Aufgabe 1.1: Einfachauswahl-Fragen (22 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-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) Für welchen Zweck wird der Systemaufruf `listen()` benutzt? 2 Punkte

- ❏ Der Aufruf von `listen()` wartet solange an einem Socket, bis eine einkommende Verbindungsanfrage vorliegt.
- ❏ Der Aufruf von `listen()` erzeugt eine leere verkettete Liste, die zum Speichern von Daten verwendet werden kann.
- ❏ Mit `listen()` wird ein Socket für die Verbindungsannnahme vorbereitet. Ein Parameter gibt an, wieviele Verbindungsanfragen vor deren Annahme gepuffert werden können.
- ❏ Mit `listen()` wird ein Socket für die Verbindungsannahme vorbereitet. Ein Parameter gibt an, wieviele laufende Verbindungen maximal möglich sind.

b) Welche der genannten Attribute sind in einem Inode eines UNIX-Dateisystems gespeichert? 1 Punkt

- ❏ Dateityp, Eigentümer und Dateiname
- ❏ Gruppenzugehörigkeit, Anzahl der Verweise und bei Verzeichnissen zusätzlich die Anzahl der enthaltenen Unterverzeichnisse
- ❏ Eigentümer, Dateigröße und Dateityp
- ❏ Zeitpunkt des letzten Dateizugriffes, Erstellungszeitpunkt und aus Sicherheitsgründen der Zeitpunkt der letzten Rechteänderung.

c) In welcher der folgenden Situationen wird ein *laufender* Prozess in den Zustand *blockiert* überführt? 3 Punkte

- ❏ Ein Kindprozess des Prozesses terminiert.
- ❏ Der Prozess hat einen Seitenfehler für eine Seite, die bereits in den Freiseitenpuffer eingetragen, aber noch im Hauptspeicher vorhanden ist.
- ❏ Der Prozess ruft eine V-Operation auf einen Semaphor auf und der Semaphor hat gerade den Wert 0.
- ❏ Der Prozess greift lesend auf eine Datei zu und der entsprechende Datenblock ist noch nicht im Hauptspeicher vorhanden.

d) Welche Aussage zum Thema Ablaufplanung ist richtig? 2 Punkte

- ❏ Bei der FCFS-Strategie kann es aufgrund des Konvoieffekts zu hohen Antwortzeiten kommen.
- ❏ Offline-Einplanungsverfahren eignen sich vor allem für den Einsatz auf mobilen Geräten, da diese auch ohne Internetverbindung arbeiten können.
- ❏ In Echtzeitsystemen kommt es auf maximalen Durchsatz an, weshalb hier ausschließlich nicht-unterbrechbare Schedulingverfahren verwendet werden.
- ❏ Asymmetrische Einplanungsverfahren zielen auf eine optimale Behandlung von Prozessmengen, die sich in E/A- und CPU-Intensität stark voneinander unterscheiden.

e) Welche Aussage bezüglich der Freispeicherverwaltung mittels einer Bitliste ist richtig? 2 Punkte

- ❏ Der zu verwaltende Speicher wird in Speichereinheiten unterschiedlicher Größe unterteilt.
- ❏ Zur Suche nach freiem Speicher kann es nötig sein, die gesamte Bitliste zu durchsuchen.
- ❏ Das Zusammenfassen von benachbarten freien Speichereinheiten ist besonders aufwändig.
- ❏ Je feiner die Granularität der Speichereinheiten ist, desto kürzer ist die Bitliste.

f) Sie kennen den Begriff Seitenflattern (Thrashing). Welche Aussage ist richtig? 3 Punkte

- ❏ Als Seitenflattern bezeichnet man das wiederholte Löschen und Neuladen des Translation-Look-Aside-Buffer (TLB), ausgelöst durch häufigen Prozesswechsel.
- ❏ Als Seitenflattern bezeichnet man das wiederholte Einlagern einer erst vor kurzem verdrängten Speicherseite. Die Prozesse verbringen als Folge die meiste Zeit mit dem Warten auf die Behebung von Seitenfehlern.
- ❏ Seitenflattern erkennt man an der starken Geräuschentwicklung der Festplatte, da auf Grund häufiger Seitenzugriffe der Lesekopf ständig neu positioniert wird. Bei Systemen ohne Festplatte (z. B. Thin-Clients) kann das Seitenflattern nicht auftreten.
- ❏ Bei Verwendung der LRU-Seitenersetzungstrategie kann Seitenflattern prinzipbedingt nicht auftreten.

