Aufgabe 1: Ankreuzfragen (30 Punkte)

1) Einfachauswahlfragen (22 Punkte)

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (☒) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Welche der folgenden Aussagen über Schedulingverfahren ist richtig? 2 Punkte

- ☐ Kooperatives Scheduling ist für Steuerungssysteme mit Echtzeitanforderungen völlig ungeeignet.
- ☐ Preemptives Scheduling ist für Mehrbenutzerbetrieb geeignet.
- ☐ Bei kooperativem Scheduling sind Prozessumschaltungen unmöglich, wenn ein Prozess in einer Endlosschleife läuft, selbst wenn er bei jedem Schleifendurchlauf einen Systemaufruf macht.
- ☐ Bei preemptivem Scheduling sind Prozessumschaltungen unmöglich, wenn ein Prozess in einer Endlosschleife läuft.

b) Gegeben sei folgendes Szenario: zwei Fäden werden auf einem Monoprozessorsystem mit der Strategie "First Come First Served"verwaltet. In jedem Faden wird die Anweisung `i++;` auf die gemeinsame, globale Variable i ausgeführt. Welche der folgenden Aussagen ist richtig: 2 Punkte

- ☐ Die Inkrementoperation muss mit einer CAS-Anweisung nicht-blockierend synchronisiert werden.
- ☐ In einem Monoprozessorsystem ohne Verdrängung ist keinerlei Synchronisation erforderlich.
- ☐ Die Operation i++ ist auf einem Monoprozessorsystem immer atomar.
- ☐ Während der Inkrementoperation müssen Interrupts vorübergehend unterbunden werden.

c) Welche Aussage zum Thema RAID ist richtig? 2 Punkte

- ☐ Bei RAID 5 liegen die Paritätsinformationen auf einer dedizierten Platte.
- ☐ Bei RAID 4 werden alle im Verbund beteiligten Platten gleichmäßig beansprucht.
- ☐ Bei RAID 1 wird beim Lesen ein Geschwindigkeitsvorteil erzielt.
- ☐ RAID 5 ist nur mit drei Platten verwendbar, da die Berechnung der Paritätsinformationen bei mehr Platten nicht möglich ist.

d) Welche Aussage zu nicht-blockierender Synchronisation ist richtig? 2 Punkte

- ☐ Nicht-blockierende Synchronisationsverfahren setzen besondere Unterstützung durch das Betriebssystem voraus.
- ☐ Bei allen nicht-blockierenden Verfahren tritt das ABA-Problem auf.
- ☐ In vielen Fällen sind die Algorithmen bei Verwendung nicht-blockierender Synchronisation einfacher zu beschreiben als bei blockierender Synchronisation.
- ☐ Bei nicht-blockierenden Verfahren können keine Verklemmungen auftreten.

e) Wie werden im Rahmen von Seitenersetzungsverfahren Freiseitenpuffer genutzt? 2 Punkte

- ☐ Wenn zu wenige freie Seitenrahmen im System vorhanden sind (low water mark erreicht) werden prophylaktisch Seiten ausgelagert, selbst wenn kein akuter Bedarf besteht.
- ☐ Wird ein Seitenrahmen in den Freiseitenpuffer eingetragen, wird seine bisherige Zuordnung aus dem entsprechenden Seitendeskriptor entfernt.
- ☐ Auf eine Seite im Freiseitenpuffer darf von dem bisherigen "Besitzer"(Prozess) noch weiterhin zum Lesen und Schreiben zugegriffen werden, bis der Seitenrahmen tatsächlich für eine neue Seite genutzt wird.
- ☐ Auf eine Seite im Freiseitenpuffer darf von dem bisherigen "Besitzer"(Prozess) noch weiterhin, allerdings nur zum Schreiben, zugegriffen werden, bis der Seitenrahmen tatsächlich für eine neue Seite genutzt wird.

