Probeklausur Betriebssysteme und Sicherheit, 25. 01. 2022

1	2	3	4	5	6	$\sum$
4	8	10	9	12	9	52

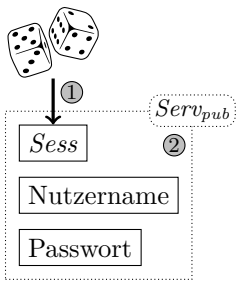
Alle Aussagen sind so ausführlich wie nötig, aber so knapp wie möglich zu begründen.

Aufgabe 1 – Kryptographie und Sicherheit

4 Punkte

Fletnix, ein Anbieter für den Verleih von Videos über das Internet, betreibt einen Server mit den Kunden- und Videodaten. Zur Nutzung des Dienstes stellt er seinen Kunden das ausführbare Programm *DRaMa* bereit. Verbindungen zum Server sollen stets sicher verschlüsselt erfolgen.

Aus Effizienzgründen kommt ein hybrides Verschlüsselungssystem zum Einsatz: Der öffentliche RSA-Schlüssel des Servers ($Serv_{pub}$) ist in *DRaMa* fest integriert. Beim Verbindungsaufbau erzeugt das Programm zunächst einen neuen, zufälligen Sitzungsschlüssel *Sess* ①. Anschließend werden die Anmeldeinformationen (Nutzername und Passwort) sowie *Sess* unter Verwendung von $Serv_{pub}$ verschlüsselt ② und an den Server gesandt. Der Server entschlüsselt die Nachricht und prüft die Anmeldeinformationen. Für alle nachfolgende Kommunikation wird *Sess* verwendet.



a) Nennen Sie einen geeigneten kryptografischen Algorithmus, der für die Verschlüsselung der Kommunikation mittels *Sess* eingesetzt werden kann. **1 P**

b) Welches Problem träte auf, wenn $Serv_{pub}$ nicht in *DRaMa* integriert wäre, sondern vor jeder Anmeldung vom Server geladen würde? **1 P**

c) Eva findet das beschriebene Verfahren viel zu aufwendig. Sie schlägt vor, stattdessen einmalig einen Schlüssel (*Serv*) für ein symmetrisches Verschlüsselungsverfahren zu generieren und diesen in *DRaMa* zu hinterlegen. *Serv* soll dann direkt für die Absicherung aller Kommunikation mit dem Server genutzt werden. Wie beurteilen Sie diesen Vorschlag? **2 P**

Aufgabe 2 – Unix

8 Punkte

```
1  int main() {
2     int pid = fork();
3     printf("A");
4
5     if (pid < 0) {
6        printf("F");
7        exit(1);
8     } else if (pid == 0) {
9        printf("B");
10       wait();
11       exit(0);
12    } else {
13       printf("C");
14       wait();
15    }
16
17    printf("D");
18    exit(0);
19 }
```

a) Wählen Sie aus den vorgegebenen Ausgaben alle aus, die bei Ausführung des nebenstehenden Unix-Programms entstehen können und notieren Sie die zugeordneten Buchstaben. **5 P**

Hinweis:

- Die Funktion `wait` blockiert den aufrufenden Prozess so lange, bis sich einer seiner Kindprozesse beendet. Existiert kein solcher, so kehrt die Funktion sofort zurück.
- Nehmen Sie an, dass die Ausgabe von `printf` nicht gepuffert wird.

(a) ABACD	(e) ACABD	(i) ACADB
(b) AF	(f) AFD	(j) AACBD
(c) AACCDD	(g) ABBA	(k) ACDAB
(d) AABCD	(h) AFAF	(l) ABABDD

```
1  int main() {
2     int fd, pid;
3     pid = fork();
4
5     // Öffnen der existierenden
6     // und leeren Datei 'buffer'
7     fd = open("buffer", O_WRITE);
8
9     if (pid == 0) {
10       write(fd, "foo", 3);
11    } else {
12       wait();
13       write(fd, "bar", 3);
14    }
15
16    return 0;
17 }
```

b) Welchen Inhalt hat die Datei `buffer` nach Beendigung des nebenstehenden Unix-Programms? Wenn es mehrere Möglichkeiten gibt, nennen Sie diese. Wenn nicht, erklären Sie kurz warum. **3 P**

Aufgabe 3 – Scheduling **10 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 relativen Deadline. Die Parameter der Tasks beschreiben jeweils die Periode p und die konstante Bearbeitungszeit e der Jobs.

