Betriebssysteme

Kapitel 6 Deadlocks

Deadlocks / Verklemmungen

- Einführung
  - -Historischer Überblick
  - -Betriebssystemkonzepte
- Prozesse und Threads
  - -Einführung in das Konzept der Prozesse
  - -Prozesskommunikation
  - -Scheduling von Prozessen
  - -Threads
- Speicherverwaltung
  - -Einfache Speicherverwaltung
  - -Virtueller Speicher
  - -Segmentierter Speicher
- Dateien und Dateisysteme
  - -Dateien
  - -Verzeichnisse
  - -Implementierung von Dateisystemen
- Grundlegende Eigenschaften der I/O-Hardware
  - -Festplatten
  - -Terminals
  - -Die I/O-Software
- Deadlocks/Verklemmungen
- Virtualisierung und die Cloud
- Multiprozessor-Systeme
- IT-Sicherheit
- Fallstudien

Deadlocks / Verklemmungen

Deadlocks / Verklemmungen

Wiederholung

- Betriebssystem verwaltet Ressourcen
  - Ressourcen werden Prozessen zugeteilt
    - Prozesse sind wesentliche Abstraktion im Betriebssystem
    - Das Betriebssystem teilt die Ressource CPU mit Hilfe des Scheduling zu
  - eine Ressource kann (oft) nur jeweils von einem Prozess genutzt werden
    - kritische Abschnitte, wechselseitiger Ausschluss, …
    - Verschiedene Synchronisations-Primitive (busy wait, Semaphore, Monitor, …)
  - Trotzdem können Verklemmungen auftreten!
    - Beispiel:
      - Prozess A hat Scanner belegt und will CD-Brenner
      - Prozess B hat CD-Brenner belegt und will Scanner
        → A wartet auf B, B wartet auf A,....

Deadlocks / Verklemmungen

- Unterbrechbare Ressource (preemptable)
  – kann einem Prozess ohne Schaden entzogen werden
    - z.B. CPU (Sichern und Wiederherstellen beim Context-Switch)
    - z.B. Hauptspeicher (Auslagern auf Platte)

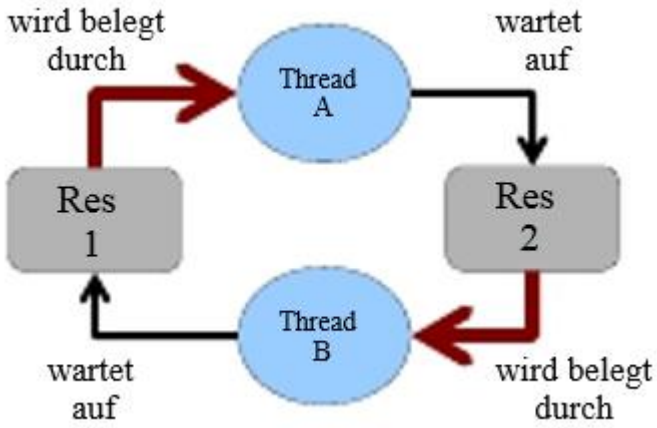
    → Deadlock können verhindert werden

- Ununterbrechbare Ressource (nonpreemptable)
  – Kann einem Prozess nicht entzogen werden, ohne dass dessen Ausführung fehlschlägt
    - z.B. CD-Brenner

→ Deadlocks haben immer mit nicht-unterbrechbaren Ressourcen zu tun

Deadlocks / Verklemmungen

- Verhungern oder Verklemmung??
  - Verhungern/Starvation
    - Prozess/Thread wartet unendlich lang
      - z.B. bei einer sehr ungünstigen Scheduling-Strategie
  - Deadlock
    - Zyklisches Warten auf Ressourcen
      - Thread A besitzt Res1 und wartet auf Res 2
      - Thread B besitzt Res2 und wartet auf Res 1



  - Deadlock führt zum Verhungern (nicht umgekehrt)
    - Verhungern kann enden
    - Verklemmung braucht externe Eingriffe

Deadlocks / Verklemmungen

- Definition
  - Deadlock
    - *Eine Menge von Prozessen befindet sich in einem Deadlock-Zustand (Verklemmungs-Zustand), wenn jeder Prozess aus der Menge auf ein Ereignis wartet, das nur ein anderer Prozess aus dieser Menge auslösen kann*
      - Hier: Ereignis = Freigabe einer Ressource

        → Alle Prozesse warten
        → Die Ereignisse werden niemals ausgelöst
        → Keiner der Prozess wird jemals wieder aufwachen

  - Deadlock sind nicht-deterministisch
    - nur wenn Scheduler und falsches Design zusammenkommen!
    - „Falsches Timing“ notwendig!

