Name: Matrikel-Nr.: Studiengang:

Klausur Betriebssysteme und Sicherheit, 16. 08. 2018

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

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

Aufgabe 1 – Sicherheit

15 Punkte

a) Nennen Sie die drei in der Vorlesung besprochenen (Haupt-)Schutzziele der Datensicherheit sowie konkrete Verfahren bzw. Algorithmen, um diese Schutzziele im Rahmen der E-Mail-Kommunikation zu erreichen. **6 P**

b) Wir betrachten als Beispiel weiterhin die Kommunikation per E-Mail, konkret das Versenden einer Nachricht. Wählen Sie eines der in Teilaufgabe a) genannten kryptografischen Verfahren aus und erläutern Sie, wie es eingesetzt wird. Nehmen Sie dabei auf die verwendeten Schlüssel Bezug.

HINWEIS: Sie können davon ausgehen, dass ein ggf. nötiger Schlüsselaustausch bereits erfolgreich stattgefunden hat. **3 P**

c) Nennen und erklären Sie einen Vorteil symmetrischer und asymmetrischer Verschlüsselungsalgorithmen gegenüber der jeweils anderen Klasse. Zu welcher Klasse gehört das „One Time Pad“, auch bekannt als „Vernam-Chiffre“? Begründen Sie Ihre Antwort. **3 P**

d) Auf einem Linux-System existieren die Nutzer A und B, die beide der Gruppe G angehören und Zugriff auf ein gemeinsames Verzeichnis haben. Nutzer A möchte eine Datei D erstellen, die auch Nutzer B lesen kann.

Geben Sie drei mögliche Zuweisungen von Unix-Rechten (Besitzer, Gruppe und Rechtebits) für die Datei D an, so dass beide Nutzer Lesezugriff auf die Datei D haben. **3 P**

Aufgabe 2 – Seitenersetzung **15 Punkte**

Ein System verwaltet 4 Rahmen physischen Speichers. Dabei seien zunächst alle Rahmen frei. Im System existieren 3 Prozesse (`A`, `B`, `C`), die die folgende Seitenreferenzfolge (gegeben als `Prozess:Seitennummer`) ausführen:

`A:4` `B:0` `A:4` `C:1` `C:3` `A:6` `C:4` `B:0` `A:4` `A:6` `C:1` `C:4` `A:4`

a) Wenden Sie für diese Seitenreferenzfolge zunächst den optimalen Seitenersetzungsalgorithmus (OPT) an. Markieren Sie Seitenfehler durch ein Kreuz in der Zeile *SF*. **5 P**

Rahmen	`A:4`	`B:0`	`A:4`	`C:1`	`C:3`	`A:6`	`C:4`	`B:0`	`A:4`	`A:6`	`C:1`	`C:4`	`A:4`
1													
2													
3													
4													
SF													

b) Warum wird OPT in der Praxis nicht eingesetzt? **1 P**

c) Die Strategie *Least Recently Used* (LRU) gilt als gute Näherung von OPT. Aufgrund welcher Annahme/Beobachtung ist das so? **1 P**

d) Wenden Sie LRU auf die eingangs genannte Seitenreferenzfolge an. Machen Sie dabei wiederum Seitenfehler kenntlich. **5 P**

Rahmen	`A:4`	`B:0`	`A:4`	`C:1`	`C:3`	`A:6`	`C:4`	`B:0`	`A:4`	`A:6`	`C:1`	`C:4`	`A:4`
1													
2													
3													
4													
SF													

e) Auch LRU hat jedoch ein Problem, das eine Implementierung in einem realen System verhindert bzw. stark erschwert. Welches? **1 P**

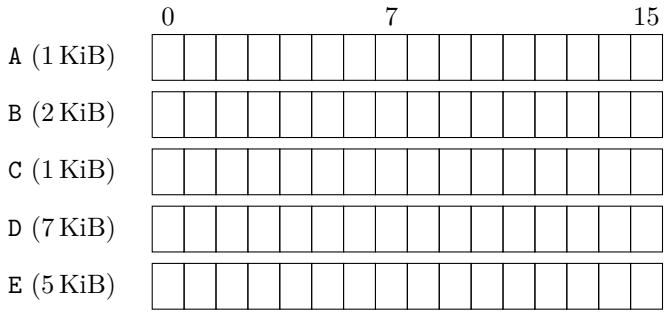
f) Nennen Sie ein praktisch nutzbares Seitenverdrängungsverfahren, das versucht OPT bzw. LRU anzunähern. **1 P**

g) Was geschieht mit Seiten, die aus dem physischen Speicher verdrängt werden? **1 P**

Aufgabe 3 – Speicherverwaltung **14 Punkte**

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

a) Geben Sie den Ablauf der Einlagerungen und die Lage der Segmente im Speicher an. Kennzeichnen Sie nicht erfüllbare Anfragen. **5 P**



