Name: Matrikel-Nr.: Studiengang:

Klausur Betriebssysteme und Sicherheit, 02.08.2017

1	2	3	4	5	6	$\sum$
7	7	6	8	7	7	42

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

Aufgabe 1 – Dateisysteme

7 Punkte

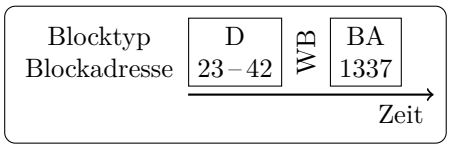
Ein Unix-artiges Dateisystem organisiert alle Dateiinhalte und Verwaltungsstrukturen in Blöcken auf dem Speichermedium. Jeder Block enthält entweder Daten (D) oder eine der üblichen Arten von Metadaten: Inodes (I), Verzeichniseinträge (V), Allokations-Bitmap für Inodes bzw. Blöcke (IA bzw. BA). Zur Leistungssteigerung puffert das Betriebssystem modifizierte Blöcke zunächst im Arbeitsspeicher.

Um die neu erzeugte Datei `kernel.img` persistent zu speichern, müssen die nebenstehenden 135 Blöcke an die angegebenen Blockadressen auf das Speichermedium geschrieben werden.

IA → 1
I → 32, 34
BA → 65, 66
V → 260
D → 400 – 528

a) Das Dateisystem verwendet ein Journal, für das 32 Blöcke mit den Blocknummern 128 – 159 fest reserviert sind. Transaktionen werden durch einen speziellen Block vom Typ T abgeschlossen.

Tragen Sie auf der untenstehenden Zeitachse eine gültige Sequenz von Schreibzugriffen und Write-Barriers (WB) ein, so dass alle oben angegebenen Blöcke *mit einer Transaktion* gemäß dem Journaling-Verfahren sicher auf das Speichermedium zurückgeschrieben werden! Nutzen Sie dazu eine Notation gemäß dem nebenstehenden Beispiel.



Hinweis: Wir nehmen an, dass das Journal leer ist und beginnend bei Block 128 beschrieben werden kann. Das etwaige Löschen von Journal-Transaktionen wird nicht betrachtet. **4 P**

Zeit

b) Wir nehmen an, das System stürzt während der in a) ausgeführten Schreibsequenz ab. Wovon hängt es ab, ob die Datei `kernel.img` nach einem Neustart gelesen werden kann und welche Aussagen lassen sich über den Inhalt der Datei treffen? Erläutern Sie hierzu die möglichen Fälle, die beim Wiedereinhängen des Dateisystems gemäß dem Journaling-Verfahren eintreten können! **3 P**

Aufgabe 2 – Synchronisation **7 Punkte**

Zur Absicherung von sehr kurzen kritischen Abschnitten nutzt ein Programmierer die folgende Implementierung eines Locks. Dabei sind lediglich Sicherheit und Verklemmungsfreiheit von Bedeutung.

```
1 void lock() {
2    while (true) {
3       if (xadd(&lck, 1) == 0) {
4          break;
5       } else {
6          while (lck != 0) {}
7       }
8    }
9 }
```

```
1 void unlock() {
2    xadd(&lck, -1);
3 }
```

Am Beginn des Programms wird die Lock-Variable wie folgt initalisiert:

```
1 int lck = 0;
```

HINWEIS: Die verwendete Funktion `xadd(&a, i)` erhöht *atomar* den Wert der Variable `a` um den Wert `i` und gibt den Wert zurück, der *zuvor* in der Variable `a` gespeichert war.

a) Während eines Code-Reviews des Programms wird die verwendete Lock-Implementierung als fehlerhaft angekreidet. Begründen Sie wieso der Reviewer Recht hat und zeigen Sie anhand eines Beispiels wann und wie sich das Problem manifestiert! **2 P**

Der Reviewer schlägt zur Lösung des Problems die nebenstehende alternative Implementierung der `unlock`-Funktion vor.

```
1 void unlock() {
2    xchg(&lck, 0);
3 }
```

HINWEIS: Die verwendete Funktion `xchg(&a, n)` ändert *atomar* den Wert der Variablen `a` zu dem Wert `n` und gibt den ursprünglichen Wert der Variablen `a` zurück.

b) Führt die Umsetzung dieses Änderungsvorschlags unter Beibehaltung der `lock`-Funktion zu einer korrekten Implementierung des Locks? Begründen Sie Ihre Antwort! **2 P**

Statt dem Vorschlag des Reviewers zu folgen, implementiert der Programmierer die folgende neue `lock`-Funktion und nutzt diese in Kombination mit der ursprünglichen `unlock`-Funktion.

```
1 void lock() {
2    while (true) {
3       while (lck != 0) {}
4
5       if (xchg(&lck, 1) == 0) {
6          break;
7       }
8    }
9 }
```

```
1 void unlock() {
2    xadd(&lck, -1);
3 }
```

Hinweis: Die verwendete Funktion `xchg(&a, n)` ändert *atomar* den Wert der Variablen `a` zu dem Wert `n` und gibt den ursprünglichen Wert der Variablen `a` zurück.

