ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2018 Prof. R. Wattenhofer

Technische Informatik II

Klausur

Mittwoch, 23. Januar 2019, 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. **Begründen Sie alle Ihre Antworten** sofern nichts anderes steht. Beschriften Sie Skizzen und Zeichnungen verständlich. **Was wir nicht lesen können gibt keine Punkte!**

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.

Familienname	**Vorname**	**Legi-Nr.**

Aufgabe	**Erreichte Punktzahl**	**Maximale Punktzahl**
1 - Multiple Choice		9
2 - Transport Layer: LPs		15
3 - Storage		20
4 - Queue-Counter-Lock		20
5 - Markov Chains		26
Summe		**90**

1 Multiple Choice (9 Punkte)

Geben Sie bei jeder Aussage an, ob sie wahr oder falsch ist. Jede korrekte Antwort gibt 1 Punkt. Jede fehlerhafte Antwort und unbeantwortete Aussage gibt 0 Punkte. **In dieser Aufgabe können Sie Ihre Antworten nicht begründen oder erklären.**

a) `[3 Punkte]` Auf dem Netzwerk-Layer und auf dem Link-Layer werden unterschiedliche Adressen verwendet.

	wahr	**falsch**
IP-Adressen werden benötigt, da die MAC-Adressen nicht global einmalig zugewiesen werden.	☐	☐
Ein Netzwerk-Gerät kann gleichzeitig eine IPv4 und eine IPv6 Adresse haben.	☐	☐
Beim Zugriff auf eine Webseite über deren Domain-Namen wird der Domain-Name im IP-Paket anstatt der IP-Adresse angegeben.	☐	☐

b) `[3 Punkte]` Wir betrachten das Transmission Control Protocol (TCP):

	wahr	**falsch**
Congestion-Control funktioniert bei TCP trotz bösartiger Teilnehmer.	☐	☐
TCP beendet automatisch eine Verbindung, die eine Minute lang unbenutzt ist.	☐	☐
TCP verwendet die RTT um zu bestimmen, wie lange ein Sender auf das ACK wartet, bevor der Sender erneut sendet.	☐	☐

c) `[3 Punkte]` Im Skript ist beschrieben wie Prozesse und Threads miteinander kommunizieren können.

	wahr	**falsch**
Zwischen mehreren Prozessen, welche mittels Remote Procedure Calls kommunizieren, kann es nie zu Deadlocks kommen.	☐	☐
Ein Betriebssystem mit non-preemptive Scheduling läuft auf einem Einkernprozessor. In diesem Fall könnte man komplett auf Locks verzichten.	☐	☐
Abgesehen von Performance, könnte man jedes Programm, welches in mehreren Threads ausgeführt wird, so umschreiben, dass es stattdessen in mehreren Prozessen läuft.	☐	☐

Lösungen

a) `[3 Punkte]` Auf dem Netzwerk-Layer und auf dem Link-Layer werden unterschiedliche Adressen verwendet.

		wahr	**falsch**
	IP-Adressen werden benötigt, da die MAC-Adressen nicht global einmalig zugewiesen werden. *Begründung: MAC-Adressen werden einmalig zugewiesen (ausser man ändert sie). Routing nicht möglich, mit MAC-Adressen, da jeder Router den Weg zu jeder Adresse speichern müsste. IPs werden hierarchisch zugewiesen.*		✓
	Ein Netzwerk-Gerät kann gleichzeitig eine IPv4 und eine IPv6 Adresse haben. *Begründung: Stimmt.*	✓	
	Beim Zugriff auf eine Webseite über deren Domain-Namen wird der Domain-Name im IP-Paket anstatt der IP-Adresse angegeben. *Begründung: Falsch. Wird über DNS erst zu IP übersetzt.*		✓

b) `[3 Punkte]` Wir betrachten das Transmission Control Protocol (TCP):

