2405 Betriebssysteme
VI. Verklemmung von Prozessen

Thomas Staub, Markus Anwander
Universität Bern

Inhalt

1.1 Verklemmungen

- Eine Menge von Prozessen befindet sich in einer Verklemmung (*Deadlock*), wenn jeder Prozess ein Betriebsmittel belegt und darauf wartet ein anderes, von einem anderen Prozess belegtes Betriebsmittel (Ressource) zu belegen.
- Beispiel:
  - System mit 2 Bandlaufwerken
  - P_0 und P_1 belegen jeweils ein Bandlaufwerk und wollen das jeweils andere auch belegen.

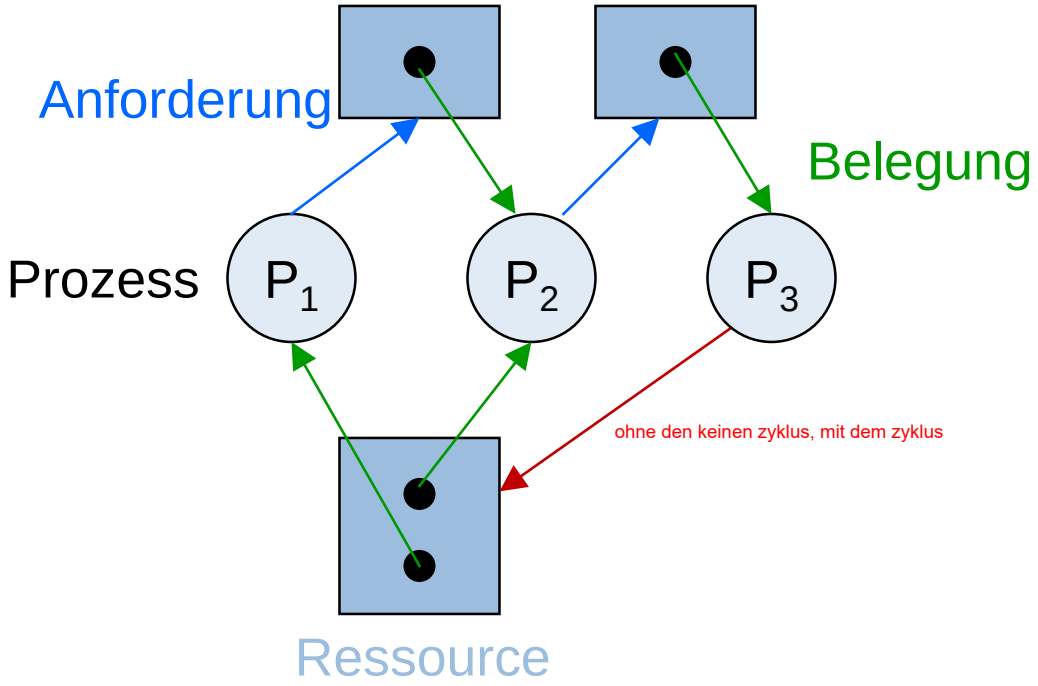
P_0	P_1
`wait(A);`	`wait(B);`
`wait(B);`	`wait(A);`

1.2 Charakterisierung von Verklemmungen

Verklemmungen treten auf, wenn ***alle*** der folgenden Bedingungen erfüllt sind:

- Wechselseitiger Ausschluss
  - Ressource kann nur von einem Prozess benutzt werden.
- Halten und Warten
  - Prozess, welcher eine Ressource hält, wartet auf eine andere.
- Keine Verdrängung
  - Ressource kann durch den haltenden Prozess nur freiwillig freigegeben werden.
- Zirkulierendes Warten
  - P_i (i=0,...,n-1) wartet auf durch $P_{i+1 \bmod n}$ belegte Ressource.hinreichend

1.3 Ressourcenbelegungsgraphen



Graph ohne Zyklen

keine Verklemmung

Graph mit Zyklen

mögliche Verklemmung !

