Belegungs-Anforderungs-Graphen

Hochschule RheinMain

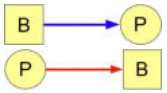
Graphische Darstellung der Beziehung von Prozessen zu Betriebsmitteln (Holt, 1972)

Es gibt zwei Knotentypen:

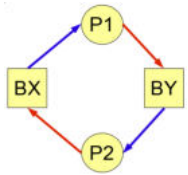
- Prozesse, repräsentiert durch Kreise:
- Betriebsmittel, repräsentiert durch Quadrate:

Pfeile:

- P belegt B
- P wartet auf B



Zyklus im Graphen → Deadlock



Notizen

Beispiel

Hochschule RheinMain

Gegeben:

- drei Prozesse A, B, C und
- drei Betriebsmittel R,S,T

Prozess A	**Prozess B**	**Prozess C**
- Anforderung R	- Anforderung S	- Anforderung T
- Anforderung S	- Anforderung T	- Anforderung R
- Freigabe R	- Freigabe S	- Freigabe T
- Freigabe S	- Freigabe T	- Freigabe R

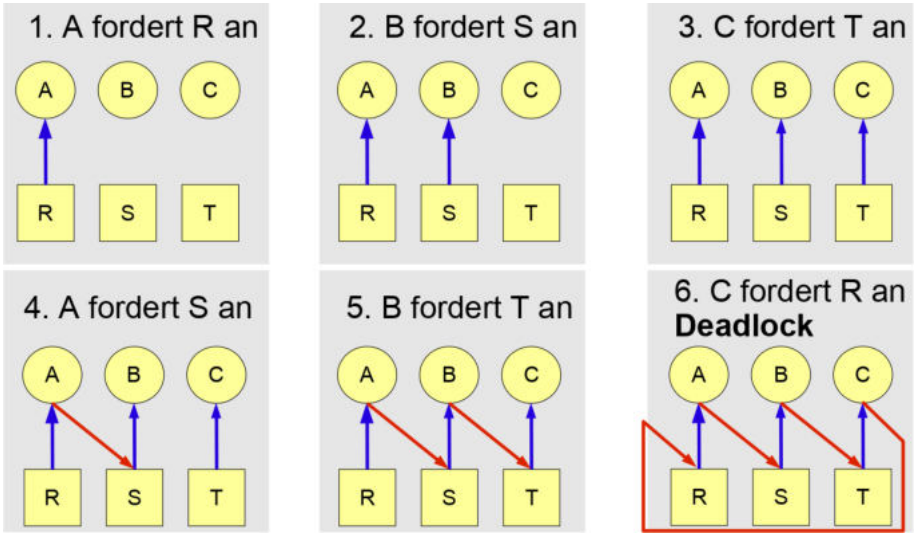
Das Betriebssystem kann jeden (nicht blockierten) Prozess **jederzeit** ausführen

Sequentielle Ausführung von A, B, C wäre unproblematisch (dann aber auch keine Nebenläufigkeit)

Wie sieht es bei nebenläufiger Ausführung aus?

Notizen

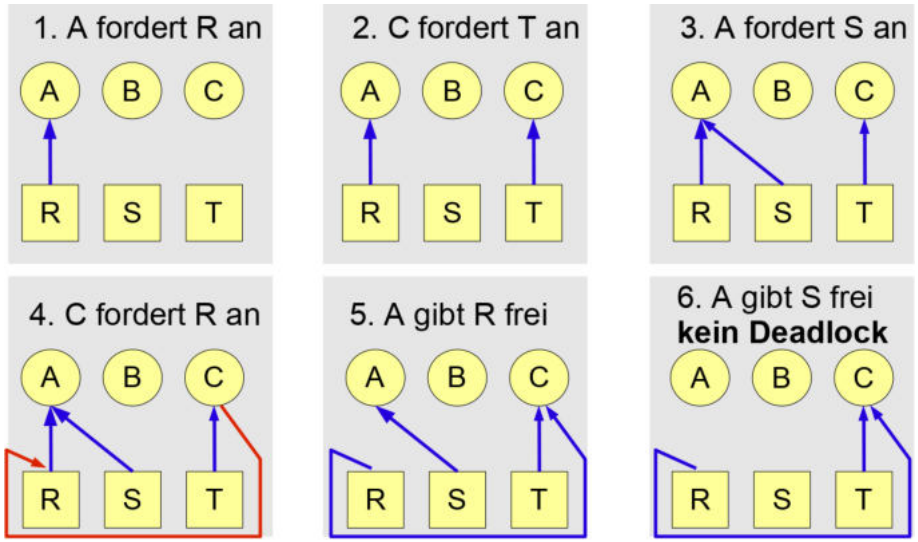
Ausführung I



Notizen

Ausführung II

(B zunächst suspendiert)



