Hochschule RheinMain
Studiengänge Angewandte Informatik / Informatik - Technische Systeme
Prof. Dr. Robert Kaiser

Probeklausur Betriebssysteme (LV3121)

Wintersemester 2020/2021

Nachname: **Mustermeier**	Vorname: **Theo**
Matrikelnummer: **VOID**	
Datum: 03.03.2021	**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 46 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 beschriebenen DIN A4 Seite.

Punkte:

Aufgabe	Soll-Punkte	Ist-Punkte
1	10	
2	12	
3	4	
4	17	
5	15	
6	8	
7	10	
8	4	
9	12	
Gesamt	92	

Note:

Aufgabe 1: (10 Punkte)

Beantworten Sie bitte folgende Fragen:

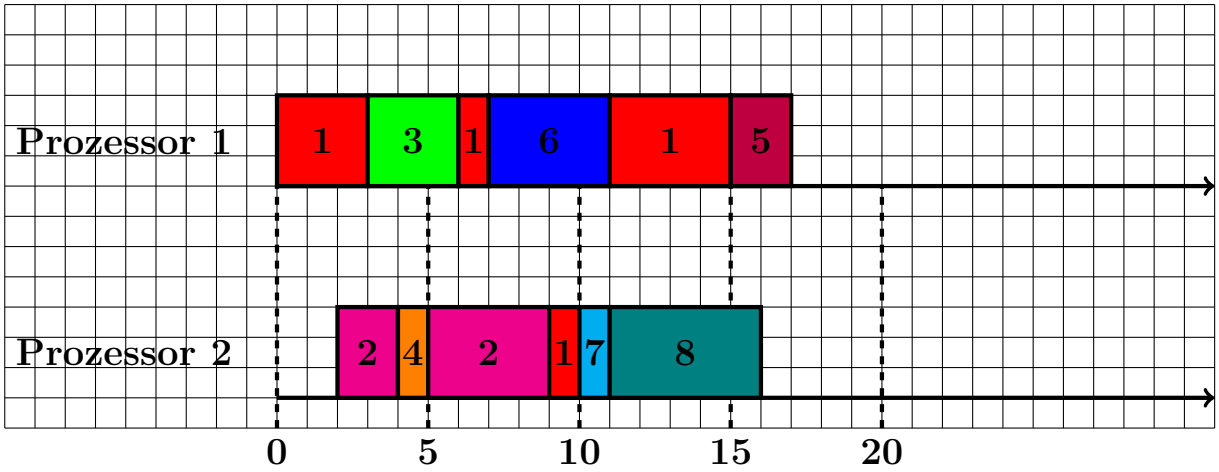
Frage	**Antwort**
Bei einem UNIX-Rechner liefert `getpagesize()` einen Wert von 8192 und `sizeof(void*)` ist 4. Wie viele Einträge müsste eine eindimensionale Seitentabelle dieses Rechners haben?	**524288**
Was kennzeichnet eine „race condition“?	**Ergebnis abhängig von Prozessreihenfolge**
Werden mit `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: (12 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. **(6P)**



b) Bestimmen Sie die mittlere Verweilzeit. (Sie können die freien Spalten in der Tabelle verwenden). **(2P)**

(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. **(4P)**

$Q \to 0, c = 0 \Rightarrow$ **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: (17 Punkte)

Das folgende C-Programm:

```
#include <stdio.h>

void main()
{
    printf("Zeigergroesse: %d\n", sizeof(void*));
    printf("Zahlengroesse: %d\n", sizeof(int));
    printf("Seitengroesse: %d\n", getpagesize());
}
```

liefert auf einem realen Rechner folgende Ausgabe:

```
Zeigergroesse: 8
Zahlengroesse: 4
Seitengroesse: 16384
```

a) Wieviele Adressbits besitzt dieser Prozessor? **(1P)**

64

b) Wie viele Seitentabelleneinträge müsste demnach eine 1-stufige Seitentabelle haben? **(2P)**
Hinweis: Geben Sie bei dieser und den weiteren Fragen größere Zahlenwerte soweit das möglich ist als Zweierpotenzen an (also z.B. 2^{12} statt 4096).

