Lernskript

Praktische Informatik

Themen:

- Betriebssysteme
  (Andrew S. Tanenbaum)
- Verteilte Systeme
  (Andrew S. Tanenbaum)
- Datenbanken
  (C. J. Date)
- Programmiersprachen
  (Kenneth C. Louden)

„Ein verteiltes System ist ein System, mit dem man nicht arbeiten kann, weil ein Rechner ausgefallen ist, von dem man noch nie etwas gehört hat…“

(Leslie Lamport)

Christoph Haas

19.10.99

I Traditionelle Betriebssysteme

(1) Einführung

Definition Betriebssystem

Systemsoftware eines Rechensystems

- Schnittstelle für den Zugriff auf die Hardwarekomponenten des Systems (Systemaufrufe / virtuelle Maschine)
- Verwaltung (und Koordination) von Ressourcen

Betriebssystemkonzepte

Prozesse

Ein Prozess ist ein Programm während seiner Ausführung. Er besteht aus:

- Programm
- Programmzähler
- Programmdaten
- Stack
- Stackpointer
- uid (User Identification)
- gid (Group Identification)

Bei der Prozessumschaltung werden alle für die Fortführung benötigten Daten als *core image* gespeichert.

Nur der Eigentümer (uid) oder der Vater-Prozess haben Zugriff auf einen Prozess.

Signale unterbrechen die Ausführung eines Prozesses vom Betriebssystem aus (➔bei Benutzung illegaler Adressbereiche oder Ablaufen eines Timers) und erlauben das Abarbeiten eines *Handlers* zur Auswertung des Signals. Signale entsprechen den Hardware-Interrupts in der Software.

Dateien

Dateien sind Datenbereiche, die vom Betriebssystem verwaltet werden. Zur Strukturierung des Dateisystems werden *Verzeichnisse* verwendet. Dateien werden durch (relative/absolute) *Pfade* bezeichnet.

Zur Benutzung von Dateien werden vom Betriebssystem Systemaufrufe bereitgestellt.

Der Zugriff auf Dateien kann genau geregelt werden (➔rwx bei UNIX). Wird eine Datei geöffnet, erhält der verantwortliche Prozess zum Zugriff einen *Filehandle*.

Peripherie kann durch *special devices* angesprochen werden.

Es gibt besondere Dateideskriptoren (*file handle*) für die Standardeingabe (STDIN = 0), Standardausgabe (STDOUT = 1) und Standardfehlerausgabe (STDERR = 2).

Pipes verbinden zwei Dateien miteinander, um Daten auszutauschen. Die Standardausgabe eines Prozesses wird dabei an die Standardeingabe des nächsten Prozesses weitergeleitet.

Systemaufrufe

Aufruf einer Systemfunktion des Betriebssystems. Erlaubt einem Benutzerprogramm den Zugriff auf Ressourcen oder Peripherie. Die Systemfunktion liefert einen Rückgabewert, der den Erfolg der Aktion bezeichnet.

Shell

Kommandointerpreter (in UNIX) zur Interaktion des Benutzers mit dem Betriebssystems durch Befehle. Die Shell wird nach der Anmeldung eines Benutzers an das System gestartet.

Monilithische Struktur

Das Rechensystem unterscheidet nur zwischen *Kernelmodus* und *Benutzermodus*. Im Benutzermodus sind I/O-Operationen, Speicherzugriffe und andere Befehle nicht erlaubt. Das Betriebssystem selbst läuft im Kernelmodus ab. Das Anwendungsprogramm aktiviert einen TRAP und springt so in den Kernelmodus.

[Geschichtete Systeme]

Virtuelle Maschinen

Auf der untersten Schicht verwaltet ein *virtual machine monitor* mehrere virtuelle Maschinen. Der darüberliegenden Schicht wird jeweils eine exakte Kopie der CPU (mit Kernel- und Benutzermodus) angeboten. So können verschiedenen Betriebssysteme gleichzeitig auf einer realen CPU ablaufen.

Client-Server-Modell

Jeder Dienst im System ist ein Server. Die Client-Prozesse können Nachrichten an einen Server schicken und erhalten eine Antwort. Im Kernelmodus werden nur Nachrichten zwischen Client und Server übertragen. Für besondere (zeitkritische) Aufgaben kann ein Client-Prozess mit Sonderrechten direkt teilweise im Kernelmodus ablaufen.

(2) Prozesse

Multitasking

Durch den Wechsel des Betriebssystems zwischen verschiedenen Benutzerprozessen erhält der Benutzer den Eindruck einer gleichzeitigen Abarbeitung von Prozessen. Dies nennt man Pseudo-Multitasking.

Das Prozessmodell

Ein (sequentieller) Prozess ist ein Programm während seiner Ausführung. Jeder Prozess belegt eine *virtuelle CPU* (die reale CPU wird zwischen den Prozessen zeitlich geteilt). So arbeitet das Rechensystem im *Mehrprogrammbetrieb*. Die Art des Umschaltens wird durch das *Scheduling* definiert.

Prozesshierarchien

Prozesse können weitere Prozesse erzeugen. Unter UNIX gibt es einen *fork*-Aufruf, der den aktuellen Prozess dupliziert. Dadurch entsteht eine baumartige Prozesshierarchie.

Prozesszustände

rechnend:
: Die CPU bearbeitet gerade diesen Prozess.

bereit:
: Der Prozess ist ausführbar, aber die CPU bearbeitet gerade einen anderen Prozess.

blockiert:
: Der Prozess wartet auf ein externes Ereignis (➔eine Eingabe). Sobald das Ereignis eintrifft, wechselt der Prozess zurück in den Zustand „bereit“.

Implementierung von Prozessen

Prozesstabelle

Das Betriebssystem verwaltet Prozesse über die *Prozesstabelle*. Dort stehen u.a. folgende Informationen über jeden Prozess, damit er vom Scheduler wieder in den Zustand „rechnend“ versetzt werden kann:

- Register
- Prozesszustand (rechnend, bereit, blockiert)
- pid (Process Identification)
- Programmzähler
- Stackpointer
- Belegte Ressourcen (Speicher, Dateien)
- …

Interrupt-Vektor

Der Interrupt-Vektor eines Gerätes enhält die Speicheradresse der Routine für den Interrupt-Handler. Wird ein Hardware-Interrupt ausgelöst, so wird der aktive Prozess verdrängt und der Handler aktiviert. Der blockierte Prozess, der auf eine Eingabe von diesem Gerät wartet, wird jetzt reaktiviert (von „blockiert“ in den Zustand „bereit“ gesetzt).

Prozesskommunikation (IPC)

In einem Einprozessorsystem können Prozesse einfach über gemeinsame Speicherbereiche miteinander kommunizieren.

In verteilten Systemen ist die Kommunikation nur über Nachrichtenaustausch möglich.

Beispiel: Druckerspooler

Wenn ein Prozess eine Datei ausdruckt, schreibt er eine Datei in ein *Spool-Verzeichnis*. Der verantwortliche Druckprozess (*Printer-Daemon „lpd“*) überprüft periodisch den Inhalt des Spool-Verzeichnisses und druckt die Dateien einzeln nacheinander aus.

Wechselseitiger Ausschluss

Um den gleichzeitigen Zugriff auf eine Ressource zu verhindern, verwendet man den *wechselseitigen Ausschluss*. Die Prozesse können dann nur einzeln z.B. auf eine Variable zugreifen. Ist die Ressource ein Programmteil, der nicht parallel ausgeführt werden darf, nennt man ihn einen *kritischen Bereich*.

Dabei sollten folgende Regeln eingehalten werden:

1. nur ein Prozess ist jeweils im kritischen Bereich
2. keine Annahmen über die Rechenleistung der CPU
3. ist ein Prozess im kritischen Bereich, darf er keine anderen Prozesse blockieren
4. wer in den kritischen Bereich möchte, darf das auch in endlicher Zeit (Fairness)

Lösungen mit aktivem Warten (busy waiting)

Sperren der Interrupt:
: Alle Interrupts im System werden während der Ausführung des kritischen Bereichs abgeschaltet.
Nachteil: System während eines Benutzerprozesses komplett blockiert

Striktes Alternieren:
: Zwei Prozesse können wechselseitig einen kritischen Bereich durchlaufen und dann ein Flag (Sperrvariable) setzen, das dem jeweils anderen Prozess signalisiert, dass er jetzt den kritischen Bereich betreten kann.
Nachteil: Verschwendung von Rechenzeit und Verstoß gegen Regel 3 (anderer Prozess wird blockiert)

Decker-Peterson-Algorithmus:
: Es gibt ein Array *interested[]*, wo ein Prozess sein Interesse anzeigt, den kritischen Bereich zu

betreten. Vor dem Eintritt in den kritischen Bereich prüft er, ob kein anderer Prozess interessiert ist.

TSL-Anweisung:

In Mehrprozessor-Systemen gibt es häufig eine Anweisung „TSL“ (*test and set lock*). Sie prüft unteilbar (!) den Wert eines Flags und setzt ihn auf 1, wenn er 0 war. Der Prozess kann dann den kritischen Bereich betreten und anschließend das Flag zurücksetzen. Ansonsten muss er warten, bis das TSL das Flag erfolgreich auf 1 gesetzt hat.

Nachteile bei aktivem Warten:

- Verschwendung von CPU-Zeit
- Mögliche *race conditions*
  (durch ungünstiges Scheduling könnte der andere Prozess nie zum Zug kommen)

Lösung durch SLEEP / WAKEUP

Erzeuger-Verbraucher-Problem:

Zwei Prozesse haben einen Puffer endlicher Größe. Der Erzeuger füllt den Puffer – der Verbraucher leert den Puffer. Ist der Puffer voll, blockiert der Erzeuger (SLEEP) bis der Verbraucher wieder Platz geschaffen hat. Genauso blockiert der Verbraucher, wenn der Puffer leer ist. Der jeweils aktive Prozess überprüft, ob der andere Prozess blockiert ist und aktiviert ihn. Da der Zugriff auf die Zählervariable (Füllung des Puffers) nicht ungeteilt ist, kann es zu Deadlocks kommen (jeder Prozess hat ein SLEEP ausgeführt).

Semaphore

Ein *Semaphor* ist eine ganzzahlige Variable, die mit der Anzahl der Prozesse initialisiert wird, die gleichzeitig in einem kritischen Bereich sein dürfen.

Betritt ein Prozess den kritischen Bereich, so wird DOWN aufgerufen - die Variable wird dekrementiert (falls sie größer als 0 war) und der Bereich betreten. Anderenfalls wird der Prozess blockiert. Verlässt der Prozess den kritischen Bereich, wird UP aufgerufen und die Variable wieder um 1 erhöht bzw. der nächste blockierte Prozess freigegeben (falls noch einer wartet). Vorteil: kein busy waiting!

Wichtig: die beiden Anweisungen für das verändern des Semaphors werden im Betriebssystem unteilbar ausgeführt. Während dieser Prozeduren werden die Interrupts gesperrt.

Bei falscher Anwendung von Semaphoren kann es trotzdem zu Deadlocks kommen.

Semaphoren funktioneren auch nur lokal auf einem System. Im Netz sind sie unbrauchbar.

Lösung des Erzeuger-Verbraucher-Problems mit Semaphoren

Mit Hilfe von Semaphoren kann das Erzeuger-Verbraucher-Problem gelöst werden. Es gibt dazu drei Semaphore:

- *full* (Puffer voll?)
  anfangs: full = 0
- *empty* (Puffer leer?)
  anfangs: empty = n
- *mutex* (mutual exclude für Zugriff auf Puffer)
  anfangs: mutex = 1

Algorithmus siehe Tanenbaum.

Semaphore werden in der Praxis selten eingesetzt.

Ereigniszähler

Ereigniszähler sind nicht-negative Integer-Variablen, die sich nur erhöhen lassen. Der Erzeuger zählt seinen Ereigniszähler jeweils um 1 hoch, wenn er ein Paket in den Puffer gelegt hat. Der Verbraucher zählt seinen Ereigniszähler jeweils um 1 hoch, wenn er ein Paket aus dem Puffer entnommen hat.

Es gibt drei Operationen für Ereigniszähler:

- Read(E)
  liest den Wert von E
- Advance(E)
  inkrementiert E (unteilbar)
- Await(E,i)
  wartet bis $E \geq i$

Monitore

Hochsprachliche Implementation einer unteilbaren Handlung. Der Monitor ist eine Menge von Prozeduren (und Variablen), die nur im wechselseitigen Ausschluss ausgeführt werden dürfen. Der Compiler ist hierfür verantwortlich. Menschliche Fehler werden vermieden (z.B. eine falsche Down-Anweisung, wenn der Semaphor bereits 0 ist). Er verwendet dazu intern binäre Semaphore.

Lösung des Erzeuger-Verbraucher-Problems mit Monitoren

Zusätzlich werden die Systemfunktionen *WAIT* und *SIGNAL* mit *Bedingungsvariablen* verwendet. WAIT(x) blockiert den aktuellen Prozess. Eine andere Prozedur des Monitors könnte dann mit SIGNAL(x) den blockierten Prozess aktivieren. Dabei wird er selbst blockiert, um die Monitorbedingung zu erfüllen.

Monitore sind nur in sehr wenigen Programmiersprachen implementiert.
(➔Concurrent Euclid, Java)

Nachrichtenaustausch (Send/Receive)

Probleme beim Nachrichtenaustausch:

- Nachrichten können in einem Netzwerk verlorengehen (Empfangsbestätigungen, die aber auch verloren gehen könnten)
- Nachrichten müssen eindeutig zugestellt werden (➔*process@machine.domain*)
- Authentifizierung des Senders/Empfängers
- Performance

SEND verschickt asynchron (ohne zu warten) ein Paket an einen anderen Prozess. RECEIVE wartet (blockierend) auf eine Nachricht von einem Prozess.

Lösungen des Erzeuger-Verbraucher-Problems mittels Nachrichtenaustausch

Mailbox-Synchronisation:

Jeder Rechner hat eine Mailbox für n Pakete. Hat er ein Paket erhalten, schickt er ein leeres Paket zurück.
In UNIX werden hierzu Pipes benutzt.

Rendezvous-Synchronisation:

Der Sender führt ein synchrones SEND aus, das solange blockiert, bis der Empfänger ein entsprechendes RECEIVE ausführt. Die Prozesse sind somit eng aneinander gekoppelt.

[Äquivalenz der Kommunikationsprimitiven]

Klassische IPC-Probleme

Das Philosophenproblem

Fünf Philosophen sitzen an einem runden Tisch, in dessen Mitte eine Schüssel Spaghetti steht. Jeder Philosoph kann denken oder essen. Zum Essen stehen insgesamt fünf Gabeln (jeweils zwischen den Tellern) zur Verfügung. Will ein Philosoph essen, so nimmt er in beliebiger Reihenfolge die beiden Gabeln auf.

Problem: es kommt zu einer Verklemmung (Deadlock), wenn alle Philosophen die linke Gabel genommen haben und nicht wieder zurücklegen.

Selbst wenn die linke Gabel wieder zurückgelegt würde, falls die rechte Gabel nicht verfügbar ist, könnte es zum *Verhungern* kommen (nämlich wenn alle Philosophen zeitgleich die Gabeln zurücklegen und wieder aufnehmen).

Der Einsatz von Semaphoren ist suboptimal, da immer nur ein Philosoph essen kann.

Lösung: das Aufnehmen von rechter und linker Gabel muss unteilbar gemacht werden.

Leser-Schreiber-Problem

Das Leser-Schreiber-Problem modelliert Datenbankzugriffe. Es können mehrere Prozesse eine Datei lesen. Falls aber ein Schreibzugriff verlangt wird, darf kein Leseprozess mehr ablaufen.

Lösung: ein Semaphor muss die Zugriffe kontrollieren (vgl. Rechensysteme). Nachteil: falls permanent Leseprozesse stattfinden, könnte der Schreibprozess verhungern.

[Der schlafende Friseur]

Prozess-Scheduling

Der *Scheduler* ist ein Teil des Betriebssystem, der unter Verwendung eines festgelegten *Scheduling-Algorithmus* über die Zuteilung von Rechenzeit an Prozesse entscheidet.

