Name: Matrikel-Nr.: Studiengang:

Klausur Betriebssysteme und Sicherheit, 27.02.2014

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

Aufgabe 1 – Sicherheit

6 Punkte

a) Wie kann man auf einem Computer Zufall (Entropie) erzeugen? Nennen Sie zwei Möglichkeiten! **2 P**

b) Nennen Sie einen prinzipiellen Vorteil von asymmetrischen gegenüber symmetrischen Verschlüsselungsverfahren! **1 P**

c) Kann der folgende Modul Bestandteil eines sicheren RSA-Schlüssels sein?
Begründen Sie! n = 0x2D16E6ADE5C **1 P**

d) Einige Endgeräte haben schlechte Entropiequellen. Was bedeutet es für die Sicherheit von RSA, wenn zwei Teilnehmer unbeabsichtigt einen gleichen Faktor wählen? **2 P**

Name: Matrikel-Nr.: Studiengang:

Aufgabe 2 – UNIX

5 Punkte

Ein UNIX-Prozeß führt folgenden Code aus:

```
int fd1  = open("datei1.txt", O_RDWR | O_CREAT);
int fd2  = open("datei2.txt", O_RDWR | O_CREAT);

int pid1 = fork();

if (pid1 == 0)
{
   write(fd1, "abc", 3);
   exit(0);
}
else
{
    int pid2 = fork();
    if (pid2 == 0)
    {
       write(fd2, "abc", 3);
       exit(0);
    }

    waitpid(pid1, NULL, 0);
    write(fd1, "def", 3);
    write(fd2, "def", 3);
}
```

Die Dateien `datei1.txt` und `datei2.txt` existieren vor Programmausführung nicht, sondern werden durch das Öffnen angelegt. Alle Systemaufrufe laufen korrekt zu Ende, d.h. es treten keine Laufzeitfehler auf. Prozesse können zu einem beliebigen Zeitpunkt (auch während der Ausführung eines Systemaufrufs) unterbrochen werden.

Welchen Inhalt können die Dateien `datei1.txt` und `datei2.txt` nach erfolgreicher Ausführung des Programms haben?

Name: Matrikel-Nr.: Studiengang:

Aufgabe 3 – Echtzeit

8 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	T_2	T_3
$p_1 = 5 \quad e_1 = 2$	$p_2 = 10 \quad e_2 = 3$	$p_3 = 4 \quad e_3 = 1$

Alle Tasks starten zum Zeitpunkt 0, können an beliebiger Stelle unterbrochen werden und sind unabhängig voneinander. Der Scheduling-Overhead werde vernachlässigt. Es stehe genau ein Prozessor zur Verfügung.

a) Ist die gegebene Taskmenge mit statischen Prioritäten einplanbar? Wenn ja, geben Sie eine funktionierende Prioritätszuweisung an. **2 P**

b) Ist die gegebene Taskmenge mit dem Verfahren EDF einplanbar? Wenn ja, erklären Sie, welche Task zum Zeitpunkt 182 die höchste Priorität hat! **2 P**

Um Energie zu sparen sollen mehrere langsamere Prozessoren anstelle eines schnellen Prozessors verwendet werden. Die sparsamen Prozessoren laufen mit halber Geschwindigkeit, so dass sich die Bearbeitungszeiten e der Jobs verdoppeln. Tasks sollen nicht zwischen Prozessoren migrieren, sondern werden fest zugeteilt.

Hinweis: *Migration* bezeichnet den Vorgang der Neu-Zuteilung von Tasks zu Prozessoren, sowohl zwischen einzelnen Jobs als auch innerhalb eines Jobs.

c) Wie viele der sparsamen Prozessoren sind minimal nötig, um die Taskmenge unter diesen Bedingungen einzuplanen? Belegen Sie Ihre Antwort! **2 P**

d) Ändert sich diese minimale Prozessoranzahl, wenn Jobs während ihrer Laufzeit zwischen Prozessoren migrieren können? Der Migrations-Overhead werde vernachlässigt. Begründen Sie Ihre Antwort durch einen geeigneten Ablaufplan. **2 P**

Name: Matrikel-Nr.: Studiengang:

Aufgabe 4 – Speicherverwaltung **3 Punkte**

Ein System verwendet das Buddy-Verfahren zur Verwaltung des Speichers. Der freie Speicher umfasst 16 Seiten. Es werden nacheinander 5 Segmente (A–E), welche jeweils aus 1, 7, 3, 3 und 2 Seiten bestehen in den Speicher eingelagert. Kann eine Anfrage nicht erfüllt werden, so wird diese Anfrage ignoriert und mit der folgenden Anforderung fortgefahren.