Notizen

Verfahren zur Deadlock-Behandlung

Hochschule RheinMain

Mit Betriebsmittelzuteilungsgraphen (*„Belegungs/Anforderungs-Graphen“*) lassen sich Deadlocks erkennen (→Zyklus im Graph)
Wie weiter verfahren?

- **Ignorieren** („Vogel-Strauß-Verfahren“)
- Deadlocks **erkennen** und **beheben**
- **Verhinderung** durch Planung der Betriebsmittelzuordnung (*deadlock avoidance*)
- **Vermeidung** durch Nichterfüllung (mindestens) einer der vier Voraussetzungen für Deadlocks (*deadlock prevention*)

Diese Strategien werden im folgenden untersucht.

Notizen

Ignorieren des Problems

Hochschule RheinMain

„Vogel-Strauß-Algorithmus“

Ausdruck optimistischer Lebenshaltung:

„Deadlocks kommen in der Praxis sowieso nie vor“

http://clipart.coolclips.com/480/vectors/tf05038/CoolClips_anim0613.png

...warum also dann Aufwand in ihre Vermeidung stecken?

Beispiel:

- UNIX-System mit z.B. 100 Einträge großer Prozesstabelle
- 10 Programme versuchen gleichzeitig, je 12 Kindprozesse zu erzeugen
- Deadlock nach 90 erfolgreichen fork()-Aufrufen (wenn keiner der Prozesse aufgibt)

Ähnliche Beispiele sind mit anderen begrenzt großen Systemtabellen möglich (z.B. inode-Tabelle)

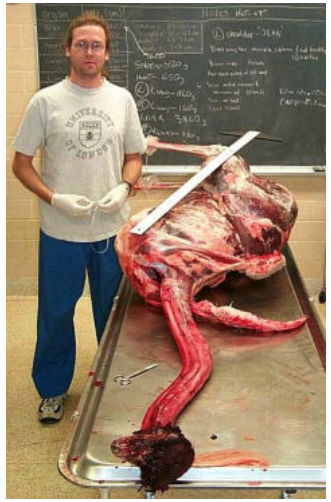
Notizen

...manchmal nicht so gut

Hochschule RheinMain

http://clipart.coolclips.com/480/vectors/tf05038/CoolClips_anim0613.png

→



http://www.instore.si/newsarticle/newsarticle/Septembra-v-Aldiju-nojevo-meso

Notizen

Deadlock-Erkennung und Behebung

Hochschule RheinMain

Engl.: ***deadlock detection and resolution / recovery***

Vorgehensweise: Das Auftreten von Deadlocks wird vom Betriebssystem nicht verhindert. Es wird versucht, Deadlocks zu erkennen und anschließend zu beheben.

Betrachtet werden im folgenden:

- Deadlock-Erkennung mit einem Betriebsmittel je Klasse (Einfacher Fall)
- Deadlock-Erkennung mit mehreren Betriebsmitteln je Klasse (Allgemeiner Fall)
- Verfahren zur Deadlock-Behebung

Notizen

Deadlocks erkennen (Einfacher Fall)

Hochschule RheinMain

Vereinfachende Annahme: **Ein Betriebsmittel** je Betriebsmitteltyp

Vorgehen:

- erzeuge Belegungs-/Anforderungs-Graph
- suche nach Zyklen
- falls ein Zyklus gefunden wurde: Deadlock beheben (s.u.)

Wann wird die Untersuchung durchgeführt?

- bei jeder Betriebsmittelanforderung?
- in **regelmäßigen** Zeitabständen?
- wenn **„Verdacht“** auf Deadlock besteht
  (z.B. Abfall der CPU-Auslastung unter eine Grenze)

Notizen

Beispiele: Sicher?

