Hochschule RheinMain
Studiengang Angewandte Informatik B.Sc.
Prof. Dr. Robert Kaiser

# Probeklausur Betriebssysteme ohne Rechnerarchitektur!(LV????)

Wintersemester 2017/2018

| Nachname: **Mustermeier** | Vorname: **Theo** |
|---|---|
| Matrikelnummer: **VOID** | |
| Datum: 22.02.2018 | **Unterschrift:** |

Sie erhalten eine geheftete Klausur. **Bitte lösen Sie die Heftung nicht**. Bitte tragen Sie zu Beginn der Bearbeitungszeit Ihren Namen und Ihre Matrikelnummer an den dafür vorgesehenen Stellen ein und unterschreiben Sie die Klausur. Die Klausur ist **nur mit Unterschrift** gültig. Die Klausur muss mit dem Verlassen des Raumes abgegeben werden.

**Zum Bestehen der Klausur sind 22 Punkte (50%) notwendig**

Im Falle nicht ausreichenden Platzes benutzen Sie bitte zusätzliche Blätter, die Sie mit Name und Matrikelnummer versehen. Machen Sie bitte eindeutig kenntlich, auf welche Aufgabe sich Ihre Antwort bezieht.

Dauer: 90 min
Hilfsmittel: Taschenrechner für arithmetische Operationen, eigene Formelsammlung von maximal einer doppelseitig handbeschriebenen DIN A4 Seite.

Punkte:

| Aufgabe | Soll-Punkte | Ist-Punkte |
|---|---|---|
| 1 | 5 | |
| 2 | 8 | |
| 3 | 4 | |
| 4 | 4 | |
| 5 | 6 | |
| 6 | 4 | |
| 7 | 4 | |
| 8 | 4 | |
| 9 | 6 | |
| **Gesamt** | 45 | |

Note:

## Aufgabe 1: (5 Punkte)

Beantworten Sie bitte folgende Fragen:

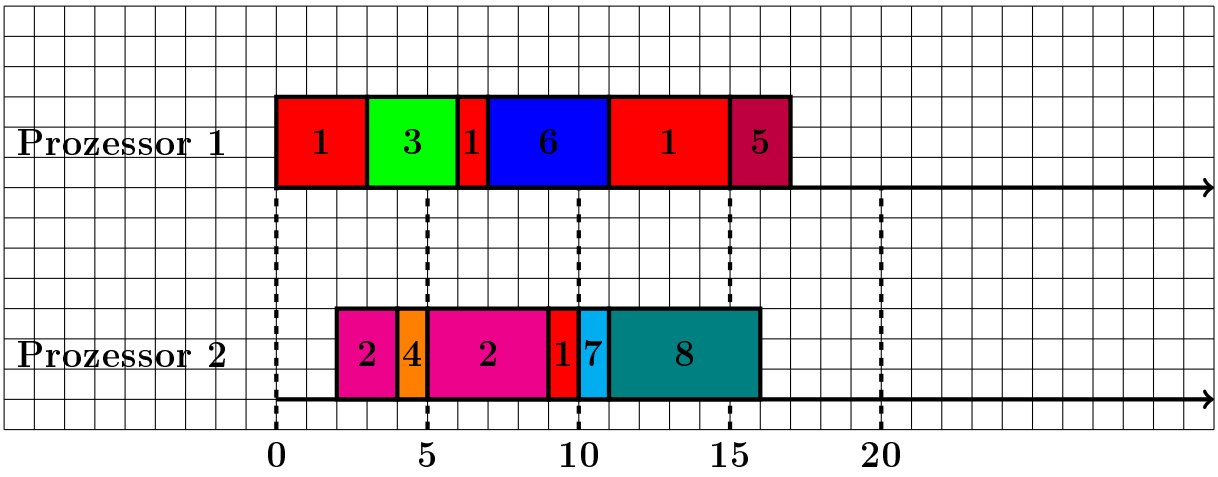
| **Frage** | **Antwort** |
|---|---|
| Warum sind Textdateien zwischen verschiedenen Betriebssystemen (UNIX / MacOS / Windows) nicht ohne Weiteres übertragbar? | **Unterschiedliche Sonderzeichen für Zeilenende** |
| Was kennzeichnet eine „race condition“? | **Ergebnis abhängig von Prozessreihenfolge** |
| Werden mit `semget()` erzeugte Semaphore in C automatisch zerstört, wenn der erzeugende Prozess terminiert? | **Nein** |
| Was kann bei preemptivem Multithreading vorkommen, bei kooperativem Multithreading hingegen nicht? | **Preemption, Unterbrechung lfd. Prozess, Prozessor-Entzug,...** |
| Wie viele verschiedene Werte kann eine Variable vom Typ `char` annehmen? | 256 |
| Welche Prozess-ID hat bei einem UNIX-System der init-Prozess? | **eins** |
| Wodurch wird bei einem Betriebssystemaufruf der Wechsel in den privilegierten Modus bewerkstelligt? | **Durch einen „Trap”-Befehl** |
| Mit dem „Peterson-Algorithmus“ kann wechselseitiger Aussschluss zwischen maximal .... konkurrierenden Prozessen oder Threads sichergestellt werden. | **zwei** |
| Wie sind unter UNIX Dateiverweise („Links“) über Dateisystemgrenzen realisierbar | **Als symbolische Links** |
| Warum sind SSD-Festplatten i.d.R. schneller als herkömmliche magnetisch aufzeichnende Festplatten? | **Keine mechanisch bewegten Teile** |

