Hans-Georg Eßer
Betriebssysteme
Fakultät Informatik/Mathematik
Hochschule München

Datum: 02.07.2009 **Name:** _____ **Vorname:** _____

Arbeitszeit: 70 Minuten **Matr.-Nr.:** _____

Hilfsmittel: Taschenrechner **Unterschrift:** _____

Muster-lösung

wird vom Prüfer ausgefüllt

1	2	3	4	5	6	7	8	9	10	11	Σ

Diese Probeklausur hat **reduzierten Umfang** (70 statt 90 Minuten in der Prüfung). Die tatsächliche Prüfung wird eine „**Auswahlklausur**" sein (es reicht, einen Teil der Aufgaben zu bearbeiten, um 100 oder mehr Prozent der Punkte zu erreichen), in dieser Probeklausur bearbeiten Sie bitte alle Aufgaben (Summe: 80 Punkte).

1. Speicherverwaltung (6/80 Punkte)

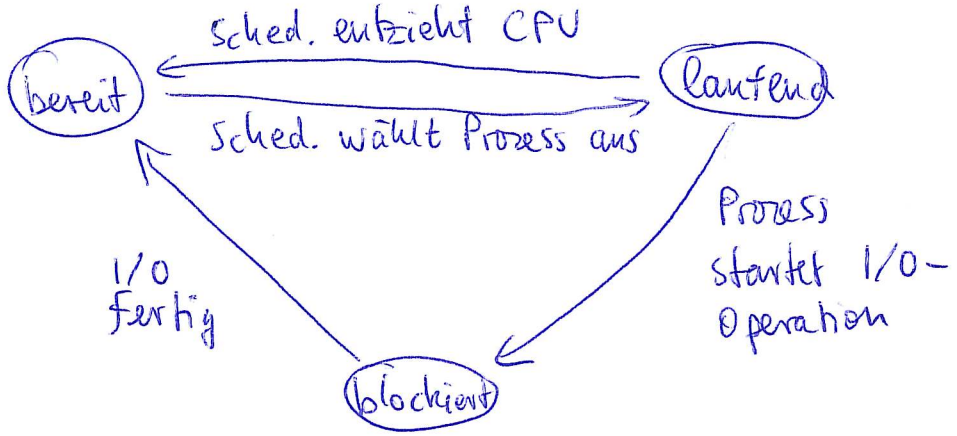
Bei der Speicherverwaltung gibt es die Phänomene der **internen Fragmentierung** und **externen Fragmentierung**. Erläutern Sie den Unterschied zwischen den beiden Fragmentierungstypen und nennen Sie je ein Beispiel, bei dem es zu diesen Arten der Fragmentierung kommt.

Intern: Speicherbereiche, die einem Prozess zugeordnet wurden, werden nicht vollständig genutzt, z. B. bei Aufteilung des Speichers in Partitionen fester, gleicher Größe und einem Prozess, der viel weniger Speicher benötigt

Extern: Im Zuge mehrfacher Vergabe und Rückgabe von Speicher bleiben kleine „Lücken": Speicherbereiche, die keinem Prozess zugeordnet sind und die so klein sind, dass vermutlich auch in der Zukunft kein Prozess sie verwenden kann. Z. B. bei dynamischer Aufteilung in (zusammenhängende) Partitionen.

2. Prozess-Zustände (8/80 Punkte)

a) Die drei wichtigsten Prozesszustände sind **bereit**, **laufend** und **blockiert**. Beschreiben Sie, welche Übergänge zwischen diesen Prozessen möglich sind und warum es zu diesen Übergängen kommt. (Es reicht aus, die Zustände und Übergänge in einer Grafik zu zeichnen und die Übergangspfeile mit Stichworten zu erläutern.)



b) **Threads** und **Prozesse** wechseln zwischen unterschiedlichen Zuständen hin und her. Nennen Sie zwei Zustände, die es nur bei Prozessen gibt, und begründen Sie jeweils kurz, warum es nicht sinnvoll ist, diese Zustände für Threads zu definieren.