Hochschule RheinMain

4 Prozesse, ein Betriebsmitteltyp (10 Stück vorhanden)

verfügbar: 10

Proz.	hat	max.
A	0	6
B	0	5
C	0	4
D	0	7

sicher!

z.B. sequenzielle Ausführung von A, B, C, D in beliebiger Reihenfolge ist möglich.

verfügbar: 2

Proz.	hat	max.
A	1	6
B	1	5
C	2	4
D	4	7

sicher!

C ist ausführbar,(→ dann 4 verfügbar) dann D, B, A möglich.

verfügbar: 1

Proz.	hat	max.
A	1	6
B	2	5
C	2	4
D	4	7

unsicher!

Differenz $max - hat$ immer $> verfgbar$. Deadlock, sobald irgend ein Prozess auf sein Maximum zugeht

Notizen

Beispiel

Hochschule RheinMain

A belegt R und fordert S an.

B fordert T an.

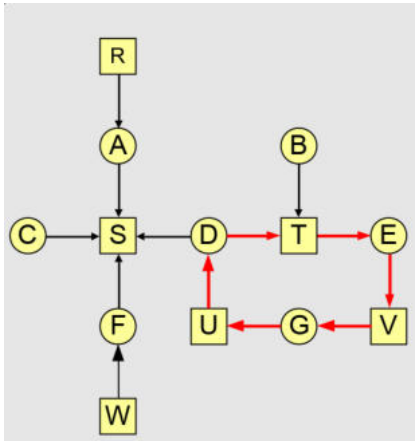
C fordert S an.

D belegt U und fordert S und T an.

E belegt T und fordert V an.

F belegt W und fordert S an.

G belegt V und fordert U an.



Notizen

Deadlocks erkennen

Hochschule RheinMain

Erweiterung: Mehrere (E_i-viele) Betriebsmittel je Betriebsmitteltyp i (z.B. mehrere Drucker)

Prozesse $P_1, \ldots, P_n$

$$E = (E_1, E_2, \ldots, E_m)$$

Betriebsmittelvektor E: Gesamtzahl der BM je Typ i

$$A = (A_1, A_2, \ldots, A_m)$$

Verfügbarkeitsvektor A: Gesamtzahl der BM je Typ i

Belegungsmatrix C: Zeile j gibt BM-Belegung durch Prozess j an („Prozess j belegt C_{jk} Einheiten von BM k")

$$C = \begin{pmatrix} C_{11} & C_{12} \ldots & C_{1m} \\ C_{21} & C_{22} \ldots & C_{2m} \\ \ldots & \ldots\ldots & \ldots \\ C_{n1} & C_{n2} \ldots & C_{nm} \end{pmatrix}$$

Anforderungsmatrix R: Zeile j gibt BM-Belegung durch Prozess j an („Prozess j belegt R_{jk} Einheiten von BM k")

$$R = \begin{pmatrix} R_{11} & R_{12} \ldots & R_{1m} \\ R_{21} & R_{22} \ldots & R_{2m} \\ \ldots & \ldots\ldots & \ldots \\ R_{n1} & R_{n2} \ldots & R_{nm} \end{pmatrix}$$

Notizen

Erkennungsalgorithmus

Hochschule RheinMain

Zu Beginn sind alle Prozesse aus P unmarkiert
(Markierung heißt, dass der Prozess in keinem DL steckt)

Suche einen Prozess, der ungehindert durchlaufen kann, also einen unmarkierten Prozess P_i, dessen Zeile in der Anforderungsmatrix-Zeile R_i (komponentenweise) kleiner oder gleich dem Verfügbarkeitsvektor A ist

Kein passendes P_i gefunden? Dann → **Ende**

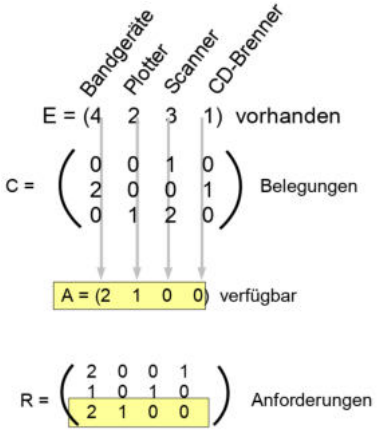
Gefunden? Dann kann P_i durchlaufen und gibt danach seine belegten Betriebsmittel zurück: $A = A + C_i$, wird markiert und es geht beim nächsten unmarkierten Prozess weiter

Beim Ende des Verfahrens sind **alle unmarkierten** Prozesse an einem **Deadlock beteiligt**.

