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 Adressräume ist richtig? 2 Punkte

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

b) Ausnahmesituationen bei einer Programmausführung werden in die beiden Kategorien Trap und Interrupt unterteilt. Welche der folgenden Aussagen ist zutreffend? 2 Punkte

- ☐ Ein Trap signalisiert einen schwerwiegenden Fehler und führt deshalb immer zur Beendigung des unterbrochenen Programms.
- ☐ Die CPU sichert bei einem Interrupt einen Teil des Prozessorzustands.
- ☐ Ein Systemaufruf im Anwendungsprogramm ist der Kategorie Interrupt zuzuordnen.
- ☐ Ein durch einen Interrupt unterbrochenes Programm darf je nach der Interruptursache entweder abgebrochen oder fortgesetzt werden.

c) Welche der folgenden Aussagen zum Thema Speicherverwaltung ist richtig? 2 Punkte

- ☐ Speicherbereiche, die im logischen Adressraum zusammenhängend sind, müssen auch im physikalischen Hauptspeicher zusammenhängend sein.
- ☐ Bei Segmentierung kann externe Fragmentierung des Arbeitsspeichers auftreten.
- ☐ Bei der Auslagerung einer Seite ist keine Anpassung des TLBs erforderlich.
- ☐ Beim Buddy-Verfahren können zwei aneinandergrenzende Blöcke gleicher Größe immer verschmolzen werden.

d) Man unterscheidet die Begriffe Programm und Prozess. Welche der folgenden Aussagen zu diesem Themengebiet ist richtig? 2 Punkte

- ☐ Der Binder erzeugt aus einer oder mehreren Objekt-Dateien einen Prozess.
- ☐ Mit Hilfe von Threads kann ein Prozess mehrere Programme gleichzeitig ausführen.
- ☐ Ein Programm kann immer nur von einem Prozess ausgeführt werden.
- ☐ Mit Hilfe des Systemaufrufs exec(2) wird das bestehende Programm im aktuell laufenden Prozess ersetzt.

e) Welche der folgenden Aussagen zum Thema RAID ist richtig? 2 Punkte

- ☐ Bei RAID 1 kann beim Lesen ein Geschwindigkeitsvorteil erzielt werden.
- ☐ Bei RAID 0 führt der Ausfall einer der beteiligten Platten nicht zu Datenverlust.
- ☐ Bei RAID 5 liegen die Paritätsinformationen auf einer dedizierten Platte.
- ☐ Bei RAID 4 werden alle im Verbund beteiligten Platten gleichmäßig beansprucht.

f) Welche Aussage zu Prozesszuständen ist richtig? 2 Punkte

- ☐ Ein Prozess, der sich im Zustand gestoppt befindet, kann sich selbst durch den Aufruf der Funktion fork() in den Zustand bereit überführen.
- ☐ Ein Prozess kann nur durch seine eigene Aktivität vom Zustand laufend in den Zustand blockiert überführt werden.
- ☐ Ein Prozess kann sich nicht selbst in den Zustand beendet überführen.
- ☐ Ein Prozess, der sich im Zustand laufend befindet, kann im Rahmen der mittelfristigen Einplanung in den Zustand schwebend-laufend überführt werden.

g) Welche Aussage zum Aufbau einer Kommunikationsverbindung zwischen einem Client und einem Server über eine Socket-Schnittstelle ist richtig? 2 Punkte

- ☐ Der Client kann erst connect(2) aufrufen, nachdem der Server accept(2) aufgerufen hat – vorher würde der Verbindungsversuch mit der Meldung „connection rejected“ abgewiesen werden.
- ☐ Der Server erzeugt einen Socket und ruft anschließend listen(2) auf – der Client kann daraufhin mit connect(2) eine Verbindung herstellen und sofort Daten übertragen.
- ☐ Der Server signalisiert durch den Aufruf von connect(2), dass er zur Annahme von Verbindungen bereit ist; ein Client kann dies durch accept(2) annehmen.
- ☐ Der Server richtet am Socket eine Warteschlange für ankommende Verbindungen ein und kann dann mit accept(2) eine konkrete Verbindung annehmen. accept(2) blockiert so lange die Warteschlange leer ist.

