Prüfungsprotokoll Praktische Informatik

Ewgenij Sokolovski

6. Dezember 2006

Prüfungsfächer:

1. Betriebssysteme I und II (Buch Silberschatz 7. Auflage)
2. Implementation of Databases (Vorlesung Jarke)
3. Introduction to Database Systems (Vorlesung Jarke)

Ich bedanke mich sehr bei all denen, die ihre Prüfungsprotokolle ins Netz gestellt oder diese bei der Fachschaft abgegeben haben. Diese Unterlagen haben mir SEHR bei der Vorbereitung zu meiner eigenen Prüfung geholfen!

Zusammenfassung

Das ist ein Protokoll über meine Diplomprüfung im Fach Praktische Informatik, die am 06.12.06 von 9:33 bis etwa 10:27 stattgefunden hat. Der Prüfer war Professor Matthias Jarke. Die Prüfung wurde mit der Note 2.0 bewertet. Die Dauer der Prüfung betrug ungefähr 55 Minuten.

1 Betriebssysteme

- **Jarke:** Wir beginnen mit den Betriebssystemen. Wie sie wissen, bestehen die Computersysteme aus verschiedenen Geräten, die jeweils verschiedene Funktionen haben.
- **Ich:** Die Ressourcen.
- **Jarke:** Ja, die Ressourcen. Es gibt ja verschiedene Arten von Ressourcen...

- **Ich:** (verstehe nicht wirklich, worauf er hinaus will) Also es gibt die Eingabe-/Ausgabegeräte, den Prozessor, den Speicher.
- **Jarke:** (nicht wirklich zufrieden mit der Antwort) Hmm, hmm, ja so könnte man das auch formulieren, dann kann man natürlich diese Kategorien noch weiter aufteilen...
- **Ich:** Soll ich jetzt einzelne Geräte nennen?
- **Jarke:** Nee, das brauchen wir nicht. Ich wollte eigentlich auf etwas Anderes hinaus, es gibt verschiedene Arten von Ressourcen in Bezug auf ihre Verwendung. Wir haben ja mehrere Prozesse, die jeweils Ressourcen beanspruchen.
- **Ich:** Ah, Sie meinen jetzt die Ressourcen, die gleichzeitig von mehreren Prozessen benutzt werden können und die, die nur von einem belegt werden dürfen.
- **Jarke:** Ja, nennen Sie doch ein Beispiel.
- **Ich:** Also ein Drucker kann zum Beispiel nur von einem Prozess gleichzeitig betrieben werden. Und der Prozessor wird gleichzeitig von mehreren Prozessen benutzt.
- **Jarke:** Na, das mit dem Prozessor nicht so wirklich...
- **Ich:** Ja, OK, ich meine durch Multitasking wechseln sich die Prozesse sehr schnell ab, aber in jedem einzelnen Moment wird nur ein Prozess vom Prozessor ausgeführt. Hmm, hmm, dann wäre der Speicher ein Beispiel für eine Ressource, die gleichzeitig von mehreren Prozessen benutzt werden kann. Im Falle von nur lesenden Zugriffen, kann derselbe Speicherbereich von mehreren Prozessen gleichzeitig genutzt werden, da es dann keine Gefahr von inkonsistenten Zuständen besteht.
- **Jarke:** OK. (Er fängt an, darüber zu erzählen, dass wenn man viele Geräte hat, und viele Prozesse sie belegen möchten, dann müssen sie sich bei jedem Gerät anstellen, und die Schlangen müssen verwaltet werden und es gibt ja viele Möglichkeiten, wie man sie verwaltet usw. Dabei verstehe ich wieder nicht wirklich, was er jetzt von mir hören will)
- **Ich:** Wollen Sie auf die Schedulingverfahren für die Geräte hinaus? Also es gibt die FIFO,...

