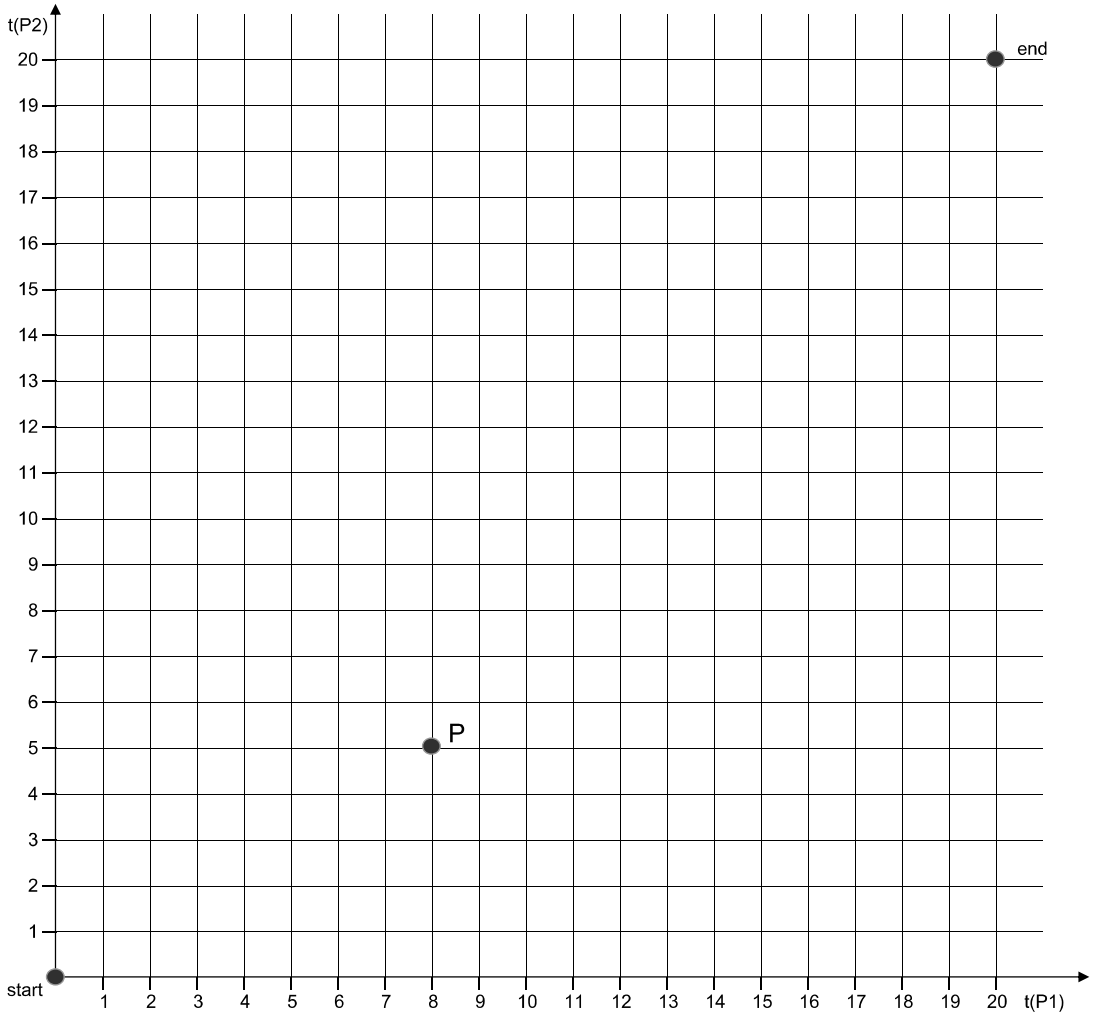


Abbildung 1: Abarbeitungs Diagramm

1.2 Deadlock Avoidance

Bestimmen Sie die Claim-Matrix C der Prozesse, sowie die Allokationsmatrix A im Punkt P. (Claim: 2 P, Alloc: 2P)

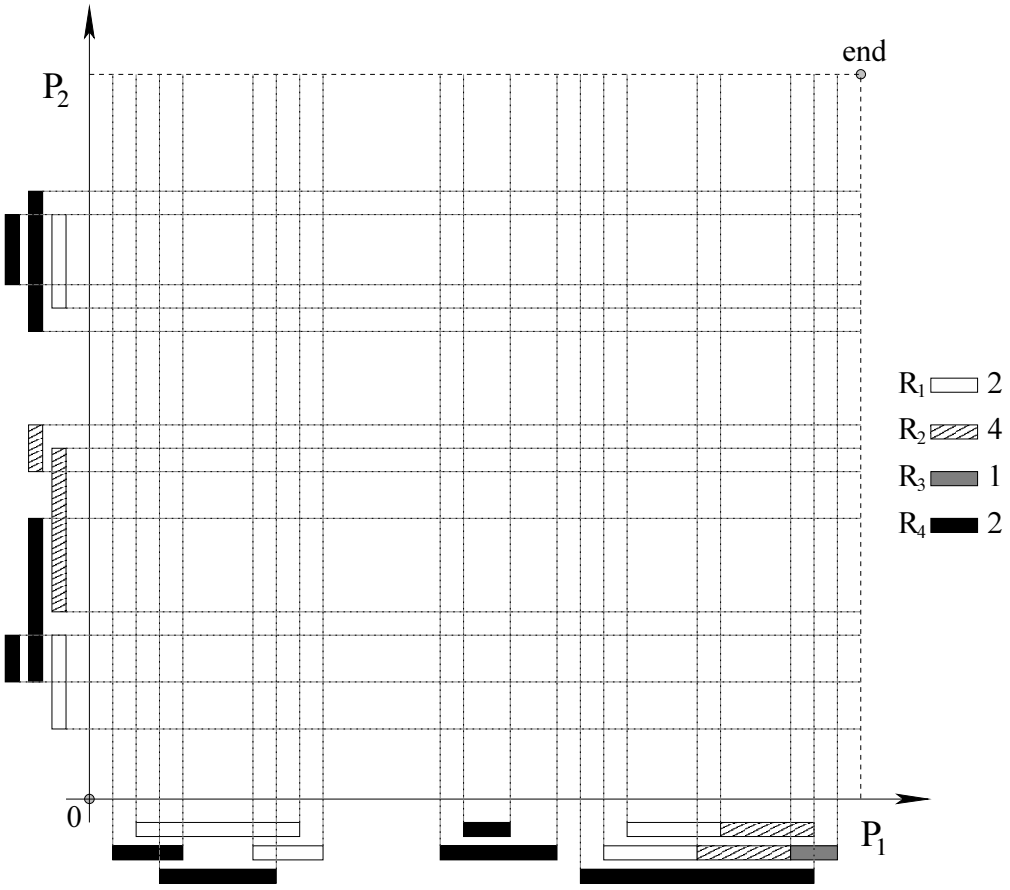
Ist P ein Safestate? Wenn ja, geben Sie mit dem Banker's Algorithm eine Ableitung an. Wenn nein, begründen Sie warum P kein Safestate ist. (3P: 1P Ja/Nein, 2P Begruendung)

3 Deadlock (20)

Zwei Prozesse P_1 und P_2 teilen sich gemeinsam Ressourcen des Systems. Der zum System gehörige Ressource Vector ist: $(R_1, R_2, R_3, R_4) := (2, 4, 1, 2)$. Werden von einem Prozess Ressourcen benötigt, welche der andere Prozess gerade belegt, so wird er mindestens bis zum Freiwerden der notwendigen Ressourcen verzögert. Welche Ressourcen von einem Prozess im Laufes seines „Lebens“ benötigt werden, entnehmen Sie bitte der Abbildung.

Durch einen Kantenzug von 0 nach end lässt sich nun der Fortschritt der beiden Prozesse bei der (quasi)parallelen Abarbeitung darstellen.