h) 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(4, 4)) * 2`

- ☐ 12
- ☐ 10
- ☐ 6
- ☐ -4

i) Welche Antwort trifft für die Eigenschaften eines UNIX/Linux-Dateideskriptors zu? 2 Punkte

- ☐ Ein Dateideskriptor ist ein Zeiger auf Betriebssystem-interne Strukturen, der von den Systemaufrufen ausgewertet wird, um auf Dateien zuzugreifen.
- ☐ Ein Dateideskriptor ist eine Verwaltungsstruktur, die auf der Festplatte gespeichert ist und Informationen über Größe, Zugriffsrechte, Änderungsdatum usw. einer Datei enthält.
- ☐ Beim Öffnen ein und derselben Datei erhält ein Prozess jeweils die gleiche Integerzahl als Dateideskriptor zum Zugriff zurück.
- ☐ Ein Dateideskriptor ist eine prozesslokale Ganzzahl, die der Prozess zum Zugriff auf eine geöffnete Datei benutzen kann.

j) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ☐ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.
- ☐ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-reference.
- ☐ Zeiger können verwendet werden, um in C eine call-by-reference Übergabesemantik nachzubilden.
- ☐ Zeiger vom Typ `void*` existieren in C nicht, da solche „Zeiger auf Nichts“ keinen sinnvollen Einsatzzweck hätten.

k) Welche Seitennummer und welcher Versatz gehören bei einer Seitengröße von 4096 Bytes zu folgender logischer Adresse: 0xcafe 2 Punkte

- ☐ Seitennummer 0x19, Versatz 0x2fe
- ☐ Seitennummer 0xca, Versatz 0xfe
- ☐ Seitennummer 0xc, Versatz 0xafe
- ☐ Seitennummer 0x32, Versatz 0x2fe

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) Welche der folgenden Aussagen zum Thema Synchronisation sind richtig? 4 Punkte

- ○ Gibt ein Faden einen Mutex frei, den er selbst zuvor nicht angefordert hatte, stellt dies einen Programmierfehler dar; der fehlerhafte Prozess sollte dann abgebrochen werden.
- ○ Ein Mutex kann ausschließlich für einseitige Synchronisation verwendet werden.
- ○ Zur Synchronisation eines kritischen Abschnitts ist passives Warten immer besser geeignet als aktives Warten.
- ○ Der Einsatz von nicht-blockierenden Synchronisationsmechanismen kann zu Verklemmungen (deadlocks) führen.
- ○ Für nichtblockierende Synchronisation werden spezielle Befehle der Hardware genutzt, die wechselseitigen Ausschluss garantieren.
- ○ Ein Anwendungsprozess muss bei der Verwendung von Semaphoren Interrupts sperren, um Probleme durch Nebenläufigkeit zu verhindern.
- ○ Semaphore können sowohl für einseitige als auch für mehrseitige Synchronisation verwendet werden.
- ○ Die V-Operation kann auf einem Semaphor auch von einem Faden aufgerufen werden, der zuvor keine P-Operation auf dem selben Semaphor ausgeführt hat.

b) Welche der folgenden Aussagen zu UNIX-Dateisystemen sind richtig?

