ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2016 Prof. R. Wattenhofer

Technische Informatik II

Klausur

Samstag, 11. Februar 2017, 09:00 - 10:30 Uhr

Nicht öffnen oder umdrehen bevor die Prüfung beginnt!
Lesen Sie die folgenden Anweisungen!

Die Prüfung dauert 90 Minuten und es gibt insgesamt 90 Punkte. Die Anzahl Punkte pro Teilaufgabe steht jeweils in Klammern bei der Aufgabe. Sie dürfen die Prüfung auf Englisch oder Deutsch beantworten. **Begründen Sie** alle Ihre Antworten sofern nichts anderes dabeisteht. Beschriften Sie Skizzen und Zeichnungen verständlich. Was wir nicht lesen können, können wir auch nicht bewerten!

Schreiben Sie zu Beginn Ihren Namen und Ihre Legi-Nummer in das folgende dafür vorgesehene Feld und beschriften Sie **jedes Zusatzblatt** ebenfalls mit Ihrem Namen und Ihrer Legi-Nummer.

Name	**Legi-Nr.**

Aufgabe	**Erreichte Punktzahl**	**Maximale Punktzahl**
1 - Multiple Choice		10
2 - Verschlüsseltes Testament		12
3 - SQL		23
4 - Locking & Concurrency		23
5 - Markovketten		22
Summe		**90**

1 Multiple Choice (10 Punkte)

Geben Sie bei jeder Aussage an, ob sie wahr oder falsch ist. Jede korrekte Antwort gibt 1 Punkt, **jede fehlerhafte Antwort -1 Punkt**. Jede unbeantwortete Aussage gibt 0 Punkte. Wenn die Summe der Punkte für alle Aussagen negativ ist, dann wird die Aufgabe insgesamt mit 0 Punkten bewertet. **In dieser Aufgabe können Sie Ihre Antworten nicht begründen.**

Aussage	**wahr**	**falsch**
Ein externer Host kann mittels NAT eine Verbindung zu einer privaten internen Adresse aufbauen.	☐	☐
Es gibt IP-Adressen, die von mehreren verschiedenen Maschinen genutzt werden.	☐	☐
6ffe:8d::f023:a349::8917 ist eine gültige IPv6 Adresse.	☐	☐
Ein einzelnes TCP Paket geht nie an einem Knoten im Netzwerk verloren.	☐	☐
CSS wird vom Webserver auf die HTML Seiten angewandt, bevor diese zum Client geschickt und im Browser angezeigt werden.	☐	☐
<tag1><tag2>Hallo</tag1></tag2>ist korrekte HTML-Syntax.	☐	☐
HTTP 1.0 kann als unterliegendes Protokoll zum Streamen von Video benutzt werden.	☐	☐
Disk scheduling ist für Lese- und Schreibeffizienz bei SSDs ähnlich wichtig wie bei HDDs.	☐	☐
Die Anzahl speicherbarer Dateien ist (zusätzlich zum verfügbaren Speicherplatz) abhängig vom Dateisystem.	☐	☐
Dateinamen werden in inodes gespeichert.	☐	☐

