

Lösungsvorschläge zur
Theoretischen Übungsserie A
Verteilte Systeme im HS 2011

Benedikt Ostermaier
(ostermaier@inf.ethz.ch)

Research Group for Distributed Systems
ETH Zürich

23. Dezember 2011
Überarbeitet am 9. Januar 2012

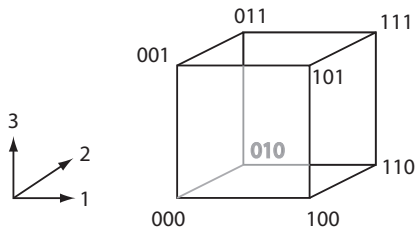
1 Topologien und Pfadlängen

1.1.) 256 Knoten, 16×16 Gitter

Maximale Pfadlänge: 30 Schritte

1.2.) 256 Knoten, Hypercube

- ▶ Dimension des Hypercubes: $d = \log_2(256) = 8$



- ▶ Max. Anzahl von Schritten = Max. Anzahl Bitflips in der Adresse = $d = 8$

2 Pfade im Hypercube

2.1) Abstand der Knoten $a = (0, 1, 1, 0, 1, 0)$ und $b = (1, 1, 0, 0, 1, 1)$ eines 6-dimensionalen Hypercubes?

Zwei Knoten mit Abstand k unterscheiden sich in k Bits ihrer Adresse. Der Abstand lässt sich wie folgt berechnen:

$$\delta = \sum_i |a_i - b_i| =$$
$$|0 - 1| + |1 - 1| + |1 - 0| + |0 - 0| + |1 - 1| + |0 - 1| = 3$$

2.2) Wieviele Pfade gibt es zwischen zwei Knoten im Abstand k eines Hypercubes der Dimension d ?

- ▶ Annahme: k Bits unterscheiden sich in Quell- und Zieladresse
- ▶ Im ersten Schritt k Möglichkeiten, im zweiten $(k - 1)$, ...
- ▶ Das ergibt $k!$ Pfade

2 Pfade im Hypercube (II)

2.3) Wieviele dieser Pfade sind *knotendisjunkt*?

Gegeben: Zwei Knoten im Abstand k

Behauptung: k solche Pfade

- ▶ Maximal k knotendisjunkte Pfade, denn Startknoten kann auf dem Weg zum Ziel zwischen k Nachbarn wählen
- ▶ Mindestens k Pfade *existieren*:
 - ▶ Pfad ist gegeben durch Folge von Dimensionswechslern (z.B. 1,4,3,2)
 - ▶ Anordnung der *Dimensionswechsel*, so dass sich keine gemeinsamen Zwischenknoten auf verschiedenen Pfaden ergeben
 - ▶ Start mit n_1, n_2, \dots, n_k
 - ▶ 1. Rotieren um eine Stelle nach links: $n_2, n_3, \dots, n_k, n_1$
 - ▶ 2. Rotieren: $n_3, n_4, \dots, n_k, n_1, n_2$
 - ▶ ...
 - ▶ $(k - 1)$. Rotieren: n_k, n_1, \dots, n_{k-1}
 - ▶ Präfixe dieser Pfade enthalten alle unterschiedliche Elemente!

3 Fehlermodelle

Anmerkung: Nicht alle Fehlerfälle lassen sich eindeutig einer bestimmten Fehlerklasse zuordnen.

1. (Webserver) Fail-Stop. Sender: Browser, Empfänger: Server. Der Server antwortet mit einer Fehlermeldung.
2. (Mobiltelefon) Crash bzw. Fail-Stop. Sender: GSM-Netz/anderer Teilnehmer, Empfänger: Mobiltelefon. Das Netz erkennt "Verlust" durch Timeout.
3. (Virus) Byzantinischer Fehler. Das Verhalten des Virus ist beliebig. Dass ein Fehler auftritt, kann nicht in jedem Fall festgestellt werden.
4. (W-LAN) Fehlerhaftes Senden und/oder Empfangen bzw. fehlerhaftes Übertragen. Hier ist der Link selbst betroffen, nicht einer der Kommunikationsteilnehmer.

3 Fehlermodelle (II)

5. (E-Mail) Zeitfehler. Der Empfänger (Benutzer) erhält die Nachricht zu spät.
6. (Spam-Filter) Fehlerhaftes Empfangen. Die Nachricht (Mail) wird vom Empfänger (Spam-Filter) falsch behandelt. Der Sender merkt davon nichts.
7. (Drucker) Byzantinischer Fehler oder fehlerhaftes Empfangen. Entweder hat der Druckertreiber einen Bug (byz. Fehler) oder bei der Übertragung zum Drucker gehen Daten verloren, so dass der Drucker die Postscript-Datei nicht als solche erkennt.

4 Fehlertoleranz

Gegeben: Zuverlässigkeit eines Dienstes pro Rechner bei 60%.

4.1.) Zuverlässigkeit bei 3 Rechnern?

$$1 - (1 - 0.6)^3 = 0.936 = 93.6\%$$

4.2.) Benötigte Replikate um 99% Verfügbarkeit zu erreichen?

$$1 - (1 - 0.6)^n = 0.99 \Rightarrow 0.01 = 0.4^n \Rightarrow n = \log_{0.4}(0.01) = 5.03$$

Es werden also $\lceil 5.03 \rceil = 6$ Rechner (5 Replikate) benötigt.