Deadlocks / Verklemmungen

- Bedingungen für einen (Ressourcen-) Deadlock
  - **Wechselseitiger Ausschluss**

    Jede Ressource kann zu einem Zeitpunkt von höchstens einem Prozess genutzt werden

  - **Hold-and-Wait-Bedingung (Besitzen und Warten)**
    Ein Prozess, der bereits Ressourcen besitzt, kann noch weitere Ressourcen anfordern

  - **Ununterbrechbarkeit (kein Ressourcenentzug)**
    Einem Prozess, der im Besitz einer Ressource ist, kann diese nicht gewaltsam entzogen werden

  - **Zyklisches Warten**
    Es gibt eine zyklische Kette von Prozessen, bei der jeder Prozess auf eine Ressource wartet, die vom nächsten Prozess in der Kette belegt ist.

Deadlocks / Verklemmungen

- Bedingungen für einen Deadlock

  – Bedingungen 1-3 sind notwendige Bedingungen
    - aber nicht hinreichend

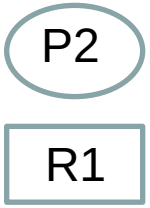
  – Bedingung 4 ist eine mögliche Konsequenz aus 1-3
    - Die Unauflösbarkeit des zyklischen Warten ist eine Folge aus 1-3

  – Alle vier Bedingungen zusammen sind **notwendig** und **hinreichend** für eine Verklemmung

  → **Wenn eine der Bedingungen unerfüllbar ist, können keine Deadlocks auftreten**

Deadlocks / Verklemmungen

- Modellierung von Deadlocks mithilfe von gerichteten Graphen
  - Ressourcen-Belegungs-Graph
  - Zwei Arten von Knoten
    - Prozesse/Thread = Kreis
    - Ressourcen = Quadrate

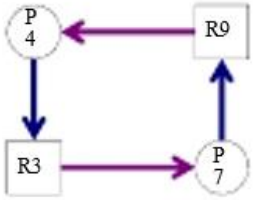


  - Zwei Arten von Kanten
    - Anforderungskante
      - Ressource wird vom Prozess angefordert (z.B. R1). Prozess wartet
      - Von Prozess/Thread zu Ressource

    - Belegungskante
      - Ressource ist vom Prozess belegt (z.B. R4)
      - Von Ressource zum Prozess/Thread

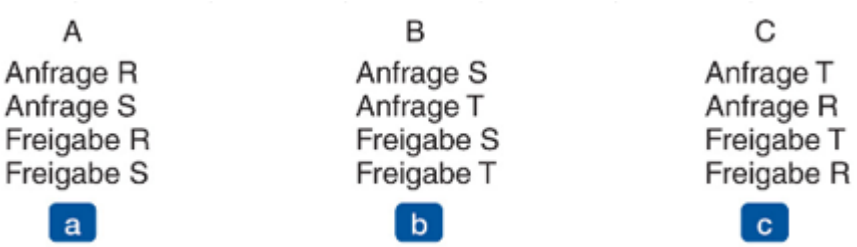
  - Kriterium für Deadlock: Zyklus im Graph

    z.B. P4 wartet auf R3, R3 ist belegt von P7, P7 wartet auf R9, R9 ist von P4 belegt



Deadlocks / Verklemmungen

- Beispiel: mit Verklemmung (ungünstiger Verlauf)



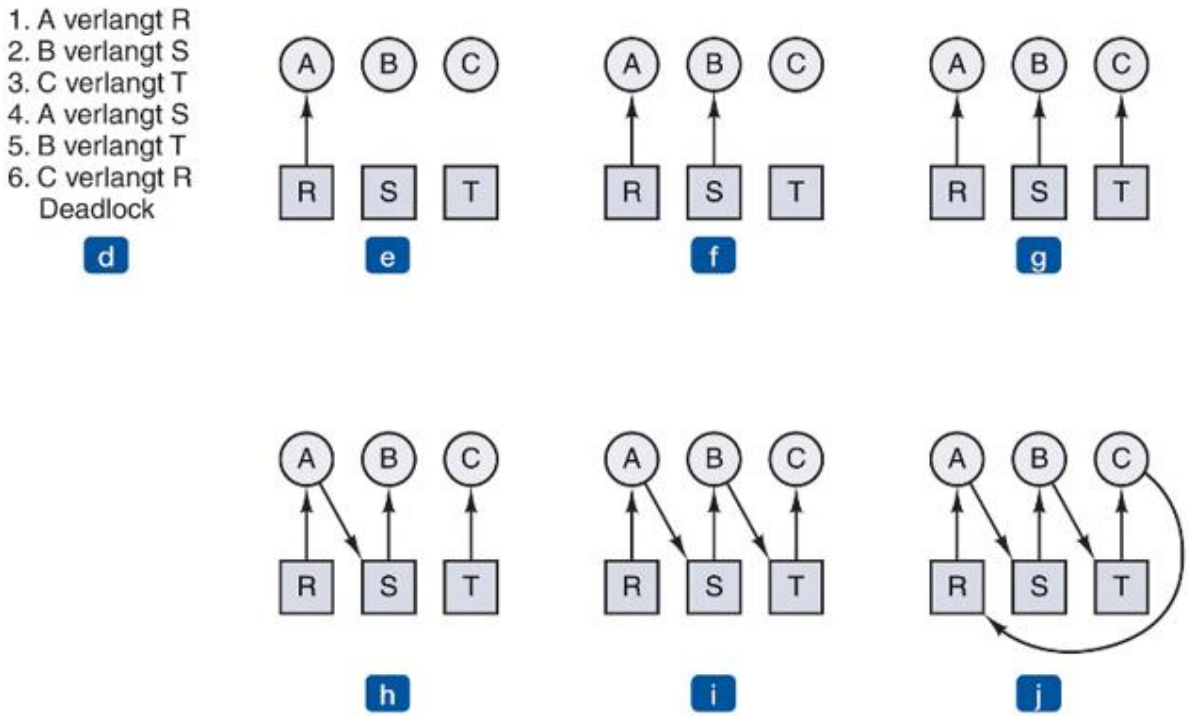
Beispiel:
- 3 Prozesse: A, B, C, 3 Ressourcen: R, S, T
- a-c zeigt, wie Ressourcen reserviert und freigegeben werden
- Betriebssystem kann zu jedem Zeitpunkt jeden nicht blockierten Prozess ausführen
  → Kein Deadlock, keine Konkurrenz um die Ressourcen, aber keine parallele Ausführung