f) Welche Aussage zu Terminvorgaben in Echtzeitsystemen ist korrekt? 2 Punkte

- ☐ Beim Überschreiten einer weichen Terminvorgabe wird das Berechnungsergebnis wertlos; die Ausführung wird daher abgebrochen.
- ☐ Das Überschreiten einer harten Terminvorgabe kann zur Katastrophe führen; daher muss eine Ausnahmebehandlung im Anwendungsprogramm erfolgen.
- ☐ Bei festen Terminvorgaben ist eine Terminverletzung tolerierbar, das Ergebnis verliert im Laufe der Zeit aber an Wert.
- ☐ Das Überschreiten einer harten Terminvorgabe ist nicht tolerierbar; daher muss das System in so einem Fall heruntergefahren werden.

g) Nehmen Sie an, der Ihnen bekannte Systemaufruf stat(2) wäre analog zu der Funktion readdir(3) mit folgender Schnittstelle implementiert: `struct stat *stat(const char *path);` Welche Aussage ist richtig? 2 Punkte

- ☐ Der Systemaufruf liefert einen Zeiger zurück, über den die aufrufende Funktion direkt auf eine Datenstruktur zugreifen kann, die die Dateiattribute enthält.
- ☐ Der Aufrufer muss sicherstellen, dass er den zurückgelieferten Speicher mit free(3) wieder freigibt, wenn er die Dateiattribute nicht mehr weiter benötigt.
- ☐ Ein Zugriff über den zurückgelieferten Zeiger liefert völlig zufällige Ergebnisse oder einen Segmentation fault.
- ☐ Solch eine Schnittstelle ist nicht schön, da dadurch die aufrufende Funktion auf internen Speicher des Betriebssystems zugreifen könnte.

h) Virtualisierung kann als Maßnahme gegen Verklemmungen genutzt werden. Warum?

2 Punkte

- ☐ Im Fall einer Verklemmung können zusätzliche virtuelle Betriebsmittel neu erzeugt werden. Diese können dann eingesetzt werden, um die fehlenden physikalischen Betriebsmittel zu ersetzen.
- ☐ Durch Virtualisierung kann man über Abbildungsvorgänge Zyklen, die auf der logischen Ebene vorhanden sind, auf der physikalischen Ebene auflösen.
- ☐ Eine Verklemmungsauflösung ist einfacher, weil virtuelle Betriebsmittel jederzeit ohne Schaden entzogen werden können.
- ☐ Durch Virtualisierung ist ein Entzug von physikalischen Betriebsmitteln möglich, obwohl dies auf der logischen Ebene unmöglich ist.

i) Welche der folgenden Aussagen zum Thema Seitenfehler (page fault) ist richtig?

2 Punkte

- ☐ Ein Seitenfehler zieht eine Ausnahmebehandlung nach sich. Diese wird dadurch ausgelöst, dass die MMU das Signal SIGSEGV an den aktuell laufenden Prozess schickt.
- ☐ Wenn der gleiche Seitenrahmen in zwei verschiedenen Seitendeskriptoren eingetragen wird, löst dies einen Seitenfehler aus (Gefahr von Zugriffskonflikten).
- ☐ Seitenfehler können auch auftreten, obwohl die entsprechende Seite gerade im physikalischen Speicher vorhanden ist.
- ☐ Ein Seitenfehler wird ausgelöst, wenn der Offset in einer logischen Adresse größer als die Länge der Seite ist.

j) Welche der folgenden Aussagen zum Thema Threads ist richtig?

2 Punkte

- ☐ Bei User-Threads ist die Scheduling-Strategie nicht durch das Betriebssystem vorgegeben.
- ☐ Kernel-Threads können Multiprozessoren nicht ausnutzen.
- ☐ Die Umschaltung von User-Threads ist eine privilegierte Operation und muss deshalb im Systemkern erfolgen.
- ☐ Zu jedem Kernel-Thread gehört ein eigener, geschützter Adressraum.

k) Wodurch kann Nebenläufigkeit in einem System entstehen?

2 Punkte

- ☐ durch Seitenflattern
- ☐ durch langfristiges Scheduling
- ☐ durch Compiler-Optimierungen
- ☐ durch Interrupts

2) Mehrfachauswahlfragen (8 Punkte)

Bei den Mehrfachauswahlfragen in dieser Aufgabe sind jeweils m Aussagen angegeben, davon sind n ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie alle richtigen Aussagen an.
Jede korrekte Antwort in einer Teilaufgabe gibt einen Punkt, jede falsche Antwort einen Minuspunkt. Eine Teilaufgabe wird minimal mit 0 Punkten gewertet, d. h. falsche Antworten wirken sich nicht auf andere Teilaufgaben aus.

