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 Aussage zu Prozesszuständen ist richtig? 2 Punkte

- ☐ Ein Prozess kann sich nicht selbst in den Zustand *beendet* überführen.
- ☐ 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, der sich im Zustand *laufend* befindet, kann im Rahmen der mittelfristigen Einplanung in den Zustand *schwebend-laufend* überführt werden.

b) Welche der folgenden Informationen wird typischerweise in dem Seitendeskriptor einer Seite eines virtuellen Adressraums gehalten? 2 Punkte

- ☐ die Identifikation des Prozesses, dem die Seite zugeordnet ist
- ☐ die Position der Seite im virtuellen Adressraum
- ☐ Zugriffsrechte (z. B. lesen, schreiben, ausführen)
- ☐ die Zuordnung zu einem Segment (Text, Daten, Stack, ...)

c) Welche Aussage ist in einem Monoprozessor-Betriebssystem richtig? 2 Punkte

- ☐ Ein Prozess im Zustand *blockiert* muss warten, bis der laufende Prozess den Prozessor abgibt und kann dann in den Zustand *laufend* überführt werden.
- ☐ In den Zustand *blockiert* gelangen Prozesse nur aus dem Zustand *bereit*.
- ☐ Ist zu einem Zeitpunkt kein Prozess im Zustand *bereit*, so ist auch kein Prozess im Zustand *laufend*.
- ☐ Es befindet sich zu jedem Zeitpunkt maximal ein Prozess im Zustand *laufend*.

d) Welche der folgenden Aussagen zum Thema Threads ist richtig? 2 Punkte

- ☐ Die Umschaltung von User-Threads ist eine privilegierte Operation und muss deshalb im Systemkern erfolgen.
- ☐ Kern-Threads blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ☐ User-Threads blockieren sich bei blockierenden Systemaufrufen gegenseitig.
- ☐ Zu jedem Kernel-Thread gehört ein eigener Adressraum.

e) Welche der folgenden Aussagen zu UNIX-Dateisystemen ist richtig? 2 Punkte

- ☐ Beim Anlegen einer Datei wird die maximale Größe festgelegt. Wird sie bei einer Schreiboperation überschritten, wird ein Fehler gemeldet.
- ☐ Ein Pfadname, der nicht mit einem '/'-Zeichen beginnt, wird relativ zum Home-Verzeichnis des Benutzers interpretiert.
- ☐ Zur Anzeige des Inhaltes einer Datei ist es notwendig, das Leserecht auf dem übergeordneten Verzeichnis zu besitzen.
- ☐ Der Name einer Datei wird getrennt von ihrem Indexknoten (inode) gespeichert.

f) Virtualisierung kann als Maßnahme gegen Verklemmungen genutzt werden. Warum? 2 Punkte

- ☐ Eine Verklemmungsauflösung ist einfacher, weil virtuelle Betriebsmittel jederzeit ohne Schaden entzogen werden können.
- ☐ Durch Virtualisierung kann man über Abbildungsvorgänge Zyklen, die auf der logischen Ebene vorhanden sind, auf der physikalischen Ebene auflösen.
- ☐ Durch Virtualisierung ist ein Entzug von physikalischen Betriebsmitteln möglich, obwohl dies auf der logischen Ebene unmöglich ist.
- ☐ Im Fall einer Verklemmung können zusätzliche virtuelle Betriebsmittel neu erzeugt werden. Diese können dann eingesetzt werden, um die fehlenden physikalischen Betriebsmittel zu ersetzen.

g) Namensräume dienen u. a. der Organisation von Dateisystemen. Welche Aussage ist richtig? 2 Punkte

- ☐ In einem hierarchisch organisierten Namensraum dürfen gleiche Namen in unterschiedlichen Kontexten enthalten sein.
- ☐ Der Nachteil von hierarchischen Namensräumen besteht darin, dass das Dateisystem spezielle Funktionen zum Auflösen von Namenskonflikten implementieren muss.
- ☐ Hierarchische Namensräume werden erzeugt, indem man in einem Kontext symbolische Verweise auf Dateien einträgt.
- ☐ Flache Namensräume sind besonders einfach implementierbar und damit vor allem für Mehrbenutzersysteme gut geeignet.

h) 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 durch einen Interrupt unterbrochenes Programm darf je nach der Interrupt-Ursache entweder abgebrochen oder fortgesetzt werden.
- ☐ Ein Systemaufruf im Anwendungsprogramm ist der Kategorie Interrupt zuzuordnen.

i) Welche Aussage zu Programmbibliotheken ist richtig?

