Name: Matrikel-Nr.: Studiengang:

Klausur Betriebssysteme und Sicherheit, 24.02.2016

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

Aufgabe 1 – Sicherheit

7 Punkte

a) Erklären Sie die Begriffe semantische Sicherheit (Semantic Security) und beweisbare Sicherheit (Provable Security)! **2 P**

b) Betrachtet werde das Verschlüsselungsverfahren „One-Time-Pad“. Ist das Verfahren symmetrisch oder asymmetrisch? Wie lang muss ein Schlüssel sein, um eine Nachricht der Länge m Bit sicher zu verschlüsseln? Welche Eigenschaft im Sinne der Sicherheit hebt dieses Verfahren von allen anderen bekannten Verschlüsselungsverfahren ab? **2 P**

Gegeben seien die folgenden Parameter für das Verschlüsselungsverfahren RSA:

- Modul $n = 65$
- Öffentlicher Schlüssel $c = 5$

Die Nachricht lautet $x = 8$.

c) Bestimmen Sie zunächst den privaten Schlüssel d! Berechnen Sie außerdem den Schlüsseltext y und die Signatur t für die Nachricht x (d.h. signieren Sie x)! **3 P**

Aufgabe 2 – Unix

6 Punkte

Betrachtet werde folgender C-Code:

```
1  int fd1, fd2 = 23;
2
3  int main() {
4     int pid;
5
6     fd1 = creat("/tmp/file1");
7     pid = fork();
8
9     fd2 = creat("/tmp/file2");
10    write(fd1, "A", 1);
11    write(fd2, "B", 1);
12
13    if (pid == 0) { exec("/bin/nichts"); }
14    printf("fd1: %d, fd2: %d, pid: %d\n", fd1, fd2, pid);
15    close(fd1);
16    close(fd2);
17
18    return 0;
19 }
```

Hinweis: Der Platzhalter `%d` in den `printf`-Aufrufen wird jeweils durch die als Argument übergebene `int`-Variable ersetzt. Die Funktion `creat` legt die angegebene Datei an und öffnet sie. Sollte die Datei bereits existieren, so wird ihr Inhalt beim Öffnen komplett gelöscht. Das Programm `/bin/nichts` erzeugt *keine* Ausgabe.

a) In welchen Segmenten des Unix-Adressraumes liegen die Symbole `main`, `fd1`, `fd2` und `pid`? **1 P**

Das Programm wird nun ausgeführt. Alle auftretenden Funktionsaufrufe sind dabei erfolgreich.

b) Was wird auf der Konsole ausgegeben? Sollte es mehrere Möglichkeiten geben, so geben Sie die Ursache dafür an und nennen Sie eine zweite mögliche Ausgabe! Wird sie durch die Systemaufrufe `fork` und `exec` beeinflusst? Begründen Sie Ihre Antwort! **3 P**

c) Welchen Inhalt haben die beiden angelegten Dateien? Sollte es mehrere Möglichkeiten geben, so geben Sie die Ursache dafür an und nennen Sie einen zweiten möglichen Dateiinhalt! **2 P**

Aufgabe 3 – Scheduling **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: $p_1 = 3$, $e_1 = 1$
T_2: $p_2 = 12$, $e_2 = 2$
T_3: $p_3 = 4$, $e_3 = 1$

Alle Tasks starten zum Zeitpunkt $t = 0$ und können an beliebiger Stelle unterbrochen werden. Der Scheduling-Overhead werde vernachlässigt und es stehe genau ein Prozessor zur Verfügung.

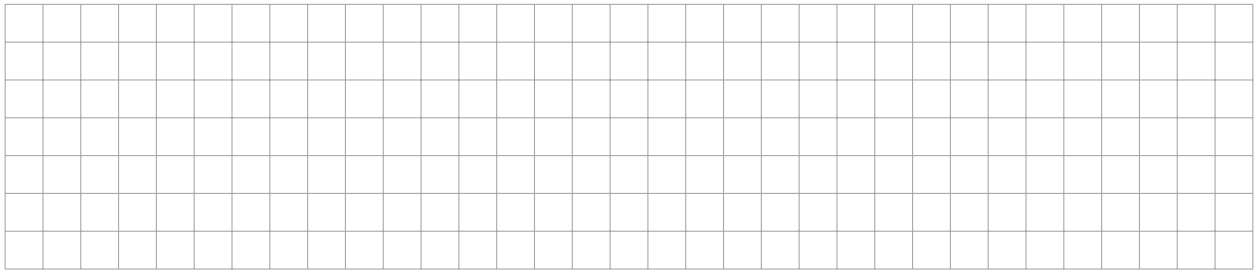
a) Zunächst werden den Tasks statisch Prioritäten zugeteilt, so dass T_1 die höchste, T_2 eine mittlere und T_3 die niedrigste Priorität erhält. Weisen Sie nach, dass die gegebene Task-Menge unter den genannten Bedingungen nicht einplanbar ist! **1 P**