g) Welche Aussage zum Thema "Aktives Warten" ist richtig? 2 Punkte

- ❏ Aktives Warten vergeudet gegenüber passivem Warten immer CPU-Zeit
- ❏ Auf Mehrprozessorsystemen ist aktives Warten unproblematisch und deshalb dem passiven Warten immer vorzuziehen
- ❏ Aktives Warten sollte bei einer nicht-verdrängenden Scheduling-Strategie auf einem Monoprozessorsystem dem passiven Warten vorgezogen werden
- ❏ Zur Implementierung einer Schlossvariable mit aktivem Warten ist keine Unterstützung durch das Betriebssystem notwendig.

h) Nehmen Sie an, der Ihnen bekannte Systemaufruf stat(2) wäre analog zu der Funktion readdir(3) mit folgender Schnittstelle implementiert: 3 Punkte

```
struct stat *stat(const char *path);
```

Welche Aussage ist richtig?

- ❏ Ein Zugriff über den zurückgelieferten Zeiger liefert völlig zufällige Ergebnisse oder einen Segmentation fault.
- ❏ Der Systemaufruf liefert einen Zeiger zurück, über den die aufrufende Funktion direkt auf eine Datenstruktur zugreifen kann, welche die Dateiattribute enthält.
- ❏ Solch eine Schnittstelle ist nicht schön, da dadurch die aufrufende Funktion auf internen Speicher des Betriebssystems zugreifen könnte.
- ❏ Der Aufrufer muss sicherstellen, dass er den zurückgelieferten Speicher mit free(3) wieder freigibt, wenn er die Dateiattribute nicht mehr weiter benötigt.

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

- ❏ Zur Umschaltung von User-Threads verschiedener Prozesse ist kein Adressraumwechsel erforderlich.
- ❏ Kern-Threads teilen sich den kompletten Adressraum und verwenden daher den selben Stack.
- ❏ Auf Multiprozessorsystemen kann die Umschaltung von Kern-Threads ohne Mitwirken des Systemkerns erfolgen.
- ❏ Der Synchronisationsbedarf im Anwendungsprogramm kann von der Ablaufplanung der Kernfäden abhängen.

j) In einem UNIX-UFS-Dateisystem gibt es symbolische Namen/Verweise (Symbolic Links). Welche Aussage ist richtig? 2 Punkte

- ❏ Der Systemaufruf `stat()` liefert im Gegensatz zum Systemaufruf `lstat()` die Dateiattribute des symbolischen Verweises und nicht die Attribute vom Ziel des Verweises.
- ❏ Ein symbolischer Verweis kann ausschließlich auf reguläre Dateien verweisen.
- ❏ Beim Zugriff auf einen Symbolic Link kann ein “No such file or directory.”-Fehler auftreten (`errno==ENOENT`), obwohl der Symbolic Link existiert.
- ❏ In jedem Inode ist ein Referenzzähler gespeichert, welcher die Anzahl der Symbolic Links angibt, die auf ihn verweisen.

Aufgabe 1.2: Mehrfachauswahl-Fragen (8 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe sind jeweils m Aussagen angegeben, n ($0 \leq n \leq m$) Aussagen davon sind richtig. Kreuzen Sie **alle richtigen** Aussagen an. Jede korrekte Antwort in einer Teilaufgabe gibt einen halben Punkt, jede falsche Antwort einen halben 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 persistenter Datenspeicherung sind richtig? 4 Punkte

- ❍ Bei kontinuierlicher Speicherung von Daten ist es unter Umständen mit enormem Aufwand verbunden, eine bestehende Datei zu vergrößern.
- ❍ Bei verketteter Speicherung dauert der wahlfreie Zugriff auf eine bestimmte Dateiposition immer gleich lang, wenn Cachingeffekte außer Acht gelassen werden.
- ❍ Bei indizierter Speicherung von Dateien müssen unter Umständen mehrere Blöcke geladen werden, bevor der Dateiinhalt gelesen werden kann.
- ❍ Extents finden aus Performanzgründen keine Anwendung in modernen Dateisystemen.
- ❍ Journaling-Dateisysteme garantieren, dass auch nach einem Systemausfall alle Metadaten wieder in einen konsistenten Zustand gebracht werden können.
- ❍ Journaling-Dateisysteme sind immun gegen defekte Plattenblöcke.
- ❍ Durch den Einsatz von mehreren Platten wird bei RAID 0 die Ausfallsicherheit des Gesamtsystems so stark erhöht, dass Datensicherungen überflüssig sind.
- ❍ Festplatten eignen sich besser für sequentielle als für wahlfreie Zugriffsmuster.