- *„swapped": Wäre der gesamte Speicher eines Threads ausgelagert, dann zwingend auch der (identische) gesamte Speicher des Prozesses, zu dem der Thread gehört*
- *„zombie": Zombies sind Prozesse, die bereits beendet sind, deren Rückgabewert aber noch nicht vom Vaterprozess abgeholt wurde. Das gibt es bei Threads nicht.*

3. System calls (10/80 Punkte)

a) Nach dem Öffnen einer Datei mit **`fd=open(...)`** wollen Sie lesend und schreibend auf die Datei zugreifen. Geben Sie die Syntax der Befehle zum Lesen und Schreiben an, d. h. nennen Sie die Parameter der beiden Funktionen und erklären Sie die Bedeutungen der Parameter. (Die korrekte Reihenfolge der möglichen Parameter ist dabei nicht wichtig.)

```
int buffer[20];
fd = open ("test.dat", O_RDWR);
read (fd, &buffer, 20);
write (fd, &buffer, 20);
```

fd ist der File-Descriptor, der aus open() kommt, buffer ist der Puffer, der beim Lesen die Daten aufnimmt bzw. sie beim Schreiben enthält, das letzte Argument bei read/write ist die Anzahl Bytes, die die Funktion lesen/schreiben soll.

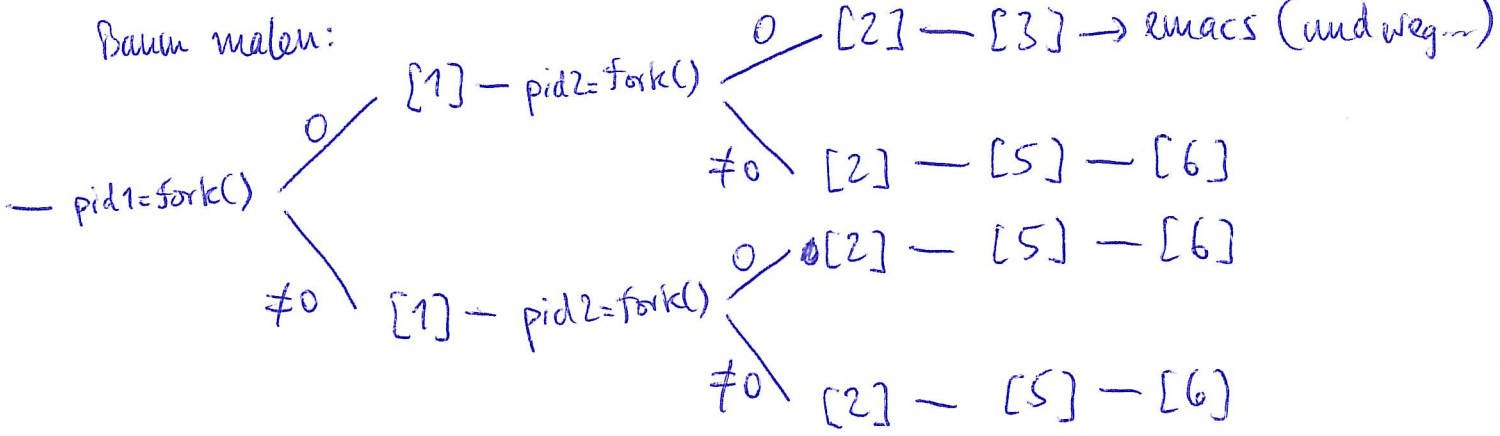
b) Betrachten Sie den folgenden Programmausschnitt:

```
...
int pid1 = fork();
printf ("%s\n","[1] Ein Fork ist durch, einer muss noch.");
int pid2 = fork();
printf ("%s\n","[2] Zeit für eine Fallunterscheidung.");
if ( (pid1==0) && (pid2==0) ) {
  printf ("%s\n","[3] Ich starte jetzt emacs.");
  execl ("/bin/emacs", "/etc/fstab", (char *)NULL);
  int pid3 = fork();
  printf ("%s\n","[4] Nach dem dritten Fork.");
} else {
  printf ("%s\n","[5] Ich gucke nur zu.");
};
printf ("%s\n","[6] Nach der if-Abfrage endet das Programm.");
...
```

} wird nie ausgeführt!

Wie oft und warum erscheinen die mit [1] bis [6] durchnummerierten Ausgaben? Schreiben Sie zu jeder Ausgabe die Anzahl und begründen Sie Ihre Antwort stichwortartig.