→ Prozesse fordern Ressourcen an und geben sie frei UND
→ Prozesse rechnen aber auch und machen Eingabe-/Ausgabeoperationen
→ Wenn Prozesse sequ. ausgeführt werden, ist während der Ausführung des Prozesse die CPU in Benutzung, während ein Prozess auf Eingabe-/Ausgabeoperationen wartet.
  → Nicht optimal

Fortsetzung nächste Folie

Deadlocks / Verklemmungen

- Beispiel: mit Verklemmung (ungünstiger Verlauf)



Annahme: Prozesse rechnen und führen Eingabe-/Ausgabeoperationen durch; Round Robin könnte sinnvoll sein

- Bei 4. Anforderung wird S von A angefordert → A blockiert, wartet bis S frei wird
- Bei 5. Anforderung wird T von B angefordert → B blockiert, wartet bis T frei wird
- Bei 6. Anforderung wird R von C angefordert → C blockiert, wartet bis R frei wird

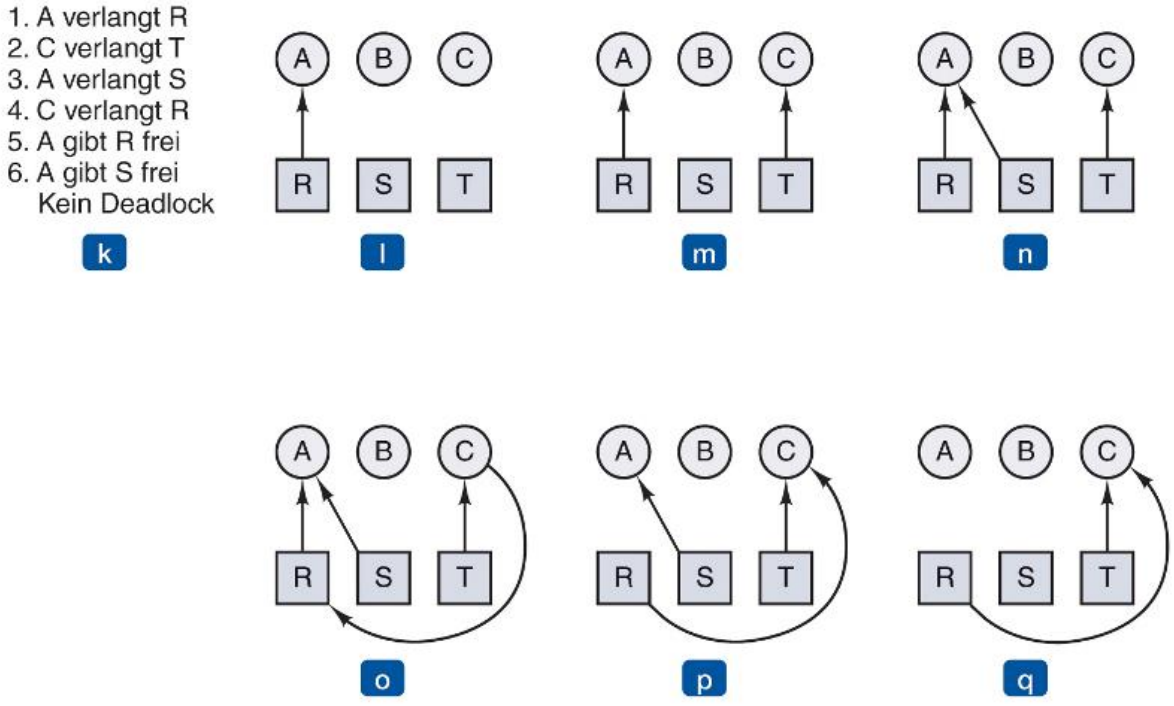
Fortsetzung nächste Folie

→Zyklus → Deadlock

- Beispiel: mit Verklemmung (günstiger Verlauf)