1. Umranden und schraffieren Sie in der Abbildung all jene Bereiche, durch welche ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen darf.
2. Umranden Sie all jene Bereiche, welche ein Kantenzug nicht passieren kann ohne in einem Deadlock zu enden. Kennzeichen Sie diese Bereiche durch ein „D“.
3. Umranden Sie all jene Bereiche, welche ein Kantenzug nicht erreichen kann obwohl kein Ressourcenkonflikt in diesem Bereich besteht. Kennzeichnen Sie diese Bereiche durch ein „U“.
4. Zeichen Sie einen deadlockfreien Kantenzug von 0 nach end ein.



4 Deadlock (25)

In einer Firma die sich mit Software und Web Design beschäftigt gibt es folgende Geräte: 4 Computer (C), 1 Scanner (S), 2 Drucker (D) und 3 Telefone (T).

In der Firma gibt es 4 Angestellte, einer ist zuständig für Web Design (WD), der zweite für Datenbanksoftware (DS) der dritte betreut die Kunden (KB), und der vierte ist System Administrator (*root*).

WD benötigt für seine Tätigkeit 3 Computer, 1 Scanner, 1 Drucker, und 2 Telefone. DS benötigt für seine Tätigkeit 1 Computer, 1 Drucker, und 1 Telefon. KB benötigt für seine Tätigkeit 1 Computer, 1 Scanner, 1 Drucker, und 2 Telefone. Root benötigt für seine Tätigkeit 1 Computer und 1 Telefon.

Zur Zeit belegt WD 2 Computer und 1 Telefon. DS belegt 1 Computer und 1 Telefon. KB belegt 1 Computer und 1 Drucker. Root belegt 1 Telefon.

a) (5)

Tragen sie den Resource-Vektor, die Claim Matrix und die Allocation-Matrix ein. Berechnen Sie auch den Availability-Vektor.

$$Resource = \begin{pmatrix} \texttt{C} & \\ \texttt{S} & \\ \texttt{D} & \\ \texttt{T} & \end{pmatrix} \qquad Claim = \begin{pmatrix} \text{WD} & \text{DS} & \text{KB} & \text{root} \\ & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

$$Available = \begin{pmatrix} \texttt{C} & \\ \texttt{S} & \\ \texttt{D} & \\ \texttt{T} & \end{pmatrix} \qquad Allocation = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

b) (12)

Suchen Sie jetzt eine Abarbeitungsreihenfolge für die Tätigkeiten der Angestellten, bei der alle Tätigkeiten ausgeführt werden können. Verwenden Sie dazu den *banker's algorithm* und geben Sie für *jeden* Schritt die Claim- und Allocationmatrix, sowie den Availability Vector und die als nächstes durchzuführende Tätigkeit (WD, DS, KB oder root) an. Wenn

keine Tätigkeit mehr auszuführen ist, dann schreiben sie ‘fertig’, falls ein Deadlock auftritt, schreiben Sie ‘Deadlock’ in den nächsten Schritt.

$$Claim = \begin{pmatrix} \texttt{C} & & & & \\ \texttt{S} & & & & \\ \texttt{D} & & & & \\ \texttt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{C} & & & & \\ \texttt{S} & & & & \\ \texttt{D} & & & & \\ \texttt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{C} & & & & \\ \texttt{S} & & & & \\ \texttt{D} & & & & \\ \texttt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{C} & & & & \\ \texttt{S} & & & & \\ \texttt{D} & & & & \\ \texttt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{C} \\ \texttt{S} \\ \texttt{D} \\ \texttt{T} \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{C} \\ \texttt{S} \\ \texttt{D} \\ \texttt{T} \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

c) (3)

Was ist Deadlock?

d) (5)

Welche 4 Bedienungen müssen erfüllt werden dass es zu einen Deadlock kommt?

3 Deadlock (25)

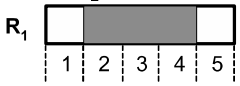
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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

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

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 unterhalb bzw. links der Diagrammachsen ein. Dabei ist anzunehmen, dass eine Roussource bereits ab Start der Anweisung `get()` als allokiert gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt, wie im folgenden Beispiel verdeutlicht ist:

   ```
   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 aufgrund von Ressourcenkonflikten nicht gehen kann.

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

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

5. Beschriften Sie jeweils ein Kästchen im Diagramm

   mit 'A', von welchem aus der Punkt *end* erreichbar ist

   mit 'B', welches unweigerlich zu einen Deadlock führt

   mit 'C', welches ein nicht erreichbarer Zustand ist

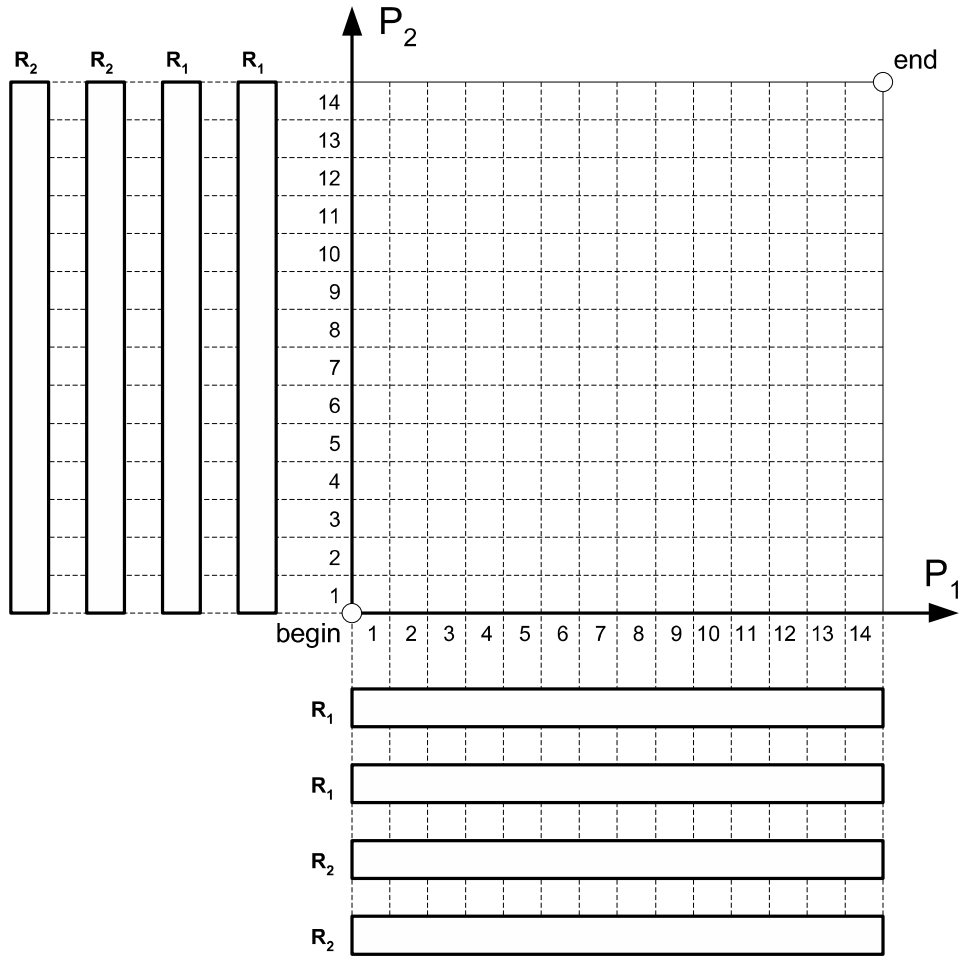
Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=5;
 2: get(R2);
 3: get(R2);
 4: b=a+6;
 5: a=a+4;
 6: get(R1);
 7: c=b+a;
 8: free(R2);
 9: a=0;
10: c=c/2;
11: b=0;
12: free(R1);
13: free(R2);
14: return;
```

Program P_2:

```
 1: a=0;
 2: get(R1);
 3: get(R1);
 4: b=a+2;
 5: get(R2);
 6: get(R2);
 7: free(R1);
 8: c=a+2;
 9: free(R1);
10: free(R2);
11: d=c;
12: free(R2);
13: a=d;
14: return;
```



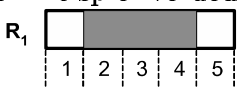
4 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und die Ressourcen R_1, R_2 und R_3. Ressource R_1 und R_2 sind einmal vorhanden. Ressource R_3 ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Zu Beginn sind alle Ressourcen verfügbar.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *start* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 unterhalb bzw. links der Diagrammachsen ein. Dabei ist anzunehmen, dass eine Roussource bereits ab Start der Anweisung `get()` als allokiert gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt, wie im folgenden Beispiel verdeutlicht ist:

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



2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

4. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein. Dieser Kantenzug soll durch möglichst viele Punkte (S_1-S_6) führen.

