FernUniversität in Hagen

Fakultät für
Mathematik und Informatik

Lehrgebiet Kooperative Systeme
Prof. Dr. Jörg M. Haake

Klausur zum Kurs

Betriebssysteme (1802)

am 17. September 2011

Klausurort:	
Vorname Name:	
Adresse:	
Matrikelnummer:	

Aufgabe	1	2	3	4	5	6	7	8	Summe
Punkte	25	25	30	35	15	36	20	14	200
bearbeitet									
erreichte Punkte									

Hinweise zur Bearbeitung der Klausur zum Kurs Betriebssysteme (1802)

Wir begrüßen Sie zur Klausur *Betriebssysteme* und bitten Sie, diese Hinweise vollständig und aufmerksam durchzulesen, bevor Sie mit der Bearbeitung der Aufgaben beginnen.

1. Prüfen Sie bitte die Vollständigkeit dieser Unterlagen:
   - Deckblatt, diese Hinweise und 8 Aufgaben auf den Seiten 1 bis 18
   - eine Teilnahmebescheinigung zur Vorlage beim Finanzamt
2. Bevor Sie mit der Bearbeitung der Aufgaben beginnen, tragen Sie bitte auf dem Deckblatt Name, Anschrift und Matrikelnummer ein.
3. Falls Sie eine Teilnahmebestätigung wünschen, füllen Sie diese bitte aus.
4. Schreiben Sie Ihre Lösungen bitte auf die Aufgabenblätter bzw. die dafür vorgesehenen Leerseiten und benutzen Sie auch die Rückseiten, wenn der Platz nicht reicht.
5. Auf jedes Blatt, auf dem sich Teile Ihrer Lösung befinden, schreiben Sie bitte oben Ihren Namen und Ihre Matrikelnummer.
6. Wenn Sie eine Prozedur oder ein Programm schreiben sollen, achten Sie auf eine klare Gliederung und eine *ausführliche* Kommentierung.
7. Wenn Sie Zahlenwerte ausrechnen sollen, skizzieren Sie auch den Rechenweg.
8. Als Hilfsmittel ist unbeschriebenes Konzeptpapier zugelassen.
9. Zum Bestehen der Klausur reichen 100 von 200 Punkten auf jeden Fall aus.

Wir wünschen Ihnen bei der Bearbeitung der Klausur viel Erfolg!

Aufgabe 1: Betriebssysteme 8 + 5 + 12 Punkte

Geben Sie bitte auf jede Frage eine kurze Antwort.

1. Geben Sie mindestens 4 Hardware-Ressourcen an, die vom *Betriebssystem* verwaltet werden.

2. Viele moderne Betriebssysteme, z. B. UNIX/Linux monolithisch, d. h. jede Komponente des Systems läuft im *Kern*, insbesondere die Gerätetreiber. Wodurch ist so ein Betriebssystem fehleranfällig?

3. Welche drei Aufgaben von Betriebssystemen können mit Hilfe des *Unterbrechungs-Mechanismus* erledigt werden?

Aufgabe 2: **Prozesse** **6 + 5 + 9 + 5 Punkte**

Geben Sie bitte auf jede Frage eine kurze Antwort.

1. Was ist ein *Prozess*? Welche Informationen über einen Prozess sind für das Betriebssystem wichtig und wo werden sie gespeichert?

2. In welcher Datenstruktur werden Informationen über *alle vorhandenen Prozesse* abgelegt?

3. Wenn ein *Prozesswechsel* stattfindet, muss ein bestimmtes Ereignis vorliegen. Geben Sie drei solche Ereignisse an.

4. Warum können die *Threads* eines Prozesses effizienter miteinander kommunizieren als separate Prozesse?

Aufgabe 3: Scheduling 10 + 10 + 10 Punkte

Geben Sie bitte auf jede Frage eine kurze Antwort.

1. In einem *nicht-präemptiven System* ist es möglich, dass ein Prozess in eine Endlos-Schleife gerät und den Prozessor für sich allein behält. Kann so ein Problem auch in einem *präemptiven System* entstehen? Falls ja, unter welchen Umständen?

2. Wenn man die *SJF*- und die *FCFS-Strategie* als Prioritäts-Strategien auffasst, wie sind dann die *Prioritäten* festgelegt?

3. Wir nehmen an, dass eine Prozessumschaltung S Zeiteinheiten benötigt und betrachten das *Round-Robin-Verfahren* mit dem Zeitquantum Q. Warum ist die CPU-Effizienz dann keinesfalls größer als $\frac{Q}{S+Q}$?

   Wenn der maximale *CPU burst* bekannt ist, sagen wir $T_{\max}$, wie groß ist dann höchstens die CPU-Effizienz, egal wie groß Q gewählt wird?

Aufgabe 4: Hauptspeicher 10 + 10 + 15 Punkte

1. Wir nehmen an, dass in einem *Paging-System* eine virtuelle Adresse $v = (s, d)$ 32 Bit lang ist, wobei s die Seitennummer und d der Offset sind. Wie viele Seiten gibt es, wenn der Offset n Bit lang ist? Warum beeinflusst die Auswahl der Zahl n die *interne Fragmentierung* und den Speicherbedarf der *Seitentabelle*?

2. Wie wird der Speicherschutz bei *zusammenhängenden Speicherzuweisungen* realisiert? Wie ist der Speicherschutz beim Paging-Verfahren?

3. In einer andauernd wiederholten Schleife benutzt ein Prozess immer wieder n Speicherseiten nacheinander in derselben Reihenfolge, bei jedem Schleifendurchlauf jede Seite einmal.