Ein System soll byte-adressierten, virtuellen Speicher verwenden. Dabei werden folgende Anforderungen an die Speicherverwaltung gestellt:

- Die virtuellen Adressen sind 64 Bit breit.
- Die physischen Adressen sind ebenfalls 64 Bit breit.
- Es wird eine Adressumsetzung mit fünf Stufen verwendet.
- Jede Stufe verwendet 10 Bit der virtuellen Adresse als Index:

Idx_1	Idx_2	Idx_3	Idx_4	Idx_5	Offset

- Die Tabellen aller Stufen sind gleich groß und füllen jeweils einen Rahmen.

b) Beantworten Sie folgende Fragen zum beschriebenen Verwaltungsschema. Geben Sie dabei jeweils auch passende Einheiten an. **9 P**

I. Wie viel physischen Speicher kann das System maximal adressieren?

II. Wie viele Einträge hat die Seitentabelle der ersten Stufe?

III. Wie viele Einträge hat eine Seitentabelle der zweiten Stufe?

IV. Wie viele Bits der virtuellen Adresse entfallen auf den Offset?

V. Wie groß ist eine Seite?

VI. Wie groß ist ein Rahmen?

VII. Wie viele Rahmen kann das System maximal adressieren?

VIII. Ein virtueller Adressraum enthält zwei Seiten an beliebigen Adressen. Wie viel Speicher ist für die Verwaltung dieses Adressraums maximal nötig?

Aufgabe 4 – Unix

16 Punkte

Beim Start eines neuen Programms richtet das Betriebssystem für alle existierenden Signale eine Standardreaktion ein, die ausgeführt wird, wenn das entsprechende Signal auftritt. Mit der Funktion `signal` kann man diese Standardreaktion verändern.

a) Ist das für alle Signale, die in einem Unixsystem existieren, möglich? Wenn nicht, nennen Sie ein Signal, für das keine selbstgewählte Reaktion registriert werden kann. **1 P**

b) Nennen Sie zwei andere Signale, die in einem Unix-System gesendet werden können. **2 P**

Der unten stehende Code implementiert die Nutzung einer Pipe, um die Aus- und Eingabe zweier Prozesse miteinander zu verbinden. Derselbe Ablauf kann in einer Shell mit der folgenden Befehlssequenz realisiert werden: `cat myfile.dat | sort`

```
1  int main(void)
2  {
3     pipe();
4
5     int pid_cat = fork();
6
7     if (pid_cat == 0) {
8        close(4);
9        dup2(3, 1);
10       close(3);
11
12       exec("cat myfile.dat");
13    }
14
15    close(3);
16
17    int pid_sort = fork();
18    if (pid_sort == 0) {
19       dup2(4, 0);
20       close(4);
21
22       exec("sort");
23    }
24
25    close(4);
26
27    wait(pid_sort);
28
29    return 0;
30 }
```

`pipe()` erzeugt eine neue Pipe. Der Eingang der Pipe (PIPEIN) wird dabei durch File-Deskriptor 3 repräsentiert, ihr Ausgang (PIPEOUT) durch File-Deskriptor 4.

`dup2(fd1, fd2)` kopiert, den File-Deskriptor, der in `fd1` gespeichert ist, nach `fd2`. Nach dem Aufruf zeigen somit `fd1` und `fd2` auf die gleiche Datei. Sollte `fd2` vor dem Aufruf bereits auf eine andere Datei gezeigt haben, wird diese automatisch mit geschlossen.

`wait(pid)` blockiert den aktuellen Prozess, bis der Kindprozess mit der angegebenen `pid` sich beendet.

c) Die folgende Tabelle repräsentiert die offenen File-Deskriptoren (*open file table*) im Betriebssystemkern. Tragen Sie in den leeren Feldern ein, worauf die File-Deskriptoren der jeweiligen Prozesse zu den angegebenen Punkten im Programmablauf zeigen.

Nutzen Sie dafür die gegebenen Abkürzungen für die relevanten Dateien (STDIN, STDOUT, ...). Wenn der File-Deskriptor geschlossen ist, tragen Sie CLOSED ein. Sollte der jeweilige Prozess einen der angegebenen Punkte nicht erreichen können, kreuzen Sie die zugehörige Zelle durch. Die Spalte für Zeile 4 ist bereits als Beispiel vollständig vorgegeben. **9 P**

P	FD	Zeile 4	Zeile 6	Zeile 11	Zeile 14	Zeile 21	Zeile 24
main	0	STDIN					
	1	STDOUT					
	2	STDERR					
	3	PIPEIN					
	4	PIPEOUT					
cat	0						
	1						
	2						
	3						
	4						
sort	0						
	1						
	2						
	3						
	4						

d) In Unix kann ein Prozess in den Zustand „Zombie“ kommen. Erklären Sie, wie das passieren kann und welche Eigenschaften ein Zombie-Prozess hat. **2 P**

e) Entsteht im gegebenen Beispielprogramm ein Zombie-Prozess? Begründen Sie ihre Antwort. **2 P**

