Vorlesung Betriebssysteme I

UNIVERSITÄT SIEGEN

Fachbereich 12, Elektrotechnik und Informatik
Betriebssysteme / verteilte Systeme
Prof. Roland Wismüller

Name: _____

Vorname: _____

Matrikelnummer: _____

Beispielklausur Lösungsskizze

Achtung: die hier angegebenen Lösungen sind teilweise sehr stichpunktartig und sollen Ihnen lediglich Hinweise für die korrekte Antwort geben. In der realen Klausur müssten Sie ggf. ausführlicher antworten!

Eine Gewähr für die Richtigkeit der Antworten kann nicht gegeben werden!

Aufgabe 1: Prozesse und Threads (9 Punkte)

a) (*2 Punkte*) Geben Sie die **drei** wichtigsten Zustände eines Prozesses (oder Threads) an! In welchem der drei Zustände befindet sich ein **neu erzeugter** Prozeß zunächst?

Bereit, rechnend, blockiert
Neu erzeugter Prozeß startet im Zustand „bereit“.

b) (*3 Punkte*) Welche Zustandsübergänge

(i) können durch einen Interrupt (z.B. Uhr, E/A-Gerät) ausgelöst werden?
rechnend → bereit
blockiert → bereit
[bereit → rechnend] (über Scheduler ...)

(ii) können beim **aufrufenden Prozeß/Thread** durch einen Systemaufruf ausgelöst werden?
rechnend → bereit
rechnend → blockiert

(iii) werden durch den (präemptiven) Scheduler des Betriebssystems verursacht?
rechnend → bereit
bereit → rechnend

c) (*4 Punkte*) Geben Sie die einzelnen Schritte an, die das Betriebssystem ausführt, um einen **Prozeß**wechsel durchzuführen!

Ein **Thread**wechsel läuft prinzipiell genauso ab, beim Threadwechsel innerhalb **desselben** Prozesses fehlt aber im Vergleich zum Prozeßwechsel ein wesentlicher Schritt. Welcher?

- Sichern des kompletten CPU-Zustands im Prozeßkontrollblock
- Prozeßzustand aktualisieren, Verwaltungsinformation (CPU-Zeit etc.) aktualisieren, Einreihen in passende Warteschlange
- Scheduler: neuen Prozeß aus Bereit-Liste auswählen und entfernen
- Prozeßzustand aktualisieren: „rechnend“
- Speicherabbildung (Seitentabelle der MMU) aktualisieren
- CPU-Zustand aus Prozeßkontrollblock restaurieren, Benutzermodus und Rückkehr aus dem BS

Beim Threadwechsel innerhalb desselben Prozesses fehlt das Ändern der Speicherabbildung.

Aufgabe 2: Synchronisation (14 Punkte)

a) (*6 Punkte*) Die folgende Realisierung eines binären Semaphors durch einen Monitor soll vervollständigt werden. Durch den Aufruf von **`Lock()`** wird das Semaphor gesperrt und durch **`Unlock()`** wieder freigegeben. Ein Thread kann somit für einen kritischen Bereich durch Einklammern in **`Lock()`** und **`Unlock()`** den wechselseitigen Ausschluß sicherstellen.

Falls das Semaphor bereits belegt ist (**`locked = true`**), soll **`Lock()`** den aufrufenden Thread so lange blockieren, bis ein anderer Thread **`Unlock()`** ausführt, d.h. **`locked = false`** gilt. Realsieren Sie die dazu notwendige Synchronisation über eine **Bedingungsvariable**!

```
monitor BinSema

    condition  unlock;
    boolean    locked;

    procedure  Lock()
    begin
        if (locked)        // oder: while(locked)
            wait(unlock);
            .....
        locked = true;
    end;
    ⋮
```

```
    ⋮
    procedure  Unlock()
    begin
        locked = false;
        signal(unlock);
        .....
    end;

    locked = false;    // Initialisierung
end monitor;
```

b) (*2 Punkte*) Begründen Sie, warum zur Lösung von Aufgabe 2a) in der Monitor-Prozedur **`Lock()`** kein *Busy Waiting* verwendet werden kann! (Tip: was würde passieren, wenn in **`Lock()`** die Zeile

```
while (locked);    // warte in der Schleife bis locked==false
```

eingefügt würde?)