2. Behandeln von Verklemmungen

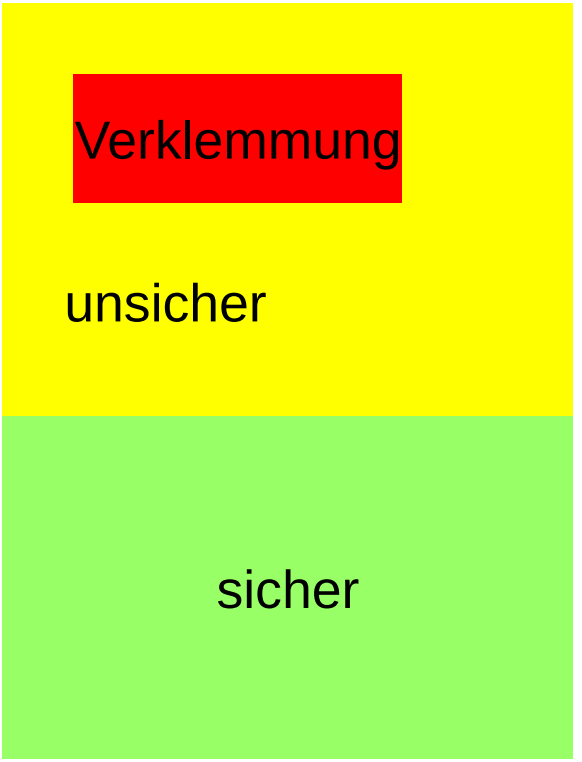
- Verhindern/Vermeiden von Verklemmungen
  - Verhindern (Deadlock Prevention):
    Methoden um zu verhindern, dass eine der vier Verklemmungsbedingungen zutrifft
  - Vermeiden (Deadlock Avoidance):
    Für jede einzelne Ressourcenanforderung wird entschieden,
    ob dadurch eine Verklemmung auftreten kann.
- Aufheben von Verklemmungen
  - Es wird erlaubt, dass eine Verklemmung auftreten kann und falls sie erkannt wird,
    werden entsprechende Massnahmen ergriffen.

2.1 Verhindern von Verklemmungen

- Wechselseitiger Ausschluss
  - nicht für teilbare Ressourcen (Betriebsmittel) notwendig, z.B. read-only Dateien
- Halten und Warten
  - Anforderung von Ressourcen nur wenn der Prozess aktuell keine Ressourcen belegt
    - Belegen aller Ressourcen vor Ausführung des Prozesses
    - Abgabe aller Betriebsmittel bevor neue belegt werden
- Keine Verdrängung
  - Entzug von bereits zugewiesenen Ressourcen
- Zirkulierendes Warten
  - Totalordnung von Ressourcentypen
    - z.B. Bandlaufwerk (0), Festplatte (1), ..., Drucker (15) fixe reihenfolge von anfragen
  - aufsteigende Anforderung
  - atomare Anforderung mehrerer Instanzen eines Ressourcentyps

2.2 Vermeiden von Verklemmungen

- Prozesse Pi beschreiben a priori die maximale Menge benötigter Ressourcen.
- Request wird erfüllt, wenn das System in einem sicheren Zustand (safe state) bleibt.
- Ein unsicherer Zustand kann zu einer Verklemmung führen.
- Ein System befindet sich in einem sicheren Zustand, wenn es eine sichere Sequenz gibt.
- Sequenz $<P_1, P_2,..., P_n>$ ist sicher, wenn für alle i jeder Request von P_i durch verfügbare Ressourcen und belegte Ressourcen von P_j (j<i) erfüllt werden kann.



2.2.1 Safe State Algorithmus

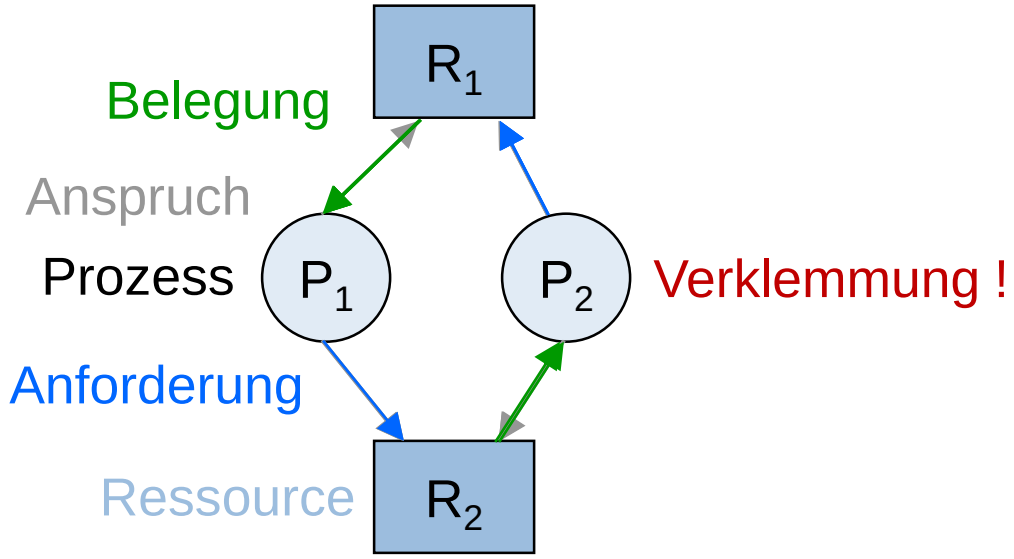
- 12 Bandlaufwerke
- t_0: 3 freie Laufwerke
- <P_1,P_0 ,P_2> ist sicher
  - P_1 kann 2 weitere Ressourcen belegen und dann freigeben. → 5 freie Ressourcen.
  - P_0 kann 5 weitere Ressourcen belegen und dann freigeben. → 10 freie Ressourcen.
  - P_2 kann dann 7 weitere Ressourcen belegen.