Lösungen

Aussage	**wahr**	**falsch**
Ein externer Host kann mittels NAT eine Verbindung zu einer privaten internen Adresse aufbauen. *Begründung: NAT funktioniert beim Aufbau der Verbindung in die andere Richtung, also von privat nach öffentlich.*		✓
Es gibt IP-Adressen, die von mehreren verschiedenen Maschinen genutzt werden. *Begründung: Private Adressen - bspw. die im Netz 10.0.0.0/8 - werden in vielen Netzen verwendet.*	✓	
6ffe:8d::f023:a349::8917 ist eine gültige IPv6 Adresse. *Begründung: Es kommt zwei mal :: vor, ist deshalb nicht eindeutig wo sich die 0000 Blöcke befinden.*		✓
Ein einzelnes TCP Paket geht nie an einem Knoten im Netzwerk verloren. *Begründung: TCP Pakete können auch verloren gehen, wird aber erkannt und werden neu gesendet.*		✓
CSS wird vom Webserver auf die HTML Seiten angewandt, bevor diese zum Client geschickt und im Browser angezeigt werden. *Begründung: CSS und HTML werden unverändert zum Client geschickt, und der Browser übernimmt dann das ganze rendering.*		✓
<tag1><tag2>Hallo</tag1></tag2>ist korrekte HTML-Syntax. *Begründung: Tags müssen richtig geschachtelt sein.*		✓
HTTP 1.0 kann als unterliegendes Protokoll zum Streamen von Video benutzt werden. *Begründung: Ein Streaming-Protokoll das über HTTP 1.0 läuft (e.g. DASH) ermöglicht das Streamen von e.g. Video. HTTP1.1+ sind zwar deutlich besser geeignet für Streaming, aber nicht zwingend notwendig.*	✓	
Disk scheduling ist für Lese- und Schreibeffizienz bei SSDs ähnlich wichtig wie bei HDDs. *Begründung: SSDs haben random access.*		✓
Die Anzahl speicherbarer Dateien ist (zusätzlich zum verfügbaren Speicherplatz) abhängig vom Dateisystem. *Begründung: Beschränkt durch die Anzahl inodes.*	✓	
Dateinamen werden in inodes gespeichert. *Begründung: Dateinamen stehen nur in Datenblöcken, die den Inhalt von Ordnern speichern. Darin stehen Einträge wie "hallo.txt : 123", was bedeutet "die Daten für Datei hallo.txt sind über inode 123 auffindbar".*		✓

2 Verschlüsseltes Testament (12 Punkte)

Bob will sein Testament T (encodiert als Bitstring) verschlüsselt hinterlassen. Er hinterlässt drei Personen drei verschiedene geheime Nachrichten N_1, N_2, N_3: seinem Notar, seiner Tochter, und seinem Geschäftspartner. Jede der drei Personen erfährt nur die ihr hinterlassene geheime Nachricht N_i. Der Einfachheit halber will Bob zum Ver- und Entschlüsseln nur die Operationen $+, -$ sowie bitweises AND, OR, NOT verwenden.

a) `[5 Punkte]` Bob will, dass jemand das Testament nur entschlüsseln kann, wenn er alle drei geheimen Nachrichten N_1, N_2, N_3 hat, und dass ohne alle drei zusammen Perfect Secrecy gewährleistet ist. Wie kann Bob die drei geheimen Nachrichten konstruieren, sodass diese Anforderungen gewährleistet sind?

b) `[7 Punkte]` Bob will zur Ausfallsicherheit das System verändern: das Testament soll nun genau dann entschlüsselbar sein, wenn man mindestens zwei beliebige geheime Nachrichten hat (egal welche zwei), und ansonsten soll Perfect Secrecy gewährleistet sein. Wie kann Bob die drei geheimen Nachrichten konstruieren, um das zu erreichen?

Lösungen

a) [5 Punkte] Bob wählt unabhängig voneinander uniform zwei Schlüssel k_1 und k_2 derselben Länge wie T aus und setzt $N_1 = k_1$, $N_2 = k_2$ und $N_3 = T \oplus k_1 \oplus k_2$. Es handelt sich um $(3, 3)$-Threshold Secret Sharing, das laut Vorlesung perfect secrecy gewährleistet, wenn nicht alle Teilnehmer kollaborieren.

b) [7 Punkte] Nun wählt Bob unabhängig voneinander uniform drei Schlüssel k_1, k_2, k_3 derselben Länge wie T aus und setzt die Nachrichten auf $N_1 = (T \oplus k_1, k_2, 1)$ und $N_2 = (T \oplus k_2, k_3, 2)$ und $N_3 = (T \oplus k_3, k_1, 3)$. Für zwei beliebige verschiedene Nachrichten $N_i = (C_i, k_i, i) \neq (C_j, k_j, j) = N_j$ gilt entweder $C_i \oplus k_j = T$ oder $C_j \oplus k_i = T$, und anhand der Indizes i und j ist klar, welches der beiden Ergebnisse stimmt. Lediglich eine Nachricht genügt zum Entschlüsseln nicht, da es sich dann um eine OTP-Verschlüsselung handelt, die perfectly secret ist.