Kriterien:

- Fairness (gerechte Verteilung der CPU-Zeit)
- Antwortzeit (kurze Reaktionszeit)
- Verweilzeit (geringe Gesamtlaufzeit eines Prozesses)

Bei jedem Zeit-Interrupt (z.B. bei jede 1ms) überprüft der Scheduler die Situation und verdrängt möglicherweise rechnende Prozesse in den Zustand „rechenbereit“, um andere Prozesse zu aktivieren (➔präemptives Scheduling).

Round-Robin-Scheduling

Round-Robin ist ein fairer und verbreiteter Scheduling-Algorithmus. Jeder Prozess erhält ein gleiches *Quantum* an Rechenzeit. Ist das Quantum verbraucht, wird der Prozess verdrängt. (Bei einem blockierten Prozess wird nicht auf den Ablauf des Quantums gewartet.) Das Quantum ist in der Praxis bei 1ms gewählt und ist abhängig von der Dauer des Prozesswechsels (Overhead). Ein zu großes Quantum wirkt sich negativ auf die Antwortzeit aus – ein geringes Quantum erzeugt einen zu hohen Overhead.

Prioritäts-Scheduling
(wenn viele Prozesse im System)

Ähnliches Scheduling wie Round-Robin, nur werden immer erst alle Prozesse einer Prioritätsklasse abgearbeitet. Zusätzlich wird bei jedem Zeit-Interrupt die Priorität eines Prozesses erniedrigt (damit ein Prozess nicht auf Dauer die CPU belegen kann. Innerhalb einer Klasse wird Round-Robin-Scheduling verwendet.

Mehrere Warteschlagen
(wenn viel Swapping)

Jeder Prozess erhält zunächst 1 Quantum, dann 2, dann 4…

Dieses Scheduling ist günstiger, wenn der Prozesswechsel lange dauert (bei zuwenig Hauptspeicher und damit intensivem Swapping). Bei kurzen interaktiven Aufträgen ist die Antwortzeit trotzdem gut.

Zweistufiges Scheduling (wenn viel Swapping)

Wenn sich nicht alle Prozesse im Arbeitsspeicher befinden können (Speichermangel), gibt es zwei Scheduler.
Der *Scheduler der oberen Stufe* transportiert Prozesse zwischen Hauptspeicher und Auslagerungsspeicher.
Der *Scheduler der unteren Stufe* übernimmt das eigentliche Scheduling, bei dem nur zwischen Prozessen im Arbeitsspeicher umgeschaltet wird.

Shortest-Job-First (wenn Stapelverarbeitung)

Voraussetzung: die Bedienzeit der Aufträge ist dem Scheduler vor der Ausführung bekannt. Dieser Algorithmus eignet sich gut für Stapelaufträge.

Die Aufträge werden in der Reihenfolge ihrer erwarteten Bedienzeit bearbeitet. Damit wird die mittlere Verweilzeit wird minimiert.

Garantiertes Scheduling

In manchen Systemen (➔Realzeitsteuerung) muss ein Auftrag regelmäßig ausgeführt werden, um wichtige Aufgaben zu erfüllen.

(3) Speicherverwaltung

Der *Speicherverwalter* ist der Teil des Betriebssystems, der sich um die Verwaltung des freien und belegten Speichers sowie das Ein- und Auslagern von Bereichen kümmert.

Einprogrammbetrieb (ohne Swapping)

Der Speicher ist zwischen Betriebssystem und dem (einzigen) Benutzerprozess aufgeteilt. Ein neues Prozess verdrängt den alten, sobald dieser beendet ist.
➔ MS DOS

Multitasking

Zur besseren Ausnutzung der CPU werden mehrere Prozesse (pseudo-) gleichzeitig ausgeführt. Da Prozesse einen hohen Zeitanteil auf Ein- und Ausgaben warten (und dazu nicht die CPU benötigen), ist es sinnvoll, mehrere Prozesse im Speicher zu halten.

Bei einem I/O-Anteil von p jedes Prozesses und n Prozessen im Speicher ergibt sich eine CPU-Auslastung von $1-p^n$.

Beispielsweise liegt der I/O-Anteil bei 80%. Es müssten dann 10 Prozesse im System sein, damit die CPU zu 90% ausgelastet wäre.

(Dieses stochastische Modell ist eine Approximation, da die Prozesse nicht immer stochastisch unabhängig voneinander sind.)

Multitasking mit fixierten Partitionen

Bei fixierten Partitionen wird der Arbeitsspeicher in feste Speicherbereiche verschiedener Größe aufgeteilt. Aus der Menge der wartenden Prozesse wir derjenige ausgewählt, der eine Partition am besten ausnutzt. Zusätzlich wird die Fairness garantiert, damit kleine Prozesse nicht ewig warten müssen.

Nachteil: interne Fragmentierung

Relokation und Schutz

Relokation: beim Compilieren und Linken eines Programms ist noch nicht bekannt, in welchem Speicherbereich das Programm später ausgeführt werden soll. Beim Laden des Programms müssen Anweisungen mit Speicherzugriffen umgeschrieben werden, um an der aktuellen Position im Speicher lauffähig zu sein.

Mögliche Lösungen:

- Softwaremäßig:
  Zu jeder Adresse im Programm wird die Basisadresse des Speicherbereichs addiert (der Linker fügt eine Liste der Adressen an das Programm, die relokiert werden müssen).
- Hardwaremäßig:
  Es gibt zwei Hardware-Register: *Basisregister* (Startadresse des Speicherbereichs) und *Grenzregister* (Länge der Partition). Das System addiert dann selbständig das Basisregister zu jeder Adresse einer Anweisung. Das Grenzregister stellt sicher, dass es zu keinen Schutzfehlern (Zugriffe auf Adressbereiche anderer Prozesse) kommt. Bei dieser Methode kann das Programm sogar während seiner Ausführung im Speicher verschoben werden. (vgl. Segmentierung)

Multitasking mit variablen Partitionen

Swapping:
Verschieben von Prozessen zwischen Hauptspeicher und Auslagerungsspeicher (Festplatte)

Swapping mit fixierten Partitionen führt zu Overhead, da die gesamte Partition ausgelagert wird, obwohl der Speicher eventuell gar nicht vom Prozess verwendet wird. Bei variablen Partitionen werden nur die Nutzdaten ausgelagert.

Ein Problem bei variablen Partitionen ist die zunehmende externe Speicherfragmentierung mit der Zeit. Falls eine Defragmentierung zur Laufzeit möglich ist, kostet sie viel Zeit.

Speicherverwaltung mit Bitmaps

Der Speicher wird in Einheiten unterteilt, für die in der Belegungs-Bitmap jeweils ein Bit reserviert wird. Ist das Bit 0, so ist der Speicherbereich noch frei. Je kleiner die Einheiten, um so größer die Bitmap.

Soll ein Speicherbereich belegt werden, so muss der Speicherverwalter einen passenden Bereich in der Bitmap mit zusammenhängenden Nullen finden. Dies ist ineffizient und wird selten eingesetzt.

Speicherverwaltung mit verketteten Listen

Jeder Speicherbereich führt Referenzen zu dem vorhergehenden und dem nachfolgenden Speicherbereich. Beim Belegen eines Speicherbereichs wird ein ausreichend großes Loch gesucht und die Zeiger der beiden umgebenden Speicherbereiche verändert. Bei der Freigabe des Speichers werden diese Referenzen wiederum aufeinander gesetzt. (Die verkettete Liste ist nach Adressen sortiert.)

Speicherzuteilungsalgorithmen

First Fit:
: Der Speicherverwalter sucht vom Anfang der Segmentliste an ein Loch, das groß genug ist. Dieses Loch wird zweigeteilt. Einen Teil erhält das Programm, der andere Teil wird als frei markiert.

Next Fit:
: Der Speicherverwalter merkt sich, wo er zuletzt ein Loch gefunden hat und sucht von dort aus weiter.
Next Fit ist etwas weniger performant als First Fit.

Best Fit:
: Der Speicherverwalter sucht im gesamten Speicher ein Loch, das möglichst wenig größer als die angeforderte Größe ist. Best Fit ist nicht optimal, weil er kleine (unbrauchbare) Löcher erzeugt.

Quick Fit:
: Der Speicherverwalter führt Listen für vorhandene Löcher verschiedener Größe (2K, 5K, 12K, 20K…). Das Finden von passendem Speicher ist somit extrem schnell.

Speicherverwaltung nach dem Buddy-System

Wie beim Quick-Fit verwaltet das System mehrere Listen von Löchern – hier aber mit Zweierpotenzen (1K, 2K, 4K…1M, 2M, 4M…), denn Zweierpotenzen erlauben eine effizientere Verschmelzung auf Bit-Ebene. Wird ein Speicherbereich angefordert, wird die Größe auf die nächst größere Zweierpotenz aufgerundet (48K➔64K) und dieser Bereich belegt. Der verbleibende Speicher wird in entsprechend kleinere Blöcke (*„Buddies“*) aufgeteilt (➔interne Fragmentierung). Buddies werden bei der Freigabe von Speicher möglichst wieder zu größeren Buddies verschmolzen.

Leider wird durch die sehr seltene exakte Passung Speicherplatz verschwendet.

Interne Fragmentierung:
: Verschwendung durch keine volle Ausnutzung eines Segments
(innerhalb des Segments)

Externe Fragmentierung:
: Verschwendung durch viele verteilte Speicherlöcher
(zwischen den Segmenten)

50%-Regel

Angrenzende Speicherlöcher können verschmolzen werden, Prozesse aber nicht. Dadurch sind über die Zeit nur halb so viele Löcher wie Prozesse vorhanden.

[Ungenutzte-Speicher-Regel]

Virtueller Speicher

Der adressierbare virtuelle Speicher übersteigt die Größe des physikalischen Hauptspeichers.

Der virtuelle Adressraum wird in gleich große *Seiten* unterteilt. Die Seiten entsprechen im realen Speicher *Seitenrahmen*.

Eine *Seitentabelle* zeigt an, welcher Bereich im virtuellen Adressraum (Seitenrahmen!) auf welchen Adressraum im Arbeitsspeicher verweist. Fehlt dort ein Eintrag (bzw. ist das *Absent-Bit* gesetzt), so befindet sich der Adressbereich nicht im Arbeitsspeicher und muss erst eingelagert werden. Der Zugriff auf die Seitentabelle selbst ist schnell, denn die Adresse kann direkt (über den Offset) angesprochen werden.

MMU (Memory Management Unit)

Die MMU wandelt die virtuellen in reale Adressen um – entweder direkt an den Arbeitsspeicher oder den Auslagerungsspeicher (➔Festplatte). Die MMU ist Teil der CPU und bedient den Speicherbus. Bei der Umsetzung wird die in die MMU eingehende virtuelle Adresse binär aufgeteilt. Die niederwertigen Bits (*Offset*) bestimmen die Position innerhalb einer Seite (eine Seite hat immer die Größe einer Zweierpotenz) und die höherwertigen Bits geben die virtuelle Seitennummer an.

Wird auf eine Seite über die MMU zugegriffen, die derzeit nicht im Arbeitsspeicher ist, erzeugt die MMU einen *Seitenfehler*, der das Betriebssystem veranlasst, die entsprechende Seite aus dem Auslagerungsspeicher in den Arbeitsspeicher einzulagern. (Diese Adresse steht nicht in der Seitentabelle sondern in anderen Softwaretabellen.) Falls kein Platz mehr für diese Seite ist, wird die am seltensten benötigte Seite ausgelagert (➔Seitenersetzungsalgorithmus).

Ist die Seite aber bereits im Arbeitsspeicher vorhanden, so werden bei der ausgehenden physikalischen Adresse lediglich die höherwertigen Bits gegen die physikalische Seitennummer ausgetauscht. Da die Position innerhalb der Seite gleich bleibt, wird der niederwertige Teil der Adresse nicht verändert.

Mehrstufige Seitentabellen

Selten ist der gesamte virtuelle Speicher belegt. Um nicht eine Tabelle für alle allokierten Seiten im Arbeitsspeicher halten zu müssen, verwendet man mehrstufige Tabellen. Die Tabelle der Hauptebene befindet sich immer im Arbeitsspeicher und verweist auf die Tabellen der zweiten Ebene. Ist eine Tabelle der zweiten Ebene zur Zeit nicht im Arbeitsspeicher vorhanden, wird sie erst eingelagert.

Die Adressierung der mehrstufigen Seitentabellen ist kaum schwieriger. Die höchsten Bits der Adresse sind der Index der Seitentabelle der ersten Stufe. Die nächsten Bits stehen für den Index der Seitentabelle der zweiten Stufe etc. Die niederwertigsten Bits geben wieder den Offset innerhalb der Seite an.

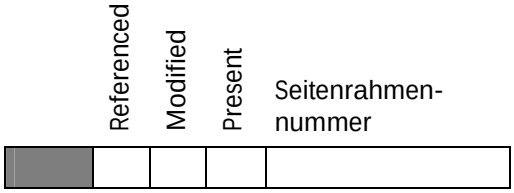


Tabelle 1: Eintrag in der Seitentabelle

Present (Anwesend):
: Ist die Seite im Arbeitsspeicher oder nur im Auslagerungsspeicher?

Modified (geschrieben):
: Wurde die Seite seit dem letzten Zugriff verändert? (Wenn nicht, kann bei einem Seitenfehler der Inhalt einfach verworfen werden, anstatt die Seite auszulagern.)

Referenced (gelesen oder geschrieben):
: Wann wurde zuletzt auf diese Seite zugegriffen? (Information für den Seitenersetzungsalgorithmus)

[Beispiele: PDP11…]

Assoziativspeicher

Der Assoziativspeicher ist ein „MMU-Cache“ zur Beschleunigung des Paging. Ist die angefragte Seite noch im Assoziativspeicher, so erübrigt sich der Zugriff auf die Seitentabelle.

Assoziativspeicher lohnt sich, da häufig Prozesse mehrfach hintereinander auf dieselben Speicherbereiche zugreifen.

Invertierte Seitentabellen

Bei großen Adressräumen (64 Bit = 20 Mio. Terabytes) baut man die Seitentabelle genau umgekehrt auf. Die invertierte Seitentabelle bildet den Hauptspeicher auf den virtuellen Speicher ab. Dadurch muss natürlich bei Zugriffen die gesamte Tabelle nach dem passenden Eintrag durchsucht werden.

Seitenersetzungsalgorithmen

Muss eine Seite eingelagert werden, aber es ist kein Platz im Arbeitsspeicher, dann wählt der Seitenersetzungsalgorithmus eine Seite aus, die dafür ausgelagert wird. Optimal wäre eine Auslagerung der Seite, die so lange wie möglich nicht mehr benötigt wird – dies zu berechnen ist aber leider nicht möglich. Diese Algorithmen sind Annäherungen:

NRU (Not Recently Used)

Der NRU-Algorithmus differenziert die Seiten in vier Kategorien:

Klasse 0: nicht referenziert, nicht modifiziert

Klasse 1: nicht referenziert, modifiziert

Klasse 2: referenziert, nicht modifiziert

Klasse 3: referenziert, modifiziert

Seiten haben ein M- (modified) und R- (referenced) Flag in der Seitentabelle. Bei jedem Zeit-Interrupt wird das R-Flag gelöscht – es wird also angezeigt, dass die Seite längere Zeit nicht mehr angesprochen wurde (so kommen auch Klasse-1-Seiten zustande). NRU wählt immer Seiten der niedrigsten Klasse als Kandidaten für die Auslagerung aus.

FIFO (First In First Out)

Der Algorithmus verwaltet eine eigene Tabelle mit den Seiten, in welcher Reihenfolge auf sie zugegriffen wurde. Es wird jeweils die Seite am Ende der Tabelle ausgelagert.

