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) Wie funktioniert Adressraumschutz durch Einfriedung? 2 Punkte

- ☐ Der Übersetzer grenzt bei Erzeugung des Codes die Adresszugriffe auf einen bestimmten Bereich ein.
- ☐ Die MMU kennt die Länge eines Segments und verhindert Speicherzugriffe darüber hinaus.
- ☐ Der Lader positioniert Programme immer so im Arbeitsspeicher, dass unerlaubte Adressen mit nicht-existierenden physikalischen Speicherbereichen zusammenfallen.
- ☐ Begrenzungsregister legen einen Adressbereich im physikalischen Adressraum fest, auf den alle Speicherzugriffe beschränkt werden.

b) Welche der folgenden Aussagen über Schedulingverfahren ist richtig? 2 Punkte

- ☐ Bei kooperativem Scheduling sind Prozessumschaltungen unmöglich, wenn ein Prozess in einer Endlosschleife läuft. Selbst wenn er bei jedem Schleifendurchlauf einen Systemaufruf macht.
- ☐ Online-Schedulingverfahren sind für den Einsatz in Rechnern ohne Netzwerkschnittstelle ungeeignet.
- ☐ Verdrängende Schedulingverfahren können nur mit Hilfe von Unterbrechungen realisiert werden.
- ☐ Deterministische Schedulingverfahren sind nur in der Theorie relevant, da die genaue Länge der CPU-Stöße nie vorhergesagt werden kann.

c) Welche der folgenden Aussagen zum Thema Prozesse und Threads ist richtig? 2 Punkte

- ☐ Bei schwergewichtigen Prozessen ist die Schedulingstrategie durch das Betriebssystem vorgegeben.
- ☐ Jeder federgewichtige Prozess (User-Thread) und jeder leichtgewichtige Prozess (Kern-Thread) hat seinen eigenen, geschützten Adressraum.
- ☐ Bei Blockade eines schwergewichtigen Prozesses werden alle anderen schwergewichtigen Prozesse, die das selbe Progamm ausführen, ebenfalls blockiert.
- ☐ Unabhängig von leichtgewichtigen Prozessen (Kernel-Threads) können federgewichtige Prozesse (User-Threads) Multiprozessoren ausnutzen.

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

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

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

- ☐ Bei RAID 5 werden alle im Verbund beteiligten Platten gleichmäßig beansprucht.
- ☐ Bei RAID 0 können nach Ausfall einer der beteiligten Platten die Daten durch die Information der anderen rekonstruiert werden.
- ☐ Der Lesedurchsatz eines RAID-Systems mit mehreren Platten ist prinzipbedingt geringer als der Lesedurchsatz einer einzelnen Platte.
- ☐ Bei RAID 4 werden alle im Verbund beteiligten Platten gleichmäßig beansprucht.

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

- ☐ Bei Eintreffen eines Interrupts wird der aktuell laufende Prozess für die Dauer der Interrupt-Abarbeitung in den Zustand blockiert überführt.
- ☐ Das Auftreten eines Seitenfehlers kann dazu führen, dass der aktuell laufende Prozess in den Zustand beendet überführt wird.
- ☐ Im Rahmen der mittelfristigen Einplanung kann ein Prozess von Zustand laufend in den Zustand schwebend laufend wechseln.
- ☐ Bei kooperativem Scheduling ist kein direkter Übergang vom Zustand laufend in den Zustand bereit möglich.

g) Welche Aussage zur Seitenumlagerung in virtuellen Adressräumen ist richtig? 2 Punkte

- ☐ Bei der Seitenersetzungsstrategie LRU wird diejenige Seite ausgelagert, auf die in der Vergangenheit am seltensten zugegriffen wurde.
- ☐ Beim Auslagern einer Speicherseite muss der zugehörige Seitendeskriptor angepasst werden. Beim Einlagern ist dies nicht nötig.
- ☐ Unter Seitenflattern versteht man das ständige Ein- und Auslagern von Speicherseiten, wenn der physisch vorhandene Hauptspeicher nicht ausreicht.
- ☐ Die Seitenersetzungsstrategie Second Chance (Clock) ist nur in der Theorie interessant, weil ihre Implementierung komplexe Datenstrukturen erfordert.

h) Gegeben sei folgendes Szenario: zwei Fäden werden auf einem Monoprozessorsystem mit der Strategie „First Come First Served“ verwaltet. In jedem Faden wird die Anweisung `i++;` auf der gemeinsamen, globalen **`volatile`** Variablen `i` ausgeführt. Welche der folgenden Aussagen ist richtig?

