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
  - Vorhandene Ressourcen werden Prozessen zugeteilt
    - Prozesse sind wesentliche Abstraktion im Betriebssystem
    - Das Betriebssystem teilt z.B. die Ressource CPU mit Hilfe des Schedulings unterschiedlichen Prozessen zu
  - Zahlreiche Ressourcen können (oft) nur jeweils von einem Prozess genutzt werden, z.B. Drucker, Einträge in Systemtabellen,…
    - Wenn z.B. zwei Prozesse gleichzeitig in denselben Eintrag einer Dateizuordnungstabelle schreiben, wird das Dateisytsem inkonsistent.
    - kritische Abschnitte, wechselseitiger Ausschluss, …
    - Verschiedene Synchronisations-Primitive (busy wait, Semaphore, Monitor, …)

Deadlocks / Verklemmungen

Wiederholung

– **Viele Anwendungen verlangen nicht nur auf eine Ressource alleinigen Zugriff, sondern gleich auf mehrere, z.B. 2 Prozesse wollen beide ein Dokument einscannen und dann auf eine DVD-Disk brennen.**

Beispiel:

- Prozess A reserviert den Scanner,
- Prozess B verlangt den DVD-Brenner.
- Als nächstes versucht Prozess A den DVD-Brenner zu reservieren.
- Wird zurückgestellt, weil Prozess B ihn noch nicht freigegeben hat.
- Prozess B verlangt nun den Scanner, gibt den DVD-Brenner aber nicht frei.
- Prozess A wartet auf B, B wartet auf A,....

➡ **Deadlock (deutsch: Verklemmung)**

– Beispiel:
  – Lokales Netz: Deadlocks können auch über mehrere Rechner verteilt sein, z.B. in einem lokalen Netz, in dem gemeinsame Ressourcen von jedem Benutzer benutzt werden können.
  – Datenbankanwendung: Wenn Prozess A den Datensatz R1 und Prozess B den Datensatz R2 sperrt und anschliessend jeder Prozess versucht, den Datensatz des anderen zu sperren.

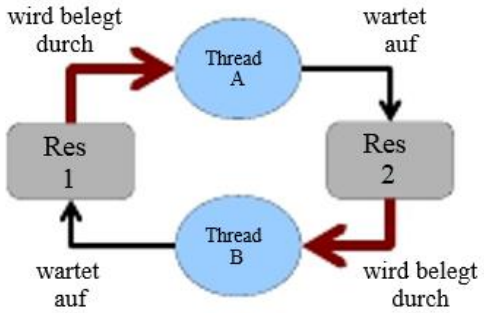
Deadlocks / Verklemmungen

- Die meisten Deadlocks betreffen Betriebsmittel, für welche einigen Prozessen das alleinige Zugriffsrecht erteilt wird. Die reservierten Objekte (Betriebsmittel) nennt man Ressourcen (physisch, Datensatz,..).
  - **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)
        → Deadlocks 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
      - bei einer sehr ungünstigen Scheduling-Strategie, z.B. Scheduling-Strategie sagt, es werden beim Drucken immer kleine Dateien vorgezogen; solange bei der Auswahl immer wieder kleine Dateien vorhanden sind, werden nie große Dateien gedruckt. Der Prozess, der die große Datei drucken möchte, verhungert (kein Deadlock),
      - **Welche Strategie verhindert das Verhungern?**
  - **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
      - Deadlock/Verklemmung braucht externe Eingriffe

Deadlocks / Verklemmungen

- Definition
  - Deadlock
    - ***Eine Menge von Prozessen befindet sich in einem Deadlock-ustand (Verklemmungs-Zustand), wenn jeder Prozess aus der Menge auf ein Ereignis wartet, das nur ein anderer Prozess aus dieser Menge auslösen kann***
      - Meist ist das Ereignis, auf das gewartet wird, die Freigabe einer Ressource
    - Alle Prozesse warten
    - Prozesse werden die Ereignisse, auf die die anderen Prozesse warten, nicht auslösen.
    - Keiner der Prozess wird jemals wieder aufwachen
  - Deadlock sind nicht-deterministisch
    - nur wenn Scheduler und falsches Design zusammenkommen!
    - „Falsches Timing“ notwendig!

