Name: Matrikel-Nr.: Studiengang:

Klausur Betriebssysteme und Sicherheit, 25.02.2015

1	2	3	4	5	6	$\sum$
6	6	9	8	6	7	42

Aufgabe 1 – Speicher I

6 Punkte

Ein System verwaltet 5 Rahmen mit Hilfe des Arbeitsmengenmodells. Zu Beginn der Ausführung sind alle Rahmen frei. Im System existieren 3 Prozesse (A,B,C), die Fenstergröße beträgt 3 Seiten und die Seitenreferenzfolge lautet:

```
A:3 B:5 B:2 B:3 C:1 B:3 C:2 C:2 B:4
```

a) Vervollständigen Sie die Zuordnung der Seiten zu Rahmen in der folgenden Tabelle! Markieren Sie prozess-lokale Ersetzungen durch Umkreisen, globale durch Unterstreichen. Tragen Sie bei Seitenfehlern ein Kreuz in der Zeile *PF* ein. **4 P**

Rahmen	A:3	B:5	B:2	B:3	C:1	B:3	C:2	C:2	B:4
1									
2									
3									
4									
5									
PF									

b) Welche Seiten enthält die Arbeitsmenge von B nach der letzten Referenz? **1 P**

c) Was passiert, wenn A im Anschluss noch auf Seite 2 zugreifen will? **1 P**

Aufgabe 2 – Echtzeit **6 Punkte**

In einem Echtzeitsystem ist eine Menge von drei periodischen Tasks so einzuplanen, dass deren Jobs in jeder Periode erfolgreich beendet werden. Das Periodenende entspricht der Deadline. Die Parameter der Tasks beschreiben jeweils die Periode p und die konstante Bearbeitungszeit e der Jobs:

T_1: $p_1 = 4$, $e_1 = 1$
T_2: $p_2 = 5$, $e_2 = 1$
T_3: $p_3 = 10$, $e_3 = 2$

Den Tasks werden statisch Prioritäten zugeteilt, so dass T_1 die höchste, T_2 eine mittlere und T_3 die niedrigste Priorität erhält. Alle Tasks starten zum Zeitpunkt 0 und können an beliebiger Stelle unterbrochen werden. Der Scheduling-Overhead werde vernachlässigt und es stehe genau ein Prozessor zur Verfügung.

a) Ist die gegebene Taskmenge unter den genannten Bedingungen einplanbar, wenn zwischen den Tasks keine Abhängigkeiten bestehen? **2 P**

b) Eine zusätzliche Task T_4 mit $p_4 = 3$ und $e_4 = 1$ soll ebenfalls eingeplant werden. Diskutieren Sie, inwiefern dies umsetzbar ist! **2 P**

In der ursprünglichen Taskmenge (T_1, T_2, T_3) wird festgestellt, dass T_1 und T_3 ein gemeinsames Spin-Lock benutzen. Somit kann T_1 nur laufen, wenn T_3 nicht läuft und umgekehrt. Fordern beide zum gleichen Zeitpunkt die Ressource an, so wird sie gemäß der Task-Priorität an T_1 vergeben.

c) Zeigen Sie entstehende Probleme anhand eines Ablaufplans! **2 P**

Sie können diesen Bereich zum Zeichnen von Ablaufplänen nutzen:

Aufgabe 3 – Wechselseitiger Ausschluss **9 Punkte**

Eine Thread-Bibliothek stellt als Synchronisationsprimitive die Funktionen `lock()` und `unlock()` bereit. Diese sind wie folgt implementiert:

```
1 void lock(bool *L)
2 {
3   while (*L == true) {}
4   *L = true;
5 }
```

```
1 void unlock(bool *L)
2 {
3    *L = false;
4 }
```

Zwei Threads (*T1* und *T2*) führen nun folgenden Code aus:

```
1 bool global_lock = false; // global zugreifbare Lock-Variable
2
3 void thread_fn()
4 {
5    lock(&global_lock);
6    work();
7    unlock(&global_lock);
8 }
```

a) Zeigen Sie, dass die oben gegebene Lock-Implementierung nicht in der Lage ist, wechselseitigen Ausschluss für die beiden Threads zu realisieren. **3 P**

b) Welche Hilfsmittel stellen moderne Mehrkern-CPUs in der Regel bereit, um das in a) gefundene Problem zu lösen? **1 P**

c) Wie müssen die oben angegebenen Funktionen `lock()` und `unlock()` angepasst werden, um die in b) genannten Hardware-Mittel zu nutzen? **3 P**

d) Welches Problem hat die nunmehr in c) gefundene `lock()`-Implementierung im Hinblick auf die Prozessor-Auslastung? **1 P**

Als Alternative zur obigen Lock-Implementierung werden u. a. Ticket-Locks verwendet. Diese arbeiten auf einer zweigeteilten Lock-Datenstruktur:

```
1 struct lock_t {
2    int next;
3    int current;
4 }
```

`next` und `current` werden mit `0` initialisiert.

