Aufgabe 1: Ankreuzfragen (22 Punkte)

1) Einfachauswahlfragen (18 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) Gegeben seien die folgenden Präprozessor-Makros: 2 Punkte

```
#define SUB(a, b) a - b
#define MUL(a, b) a * b
```

Was ist das Ergebnis des folgenden Ausdrucks? `4 * MUL ( SUB(3,5), 2)`

- ☐ 16
- ☐ 2
- ☐ -2
- ☐ -16

b) Was versteht man unter einer Unterbrechung bei der Ausführung von Instruktionen durch einen Prozessor? 2 Punkte

- ☐ Eine Signalleitung teilt dem Prozessor mit, dass er den aktuellen Prozess anhalten und auf das Ende der Unterbrechung warten soll.
- ☐ Mit einer Signalleitung wird dem Prozessor eine Unterbrechung angezeigt. Der Prozessor sichert den aktuellen Zustand bestimmter Register, insbesondere des Programmzählers, und springt eine vordefinierte Behandlungsfunktion an.
- ☐ Der Prozessor wird veranlasst eine Unterbrechungsbehandlung durchzuführen. Der gerade laufende Prozess kann die Unterbrechungsbehandlung ignorieren.
- ☐ Durch eine Signalleitung wird der Prozessor veranlasst, die gerade bearbeitete Maschineninstruktion zu unterbrechen und in den Benutzermodus umzuschalten.

c) Welche Aussage über fork() ist richtig? 2 Punkte

- ☐ Der an fork() übergebene Funktionszeiger wird durch einen neuen Thread im aktuellen Prozess ausgeführt.
- ☐ Der Kind-Prozess bekommt im Erfolgsfall die Prozess-ID des Elternprozesses zurückgegeben.
- ☐ Dem Eltern-Prozess wird im Erfolgsfall die Prozess-ID des Kindes zurückgeliefert.
- ☐ Der Aufruf von fork() ersetzt das im aktuellen Prozess laufende Programm durch das als Parameter angegebene Programm.

d) In einem UNIX-Dateisystem gibt es symbolische Verweise (symbolic links) und feste Verweise (hard links). Welche der folgenden Aussagen ist richtig?

2 Punkte

- ☐ Ein „hard link“ kann nicht auf Dateien, sondern nur auf Verzeichnisse verweisen.
- ☐ Auf ein Verzeichnis verweist immer genau ein symbolischer Verweis.
- ☐ Die Anzahl der „hard links“, die auf ein Verzeichnis verweisen, hängt von der Anzahl seiner Unterverzeichnisse ab.
- ☐ Ein symbolischer Verweis kann nicht auf Dateien in anderen Dateisystemen verweisen.

e) Welche Aussage über Variablen in C-Programmen ist richtig?

2 Punkte

- ☐ Es ist nicht möglich, Zeiger als Parameter an Funktionen zu übergeben.
- ☐ Lokale automatic-Variablen, die auf dem Stack angelegt werden, werden immer mit dem Wert 0 initialisiert.
- ☐ Eine Funktion, die mit dem Schlüsselwort static definiert wird, kann nur innerhalb des Moduls aufgerufen werden, in dem sie definiert wurde, nicht jedoch aus einem anderen Modul heraus.
- ☐ Wird dem Parameter einer Funktion innerhalb der Funktion ein neuer Wert zugewiesen, so ändert sich auch der Wert der Variablen, welche in der aufrufenden Funktion als Parameter angegeben wurde.

f) Was versteht man unter virtuellem Speicher?

2 Punkte

- ☐ Unter einem virtuellen Speicher versteht man einen physikalischen Adressraum, dessen Adressen durch eine MMU vor dem Zugriff auf logische Adressen umgesetzt werden.
- ☐ Speicher, der einem Prozess durch entsprechende Hardware (MMU) und durch Ein- und Auslagern von Speicherbereichen vorgespiegelt wird, aber möglicherweise größer als der verfügbare physikalische Hauptspeicher ist.
- ☐ Speicher, der nur im Betriebssystem sichtbar ist, jedoch nicht für einen Anwendungsprozess.
- ☐ Virtueller Speicher kann dynamisch zur Laufzeit von einem Programm erzeugt werden.

g) Welche Aussage zum Thema Prozesszuständen ist richtig?