3 SQL (23 Punkte)

In dieser Aufgabe verwenden wir die Film-Datenbank aus der Vorlesung und der Übung. Das Schema ist in Abbildung 1 dargestellt. Das Schema enthält nur die Unique Identifier und Foreign Keys für jede Tabelle. Die Namen einiger für die Aufgabe relevanter Spalten sind in Tabelle 3 aufgelistet.

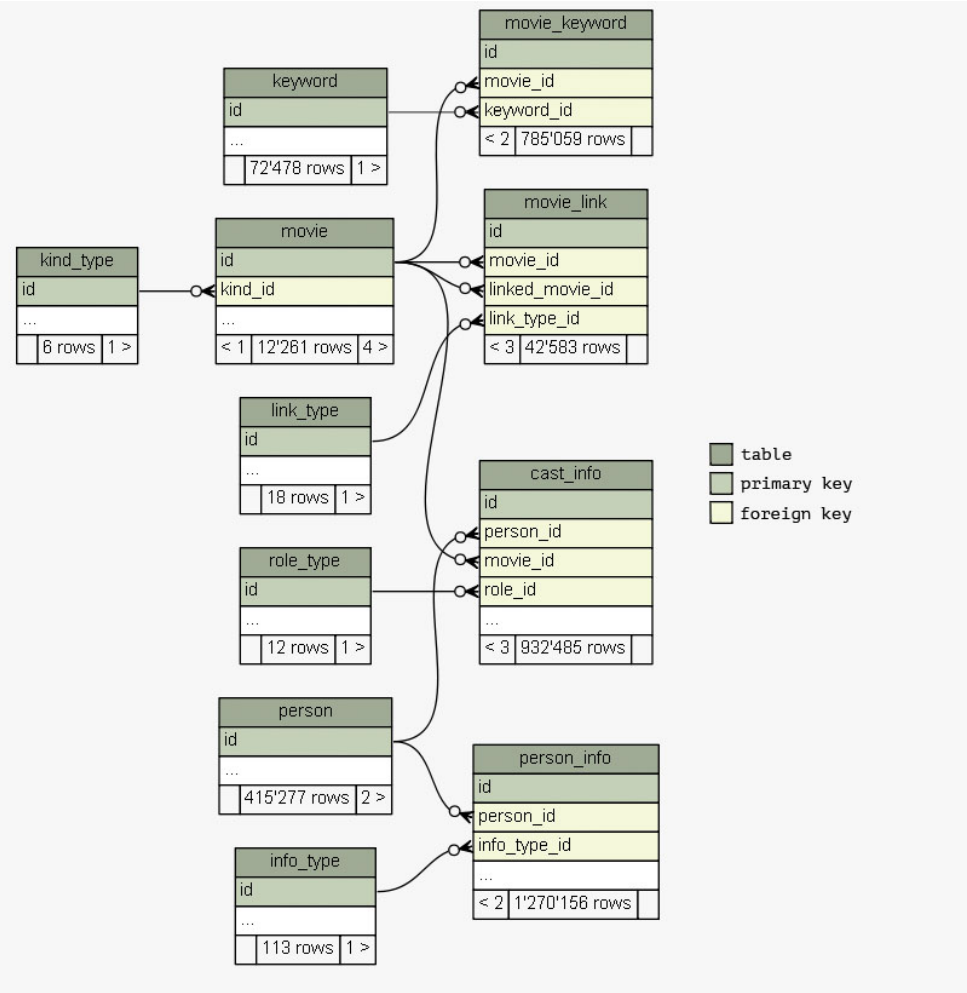


Abbildung 1: Datenbankschema welches die Tabellen mit ihren Primary und Foreign Keys zeigt.

movie
id
kind_id
title
year
tomatometer

person
id
name
gender

Tabelle 3: Relevante Spaltennamen der Datenbanktabellen *movie* und *person*.

a) Geben Sie für jeden der nachfolgenden Punkte an, welche Art von JOIN man verwenden müsste um eine korrekte und möglichst kompakte MySQL-Query zu schreiben. Sie müssen KEINE MySQL-Query schreiben.