4.3.) Ausfallzeit in Stunden/Jahr bei 99% Verfügbarkeit?

$$(1 - 0.99) * (365 * 24) = 87.6 \text{ Stunden}$$

5 Synchroner Kommunikation mit Mars-Rover?

Wie schnell darf der Mars-Rover maximal fahren?

- ▶ Signallaufzeit Erde \leftrightarrow Mars:
55.7 Mio km / 300'000 km/s \approx 186s.
- ▶ Benötigte Zeit um auf ein Hindernis zu reagieren:
[0.999...]s Kompensation Bildwiederholungsrate +
2 * Signallaufzeit + 4s Reaktionszeit + $\frac{1}{10}$ s für Steuerbefehl
= 377.1 Sekunden.
- ▶ Max. erkennbare Distanz zum Hindernis: 10m

\Rightarrow In 377.1 Sekunden darf der Rover max. 10m zurücklegen, d.h.

$$v_{max} = 10m/377.1s = 0.0265m/s = 2.65cm/s = 95m/h$$

Anmerkung:

- ▶ Ein echter Mars-Rover ist auf maximal 5 cm/s ausgelegt und bewegt sich real mit ca. 1 cm/s.

6 Kommunikation

6.1) Wie kann es bei synchroner Kommunikation zwischen zwei Prozessen zu einem Deadlock kommen?

Durch gleichzeitiges, gegenseitiges Warten, z.B. wenn sich zwei Teilnehmer gleichzeitig einen Auftrag schicken.

6.2) Bei welchem Kommunikationsmechanismus besteht nur eine geringe Gefahr für Deadlocks? Begründen Sie Ihre Antwort.

Bei mitteilungsorientierter, asynchroner Kommunikation wird nie gewartet, also kann es prinzipiell zu keinem gleichzeitigen, gegenseitigen Warten kommen.

6 Kommunikation (II)

6.3) Warum bevorzugen Programmierer trotzdem RPC?

Bei RPC ist der Aufwand für die Verwendung entfernter Prozeduren minimal. Der Aufruf von lokalen und entfernten Prozeduren folgt der gleichen Syntax. Programmierer können also ihre gewohnten, prozeduralen/objektorientierten Lösungsmuster verwenden.

6.4) Was ist der Unterschied zwischen synchroner, mitteilungsbasierter und synchroner, auftragsorientierter Kommunikation ohne Rückgabewert?

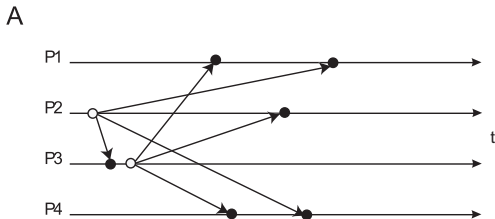
Im ersten Fall wartet der Sender nur, bis eine Bestätigung des Eingangs der Mitteilung vorliegt. Im zweiten Fall wartet er, bis auf der Empfängerseite der Auftrag tatsächlich abgearbeitet wurde.

7 RPC

1. Referenzen sind grundsätzlich nicht erlaubt, da eine Referenz (Pointer) auf Rechner A auf Rechner B keine Bedeutung hat. Das gilt sowohl für Eingabe- wie Ausgabeparameter. Man könnte jedoch die Datenstruktur hinter der Referenz sequenzialisieren und in dieser Form übertragen.
2. Es besteht die Gefahr, dass eine Anfrage mehrfach bearbeitet wird. Sequenznummern helfen.
3. At-least-once

8 Broadcast

8.1) Atomarer Broadcast? Beispiel A



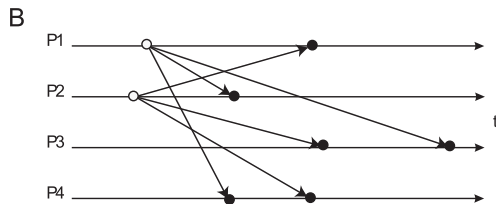
Def. atomarer (totaler) Broadcast: wenn P_1 und P_2 die Nachrichten N, M erhalten, ist die Empfangsreihenfolge bei beiden Prozessen die gleiche.

Beachte: Das Senden einer Nachricht zählt nicht als Empfang!

Hier: Empfangsreihenfolge bei P_1 und P_4 gleich \Rightarrow **atomarer Broadcast**

8 Broadcast (II)

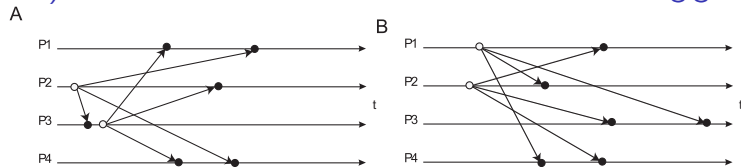
8.1) Atomarer Broadcast? Beispiel B



Empfangsreihenfolge bei P_3 und P_4 unterschiedlich \Rightarrow **kein atomarer Broadcast**

8 Broadcast (III)

8.2) Sind die Broadcasts voneinander kausal abhängig?