- **Jarke:** Nein, ich wollte was Anderes. Also wenn es Ressourcen gibt, die nur von einem einzigen Prozess gleichzeitig genutzt werden können, dann...
- **Ich:** Dann kann es zu Deadlocks kommen. Es gibt ja die vier Bedingungen, die Prozesse müssen in einer Kette jeweils auf Ressourcen warten, die von einem anderen Prozess belegt ist. Es gibt die 4 Bedingungen...
- **Jarke:** (Unterbricht. Die genaue Definition von Deadlock bzw. von der vier dafür notwendigen Bedingungen will er nicht hören) Ja, OK. Und bei den um Betriebsmittel konkurrierenden Prozessen gibt es noch die Problematik, wann welcher Prozess Zugang bekommt. Da gibt es viele Algorithmen, die es bewerkstelligen.
- **Ich:** (hoffe, dass ich seine Richtung richtig erraten hab) Sie meinen die Synchronisationsproblematik, die kritischen Bereiche.
- **Jarke:** Ja, die meine ich.
- **Ich:** Hier gibt es mehrere Methoden. Es gibt Semaphoren, Monitore, Bakery Algorithmus.
- **Jarke:** Erläutern Sie mal diese.
- **Ich:** Welche jetzt davon?
- **Jarke:** Ja alle, fangen Sie, sagen wir, mit den Semaphoren an. Wir haben auch eine konkrete Anwendung dafür, die Producer-Consumer Problematik.
- **Ich:** Zuerst erkläre ich die Semaphoren. (Ich erkläre die Variable S und schreibe die *wait* und *signal* Prozeduren auf) Und in der Anwendung benutzt man die Semaphoren wie folgt:

  Prozess$_i${
      wait(S);
      Kritischer Bereich;
      signal(S);
  }

  (Habe dann auch erklärt, warum jeweils nur einer der Prozesse in den kritischen Bereich gelangen kann.)

- **Jarke:** Kommen wir dann auf unser Beispiel mit Producer und Consumer. Ist es da so einfach?

- **Ich:** Ähhhh (überleg, überleg), nein, da ist es nicht so einfach! Bei Producer-Consumer geht es ja nicht nur darum, dass nur einer der Prozesse in den kritischen Bereich darf. Hier kommt noch das zusätzliche Problem, dass der Producer nur dann in den Puffer schreiben darf, wenn ein Platz frei ist. Und der Consumer darf nur dann auf den Puffer zugreifen, wenn dieser auch mindestens ein Objekt enthält. Deswegen muss man noch zwei zusätzliche Semaphore einführen: *empty* und $full$. *empty* zeigt an, wie viele Pufferplätze frei, und $full$ - wie viele Pufferplätze belegt sind. Jedes Mal, wenn der Producer ein Element ablegt, dekrementiert er *empty* und inkrementiert $full$. Jedes Mal, wenn der Consumer ein Element entnimmt, dekrementiert er $full$ und inkrementiert *empty*. Das alles sieht dann so aus:

  ```
  Producer{
     wait(empty);
     Element ablegen;
     signal(full);
  }
  ```

  ```
  Consumer{
     wait(full);
     Element entnehmen;
     signal(empty)
  }
  ```

  (Habe dann noch etwas ausführlicher die Funktionsweise von *Producer* und *Consumer* mit Semaphoren erklärt)

- **Jarke:** (guckt sich meine Algorithmen an) Ist das denn alles? Haben Sie alles berücksichtigt?

- **Ich:** Hmm... Ach ja, ich habe noch den exklusiven Zugang zum Puffer vergessen. Es muss sichergestellt werden, dass in einem Zeitpunkt nur ein Prozess: *Producer* oder *Consumer* auf den Puffer zugreift. Dafür verwendet man noch einen Semaphor S. Dann sieht es so aus:

  ```
  Producer{
     wait(S);
     wait(empty);
     Element ablegen;
     signal(full);
     signal(S);
  }
  ```

  ```
  Consumer{
     wait(S);
     wait(full);
     Element entnehmen;
     signal(empty)
     signal(S);
  }
  ```

- **Jarke:** Warum haben Sie in beiden Prozessen *wait(S)* vor *wait(empty)* bzw. *wait(full)* gesetzt?