1. A verlangt R
2. B verlangt S
3. C verlangt T
4. A verlangt S
5. B verlangt T
6. C verlangt R
   Deadlock

d



Betriebssystem kann sich Reihenfolge der Ausführung aussuchen.
Betriebsystem kann Zuteilung einer freien Ressource verweigern, falls diese zu Deadlock führen könnte (Prozess wird blockiert, er bekommt keine Rechenzeit)

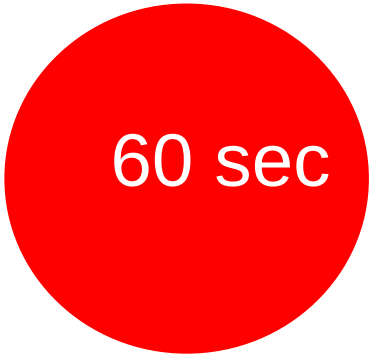
- Im Beispiel: Betriebssystem erkennt Deadlock, kann B blockieren (gibt S nicht an B); C und A werden ausgeführt

→ kein Deadlock

Aufgabe/Frage

- Überlegen Sie welche Möglichkeiten das Betriebssystem hat, um Deadlocks zu behandeln!

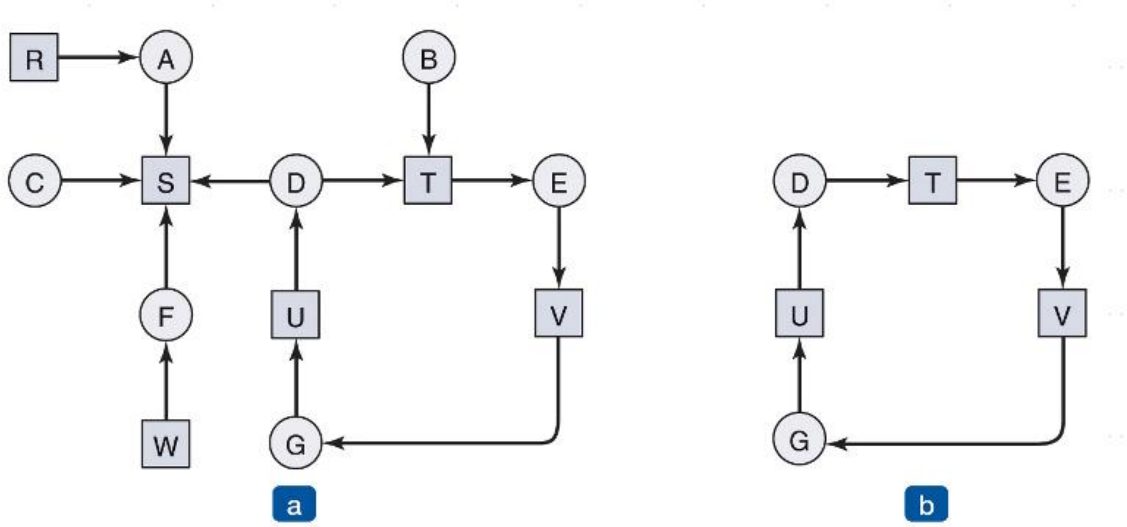
Bedingung:
→ 60 sec



Deadlocks / Verklemmungen

Beispiel

1. B belegt nichts, verlangt T
2. C belegt nichts, verlangt S
3. D belegt U und verlangt Sund T
4. E belegt T und verlant V
5. F belegt W und verlangt S
6. G belegt V und verlangt



Deadlocks / Verklemmungen

- Erkennung von Deadlocks
  - Verallgemeinerung: mehrere Ressourcen pro Typ
    - Prozess wartet nicht mehr auf eine bestimmte Ressource, sondern auf irgendeine Ressource des passenden Typs
      Ressourcen-Belegungs-Graph nicht mehr ausreichend
      => Matrix basierter Algorithmus
  - Modellierung des Systems:
    - n Prozesse P1, … Pn
    - m Klassen/Typen von Ressourcen
    - Klasse/Typ i (1 <= i <= m) enthält Ei Ressource-Instanzen
      - Ressourcen-Vektor E
        » Anzahl verfügbarer Ressourcen für jede(n) Klasse/Typ