		wahr	**falsch**
	Congestion-Control funktioniert bei TCP trotz bösartiger Teilnehmer. *Begründung: Ein bösartiger Teilnehmer kann bei Paketverlust sein Congestion Window gleich gross lassen (AIMD nicht befolgen), gleichzeitig verkleinern die anderen Teilnehmer ihr Congestion Window. Dadurch ist die Zuteilung der Raten nicht mehr fair.*		✓
	TCP beendet automatisch eine Verbindung, die eine Minute lang unbenutzt ist. *Begründung: Eine TCP-Verbindung bleibt offen bis sie beendet wird.*		✓
	TCP verwendet die RTT um zu bestimmen, wie lange ein Sender auf das ACK wartet, bevor der Sender erneut sendet. *Begründung: Die smoothed RTT (anfangs RTT) wird verwendet um zu entscheiden wie lange der Sender auf das ACK wartet.*	✓	

c) `[3 Punkte]` Im Skript ist beschrieben wie Prozesse und Threads miteinander kommunizieren können.

	wahr	**falsch**
Zwischen mehreren Prozessen, welche mittels Remote Procedure Calls kommunizieren, kann es nie zu Deadlocks kommen. *Begründung: Prozess 1 ruft Prozess 2 per RPC auf und Prozess 2 ruft dann Prozess 1 per RPC auf. Deadlock.*		✓
Ein Betriebssystem mit non-preemptive Scheduling läuft auf einem Einkernprozessor. In diesem Fall könnte man komplett auf Locks verzichten. *Begründung: Jeder Thread wird nur dann geblockt wenn er selber yieldet, das heisst wenn die Threads nie in Critical Sections yielden und wenn sie immer alle Resourcen freigeben, braucht es keine locks für eine korrekte Programmausführung.*	✓	
Abgesehen von Performance, könnte man jedes Programm, welches in mehreren Threads ausgeführt wird, so umschreiben, dass es stattdessen in mehreren Prozessen läuft. *Begründung: Threads können durch Prozesse ersetzt werden, es ist einfach aufwändiger da man Inter-Prozess Kommunikation implementieren muss.*	✓	

2 Transport Layer: LPs (15 Punkte)

a) `[7 Punkte]` Wir betrachten das folgende LP:

Maximiere: $f(x) = 7x_1 - 12x_2$ unter den Bedingungen

1: $x_1 \geq 0$
2: $x_1 \leq 4$
3: $x_2 \geq 0$
4: $x_2 \leq 3$
5: $x_1 - x_2 \leq 3$

Führen Sie den Simplex Algorithmus beginnend bei $(0, 0)$ durch und beantworten Sie folgende Fragen:

(i) Was sind die Eckpunkte des Polytops?

(ii) Was ist die Lösung des LPs?

Hinweis: *Eine aussagekräftig beschriftete Skizze ist als Lösung zugelassen.*

b) `[8 Punkte]` Wir betrachten das folgende LP:

Maximiere: $f(x) = 3x_1 - x_2 + x_3$ unter den Bedingungen

1: $x_1 \geq 1$
2: $-x_1 \geq -5$
3: $2x_2 \geq 0$
4: $\frac{1}{3}x_2 \leq 1$
5: $x_3 \geq 1$
6: $2x_3 \leq 2$
7: $x_1 - 1 \geq 2x_2$
8: $5 - x_1 + x_2 - 2x_3 \geq 0$

Berechnen Sie die Lösung des LPs.

Hinweis: *Der Punkt* $(1, 0, 1)$ *ist ein Eckpunkt des Polytops.*

Lösungen

a) Die Eckpunkte des Polytops sind:

- $x = (0, 0)$,
- $x = (3, 0)$,
- $x = (4, 1)$,
- $x = (4, 3)$,
- $x = (0, 3)$.

Der Simplex-Algorithmus findet das Ergebnis in einem Schritt: $(0, 0) \to (3, 0)$.

b) Zunächst beobachten wir, dass:

- $-x_1 \geq -5$ äquivalent zu $x_1 \leq 5$ ist,
- $2x_2 \geq 0$ äquivalent zu $x_2 \geq 0$ ist,
- $\frac{1}{3}x_2 \leq 1$ äquivalent zu $x_2 \leq 3$ ist,
- $x_3 \geq 1$ und $2x_3 \leq 2$ impliziert, dass $x_3 = 1$ gilt,
- $x_1 - 1 \geq 2x_2$ äquivalent zu $x_1 - 2x_2 \geq 1$ ist,
- und $5 - x_1 + x_2 - 2x_3 \geq 0$ nun äquivalent zu $x_1 - x_2 \leq 3$ ist.