2 Punkte

- ☐ Eine statische Bibliothek, die in ein Programm eingebunden wurde, muss zum Ladezeitpunkt dieses Programms im Dateisystem vorhanden sein.
- ☐ Programm-Module, die von mehreren Anwendungen gemeinsam genutzt werden, können in Form einer dynamischen Bibliothek zentral installiert werden, um Speicherplatz zu sparen.
- ☐ Statische Bibliotheken können nicht in C implementiert werden.
- ☐ Änderungen am Code einer dynamischen Bibliothek (z. B. Bugfixes) erfordern immer das erneute Binden aller Programme, die diese Bibliothek benutzen.

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 = 2018;

int f1 (const int *y) {
    static int b;
    int c;
    char *d = malloc(1337);
    int (*e)(const int *) = f1;
    y++;
    // ...
}
```

Welche der folgenden Aussagen zum obigen Programmfragment sind richtig?

- ○ `a` liegt im Datensegment.
- ○ `c` ist mit dem Wert `0` initialisiert.
- ○ Die Speicherstelle, auf die `d` zeigt, verliert beim Rücksprung aus der Funktion `f1()` ihre Gültigkeit.
- ○ `b` liegt im Stacksegment.
- ○ `e` liegt im Stacksegment und zeigt in das Textsegment.
- ○ Die Anweisung y++ führt zu einem Laufzeitfehler, da `y` konstant ist.
- ○ `d` ist ein Zeiger, der in den Heap zeigt.
- ○ `y` liegt im Stacksegment.

Aufgabe 2: gitbox (45 Punkte)

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

Sie sind einer der letzten verbleibenden Entwickler von gitbox.com nach der Übernahme durch einen großen Betriebssystemhersteller. Schreiben Sie ein Programm *gitbox*, das in der Lage ist eine Sicherheitskopie des kompletten Datenbestands herzustellen, um so den Verlust einer großen Zahl von Open-Source-Projekten zu verhindern.

Das Programm *gitbox* kopiert ein als Befehlszeilenargument übergebenes Verzeichnis rekursiv in ein anderes, ebenfalls per Befehlszeile übergebenes, Verzeichnis. Um die Kopierdauer zu reduzieren, soll der Kopierprozess in `MAX_THREAD` Threads gleichzeitig ausgeführt werden.

Das Programm soll folgendermaßen arbeiten:

Das Hauptprogramm prüft die Anzahl der Befehlszeilenargumente, initialisiert notwendige Datenstrukturen und startet den parallelen Kopierprozess durch Aufruf der Funktion `copy_dir`. Im Anschluss daran wartet das Hauptprogramm aktiv auf die Beendigung aller verbleibender Kopierthreads.

Funktion `void copy_dir(const char *src, const char *dst)`*:* Kopiert alle regulären Dateien und Verzeichnisse in *src* (rekursiv) nach *dst*. Einträge, die weder eine reguläre Datei, noch ein Verzeichnis sind, sollen dabei ignoriert werden.
Falls die Anzahl der aktuell laufenden Threads unter `MAX_THREAD` liegt, wird zum rekursiven Kopieren von Unterverzeichnissen einen Thread erzeugt (Einstiegspunkt `tstart`).
Hinweis: Nutzen Sie zur Übergabe des Kopierauftrags eine Struktur, die *src* und *dst* beinhaltet.
Andernfalls wird das Unterverzeichnis durch Aufruf von `copy_dir` im aktuellen Thread kopiert.

Funktion `void* tstart(void *arg)`: Einstiegspunkt erzeugter Threads. Führt den als `arg` übergebenen Kopierauftrag durch Aufruf von `copy_dir` aus.

Die *Funktion* `void copy_file(const char *src, const char *dst)` kopiert die Datei *src* byteweise nach *dst*.

Achten Sie auf korrekte Synchronisation.

Hinweise:

- Ihnen steht das aus der Übung bekannte Semaphor-Modul zur Verfügung. Die Schnittstelle finden Sie im folgenden Programmgerüst nach den #include-Anweisungen.
- Einzelne Verzeichnisse können mit Hilfe der vorgegebenen Funktion `create_dir` (vgl. folgende Seite) erzeugt werden.
- Nutzen Sie eine Zählvariable zur Verwaltung der aktuell laufenden Threads.
- Zur Vereinfachung dürfen Sie annehmen, dass das auf der Befehlszeile übergebene Zielverzeichnis nicht existiert.
- Zur Vereinfachung dürfen Sie die Ausführung von gitbox mittels `die()` (vgl. nächste Seite) im Fehlerfall abbrechen.


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

#define MAX_THREAD 8

// Interface sem.h
SEM *semCreate(int initVal);
void P(SEM *sem);
void V(SEM *sem);
void semDestroy(SEM *sem);

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

/** Creates the directory specified by parameter ‘path‘.
 *  Only creates a single directory; recursive creation is not supported.
 *  Returns true on success, false otherwise.
 */
static bool create_dir(const char *path) {
  return 0 == mkdir(path, 0777);
}

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