Wollen Sie eine falsch angekreuzte Antwort korrigieren, streichen Sie bitte das Kreuz mit drei waagrechten Strichen durch (☒).

Lesen Sie die Frage genau, bevor Sie antworten.

a) Ausnahmesituationen bei einer Programmausführung werden in die beiden Kategorien Trap und Interrupt unterteilt. Welche der folgenden Aussagen sind zutreffend?

4 Punkte

- ○ Die CPU sichert bei einem Interrupt einen Teil des Prozessorzustands.
- ○ Der Zeitgeber (Systemuhr) unterbricht die Programmbearbeitung in regelmäßigen Abständen. Die genaue Stelle der Unterbrechungen ist damit vorhersagbar. Somit sind solche Unterbrechungen in die Kategorie Trap einzuordnen.
- ○ Normale Rechenoperationen können zu einem Trap führen.
- ○ Bei einem Trap wird der gerade in Bearbeitung befindliche Befehl immer noch vollständig zu Ende bearbeitet, bevor mit der Trapbehandlung begonnen wird.
- ○ Greift ein Programm auf eine ungültige Adresse zu, dann sendet die MMU einen Interrupt an das Betriebssystem.
- ○ Wenn ein Interrupt einen schwerwiegenden Fehler signalisiert, muss das unterbrochene Programm abgebrochen werden.
- ○ Der Zugriff auf eine virtuelle Adresse kann zu einem Trap führen.
- ○ Der Zugriff auf eine logische Adresse kann zu einem Trap führen.

b) Welche der folgenden Aussagen zu UNIX/Linux-Dateideskriptoren sind korrekt?

4 Punkte

- ○ Ein Dateideskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei benutzen kann.
- ○ Dateideskriptoren sind Zeiger auf Betriebssystem-interne Strukturen, die von den Systemaufrufen ausgewertet werden, um auf Dateien zuzugreifen.
- ○ Beim Öffnen ein und derselben Datei erhält ein Prozess jeweils die gleiche Integerzahl als Dateideskriptor zum Zugriff zurück.
- ○ Ein Dateideskriptor ist eine Integerzahl, die über gemeinsamen Speicher an einen anderen Prozess übergeben werden kann und von letzterem zum Zugriff auf eine geöffnete Datei verwendet werden kann.
- ○ Der Aufruf `newfd = dup(fd)` erzeugt eine Kopie der dem Dateideskriptor `fd` zugrundeliegenden Datei; `newfd` enthält einen Dateideskriptor auf die neu erzeugte Datei.
- ○ Beim Aufruf von `fork()` werden zuvor geöffnete Dateideskriptoren in den Kindprozess vererbt.
- ○ Ein Dateideskriptor ist eine Verwaltungsstruktur, die auf der Festplatte gespeichert ist und Informationen über Größe, Zugriffsrechte, Änderungsdatum usw. einer Datei enthält.
- ○ Ist das Flag `FD_CLOEXEC` eines Dateideskriptors gesetzt, dann wird dieser Dateideskriptor geschlossen, sobald der Prozess eine Funktion der `exec`-Familie aufruft.

Aufgabe 2: rofl - RemOte Folder Lister (60 Punkte)

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Schreiben Sie ein Programm `rofl`, das auf dem TCP/IPv6-Port 2017 (`LISTEN_PORT`) einen Dienst anbietet, über den ein Benutzer Dateien vom Server herunterladen und Verzeichnisse auf dem Server auflisten kann. Die Abarbeitung parallel eintreffender Anfragen soll von einem Arbeiter-Thread-Pool aus initial 3 (`DEFAULT_THREADS`) Arbeiter-Threads übernommen werden; die Threads werden über einen entsprechend synchronisierten Ringpuffer mit Verbindungen versorgt. Der verwendete Ringpuffer soll hierbei maximal 64 (`BB_SIZE`) Einträge speichern können.

