Lösungsskizzen zur Abschlussklausur

Betriebssysteme (BTS)

6. Juli 2007

Name: _____

Vorname: _____

Matrikelnummer: _____

Studiengang: _____

Hinweise:

- Tragen Sie zuerst auf allen Blättern (einschließlich des Deckblattes) Ihren *Namen*, Ihren *Vornamen* und Ihre *Matrikelnummer* ein. Lösungen ohne diese Angaben können nicht gewertet werden.
- Schreiben Sie die Lösungen jeder *Teil*aufgabe auf das jeweils vorbereitete Blatt. Erst wenn Vorder- und Rückseite nicht ausreichen, können die leeren Blätter am Ende der Heftung benutzt werden. In diesem Fall ist ein Verweis notwendig. Eigenes Papier darf nicht verwendet werden.
- Legen Sie bitte Ihren *Lichtbildausweis* bereit.
- Als *Hilfsmittel* sind zwei selbstständig beschriebene DIN-A4-Blätter und Taschenrechner zugelassen.
- Mit Bleistift oder Rotstift geschriebene Ergebnisse werden *nicht* gewertet.
- Die Bearbeitungszeit für die Klausur beträgt *70 Minuten*.
- Stellen Sie sicher, dass Ihr Mobiltelefon ausgeschaltet ist. Klingelnde Mobiltelefone werden als Täuschungsversuch angesehen und der/die entsprechende Student/in wird von der weiteren Teilnahme an der Klausur **ausgeschlossen**!

Bewertung:

1a)	1b)	2a)	2b)	2c)	3)	4)	5a)	5b)	5c)	5d)

6)	7a)	7b)	7c)	8a)	8b)	**Σ**	**Note**

Lösungsskizzen zur Abschlussklausur

Betriebssysteme (BTS)

6.7.2007

MSc Christian Baun

Aufgabe 1 (1+1+1+1 Punkte)

a) Nennen Sie 5 **Ersetzungsstrategien** aus der Cache-Datenverwaltung. Es ist nur eine Aufzählung und keine Beschreibung der Funktionalität verlangt!

b) Es existieren zwei Möglichkeiten für Schreibzugriffe auf den Cache.

- Nennen Sie die beiden **Cache-Schreibstrategien**.
- Beschreiben Sie kurz (1-2 Sätze) die Funktionsweise der beiden Cache-Schreibstrategien.
- Nennen Sie für jede der beiden Cache-Schreibstrategien jeweils eine vorteilige und eine nachteilige Eigenschaft.

Aufgabe 2 (2+1+3 Punkte)

a) Das **2-Zustands-Prozessmodell** ist das kleinste, denkbare Prozessmodell. Zeichnen Sie das 2-Zustands-Prozessmodell mit seinen Zuständen und allen Prozessübergängen.

b) Ist das 2-Zustands-Prozessmodell sinnvoll? Begründen Sie kurz ihre Antwort.

c) Zeichnen Sie das 5-Zustands-Prozessmodell mit seinen Zuständen und allen Prozessübergängen.

Aufgabe 3 (4 Punkte)

Beschreiben Sie die Funktionsweise der beiden UNIX-Systemaufrufe `fork()` und `exec()`. Heben Sie dabei die Unterschiede zwischen beiden Systemaufrufen hervor.

Aufgabe 4 (2 Punkte)

Die existierenden **Schedulingverfahren** können in zwei grundsätzliche Klassen unterteilt werden. Welche sind das und wie unterscheiden sich diese?

Aufgabe 5 (2,5+2,5+2,5+2,5 Punkte)

Auf einem Einprozessorrechner sollen fünf Prozesse verarbeitet werden. Hohe Prioritäten sind durch hohe Zahlen gekennzeichnet.