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
  - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
  - Modellierung des Systems: ...
    - Klasse i (1 <= i <= m) wurde x_i mal belegt. Damit sind noch A_i Instanzen der Klasse frei
      - Ressourcenrest-Vektor ***A***
        - » Anzahl noch freien Ressourcen für jede(n) Klasse/Typ
    - Klasse j (1 <= i <= m) wurde vom Prozess i (1 <= j <= n) C_{ij} mal belegt
      - Belegungsmatrix ***C***
        - » Anzahl der Ressourcen der Klasse j, die durch den Prozess Pi belegt sind
      - Invariante: $\forall j = 1 \ldots \mathrm{m}: \sum_{i=1}^{n} C_{ij} + A_j = E_j$

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
  - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
  - Modellierung des Systems: ...
    - Klasse j (1 <= i <= m) wurde vom Prozess i (1 <= j <= n) R_{ij} mal angefordert
      - Anforderungsmatrix ***R***
        - » Anzahl der Ressourcen der Klasse j, die durch den Prozess P_i angefordert sind

$$\boldsymbol{E} = (E_1, E_2, E_3, \ldots, E_m) \qquad \boldsymbol{A} = (A_1, A_2, A_3, \ldots, A_m)$$

$$\boldsymbol{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix} \qquad \boldsymbol{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

- Erkennung von Deadlocks
  - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
    - Beispiel:

Bandlaufwerke, Plotter, Scanner, CD-ROM

insgesamt

$$\boldsymbol{E} = (4, 2, 3, 1)$$

frei

$$\boldsymbol{A} = (2,1,0,0)$$

belegt

$$\boldsymbol{C} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

angefordert

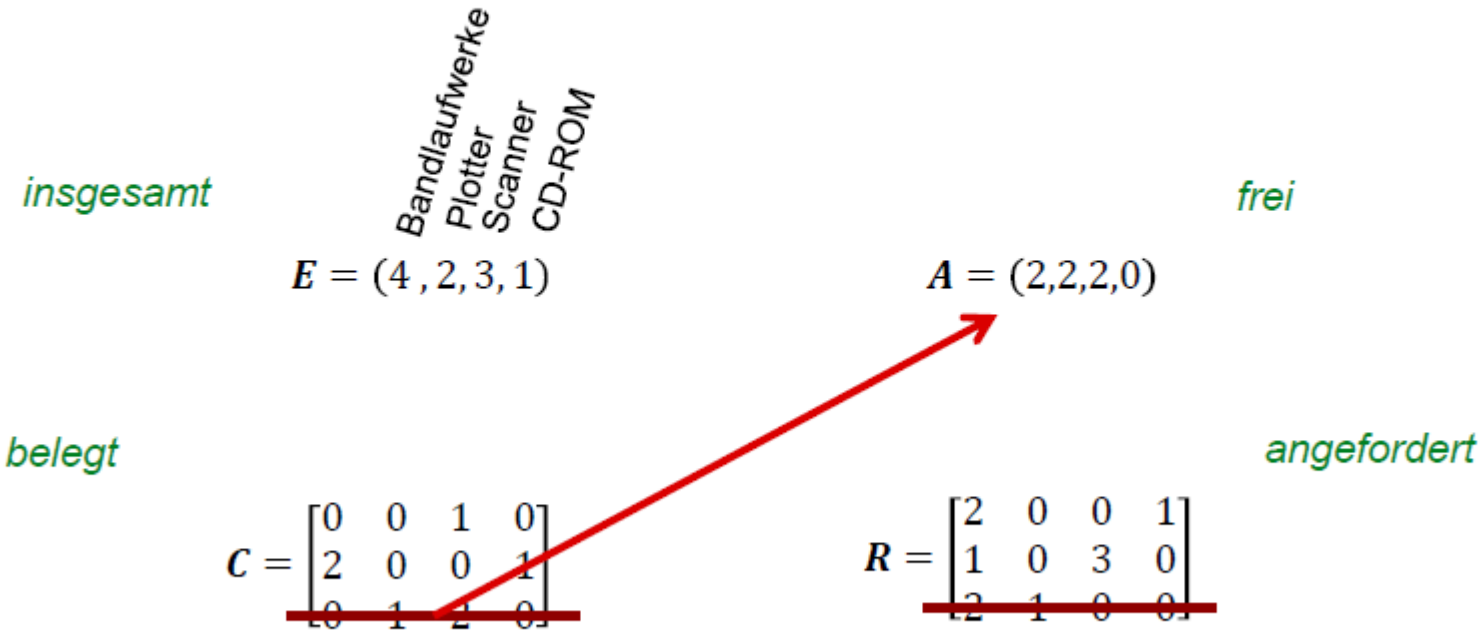
$$\boldsymbol{R} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 3 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

→

Prozess 3 erfüllt die Bedingung

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
  - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
    - Beispiel:



Weder Prozess 1 oder 2 erfüllen die Bedingung, => Deadlock

Aufgabe/Frage

- Sie kennen jetzt eine Vorgehensweise zur Erkennung von Deadlocks.