	maximale Anforderungen	aktuelle Belegungen
P_0	10	5
P_1	4	2
P_2	9	~~2~~ 3

t_1: P_2 fordert 1 Ressource an.
→ Anforderung von P_1 könnte zwar erfüllt werden, danach würden aber nur 4 freie Ressourcen zur Verfügung stehen

2.2.2 Ressourcenbelegungsgraph-Algorithmus

- für Systeme mit ***einer*** Instanz für jede Ressource !
- Anspruch-Pfeil zur Anzeige (a priori),
  dass Prozess möglicherweise eine Ressource belegen wird.
- Anspruch-Pfeil wird bei Anforderung zu Anforderung-Pfeil.
- Belegungs-Pfeil wird bei Freigabe zu Anspruch-Pfeil.



2.2.3 Bankers-Algorithmus

- geeignet für Ressourcen mit mehrfachen Instanzen
- Jeder Prozess beschreibt a priori seinen maximalen Anspruch.
- Annahme: Nach Ressourcenbelegung gibt ein Prozess nach endlicher Zeit alle Ressourcen frei.
- Datenstrukturen (n: Anzahl Prozesse, m: Anzahl Ressourcentypen)
  - Available[j]==k (j<m): k Ressourcen des Typs j sind verfügbar.
  - Max[i,j]==k (i<n, j<m) : Prozess P_i belegt höchstens k Instanzen der Ressource R_j.
  - Allocation[i,j]==k (i<n, j<m):
    Prozess P_i hat aktuell k Instanzen der Ressource R_j belegt.
  - Need[i,j]==k (i<n, j<m) : Prozess P_i benötigt höchstens weitere k Instanzen der Ressource R_j zur Beendigung seiner Aufgabe. Need = Max – Allocation

2.2.3.1 Safety-Algorithmus

1. Work=Available;
   Finish[i]=false für alle i=0,1,...,n-1
2. Finde i, so dass Finish[i] == false und $Need_i \leq Work$,
   d.h. Need[i,j] ≤ Work[j] für alle j=0,..,m-1
   Falls kein solches i existiert gehe zu 4.
3. Work=Work+$Allocation_i$;
   Finish[i]=true;
   gehe zu 2.
4. Das System ist in einem sicheren Zustand,
   falls Finish[i]==true für alle i=0,1,...,n-1

2.2.3.2 Ressourcenbelegungs-Algorithmus

$Request_i$: Anforderungsvektor für Prozess P_i,
Request[i,j]==k: Prozess P_i fordert k Instanzen von Ressourcentyp R_j.

1. Falls $Request_i \leq Need_i$ gehe zu 2.,
   sonst: Fehler, da Maximum überschritten
2. Falls $Request_i \leq Available_i$ gehe zu 3.,
   sonst: warte, da nicht genug Ressourcen verfügbar
3. $Available$ -= $Request_i$;
   $Allocation_i$ += $Request_i$;
   $Need_i$ -= $Request_i$
   Belege Ressourcen, falls System in sicherem Zustand,
   sonst: P_i muss warten, stelle alten Zustand wieder her.

