Name: Matrikel-Nr.: Studiengang:

Klausur Betriebssysteme und Sicherheit, 13. 02. 2019

1	2	3	4	5	6	$\sum$
12	15	14	15	16	18	90

Alle Aussagen sind so ausführlich wie nötig, aber so knapp wie möglich zu begründen.

Aufgabe 1 – Synchronisation

12 Punkte

a) Nennen Sie die Anforderungen, die an ein Lock gestellt werden. **2 P**

b) Gegeben ist die unten stehende Implementierung eines Locks. Welche der Anforderungen erfüllt die gegebene Implementierung nicht? Beschreiben Sie ein Szenario, in dem die Implementierung fehlerhaft arbeitet. **3 P**

```
1  int lck = 0; // globale Variable
2
3  void lock() {
4      while (lck == 1) {
5      }
6      lck = 1;
7  }
8
9  void unlock() {
10     lck = 0;
11 }
```

c) Das Primitiv `cmpxchg(&var, alt, neu)` vergleicht zunächst `var` und `alt`. Nur wenn sie gleich sind, wird der Wert von `neu` an `var` zugewiesen. In jedem Fall gibt die Funktion den alten Wert von `var` zurück.

Nutzen Sie `cmpxchg`, um obige Implementierung zu korrigieren. Welche Eigenschaft muss `cmpxchg` dabei aufweisen? **4 P**

d) Welcher Synchronisationsmechanismus wird vom Betriebssystem angeboten und kann statt eines Locks verwendet werden? Nennen Sie jeweils einen Vor- und Nachteil gegenüber einem einfachen Lock. **3 P**

Aufgabe 2 – Scheduling

15 Punkte

a) Kreuzen Sie für jede der folgenden Aussagen an, ob diese richtig (R) oder falsch (F) ist. **5 P**

Ⓡ Ⓕ Bei RMS existiert ein ausführbarer Ablaufplan genau dann, wenn für die Auslastung $U \leq n(\sqrt[n]{2} - 1)$ gilt.

Ⓡ Ⓕ Es existieren Task-Mengen mit $U = 1$, die sich mittels RMS einplanen lassen und dabei alle Zeitschranken einhalten.

Ⓡ Ⓕ LPT minimiert die Gesamtbearbeitungszeit in Systemen mit mehreren Prozessoren.

Ⓡ Ⓕ Mindestens einer der Algorithmen FIFO, LIFO oder SPT ist optimal bezüglich Minimierung der Gesamtbearbeitungszeit in Ein-Prozessor-Systemen.

Ⓡ Ⓕ Jobsysteme mit unabhängigen Jobs lassen sich mittels SPT optimal bezüglich Minimierung der mittleren Verweildauer einplanen.

b) Gegeben sei folgende Jobmenge mit entsprechenden Ausführungszeiten: $\{A : 3, B : 2, C : 1, D : 5, E : 1\}$

Planen Sie diese Jobmenge jeweils mit SPT und LPT auf einem Zwei-Prozessor-System ein. Nennen Sie die Optimierungsziele der beiden Algorithmen. **4 P**

c) Planen Sie die folgenden periodischen Echtzeit-Tasks (Periodendauer p, Ausführungszeit e, relative Zeitschranke d = p) mittels EDF auf einem Prozessor grafisch ein. Zeichnen Sie dazu das Gantt-Diagramm über 20 Zeiteinheiten. Alle Tasks sind zum Zeitpunkt $t = 0$ bereit, voneinander unabhängig und jederzeit unterbrechbar. Der Scheduling-Overhead wird vernachlässigt. **4 P**

	A	B	C	D
p	8	5	6	3
e	2	1	2	1

d) Welche Schlussfolgerung bezüglich der Einplanbarkeit der Task-Menge können Sie aus c) ableiten? **1 P**

e) Ist die Task-Menge aus c) mittels RMS einplanbar? **1 P**

Aufgabe 3 – Unix

14 Punkte

a) Threads können sich in einem von mehreren Zuständen befinden, u.a. *aktiv*, *bereit* oder *blockiert*. Nennen Sie jeweils ein Ereignis, das zu den genannten Zustandsübergängen führt. **3 P**

- aktiv → blockiert
- aktiv → bereit
- bereit → aktiv

b) Nennen Sie ein Signal in Unix-Systemen, bei dem keine Abweichung von der Standardreaktion zugelassen ist und einen Grund dafür, dass es nicht sinnvoll wäre, eine andere Reaktion zuzulassen. **2 P**

Der nebenstehende C-Code soll die folgenden Vorgaben umsetzen: Der Hauptprozess P_1 startet einen neuen Prozess P_2, der seinerseits Prozess P_3 startet. P_3 schreibt seine Ergebnisse in eine neu angelegte Datei *buffer*, die später wieder entfernt wird. Diese Ergebnisse werden anschließend von Prozess P_1 eingelesen und weiter verarbeitet. Außerdem werden die PIDs der Prozesse P_2 und P_3 ausgegeben.