Prozess	CPU-Laufzeit (ms)	Priorität
1	4	5
2	8	12
3	4	8
4	8	2
5	6	10
6	4	4

a) Skizzieren Sie die Ausführungsreihenfolge der Prozesse mit einem Gantt-Diagramm (Zeitleiste) für **Round Robin** (Zeitquantum $q = 2$ ms), **FCFS**, **SJF**, **LJF** und **Prioritätengesteuertes Scheduling**.

b) Berechnen Sie die mittleren Laufzeiten der Prozesse.

c) Berechnen Sie die mittleren Wartezeiten der Prozesse.

d) Welche der obigen Scheduling-Verfahren können zu einem Verhungern der Prozesse führen und welche nicht?

Aufgabe 6 (15 Punkte)

In einer Lagerhalle der John Doe Werke werden ständig Pakete von einem Lieferanten angeliefert und von zwei Auslieferern abgeholt. Der Lieferant und die Auslieferer müssen dafür ein Tor durchfahren. Das Tor kann immer nur von einer Person durchfahren werden. Der Lieferant bringt mit jeder Lieferung 3 Pakete zum Wareneingang, an der Ausgabe holt ein Auslieferer jeweils 2 Pakete ab, der andere Auslieferer 1 Paket. Die Prozesse laufen folgendermassen ab:

```
Lieferant
{
  while (TRUE)
  {

    <Tor durchfahren>;

    <Wareneingang betreten>;

    <3 Pakete entladen>;

    <Wareneingang verlassen>;

    <Tor durchfahren>;

    }
}
```

```
Auslieferer_X
{
  while (TRUE)
  {

    <Tor durchfahren>;

    <Warenausgabe betreten>;

    <2 Pakete aufladen>;

    <Warenausgabe verlassen>;

    <Tor durchfahren>;

    }
}
```

```
Auslieferer_Y
{
  while (TRUE)
  {

    <Tor durchfahren>;

    <Warenausgabe betreten>;

    <1 Paket aufladen>;

    <Warenausgabe verlassen>;

    <Tor durchfahren>;

    }
}
```

Es existiert genau ein Prozess `Lieferant`, ein Prozess `Auslieferer_X` und ein Prozess `Auslieferer_Y`.

Synchronisieren Sie diese drei Prozesse, indem Sie die notwendigen **Semaphoren** deklarieren, diese mit Startwerten versehen und Semaphor-Operationen einfügen. Folgende Bedingungen müssen erfüllt sein:

- Es darf immer nur ein Prozess das Tor durchfahren.
- Es darf immer nur einer der beiden Auslieferer die Warenausgabe betreten. Es soll aber zugelassen werden, dass der Lieferant und ein Auslieferer gleichzeitig Waren entladen bzw. aufladen.
- Die Lagerhalle hat nur eine beschränkte Kapazität. es können maximal 10 Pakete aufgenommen werden.
- Es dürfen keine Verklemmungen auftreten, da sonst die John Doe Werke viel Zeit und Geld verlieren.
- Zu Beginn sind keinerlei Pakete in der Lagerhalle vorrätig und das Tor, der Wareneingang und die Warenausgabe sind frei.

Aufgabe 7 (1+1+1 Punkte)

a) Was ist die Aufgabe eines **Mutex**?

b) Welche Zustände kann ein Mutex annehmen?

c) Was sind die Unterschiede zwischen Mutexen und Semaphoren?

Aufgabe 8 (2+4 Punkte)

a) Welche Auswirkungen hat die Größe der **Cluster** im Dateisystem?

b) Was sind **Journaling-Dateisysteme**? Wie ist die Funktionsweise von Journaling-Dateisystemen? Was sind die Vorteile von Journaling-Dateisystemen gegenüber Dateisystemen ohne Journal?

Name: Vorname: Matr.Nr.:

Aufgabe 1)

Punkte: .....

a) *Hier waren 5 der folgenden Ersetzungsstrategien verlangt*

- **OPT** (Optimale Strategie)
- **LRU** (Least Recently Used)
- **LFU** (Least Frequently Used)
- **FIFO** (First In First Out)
- **TTL** (Time To Live)
- **Random**
- **WS** (Working Set)
- **Climb**

b) **Write-Back**: Schreibzugriffe werden nicht direkt an die tieferen Speicherebene weitergegeben. Dadurch kommt es zu Inkonsistenzen zwischen den Daten im Cache und auf dem zu cachenden Speicher. Die Daten werden erst zurückgeschrieben, wenn der betreffende Datenblock aus dem Cache verdrängt wird. $\Longrightarrow$ Zurückschreiben

- **Vorteil**: Höhere System-Geschwindigkeit.
- **Nachteil**: Daten können beim Systemausfall verloren gehen.