(i) [2] Finden Sie für jeden **movie** die zugehörigen **movie_keyword**s.

(ii) [2] Geben Sie alle **person**en aus welche eine zugehörige **person_info** haben.

(iii) [2] Finden Sie heraus, wie viele **person**en kein zugehöriges **person_info** haben.

b) [4] Schreiben Sie eine MySQL-Query um die Anzahl **movie**s welche das Wort "of" im *title* enthalten auszugeben.

c) [6] Schreiben Sie eine MySQL-Query, die ausgibt wie viele männliche ("m") **person** mit Vornamen "Robert" beim **movie** "The Departed" mitgespielt haben. Sie können annehmen, dass kein Schauspieler mehr als eine Rolle spielt.

d) [4] Wir suchen eine MySQL-Query welche für jeden *tomatometer* grösser als 74 die Anzahl **movie**s mit dieser Bewertung finden soll. Das Resultat sollte absteigend nach der Anzahl gefundener **movie**s sortiert sein. Das Resultat der Query sollte also 25 Zeilen haben. Ein Freund schlägt folgende Query vor:

```
SELECT tomatometer,COUNT(title) AS cnt
FROM movie
WHERE tomatometer>74
ORDER BY cnt DESC;
```

Wenn Sie die Query laufen lassen, hat das Resultat nur eine Zeile. Wo liegt der Fehler, und wie können Sie die Query reparieren?

e) [3] MySQL kennt keine FULL OUTER JOINS. Wie könnte man mit bestehenden Befehlen dennoch einen FULL OUTER JOIN ausführen?

Lösungen

a) Joins

(i) Left Join

(ii) Join or Right Join

(iii) Left Join

```
SELECT COUNT(title)
FROM movie
WHERE title LIKE "% of %";
```

b)

```
SELECT COUNT(person.name)
FROM cast_info
JOIN movie ON movie.id = cast_info.movie_id
JOIN person ON person.id = cast_info.person_id
WHERE movie.title='The Departed' and person.gender='m'
AND person.name LIKE '%Robert%';
```

c)

d) Eve hat den "GROUP BY tomatometer" Befehl vergessen. ODER: GROUP BY und HAVING

```
SELECT tomatometer,COUNT(title) AS cnt
FROM movie
WHERE tomatometer>74
GROUP BY tomatometer
ORDER BY cnt DESC;
```

e) Die Resultate von Left und Right Outer Join kombinieren. (Eigentlich mit UNION, aber es ist nicht nötig dies explizit zu schreiben)

4 Locking & Concurrency (23 Punkte)

Als Lock für ein faires Multithread-System soll eine FIFO Queue zum Einsatz kommen, die als doppelt verlinkte Liste implementiert ist. Die Liste hat einen *head*- und einen *tail*-Block, die vor dem ersten, beziehungsweise nach dem letzten, Element der Liste stehen. Um sich in die Queue einzureihen, können Threads einen Knoten am Ende der Liste (vor dem tail) anhängen. Jeweils der Thread, von dem der vorderste Knoten (nach dem head) stammt, hat das Lock. Wenn er fertig ist, löscht er den Knoten, wodurch der nächste Knoten nachrückt, in dem er erkennt, dass er der vorderste ist.

```
 1: Struct Block {
 2:     Boolean locked
 3:     Pointer next
 4:     Pointer previous
 5: }
 6:
 7: Block head, tail
 8: head.locked = false
 9: head.next = tail
10: tail.locked = false
11: tail.previous = head
12:
13: procedure ACQUIRE_LOCK
14:     do
15:         res = TestAndSet(tail.locked)
16:     while res = true
17:
18:     Block prev = tail.previous
19:     do
20:         res = TestAndSet(prev.locked)
21:     while res = true
22:
23:     Block new_block
24:     prev.next = new_block
25:     new_block.previous = prev
26:     tail.previous = new_block
27:     new_block.next = tail
28:
29:     tail.locked = false
30:     prev.locked = false
31:     return new_block
32: end procedure
```

```
31: procedure Thread implementation
32:     mine = acquire_lock()
33:     while head.next != mine do
34:         wait 10 ms
35:     end while
36:     [Critical Section]
37:     release_lock()
38: end procedure
39:
40: procedure RELEASE_LOCK
41:     do
42:         res = TestAndSet(head.locked)
43:     while res = true
44:
45:     Block mine = head.next
46:     do
47:         res = TestAndSet(mine.locked)
48:     while res = true
49:
50:     Block next = mine.next
51:     do
52:         res = TestAndSet(next.locked)
53:     while res = true
54:
55:     head.next = next
56:     next.previous = head
57:
58:     head.locked = false
59:     mine.locked = false
60:     next.locked = false
61: end procedure
```

