Name: Matrikel-Nr.: Studiengang:

Klausur Betriebssysteme und Sicherheit, 19. 07. 2022

— Bearbeitungszeit: 90 Minuten — Prüfer: Prof. Dr. Schirmeier, Dr. Köpsell —

1	2	3	4	5	6	$\sum$
19	13	17	16	12	13	90

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

Aufgabe 1 – Sicherheit

19 Punkte

a) Warum ist es **notwendig**, Angreifermodelle bei der Erstellung einer Sicherheitslösung zu berücksichtigen? **1 P**

b) Nennen und erläutern Sie vier Kriterien, die zur Beschreibung eines Angreifermodells benutzt werden können. **4 P**

c) Welche Analysen sind zur Identifikation von Risiken durchzuführen? Mit welchen beiden Aspekten können die ermittelten Risiken eingeschätzt und bewertet werden? **2 P**

d) RSA wird in der einfachen, unsicheren Variante als Konzelationssystem verwendet. Die folgenden Parameter sind gegeben: $p = 3$, $q = 23$, $k_e = 5$. Der Schlüsseltext $c = 2$ ist zu entschlüsseln. Wie lautet die resultierende Nachricht? **4 P**

e) Welche Angriffe sind möglich, wenn RSA in dieser einfachen, unsicheren Variante eingesetzt wird? Beschreiben Sie in knapper Form die Angriffe und die entsprechenden Gegenmaßnahmen! **4 P**

f) Nennen und erläutern Sie zwei Datenschutzprinzipien. **4 P**

Aufgabe 2 – Virtueller Speicher **13 Punkte**

Eine hypothetische Architektur verwende virtuelle Adressen von 48 Bit Länge. Eine zweistufige Adressübersetzung wird verwendet, um virtuelle Adressen in physische zu übersetzen. Die Indizes sind jeweils 16 Bit lang. Es sollen einzelne Bytes des virtuellen Adressraumes adressiert werden können.
Hinweise:

- Bei den folgenden Aufgaben genügt die Angabe von Zweierpotenzen.
- Es ist immer auch eine sinnvolle Einheit anzugeben.

a) Wie groß ist der virtuelle Adressraum? **1 P**

b) Wie groß ist eine Seite? **1 P**

c) Wie viele Einträge hat eine Seitentabelle einer Stufe? **1 P**

Die folgenden Regionen sind für einen Adressraum definiert. Daneben gibt es keine weiteren Regionen.

Code	0x0000 0001 0000 - 0x0000 0003 FFFF
Data	0x0000 0005 0000 - 0x0000 0005 FFFF
Stack	0xFFFF FFFA 0000 - 0xFFFF FFFF FFFF

d) Geben Sie für jede der folgenden Adressen an, ob auf diese zugegriffen werden kann. Füllen Sie dafür das entsprechende Feld mit einem J (ja) oder einem N (nein) aus.

Adresse	Zugriff möglich?
0x0000 0002 ABCD	
0x0000 0004 FFEE	
0xFFFF 0123 4567	
0xFFFF FFFF FFFE	

4 P

Der Adressraum habe die folgenden Seitentabellen. Die Wurzel-Seitentabelle ist an der Adresse 0x0816 0000.

Addr.: 0x0816 0000

0x0	0x0000 0012	p
0x1	0x0000 0066	
...		
0xFFFF	0x0000 000C	p

Addr.: 0x0000 000C

0x0	0x0000 00A0	
...		
0xFFF8	0x0000 000D	p
...		
0xFFFF	0x0000 00A0	p

Addr.: 0x0000 0012

0x0	0x0F01 FBAD	
0x1	0x0000 ABC0	p
0x2	0xBEEF 0000	p
0x3	0x0000 0123	p
0x4	0x0000 A110	p
...		
0xFFFF	0xA0B0 00FF	

Dabei sind gültige Einträge mit dem Bit p (*present*) gekennzeichnet. Nicht genannte Einträge haben dieses Bit nicht gesetzt. Ein Zugriff ist nur gültig, wenn in den Einträgen aller Stufen das *present*-Bit gesetzt ist.

e) An welchen virtuellen Adressen liegen eingeblendete Seiten, die nicht zu den oben genannten Regionen gehören? **2 P**

f) Nennen Sie die physische Adresse, auf die die folgenden lesenden Zugriffe jeweils zugreifen. Sollte ein Zugriff ungültig sein, geben Sie den Grund an.