Die Prozeduren einer Monitors stehen unter wechselseitigem Ausschluß. Wenn **`Lock()`** aktiv warten würde, bis **`locked==false`** gilt, gäbe es einen Deadlock, da **`Unlock()`** nicht betreten werden kann, bevor **`Lock()`** verlassen wird.

c) (*6 Punkte*) Gegeben ist der folgende Code für ein Erzeuger-/Verbraucher-Problem mit **unbeschränktem** Puffer. Die Prozedur **`insert_item()`** fügt ein Element in den Puffer ein, die Funktion **`remove_item()`** entfernt das erste Element und gibt seinen Inhalt als Ergebnis zurück. Ergänzen Sie den Code um die notwendige Synchronisation, um sicherzustellen, daß

- die Aufrufe von **`insert_item()`** und **`remove_item()`** unter wechselseitigem Ausschluß stehen,
- Thread 2 beim Versuch, ein Element aus einem **leeren Puffer zu entfernen**, so lange blockiert wird, bis der Puffer nicht mehr leer ist.

Hinweise: Verwenden Sie zur Lösung zwei **Semaphore** und, falls erforderlich, weitere Hilfsvariablen. Lösungen mit *Busy Waiting* werden **nicht** akzeptiert! Eine Synchronisation, um das Schreiben auf einen vollen Puffer zu verhindern, soll **nicht** realisiert werden.

Semaphore und globale Variablen

```
Semaphore mutex = 1;
Semaphore count = 0;
```

Thread 1

```
while (true) {
    item = produce();  // Produziere item
    P(mutex);

    insert_item(item);  // Einfügen in Puffer
    V(count);
    V(mutex);

}
```

Thread 2

```
while (true) {
    P(count);
    P(mutex);

    item = remove_item();  // Entfernen aus Puffer
    V(mutex);

    consume(item);  // Verarbeite item
}
```

Aufgabe 3: Verklemmungen (9 Punkte)

a) (*4 Punkte*) Gegeben sind die folgenden drei Prozesse, die in der angegebenen Reihenfolge Betriebsmittel belegen und wieder freigeben:

Prozeß 1	Prozeß 2	Prozeß 3

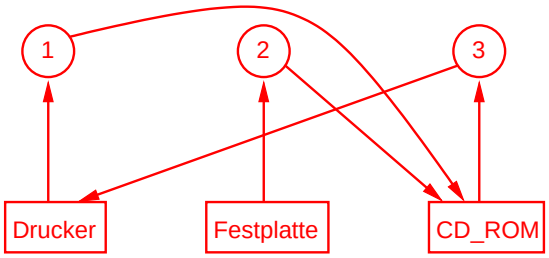
P(Drucker);
P(Festplatte);
// drucke von Plate
V(Festplatte);
P(CD_ROM); // (*)
// drucke von CD-ROM
V(CD_ROM);
V(Drucker);
	P(Festplatte);
	P(CD_ROM); // (*)
	// kopiere von CD auf Platte
	V(CD_ROM);
	V(Festplatte);
		P(CD_ROM);
		P(Drucker); // (*)
		// drucke von CD-ROM
		V(Drucker);
		V(CD_ROM);

Die (ununterbrechbaren) Betriebsmittel sind jeweils nur einmal vorhanden.

Betrachten Sie den Zustand, bei dem sich die Prozesse an den mit (*) markierten Stellen befinden (die P-Operationen sind aufgerufen, aber noch nicht abgeschlossen).

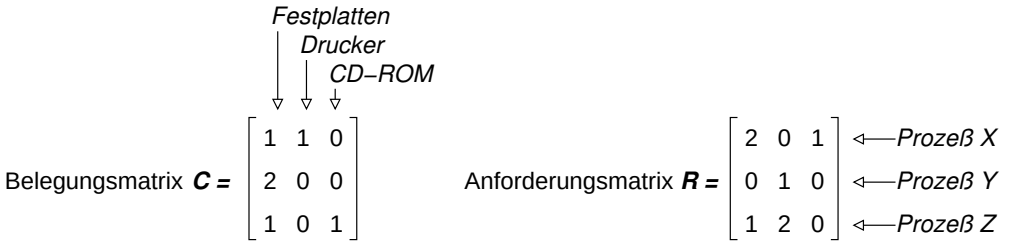
(i) Zeichnen Sie für diesen Zustand einen Belegungs-Anforderungs-Graphen!