Simpler Test auf kausale Abhängigkeit: gibt es einen Pfad (von links nach rechts), der vom Sendeereignis von X zum Sendeereignis von Y führt?

A: Ja! **Aber:** Kausale Reihenfolge ist nicht gewahrt, denn die Reihenfolge bei den Empfängern P_1 und P_4 entspricht nicht der Kausalität!

B: Nein, keine kausale Abhängigkeit der Broadcasts

8 Broadcast (IV)

8.3) Totale Ordnung mit zentralem Sequencer

Frage: Sind Broadcasts, die über einen zentralen Sequencer gesendet werden, notwendigerweise total geordnet? Welche Bedingung muss gelten?

Falls Kanäle vom Sequencer zu den Empfängern **FIFO-Eigenschaft** besitzen, dann gilt totale Ordnung, denn: wenn Nachricht X vor Y gesendet, dann kommt X vor Y an (Nachrichten überholen sich auf FIFO-Kanal nicht)

FIFO-Eigenschaft kann z.B. über Acknowledgements hergestellt werden (Sequencer braucht aber ein ACK von *allen* Empfängern)

9 Lamport-Zeit

Wiederholung: Kausalrelation \prec auf Ereignissen
("happened before")

Es sei $x \prec y$ genau dann, wenn:

1. x und y auf dem gleichen Prozess stattfinden und x vor y kommt, *oder*
2. x ist ein Sendeereignis und y ist das korrespondierende Empfangsereignis *oder*
3. $\exists z$ mit $x \prec z \wedge z \prec y$

9 Lamport-Zeit (II)

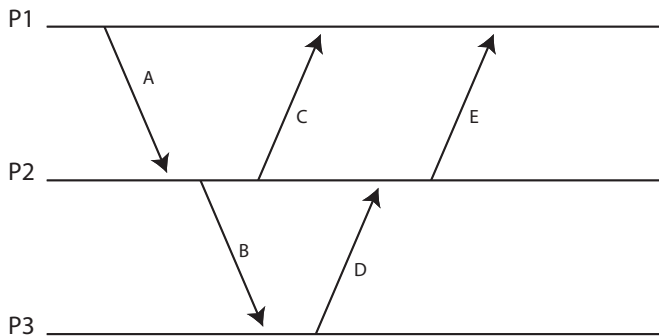
Wiederholung: Uhrenbedingung

- ▶ $C : E \rightarrow N$ ist Abbildung von Ereignissen auf Zeitstempel
- ▶ Es soll gelten: $e \prec e' \Rightarrow C(e) < C(e')$

Protokoll, das Uhrenbedingung implementiert:

- ▶ Lokale Uhr (= „Zähler“) tickt bei jedem Ereignis
- ▶ Sendeereignis: Uhrwert mitsenden (Zeitstempel)
- ▶ Empfangsereignis: $\max(\text{lokale Uhr, Zeitstempel})$,
anschliessend ticken!

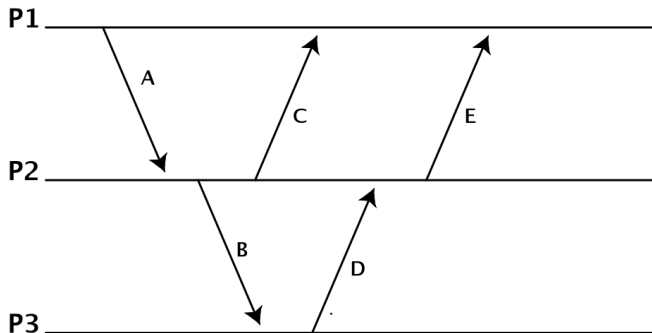
9 Lamport-Zeit (III)



9.1) Gesucht: Paar von Ereignissen, welche nicht kausal abhängig sind (Korrigiert!).

Beispiele: (B.receive, C.receive), (C.receive, E.send)

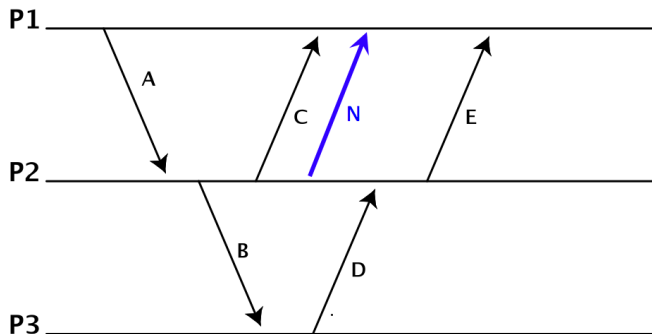
9 Lamport-Zeit (IV)



9.2) Einfügen einer Nachricht N, so dass

- ▶ $A.\text{receive} \prec N.\text{send} \wedge C(N.\text{receive}) < C(E.\text{send})$
- ▶ Absender und Empfänger der Nachricht können frei gewählt werden, müssen aber verschieden sein.