Deadlocks / Verklemmungen

- Voraussetzungen für einen (Ressourcen)-Deadlock
  - Bedingung des **Wechselseitigen Ausschlusses:**
    Jede Ressource ist entweder verfügbar oder genau einem Prozess zugeordnet ODER
    Jede Ressource kann zu einem Zeitpunkt von höchstens einem Prozess genutzt werden.

  - **Hold-and-Wait-Bedingung (Besitzen und Warten):**
    Prozesse, die schon Ressourcen reserviert haben, können noch weitere Ressourcen anfordern ODER
    Ein Prozess, der bereits Ressourcen besitzt, kann noch weitere Ressourcen anfordern

  - **Ununterbrechbarkeit (kein Ressourcenentzug):**
    Ressourcen, die einem Prozess bewilligt wurden, können diesem nicht gewaltsam wieder entzogen werden. Der Prozess muss sie explizit freigeben.

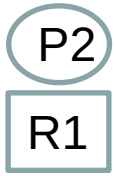
  - **Zyklische Wartebedingung:**
    Es muss eine zyklische Liste/Kette von Prozessen geben, von denen jeder auf eine Ressource wartet, die dem nächsten Prozess in der Kette gehört.

Deadlocks / Verklemmungen

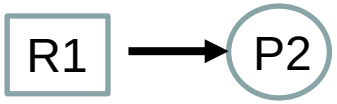
- Bedingungen für einen Deadlock (Coffman et al., 1971)
  - **Alle vier Bedingungen müssen gleichzeitig erfüllt sein**, damit ein Ressourcen-Deadlock entstehen kann. Wenn eine fehlt ist kein Ressourcen-Deadlock möglich.
  - Bedingungen 1-3 sind notwendige Bedingungen
    - aber nicht hinreichend
  - Bedingung 4 ist eine mögliche Konsequenz aus 1-3
    - Die Unauflösbarkeit des zyklischen Warten ist eine Folge aus 1-3
  - **Wenn eine der Bedingungen unerfüllbar ist, können keine Deadlocks auftreten**

Deadlocks / Verklemmungen

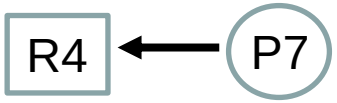
- **Modellierung von Deadlocks mithilfe von gerichteten Graphen (Holt, 1972)**
  - Ressourcen-Belegungs-Graph
  - Zwei Arten von Knoten
    - Prozesse/Thread = Kreis
    - Ressourcen = Quadrate



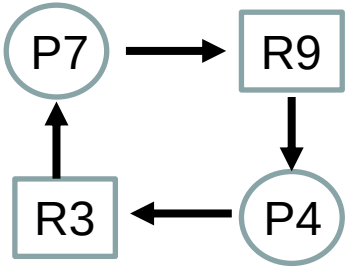
  - Zwei Arten von Kanten
    - **Belegungskante**
      - Ressource wird vom Prozess angefordert (z.B. R1) und belegt
      - gerichtete Kante von einer Ressource zu einem Prozessknoten



    - **Anforderungskante**
      - Prozess wartet auf Ressource (z.B. R4)
      - gerichtete Kante von einem Prozess zu einer Ressource

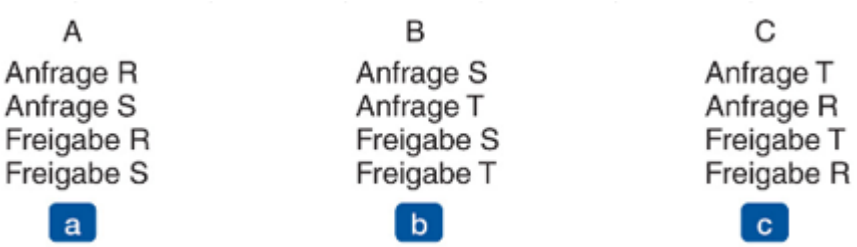


  - **Kriterium für Deadlock: Zyklus im Graph**
    z.B. P4 wartet auf R3, R3 ist belegt von P7, P7 wartet auf R9, R9 ist von P4 belegt



Deadlocks / Verklemmungen

- Beispiel: mit Verklemmung (ungünstiger Verlauf)



Erklärung Beispiel:

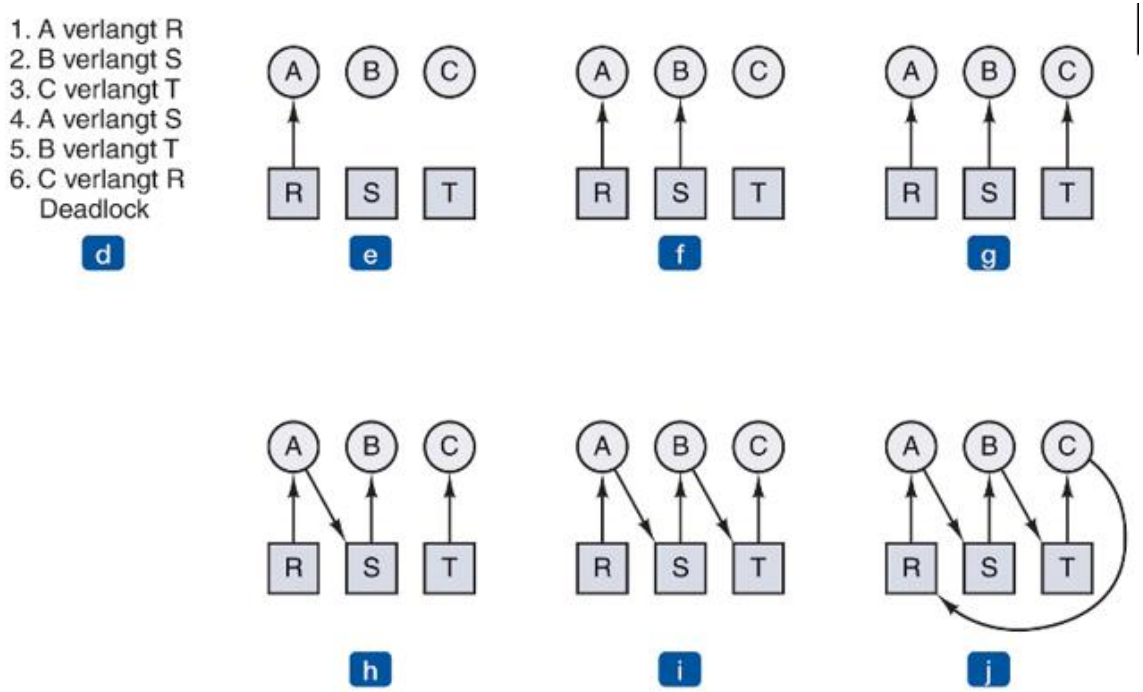
- 3 Prozesse: A, B, C,
- 3 Ressourcen: R, S, T
- **a-c zeigt, wie Ressourcen reserviert und freigegeben werden**
- Betriebssystem kann zu jedem Zeitpunkt jeden nicht blockierten Prozess ausführen
  → Kein Deadlock, keine Konkurrenz um die Ressourcen, aber keine parallele Ausführung

→ Prozesse fordern Ressourcen an und geben sie frei UND
→ Prozesse rechnen aber auch und machen Eingabe-/Ausgabeoperationen
→ Wenn Prozesse sequentiell ausgeführt werden, kann kein anderer Prozess die CPU benutzen, während ein Prozess auf Ein-/Ausgabe wartet. Strenge sequ. Ausführung ist also nicht unbedingt optimal. SJF ist besser als RR, solange keiner der Prozesse Ein-/Ausgabeoperationen ausführt.

Fortsetzung nächste Folie

- Beispiel: mit Verklemmung (ungünstiger Verlauf)

Annahme: Prozesse rechnen und führen Eingabe-/Ausgabeoperationen durch; Round Robin ist sinnvolle Schedulingstrategie



Erklärung:

- 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

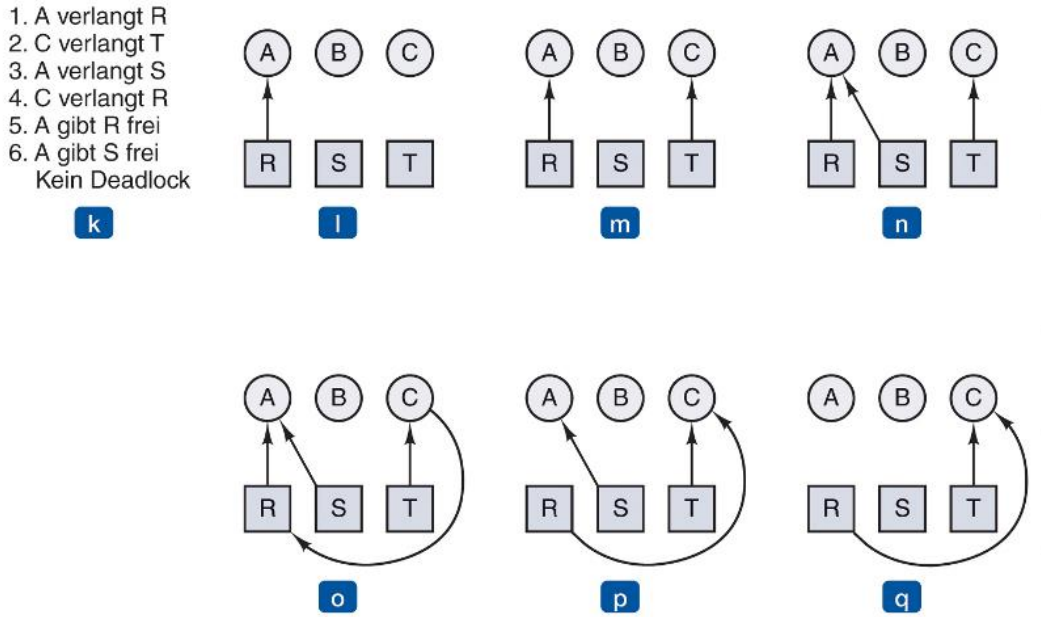
→Zyklus → **Deadlock**

Fortsetzung nächste Folie

- Beispiel: mit Verklemmung (günstiger Verlauf)
  - Betriebssystem kann sich Reihenfolge der Ausführung aussuchen.
  - Betriebssystem kann Zuteilung einer freien Ressource verweigern, falls diese zu einem Deadlock führen könnte (Prozess wird blockiert, er bekommt keine Rechenzeit zugeteilt)

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



Erklärung:
- Betriebssystem erkennt Deadlock, kann B blockieren (gibt S nicht an B); C und A werden ausgeführt
- Nach Schritt (q) kann B dann die Ressource S zugeteilt bekommen, weil A fertig ist und C schon alles hat

→ kein Deadlock

Aufgabe/Frage

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

Bedingung:

→ ad hoc

Lösung

Antwort:

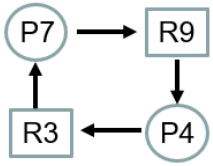
- Behandlung von Deadlocks
  - Vogel-Strauß-Methode
    - hoffen, daß alles gut geht! (Ignoranz)
    - Nicht unüblich (UNIX)
  - Deadlock-Erkennung und Behebung
    - Es werden alle Anforderungen zugelassen, um bei Bedarf Deadlock aufzulösen
    - Erkennen von Deadlocks notwendig
  - Vermeidung/Verhinderung von Deadlock
    - Lasse das System nie in eine Deadlock-Situation kommen
    - Verhindern von Deadlocks durch vorsichtige Ressourcen-Zuteil ung
    - Vermeiden von Deadlock durch Unerfüllbarkeit einer der Deadlock-Bedingungen (1-4)

Deadlocks / Verklemmungen

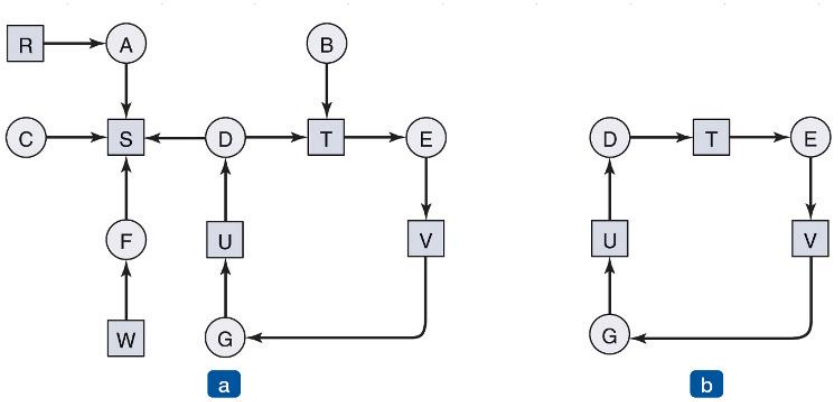
- Behandlung von Deadlocks
  - **Vogel-Strauß-Methode**
    - Einfachster Ansatz: Kopf in den Sand stecken und so tun als gäbe es kein Problem
    - Mathematiker finden diese Methode indiskutabel; Aussage: Deadlocks müssen unbedingt vermieden werden
    - Ingenieure fragen erst mal nach, wie oft das Problem auftritt, wie oft das System aus anderen Gründen abstürzt und wie schwerwiegend ein Deadlock ist.