(ii) Liegt ein Deadlock vor? Falls ja, zwischen welchen Prozessen? Kurze Begründung!



Es liegt ein Deadlock vor, zwischen den Prozessen 1 und 3 [oder: 1, 2 und 3], da es im Belegungs-Anforderungs-Graph einen Zyklus gibt, der die Prozesse 1 und 3 enthält [und 2 auf 3 wartet].
(oder: da beide Prozesse auf ein Ereignis warten, das nur der andere auslösen kann)

b) (*5 Punkte*) In einem System stehen 5 Festplatten, 2 Drucker und 1 CD-Laufwerk als (ununterbrechbare) Betriebsmittel zur Verfügung. Betrachten Sie den folgenden Belegungszustand:



Die Anforderungsmatrix R beschreibt hier die **maximalen** zukünftigen Forderungen.

(i) Geben Sie den Ressourcenvektor E und den Ressourcenrestvektor A an.

(ii) Führen Sie den Bankiers-Algorithmus aus! Geben Sie an, ob Prozesse zu Ende laufen können und wenn ja, welche!

(iii) Ist der Zustand sicher? Was bedeutet dieses Ergebnis genau?

$E = (5, 2, 1), \quad A = (1, 1, 0)$

1. Zunächst können nur die Anforderungen von Prozeß Y erfüllt werden. Y kann zu Ende laufen. Zustand danach: $A = (3, 1, 0)$.
2. Nun können weder die Anforderungen von Prozeß X noch die von Prozeß Z erfüllt werden.

Der Zustand ist nicht sicher. Das bedeutet, daß es einen Deadlock gibt, wenn alle Prozesse sofort ihre maximalen Anforderungen stellen. (Oder: Das bedeutet, daß es im weiteren Ablauf im ungünstigsten Fall einen Deadlock geben *kann*.)

Aufgabe 4: *Scheduling* (11 Punkte)

Drei Prozesse treffen zu den in der Tabelle angegebenen Zeiten in der Bereitliste eines Schedulers ein. Von jedem Prozeß ist die Bedienzeit (= benötigte Rechenzeit) bekannt. Zusätzlich hat jeder Prozeß eine Priorität (0 stellt die höchste Priorität dar).

Prozeß	Ankunftszeit	Bedienzeit	Priorität
A	0	7	2
B	2	5	1
C	5	3	0

Die Prozesse benutzen nur die CPU und werden nie durch E/A oder sonstige Gründe blockiert. Der Scheduler entscheidet online, d.h. nur aufgrund der zum Scheduling-Zeitpunkt bereits vorliegenden Prozesse.

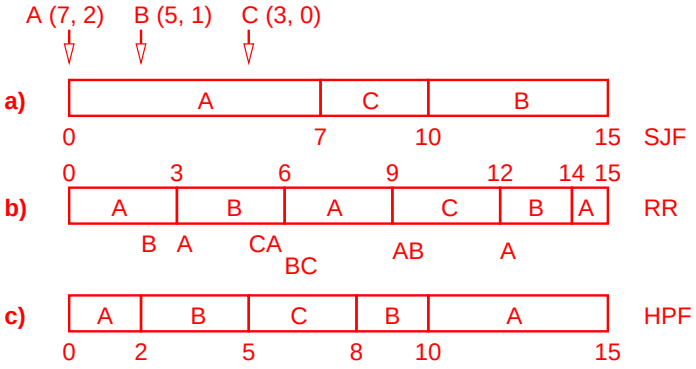
Betrachten Sie die folgenden Scheduler-Strategien:

a) *Shortest Job First* (SJF), nicht-präemptiv,

b) *Round Robin* (RR) mit einer Zeitscheibenlänge (Quantum) von 3,

c) *Prioritätsbasiertes Scheduling* (HPF), präemptiv.

Zeichnen Sie für jede der Strategien den zeitlichen Ablauf der Prozeßausführung als Gantt-Diagramm!



Aufgabe 5: Schutz (2 Punkte)

Definieren Sie den Begriff **Zugriffskontroll-Liste (ACL)**! Welche Informationen sind in einer solchen Liste gespeichert?