- 0xFFFF 0000 1234:
- 0x0000 0002 0F00:
- 0xFFFF FFFF 0005:
- 0x0000 0007 5432:

4 P

Aufgabe 3 – Scheduling

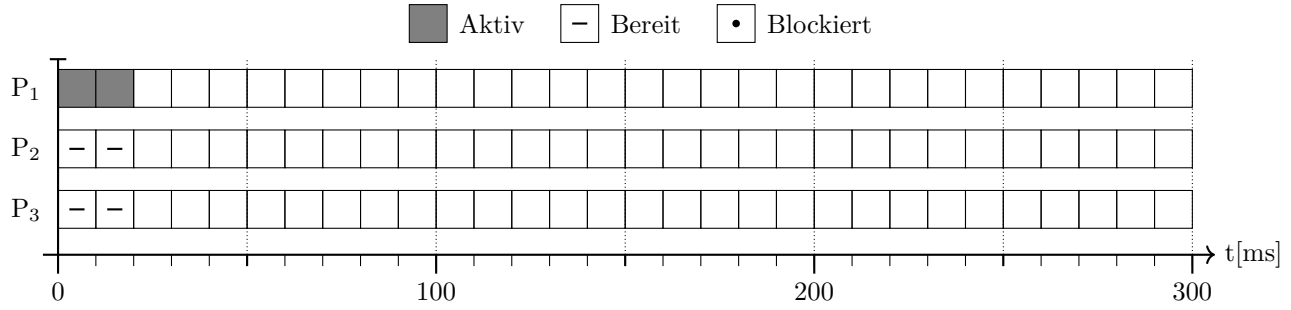
17 Punkte

a) Erklären Sie *knapp* in eigenen Worten, was präemptives von nicht-präemptivem Scheduling unterscheidet. Nennen Sie für beide Varianten beispielhaft jeweils eine Scheduling-Strategie. **3 P**

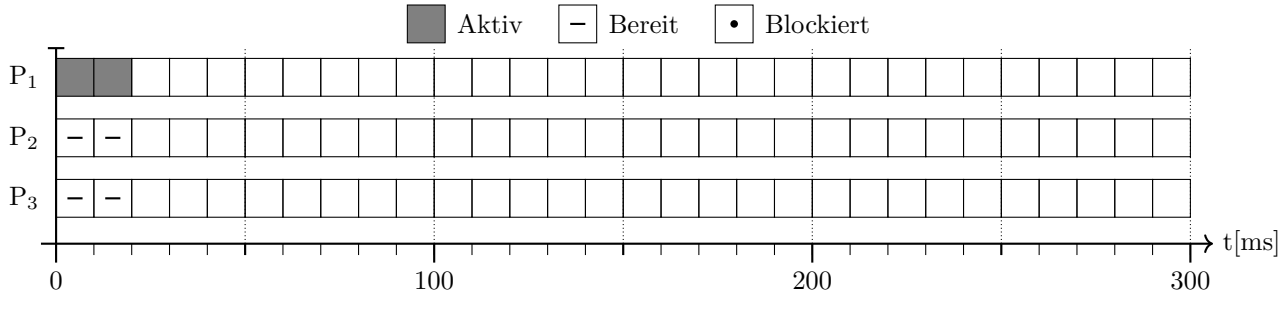
b) Ein Einprozessor-Betriebssystem verwaltet drei Prozesse P_1, P_2 und P_3. Die Prozesse treffen in dieser Reihenfolge im System ein und sind alle zum Zeitpunkt $t = 0$ rechenbereit. Die Prozesse wiederholen sich unendlich lange. Nach jedem CPU-Stoß führt der Prozess einen E/A-Stoß von 20 ms durch. Die CPU-Stöße (in ms) der Prozesse sind in der folgenden Tabelle angegeben:

Prozesse	P_1	P_2	P_3
CPU-Stöße	40	80	90

Zeichnen Sie in das folgende Gantt-Diagramm ein, wie die drei Prozesse P_1, P_2 und P_3 bearbeitet werden würden, wenn das Scheduling nach der „Virtual Round Robin"-Strategie vorgenommen wird. Die gewählte Zeitscheibe beträgt 30 ms.
Hinweis: Die ersten beiden Zeiteinheiten sind bereits fertig ausgefüllt. **7 P**



<u>Ersatzdiagramm</u> (Streichen Sie ungültige Lösungen deutlich durch):



In einem Einprozessor-Echtzeitsystem sind vier periodische Tasks T_1, T_2, T_3 und T_4 einzuplanen mit folgenden Werten (p_i Periodenlänge, t_i Bearbeitungszeit, Periodenende = Zeitschranke):

$p_1 = 6$, $t_1 = 1$ $\quad$ $p_2 = 12$, $t_2 = 3$ $\quad$ $p_3 = 9$, $t_3 = 2$ $\quad$ $p_4 = 18$, $t_4 = 1$

Zwischen den Tasks bestehen keine Abhängigkeiten und sie sind an beliebiger Stelle unterbrechbar.