4 Punkte

- ◯ Wird eine Datei gelöscht, so werden auch alle symbolic links, die auf diese Datei verweisen, gelöscht.
- ◯ Die Anzahl der hard links, die auf ein Verzeichnis verweisen, hängt von der Anzahl seiner Unterverzeichnisse ab.
- ◯ In einem Verzeichnis darf es mehrere Einträge mit identischem Namen geben, sofern sie auf unterschiedliche Inodes verweisen.
- ◯ Der Inode einer Datei wird in der Regel getrennt von ihrem Inhalt auf der Platte gespeichert.
- ◯ Der selbe Inode kann im Dateisystem im selben Verzeichnis mehrfach über verschiedene Namen referenziert werden.
- ◯ Beim lesenden Zugriff auf eine Datei über einen symbolic link kann ein Prozess den Fehler `Permission denied` erhalten, obwohl er das Leserecht auf dem symbolic link besitzt.
- ◯ Ein Inode enthält u.a. den Namen der entsprechenden Datei.
- ◯ Ein hard link kann nur auf Verzeichnisse, nicht jedoch auf Dateien verweisen.

Aufgabe 2: easi (59 Punkte)

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

Schreiben Sie ein Programm `easi` (**e**asy **a**utomated **s**oftware **i**nstallation), das parallel Dateien von Webseiten herunterlädt und diese mithilfe von `bash` und `sudo` mit root-Rechten ausführt. Das Programm ersetzt `curl | sudo bash`, da es einfacher auszuführen ist und dank Parallelität den Code von mehreren Webseiten schneller ausführen kann. Die einzelnen URLs werden als Argumente auf der Kommandozeile übergeben. Um die ausgeführten Befehle zu dokumentieren, schreibt das Programm den heruntergeladenen Code sowie die Ausgabe der Programme in eigene Dateien. Fehler zur Laufzeit beenden den jeweiligen Prozess mit einem Fehlercode, dürfen allerdings **keine Fehlermeldungen** erzeugen, um die Nutzer nicht zu verunsichern.

Das *Hauptprogramm* konfiguriert die Signalbehandlung um Zombies **sofort** aufzusammeln und startet dann für jede URL einen Prozess, der diese URL mithilfe der Funktion `handle_url()` bearbeitet. Prozesse für mehrere URLs laufen parallel. Wenn das Limit von `PROCS_MAX` gleichzeitig **laufenden** Prozessen erreicht ist, wird vor dem Starten eines neuen Prozesses **passiv** gewartet, bis sich ein Prozess beendet hat. Die Anzahl von erfolgreich beendeten (anhand des Exitcodes) und den übrigen, fehlgeschlagenen Prozessen wird gezählt.

Wurden alle Prozesse gestartet, wartet easi **passiv** bis sich alle gestarteten Prozesse beendet haben und gibt dann die Anzahl erfolgreicher/fehlgeschlagener Prozesse aus. Vor dem Beenden wird `delete_files()` aufgerufen (siehe unten). Die Ausgabe soll dem folgenden Beispiel entsprechen:

```
status: success 84, failure 72
```

`void delete_files(void)` löscht mithilfe von `unlink(2)` nicht-rekursiv alle **leeren** regulären Dateien im aktuellen Verzeichnis, deren Name mit `isae.` beginnt.

`void fetch_file(const char *dest, const char *host, const char *path)` baut mithilfe der **vorgegebenen** Funktion `tcp_connect()` eine Verbindung zu Port 80 des Zielhosts `host` auf, fordert die Datei `path` per GET-Request (`GET <path> HTTP/1.0\r\n\r\n`) an und schreibt die Antwort des Servers in die Zieldatei `dest`. Alle Zeilen des HTTP-Headers der Antwort müssen ignoriert werden. Der Header endet mit einer Leerzeile (`\r\n`). Gehen Sie davon aus, dass der Server keine HTTP-Fehler schickt, jede Zeile maximal `LINE_MAX` Zeichen lang ist und keine Nullbytes enthält.

