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) Welche der folgenden Aussagen zu statischem bzw. dynamischem Binden ist richtig? 2 Punkte

- ☐ Bei dynamischem Binden müssen zum Übersetzungszeitpunkt alle Adressbezüge vollständig aufgelöst werden.
- ☐ Bei dynamischem Binden können Fehlerkorrekturen in Bibliotheken leichter übernommen werden, da nur die Bibliothek selbst neu erzeugt werden muss. Programme, die die Bibliothek verwenden, müssen nicht neu kompiliert und gebunden werden.
- ☐ Bei statischem Binden werden durch den Compiler alle Adressbezüge vollständig aufgelöst.
- ☐ Beim statischen Binden werden alle Adressen zum Ladezeitpunkt aufgelöst

b) Ein Betriebssystem setzt logische Adressräume auf der Basis von Segmentierung ein. Welche Aussage ist richtig? 2 Punkte

- ☐ Segmente können verschiedene Längen haben. Die Einhaltung der Längenbegrenzung wird vom C-Compiler überprüft.
- ☐ Die Segmentierung schränkt den logischen Adressraum derart ein, dass nur auf gültige Speicheradressen erfolgreich zugegriffen werden kann.
- ☐ Adressraumschutz durch Segmentierung erfordert keine Hardwareunterstützung.
- ☐ Die Bindung von Programm- an Arbeitsspeicheradressen erfolgt zur Ladezeit des Programms.

c) Wie funktioniert Adressraumschutz durch Eingrenzung? 2 Punkte

- ☐ Beim Laden eines Programms prüft das Betriebssystem, ob alle Speicherzugriffe im gültigen Bereich liegen.
- ☐ Begrenzungsregister legen einen Adressbereich im logischen Adressraum fest, auf den alle Speicherzugriffe beschränkt werden.
- ☐ Zur Laufzeit eines Programms ist durch Begrenzungsregister festgelegt, auf welchen Bereich des physikalischen Speichers das Programm zugreifen darf.
- ☐ Der Compiler grenzt beim Erzeugen des Codes die Adresszugriffe auf einen bestimmten Bereich ein.

d) Welche der folgenden Aussagen zum Thema Adressräume ist richtig?

2 Punkte

- ☐ Die maximale Größe des virtuellen Adressraums kann unabhängig von der verwendeten Hardware frei gewählt werden.
- ☐ Virtuelle Adressräume sind Voraussetzung für die Realisierung logischer Adressräume.
- ☐ Der physikalische Adressraum ist durch die gegebene Hardwarekonfiguration definiert.
- ☐ Der virtuelle Adressraum eines Prozesses kann nie größer sein als der physikalisch vorhandene Arbeitsspeicher.

e) Welche der folgenden Aussagen zum Thema Prozesszustände ist richtig?

2 Punkte

- ☐ Das Auftreten eines Seitenfehlers kann dazu führen, dass der aktuell laufende Prozess in den Zustand beendet überführt wird.
- ☐ 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.
- ☐ In einem Achtkernsystem gibt es stets genau acht laufende Benutzerprozesse.

f) Gegeben seien die folgenden Präprozessor-Makros:

2 Punkte

```
#define ADD(x, y) x + y
#define SUB(x, y) x - y
```

Was ist das Ergebnis des folgenden Ausdrucks? `SUB(3, ADD(1, 4)) * 2`

- ☐ -7
- ☐ 10
- ☐ -4
- ☐ 12

g) Welche der folgenden Aussagen zum Thema Synchronisation sind richtig?

2 Punkte

- ☐ Die V-Operation eines Semaphors kann ausschließlich von einem Thread aufgerufen werden, der zuvor mindestens eine P-Operation auf dem selben Semaphor aufgerufen hat.
- ☐ Semaphore lassen sich nicht für einseitige Synchronisation einsetzen.
- ☐ Durch den Einsatz von Semaphoren kann ein wechselseitiger Ausschluss erzielt werden.
- ☐ Ein Semaphor kann ausschließlich für mehrseitige Synchronisation verwendet werden.

h) Wie wird in einem UNIX-Dateisystem (z.B. EXT2 oder Sun UFS) das Attribut des Eigentümers einer Datei realisiert?

2 Punkte

- ☐ Der Benutzername des Eigentümers wird als String im Dateiinhalt abgespeichert.
- ☐ Eine Integerzahl repräsentiert die *user ID* (UID) des Eigentümers im Inode der Datei.
- ☐ Die *group ID* des Eigentümers wird im Dateiinhalt gespeichert.
- ☐ Die *user ID* des Eigentümers steht im Verzeichniseintrag der Datei.

i) Welche der folgenden Aussagen zum Thema Betriebsarten ist richtig?