Write-Through: Schreibzugriffe werden sofort an die tieferen Speicherebene weitergegeben. $\Longrightarrow$ Durchschreiben

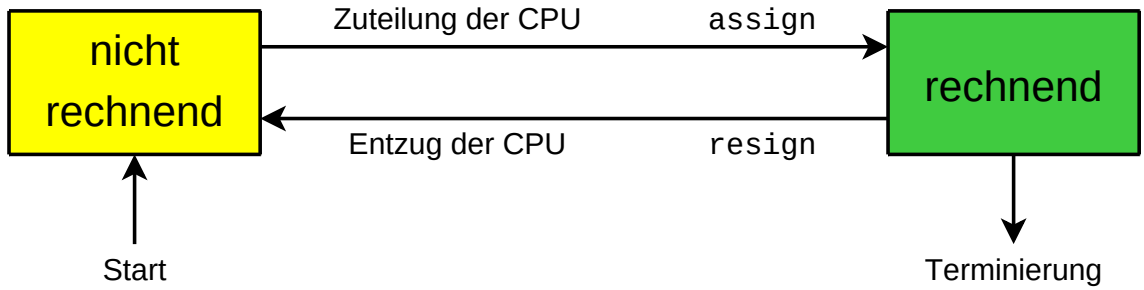
- **Vorteil**: Datenkonsistenz ist gesichert.
- **Nachteil**: Geringere System-Geschwindigkeit.

Name: Vorname: Matr.Nr.:

Aufgabe 2)

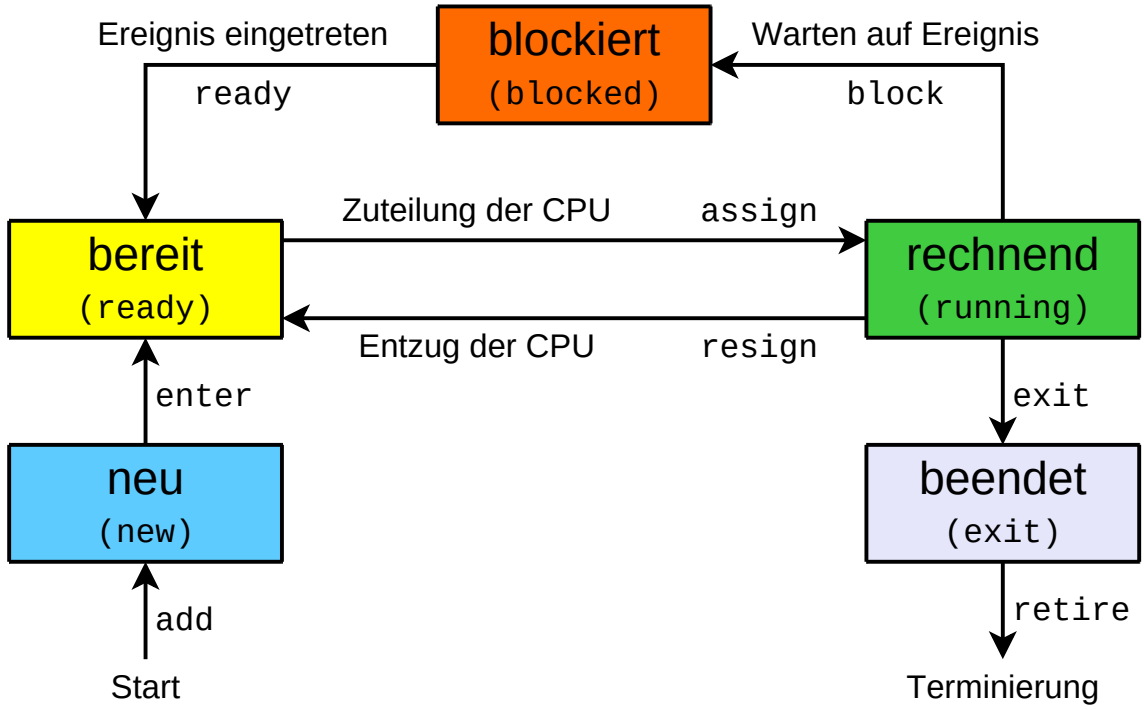
Punkte: .....