Wir erhalten das LP:
Maximiere: $f(x) = 3x_1 - x_2 + 1$ unter den Bedingungen

1: $x_1 \geq 1$
2: $x_1 \leq 5$
3: $x_2 \geq 0$
4: $x_2 \leq 3$
5: $x_1 - 2x_2 \geq 1$
6: $x_1 - x_2 \leq 3$

welches analog zur ersten Teilaufgabe gelöst werden kann. Die Eckpunkte sind:

- $x = (1, 0)$,
- $x = (3, 0)$,
- $x = (5, 2)$.

Der Simplex-Algorithmus beginnend bei $(1, 0)$ findet das Ergebnis direkt: $(1, 0) \to (5, 2)$. Das Ergebnis des originalen LPs ist somit $x = (5, 2, 1)$.

3 Storage (20 Punkte)

Sie schreiben ein Programm, um den verfügbaren Platz auf einem auf Inodes basierenden Dateisystem zu berechnen. Dazu startet das Programm im Root-Verzeichnis und geht rekursiv durch alle Verzeichnisse. Das Programm summiert die Grösse aller gefundenen Dateien. Dieser Wert wird dann von der Grösse der Data-Region abgezogen.

a) `[6 Punkte]` Weshalb stimmt der berechnete freie Speicherplatz nicht mit dem tatsächlich freien Platz in der Data-Region überein? Geben Sie zwei Gründe an.

b) `[4 Punkte]` Geben Sie jeweils eine Lösungsmöglichkeit an, um die gefundenen Probleme zu beheben. (Falls Sie nur ein Problem gefunden haben können Sie auch zwei Lösungsmöglichkeiten für dieses angeben.)

Eine Kollegin erstellt zum Spass ein Programm, das in Ihrem Dateisystem viele neue Inodes erstellt, ohne Verzeichniseinträge (directory entries) auf diese zu erstellen.

c) `[5 Punkte]` Weshalb ist nach dem Ausführen dieses Programms das Dateisystem möglicherweise nicht mehr korrekt nutzbar (Dateien und Verzeichnisse lesen, erstellen, ändern, löschen)?

d) `[5 Punkte]` Wie können Sie herausfinden, ob dieses Programm auf Ihrem Dateisystem ausgeführt wurde, und wie können Sie den ursprünglichen Zustand des Dateisystems wiederherstellen?

Lösungen

a) Mehrere Gründe:

- Datenblöcke nur teilweise durch Dateien gefüllt
- Platzverbrauch der Verzeichnisse nicht mitgezählt
- Platzverbrauch der Symlinks nicht mitgezählt
- Platzverbrauch der Datenblöcke mit indirect Pointern
- Hardlinks mehrfach gezählt

b)

- Datenblöcke nur teilweise durch Dateien gefüllt: Datenblöcke pro Inode zählen oder totale Belegung aus Daten-Bitmap bestimmen.
- Platzverbrauch der Verzeichnisse/Symlinks nicht mitgezählt: Bei Verzeichnissen zählen wie viel Platz durch Dateiliste belegt ist, oder totale Belegung aus Daten-Bitmap bestimmen.
- Platzverbrauch der Datenblöcke mit indirect Pointern: Zählen, wie viele Datenblöcke in Inode für die indirect Pointern verbraucht.
- Hardlinks mehrfach gezählt: $(reference\_counter - 1) * file\_size$ von totaler Grösse abziehen oder mit Liste der gesehenen Inodes sicherstellen dass jeder Inode nur einmal gezählt wird oder totale Belegung aus Daten-Bitmap bestimmen.

c) Wenn alle Inodes belegt sind können keine neuen Dateien mehr erstellt werden. Oder es könnte auch die Daten-Region vollständig belegt sein.

d) Alle Verzeichnisse durchgehen und Inode-Bitmap mit den referenzierten Inodes machen. Diese vergleichen mit der Inode-Bitmap des Dateisystems. Falls sie nicht übereinstimmen wurde das Programm ausgeführt. Effekt kann rückgängig gemacht werden durch Löschen der zusätzlichen Inodes, also auch der gebrauchten Datenblöcke. Oder es können alle Inodes durchgegangen werden und diejenigen Löschen, bei denen der Referenz-Counter 0 ist.