2 Punkte

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

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) Gegeben sei folgendes Programmfragment:

4 Punkte

```
static int a = 81034;
int main(int argc, char *argv[]) {
  static int b;
  int c;
  int (*d)(int, char **) = main;
  long *e = malloc(800);
  argc++;
  // ...
}
```

Welche der folgenden Aussagen zu den Variablen im Programm sind richtig?

- ○ `c` verliert beim Rücksprung aus `main` seine Gültigkeit.
- ○ Die in `e` nach der Zuweisung enthaltene Speicheradresse kann problemlos verwendet werden.
- ○ `c` ist uninitialisiert und enthält einen undefinierten Wert.
- ○ Die Anweisung `argc++` ändert den Wert von `argc` und beeinflusst somit den Aufrufer.
- ○ `b` ist mit 0 initialisiert und liegt im BSS-Segment.
- ○ Die Adresse `e` liegt auf dem Stack.
- ○ `e` zeigt auf ein Array, in dem Platz für 800 Ganzzahlen vom Typ **`long`** ist.
- ○ Das Ergebnis des Aufrufs der Funktion `main` wird in `d` gespeichert.

Aufgabe 2: `pdu` (45 Punkte)

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

Schreiben Sie ein mehrfädiges Programm `pdu` (**p**arrallel **d**isk **u**sage), das den Gesamtspeicherplatzverbrauch pro Benutzer in frei wählbaren Verzeichnissen ermittelt und eine Übersicht erstellt. Der Aufruf an `pdu` und die resultierende Ausgabe sehen aus wie folgt:

```
luis@cipterm0: ./pdu /home/luis /usr/local
luis (21617): 5030322 Bytes
thomas (21601): 173034 Bytes
```

Dabei soll für jeden als Argument an `pdu` übergebenen Verzeichnispfad ein eigener Thread gestartet werden, der den Speicherverbrauch aller darin enthaltenen **regulären Dateien** ermittelt und dem Besitzer zuordnet. Benutzer werden durch die `uid`, einer Ganzzahl vom Typ `uid_t`, identifiziert. Gehen Sie davon aus, dass die `uid` vom Betriebssystem fortlaufend für jeden neu angelegten Benutzer vergeben wird. Dateien mit einer `uid` < 1000 sollen von `pdu` ignoriert werden. Die Anzahl der verfügbaren Benutzer ist nicht bekannt. Die `uid` ist Teil der `struct stat`.

Der Hauptthread startet die Arbeiterthreads und wartet passiv bis diese terminieren. Die Arbeiter aggregieren den Speicherverbrauch in einer globalen Variable vom Typ `struct agg`. Dabei zeigt das Strukturmitglied `usage` auf ein Array mit aggregiertem Speicherverbrauch je Benutzer, während `count` die Anzahl der Einträge im Array und damit die höchste gefundene `uid` speichert. Nach dem Aufsammeln der Arbeiter wird vom Hauptthread eine Rangfolge erstellt, in der die Benutzer **absteigend** nach ihrem Speicherverbrauch sortiert werden. Dafür sollen die in `struct agg` gesammelten Informationen in je ein `struct summary` pro Benutzer kopiert werden. Zur Sortierung der `struct summary` soll auf eine eigene Vergleichsfunktion zurückgegriffen werden.

Zum Abschluss wird die Zusammenfassung des Speicherverbrauchs für alle Nutzer mit einem Verbrauch > 0 Byte auf `stdout` ausgegeben. Zur Ausgabe der Ergebnisse soll die **vorgegebene** Funktion `print_user_usage()` genutzt werden. Diese ermittelt zu jeder `uid` den passenden Benutzernamen und tätigt die Ausgabe auf `stdout`. Sie kann nicht fehlschlagen.

Die Arbeiterthreads, zu implementieren als **`void *dir_iter(void *path)`**, steigen rekursiv die Verzeichnishierarchie hinab und addieren für jede relevante **reguläre Datei** den Speicherverbrauch auf den Gesamtverbrauch des Besitzers. Verzeichnisse, auf die nicht zugegriffen werden kann, sollen nach Ausgabe einer Fehlermeldung ignoriert werden. Zur Aggregation des Speicherverbrauchs soll die Funktion `add_usage()` genutzt werden.

`void add_usage(uid_t uid, off_t bytes)` erwartet eine Benutzer-Id als `uid_t` und den aufzuaddierenden Speicherverbrauch vom Typ `off_t` als Argument. Auf dem Datentyp `off_t` kann normale Ganzzahl-Arithmetik angewendet werden. Dabei arbeitet die Funktion auf einer globalen Variable vom Typ `struct agg`. Für bisher unbekannte `uid`s muss das Array `usage` ggf. vergrößert werden. Dabei entstehende Lücken müssen richtig initialisiert werden. Bedenken Sie mögliche Probleme, die durch nebenläufige Zugriff entstehen.

