FernUniversität in Hagen

Fakultät für Mathematik und Informatik

Lehrgebiet Kooperative Systeme

Prof. Dr. Jörg M. Haake

FernUniversität in Hagen 58084 Hagen

Informatikzentrum
Universitätsstr. 1
58084 Hagen

Vorname Name

Straße Hausnr.

PLZ Ort

Klausur zum Kurs

Betriebssysteme (1802)

am 19. September 2009

Klausurort:	
Vorname Name:	
Matrikelnummer:	

Aufgabe	1	2	3	4	5	6	7	Summe
Punkte	35	30	10	30	40	45	10	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 7 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 (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: **Prozesse** **15 + 10 + 10 Punkte**

1. Ein Betriebssystem hat die Aufgabe, Prozesse im System zu kontrollieren und zu koordinieren. Welche Informationen sind wichtig dafür und in welcher Datenstruktur werden sie gespeichert?

2. Wann findet eine Prozessumschaltung statt?

3. Was ist der Unterschied zwischen einem Prozesswechsel und einem Moduswechsel?

Aufgabe 2: Hauptspeicher 15 + 15 Punkte

Welche Mechanismen unterstützen die folgenden Anforderungen zur Speicherverwaltung?

1. Es soll möglich sein, einem Prozess an einer beliebigen Stelle des physischen Hauptspeichers einen Speicherbereich zuzuweisen, dies soll transparent für den Programmier sein.

2. Der Speicherbereich eines Prozesses muss vor dem Zugriff durch andere Prozesse geschützt werden. Unterscheiden Sie hier *zusammenhängende Speicherzuweisung* und *Paging*.

Aufgabe 3: **Synchronisation** **10 Punkte**

Synchronisieren Sie die folgenden Prozesse A und B mit Semaphoren.

Prozess A	Prozess B
`1. Anweisung a1;`	`1. Anweisung b1;`
`2. Anweisung a2;`	`2. Anweisung b2;`

Bei der Ausführung der Programme müssen folgende Regeln eingehalten werden: Die Anweisung `a1` muss vor der Anweisung `b2` und die Anweisung `b1` muss vor der Anweisung `a2` ausgeführt werden. Die Ausführungsreihenfolge von `a1` und `b1` kann beliebig sein, ebenso die von `a2` und `b2`.
Deklarieren und initialisieren Sie also passende Semaphore und fügen Sie die notwendigen `up`- und `down`-Anweisungen in den obigen Programmcode ein.

Aufgabe 4: Synchronisation 15 + 15 Punkte

Betrachten wir das Sushi-Bar-Problem mit den folgenden Bedingungen:

- Eine Sushi-Bar hat 10 Plätze.
- Solange zunächst noch ein leerer Platz vorhanden ist, darf ein neuer Gast sich einen Platz nehmen.
- Sobald alle 10 Plätze besetzt sind, müssen danach zunächst alle Plätze leer werden, bevor sich ein weiterer Gast einen Platz nehmen darf.

Das folgende Programm ist ein Vorschlag zur Lösung.

```
1. waiting, eating: integer; (* Anzahl von wartenden und essenden Gästen *)
2. mutex, wait_next: semaphor;
3. must_wait: boolean;       (* Der Zustand der Bar *)
4. n: integer  (* Hilfsvariable *)

5. (* Initialisierung *)
6. waiting := 0; eating := 0; (* Anfangs ist die Anzahl der Gäste 0 *)
7. must_wait := false;        (* Am Anfang ist die Bar leer *)
8. mutex := 1; wait_next := 0;(* Neue Gäste müssen warten, wenn nicht *)
9.                            (* alle Gäste ihre Plätze verlassen haben *)
10.
11. down(mutex);             (* Schütze den Zugriff auf must_wait *)
12. if (must_wait) {         (* Alle Plätze sind besetzt *)
13.   waiting := waiting + 1;  (* Erhöhe die Anzahl der Wartenden *)
14.   up(mutex);             (* Gib Mutex wieder frei *)
15.   down(wait_next);       (* Warte, bis alle Plätze wieder frei sind *)
16.   down(mutex);           (* Schütze den Zugriff auf waiting *)
17.   waiting := waiting - 1;(* Die nächste Runde kann wieder anfangen *)
18. }
19. eating := eating + 1;  (* Noch sind Plätze frei *)
20. if (eating = 10 ) {    (* Sind alle Plätze jetzt besetzt? *)
21.    must_wait := true;(* Falls ja, müssen die weiteren Gäste warten *)
22. } else {must_wait := false}
23. up(mutex)              (* Gib Mutex wieder frei *)
24.
25. Kritischer Abschnitt;
26.
27. down(mutex);           (* Will den Platz verlassen *)
28. eating := eating - 1  (* Einer ist mit dem Essen fertig *)
29. if (eating = 0) {     (* Der als letzter gehende Gast muss das Signal *)
30.    if (waiting < 10) { (* für die nächste Runde geben *)
31.        n := waiting   (* Nur höchstens 10 wartende Gäste *)
32.     } else {          (* dürfen die neue Runde starten *)
33.       n=10;
34.     }
```

```
35.
36.     while (n > 0) {
37.        up(wait_next);   (* Die wartenden Gäste werden aufgeweckt*)
38.        n := n-1;        (* und die nächste Runde kann starten *)
39.     }
40.     must_wait := false; (* Alle Gäste haben die Plätze verlassen *)
41. }
42. up(mutex);     (* Gib Mutex wieder frei *)
```

1. Betrachten Sie Zeilen 16 und 40. Konstruieren Sie ein Szenario, in dem mehr als 10 Gäste in den kritischen Abschnitt eintreten können.

2. Korrigieren Sie das Programm so, dass das Problem mit mehr als 10 Gästen im kritischen Abschnitt gelöst wird. (Hinweise: Sind die Zeilen 16 und 17 nötig? Könnte eine `else`-Anweisung vor Zeile 19 hilfreich sein? Kann die Zeile 40 so bleiben?)

Aufgabe 5: Dateisysteme 10 + 10 + 20 Punkte

Auf einer Festplatte gibt es eine kleine Partition von 512 MByte mit einer Blockgröße von 512 Byte.

1. Wie viel Platz belegt die File Allocation Table (FAT) für ein Dateisystem auf der Partition mindestens, wenn eine Sektoradresslänge von 4 Byte verwendet wird?

2. Jetzt soll die Partition mit einem Dateisystem mit i-nodes verwaltet werden, wobei wieder die Sektoradresslänge 4 Byte benutzt wird. Wird die dreifach-indirekte Adresse im i-node für die sehr große Dateien nun wirklich benötigt?

3. Wie werden mit i-nodes die Sektoradressen einer 1128 KByte großen Datei abgelegt?

Aufgabe 6: Sicherheit 15 + 15 + 15 Punkte

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

Antworten Sie die folgenden Fragen bitte so einfach wie möglich.

1. Was ist der Unterschied zwischen *Identifikation* und *Authentisierung*? Für welches der oben genannten Sicherheitsziele kann man diese Mechanismen 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 7: Kommandosprachen 10 Punkte

Schreiben Sie ein `Makefile` für die folgende Anwendung.

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

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

ausgeführt.
Die Objektmoduln sind wiederum abhängig von den Quelldateien `a.c` und `b.c` und werden durch Kompilieren erzeugt, z. B.

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