Die verwendete Funktion `xadd(&a, i)` erhöht *atomar* den Wert der Variable `a` um den Wert `i` und gibt den Wert zurück, der *zuvor* in der Variable `a` gespeichert war.

c) Ist diese Implementierung des Locks korrekt? Begründen Sie Ihre Antwort! **2 P**

In seiner Freizeit liest der Programmierer einen Artikel[1] von Dijkstra, in dem dieser das Konzept und die Funktionsweise von Semaphoren vorstellt. Aufgrund dieses Wissens entfernt der Programmierer seine Lock-Implementierung wieder aus seinem Programm und verwendet stattdessen einen Semaphor, um seine sehr kurzen kritischen Abschnitte abzusichern.

d) Ist diese Verwendung eines Semaphors anstelle eines einfachen Locks vorteilhaft? Begründen Sie Ihre Antwort! **1 P**

[1] Edsger Wybe Dijkstra. „Over de sequentialiteit van procesbeschrijvingen“. URL: `https://www.cs.utexas.edu/users/EWD/ewd00xx/EWD35.PDF`.

Aufgabe 3 – Sicherheit **6 Punkte**

Alice und Bob kommunizieren über ein verschlüsselndes Chatsystem. Dafür haben beide vor Beginn der Kommunikation asymmetrische Schlüsselpaare erstellt und die geeigneten Teilschlüssel sicher miteinander ausgetauscht. Im laufenden Chat wird jede Nachricht sowohl verschlüsselt als auch signiert.

a) Sind die folgenden Aussagen korrekt? *Falls ja, genügt eine einfache Angabe ohne Begründung.* Falsche Aussagen sind zu inhaltlich entsprechenden, sinnvollen Aussagen zu berichtigen.

HINWEIS: Bei Korrekturen von falschen Aussagen genügt es Teile der vorgegebenen Sätze zu streichen bzw. zu ergänzen. Eine einfache Negation der Aussage ist jedoch nicht zulässig. **4 P**

- Alice und Bob verwenden jeweils das Verfahren AES für die Erstellung der Schlüsselpaare.
- Alice hat vor Beginn der Kommunikation Bob ihren privaten Schlüssel zur Verfügung gestellt.
- Beim Schlüsselaustausch musste auf mögliche Man-in-the-Middle-Angriffe geachtet werden.
- Ein passiver Angreifer kann den Klartext-Inhalt der verschlüsselten Chat-Nachrichten nicht lesen.
- Ein passiver Angreifer erhält keinerlei Informationen über den Chatverlauf.
- Durch Prüfen der Signatur kann Alice sicher erkennen, dass eine Chatnachricht von Bob stammt.
- Gelangt ein Angreifer an den geheimen Schlüssel von Alice, so kann er damit alle zukünftigen und vergangenen Nachrichten entschlüsseln, die an Alice gerichtet sind.
- Besitzt ein Angreifer lediglich die öffentlichen Schlüssel, so kann er unter keinen Umständen die Vertraulichkeit des Chatsystems brechen.

b) Alice und Bob möchten über das Chatsystem auch Dateien austauschen, die potenziell mehrere Gigabyte groß sind. Schlagen Sie eine geeignete Erweiterung des Chatsystems vor, die dies effizient ermöglicht und begründen Sie ihren Vorschlag! **2 P**

Aufgabe 4 – Echtzeit

8 Punkte

HINWEIS: Am Ende der Seite steht ein karierter Bereich zum Zeichnen von Ablaufplänen *für alle Teilaufgaben* zur Verfügung. Bitte notieren Sie dabei jeweils zu welcher Teilaufgabe die Zeichnung gehört.

Ein um die Erde kreisender Satellit bestimmt seine Position mittels dreier verschiedener Lagesensoren (Sternensensor A, Magnetometer B, Gyroskop C). Zur genauen Lageregelung müssen diese periodisch ausgewertet werden, um die Antriebe entsprechend zu steuern. Die notwendigen Perioden p und Ausführungszeiten e der Auswertungsalgorithmen sind in nebenstehender Tabelle zusammengefasst. Die relative Deadline d ist gleich der Periodenlänge ($d = p$). Alle Tasks sind voneinander unabhängig und laufende Jobs zu beliebiger Zeit unterbrechbar.