   Ungünstigerweise stehen dem Prozess nur $n - 1$ Seitenrahmen zur Verfügung. Mit wie vielen Seitenfehlern pro Schleifendurchlauf müssen wir nun bei Verwendung der LRU- oder der FIFO-Auslagerungsstrategie rechnen? Könnte eine Zufallsstrategie bessere Ergebnisse liefern?

Aufgabe 5: **Synchronisation** **15 Punkte**

Betrachten wir den Algorithmus von Peterson zur *Synchronisierung*:

```
var turn: integer; /* wer ist gerade dran? */
    flag[0], flag[1]: boolean; /* Interesse */
    flag[0] := false; /* Initialisierung */
    flag[1] := false;
```

Prozess(i) für $i = 0, 1$:

```
repeat

   /* enter_critical_section */
   flag[i] := true; /* Interesse zeigen */
   turn := i;       /* Flag setzen */
   while (flag[1-i] and turn = i) do begin end; /* busy waiting */

   kritischer Abschnitt;

   /* leave_critical_section */
   flag[i] := false;

   unkritischer Abschnitt;

until false;
```

Begründen Sie, warum dieser Algorithmus die folgende Bedingung erfüllt:

> Wenn Prozess 0 und Prozess 1 gleichzeitig versuchen, in den kritischen Abschnitt einzutreten, so wird innerhalb einer endlichen Zeit eine Entscheidung getroffen, welcher Prozess zuerst den kritischen Abschnitt betritt.

Aufgabe 6: Dateisysteme 10 + 8 + 9 + 9 Punkte

1. Gegeben ist ein sehr vereinfachter Ausschnitt eines FAT-basierten Dateiverzeichnisses, bei dem nur die Dateinamen und jeweils rechts davon die Nummern der zugehörigen Startsektoren angegeben sind:

a.tex	5	b.pdf	12	c.txt	16	d.prog	4	e.html	11

Außerdem ist ein Anfangsausschnitt der *File Allocation Table* gegeben, in der die erste Zeile die Plattensektornummern und die zweite Zeile die FAT-Einträge sind.

0	1	2	3	4	5	6	7	8	9	10	...
3	nil	bad	21	17	13	1	free	10	nil	6	...

...

...	11	12	13	14	15	16	17	18	19	20	21
...	19	18	0	nil	free	8	9	14	nil	free	nil

(a) Geben Sie für die fünf Dateien des Verzeichnisses jeweils die zugehörigen Plattensektoren an, und zwar in der Reihenfolge, in der die Daten der Datei abgespeichert sind.

(b) Betrachten Sie die folgenden Paare aus Dateinamen und Nummern eines Bytes der Datei:

(b.pdf, 1998), (b.pdf, 2137), (e.html, 2137), (a.tex, 7512)

Geben Sie jeweils die Nummer des Plattensektors an, in dem sich das Byte befindet. Die Größe eines Blocks beträgt 2 KByte ($= 2^{11}$ Byte). Die Bytenummer einer Datei beginnt immer bei 0.

2. Folgende drei Dateien sind jeweils in mehreren Blöcken mit den angegebenen Blocknummern auf der Festplatte gespeichert.

f.tex:	2	7	3	1										
g.txt:	20	11	5	17	29	19	12	8	4	30	37	9	6	13
h.fig:	35	14	15	16	21	22	23	24	25	26	27	28		

Hierbei ist die Reihenfolge der Blocknummern wichtig: der erste Block von f.tex steht im Plattenblock mit Nummer 2, der zweite hat Nummer 7 usw. Die Größe eines Blocks beträgt wieder 2 KByte ($= 2^{11}$ Byte).

(a) Wie sehen die direkten und die indirekten Sektoradressen in den drei zugehörigen i-nodes aus?

(b) Gegeben sind die folgenden Paare aus Dateinamen und Bytenummern:
(f.tex, 600), (g.tex, 20481), (h.fig, 18433)
Wie viele Plattenblöcke müssen jeweils gelesen werden, um diese Bytes zu finden? Wir nehmen an, dass sich die inodes der Dateien schon im Hauptspeicher befinden.

Aufgabe 7: Sicherheit 8 + 8 + 4 Punkte

Die beiden primären Ziele von Sicherheitsmechanismen in einem Rechnersystem sind die Kontrolle von Zugriffen auf das System und von Zugriffen auf Informationen innerhalb des Systems.

Antworten Sie bitte auf die folgenden Fragen so kurz wie möglich.

1. Was ist der Unterschied zwischen *Identifikation* und *Authentisierung*? Für welches der oben genannten Sicherheitsziele kann man diese Sicherheitsfunktionen einsetzen?

2. Was bedeuten *diskretionäre Zugriffs- und Informationsflusskontrollen*? Für welches der genannten Ziele kann man sie einsetzen?

3. Warum kann man mit diskretionären Zugriffskontrollen einen Angriff durch *Trojaner* nicht verhindern?

Aufgabe 8: Kommandosprachen 7 + 7 Punkte

1. Das Kommando `make` ist ein Beispiel für eine Kommandoprozedur unter UNIX. Was ist das Funktionsprinzip von `make`?

2. Schreiben Sie ein `Makefile` für die folgende Anwendung.

   Es gibt eine ausführbare Datei namens `prog`, die immer wieder neu gebunden wird, wenn sich eines der Objektmodule `a.o` und `b.o` geändert hat. Für das Binden wird das Kommando

   ```
   cc a.o b.o -o prog
   ```

   ausgeführt.

   Die Objektmodule sind wiederum abhängig von den Quelldateien `a.c` und `b.c` und werden durch Kompilieren erzeugt mit den Befehlen

   ```
   cc -o a.o a.c
   cc -o b.o b.c
   ```