- Wie muss die Gesamtanzahl R verändert werden, so daß es zu keinem Deadlock kommt? Bitte um Erklärung und Darstellung.

Bedingung:

→ 10 min

Aufgabe/Frage

Im realen Leben kommt es oft zu Situationen von ‚Verklemmungen' (Deadlocks).
Im folgenden Beispiel sollten Sie entscheiden, ob ein Deadlock vorliegt. Wenn Ja, warum? Wenn Nein, warum nicht?

Hier das Beispiel:
Sie und ihr Freund/in leihen sich bei der örtlichen Bibliothek jeweils ein Buch aus. Nun stellen Sie beide fest, dass Sie auch noch das Buch des jeweils anderen dazu benötigen und reservieren es. Auf Nachfrage bei der Bibliothek ist jedoch kein weiteres Exemplar beschaffbar. Kommt es zu einem Deadlock?

Bedingung:
→ 5 min

Deadlocks / Verklemmungen

Sie wissen nun, wie man Deadlocks erkennt (zumindest wenn die statischen Ressourcenanforderungen bekannt sind).

Frage:
Wann soll die Überprüfung durchgeführt werden

Deadlocks / Verklemmungen

- Deadlock-Erkennung
  - Zeitpunkt
    - bei jeder Ressourcen-Anforderung
      - Sofortiges Erkennen von Deadlocks
      - Hoher Rechenaufwand
    - Regelmäßiges Suchen (z.B. nach k Minuten)
    - Prozessorauslastung fällt unter eine gewisse Grenze
      - CPU befindet sich im Leerlauf, wenn ausreichend viele Prozesse an einem Deadlock beteiligt sind
      - Rechenkapazität ist in dieser Situation ausreichend verfügbar
      - Deadlock werden bei ständiger Last der CPU nicht entdeckt

Deadlocks / Verklemmungen

- Behebung von Deadlocks (Gesucht: Möglichkeit zur Behebung von Deadlocks und System wieder in Gang bringen)
  - Temporärer Entzug einer Ressource
    - Schwierig bis unmöglich
      - → Meist nur manuell möglich!
  - Rücksetzen eines Prozesses (Rollback)
    - Voraussetzung: Checkpoints
      - ➢ Status des Prozesses wird regelmäßig gesichert
    - Bei einem Deadlock wird der Prozess aus einem früheren Checkpoint zurückgesetzt und neu gestartet
    - Nachteil: Checkpoint Overhead
  - Abbruch eines Prozesses
    - Brutalstmöglich
    - Geht nur mit Prozessen, die problemlos neu gestartet werden können!

Deadlocks / Verklemmungen

- Behebung von Deadlocks
  - Schwierig
  - Welche Alternativen?
    - → Ist die **Vermeidung/Verhinderung von Deadlocks** besser?
    - → Gibt es einen Algorithmus, der Deadlocks zuverlässig verhindert, indem er immer die **richtige Scheduling-Entscheidung** trifft.
    - → Ja, aber… dann müssen bestimmte Informationen **im Voraus** zur Verfügung stehen
      - Welcher Prozess wird wann gestartet?
      - In welcher Reihenfolge werden die Ressourcen angefordert?
      - Wann werden Ressourcen wieder freigegeben?

Deadlocks / Verklemmungen

- Verhindern von Deadlocks

2 Prozesse
2 Ressourcen
Horizontale Achse=Anzahl der Anweisungen von Prozess A
Vertikale Achse=Anzahl der Anweisungen von Prozess B

Bei I1 verlangt A den Drucker, bei I2 den Plotter
Bei I3 und I4 Freigabe der beiden Geräte