Nachteil: die älteste Seite der Liste kann trotzdem intensiv benutzt werden

Second Chance

Dieser Algorithmus ähnelt FIFO, beachtet aber auch das R-Flag. In der FIFO-Tabelle werden zuerst Seiten ausgelagert, bei denen das R-Flag nicht gesetzt ist. Bei jedem Durchlauf werden die R-Flags gelöscht.

Falls bei allen Seiten das R-Flag gesetzt ist, funktioniert Second-Chance exakt wie FIFO.

Seiten, die bei jedem Durchlauf benutzt werden, können also nicht ausgelagert werden und erhalten immer eine zweite Chance.

Uhr

Verhält sich wie „Second Chance“, rotiert allerdings nicht die eigene Seitenliste sondern verwaltet einen Zeiger auf die Position in der Seitenliste. Der „Uhrzeiger“ zeigt immer auf eine Seite, bei der das R-Flag nicht gesetzt ist.

LRU (Least Recently Used)

LRU lagert die Seite aus, die am längsten nicht mehr referenziert wurde. Um keine komplette sortierte Liste vorhalten zu müssen, gibt es drei Approximationen:

- Zähler (Hardware):
  Für jeden Seitentabelleneintrag gibt es ein 64-Bit-Feld. Ein Zähler wird bei jedem Zugriff inkrementiert und der Wert in der Tabelle gespeichert. Der niedrigste Wert entspricht der ältesten Seite.
- Matrix (Hardware):
  Es gibt eine n-mal-n Matrix (n = Anzahl Seiten in der Seitentabelle). Bei jedem Zugriff werden die Bits der Zeile auf 1 gesetzt, die Bits der Spalte dann auf 0. Der kleinste binäre Wert der Tabelle ist die älteste Seite.
- Shifting (Software):
  Jede Seite hat einen 8-Bit-Zähler. Bei jedem Uhr-Interrupt werden alle Bits aller Register nach rechts geshiftet und links eine 0 (kein Zugriff seit letztem Tick) oder 1 (Zugriff) eingefügt. Der geringste Wert gibt die älteste Seite an.

Modellierung von Paging-Algorithmen

Seitengröße

Kleine Seitengrößen reduzieren die interne Fragmentierung.

Große Seitengrößen sorgen für eine kleinere Seitentabelle.

Seitengröße hängt von der mittleren Prozessgröße und der Größe eines Seiteneintrags ab. In der Praxis liegt sie bei 1-2K.

Paging-Daemon

Der Paging-Daemon sorgt dafür, dass immer einige Seiten im Hauptspeicher frei sind, falls ein Seitenfehler auftritt.

[Modellieren von Paging-Algorithmen]

Segmentierung

Segmentierung ist die Aufteilung des (eindimensionalen) Adressraums in verschiedene Segmente, bei denen die Adressen jeweils wieder von 0 bis x gezählt werden. Die Adressierung erfolgt zweidimensional über die Segmentnummer und den Offset.

Vorteile:

- Die Segmente können wachsen und schrumpfen (wie es z.B. beim Compilieren passiert).
- Der Speicherverwalter kann den Speicher automatisch defragmentieren, da sich die Adressierung dadurch nicht ändert.
- Es können Schutzflags für jedes Segment (ähnlich rwx) gesetzt werden (Segmente können gemeinsam benutzt werden ➔shared libraries.)

(4) Dateisysteme

Dateien

Dateien haben drei Vorteile:

- *persistent* Speicherung von Daten
  (über die Laufzeit von Prozessen hinaus)
- Speicherung großer Datenmengen
  (mehr als ein Prozess Speicher reservieren kann)
- mehrere verschiedene Prozesse können auf diese Daten zugreifen
  (z.B. IPC durch Spooling)

Das Betriebssystem verwaltet Dateien über das *Dateisystem*. Der Programmierer erhält Zugriff zu den Dateien über bestimmte Systemaufrufe.

Dateitypen

Es gibt verschiedene Arten von Dateien im Dateisystem:

Reguläre Dateien:
: enthalten Bytesequenzen, denen nur vom passenden Anwendungsprogramm eine Bedeutung beigemessen wird (ASCII = Textdateien / Binär = sonstiges, ausführbare Programme werden durch eine Magic-Number gekennzeichnet)

Verzeichnisse:
: bringen hierarchische Strukturen ins Dateisystem

Zeichenorientierte Spezialdateien:
: modellieren Geräte, die byte-orientiert arbeiten (➔Terminals, Drucker)

Blockorientierte Spezialdateien:
: modellieren Geräte, in denen immer auf einen Datenblock zugegriffen wird (➔Festplatte)

Dateizugriff

Sequentieller Zugriff (*sequential access*):
: Eine Datei wird immer vom Anfang zum Ende gelesen. Die Datei kann lediglich „zurückgespult“ werden.
  (Angemessen für Magnetbänder.)

Direkter Zugriff (*random access*):
Auf Daten in Datei kann anhand der Position in der Datei zugegriffen werden. Die Position in der Datei kann mit *tell* abgefragt und mit *seek* festgelegt werden. (Angemessen für Platten.)

Dateiattribute

Dateiattribute werden automatisch mit der Datei zusammen gespeichert. Bei UNIX:

- uid des Besitzers
- gid des Besitzers
- Schutzflags (rwxst)
- Spezialflags (block/character)
- Größe
- Erstellungsdatum
- Änderungsdatum

Speicherabgebildete Dateien

Hier wird eine Datei in einen Speicherbereich (des *virtuellen* Speichers) eingeblendet (*MAP*) und kann mit normalen Speicherzugriffen bearbeitet werden. Beim Zugriff werden Seitenfehler ausgelöst. Beim Beenden der Abbildung (*UNMAP*) werden alle eingelagerten Seiten zurückgeschrieben.

Dabei kann die Größe einer Datei allerdings nicht verändert werden. Beim Versuch käme es zu einem Zugriffsfehler.

Verzeichnisse

Hierarchische Verzeichnissysteme

Zur Strukturierung des Dateisystems werden Verzeichnisse (spezielle Dateien) verwendet. Jedem Benutzer wird automatisch ein Home-Verzeichnis zugeordnet, in dem er persönliche Dateien speichern kann.

Auf ein Verzeichnis oder eine Datei wird mit Hilfe des Pfadnamens zugegriffen, der den Pfad in der Baumstruktur des Dateisystems angibt.

Pfadnamen

Der *absolute Pfad* wird immer vom Hauptverzeichnis aus gerechnet. Der *relative Pfad* bezieht sich auf ein Verzeichnis relativ zum aktuellen Arbeitsverzeichnis (*cwd* = current working directory). Jedem Prozess ist ein eigenes Arbeitsverzeichnis zugeordnet.

Zudem gibt es zwei spezielle Verzeichnisse in jedem Verzeichnis: „." steht für das aktuelle Verzeichnis und „.." für das übergeordnete Verzeichnis (*parent directory*).

Dateisystem-Implementation

Dateien

Kontinuierliche Allokation:
Eine Dateien wird als ein zusammenhängender Datenblock auf der Festplatte angelegt. (Wie Allokation im Hauptspeicher.)
Vorteil: einfache Implementation
Nachteil: Fragmentierung, Größe der Datei muss bei der Erzeugung bekannt sein, Defragmentierung auf Platte deutlich langsamer als im Hauptspeicher.

Verkettete Listen:
Jeder Block der Datei zeigt auf einen Folgeblock.
Vorteil: keine Fragmentierung
Nachteil: langsamerer Zugriff

Verkettete Listen mit Index:
Im Speicher wird eine Tabelle der Blöcke mit einem Verweis auf den jeweils nächsten Block (der Datei) gehalten.
Vorteil: keine Fragmentierung; schnellerer Zugriff, wenn nicht die gesamte Datei durchlaufen werden muss
Nachteil: Overhead

I-Nodes (*index nodes*):
Einige Zeiger auf Blöcke der Datei werden im ersten i-node-Block gespeichert. Bei größeren Dateien wird auf weitere (indirekte) i-nodes verwiesen. Eine sehr kleine Datei besteht also aus der i-node und einem Datenblock (= 2 Blöcke).
➔UNIX

Verzeichnisse

Das Verzeichnissystem verwaltet eine Zuordnung der Dateinamen (mit Pfad) auf die i-nodes. In den i-nodes werden auch Informationen über das Verzeichnis gespeichert.

Um eine Datei in einem komplexeren Pfad zu lokalisieren, werden die einzelnen Verzeichnisse nach dem gewünschten Unterverzeichnis in der i-node durchsucht bis die Datei gefunden wurde.

/usr/local/bin wird zu /usr ➔ local ➔ bin

Links

Eine Datei (oder ein Verzeichnis) kann durch Links auch von anderen Pfaden referenziert werden. Die Datei ist physikalisch nur einmal vorhanden, kann aber von zwei Pfade aus angesprochen werden. Links führen zu Zyklen im Baum.

Symbolischer Link:
Eine Datei wird erzeugt und als Link markiert, die lediglich den Pfad der eigentlichen Datei als ASCII enthält. Die Datei kann also gefahrlos

gelöscht werden – die Datei im Link wäre einfach nicht mehr vorhanden.

Hard Link:
: Die eigentliche Datei enthält einen Link-Zähler in der i-node, der beim Erzeugen eines neuen Links inkrementiert wird. Der Link zeigt direkt auf die i-node der gewünschten Datei. Soll die Datei gelöscht werden, so kann dies nur erlaubt werden, wenn der Link-Zähler 1 ist.

Plattenplatz-Verwaltung

Bei kleinen Blockgrößen wird zwar weniger Plattenplatz verschwendet (interne Fragmentierung), dafür kommt es zu höheren Zugriffszeiten. Bei großen Blockgrößen wird viel Plattenplatz verschwendet, dafür sind die Zugriffszeiten geringer. Gängige Blockgrößen sind 512 oder 1024 Bytes.

Der freie Plattenplatz kann über verkettete Listen oder eine Bitmap verwaltet werden. Bei verketteten Listen enthält jeder Block 511 Nummern von freien Blöcken (bei einer Blockgröße von 1024 Byte und 2 Bytes pro Eintrag). Die Bitmap benötigt meistens weniger Speicherplatz (außer wenn fast keine freien Blöcke mehr vorhanden sind).

Beim Einsatz von Quotas darf ein Benutzer nur eine festgelegte Datenmenge speichern. Für jeden Benutzer wird der aktuelle Platzverbrauch festgehalten. Beim Erreichen des *Soft-Limits* wird eine Gnadenfrist (*grace period*) festgelegt. Ist diese Frist verstrichen, wird der Benutzer aus dem System bis zur Freigabe durch den Systemadministrator ausgeschlossen. Wird das höhere *Hard-Limit* erreicht, werden sofort keine Schreibzugriffe mehr erlaubt.

Zuverlässigkeit

Platten oder Disketten enthalten von der Produktion oder durch Abnutzung fehlerhafte Blöcke. Bei Festplatten wird eine Liste der defekten Blöcke vom Hersteller auf der Platte abgespeichert. Entweder verwaltet das Dateisystem eine Liste der defekten Blöcke mit einer Abbildung auf Alternativblöcke oder es wird eine Pseudodatei angelegt, die diese defekten Blöcke allokiert.

Es ist immer ratsam, eine Sicherheitskopie des Dateisystems zu halten. Dazu wird meist ein Level-0-Dump auf Band gemacht, auf das inkrementell alle Änderungen gesichert werden. Bei Hochverfügbarkeitssystemen werden auch RAID-Arrays eingesetzt, die im besten Fall auf zwei Platten dieselben Inhalte verwalten, falls eine Platte ausfällt.

Konsistenz

Stürzt das System während des Betriebs ab, können nicht-gespeicherte Daten verloren gehen. Dienstprogramme können dann das Dateisystem wieder in einen konsistenten Zustand bringen.

Dazu werden zwei Zähler pro Block angelegt, die zählen, wie oft der Block in einer Datei bzw. in der Liste der freien Blöcke vorhanden ist. Nach der Untersuchung aller i-nodes darf der Block nur entweder im ersten oder zweiten Zähler eine 1 haben, sonst ist das Dateisystem inkonsistent.

Steht in beiden Listen eine 0, so ist es ein *verlorener Block*. Er kann wieder freigegeben werden.

Steht in beiden Listen eine 1, so wird der Block aus der Freiliste entfernt.

Steht in der Freiliste an einer Stelle ein Wert größer 1, so wird der Wert auf 1 gesetzt.

Steht in der Belegungsliste an einer Stelle ein Wert größer 1, so ist die entsprechende Datei verloren. Man kann zwar den Block duplizieren, aber eine Datei ist in jedem Fall irreparabel.

Performance

Cache:
: Speicherbereich, in dem aus Performancegründen Daten der Festplatte gehalten werden.

Bei jedem Plattenzugriff wird geprüft, ob der Datenblock bereits im Cache vorhanden ist. Falls nicht, wird der Block zuerst in den Cache geladen. Das Verwerfen von Cache-Einträgen verläuft ähnlich wie beim Paging (NRU). Allerdings behandelt der Caching-Algorithmus das Schreiben von Blöcken priorisiert, die der Konsistenz des Dateisystems dienen. Unter UNIX läuft zusätzlich alle 30 Sekunden ein *update*-Kommando ab, das den Write-Cache zurückschreibt.

Zur weiteren Beschleunigung können häufig angesprochene Blöcke (z.B. i-nodes) über die Platte verteilt werden, damit der Lesekopf möglichst kürzere Strecken zurücklegen muss. Es können auch zusammenhängende Blöcke mit *Interleave* geschrieben werden, falls der Lesekopf keinen zusammenhängenden Datenstrom von der Magnetoberfläche lesen kann.

Interleave:
: Faktor, um den versetzt die Blöcke auf einer Spur auf Festplatte geschrieben werden. Dadurch kann der Zugriff auf Blöcke beschleunigt werden, wenn der Controller nicht in Echtzeit Blöcke lesen kann sondern nach jedem Block eine Zeit braucht. Es müsste sonst fast eine Rotationszeit gewartet werden, bis der nächste Block unter dem Kopf ist.
(Interleave ist ein veraltetes Konzept!)

Schutzmechanismen

Schutzdomänen

Unter UNIX werden die uid und gid benutzt, um die Rechte auf Ressourcen (Dateien/Geräte) zu bestimmen. Besitzt eine Datei das SETUID/SETGID-Bit, so ändern sich uid und gid des Prozesses, der diese Datei ausführt. Außerdem werden alle Systemaufrufe im

Kernmodus (*kernel mode*) ausgeführt, was einen vollen Zugriff auf alle Ressourcen impliziert.

Zugriffskontrolllisten (ACLs)

ACLs benötigen weniger Speicherplatz als Schutzmatrizen. In UNIX werden pro Datei die RWX-Bits als Schutz verwendet. Nur der Eigentümer kann diese ACL ändern.

[Capabilities]

[Schutzmodelle]

Verdeckte Kanäle

Selbst wenn sorgfältig ACLs gesetzt werden, können Prozesse verdeckt miteinander kommunizieren (Veränderung der CPU-Auslastung, systematisches Blockieren von Dateien…).

(5) Eingabe / Ausgabe

Prinzipien der Hardware

I/O-Geräte

Block devices modellieren Geräte, deren kleinste zugreifbare Datenmenge ein Block (eine Menge von Bytes) ist (➔Festplatte, Floppy).
Character devices modellieren Geräte, die Zeichenströme (à 1 Byte) bearbeiten (➔Drucker, Terminal, Netzwerkkarte, Bandlaufwerk).

Nicht alle Geräte passen in dieses Schema (➔Uhren, die nur Interrupts auslösen).

Geräte-Controller

Der *Controller* ist ein elektronischer Teil des Geräts und übernimmt die Ansteuerung des mechanischen Teils. Dazu blendet er seine Register in einen Teil des Arbeitsspeichers ein (*memory-mapped I/O*). Das Betriebssystem schreibt Befehle und Parameter in diesen Bereich. Nach der Durchführung des Befehls löst das Gerät einen Interrupt aus und das Betriebssystem kann das Ergebnis (➔Auslesen eines Datenblocks von Festplatte) aus den Registern auslesen.