Notizen

Beispiel

Hochschule RheinMain



Ausführbar ist zunächst nur P_3
Freigabe $C_3 = (0120)$
⇒ $A = (2100) + (0120)$
⇒ $A = (2220)$
Nun ausführbar: P_2
(benötigt $R_2 = (1010)$)
Freigabe $C_2 = (2001)$
⇒ $A = (4221)$
Schließlich auch P_1 ausführbar
⇒ $A = (4231)$
⇒ Alle Prozesse markiert,
kein Deadlock aufgetreten.

Notizen

Beheben von Deadlocks

Hochschule RheinMain

Wie kann man auf erkannte Deadlocks reagieren?

Prozessunterbrechung

- Betriebsmittel zeitweise entziehen, anderem Prozess bereitstellen und dann zurückgeben
- Kann je nach Betriebsmittel schwer oder nicht möglich sein

Teilweise Wiederholung (*rollback*)

- System sichert regelmäßig Prozesszustände (*checkpoints*)
- Dadurch ist Abbruch und späteres Wiederaufsetzen möglich
- Arbeit seit letztem Checkpoint geht beim Rücksetzen verloren und wird beim Neuaufsetzen wiederholt (ungünstig z.B. bei seit Checkpoint ausgedruckten Seiten)
- Beispiel: Transaktionsabbruch bei Datenbanken

Prozessabbruch

- Härteste, aber auch einfachste Maßnahme
- Nach Möglichkeit Prozesse auswählen, die relativ problemlos neu gestartet werden können (z.B. Compilierung)

Notizen

Verhindern von Deadlocks

Hochschule RheinMain

Bisher: Erkennung von Deadlocks, gegebenenfalls „drastische“ Maßnahmen zur Auflösung

Annahme bisher: Prozesse fordern alle Betriebsmittel „auf ein Mal“ an (vgl. 7.4.2).

In den meisten praktischen Fällen werden BM jedoch nacheinander angefordert

Das Betriebssystem muss dann dynamisch über die Zuteilung entscheiden

Notizen

Verhindern von Deadlocks

Hochschule RheinMain

Kann man **Deadlocks** durch „geschicktes“ Vorgehen bei der Betriebsmittelzuteilung **von vornherein verhindern**?

Welche Informationen müssen dazu vorab zur Verfügung stehen?

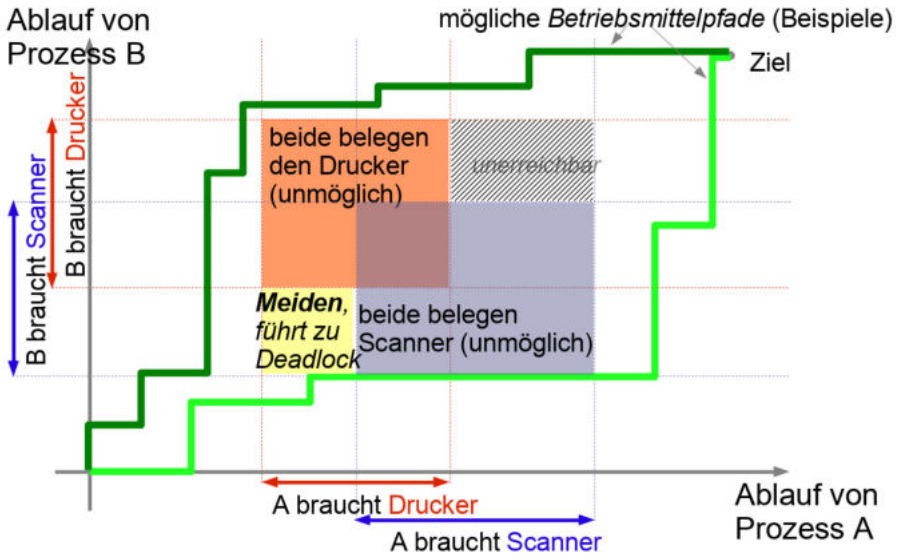
Im folgenden betrachtet

- Betriebsmittelpfade (Grafische Veranschaulichung)
- Sichere und unsichere Zustände
- Der vereinfachte Bankiersalgorithmus für eine BM-Klasse
- Der Bankiersalgorithmus für mehrere BM-Klassen

