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) Wie funktioniert Adressraumschutz durch Eingrenzung? 2 Punkte

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

b) 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.

c) Welcher UNIX-Systemaufruf wird bei der Verwendung von Sockets auf keinen Fall gebraucht? 2 Punkte

- ☐ shutdown()
- ☐ wait()
- ☐ close()
- ☐ accept()

d) In einem UNIX-UFS-Dateisystem gibt es symbolische Namen/Verweise (*symbolic links*) und feste Links (*hard links*) auf Dateien. Welche Aussage ist richtig? 2 Punkte

- ☐ Ein *hard link* kann nur auf Verzeichnisse verweisen, nicht jedoch auf Dateien.
- ☐ Wird der letzte *symbolic link* auf eine Datei gelöscht, so wird auch die Datei selbst gelöscht.
- ☐ Für jede reguläre Datei existiert mindestens ein *hard link* im selben Dateisystem.
- ☐ Ein *symbolic link* kann nicht auf Dateien anderer Dateisysteme verweisen.

e) Welche Aussage über einem UNIX-Systemaufruf trifft zu? 2 Punkte

- ☐ Nach dem Umschalten in den privilegierten Prozessormodus wird die vom Benutzer übergebene Folge an Maschineninstruktionen ausgeführt.
- ☐ Lediglich der Programmzähler muss gesichert werden.
- ☐ Mit Hilfe von Systemaufrufen kann ein Benutzerprogramm privilegierte Operationen durch das Betriebssystem ausführen lassen, die es im normalen Ablauf nicht selbst ausführen dürfte.
- ☐ Durch einen Systemaufruf wird die Kontrolle vom System an den unpriviligierten Prozess abgegeben.

f) Welche der folgenden Aussagen zum Thema Prozesszustände ist korrekt? 2 Punkte

- ☐ Der Planer (*scheduler*) kann einen Prozess in den Zustand „blockiert“ überführen, indem er einen anderen Prozess einlastet.
- ☐ In einem Vierkernsystem können sich maximal vier Prozesse gleichzeitig im Zustand „bereit“ befinden.
- ☐ Blockierte Prozesse können nicht in den Zustand „bereit“ überführt werden, solange nicht alle benötigten Betriebsmittel zur Verfügung stehen.
- ☐ In einem Vierkernsystem gibt es stets genau vier laufende Prozesse.

g) User-Level- und Kernel-Level-Threads unterscheiden sich in verschiedenen Eigenschaften. Welche Aussage ist richtig? 2 Punkte

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

h) Welche Aussage zum Thema Betriebsarten ist richtig?

2 Punkte

- ☐ Echtzeitsysteme findet man hauptsächlich auf großen Serversystemen, die eine enorme Menge an Anfragen zu bearbeiten haben.
- ☐ Beim Stapelbetrieb können keine globalen Variablen existieren, weil alle Daten im Stapel-Segment (Stack) abgelegt sind.
- ☐ Mehrprogrammbetrieb ermöglicht die simultane Ausführung mehrerer Programme innerhalb desselben Prozesses.
- ☐ Mehrzugangsbetrieb ist nur in Verbindung mit CPU- und Speicherschutz sinnvoll realisierbar.

i) Wie viele Prozesse existieren nach erfolgreicher Ausführung der folgenden Programmsequenz?

2 Punkte

```
if (fork())
    fork();
exit(EXIT_SUCCESS);
```

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 3

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

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

Aufgabe 2: `wosch` (45 Punkte)

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

Um den immer häufiger werdenden Anglizismen in der deutschen Sprache entgegenzuwirken, ist es Ihre Aufgabe, den **Wo**rtver**sch**önerer, kurz `wosch`, zu implementieren. Dieser ersetzt mithilfe eines Glossars Begriffe durch eine entsprechende Verschönerung.

Ihr Programm nimmt als ersten Parameter den Pfad zum Glossar entgegen, welches die Verschönerungen von Begriffen enthält. Alle weiteren Parameter werden als zu bearbeitende Pfade interpretiert. Falls keine weiteren Parameter neben dem Pfad zum Glossar vorhanden sind, soll die Standardeingabe verarbeitet werden. Die Ausgabe der verschönerten Texte geschieht auf der Standardausgabe.

Bespiel: `./wosch glossar.wosch text.txt`

glossar.wosch:
```
Hello Hallo
World Welt
Nice Nette
```

text.txt:
```
Hello World,
wie geht es?
Nice Aufgabe, oder?
```

Ausgabe:
```
Hallo Welt,
wie geht es?
Nette Aufgabe, oder?
```

Implementieren Sie zur Umsetzung folgende Funktionen:

`int main(int argc, char *argv[])`
Liest mit der Funktion `glossar` das Glossar ein. Danach werden die Verschönerungen mithilfe von `lese_ein` und `bearbeite` vorgenommen. Abschließend werden noch belegte Ressourcen freigegeben.

`struct glossar *glossar(const char *p)`
Speichert alle vorkommenden Verschönerungen aus der Datei hinter dem Pfad `p` in einem **`struct`** `glossar` und gibt einen Zeiger darauf zurück. Die Einträge im **`struct`** `glossar` sollen mithilfe der gegebenen Vergleichsfunktion `vergleiche` sortiert werden. Eine valide Zeile besteht aus einem zu ersetzenden Begriff (siehe Hinweis 1) und der zugehörigen, beliebig langen Verschönerung. Beide Bestandteile sind mit Leerzeichen und/oder Tabulatoren voneinander getrennt. Jede Zeile der Glossardatei beinhaltet maximal 200 Zeichen, ebenso kommt kein zu ersetzender Begriff doppelt vor. Nicht valide Zeilen werden ignoriert.