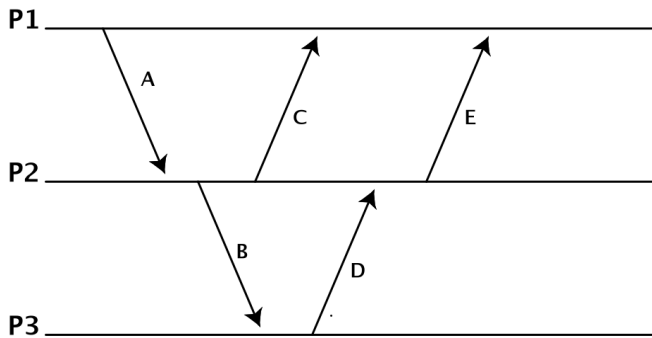
9 Lamport-Zeit (IV)



9.2) Einfügen einer Nachricht N, so dass

- ▶ $A.\text{receive} \prec N.\text{send} \wedge C(N.\text{receive}) < C(E.\text{send})$
- ▶ Absender und Empfänger der Nachricht können frei gewählt werden, müssen aber verschieden sein.

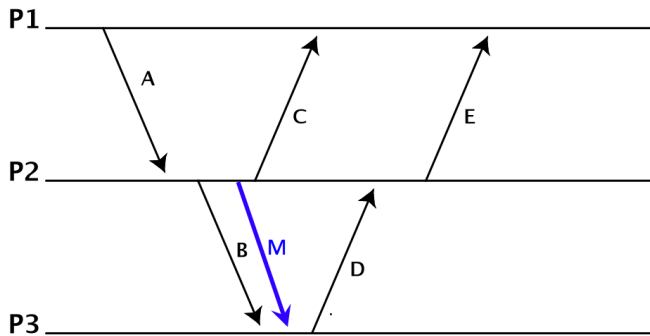
9 Lamport-Zeit (V)



9.3) Einfügen einer Nachricht M, so dass

- ▶ $M.send \prec C.send \wedge C(B.receive) < C(M.receive)$
- ▶ Sender: P2, Empfänger: P3

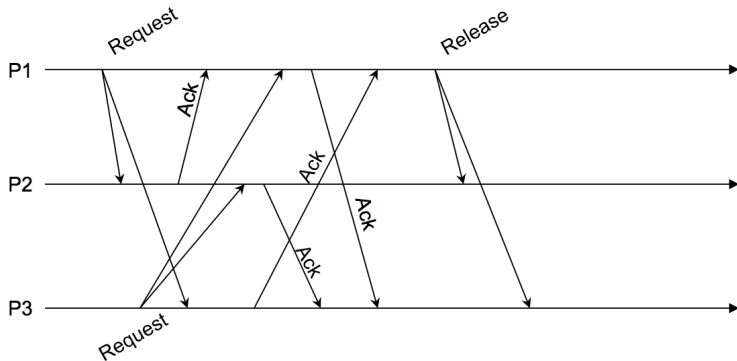
9 Lamport-Zeit (V)



9.3) Einfügen einer Nachricht M, so dass

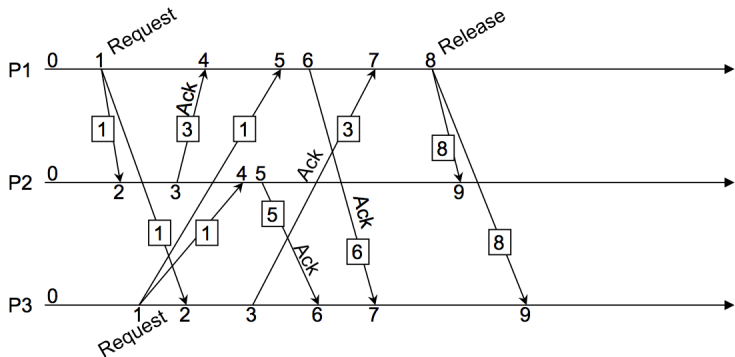
- ▶ $M.send \prec C.send \wedge C(B.receive) < C(M.receive)$
- ▶ Sender: P2, Empfänger: P3

10 Wechselseitiger Ausschluss mit Lamport-Zeit



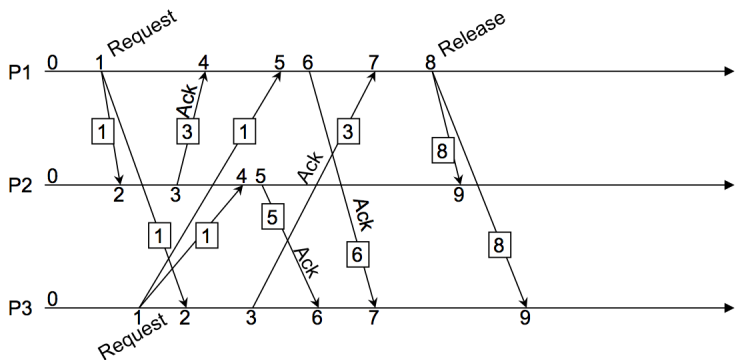
10.1 Sende- und Empfangszeitstempel für jedes Ereignis?

10 Wechselseitiger Ausschluss mit Lamport-Zeit



10.1 Sende- und Empfangszeitstempel für jedes Ereignis?

10 Wechselseitiger Ausschluss mit Lamport-Zeit (II)



Die Prozesse wenden das aus der Vorlesung bekannte Verfahren zum wechselseitigen Ausschluss an, das Lamport-Zeit und verteilte Warteschlangen benutzt.

