# 1 Prozesse und Scheduling (12 Punkte)

**a) Prozesserzeugung (insgesamt 7 Punkte)** Was geben die folgenden C-Programme aus? Wieso ist die Ausgabe unterschiedlich? *#include* der Systemheader und Fehlerabfragen wurden der Einfachheit halber weggelassen. Gehen Sie von einer fehlerfreien Abarbeitung aller Systemaufrufe aus.

```
1  int pferd;
2
3  void* erzeugePferd(void* param) {
4          pferd++;
5          printf("%d\n",pferd);
6
7          return NULL;
8  }
9
10 int main(void) {
11         pferd = 42;
12         pid_t ret;
13
14         if (fork() == 0) {
15                 erzeugePferd(NULL);
16         } else {
17                 erzeugePferd(NULL);
18         }
19         return 0;
20 }
```

Ausgabe:

```
1  int pferd;
2
3  void* erzeugePferd(void* param) {
4          pferd++;
5          printf("%d\n",pferd);
6          pthread_exit(NULL);
7
8          return NULL;
9  }
10
11 int main(void) {
12         pferd = 42;
13         pthread_t id;
14
15         pthread_create(&id,
16                 NULL,
17                 erzeugePferd,
18                 NULL);
19         erzeugePferd(NULL);
20         pthread_exit(NULL);
21         return 0;
22 }
```

Ausgabe:

**b) Prozesse (insgesamt 2,5 Punkte)**

**1. Prozessstruktur (1 Punkt)** Was geschieht mit den Kindprozessen, wenn der Elternprozess terminiert?

**2. Prozessverwaltung (1,5 Punkte)** Was ist der *copy-on-write* Mechanismus? Wo kommt er im Kontext von Prozessen zum Einsatz?

c) **Scheduling (2,5 Punkte)** Welche der folgenden Aussagen zum Scheduling treffen zu? Kreuzen Sie bei **J** (ja, zutreffend) oder **N** (nein, nicht zutreffend) an!

| J | N | |
|---|---|---|
| | | Bei Round Robin bekommt der nächste Prozess die restliche Zeitscheibe des Vorgängers. |
| | | Shortest Remaining Time First (SRTF) kann Prozesse zum Verhungern bringen. |
| | | Bei Round Robin ist die CPU-Zeit zu gunsten CPU-lastiger Prozesse ungleich verteilt. |
| | | Bei *präemptivem Scheduling* wird davon ausgegangen, dass Prozesse freiwillig die CPU abgeben. |
| | | First-Come First-Served (FCFS) ist optimal geeignet bei einem Mix von CPU- und E/A-lastigenProzessen. |

## 2 Synchronisation und Verklemmungen (13 Punkte)

**a) Erzeuger-/Verbraucher-Problem (7 Punkte)** Die Funktionen producer() und consumer() in folgendem Pseudocode-Abschnitt werden potentiell gleichzeitig ausgeführt, wobei producer() Elemente erzeugt und consumer() diese verbraucht. Synchronisieren Sie die beiden Funktionen mittels *einseitiger Synchronisation* und beachten Sie dabei, dass die Warteschlange zwar beliebig viele Elemente aufnehmen kann, die Operationen enqueue() und dequeue() aber selbst kritisch sind und nicht gleichzeitig ausgeführt werden dürfen. Der consumer() soll nur Elemente aus der Warteschlange entfernen, wenn diese nicht leer ist!

Legen Sie dazu geeignet benannte Semaphore an, initialisieren Sie diese, und setzen Sie an den freien Stellen Semaphor-Operationen (P, V bzw. wait, signal) ein (z.B. `P(MeinSemaphor)`). (P(), V(), produce_element(), consume_element(), enqueue() und dequeue() können als gegeben angesehen werden und müssen *nicht* implementiert werden!)

Namen und Initialwerte der Semaphore: [ ] = [ ]
[ ] = [ ]

```
producer() {
    while (1) {
        Element e = produce_element();



        enqueue(e);









   }
}

consumer() {
    while (1) {









        Element e = dequeue();



        consume_element(e);
    }
}
```

**b) Race-Conditions (1.5 Punkte)** Erläutern Sie, was man unter sogenannten *Race Conditions* versteht und wie man diese verhindern kann.

**c) Verklemmungsbedingungen (3 Punkte)** Nennen und erläutern Sie zwei Bedingungen, die notwendig sind, damit eine Verklemmung auftreten kann.

**d) Verklemmungsvorbeugung (1.5 Punkte)** Nennen Sie eine Möglichkeit zur Verklemmungsvorbeugung (*deadlock prevention*).

# 3 Speicherverwaltung und Virtueller-Speicher (8 Punkte)

**a) Adressabbildung (4 Punkte)** In einem System mit Seitenadressierung (*paging*) befindet sich die Seitenkacheltabelle in unten angegebenem Zustand. Die Adresslänge beträgt 16 Bit, wovon 12 Bit für den Offset innerhalb der Seite verwendet werden (Seitengröße: 4096 Bytes). Bilden Sie die logischen Adressen $6AB1_{16}$ und $F1B7_{16}$ auf ihre physikalischen Adressen ab. (Hinweis: Eine Hexadezimalziffer stellt immer genau vier Bit der Adresse dar.)