2 Punkte

- ☐ Terminiert ein laufender Prozess, so wird er vom Betriebssystem in den Zustand blockiert überführt.
- ☐ Ein Prozess kann nicht direkt vom Zustand laufend in den Zustand bereit überführt werden.
- ☐ Pro Prozessor kann es stets nur einen laufenden, jedoch mehr als einen bereiten Prozess geben.
- ☐ Ein Prozess kann mit Hilfe von Spezialbefehlen selbst vom Zustand bereit in den Zustand laufend wechseln.

h) Was muss geschehen, damit ein Linux-Prozess sich im Zustand Zombie befindet?

2 Punkte

- ☐ Der Elternprozess muss terminieren, aber der init Prozess darf den Kindprozess noch nicht adoptieren.
- ☐ Die Terminierung des Prozesses muss fehlschlagen.
- ☐ Der Prozess muss terminieren, aber der Exitstatus darf noch nicht abgefragt worden sein.
- ☐ Der Prozess muss blockieren und diesen Zustand nie wieder verlassen.

i) Welche Aussage zu Semaphoren ist richtig?

2 Punkte

- ☐ Die V-Operation eines Semaphors erhöht den Wert des Semaphors um 1 und deblockiert gegebenenfalls wartende Prozesse.
- ☐ Ein Semaphor kann nur zur Signalisierung von Ereignissen, nicht jedoch zum Erreichen gegenseitigen Ausschlusses verwendet werden.
- ☐ Die V-Operation eines Semaphors kann ausschließlich von einem Thread aufgerufen werden, der zuvor mindestens eine P-Operation auf dem selben Semaphor aufgerufen hat.
- ☐ Die P-Operation eines Semaphors erhöht den Wert des Semaphors um 1 und deblockiert gegebenenfalls wartende Prozesse.

2) Mehrfachauswahlfragen (4 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) Welche der folgenden Aussagen zum Thema Threads sind richtig?

4 Punkte

- ○ Kern-Threads können Multiprozessoren nicht ausnutzen.
- ○ Kern-Threads blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ○ Die Umschaltung von User-Threads ist sehr effizient.
- ○ Zur Umschaltung von Kernel-Threads ist kein Adressraumwechsel erforderlich.
- ○ Zu jedem Kern-Thread gehört ein eigener Adressraum.
- ○ Bei User-Threads ist die Schedulingstrategie keine Funktion des Betriebssystemkerns.
- ○ Die Umschaltung von Kern-Threads muss immer im Systemkern erfolgen.
- ○ Bei Kern-Threads ist die Schedulingstrategie meist durch das Betriebssystem vorgegeben.

Aufgabe 2: berch (45 Punkte)

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

Implementieren sie das Kommandozeilenprogramm **BE**dienungs-**R**ichtlinien-**CH**ecker (kurz berch). Dieses überprüft für jede Zeile auf der Standardeingabe, ob mit einem gewissen Alter ein bestimmer Artikel gekauft werden darf. Dazu wird das Programm mit dem Pfad zu einem Verzeichnis als einzigem Parameter gestartet, welches in Dateien Informationen zu verfügbaren Artikeln bereit hält. Für jede Anfrage wird in einer eigenen Zeile eine Zahl ausgegeben, die angibt, ob der Artikel erworben werden darf (`1`), nicht erworben werden darf (`0`), oder ob der Artikel nicht gefunden wurde (`-1`). Ein Beispielaufruf könnte also wie folgt aussehen:

```
anfragen.txt:
Kellerbier 14
Weiße_Limonade 8
Unbekannter_Artikel 18
```

```
bedienung@i4keller: ./berch /usr/share/articles < anfragen.txt
0
1
-1
```

Implementieren Sie dazu folgende Funktionen:

`int main(int argc, char **argv)`
Überprüft zunächst die Eingabe, durchsucht den übergebenen Artikel-Ordner mithilfe der Funktion `load()` und liest dann mittels `process()` Anfragen von der Standardeingabe ein. Sobald alle Anfragen bearbeitet wurden, werden die in der globalen Variable `requests` hinterlegten Ergebnisse zeilweise ausgegeben. Zum Schluss wird der genutzte Speicher freigegeben.