5. Entscheiden Sie für jeden der 6 Punkte (S_1-S_6) im Diagramm, ob der Punkt erreicht werden kann, oder nicht und kreuzen Sie dementsprechend in der untenstehenden Tabelle an.

Punkt	erreichbar	nicht erreichbar
S_1	○	○
S_2	○	○
S_3	○	○
S_4	○	○
S_5	○	○
S_6	○	○

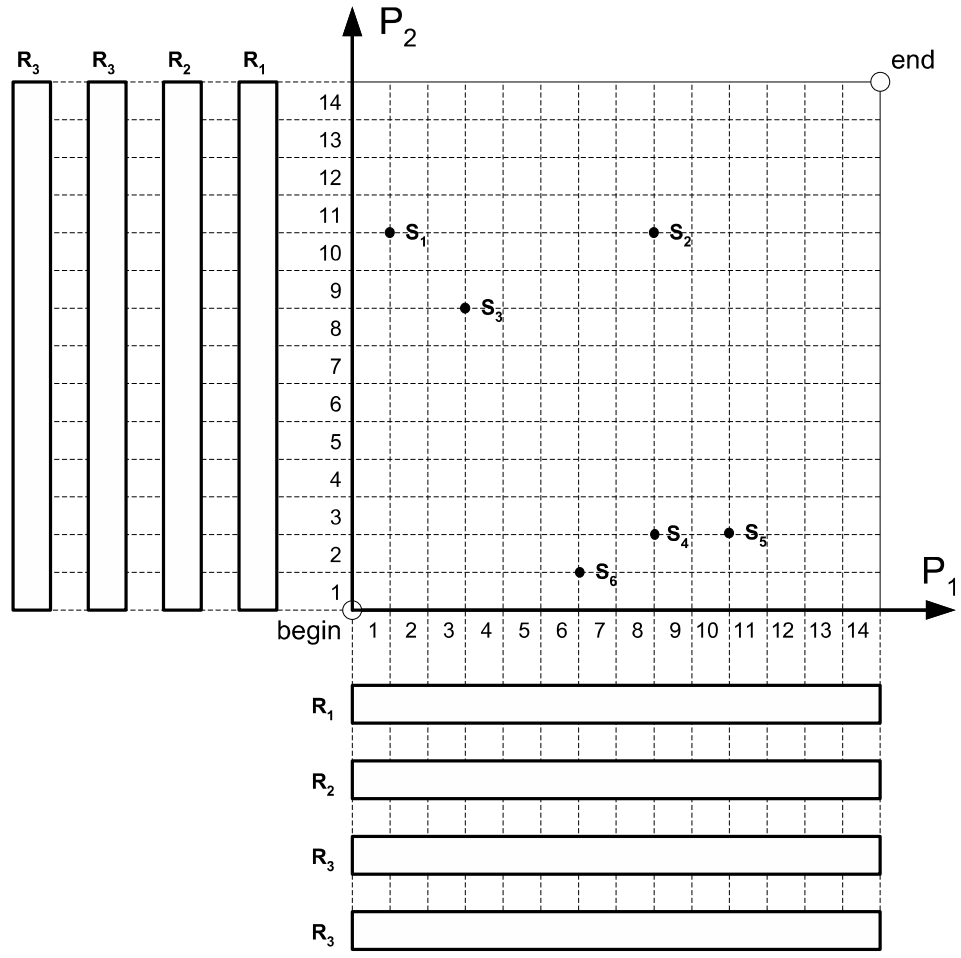
Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=0;
 2: get(R2);
 3: b=a+2;
 4: a=a+3;
 5: get(R3);
 6: get(R1);
 7: c=b*a;
 8: free(R1);
 9: get(R3);
10: b=a+b;
11: free(R2);
12: free(R3);
13: free(R3);
14: return;
```

Program P_2:

```
 1: get(R3);
 2: a=0;
 3: get(R1);
 4: get(R3);
 5: b=0;
 6: get(R2);
 7: free(R3);
 8: c=a+b;
 9: free(R2);
10: d=c;
11: free(R3);
12: free(R1);
13: a=d+2;
14: return;
```



3 Deadlock (20)

Ein Stahlhersteller produziert abhängig von der Nachfrage unterschiedliche legierte Stähle mit spezifischen chemischen Zusammensetzungen.

Um die laufenden Aufträge erfüllen zu können, müssen Ressourcen wie Container, Frachtschiffe und Güterzüge koordiniert werden.

Dem Stahlhersteller stehen 80 Container, 3 Frachtschiffe und 5 Züge zur Verfügung.

Zur Zeit sind drei Aufträge in Arbeit:

- Auftrag: # 1 beinhaltet den Transport von 1000 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 10 Container, 1 Frachtschiff und 2 Güterzüge notwendig. Für diesen Auftrag sind bereits 10 Container und 2 Güterzüge reserviert.
- Auftrag: # 2 beinhaltet den Transport von 4000 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 40 Container, 1 Frachtschiff und 4 Güterzüge notwendig. Die Bearbeitung dieses Auftrags hat noch nicht begonnen.
- Auftrag: # 3 beinhaltet den Transport von 1600 Tonnen Eisenerz. Um diesen Auftrag zu erfüllen, sind 16 Container, 2 Frachtschiffe und 2 Güterzüge notwendig. Für diesen Auftrag sind bereits 16 Container und ein Frachtschiff reserviert.

Process Initiation Denial (10)

Verwenden Sie die Strategie des *Process Initiation Denial*.

Beschreiben Sie *Resource-* und *Available-Vektor* sowie *Claim-* und *Allocation-Matrix*. Die Matrizen sind so auszufüllen, dass die Prozesse (= Aufträge) zeilenweise und die Ressourcen spaltenweise aufgezählt werden.

$$Resource = \begin{pmatrix} & & \end{pmatrix} \quad Available = \begin{pmatrix} & & \end{pmatrix}$$

$$Claim = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Allocation = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Darf Auftrag 2 gemäß *Process Initiation Denial* begonnen werden?

◯ Ja ◯ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

Resource Allocation Denial (10)

Verwenden Sie die Strategie des *Resource Allocation Denial*.
Beschreiben Sie für die gegebene Ausgangssituation mit Hilfe des Banker's-Algorithmus eine Lösung, bei der zuerst Auftrag 1, dann Auftrag 2 und zuletzt Auftrag 3 erfüllt werden.

Führen Sie alle Schritte bis zum Abschluss aller Aufträge durch und geben Sie zu jedem Schritt die zugehörige Claim- und Allocation-Matrix, sowie den Available-Vektor an.

Auftrag 1 wird durchgeführt:
(Alle für Auftrag 1 notwendigen Ressourcen sind in Verwendung):

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

Nachdem Auftrag 1 beendet ist und Auftrag 2 durchgeführt wird:
(Alle für Auftrag 2 notwendigen Ressourcen sind in Verwendung)

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

Nachdem Auftrag 2 beendet ist und Auftrag 3 durchgeführt wird:
(Alle für Auftrag 3 notwendigen Ressourcen sind in Verwendung)

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

Nachdem alle Aufträge durchgeführt worden sind:

$$C = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} & & \end{pmatrix}$$

3 Deadlock (25)

Bedingungen für Deadlock (6)

Erklären Sie die für das Auftreten eines Deadlocks notwendigen und hinreichenden Bedingungen.

Deadlock-Vermeidung (19)

Die Firma InTheRed hat unter ihren Angestellten 5 Ingenieure (I), 2 Tester (T) und 3 Patentanwälte (P) und bearbeitet derzeit drei Projekte, FIZZLE, T&E und HOTAIR.

Projektdaten:

Projektname:	FIZZLE
Benötigte Ressourcen:	4 Ingenieure, 2 Tester, 2 Patentanwälte
Bereits dem Projekt zugeteilt:	1 Ingenieur
Projektname:	T&E
Benötigte Ressourcen:	4 Ingenieure, 2 Tester, 0 Patentanwälte
Bereits dem Projekt zugeteilt:	1 Ingenieur, 1 Tester
Projektname:	HOTAIR
Benötigte Ressourcen:	2 Ingenieure, 1 Tester, 3 Patentanwälte
Bereits dem Projekt zugeteilt:	1 Ingenieur, 1 Tester, 1 Patentanwalt

Tragen Sie den Resource-Vektor, die Claim-Matrix und die Allocation-Matrix ein. Berechnen Sie auch den Availability-Vektor.

$$Resource = \begin{pmatrix} \texttt{I} & \\ \texttt{T} & \\ \texttt{P} & \end{pmatrix} \qquad Claim = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

$$Available = \begin{pmatrix} \texttt{I} & \\ \texttt{T} & \\ \texttt{P} & \end{pmatrix} \qquad Allocation = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Geben Sie in Ihrer Lösung die einzelnen Schritte der Ausführung des Deadlockerkennungsalgorithmus an.

Verwenden Sie nun den *Bankier-Algorithmus* (Banker's Algorithm) und geben Sie für *jeden* Schritt die Claim- und Allocationmatrix, sowie den Availability Vector und das als nächstes durchzuführende Projekt an. Wenn kein Projekt mehr auszuführen ist, dann schreiben sie 'fertig', falls ein Deadlock auftritt, schreiben Sie 'Deadlock' in den nächsten Schritt.

`Nächstes Projekt/fertig/Deadlock:`

`Alle für das Projekt benötigten Ressourcen sind in Verwendung:`

$$Claim = \begin{pmatrix} \texttt{I} & & & \\ \texttt{T} & & & \\ \texttt{P} & & & \end{pmatrix} \qquad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \qquad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

`Nach der Ausführung des letzten Projektes:`

$$Claim = \begin{pmatrix} \texttt{I} & & & \\ \texttt{T} & & & \\ \texttt{P} & & & \end{pmatrix} \qquad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \qquad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächstes Projekt/fertig/Deadlock:

Alle für das Projekt benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \text{I} \\ \text{T} \\ \text{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des letzten Projektes:

$$Claim = \begin{pmatrix} \text{I} \\ \text{T} \\ \text{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächstes Projekt/fertig/Deadlock:

Alle für das Projekt benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \text{I} \\ \text{T} \\ \text{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des letzten Projektes:

$$Claim = \begin{pmatrix} \text{I} \\ \text{T} \\ \text{P} \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

4 Deadlock (25)

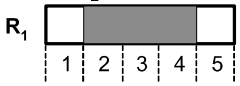
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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

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

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 unterhalb bzw. links der Diagrammachsen ein. Dabei ist anzunehmen, dass eine Roussource bereits ab Start der Anweisung `get()` als allokiert gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt, wie im folgenden Beispiel verdeutlicht ist:

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

   Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

2. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die der Kantenzug einer (quasi)parallelen Abarbeitung aufgrund von Ressourcenkonflikten nicht gehen kann.

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

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

5. Beschriften Sie jeweils ein Kästchen im Diagramm

   mit 'A', von welchem aus der Punkt *end* erreichbar ist

   mit 'B', welches unweigerlich zu einen Deadlock führt

   mit 'C', welches ein nicht erreichbarer Zustand ist

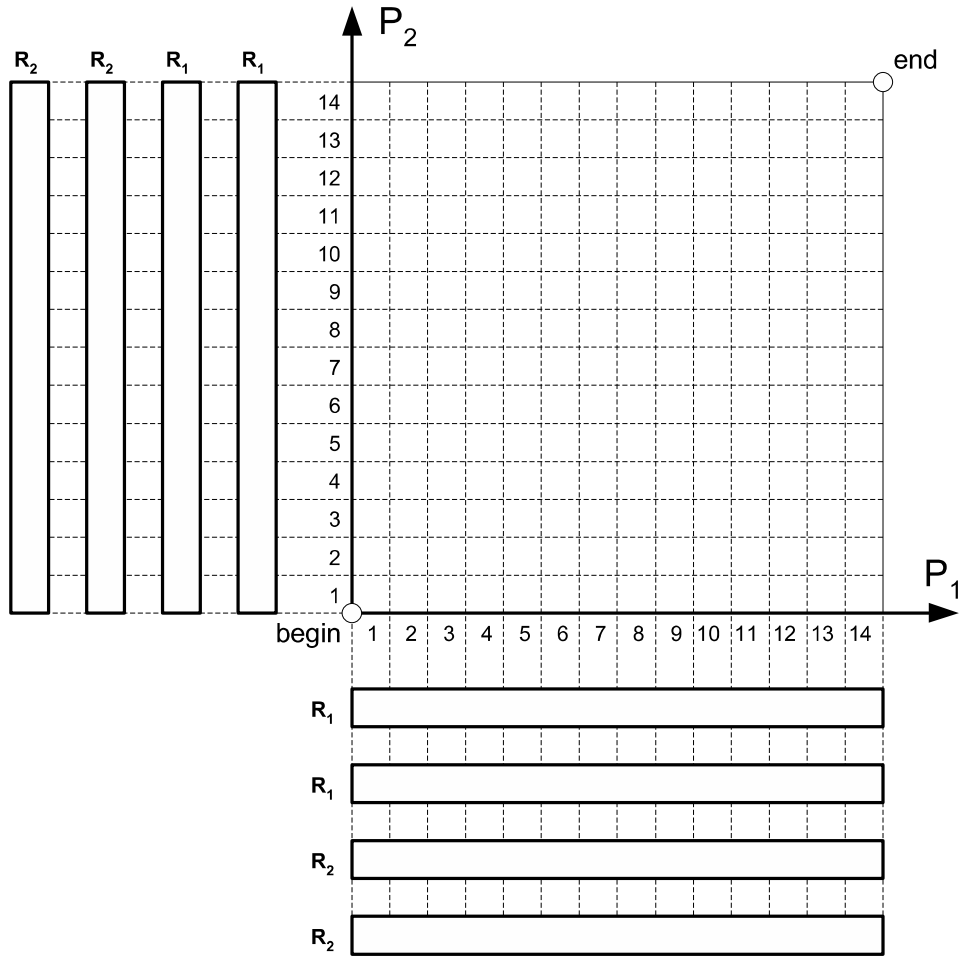
Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=0;
 2: get(R2);
 3: a=b-6;
 4: get(R2);
 5: a=a*2;
 6: a=b*3;
 7: get(R1);
 8: free(R2);
 9: a=5;
10: free(R1);
11: c=c-2;
12: b=10;
13: free(R2);
14: return;
```

Program P_2:

```
 1: get(R1);
 2: a=10;
 3: b=a+12;
 4: get(R1);
 5: get(R2);
 6: get(R2);
 7: free(R1);
 8: c=b*2;
 9: free(R1);
10: free(R2);
11: d=c-2;
12: a=b;
13: free(R2);
14: return;
```



3 Deadlock (25)

Ein Computer-Server besitzt 5 Festplatten, 2 Speichermodule mit je 10 GByte und 3 Gigabit Ethernet Netzwerkkarten. Die Ressourcen, die das installierte Betriebssystem benötigt, brauchen Sie zur Lösung der folgenden Aufgaben nicht berücksichtigen.