Geben Sie den Ablauf der Einlagerungen und die Lage der Segmente im Speicher an. Kennzeichnen Sie nicht erfüllbare Anfragen.

A:1

B:7

C:3

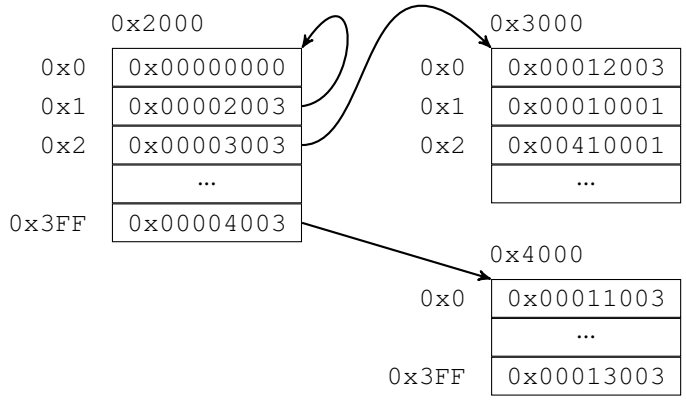
D:3

E:2

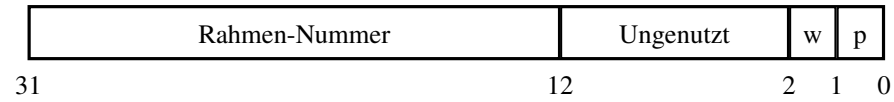
Name: Matrikel-Nr.: Studiengang:

Aufgabe 5 – Speicherverwaltung **6 Punkte**

Gegeben seien folgende Seitentabellen für eine Architektur mit einem 32 Bit großen virtuellen und physischen Adressraum. Die Seitengröße beträgt 4096 Byte. Es wird eine zweistufige Abbildung mit jeweils 10 Bit pro Stufe verwendet.



Die Abbildung zeigt ein Seitenverzeichnis und zwei Seitentabellen jeweils mit ihrer physische Adresse. Vor den einzelnen Seitentabellen-Einträgen steht der Index. Der Eintrag selber habe folgendes Format (wobei p für present und w für writable steht):



a) Führen Sie für folgende Zugriffe die Adressumsetzung durch und geben Sie an, ob dabei ein Seitenfehler auftritt (fehlendes present oder writable Bit) oder anderenfalls die resultierende physische Adresse. Geben Sie bei den lesenden Zugriffen, sofern möglich, außerdem an welcher Wert gelesen wird. **4 P**

I. Lesen von 4 Byte von virtuell `0x000000FF`

II. Schreiben von 4 Byte an virtuell `0x00801012`

III. Schreiben von 4 Byte an virtuell `0xFFFFF556`

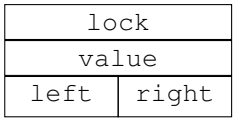
IV. Lesen von 4 Byte von virtuell `0x00402004`

b) Angenommen auch der Betriebssystemkern kann nicht direkt auf physische Adressen zugreifen, sondern muss ebenfalls virtuelle Adressen benutzen. An welche virtuelle Adresse muss er schreiben um Eintrag 0x0 in der Seitentabelle unten rechts zu ändern? **2 P**

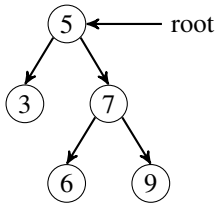
Aufgabe 6 – Synchronisation **6 Punkte**