10 Wechselseitiger Ausschluss mit Lamport-Zeit (III)

Wiederholung: Algorithmus von Lamport (1978)

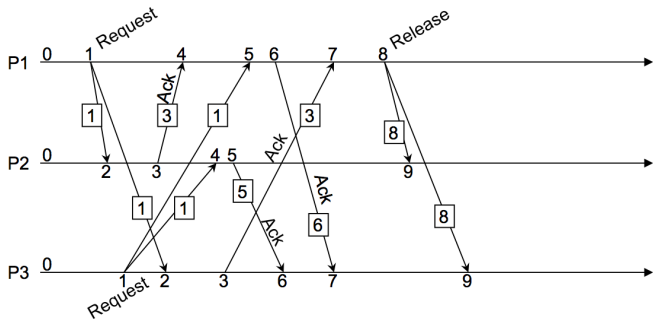
1. Bei „request“ des Betriebsmittels: Request mit Zeitstempel und Absender an alle versenden und in eigene queue einfügen.
2. Bei Empfang einer request-Nachricht: Request in eigene queue einfügen, ack versenden.
3. Bei „release“ des Betriebsmittels: Aus eigener queue entfernen, release-Nachricht an alle versenden.
4. Bei Empfang einer release-Nachricht: Zugehörigen request aus eigener queue entfernen.
5. Ein Prozess darf das Betriebsmittel benutzen, wenn gilt:
 - ▶ Eigener request ist frühester in seiner queue *und*
 - ▶ er hat bereits von jedem anderen Prozess (irgendeine) spätere Nachricht bekommen.

10 Wechselseitiger Ausschluss mit Lamport-Zeit (IV)

Wiederholung: Algorithmus von Lamport (1978)

- ▶ Problem: Lamport-Zeit ist nicht injektiv
- ▶ Erweiterung unter Einbeziehung einer Ordnung \triangleleft auf Prozessen:
- ▶ $(C(e), P_i) < (C(e'), P_j)$
 $\Leftrightarrow C(e) < C(e') \vee C(e) = C(e') \wedge P_i \triangleleft P_j$
- ▶ Bei gleichem Zeitstempel ist das Ereignis auf dem „kleineren“ Prozess das „frühere“
- ▶ Nun haben wir eine total geordnete Lamport-Zeit

10 Wechselseitiger Ausschluss mit Lamport-Zeit (V)



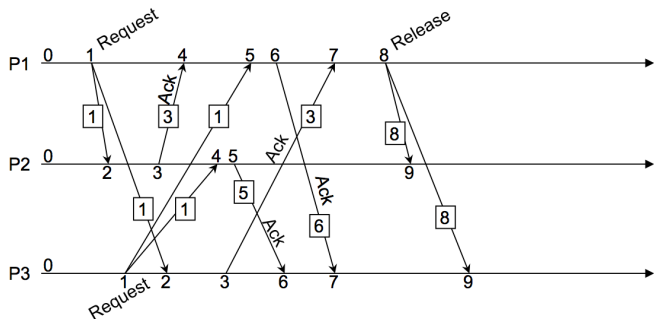
10.2. Geben Sie für die Prozesse 1 und 3 an, wie die Warteschlange des jeweiligen Prozesses nach jedem Sende- bzw. Empfangsereignis aussieht.

10 Wechselseitiger Ausschluss mit Lamport-Zeit (VI)

10.2. Beispiel Prozess 1:

- 1) Eigene Requestnachricht: Request(1,P1)
- 2) Ack von P2: Request(1,P1)
(Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3)
- 3) Request von P3: Request(1,P1); Request(1,P3)
(Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3)
- 4) Ack an P3: Request(1,P1); Request(1,P3)
(Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3)
- 5) Ack von P3: Request(1,P1); Request(1,P3)
(Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3, letzte Nachricht von P3 mit Zeitstempel 3)
- 6) Eigene Releasenachricht: Request(1,P3)
(Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3, letzte Nachricht von P3 mit Zeitstempel 3)

10 Wechselseitiger Ausschluss mit Lamport-Zeit (VII)

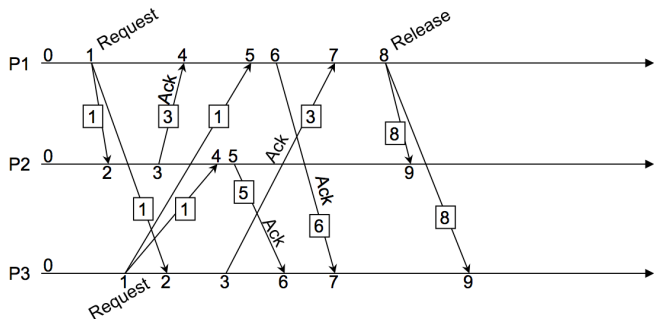


10.3. Sind beim Einreihen in die Warteschlange die Sende- oder die Empfangszeitstempel zu verwenden? Warum?

Die Sendezeitstempel, da sonst keine global-konsistente Sicht garantiert werden kann:

- ▶ Q1: Request(1,P1); Ack(4,P2); Request(5,P3); ...
- ▶ Q3: Request(1,P3); Request(2,P1); Ack(6,P2); ...