Deadlocks / Verklemmungen

- Behandlung von Deadlocks
  - **Deadlock-Erkennung und Behebung**
    - Es werden Deadlocks zugelassen und das System soll diese erkennen und anschliessend etwas dagegen unternehmen
    - Voraussetzung: Erkennen von Deadlocks
    - **Einfacher Fall:** nur eine Ressource in jeder Klasse (System hat z.B. ein Scanner, einen DVD-Brenner, einen Plotter,…, aber jeweils nur ein Gerät jeder Sorte. Enthält dieser Graph einen Zyklus, dann gibt es auch einen Deadlock.



    - **Komplexer:** 7 Prozesse A-G, 6 Ressourcen R-W.
      1. A belegt R und verlangt S
      2. B belegt nichts, verlangt T
      3. C belegt nichts, verlangt S
      4. D belegt U und verlangt S und T
      5. E belegt T und verlangt V
      6. F belegt W und verlangt S
      7. G belegt V und verlangt



Frage: Ist dieses System in einem Deadlock-Zustand? Welche Prozesse sind beteiligt?

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 reicht nicht mehr aus
      => Matrix basierter Algorithmus zur Erkennung von Deadlocks in einer Menge von n Prozessen P1 bis Pn

  - Modellierung des Systems:
    - Menge von n Prozesse P1, … Pn
    - m ist Anzahl der Ressourcenklassen
    - E mit i bezeichnen die Ressourcen der Klasse/Typ i (1 <= i <= m)
      → E heißt **Ressourcen-Vektor E (existing resource vector)** und gibt die Anzahl der Ressourcen an, die von jeder Klasse **insgesamt** vorhanden sind.
      » E ist Anzahl verfügbarer Ressourcen für jede(n) Klasse/Typ
      z.B. DVD-Brenner=Klasse1, vorhanden 2 → E von 1=2

$$\boldsymbol{E} = (E_1, E_2, E_3, \ldots, E_m)$$

- Erkennung von Deadlocks
  - Modellierung des Systems:
    - Zu jedem Zeitpunkt sind einige Ressourcen belegt.
      Der **Ressourcenrestvektor A (available resource vector)** enthält für jede Ressource i die Anzahl der freien Instanzen A von i.
      z.B. beide Klasse1-DVD-Brenner belegt → A von 1=0, es gibt keine freie Instanz von DVD-Brenner.

$$\boldsymbol{A} = (A_1, A_2, A_3, \ldots, A_m)$$

- Erkennung von Deadlocks
  - Modellierung des Systems:
    - Es werden noch weitere Matrizen benötigt:
      - Die aktuelle **Belegungsmatrix C** (current allocation matrix) und die **Anforderungsmatrix R** (request matrix)
      - Die i-te Zeile von C enthält die Anzahl der Ressorucen, die der Prozess P von von jeder Klasse belegt.
        → C von i und j ist die Anzahl der Ressourcen der Klasse j, die Protess i belegt.
        → R von i und j ist die Anzahl der Ressorucen der Klasse j, die Prozess i gerne hätte.

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

- Erkennung von Deadlocks
  - Modellierung des Systems (zusammengefasst):
    - **Ressourcen-Vektor E** (existing resource vector)
    - **Ressourcenrestvektor A** (available resource vector)
    - **Belegungsmatrix C** (current allocation matrix)
    - **Anforderungsmatrix R** (request matrix)

$$\boldsymbol{E} = (E_1, E_2, E_3, \ldots, E_m)$$

$$\boldsymbol{A} = (A_1, A_2, A_3, \ldots, A_m)$$

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

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

- Erkennung von Deadlocks
  - Beispiel zur ‚Berechnung' eines Deadlocks
    Gegeben ist:

Gesamt vorhanden sind
4 Band, 2 Plotter, 3 Scanner, 1 DVD-LW
E= 4, 2, 3, 1

Frei
A= 2, 1, 0, 0

Belegt

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix} \begin{matrix} P1 \\ P2 \\ P3 \end{matrix}$$

Angefordert

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

Gibt es einen Deadlock?

