Gedächtnisprotokoll mündliche Diplomprüfung Informatik Praxis / Jarke (i5)

Datum: 2007 *Dauer*: 45 Minuten *Note*: 1,3

Vorlesungen

- Einführung in Datenbanken (EDB),
- Implementierung von Datenbanken (IDB),
- Betriebssysteme (BS).

Wichtig: Zu Beginn der Prüfung sagen, wann man die Vorlesungen gehört hat, die Inhalte unterscheiden sich bei neueren Themen teilweise deutlich, auch wenn es erst zwei Jahre her ist.

Wer wie ich BS nicht gehört hat, dem nennt Prof. Jarke einige Kapitel aus dem Buch von Silberschatz/Galvin, meist alles außer Verteilte System und Netzwerke, weil es dafür eigene Vorlesungen gibt. Außerdem soll man sich eines der Fallbeispiele für Betriebssysteme aussuchen, was in meiner Prüfung allerdings nicht drankam.

Materialien

- Folien der beiden Datenbank-Vorlesungen von 2004/5
- Einige der IDB-Videoaufzeichnungen von s-inf.de.
- Ramez Elmasri, Shamkant B. Navathe: *Fundamentals of Database Systems*. 2. Auflage. The Benjamin/Cummings Publishing Company, Redwood City, 1994. ISBN 0-805-31753-8.
- Gottfried Vossen: *Datenbankmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 3. Auflage. R. Oldenbourg Verlag, München Wien 1999. ISBN 3-486-24544-9.
- Abraham Silberschatz, Peter B. Galvin: *Operating System Concepts*. 4. Auflage. Addison-Wesley, Reading etc., 1994. ISBN 0-201-59292-4.
- Prüfungsprotokolle von s-inf.de

Der Vossen ist sehr trocken, und gerade bei den komplexeren Dingen fehlen gute Beispiele bzw. eine praktische Betrachtung der Theorie.

Der Elmasri/Navathe ist sehr lesbar, und die Beispiele stimmen. Allerdings geht er bei den mathematischen Konzepten rund um Relationenalgebra und funktionale Abhängigkeiten nicht ausreichend in die Tiefe.

Der Silberschatz/Galvin ist ebenfalls lesbar, erklärt aber manchmal Einfaches zu ausführlich und Komplexes gar nicht, was natürlich anders herum sein sollte.

Allerdings habe ich ziemlich alte Auflagen dieser Bücher benutzt, vielleicht hat sich das in neueren geändert.

Vorwort

Die Prüfung war angenehm, Jarke mag längere Antworten und lässt den Prüfling ausreden. Wenn man mal etwas nachdenken muss, ist das auch nicht schlimm, scheint mir. Falls man um den heißen Brei redet, weil einem die präzise Lösung fehlt, sagt er einem das allerdings auf den Kopf zu (Zitat: „Das war jetzt aber eine sehr generische Antwort, oder?“).

Prüfungsverlauf

Bibliothek des i5, Jarke, Beisitzer, ich.

Betriebssysteme

- **Speicherhierarchie erklären:** von CPU-Registern, L1/L2-Cache, Hauptspeicher, Platte, optisches Laufwerk, Bandlaufwerk. Evtl. noch Netzwerk. Je weiter von der CPU weg, desto langsamer und desto preiswerter pro Speichereinheit.
- **Wo ist der größte Sprung?** Zwischen Hauptspeicher und Platte, Zugriff Hauptspeicher ca. 60 ns, Platte ca. 6 ms, Faktor 100.000.
- **Wie funktioniert Hauptspeicherverwaltung?**

  Prozesse hatten ursprünglich zusammenhängenden Speicher. Führt zu externer Fragmentierung, viele Löcher; neuer Prozess passt u.U. nicht in den Speicher, obwohl eigentlich die Summe der Löcher groß genug wäre. Kann man zwar durch Umkopieren lösen, aber besser: mittels Paging Unterteilung des physikalischen Hauptspeichers in gleich große Seiten.