| Seitennummer | Startadresse |
|---|---|
| 0 | $F000_{16}$ |
| 1 | $3000_{16}$ |
| 2 | $8000_{16}$ |
| 3 | $1000_{16}$ |
| 4 | $C000_{16}$ |
| 5 | $2000_{16}$ |
| 6 | $4000_{16}$ |
| 7 | $B000_{16}$ |
| ... | ... |
| 15 | $5000_{16}$ |

logische Adresse: $6AB1_{16}$
$\rightarrow$ physikalische Adresse: ☐

logische Adresse: $F1B7_{16}$
$\rightarrow$ physikalische Adresse: ☐

**b) LRU** In einem System mit Seitenadressierung und der Seitenersetzungsstrategie *LRU* (*Least Recently Used*) tätigt ein Prozess Seitenzugriffe entsprechend folgender

*Referenzfolge*: **3, 1, 2, 4, 2, 1, 5, 3, 2**

Das Betriebssystem sieht für diesen Prozess eine feste Anzahl von drei Hauptspeicherkacheln vor. Tragen Sie in die Tabelle unter „Hauptspeicher" jeweils die Nummer der Seite ein, die zum gegebenen Zeitpunkt in der jeweiligen Kachel eingelagert ist. Die Felder unter „Kontrollzustände" *können* Sie zum Notieren des Alters der eingelagerten Seiten zuhilfenehmen. **(4 Punkte)**

| Referenzfolge | | 3 | 1 | 2 | 4 | 2 | 1 | 5 | 3 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| Hauptspeicher | Kachel 1 | 3 | | | | | | | | |
| | Kachel 2 | | | | | | | | | |
| | Kachel 3 | | | | | | | | | |
| Kontrollzustände | Kachel 1 | 0 | | | | | | | | |
| | Kachel 2 | | | | | | | | | |
| | Kachel 3 | | | | | | | | | |

# 4 Ein-/Ausgabe und Dateisysteme (12 Punkte)

**a) I/O-Scheduling (3 Punkte)** Gegeben sei ein Plattenspeicher mit 16 Spuren. Der jeweilige I/O-Scheduler bekommt immer wieder Leseaufträge für eine bestimmte Spur. Die Leseaufträge in $L_1$ sind dem I/O-Scheduler bereits bekannt. Nach einem bearbeiteten Auftrag erhält er die Aufträge in $L_2$. Nach weiteren drei (d.h. nach insgesamt vier) bearbeiteten Aufträgen erhält er die Aufträge in $L_3$. Zu Beginn befindet sich der Schreib-/Lesekopf über Spur 0.

$L_1 = \{4, 7, 11, 3\}, L_2 = \{2, 13, 1\}, L_3 = \{15, 5, 6\}$

Bitte tragen Sie hier die Reihenfolge der gelesenen Spuren für einen I/O-Scheduler, der nach der **First In First Out (FIFO)** Strategie arbeitet, ein:

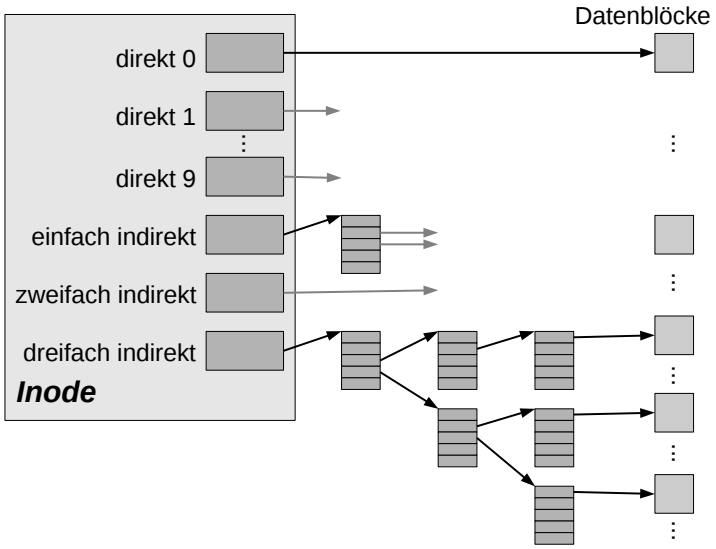
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

**b) Kontinuierliche Speicherung (2 Punkte)** Nennen Sie je einen Vorteil und einen Nachteil der kontinuierlichen Datenspeicherung

**Vorteil:**

**Nachteil:**

**c) Indizierte Speicherung (insgesamt 7 Punkte)** Beantworten Sie die folgenden Fragen zu dem aus der Vorlesung bekannten Schema der indizierten Speicherung von Dateien (siehe nachfolgende Abbildung).




1. **Dateisystem (1 Punkt)** Nennen Sie beispielhaft ein Dateisystem, bei dem Dateien in der dargestellten Weise abgelegt werden.

2. **Maximale Dateigröße (3 Punkte)** Ein hypothetisches Dateisystem verwendet Inodes wie oben dargestellt, nur ***ohne*** Dreifach-Indirektion. Wie lässt sich die maximale Dateigröße für dieses Dateisystem berechnen, wenn die Blockgröße 1024 Bytes beträgt und für die Speicherung eines Blockverweises 4 Bytes benötigt werden. Hinweis: Es ist nicht erforderlich die Zahl auszurechnen. Beschreiben Sie den Rechenweg Schritt für Schritt oder geben Sie eine Formel an.

3. **Vor-/Nachteil (3 Punkte)** Nennen Sie einen Vorteil und einen Nachteil der indizierten Speicherung im Vergleich zur kontinuierlichen Speicherung.

   **Vorteil:**

   **Nachteil:**