

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

2 Punkte

- Durch eine Signalleitung wird der Prozessor veranlasst, die gerade bearbeitete Maschineninstruktion zu unterbrechen und in den Benutzermodus umzuschalten.
- 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.
- Eine Signalleitung teilt dem Prozessor mit, dass er den aktuellen Prozess anhalten und auf das Ende der Unterbrechung warten soll.

b) 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.
- 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.
- Auf ein Verzeichnis verweist immer genau ein symbolischer Verweis.

c) Was versteht man unter virtuellem Speicher?

2 Punkte

- Speicher, der nur im Betriebssystem sichtbar ist, jedoch nicht für einen Anwendungsprozess.
- Virtueller Speicher kann dynamisch zur Laufzeit von einem Programm erzeugt werden.
- 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.

d) Welche Aussage über `fork()` ist richtig?

2 Punkte

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

e) Welche Aussage zum Thema Prozesszustände ist richtig?

2 Punkte

- Terminiert ein laufender Prozess, so wird er vom Betriebssystem in den Zustand blockiert überführt.
- Ein Prozess kann mit Hilfe von Spezialbefehlen selbst vom Zustand bereit in den Zustand laufend wechseln.
- 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.

f) Welche Aussage zu Semaphoren ist richtig?

2 Punkte

- Die V-Operation (*up*) eines Semaphors erhöht den Wert des Semaphors um 1 und deblockiert gegebenenfalls wartende Prozesse.
- Ist oder war der Wert des Semaphors bei der P-Operation (*down*) 0, werden wartende Prozesse aufgeweckt.
- Die V-Operation (*up*) eines Semaphors kann ausschließlich von einem Thread aufgerufen werden, der zuvor mindestens eine P-Operation (*down*) auf dem selben Semaphor aufgerufen hat.
- Ein Semaphor kann nur zur Signalisierung von Ereignissen, nicht jedoch zum Erreichen gegenseitigen Ausschlusses verwendet werden.

g) Wie funktioniert Adressraumschutz durch Eingrenzung?

2 Punkte

- Jedes Programm bekommt zur Ladezeit mehrere Wertepaare aus Basis- und Längenregistern zugeordnet, die die Größe aller Segmente des darin laufenden Prozesses festlegen.
- Begrenzungsregister legen einen Adressbereich im physikalischen Adressraum fest, auf den alle Speicherzugriffe beschränkt werden.
- Der Lader positioniert Programme immer so im Arbeitsspeicher, dass unerlaubte Adressen mit nicht-existierenden physikalischen Speicherbereichen zusammenfallen.
- Die MMU kennt die Länge eines Segments und verhindert Speicherzugriffe darüber hinaus.

h) Welche der folgenden Aussagen trifft auf das Programmfragment zu?

2 Punkte

```
int f1 (const int *y) {
    static int b;
    char *d = malloc(2407);
    int (*e)(const int *) = f1;
    b += *y;
    y++;
    return b;
}
```

- Die Variable b liegt im Stacksegment.
- Die Anweisung y++ führt zu einem Laufzeitfehler, da y konstant ist.
- Die Speicherstelle, auf die d zeigt, verliert beim Rücksprung aus der Funktion f1() ihre Gültigkeit.
- Die Variable e liegt im Stacksegment und zeigt auf eine Stelle im Textsegment.

i) User-Level- und Kernel-Level-Threads unterscheiden sich in verschiedenen Eigenschaften. Welche Aussage ist richtig?

2 Punkte

- Kernel-Level-Threads werden effizienter umgeschaltet als User-Level-Threads.
- Nur durch User-Level-Threads können mehrere Prozessoren genutzt werden.
- Bei User-Level-Threads können anwendungsabhängig Schedulingstrategien eingesetzt werden.
- Blockiert ein Systemaufruf, so wird automatisch der nächste User-Level-Thread ausgeführt.

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) Man unterscheidet zwischen Traps und Interrupts. Welche der folgenden Aussagen ist richtig?

4 Punkte

- Ein Programm darf im Rahmen einer Trapbehandlung abgebrochen werden.
- Der Zugriff auf eine virtuelle Adresse kann zu einem Trap führen.
- Rechenoperationen können zu einem Interrupt führen.
- Ganzzahl-Rechenoperationen können nicht zu einem Trap führen.
- Die CPU sichert bei einem Interrupt einen Teil des Prozessorzustands.
- Weil das Betriebssystem nicht vorhersagen kann, wann ein Prozess einen Systemaufruf tätigt, sind Systemaufrufe in die Kategorie Interrupt einzuordnen.
- Der Zugriff auf eine logische Adresse kann zu einem Trap führen.
- Der Zugriff auf eine physikalische Adresse kann keinen Trap auslösen.

Aufgabe 2: griff (45 Punkte)

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

Implementieren sie das Kommandozeilenprogramm **Grundlegender Rekursiver Intelligenter Fragmentfinder** (kurz **griff**) welches eine vereinfachte Variation von *grep* darstellt. Analog zum Original soll **griff** als ersten Parameter die gesuchte Zeichenkette sowie mindestens ein Verzeichnis übergeben bekommen.

Damit sucht *griff* rekursiv in allen gegebenen Verzeichnissen nach regulären Dateien und durchsucht diese jeweils zeilenweise nach der gegebenen Teilzeichenkette (*strstr(3p)*). Dabei wird als Optimierung für jede reguläre Datei ein eigener Thread gestartet. Enthält eine Datei Zeilen mit mehr als *LMAX* Nutzzeichen, soll das durchsuchen der Datei abgebrochen werden. Gefundene Übereinstimmungen werden zusammen mit dem Pfad der Datei ausgegeben.

```
boulder@halle: ./griff Route dir other/dir
dir/fileA: Gelbe Route:
dir/fileA: Rote Route:
other/dir/fileC: Routenplanung
```

Für die Speicherung von `pthread_t`-Werten wird eine einfache Listenimplementierung bereitgestellt (siehe Funktionsdeklarationen auf der nächsten Seite).