2.2.3.3.1 Beispiel: Bankers-Algorithmus

	Allocation	Max	Need	Available	Work
	A B C	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	3 3 2	10 4 7
P_1	2 0 0	3 2 2	1 2 2		
P_2	3 0 2	9 0 2	6 0 0		
P_3	2 1 1	2 2 2	0 1 1		
P_4	0 0 2	4 3 3	4 3 1		

(P_1,P_3,P_4,P_2,P_0) ist eine sichere Sequenz, d.h. System ist sicher

3 3 2
5 3 2
7 4 3
7 4 5
10 4 7

dann haben wir alle sequenzen durchgearbeitet

2.2.3.3.2 Beispiel: Bankers-Algorithmus

P_1 fordert Ressourcen (1 0 2) an.
(P_1,P_3,P_4,P_2,P_0) ist eine sichere Sequenz, d.h. System ist sicher.

	Allocation	Max	Need	Available	Work
	A B C	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	2 3 0	10 4 7
P_1	3 0 2	3 2 2	0 2 0		
P_2	3 0 2	9 0 2	6 0 0		
P_3	2 1 1	2 2 2	0 1 1		
P_4	0 0 2	4 3 3	4 3 1		

P_4 fordert Ressourcen (3 3 0) an, Ressourcen sind nicht verfügbar.

P_0 fordert Ressourcen (0 2 0) an, Ressourcen sind verfügbar.

2.2.3.3.3 Beispiel: Bankers-Algorithmus

P_0 fordert Ressourcen (0 2 0) an, es gibt aber keine sichere Sequenz.

	Allocation	Max	Need	Available	Work
	A B C	A B C	A B C	A B C	A B C
P_0	0 3 0	7 5 3	7 2 3	2 1 0	2 1 0
P_1	3 0 2	3 2 2	0 2 0		
P_2	3 0 2	9 0 2	6 0 0		
P_3	2 1 1	2 2 2	0 1 1		
P_4	0 0 2	4 3 3	4 3 1		

2.3 Erkennen von Verklemmungen

- Periodisches Berechnen des Ressourcenbelegungsgraphen bei einer Instanz pro Ressourcentyp
- Verklemmungserkennungsalgorithmus für mehrere Instanzen
  - Datentypen wie bei Bankers-Algorithmus
  - Request[i,j] == k: Prozess P_i fordert weitere k Instanzen des Ressourcentyps R_j an.

2.3.1 Verklemmungserkennungsalgorithmus

1. Work=Available;
   Falls $Allocation_i \neq 0$: Finish[i] = false sonst Finish[i] = true, für alle i=0,1,...,n
2. Finde i, so dass Finish[i] == false *und* $Request_i \leq Work$
   Falls kein solches i existiert, gehe zu 4.
3. $Work = Work + Allocation_i$;
   Finish[i] = true;
   gehe zu 2.
4. Das System ist in einem Verklemmungszustand,
   falls Finish[i] == false für ein i=0,1,...,n.

> Unterschied zu Safety-Algorithmus: Vergleich von Request (nicht Need) mit Work
> $\rightarrow$ optimistischer Ansatz (Annahme: keine weiteren Ressourcenanforderungen)

> Ansonsten werden Verklemmungen später entdeckt.

2.3.2.1 Beispiel: Verklemmungserkennung

	Allocation	Request	Available	Work
	A B C	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0	7 2 6
P_1	2 0 0	2 0 2		
P_2	3 0 3	0 0 0		
P_3	2 1 1	1 0 0		
P_4	0 0 2	0 0 2		

Sequenz (P_0,P_2,P_3,P_1,P_4) resultiert in Finish(i) == true für alle i.

2.3.2.2 Beispiel: Verklemmungserkennung

	Allocation	Request	Available	Work
	A B C	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0	0 1 0
P_1	2 0 0	2 0 2		
P_2	3 0 3	0 0 1		
P_3	2 1 1	1 0 0		
P_4	0 0 2	0 0 2		

Verklemmung !

2.4 Aufheben von Verklemmungen

- Optionen zum Beenden von Prozessen:
  - Beende alle verklemmten Prozesse.
  - Beende einen Prozess nach dem anderen bis Verklemmung aufgehoben ist.
- Entzug von Ressourcen
  - Auswahl eines „Opfers“
    - Parameter: belegte Ressourcen, bereits benötigte Ausführungszeit
  - Rollback der betroffenen Prozesse
    - Rückkehr zu einem sicheren Zustand und erneutes Starten der Prozesse
    - Unterstützung von Rollback durch periodisches Speichern der Prozesszustände (inkl. Ressourcenbelegung) → Checkpoint
  - Vermeiden von Aushungern
    - maximale Anzahl von Rollbacks;
      Berücksichtigung der Anzahl von Rollbacks als Parameter bei der Auswahl eines Opfers