2 Punkte

- ☐ Die Operation `i++` ist auf einem Monoprozessorsystem immer atomar.
- ☐ Während der Inkrementoperation müssen Interrupts vorübergehend unterbunden werden.
- ☐ Die Inkrementoperation muss mit einer CAS-Anweisung synchronisiert werden.
- ☐ In einem Monoprozessorsystem ohne Verdrängung ist keinerlei Synchronisation erforderlich.

i) Was passiert, wenn auf einer Hardwarearchitektur mit MMU in einem C-Programm über einen Zeiger auf ungültigen Speicher zugegriffen wird?

2 Punkte

- ☐ Der Übersetzer erkennt die problematische Code-Stelle und generiert Code, der zur Laufzeit bei dem Zugriff einen entsprechenden Fehler auslöst.
- ☐ Beim Zugriff muss die MMU die erforderliche Adressumsetzung vornehmen. Sie erkennt die ungültige Adresse und löst einen Trap aus.
- ☐ Das Betriebssystem erkennt die ungültige Adresse bei der Weitergabe des Befehls an die CPU (partielle Interpretation) und leitet eine Ausnahmebehandlung ein.
- ☐ Der Speicher schickt an die CPU einen Interrupt. Hierdurch wird das Betriebssystem angesprungen, das dem gerade laufenden Prozess das Signal SIGSEGV(Segmentation Violation) zustellt.

j) Für lokale Variablen, Aufrufparameter usw. einer Funktion wird bei vielen Prozessoren ein Stack-Frame angelegt. Welche Aussage ist richtig?

2 Punkte

- ☐ Ein Pufferüberlauf eines lokalen Arrays wird immer zu einem Segmentation Fault führen und kann somit keine sicherheitskritischen Auswirkungen haben.
- ☐ Es ist nicht möglich auf lokale automatic-Variablen zuzugreifen, die sich im Stack-Frame einer anderen Funktion befinden.
- ☐ Bei rekursiven Funktionsaufrufen kann der Speicher des Stack-Frames in jedem Fall wiederverwendet werden, weil die gleiche Funktion aufgerufen wird.
- ☐ Wenn in einem UNIX-Prozess mehrere Threads parallel laufen, benötigt jeder von ihnen einen eigenen Stack.

k) 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.
- ☐ Ein Symbolic Link kann nicht auf Dateien anderer Dateisysteme verweisen.
- ☐ Für jede reguläre Datei existiert mindestens ein Hard-Link im selben Dateisystem.
- ☐ Auf jedes Verzeichnis verweist immer genau ein Hard-Link.

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 zu UNIX/Linux-Dateideskriptoren sind korrekt?

4 Punkte

- ○ Ein Dateideskriptor ist eine Verwaltungsstruktur, die auf der Festplatte gespeichert ist und Informationen über Größe, Zugriffsrechte, Änderungsdatum usw. einer Datei enthält.
- ○ Nach dem Aufruf von `fork(2)` teilen sich die Prozesse die den gemeinsamen Dateideskriptoren zu Grunde liegenden Kernel-Datenstrukturen.
- ○ Ist das Flag `FD_CLOEXEC` eines Dateideskriptors gesetzt, dann wird dieser Dateideskriptor geschlossen, sobald der Prozess eine Funktion der `exec`-Familie aufruft.
- ○ Ein Dateideskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine Datei benutzen kann.
- ○ Ein Dateideskriptor ist eine Integerzahl, die über gemeinsamen Speicher an einen anderen Prozess übergeben werden kann und von letzterem zum Zugriff auf eine geöffnete Datei verwendet werden kann.
- ○ Auch Netzwerkverbindungen werden über einen Dateideskriptor referenziert.
- ○ Wird ein Dateideskriptor mittels des `dup(2)`-Systemaufruf vervielfältigt, können die Zugriffsrechte auf dem resultierendem Deskriptor unabhängig vom ursprünglichen geändert werden.
- ○ Dateideskriptoren sind Zeiger auf Betriebssystem-interne Strukturen, die von den Systemaufrufen ausgewertet werden, um auf Dateien zuzugreifen.

b) Welche der folgenden Aussagen zum Thema persistenter Datenspeicherung sind richtig?