c) Ist die Taskmenge einplanbar mittels statischer Prioritäten? Falls ja, schlagen Sie einen Algorithmus vor und geben Sie die Zuordnung der Prioritäten an. Ansonsten begründen Sie Ihre Aussage.
Hinweis: $2^{1/2} \approx 1{,}41$; $\quad 2^{1/3} \approx 1{,}26$; $\quad 2^{1/4} \approx 1{,}19$; $\quad 2^{1/5} \approx 1{,}15$ **3 P**

d) Die Taskmenge aus Teilaufgabe (c) soll nun mit dynamischen Prioritäten eingeplant werden. Außerdem sollen, wenn möglich, noch zusätzliche Tasks hinzugefügt werden. Diese zusätzlichen Tasks T_n haben alle die gleichen Eigenschaften ($p_n = 100$, $t_n = 1$). Es muss jedoch beachtet werden, dass jeder zusätzliche Task T_n die Bearbeitungszeit des Task T_1 um 0,1 erhöht, da dieser Daten aus T_n verarbeiten muss. Um wie viele solcher Tasks kann die Taskmenge maximal erweitert werden, ohne die Einplanbarkeit zu gefährden? Begründen Sie Ihre Aussage. **4 P**

Aufgabe 4 – Prozesse und Unix **16 Punkte**

a) Erklären Sie *knapp* in eigenen Worten den Unterschied zwischen den Begriffen Programm und Prozess. **2 P**

b) Die folgenden Shell-Befehle geben die letzten zehn Vorkommen des Wortes „klausurrelevant“, die in der Datei `protokoll` stehen, aus. Sie nutzen dazu die temporäre Datei tmp:

```
grep klausurrelevant < protokoll > tmp
tail -n 10 < tmp
```

Geben Sie einen funktional äquivalenten Shell-Befehl an, der keine temporäre Datei verwendet. **2 P**

c) Warum führt das Betriebssystem einen Prozess nach dessen Terminierung noch in der Prozessverwaltung auf und weshalb ist der Aufruf von `wait()` in dem Zusammenhang notwendig? **3 P**

d) In der Vorlesung wurden verschiedene Prozessmodelle behandelt. Gegeben sei ein Programm, das durch mehrere, voneinander unabhängige Kontrollflüsse realisiert ist. Diese können als schwergewichtige, leichtgewichtige oder federgewichtige Prozesse ausgeführt werden. Geben Sie für jeden dieser Fälle an, ob die links aufgeführten Eigenschaften zutreffen, indem Sie in das entsprechende Feld ein J (ja) oder ein N (nein) eintragen. **3 P**

	schwergewichtige Prozesse	leichtgewichtige Prozesse	federgewichtige Prozesse
Mehrere Kontrollflüsse teilen sich einen Adressraum.			
Das Programm lässt sich durch die Verwendung von Multiprozessor-Hardware beschleunigen.			
Die Blockierung eines Kontrollflusses führt zur Blockierung des ganzen Programms.			

Gegeben sei folgendes Programm. Gehen Sie von einer fehlerfreien Ausführung aus. Des Weiteren wurden die Präprozessordirektiven zur besseren Lesbarkeit weggelassen.

Hinweise:

- Die Funktion `wait` blockiert den aufrufenden Prozess so lange, bis sich einer seiner Kindprozesse beendet. Existiert kein solcher, so kehrt die Funktion sofort zurück.
- Der beispielhafte Aufruf von `printf("x: %d ", y)` gibt `"x: <Variable y> "` auf der Konsole aus, wobei `<Variable y>` mit der zu diesem Zeitpunkt in Variable `y` gespeicherten Zahl zu ersetzen ist.

```
1  int a = 10;
2
3  void foo() {
4     a = a + 2;
5     printf("f: %d ", a);  // Gibt "f: <Variable a> " aus
6  }
7
8  void bar() {
9     a = 815;
10    printf("b: %d ", a);  // Gibt "b: <Variable a> " aus
11 }
12
13 int main() {
14    pid_t pid = fork();
15    if ( pid == 0 ) {
16       foo();
17    } else {
18       wait();
19       bar();
20    }
21    printf("m: %d ", a);  // Gibt "m: <Variable a> " aus
22    exit(0);
23 }
```

e) Welche Bedeutung hat der Rückgabewert von `fork()`, der in Zeile 15 abgefragt wird? **2 P**

f) Geben Sie alle möglichen vollständigen Ausgaben des Programms an. **4 P**

Aufgabe 5 – Ein-/Ausgabe und Dateisysteme **12 Punkte**

a) Nennen sie zwei Hardwarefunktionalitäten, die für die Kommunikation zwischen dem Betriebssystem und einem Eingabe-/Ausgabe-Gerät verwendet werden. **2 P**

Gegeben sei folgender Zustand eines Unix-ähnlichen Dateisystems. Die Festplatte nutzt eine Blockgröße von 4 KiB. Einträge in einem Verzeichnisblock sind jeweils 32 Byte groß. Die dargestellten Blöcke enthalten entweder den Superblock (SB), Inodes (IB), Verzeichniseinträge (VB) oder eine Allokations-Bitmap für Blöcke (AB) bzw. für Inodes (IAB). Weitere Blöcke (z.B. für Dateiinhalte) existieren, sind aber nicht dargestellt.

