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 zum Thema Prozesszustände ist richtig? 2 Punkte

- ☐ Bei Eintreffen eines Interrupts wird der aktuell laufende Prozess für die Dauer der Interrupt-Abarbeitung in den Zustand blockiert überführt.
- ☐ Im Rahmen der mittelfristigen Einplanung kann ein Prozess von Zustand laufend in den Zustand schwebend laufend wechseln.
- ☐ Das Auftreten eines Seitenfehlers kann dazu führen, dass der aktuell laufende Prozess in den Zustand beendet überführt wird.
- ☐ Bei kooperativem Scheduling ist kein direkter Übergang vom Zustand laufend in den Zustand bereit möglich.

b) Welche Aussage zur Seitenumlagerung in virtuellen Adressräumen ist richtig? 2 Punkte

- ☐ Unter Seitenflattern versteht man das ständige Ein- und Auslagern von Speicherseiten, wenn der physisch vorhandene Hauptspeicher nicht ausreicht.
- ☐ Bei der Seitenersetzungsstrategie LRU wird diejenige Seite ausgelagert, auf die in der Vergangenheit am seltensten zugegriffen wurde.
- ☐ Beim Auslagern einer Speicherseite muss der zugehörige Seitendeskriptor angepasst werden. Beim Einlagern ist dies nicht nötig.
- ☐ Die Seitenersetzungsstrategie Second Chance (Clock) ist nur in der Theorie interessant, weil ihre Implementierung komplexe Datenstrukturen erfordert.

c) Beim Blockieren in einem Monitor muss der Monitor freigegeben werden. Warum? 2 Punkte

- ☐ Weil kritische Abschnitte immer nur kurz belegt sein dürfen.
- ☐ Weil sonst die Monitordaten inkonsistent sind.
- ☐ Weil ein anderer Thread die Blockierungsbedingung nur aufheben kann, wenn er den Monitor vorher betreten kann.
- ☐ Weil der Thread sonst aktiv warten würde.

d) Welche Aussage zum Thema „Aktives Warten“ ist richtig? 2 Punkte

- ☐ Aktives Warten auf andere Prozesse bei Scheduling-Strategien ohne Verdrängung auf einem Monoprozessorsystem kann zu Verklemmungen (*Deadlocks*) führen.
- ☐ Bei verdrängenden Scheduling-Strategien verzögert aktives Warten nur den betroffenen Prozess, behindert oder verzögert aber nicht andere.
- ☐ Auf Mehrprozessorsystemen ist aktives Warten unproblematisch und deshalb dem passiven Warten immer vorzuziehen.
- ☐ Aktives Warten vergeudet gegenüber passivem Warten immer CPU-Zeit.

e) Welche Aussage zu Prozessen und Threads ist richtig? 2 Punkte

- ☐ Mittels `fork()` erzeugte Kindprozesse können in einem Multiprozessor-System nur auf dem Prozessor ausgeführt werden, auf dem auch der Elternprozess ausgeführt wird.
- ☐ Der Aufruf von `fork()` gibt im Elternprozess die Prozess-ID des Kindprozesses zurück, im Kindprozess hingegen den Wert 0.
- ☐ Threads, die mittels `pthread_create()` erzeugt wurden, besitzen jeweils einen eigenen Adressraum.
- ☐ Die Veränderung von Variablen und Datenstrukturen in einem mittels `fork()` erzeugten Kindprozess beeinflusst auch die Datenstrukturen im Elternprozess.

f) Welche Aussage zur Verklemmungsverhinderung ist richtig? 2 Punkte

- ☐ Bei einem Zyklus im erweiterten Betriebsmittelgraph liegt ein unsicherer Zustand vor.
- ☐ Bei Verklemmungsverhinderung wird dafür gesorgt, dass eine der vier notwendigen Bedingungen für Verklemmungen nicht auftreten kann.
- ☐ Das System überprüft vor dem Freigeben von Betriebsmitteln, ob ein unsicherer Zustand eintreten würde.
- ☐ Mit Hilfe des Bankiers-Algorithmus wird beim Belegen eines Betriebsmittels geprüft, ob mit erfolgreicher Belegung ein unsicherer Zustand eintreten würde.

g) Welche Aussage zum Thema Synchronisation ist richtig? 2 Punkte

- ☐ Die V-Operation kann auf einem Semaphor nur von dem Thread aufgerufen werden, der zuvor auch die P-Operation aufgerufen hat.
- ☐ Durch den Einsatz von Semaphoren kann ein wechselseitiger Ausschluss erzielt werden.
- ☐ Ein Semaphor kann ausschließlich für mehrseitige Synchronisation (*multilateral synchronisation*) verwendet werden.
- ☐ Einseitige Synchronisation (*unilateral synchronisation*) erfordert immer Betriebssystem-Unterstützung.

h) Was ist ein Stack-Frame?