`void load(char *folder)`
Durchläuft das Verzeichnis `folder` rekursiv und übergibt den Pfad jeder regulären Datei an die bereits vorgegebene Funktion `load_article()`. Symbolischen Verknüpfungen soll **nicht** gefolgt werden. Sollte `load_article` **`NULL`** zurückgeben und `errno` auf `ENOMEM` setzen, so wird das Programm mit einem Fehlercode beendet. Alle anderen Fehler sollen lediglich mit Ausgabe einer Fehlermeldung quittiert werden. Alle erfolgreich gelesenen Artikel werden im globalen Array `articles` gesammelt.

`void process(FILE *stream)`
Liest bis zum Abbruch durch EOF zeilenweise Anfragen von `stream`. Zeilen, die länger als die Maximallänge `LMAX` sind, sollen zum Abbruch des Programmes führen. Jede Zeile enthält zuerst den Artikelnamen und dann das Alter der Person. Beide Werte sind mit Leerzeichen voneinander getrennt. Sie dürfen davon ausgehen, dass Leerzeichen ausschließlich zum Trennen der beiden Werte verwendet werden. Gültige Anfragen werden anschließend an `check()` übergeben. Für die Konvertierung des Alters existiert die Funktion `parse_int()`.

`void check(const char *article_name, int age)`
Erweitert das globale Array `requests` um ein zusätzliches Element und befüllt dieses mit `article_name` und `age`. In der globalen Struktur `articles` wird nach einem Eintrag mit demselben Namen gesucht und gegebenenfalls eine Altersprüfung durchgeführt. Das Ergebnis der Prüfung wird im Strukturmitglied `result` des neuen Elements vermerkt.

Hinweis: Nicht-fatale Fehler, wie unzureichende Dateizugriffsrechte, sollen mit einer Fehlermeldung quittiert werden.

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

```c
#include <errno.h>
#include <dirent.h>
#include <stdlib.h>
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#define LMAX 1024 // Maximale Zeilenlänge
#define NMAX 256 // Maximale Länge eines Artikelnamens

// Vorgegebene Strukturen und Funktionen:
struct article {
    char name[NMAX + 1]; // Artikelname
    unsigned min_age; // Mindestalter
};

struct request {
    char article_name[NMAX + 1];
    unsigned age; // Kundenalter
    int result;
};

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

static void exit_usage(char *cmd) {
    fprintf(stderr, "%s <article_dir>\n", cmd);
    exit(EXIT_FAILURE);
}

// Wandelt str in vorzeichenlose Ganzzahl um und speichert diese in n.
// @return 0 im Erfolgsfall und -1, falls ein Fehler auftritt.
static int parse_int(const char *str, unsigned *n);

// Lädt Artikelinformationen aus Datei. Es ist Aufgabe des Aufrufers,
// den zurückgegebenen Speicher mittels free(3) freizugeben.
// @return den geladenen Artikel im Erfolgsfall und NULL im Fehlerfall.
static struct article *load_article(char *filename);

// Globale Variablen
static struct request **requests = NULL;
static size_t num_requests = 0;
static struct article **articles = NULL;
static size_t num_articles = 0;
```

```
int main(int argc, char **argv) {
    // Argumente prüfen und verarbeiten

    // Ergebnisse ausgeben

    // Ressourcen freigeben

    return EXIT_SUCCESS;
}
```