```
// Hauptfunktion (main)
int main(int argc, char *argv[]) {
  if(argc != 3) {
    fprintf(stderr, "Usage: %s <src> <dst>\n", argv[0]);
    return EXIT_FAILURE;
  }
```

```
  return EXIT_SUCCESS;
} // Ende Funktion main
```

M:


```
// Funktion copy_dir
```

```
  // Verzeichnis durchlaufen und Pfade zusammenbauen
```

```
    // Kopieren der Datei / des Verzeichnisses
```



```
// Ende Funktion copy_dir
// Funktion tstart
```

```
// Ende Funktion tstart
```

D:

```
// Funktion copy_file
```

```
// Ende Funktion copy_file
```

F:

Aufgabe 3: Adressräume (14 Punkte)

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

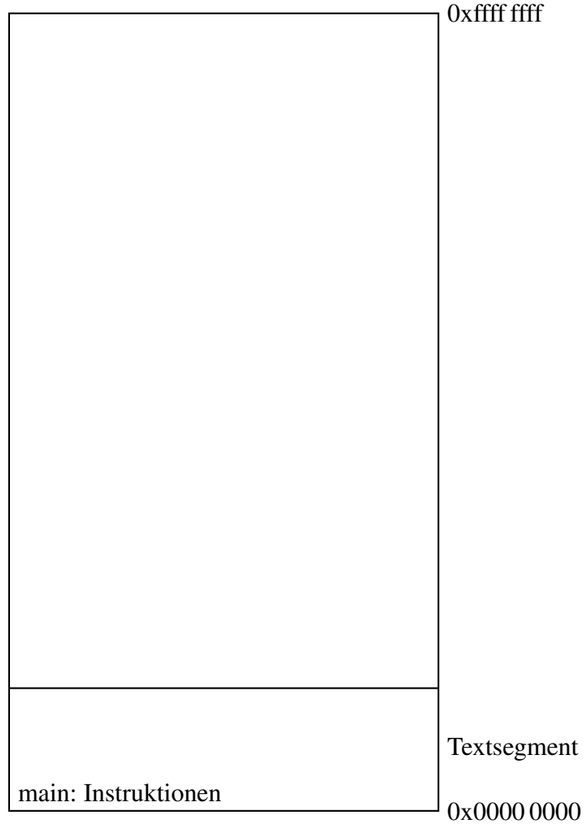
1) Tragen Sie die Segmente und deren Namen (analog zum schon vorgegebenen Textsegment) in unten stehende Zeichnung ein. Unterscheiden Sie hierbei die Bereiche zur Speicherung von initialisierten und nicht initialisierten Variablen. Zeichnen Sie für jede Variable ein, wo diese ungefähr im logischen Adressraum zu finden sein wird und welchen Wert sie enthält. 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. (9 Punkte)

```
static int a = 3;
static int b = 0;

const char *s = "Hello World\n";

int main(void) {
    int g = 5;
    static int h = 12;
    void *arr = malloc(5);
    int (*func)(void) = main;
}
```



2) Wenn das folgende Programmstück in einem UNIX-System abläuft, wird ein Fehler auftreten. (5 Punkte)

```
// ...
int *p = NULL;
// ...
*p = -1;
// ...
```

a) Welcher Fehler tritt auf? (1 Punkt)

b) Warum tritt der Fehler auf? (1 Punkt)

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

d) Mit welchem Mechanismus wird der Fehler dem Betriebssystem mitgeteilt? (1 Punkt)

e) Was macht das Betriebssystem mit dem Prozess, der gerade das Programmstück ausführt? (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)