## Aufgabe 2: (8 Punkte)

Die folgenden Aufträge treffen in einem 2-Prozessorsystem zu den angegebenen Zeitpunkten ein. Eine höhere Zahl für die Priorität drücke eine höhere Wichtigkeit aus. Die Aufträge seien reine Rechenaufträge ohne I/O. Die Prozesswechselzeiten werden vernachlässigt.

| Auftrag | Ankunftszeit | Bedienzeit | Priorität | **Abschlusszeit** | **Verweilzeit** | **Mittel** |
|---|---|---|---|---|---|---|
| 1 | 0 | 9 | 2 | **15** | **15** | |
| 2 | 2 | 6 | 4 | **9** | **7** | |
| 3 | 3 | 3 | 5 | **6** | **3** | |
| 4 | 4 | 1 | 6 | **5** | **1** | |
| 5 | 5 | 2 | 1 | **17** | **12** | |
| 6 | 7 | 4 | 3 | **11** | **4** | |
| 7 | 10 | 1 | 4 | **11** | **1** | |
| 8 | 11 | 5 | 2 | **16** | **5** | |
| | | | | | **48** | **6** |

a) Geben Sie für das unterbrechende Prioritäts-Scheduling-Verfahren ein Belegungs-Diagramm für die beiden CPUs für die Abarbeitung der Prozesse an. **(3P)**




b) Bestimmen Sie die mittlere Verweilzeit. (Sie können die freien Spalten in der Tabelle verwenden). **(1P)**

**(Siehe Werte in Tabelle), Verweilzeit = Abschlußzeit - Ankunftszeit**

**Mittlere Verweilzeit = Summe aller Verweilzeiten / Anzahl Prozesse**

c) Diskutieren Sie das Verhalten eines Round-Robin-Schedulers, wenn das Quantum $Q$ entweder gegen 0 oder gegen unendlich konvergiert und die Zeitdauer für einen Kontextwechsel entweder als Null oder als Konstante $c$ angenommen wird. **(2P)**

$Q \to 0, c = 0 \Rightarrow$ **Processor sharing**
$Q \to 0, c \neq 0 \Rightarrow$ **Massiver Overhead: Alle Rechenzeit wird für Kontextwechsel verbraucht**
$Q \to \infty \Rightarrow$ **Konvergiert gegen First Come First Serve, Run to Completion**