- **Ich:** Weil ich zuerst prüfen muss, ob überhaupt auf den Puffer zugegriffen werden kann, bevor ich die Zähler der leeren bzw. vollen Plätze verändere. Wenn ich das anders herum machen würde, also z.B. zuerst *wait(empty)* und erst dann *wait(S)* würde ich direkt den Zähler der leeren Plätze dekrementieren. Wäre danach der Prozess wegen *wait(S)* daran verhindert, ein Objekt in dem Puffer abzulegen, wäre das System im inkonsistenten Zustand. (Im nachhinein war es mir klar, dass es nicht richtig war und dass die Reihenfolge von *wait*s vertauscht werden sollte - siehe Silberschatz. Aber in dem Moment fiel mir es nicht ein:)))

- **Jarke:** Ja, man könnte so argumentieren. Sagen Sie können hier Deadlocks auftreten?

- **Ich:** Ja.

- **Jarke:** Dann geben Sie mal ein Beispiel von so einem Deadlock.

- **Ich:** (hab versucht, mir so eine Situation auszudenken, kam aber nicht drauf, wie es aussehen könnte. Sagte dann nur, dass es dann dazu kommen muss, dass der eine Prozess auf den anderen wartet und der, seinerseits, auf den ersten)

- **Jarke:** Also zum Beispiel, wenn der *Producer* nach dem *wait(S)* feststellt, dass *empty* schon 0 ist, dann kann er nicht weiter machen, und der *Consumer* kann auch nicht weiter machen, da er an *wait(S)* scheitert.

- **Ich:** Ah, ja, und dann warten sie unendlich aufeinander und keiner kann fortschreiten.

- **Jarke:** Also ist es kein so tolles Protokoll, was Sie da entworfen haben. Wenn es Deadlocks gibt, ist das schlecht. Bei der Implementierung von Datenbanken haben wir gelernt, dass sie bei Datenbanksystemen toleriert werden. Hier sind sie aber sehr unerwünscht. Machen wir aber jetzt mit der Implementierung von Datenbanken weiter.

2 Implementation of Databases

- **Jarke:** In der Implementierung haben wir Transaktionen behandelt, und da gibt es auch Synchronisationsprobleme...

- **Ich:** Ja, man muss verhindern, dass die Datenbank in einem inkonsistenten Zustand landet. Ich erkläre zunächst das Read-Write-Model... (Habe dann das Modell erklärt und erklärt, was ein Schedule ist. Dann habe ich den Begriff der Serialisierbarkeit erklärt, die verschiedenen Arten davon aufgezählt: Final-State-, View- und Konfliktserialisierbarkeit. Definition davon oder überhaupt irgendwelche Erläuterungen wollte Jarke nicht hören. Ich habe nur die Namen aufgezählt. Dann habe ich erzählt, dass die ersten zwei für praktische Zwecke nicht geeignet sind, weil die entsprechenden Entscheidungsprobleme NP-vollständig sind.) Die Konfliktserialisierbarkeit ist dagegen für die Praxis geeignet, weil sie mit dem Konfliktgraphen (habe erklärt, was das ist) in Linearzeit entscheidbar ist (falsch, in $O(n^2)$). Soll ich jetzt die anderen Serialisierbarkeiten erklären?

- **Jarke:** Nein, wir wollen ja nicht bis Mitternacht hier sitzen:) Wie ist es denn mit dem Konfliktgraphen, wie sieht man daran, ob ein Schedule Konflikt-Serialisierbar ist?

- **Ich:** Es gibt keine Zycklen in dem Graphen

- **Jarke:** Können Sie das beweisen?