4 Queue-Counter-Lock (20 Punkte)

Sie entwerfen ein Programm, das den Zugriff von mehreren Threads auf ein lese- und schreibbares Dokument kontrollieren soll. Sie basieren dabei ihre Implementierung auf zwei Locks, welche Sie aus der Vorlesung/Übung kennen: ein MCS Queue Lock und ein Counter Lock. Das Counter Lock gibt Ihnen folgende mit CompareAndSwap implementierten Funktionen:

- `write_lock`: wartet darauf, dass der AtomicInteger 0 ist und setzt ihn dann auf -1
- `read_lock`: wartet darauf, dass der AtomicInteger ≥ 0 ist und erhöht ihn dann um 1
- `write_unlock`: setzt den AtomicInteger zurück auf 0
- `read_unlock`: liest den Wert des AtomicIntegers und probiert ihn um eins reduziert wieder zu schreiben. Wiederholt dies falls CompareAndSwap misslingt.

Sie können davon ausgehen, dass das MSC Queue Lock und das Counter Lock korrekt implementiert sind. Sie schlagen nun folgende Implementation für ein Queue-Counter-Lock vor:

Algorithm 1 Queue-Counter-Lock

```
 1:  MCSQueueLock queueLock = new MCSQueueLock()
 2:  CounterLock counterLock = new CounterLock()
 3:
 4:  void write_lock() {
 5:     queueLock.lock();
 6:     counterLock.write_lock();
 7:     queueLock.unlock();
 8: }
 9:  void read_lock() {
10:     queueLock.lock();
11:     counterLock.read_lock();
12:     queueLock.unlock();
13: }
14:  void write_unlock() {
15:     counterLock.write_unlock();
16: }
17:  void read_unlock() {
18:     counterLock.read_unlock();
19: }
```

a) [6 Punkte] Gegeben sei der Anfangszustand, in welchem der AtomicInteger des Counter Locks 0 ist und die Queue des Queue Locks leer ist. Threads A, B, C und D rufen in der folgenden Reihenfolge die Funktionen des Queue-Counter-Locks auf:

- A: `read_lock()`
- B: `read_lock()`
- C: `write_lock()`
- D: `read_lock()`

Geben Sie für jeden Thread an, ob er auf ein Lock wartet, und wenn ja, auf welches.

b) [4 Punkte] Ist das Queue-Counter-Lock First-Come-First-Served Fair?

c) [6 Punkte] Ein Kollege von Ihnen schlägt vor, Zeilen 11 und 12 im Code oben auszutauschen, damit Lese-Prozesse schneller Zugriff auf das Dokument haben. Warum ist dies eine schlechte Idee?

d) [4 Punkte] Threads E und F führen folgende Befehle in dieser Reihenfolge aus:

- E: `read_lock()`
- F: `read_lock()`
- E: `read_unlock()`
- E: `read_unlock()`
- F: `read_unlock()`

Was hat dies zur Folge?

Lösungen

a) Thread A und B warten auf kein Lock (sind am Lesen), Thread C wartet auf das Counter Lock (darauf, dass A und B mit Lesen fertig sind) und Thread D wartet auf das Queue Lock (welches von C gehalten wird).