Baum malen:



c) Beim Aufruf des System calls **`fork()`** erhält der Sohn den Wert 0 und der Vater die Prozess-ID des neu erzeugten Sohnes (≠0) zurück. Für eine reine Unterscheidung („wer bin ich?“) könnte man auch umgekehrt arbeiten, also im Vater den Wert 0 und im Sohn die Prozess-ID des Vaters (≠0) zurückgeben. Warum ist diese Alternative schlecht?

Der Sohn kann immer über getppid() (get parent process id) die ID des Vaterprozesses rausfinden, denn da gibt es ja nur einen. Anders rum kann aber der Vater nicht die PID eines bestimmten Sohnes über einen Funktionsaufruf erfahren – es könnte ja mehrere geben.

4. Scheduling-Verfahren (Uni-Prozessor) (11/80 Punkte)

a) Aus der Vorlesung kennen Sie die Scheduling-Verfahren **FCFS** (First Come First Served), **SRT** (Shortest Remaining Time Next) und Round Robin (**RR**).

Es gebe die folgenden fünf Prozesse mit den angegebenenen Ankunftszeiten und Gesamtrechenzeiten:

Prozess	Ankunft	Rechenzeit	Prozess	Ankunft	Rechenzeit
P	0	10	S	6	1
Q	4	8	T	12	2
R	5	7			

Für First Come First Served sieht die Ausführreihenfolge wie folgt aus:

Zeit	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7
		10	20
FCFS	**P** P P P P P P P P P	**Q** Q Q Q Q Q Q Q **R** R	R R R R R **S** **T** T
SRT	**P** P P P P P P P P P	**S** **R** R R R R R R **T** T	**Q** Q Q Q Q Q Q Q
RR (q=2)	**P** P P P **Q** Q **P** P **R** R	**S** **Q** Q **P** P **R** R **T** T **Q**	Q **P** P **R** R **Q** Q **R**
RR (q=8)	**P** P P P P P P P **Q** Q	Q Q Q Q Q Q **R** R R R	R R R **S** **P** P **T** T

Achtung: Lösung beschreibt SJF, nicht SRT!

Ergänzen Sie für SRT und RR hier in der Tabelle die Ausführreihenfolgen. Für RR nehmen Sie ein Zeitquantum von q=2 bzw. q=8 Zeiteinheiten an. (Neue Jobs werden bei RR im Moment ihrer Ankunft hinten an die aktuelle Warteschlange angehängt.)

q=2:

0: [P]
[] (P)

4: [Q,P]
[P] (Q)
erst neues Q, dann unterbrochenes P

5: [P,R] (Q)
neues R

6: [P,R,S,Q]
[R,S,Q] (P)
neues S

8: [R,S,Q,P]
[S,Q,P] (R)

10: [S,Q,P,R]
[Q,P,R] (S)

11: [Q,P,R]
[P,R] (Q)

12: [P,R,T] (Q)
neues T

13: [P,R,T,Q]
[R,T,Q] (P)

15: [R,T,Q,P]
[T,Q,P] (R)

usw.

b) Wenn Sie FCFS mit RR(q=2) und RR(q=8) vergleichen, was fällt Ihnen dann auf? Bewerten Sie die Wahl des Zeitquantums q=2 bzw. q=8.

Kurze Prozesse (wie S) kommen früher dran (und werden damit auch früher fertig), weil die Warteschlange schneller rotiert. Dafür brauchen lange Prozesse (wie P) etwas länger, das ist aber nicht so tragisch. Bei q=2 fallen natürlich bis zu 4x so viele Context-Switches an wie bei q=8.

c) Was ist der Unterschied zwischen **I/O-lastigen Prozessen** und **CPU-lastigen Prozessen**?

I/O-lastig: verbringt die meiste Zeit mit dem Warten (Blockiert-sein) auf I/O-Operationen

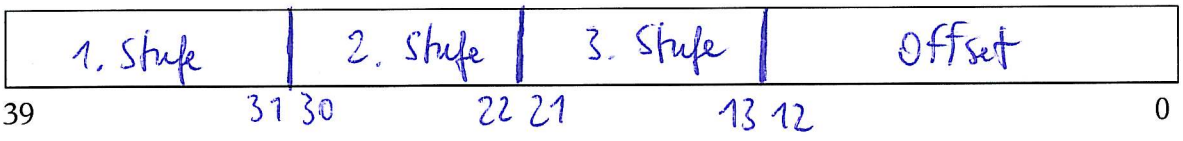
CPU-lastig: rechnet die meiste Zeit

5. Virtuelle Adressen (6/80 Punkte)

Eine CPU arbeitet mit folgenden Werten:

- Seitengröße: 8 KByte = 8192 Byte = 2^{13} Byte ⇒ 13 Bit Offset
- 40 Bit lange virtuelle Adressen → 40 − 13 = 27 Bit für Seitennummer
- 3-stufiges Paging; alle Seitentabellen sind gleich groß → 27/3 = 9 Bit pro Stufe
- Seitentabelleneinträge sind 8 Byte lang

a) Wie ist eine virtuelle Adresse aufgebaut (welche Bits der Adresse haben welche Bedeutung)?



Zeichnen Sie die die Unterteilung hier ein und beschriften Sie die Abschnitte geeignet.

b) Wie viele Seitentabellen der verschiedenen Stufen gibt es? Wie groß sind diese Tabellen?

Zur Größe: Auf jeder Stufe 2^9 = 512 (partielle) Seitennummern, alle je 2^9 Einträge. Jeder Eintrag ist 8 Byte lang
⇒ Größe = 512 × 8 = 4096 = 4 K (oder ½ Seite)

Zur Anzahl:

- 1. Stufe: 1 äußere Seitentabelle
- 2. Stufe: 2^9 = 512 mittlere Seitentabellen
- 3. Stufe: $2^9 \times 2^9$ = 262144 = 256 K innere Tabellen

6. Synchronisation (8/80 Punkte)

a) Der **Linux-Kernel** verwendet zur Synchronisation **Semaphore**. Wie unterscheiden sich diese von **Spin-Locks** und welche Auswirkungen hat das auf ihre Verwendbarkeit innerhalb von Interrupt-Handlern?

Semaphore schlafen, wenn eine Ressource nicht mehr verfügbar ist. Darum können sie nicht in Interrupt-Handlern benutzt werden.

b) Mutexe und Semaphore helfen Programmierern bei der Synchronisation, ersparen ihnen aber nicht, den Programmcode sorgfältig nach kritischen Bereichen zu durchsuchen. **Monitore** sind eine Alternative – auf welche Weise erleichtern sie Programmierern die Arbeit?

Monitore kapseln die Daten (auf die potenziell von mehreren Threads zugegriffen wird) und die kritischen Bereiche (in denen diese Zugriffe stattfinden). Wie bei einer „private" deklarierten Variable kommen Programme nur noch über im Monitor definierte Monitorprozeduren (Funktionen) an diese Daten heran. Damit entfällt das manuelle Schützen aller kritischen Bereiche mit Mutexen.

7. Dateisysteme (4/80 Punkte)

a) Ein Dateisystem arbeite mit bis zu zweifacher **Indirektion**. Erklären Sie, warum aus einer Verdopplung der Blockgröße eine (ca.) Ver-8-fachung der maximalen Dateigröße resultiert.

Jeder Datenblock verdoppelt seine Größe -> Faktor 2

Jeder Index-Block verdoppelt seine Größe, also gibt es darin doppelt so viele Blockadressen wie vorher. Da es zweistufige Indirektion gibt -> Faktor 2 x 2 = 4

Zusammen: Faktor 2 x 4 = 8

b) Erklären Sie den Unterschied zwischen **symbolischen Links** (**Soft Links**) und **Hard Links**.

Symbolische Links enthalten einen Pfadnamen, um auf eine Datei zu verweisen (die sich z. B. auch auf einem anderen Dateisystem befinden darf)

Hard Links sind zusätzliche Verzeichniseinträge, die auf denselben Inode (wie ein bereits vorhandener Eintrag) zeigen.

8. Dateigrößen (7/80 Punkte)

Ein Unix-Dateisystem habe folgende Eigenschaften:
- Blockgröße: 16 KByte
- Dateisystemgröße: 16 GByte
- zweifache Indirektion;
  - 8 direkte Verweise
  - 4 einfach indirekte Verweise
  - 2 zweifach indirekte Verweis

Berechnen Sie

a) die Mindestgröße einer Blockadresse in Bit (und daraus abgeleitet die tatsächlich verwendete Größe in Byte),

b) die Anzahl der Blockadressen, die in einen Block passen,

c) und daraus die maximale Dateigröße.