Zur Synchronisierung soll die aus der Vorlesung und Übung bekannte Semaphor-Schnittstelle verwendet werden. Diese ist auf nächste Seite kurz beschrieben.

Hinweis: Symbolischen Verknüpfungen soll **nicht** gefolgt werden.

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

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

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

static void print_user_usage(uid_t uid, off_t bytes) {
  // Implementierung vorgegeben
}

#define MIN_UID 1000

// Erstellt eine neue Semaphor mit dem initialen Wert "initVal".
// Bei Erfolg ist der Rückgabewert ein Zeiger auf eine Semaphor.
// Andernfalls ist der Rückgabewert NULL und die errno wird
// entsprechend gesetzt.
SEM *semCreate(int initVal);

// Gibt alle von der Semaphor beanspruchten Ressourcen frei
void semDestroy(SEM *sem);

void P(SEM *sem);
void V(SEM *sem);

// Aggregieren des Speicherverbrauchs
struct agg {
  off_t        *usage;
  unsigned int  count;
};

// Speicherverbrauch pro einzelnem Nutzer zum Sortieren
struct summary {
  uid_t         uid;
  off_t         bytes;
};
```

```
// Deklaration globaler Variablen
// Vorausdeklaration von Funktionen ist nicht notwendig


// Hauptfunktion (main)
int main(int argc, char **argv) {
  // Initialisierung von Variablen


  // Threads starten


  // Auf Threads warten
```

```
  // Ausgabe vorbereiten und tätigen
```

```
  return EXIT_SUCCESS;
}
// Hilfsfunktion für die Sortierung
```

M:

```
// Funktion zum rekursiven Absteigen in Verzeichnishierarchien
```

```
  // Einträge des aktuellen Verzeichnis lesen
```

```
    // Auslesen der Dateiinformationen
    // Für Verzeichniseinträge ggf. rekursiv absteigen
```

```
  // Freigabe von Systemressourcen
```

F:

```
// Funktion um Speicherverbrauch pro Nutzer zu aggregieren
```

H:

Aufgabe 3: Fehler und Ausnahmen (12 Punkte)

Gegeben ist folgende C-Funktion, die Programmierfehler enthält.

```
int *new_array(size_t size, int value) {
    int *array = calloc(size, sizeof(*array));
    for (size_t i = 0; i <= size; i++)
        array[i] = value;
    return array;
}
```

1) Welcher der Fehler wird -wenn er auftritt- zuverlässig von der Hardware erkannt? (1 Punkt)

---

2) Welche Hardwarekomponente ist dafür verantwortlich und wie wird die Ausnahme dem Betriebssystem signalisiert? (2 Punkte)

---

3) Aunahmesituationen lassen sich in die Kategorien *Trap* und *Interrupt* einteilen. (6 Punkte)

a) Beschreiben Sie, wodurch Ausnahmesituationen der einzelnen Kategorien entstehen. (2 Punkte)

---

b) Geben Sie je ein Beispiel pro Kategorie. (2 Punkte)

---

c) Nennen Sie zwei Eigenschaften, in denen sich die Kategorien unterscheiden. (2 Punkte)

---

4) Ein weiterer Fehler kann nicht so zuverlässig durch die Hardware entdeckt werden. Nennen Sie die fehlerhafte Stelle und begründen Sie mit einem Beispiel weshalb der Fehler von der Hardware unerkannt bleiben kann. (3 Punkte)

Aufgabe 4: Speicherverwaltung (7 Punkte)

1) Speicherzuteilung erfolgt typischerweise auf verschiedenen Ebenen. Zum einen durch das Laufzeitsystem im Anwendungsprogamm (Maschinenprogramm), zum anderen auf Betriebssystemebene. Welche Art von Speicher wird auf diesen Ebenen jeweils verwaltet, was sind jeweils Ziele und Kriterien bei der Verwaltung und in welcher Granularität wird der Speicher gehandhabt? (7 Punkte)

Aufgabe 5: Betriebsmittel (4 Punkte)

1) Das folgende Programmstück zeigt einen Ausschnitt aus dem Code. Ordnen Sie die verwendeten Betriebsmittel *buffer* und *message* nach der Ihnen bekannten Klassifikation ein. Welche Konsequenzen in Bezug auf Koordinierungsmaßnahmen ergeben sich aus der Klassifikation für diese Betriebsmittel? (4 Punkte)

```
SEM *buffer_mutex;
char buffer[256];

void receive(char *message) {
  if (strlen(message) < 256) {
    P(buffer_mutex);
    strcpy(buffer, message);
    V(buffer_mutex);
  }
  free(message);
}
```