a)



b) Das 2-Zustands-Prozessmodell geht davon aus, dass alle Prozesse immer zur Ausführung bereit sind. Das ist aber unrealistisch. Es gibt fast immer Prozesse, die blockiert sind und z.B. auf das Ergebnis eines E/A-Geräts oder eines anderen Prozesses warten.

c)



Name: Vorname: Matr.Nr.:

Aufgabe 3)

Punkte: .....

- Ruft ein Prozess `fork()` auf, wird eine identische Kopie als neuer Prozess gestartet. Nach der Erzeugung eines neuen Prozesses mit `fork()` hat der Kindprozess den gleichen Programmcode und die Befehlszähler haben den gleichen Wert, verweisen also auf die gleiche Stelle im Programmcode. Geöffnete Dateien und Speicherbereiche des Elternprozesses werden für den Kindprozess kopiert und stehen ihm nun getrennt zur Verfügung. Die Speicherbereiche von Kindprozess und Elternprozess sind, wie bei allen anderen Prozessen auch, streng voneianander getrennt. Beide besitzen ihren eigenen Prozesskontext.

- Mit dem Systemaufruf `exec()` wird ein Prozess durch einen anderen ersetzt. Es findet eine **Verkettung** statt. Im Gegensatz zu `fork()`, wo vom aufrufenden Prozess eine identische Kopie als neuer Prozess erzeugt wird, wird bei `exec()` der aufrufende Prozess beendet und ein neuer gestartet. Dieser neue Prozess erbt sogar die Prozess-ID (PID) des aufrufenden Prozesses. Wenn man aus einem Prozess heraus ein Programm starten möchte, ist es notwendig, erst mit `fork()` einen neuen Prozess zu erzeugen und dann mit `exec()` den neuen Prozess zu ersetzen.

Name: Vorname: Matr.Nr.:

Aufgabe 4)

Punkte: .....

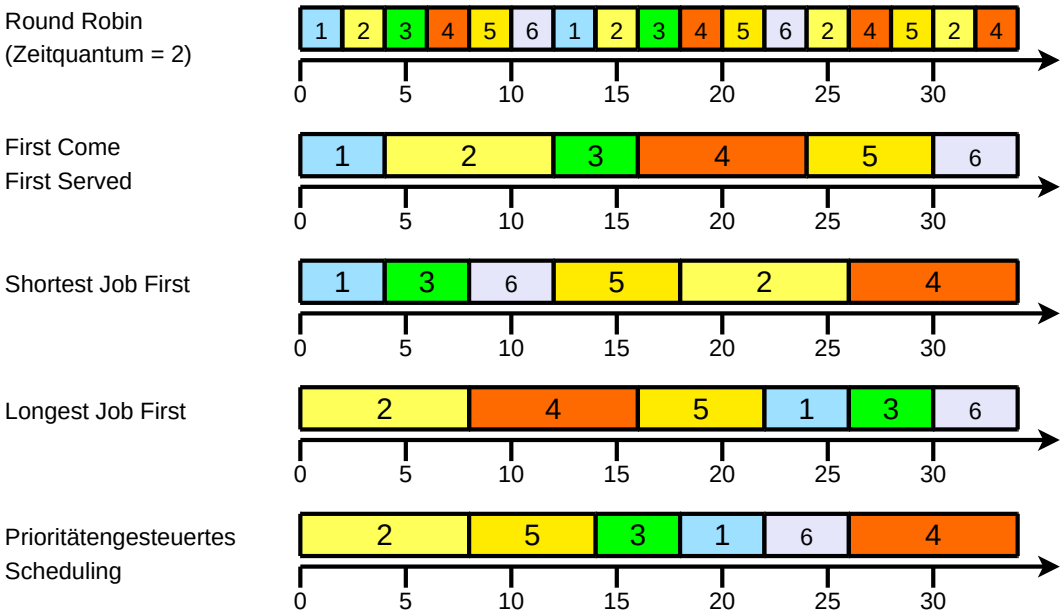
- **Nicht-präemptives Scheduling** (nicht-verdrängendes Scheduling): Ein Prozess, der vom Scheduler die CPU zugewiesen bekommen hat, behält die Kontrolle über diese bis zu seiner vollständigen Fertigstellung. Eine vorzeitige Entziehung der CPU durch den Scheduler ist nicht vorgesehen. Problematisch ist dabei, dass ein Prozess die CPU so lange belegen kann, wie er möchte und andere, vielleicht dringendere Prozesse für lange Zeit nicht zum Zuge kommen.
- **Präemptives Scheduling** (verdrängendes Scheduling): Einem Prozess kann die CPU vor seiner Fertigstellung wieder entzogen werden, um diese anderen Prozessen zuzuteilen. Der Prozess pausiert so lange in seinem aktuellen Zustand, bis ihm wieder die CPU vom Scheduler zugeteilt wird.