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

- ❍ Kooperative Schedulingverfahren ermöglichen keinen wirksamen CPU-Schutz.
- ❍ Verdrängende Schedulingverfahren können nur mit Hilfe von Unterbrechungen realisiert werden.
- ❍ Asymmetrische Schedulingverfahren sind nur bei Systemen mit heterogenen Prozessoren (z. B. Grafikprozessoren und CPUs) einsetzbar.
- ❍ Symmetrische Schedulingverfahren sorgen für eine gleichmäßige Lastverteilung auf homogenen Multiprozessorsystemen.
- ❍ Online-Schedulingverfahren sind für den Einsatz in Rechnern ohne Netzwerkschnittstelle ungeeignet.
- ❍ Bei Offline-Schedulingverfahren wird der vollständige Ablaufplan bereits vor dem Start des Systems erstellt.
- ❍ Probabilistische Schedulingverfahren eignen sich für Systeme mit harten Echtzeitanforderungen.
- ❍ Deterministische Schedulingverfahren sind nur in der Theorie relevant, da die genaue Länge der CPU-Stöße nie vorhergesagt werden kann.

Aufgabe 2: msgpipe (60 Punkte)

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

Schreiben Sie ein Programm **`msgpipe`**, das über einen Netzwerk-Socket Nachrichten entgegennimmt und an den gewünschten Empfänger auf dem lokalen System ausliefert. Der Inhalt jeder Nachricht soll dabei durch ein vom jeweiligen Empfänger definiertes Filter-Programm verarbeitet werden.

`msgpipe` wird mit einem optionalen Parameter aufgerufen, der angibt, auf welchem Port Verbindungen angenommen werden sollen. Gibt der Aufrufer kein Befehlszeilenargument mit, soll standardmäßig auf Port 2012 gelauscht werden. Bei einer ungültigen Anzahl von Argumenten soll das Programm den Benutzer auf die korrekte Verwendung hinweisen und sich beenden.

Im Konfigurationsverzeichnis **`/etc/msgpipe`** liegt für jeden dem System bekannten Empfänger eine gleichnamige (d. h. Name des Empfängers) Datei, die eine Zeile mit dem Namen des von ihm gewünschten Filter-Programms (Maximallänge: 256 Zeichen) enthält. Existiert für einen Empfänger keine Konfigurationsdatei oder schlägt das Lesen der Datei fehl, wird die Nachricht abgewiesen.

Jedes Filter-Programm erwartet den Namen des Empfängers als Befehlszeilenargument und den Nachrichtentext auf dem Standardeingabe-Kanal.

Die Kommunikation läuft wie folgt ab: Pro Verbindung wird genau eine Nachricht übertragen. Der Client sendet zunächst eine Zeile mit dem Namen des Empfängers (Maximallänge: 256 Zeichen; es kann davon ausgegangen werden, dass der Name nur aus alphanumerischen Zeichen besteht).

Wird die Nachricht abgewiesen, so antwortet der Server mit der Zeile **`Rejected`** und beendet die Verbindung, andernfalls wird die Antwortzeile **`Accepted`** an den Client gesendet. Dieser schickt daraufhin einen beliebig langen Nachrichtentext, der durch das Filter-Programm bearbeitet wird.

Nach erfolgreicher Abarbeitung durch das Filter-Programm sendet der Server die Antwortzeile **`OK`** zurück, ansonsten **`Failed`**.