B benötigt den Plotter von I5 bis I7 und der Drucker von I6 bis I8

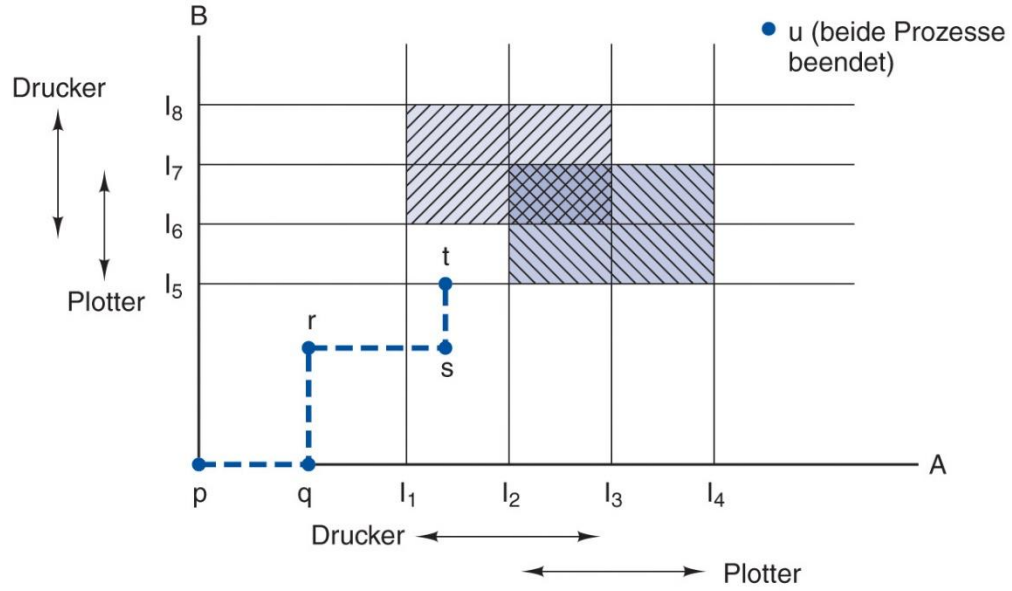


Abbildung 6.8: Ressourcenspur für zwei Prozesse

- Jeder Punkt im Diagramm repräsentiert einen gemeinsamen Zustand der beiden Prozessen
- Mit einem Prozessor verlaufen die Pfade immer horizontal oder vertikal, niemals diagonal. Bewegung geht immer nach rechts oder nach oben, niemals nach links oder unten.

- Ursprung p: Keiner der Prozesse wird ausgeführt
- Scheduler bringt A zur Ausführung: q
- A hat bis q schon eine Anzahl von Ausführungen ausgeführt, B noch keine
- Bei q führt B Ausführungen aus
- Sobald A auf dem Weg von r nach s die Linie I1 überschreitet, reserviert sich A den Drucker
- Am Punkt t reserviert sich B den Plotter
- Schraffierte Bereiche /: enthält die Zustände, in denen A und B den Drucker belegen
  → nicht möglich, Bereich unerreichbar
- Schraffierte Bereiche \: enthält die Zustände, in denen A und B den Plotter belegen
  → nicht möglich, Bereich unerreichbar
- Falls System in Rechteck links und rechts von I1 und I2 und oben und unten von I6 und I5 eintritt, ist Deadlock unvermeidbar. An diesem Punkt verlangt A den Plotter und B den Drucker. Beide sind vergeben
- Das gesamte Rechteck ist unsicher und darf nicht betreten werden
- In t ist die einzige sichere Möglichkeit, A bis I4 auszuführen
- Rechte
- usw.

Deadlocks / Verklemmungen

- Jeder Punkt im Diagramm repräsentiert einen gemeinsamen Zustand der beiden Prozessen
- Mit einem Prozessor verlaufen die Pfade immer horizontal oder vertikal, niemals diagonal. Bewegung geht immer nach rechts oder nach oben, niemals nach links oder unten.

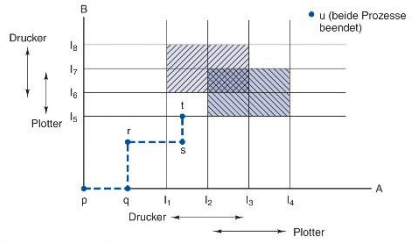


Abbildung 6.8: Ressourcenspur für zwei Prozesse

- Ursprung p: Keiner der Prozesse wird ausgeführt
- Scheduler bringt A zur Ausführung: q
- A hat bis q schon eine Anzahl von Ausführungen ausgeführt, B noch keine
- Bei q führt B Ausführungen aus
- Sobald A auf dem Weg von r nach s die Linie I1 überschreitet, reserviert sich A den Drucker
- Am Punkt t reserviert sich B den Plotter
- Schraffierte Bereiche /: enthält die Zustände, in denen A und B den Drucker belegen
  → nicht möglich, Bereich unerreichbar
- Schraffierte Bereiche \: enthält die Zustände, in denen A und B den Plotter belegen
  → nicht möglich, Bereich unerreichbar
- Falls System in Rechteck links und rechts von I1 und I2 und oben und unten von I6 und I5 eintritt, ist Deadlock unvermeidbar. An diesem Punkt verlangt A den Plotter und B den Drucker. Beide sind vergeben
- Das gesamte Rechteck ist unsicher und darf nicht betreten werden
- In t ist die einzige sichere Möglichkeit, A bis I4 auszuführen

- WICHTIG: B fordert am Punkt t eine Ressource an → System entscheidet, ob B sie bekommt oder nicht
- Wenn JA: System geht in unsicheren Bereich und es entsteht ein Deadlock
- Daher sollte bei t Prozess B blockiert werden

Deadlocks / Verklemmungen

- Verhindern von Deadlocks
  - Sichere und unsichere Zustände
    - Sicherer Zustand:
      Es gibt eine Ausführungsreihenfolge (Scheduling-Reihenfolge), in der alle Prozesse ohne Deadlock zu Ende laufen, selbst wenn die Prozesse sofort die maximalen Resourcenanforderungen stellen
      → Garantie für Beendigung aller Prozesse möglich!