$\frac{2^{64}}{16384} = \frac{2^{64}}{2^{14}} = 2^{64-14} = 2^{50}$

c) Wie viele Petabyte[1] wären das, wenn ein Seitentabelleneintrag 8 Byte groß ist? **(2P)**

$2^{50} \cdot 8 = 2^{50+3} = 2^{53}$ **Byte** $\hat{=} 8$ **PiB**

d) Das Betriebssystem des Rechners läßt maximal $2^8 = 256$ Prozesse zu. Jeder dieser Prozesse hat seinen eigenen Adressraum und damit seine eigene Seitentabelle. Wieviel Speicherplatz müsste damit das System für das gleichzeitige Halten aller Seitentabellen bereitstellen? **(2P)**

[1] 1 PiB = 2^{50} Byte

256·8 **Petabyte** = 2^{11} **PiB** $\hat{=}$2 **Exabyte**

e) Der reale Rechner, auf dem das Programm ausgeführt wurde, hat mit insgesamt 2^{34} Byte = 16 GiB deutlich weniger physikalischen Speicher, als alleine schon zum Halten einer der Seitentabellen erforderlich wäre. Mit welchen „Trick“ schafft es dieses System, den Platzbedarf für die Seitentabellen so weit zu reduzieren, dass es mit dieser Speichermenge auskommt und dass dabei noch effizientes Arbeiten möglich ist? **(1P)**

Mehrstufige Seitentabellen

f) Erklären Sie, warum dieser „Trick“ Speicher spart. **(3P)**

Der größte Teil des virtuellen Adressraums bleibt ohnehin ungenutzt, d.h. es existieren große, zusammenhängende Bereiche im Adressraum, denen kein physikalischer Speicher zugeordnet ist, die also ungültig sind. Dazu genügt bereits auf der ersten Stufe der Seitentabelle ein Eintrag. Weitere Tabelleneinträge auf den niedrigeren Stufen erübrigen sich damit. Vollständige Einträge in der Seitentabelle sind nur für die wenigen Seiten erforderlich, denen physischer Speicher zugeordnet ist.

g) Da jede vom Prozessor referenzierte virtuelle Adresse erst über die Seitentabelle in eine physische Adresse übersetzt werden muss, und da die Seitentabelle im Speicher liegt, erfordert jeder Speicherzugriff des Prozessors mindestens einen weiteren Speicherzugriff auf die Seitentabelle. Die Speicherzugriffszeit würde sich dadurch also mindestens verdoppeln (bei mehrstufigen Seitentabellen sogar vervielfachen). Dieses Problem lösen moderne Rechnerarchitekturen mithilfe eines „TLB“.
Was ist ein TLB? **(1P)**

TLB = Translation Lookaside Buffer

h) Wie löst ein TLB das oben beschriebene Problem? **(3P)**

Der TLB ist ein schneller Assoziativspeicher (ein „Cache“), der häufig benötigte Seitentabelleneinträge praktisch ohne Zeitverzug liefen kann. Wird eine bisher nicht referenzierte Seite adressiert, so wird der zughörige Seitentabelleneintrag in der Seitentabelle lokalisiert („Page Table Walk“) und der Eintrag wird im TLB gespeichert. Wird dieselbe Seite später wieder gebraucht, so wird die Anfrage ohne Verzug aus dem TLB beantwortet.

i) Welches Performance-Problem verursacht der TLB in Verbindung mit der Prozessumschaltung durch das Betriebssystem? **(2P)**

Bei einer Prozessumschaltung muss der Adressraum gewechselt werden. Dadurch werden alle im TLB befindlichen Einträge ungültig, weshalb der TLB gelöscht werden muss. Der neue Prozess findet den TLB somit „kalt“ vor und muss ihn nach und nach mit den nötigen Einträgen befüllen. Er läuft deshalb zunächst langsamer. Bei zu häufigen Prozesswechseln wird der TLB u.U. gar nicht mehr „warm“.

Aufgabe 5: (15 Punkte)

Auf einem Rechner läuft seit mehreren Sekunden ein Compiler-Programm, ohne zu blockieren. Das Betriebssystem verwendet das „Aging"-Seitenersetzungsverfahren. Dabei ergibt sich die unten angegebene Liste eingelagerter Seiten[2]. Die Spalte „Zähler" gibt die den Seiten jeweils zugeordneten Zähler des Aging-Verfahrens als Dualzahl an. Der Aging-Algorithmus wird regelmäßig alle 10 Millisekunden ausgeführt, um die Zählerstände zu aktualisieren

Seitennummer	**Zähler**
9	01001000
1	11111111
32	11101011
51	00111111
8	00011101
33	00011111
34	00100000
11	00110000

a) Welche der Seiten enthält das `.text`-Segment des Compiler-Programmes? **(1P)**

Seite 1

b) Begründen Sie Ihre Antwort **(3P)**

Der Compiler blockiert nicht, führt also seit mehreren Sekunden Code aus seinem .text-Segment aus. In den vergangenen 80 Millisekunden wurde nur Seite 1 ständig referenziert.

c) Es gelte die Festlegung, dass alle Seiten, die innerhalb der letzten 80 Millisekunden referenziert wurden, zum Working Set des laufenden Prozesses gehören. Für wie lange würden demnach die einzelnen Seiten im Working Set des Compilers verbleiben, wenn dieser bis auf Weiteres keine Daten mehr referenzieren würde? **(4P)**
Hinweis: Verwenden Sie die nachfolgende Tabelle.