Block	0	1	2	3	4	5	6
Typ	SB	AB	IAB	IB	IB	VB	VB
Inhalt	free: 80 root: I_0	0 – 11	0, 1, 2	$I_0 \rightarrow 5, 32$ $I_1 \rightarrow 6, 64$ $I_2 \rightarrow 7 – 10, 16000$	$I_3 \rightarrow 11, 3000$	`files` $\rightarrow I_1$	`bar` $\rightarrow I_2$ `tr` $\rightarrow I_3$

Inode-Verweis für Wurzelverzeichnis

Blöcke 0 bis 11 belegt

Inodes 0 bis 2 belegt

Blockzeiger für Inodes, Größe (in Byte)

Abbildung von Dateinamen auf zugehörige Inodes

b) Ein Nutzer möchte gerne die ersten 8 KiB der Datei `/files/bar` einlesen. Welche Blöcke (Blocknummer) müssen dafür von dem Betriebssystem gelesen werden? **6 P**

c) Bei der Übertragung von Daten an eine rotierende Festplatte muss das Betriebssystem die physikalischen Eigenschaften des Gerätes berücksichtigen. Dafür kann der Treiber die Aufträge entsprechend vorsortieren, um so eine bessere Schreib- oder Leseauslastung des Gerätes zu erreichen. Nennen Sie ein Verfahren, das genutzt wird, um die Daten für die Übertragung an die Festplatte zu sortieren. **1 P**

In einem Unix-ähnlichen System gibt es genau drei Nutzer: Alice, Bob und Charlie. Alle diese Nutzer sind in der Gruppe `user`. Alice und Bob sind zusätzlich noch in der Gruppe `admin`. Alice möchte gerne eine Datei `foo` mit Bob teilen und legt diese dafür in folgendem Verzeichnis an:

Name	Besitzer	Gruppe	Rechte
`files`	`alice`	`user`	`rwx rwx rwx`

d) Geben Sie passende Rechte, Besitzer und Gruppe (gemäß Unix-Rechtesystem) für die Datei `foo` an, sodass nur Alice die Datei lesen und schreiben darf und Bob die Datei lesen darf. Alle anderen Nutzer im System (in diesem Fall nur Charlie) dürfen keinen Zugriff auf die Datei haben. **3 P**

Aufgabe 6 – Synchronisation **13 Punkte**

a) Nennen Sie jeweils ein Beispiel für einen Synchronisationsmechanismus mit aktivem und mit passivem Warten. **2 P**

b) Nennen Sie einen Vorteil und einen Nachteil von Synchronisation mit passivem gegenüber aktivem Warten. **2 P**

Es ist eine C++ Klasse `sem_t` gegeben, die einen zählenden Semaphor implementiert. Die Klasse besitzt zwei Methoden, `wait` und `signal`, die wie in der Vorlesung definiert die typischen Funktionen eines Semaphor zur Verfügung stellen.

c) Vervollständigen Sie folgenden Code unter Verwendung der Klasse `sem_t` um wechselseitigen Ausschluss für den markierten kritischen Abschnitt zu gewährleisten. Achten Sie darauf den seriellen Abschnitt so kurz wie möglich zu dimensionieren. Geben Sie auch an, wie der Zähler der `sem_t` Variable initialisiert werden soll. **3 P**

```
sem_t lock(    );

void my_important_function() {
    
    work_item_1();
    
    work_item_2(); /* <- Kritischer Abschnitt */
    
    work_item_3();
    
}
```

d) Das folgende Programm mit mehreren Threads nutzt die Funktion `printf` um eine Zeichenkette an den Nutzer auszugeben. Synchronisieren Sie die Threads mit Hilfe der Klasse `sem_t` so untereinander, dass nach Beendigung des Programms die Zeichenkette „Come to the dark side, we have cookies.“ entsteht. Geben Sie dabei für jede Variable der Klasse `sem_t` auch den initialen Wert des Zählers mit an. **6 P**

```
/* Initialisierung der sem_t-Variablen */
```

T_1

```
void do_print1() {

    printf("Come to ");

    printf("side, ");

}
```

T_2

```
void do_print2() {

    printf("dark ");

    printf("cookies.");

}
```

T_3

```
void do_print3() {

    printf("the ");

    printf("we ");

    printf("have ");

}
```