a) [4] Ist es besser, wenn die Threads prüfen, ob ihr Knoten auf den head der Liste folgt, oder wenn der Thread mit dem Lock jeweils den folgenden Knoten freigibt, wenn er fertig ist, indem er ein Flag in dem nachfolgenden Knoten setzt?

b) [8] Die Idee ist im gegebenen Pseudocode umgesetzt. Es wird angenommen, dass alle Reads und Writes atomar sind. Wieso ist die Implementierung nicht korrekt? Mit welcher Strategie kann das Problem gelöst werden?

c) [8] Ist der gegebene Pseudocode korrekt wenn die Liste immer mehr als 100 Blöcke enthält? Zeigen Sie wieso oder geben Sie ein Gegenbeispiel.

d) [3] Welches Performanceproblem kann im gegebenen Pseudocode auftreten?

Lösungen

a) Die zweite Variante (wie MCS Lock) ist besser. Die erste invalidiert die Caches von allen Prozessoren, immer wenn der HEAD geändert wird. Das braucht hohe Bandbreite vom gemeinsamen Memory Bus.

b) In der Implementierung können Deadlocks entstehen: Release locks from front, acquire from back. If queue contains only one block and both operations are issued concurrently, a deadlock can happen when the release thread locks head and the only block and the acquire thread locks the tail and then both wait for each other to release a lock.

Solution: Acquire locks in the same order.
Correct code (not required to get points):

```
1: procedure WAIT_FOR_LOCK
2:     prev = tail.previous
3:     lock(prev)
4:     lock(tail)
5:     if tail.previous != prev then repeat from beginning
6:     end if
7:     [critical section]
8: end procedure
```

c) Die Lösung ist korrekt: Bei beiden Methoden kann immer nur ein Thread tail.locked, bzw. head.locked auf true setzen und false als Rückgabewert kriegen. Deshalb müssen dann alle anderen Threads warten, bis tail, bzw. head, wieder "frei" sind. Überholen innerhalb einer Methode ist also nicht möglich. Zwischen den beiden Methoden ist auch keine Beeinflussung möglich, da immer die Blöcke vor und hinter der zu bearbeitenden Position "gesperrt" werden durch das setzen der locked Flags.

d) Das Warten in der Thread-Implementierung ist das Problem. Der Thread, der an der Reihe ist, kann noch bis zu (knapp unter) 10 ms warten, bis er den kritischen Abschnitt ausführt. Währenddessen macht kein einziger Thread Fortschritt.

5 Markovketten (22 Punkte)

Betrachten wir einen Kassierer bei einem Supermarkt mit zwei Gemütszuständen: gutgelaunt oder launisch. Kunden sind mit einer Wahrscheinlichkeit von 2/3 freundlich, ansonsten unfreundlich. Der Kassierer ist zu Arbeitsbeginn gutgelaunt, aber jedes Mal, wenn er direkt nacheinander drei unfreundliche Kunden bedient, wird er launisch. Wenn er launisch ist und einen freundlichen Kunden bedient, dann wird er mit einer Wahrscheinlichkeit von 1/4 wieder gutgelaunt, sonst bleibt er launisch. Wir nehmen an, dass am Ende des Arbeitstages die stationäre Verteilung erreicht ist.

a) [5] Zeichnen Sie den Prozess als Markovkette!

b) [3] Ist die Markovkette ergodisch?