- **Ich:** Ja (male einen Konflikt-Graphen mit Zycklus auf und erkläre, warum der Zyckel bei einem Konflikt-Serialisierbaren Schedule ein solcher Zycklus unmöglich ist)

- **Jarke:** Hier können aber Deadlocks auftreten.

- **Ich:** Ja, klar. Das hat aber mit der Serialisierbarkeit nichts zu tun. Ist aus einem anderen Bereich.

- **Jarke:** Womit realisiert man einen Konflikt-Serialisierbaren Scheduler?

- **Ich:** Mit einem Zwei-Phasen-Scheduler, soll ich seine Funktionsweise beschreiben?

- **Jarke:** Ja, machen Sie.

- **Ich:** (Erkläre zuerst, was Sperren sind, die Schreib- und die Lesesperren. Erwähne dabei, was denn ein Konflikt eigentlich ist - dass mindestens eine der Sperren auf demselben Objekt eine Schreibsperre ist. Dann erkläre die Funktionsweise des Zwei-Phasen-Schedulers (2PL))

- **Jarke:** Sind dabei Deadlocks möglich?

- **Ich:** Ja.
- **Jarke:** Was macht man dann?
- **Ich:** Also man kann den Banker's Algorithmus benutzen...
- **Jarke:** Nein, nein, das macht man hier nicht bei Datenbanksystemen. Außerdem ist der Banker's Algorithmus dafür da, Deadlocks vorzubeugen.
- **Ich:** Man kann damit Deadlocks auch erkennen. Hier sucht man dann eine Transaktion im Deadlock-Zycklus aus und bricht sie ab. Damit wird der Deadlock gelöst. Es ist wichtig, welche Transaktion man aussucht, es ist ein wichtiger Tuning-Faktor. Man könnte zum Beispiel die Transaktion abbrechen, die noch am wenigsten gelaufen ist - man könnte vermuten, dass die Auswirkungen auf die anderen Transaktionen dadurch minimiert werden.
- **Jarke:** Ja, gut, dass Sie das erwähnt haben. Sie sprachen von Auswirkungen auf andere Transaktionen. Was meinen Sie damit?
- **Ich:** Wenn eine Transaktion abgebrochen wird, müssen alle ihre Auswirkungen auf die Datenbank und auf die anderen Transaktionen rückgängig gemacht werden. Alle Werte, die von ihr in die Datenbank geschrieben wurden, müssen durch ältere Versionen ersetzt werden und alle Transaktionen, die solche Werte gelesen haben, müssen ebenfalls abgebrochen werden.
- **Jarke:** Jaaa, hier haben wir den Fall kaskadierender Aborts. Können Sie sich einen Scheduler vorstellen, der diese vermeidet?
- **Ich:** Hmm...(überleg, überleg, hab ihm dann fälschlicherweise die Definition der ST-Klasse gegeben. Jarke meinte dann, es wäre die ST-Klasse und nicht die ACA-Klasse. Ich hab mich dann korrigiert und die Definition der ACA-Klasse gegeben, damit war er aber nicht zufrieden. Er wollte einen speziellen Scheduler-Typ hören.) Nee, fällt mir jetzt nicht ein. Was ist das denn für einer?
- **Jarke:** Der strenge 2PL-Scheduler.
- **Ich:** Aahh, ja.

- **Jarke:** (Erklärt, warum der strenge 2PL-Scheduler einen ACA-Schedule ausgibt.) Sie haben davon gesproche, dass die Auswirkungen der abgebrochenen Transaktionen rückgängig gemacht werden sollen. Wie geschieht das?
- **Ich:** Das geschieht mithilfe des Logs und des Undo/Redo Protokolls.
- **Jarke:** Also in diesem Fall nur Undo-Protokoll, Redo brauchen wir hier nicht.
- **Ich:** Soll ich das Undo/Redo-Protokoll erklären?
- **Jarke:** Ja, machen Sie bitte.
- **Ich:** (Erkläre das Protokoll: Checkpoints, Logeinträge, Rück- und Vorwärtslaufen im Log beim Wiederherstellen...)
- **Jarke:** Ja, gut. Jetzt noch etwas zur Einführung in Datenbanken.