	A	B	C
p	9	6	3
e	3	2	1

a) Ist die Lageregelung auf einem Prozessor mittels dynamischer Prioritäten möglich, falls der Scheduling-Overhead vernachlässigbar ist? Wenn nein, begründen Sie. Falls ja, schlagen Sie einen Algorithmus für die Einplanung vor. **2 P**

b) Während der Entwicklung der Software wird festgestellt, dass der Scheduling-Overhead für das Einplanen eines neuen Jobs in einem System mit dynamischen Prioritäten 0,2 Zeiteinheiten beträgt. Ist die Taskmenge weiterhin auf einem Prozessor einplanbar? **1 P**

c) Existiert eine statische Prioritätszuordnung mit welcher die Taskmenge auf einem Prozessor eingeplant werden kann? Falls ja, geben Sie diese Zuordnung an! Ansonsten begründen Sie Ihre Aussage!

HINWEIS: Bei statischen Prioritäten ist der Scheduling-Overhead vernachlässigbar. **2 P**

Als Lösung schlägt der System-Designer vor, den Sternensensor (A) durch einen Sonnensensor (A′) zu ersetzen. Dessen Auswertungsalgorithmus hat eine Periode von 4 und eine Ausführungszeit von 1 Zeiteinheiten. Das System besteht damit jetzt aus den nebenstehenden Tasks.

	A′	B	C
p	4	6	3
e	1	2	1

d) Ist dieses System mittels statischer Prioritäten einplanbar? Falls ja, nennen Sie eine Prioritätszuteilung! Ansonsten begründen Sie Ihre Aussage!

HINWEIS: Bei statischen Prioritäten ist der Scheduling-Overhead vernachlässigbar. **2 P**

e) Wäre das System mittels dynamischer Prioritäten einplanbar, wenn hier weiterhin von einem Overhead für den Jobwechsel von 0,2 Zeiteinheiten ausgegangen wird? **1 P**

Aufgabe 5 – Virtueller Speicher **7 Punkte**

In einer 32-Bit-Architektur werden zweistufige Seitentabellen zum Übersetzen von virtuellen in physische Adressen verwendet. Die Indizes sind jeweils 10 Bit lang; die letzten 12 Bit bilden den Offset in die Seite. Jeder Tabelleneintrag enthält zusätzlich ein *present bit* (`p`) und ein *writeable bit* (`w`). Bei jedem Zugriff werden die Zugriffsrechte in beiden Stufen überprüft.

Ein Prozess A verwende die folgenden Seitentabellen; alle nicht genannten Einträge werden als ungültig angenommen. Die Tabelle der ersten Stufe, das Seitenverzeichnis, liegt dabei an der Adresse `0x00104000`.

Addr.: `0x00104000`

0x0	0x4D800000	pw
0x1	0x4D801000	p
0x2	0x4D800000	w
0x3	0xCC432000	pw

Addr.: `0x4D800000`

0x0	0x59083000	
0x1	0x4D801000	pw
0x2	0x84156000	p

Addr.: `0x4D801000`

0x0	0xC5635000	pw
0x1	0x688C0000	pw
0x2	0xF190A000	p

Addr.: `0xCC432000`

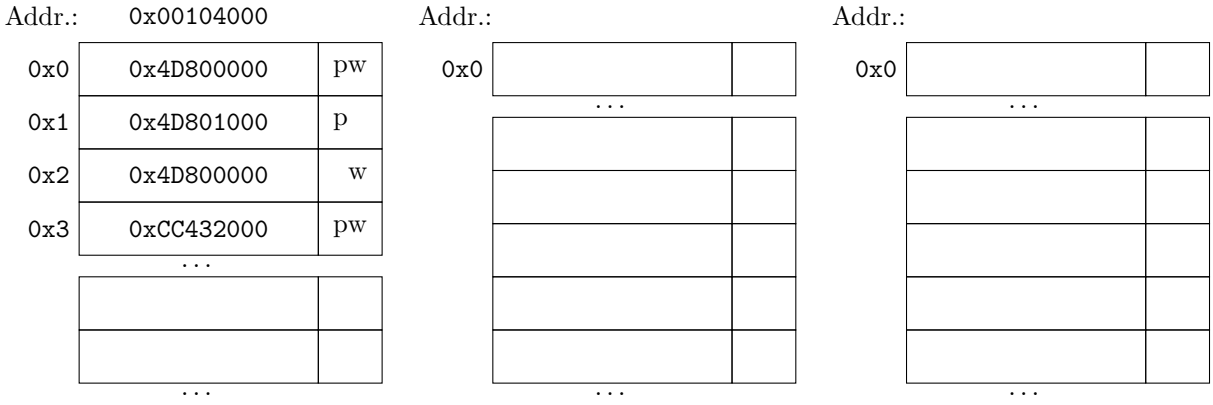
0x0	0x59083000	p
0x1	0xF190A000	pw
0x2	0x1E817000	p