DMA (Direct Memory Access)

DMA erlaubt es blockorientierten Geräten, den Arbeitsspeicher ohne (verschwenderische) Interaktion der CPU zu beschreiben.

Das System weist dem Gerätecontroller einen Speicherbereich (Anfang und Größe) zu, in dem die zu holenden Daten abgelegt werden sollen. Nachdem der Datenblock in den Arbeitsspeicher kopiert wurde, wird ein Interrupt ausgelöst. Das Betriebssystem kann direkt mit den neuen Daten arbeiten.

Wichtig: aus Timing-Gründen wird trotzdem ein eigener interner Gerätepuffer zum Zwischenspeichern benutzt, um alle Daten zu sammeln, bevor sie in den Speicher kopiert werden.

Eigenschaften der I/O-Software

Ziele der I/O-Software

Geräteunabhängigkeit (*device independence*):
: Programme können auf beliebige Geräte gleichartig zugreifen ohne sich um die Eigenheiten der Kommunikation kümmern zu müssen (➔sort <input >output).

Gleiche Bezeichnungen (*uniform naming*):
: Geräte werden durch Namen bezeichnet, die für alle Geräte gleichartig sind.

Fehlerkorrektur:
: Untere Hardwareebenen sollten möglichst automatisch Fehler korrigieren (➔erneutes Lesen eines Blocks bei einem transienten Fehler). Erst wenn Fehler nicht ausgeglichen werden können, soll die Software davon in Kenntnis gesetzt werden.

Synchroner Transfer:
: für eine synchrone Datenübertragung muss das Gerät dediziert blockiert (➔blocked) werden

Asynchroner Transfer:
: Daten werden übertragen und dann ein Interrupt ausgelöst

Dedizierte Geräte (*dedicated devices*):
: können nur von einem Prozess zur Zeit benutzt werden (➔Drucker)

Gemeinsame Geräte (*shared devices*):
: können von mehreren Prozessen gleichzeitig benutzt werden (➔Festplatte)

1	Interrupt-Handler Treiber entblockieren, wenn Eingabe
2	Gerätetreiber Kommunikation mit dem Gerät über Register, Statusüberwachung
3	Geräteunabhängiges Betriebssystem Schutz, Blockierung, Pufferung, Allokation
4	Benutzerprozesse Aufruf von Betriebssystemprozeduren, Spooling

Tabelle 2: Schichten des I/O-Systems

Interrupt-Handler

Sobald ein Gerät eine Operation beendet hat, erzeugt es einen Interrupt. Dieser startet den Interrupt-Handler und reaktiviert so den wartenden (blockierten) Gerätetreiber.

Gerätetreiber

Gerätetreiber enthalten die Software zur Ansteuerung eines bestimmten Geräts. Sie übernehmen die Kommunikation mit dem Gerät über die Register. Das Betriebssystem stellt eine geräteunabhängige Anfrage (➔lies Block x) an den Treiber, der diesen für das Gerät passend umsetzen muss. Kann ein Gerät nicht in Echtzeit antworten, so wird der Treiber blockiert (➔blocked) und durch einen Interrupt wieder aktiviert, sobald die Aufgabe erledigt wurde.

Geräteunabhängiges Betriebssystem

Das Betriebssystem stellt abstrakte geräteunabhängige Anfragen an den Treiber. Unter UNIX identifiziert er anhand des verlangten Gerätenamens (➔/dev/ttyS3) die *major device number* (die den verwendeten Treiber angibt) und die *minor device number* (die als Parameter für den Treiber das gewünschte Gerät angibt) und ruft den Treiber auf.

Die Benutzung der Geräte wird (unter UNIX) durch rwx-Flags gesteuert.

Außerdem müssen Ein- und Ausgaben gepuffert werden. Soll ein Byte in einer Datei geändert werden, so muss der gesamte Block eingelesen, gepuffert, ein Byte verändert und der Block wieder gespeichert werden. Auch Tastatureingaben werden gepuffert.

Benutzerprozesse

Viele Routinen für die Ein- und Ausgabe sind in einer Standard-I/O-Library zusammengefasst und werden beim Compilieren zum Programm gelinkt.

Spooling:
: wird eingesetzt, wenn mehrere Prozesse auf ein dediziertes Gerät zugreifen möchten. Ein *Daemon* (➔lpd = line printer daemon) überwacht den Inhalt des *Spooling-Verzeichnisses*. Die Benutzerprozesse legen Dateien dort ab, die dann vom Daemon nacheinander an das entsprechende Gerät geschickt werden.
Dies ist nur unkritisch, wenn eine Verzögerung akzeptabel ist.
Auch bei Netzwerkübertragungen im USENET ist Spooling (über UUCP) gängig.

Festplatten (*disks*)

Festplatten haben drei Vorteile:

- Hohe Speicherkapazität
- Geringer Preis je Bit
- Persistente Speicherung

Hardware

Platten bestehen aus Zylindern, auf denen die Spuren gespeichert werden. Für jede Spur ist ein Schreib-/Lesekopf zuständig. Die Spuren werden weiter in gleichartige Sektoren (8-32) unterteilt.

Plattenarm-Scheduling-Algorithmen

Die Zugriffszeit auf einen Block ist von folgenden Faktoren abhängig:

- Suchzeit (*seek time*):
  Zeit, bis sich der Lesekopf über den richtigen Zylinder bewegt hat
- Umdrehungsverzögerung (*rotational delay*):
  Zeit, bis sich der richtige Sektor unter den Kopf gedreht hat
- Übertragungszeit (*transfer time*):
  Zeit, bis der Block bis zum Ende eingelesen wurde

In der Praxis entspricht die Rotationsverzögerung etwa der Suchzeit.

Standardmäßig werden Blöcke nach FCFS gelesen. Um den Zugriff auf Blöcke zu optimieren, verwalten Platten eigene verkettete Listen auf die angeforderten nachfolgenden Blöcke. Algorithmen können mit diesen Informationen die Suchzeiten optimieren.

FCFS (First Come First Served):
: Liest die Blöcke in der angefragten Reihenfolge. Einzig möglicher Algorithmus, wenn die Anfragen nacheinander kommen.

Shortest Seek First (SSF):
: Bedient den jeweils nächstgelegenen Zylinder. Nachteil: unfair, denn bei vielen Anfragen könnten Zylinder am äußeren Rand möglicherweise nie angesprochen werden.

Fahrstuhl-Algorithmus:
: Anfragen werden immer nur in einer Richtung bearbeitet. Erst werden alle Anfragen bis zum äußersten Zylinder nacheinander sortiert ausgeführt. Danach wechselt der Kopf die Richtung und bearbeitet alle Anfragen nacheinander bis zum innersten Zylinder.

In jedem Fall können die verschiedenen Köpfe ohne Wartezeit umgeschaltet werden, um Blöcke von verschiedenen Spuren auf demselben Zylinder zu lesen.

RAID (Redundant Array of Inexpensive Disks):
: Daten werden über mehrere Platten verteilt gespeichert (z.B. ein Bit je Platte von jedem Byte). Im Falle eines Plattenausfalls könnte man durch fehlerkorrigierende Hamming-Codes den Fehler ausgleichen. Außerdem sind in der Praxis mehrere kleinere Platten meist billiger als eine große.

Fehlerbehandlung

Folgende Fehler können bei Platten auftreten:

- Programmierfehler
  Zugriffe auf nichtexistente Blöcke, Zylinder, Spuren, Köpfe oder Speicherbereiche
- Transiente Prüfsummen-Fehler
  Durch physikalische Störungen konnte ein Block nicht einwandfrei gelesen werden. Das Wiederholen der Operation könnte den Fehler beheben.
- Permanente Prüfsummen-Fehler
  Konnte der Fehler nicht durch Wiederholung der Operation behoben werden, muss der Block als defekt markiert werden.
- Such-Fehler
  Durch mechanische Probleme (➔Erwärmung) kann der Kopf auf eine falsche Spur zeigen. Durch eine Rekalibrierung wird dieser Fehler meist behoben.
- Controller-Fehler
  Controller sind kleine spezialisierte Computer, die natürlich auch Fehler enthalten können. Im Fehlerfall setzt sich der Controller zurück und wiederholt die letzte Operation.

Spur-Caching

Einige Treiber lesen beim Zugriff auf einen Sektor gleich die ganze Spur in einen Cache-Speicher ein, da dies kaum mehr Zeit in Anspruch nimmt. Bei Zugriffen auf Daten im Cache, muss dann allerdings die CPU die Daten kopieren (kein DMA).

RAM-Disk

Die Struktur eines Block-Gerätes wird auf den Arbeitsspeicher projiziert. Durch spezielle Zugriffsmethoden können dann „Blöcke“ aus dem Arbeitsspeicher gelesen oder geschrieben werden.

Uhren (*timers / clocks*)

Uhren verwalten die aktuelle Uhrzeit und werden vom Prozess-Scheduler für die Prozessumschaltung verwendet. Die Uhr wird als Device angesprochen (aber nicht block- oder zeichenorientiert).

Hardware

Uhren orientieren sich entweder an der Netzfrequenz (50/60 Hz) oder an einem Quarz-Oszillator. Dabei schwingt der Quarz mit 5 bis 100 MHz und die Schwingungen werden an einen Zähler übermittelt. Ist der Zähler 0, dann wird ein Zeit-Interrupt ausgelöst (*„tick“*).

Software

Die Zeit-Software kümmert sich um diese Aufgaben:

- Realzeit
- Prozess-Umschaltung
- Alarme von Benutzerprozessen
- Watchdog-Timer

Die reale Uhrzeit (batteriegepuffert) wird bei jede Sekunde (nach einer Anzahl von „ticks“) hochgezählt. Die Anzahl Sekunden werden ab dem 1.1.1970 gerechnet. Mit einem 32-Bit-Register (signed integer!) lassen sich damit Zeiten bis ins Jahr 2038 abbilden.

Für die Prozess-Umschaltung wird dem laufenden Prozess das Quantum zugeteilt und ein anderer Prozess beim Ablaufen dieser Zeit ausgewählt.

Watchdog-Timer werden vom Betriebssystem verwendet, wenn z.B. der Motor der Floppy nach einige Zeit abgeschaltet werden soll. Watchdog-Timer haben immer eine Callback-Routine.

Terminals

Hardware

Es gibt zwei Kategorien von Terminals:

- Serielle Terminals (RS-232)
- Speicherbasierte Terminals (*memory mapped*)

RS-232-Terminals bestehen aus einer Tastatur, einem Bildschirm und einer seriellen Schnittstelle, die Daten bitweise überträgt (je ein Pin für Senden, Empfangen und Erdung). Gängige Übertragungsraten sind 1200, 2400 und 9600 bps.

Die Umwandlung von Bits zu Zeichen (8-Bit) übernimmt der UART (*Universal Asynchronous Receiver Transmitter*). Je nach Größe des UART-Puffers können mehrere Zeichen gepuffert werden, die dann automatisch geshiftet übertragen werden.

Hardcopy-Terminals senden Tastatureingaben an den Computer und drucken die Ausgabe direkt aus (➔Fernschreiber).

Bildschirm-Terminals (*CRT=cathode ray tube*) geben hingegen Ausgaben auf einem Bildschirm aus. Man nennt sie auch TTY (teletype).

Speicher-Terminals

Speicher-Terminals kommunizieren nicht seriell mit dem Computer sondern sind ein direkter Bestandteil des Systems. Sie haben ein Video-RAM (als Teil des adressierbaren Arbeitsspeichers) und einen Video-Controller, der das Video-RAM ausliest und daraus ein Videosignal erzeugt. Im Video-RAM werden entweder ASCII-Zeichen oder direkt eine Bitmap der Pixel gespeichert.

Eingabe-Software

Der Tastatur-Treiber sammelt Eingaben von der Tastatur und gibt sie an das Benutzerprogramm weiter. Im *raw-mode* wird jeder einzelne Tastendruck als ASCII-

Zeichen an das Programm gesendet. Im *cooked-mode* erhält das Programm eine ganze Zeile, bei der Steuerzeichen wie Backspace bereits interpretiert wurden.

Alle Eingaben werden gepuffert, damit der Anwender weiter tippen kann, auch wenn das Benutzerprogramm noch nicht bereit ist. Das *Echoing* der Eingaben sorgt dafür, dass jeder Tastendruck direkt auf dem Bildschirm quittiert wird. Es kann wahlweise abgeschaltet werden, wenn z.B. ein Passwort blind eingegeben werden soll.

Der Tastatur-Treiber kann auch Escape-Sequenzen (➔CTRL-S/-Q zum Anhalten der Ausgabe) verarbeiten oder Signale an Prozesse schicken (➔CTRL-C zum Senden eines SIGINT).

Ausgabe-Software

Bei RS-232-Terminals werden Ausgaben vor der Ausgabe gepuffert (Interrupt). Bei Speicher-Terminals werden Ausgaben direkt ins Video-RAM geschrieben (kein Interrupt).

Die Ausgabe-Software kümmert sich auch um die Cusor-Position, erweiterte Ausgaben (➔CTRL-G für die „Klingel“), die Interpretation von Tabulatoren und das Scrolling.

(6) Deadlocks

In vielen Anwendungen benötigt ein Prozess exklusiven Zugriff auf ein unteilbares Betriebsmittel. Wenn der Prozess bereits ein Betriebsmittel belegt hat, auf die ein anderer Prozess wartet (und umgekehrt), dann kommt es zu einer Verklemmung (Deadlock).

Betriebsmittel

Betriebsmittel sind Hardwaregeräte oder Informationen, auf die ein Prozess für die Erfüllung seiner Aufgabe zugreift und die nur einem Prozess zur Zeit zur Verfügung stehen.

Entziehbare Betriebsmittel:

…können einem Prozess ohne negative Folgen entzogen werden (➔Arbeitsspeicher)

Exklusive Betriebsmittel:

…dürfen einem Prozess nicht entzogen werden, weil sonst Probleme entstehen (➔Drucker)

Deadlocks (Verklemmungen)

Deadlocks können nur bei exklusiven Ressourcen auftreten, denn bei entziehbaren Ressourcen kann das Betriebssystem eingreifen.

Eine Menge von Prozessen ist verklemmt (*deadlocked*), wenn jeder Prozess auf ein Ereignis wartet, das nur ein anderer Prozess auslösen kann.

Ein verklemmter Prozess bekommt meist nicht einmal die Gelegenheit, eine andere Ressource wieder freizugeben, um den Deadlock zu beheben.

Bedingungen für einen Deadlock

Vier Bedingungen müssen erfüllt sein, bevor es zu einem Deadlock kommen kann.

- Wechselseitiger Ausschluss:
  Nur ein Prozess zur Zeit darf eine Ressource belegen.
- Belegen-und-Warten:
  Prozesse, die bereits Ressourcen belegen, dürfen weitere Ressourcen anfordern.
- Ununterbrechbarkeit:
  Eine einmal vergebene Ressource darf dem Prozess nicht gewaltsam entzogen werden.
- Zyklisches Warten:
  Es muss zu einem Zyklus von mindestens zwei Prozessen kommen, die jeweils auf eine Ressource warten, die von einem anderen Prozess im Zyklus freigegeben werden muss.

Wenn nur eine Bedingung nicht zutrifft, kann es nicht zu Deadlocks kommen.

Modellierung von Deadlocks

Deadlocks modelliert man mit gerichteten Graphen (ähnlich wie Petrinetze). Prozesse sind Kreise und Betriebsmittel sind Quadrate.

Ein Pfeil von einem Betriebsmittel zu einem Prozess bedeutet, dass der Prozess dieses Betriebsmittel erfolgreich belegt hat und noch hält.

Ein Pfeil von einem Prozess zu einem Betriebsmittel zeigt den Versuch an, ein Betriebsmittel zu belegen.

Enthält der Graph einen Zyklus, so liegt ein Deadlock vor.

Ignorieren von Verklemmungen

Verklemmungen werden nicht vom Betriebssystem behandelt (UNIX). Es liegt in der Hand des Programmierers.