4 Punkte

- ○ Bei verketteter Speicherung dauert der wahlfreie Zugriff auf eine bestimmte Dateiposition immer gleich lang, wenn Cachingeffekte außer Acht gelassen werden.
- ○ Bei verketteter Speicherung mittels FAT-Ansatz kann die Verkettungsinformation redundant gespeichert werden, um die Fehleranfälligkeit zu reduzieren.
- ○ Journaling-Dateisysteme sind immun gegen defekte Plattenblöcke.
- ○ Bei indizierter Speicherung kann es prinzipbedingt nicht zu Verschnitt kommen.
- ○ Bei kontinuierlicher Speicherung von Daten ist es unter Umständen mit enormem Aufwand verbunden, eine bestehende Datei zu vergrößern.
- ○ Im Vergleich zu den anderen Verfahren ist bei indizierter Speicherung die Positionierzeit des Festplatten-Armes beim Zugriff auf alle Datenblöcke einer Datei minimal.
- ○ Journaling-Dateisysteme garantieren, dass auch nach einem Systemausfall alle Metadaten wieder in einen konsistenten Zustand gebracht werden können.
- ○ Festplatten eignen sich besser für sequentielle als für wahlfreie Zugriffsmuster.

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

Aufgabe 2: `spam` (60 Punkte)

Schreiben Sie das Programm `spam` (**s**ophisticated **p**arallel **a**ppointment **m**aker), das dabei helfen kann, einen Termin bei einer Vielzahl von Empfängern anzukündigen. Dafür werden `spam` ein Ersetzungstext und als weitere Argumente die Namen von Verzeichnissen übergeben. In jedem Verzeichnis wird rekursiv nach Vorlagen gesucht, um diese jeweils in einem eigenen Faden an den jeweiligen Adressaten zu schicken. Eine Vorlage wird dabei jeweils unter dem Dateinamen des Empfängers gespeichert und enthält beliebigen Text. Beim Versenden wird jedes Vorkommen des Zeichens `'$'` durch den gewünschten Ersetzungstext substituiert. Um eine Überlastung des Kommunikationssystems auszuschließen, dürfen nie mehr als 42 (`MAX_THREADS`) Arbeiterfäden aktiv sein.

Implementieren Sie die folgenden Funktionen:

`void main(int argc, char *argv[])` Prüft zu Beginn die Parameterzahl und initialisiert den globalen Zustand. Anschließend wird für jeds Verzeichnisargument `crawl` aufgerufen. Schlussendlich muss sichergestellt werden, dass alle Fäden ihre Nachricht gesendet haben und belegte Ressourcen freigegeben wurden.

`void crawl(const char *dir)` Durchsucht das Verzeichnis nach Vorlagedateien, wobei versteckte Verzeichniseinträge (Name beginnt mit `'.'`) ignoriert werden. Unterverzeichnisse sollen rekursiv durchsucht werden. Der Pfadname jeder regulären Datei, die dem ausführenden Benutzer gehört, wird einem eigenen Faden (`worker`) zur Bearbeitung übergeben. Dabei muss das Limit von aktiven Fäden beachtet werden. Auftretende Fehler sollen (außer einer Fehlermeldung) keine weitere Auswirkung auf die Bearbeitung anderer Dateien/Verzeichnisse haben.

`void *worker(void *)` Einstiegsfunktion der Fäden zur Bearbeitung einer Vorlage. Ein `worker` extrahiert aus dem übergebenen Vorlagenpfad den Dateinamen als Verbindungsinformation und führt die Funktion `request` aus. Der Rückgabewert dieser Funktion wird anschließend zusammen mit dem Vorlagenpfad auf `stdout` ausgegeben. Threads sollen die Freigabe ihrer Ressourcen nach Terminierung selbst veranlassen.