## Aufgabe 3: (4 Punkte)

Betrachten Sie das **Leser-Schreiber**-Problem: mehrere Prozesse versuchen, lesend bzw. schreibend auf einen Datenbestand zuzugreifen. Dabei sind **mehrere gleichzeitig lesende** Prozesse **zugelassen**, **mehrere, gleichzeitig schreibende** Prozesse oder gleichzeitiges Lesen und Schreiben jedoch **nicht**. Geben Sie eine Lösung für die Synchronisation an, basierend auf Semaphoren, die als Typ `sem` gegeben seien. Folgende Funktionen seien verfügbar:

| | |
|---|---|
| `void C(sem *s, int anfwert)` | Initialisieren eines Semaphors mit vorgegebenem Anfangswert für den Zähler |
| `int S(sem *s)` | Aktuellen Zählerstand eines Semaphors ermitteln |
| `P(sem *s)` | P-Operation nach Dijkstra: Zähler dekrementieren, ggf. Blockieren |
| `V(sem *s)` | V-Operation nach Dijkstra: Zähler inkrementieren, ggf. De-Blockieren |

Ergänzen Sie das im folgenden angegebene C-Code-Skelett um die Aufrufe zur Initialisierung der Semaphoren sowie zur korrekten Synchronisation im Leser-Programm (reader) und im Schreiber-Programm (writer).
Hinweis: Beachten Sie, dass es **keinen** gemeinsamen Speicher zwischen den beteiligten Prozessen gibt. Prozesskommunikation und -Synchronisation kann somit nur über Semaphoren, nicht über gemeinsame Variablen erfolgen.

```
/* Definition der Semaphoren */

sem

mutex, db, nLeser;   /* 0,5 Punkte */

/* Initialisierung der Semaphoren */

C(&mutex, 1);
C(&db, 1);        /* 0,5 Punkte */
C(&nLeser, 0);
```

```
void writer(void)    /* Schreiber */
{
   int item;
   while(TRUE)
   {


      produce_item(&item);  /* erzeuge Datum */


       P(&db);          /* 1 Punkt: Exklusiv-Zugriff sichern */
      write_item(item); /* Schreibe Datum in  den Puffer */


       V(&db);          /* Zugriff wieder freigeben */
} }
void reader(void)     /* Leser */
{
   int item;
   while(TRUE)
   {
       P(&mutex);
       V(&nLeser);
       if(S(&nLeser) == 1)                    /* 2 Punkte */
           P(&db);
       V(&mutex);
      read_item(&item); /* lies Eintrag  aus Puffer */
       P(&mutex);
       P(&nLeser)
       if(S(&nLeser) == 0)
           V(&db);
       V(&mutex);
      use_data(item);  /* verarbeite Eintrag */



} }
```

## Aufgabe 4: (4 Punkte)

Gegeben sei ein Rechner mit 64 Byte physischem Speicher. Die Seitengröße des Rechners betrage 16 Byte. Die Seitentabelle habe folgenden Inhalt:

| **Index** | **Inhalt** |
|---|---|
| 0 | 3 |
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |

Tragen Sie in der unten angegebenen, tabellarischen Darstellung des physischen Speichers ein, welche Inhalte durch die Ausführung des folgenden Programms in welche Speicherzellen geschrieben werden. (**4 P**)

**Hinweis:** Skizzieren Sie am besten zunächst die Inhalte des virtuellen Speichers in einer eigenen tabellarischen Darstellung und übertragen Sie diese anschließend anhand der gegebenen Seitentabelle in den physischen Speicher. Achten Sie dabei auf „Aliases“, d.h. physische Seitenrahmen die unter mehreren verschiedenen Adressen im virtuellen Adressraum erscheinen.

```
main()
{
    char *string = (char*)10;
    char *s      = (char*)48;

    strcpy(string, "Hallo Welt");
    strcpy(s, string);
}
```

| **Adresse** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | |
| 16 | **W** | **e** | **l** | **t** | **\0** | | | | | | | | | | | |
| 32 | **H** | **a** | **l** | **l** | **o** | **’ ’** | **W** | **e** | **l** | **t** | **\0** | | | | | |
| 48 | | | | | | | | | | | **H** | **a** | **l** | **l** | **o** | **’ ’** |