Um die auf dem Server befindlichen Programme schedulen zu können, müssen die Ressourcen wie Festplatten (F), Speicher (S) und Netzwerkkarten (N) koordiniert werden.

Zurzeit befinden sich drei Programme am Server:

- **Programm 1** benötigt 4 Festplatten, 20 GByte Speicher und 2 Netzwerkkarten. Für dieses Programm ist bereits 1 Festplatte reserviert.
- **Programm 2** benötigt 4 Festplatten, 20 GByte Speicher und keine Netzwerkkarte. Für dieses Programm sind bereits 1 Festplatte und 10 GByte Speicher reserviert.
- **Programm 3** benötigt 2 Festplatten, 10 GByte Speicher und 3 Netzwerkkarten. Für dieses Programm sind bereits 1 Festplatte, 10 GByte Speicher und 1 Netzwerkkarte reserviert.

Process Initiation Denial (10)

Beschreiben Sie *Resource-* und *Available-Vektor* sowie *Claim-* und *Allocation-Matrix*. Die Matrizen sind so auszufüllen, dass die Ressourcen zeilenweise und die Programme (Prozesse) spaltenweise aufgezählt werden.

$$Resource = \begin{pmatrix} \texttt{F} & \\ \texttt{S} & \\ \texttt{N} & \end{pmatrix} \qquad Available = \begin{pmatrix} \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{F} & & & \\ \texttt{S} & & & \\ \texttt{N} & & & \end{pmatrix} \qquad Allocation = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Darf mit Programm 2 gemäß *Process Initiation Denial* begonnen werden?

◯ Ja ◯ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

Bedingungen für Deadlock (5)

Erklären Sie die für das Auftreten eines Deadlocks notwendigen und hinreichenden Bedingungen.

Deadlock Vermeidung (10)

Verwenden Sie nun von obigem Initialzustand ausgehend den *Bankier-Algorithmus* (Banker's Algorithm) zum Schedulen der Prozesse. Geben Sie für *jeden* Schritt die Claim- und Allocation-Matrix, sowie den Availability Vector und das als nächstes durchzuführende Programm an. Wenn kein Programm mehr auszuführen ist, dann schreiben sie 'fertig', falls ein Deadlock auftritt, schreiben Sie 'Deadlock' in den nächsten Schritt.

Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \text{F} & & & \\ \text{S} & & & \\ \text{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} \text{F} & & & \\ \text{S} & & & \\ \text{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \text{F} & & & \\ \text{S} & & & \\ \text{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} \text{F} & & & \\ \text{S} & & & \\ \text{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{F} & & & \\ \texttt{S} & & & \\ \texttt{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} \texttt{F} & & & \\ \texttt{S} & & & \\ \texttt{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

Nächstes Programm: . Alle für das Programm benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{F} & & & \\ \texttt{S} & & & \\ \texttt{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Programms:

$$Claim = \begin{pmatrix} \texttt{F} & & & \\ \texttt{S} & & & \\ \texttt{N} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

3 Deadlock (23)

Das Catering Service *Fein und Gesund* bekommt von verschiedenen Kunden Aufträge um auf diversen Veranstaltungen für das leibliche Wohl zu sorgen.

Um die Aufträge erfolgreich erfüllen zu können müssen bestimmte Ressourcen wie Kellner (K), Teller (T) und Sektgläser (S) koordiniert werden. Dem Unternehmen stehen 15 Kellner, 200 Teller und 500 Sektgläser zur Verfügung.

Im Augenblick sind drei Aufträge in Arbeit:

- Für **Auftrag 1** werden 11 Kellner, 150 Teller und 300 Sektgläser benötigt. Reserviert sind für diesen Auftrag bereits 10 Kellner, 130 Teller und 200 Sektgläser.
- Für **Auftrag 2** werden 5 Kellner, 70 Teller und 300 Sektgläser benötigt. Dieser Auftrag hat noch nicht begonnen.
- Für **Auftrag 3** werden 7 Kellner, 10 Teller und 80 Sektgläser benötigt. Reserviert sind für diesen Auftrag bereits 1 Kellner, 1 Teller und 1 Sektglas.

Process Initiation Denial

Beschreiben Sie *Resource-* und *Available-Vektor* sowie *Claim-* und *Allocation-Matrix*. Die Matrizen sind so auszufüllen, dass die Ressourcen zeilenweise und die Aufträge spaltenweise aufgezählt werden.

$$Resource = \begin{pmatrix} \mathtt{K} & \\ \mathtt{T} & \\ \mathtt{S} & \end{pmatrix} \qquad Available = \begin{pmatrix} & \\ & \\ & \end{pmatrix}$$

$$Claim = \begin{pmatrix} \mathtt{K} & & & \\ \mathtt{T} & & & \\ \mathtt{S} & & & \end{pmatrix} \qquad Allocation = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Darf mit Auftrag 2 gemäß *Process Initiation Denial* begonnen werden?

◯ Ja ◯ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

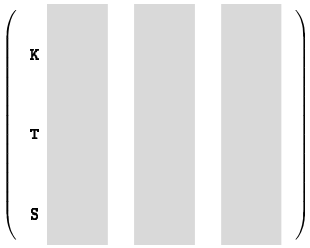
Deadlock Vermeidung

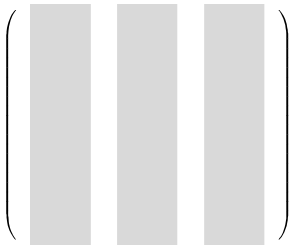
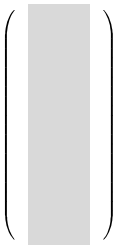
Verwenden Sie nun von obigem Initialzustand ausgehend den *Bankier-Algorithmus* (Banker's Algorithm) zum Schedulen der Aufträge. Geben Sie für *jeden* Schritt die Claim- und Allocation-Matrix, sowie den Available Vector und den als nächstes durchzuführenden Auftrag an. Wenn kein Auftrag mehr auszuführen ist, dann schreiben sie 'fertig' in den nächsten Schritt, falls ein Deadlock auftritt, schreiben Sie 'Deadlock' .

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

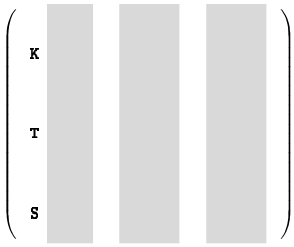
$$Claim = \begin{pmatrix} K & & & \\ T & & & \\ S & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

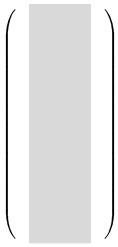
Nach der Ausführung des zuletzt markierten Auftrags:

$Claim =$ 

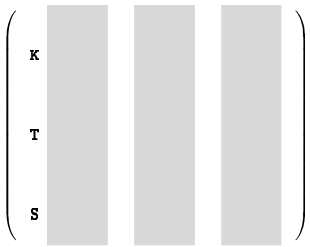
$Alloc =$ 
$Avail =$ 

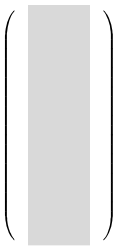
Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

$Claim =$ 

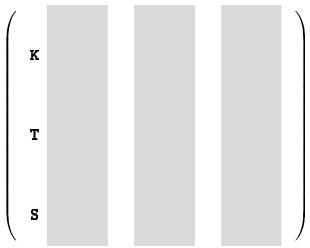
$Alloc =$ 
$Avail =$ 

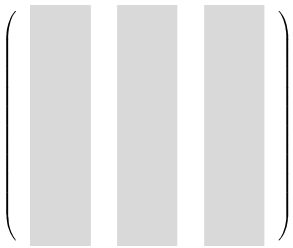
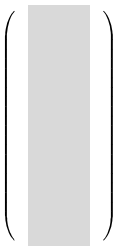
Nach der Ausführung des zuletzt markierten Auftrags:

$Claim =$ 

$Alloc =$ 
$Avail =$ 

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

$Claim =$ 

$Alloc =$ 
$Avail =$ 

Nach der Ausführung des zuletzt markierten Auftrags:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nächster Auftrag: . Alle für den Auftrag benötigten Ressourcen sind in Verwendung:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

Nach der Ausführung des zuletzt markierten Auftrags:

$$Claim = \begin{pmatrix} \texttt{K} & & & \\ \texttt{T} & & & \\ \texttt{S} & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \end{pmatrix}$$

3 Deadlock (18)

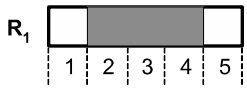
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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

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

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

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

   Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

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

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll.