Erkennung von Deadlocks

Erkennung

Das Betriebssystem verhindert keine Deadlocks. Es kann sie aber erkennen. Dazu wird der Betriebsmittelgraph aufgebaut und nach einem Zyklus gesucht.

[Bei mehreren Betriebsmitteln von jedem Typ]

Behebung

Bei entziehbaren Betriebsmitteln ist die Behebung einfach. Das Betriebsmittel wird dem nächsten Prozess zugewiesen.

Wenn Deadlocks regelmäßig auftreten, kann das Betriebssystem den Zustand (mit derzeit belegten Betriebsmitteln) eines Prozesses in regelmäßigen Abständen speichern. Kommt es zu einem Deadlock, wird ein älterer Stand eingespielt und der Prozess muss auf die Ressource warten (➔Backtracking).

Ansonsten muss der Prozess abgebrochen werden. Bei manchen Prozessen (➔make) ist es unkritisch, den Prozess noch einmal zu starten.

Verhinderung von Deadlocks

[Betriebsmittel-Flugbahnen]

Sichere und unsichere Zustände

Es wird ein Residuum berechnet und die sicheren und unsicheren Zustände werden markiert. Kommt das System in einen unsicheren Zustand, so liegt noch nicht direkt ein Deadlock vor – bei bestimmten Betriebmittelanforderungen kann es aber zu einem Deadlock kommen. Das System muss unsichere Zustände vermeiden.

Beispiel: Bankiersalgorithmus

Beispiel zur Problematik von Deadlocks und deren Umgehung. Der Bankier verleiht verschiedene Beträge aus seinem Kapital an seine Kunden. Ein Kunde muss den Kredit auf einmal nach endlicher Zeit zurückzahlen. Damit kann es für den Bankier zu Deadlocks führen, da er Teilbeträge bereits ausgegeben haben kann, aber noch kein Kunde den vollen Kredit erhalten hat und auch nicht verpflichtet ist ihn zurückzuzahlen.

Hier empfiehlt es sich im Voraus das Residuum zu berechnen und unsichere Zustände zu vermeiden.

Für Betriebssysteme ist der Bankiersalgorithmus bedeutungslos, da kaum ein Prozess von Anfang an seine Betriebsmittelbedürfnisse kennt.

[Bankiersalgorithmus für mehrere Ressourcen]

4 Bedingungen verhindern

Wenn eine der vier Bedingungen für eine Verklemmung nicht zutrifft, kann es zu keiner Verklemmung kommen.

- **Kein „Wechselseitiger Ausschluss“**
  Es darf nur ein Prozess im System eine Ressource belegen. Beim könnte Spooling eingesetzt werden. Nur der lpd würde dann auf den Drucker zugreifen.
- **Kein „Belegen-und-Warten“**
  Falls ein Prozess weiß, welche Betriebsmittel er benötigt, kann er sie am Anfang der Ausführung belegen.
  Ist dies nicht bekannt und schlägt die Belegung eines weiteren Betriebsmittels fehl, werden alle Betriebsmittel freigegeben. (Zwei-Phasen-Belegung)
- **Keine „Ununterbrechbarkeit“**
  Dem Prozess ein Betriebsmittel zu entziehen, ist praktisch unmöglich.
- **Kein „Zyklisches Warten“**
  Wenn alle Ressourcen durchnummeriert werden, belegen alle Prozesse die benötigten Ressourcen in derselben Reihenfolge.

Zwei-Phasen-Belegung

Hierbei werden erst alle Ressourcen angefordert (und sonst die Anforderung komplett abgebrochen) bevor mit den Ressourcen gearbeitet wird.

Verschiedenes

Nicht-Betriebsmittel-Deadlocks

Deadlocks können u.a. auch bei Semaphoren auftreten, wenn z.B. zwei Prozesse einen DOWN ausführen, wenn der Semaphor schon 0 ist.

Verhungern

Wenn Strategien über Betriebsmittelvergaben entscheiden, kann es zum Verhungern kommen, wenn ein Prozess (➔der eine besonders lange Datei drucken möchte) bei einem ausgelasteten System aufgrund einer unfairen Strategie immer weiter ignoriert wird.

(7) Fallstudie: UNIX

UNIX ist das am weitesten verbreitete Betriebssystem für Workstations und Server.

Übersicht über UNIX

UNIX ist ein interaktives Timesharing-Betriebssystem. Es wurde primär für „eingeweihte“ Programmierer entwickelt.

Login

Der Zugang zu UNIX ist durch eine User/Passwort-Authentisierung geschützt. Diese Identifikation wird auch für die Autorisierung für Zugriffe auf Dateien verwendet.

Usernamen und Passwörter (verschlüsselt) werden in einer Passwortdatei gespeichert. Jedem User entspricht eine UID (*user identification*) und eine GID (*group identification*).

Shell

Nach dem Login wird die Shell (Kommandointerpreter) gestartet, in der der Anwender Befehle eingeben kann. Die Shell interpretiert Wildcards („*“ und „?“), ermöglicht Pipelines (*sort | head -10*) und verwaltet Hintergrundprozesse (*sort <x >y &*).

Dateien und Verzeichnisse

Dateien werden durch Dateinamen bis zu 255 Zeichen beschrieben. Ein 9-Bit-Eintrag beschreibt die Zugriffsrechte auf eine Datei (*rwxrwxrwx = 777 oktal*).

Es werden bis zu dreifach indirekte i-nodes verwendet.

Speicher

Jeder Prozess besteht aus Textsegment, Datensegment und Stacksegment.

Prozesse

Prozesse werden vom init-Prozess aus baumartig gestartet. Es gibt einen fork-Aufruf.

Scheduling

Zweistufiges Prioritätsscheduling.

Swapping

Es werden bevorzugt seit längerer Zeit blockierte (auf I/O wartende) Prozesse ausgelagert.

Seitenersetzungsalgorithmus

Der page-Daemon prüft regelmäßig, ob mindestens ein Viertel des Hauptspeichers frei für das Einlagern von Speicherseiten ist. Wenn nicht, werden Seiten ausgelagert.

Ausgewählt werden die Seiten nach dem Uhr-/Second-Chance-Algorithmus.

II Verteilte Systeme

(1) Einführung

Vorteile verteilter Systeme

- Wirtschaftlichkeit
  (mehrere langsame CPUs sind günstiger als eine schnelle)
- Parallelverarbeitung
  (höhere Rechenleistung, Lastausgleich)
- Skalierbarkeit
  (weitere Systeme können hinzugenommen werden, ohne dass man alles wegwerfen muss)
- Ausfallsicherheit
  (fällt eine CPU aus, kann das System trotzdem noch laufen)
- Kooperative Arbeit
  (Zugriff mehrerer Benutzer auf dieselben Daten, Email)

Nachteile

- Netzwerk
  (Nachrichten können im Netzwerk verlorengehen)
- Datenschutz
  (Schutzmechanismen bei Netzzugriffen)

(2) Hardware-Konzepte

Klassifikation nach Flynn

SISD (*single instruction, single data*):
: Zentrales System
(ein Prozessor führt jeweils einen Befehl aus)

SIMD (*single instruction, multiple data*):
: Supercomputer
(mehrere Prozessoren führen einheitliche Befehle aus)

MIMD (*multiple instruction, multiple data*):
: Verteilte Systeme

Multiprozessorsysteme:
: eng gekoppelte MIMD-Systeme mit gemeinsamem Speicher
(bus-basiert ➔ Sun Enterprise, Dual-CPU)
(geswitcht ➔ Supercomputer)

Multicomputersysteme:
: lose gekoppelte MIMD-Systeme mit jeweils privatem Speicher
(bus-basiert ➔ PCs im LAN)
(geswitcht ➔ Transputer)

Bus-basierte Multiprozessorsysteme

Koheränz:
: Gewährleistung eines einheitlichen Speicherinhalts bei mehreren Prozessoren

Um den Speicherbus zu entlasten, kann ein Prozessor-Cache benutzt werden, der Speicherzugriffe zwischenspeichert.

Problem: nicht kohärent / aufwendig zu programmieren

Semantiken

Write-Through-Cache:
: Prozessorcache, der Schreibvorgänge direkt und ohne Caching in den Arbeitsspeicher zurückschreibt.

Snoopy-Cache:
: Der Cache überwacht den Speicherbus auf Schreibvorgänge und aktualisiert seine gemerkten Informationen.

Geswitchte Multiprozessorsysteme

Bei vielen Prozessoren (>64) werden *crossbar-switches* eingesetzt, die über ein Gitter Prozessoren mit Speichermodulen verbinden. So können verschiedenen Prozessoren gleichzeitig über die Knotenpunkt *(crosspoint switches)* auf verschiedene Speichermodule zugreifen.

Um das Gitter zu verkleinern werden *Omega-Netzwerke* eingesetzt, bei denen statt Kreuzschaltern 2x2-Schalter eingesetzt werden, die über einen Bus Prozessoren und Speichermodule miteinander verbinden. Bei 1024 Prozessoren wären dann 10 Schalterstufen nötig (binär-logarithmisch).

Bus-basierte Multicomputersysteme

Eigener Speicher, Koppelung über LAN-Strecken im 100 Mbit/s-Bereich.

Geswitchte Multicomputersysteme

Prozessoren haben hier immer einen eigenen Speicher.

Gitter eignen sich für zweidimensionale Probleme.

Hypercubes sind n-dimensionale Würfel, in denen jeder Prozessor n Verbindungen zu anderen Prozessoren hat. Es sind immer nur benachbarte Prozessoren miteinander verbunden, so dass weitere Strecken geschaltet werden müssen. Der Schaltweg ist aber nur logarithmisch zur Anzahl der Prozessoren. Hypercubes verbinden ca. 10.000 Prozessoren.

(3) Software-Konzepte

Netzwerkbetriebssysteme und NFS

Clients (Workstations) greifen direkt auf Dateien auf einem Datei-Server zu. Die vom Server exportierten Verzeichnisse können auf dem Client-Rechner ins Dateisystem eingebunden werden. NFS ist hard- und softwaremäßig lose gekoppelt.

Netzwerkbetriebssystem:
: Betriebssystem für autonome Rechner, die aber über ein Netzwerk miteinander kommunizieren können.

Netzwerkbetriebssysteme kommunizieren nur über Datei-Sharing miteinander.

NFS-Architektur

Der Server stellt in /etc/exports eine Liste der freigegebenen Dateibäume zur Verfügung.

Auf der Client-Seite wird der Dateibaum transparent in den eigenen Dateibaum an einem Mount-Point eingefügt (nach den Vorgaben der /etc/mnttab). Es ist dann für den Benutzer transparent, ob er lokal oder auf einem anderen Rechner Dateien benutzt.

Ein Rechensystem kann sowohl Client als auch Server sein.

NFS ist kein Betriebssystem sondern eine Erweiterung des Dateisystems. Rechensysteme unterschiedlicher Architektur können über NFS Dateibäume im- und exportieren.

Ein Protokoll kümmert sich um das mounten. Ein anderes Protokoll regelt den Dateizugriff (es wird einfach zustandslos auf Dateien zugegriffen – sie müssen nicht geöffnet werden).
(➔UNIX-/Sitzungs-Semantik)

[NFS-Implementierung]

Echte verteilte Systeme

Verteiltes System:
: Ein verteiltes System ist ein System, das auf einer Menge von Rechnern ausgeführt wird, die nicht über einen gemeinsamen Speicher verfügen, und das sich dem Benutzer transparent wie ein einzelner Rechner darstellt.

In Multiprozessorsystemen müssen gleichartige Systemaufrufe zur Verfügung stehen (eine Transparenz wie bei NFS reicht nicht aus). Häufig wird auf allen Prozessoren derselbe Kernel verwendet.

Echte verteilte Systeme kommunizieren über Nachrichten miteinander.

Multiprozessor-Timesharing-Systeme

Sie bestehen aus mehreren Prozessoren mit Cache und einem gemeinsamen Arbeitsspeicher. Das Betriebssystem ist ebenso eng gekoppelt wie die Hardware.

Entwurfsentscheidungen

Transparenz

Ein *transparentes* verteiltes System stellt sich dem Benutzer wie ein Einprozessorsystem dar.

- Ortstransparenz
  (der Benutzer weiß nicht, wo die Daten sind)
- Migrationstransparenz
  (Betriebsmittel können unbemerkt ihren Ort verändern)
- Replikationstransparenz
  (es können mehrere Kopien der Daten existieren)
- Nebenläufigkeitstransparenz
  (mehrere Benutzer können Betriebsmittel gemeinsam benutzen)
- [Parallelitätstransparenz]

Flexibilität

Monolithischer Kernel:
: Enthält ein komplettes Betriebssystem plus Schnittstellen für die Netzwerkkommunikation.

Micro-Kernel:
: Enthält nur grundlegende Funktionen und greift sonst auf einen Server zu.

Zuverlässigkeit

Nicht alle Rechner in einem verteilten Systeme sind (aufgrund verschiedener Aufgaben) voneinander unabhängig. Der Ausfall eines Rechners kann größere Folgen haben.

Außerdem müssen Kopien von Datenbeständen konsistent gehalten werden.

[Leistung]

Skalierbarkeit

Der Aufgabenbereich von verteilten Systemen kann sich mit der Zeit stark erweitern. Die Struktur sollte sich daran anpassen können.

(4) Kommunikation in verteilten Systemen

Bei verteilten Systemen ist die Prozesskommunikation über einen gemeinsamen Speicher nicht mehr möglich. Bei WANs wird das OSI-Schichtenmodell ange-

wendet (OSI = Open Systems Interconnection), das Protokollschichten definiert.

Schichtenprotokolle

Verbindungsorientiertes Protokoll:
Es wird erst eine logische Verbindung zwischen zwei Systemen aufgebaut, über die dann Daten ausgetauscht werden können. Interessant für weitläufige Netze. (➔Telefon)

Verbindungsloses Protokoll:
Daten werden ohne Verbindung zum Zielsystem abgesendet. Interessant für eng zusammenhängende Systeme mit hörerer Sicherheit. (➔Snailmail, LAN)

Der Sender verschickt eine Nachricht, in dem sie schichtweise bis zur Schicht 1 gereicht wird (dabei können von jeder Schicht zusätzliche Informationen an die Nachricht angefügt werden), dann den Empfänger erreicht, der die Nachricht schichtweise wieder zur oberen Schicht überträgt.

Schichten (OSI-Protokoll-Stack)

7 = Anwendungsschicht (*application*)
6 = Präsentationsschicht (*presentation*)
5 = Sitzungsschicht (*session*)
4 = Transportschicht (*transport*)
3 = Netzwerkschicht (*network*)
2 = Verbindungsschicht (*connection*)
1 = Bitübertragungsschicht (*physical*)

Schicht 1 – Bitübertragungsschicht

Spezifizierung der elektrischen und signalisierenden Schnittstellen.

Schicht 2 – Verbindungsschicht

Bits senden und empfangen. Korrektur von Übertragungsfehlern durch eigene Prüfsummen.

Schicht 3 – Netzwerkschicht

Routing (Wegewahl) und permanente Überwachung der Auslastungen der Netzwerkstrecken. (➔IP)

Schicht 4 – Transportschicht

Sequenzierung (Zerlegen) von Paketen und Zusammenbau beim Empfänger in der richtigen Reihenfolge. Neue Sendung, wenn ein Paket verloren gegangen ist. (➔TCP / UDP)

Schicht 5 – Sitzungsschicht

Dialogkontrolle und Neusynchronisation im Fehlerfall (damit nicht alle Pakete komplett neu gesendet werden müssen).

Schicht 6 – Präsentationsschicht

Strukturierung der Daten in entsprechende Felder.

Schicht 7 – Anwendungsschicht

Abhängig von der Anwendung.
(➔X.400 für Email)

Das Client/Server-Modell

Für WANs ist OSI interessant, da die Übertragungsverzögerungen relativ groß sind.

Im LAN ist der Verwaltungsaufwand für das Durchlaufen der Schichten zu groß (zuviel Netzlast). Es wird dann höchstens ein Teil der Schichten benutzt.