[2]Es sind nur 8 Seitenrahmen verfügbar.

Seitennummer	**scheidet aus WS aus nach ... ms**
9	**70**
1	**niemals**
32	**80**
51	**60**
8	**50**
33	**50**
34	**60**
11	**60**

d) Es wird nun eine neue Seite mit der Nummer 17 angefordert. Welche Seite wird ausgelagert, um für Seite 17 Platz zu machen (Begründung)? **(2P)**

Seite 8 wird ausgelagert (hat den niedrigsten Zähler)

e) Nach Einlagerung der Seite 17 werden nun während der folgenden 10 Millisekunden alle eingelagerten Seiten referenziert. Anschließend wird der Aging-Algorithmus wieder aufgerufen. Tragen Sie bitte in der folgenden Tabelle die Nummern der nun eingelagerten Seiten und die zugehörigen, aktualisierten Zählerstände ein. **(4P)**

Seitennummer	**Zähler**
9	**10100100**
1	**11111111**
32	**11110101**
51	**10011111**
17	**10000000**
33	**10001111**
34	**10010000**
11	**10011000**

f) Welche Seite würde nun ausgelagert, um für eine neue Seite 42 Platz zu machen? **(1P)**

Seite 17 wird ausgelagert (hat den niedrigsten Zähler)

Aufgabe 6: (8 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. **(2P)**

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? **(2P)** 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.**(2P)**

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

d) Welche Ausgabe erzeugt das folgende Programm? **(2P)**
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:

```
0
Hallo Otto!!
```

Aufgabe 7: (10 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 **(4P)**.
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.**(4P)**
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.**(4P)**

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) **(4P)**.

			La	Ar	RPi		
E	**=**	**(**	**3**	**2**	**8**	**)**	
		|	**1**	**0**	**2**	**|**	**(Gergeleit)**
C	**=**	**|**	**0**	**1**	**1**	**|**	**(Reith)**
		|	**0**	**1**	**1**	**|**	**(Thoss)**

A	**=**	**(**	**2**	**0**	**4**	**)**
		|	**2**	**0**	**2**	**|**
R	**=**	**|**	**1**	**1**	**0**	**|**
		|	**3**	**0**	**3**	**|**

Reihenfolge:

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

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

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

Aufgabe 9: (12 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? **(4P)**

	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**	

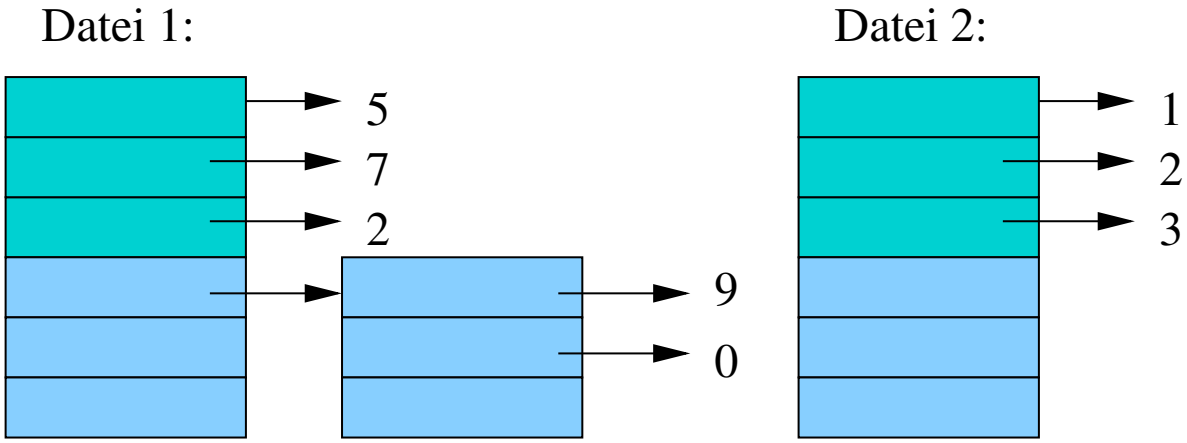
Das Dateisystem ist inkonsistent: Block 2 wird von Dateien 1 und 2 beansprucht.

b) Was ist der wesentliche Nachteil von FAT Dateisystemen bei der Verwaltung großer Platten?**(2P)**

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.**(4P)**

Inode-Struktur:



d) Wie groß kann eine Datei bei diesem Dateisystem maximal werden, wenn die Blockgröße 512 Bytes beträgt?**(2P)**

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