Das Programm soll im Detail folgendermaßen arbeiten:

- Das Hauptprogramm nimmt auf dem angegebenen Port Verbindungen an. Die Abarbeitung jeder Anfrage erfolgt innerhalb eines eigenen Prozesses durch Aufrufen der Funktion **`handleRequest()`**. Hierbei sollen nicht mehr als 16 Kindprozesse gleichzeitig aktiv sein. Ist die Maximalanzahl bereits erreicht und eine neue Verbindung wird akzeptiert, wird passiv gewartet, bis ein Kindprozess sich beendet hat. Falls währenddessen noch weitere Verbindungsanfragen von Clients eintreffen, sollen nicht mehr als 8 von ihnen zwischengespeichert werden - darüber hinausgehende werden abgewiesen.
- Funktion **`void handleRequest(FILE *client)`**: Führt die Anfragebearbeitung durch. Der Name des Empfängers wird gelesen, anschließend wird die zugehörige Datei aus dem Konfigurationsverzeichnis eingelesen und dann das Filter-Programm ausgeführt (Funktion **`runFilterApp()`**). Das Trennen der Verbindung ist Aufgabe des Aufrufers.
- Funktion **`int runFilterApp(const char program[], const char recipient[], FILE *client)`**: Startet einen Prozess, in dem das angegebene Programm mit dem Namen des Empfängers als Argument ausgeführt wird, und wartet auf seine Beendigung. Der Nachrichtentext, der vom Client kommt, wird in den Standardeingabe-Kanal des Filter-Programms geleitet. Hat sich der Prozess regulär mit Exit-Status 0 beendet, gibt die Funktion 0 zurück, sonst -1.

Auf den folgenden Seiten finden Sie ein Gerüst für das beschriebene Programm. In den Kommentaren sind nur die wesentlichen Aufgaben der einzelnen zu ergänzenden Programmteile beschrieben, um Ihnen eine gewisse Leitlinie zu geben. Es ist überall sehr großzügig Platz gelassen, damit Sie auch weitere notwendige Programmanweisungen entsprechend Ihrer Programmierung einfügen können.

Einige wichtige Manual-Seiten liegen bei - es kann aber durchaus sein, dass Sie bei Ihrer Lösung nicht alle diese Funktionen oder ggf. auch weitere Funktionen benötigen.


```
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>

// Konstanten, Funktions- und Strukturdeklarationen,
// globale Variablen
```

```
// Funktion main

    // Argumente auswerten
    // und weitere allgemeine Vorbereitungen
```

A:


```
//  Signalbehandlung und Signalmasken initialisieren
```

S:

```
// Socket erstellen und auf die
// Verbindungsannahme vorbereiten

// Verbindungen annehmen
```

O:


```
    // Warten, falls schon zu viele Kindprozesse laufen

    // Anfrage abarbeiten

// Ende Funktion main
```

C:

```
// Signalbehandlung SIGCHLD
```



```
// Funktion handleRequest()
```

H:

```
// Funktion runFilterApp()
```

R:

Aufgabe 3: (18 Punkte)

a) Erläutern Sie das Synchronisationskonzept "Monitor" am Beispiel der Hansen-/ Mesa-Monitore und gehen Sie hierbei auf die folgenden Fragen ein:
   a1) Worin besteht der Unterschied zu den pthread-Synchronisationsmechanismen?
   a2) Welche Synchronisationsmechanismen gehören zu einem Monitor und wie funktionieren sie im Detail?
   a3) Welche Art von Synchronisation in Bezug auf die Wirkung zwischen den Threads liegt bei diesen Synchronisationmechanismen vor? Bitte mit kurzer Erläuterung!
   (14 Punkte)

b) Was versteht man unter dem *lost wakeup*-Problem?
   (4 Punkte)

Aufgabe 4: (12 Punkte)

a) Bei virtuellem Speicher kann es zu Seitenfehlern (*page faults*) kommen. In welchen Situationen tritt ein Seitenfehler auf und was muss das Betriebssystem jeweils zur Behebung tun (beschreiben Sie drei solche Situationen)? (6 Punkte)

b) Falls kein freier Seitenrahmen im Hauptspeicher verfügbar ist, muss eine Seite ausgelagert werden. Eine mögliche Strategie zur Bestimmung der zu verdrängenden Seite ist LRU. Beschreiben Sie diese Strategie kurz und erläutern Sie, welche Probleme bei der Realisierung bestehen. (2 Punkte)

c) Eine in der Praxis gut einsetzbare Approximation dieser Strategie ist Second Chance (oder Clock). Beschreiben Sie die Funktionalität dieser Strategie und welche Unterstützung durch die MMU dafür erforderlich ist. (4 Punkte)