5. Semester Informatik

Betriebssysteme

Schriftliche Prüfung SS 2000

GEORG-SIMON-OHM
FACHHOCHSCHULE
NÜRNBERG

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

Prüfungstermin: 18.7.00, 8:30 Uhr
6 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 Betriebssystemarten (Gewichtung: mittel)

Gegeben sind folgende drei Prozesse.

Name	Ankunftszeit	Rechenzeitbedarf	Priorität
A	20	30	2
B	10	20	3
C	30	10	1

(Die Priorität ist nur für den Echtzeitbetrieb relevant, wobei die Zahl 1 die beste Priorität und 3 die schlechteste Priorität kennzeichnet)

Diese Prozesse werden auf drei unterschiedlichen Betriebssystemarten gestartet.

a) Stapel-Betrieb (Batch)

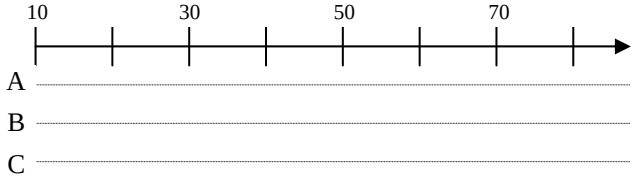
b) Teilnehmer-Betrieb (time sharing) mit Rechenzeitvergabe zu gleichen Teilen

c) Echtzeitbetrieb

Nicht alle der obigen Angaben zu den Prozessen werden bei der jeweiligen Betriebssystemart benötigt! Geben Sie bitte für die jeweilige Betriebssystemart an, wann welcher Prozess beendet ist und wie viel Zeit zwischen Prozessstart und Prozessende vergangen ist (also der Parameter 'T'). Tragen Sie Ihre Ergebnisse bitte direkt in die nachfolgenden Tabellen ein und markieren Sie in der Skizze, wann welcher Prozess bzw. wann welche Prozesse aktiv sind.

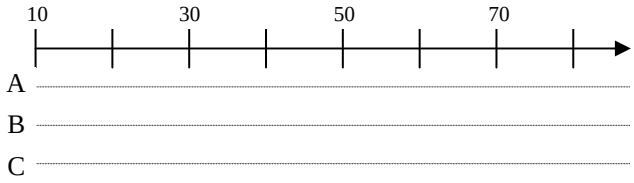
Stapel-Betrieb (Batch)

Name	Ankunft	Ende	T
A	20	60	40
B	10	30	20
C	30	70	40



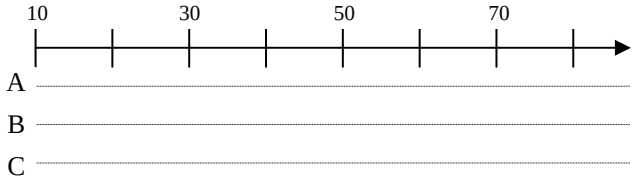
Teilnehmer-Betrieb

Name	Ankunft	Ende	T
A	20	70	50
B	10	45	35
C	30	55	25



Echtzeitbetrieb -Betrieb

Name	Ankunft	Ende	T
A	20	60	40
B	10	70	60
C	30	40	10



2 Shell-Programmierung (Gewichtung: niedrig)

Schreiben Sie ein Shell-Programm 'suche', welches analog zu (aber **ohne (!) Verwendung** von) grep oder ähnlichen Dienstprogrammen eine oder mehrere in der Kommandozeile gegebene Dateien hinsichtlich einer gegebenen Zeichenkette durchsucht. Wird die Zeichenkette in einer Zeile gefunden, so ist der Dateiname, ein Doppelpunkt und die Zeile – in der die Zeichenkette vorkommt – auszugeben.

Der Aufruf von suche hat folgende Syntax: `suche <Zeichenkette> <Dateiname1> ...`

Es muss mindestens eine zu suchende Zeichenkette und ein Dateiname angegeben sein, sonst ist eine Fehlermeldung auszugeben.

