5. Semester Informatik

Betriebssysteme

Schriftliche Prüfung WS 1998/99

GEORG-SIMON-OHM FACHHOCHSCHULE NÜRNBERG

Prüfer: Prof. Dr. Kern / Prof. Dr. Wienkop

Prüfungstermin: 29.1.98, 11:00 Uhr
8 Aufgabenblätter, Bearbeitungszeit: 90 Min.
Hilfsmittel: Vorlesungsmitschrift, Vorlesungsskript/-foliensatz und Taschenrechner

---

Name: .....

Vorname: .....

Matr.-Nr. .....

(Bitte liefern Sie die vollständigen Informationen auf einem der Standardprüfungsbögen!)

Die Aufgaben sind z.T. auf diesem Aufgabenblatt zu bearbeiten.
Geben Sie daher unbedingt dieses Blatt wieder mit ab!

---

1 Shell-Programmierung (Gewichtung: niedrig)

Erstellen Sie eine Shell-Prozedur, die bei Aufruf durch den Superuser (der die dazu nötigen Rechte hat) jedes Terminal im System zunächst prüft, ob es betriebsbereit ist (also, ob darauf geschrieben werden kann; Tip: test-Kommando). Wenn ja, gibt die Shell-Prozedur den Text "Meldung an", dann den Terminalnamen, Datum und Uhrzeit und schließlich das Wort "Feierabend!" aus. Dann teilt sie dem Aufrufer mit: "Meldung an <Terminalname> gelungen." Wenn auf das Terminal nicht geschrieben werden kann, wird lediglich dem Aufrufer mitgeteilt: "<Terminalname> nicht betriebsbereit!".

Die Namen aller Terminals beginnen mit tty, und die zugehörigen Gerätedateien sind alle im Katalog /dev enthalten.

Beispiel für die Ausgabe am Zielterminal:

```
Meldung an /dev/tty1B: Mi 13. Jan 1999, 10:30 Uhr. Feierabend!
```

Mitteilung an den Aufrufer, z.B.:

```
Meldung an /dev/tty1B gelungen.
```

oder

```
/dev/tty1B nicht betriebsbereit
```

```
set `date`

for i in /dev/tty*
do
  if test -w $i
  then
    echo Meldung an $i: $1 $3.$2 $6 $4 Uhr. Feierabend! >> $i
    echo Meldung an $i gelungen
  else
    echo $i nicht betriebsbereit
  fi
done
```

2 Aufrufschnittstelle (Gewichtung: niedrig)

Bitte geben Sie für den Systemaufruf

```
size_t fread (void *buffer, size_t size, size_t count, FILE *stream );
```

(Hierbei bezeichnet 'size' die Größe eines zu lesenden Elements (d.h. eines records) und 'count' die Anzahl der zu lesenden Elemente!)
size_t ist definiert als vorzeichenloser, ganzzahliger Typ, z.B.: `typedef unsigned int size_t;`

mögliche **Fehlerursachen** bzw. **Rückgabezustände** an. Sie können davon ausgehen, daß die Korrektheit bzgl. Anzahl und Typ der übergebenen Parameter vom Compiler überprüft wurde.

Lösung:

Fehlerursachen:

- stream ungültig
- Datei nicht zum Lesen geöffnet
- ungültige Adresse
- HW-Lesefehler
- EOF erreicht

Rückgabewerte:

- Anzahl der tatsächlich gelesenen Datensätze (vgl. auch EOF)

3 Cache (Gewichtung: mittel)

Der Speicher eines Rechners bestehe aus 256 KByte Cache mit 25ns Zugriffszeit und 32 MByte Hauptspeicher mit 100ns Zugriffszeit. Der Programmdurchsatz soll beschleunigt werden, indem

- entweder bei unverändertem Hauptspeicher der Cache mit Speicherbausteinen mit 10ns Zugriffszeit
- oder aber bei unverändertem Cache der Hauptspeicher mit Speicherbausteinen mit 60ns Zugriffszeit

ausgerüstet wird. Der Preis für die Umrüstung braucht hier nicht berücksichtigt werden.

a) Geben Sie in Abhängigkeit von der Trefferrate r des Cache-Speichers (mit 0 <= r <= 1) an, welche Umbauvariante den Durchsatz mehr erhöht.

b) In welche Richtung verschiebt sich die Trefferrate r unter sonst gleichbleibenden Verhältnissen, wenn der Cache-Speicher auf 512 KByte ausgebaut wird?

c) Manche Chipsätze für die Cache-Verwaltung können nur Hauptspeicher bis 64 MByte handhaben. Eventuell weiterer Speicher wird dann ohne Cache direkt adressiert. Ist bei einem solchen System ein Speicherausbau auf z.B. 128 MByte sinnvoll (Begründen Sie Ihre Antwort wiederum unter Berücksichtigung der Trefferrate r. Nehmen Sie dabei an, daß der Speicher vom Betriebssystem gleichverteilt benutzt wird!) ?