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
  - Beispiel

Gesamt vorhanden sind
4 Band, 2 Plotter, 3 Scanner, 1 DVD-LW
E= 4, 2, 3, 1

Frei
A= 2, 1, 0, 0

Belegt

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix} \begin{matrix} P1 \\ P2 \\ P3 \end{matrix}$$

Angefordert

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

Erklärung zu C:
P1 belegt 1 Scanner
P2 belegt 2 Band-LW + 1 DVD-LW
P3 belegt 1 Plotter + 2 Scanner

Erklärung zu R:
P1 fordert 2 Band-LW + 1 DVD-LW
P2 fordert 1 Band-LW + 1 Scanner
P3 fordert 2 Band-LW + 1 Plotter +1 DVD-LW

Step 1:
P1 fordert 2 Band-LW + 1 DVD-LW → 1 DVD-LW nicht frei
P2 fordert 1 Band-LW + 1 Scanner → 1 Scanner nicht frei
P3 fordert 2 Band-LW + 1 Plotter + 1 DVD-LW → 1 DVD-LW nicht frei

⇒ 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:

→ 5 min

Lösung

Antwort:
P3 darf kein DVD-LW anfordern.

Gesamt vorhanden sind
4 Band, 2 Plotter, 3 Scanner, 1 DVD-LW
E= 4, 2, 3, 1

Frei
A= 2, 1, 0, 0

Belegt

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix} \begin{matrix} P1 \\ P2 \\ P3 \end{matrix}$$

Angefordert

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

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
      - Vorteil: Sofortiges Erkennen von Deadlocks
      - Nachteil: Hoher Rechenaufwand
    - Regelmäßiges Suchen (z.B. nach k Minuten)
    - Wenn Prozessorauslastung unter eine gewisse Grenze fällt
      - CPU befindet sich oft im Leerlauf, wenn ausreichend viele Prozesse an einem Deadlock beteiligt sind
      - Wenige Prozesse ausführbar → Rechenkapazität ist in dieser Situation ausreichend verfügbar
      - Nachteil: Deadlocks werden bei ständiger Last der CPU nicht entdeckt, weil die Prozessorauslastung nie unter eine gewisse Grenze fällt.

Deadlocks / Verklemmungen

- **Behebung von Deadlocks**
  (Gesucht: Möglichkeit zur Behebung von Deadlocks und System wieder in Gang bringen)
  - Temporärer Entzug einer Ressource und Vergabe an einen anderen Prozess
    - Schwierig bis unmöglich → Meist nur manuell möglich!
  - Rücksetzen eines Prozesses (Rollback)
    - Zustand eines Prozesses wird in regelmäßigen Abständen (checkpoints) in eine Datei geschrieben
      Voraussetzung: Checkpoints
    - Bei einem Deadlock wird der Prozess aus einem früheren Checkpoint zurückgesetzt und neu gestartet
    - Nachteil: Checkpoint Overhead
  - Abbruch eines Prozesses
    - Brutal, aber mit Glück laufen andere Prozesse weiter.
    - Geht nur mit Prozessen, die problemlos neu gestartet werden können!