Achten Sie bitte darauf, die gelesene Zeile auch dann noch korrekt auf dem Bildschirm auszugeben, wenn diese möglicherweise für die Shell relevante Sonderzeichen (z.B. '*') enthält!

3 Bestimmung einer Byteposition einer Datei in MSDOS und Unix (Gewichtung: hoch)

Geben Sie ein **Programmfragment (in C- oder PASCAL-Pseudocode)** an, wie eine Realisierung eines GetByteAt(unsigned BytePos) – Befehls für Dateien aussehen könnte. Es soll also an eine beliebige Stelle in der Datei positioniert und anschließend das eine Byte an dieser Stelle gelesen und zurückgegeben werden. Hierzu dürfen Sie natürlich seek oder ähnliche Systemaufrufe nicht verwenden, denn so etwas sollen Sie ja hier nachbilden!

Geben Sie bitte **jeweils eine Realisierung für das FAT und das Unix-Dateisystem** an. Sie können dabei davon ausgehen, dass die Datei bereits in einem Katalog gefunden wurde und dass sich der Inhalt der gefundenen Katalogdatenstruktur bereits in der Variable "Kat" im Hauptspeicher befindet.

Bei der Unix-Variante können Sie sich auf Dateien mit einer max. Länge von ~64 MByte, d.h. zwei Indirektionen beschränken. Darüber hinaus können Sie bei Unix von einer Blockgröße von 1KByte und vier Byte großen Adressen ausgehen.

Bitte denken Sie auch an mögliche Fehlersituationen!

Den Zugriff auf benötigte Datenstrukturen können Sie einfach in der Form von Array- oder Strukturkomponentenadressierungen schreiben, z.B. Kat.Datum, Kat.Name, FAT[4711], Inode.Datum, Inode.Block[5] (d.h. Blocknummer des fünften Inode-Verweises), usw. Plattenzugriffe können in der Form LoadInode(345), LoadBlock(777) oder LoadCluster(555) realisiert werden.

Lösungen:

FAT-Dateisystem:

```
char GetByteAt(unsigned Pos)
{
      unsigned Cluster = Kat.ClusterVerw;
      unsigned NthCluster = Pos / ClusterSize;

      while (NthCluster > 0)
            Cluster = FAT[Cluster];
      LoadCluster(Buf, Cluster);
      return Buf[Pos % ClusterSize];
}
```

Unix-Dateisystem:

```
char GetByteAt(unsigned Pos)
{
      Inode = LoadInode(Kat.Inode);
      if (Pos < 10k)
      {
            Buffer = LoadBlock(Inode.Block[Pos / 1024]);
            return Buffer[Pos % 1024];
      }
      else {
            Pos = Pos – 10k;
            if (Pos < 256k)
            {
                  Buffer = LoadBlock(Inode.Block[10]);
                  Buffer = LoadBlock(Buffer[Pos/1024]);
                  return Buffer[Pos % 1024];
            }
            else {
                  Pos = Pos – 256k;
                  if (Pos < 64MByte)
                  {
                        Buffer = LoadBlock(Inode.Block[11]);
                        Buffer = LoadBlock(Buffer[Pos / 1024 / 256k]);
                        Buffer = LoadBlock(Buffer[Pos / 1024]);
                        return Buffer[Pos % 1024];
                  }
            }
      }
}
```

4 Unix- und NT-Dateisysteme (Gewichtung: mittel)

Pro Festplattenblock wird bei dem in der Vorlesung betrachteten Unix-Dateisystem in dem Inode ein Zeiger (4 Byte) auf diesen Datenblock abgelegt und ggf. bei größeren Dateien werden noch zusätzliche Blöcke für die indirekte Adressierung benötigt. Ferner haben wir gesehen, dass bei einer Blockgröße von 1KByte die größte auf diese Weise adressierbare Datei bei ca. 16 GByte liegt.