Name: Vorname: Matr.Nr.:

Aufgabe 5)

Punkte: .....

a)



b) Laufzeit (Turnaround Time) der Prozesse

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
Round Robin	14	32	18	34	30	24
First Come First Served	4	12	16	24	30	34
Shortest Job First	4	26	8	34	18	12
Longest Job First	26	8	30	16	22	34
Prioritätengesteuertes Scheduling	22	8	18	34	14	26

Round Robin	$\frac{14+32+18+34+30+24}{6}$	$=$	$25,\overline{3}$ ms
First Come First Served	$\frac{4+12+16+24+30+34}{6}$	$=$	20 ms
Shortest Job First	$\frac{4+26+8+34+18+12}{6}$	$=$	17 ms
Longest Job First	$\frac{26+8+30+16+22+34}{6}$	$=$	$22,\overline{6}$ ms
Prior. Scheduling	$\frac{22+8+18+35+14+26}{6}$	$=$	$20,\overline{3}$ ms

Name: Vorname: Matr.Nr.:

Aufgabe 5)

Punkte: .....

c) Wartezeit der Prozesse – Zeit in der `bereit`-Liste

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
Round Robin	10	24	14	26	24	20
First Come First Served	0	4	12	16	24	30
Shortest Job First	0	18	4	26	12	8
Longest Job First	22	0	26	8	16	30
Prioritätengesteuertes Scheduling	18	0	14	26	8	22

Round Robin	$\frac{10+24+14+26+24+20}{6}$	$=$	$19,\overline{6}$ ms
First Come First Served	$\frac{0+4+12+16+24+30}{6}$	$=$	$14,\overline{3}$ ms
Shortest Job First	$\frac{0+18+4+26+12+8}{6}$	$=$	$11,\overline{3}$ ms
Longest Job First	$\frac{22+0+26+8+16+30}{6}$	$=$	17 ms
Prior. Scheduling	$\frac{18+0+14+26+8+22}{6}$	$=$	$14,\overline{6}$ ms

d)

- Zu einem Verhungern der Prozesse kann es bei **SJF**, **LJF** und **Prioritätengesteuertes Scheduling** kommen.
- Bei **Round Robin** und **FCFS** kann es unmöglich zu einem Verhungern der Prozesse kommen.

Name: Vorname: Matr.Nr.:

Aufgabe 6)

Punkte: .....