Ein Client sendet nach erfolgreicher Verbindung eine Zeile, die das abzurufende Datum enthält. Zur Vereinfachung dürfen Sie davon ausgehen, dass eine Zeile aus maximal 256 (`MAX_LINE`) Zeichen besteht. Nach Einlesen der Zeile sendet der Server die angeforderte Datei oder das angeforderte Verzeichnis an den Client.

Das Programm soll folgendermaßen strukturiert sein:

- Das Hauptprogramm initialisiert zunächst alle benötigten Datenstrukturen, startet die benötigte Anzahl an Arbeiter-Threads und nimmt auf einem Socket Verbindungen an. Eine erfolgreich angenommene Verbindung soll zur weiteren Verarbeitung in den Ringpuffer eingefügt werden. Nutzen Sie hierzu den aus der Übung bekannten Ringbufer `jbuffer` (siehe Teilaufgabe auf Seite 16).
- Funktion `void* thread_worker(void *arg)`: Hauptfunktion der Arbeiter-Threads. Entnimmt in einer Endlosschleife dem Ringpuffer eine Verbindung, und ruft – falls es sich um "normale" Dateideskriptoren handelt (s.u.) – die Funktion `handle_connection` zur weiteren Verarbeitung auf. Achten Sie darauf, dass während der Beantwortung von Clientanfragen aufgetretene Fehler (bspw. nicht vorhandene Dateien oder fehlende Zugriffsrechte) nicht zur Terminierung des Servers führen dürfen.
- Funktion `void handle_connection(int clientSock)`:
  Liest den Pfad (= eine Zeile) vom Client und sendet den Inhalt an den Client.
  - Verzeichnis: Rekursives Auflisten aller Dateien und Unterverzeichnisse via `dump_dir`
  - Reguläre Datei: Dateiinhalt an Client senden
  - Sonst: Anfrage ignorieren

  Zur Vereinfachung dürfen Sie davon ausgehen, dass kein Client Zeilen länger als `MAX_LINE` Zeichen sendet.
- Funktion `void dump_dir(FILE *fh, char *path)`:
  Schreibt den Inhalt des übergebenen Verzeichnisses (und aller Unterverzeichnisse) auf den übergebenen Dateizeiger `fh`.

Mithilfe der Signale `SIGUSR1` (bzw. `SIGUSR2`) soll die Anzahl der laufenden Arbeiter-Threads um einen Thread erhöht respektive verringert werden. Hierzu fügt der jeweilige Signalhandler die Werte `BIRTH` (`POISON`) in den Ringpuffer ein; das Erzeugen (Terminieren) von Threads soll in der Funktion `thread_worker` durchgeführt werden. Zur Vereinfachung dürfen Sie den Fall "kein Arbeiter-Thread läuft mehr" ignorieren.

Achten Sie auf korrekte Synchronisation potenziell nebenläufig ausgeführter Programmteile. Beachten Sie insbesondere die vorgegebene Implementierung von `bbPut` auf Seite 16. Achten Sie ebenfalls auf korrekte und sinnvolle Fehlerbehandlung; Insbesondere dürfen Fehler während der Abhandlung von Clientanfragen nicht zum Abbruch des Programms führen.

```c
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>

// Prototypen des Ringpuffers
BNDBUF* bbCreate(size_t size);
void bbPut(BNDBUF *bb, int value);
int bbGet(BNDBUF *bb);

// Konstanten, Hilfsfunktionen
#define MAX_LINE 256
#define BB_SIZE 64
#define LISTEN_PORT 2017
#define DEFAULT_THREADS 3
#define POISON (-1)
#define BIRTH  (-2)

static void die(char *msg) {
    perror(msg); exit(EXIT_FAILURE);
}

// Funktionsdeklarationen, globale Variablen usw.
```



```c
// Signalhandler

// Funktion main()

    // Threads starten
```

```
    // Signalbehandlung aufsetzen
```

```
    // Auf eingehende Verbindungen warten
```

```
// Ende Funktion main()
```

M:

```
// Funktion dump_dir()
```



```
// Ende Funktion dump_dir()
```

D:

```
// Funktion handle_connection()
```



```
// Ende Funktion handle_connection()
```