Hinweis: Eine Überprüfung der Rückgabewerte der genutzen Funktionen findet zu Gunsten der Übersichtlichkeit nicht statt; Funktionsaufrufe sind *stets erfolgreich*.

Die Funktion `wait` blockiert den aufrufenden Prozess so lange, bis sich einer seiner Kindprozesse beendet. Existiert kein solcher, so kehrt die Funktion sofort erfolgreich zurück.

```
1  int fd, pid2, pid3;
2  char result[1337];
3
4  fd = creat("buffer");
5  pid2 = fork();
6
7  if (pid2 == 0) {
8     pid3 = fork();
9     if (pid3 == 0) {
10       produce(result);
11       write(fd, result, 1337);
12       unlink("buffer");
13    }
14 }
15 else {
16    printf("PID von P2: %d, PID von P3: %d",
        ↪ pid2, pid3);
17    wait();
18    read(fd, result, 1337);
19    consume(result);
20 }
```

c) Welche drei (logischen) Fehler enthält der Code? Erläutern Sie jeweils kurz das Problem. **6 P**

d) In obigem Beispiel wurde eine Datei genutzt, um größere Datenmengen zwischen zwei Prozessen auszutauschen. Nennen sie eine mögliche Alternative, sowie je einen Vor- und Nachteil gegenüber der Nutzung einer Datei für diesen Zweck. **3 P**

Aufgabe 4 – Sicherheit **15 Punkte**

Alice, Bob und Charlie leben in einer Wohngemeinschaft. Für ihr gemeinsames Filmarchiv betreiben die drei einen Server, der ein gemeinsam genutztes Dateisystem bereitstellt. Dabei sollen folgende Anforderungen realisiert werden:

- Jeder Nutzer verwaltet eigene Filme in einem privaten Verzeichnis (`/alice`, `/bob`, `/charlie`).
- Alle Bewohner der WG können in einem gemeinsamen Verzeichnis `/shared` Filme austauschen.
- In einem weiteren Verzeichnis `/guests` ist der Austausch von Dateien mit Gästen möglich.

a) Geben Sie eine Lösung mit Hilfe des klassischen Unix-Rechtemodells an. Nennen Sie für jedes der genannten fünf Verzeichnisse dessen Eigentümer, Gruppe und Rechte. Geben Sie für verwendete Gruppen deren Mitglieder an. **4 P**

b) Um Speicherplatz zu sparen, kann jeder Nutzer ein Programm `dedup` ausführen, welches in allen Verzeichnissen nach mehrfach gespeicherten Filmen sucht. Dabei sollen auch Verzeichnisse untersucht werden, auf die der Nutzer im Normalfall keinen Zugriff hat. Geben Sie Unix-Rechte für `dedup` an, die diese Ausführung ermöglicht. **3 P**

c) Sobald `dedup` fündig wird, soll die Speicherung des mehrfach vorhandenen Films so verändert werden, dass die Datei weiterhin an mehreren Stellen im Dateisystem gefunden wird, aber nur einmal Speicherplatz belegt. Nennen Sie eine Lösung, die Unix-Dateisysteme dafür anbieten. **1 P**

d) Das Dateisystem mit dem Filmarchiv wird von einem Server über Netzwerk bereitgestellt. Dabei wird das NFS-Protokoll als entferntes Dateisystem verwendet. Welches Problem ergibt sich dadurch hinsichtlich der Durchsetzung der ACLs? **2 P**

e) Schlagen Sie eine Lösung vor, die dieses Problem beseitigt. Dabei können Sie ein aus der Vorlesung bekanntes Protokoll benennen oder einen eigenen Lösungsvorschlag entwerfen. **2 P**

f) Beschreiben Sie kurz symmetrische und asymmetrische Verschlüsselung. Was ist hybride Verschlüsselung und warum wird diese benutzt? **3 P**

Aufgabe 5 – Dateisysteme **16 Punkte**

Ein Unix-artiges Dateisystem organisiert alle Dateiinhalte und Metadaten in Blöcken auf dem Speichermedium. Die Größe eines Blocks beträgt 4 KiB.

a) Wie viele Blöcke werden benötigt, um den Inhalt einer 111 KiB großen Datei zu speichern? **1 P**

b) In jedem Inode des Dateisystems ist Speicherplatz für bis zu 15 direkte Blockzeiger vorhanden. Erläutern Sie, wie ein solches Dateisystem dennoch mehr als 15 Datenblöcke pro Datei adressieren kann. Welche zusätzlichen Metadaten sind dafür erforderlich und wo werden diese gespeichert? **3 P**

Gegeben ist der nachfolgende Ausschnitt aus dem Dateisystemzustand auf einem Speichermedium. Die dargestellten Blöcke enthalten entweder Inodes (I), Verzeichniseinträge (V) oder eine Allokations-Bitmap für Blöcke (BA) bzw. Inodes (IA). Weitere Blöcke (z. B. für Dateiinhalte) existieren, sind aber nicht dargestellt.