b) Ist eine Einplanung dieser Task-Menge mit statischen Prioritäten überhaupt möglich? Wenn ja, geben Sie eine entsprechende Prioritätszuteilung an! **2 P**

c) Die Task-Menge soll mit dynamischen Prioritäten eingeplant werden. Schlagen Sie ein passendes Einplanungsverfahren vor und geben Sie an, welcher Task im gewählten Verfahren zum Zeitpunkt $t = 3{,}5$ die höchste Priorität aufweist. **2 P**

d) Weiterhin soll die Ausführungszeit e_3 der Task T_3 erhöht werden. Bis zu welchem Maximalwert von e_3 ist eine Einplanung mit dynamischen Prioritäten möglich? **1 P**

Sofern erforderlich, können Sie diesen Bereich zum Zeichnen von Ablaufplänen für alle Teilaufgaben nutzen.



Aufgabe 4 – Speicher **8 Punkte**

Ein hypothetisches 32-Bit-System verwaltet den Speicher mittels zweistufiger Seitentabellen. Das Offset beträgt 12 Bit und für jede Stufe werden 10 Bit zur Adressierung der Einträge in den Seitentabellen verwendet. Das Betriebssystem organisiert den Adressraums eines Prozesses wie folgt:

Region	Startadresse (virtuell)	Größe	I-Node	Offset	Rechte
Code	`0x1000`	12 KiB	123	`0x2000`	`r-x`
Daten	`0x4000`	6 KiB	123	`0x5000`	`rw-`
BSS	`0x6000`	10 KiB	—	—	`rw-`
Stack	`0xBFFF F000`	4 KiB	—	—	`rw-`

Bei den Rechten steht `r` für das Leserecht, `w` für das Schreibrecht und `x` für das Recht Code auszuführen. Diese Rechte werden auch in den Seitentabellen verwendet und in jeder Stufe überprüft.

a) Wie viele Einträge enthält eine Seitentabelle? **1 P**

b) Wie viel physischen Speicher kann das System adressieren? **1 P**

Da das Betriebssystem Seiten erst beim ersten Zugriff darauf in den Arbeitsspeicher lädt (Demand Paging), ist der Adressraum beim Start leer. Beim Start des Programms wird auf folgende virtuelle Adressen in der angegebenen Reihenfolge und Art zugegriffen.

1. ausführend auf `0x1000`
2. ausführend auf `0x1004`
3. schreibend auf `0xBFFF FFFC`
4. lesend auf `0x3100`

Dem Betriebssystem stehen folgende freie Rahmen zur Verfügung, gegeben als deren physische Startadressen:

`0x4000`, `0x8000`, `0xA000`, `0xC000`, `0xD000`

Diese sollen *in der gegebenen Reihenfolge* vom Betriebssystem verwendet werden.

c) Nehmen Sie die Rolle des Betriebssystems ein, welches die dabei auftretenden Seitenfehler behandelt. Füllen Sie dafür das untenstehende Seitentabellenschema für den Zustand nach diesen Zugriffen aus! Die Seitentabellen werden dabei ebenfalls bei Bedarf allokiert. **4 P**

Seitentabelle – Stufe 1

Addr.: `0x2000`		Rechte
0		
1		

Seitentabelle – Stufe 2

Addr.:		Rechte
0		
1		

Seitentabelle – Stufe 2

Addr.:		Rechte
0		
1		

d) Geben Sie für folgende zwei Zugriffe die resultierende physische Adresse bzw. den Grund für das Scheitern der Adressumsetzung an! **2 P**

- lesend auf `0xBFFF FFFC`:
- schreibend auf `0x3FF1`:

Aufgabe 5 – Seitenersetzung **8 Punkte**

Es wird erneut ein 32-Bit-System betrachtet: Der Speicher wird mittels zweistufiger Seitentabellen verwaltet. Das Offset beträgt 12 Bit und für jede Stufe werden 10 Bit zur Adressierung der Einträge in den Seitentabellen verwendet.