T_1: $p_1 = 6, e_1 = 2$ T_2: $p_2 = 16, e_2 = 2$ T_3: $p_3 = 4, e_3 = 1$

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

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. **3 P**

b) Ist eine Einplanung dieser Task-Menge mit statischen Prioritäten überhaupt möglich? Begründen Sie Ihre Antwort. Geben Sie im Fall einer erfolgreichen Einplanung eine entsprechende Prioritätszuteilung an.

HINWEIS: Sie können folgende Werte annehmen: $\sqrt{2} < 1{,}42$ und $\sqrt[3]{2} < 1{,}26$ **4 P**

c) Die Task T_1 soll häufiger ausgeführt werden. Bis zu welcher Periode p_1 ist eine Einplanung mit *dynamischen Prioritäten* möglich? **3 P**

Aufgabe 4 – Seitenersetzung **9 Punkte**

Wir betrachten nun ein System welches über 5 Rahmen verfügt und diese mit Hilfe des Arbeitsmengenmodells verwaltet. Dabei nutzt das System eine Fenstergröße (Δ) von 2 Referenzen. Im System existieren 3 Prozesse (A,B,C), von denen jeder bisher auf genau eine Seite zugegriffen hat (siehe vorgegebene Spalte „Init“).

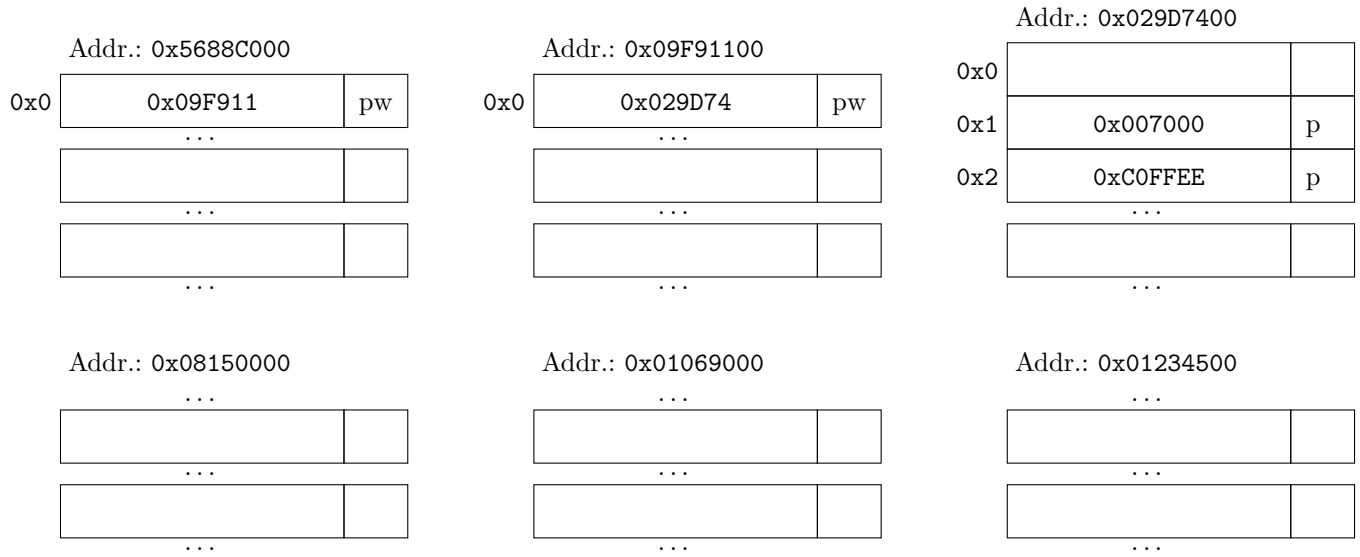
a) Vervollständigen Sie die Zuordnung der Seiten zu Rahmen in der folgenden Tabelle. Tragen Sie dazu in jeder Spalte ein, in welchem Rahmen die entsprechende Seite liegt bzw. in welchen sie eingelagert wird. Markieren Sie Seitenfehler durch ein Kreuz in der Zeile *PF*. **9 P**