**Virtueller Speicher:**

| **Adresse** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | | | **H** | **a** | **l** | **l** | **o** | **' '** |
| **16** | **W** | **e** | **l** | **t** | **\0** | | | | | | | | | | | |
| **32** | | | | | | | | | | | | | | | | |
| **48** | **H** | **a** | **l** | **l** | **o** | **' '** | **W** | **e** | **l** | **t** | **\0** | | | | | |

## Aufgabe 5: (6 Punkte)

Eine Speicherverwaltung arbeitet mit einer Allokations-Bitmap: Ein belegter Speicherblock wird durch eine „1“ in der Bitmap repräsentiert, d.h. Bit Nummer $N$ der Bitmap ist gesetzt, wenn Block Nummer $N$ belegt ist. Der gesamte von der Speicherverwaltung verwaltete Speicher ist 160.000 Bytes groß. Die Zuteilung geschieht nach dem „first fit“-Verfahren, d.h. bei einer Anfrage wird stets ausgehend vom Anfang des Speichers das erste hinreichend große Stück zugeteilt.

a) Wie groß ist bei dieser Speicherverwaltung der Verwaltungs-Overhead[1] (in Prozent), wenn die gewählte Blockgröße

   a1) 1000 Byte oder

   a2) 256 Byte

   beträgt? **(1P)**

   **Zur Verwaltung wird ein Bit pro Speicherblock gebraucht. Somit:**

   **a1) Overhead bei 1000 Byte Blöcken:** $\frac{1}{1000 \cdot 8} \cdot 100 = 0,0125\%$

   **a2) Overhead bei 256 Byte Blöcken:** $\frac{1}{256 \cdot 8} \cdot 100 = 0,048828125\%$

b) Die Speicherverwaltung beantwortet nun eine Folge von sechs Anfragen:

| **Anfrage Nr.** | **Angeforderte Anzahl Bytes** |
|---|---|
| 1 | 999 |
| 2 | 255 |
| 3 | 158.000 |
| 4 | 200 |
| 5 | 256 |
| 6 | 1 |

Ab welchem Punkt ist der gesamte Speicher erschöpft, wenn die Blockgröße

[1] Verhältnis zwischen für die Verwaltung benötigtem Speicher zu Nutz-Speicher

b1) 1000 Byte oder

b2) 256 Byte

beträgt? **(4P)**

Hinweise:

- Verwenden Sie die beiden nachfolgenden Tabellen
- Tragen Sie zunächst jeweils die Anzahl der belegten Blöcke ein
- Müssen mehr Blöcke belegt werden als vorhanden sind, so ist der Speicher erschöpft

b1) Tabelle für Blockgröße 1000 Byte:

| Nr. | Bytes | Kummuliert | Bel.Blöcke | erschöpft? | Bel.Speicher | Verlust | Fragm. |
|---|---|---|---|---|---|---|---|
| 1 | 999 | 999 | **1** | **-** | **1.000** | **1** | **0,10%** |
| 2 | 255 | 1254 | **2** | **-** | **2.000** | **746** | **37,30%** |
| 3 | 158.000 | 159.254 | **160** | **-** | **160.000** | **746** | **0,47%** |
| 4 | 200 | 159.454 | **161** | **ja** | **161.000** | **1546** | **0,96%** |
| 5 | 256 | 159.710 | **162** | **ja** | **162.000** | **2290** | **1,41%** |
| 6 | 1 | 159.711 | **163** | **ja** | **163.000** | **3289** | **2,02%** |

b2) Tabelle für Blockgröße 256 Byte:

| Nr. | Bytes | Kummuliert | Bel.Blöcke | erschöpft? | Bel.Speicher | Verlust | Fragm. |
|---|---|---|---|---|---|---|---|
| 1 | 999 | 999 | **4** | **-** | **1.024** | **25** | **2,44%** |
| 2 | 255 | 1254 | **5** | **-** | **1.280** | **26** | **2,03%** |
| 3 | 158.000 | 159.254 | **623** | **-** | **159.488** | **234** | **0,15%** |
| 4 | 200 | 159.454 | **624** | **-** | **159.744** | **290** | **0,18%** |
| 5 | 256 | 159.710 | **625** | **-** | **160.000** | **290** | **0,18%** |
| 6 | 1 | 159.711 | **626** | **ja** | **160.256** | **545** | **0,34%** |