3 Introduction to Database Systems

- **Jarke:** In letzter Zeit kommen neue Datenbankarchitekturen in Gebrauch, die XML-Datenbanken. Erläutern Sie doch die wesentlichen Unterschiede zum herkömmlichen relationalen Datenbankmodell.
- **Ich:** (Erkläre den Unterschied zwischen den strukturierten und semistrukturierten Datenbanken)
- **Jarke:** Und wie wird das denn realisiert in XML?
- **Ich:** (Erkläre, wie man ein XML-Dokument aufbaut, wie die einzelnen Entities in Tags dargestellt werden)
- **Jarke:** Und wie wird nach einem speziellen Knoten gesucht?
- **Ich:** (verstehe nicht ganz, was er meint) Hmm..., also die Entities werden nacheinander mit dem Suchkriterium verglichen und die, die es erfüllen, werden ausgegeben. Also sequentielle Suche.
- **Jarke:** Sequentielle Suche? Bei einer Baumstruktur?
- **Ich:** Ja, schon. Die Entities müssen ja nacheinander durchlaufen und mit dem Suchkriterium verglichen werden. Aber die Anfrage geschieht zum Beispiel mit XPath

- **Jarke:** Aha! (das wollte er wohl von Anfang an hören)
- **Ich:** (Erkläre, wie eine Anfrage mit XPath aussieht, ihre Syntaxis und Semantik.)
- **Jarke:** Welche Anfragesprache noch gibt es für XML-Dokumente?
- **Ich:** XQuery, sieht ähnlich dem SQL aus, aber ich glaube, XQuery erlaubt keine Rekursion.
- **Jarke:** (sagt etwas dazu, weiß ich nicht mehr was) Stellen Sie doch die beiden Anfragesprachen gegenüber und zeigen Sie die Unterschiede.
- **Ich:** (Schreibe die Grundstrukturen der SQL- und XQuery-Anfrage auf. Erkläre was was ist. Also bei SQL ist FROM - kartesisches Produkt, SELECT - Projektion und WHERE - Selektion. Und bei XQuery definiert FOR die Variablengültigkeit, WHERE - Selektion, RETURN die Rückgabewerte. Erzähle noch nebenbei darüber, vergleiche die beiden Sachen und sage, dass sie sehr ähnlich sind)
- **Jarke:** Wo ist denn der wesentliche Unterschied?
- **Ich:** (keine Ahnung, fang an, rumzurätseln)
- **Jarke:** Was wird bei SQL als Ergebnis zurückgeliefert?
- **Ich:** Die Werte von Attributen, die in der SELECT-Klausel stehen. Die Ergebnistupel werden darauf projeziert, das SELECT eine Projektion darstellt.
- **Jarke:** Und bei XQuery?
- **Ich:** (Ahh!! Jetzt fällt der Groschen!) Bei XQuery werden die kompletten Knoten mit daran hängenden Teilbäumen zurückgeliefert.
- **Jarke:** Also im Unterschied zu einfachen Werten bei SQL die strukturierten Rückgaben bei XQuery. Und wie ist es mit der FOR-Klausel, wie wird da auf die Elemente zugegriffen?
- **Ich:** Mit Hilfe eines XPath-Ausdrucks (erkläre wie das funktioniert).
- **Jarke:** Ja, genau. (macht Themawechsel) Erzählen Sie mir jetzt über das relationale Datenbankschema.
- **Ich:** (Erkläre das Schema einer Relation: Name,Signatur, Attributdomänen, FDs usw.)