Clients und Server

Das Client/Server-Modell basiert auf einem verbindungslosen Anfrage/Antwort-Protokoll. Die Antwortnachricht dienst als Empfangsbestätigung.

Bei identischen Rechner werden nur die OSI-Schichten 1 (Bitübertragung), 2 (Verbindung) und 5 (Sitzung) verwendet. Die Schichten 1 und 2 werden durch Hardware realisiert. Schicht 5 liefert die möglichen Anfragen/Antworten.

Die Kommunikation läuft über die Systemaufrufe *send(destination,&msg)* und *receive(port,&msg)*.

Addressierung (von Prozessen)

- feste Prozess-Nummer im Ziel-System (gegeben durch Systemnummer)
  ➔ 243@114
- Lokalisierung durch Broadcast:
  netzwerkweit eindeutige Prozessnummern, Prozess sendet ein Lokalisierungspaket (ähnlich ARP-Request) an alle Rechner und erhält vom Zielrechner eine Antwort mit der Systemnummer
  ➔ wo ist der Prozess 2342? System 114!
- ein Nameserver setzt Zeichenketten in Systemnummern um
  ➔ 243@systemname

Blockierende Primitive

Der Senderprozess wird beim Absenden einer Nachricht blockiert, bis die Nachricht übertragen wurde.

Nicht blockierende Primitive

Der Senderprozess legt die zu versendende Nachricht in einem Sendepuffer ab und kann weiterlaufen. Erst wenn mehr als eine Nachricht verschickt werden muss, wird der Prozess blockiert.

Puffernde Primitive

Der Zielrechner fordert eine Datenstruktur *Mailbox* an, in der ankommende Nachrichten jederzeit bis auf Abruf gespeichert werden. Die Mailbox hat eine be-

grenzte Kapazität und empfängt notfalls keine neuen Nachrichten.

Nicht puffernde Primitive

Der Zielrechner muss erst *receive* ausführen, bevor der Absender ein *send* ausführt. Sonst geht die Nachricht verloren.

Zuverlässige Primitive

Sinnvollerweise sendet der Kernel die Nachricht solange, bis er vom Kernel des Zielsystems eine Empfangsbestätigung erhält.

[Implementierung des Client/Server-Modells]

RPC (Remote Procedure Call)

Ein RPC soll möglichst ähnlich aussehen wie ein normaler Systemaufruf (Transparenz). Ein Programm kann ein Unterprogramm auf einem anderen Rechner aufrufen. Es werden die Parameter in eine Nachricht gepackt (*marshalling*) und über das Netz geschickt. Skalare Werte werden direkt übertragen – komplexere Datentypen (Arrays, …) müssen vom call-by-reference dereferenziert werden. Der Rückgabewert vom Zielsystem wird dann an den Prozess zurückgeliefert.

RPC bedient sich dem UDP-Protokoll (von TCP/IP), das verbindungslos arbeitet.

Stubs

Client-Stub:
: Systemprozedur aus einer Bibliothek, die eine Anfrage über den Kernel mittels eines blockierenden Systemaufrufs an einen Server weiterleitet, statt sie selbst zu bearbeiten.

Server-Stub:
: Systemprozedur auf dem Server, der auf Nachrichten von Clients wartet, sie bearbeitet und den Rückgabewert zurückschickt.

Da Datenstrukturen (➔integer) in verschiedenen Architekturen verschieden (➔*little endian* versus *big endian*) benutzt werden, muss eventuell eine Konvertierung der Parameter erfolgen.

Zeiger werden bei der Übergabe über RPC dereferenziert (nur bei einfachen Datenstrukturen möglich).

Dynamisches Binden

Da der Zielrechner nicht hardcoded im Quelltext erscheinen sollte, wird *dynamisches Binden* eingesetzt. Der Server registriert sich beim Binder (mit einer Versionsnummer seiner Schnittstelle). Ein Client fragt dann im Netz nach einem passender Server (mit dieser Versionsnummer) und bekommt einen oder mehrere Zielrechner zurück, falls diese existieren.

RPC-Semantik beim Auftreten von Fehlern

RPC soll der Transparenz dienen, die im Fehlerfall nicht gegeben ist, da der Absender auf den Fehler reagieren muss.

Client kann den Server nicht lokalisieren

Es wird eine Ausnahmebehandlung eines Signals gestartet (➔SIGNOSERVER), die vom Prozess abgefragt werden muss.

Verlust von Anfragenachrichten

Falls die Nachricht innerhalb einer gewissen Zeit vom Server nicht bestätigt wurde, wird sie erneut angefordert. Passiert dies mehrmals, gilt der Server als nicht erreichbar.

Verlust von Antwortnachrichten

Idempotente Anfragen (die ohne Nebenwirkungen wiederholt werden können ➔ Lesen eines Blocks von Festplatte) können noch einmal gestellt werden. Ansonsten muss eine Sequenznummer mit jedem Paket übertragen werden, um mehrfache Ausführung zu verhindern.

Server-Ausfall

Mindestens-einmal-Semantik:
: Der RPC wird eventuell mehrfach bis zur Erledigung ausgeführt.

Höchstens-einmal-Semantik:
: Der RPC wird nach einem Fehler sofort mit einer Fehlermeldung abgebrochen.

Client-Ausfall

Der Server hat in diesem Fall die (nicht idempotente) Anfrage bereits bearbeitet, aber der Client ist vor dem Empfang des Ergebnisses ausgefallen. Diese Anfrage ist ein *Waise* (*orphan*).

Lösung 1: Ausrottung
: führen einer Protokolldatei vor der Ausführung, bereits ausgeführte Anfragen werden nicht wiederholt.

Lösung 2: Reinkarnation
: beim Neustart des Clients wird eine neue *Epoche* angekündigt, die alle aktiven RPC-Ausführungen auf dem Server abbricht.

Lösung 3: sanfte Reinkarnation
: bei einer neuen Epoche überprüft jeder Client, ob er Berechnungen auf einem Server ausführt und versucht, den verantwortlichen Prozess zu finden.

Lösung 4: Verfallszeitpunkte

ein RPC hat nur eine begrenzte Laufzeit, nach der manuell ein neues Laufzeitintervall angefordert werden muss – ansonsten terminiert der Prozess.

[Implementationsaspekte]

(5) Gruppenkommunikation

Gruppen ermöglichen es einem Prozess mit einer Menge anderer Prozesse (als eine abstrahierte Einheit) zu kommunizieren.

Zur Erhöhung der Zuverlässigkeit kann ein Client mit mehreren redundanten Servern kommunizieren.

Multicasting:

Ein Paket wird an eine bestimmte Gruppenadresse geschickt, die die interessierten Rechner abhören.
Die Netzwerkhardware entscheidet, ob das System in der Gruppe ist.

Broadcasting:

Ein Paket wird an alle im Netz befindlichen Rechner geschickt.
Die Software entscheidet, ob das Paket für dieses System gedacht war.

Unicasting:

Das Paket wird einzeln an jedes andere Mitglied der Gruppe geschickt, wenn weder Multicasting noch Broadcasting zur Verfügung stehen.

Entwurfsentscheidungen

Geschlossene Gruppen

Nur Mitglieder der Gruppe können an andere Mitglieder in der Gruppe senden. Außenstehende können lediglich Nachrichten an einzelne Systeme schicken

Offene Gruppen

Keine Beschränkung der Kommunikation.

Demokratische/Hierarchische Gruppen

Demokratische Gruppen können beim Ausfall eines Mitglieds weiter arbeiten.

Bei hierarchischen Gruppen ist ein Prozess für die Koordination zuständig. Er darf nicht ausfallen.

Gruppenmitgliedschaft

Ein Gruppen-Server (der nicht ausfallen darf) kann die Mitgliedschaften zentral verwalten.

Werden die Mitgliedschaften verteilt verwaltet (auf jedem einzelnen Rechner), müssen Protokolle den Eintritt und Austritt (auch bei Ausfall eines Rechners) regeln.

Gruppenadressierung

Ein Prozess kann an die Multicast-Adresse senden, die jeder Rechner der Gruppe abhört.

Beim Broadcasting prüft jeder Empfänger, ob der Rechner in der Gruppe ist und verwirft sonst die Nachricht.

Bei der *Prädikatadressierung* wird ein Paket von der Gruppe nur akzeptiert, wenn bestimmte mitgelieferte Bedingungen passen (➔"nur an Rechner mit mindestens 4 MB RAM").

[Sende- und Empfangsprimitive]

Atomarer Broadcast

Es müssen alle Mitglieder der Gruppe die Nachricht empfangen. Ablehnungen oder Verlorengehen einer Nachricht werden nicht akzeptiert. Eine hilfreiche Bedingung, wenn sich nicht jeder Prozess um die Fehlerbehandlung kümmern möchte. („Alles-oder-nichts-Übertragung")

Nachrichtenreihenfolge

Alle Nachrichten müssen in derselben Reihenfolge am Empfänger ankommen, wie sie gesendet wurden (*globale Zeitordnung*).

[Überlappende Gruppen]

[Skalierbarkeit]

Gruppenkommunikation in ISIS

ISIS ist eine Sammlung von Programmen unter UNIX, die die Entwicklung verteilter Anwendungen erleichtern soll.

In einem *synchrones System* treten alle Ereignisse sequentiell ein. Das Versenden von Nachrichten ist atomar (➔atomarer Broadcast) und die Übertragung zeitlos. Es gibt aber kein synchrones System.

Ein *lose synchrones System* sorgt für Ereignisse in der korrekten Reihenfolge (➔konsistente Zeitordnung).

ISIS stellt einige Kommunikationsprimitive zum Broadcasting zur Verfügung.

(6) Synchronisation in verteilten Systemen

Semaphore und Monitore sind auf einen gemeinsamen Speicher angewiesen und können so in verteilten Umgebungen nicht funktionieren.

Synchronisation von Uhren

Uhren müssen synchron laufen, damit z.B. *make* funktioniert (sonst würden geänderte Programmteile nicht neu übersetzt werden).

Die interne Uhr eines Rechner besteht aus einem *Zählerregister* und einem *Laderegister*. Das Zählerregister wird bei jedem Tick dekrementiert und bei Null wieder auf das Laderegister gesetzt.

Logische Uhren

Bei vielen Anwendungen reichen logische Uhren aus, die zwar eine konstante Uhrabweichung von der Realzeit haben, aber untereinander synchron sind.

nebenläufig:
: Prozesse laufen nicht sequentiell (vielleicht sogar gleichzeitig) ab

Algorithmen zur Synchronisation logischer Uhren

Algorithmus von Lamport:
: Eine Nachricht bekommt immer die Absendezeit des Senders angehängt. Kommt die Nachricht beim Empfänger an, darf sie nicht vor dem Sendezeitpunkt empfangen werden. Sonst setzt der Algorithmus die Uhrzeit auf Absendezeit + 1.

Berkeley-Algorithmus:
: Hat kein System einen Atomzeit-Empfänger, fragt der Zeit-Server alle Clients regelmäßig nach der lokalen Zeit und übermittelt wiederum an alle Clients die durchschnittliche Uhrzeit. Die Uhrzeit am Server muss manuell gesetzt werden.

Dezentrale Durchschnittsbildung:
: Jeder Rechner fragt regelmäßig alle anderen Rechner im Netz nach deren Zeit und bildet einen Durchschnitt darüber – abzüglich des maximalen und minimalen Wertes, um sich vor defekten Uhren zu schützen.

Physikalische Uhren

Physikalische Uhren dürfen lediglich um einen bestimmten Betrag von der Realzeit abweichen.

UTC (Universal Coordinated Time):
: Gängiges Zeitsystem, das sich astronomisch an der Bewegung der Sonne orientiert.

Algorithmen zur Synchronisation physikalischer Uhren

Algorithmus von Cristian:
: Der Client fragt einen Zeit-Server ab, der mit einem Atomzeit-Empfänger ausgerüstet ist. Der Algorithmus misst die Übertragungszeit zum Client, um sie abzuziehen. Die Client-Uhr wird langsam schrittweise an die Server-Uhr angeglichen (damit die Zeit nicht merklich rückwärts läuft).

Mehrere externe Zeitquellen:
: Jeder Rechner fragt mehrere UTC-Quellen ab und sucht den Schnittbereich der UTC-Bereiche. Die Zeit wird dann auf den Durchschnitt des Schnittbereichs gesetzt.

Wechselseitiger Ausschluss

Zentraler Algorithmus

Ein Server ist der Koordinator, bei dem das Eintreten in den kritischen Bereich angemeldet werden muss.

Möchte ein zweiter Prozess in den kritischen Bereich eintreten, so ignoriert der Server seine Anfrage vorerst und schickt erst dann ein OK zurück, wenn der kritische Bereich frei ist. Die Anfragen werden in einer Warteschleife gespeichert.

Nachteil: Unsicherheit falls der Server ausfällt

Vorteil: geringe Netzlast

Verteilter Algorithmus

Ist das Versenden von Nachrichten zuverlässig, dann schickt ein Prozess eine Anfrage an alle anderen Prozesse im Netz, ob der Bereich frei ist. Bekommt er ein OK, tritt er in den kritischen Bereich ein.

Falls bereits ein Prozess im kritischen Bereich ist, wird die OK-Meldung erst nach dem Austritt aus dem kritischen Bereich verschickt.

Antwortet ein Prozess gar nicht auf diese Anfrage, dann muss eine Fehlerbehandlung aktiviert werden.

Token-Ring-Algorithmus

Ein Token zirkuliert im Ring. Ein System kann den Token während des Eintritts in den kritischen Bereich behalten und danach weitergeben.

Nachteil: das Token könnte verlorengehen oder ein Prozess könnte ausfallen, Overhead

Wahlalgorithmen

In verteilten Systemen gibt es oft einen Koordinator zur Synchronisation.

Wenn ein Prozess ausfällt kann durch Wahlalgorithmen ein neuer Koordinator bestimmt werden.

Bully-Algorithmus

Wenn ein Prozess den Ausfall des Koordinators bemerkt, schickt er *WAHL*-Nachrichten an alle Prozesse mit höheren Prozessnummern als der eigenen.

Erhält ein Prozess eine *WAHL*-Nachricht, so bestätigt er mit OK und wiederholt diesen Schritt.

Antwortet kein höherer Prozess, so ist dieser Prozess der neue Koordinator, was er durch eine *KOORDINATOR*-Nachricht bekanntgibt.

Ein Ringalgorithmus

Im Ring schickt ein Prozess eine *WAHL*-Nachricht mit seiner Prozessnummer durch den Ring (und überspringt ausgefallene Prozesse), bis diese Nachricht wieder bei ihm ankommt. Bei jedem Prozess wird die jeweilige Prozessnummer angehängt.

Kommt die Nachricht wieder beim Absender an, extrahiert er die höchste Prozessnummer und gibt sie als *KOORDINATOR*-Nachricht durch den Ring.

(Atomare) Transaktionen

Transaktionen sind eine höhersprachliche Implementation von Prozesskommunikation.

Ein Prozess kündigt eine Transaktion bei anderen Prozessen an und nennt die genaueren notwendigen Operationen. Wird die Durchführung der Transaktion von allen beteiligten Prozessen akzeptiert, so wird sie in einem Schritt (*atomar*) durchgeführt. (➔Alles-Oder-Nichts-Eigenschaft)

Lehnt ein Prozess die Transaktion ab, kehrt alles in den Ursprungszustand zurück.

Beispiel: Bank-Überweisung

Das Transaktionsmodell

Der Nachrichtenaustausch wird als zuverlässig angemommen.

Stabiler Speicher:
: Hochsicherheitsspeicherung von Daten auf einem RAID mit Plattenspiegelung und Prüfsummenbildung.

Transaktionsprimitive:
: Befehle zur Steuerung von Transaktionen.
➔BEGIN_TRANSACTION
➔END_TRANSACTION
➔ABORT_TRANSACTION

Eigenschaften von Transaktionen

- Serialisierbarkeit
  (nebenläufige Transaktionen beeinflussen sich nicht)
- Atomarität
  (eine Transaktion wird ohne Zwischenschritte unteilbar ausgeführt)
- Permanenz
  (bei Erfolg einer Transaktionen ist das Ergebnis festgelegt)

Implementation von Transaktionen

Privater Arbeitsbereich:
: Sollen durch eine Transaktion Daten verändert werden (Schreibzugriff), werden die Originaldaten in einen *privaten Arbeitsbereich* kopiert und dort verändert. Wird die Transaktion abgebrochen, wird der private Arbeitsbereich verworfen.