2 Punkte

- ☐ Der Speicherbereich, in dem der Programmcode einer Funktion abgelegt ist.
- ☐ Ein Fehler, der bei unberechtigten Zugriffen auf den Stack-Speicher entsteht.
- ☐ Ein Bereich des Speichers, in dem u.a. lokale automatic-Variablen einer Funktion abgelegt sind.
- ☐ Ein spezieller Registersatz des Prozessors zur Bearbeitung von Funktionen.

i) Welche der folgenden Informationen wird typischerweise in dem Seitendeskriptor einer Seite eines virtuellen Adressraums gehalten?

2 Punkte

- ☐ Die Zugriffsrechte auf die jeweilige Seite (z. B. lesen, schreiben, ausführen).
- ☐ Die Identifikation des Prozesses, dem die Seite zugeordnet ist.
- ☐ Die Zuordnung zu einem Segment (Text, Daten, ...).
- ☐ Die Position der Seite im virtuellen Adressraum.

j) Welche der folgenden Aussagen zu statischem bzw. dynamischem Binden ist richtig?

2 Punkte

- ☐ Änderungen am Code einer dynamischen Bibliothek (z. B. Bugfixes) erfordern immer das erneute Binden aller Programme, die diese Bibliothek benutzen.
- ☐ Beim statischen Binden werden alle Adressen zum Ladezeitpunkt aufgelöst.
- ☐ Beim dynamischen Binden erfolgt die Adressauflösung beim Laden des Programms oder zur Laufzeit.
- ☐ Statisch gebundene Programme können zum Ladezeitpunkt an beliebige virtuelle Speicheradressen platziert werden.

k) Sie kennen den Translation-Lookaside-Buffer (TLB). Welche Aussage ist richtig?

2 Punkte

- ☐ Verändert sich die Speicherabbildung von logischen auf physikalische Adressen aufgrund einer Adressraumumschaltung, so werden auch die Daten im TLB ungültig.
- ☐ Der TLB verkürzt die Zugriffszeit auf den physikalischen Speicher da ein Teil des möglichen Speichers in einem schnellen Pufferspeicher vorgehalten wird.
- ☐ Der TLB puffert Daten bei der Ein-/Ausgabebehandlung und beschleunigt diese damit.
- ☐ Wird eine Speicherabbildung im TLB nicht gefunden, wird der auf den Speicher zugreifende Prozess mit einer Schutzraumverletzung (*Segmentation Fault*) abgebrochen.

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) Gegeben sei folgendes Programm:

4 Punkte

```
 1 #include <stdlib.h>
 2 #include <stdio.h>
 3
 4 extern long a;
 5
 6 #define GOLD 1.618
 7
 8 int main(int argc, char **argv) {
 9     static int x;
10     int y = GOLD;
11
12     a = x * y;
13
14     fprintf(stderr, "%f\n", y);
15
16     return EXIT_SUCCESS;
17 }
```

Welche der folgenden Aussagen bzgl. dieses Programms sind korrekt?

- ○ Beim Überschreiben der Variable `a` in Zeile 12 tritt ein Fehler auf, weil externe Variablen nicht überschrieben werden dürfen.
- ○ Der Aufruf von `fprintf()` in Zeile 14 gibt den Wert `1.618` auf `stderr` aus.
- ○ `argv` ist ein Array aus Zeigern, die jeweils auf ein Array aus `char`s zeigen.
- ○ Die globale Variable `GOLD` enthält den Wert `1.618`.
- ○ An Index `0` des `argv`-Arrays liegt ein Zeiger auf den Programmnamen oder -pfad.
- ○ Beim Linken des Programms kann ein Fehler auftreten.
- ○ Der Inhalt der Datei *stdio.h* wird vor dem Übersetzen an die Stelle der entsprechenden `include`-Anweisung einkopiert.
- ○ Die Variable `x` ist uninitialisiert und wird daher auf einen zufälligen Wert gesetzt.

b) Man unterscheidet zwei Kategorien von Ausnahmesituationen bei einer Programmausführung: Traps und Interrupts. Welche der folgenden Aussagen sind zutreffend?