**(Siehe Tabellen)**

**b1) Bei 1000 Byte Blöcken: ab Anfrage Nr. 4 erschöpft**

**b2) Bei 256 Byte Blöcken: ab Anfrage Nr. 6 erschöpft**

c) Berechnen Sie die interne Fragmentierung[2] (in %), die sich jeweils nach den oben angegebenen Speicheranfragen ergibt. Betrachten Sie auch hier wieder die beiden Blockgrößen 256 Byte und 1000 Byte. **(3P)**
Hinweise:

- Verwenden Sie auch hier die beiden Tabellen
- Der belegte Speicher ergibt sich aus den belegten Blöcken durch Multiplikation mit der Blockgröße
- Der Verlust ist die Differenz aus angefordertem Speicher und belegtem Speicher
- Die Fragmentierung ist das Verhältnis aus Verlust und angefordertem Speicher

**(Ergebnisse siehe Tabellen)**

[2]Anteil an nicht genutztem Speicher innerhalb der zugeteilten Speicherblöcke

## Aufgabe 6: (4 Punkte)

In der UNIX-Prozessverwaltung spielen die Systemfunktion `fork()` und die `exec()`-Familie eine zentrale Rolle.

a) Erläutern Sie kurz, was beim Aufruf von `fork()` geschieht. (**1P**)

**Kopie des aufrufenden Prozesses, Neuer Prozess (Sohn) mit Kopie der Speicherinhalte, offenen Dateien usw., Rückgabewert:**

- **Vater: PID des Sohns (oder -1 bei Fehler)**
- **Sohn: 0**

b) Welche Ausgabe erzeugt das folgende Programm? (**1P**) Hinweis: Nehmen Sie dabei an, dass die nächste vom System vergebene Prozess-ID die 1234 ist.

```
1  #include <unistd.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdio.h>
5  int main(void) {
6          int pid, n = 0;
7          printf("Ich habe pid %d\n",getpid());
8          pid = fork();
9          if(pid == 1) {
10                 perror("Fehler bei fork()");
11         } else if (pid == 0) {
12                 printf("Ich bin der Sohn!\n");
13         } else {
14                 printf("Ich bin Vater von pid %d\n",pid);
15         }
16         n = n + 1;
17         printf("%d Tschuess von %d\n", n, getpid());
18         return 0;
19 }
```

Ausgabe:

```
1  Ich habe pid 1234
2  Ich bin Vater von pid 1235
3  1 Tschuess von 1234
4  Ich bin der Sohn!
5  1 Tschuess von 1235
```

**(andere Ausgabe-Reihenfolge möglich!)**

c) Erläutern Sie kurz, was beim Aufruf von `exec()` geschieht.(**1P**)

**Angegebener Prozess wird geladen, überschreibt / ersetzt den laufenden Prozess, erbt dessen PID.**

d) Welche Ausgabe erzeugt das folgende Programm? (**1P**)
Hinweis: Dabei ist davon auszugehen, dass `/bin/echo` ein ausführbares Programm ist, das seine Argumente auf der Standardausgabe ausgibt.

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(void)
5  {
6      char *argv1[] = {"echo", "Hallo Otto!!", NULL};
7      int i;
8          for(i = 0; i < 15; i++)
9          {
10                 printf("%d\n", i);
11                 execv("/bin/echo", argv1);
12         }
13     printf("Ende\n");
```

```
14      return 0;
15  }
```

Ausgabe:

```
1  0
2  Hallo Otto!!
```

## Aufgabe 7: (4 Punkte)

Besonders im Zusammenhang mit Serversystemen hört man häufig den Begriff „RAID".

a) Was versteht man unter einem „RAID-System"? (**1P**) **Redundant Array of inexpensive/independent Disks: Mehrere Festplatten zusammengeschaltet, sehen nach außen wie eine aus.**

b) Erläutern Sie bitte die Eigenschaften und je einen Vorteil von RAID 0 und RAID 1 (**5P**).
RAID 0 bedeutet:**„Striping": → Keine Redundanz**

Ein Vorteil:**Schneller, hohe Kapazität**

RAID 1 bedeutet:**„Mirroring": Spiegelplatte, volle Redundanz**

Ein Vorteil:**Ausfallsicher**

c) Ein RAID 5 System werde mit vier Festplatten betrieben. Platten 1 bis 3 dienen der Nutzdatenhaltung, Platte 4 enthält Paritätsdaten. Plötzlich knirscht es vernehmlich in Platte 2 ihre Daten sind nicht mehr lesbar. Die ersten Bits der einzelnen Platten sehen nach diesem Unfall wie folgt aus:

Platte 1: 1 0 1 1 0 1 0 0 1 0 1 1

Platte 2: ? ? ? ? ? ? ? ? ? ? ? ? ?

Platte 3: 0 1 0 1 1 1 0 0 1 1 1 0

Platte 4: 1 1 1 0 0 1 1 0 0 0 1 1

Kann der Inhalt von Platte 2 automatisch wiederhergestellt werden? (bitte ankreuzen) (**1P**)

☑ Ja

☐ Nein

**Falls ja:** Erläutern Sie bitte den Wiederherstellungsvorgang und geben Sie die ersten 12 verlorenen Bits für Platte 2 an.(**2P**)
**Redundanzplatte enthält XOR über alle Platten**
**Wiederherstellung duch XOR-Verknüpfen der übrigen Disk-Inhalte:**
**0 0 0 0 1 1 1 0 0 1 1 0**

**Falls nein:** Erläutern Sie bitte Funktionsweise von RAID 5 und insbesondere die Bedeutung der Daten auf Platte 4.(**2P**)

## Aufgabe 8: (4 Punkte)

Der Informatik stehen insgesamt drei Laptops, zwei Arduinos und acht Raspberry Pi zur Verfügung, die von Herrn Beckmann an die Professoren verliehen werden.

Herr Gergeleit hat bereits einen Laptop und zwei Raspberry Pis, Herr Reith und Herr Thoss jeweils einen Arduino und einen Raspberry Pi. Zur Beendigung seiner Arbeiten benötigt Herr Reith zusätzlich einen Laptop und einen weiteren Arduino, während Herr Thoss drei weitere Laptops und drei Raspberry Pis haben möchte. Herr Gergeleit benötigt hingegen noch zwei Laptops und zwei Raspberry Pis. Nun stehen die drei Herren vor Herrn Beckmann und möchten bedient werden.

Untersuchen Sie bitte mit Hilfe des verallgemeinerten Bankier-Algorithmus, ob alle Professoren ihre Arbeiten sicher abschließen können, und schlagen Sie Herrn Beckmann eine entsprechende, verklemmungsfreie Reihenfolge zur Bedienung der Professoren vor. (Rechenweg/Begründung).