Notizen

Betriebsmittelpfade

Hochschule RheinMain



Notizen

(Un-)Sichere Zustände

Hochschule RheinMain

Definition

Ein Systemzustand ist **sicher**, wenn er

keinen Deadlock repräsentiert und
es eine geeignete Prozessausführungsreihenfolge gibt, bei der alle Anforderungen erfüllt werden
(die also **auch dann** nicht in einen Deadlock führt, wenn alle Prozesse gleich ihre max. Ressourcenanzahl anfordern)

Sonst heißt der Zustand **unsicher**.

Bei einem sicherem Zustand kann das System **garantieren**, dass alle Prozesse bis zum Ende durchlaufen können.
Bei unsicherem Zustand ist das nicht garantierbar (aber auch nicht ausgeschlossen!).

Beispiel: Ein Prozess gibt ein BM zu einem „glücklichen Zeitpunkt“ kurzzeitig frei, wodurch eine Deadlock-Situation „zufällig“ vermieden wird. (→ „Glück“ nicht vorhersehbar)

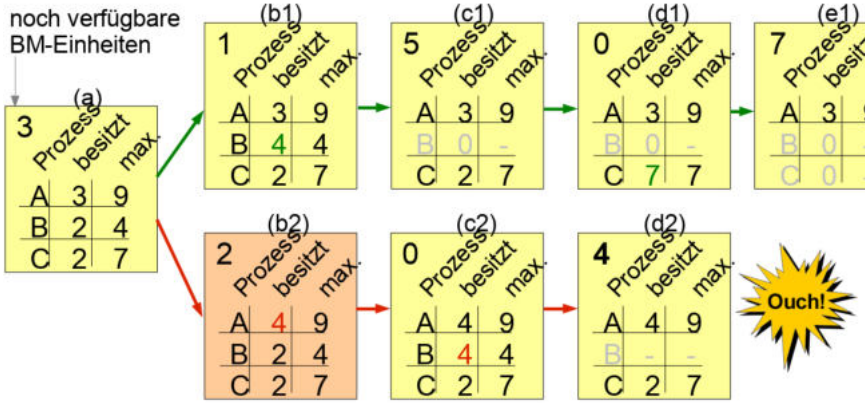
„Unsicher“ bedeutet also nicht „Deadlock unvermeidlich“.

Notizen

Beispiel

Hochschule RheinMain

3 Prozesse A,B,C; jeweils mit BM-Besitz und max. Bedarf
ein Betriebsmitteltyp, 10x vorhanden



Zustand (a) ist sicher (es gibt eine DL-freie Lösung)
(b2) ist **nicht** sicher (A und C brauchen je 5, frei sind nur 4)

Notizen

Bankier-Algorithmus (1 BM-Klasse)

Hochschule RheinMain

- Dijkstra (wer sonst? 1965):
- Ein **Bankier kennt** die **Kreditrahmen** seiner Kunden.
- Er geht davon aus, dass **nicht alle** Kunden **gleichzeitig** ihre Rahmen **voll** ausschöpfen werden.
- Daher hält er **weniger Bargeld** bereit als die **Summe** der Kreditrahmen.
- Gegebenenfalls **verzögert** er die **Zuteilung** eines Kredits, bis ein anderer Kunde zurückgezahlt hat.
- **Zuteilung** erfolgt **nur**, wenn sie **"sicher"** ist (also letztlich alle Kunden bis zu ihrem Kreditrahmen bedient werden können).

Bankier = Betriebssystem, Bargeld = Betriebsmitteltyp,
Kunden = Prozesse, Kredit = BM-Anforderung

Notizen

Bankier-Algorithmus (2)

Hochschule RheinMain

Prüfe bei jeder Anfrage, ob die Bewilligung in einen sicheren Zustand führt:

Prüfe dazu, ob ausreichend Betriebsmittel bereitstehen, um **mindestens einen** Prozess **vollständig** zufrieden zu stellen.