`void handle_url(const char *url)` bearbeitet eine URL wie folgt:

1. Um zu verhindern, dass Befehle verschlüsselt übertragen werden, müssen alle URLs mit `http://` beginnen. Sonst wird der bearbeitende Prozess mit einem Fehlercode beendet.
2. Die Datei wird mit `fetch_file()` heruntergeladen und unter `isae.<host>.<pid>` gespeichert (`getpid(2)` liefert die aktuelle PID, die immer kleiner als 65535 sei).
3. Die erstellte Datei wird mit `/usr/bin/sudo /bin/bash <file>` mithilfe einer Funktion der `exec()`-Familie ausgeführt. Vorher ist die Ausgabe (aber nicht die Fehlerausgabe) in eine Datei der Form `isae.<host>.<pid>.log` umzuleiten.

Hinweise:

- `waitpid(2)` benötigt in dieser Aufgabe keine Fehlerbehandlung.
- Um die URL in Host und Pfad zu zerlegen, bieten sich Zeigerarithmetik und bekannte String-Funktionen an. Für das Beispiel `http://example.org/file/path` ist `example.org` der Host und `/file/path` der Pfad. Hosts können kein `/` enthalten. Sie dürfen annehmen, dass Pfade nicht leer sind.
- Ausgaben auf der Fehlerausgabe von gestarteten Programmen gelten nicht als Fehlermeldungen und müssen nicht gesondert behandelt werden.
- Die vorgegebene Funktion `tcp_connect()` hat folgende Signatur:
  `int tcp_connect(const char *host, int port, FILE **read, FILE **write)`

```c
#include <dirent.h>
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define LINE_MAX    1024
#define PROCS_MAX   42

#define STDOUT_FILENO 1 /* fd for stdout */

/*
 * tcp_connect() baut eine TCP-Verbindung zu dem Host host und dem Port
 * port auf, erstellt je einen FILE-Pointer zum Lesen und Schreiben und
 * schreibt diese in die übergebenen Parameter. Der Aufrufer muss die
 * FILE-Pointer nach Benutzung selbst schließen. Im Erfolgsfall wird 0
 * zurück gegeben, im Fehlerfall eine negative Zahl.
 */
static int tcp_connect(const char *host, int port,
                       FILE **read, FILE **write);

static void fail(void) {
    exit(EXIT_FAILURE);
}

// Funktions- & Strukturdekl., globale Variablen, etc.
```



```c
// Hauptfunktion (main)
int main(int argc, char *argv[]) {
    // Signalbehandlung initialisieren


    // Argumente verarbeiten


        // Anzahl laufender Prozesse auf PROCS_MAX limitieren
```

```
        // Prozess zur Bearbeitung starten
```

```
    // Statusausgabe
```

```
}
// Ende Hauptfunktion
```

M:


```
// Funktion Signalbehandlung
```

S:

```
// Funktion delete_files
```

D:


```
// Funktion fetch_file

    // Verbindung aufbauen

    // HTTP-Anfrage stellen

    // Zieldatei erstellen
```

```
    // Antwort verarbeiten
```

```
    // Cleanup
```

F:


```
// Funktion handle_url
```

```
    // URL zerlegen und Datei herunterladen
```

```
    // Ausgabe umleiten
```

```
    // Programm ausführen
```

H:

Aufgabe 3: Prozesse/Threads (9 Punkte)

1) Man unterscheidet verschiedene Gewichtsklassen von Prozessen/Threads. Welche sind dies und wodurch entsteht das unterschiedliche "Gewicht"? (6 Punkte)

2) Welche Eigenschaften haben die unterschiedlichen Prozess/Thread-Arten im Hinblick auf Multiprozessoren bzw. Mehrkernprozessoren? (3 Punkte)

Aufgabe 4: Speicherverwaltung (12 Punkte)

1) Falls kein freier Seitenrahmen im Hauptspeicher verfügbar ist, muss eine Seite ausgelagert werden. Nennen Sie **zwei** mögliche Strategien zur Bestimmung der zu verdrängenden Seite (ausgenommen Second Chance, CLOCK). Beschreiben Sie die Strategien jeweils kurz. (2 Punkte)