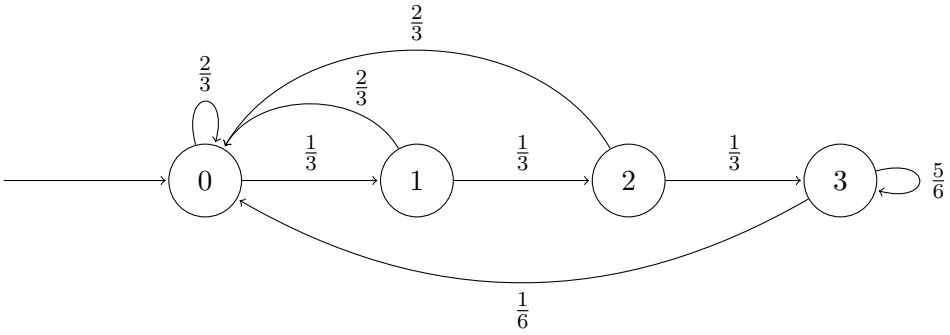
c) [3] Wie viele Kunden liegen in Erwartung zwischen einem Kunden, der den Kassierer launisch stimmt, und dem nächsten Kunden, der ihn wieder gutgelaunt stimmt?

d) [8] Wie hoch ist die Wahrscheinlichkeit, dass der Kassierer am Ende des Arbeitstages gutgelaunt nach Hause geht?

e) [3] Nehmen wir nun Folgendes an: Wenn der Kassierer schon launisch ist und ein weiterer unfreundlicher Kunde kommt, dann wird der Kassierer wütend und bleibt bis zum Ende des Arbeitstages wütend. Wie hoch ist nun die Wahrscheinlichkeit, dass der Kassierer gutgelaunt nach Hause geht?

Lösungen

a) [5]



Jeder Zustand ist beschriftet mit der Anzahl an unfreundlichen Kunden, die der Kassierer zuletzt gesehen hat, ohne einen freundlichen Kunden gesehen zu haben.

b) [3] Jeder Zustand ist von jedem anderen Zustand erreichbar, also ist die Markovkette irreduzibel. Wegen der Loops bei Zuständen 0 und 3 ist die Markovkette auch aperiodisch. Weil sie sowohl irreduzibel als auch aperiodisch ist, ist sie also ergodisch.

c) [3] Wir verlassen den Zustand 3 mit einer Wahrscheinlichkeit von $\frac{1}{6}$, in Erwartung verlassen wir ihn also in der $\frac{1}{\frac{1}{6}} = 6$ten Runden. Wir verweilen also in Erwartung für $6-1=5$ Runden in Zustand 3.

d) [8] Die Übergangsmatrix der Markovkette ist

$$P = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{3} & 0 \\ \frac{2}{3} & 0 & 0 & \frac{1}{3} \\ \frac{1}{6} & 0 & 0 & \frac{5}{6} \end{pmatrix}$$

Um die stationäre Verteilung zu finden lösen wir das Gleichungssystem:

$$\begin{aligned} (x_0, x_1, x_2, x_3) &= (x_0, x_1, x_2, x_3) \cdot P \\ 1 &= x_0 + x_1 + x_2 + x_3 \end{aligned}$$

Wir erhalten: $x_0 = \frac{3}{5}$, $x_1 = \frac{1}{5}$, $x_2 = \frac{1}{15}$, $x_3 = \frac{2}{15}$. der Kassierer ist gutgelaunt in allen Zuständen ausser 3, die Wahrscheinlichkeit, dass er am Ende gutgelaunt ist, ist also $1 - \frac{2}{15} = \frac{13}{15}$.

e) [3] Die Kette wird um einen Zustand "wütend" erweitert, der eine eingehende Kante von Zustand 3 hat und über diese von allen anderen Zuständen aus erreichbar ist; "wütend" hat nur eine ausgehende Kante, die zu sich selbst führt. In der stationären Verteilung hat dieser Zustand eine Wahrscheinlichkeit von 1.