Davon ausgehend, dass dieser Prozess nach Durchlauf seine Betriebsmittel freigibt: führe **Test** mit dem Prozess aus, der dann am nächsten am Kreditrahmen ist

usw., **bis alle** Prozesse positiv getestet sind;

Falls **ja**, kann die aktuelle Anfrage **bewilligt** werden.

Sonst: Anforderung **verschieben** (warten)

Notizen

Verallgemeinerter Bankier-Algorithmus

Hochschule RheinMain

Mehrere Betriebsmittelklassen

Datenstrukturen wie bei „Deadlockerkennung“ (7.4.2)

Matrizen mit belegten / angeforderten Betriebsmitteln

Vektoren mit BM-Bestand, verfügbaren BM und belegten BM je Betriebsmitteltyp

- E Betriebsmittelvektor
- A Verfügbarkeitsvektor
- C Belegungsmatrix
- R Anforderungsmatrix

Notizen

Beispiel

Hochschule RheinMain

$$E = \begin{pmatrix} 6 & 3 & 4 & 2 \end{pmatrix} \text{ vorhanden}$$

$$C = \begin{pmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ zugewiesen}$$

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 \end{pmatrix} \text{ verfügbar}$$

$$P = \begin{pmatrix} 5 & 3 & 2 & 2 \end{pmatrix} \text{ belegt}$$

$$R = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 \end{pmatrix} \text{ angefordert}$$

Sicher? Ja, Ausführungsfolge P_4, P_1, P_5, ...ist möglich:

$$P_4 \rightarrow A = \begin{pmatrix} 2 & 1 & 2 & 1 \end{pmatrix}$$

$$P_1 \rightarrow A = \begin{pmatrix} 5 & 1 & 3 & 2 \end{pmatrix}$$

$$P_5 \rightarrow A = \begin{pmatrix} 5 & 1 & 3 & 2 \end{pmatrix}$$

$$P_2 \rightarrow A = \begin{pmatrix} 5 & 2 & 3 & 2 \end{pmatrix}$$

$$P_3 \rightarrow A = \begin{pmatrix} 6 & 3 & 4 & 2 \end{pmatrix}$$

Notizen

Beispiel

Hochschule RheinMain

$$E = \begin{pmatrix} 6 & 3 & 4 & 2 \end{pmatrix} \text{ vorhanden}$$

$$C = \begin{pmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ zugewiesen}$$

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix} \text{ verfügbar}$$

$$P = \begin{pmatrix} 5 & 3 & 3 & 2 \end{pmatrix} \text{ belegt}$$

$$R = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 \end{pmatrix} \text{ angefordert}$$

P2 fordere ein BM 3 an (rot)

Sicher? Ja, Ausführungsfolge P_4, P_1, P_5, P_2,P_3 möglich:

$P_4 \rightarrow A = \begin{pmatrix} 2 & 1 & 1 & 1 \end{pmatrix}$

$P_1 \rightarrow A = \begin{pmatrix} 5 & 1 & 2 & 2 \end{pmatrix}$

$P_5 \rightarrow A = \begin{pmatrix} 5 & 1 & 2 & 2 \end{pmatrix}$

$P_2 \rightarrow A = \begin{pmatrix} 5 & 2 & 3 & 2 \end{pmatrix}$

$P_3 \rightarrow A = \begin{pmatrix} 6 & 3 & 4 & 2 \end{pmatrix}$

also erhält P_2 ein BM3

Notizen

Beispiel

Hochschule RheinMain

$$E = \begin{pmatrix} 6 & 3 & 4 & 2 \end{pmatrix} \text{ vorhanden}$$

$$C = \begin{pmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \text{ zugewiesen}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \text{ verfügbar}$$

$$P = \begin{pmatrix} 5 & 3 & 4 & 2 \end{pmatrix} \text{ belegt}$$

$$R = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 \end{pmatrix} \text{ angefordert}$$

Nun fordere auch P_5 ein BM 3 an

→ dann würde $A = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$

Sicher? *Nein!*

→ daher Anfrage von P_5 blockieren

Notizen

Ist der Bankier-Algorithmus praktikabel?