Eine ACL gibt zu einem Objekt an, welche Subjekte welche Rechte an diesem Objekt haben. [ACL ist eine Spalte der Schutzmatrix].
Jeder Eintrag der ACL enthält ein Subjekt und ein Recht (ggf. Mengen und/oder Wildcards).

Aufgabe 6: Speicherverwaltung (15 Punkte)

a) (*9 Punkte*) Sowohl Segmentierung als auch Paging basieren auf einer Adreßübersetzung von virtuellen auf physische Adressen. Beschreiben Sie **exakt** die wesentlichen **Unterschiede** zwischen Segmentierung und Paging! Gehen Sie dabei insbesondere auf den **Aufbau des virtuellen und physischen Adreßraums**, den Unterschied zwischen **Segmenten** und **Seiten**, sowie die eigentliche **Adreßübersetzung** ein!

Achtung: Segmentierung wird in der aktuellen Vorlesung nicht mehr behandelt!

- Segmentierung:
  - virt. Adreßraum besteht aus mehreren Teilen (Segmenten) unterschiedlicher Länge.
  - Segmente können unterschiedliche Rechte haben.
  - Segemente sind Unterteilung des virt. Adreßraums nach logischer Struktur (Code, Heap, ...).
  - Zweidimensionale Adressierung: (Segmentnummer, Offset).
  - Segmente werden zusammenhängend in HS abgebildet.
  - Adreßübersetzung: Segmentnummer indiziert Eintrag in Segmenttabelle. Eintrag enthält Basisadresse und Segmentlänge.
    Physische Adresse ist Summe aus Basisadresse und Offset.
    Offset wird gegen Segmentlänge geprüft.
- Paging:
  - virt. und phys. Adreßraum sind linear, starre Unterteilung in Seiten (virt.) bzw. Kacheln (phys.) gleicher Größe, für Prozesse nicht sichtbar, ohne Rücksicht auf logische Struktur
  - Teile des virt. Adreßraums in HS eingelagert
  - Adreßübersetzung: Seitennummer bestimmt Eintrag in Seitentabelle. Eintrag enthält zugehörige Kachelnummer. (Seitentabelle bildet Seiten auf Kacheln ab).
    Seitenoffset = Kacheloffset.
    Ggf. Seitenfehler. Zusammenspiel MMU/BS.

b) (*6 Punkte*) Einem Prozeß, dessen virtueller Adreßraum 5 Seiten umfaßt, wurden vom Betriebssystem **4 Kacheln** im Hautpspeicher zugeteilt. Zu einem bestimmten Zeitpunkt hat das Betriebssystem folgende Informationen (Seitentabelle + Hilfsinformation) über die Seiten des Prozesses:

	Seitentabelle				Hilfsinformation	
Seite	Present-Bit	Kachel	R-Bit	M-Bit	Lade-zeit	Zeit des letzten Zugriffs
0	1	2	0	1	30	35
1	1	3	1	0	5	45
2	1	1	0	0	10	20
3	0	-	-	-	-	-
4	1	0	0	0	25	40

Der Prozeß führt nun

- zum **Zeitpunkt 50** einen **Schreib**zugriff auf **Seite 2** und danach
- zum **Zeitpunkt 55** einen **Lese**zugriff auf **Seite 3** durch.

Geben Sie unten an, wie Seitentabelle und Hilfsinformation nach dieser Folge von Zugriffen aussehen, wenn

(i) LRU (*Least Recently Used*)

(ii) *Second Chance*

als Seitenersetzungsalgorithmus verwendet wird!

(i) LRU

	Seitentabelle				Hilfsinformation	
Seite	Present-Bit	Kachel	R-Bit	M-Bit	Lade-zeit	Letzter Zugriff
0	**0**	**-**	**-**	**-**	**-**	**-**
1	1	3	1	0	5	45
2	1	1	**1**	**1**	10	**50**
3	**1**	**2**	**1**	**0**	**55**	**55**
4	1	0	0	0	25	40

(ii) *Second Chance*

	Seitentabelle				Hilfsinformation	
Seite	Present-Bit	Kachel	R-Bit	M-Bit	Lade-zeit	Letzter Zugriff
0	1	2	0	1	30	35
1	1	3	**0**	0	**55**	45
2	1	1	0	**1**	10	**50**
3	**1**	**0**	**1**	**0**	**55**	**55**
4	**0**	**-**	**-**	**-**	**-**	**-**