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

Aufgabe 5 – Virtueller Speicher **12 Punkte**

Ein hypothetisches System verwende virtuellen Speicher mit 32-Bit-Adressen und dreistufige Tabellen zur Adressübersetzung. Die Indizes aller Stufen seien jeweils 8 Bit groß und der Speicher sei Byte-adressierbar. In den Tabellen werden die Zugriffsrechte eines Eintrags durch die Bits „present“ (`p`) und „writable“ (`w`) bezeichnet. Bei einem Zugriff werden diese Rechte jeweils auf allen Stufen überprüft.

Einem Prozess stehen folgende Seitentabellen zur Verfügung. Alle nicht aufgeführten oder frei gelassenen Einträge werden dabei als ungültig („not present“) angenommen. Der Einsprungpunkt für die Adressübersetzung, also die Tabelle der ersten Stufe, liegt an Adresse `0x5688C000`.



Weiterhin habe der Prozess die nebenstehenden Regionen:

`0x00000100` – `0x000002FF`	Code (lesbar)
`0x00000300` – `0x000004FF`	Heap (lesbar, schreibbar)
`0x7FFFFF00` – `0x7FFFFFFF`	Stack (lesbar, schreibbar)

a) Prüfen Sie für jeden der folgenden Zugriffe, ob dieser durch die MMU ausgeführt werden kann. Geben Sie in diesem Fall die zugehörige physische Adresse an. Für nicht unmittelbar auflösbare Adressen, prüfen Sie zunächst, ob das Betriebssystem Änderungen an den Seitentabellen vornehmen würde und tragen Sie diese ggf. in obiges Schema ein. Nennen Sie dann ebenfalls die sich ergebende physische Adresse des Zugriffs.

Ist ein Zugriff unzulässig, so genügt es dies anzugeben. Diese Zugriffe werden nicht weiter betrachtet und führen auch nicht zum Abbruch des Programms.

Hinweis: Freie Rahmen stehen an den Adressen `0x74E35B00`, `0xD8415600` und `0xC5635600` zur Verfügung.
12 P

1. Lesend auf `0x00000262`:
2. Lesend auf `0x000102CD`:
3. Schreibend auf `0x00000123`:
4. Schreibend auf `0x7FFFFF0A`:
5. Schreibend auf `0x00000500`:

Aufgabe 6 – Synchronisation **9 Punkte**

a) Nennen Sie die Anforderungen, die an ein Lock gestellt werden. **2 P**

b) Gegeben ist die unten stehende Implementierung eines Locks. Welche der Anforderungen erfüllt die gegebene Implementierung nicht? Beschreiben Sie ein Szenario, in dem die Implementierung fehlerhaft arbeitet. **3 P**

```
1  int lck = 0; // globale Variable
2
3  void lock() {
4      while (lck == 1) {
5      }
6      lck = 1;
7  }
8
9  void unlock() {
10     lck = 0;
11 }
```

c) Das Primitiv `cmpxchg(&var, alt, neu)` vergleicht zunächst `var` und `alt`. Nur wenn sie gleich sind, wird der Wert von `neu` an `var` zugewiesen. In jedem Fall gibt die Funktion den alten Wert von `var` zurück.

Nutzen Sie `cmpxchg`, um obige Implementierung zu korrigieren. Welche Eigenschaft muss `cmpxchg` dabei aufweisen? **4 P**