Hochschule RheinMain

In der Praxis gibt es mehrere Probleme beim Einsatz:

- Prozesse können „maximale Ressourcenanforderung“ selten im Voraus angeben
- Anzahl der Prozesse ändert sich ständig
- Ressourcen können verschwinden (z.B. durch Ausfall)

Notizen

Deadlock-Vermeidung

Hochschule RheinMain

Deadlock-Verhinderung ist wenig praktikabel ☹

Ansatz: **Vermeidung** mindestens einer der vier Deadlock-**Voraussetzungen** (vgl 7.2)

- Wechselseitiger Ausschluss
- Belegungs-/Anforderungsbedingung („Hold-and-Wait“, d.h. zu reservierten BM weitere anforderbar)
- Ununterbrechbarkeit (kein erzwungener BM-Entzug)
- zyklisches Warten

Notizen

1. Wechselseitiger Ausschluß?

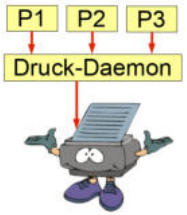
Hochschule RheinMain

Falls es keine exklusive Zuteilung eines Betriebsmittels an einen Prozess gibt, gibt es auch keine Deadlocks.

Beispiel: Zugriff auf Drucker

Einführung eines **Spool-Systems**, das

- Druckaufträge von Prozessen (schnell) entgegennimmt
- ggf. zwischenspeichert
- und der Reihe nach auf dem Drucker ausgibt



Entkopplung zwischen (konkurrierenden) Prozessen und dem (langsamen) Betriebsmittel

Vermeidung einer exklusiven Zuteilung des Betriebsmittels „Drucker“

Notizen

2. Belegungs-/Anforderungsbedingung?

Hochschule RheinMain

Vermeiden, dass neue Betriebsmittel-Anforderungen zu bereits bestehenden hinzukommen.

„Preclaiming“: Alle Anforderungen zu Beginn der Ausführung stellen („alles oder nichts“)

Vorteil: Wenn Anforderungen erfüllt werden, kann der Prozess sicher bis zum Ende durchlaufen (er hat ja dann alles, was er braucht)

Nachteil:

- Anforderungen müssen **zu Beginn bekannt** sein
- Betriebsmittel werden unter Umständen **lange blockiert**
- und können zwischenzeitlich nicht (sinnvoll) anders genutzt werden.

Beispiel: Batch-Jobs bei Großrechnern.

Notizen

3. Ununterbrechbarkeit?

Hochschule RheinMain

Hängt vom Betriebsmittel ab, aber

„gewaltsamer“ Entzug ist in der Regel nicht akzeptabel

- Drucker?
- CD-Brenner?

Notizen

4. Zyklische Wartebedingung?

Hochschule RheinMain

Wenn es kein zyklisches Auf-einander-warten gibt, entstehen auch keine Deadlocks

Idee:

- Betriebsmitteltypen **linear ordnen** und
- nur in aufsteigender Ordnung Anforderungen annehmen (wenn mehrere Exemplare eines Typs gebraucht werden: alle Exemplare auf einmal anfordern)
- z.B. „Drucker *vor* Scanner *vor* CD-Brenner *vor* …“

Dadurch entsteht **automatisch** ein **zyklenfreier** Belegungs-Anforderungs-Graph,

wodurch Deadlocks ausgeschlossen sind.

Tatsächlich praktikables Verfahren.

Notizen

Deadlock-Vermeidung im Überblick

Hochschule RheinMain

Deadlock-Vermeidung durch Verhinderung (mindestens) einer der 4 Vorbedingungen eines Deadlocks ist möglich:

Wechselseitiger Ausschluß	→ Spooling
Belegungs-/Anforderungsbed.	→ Preclaiming
Ununterbrechbarkeit	(BM-Entzug...besser nicht)
Zyklisches Warten	→ Betriebsmittel ordnen

Notizen

Verwandte Fragestellungen

Hochschule RheinMain

Deadlocks bei der Benutzung von Semaphoren (vgl. Kap. 3)

Zwei-Phasen-Locking in Datenbanken

Verhungern (Starvation), kein Deadlock, aber auch kein Fortschritt für einen Prozess (vgl. Philosophen-Problem)

Notizen