Lösung

a) t < 40/55 (0.72) → Hauptspeicher aufrüsten
t > 40/55 (0.72) → Cache aufrüsten

b) Trefferrate wird besser

c) Speicherausbau auf 128 Mbyte ist nicht sinnvoll, da sich bei gleicher Trefferrate der durch den Cache erzielte Geschwindigkeitsvorteil halbiert!

4 Vergleich von E/A-Verfahren (Gewichtung: hoch)

Gegeben sei eine Platte mit 200 Spuren (Zylindern). Im Moment befindet sich der Schreib-/ Lesekopf auf Spur 100; der vorhergehende Zugriff bezog sich auf Spur 132. Es sind Zugriffswünsche für folgende Spuren eingetroffen (ohne Unterscheidung von Schreib- oder Leseaufträgen):

187 165 21 34 101 102 34

Alle 80 ms (= 50 Spurwechsel) komme ein neuer Auftrag hinzu. Diese Aufträge seien:
10 20 30 40 50.

Falls ggf. ein neuer Auftrag genau zu einem Zeitpunkt eintrifft, wo sich der Kopf gerade auf dieser Position befindet, so wird dieser Auftrag sofort mit ausgeführt.

Bestimmen Sie für die Verfahren
- einfache Fahrstuhlstrategie (Pickup)
- volle Fahrstuhlstrategie (Scan)

die Abarbeitungsreihenfolge der Aufträge unter Berücksichtigung der neu eingetroffenen Aufträge und berechnen Sie die Anzahl Σ der gewechselten Spuren.

Lösung einfache Fahrstuhlstrategie:

100 – 101 – 102 – 165 – 187 – 34 – 34 – 21, Summe: 304

Lösung volle Fahrstuhlstrategie:

34 – 34 – 21 – 10 – 20 – 101 – 102 – 165 – 187 – 50 – 40 – 30, Summe: 424

5 Vergleich der FAT-, Unix- und NTFS Dateisysteme (Gewichtung: mittel)

Vergleichen Sie die Dateisysteme von MSDOS, Unix System V und NTFS am Beispiel einer **1048 KByte großen Datei** in einer 1023 MByte großen Partition. Bei Unix können Sie annehmen, daß die Blockgröße 1 KByte beträgt und daß 4 Byte pro Verweis verwendet werden.

Ferner können Sie von nicht-fragmentierten Dateien ausgehen!

Die Dateisysteme sollen hinsichtlich folgender Kriterien verglichen werden:	MSDOS	Unix	NTFS
a) Größe des Directory-Eintrags bei MSDOS bzw. Bestandteile des Directory-Eintrags bei Unix	32 Byte	Dateiname + Inode	
b) Plattenplatzbedarf (Cluster/Blöcke) sowie Verschnitt (KByte) (der Platzbedarf für FAT, MFT und INode-Eintrag darf vernachlässigt werden.	66.....Cluster 8..... Kbyte	1054.....Blöcke 0..... KByte	1048.....Cluster 0..... KByte
c) Aufwand beim Positionieren auf eine beliebige Stelle **innerhalb** der Datei, d.h. wieviele Einträge in FAT/MFT-Eintrag/ INode müssen im **worst-case** betrachtet werden. Geben Sie zusätzlich an, ob es sich beim Zugriff auf diese Einträge um Hauptspeicher oder Plattenzugriffe handelt	66 Hauptsp. Zugriffe	3 Plattenzugriffe	1 Hauptspeicherzugriff

6 Ablaufplanung (Gewichtung: hoch)

Auf einem Unix System V – System mit einer Zeitscheibendauer von 1sec und 50 Ticks / Zeitscheibe laufen drei Prozesse A, B und C.

C ist ein reiner Rechenprozeß ohne sonstige Blockierungen und benötigt eine reine Rechenzeit von 60 Ticks. Alle drei Prozesse werden zum Zeitpunkt 0 gestartet.

Die "Programme" der Prozesse A und B sind nachfolgend abgedruckt. Zu den angegebenen Zeiten (in Anzahl Ticks seit Prozeßstart) erfolgen jeweils Semaphorfunktionsaufrufe. Die bei einem V() – Aufruf evtl. wieder freigegebenen Prozesse werden lediglich "bereit"-gesetzt, d.h. sie erhalten nicht sofort wieder die CPU!

Initialisierung der Semaphoren:

```
S0 := 0
S1 := 1
```

Prozeß A:	**Prozeß B:**
... 10: P(S0) ... 20: P(S1) ... 30: V(S1) ... 40: "*Prozeßende*"	... 20: V(S0) P(S1) ... 50: V(S1) ... 60: "*Prozeßende*"

Die Startprioritäten seinen wie folgt vergeben: A: 60, B: 62, C: 64
Alle drei Prozesse werden zum Zeitpunkt 0 gestartet.