Man benötigt folgende Semaphore:

- Boolscher Semaphor `tor` zum wechselseitigen Ausschluss des Tores mit Startwert `1`. Zu Beginn ist das Tor frei.
- Boolscher Semaphor `ausgabe` zum wechselseitigen Ausschluss der Warenausgabe mit Startwert `1`. Zu Beginn ist die Warenausgabe frei.
- Semaphor `frei` zum Zählen der freien Plätze in der Lagerhalle mit Startwert `10`. Zu Beginn sind alle Plätze frei.
- Semaphor `belegt` zum Zählen der belegten Plätze in der Lagerhalle mit Startwert `0`. Zu Beginn ist kein Platz belegt.

```
Lieferant
{
  while (TRUE)
  {
    tor.P;
    <Tor durchfahren>;
    tor.V;

    <Wareneingang betreten>;
    frei.P;
    frei.P;
    frei.P;
    <3 Pakete entladen>;
    belegt.V;
    belegt.V;
    belegt.V;
    <Wareneingang verlassen>;

    tor.P;
    <Tor durchfahren>;
    tor.V;
    }
}
```

```
Auslieferer_X
{
  while (TRUE)
  {
    tor.P;
    <Tor durchfahren>;
    tor.V;
    ausgabe.P;
    <Warenausgabe betreten>;

    belegt.P;
    belegt.P;
    <2 Pakete aufladen>;
    frei.V;
    frei.V;

    <Warenausgabe verlassen>;
    ausgabe.V;
    tor.P;
    <Tor durchfahren>;
    tor.V;
    }
}
```

```
Auslieferer_Y
{
  while (TRUE)
  {
    tor.P;
    <Tor durchfahren>;
    tor.V;
    ausgabe.P;
    <Warenausgabe betreten>;


    belegt.P;
    <1 Paket aufladen>;
    frei.V;


    <Warenausgabe verlassen>;
    ausgabe.V;
    tor.P;
    <Tor durchfahren>;
    tor.V;
    }
}
```

Quelle der Aufgabe: TU-München, Übungen zur Einführung in die Informatik III, WS01/02

Name: Vorname: Matr.Nr.:

Aufgabe 7)

Punkte: .....

a) **Mutexe** dienen dem Schutz kritischer Abschnitte, auf denen zu jedem Zeitpunkt immer nur **ein Prozess** zugreifen darf.

b) Mutexe können zwei Zustände annehmen, nämlich **belegt** und **nicht belegt**.

c) Wenn die Möglichkeit eines Semaphors zu zählen nicht benötigt wird, kann man Mutexe vorziehen. Mutexe sind im Vergleich zu Semaphoren effizienter und einfacher zu realisieren. Mutexe können nur zwei Zustände annehmen. Folglich wird nur ein Bit benötigt, um die Information darzustellen.

Name: Vorname: Matr.Nr.:

Aufgabe 8)

Punkte: .....

a)
- Je kleiner die Cluster, desto größer ist der entstehende Overhead.
- Je größer die Cluster, desto mehr Speicher geht durch interne Fragmentierung verloren.

b) Ein Journaling-Dateisystem führt ein Journal über die Daten, auf die Schreibzugriffe durchgeführt werden sollen. Eine zu ändernde Datei behält ihre Gültigkeit, bis die Schreibzugriffe durchgeführt wurden. In festen Zeitabständen wird das Journal geschlossen und die Schreiboperationen werden durchgeführt. Während der Abarbeitung des Journals wird festgehalten, welche Schreiboperationen bereits erfolgreich durchgeführt wurden. Gleichzeitig werden Prüfsummen von Daten von der Änderung erstellt. Mit Hilfe des Journals und den Prüfsummen können nach einem Systemabsturz die zu überprüfenden bzw. wiederherzustellenden Dateien schnell identifiziert und repariert werden und das Journal wenn möglich weiter abgearbeitet werden. Im schlimmsten Fall gehen Änderungsanforderungen, die im Journal vermerkt waren, verloren. Die Dateien auf dem Medium bleiben aber in einem konsistenten Zustand. Das führen eines Journals führt zu geringen Leistungseinbußen. Der Vorteil von Journaling-Dateisystemen ist, dass bei einem Absturz des Betriebssystems nicht alle Daten überprüft, sondern nur die zu dem Zeitpunkt geöffneten Daten repariert werden müssen. Das führt zu großen Zeitersparnissen.