Aufgabe 5 – Dateisysteme **15 Punkte**

Ein Unix-artiges Dateisystem organisiert alle Nutz- und Metadaten in Blöcken. Das verwendete Speichermedium erfordert eine Blockgröße von 4 KiB.

a) Wie viel Speicherplatz kann das Dateisystem maximal verwalten, wenn für die Block-Bitmap 32 Blöcke vorgesehen sind? **1 P**

Bei dem in dieser Aufgabe betrachteten Dateisystem sind die Verwaltungsinformationen für einzelne Dateien über verschiedene Arten von Metadaten verteilt: Verzeichniseinträge (V), Inodes (I), belegt/frei-Status von Inodes (IA) und Blöcken (BA) sowie Blöcke mit den Dateiinhalten (D).

b) Das unten stehende Schema repräsentiert die Metadaten für eine Datei mit dem Namen `A` und das Wurzelverzeichnis, in dem der Verzeichniseintrag für `A` abgelegt ist. Die Datei `A` enthält 11.000 Byte Daten. Verzeichniseinträge sind generell 16 Byte groß.

Im Wurzelverzeichnis wird eine neue Datei `B` der Größe 4.096 Byte abgelegt. Ergänzen Sie die Metadaten bzw. passen Sie vorhandene Metadaten entsprechend an. **6 P**

Block	0	1	2	3	4	5	...
Typ	SB	IA	BA	I	V	D	...
Inhalt	$/ \to 0$	$0, 1$	$0, 1, 2, 3,$ $4, 5, 6, 7$	$I_0 \to V \mid 16 \mid 4$ $I_1 \to D \mid 11.000 \mid 5, 6, 7$	`"A"` $\to I_1$	Nutz-daten	...

Inode-Verweis für Wurzelverzeichnis

Inodes 0 und 1 belegt

Blöcke 0 bis 7 belegt

Typ | Größe | Blockzeiger für Inodes 0 und 1

Abbildung des Dateinamens `A` auf zugehöriges Inode 1

c) Nennen Sie zwei konkrete Beispiele für Inkonsistenzen im Dateisystem, die durch unvollständige Aktualisierung der betroffenen Verwaltungsinformationen und/oder Datenblöcke auftreten können. Beschreiben Sie die dabei mögliche Auswirkungen dieser Inkonsistenzen. **4 P**

d) Erläutern Sie die Grundidee des in der Vorlesung vorgestellten Journaling-Verfahrens. Gehen Sie dazu kurz auf die Reihenfolge ein, in der Blöcke auf das Speichermedium geschrieben werden und welche Bedingungen dabei einzuhalten bzw. durchzusetzen sind. Falls das Betriebssystem beim Einhängen des Dateisystems bestimmte Operationen ausführen muss, so erläutern Sie diese ebenfalls. **4 P**

Aufgabe 6 – Nebenläufigkeit **15 Punkte**

a) Nennen Sie jeweils einen Nachteil der folgenden Mechanismen für wechselseitigen Ausschluss. **3 P**

- Interrupts ausschalten

- Atomare Instruktionen

- Dekker-Algorithmus

b) In einem Supermarkt wird neue Ware von **einem** Lieferant angeliefert und von **zwei** Lageristen eingeräumt. Dabei stehen auf der Laderampe insgesamt 10 Plätze zum Ablegen von Paketen zur Verfügung.

Ergänzen Sie folgenden Code um die nötige Synchronisierung zwischen dem Lieferant und den Lageristen mittels Semaphoren (Datentyp `sem_t`). Fügen Sie außerdem Code ein, um die Pakete auf der Laderampe abzulegen bzw. dieser zu entnehmen. Dabei sollen alle Beteiligten maximal parallel arbeiten und kein aktives Warten (*busy waiting*) auftreten. **9 P**

```
int n_frei =      ; int n_voll =      ;
paket_t plaetze[     ];



void lieferant(paket_t ware) {


    plaetze[n_frei] = ware;


}

void lagerist() {


    paket_t ware = plaetze[n_voll];


    einraeumen(ware);


}
```

c) Zwei Threads rufen parallel die nebenstehende Funktion auf. Nennen Sie Aufrufe von `f`, die zu einer Verklemmung (*deadlock*) führen können. Jeder Knoten besitzt dabei einen binären Semaphore, der mit `down()` angefordert und `up()` freigegeben wird. **3 P**

```
1 void f(Knoten *k1, Knoten *k2) {
2    k1->down();
3    k2->down();
4    // Zugriff auf k1 und k2
5    k2->up();
6    k1->up();
7 }
```