Block	0	1	2	3	4	...
Typ	SB	BA	IA	I	V	...
Inhalt	/ → 0	0,1,2,3, 4,5,6,7	0,1,2	$I_0 \to V$ | 48 | 4 $I_1 \to D$ | 11.000 | 5, 6, 7 $I_2 \to D$ | 13 | 7 $I_3 \to D$ | 4.096 | 8	`"A"` → I_1 `"tmp"` → I_2 `"X"` → I_3	...

Inode-Verweis für Wurzelverzeichnis

Blöcke 0 bis 7 belegt

Inodes 0 bis 2 belegt

Typ | Größe (in Byte) | Blockzeiger für Inodes

Abbildung von Dateinamen auf zugehörige Inodes

c) Die im Schema dargestellten Metadatenstrukturen sind durch einen Systemabsturz inkonsistent geworden. Worin genau bestehen die vorhandenen Inkonsistenzen? Erläutern Sie, welche Probleme auftreten können, wenn das Betriebssystem diese Datenstrukturen ohne vorherige Reparatur für zukünftige Lese- und Schreiboperationen konsultiert und/oder verändert. **6 P**

Die Entwickler des Dateisystems haben das Journaling-Verfahren in ihre Implementierung integriert.

d) Wie kann es trotz Anwendung des Journaling-Verfahrens passieren, dass auf dem Speichermedium Inkonsistenzen (ähnlich wie im Schema zu Teilaufgabe c) vorliegen? Erläutern Sie, wie das Betriebssystem diese Inkonsistenzen wieder korrigiert, woher die dazu nötigen Informationen kommen und wann diese Korrektur stattfindet. **4 P**

e) Ein Nutzer ist besorgt, weil er gehört hat, dass „Journaling ein Dateisystem langsamer macht“. Was kann ein informierter Entwickler antworten, um diese Sorge zu entkräften bzw. inwiefern ist sie begründet? **2 P**

Aufgabe 6 – Speicherverwaltung **18 Punkte**

Ein 48-Bit-System verwende eine dreistufige Adressübersetzung, wobei die Indizes für jede der drei Stufen jeweils 12 Bit groß seien: $\boxed{\text{Idx}_1}\boxed{\text{Idx}_2}\boxed{\text{Idx}_3}\boxed{\text{Offset}}$. Die Tabellen aller Stufen sind gleich groß und füllen jeweils fünf Rahmen. Die Rechte werden mittels eines *present*-Bits (p) und eines *writable*-Bits (w) repräsentiert, die bei einem Zugriff auf allen drei Stufen überprüft werden.

a) Wie viele Bit bilden den Offset? **1 P**

b) Wie viele Byte benötigt ein Eintrag in einer Seitentabelle der zweiten Stufe (Idx_2)? **1 P**

c) Der virtuelle Adressraum eines Prozesses ist vollständig belegt und alle Seiten sind im physischen Speicher eingelagert. Wie viele Rahmen verwendet das vorliegende System für die Seitentabellen (aller Stufen) dieses Adressraums? **2 P**

Nach dem Start eines neuen Prozesses mit leerem Adressraum werden die folgenden Speicherzugriffe ausgeführt:

- Lesend auf `0x123456 123456`
- Schreibend auf `0xABCDEF FEDCBA`
- Lesend auf `0x123456 12345A`
- Lesend auf `0xABCDEF FEDFAB`

Alle Seiten liegen dabei in Regionen, auf die der Prozess die erforderlichen Zugriffsrechte hat, d.h. alle Zugriffe sollen gültig sein. Dem System stehen freie Rahmen an den Adressen `0x081508 150000`, `0x007007 000000`, `0x802860 000000` und `0xA57A53 000000` zur Verfügung.

d) Welchen Zustand hat das System nach diesen Zugriffen? Tragen Sie im nachstehenden Schema die entsprechenden Zeilenindizes, Rahmennummern sowie Rechtebits ein. Der Einsprungpunkt für die Adressübersetzung, also die Tabelle der ersten Stufe, liegt an Adresse `0x10000`. Vier weitere Tabellen stehen an den im Schema angegebenen Adressen bereit. **14 P**