4 Punkte

- ○ Die CPU sichert bei einem Interrupt einen Teil des aktuellen Prozessorzustands.
- ○ Ein Systemaufruf im Anwendungsprogramm ist der Kategorie Interrupt zuzuordnen.
- ○ Bei einem Trap wird der gerade in Bearbeitung befindliche Maschinenbefehl immer noch vollständig zu Ende bearbeitet, bevor mit der Trapbehandlung begonnen wird.
- ○ Da Traps immer synchron auftreten, kann es im Rahmen ihrer Behandlung nicht zu Wettlaufsituationen mit dem unterbrochenen Programm kommen.
- ○ Wenn ein Interrupt einen Fehler signalisiert, wird das unterbrochene Programm im Rahmen der Interruptbearbeitung immer abgebrochen.
- ○ Ein durch einen Interrupt unterbrochenes Programm darf je nach der Interruptursache entweder abgebrochen oder fortgesetzt werden.
- ○ Ein Programm darf im Rahmen einer Trapbehandlung abgebrochen werden.
- ○ Das Dereferenzieren eines Zeigers kann zu einem Trap führen.

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

Aufgabe 2: Cell Broadcast Server (60 Punkte)

Implementieren Sie einen einfachen TCP-basierten IPv6 Cell Broadcast Server. Dieser nimmt auf Port 8888 Nachrichten von Admins entgegen, die im folgenden an alle Clients, die sich auf Port 2222 verbinden, verteilt wird. Jede Nachricht wird so lange verteilt, bis durch einen Admin eine neue gültige Nachricht gesetzt wird. Initial wird die leere Nachricht mit ID `0` verteilt. Zu jedem Zeitpunkt kann sich nur ein Admin verbinden, erst wenn dessen Behandlung abgeschlossen ist, wird die nächste Admin-Verbindung angenommen. Das Senden der aktuellen Nachricht an die Clients geschieht in `NUMTHR` parallelen Arbeiterthreads.

Kommunikation und Synchronisation mit den Arbeiterthreads: Für Client-Anfragen werden die Verbindungs-Filedeskriptoren, und zur Signalisierung von Ereignissen Event-Pillen, über einen zentralen Bounded-Buffer (Größe `BBSIZE`) an die Arbeiterthreads übermittelt. Wird von einem Admin eine neue gültige Nachricht gelesen, wird die von den Arbeiterthreads zu verteilende Nachricht `msg` geändert. Damit nicht bei jedem lesenden Zugriff der Arbeiterthreads auf `msg` synchronisiert werden muss, werden bei Vorliegen einer neuen Nachricht alle Arbeiterthreads durch das Ereignis `NEWMSG_PILL` angehalten. Sobald alle Threads gestoppt sind (Semaphore `sold`) wird die Nachricht ersetzt. Danach können die Arbeiterthreads weiterlaufen (Semaphore `snew`).

Erhält der Prozess nach der Initialisierung (beim Warten) das Signal `SIGINT`, räumt er alle allokierten Ressourcen und Threads auf und beendet sich geregelt mit `EXIT_SUCCESS`.

Implementieren Sie folgende Funktionen:

`int main(int argc, char *argv[])` Initialisiert den globalen Zustand des Prozesses, startet mit `spawn()` die Threads zum Akzeptieren von Client- und Admin-Verbindungen (`accept_clients()` bzw. `accept_admins()`) sowie die `NUMTHR` Client-Arbeiterthreads (`serve_clients()`). Wartet dann **passiv** auf das Auftreten von `SIGINT` woraufhin sich der Prozess mit all seinen Threads geregelt beendet. (Beim Warten sind alle Signale bis auf `SIGINT` blockiert.) Um eventuell in `accept()`-Aufrufen blockierte Threads zu deblockieren, wird die vorgegebene Funktion `dummy_connect()` verwendet. Den Client-Arbeiterthreads wird durch das Ereignis `SIGINT_PILL` die Aufforderung zum Beenden übermittelt.

`void spawn(pthread_t *tid, void *(*routine)(void *))` Hilfsfunktion zum Erstellen eines Threads. Beendet im Fehlerfall den Prozess.

`int listen_or_die(uint16_t port)` Erstellt ein Verbingungssocket und gibt es zurück.

`void *accept_admins(void *arg)` Öffnet das Socket zur Admin-Verbindungsannahme mit `listen_or_die()`. Nimmt Verbindungen an und liest pro Verbindung max. eine Nachricht mit `receive_msg()` ein. Ist die Nachricht gültig, wird die globale `msg`, wie oben beschrieben synchronisiert, geändert.