`int request(const char *target, const char *template)` Öffnet mit der gegebenen Funktion `start_connection` (siehe nächste Seite) eine Verbindung zu `target` sowie die Vorlagedatei. `FILE`-Objekte auf Verbindung und Vorlagedatei werden an `send_template` übergeben und belegte Ressourcen anschließend freigeräumt. Auftretende Fehler werden ausgegeben und dem Aufrufer mit Rückgabewert `-1` gemeldet. Andernfalls wird der Rückgabewert von `send_template` zurückgegeben.

`int send_template(FILE *tx, FILE *template)` Schreibt den Inhalt aus `template` nach `tx`. Dabei wird jedoch das Zeichen `'$'` durch den Ersetzungstext ersetzt. Tritt ein Fehler auf, wird dieser ohne Ausgabe mit dem Rückgabewert `-1` gemeldet. Wurde die gesamte Datei erfolgreich versendet, wird die Zahl an vorgenommenen Ersetzungen zurückgegeben.

Beispielhafter Aufruf mit Ausgabe

```
$ ./spam "20. Juli 2021" sp1/ sp2/
[stdout] sp1/students/yn95wpzm@cip.cs.fau.de: 3
[stderr] fopen: Permission denied
[stdout] sp1/staff/rabenstein@cs.fau.de: 2
[stdout] sp2/students/om79jtnd@cip.cs.fau.de: -1
[stdout] sp2/staff/nguyen@cs.fau.de: 1
```

Vorlage: `sp2/staff/nguyen@cs.fau.de`

```
Am $ ist SP2 Klausur.
```

Versendeter Text

```
Am 20. Juli 2021 ist SP2 Klausur.
```

Hinweise:

- `strrchr(3)` ermittelt die letzte Position eines vorgegebenen Zeichen in einer Zeichenkette.
- Die Schnittstelle des vorgegeben Semaphor-Moduls ist identisch zu der aus den Übungen.
- So lange noch eine Vorlage erfolgreich bearbeitet werden kann, darf sich `spam` nicht beenden.
- Der Dateideskriptor von `setup_connection` kann ein Socket sein.
- Die Funktionen auf einem `FILE`-Objekt sind thread-safe.
- Vorausdeklarationen der Funktionen sind nicht nötig.

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

```c
#include <stddef.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>
#define MAX_THREADS  42

/** Opaque type of a semaphore. */
typedef struct SEM SEM;

/* @brief Creates a new semaphore.
 * @param initVal The initial value of the semaphore.
 * @return Handle for the created semaphore, or NULL if an error
 *         occurred. */
SEM *semCreate(int initVal);

/* @brief Destroys a semaphore and frees all associated resources.
 * @param sem Handle of the semaphore to destroy. If a NULL pointer is
 *            passed, the implementation does nothing. */
void semDestroy(SEM *sem);

/* @brief P-operation.
 * @param sem Handle of the semaphore to decrement. */
void P(SEM *sem);

/* @brief V-operation.
 * @param sem Handle of the semaphore to increment. */
void V(SEM *sem);

/* @brief Start a connection
 * @param target String representation of the target.
 * @return File descriptor for the connection or -1 on error.
 *         If -1 is returned, errno is set appropriately. */
int start_connection(const char *target);

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

static void usage(const char *name) {
  fprintf(stderr, "USAGE: %s <replacement> <template-dir...>\n", name);
}
```



```c
// Globale Variablen und Definitionen

int main(int argc, char *argv[]) {
  // Globalen Zustand vorbereiten

  // Vorlagen suchen und Fäden starten

  // Finale Synchronisierung und Aufräumen

}
```

G:

M:

```
static void crawl(const char *dir) {

  // Über Verzeichniseinträge iterieren

    // Verzeichniseintrag prüfen
```



```
    // Arbeiterfaden mit Vorlage beauftragen

  // Aufräumen

}
```

C:

```
void *worker(void *arg) {

  // Anfrage absetzen

  // Statusmeldung

  // Ressourcenfreigabe

}
```