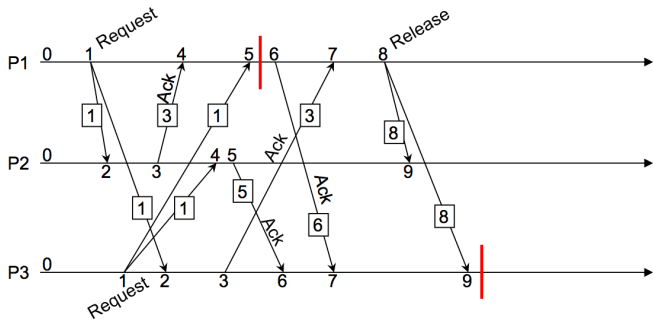
10 Wechselseitiger Ausschluss mit Lamport-Zeit (VIII)



10.4. Welche Bedingung muss erfüllt sein, damit Prozess 1 auf die Ressource zugreifen kann?

- ▶ Eigener Request ist der „früheste“
- ▶ „Spätere“ Nachrichten von allen anderen Teilnehmern erhalten
- ▶ Achtung: Injektive Lamport-Zeit!

10 Wechselseitiger Ausschluss mit Lamport-Zeit (IX)



10.5. Markieren Sie den Zeitpunkt im Zeitdiagramm, zu dem Prozess 1 bzw. Prozess 3 auf die Ressource zugreifen kann.

11 Client/Server

11.1. Bei Web-basierten Diensten wird oft ein Bezeichner in Links codiert („URL rewriting“), um die aktuelle Transaktion zu identifizieren. Wie könnte ein Unbefugter eine laufende Transaktion „übernehmen“ und was kann man gegen diese Gefahr tun?

- ▶ Unbefugter verwendet eine URL mit gültiger Transaktions-ID
- ▶ Timeouts erhöhen Sicherheit

Besser: Keine sicherheitsrelevanten Merkmale in der URL codieren.

11 Client/Server (II)

11.2. Welches Problem entsteht bei einem zustandsbehafteten Server, wenn viele Clients abstürzen, bevor sie ihre Transaktionen beendet haben?

- ▶ Es werden Ressourcen auf dem Server blockiert, obwohl diese nicht mehr benötigt werden.

11.3. Erläutern Sie kurz ein paar Vorteile von zustandslosen gegenüber zustandsbehafteten Client/Server-Protokollen und umgekehrt.

- ▶ Vorteile zustandslos: Effizienz, Robustheit des Servers gegen eigenen Crash und Client-Crash
- ▶ Vorteile zustandsbehaftet: Sitzung kann über mehrere Interaktionen gehalten werden, potentielle Reduktion der übertragenen Datenmenge

12 Jini

12.1. Erläutern Sie kurz die Funktion von Leases.

- ▶ Zeitlich befristeter Vertrag über einen Service zwischen zwei Parteien.

12.2. Eine Besonderheit von Jini ist das Ausnutzen der Mobilität von Java-Code. Welche Code-Teile werden übertragen und welche Möglichkeiten ergeben sich dadurch?

- ▶ Proxy-Objekte implementieren die Serviceschnittstelle. Sie werden bei der Registrierung des entsprechenden Services beim Lookup-Service hinterlegt und bei der Nutzung des Services zum Client übertragen.
- ▶ Dies ermöglicht die Verwendung spezieller Protokolle sowie „smart proxies“: Zusätzliche Intelligenz in den stubs um z.B. Vorverarbeitung oder Datenkompression durchzuführen.

13 Modellierung von Web-Schnittstellen

Zwei Paradigmen aus der Vorlesung:

- ▶ Service-orientiert: WS-*/SOAP
- ▶ Ressourcen-orientiert: REST/HTTP

Aufgabe: Modellierung der API eines digitalen Bilderrahmens

- ▶ Ziel: Auseinandersetzung mit den verschiedenen Paradigmen
- ▶ Viele mögliche Lösungen. Hier nur *ein* Lösungsvorschlag.

13 Modellierung von Web-Schnittstellen (II)

13.1. Darstellung eines unformatierten Textes auf dem Bildschirm

- ▶ Service-orientiert: `showText([string])`
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: `/display/presentation`
 - ▶ Methoden: PUT
 - ▶ Repräsentation: `text/plain`
 - ▶ Erlaubte Werte: (Beliebiger Text)

13.2. Darstellung eines JPEG-Bildes auf dem Bildschirm

- ▶ Service-orientiert: `showImage([image])`
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: `/display/presentation`
 - ▶ Methoden: PUT
 - ▶ Repräsentation: `image/jpeg`
 - ▶ Erlaubte Werte: JPEG-kodiertes Bitmap

13 Modellierung von Web-Schnittstellen (III)

13.3. Darstellung einer Diashow, welche im Atom- bzw. RSS-Format übertragen wird

- ▶ Service-orientiert:
 - ▶ `showRSS([string])`
 - ▶ `showATOM([string])`
 - ▶ Alternativ: `showFeed([string])`
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: `/display/presentation`
 - ▶ Methoden: PUT
 - ▶ Repräsentationen: `application/atom+xml`,
`application/rss+xml`
 - ▶ Erlaubte Werte: RSS-Feed, ATOM-Feed