- 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?
      - …

- 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

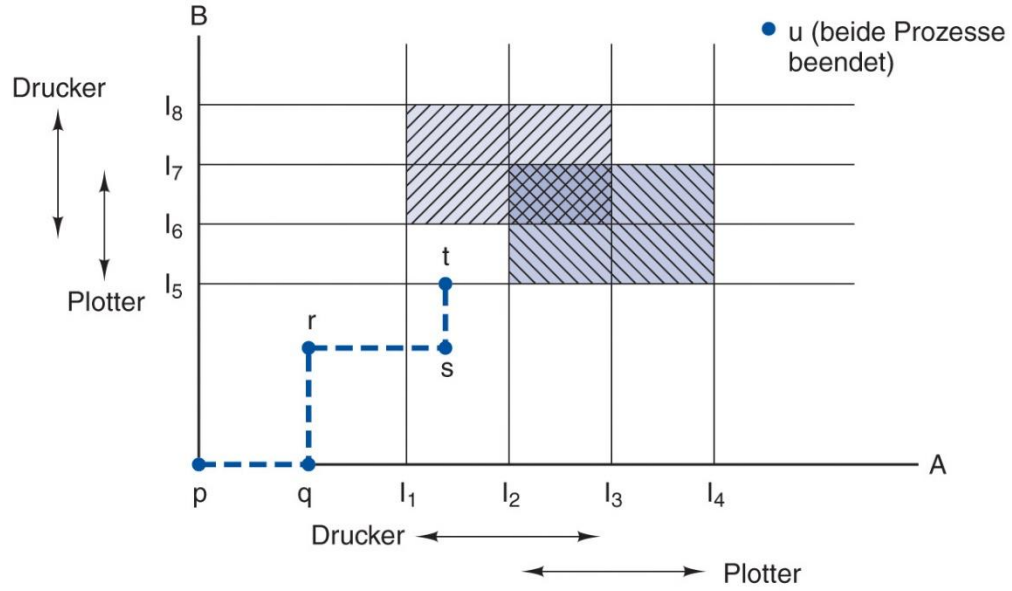


Abbildung 6.8: Ressourcenspur für zwei Prozesse

- 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.

- Verhindern von Deadlocks
  - Jeder Punkt im Diagramm repräsentiert einen gemeinsamen Zustand der beider 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.

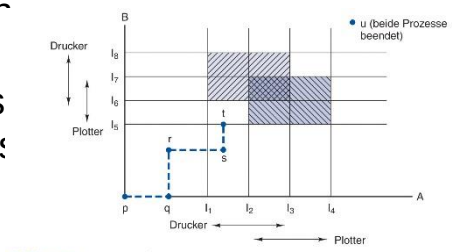


Abbildung 6.8: Ressourcenspur für zwei Prozesse

  - 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

- 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!

- Verhindern von Deadlocks
  - Sichere und unsichere Zustände
    Beispiel: sicherer Zustand

	Belegt	Max.
A	3	9
B	2	4
C	2	7

Frei: 3

- Prozess A hat 3 Instanzen belegt; benötigt später bis zu 9 Instanzen
- Prozess B hat 2 Instanzen belegt; benötigt später bis zu 4 Instanzen
- Prozess C hat 2 Instanzen belegt; benötigt später bis zu 7 Instanzen
- Insgesamt existieren 10 Instanzen → Verfügbar: 3

Scheduler kann ausschliesslich B ausführen.

	Belegt	Max.
A	3	9
B	4	4
C	2	7

Frei: 1

Sobald B beendet:

	Belegt	Max.
A	3	9
B	0	-
C	2	7

Frei: 5

Scheduler kann C ausführen:

	Belegt	Max.
A	3	9
B	0	-
C	7	7

Frei: 0

Sobald C beendet:

	Belegt	Max.
A	3	9
B	0	-
C	0	-

Frei: 7

Jetzt sind die sechs Instanzen, die A benötigt, frei und A kann zu Ende laufen.

→ Der Zustand am Anfang ist sicher, weil das System durch überlegtes und vorausschauendes Scheduling einen Deadlock verhindern kann.

- Verhindern von Deadlocks
  - Sichere und unsichere Zustände
    Beispiel: unsicherer Zustand

	Belegt	Max.
A	4	9
B	2	4
C	2	7

Frei: 2

- Gleicher Anfangszustand, nur A wurde noch eine weitere Ressource zugeteilt, so dass A 4 Instanzen belegt und später bis zu 9 Instanzen benötigt.
- Prozess B hat 2 Instanzen belegt; benötigt später bis zu 4 Instanzen
- Prozess C hat 2 Instanzen belegt; benötigt später bis zu 7 Instanzen
- Insgesamt existieren 10 Instanzen → Verfügbar: 3

Scheduler könnte B ausführen, bis B alle seine Ressourcen anfordert.

	Belegt	Max.
A	4	9
B	4	4
C	2	7

Frei: 0

Sobald B beendet:

	Belegt	Max.
A	4	9
B	0	-
C	2	7

Frei: 4

Es gibt vier freie Instanzen; jeder aktive Prozess braucht fünf.

→ **Es gibt keine Reihenfolge, die garantiert, dass alle Prozesse zu Ende laufen können.**

Deadlocks / Verklemmungen

- Verhinderung von Deadlocks
  - **Bankier-Algorithmus (banker‘s algorithm, Dijkstra, 1965)***
    - Bei jeder Anforderung wird überprüft, ob das System danach in einem sicheren Zustand ist
      - Bei JA: Ressource wird zugeteilt
      - Bei NEIN: anfordernder 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 für spezielle Anwendungsfälle tauglich!