W:

```
int request(const char *target, const char *template) {
```



```
}
```

R:

```
int send_template(FILE *tx, FILE *template) {
```

```
}
```

T:

Aufgabe 3: Koordinierung (15 Punkte)

1) Zur Koordinierung von nebenläufigen Vorgängen, die auf gemeinsame Betriebsmittel zugreifen, unterscheidet man zwischen einseitiger und mehrseitiger Synchronisation. Wie unterscheidet sich einseitige von mehrseitiger Synchronisation? (2 Punkte)

2) Betriebsmittel lassen sich in zwei Kategorien einteilen. Nennen und beschreiben Sie diese und geben Sie je zwei Beispiele. (6 Punkte)

3) Statt blockierend zu synchronisieren kann in manchen Situationen mit optimistischen, nicht-blockierenden Synchronisationsverfahren gearbeiten werden. Beschreiben Sie, wie solch ein Verfahren grundsätzlich funktioniert. (3 Punkte)

4) Nennen Sie je zwei Vor- und Nachteile solcher nicht-blockierender Verfahren. (4 Punkte)

Aufgabe 4: Dateisystem (15 Punkte)

1) Beschreiben Sie kurz (in Stichworten) die grundsätzliche Funktionsweise eines *Journaling-File-Systems.* (3 Punkte)

---

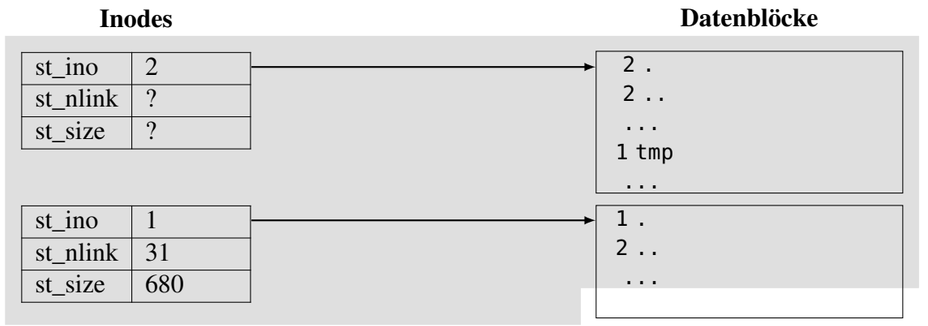
2) Gegeben ist die folgende Ausgabe des Kommandos `ls -aoRi /tmp/sp` auf einem Linux-System; eine rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `/tmp/sp` mit Angabe der Inode-Nummer (erste Spalte), des Referenzzählers (dritte Spalte) und der Dateigröße (fünfte Spalte). (12 Punkte)

```
dust@faui0sr0:~$ ls -aoRi /tmp/sp
/tmp/sp:
total 4
22 drwxr-xr-x  3 dust 100 Jul 16 17:32 .
 1 drwxrwxrwt 31 root 680 Jul 16 17:38 ..
31 lrwxrwxrwx  1 dust  13 Jul 16 17:29 file42 -> ./folder/narf
26 drwxr-xr-x  3 dust 100 Jul 16 17:57 folder
35 -rw-r--r--  2 dust 432 Jul 16 17:33 zorg

/tmp/sp/folder:
total 4
26 drwxr-xr-x 3 dust 100 Jul 16 17:57 .
22 drwxr-xr-x 3 dust 100 Jul 16 17:32 ..
34 drwxr-xr-x 2 dust  80 Jul 16 17:57 deep
32 -rw-r--r-- 1 dust 633 Jul 16 17:29 foo
30 lrwxrwxrwx 1 dust   9 Jul 16 17:27 nested -> ../file42

/tmp/sp/folder/deep:
total 12
34 drwxr-xr-x 2 dust   80 Jul 16 17:57 .
26 drwxr-xr-x 3 dust  100 Jul 16 17:57 ..
35 -rw-r--r-- 2 dust  432 Jul 16 17:33 narf
```

Ergänzen Sie im weißen Bereich die auf der folgenden Seite im grauen Bereich bereits angefangene Skizze der Inodes und Datenblöcke des Linux-Dateisystems um alle entsprechenden Informationen, die aus obiger Ausgabe entnommen werden können.



st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	