13 Modellierung von Web-Schnittstellen (IV)

13.4. Abfrage der aktuellen Darstellungskonfiguration

- ▶ Service-orientiert:
 - ▶ [int] getShowType()
 - ▶ [string] getText()
 - ▶ [image] getImage()
 - ▶ [string] getFeed()
 - ▶ Alternativ: setPresentation([presentation]) für 13.1-13.3 und [presentation] getPresentation(). Klasse presentation kapselt die verschiedenen Präsentationsmodi.
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: /display/presentation
 - ▶ Methoden: GET
 - ▶ Repräsentationen: Hier muss im Request der Accept-Header weggelassen oder auf */* gesetzt werden
 - ▶ Rückgabe: Aktueller Stand, welcher mittels PUT auf derselben Ressource gesetzt wurde
 - ▶ Entsprechender Content-Type
 - ▶ Entsprechender Inhalt (Text, JPEG-Bild, RSS-Feed, ...)

13 Modellierung von Web-Schnittstellen (V)

13.5. Abfrage des momentanen Bildschirminhalts

- ▶ Service-orientiert: [image] `getCurrentImage()`
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: `/display/currentimage`
 - ▶ Method: `GET`
 - ▶ Repräsentation: `image/jpeg`
 - ▶ Rückgabe: JPEG-Bild des aktuellen Bildschirminhalts

13 Modellierung von Web-Schnittstellen (VI)

13.6. Setzen/Auslesen der Konfiguration der Diashow

- ▶ Service-orientiert:

- ▶ `setShowConfiguration([configuration])`
- ▶ `[configuration] getShowConfiguration()`
- ▶ `[configuration]: {`
 - `[int] delay // 0-65535`
 - `[string] animation // none, fade, turn`
 - `[string] ordering // feed, random``}`

13 Modellierung von Web-Schnittstellen (VII)

13.6. Setzen/Auslesen der Konfiguration der Diashow

- ▶ Ressourcen-orientiert:
 - ▶ Pfad: `/configuration/delay`
 - ▶ Method: GET, PUT
 - ▶ Repräsentation: `text/plain`
 - ▶ Erlaubte Werte: 0-65535

 - ▶ Pfad: `/configuration/animation`
 - ▶ Method: GET, PUT
 - ▶ Repräsentation: `text/plain`
 - ▶ Erlaubte Werte: none, fade, turn

 - ▶ Pfad: `/configuration/ordering`
 - ▶ Method: GET, PUT
 - ▶ Repräsentation: `text/plain`
 - ▶ Erlaubte Werte: feed, random

13 Modellierung von Web-Schnittstellen (VIII)

13.7. Setzen/Auslesen der Bildschirmhelligkeit

- ▶ Service-orientiert:
 - ▶ `setDisplayBrightness([int])`
 - ▶ `[int] getDisplayBrightness()`
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: `/display/brightness`
 - ▶ Method: GET, PUT
 - ▶ Repräsentation: `text/plain`
 - ▶ Erlaubte Werte: 1, 2, 3, ... 16

13 Modellierung von Web-Schnittstellen (V)

13.8. Auslesen des Bewegungssensors

- ▶ Service-orientiert: [boolean] isMotion()
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: /sensors/motion
 - ▶ Method: GET
 - ▶ Repräsentation: text/plain
 - ▶ Erlaubte Werte: false, true

13.9. Auslesen des Helligkeitssensors

- ▶ Service-orientiert: [int] getBrightness()
- ▶ Ressourcen-orientiert:
 - ▶ Pfad: /sensors/brightness
 - ▶ Method: GET
 - ▶ Repräsentation: text/plain
 - ▶ Erlaubte Werte: 0, 1, 2, ... 255

14 Sicherheit

14.1. Auf welche Herausforderungen trifft man bei der Schlüsselverteilung in verteilten Systemen?

- ▶ “Offenheit” verteilter Systeme fördert Angriffe
- ▶ Heterogenität der Systeme sorgt für zusätzliche Schwachstellen
- ▶ Keine zentrale Sicherheitsautorität

14.2. Beschreiben Sie zwei mögliche Lösungsansätze zur Schlüsselverteilung.

- ▶ Schlüsselvergabe über einen sicheren Kanal oder per (aufwendigem) Public-Key-Verfahren
- ▶ Direkte Schlüsselvereinbarung per Sicherheitsprotokoll (z.B. Diffie-Hellman).

14 Sicherheit (II)

14.3. Der Diffie-Hellman-Algorithmus

a) Wofür wird der Algorithmus verwendet?

- ▶ Der Diffie-Hellman-Algorithmus stellt ein kryptographisches Protokoll dar. Dieses dient zur Erstellung eines geheimen Schlüssels zwischen Kommunikationspartnern über einen unsicheren Kanal.

14 Sicherheit (III)

14.3. Der Diffie-Hellman-Algorithmus (II)

b) Beschreiben Sie kurz das Verfahren.