Wechselseitiger Ausschluss wird dann wie folgt implementiert:

```
1 void ticket_lock(lock_t *L)
2 {
3    int tick =
     ↪ atomic_inc(L->next);
4    while (L->current != tick) {}
5 }
```

```
1 void ticket_unlock(lock_t *L)
2 {
3    L->current += 1;
4 }
```

Hinweis: Die Funktion `atomic_inc()` inkrementiert einen Wert atomar und gibt den *ursprünglichen* Wert der Variable zurück.

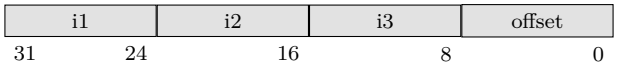
Ein Multiprozessorsystem besteht nun aus Prozessoren mit potenziell unterschiedlicher Taktfrequenz und Auslastung. Auf diesem System laufen mehrere (potenziell: viele) Threads, welche versuchen, dasselbe Lock zu greifen.

e) Was ist der zentrale Vorteil der hier gegebenen Ticket-Lock-Implementierung gegenüber der einfachen Lock-Implementierung aus c)? **1 P**

Aufgabe 4 – Speicher II

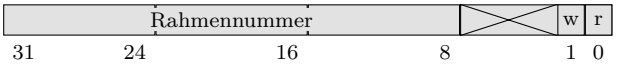
8 Punkte

Gegeben sei eine fiktive Rechnerarchitektur, die 32-bit virtuelle und 32-bit physische Adressen verwendet. Die Adressübersetzung sei 3-stufig und basiere auf folgender Aufteilung der virtuellen Adresse:



Dabei seien `i1`, `i2` und `i3` die Indizes für die jeweiligen Übersetzungsstufen, d. h. `i1` wird für die erste Stufe, `i2` für die zweite und `i3` für die dritte verwendet.

Die Seitentabellen-Einträge sehen wie folgt aus:



D. h. in den oberen 24-Bit wird die Rahmennummer abgelegt, Bit 0 gibt an, ob die Seite lesbar ist und Bit 1, ob sie schreibbar ist. Diese Rechte werden in jeder Stufe geprüft.

Das Betriebssystem hat folgende Seitentabellen für ein Nutzer-Programm angelegt, wobei das Register `root` ein Hardware-Register ist, welches auf die Seitentabelle der ersten Stufe für das aktuelle Programm zeigt. Über jeder unten angegebenen Seitentabelle steht die physische Adresse selbiger. Zudem ist vor jedem Eintrag in der Tabelle der Index des Eintrags sichtbar. Alle Zahlen sind hexadezimal angegeben.

root: 100

100	
0	103
1	503
2	503
3	502
4	501
	...
FF	503

500	
0	901
1	902
2	903
3	903
4	900
	...
FF	D01

900	
0	1801
1	2100
	...
9	7001
A	A003
	...
FF	8002

a) Wie groß ist bei dieser Architektur eine Seite und wie groß eine Seitentabelle? **1 P**

b) Was muss das Betriebssystem somit bei der Allokation von Seitentabellen beachten? **1 P**

c) Führen Sie für folgende Operationen die Adressübersetzung durch. Geben Sie dabei, sofern kein Seitenfehler auftritt, die physische Adresse an, auf die zugegriffen wird.

1. Lesen: 4 Byte von virtuell `0x020309F0`
2. Lesen: 4 Byte von virtuell `0x020400F8`
3. Schreiben: `0x0000FFFF` (4 Byte) auf virtuell `0x04030900`
4. Lesen: 4 Byte von virtuell `0x02030100`

4 P

d) Was ändert sich, wenn *vor dem 4. Zugriff* in c) zunächst `0x00006003` (4 Byte) auf virtuell `0x00020304` geschrieben wird? Beachten Sie mögliche Auswirkungen auf Seitentabellen. **2 P**

Aufgabe 5 – Sicherheit **6 Punkte**

Gegeben sei ein fiktives System des Herstellers *Herkules*. Neben dem Besitzer des Systems, *Bernd*, gibt es einen weiteren Anwender *Hans*. Sowohl *Bernd* als auch *Hans* gehören zur Gruppe *Users*.