2) Nicht-optimale Ersetzungsstrategien nutzen eine gewisse Programmeigenschaft aus, um sinnvolle Ergebnisse zu erzielen. Nennen Sie diese Eigenschaft und erklären Sie kurz, warum sie auf die meisten Programme zutrifft. (2 Punkte)

3) Virtualisierte Speicherverwaltung benötigt Unterstützung durch die Hardware. Nennen Sie die benötigte Hardware und nötige Zusatzeinträge in Seitendeskriptoren um Strategien wie CLOCK zu implementieren. (2 Punkte)

4) Eine in der Praxis gut einsetzbare Strategie ist Second Chance: CLOCK. In dieser Aufgabe soll CLOCK als (Prozess-)lokale Seitenersetzungsstrategie eingesetzt werden. (6 Punkte)

Im Folgenden sind die für die Seitenverwaltung erforderlichen Daten der aktuell anwesenden Seiten eines Prozesses dargestellt. Nehmen Sie eine Seitengröße von 4096 Bytes an (ergibt 12 Bit Offset).

CLOCK-Zeiger	Seitennummer	Anwesend	Schreibbar	Benutzt
	`0x00003`	1	1	1
	`0x00001`	1	1	0
⟶	`0x00004`	1	0	1
	`0x00006`	1	1	1

a) Welche der folgenden Zugriffe lösen Seitenfehler aus (Ursache angeben) oder verändern etwas an den dargestellten Daten (Änderung angeben)? (3 Punkte)

1. `0x000019df` (lesend)
2. `0x00002fa1` (lesend)
3. `0x00004acf` (schreibend)
4. `0x000063ad` (schreibend)

b) Welche Seite würde das Betriebssystem aufgrund der dargestellten Daten auslagern, wenn es die CLOCK-Strategie verwendet? Die Zugriffe aus a) sind **nicht** zu berücksichtigen! (1 Punkt)

c) Auf welche Seite zeigt der CLOCK-Zeiger nach der Auslagerung? (1 Punkt)

d) Wie reagiert das Betriebssystem auf einen Prozess, der mit einem Zugriff einen Seitenfehler auslöst, der gegen Einschränkungen (wie schreibbar) verstößt? (1 Punkt)

Aufgabe 5: Synchronisation (10 Punkte)

1) Was versteht man unter einer Verklemmung und was sind die (hinreichenden und notwendigen) Bedingungen, damit eine Verklemmung auftreten kann? (6 Punkte)

Die im folgenden skizzierten Programmstücke werden von 2 Threads parallel ausgeführt. Hierbei wird jeweils ein Datum aus einem Puffer gelesen, in einen anderen Puffer eingefügt und – falls dies erfolgreich war – das Datum anschließend aus dem ersten Puffer entfernt. Der Zugriff auf die Puffer darf nicht gleichzeitig von mehreren Threads erfolgen, weshalb die Zugriffe durch Semaphore synchronisiert sind.

```
thread1() {
  P(&sem_buf1);
  item = get_item(&buf1);
  P(&sem_buf2);
  if (put_item(&buf2, item) > 0) {
    remove_item(&buf1);
    ret = 1;
  } else {
    ret = -1;
  }

  V(&sem_buf2);
  V(&sem_buf1);
  return ret;
}
```

```
thread2() {
  P(&sem_buf2);
  item = get_item(&buf2);
  P(&sem_buf1);
  if (put_item(&buf1, item) > 0) {
    remove_item(&buf2);
    ret = 1;
  } else {
    ret = -1;
  }

  V(&sem_buf1);
  V(&sem_buf2);
  return ret;
}
```


2) Welches grundlegende Problem in Bezug auf Verklemmungen besteht in der angegebenen Implementierung? Beziehen Sie sich bei Ihrer Antwort auf die in 1) beschriebenen Voraussetzungen für Verklemmungen. Geben Sie zwei mögliche Lösungen für dieses Problem an. (4 Punkte)