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

5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

   ... mit 'A', von welchem aus der Punkt *end* bzw. welcher vom Punkt *begin* erreichbar ist

   ... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist

   ... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern eine solcher Punkt vorhanden ist

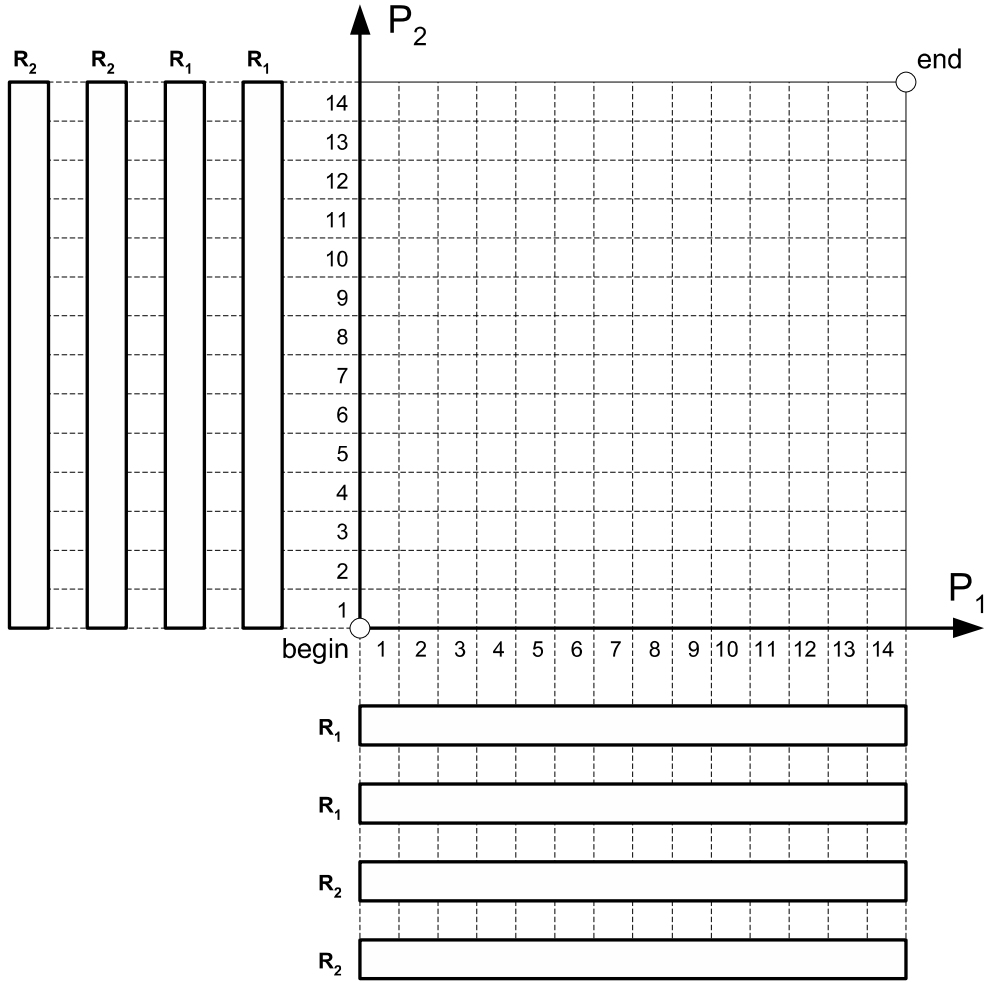
Anmerkung: Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=3;
 2: get(R1);
 3: b=4;
 4: get(R1);
 5: get(R2);
 6: get(R2);
 7: c=a+b;
 8: free(R2);
 9: a=b*4;
10: free(R2);
11: free(R1);
12: b=9;
13: free(R1);
14: return;
```

Program P_2:

```
 1: a=12;
 2: get(R2);
 3: get(R1);
 4: b=3*a;
 5: get(R2);
 6: c=a+b;
 7: a=b*c;
 8: free(R2);
 9: free(R1);
10: d=a+b;
11: a=b*5;
12: free(R2);
13: a=b+5;
14: return;
```



4 Deadlock (24)

Bedingungen für Deadlock (8)

Erklären Sie die für das Auftreten eines Deadlocks notwendigen und hinreichenden Bedingungen.

Behandlung von Deadlocks (8)

Erklären Sie den Begriff der Deadlock Prevention. Führen Sie zu jedem der beiden Arten von Deadlock Prevention ein Beispiel für eine Strategie zur Behandlung von Deadlocks an.

Worin liegt der Unterschied zwischen Deadlock Prevention und Deadlock Avoidance?

Process Initiation Denial (8)

Gegeben ist ein System von *3* Prozessen (P1, P2 und P3) und deren Geamtanforderung an Ressourcen, dargestellt durch die *Claim-Matrix C*. Die Prozesse verwenden *3* Ressourcenkategorien (R1, R2 und R3), deren insgesamt zur Verfügung stehende Gesamtanzahl durch den Vektor R abgebildet wird.

Im aktuellen Zustand des Systems werden die Prozesse P1 und P3 bereits ausgeführt. Bescreiben Sie anhand dieses Systemzustandes die gegenwärtige Ressourcenbelegung durch Ausfüllen der *Allocation-Matrix A* und des *Available-Vektors V* (in den Matrizen repräsentiert jede Spalte eine Ressource und jede Zeile einen Prozess).

$$R = \begin{pmatrix} 80 & 5 & 6 \end{pmatrix} \quad V = \begin{pmatrix} & & \end{pmatrix}$$

$$C = \begin{pmatrix} 10 & 1 & 2 \\ 40 & 1 & 4 \\ 16 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Darf mit Programm 2 gemäß *Process Initiation Denial* begonnen werden?

◯ Ja ◯ Nein

Begründen Sie Ihre Antwort (Antworten ohne Begründung werden nicht gewertet!):

2 Deadlock (18)

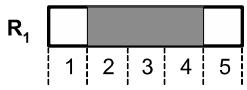
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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

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

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

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

   Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

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

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll.

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

5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

   ... mit 'A', von welchem aus der Punkt *end* bzw. welcher vom Punkt *begin* erreichbar ist

   ... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist

   ... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern eine solcher Punkt vorhanden ist

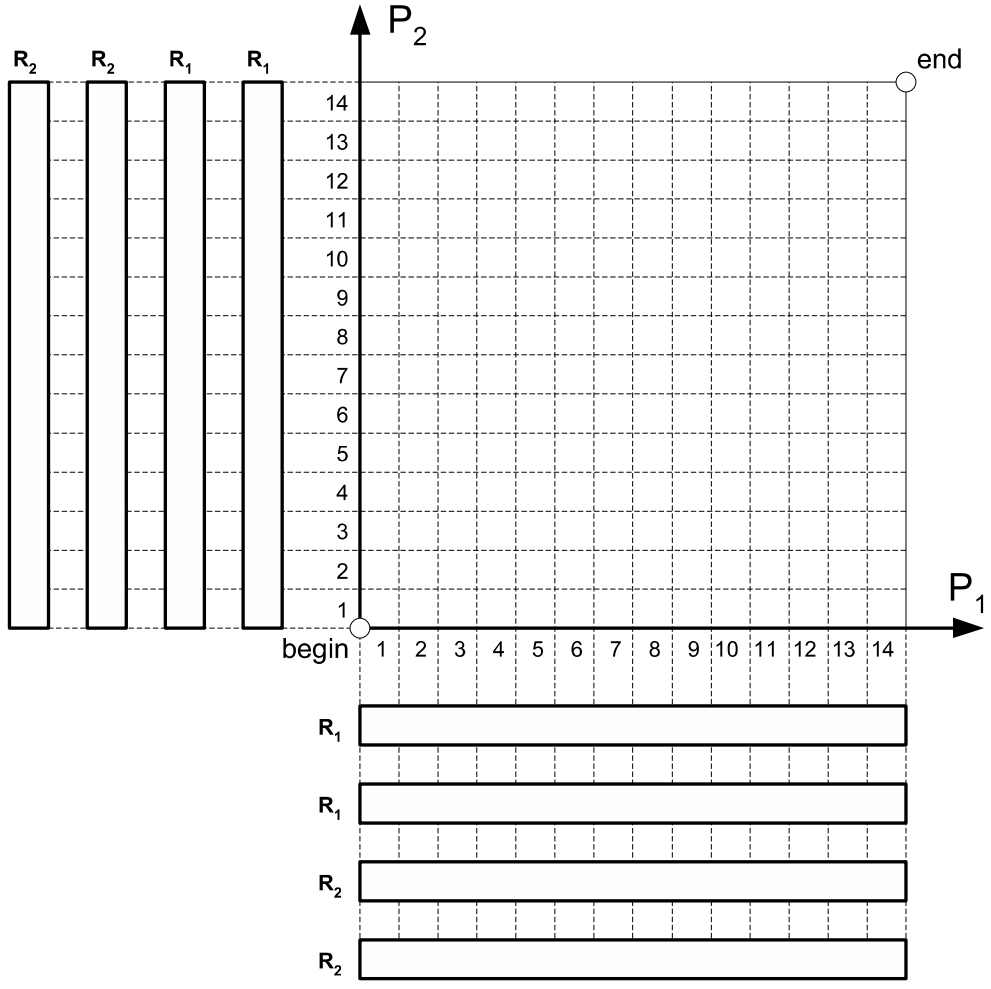
Anmerkung: Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=3;
 2: get(R1);
 3: b=4;
 4: get(R1);
 5: get(R2);
 6: get(R2);
 7: c=a+b;
 8: free(R2);
 9: a=b*4;
10: free(R2);
11: free(R1);
12: b=9;
13: free(R1);
14: return;
```