- **Jarke:** Da fehlt aber noch was.
- **Ich:** Ah ja, noch die Integrity Constraints in der Relation (erkläre, was das ist)
- **Jarke:** Ja, aber es fehlt noch was ganz wichtiges.
- **Ich:** (Was denn??? Hab doch schon alles benannt, grübel, grübel...) Hmm, können Sie vielleicht ein bisschen andeuten, was Sie meinen? Ich weiß es bestimmt!
- **Jarke:** Das freut mich zu hören:)) Also die einzelnen Relationen im Datenbankschema...
- **Ich:** Ähh? Im Datenbankschema? Wir haben doch von Relationschema gesprochen.
- **Jarke:** Also ich habe die ganze Zeit vom Datenbankschema gesprochen.
- **Ich:** Ah, OK, ich habe es wohl falsch verstanden. Also bei dem Datenbankschema kommen noch die Integrity Constraints zwischen einzelnen Relationen hinzu.
- **Jarke:** Und was sind das?
- **Ich:** (Hab das am Beispiel von Foreign Keys erklärt. Also wenn in einem Tupel ein Foreign Key enthalten ist, dann muss es einen gültigen Tupel in einer anderen Relation mit dem Foreign Key als Schlüssel geben usw.)
- **Jarke:** Ja, Foreign Keys sind ein gutes Beispiel. Würden Sie bitte kurz rausgehen?

4 Fazit

Am Ende sagte Professor Jarke, dass man mir eigentlich eine 1.7 oder eine 1.3 geben könnte, da ich eigentlich fast alles gewusst habe. Allerdings hat er bei mir bemängelt, dass ich öfters seine Fragen mißinterpretiert habe und in die falsche Richtung geredet habe. Er musste mich dann immer wieder zurückholen und in die richtige Richtung lenken. Deswegen auch die 2.0.

Na ja, ich persönlich fand seine Fragen oft recht schwammig und vaage, so dass ich nicht richtig wusste, was er denn von mir hören bzw. worauf er hinaus will. Insgesamt fand ich aber die Prüfung fair und angemessen.

Professor Jarke stellt keine fiesen Fragen und hilft gerne, wenn man nicht weiter weiß. Er versucht immer zu schauen, ob man etwas wirklich nicht weiß, oder nur einen Brett vor dem Kopf hat.

Sollte jemand noch Fragen bezüglich dieses Protokolls haben, kann er mir gerne schreiben.

5 Benutzte Literatur

- Betriebssysteme I und II
  - Operating system concepts. Silberschatz, Abraham / Galvin, Peter B. / Gagne, Greg, 7. Aufl
  - Systemprogrammierung - Skript zur Vorlesung an der RWTH Aachen. Spaniol, Günes, Wienzek, Macherey, 3. Auflage
    Link:
    `http://www-i4.informatik.rwth-aachen.de/content/teaching/scripts/downloads/system_programmierung.pdf`
- Implementation of Databases
  - Vorlesungsfolien
  - G. Vossen: Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme, Oldenbourg, 4. Auflage, 2000
  - A. Kemper, A. Eickler: Datenbanksysteme, 5. Auflage, Oldenburg, 2004
  - Query Optimization in Database Systems. ACM Computing Surveys (CSUR) Volume 16 , Issue 2 (June 1984), Pages: 111 - 152
    Link:
    `http://www-i5.informatik.rwth-aachen.de/lehrstuhl/lehre/IDB06/download/Literature/QO.pdf`
  - Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, Thomas G. Price: Access Path Selection in a Relational Database Management System. SIGMOD Conference 1979: 23-34
    (Kostenmodelle für Relationzugriffe und JOIN)
- Introduction to Database Systems
  - R. Elmasri, S.B. Navathe: Fundamentals of Database Systems, Addison Wesley, 4th edn, 2004

- What Is RDF by Joshua Tauberer July 26, 2006
  Link:
  `http://www.xml.com/pub/a/2001/01/24/rdf.html`
- UML-Tutorial
  Link:
  `http://ivs.cs.uni-magdeburg.de/~dumke/UML/inhalt.htm`