Implementieren Sie dazu folgende Funktionen:**int main(int argc, char **argv)**

Überprüft zunächst die Eingabe und initialisiert benötigte Datenstrukturen. Anschließend werden mithilfe der Funktion `crawl()` alle übergebenen Verzeichnisse rekursiv nach Dateien durchsucht, deren Inhalt die als ersten Parameter übergebene Teilzeichenkette enthält. Zum Schluss werden die von `crawl()` erzeugten Threads wieder eingesammelt, deren Treffer ausgegeben und der genutzte Speicher freigegeben.

void crawl(char *path)

Durchläuft das Verzeichnis `path` rekursiv (bei rekursiven Aufrufen soll Symbolic Links nicht gefolgt werden) und erzeugt für jede reguläre Datei einen Thread mit `processFile()` und dem Dateipfad als Argument. Einträge die keine regulären Dateien oder Verzeichnisse sind, oder mit einem Punkt beginnen, sollen still ignoriert werden. Die Thread-IDs sollen für die spätere Verarbeitung der gefundenen Zeilen in der bereitgestellten thread-sicheren Liste gesammelt werden. Falls `path` auf kein Verzeichnis zeigt, soll die Funktion eine Fehlermeldung ausgeben und zurückkehren. Alle anderen Fehler sollen das Programm mit einem Fehlercode beenden.

void *processFile(void *arg)

Öffnet die hinter dem als `void`-Pointer übergebenen Pfad liegende Datei lesend und liest zeilenweise den Dateinhalt ein. Enthält eine Zeile mehr als *LMAX* Nutzzeichen (exkl. Zeilenumbruch, `\n`), soll das durchsuchen dieser Datei mit einer Warnung auf `stderr` abgebrochen werden. Gefundene Übereinstimmungen mit der übergebenen Teilzeichenkette werden mithilfe vom vorgegebenen `prepareLine()` zwischengespeichert und als `NULL`-terminiertes Array für die abschließende Ausgabe in `main()` zurückgegeben.

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

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

```
#include <errno.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <pthread.h>

#define LMAX 100 // Maximale Zeilenlänge

// Vorgegebene Strukturen und Funktionen

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

static void exitUsage(char *cmd) {
    fprintf(stderr, "%s_<substring>_<directory_...>\n", cmd);
    exit(EXIT_FAILURE);
}

// Initialisiert eine leere, verlinkte Liste
// @return 0 im Erfolgsfall, -1 mit passend gesetzter errno im Fehlerfall
int listCreate(void);

// Fügt den übergebenen Wert in die verlinkte Liste ein (thread-safe!).
// @return 0 im Erfolgsfall, -1 mit passend gesetzter errno im Fehlerfall
int listInsert(pthread_t);

// Entfernt ältestes Element aus der verlinkten Liste und schreibt dieses an
// die Speicheradresse in @value (thread-safe!).
// @return 1 im Erfolgsfall, 0 falls die Liste leer bzw. uninitialized ist
int listRemove(pthread_t *value);

// Deallokiert den der Liste zugehörigen Speicher vollständig.
void listDestroy(void);

// Allokiert Heapspeicher für gegebene Kombination aus Dateipfad und Zeile
// und befüllt diesen mit passender Formatierung für eine spätere Ausgabe.
// @return Pointer auf gültigen Speicher, NULL mit gesetzter errno im Fehlerfall
static char *prepareLine(const char *file_path, const char *line);

// Weitere Vorausdeklarationen
static void crawl(char *path);
static void *processFile(void *args);
```

```
static char *substring = NULL;
```

```
int main(int argc, char **argv) {
```

```
    // Argumente prüfen und verarbeiten
```

```
    // Datenstrukturen initialisieren
```

```
    // Über alle übergebenen Pfade iterieren
```

```
    // Auf Beendigung aller Threads warten und  
    // jeweils Rückgaben verarbeiten + Ressourcenfreigabe
```

```
..... // Ressourcen freigeben und Ausgabe sicherstellen  
.....  
.....  
.....  
.....
```

```
..... return EXIT_SUCCESS;  
..... }  
.....
```

M:

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

```
..... // Iterieren über das Verzeichnis  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

// Eintragtyp auslesen

// Datei / Verzeichnis verarbeiten

// Schließen des Verzeichnisses

}

C:

```
static void *processFile(void *args) {
```

```
    // Zeilen aus der geöffneten Datei einlesen
```

```
    // Zeile überprüfen
```

```
    // Nach Teilzeichenkette 'substring' suchen  
    // und für Rückgabe zwischenspeichern
```


2) Was passiert in Hardware und im Betriebssystem bei einem Zugriff auf eine aktuell ausgelagerte Speicherseite? (3 Punkte)

Aufgabe 5: Echtzeitbetrieb (9 Punkte)

1) Erklären Sie die Besonderheiten des *Echtzeitbetriebs* in einigen Stichpunkten. (3 Punkte)

2) Erklären Sie die Verhaltensunterschiede eines Echtzeitbetriebssystems, das eine Terminverletzung bei festen (*firm*) bzw. harten (*hard*) Terminvorgaben erkannt hat. Gehen Sie hierbei darauf ein, welcher Teil des Systems die Behandlung der Terminüberschreitung vornimmt. (6 Punkte)