a) *Bernd* lädt von der Internetseite des Herstellers die Programmerweiterung *Zeus* herunter. Er möchte sicherstellen, dass die Erweiterung tatsächlich von der Firma *Herkules* stammt und nicht von Dritten manipuliert worden ist.
Nennen Sie ein dafür geeignetes Verfahren oder erläutern Sie kurz warum dieses Problem prinzipiell nicht lösbar ist! **1 P**

b) Geben Sie eine Rechtezuweisung im Rahmen des klassischen Unix-Rechtesystems an, die allen Benutzern *ausschließlich die Ausführung* des Programms `zeus` erlaubt!
Nutzen Sie dafür die Notation `<Rechte-Bits> <Besitzer> <Gruppe>`. **1 P**

c) Um das Prinzip der geringst-möglichen Privilegisierung umzusetzen, gibt es einen Benutzer *dialout* welcher keiner Gruppe zugehört. Nur der Benutzer *dialout* darf auf das Modem zugreifen, beispielsweise um es durch die Eingabe der PIN freizuschalten. Für das Übergeben der PIN ans Modem gibt es ein Programm `enterpin`. Nur Mitglieder der Gruppe *Users* sollen damit die PIN eingeben dürfen.
Setzen Sie die nötigen Unix-Zugriffsrechte für `enterpin`, so dass *Bernd* die PIN auf seinem System eingeben kann! **2 P**

d) Geben Sie eine Umsetzung der in b) beschriebenen Rechtezuweisung mittels Capability-Listen an und nennen Sie einen Nachteil von Capability-Listen gegenüber dem klassischen Unix-Rechtesystem! **2 P**

Aufgabe 6 – Dateisysteme

7 Punkte

Ein Unix-artiges Dateisystem organisiert alle Dateiinhalte und Metadaten in Blöcken auf dem Speichermedium. Blöcke enthalten entweder Daten (D), Inodes (I), Verzeichniseinträge (V), eine Allokations-Bitmap für Inodes bzw. Blöcke (IA bzw. BA) oder den Superblock (SB). Die Größe eines Blocks beträgt 4 KiByte. Der initiale Zustand des Dateisystems ist wie folgt:

Block	0	1	2	3	4	5	6	7	8	9	10
Typ	SB	IA	BA	I	V	D	D				
Inhalt	/: 0 frei: 4	0 – 1	0 – 6	0: 4 1: 5,6	A: 1	Nutz-daten	Nutz-daten				

Inode-Zeiger für / und Anzahl freier Blöcke

Inodes 0 und 1 belegt

Block 0 bis 6 belegt

Blockzeiger in Inodes 0 und 1

Abbildung des Dateinamens *A* auf zugehöriges Inode 1

a) Welche Blöcke muss das Betriebssystem lesen, wenn der Nutzer mit dem Kommando `ln /A /B` einen weiteren Hardlink für die Datei *A* im Wurzelverzeichnis anlegt? **2 P**

b) Nach dem Anlegen des Hardlinks erzeugt der Nutzer mit Hilfe eines Anwendungsprogramms im Wurzelverzeichnis eine neue Datei *C* mit einer Größe von 5144 Byte. Tragen Sie in das unten stehende Schema den Zustand des Dateisystems nach Abschluss dieser Operation ein. Kennzeichnen Sie alle Blöcke, deren Inhalt sich gegenüber dem oben abgebildeten Schema geändert hat.
Hinweis: In Inode- und Verzeichnisblöcken sei ausreichend Speicherplatz vorhanden. **2 P**

Block	0	1	2	3	4	5	6	7	8	9	10
Typ											
Inhalt											
geändert											

Zur Leistungssteigerung puffert das Betriebssystem neue oder modifizierte Blöcke zunächst im Hauptspeicher und schreibt diese später im Hintergrund auf das Speichermedium. Dabei werden Blöcke bestimmter, aber nicht aller, Typen zunächst in ein Log (oder Journal) geschrieben. Für das Log sind die Blöcke 11–16 exklusiv reserviert.

c) Tragen Sie in unten stehendes Schema eine mögliche Journal-Transaktion ein, bei der sichergestellt ist, dass Datei C nach einem Absturz korrekt und vollständig wiederherstellbar ist. Erläutern Sie, wann die jeweiligen „in-place“-Kopien (Blocknummern 0–10) geschrieben werden und warum nicht alle Blocktypen (welche?) ins Log geschrieben werden müssen. **3 P**

Block	11	12	13	14	15	16
Typ						