Protokollierung der Schreibabsichten:
: Die Originaldaten werden direkt verändert und dieser Zugriff wird in einem Protokoll vermerkt (alter und neuer Wert). Wird die Transaktion abgebrochen, wird durch *Rollback* der alte Wert zurückgeschrieben.

Zwei-Phasen-Commit (für verteilte Systeme):
: Der Koordinator sendet an die beteiligten Prozesse die Transaktionsdaten. Kann jeder Prozess die Transaktion durchführen, so bestätigt er dies. Sobald alle beteiligten Prozesse einverstanden sind, sendet der Koordinator das *COMMIT* und die einzelnen Prozesse bestätigen die Ausführung.

Nebenläufigkeitskontrolle

Wenn mehrere Transaktionen parallel ausgeführt werden, sorgt die Nebenläufigkeitskontrolle für konsistente Ergebnisse.

Sperren (*Locks*):
: Prozesse können Daten oder Ressourcen beim Koordinator zum lesen oder schreiben sperren. Es sind mehrere Lesesperren möglich, aber immer nur eine exklusive Schreibsperre. Bei feiner *Granularität* können Teile einer Datei einzeln gesperrt werden.

Zwei-Phasen-Sperren:
: Die für eine Transaktion benötigten Sperren werden in der ersten Phase nacheinander gesetzt und dann die Transaktion durchgeführt. In der zweiten Phase werden die Sperren nacheinander wieder freigegeben.
Vorteil: man kann die Transaktion in der ersten Phase problemlos abbrechen (bei Abbruch der Transaktion oder Behebung von Deadlocks)

[Optimistische Nebenläufigkeitskontrolle]

Zeitstempel:
: Zu Beginn einer Transaktion wird ein Zeitstempel (*Timestamp*) gemerkt. Falls eine Transaktion zeitlich gesehen vor einer anderen Transaktion beginnt, kann sie nicht durchgeführt werden, weil ein anderer Prozess sich eingemischt hat. Diese Transaktion muss abgebrochen werden.

Deadlocks in verteilten Systemen

Beispiel: verteiltes Datenbanksystem

Erkennung und Behebung

Transaktionen können beim Auftreten von Deadlocks problemlos abgebrochen (und später wiederholt) werden.

Zentrale Erkennung:
: Jeder Rechner verwaltet seinen eigenen Betriebsmittelgraphen und ein Koordinator verwaltet die Vereinigung aller Betriebsmittelgraphen. Der Koordinator achtet auf Deadlocks und bricht Prozesse ab. Durch Nachrichtenlaufzeiten können *scheinbare Deadlocks* verursacht werden.

Verteilte Erkennung (Chandy-Misra-Haas-Algorithmus):
: Wenn ein Prozess ein Betriebsmittel belegen möchte, das bereits belegt ist, sendet er eine *Untersuchungsnachricht* an alle Prozesse, die einen Deadlock überprüfen.

Verhinderung

Verhinderung von Deadlocks (➔Bankiersalgorithmus) wird in verteilten Systemen nicht eingesetzt, weil zu kompliziert.

Vermeidung

Wait-Die-Algorithmus:
: Wenn ein älterer Prozess (früherer Zeitstempel) auf Betriebsmittel eines jüngeren Prozesses (älterer Zeitstempel) wartet, wird der jüngere Prozess abgebrochen, damit der ältere Prozess sein Werk vollenden kann. Nachteil: der jüngere Prozess würde immer wieder neu gestartet werden

Wound-Wait-Algorithmus:
: umgekehrter Fall zu Wait-Die

(7) Prozesse und Prozessoren in verteilten Systemen

Threads (*leichtgewichtige Prozesse*)

Threads sind Sub-Prozesse, die auf denselben Adressraum wie ein anderer Prozess zugreifen, aber jeweils einen eigenen Kontrollfluss haben. (Normalerweise können Prozesse nur über Semaphore oder Nachrichten kommunizieren, weil es keinen gemeinsamen Speicherbereich gibt.) Dateiserver oder Webserver setzen Threads ein, um mehrere Anfragen quasi gleichzeitig auszuführen. Ein einzelner Thread kann dabei blockieren, um auf ein Betriebsmittel zu warten, während der Dienst weiter zur Verfügung steht.
(➔File-Server)

Benutzung

Threads erlauben parallele Verarbeitung von sequentiellen Prozessen trotz blockierender Systemaufrufe. Der Softwareentwurf wird damit vereinfacht.

Wenn Anfragen für die Threads kommen, können sie entweder aktiv aus einem Pool entnommen werden (Team-Modell) oder von Thread zu Thread durchgereicht werden (Pipeline-Modell).

Threads werden vom Prozess gestartet und beendet. Sie können auch bis zur eigenen Terminierung ablaufen. Beim Start eines Threads wird ihm ein eigener Stack-Bereich zugewiesen.

Entwurfskriterien für Thread-Pakete

Thread-Paket:
: Menge von Systemaufrufen, die dem Benutzer bei Threads zur Verfügung stehen

Ein Thread kann dynamisch von einem Prozess erzeugt werden. Es wird ein Zeiger auf das Programm und die Größe des Stackbereichs an einen Systemaufruf übergeben. Das System gibt dann die PID zurück.

Innerhalb eines Prozesses mit Threads kann ein Mutex (*mutual exclude*) benutzt werden, um auf gemeinsame Speicherbereiche exklusiv zugreifen zu können. Ein Mutex ist ein binärer Semaphor.

Jeder Thread benötigt eigene *private globale Variablen*, um z.B. Rückgabewerte zu speichern, die nicht mit anderen Threads geteilt werden sollen.

[Threads und RPC]

Systemmodelle

Workstation-Modell

Jeder Benutzer verfügt über einen Rechner mit eigenem Prozessor. Dadurch hat er gute Antwortzeiten. Der Rechner hat entweder keine eigene Festplatte (*diskless workstation*) oder ist vollwertig alleine nutzbar.

Nutzung ungenutzter Workstations

Server-initiierter Algorithmus:
: Ein Koordinator stellt fest, ob eine Workstations frei für weitere Arbeit ist.

Client-initiierter Algorithmus:
: Die Workstation selbst gibt seine Auslastung bekannt.

Ein Programm wird dann „remote“ auf einem freien Rechner ausgeführt. Die Systeme müssen homogen genug dafür sein (➔kein direktes Schreiben in den Bildschirmspeicher). Die Ein- und Ausgabe wird auf dem Rechner des Initiators belassen.

Wird die Workstation doch wieder von einem Anwender benutzt, so wird der Prozess auf einen anderen freien Rechner migriert oder abgebrochen.

Prozessor-Pool-Modell

Entsprechend der Warteschlangentheorie gibt es einen Pool von Prozessoren (die Bedienstation). Die Prozesse (Aufträge) werden an die Prozessoren verteilt und abgearbeitet. Die Bedienrate (μ) muss lediglich größer als die Ankunftsrate (λ) sein, damit das System stabil ist und nicht „überläuft".

Die einzelnen Arbeitsstationen haben nur noch Funktionen eines grafischen Terminals.

Mischmodell

Jeder Benutzer hat eine private Workstation mit Prozessor und Festplatte. Freie Workstations werden während der unbenutzten Zeit nicht für andere Aufgaben eingesetzt.

Prozessorzuteilung

Zuteilungsmodelle

Nicht-migrierende Zuteilung:
: Ein Prozess bleibt bis zur Terminierung einem Prozessor zugewiesen.

Migrierende Zuteilung:
: Ein Prozess kann zwischen Prozessoren migrieren. (Besser Lastausgleich, aber viel komplexer.)

Entwurfskriterien für die Prozessorzuteilung

- deterministisch/heuristisch
  Deterministisch nur möglich, wenn Bedienzeit bekannt.
- zentral/verteilt
  Zentral unsicherer aber optimaler.
- optimal/suboptimal
  Optimal ist immer teuer und aufwendig.
- Transportstrategie: lokal/global
  Bei lokaler Transportstrategie wird aufgrund einer Entscheidung des initiierenden Rechners (überschreiten einer Auslastungsgrenze) der Prozess verlagert. Bei globaler Strategie wird die Gesamtsituation betrachtet.
- Platzierungsstrategie: Sender-/Empfänger-initiiert
  Sender-inittiert: Der Sender kündigt seine Verfügbarkeit an.
  Empfänger-initiiert: Der Empfänger sucht sich Arbeit.

[Implementierung von Prozessorzuteilungsalgorithmen]

Beispiele

Graphentheoretisch-deterministisch:
: Prozesse sind Knoten, die gewichteten Kanten stellen den Netzwerkverkehr zwischen Prozessen dar. Man versucht eine Partitionierung und weist jedem Prozessor eine Partition zu, so dass zwischen den Prozessoren wenig Netzlast aufkommt.
  Nachteil: Netzlast zwischen Prozessen muss vorher bekannt sein.

Zentraler „Up-Down"-Algorithmus:
: Ein Koordinator verwaltet die Last jedes Rechners in einer Benutzungstabelle. Jeder Workstation-Benutzer hat einen Eintrag in der Benutzungstabelle. Führt er einen Prozess auf einem anderen Rechner aus, so erhält er einige Strafpunkte pro Sekunde. Lässt er andere Prozesse seinen Prozessor nutzen, bekommt er Strafpunkte abgezogen.

Hierarchischer Algorithmus:
: Prozessoren sind auf der untersten Hierarchieebene. Müssen Prozessoren belegt werden, dann versucht er erst eine Gruppe der unteren Ebene und geht solange die Ebenen höher, bis genügend Prozessoren belegt werden konnten.

Verteilter heuristischer Algorithmus:
: Soll ein neuer Prozess auf einem Rechner erzeugt werden, so fragt er zufällig andere Rechner nach deren Last und übergibt eventuell den Prozess.

[Auktionsalgorithmus]

[Scheduling in verteilten Systemen]

(8) Verteilte Dateisysteme

Dateidienst:
: Spezifikation der Zugriffsmöglichkeiten auf Dateien (Lesen, Schreiben, Anlegen, Löschen von Verzeichnissen und Dateien).

Datei-Server:
: Prozess, der Datei-Anfragen von Client-Rechnern empfängt und beantwortet.

Entwurf

Schnittstelle des Dateidienstes

Upload/Download-Modell:
: Es gibt nur zwei Operationen, die ganze Dateien auf den Dateiserver kopieren oder herunterladen. (Einfaches Konzept, ➔FTP)

Modell des entferntes Zugriffs:
: Der Dateiserver bietet umfangreiche Operationen an, um Teile von Dateien zu lesen etc. Der Client braucht hier keinen eigenen Speicherplatz und es gibt kleinere transferierte Datenmengen. (➔NFS)

Schnittstelle des Verzeichnisdienstes

Verzeichnisbäume werden in den lokalen Baum ‚gemountet'.

Ortstransparenz:
: Ein Pfadname gibt nicht an, auf welchem Server sich eine Datei befindet.

Benennung von Dateien

- Rechner+Pfadname
  host.domain:/path/file
- Mounten entfernter Dateisysteme
  /export/home/username/file
- Einheitlicher Namensraum

[Zweistufige Benennung]

Semantiken der gemeinsamen Dateinutzung

UNIX-Semantik:
: Anfragen (READ, WRITE…) werden in der Reihenfolge des Eintreffens beantwortet.

Sitzungssemantik:
: Erst wenn eine Schreiboperation beendet wurde (close), werden die Caches an den Server übertragen und damit die Änderungen sichtbar.

Transaktionen:
: Durch Transaktionen lassen sich auch hier Konflikte vermeiden.

Implementation verteilter Dateisysteme

Dateinutzung

Beobachtungen ergaben für die meisten Dateien:

- < 10 KB
- werden öfter gelesen als geschrieben
- Lesen ist sequentiell
- sind kurzlebig
- keine gemeinsame Benutzung

[Systemstruktur]

Caching

Das Caching passiert normalerweise im Arbeitsspeicher des Clients, obwohl es im Arbeitsspeicher oder auf Festplatten der Server oder Clients ablaufen könnte.

Aufrufe ins Dateisystem werden vom Kernel zum Cache-Manager umgeleitet. Wenn die Daten noch nicht im Cache sind, wird der Server nach den Daten über das Netzwerk gefragt. Ist der Cache-Bereich gefüllt, wird LRU mit verketteten Listen verwendet.

Cache-Konsistenz

Bei der Sitzungssemantik (erst schreiben, wenn die Datei geschlossen wird) kommt es zu keinen Problemen, sofern sich der Prozess dieser Semantik bewusst ist.

Write-Through:
: Die Änderungen werden zwar gecachet, aber auch direkt an den Server übermittelt. Beim erneuten Lesen aus dem Cache muss aber erst angefragt werden, ob er noch aktuell ist.

Verzögertes Schreiben:
: Änderungen werden nicht sofort auf Platte geschrieben.

Write-On-Close:
: Verwendung der Sitzungssemantik. Wenn die Datei nicht innerhalb 30 Sekunden gelöscht wird (viele Dateien sind kurzlebig), wird sie zum Server geschickt.

Zentrale Kontrolle:
: Der Client kündigt das Schreiben beim Server jedesmal an (Datei-Lock vom Server).

Replikation

Mehrere Rechner halten Kopien der Dateien.

- höhere Ausfallsicherheit / Verfügbarkeit
- Lastverteilung / Leistungsfähigkeit

Implementation:

- Einem Dateinamen entsprechen mehrere binäre Adressen (I-Nodes), auf die ein Prozess schreibt.
- Die Datei wird an eine Stelle auf dem Server geschrieben. Der Server kümmert sich um die Verteilung (*langsame Replikation*).
- Via Gruppenkommunikation wird die Datei gleichzeitig an mehrere Server auf einmal.

Replikation der ersten Kopie:
: Der Server ändert die Datei (in einem sicheren Speicher) und schickt diese Änderungen an die weiteren Server.
  Nachteil: Abhängigkeit vom Hauptserver

Abstimmung (*voting*):
: Der Client muss von der Mehrheit der Server die Erlaubnis zum Schreiben der Datei einholen. Beim Lesen muss er wieder die Mehrheit der Server nach der gespeicherten Version fragen. (Wenn 3 von 5 Servern die Version X haben, kann es nichts aktuelleres geben.)

[Quorum]

[Abstimmung mit Schatten]

III Datenbanken

(Kapitel I, II und III)

(1) Kapitel I: Grundlegende Konzepte

Ein Datenbanksystem unterstützt grundlegende Aktionen:

- Tabellen anlegen
- Tabellen löschen
- Daten in Tabellen einfügen (*insert*)
- Daten aus Dateien abfragen (*retrieve*)
- Daten in Dateien erneuern (*update*)
- Daten aus Dateien entfernen (*delete*)

Die relationalen Tabellen enthalten die Daten. Ein Datensatz entspricht einer Zeile in der Tabelle. Die Inhalte der Felder sind in Spalten angeordnet.

SQL (*structured query language*):
: Standard-Datenbanksprache zur Abfrage von Daten aus relationalen Tabellen. (Query bedeutet aber auch die Manipulation von Daten!)

Datenbanksystem

Ein Datenbanksystem besteht aus:

- Daten
- Hardware
- Software
- Benutzer

Daten

Daten sind integriert (eine Datenbank besteht aus mehreren Tabellen) und gemeinsam nutzbar (von verschiedenen Benutzern gleichzeitig).

Hardware

Die Hardware besteht aus dem Sekundärspeicher (Festplatten, Controllern), dem Prozessor und Hauptspeicher.

Software

Die Software steht zwischen Hardware und Benutzer und bildet das DBMS (*database management system*). Es schirmt den Benutzer von Details des Systems ab und bietet Operationen wie die von SQL an.