Bei der weiteren Ausführung des Systems stehen insgesamt 4 Rahmen zur Verfügung, welche der Einfachheit halber hier mit 1 – 4 bezeichnet werden. Diese sind zu Beginn frei. Es erfolgen (in der angegebenen Reihenfolge) Zugriffe auf folgende virtuelle Adressen:

`0x1000` `0x5408` `0x40D8` `0x1004` `0xBFFF FFFC` `0x600C` `0x1006` `0x4488` `0x2DCC` `0xBFFF FFF4` `0x4188`

a) Füllen Sie folgende zwei Schemata für die angegebene Seitenverdrängungsalgorithmen aus! Tragen Sie dazu in jeder Spalte die *Seite* in den Rahmen ein, in welchen das System sie platzieren würde bzw. in dem Sie sich bereits befindet. Geben Sie ferner an wenn ein Seitenfehler auftritt, indem Sie in der Zeile „SF“ ein Kreuz setzen. **5 P**

FIFO (First In – First Out):

Rahmen	`0x1000`	`0x5408`	`0x40D8`	`0x1004`	`0xBFFF FFFC`	`0x600C`	`0x1006`	`0x4488`	`0x2DCC`	`0xBFFF FFF4`	`0x4188`
1											
2											
3											
4											
SF											

LRU (Least Recently Used):

Rahmen	`0x1000`	`0x5408`	`0x40D8`	`0x1004`	`0xBFFF FFFC`	`0x600C`	`0x1006`	`0x4488`	`0x2DCC`	`0xBFFF FFF4`	`0x4188`
1											
2											
3											
4											
SF											

b) Welcher dieser Algorithmen ist, hinsichtlich der Anzahl der Seitenfehler, der bessere? Nennen Sie zudem einen Nachteil gegenüber dem anderen! **1 P**

c) Gibt es eine Strategie mit weniger Seitenfehlern? Wenn ja, zeigen Sie diese in untenstehendem Schema auf! **2 P**

Rahmen	`0x1000`	`0x5408`	`0x40D8`	`0x1004`	`0xBFFF FFFC`	`0x600C`	`0x1006`	`0x4488`	`0x2DCC`	`0xBFFF FFF4`	`0x4188`
1											
2											
3											
4											
SF											

Aufgabe 6 – Dateisystem **7 Punkte**

Der Programmcode einer Anlagensteuerung ist in einer 86 KiB großen ausführbaren Programmdatei gespeichert. Diese Datei ist unter dem Pfad `/bin/reactorcontrol` in einem Unix-artigen Dateisystem abgelegt, welches alle Dateiinhalte und Metadaten in Blöcken auf dem Speichermedium organisiert. Die Größe eines Blocks beträgt 4 KiB.

a) Wieviele Blöcke werden benötigt um den Inhalt der Programmdatei zu speichern? Nennen Sie außerdem vier verschiedene Arten von Metadaten, die zur Repräsentation der Datei im Dateisystem notwendig sind, und wo diese gespeichert werden. **3 P**

Im ausführbaren Code der Programmdatei wurde eine gefährliche Sicherheitslücke gefunden. Daher muss dringend eine neue, fehlerbereinigte Version von `/bin/reactorcontrol` installiert werden. Um die Verfügbarkeit des Systems sicherzustellen, muss die Installation der Programmdatei atomar erfolgen. Das heißt: Falls das System während der Installation abstürzt, soll nach dem Neustart unter `/bin/reactorcontrol` entweder die alte oder die neue Version der Programmdatei vollständig vorliegen. Das eingesetzte Software-Update-Tool nutzt dafür die folgenden Systemaufrufe:

`creat("pfad")`	→ Legt eine neue Datei unter „pfad“ an und liefert als Rückgabewert ein Handle
`close(handle)`	→ Schließt die durch `handle` angegebene Datei
`fsync(handle)`	→ Macht die durch `handle` angegebene Datei sicher persistent
`rename("alt", "neu")`	→ Benennt die Datei unter Pfad „alt“ in „neu“ um
`write(handle, puffer, laenge)`	→ Schreibt `laenge` Bytes aus `puffer` in die durch `handle` angegebene Datei

b) Ergänzen Sie den unten stehenden Pseudocode der Update-Routine unter Verwendung dieser Systemaufrufe, so dass die erforderliche Robustheit gewährleistet ist. Welcher Systemaufruf implementiert die kritische Operation Ihrer Lösung und welche Annahme muss dabei gelten? **4 P**

```
void *puffer = get_file_content(); // liefert Zeiger auf Inhalt der Datei
size_t laenge = get_file_size();   // liefert Länge der Datei in Bytes
int handle;
```