Bitte tragen Sie den Ablauf der drei Prozesse in das untenstehende Diagramm ein, benennen Sie die Gründe für einen Prozeßwechsel und berechnen Sie für jeden Prozeß den Parameter 'P' – Penalty Ratio.

Das Ausfüllen soll gemäß folgendem **(*nicht-korrekten*!) Muster** geschehen:

	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	P
Proz. A					Sem	Block															P = 1.0
C	0					30															
Prio	60					67															

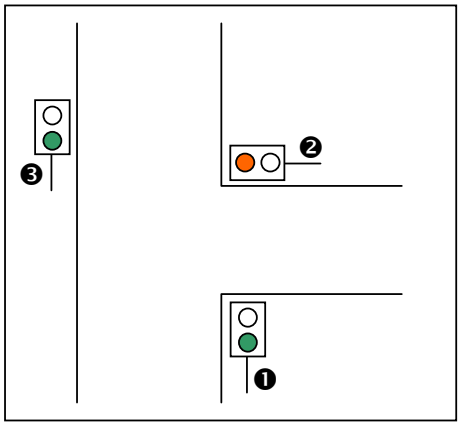
Hier bitte Ihre Lösung:

Ticks:	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	P
Proz. A																					P =
C																					
Prio	60																				
Proz. B																					P =
C																					
Prio	62																				
Proz. C																					P =
C																					
Prio	64																				

Lösung

Ticks:	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	P
Proz. A																					P = 3,5
C					10	5				15	7				27	13					
Prio	60					62					63					66					
Proz. B																					P = 2,0
C					40	20					10										
Prio	62					72					67										
Proz. C																					P = 2,6
C					0	0				40	20				30	15					
Prio	64					64					74					71					

7 **Ampelsteuerung** (Gewichtung: hoch)



Eine Straßenkreuzung nebenstehender Gestalt ist durch drei Ampeln gesichert. Dabei sollen folgende Regeln gelten:

- Die Ampeln 1 und 3 werden synchron betrieben, zeigen also immer die gleiche Farbe an
- Die Ampeln 1 und 2 werden gegenläufig "getaktet". Wenn 1 rot zeigt, muß 2 grün zeigen und umgekehrt. Analog gilt dies natürlich auch für die Ampeln 3 und 2.
- Die Länge einer Ampelphase betrage immer 30 Sekunden.
- **Jede Ampel wird durch einen eigenen Prozeß** gesteuert. Die Ampeln sollen sich **durch Semaphore miteinander synchronisieren**. Die Synchronisation geschehe nach dem folgenden Verfahren:

  Ampel 1 und Ampel 2 klären über eine Semaphore, wer gerade Grün zeigen darf. Falls Ampel 1 Grün zeigen darf, soll sie dies auch an Ampel 3 weitermelden, worauf Ampel drei dann ebenfalls auf Grün schaltet.

  Am Ende der Grün-Phase von Ampel 1 schaltet sie selbst schon einmal auf Rot um und teilt dies wieder der Ampel 3 mit (Ampel 3 soll ohne Aufforderung von Ampel 1 kein Umschalten durchführen). Dann soll Ampel 1 aus Sicherheitsgründen selbst darauf warten, bis Ampel 3 eine Bestätigung schickt, daß wieder auf Rot umgeschaltet wurde.

  Erst dann darf die Grün-Berechtigung an Ampel 2 weitergegeben werden.

Bitte entwerfen Sie eine verkehrssichere Lösung in PASCAL oder C ähnlichem Pseudocode, die obige Regeln einhält.

Erlaubte Systemaufrufe:
- Semaphoren mit den üblichen Aufrufen P() und V()
- Warte(xx Sek.)
- Setzen der Ampel durch: Ampel = grün/rot

```
Semaphore grün=1, A3grün=0, A3rot=0, A3ack=0;

Process Ampel1()
{
   Ampel = rot;
   forever {
      P(grün);
         Ampel = grün;
         V(A3grün);
         Warte(30Sek);
         V(A3rot);
         P(A3ack);
      V(grün);
      Warte(20Sek);
   }
}
```

```
process Ampel2()
{
  Ampel = rot;
  forever {
    P(grün);
      Ampel = grün;
      Warte(30Sek);
      Ampel = rot;
    V(grün);
    Warte(20Sek);
  }
}

process Ampel3()
{
  Ampel = rot;
  forever {
    P(A3grün);
    Ampel = grün;
    P(A3rot);
    Ampel = rot;
    V(A3ack);
  }
}
```

VIEL ERFOLG!

P.S.: ES WIRD NICHT ERWARTET, DAß SIE ALLE AUFGABEN VOLLSTÄNDIG LÖSEN