`void handle_sigint(int signum)` `SIGINT`-Behandlungsfunktion.

`int receive_msg(FILE *rx, msg_t *m)` Liest direkt die neue Nachricht in den Ausgabeparameter ein. Eine gültige Nachricht ist eine Zeile (max. `LMAX` Nutzzeichen und zusätzliches `\n`) die aus zwei durch einen Tabulator (`\t`) getrennten Teilen besteht: Der dezimal formatierten positiven `int` ID (`parse_positive_int()`) und dem Nachrichteninhalt. Gehen Sie davon aus, dass der Nachrichteninhalt selbst keine Tabulatoren enthält. Gibt im Fehlerfall `-1` zurück, sonst `0`.

`void *accept_clients(void *)` Öffnet das Socket zur Client-Verbindungsannahme mit `listen_or_die()`, akzeptiert Verbindungen und fügt diese in den Bounded-Buffer ein.

`void *serve_clients(void *arg)` Entnimmt Client-Sockets bzw. zu bearbeitende Events aus dem zentralen Bounded-Buffer. Sendet an Clients direkt die aktuelle Nachricht (durch `\t` getrennte ID und Inhalt) und liest dabei aus `msg` wie oben beschrieben ohne Synchronisation. Vom Client an den Server gesendete Daten werden nicht verarbeitet.

Bei Fehlern bei der Nutzeranfragenbearbeitung soll die Anfrage ohne Meldung ignoriert werden.

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

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

#include "bbuffer.h" // Vorgegeben, siehe Manpage.

// Schnittstelle des vorgegebenen Semaphor-Moduls:
typedef struct SEM SEM;
SEM *semCreate(int initVal);
void semDestroy(SEM *sem);
void P(SEM *sem); // Decrement.
void V(SEM *sem); // Increment.

static const size_t   NUMTHR = 8;
static const size_t   BBSIZE = 8;
static const uint16_t ADMIN_PORT = 2222, CLIENT_PORT = 8888;
static const int      NEWMSG_PILL = -1, SIGINT_PILL = -2;
#define               LMAX 1395

// Vorgegeben Strukturen:
typedef struct msg {
  int id;
  char cont[LMAX + 1];
} msg_t;

// Vorgegebene Funktionen:
static void die(const char *msg) {
  perror(msg);
  exit(EXIT_FAILURE);
}

static int parse_positive_int(char *str) {
    /* Gibt einen positiven int zurück, oder -1 im Fehlerfall. */
}

static void dummy_connect(uint16_t port) {
    /* Sorgt dafür, dass ein auf port blockierter accept() Aufruf zurückkehrt
     * indem eine Verbindung aufgebaut, und sofort wieder geschlossen wird. */
}
```



```c
// Vorwährtsdeklarationen von Funktionen sind nicht nötig.
static BNDBUF *bb;
static SEM *sold;
static SEM *snew;
static msg_t msg;


int main(int argc, char *argv[]) {
  if (argc > 1) {
    fprintf(stderr, "Usage: %s\n", argv[0]);
    return EXIT_FAILURE;
  }

  // Signalbehandlung und globalen Zustand initialisieren:


  // Nächste Seite: "Fäden starten"
```

```
  // Fäden starten:
```

```
  // Passives Warten auf SIGINT:
```

```
  // Geregeltes Aufräumen:
```

```
  // Nächste Seite: "Auf Arbeiterthreads warten und
  //   Ressourcen freigeben"
```



```
  // Auf Arbeiterthreads warten und Ressourcen freigeben:
```

```
  return EXIT_SUCCESS;
}
```

M:

```
static int listen_or_die(uint16_t port) {
```

```
}
```

L:

```
static void spawn(pthread_t *tid, void *(*routine)(void *)) {

}
```

P:

```
static void *accept_admins(void *arg) {

}
```

A:


```
static void handle_sigint(int signum) {

}
```

H:

```
static int receive_msg(FILE *rx, msg_t *m) {
  // Einlesen der Nachricht:

  // Verarbeiten der Nachricht:

}
```

R:

```
static void *accept_clients(void *arg) {
```

```
}
```

C:

```
static void *serve_clients(void *arg) {
```

```
}
```

S:

Aufgabe 3: Dateisystem (13 Punkte)

1) Beschreiben Sie kurz (in Stichworten) die grundsätzliche Funktionsweise eines *RAID-0 Festplattenverbund* sowie einen Vor- und Nachteil eines derartigen Betriebsmodus. (3 Punkte)

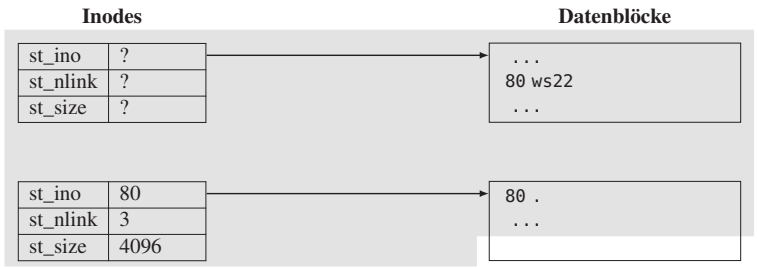
2) Gegeben ist die folgende Ausgabe des Kommandos `ls -aoRi ws22/sp2` auf einem Linux-System; eine rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `ws22/sp2` mit Angabe der Inode-Nummer (erste Spalte), des Referenzzählers (dritte Spalte) und der Dateigröße (fünfte Spalte). (10 Punkte)

```
i4sp@i4sp:~$ ls -aoRi ws22/sp2
ws22/sp2:
total 12
 81 drwxr-xr-x 3 i4sp 4096 Feb  6 13:05 .
 80 drwxr-xr-x 3 i4sp 4096 Feb  6 13:05 ..
 88 drwxr-xr-x 3 i4sp 4096 Feb  6 13:05 folder
172 lrwxrwxrwx 1 i4sp   12 Feb  6 13:05 important -> folder/file1

ws22/sp2/folder:
total 12
 88 drwxr-xr-x 3 i4sp 4096 Feb  6 13:05 .
 81 drwxr-xr-x 3 i4sp 4096 Feb  6 13:05 ..
128 drwxr-xr-x 2 i4sp 4096 Feb  6 13:05 deeper
213 lrwxrwxrwx 1 i4sp   14 Feb  6 13:05 file1 -> deeper/thefile
136 -rw-r--r-- 2 i4sp  666 Feb  6 13:05 file2

ws22/sp2/folder/deeper:
total 8
128 drwxr-xr-x 2 i4sp 4096 Feb  6 13:05 .
 88 drwxr-xr-x 3 i4sp 4096 Feb  6 13:05 ..
136 -rw-r--r-- 2 i4sp  666 Feb  6 13:05 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.



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	

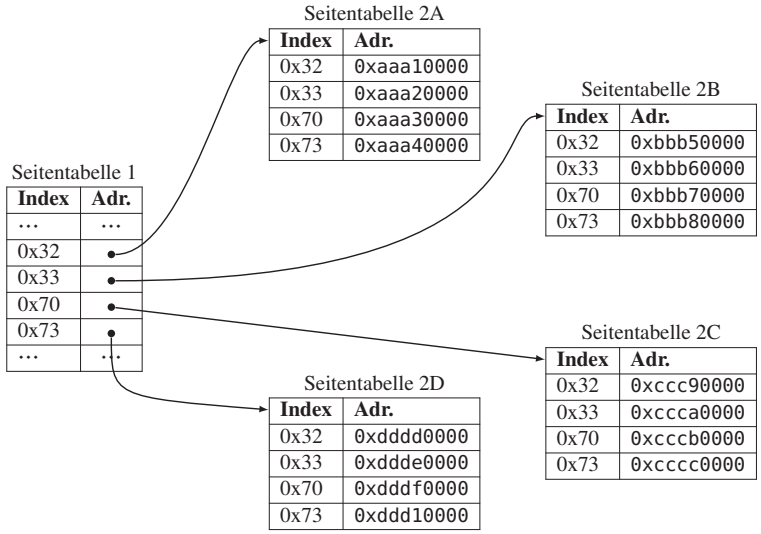
Aufgabe 4: Adressräume (17 Punkte)

1) Bei virtuellen Adressräumen können Teile des Speichers auf Hintergrundspeicher ausgelagert sein. (10 Punkte)

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

b) 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)

2) 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. (7 Punkte)



a) Bestimmen Sie die **reale Adresse** zur logischen Adresse `0x73703233`. Geben Sie hierbei die zur Bestimmung notwendigen Zwischenschritte stichpunktartig an! (5 Punkte)

b) Nehmen Sie an, dass alle gültigen Seitentabelleneinträge oben abgebildet sind. Was würde in diesem Fall mit einem Prozess passieren, der schreibend auf die Adresse `0xFFFFFFFF` zugreift und wieso? (2 Punkte)