H:

```
// Funktion thread_worker()
```

```
// Ende Funktion thread_worker
```

T:

2) Das Modul `jbuffer` (6 Punkte)

Implementieren Sie **die Funktion `bbGet`** des Ihnen aus der Übung bekannten Ringpuffers `jbuffer`. Der Puffer soll für einen Schreiber-Thread und mehrere konkurrierende Leser-Threads ausgelegt sein und FIFO-Eigenschaften aufweisen. Die Konsumenten (= Aufrufer von `bbGet`) sollen untereinander nicht-blockierend koordiniert werden.

Benutzen Sie hierfür die CAS-Funktion

```
bool __sync_bool_compare_and_swap(type *ptr, type oldval, type newval)
```

Achten Sie dabei darauf, dass mehrere Konsumenten gleichzeitig den kritischen Abschnitt durchlaufen können (keine Locks!). Zur Vereinfachung dürfen Sie mögliche ABA-Probleme ignorieren.

```
#include <errno.h>
#include <limits.h>
#include <stdint.h>

#include "jbuffer.h"

// Prototypen der Semaphore
SEM *semCreate(int initVal);
void semDestroy(SEM *sem);
void P(SEM *sem);
void V(SEM *sem);


struct BNDBUF {
    size_t size;          // Größe von data
    volatile size_t rpos; // Leseposition
    volatile size_t wpos; // Schreibposition
    SEM *full;            // Anzahl belegter Einträge
    SEM *free;            // Anzahl freier Einträge
    int data[];
};

BNDBUF* bbCreate(size_t size);

void bbPut(BNDBUF *bb, int value) {
    P(bb->free);

    bb->data[bb->wpos] = value;
    bb->wpos = (bb->wpos + 1) % bb->size;

    V(bb->full);
}

#define CAS __sync_bool_compare_and_swap
```

```
int bbGet(BNDBUF *bb) {
}
```

J:

Aufgabe 3: Dateisystem (15 Punkte)

1) Beschreiben Sie kurz (in Stichworten) die grundsätzliche Funktionsweise eines *Journaling-File-Systems.* (3 Punkte)

2) Gegeben ist die folgende Ausgabe des Kommandos `ls -aoRi /tmp/sp/` (rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `/tmp/sp/` mit Angabe der Inode-Nummer, des Referenzzählers und der Dateigröße) auf einem Linux-System. (12 Punkte)

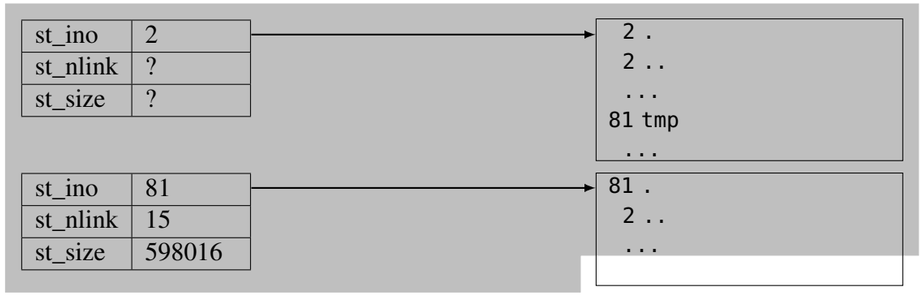
```
chris@legio:~$ ls -aoRi /tmp/sp/
/tmp/sp/:
total 596
39 drwxr-xr-x  3 chris   4096 Jul 26 12:45 ./
81 drwxrwxrwt 15 root  598016 Jul 26 12:34 ../
45 drwxr-xr-x  3 chris   4096 Jul 26 12:43 folder/
67 lrwxrwxrwx  1 chris     12 Jul 26 12:35 important -> folder/file3

/tmp/sp/folder:
total 16
45 drwxr-xr-x 3 chris 4096 Jul 26 12:43 ./
39 drwxr-xr-x 3 chris 4096 Jul 26 12:45 ../
75 drwxr-xr-x 2 chris 4096 Jul 26 12:43 deeper/
49 -rw-r--r-- 2 chris   42 Jul 26 12:38 file1
73 lrwxrwxrwx 1 chris   14 Jul 26 12:43 file2 -> deeper/thefile

/tmp/sp/folder/deeper:
total 16
75 drwxr-xr-x 2 chris 4096 Jul 26 12:43 ./
45 drwxr-xr-x 3 chris 4096 Jul 26 12:43 ../
49 -rw-r--r-- 2 chris   42 Jul 26 12:38 fileX
53 -rw-r--r-- 1 chris  666 Jul 26 12:43 thefile
```