  Zugriff auf Speicherstelle läuft dann so: Programm liefert logische Adresse (sein Speicher stellt sich für das Programm als zusammenhängender Speicherbereich von 0 bis n dar), Betriebssystem übersetzt mittels Page Table. Unterer Teil der logischen Adresse wird Offset innerhalb der Seite, oberer Teil ist Seitennummer. Zuerst in Translation Lookaside Buffer nachsehen, sonst in der Page Table und dann in TLB eintragen. (Begründung für TLB: Page-Table-Zugriff erfordert einen Speicherzugriff, also für jeden normalen Speicherzugriff des Programms einen weiteren durch das Betriebssystem, also halbe Geschwindigkeit; hier beschleunigt der ca. fünfmal schnellere TLB den Zugriff).
- **Demand Paging?** Prozess an schnell zugreifbarer Stelle auf der Platte. Page Table erweitern, für jede Page ein Bit mit der Bedeutung *im Hauptspeicher: ja/nein*. Wenn Page nicht im Speicher: Page Fault, freie Seite aussuchen, Seite hineinladen, Instruktion mit Speicherzugriff erneut ausführen.
- **Was passiert, wenn keine Seite mehr frei ist?**

  Ersetzen, Algorithmen für Page Replacement:

  - FIFO (älteste Seite wird ersetzt, älteste im Sinne von *Ladevorgang am längsten her*, nicht am längsten ungenutzt, dazu s.u.)
  - Optimal (welche Seite wird in Zukunft am längsten nicht mehr gebraucht werden – kann nur geraten werden)
  - Least Recently Used (LRU), die Seite, die am längsten nicht mehr benutzt wurde
- **LRU-Implementierungsvariante Second Chance erklären?**

  Zusätzliches Bit *Seite wurde in letzter Zeit benutzt* (ja/nein). Ringpuffer der Seiten, ein Zeiger auf *nächstes Opfer*: erste Seite nehmen, deren Bit nicht gesetzt ist. Ist das Bit gesetzt, löschen und zur nächsten. Selbst wenn alle Bits gesetzt waren, kommt man irgendwann wieder bei der ersten an, deren Bit man zuerst gelöscht hatte und nimmt diese.

- **Was sind Semaphoren?**

  Integer-Variablen, um das Problem mit dem gegenseitigen Ausschluss von Prozessen (Critical Sections) zu lösen. Prozeduren Signal und Wait aufgeschrieben:

  ```
  Wait:
    while S <= 0 do no-op;   s--;

  Signal:
    s++;
  ```

  Benutzt man zum gegenseitigen Ausschluß, zum Beispiel beim Produzenten-Konsumenten-Problem.

- **Wie kann man dieses Problem mit Semaphoren lösen?** (Steht im Silberschatz.) Puffer mit n Elementen. Drei Semaphoren:

  ```
  full := 0; empty := n; mutex := 1;
  ```

  Produzenten-Code:

  ```
  wait(empty);
  wait(mutex);
  ... Element hinzufügen ...
  signal(mutex);
  signal(full);
  ```

  Konsumenten-Code vertauscht full und empty:

  ```
  wait(full);
  wait(mutex);
  ... Element entnehmen ...
  signal(mutex);
  signal(empty);
  ```

- **Was passierte, wenn wait mutex jeweils erste Operation wäre?**

  Etwas länger überlegt, dann: Es kann zum Deadlock kommen. Wenn der Konsumenten-Prozess zuerst wait mutex aufruft, kommt er daran vorbei, und er wartet auf wait full. Der Produzenten-Prozess hingegen wartet auf sein wait mutex.

- **Bedingungen für Deadlock, warum gelten sie in diesem Fall?**

  Hold and wait: mindestens ein Prozess hält eine Ressource und möchte eine andere haben, hier der Konsument. Circular wait: minimaler Kreis, Produzent wartet auf mutex, Konsument auf full. No preemption: niemand gibt seine Ressource freiwillig auf, hier der Konsument als einziger Besitzer einer Ressource. Mutual exclusion: mindestens eine gehaltene Ressource kann nicht geteilt werden, hier mutex.