    - Unsicherer Zustand:
      - Sonstige Zustände
      - Anmerkung:
        Unsichere Zustände führen nicht zwangsläufig zu Deadlocks
        - Weil z.B. nicht alle Prozesse gleich die maximalen Ressourcen anfordern!

Deadlocks / Verklemmungen

- Verhinderung von Deadlocks
  - Bankier-Algorithmus
    - Bei jeder Anforderung wird überprüft, ob das System danach in einem sicheren Zustand ist
      - JA: Ressource wird zugeteilt
      - NEIN: anforderender Prozess wird blockiert
    - Sehr restriktive Ressourcenzuteilung
    - Verwende Matrix-Methode für Deadlock-Erkennung
      - R beschreibt die maximalen zukünftigen Forderung!
  - Problem
    - Betriebssystem muss die maximalen zukünftigen Forderungen alle kennen
      - In der Praxis i.a. nicht erfüllt
      - Nur auf spezielle Anwendungsfälle tauglich!

Deadlocks / Verklemmungen

- Vorbeugende Vermeidung von Deadlocks
  - Verhinderung von Deadlock (z.B. durch Bankier-Algorithmus) im Grunde unmöglich
  - Hmmmh – Was unternehmen reale Systeme gegen Deadlocks?
    - Deadlocks sind unmöglich, wenn eine der vier Deadlock-Bedingungen nicht erfüllt ist
  - Wechselseitiger Ausschluss
    - Ressourcen nur dann direkt an einzelne Prozesse zuteilen, wenn dies unvermeidlich ist
      - Bsp: Zugriff auf Drucker über Drucker-Spooler
        - » Keine Konkurrenz um Drucker selbst
        - » Aber evtl Deadlock an anderer Stelle
          - Zwei Prozess schreiben Spooler-Puffer voll!

Deadlocks / Verklemmungen

- Vorbeugende Vermeidung von Deadlocks
  - Hold-und-Wait-Bedingung
    - Vermeide das Warten auf Ressourcen
    - Alle benötigten Ressourcen werden gleichzeitig angefordert
      - Falls eine Ressource nicht verfügbar ist, wird gar keine Ressource belegt und der Prozess wartet
      - Problem:
        - » benötigten Ressourcen müssen im Voraus bekannt sein!
        - » Dann kann man ja den Bankier-Algorithmus anwenden!
        - » Ineffiziente Ausnutzung von Ressourcen
    - Üblich bei Großrechnern und Grid-Systemen
  - Ununterbrechbarkeit (kein Ressourcenentzug)
    - i.a. Unpraktikabel
      - weil schwierig bis unmöglich
      - weil brutalstmöglich!

Deadlocks / Verklemmungen

- Vorbeugende Vermeidung von Deadlocks
  - Unterlaufen der zyklischen Wartebedinung
    - Durchnummerieren der Ressourcen
      - (Scanner 1, Drucker 2, Bandlaufwerk 3, …)
      - Ressourcen werden zu beliebigem Zeitpunkt angefordert, aber nur in aufsteigender Reihenfolge
      - Prozess darf nur Ressourcen mit größerer Nummer anfordern als bereits belegte Ressourcen
      - Deadlocks werden unmöglich!
      - Problem:
        - » Festlegung einer für alle Prozesse tauglichen Ordnung nicht immer praktikabel

Deadlocks / Verklemmungen

- Zusammenfassung
  - Deadlock: eine Menge von Prozessen warten auf Ereignisse, die nur ein anderer Prozess der Menge auslösen kann
    - Deadlock sind nicht deterministisch!
  - Bedingungen für Deadlock
    - Wechselseitiger Ausschluss
    - Hold-und-Wait-Bedingungen
    - Ununterbrechbarkeit (kein Ressourcenentzug)
    - Zyklisches Warten
  - Behandlung von Deadlocks
    - Erkennung von Deadlocks
      - Ressourcen-Belegungs-Graph
      - Matrix-basierter Algorithmus

Deadlocks / Verklemmungen

- Zusammenfassung
  - Behandlung von Deadlocks …
    - Behandlung von Deadlock
      - Schwierig bis unmöglich
        - » meist Abbruch eines Prozesses
    - Vermeidung von Deadlocks
      - Ressourcenvergabe nur wenn sichere Zustände entstehen
        - » Sicherer Zustand: kein Deadlock, selbst wenn alle Prozesse sofort ihre Maximalanforderungen stellen
      - Bankier-Algorithmus
    - Verhinderung von Deadlock
      - Eine der vier Bedingungen unerfüllbar machen!
        - » schwierig!