a) $\frac{16\ GB}{16\ KB} = 1\,M \Rightarrow$ 1 M Blöcke

$1\,M = 2^{10} \Rightarrow$ 10 Bit für Blockadresse

$\Rightarrow$ aufrunden: 2 Byte (16 Bit)

b) $\frac{16\ KB}{2\ B} = 8\,K$, also 8192 Adressen/Block

c) max. Dateigröße:

$$\left\{ \begin{array}{rcl} 8 & \times & 1 \\ +\ 4 & \times & 8192 \\ +\ 2 & \times & 8192^2 \end{array} \right\} \times 16\,K$$

(Anzahl Blöcke) × (Blockgröße)

$= (8 + 32768 + 134217728) \times 16\,K$

$= 2\,199\,560\,257\,536$

oder als Näherung:

$(8 \cdot 1 + 4 \cdot 8K + 2 \cdot (8K)^2) \times 16K$

$= (8 + 32\,K + 128\,M) \times 16\,K$

$\approx 128 \times 16\ M \times K$

$= 2K \times M \times K$

$= 2$ Terabyte

9. Paging (6/80 Punkte)

a) Was sind **invertierte Seitentabellen** und was ist der Grund für ihren Einsatz?

Invertierte Seitentabellen ordnen Seitenrahmen Seiten zu (normale machen es umgekehrt). Das soll helfen, Platz zu sparen, weil reguläre Seitentabellen sehr groß sind.

b) Welche Aufgabe hat ein **Translation Look-Aside Buffer** (TLB)? Dank welchen Prinzips ist er in der Regel auch dann hilfreich, wenn er sehr klein ist? Erklären Sie Ihre Antwort. (Es reicht nicht aus, den Namen des Prinzips aufzuschreiben.)

Der TLB beschleunigt die Adressübersetzung (virtuelle Adresse -> physikalische Adresse), indem er Zuordnungen Seite -> Seitenrahmen cachet. Da er ein Assoziativspeicher ist, findet die MMU darin Einträge sehr schnell und kann dann auch zusätzliche Speicherzugriffe (um aus der Seitentabelle zu lesen) verzichten.

Das Lokalitätsprinzip „verspricht", dass Speicherzugriffe meist auf Adressen erfolgen, die in der Nähe von kurz vorher verwendeten Adressen (und damit evtl. in derselben Seite) liegen. Das ist z.B. beim sequentiellen Abarbeiten von Programmcode und bei Datenzugriffen in Schleifen, die sich durch Arrays arbeiten, der Fall.

10. Deadlocks (4/80 Punkte)

a) Auf einem System gebe es fünf Prozesse P_1, P_2, ..., P_5 und fünf exklusive Betriebsmittel R_1, R_2, ..., R_5. Der momentane Zustand sei wie folgt:

P_1 belegt keine Ressource und fordert R_1 an, ✓

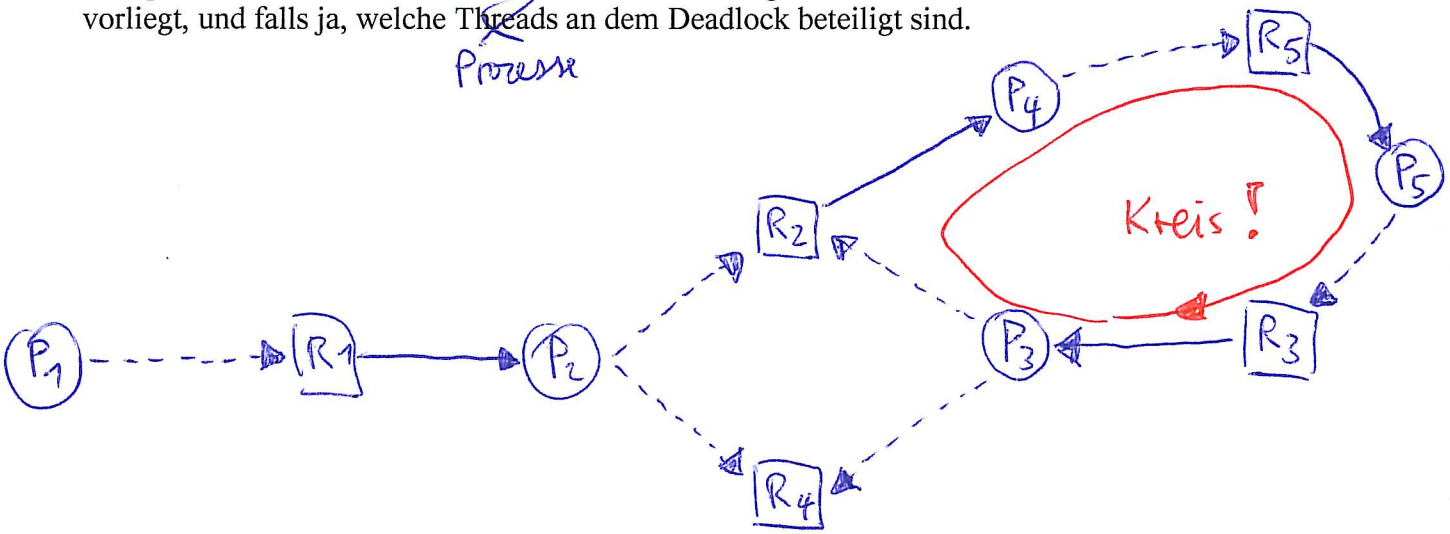
P_2 belegt R_1 und fordert R_2 und R_4 an, ✓