a) Der Prozess zieht die folgenden Zugriffe in Betracht. Welche würden gelingen, was wäre die physische Adresse bzw. der Grund für das Scheitern des Zugriffs? **2 P**

- Lesend auf `0x00402003`:
- Schreibend auf `0x00800234`:
- Schreibend auf `0x00C00F00`:
- Lesend auf `0x00002DDC`:

b) Durch einen Systemaufruf hat der Prozess lesenden Zugriff auf die physischen Rahmen `0x09F91000` und `0x1029D000` im virtuellen Speicherbereich `0x44374000` – `0x44375FFF` bekommen. Diese Abbildung ist allerdings noch nicht in den Seitentabellen eingetragen. Was passiert beim ersten Lesezugriff in diesem Bereich? Welche Schritte folgen in einem UNIX-artigen Betriebssystem um diese Situation aufzulösen? **2 P**

c) Ergänzen Sie die Einträge für die zusätzlichen Seiten aus b) in den folgenden Tabellen. Für eine zusätzliche Seitentabelle steht ggf. der Rahmen `0xE35BD000` zur Verfügung. **2 P**



d) Ein weiterer Prozess (B) verwendet den physischen Speicher `0x08150000` – `0x0815FFFF`. Wie kann Prozess A unter Ausnutzung der gegebenen Seitentabellen Zugriff auf den physischen Speicher von Prozess B erlangen? **1 P**

Aufgabe 6 – Unix

7 Punkte

Betrachtet werde das folgende Programm:

```
1  int fd1, fd2, pid1, pid2;
2
3  fd1  = creat("foo.txt");
4  pid1 = fork();
5
6  if (pid1 == 0) {
7     fd2  = creat("bar.txt");
8     write(fd1, "Lorem", 5);
9
10    pid2 = fork();
11    if (pid2 == 0) {
12       write(fd2, "Ipsum", 5);
13       wait();
14    }
15    exit();
16 }
17
18 printf("X");
19 wait();
20 exit();
```

HINWEIS: Die Funktion...

`creat` legt die angegebene Datei an und öffnet sie. Sollte die Datei bereits existieren, wird ihr Inhalt beim Öffnen vollständig gelöscht.

`wait` blockiert den aufrufenden Prozess so lange bis sich ein beliebiger Kindprozess beendet. Existiert kein solcher, so kehrt die Funktion sofort erfolgreich zurück.

`exit` beendet den aufrufenden Prozess sofort.

Ein Nutzer führt das Programm nun aus. Alle auftretenden Funktionsaufrufe sind dabei erfolgreich.

a) Was wurde auf der Konsole ausgegeben unmittelbar nachdem sich der Hauptprozess beendet hat? Skizzieren Sie kurz wie es zu dieser Ausgabe kommt. Sind auch andere Ausgaben möglich? Falls ja, geben Sie die Ursache dafür an und nennen Sie eine zweite mögliche Ausgabe! **2 P**

b) Welchen Inhalt haben die beiden Dateien `foo.txt` und `bar.txt` unmittelbar nachdem sich der Hauptprozess beendet hat? Gibt es mehrere Möglichkeiten, so geben Sie die Ursache dafür an und nennen Sie einen zweiten möglichen Dateiinhalt! **3 P**

c) Durch das Entfernen einer einzelnen Zeile des obigen Codes lässt sich erreichen, dass beim Beenden des Hauptprozesses *immer genau drei* `X` ausgegeben wurden. Um welche Zeile (Angabe der Zeilennummer genügt) handelt es sich? Erläutern Sie kurz wieso damit das beschriebene Ergebnis erreicht wird! **1 P**

d) Welches Ergebnis hätte ein Aufruf von `read(fd1, buf, 10)` – also das Einlesen von bis zu 10 Byte aus `fd1` in den (zuvor angelegten) Puffer `buf` – unmittelbar vor Zeile 20? **1 P**