Program P_2:

```
 1: a=12;
 2: get(R2);
 3: get(R1);
 4: b=3*a;
 5: get(R2);
 6: c=a+b;
 7: a=b*c;
 8: free(R2);
 9: free(R1);
10: d=a+b;
11: a=b*5;
12: free(R2);
13: a=b+5;
14: return;
```



3 Deadlock (18)

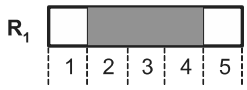
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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

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

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

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

   Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

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

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll.

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

5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

   ... mit 'A', von welchem aus der Punkt *end* bzw. welcher vom Punkt *begin* erreichbar ist

   ... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist

   ... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern eine solcher Punkt vorhanden ist

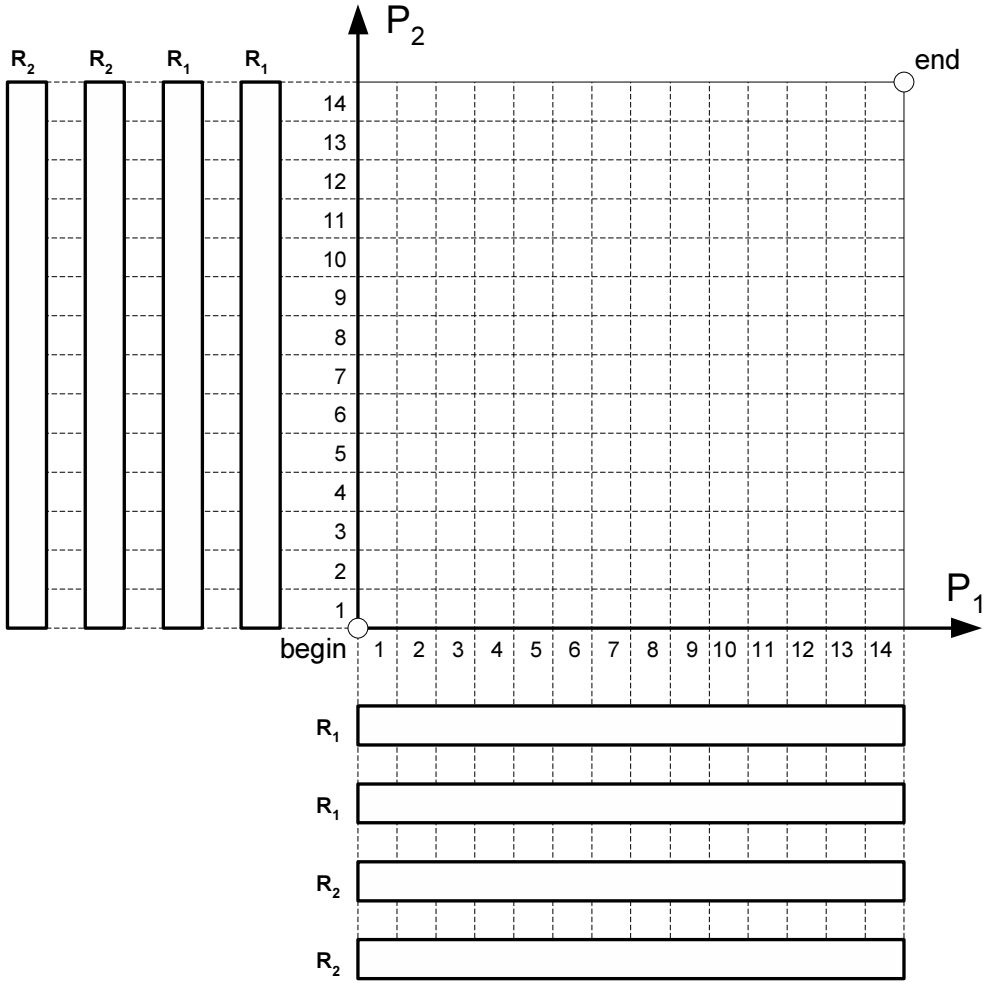
Anmerkung: Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=1;
 2: b=3;
 3: get(R1);
 4: get(R1);
 5: get(R2);
 6: get(R2);
 7: c=a+b;
 8: free(R2);
 9: a=b*4;
10: free(R1);
11: b=9;
12: free(R2);
13: free(R1);
14: return;
```

Program P_2:

```
 1: a=12;
 2: get(R2);
 3: b=3*a;
 4: c=a+b;
 5: get(R2);
 6: get(R1);
 7: a=b*c;
 8: free(R1);
 9: free(R2);
10: d=a+b;
11: free(R2);
12: a=b*5;
13: a=b+5;
14: return;
```



3 Deadlock (25)

Deadlock-Bedingungen (6)

Erklären Sie die Deadlock-Bedingungen *Hold and Wait* und *No Preemption*.

`Hold and Wait:`

[redacted]

`No Preemption:`

[redacted]

Banker's Algorithmus (6)

Wozu wird der Banker's Algorithmus verwendet?

[redacted]

Was bedeutet es, wenn der Banker's Algorithmus einen *Safe State* bzw. einen *Unsafe State* diagnostiziert?

`Safe State:`

[redacted]

`Unsafe State:`

[redacted]

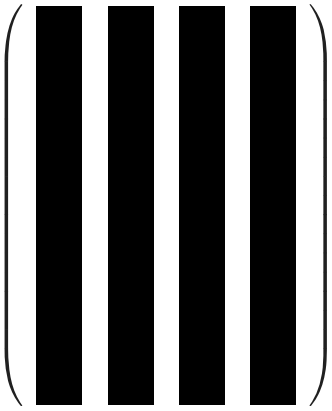
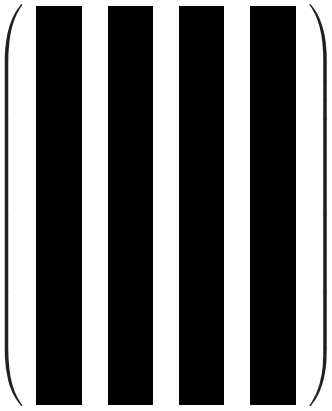
Deadlock-Erkennung (13)

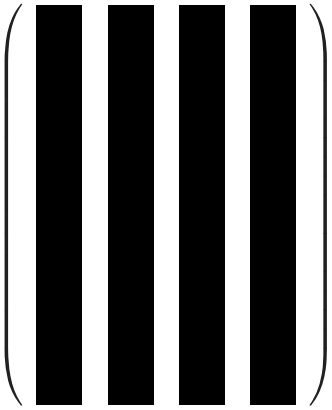
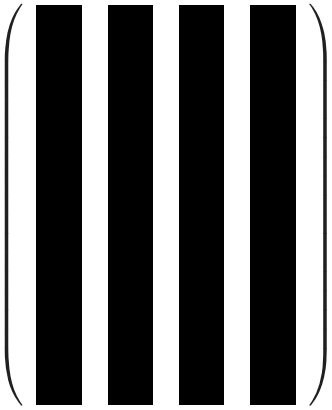
In einem Computersystem laufen fünf Prozesse, die gemeinsame Ressourcen aus vier Ressourcenkategorien verwenden. Die vorhandenen Ressourcen sind durch den Vektor $R = (6, 5, 9, 6)$ gegeben. Die unten gegebenen Matrizen beschreiben die aktuellen Ressourcenanforderungen und -allokation der fünf Prozesse.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

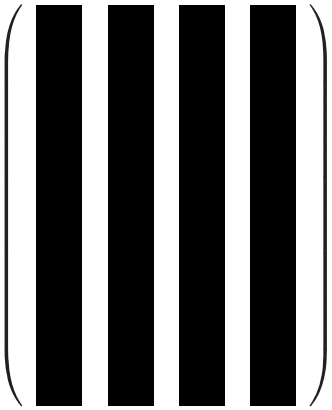
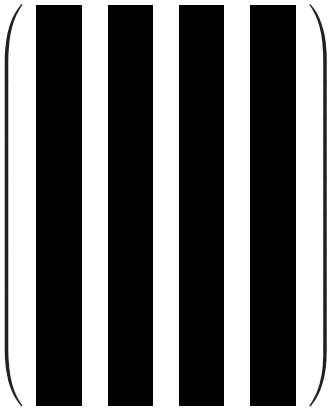
$$Q = \begin{pmatrix} 2 & 0 & 6 & 4 \\ 0 & 1 & 3 & 0 \\ 3 & 0 & 1 & 1 \\ 1 & 4 & 3 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 4 & 2 \\ 3 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad V = \begin{pmatrix} \blacksquare, & \blacksquare, & \blacksquare,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

$$Q = \begin{pmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{pmatrix} \quad A = \begin{pmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{pmatrix} \quad V = \begin{pmatrix} \blacksquare, & \blacksquare, & \blacksquare, & \blacksquare \end{pmatrix}$$

$$Q = \begin{pmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{pmatrix} \quad A = \begin{pmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{pmatrix} \quad V = \begin{pmatrix} \blacksquare, & \blacksquare, & \blacksquare, & \blacksquare \end{pmatrix}$$

$Q =$  $A =$  $V =$ 

$Q =$  $A =$  $V =$ 

$Q =$  $A =$  $V =$ 

Liegt ein Deadlock vor? Falls ein Deadlock vorliegt, beschreiben Sie diesen.

3 Deadlock (25)

Im folgenden Beispiel konkurrieren 5 Prozesse um 4 verschiedene Ressourcen. Zum betrachteten Zeitpunkt sind 2 Ressourcen vom Typ 1, und keine Ressource des Typs 2,3 oder 4 verfügbar.

Die *allocation Matrix* A gibt die gegenwärtig von jedem der 5 Prozesse allokierten Ressourcen an, die Matrix Q beschreibt die momentan von den Prozessen angeforderten und noch nicht gewährten Ressourcen (in den Matrizen repräsentiert jede Spalte eine Ressource und jede Zeile einen Prozess).

Kann in dieser Situation ein Deadlock vorliegen ? Führen Sie den *deadlock detection* Algorithmus durch, um diese Frage zu beantworten.

$$A = \begin{pmatrix} 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 2 & 0 & 1 \end{pmatrix} \quad Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 3 & 3 & 0 & 4 \\ 1 & 2 & 1 & 1 \\ 1 & 0 & 2 & 0 \end{pmatrix}$$

Ein Deadlock kann aufgehoben werden, in dem den Prozessen einige Ressourcen entzogen werden. Wie kann im vorangegangen Beispiel durch das Entziehen genau einer allokierten Ressource sichergestellt werden, dass kein Deadlock mehr vorliegt ?

[redacted]

Welche anderen Strategien zur Aufhebung eines Deadlocks kennen Sie ?

[redacted]

Beschreiben Sie eine Möglichkeit, um das Auftreten der notwendigen Deadlock Bedingung "*circular wait*" auszuschliessen (*direct deadlock prevention*). Die gleichzeitige Nutzung unterschiedlicher Ressourcen soll weiterhin möglich sein.

[redacted]

Welche weiteren Bedingungen sind für einen Deadlock notwendig ? Erklären Sie deren Bedeutung.

[redacted]

3 Deadlock (23)

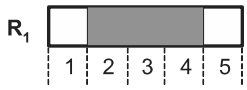
Gegeben sind zwei Prozesse, P_1 und P_2, die jeweils die Ressourcen R_1 und R_2 benötigen. Jede der zwei Ressourcen ist zweimal vorhanden. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert.

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

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

1. Tragen Sie die Anforderungen von Ressourcen für P_1 bzw. P_2 ein. Dabei ist anzunehmen, dass eine Ressource bereits ab Start der Anweisung `get()` als belegt gilt und erst nach Beendigung der Anweisung `free()` als wieder freigegeben gilt:

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

   Sind mehrere Instanzen einer Ressource belegt, so geben Sie die zuletzt belegte Instanz frei.

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

3. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll.

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

5. Beschriften Sie einen Punkt im Koordinatensystem (d.h. schreiben Sie den jeweiligen Buchstaben im Kästchen links unterhalb) ...

   ... mit 'A', von welchem aus der Punkt *end* und welcher vom Punkt *begin* erreichbar ist.

   ... mit 'B', welcher unweigerlich zu einen Deadlock führt, sofern ein solcher Punkt vorhanden ist.

   ... mit 'C', welcher einen nicht erlaubten Zustand darstellt, sofern ein solcher Punkt vorhanden ist.

Anmerkung: Achten Sie bitte darauf, dass alle Lösungen gut erkennbar und die Lösungen zu den Teilaufgaben 2 und 3 *deutlich unterscheidbar* sind.

Program P_1:

```
 1: a=8;
 2: get(R1);
 3: get(R2);
 4: get(R1);
 5: b=3;
 6: a=b+2;
 7: c=b*a;
 8: b=c-a;
 9: free(R2);
10: a=b*3;
11: free(R1);
12: b=a;
13: free(R1);
14: return;
```

Program P_2:

```
 1: a=54;
 2: get(R1);
 3: get(R2);
 4: get(R2);
 5: get(R1);
 6: c=a+b;
 7: free(R2);
 8: a=2*b*c;
 9: free(R1);
10: free(R2);
11: a=b*5;
12: free(R1);
13: b=5+d;
14: return;
```