$$
\begin{array}{lcl}
 & & \begin{array}{ccc} \textbf{La} & \textbf{Ar} & \textbf{RPi} \end{array} \\
\mathbf{E} & = & \left(\begin{array}{ccc} \mathbf{3} & \mathbf{2} & \mathbf{8} \\ \mathbf{1} & \mathbf{0} & \mathbf{2} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \end{array}\right) \begin{array}{l} \\ \textbf{(Gergeleit)} \\ \textbf{(Reith)} \\ \textbf{(Thoss)} \end{array} \\
\mathbf{A} & = & \left(\begin{array}{cccc} \mathbf{2} & \mathbf{0} & \mathbf{4} \\ \mathbf{2} & \mathbf{0} & \mathbf{2} \end{array}\right) \\
\mathbf{R} & = & \left(\begin{array}{ccc} \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{3} & \mathbf{0} & \mathbf{3} \end{array}\right)
\end{array}
$$

**Reihenfolge:**

a) **Gergeleit** $\rightarrow A = (306)$

b) **Thoss** $\rightarrow A = (317)$

c) **Reith** $\rightarrow A = (328)$

## Aufgabe 9: (6 Punkte)

Zur Abbildung von Dateien finde z.B das Konzept der Allokationstabelle (z.B. MS-DOS FAT Dateisystem) Anwendung.

a) Ein Dateisystem enthalte nun eine Datei, welche (jeweils in dieser Reihenfolge) die physischen Blöcke 5, 7, 2, 9, 0 belegt, und eine weitere Datei, welche die Blöcke 1, 2, 3 belegt. Bitte vervollständigen Sie die nachfolgende Allokationstabelle entsprechend. Kennzeichnen Sie bitte auch die Anfänge (d.h. die ersten Blöcke) der beiden Dateien. Was fällt Ihnen auf? (**2P**)

| | **phys. Block** | **Verweis** | |
|---|---|---|---|
| | 0 | **-** | |
| **Beginn Datei 2→** | 1 | **2** | |
| | 2 | **9⇔3!!** | **←Inkonsistenz!!** |
| | 3 | **-** | |
| | 4 | | |
| **Beginn Datei 1→** | 5 | **7** | |
| | 6 | | |
| | 7 | **2** | |
| | 8 | | |
| | 9 | **0** | |

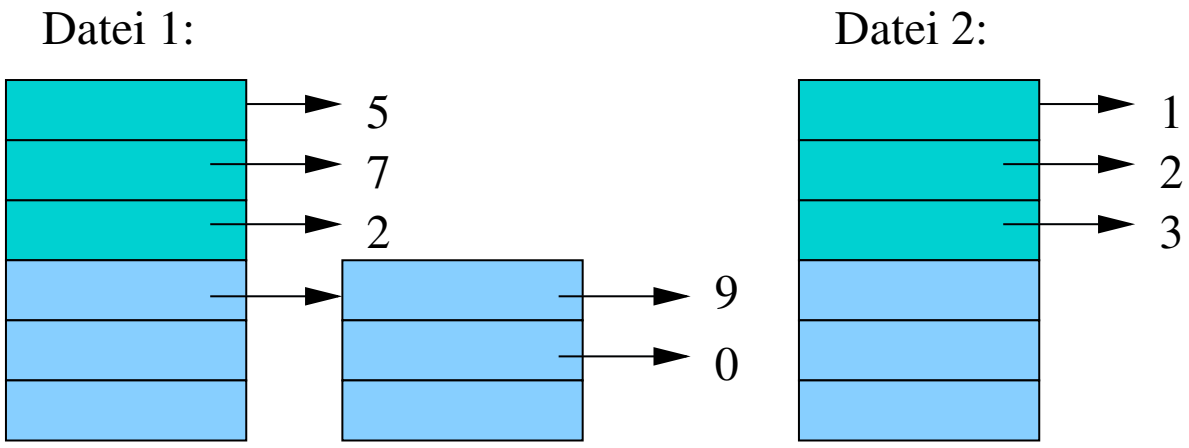
**Das Dateisystem ist inkonsistent: Block 2 wird von Dateien 1 und 2 beansprucht.**

b) Was ist der wesentliche Nachteil von FAT Dateisystemen bei der Verwaltung großer Platten?(**1P**)

**Die gesamte FAT muss im RAM gehalten werden**
**→ wird bei großen Platten sehr groß**

c) Skizzieren Sie die inode-Strukturen für die beiden Dateien, wenn statt eines FAT-ein inode-basiertes Dateisystem mit maximal drei direkten und drei einfach-indirekten Verweisen, die ihrerseits maximal je drei Verweise besitzen, zum Einsatz kommt.(**2P**)

**Inode-Struktur:**




d) Wie groß kann eine Datei bei diesem Dateisystem maximal werden, wenn die Blockgröße 512 Bytes beträgt?(**1P**)

$(3 + 3 \cdot 3) \cdot 512 = 6144$ **Bytes**