`void bearbeite(struct glossar *g, const char *p)`
Bearbeitet einen übergebenen Pfad `p` mit dem übergebenen **`struct`** `glossar *g`: Bei einer regulären Datei wird `lese_ein` aufgerufen, allerdings nur, sofern diese auf `.txt` endet. Verzeichnisse werden rekursiv abgearbeitet. Alle anderen Dateien werden ignoriert, insbesondere auch symbolische Verknüpfungen.

`void lese_ein(struct glossar *g, FILE *in, FILE *out)`
Liest den Inhalt vom übergebenen `FILE *in` ein, und gibt diesen auf `FILE *out` aus. Dabei werden Verschönerungen von Begriffen mithilfe von `verschoenere` realisiert.

`const char *verschoenere(struct glossar *g, const char *str)`
Überprüft, ob der Begriff `c` im übergebenen `glossar` vorhanden ist. Wenn ja, wird dieser zurückgegeben, anderenfalls der Begriff selbst.

Hinweise:

1. Begriffe bestehen stets aus nicht mehr als 50 Zeichen, welche nicht Leerzeichen, Tabulatoren oder Zeilenumbrüche sind.
2. Sollte der Bearbeitungsprozess bei einer Datei fehlschlagen, so soll dies zwar mit einer Fehlermeldung gemeldet werden, jedoch nicht die Bearbeitung anderer Dateien weiter beeinflussen.

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

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <dirent.h>
#include <errno.h>
#include <fnmatch.h>
#include <libgen.h>
#include <ctype.h>

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

static void usage(char *name) {
  fprintf(stderr, "Usage: %s <glossar> (<pfad...>)\n", name);
  exit(EXIT_FAILURE);
}

// Struktur zum Speichern des Glossars
struct schoen {
  char *from, *to;
};
struct glossar {
  size_t eintraege;
  struct schoen *s;
};

static int vergleiche(const void *a, const void *b) {
  struct schoen *ua = (struct schoen *) a;
  struct schoen *ub = (struct schoen *) b;
  return strcmp(ua->from, ub->from);
}

static struct glossar *glossar(const char *p);
static void bearbeite(struct glossar *g, const char *p);
static void lese_ein(struct glossar *g, FILE *in, FILE *out);
static const char *verschoenere(struct glossar *g, const char *c);
```

```
int main(int argc, char *argv[]) {
}
```

M:

```
static struct glossar *glossar(const char *p) {
  // Glossar anlegen
  // Datei öffnen
```



```
  // Einlesen der Datei
    // Teilen der gelesenen Zeile
    // Nachallozieren von Speicher
    // Eintragen in das Glossar
  // Sortierung
}
```

W:

```
static void bearbeite(struct glossar *g, const char *p) {
```



```
}
```

B:

```
static void lese_ein(struct glossar *g, FILE *in, FILE *out) {
```

```
}
```

L:

```
static const char *verschoenere(struct glossar *g, const char *c) {
}
```

V:

Aufgabe 3: Adressräume (14 Punkte)

Gegeben sei das nachfolgende Programm. Skizzieren Sie den Aufbau des logischen Adressraums eines Prozesses, der dieses Programm ausführt.

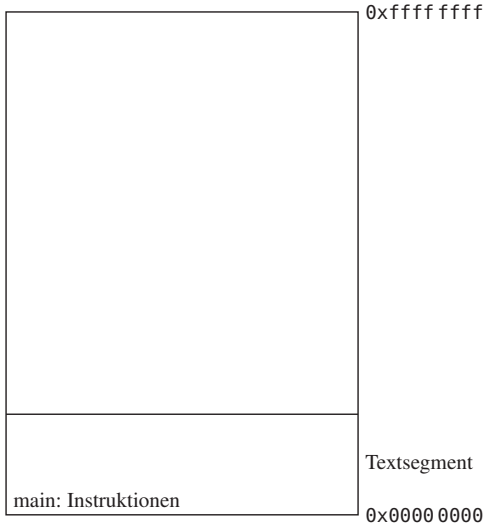
1) Tragen Sie die Segmente und deren Namen (analog zum schon vorgegebenen Textsegment) in unten stehende Zeichnung ein. Unterscheiden Sie hierbei die Bereiche zur Speicherung von initialisierten und nicht initialisierten Variablen. Zeichnen Sie für jede gültige Variable ein, wo diese ungefähr im logischen Adressraum zu finden sein wird und welchen Wert sie enthält, wenn die Programmausführung bis unmittelbar vor der **`return`**-Anweisung der `main`-Funktion fortgeschritten ist. Illustrieren Sie im Falle von Zeigervariablen mittels Pfeil, auf welches Datum die Variable jeweils zeigt.

Vermerken Sie zudem, in welche Richtung Segmente variabler Größe wachsen. Gehen Sie hierbei von einem x86-System aus. (8 Punkte)

```
static char *s = "Hello World\n";

static int * weird(int x) {
    int *arr = malloc(x);
    if (!arr) {exit(1);}
    return arr;
}

void main(void) {
    int g = 8;
    static int h;
    int *(*func)(int) = weird;
    int *w = func(g);
    return;
}
```



2) Beschreiben Sie was geschieht, wenn ein Prozess auf eine ausgelagerte Seite zugreift. (3 Punkte)

3) Nehmen Sie an, dass das folgende Programmstück in einem UNIX-System abläuft: (3 Punkte)

```
// ...
int *p = malloc(sizeof(int));
*p = -1;
// ...
```

a) Welcher Fehler kann dabei auftreten? (1 Punkt)

b) Der Fehler wird von einer Hardwarekomponente zuerst entdeckt - welche Komponente ist das? (1 Punkt)

c) Was wird das Betriebssystem mit dem Prozess machen, der den Fehler hervorruft? (1 Punkt)

Aufgabe 4: 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)