*Warum Bankier-Algorithmus?
Er funktioniert so, wie ein Kleinstadtbankier die Kreditwünsche einer Gruppe von Kunden behandeln könnte. Die Gruppe der Kunden erhält nur Geld geliehen, wenn der Bankier wusste, daß es zurückbezahlt werden konnte. Der Algorithmus überprüft bei jedem Kreditantrag, ob der Kredit zu einem sicheren oder einem unsicheren Zustand führt. Führt er zu einem sicheren Zustand, wird der _Kredit bewilligt, ansonsten wird er angelehnt.

- Vorbeugende Vermeidung von Deadlocks
  - Verhinderung von Deadlocks (z.B. durch Bankier-Algorithmus) ist im Grunde möglich
  - Was unternehmen reale Systeme gegen Deadlocks?
    - **Deadlocks sind unmöglich, wenn eine der vier Deadlock-Bedingungen nicht erfüllt ist**

      - **Unterlaufen des Wechselseitigen Ausschlusses**
        - keine Ressource darf jemals einem Prozess exklusiv zugeteilt werden
        - Ressourcen nur dann direkt an einzelne Prozesse zuteilen, wenn dies unvermeidlich ist.
          z.B. Daten: nur Lesezugriff zulassen, sodass Prozesse sie gleichzeitig nutzen können
          z.B. Drucker: Zugriff auf Drucker über Drucker-Spooler
          - Keine Konkurrenz um Drucker selbst
          - Evtl. könnte ein Deadlock an anderer Stelle auftauchen, z.B. wenn 2 Prozesse den Spooler-Puffer vollschreiben

- Vorbeugende Vermeidung von Deadlocks

  - **Unterlaufen der Hold- und Wait-Bedingung**
    - Vermeide das Warten auf Ressourcen, während sie andere Ressourcen belegen
    - Jeder Prozess fordert alle benötigten Ressourcen **IM VORAUS** an. Wenn die benötigten Ressourcen verfügbar sind, werden sie ihm gleichzeitig zugeteilt und der Prozess kann ausgeführt werden
    - Fall eine Ressource nicht verfügbar ist, wird gar keine Ressource belegt und der Prozess wartet
      - Problem:
        - Die meisten Prozesse wissen im Voaus nicht, wie viele Ressourcen sie benötigen
        - Wenn sie es wüßten → Bankier-Algorithmus
        - Ineffiziente Ausnutzung von Ressourcen
    - Üblich bei Großrechnern (Stapelverarbeitungssysteme verlangen die benötigten Ressourcen in erster Zeile eines Jobs) und Grid-Systemen

  - **Unterlaufen der Ununterbrechbarkeit (kein Ressorucenentzug)**
    - i.A. unpraktikabel
      - Schwierig bis unmöglich anzuwenden; Ressourcenvirtuali-sierung, z.B. Spooling-Drucker?

- Vorbeugende Vermeidung von Deadlocks

  - **Unterlaufen der zyklischen Wartebedingung**
    - Jedem Prozess wird zu jedem Zeitpunkt nur eine Ressource erlaubt; wenn er eine zweite Ressource benötigt muss er die Erste freigeben
    - Durchnummerieren der Ressorucen
      - (Scanner1, Scanner2, Plotter3,…)
      - Ressorucen werden zu beliebigem Zeitpunkt angefordert, aber nur in aufsteigender Reihenfolge
      - Prozess darf nur Ressourcen mit größerer Nummer anfordern als bereits belegte Ressoruce
      - Deadlocks werden unmöglich
      - Problem:
        - Festlegung einer für alle Prozesse tauglichen Ordnung ist 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-Bedingungne
      - Ununterbrechbarkeit
      - Zyklisches Warten

    - Behandlung von Deadlocks
      - Erkennung von Deadlocks
        - Ressourcen-Belegungs-Graph
        - Matrix-basierter Algorithmus

Deadlocks / Verklemmungen

- Zusammenfassung ff.
  - Behandlung von Deadlocks
    - 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 Deadlocks
    - Eine der vier Bedingungen unerfüllbar machen
    - Schwierig!