Um den Datentyp einer Menge zu implementieren, soll ein Binärbaum verwendet werden. Mehrere Threads können parallel Elemente in diesen Baum einfügen (`insert()`). Weiterhin kann auf die Existenz eines Eintrags geprüft werden (`contains()`). Aus Gründen der Skalierbarkeit soll für diese Operationen kein globales Lock für den kompletten Baum verwendet werden, sondern einzelne Locks in den jeweiligen Knoten, die dabei folgende Struktur haben:

Node:



Baum:




```
insert (Node * node, int value)
{
    while (1) {
        node->lock();
        if (value < node->value) {
            node->unlock();
            if (node->left) {
                node = node->left;
            } else {
                node->left = new Node (value);
                return;
            }
        } else if (value > node->value) {
            // similar for right path
        } else {
            // node->value == value --> do nothing
            node->unlock();
            return;
        }
}
```

a) Geben Sie für den Aufruf `insert(root, 8)` auf den obigen Baum alle Semaphore-Operationen in entsprechender zeitlicher Reihenfolge in folgendem Format an: *Knotennummer : lock/unlock* (z.B. 6 : lock oder 5 : unlock) **2 P**

b) Welches Problem kann auftreten, wenn zwei Threads gleichzeitig die Werte 1 und 2 in den Baum eintragen möchten? Verwenden Sie die gleiche Notation wie aus Aufgabe a) und zeigen Sie, wie es zu einem inkonsistenten Baum kommen kann. **2 P**

c) Geben Sie eine korrekte Implementierung von `insert()` an, die weiterhin ohne globales Lock arbeitet. Es genügt, wenn Sie nur einen der beiden if-then-else-Zweige betrachten, da sich der andere analog verhält. **2 P**

```
insert (Node * node, int value)
{
    while (1) {

        } else if (value > node->value) {
            // similar for right path
        } else {
            // node->value == value --> do nothing
            node->unlock();
            return;
        }
}
```

Name: Matrikel-Nr.: Studiengang:

Aufgabe 7 – Dateisystem

8 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 oder Blöcke (IA oder IB), 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	11
Typ	SB	IA	IB	I	V	D	D					
Inhalt	/ : 0 frei: 5	0-1	0-6	0:4 1:5,6	D1:1	Daten	Daten					

Inode-Zeiger für '/' und Anzahl freier Blöcke

Inodes 0 und 1 belegt

Block 0-6 belegt

Blockzeiger in Inodes 0 und 1

Abbildung des Namens *D1* auf Inode 1

a) Eine Anwendung öffnet die Datei *D1* im Wurzelverzeichnis des Dateisystems und liest deren Inhalt. Welche Daten- und Metadatenblöcke muss das Betriebssystem lesen? In welcher Reihenfolge erfolgen die Zugriffe? **2 P**

b) Die Anwendung legt eine neue Datei *D2* im Wurzelverzeichnis an und speichert 5000 Byte Daten darin. Tragen Sie in das unten stehende Schema den neuen Zustand des Dateisystems nach Abschluss der Operation ein. Kennzeichnen Sie alle Blöcke, deren Inhalt sich geändert hat. **3 P**

Block	0	1	2	3	4	5	6	7	8	9	10	11
Typ												
Inhalt												
Geän-dert												

Name: Matrikel-Nr.: Studiengang:

Zur Leistungssteigerung puffert das Betriebssystem neue oder modifizierte Blöcke zunächst im Hauptspeicher und schreibt diese später im Hintergrund auf das Speichermedium. Dabei wird folgende Strategie angewendet: Zunächst werden alle Datenblöcke (D) geschrieben, danach Inode-Blöcke (I) gefolgt von Verzeichnisblöcken (V) und zum Schluss alle geänderten Blöcke der Allokations-Bitmaps (IA und IB) und der Superblock (SB).

c) Welche Inkonsistenzen können auftreten, wenn das System etwa in Folge eines Stromausfalls abstürzt bevor alle Blöcke geschrieben wurden? **1 P**

d) Geben Sie eine konkrete Schreibstrategie für die in Teilaufgabe b) beschriebene Operation auf der Datei D2 an, bei der Konsistenz auch dann sichergestellt ist, wenn das System an beliebiger Stelle abstürzt. Sind nach dem Neustart des Systems noch Korrekturen erforderlich und wenn ja, welche? **2 P**