- ▶ Zwei Kommunikationspartner (A und B) kennen beide eine (grosse) Primzahl p und eine Konstante c , mit $1 < c < p$. Sie nutzen eine Einwegfunktion $f(x) = c^x \bmod p$.
- ▶ A und B wählen je eine Zufallszahl a bzw. b , die geheim gehalten werden muss.
- ▶ Aus den gegebenen Werten berechnen die Kommunikationspartner $\alpha = c^a \bmod p$ bzw. $\beta = c^b \bmod p$.
- ▶ A sendet α an B und B sendet β an
- ▶ Jetzt können A und B jeweils den gemeinsamen, geheimen Schlüssel berechnen: $G_A = \beta^a \bmod p$ und $G_B = \alpha^b \bmod p$.
- ▶ Da $(c^b \bmod p)^a \bmod p = (c^a \bmod p)^b \bmod p$ gilt, gilt auch $G_A = G_B$.

14 Sicherheit (IV)

14.3. Der Diffie-Hellman-Algorithmus (III)

- c) Was ist ein möglicher Angriff und wie könnte man sich dagegen verteidigen?
- ▶ Als „man in the middle“ könnte man in den Kanal zwischen zwei Kommunikationspartnern eindringen und sich jeweils als Gegenstelle ausgeben. So werden für beide Teilstrecken eigene Schlüssel ausgehandelt und der Angreifer kann die Nachrichten transparent weiterleiten, sie dabei aber mitlesen und auch verändern.
 - ▶ In der Vorlesung wurde ein Verfahren vorgestellt, mit dem man diesen Angriff erkennen kann (→ Kap. Sicherheit, S.30).

14 Sicherheit (V)

14.4. Wie funktioniert zertifikatsbasierte Authentifizierung? Von welchem weitverbreiteten System wird sie verwendet?

- ▶ Eine vertrauenswürdige Autorität signiert mittels asymmetrischer Verschlüsselung ein Zertifikat, welches die Identität einer Person oder Ressource zusammen mit dessen Public-Key enthält.
- ▶ Mittels eines Challenge-Response-Verfahrens überprüft die Gegenseite dann, ob die Person/Resource wirklich die ist, die sie vorgibt zu sein. Der Zertifikatsinhaber authentifiziert sich, indem er mit seinem Private-Key eine bestimmte Nachricht signiert, welche mit dem Public-Key aus dem Zertifikat validiert werden kann.
- ▶ SSL/TLS verwendet zertifikatsbasierte Authentifizierung.

14 Sicherheit (VI)

14.5. Wenn One-Time-Pads ein perfektes Verschlüsselungssystem darstellen, warum werden diese dann heutzutage nicht global eingesetzt?

- ▶ One-Time-Pads sind nur unter der Annahme perfekt, dass die beteiligten Parteien bereits im Voraus über einen sicheren Kanal genügend Pads ausgetauscht haben.
- ▶ Die Pads müssen jeweils die gleiche Länge wie die Nachrichten haben und dürfen nur ein einziges Mal verwendet werden.
- ▶ Für Standardanwendungen daher kaum praktikabel.

14 Sicherheit (VII)

14.6. Einwegfunktionen

f sei eine Einwegfunktion und x_1 ein initiales Passwort, aus dem eine Passwortkette erzeugt wird:

$$x_1 \xrightarrow{f} x_2 \xrightarrow{f} \dots \xrightarrow{f} x_{n-1} \xrightarrow{f} x_n$$

- a) Um die Passwörter zur Authentisierung nutzen zu können, muss x_n zunächst zum Server S übertragen werden. Welche der folgenden Anforderungen müssen erfüllt sein:
- i) Ein Angreifer darf nichts über x_n erfahren, die Übertragung muss also geheimnisbewahrend erfolgen. → Nicht erforderlich!
 - ii) Es muss sichergestellt sein, dass x_n bei der Übertragung nicht verändert wird. → Erforderlich!

14 Sicherheit (VII)

14.6. Einwegfunktionen (II)

- b) Annahme: $n = 100$. Server S kennt x_{100} . Bei der zweiten Anmeldung verwendet Client C aus Versehen x_{89} (statt x_{98}). Welche Gefahr besteht, wenn dieser Wert von einem Angreifer abgehört wird und S den Anmeldeversuch einfach ignoriert, weil $f(x_{89}) \neq x_{99}$?
- ▶ Man setzt normalerweise voraus, dass die Einwegfunktion bekannt ist. Ein Angreifer könnte daher $x_{89}, x_{90}, \dots, x_{98}$ berechnen und einsetzen, d.h. er könnte sich bis zu 10-mal anmelden.

14 Sicherheit (VIII)

14.7. Kann bei Kerberos ein verschlüsseltes Ticket Granting Ticket (TGT) von einem anderen Client verwendet werden, um vom Ticket Granting Service (TGS) ein Service Ticket (ST) anzufordern? Begründung!

- ▶ $TGT = \{Name, Client - ID, t, \Delta t, K, \dots\}_{KTGS}$
- ▶ Nein, da der Schlüssel K nur dem ursprünglichen Client bekannt ist. Um TGT zu verwenden, ist zusätzlich ein mit K verschlüsselter Authentizitätsnachweis erforderlich. Die Kenntnis von TGT allein reicht nicht aus.

14.8. Nennen Sie zwei Gründe, warum in Kerberos Key Distribution Center (KDC) und TGS getrennt sind.

- ▶ Skalierbarkeit
- ▶ konzeptionelle Trennung von Akkreditierung und Servicenutzung

Danke für Ihre Aufmerksamkeit

Frohe Weihnachten und einen „guten Rutsch“ ins neue Jahr!