Implementierung von Datenbanken

An den genauen Übergang kann ich mich nicht mehr erinnern. Aber wir kommen auf 2PL zu sprechen. Ich male die wachsende und die schrumpfende Phase auf und erläutere die Grundlagen.

- **Gibt es Deadlocks auch bei Datenbanken?**

  Ja, wenn Transaktionen Objekte sperren wollen, die andere besitzen, die ihrerseits Objekte sperren wollen. Wenn da ein Zykel gebildet wird, hat man einen Deadlock. Conserve 2 PL hat keine Deadlocks, dafür muss man am Anfang alle Ressourcen belegen, die man braucht; ist in der Praxis unrealistisch, da die Transaktionsbestandteile oft erst nach und nach erzeugt und an den Transaktionsmanager weitergereicht werden.

- **Wie wird der dann aufgelöst?**

  Der Vorgang ist vermutlich derselbe wie beim Abort. Hier brauche ich etwas. Der Recovery-Manager!

- **Wie funktioniert der?**

  UNDO/REDO erklärt, je nach Datenbank vier Kombinationen nur REDO, nur UNDO, UNDO und REDO, weder UNDO noch REDO. Before-Images: Datensätze im Log, die für eine Änderung den alten und den neuen Wert enthalten. Also in diesem konkreten Fall (Abort einer Transaktion): UNDO, solange bis BOT (Beginning of Transaction) gefunden wird.

- **Zurück zu 2PL.**

  Ich weiß die Überleitung nicht mehr, aber ich erkläre, was Serialisierbarkeit ist und die Fehlersicherheit: Orthogonale Konzepte, beide notwendig für eine Datenbank mit ACID-Eigenschaften. Dann erläutere ich die Scheduler-Klasse ACA (avoid cascading aborts). Ich schreibe die Bedingungen für ACA und ST auf.

- **Striktes 2PL?** Behält alle Ressourcen bis zum Ende der Transaktion. Vorteil: Keine bereits von (A) wieder freigegebenen Ressourcen können von anderer Transaktion (B) genutzt werden, die vielleicht einen Abort durchführt, bevor (A) einen Commit machen kann.

Einführung in Datenbanken

- **Relationales und objektorientiertes Datenbankmodell vergleichen. Warum ist bei OODB keine Normalisierung notwendig?**

  Normalisierung beim relationalen Datenmodell notwendig, um Redundanzen und Anomalien zu entfernen (kurz erläutert). Bei OO lädt das Datenmodell dazu ein, komplexe Dinge direkt zu modellieren.

- **Vernünftig modellieren kann man doch auch mit dem relationalen Modell?**

  Ich erwidere, man muss aber trotzdem teure Joins benutzen, um mehrere komplexe Objekte zu laden, während es bei der Implementierung des OO-Modells mithilfe der Objekt-ID möglich ist, in Objekte gekapselte andere Objekte gezielt und schnell nachzuladen.

  Jarke meint, das wäre ein sehr wichtiger Punkt, die vollständige Lösung, die er gern gehabt hätte, nennt er dann aber nicht. In meinen EDB-Folien habe ich die Antwort auch nachträglich nicht gefunden, steht vielleicht in neueren Versionen.

- **Normalformen?**

  1NF, 2NF, 3NF erklärt.

- **Synthese-Algorithmus?**

  Nur kurz eingeordnet: wozu ist er gut, es wird zunächst eine Basis gebildet, dann mittels dieser Basis nach und nach für FDs Tabellen erzeugt. Ich gebe zu, den Algorithmus nicht im Detail erklären zu können.

EDB ist offensichtlich etwas zu kurz gekommen, aber nach exakt 45 Minuten ist es vorbei, kurze Besprechung in meiner Abwesenheit, Note 1,3, wegen ein paar Ungenauigkeiten. Viel Erfolg bei Eurer Prüfung, und erstellt bitte auch ein Protokoll!