a) Bitte berechnen Sie den zusätzlichen Speicherbedarf (d.h. Overhead), der sich bei der maximal großen Datei aufgrund der indirekten Adressierung ergibt (ohne des im Inode für die Zeiger benötigten Platzes), d.h. wie viele zusätzliche Blöcke werden bei Blockgröße 1KByte und Zeigerlänge 4 Byte zusätzlich zur Datei benötigt? Welchem Platz in MByte ($=2^{20}$ Byte) entspricht dies?

b) Welcher Platzbedarf für den "Verwaltungs-Overhead" bei der gleichen Dateigröße ergibt sich bei Windows NT unter der Annahme, dass die Datei unfragmentiert ist?

c) Welche Konsequenzen würde bei NTFS ein hoher Fragmentierungsgrad haben? (nur Stichwörter!)

Lösungen:

a) 1 + 256 * (256 + 1) = 65793 =64,25 MByte

b) Kein Zusatzaufwand

c) Weitere Einträge in der Attributliste, die weitere Runs aufnehmen müssen

5 Ablaufplanung (Gewichtung: mittel)

Das Unix-Ablaufplanungsverfahren hat manche Ähnlichkeiten zu dem Round-Robin' Ablaufplanungsverfahren (RR), aber es gibt auch Unterschiede – selbst wenn die Startpriorität der Unix-Prozesse gleich gewählt ist.

Bitte geben Sie eine Situation (d.h. Prozessmenge) an, in denen sich die beiden Verfahren gleich verhalten und eine Situation, in denen die Verfahren zu unterschiedlichen Ergebnissen für die Prozessmenge führen, z.B. hinsichtlich der Ausführungsreihenfolge bzw. des Penalty-Ratios.

Lösungen:

Gleiches Verhalten: zwei langlaufende CPU-intensive Prozesse

Unterschiedlich: ein CPU-intensiver und ein I/O-intensiver Prozess

6 Ablaufplanung

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

Die Prozesse A und B tauschen dabei Daten über eine Datenstruktur (ähnlich einem Fifo) aus, die Platz für 1000 Bytes bereithält. Bei einem "Überlauf" dieses Speichers wird der schreibende Prozess blockiert, bis wieder Platz für mindestens 500 Bytes bereitsteht. Eine vorzeitige Auswahlentscheidung oder gar ein Prozesswechsel wird jedoch beim Unterschreiten dieser Grenze nicht sofort ausgelöst. Ein lesender Prozess wird mindestens solange blockiert, bis Daten aus dem Datenpuffer bereitstehen.

A braucht eine Sekunde reine Rechenzeit für die Bearbeitung seines Prozesses und produziert dabei pro Tick 100 Bytes Daten, die B benötigt. B kann pro Tick 50 Bytes Daten verarbeiten und ist nach dem Lesen und Bearbeiten des letzten Datenpakets ebenfalls beendet. A und B haben die Startpriorität 60 ohne einen nice-Zuschlag. Sie können annehmen, dass A seine Daten jeweils zum Ende eines Ticks weitergibt sowie dass B seine Daten zu Beginn eines Ticks anfordert.

Prozess C hat ebenfalls die Startpriorität 60, jedoch einen nice-Zuschlag von 6! C ist ein reiner Rechenprozess ohne sonstige Blockierungen und benötigt eine reine Rechenzeit von 1 Sekunde. 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				Spei-	cher	voll															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	X			X				X					X					X			P = 180/50
C		10			20	10			20		10			20		10					
Prio	60	65				65			70		65			70		65					
Proz. B		X	X			X	X				X	X				X	X		X	X	P = 200/100
C				20		10		30			15		35			17					
Prio	60			70		65		75			67		77			68					
Proz. C					X				X	X				X	X						P = 150/50
C						5				25	12				32						
Prio	66					68					72										

VIEL ERFOLG!

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