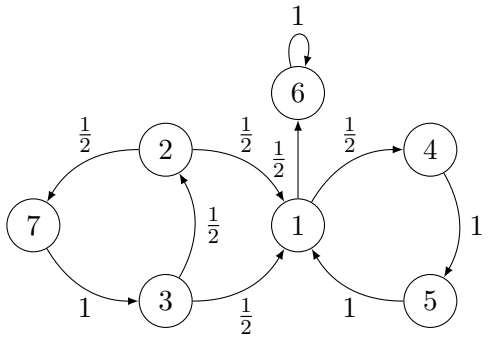
b) First-come-first-served Fairness ist durch das Queue Lock gegeben, da dieses die Lock-Anfragen auf das Counter Lock ordnet.

c)
- Hauptgrund: Der nächste Schreib-Thread könnte schneller das Counter Lock erhalten als die Lese-Threads, welche vor ihm in der Queue waren. Somit wäre First-come-first-served Fairness nicht mehr gegeben und die Lese-Threads lesen je nach dem etwas anderes als sie sollten.
- Nebengrund: Die Counter Lock Anfragen der Lese-Threads kommen nicht mehr einer nach dem anderen, was zu Contention führen kann.

d) Die falsche Verwendung des Locks durch Thread E (zwei unlocks) führt dazu, dass der Atomic Integer auf -1 gesetzt wird, obwohl kein Thread am Schreiben ist. Dies blockiert alle zukünftigen Anfragen auf das Dokument.

5 Markov Chains (26 Punkte)

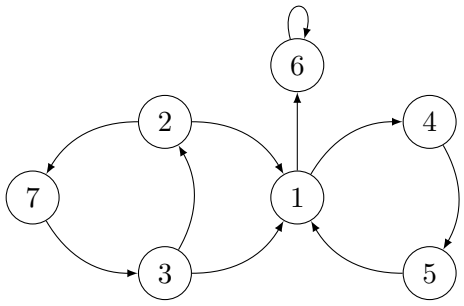
Gegeben ist folgende Markovkette:



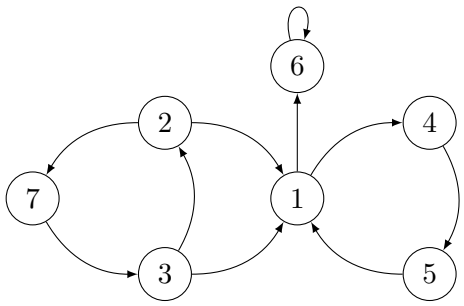
a) [2 Punkte] Ist die Markovkette aperiodic?

b) [2 Punkte] Ist die Markovkette irreducible?

c) [3 Punkte] Machen Sie die Markovkette periodic/aperiodic indem Sie die minimale Anzahl Kanten löschen/hinzufügen. Benutzen Sie dazu das folgende Bild:



d) [3 Punkte] Machen Sie die Markovkette reducible/irreducible indem Sie die minimale Anzahl Kanten löschen/hinzufügen. Benutzen Sie dazu das folgende Bild:

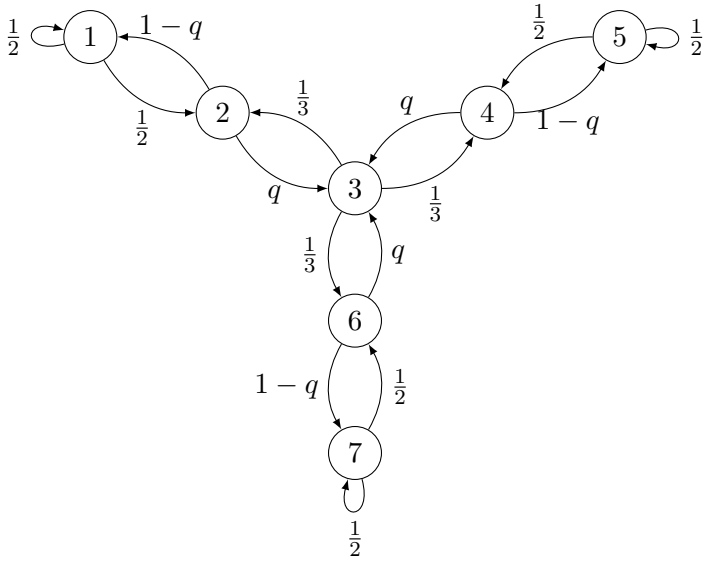


Gegeben ist folgende Markovkette:

e) [3 Punkte] Für welche Werte von q ist die Markovkette ergodisch?

f) [6 Punkte] Es gilt $q = 1/2$. Wenn man im State 3 startet, wie lange dauert es im Erwartungswert bis man das erste mal wieder im State 3 landet?

g) [7 Punkte] Es gilt $q = 0$. Berechnen Sie die stationäre(n) Verteilung(en).