Ergänzen Sie im weißen Bereich die auf der folgenden Seite im grauen Bereich bereits angefangene Skizze der Inodes und Datenblöcke des Linux-Dateisystems um alle entsprechenden Informationen, die aus obiger Ausgabe entnommen werden können.

Inodes



st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

Aufgabe 4: Adressräume (15 Punkte)

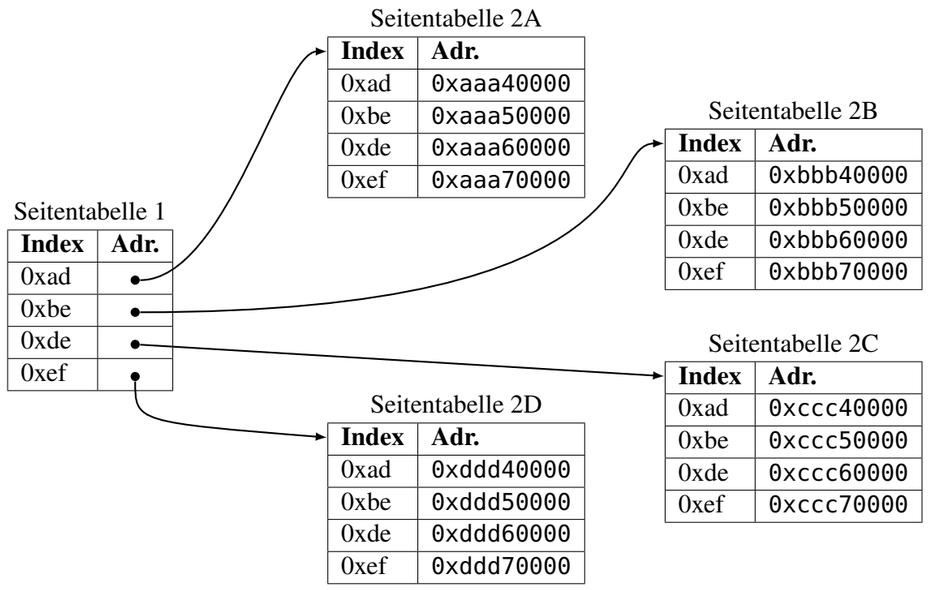
Bei virtuellen Adressräumen können Teile des Speichers auf Hintergrundspeicher ausgelagert sein.

1) Wie und durch wen wird im System bei einem Speicherzugriff erkannt, dass der entsprechende Speicherbereich ausgelagert ist? (2 Punkte)

2) Was läuft im System nach dieser Erkennung ab? Beschreiben Sie die einzelnen Schritte, die im Betriebssystem abgewickelt werden, um den Speicher verfügbar zu machen. Geben Sie an, welche Prozesszustände der auslösende Prozess dabei jeweils einnimmt. (8 Punkte)

3) Was macht das Betriebssystem mit einem Programm, das auf eine ungültige (d.h. nicht abgebildete) virtuelle Adresse (bspw. `NULL`) zugreift? (1 Punkt)

4) Gegeben sei unten dargestellte Hierarchie zweistufiger Seitentabellen. Die Adresslänge des genutzten Systems sei $32\,\mathrm{Bit}$, die Größe einer Seite $64\,\mathrm{kByte}$. Für die Indizierung der zweistufigen Abbildung werden pro Stufe $8\,\mathrm{Bit}$ genutzt. (4 Punkte)



Bestimmen Sie die **reale Adresse** zur logischen Adresse `0xdeadbeef`. Geben Sie hierbei die zur Bestimmung notwendigen Zwischenschritte stichpunktartig an!