M:

```
static void load(char *path) {
    // Öffnen des übergebenen Pfades


    // Iterieren über das Verzeichnis


        // Auslesen des Verzeichniseintrages
```

```
        // Behandlung des Verzeichniseintrages

        // Behandlung von regulären Dateien

        // Behandlung von Ordnern

    // Schließen des Verzeichnisses

}
```

L:

```
static void process(FILE *stream) {
    // Daten von der Standardeingabe einlesen

        // Eingabe überprüfen

        // Aufteilen und Prüfen der Eingabe
```

```
}
```

P:

```
static void check(const char *article_name, int age) {
    // Requests erweitern und befüllen

    // Eintragen des übergebenen Artikels

    // Ergebnis der Anfrage ermitteln

}
```

D:

Aufgabe 3: Adressräume (11 Punkte)

1) Beschreiben Sie die drei Begriffe *realer*, *logischer* und *virtueller* Adressraum. (7 Punkte)

2) Nennen und beschreiben Sie eine Möglichkeit, wie zur Laufzeit eine Abbildung auf den realen Adressraum vorgenommen werden kann. (4 Punkte)

Aufgabe 4: Dateisystem (12 Punkte)

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.

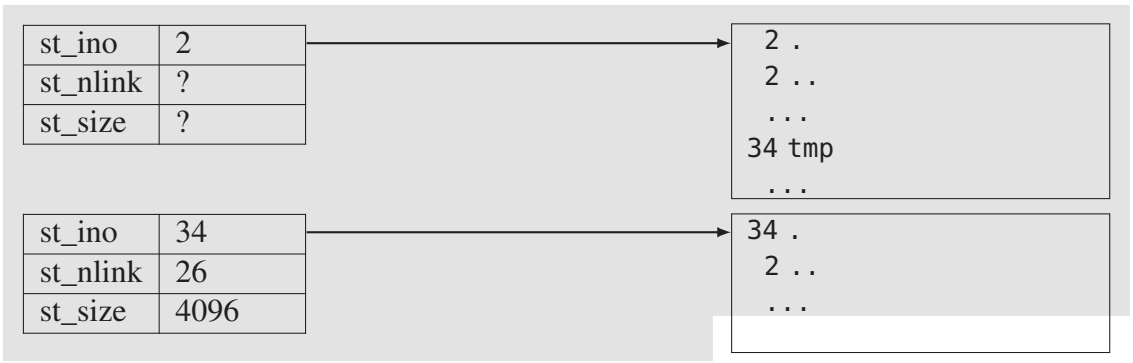
```
i4sp@korhal:~$ ls -aoRi /tmp/sp
/tmp/sp:
total 14
91 drwxr-xr-x  4 i4sp 4096 Jul 17 11:50 .
34 drwxrwxrwt 26 root  620 Jul 17 11:42 ..
81 drwxr-xr-x  2 i4sp 4096 Jul 17 11:25 tief
92 drwxr-xr-x  2 i4sp 4096 Jul 17 11:32 wichtig

/tmp/sp/tief:
total 3
81 drwxr-xr-x 2 i4sp 4096 Jul 17 11:25 .
91 drwxr-xr-x 4 i4sp 4096 Jul 17 11:50 ..
77 lrwxrwxrwx 1 i4sp    9 Jul 17 14:11 leere -> /dev/null
82 -rw-r--r-- 2 i4sp 5432 Jul 17 11:23 proviant
78 lrwxrwxrwx 1 i4sp    2 Jul 17 11:19 tiefer -> ./

/tmp/sp/wichtig:
total 2
92 drwxr-xr-x 2 i4sp 4096 Jul 17 11:32 .
91 drwxr-xr-x 4 i4sp 4096 Jul 17 11:50 ..
79 -rw-r--r-- 1 i4sp 1337 Jul 17 11:31 kaffee
82 -rw-r--r-- 2 i4sp 5432 Jul 17 11:23 kuchen
```

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	

st_ino	
st_nlink	
st_size	