1
2
3
4
5
6
7
$\frac{1}{2}$
$1-q$
$\frac{1}{2}$
$\frac{1}{3}$
q
q
$\frac{1}{3}$
$\frac{1}{2}$
$\frac{1}{2}$
$1-q$
$\frac{1}{3}$
q
$1-q$
$\frac{1}{2}$
$\frac{1}{2}$
$\frac{1}{2}$

Lösungen

a) Nein. Der linke und rechte Cycle sind periodisch und deshalb ist die ganze kette periodisch.

b) Nein. State 6 ist absorbing.

c) Verbindung von 5 nach 3.

d) Verbindung von State 6 zu, e.g., State 2.

e) Für $q = 0$ und $q = 1$ ist die Markovkette nicht irreducible und damit nicht ergodisch.

f) Da die Kette symmetrisch ist muss man nicht ein 7x7 GLS auflösen, sondern nur ein 3x3 indem man z.B. nur den Teil (3,4,5) anschaut und entsprechend die Kantengewichte von State 3 anpasst. Intuitiv kann man das so sehen: Egal in welchen der drei Branches man geht, es dauert im Erwartungswert immer gleich lange bis man wieder zurück bei 3 ist. Man hat also folgendes Gleichungssystem um die Expected Return Time r_3 zu berechnen:

$$r_3 = 1 + h_{43} \tag{1}$$

$$h_{43} = 1 + \frac{1}{2}h_{33} + \frac{1}{2}h_{54} \tag{2}$$

$$h_{54} = 1 + \frac{1}{2}h_{43} + \frac{1}{2}h_{54} \tag{3}$$

Zuerst muss man erkennen, dass $h_{33} = 0$, da man dann ja bereits zurück bei 3 ist. Man kann das GLS jetzt auflösen indem man (2) in (3) einsetzt, wodurch man $h_{54} = 6$ erhält. Einsetzen in (2) gibt $h_{43} = 4$. Einsetzen in (1) ergibt dann $r_3 = 5$.

g) Da die Kette nicht irreducible ist gibt es unendlich viele stationäre Verteilungen. Entweder durch auflösen von $\pi^T * P = \pi$ oder durch symmetrie findet man $\pi^* = (\frac{2}{3}\alpha, \frac{1}{3}\alpha, 0, \frac{1}{3}\beta, \frac{2}{3}\beta, \frac{1}{3}\gamma, \frac{2}{3}\gamma)$ wobei α, β, γ frei gewählt werden können so dass $\alpha + \beta + \gamma = 1$. Falls man $\pi^T * P = \pi$ auflöst kann man auch wieder nur einen Branch anschauen, z.B., wieder nur Knoten (1,2,3) und die Kantengewichte von State 3 entsprechend anpassen. Das GLS ist dann:

$$\frac{1}{2}\pi_1 + \pi_2 = \pi_1 \tag{4}$$

$$\frac{1}{2} + \pi_3 = \pi_2 \tag{5}$$

$$\pi_3 = 0 \tag{6}$$

$$\pi_1 + \pi_2 + \pi_3 = 1 \tag{7}$$

Auflösen ergibt trivialerweise $\pi_3 = 0$. Danach haben wir $\pi_1 = 2 * \pi_2$ und $\pi_1 + \pi_2 = 1$, was $\pi_1 = \frac{2}{3}$ und $\pi_2 = \frac{1}{3}$ ergibt. Eine mögliche stationäre Verteilung wäre also $\pi^* = (\frac{2}{3}, \frac{1}{3}, 0, 0, 0, 0, 0)$. Da die Kette nicht irreducible ist wissen wir aber, dass sie unendlich viele stationäre Verteilungen hat (Siehe Figure 12.12 im Skript). Wir können die Wahrscheinlichkeiten also beliebig auf alle States verteilen, solange die gegebenen Bedingungen eingehalten sind. Um das formal aufzuschreiben führen wir 3 freie Parameter ein (siehe oben).