P_3 belegt R_3 und fordert R_2 und R_4 an, ✓

P_4 belegt R_2 und fordert R_5 an, ✓

P_5 belegt R_5 und fordert R_3 an.

Überprüfen Sie mit einem der beiden in der Vorlesung behandelten Verfahren, ob ein Deadlock vorliegt, und falls ja, welche ~~Threads~~ Prozesse an dem Deadlock beteiligt sind.



Deadlocks der Prozesse P3, P4, P5

11. Deadlocks: Banker-Algorithmus (10/80 Punkte)

Für drei Ressourcen-Klassen A, B und C sowie vier Prozesse P_1, P_2, P_3, P_4 seien folgende Ressourcenbelegungen (**C**) und Maximalanforderungen (**Max**) gegeben:

$$\mathbf{C} = \begin{pmatrix} 2 & 0 & 0 \\ 2 & 2 & 0 \\ 0 & 3 & 1 \\ 0 & 1 & 2 \end{pmatrix} \quad \mathbf{Max} = \begin{pmatrix} 2 & 1 & 0 \\ 5 & 7 & 0 \\ 2 & 5 & 1 \\ 3 & 9 & 2 \end{pmatrix}$$

E = (5 9 4)
A = (1 3 1)

A gibt die aktuell noch verfügbaren Ressourcen an, E die Gesamtressourcen.

a) Ist dieses System in einem **sicheren Zustand**? Falls ja, geben Sie eine Ausführreihenfolge der Prozesse an, in der jeder Prozess zunächst seine maximalen Ressourcenforderungen stellt und anschließend alle belegten Ressourcen freigibt.

b) Prozess P_4 stellt die Anfrage (1 1 0) an das System. Prüfen Sie mit dem **Banker-Algorithmus**, ob das System diese Anfrage bedienen darf, wenn es Deadlocks vermeiden will.

a)

$$R = Max - C = \begin{pmatrix} 2 & 1 & 0 \\ 5 & 7 & 0 \\ 2 & 5 & 1 \\ 3 & 9 & 2 \end{pmatrix} - \begin{pmatrix} 2 & 0 & 0 \\ 2 & 2 & 0 \\ 0 & 3 & 1 \\ 0 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 3 & 5 & 0 \\ 2 & 2 & 0 \\ 3 & 8 & 0 \end{pmatrix}$$

E = (5 9 4)
A = (1 3 1)

	131
200	010
220	350
031	220
012	380

→

	331
~~200~~	~~010~~
220	350
031	220
012	380

→

	361
~~200~~	~~010~~
220	350
~~031~~	~~220~~
012	380

→

	581
~~200~~	~~010~~
~~220~~	~~350~~
~~031~~	~~220~~
012	380

✓ passt!

Reihenfolge: P_1 P_3 P_2 P_4

b) Zusätzliche Anfrage (110) durch P_4 führt zu:

	021
200	010
220	350
031	220
122	270

→

	221
~~200~~	~~010~~
220	350
031	220
122	270

→

	252
~~200~~	~~010~~
220	350
~~031~~	~~220~~
122	270

↳ geht nicht weiter

⇒ Zustand nach P_4/(110) unsicher!