Benutzer

Es gibt Programmierer, die Programme schreiben, die die Datenbank benutzen. Weiter gibt es Endanwender, die mit dem System interagieren. Und es gibt Administratoren (DBA=*database administrator*), die die Datenbank technisch am laufen halten.

Datenbanken

Persistente Daten

Daten in Datenbanken bleiben so lange erhalten, bis man sie per Query verändert oder löscht.

Transient (flüchtig, kurzlebig) sind nur Daten während der Ein- oder Ausgabe.

Entity/Relationship

Entity:
: Unterscheidbares Objekt in einer Datenbank, über das Daten (*Properties*) gespeichert werden.

Relationship (Beziehung):
: Birektionale Verbindung zwischen Entities. Es gibt binäre, tertiäre… Relationen.

Warum Datenbanken?

Vorteile von Datenbanken:

- Kompaktheit
  (keine großen Aktenordner)
- Geschwindigkeit
  (schneller Datenzugriff)
- Wenig Redundanz
  (zentrale Speicherung)
- Aktualität
  (jederzeit Zugriff auf aktuelle Daten, Vermeidung von Inkonsistenzen)
- Gleichzeitiger Zugriff
  (von mehreren Anwendern auf dieselben Daten)
- Sicherheit
  (Beschränkung des Zugriffs)
- Persistenz
  (Dauerhafte Speicherung)

Datenadministration und Datenbankadministration

Datenadministrator (DA):
: Verantwortlicher für die Korrektheit der gespeicherten Daten.

Datenbankadministrator (DBA):
: Technischer Verantwortlicher für die Umsetzung und Wartung der Datenbank und dessen Struktur entsprechend des Vorgaben des DA.

Unabhängigkeit von Daten

Daten werden in der Datenbank beliebig gespeichert, die Art der Speicherung ist für die Anwendung transparent.

Feld (*stored field*):
: Speichert einen atomaren Eintrag in einem Datensatz. Felder können numerisch, binär, zeichenorientiert etc. sein.

Datensatz (*stored record*):
: Speichert Eigenschaften eines Objekts in mehreren Feldern.

Tabelle (*stored file*):
: Speichert mehrere Datensätze im selben Format (gleiche Felder).

Relationale Systeme und andere

In relationalen Systemen werden Daten in Tabellen gespeichert. Die einzelnen Datensätze sind mathematisch gesehen Tupel. Die Tabelle entspricht einer mathematischen Relation (Kreuzprodukt von Mengen).

(2) Architektur für ein Datenbanksystem

ANSI/SPARC-Architektur

- Externe Ebene (*external level*)
  Verschiedene Ansichten der Daten für den Anwender (➔Ausschnitte aus einer Datenbank)
- Begriffliche Ebene (*conceptual level*)
  Zwischenebene, die die Art der gespeicherten Daten angibt (Informationsgehalt / Datentypen)
- Interne Ebene (*internal level*)
  Physikalische Speicherung von Daten (➔Bits, Bytes, Festplatten - maschinenabhängig)

Es gibt mehrere externe Ansichten der einzigen begrifflichen Ebene.

DSL (*data sublanguage*):
: Menge von Sprachen, mit denen man Datenbankoperationen ausführen kann. (➔SQL, DDL, DML)

Der Datenbankadministrator

Aufgaben des DBA:

- Umsetzung des Datenbankdesigns eines Datenadministrators in eine DDL (*database definition language*)
- Physikalisches Datenbankdesign des internen Schemas (Speicherung der Daten)
- Daten für Benutzer verfügbar machen
- Sicherheitsrichtlinien (➔ACL) festlegen
- Backup/Restore
- Überwachung und Skalierung

DBMS

Abfrageschema:

1. Benutzer stellt Anfrage in SQL
2. Das DBMS überprüft die Anfrage auf Gültigkeit
3. Das DBMS geht über die externe Sicht zur begrifflichen Sicht und weiter zur internen Sicht, um die Anfrage auszuführen.
4. Das DBMS führt die Operation aus und liefert die Daten zurück.

Funktionen der DBMS:

- Definition der Daten
  Sprachprozessor zur Interpretation der DDL (*database definition language*).
- Manipulation der Daten
  Sprachprozessor zur Interpretation der DML (*database manipulation language*)
- Integrität und Sicherheit der Daten
  Überprüfung der ACLs und Integrität.
- Führen einer Systemdatenbank (SYSTABLE)
- Leistung
  Effiziente Abarbeitung der Anfragen

Backend / Frontend

Backend:
: DBMS

Frontend:
: Applikationen, die auf dem DBMS basieren.

Hilfsprogramme

Für Statistik, Analyse, Backup/Wiederherstellung und Reorganisation.

Verteilte Verarbeitung

Mehrere Frontends können quasi gleichzeitig über ein Netzwerk auf ein Backend zugreifen.

In einem verteilten Datenbanksystem können auch mehrere Datenbankserver über ein Netz verteilt sein.

(3) Interne Ebene

Datenbankzugriff

1. Das DBMS entscheidet, welcher Datensatz gewünscht wird und fordert ihn vom Datei-Manager an.

2. Der Datei-Manager finden den zugehörigen Block und fordert ihn vom Disk-Manager an.
3. Der Disk-Manager (Controller) liest den physikalischen Block mittels I/O-Operationen von der Festplatte.

Clustering

Zusammenhängende Datensätze sollte das DBMS auch zusammenhängend auf Platte speichern, um die Performanz zu erhöhen.

Blöcke (*page sets*) und Dateien

Mehrere Datensätze können in einem Plattenblock[1] gespeichert werden. Innerhalb eines Blocks können Datensätze ohne I/O-Zugriffe direkt defragmentiert werden. Eine Tabelle wird in einer Folge von Blöcken (*page set*) gespeichert.

RIDs (*record identification*) werden aus der Blocknummer und dem Byte-Offset gebildet.

Indizierung

Wird eine Abfrage regelmäßig über ein bestimmtes Feld gestellt, bietet sich der Aufbau eines Index über dieses Feld an. Der Index ist eine zusätzliche Tabelle, die vom DBMS gewartet wird.

Eine Index-Datei besteht aus Einträgen mit je dem Feldinhalt des indizierten Feldes (Schlüssel) und dem Zeiger auf die RID.

Index-Dateien erlauben einen schnelleren Zugriff, da nicht jeder Datensatz aus der Hauptdatei geladen werden muss, um das Kriterium der Abfrage zu prüfen.

Ist die Index-Datei sortiert (z.B. alphabetisch), kann auch ein sequentieller Zugriff auf die Datensätze schneller erfolgen.

Es lassen sich auch Index-Dateien mit mehreren Schlüsseln bilden.

B-Bäume (B-trees / balanced trees)

Die Sequenzmenge (*sequence set*) ist eine verkette Liste mit Zeigern auf die einzelnen Seiten. Das ermöglicht einen schnellen sequentiellen Zugriff auf die Datensätze.

Die Indexmenge (*index set*) bildet eine Baumstruktur. In jedem Knoten befinden sich Zeiger auf weitere Knoten. Der Zeiger links von einem Wert zeigt auf weitere Werte, die kleiner sind. Der Zeiger rechts von einem Wert zeigt auf Werte, die größer sind.

Statt alle Zeiger in der Sequenzmenge zu durchlaufen ist nur ein Zugriff pro Baumebene nötig, um eine Seite (nach Nummer) zu finden.

Bei Baumstrukturen kann es schnell zu „unbalanced trees“ kommen. Bei B-Trees lassen sich die einzelnen Knoten leichter verändern, um dies zu verhindern.

Hashing

Indizierung über eine Funktion. Aufgrund einer Berechnungsgrundlage werden die Datensätze gespeichert und gelesen. Beispielsweise kann man dann Datensatz 251 in Block 252 finden (Block = Datensatznummer + 1).

Verkettete Zeiger

Bei häufigen Abfragen nach einem Feld bietet sich eine andere Art der Indizierung an. Von einem Suchwort werden alle passenden Datensätze über eine verkettete Liste verbunden. Einfügen und Löschen von Datensätzen ist damit einfach.

Komprimierung

Differentielle Komprimierung:
: Es werden nur die Unterschiede vom vorhergehenden Datensatz zum aktuellen gespeichert.

Hierarchische Komprimierung:
: Zusammengehörende Datensätze (mit z.B. demselben Inhalt im Index-Feld) können gruppiert werden wie bei einem hierarchischen Index.

Huffman-Kodierung:
: Basiert auf der Beobachtung, dass manche Zeichen öfter als andere in Daten vorkommen (➔‘e‘ ist häufiger als ‚y‘). Ein ‚e‘ wird z.B. durch ‚1‘ repräsentiert – ein ‚y‘ durch ‚01‘.

(4) Kapitel II: Relationale Systeme und DB2

Alle Einträge in Datensätzen sind atomar. Es gibt keine Mengen von Werten oder Zeiger in einem Datensatz – sondern nur explizite Werte.

DB2 unterstützt nur B-trees (kein Hashing etc.).

SQL

DB2 von IBM ist relational und hat SQL eingeführt (was von ANSI zum Standard erklärt wurde). SQL ist nicht-prozedural (man gibt nur an, was man will, aber man ruft direkt keine Subroutinen auf). Es enthält eine DDL- und eine DML-Komponente.

Definierende Operationen:
: CREATE TABLE TABELLE1

[1] Hier sind logische Blöcke des Dateisystems gemeint, die 512 Bytes bzw. 1 KB groß sind.

```
(
  FELD1   CHAR(10) NOT NULL,
  FELD2   CHAR(2)   NOT NULL
  PRIMARY KEY (FELD1)
);
```

Abfragende Operationen:

```
SELECT FELD1
FROM TABELLE1
WHERE FELD2 = ‚HAMBURG‘;
```

Manipulierende Operationen:

```
UPDATE TABELLE1
SET FELD2 = 2 * FELD2
WHERE FELD1 = ‚123‘;
```

Es gibt „interactive SQL“ für Operationen im direkten Dialog mit dem DBMS und „embedded SQL“ als Erweiterung von Programmiersprachen, die auf das DBMS zugreifen möchten.

Tabellen (tables=*base tables*) enthalten die volle Informationen. Ansichten (views=*virtual tables*) sind ausgewählte Teile von Tabellen davon. Tabellen existieren real, Views sind nur Ableitungen von Tabellen.

Größere Systemkomponenten

- Systemdienste (*system service*)
  (Wartung, Kommunikation, Protokollierung)
- Sperrdienste (*locking services*)
  (Steuerung konkurrenter Zugriffe)
- Datenbankdienste (*database services*)
  (Definition, Abfrage und Veränderung von Daten)
  – Precompiler
  – Bind
  – Runtime Supervisor
  – Stored Data Manager (Dateimanager)
  – Buffer Manager (Diskmanager)

Zur Vorbereitung und Ausführung einer DB2-Datenbankapplikation werden folgende Schritte durchlaufen:

- Der Precompiler extrahiert die Embedded-SQL-Anweisungen in ein DBRM (*DataBase Request Module*) und ersetzt sie durch CALL-Anweisungen im Quelltext.
- Die Applikation wird ohne SQL-Anweisungen compiliert und gelinkt zu einem *„load module“*.
- Die DBRMs werden mittels *Bind* in Objektdateien (*application plan*) compiliert.
- Während der normalen Ausführung rufen die CALL-Anweisungen im *„load module“* den entsprechenden *„application plan“* auf.

[Optimierung]

[Kompilierung und Rekompilierung]

(5) Datendefinition (DDL)

Base Table:

> Relationales Objekt einer Datenbank bestehend aus einer Reihe von Spaltenüberschriften zusammen mit Zeilen von Werten.

Die Zeilen der Tabelle sind nicht sortiert, denn eine mathematische Relation kennt auch keine Sortierung. Die Spalten hingegen sind von links nach rechts sortiert (da die Spalten verschiedene Bedeutungen haben), der Benutzer wird aber immer über den Feldnamen auf den Wert zugreifen und nicht über die Spaltennummer.

CREATE TABLE

Erzeugt eine neue Tabelle.

```
CREATE TABLE tabellenname
(
        feldname datentyp [NOT NULL],
        feldname datentyp [NOT NULL],
        feldname datentyp [NOT NULL],
        [PRIMARY KEY (feldname)],
        [FOREIGN KEY (feldname)]
);
```

„NOT NULL“ bedeutet, das Feld darf nicht leer sein. Es darf aber „0“ enthalten.

Datentypen

- INTEGER
- SMALLINT
- DECIMAL(länge, nachkommastellen)
- FLOAT(länge)
- CHAR (länge)

ALTER TABLE

Fügt eine Spalte zu einer Tabelle hinzu.

```
ALTER TABLE tabellenname ADD spalte datentyp;
```

Die Änderungen werden nicht sofort auf alle Datensätze angewendet sondern erst beim Zugriff auf einen Datensatz ergänzt.

DROP TABLE

Löscht eine Tabelle.

```
DROP TABLE tabellenname
```

Indizes

```
CREATE [UNIQUE] INDEX indexname
        ON tabellenname
        ( zeile [ASC|DESC],
          zeile [ASC|DESC],…)
```

Ist ein Index „unique“ (eindeutig), so kann man keine zwei Datensätze mit demselben Index erstellen.

```
DROP INDEX indexname
```

(6) Datenmanipulation (DML)

Einfache Abfragen

```
SELECT [DISTINCT] felder
FROM tabelle
[GROUP BY felder]
[WHERE bedingung]
[ORDER BY felder]
[HAVING bedingung];
```

Beispiel:

```
SELECT name
FROM adressen
WHERE stadt='Hamburg';
```

„DISTINCT" eliminiert doppelte Zeilen.

„GROUP BY" summiert alle Werte einer Gruppe.

„SELECT * FROM table" holt alle Einträge.

JOIN-Abfragen

Fragt alle Adressen mit gleich Städten aus zwei Tabellen ab:

```
SELECT adressen1.*, adressen2.*
FROM adressen1, adressen2
WHERE adressen1.stadt=adressen2.stadt;
```

In der SELECT-Anweisung müssen nicht alle Felder genannt werden, die mit WHERE verglichen werden. SELECT bestimmt nur die Ausgabe des Ergebnisses.

Aggregat-Funktionen

COUNT:
Anzahl Datensätze in einer Spalte

SUM:
Summe der Werte in einer Spalte

AVG:
Durchschnittswert der Spalte

MAX:
Maximalwert der Spalte

MIN:
Minimalwert der Spalte

```
SELECT COUNT(nachname) FROM adressen;
```

(Diese Funktionen liefern keinen skalaren Wert sondern eine 1x1-Tabelle zurück!)

WHERE

LIKE

SELECT…WHERE vorname LIKE 'Hans%C';

NULL

SELECT…WHERE status IS NULL;

Unterabfragen

SELECT…WHERE name IN (SELECT…WHERE);

SELECT…WHERE name = (SELECT…WHERE);

SELECT…WHERE EXISTS (SELECT…WHERE);

(Geschachteltes WHERE. Die inneren Abfragen werden zuerst evaluiert.)

UNION

SELECT…WHERE
UNION
SELECT…WHERE;

(Vereinigung der Ergebnisse)

Update-Operationen

```
UPDATE table
SET feld=ausdruck, feld=ausdruck,…
[WHERE bedingung];
```

```
DELETE FROM table
[WHERE bedingung];
```

```
INSERT INTO table (feld, feld,…)
VALUES (wert, wert,…);
```

(7) System-Katalog

Der Systemkatalog ist eine Tabelle aller Objekte einer Datenbank (base tables, views, Indizes, Benutzer, application plans, Zugriffsrechte). Bei DB2 sind es etwa 30 Objekte.

SYSTABLES
Tabellen/Views

SYSCOLUMNS
Jede Spalte jeder Tabelle

SYSINDEXES
Jeder Index

Objekte lassen sich auch über den System-Katalog löschen.

(8) Views

Views sind virtuelle Tabellen, die aus den Base-Tables abgeleitet werden. Views lassen sich also wie Tabellen auch mit SQL steuern.

```
CREATE VIEW viewname [(spalte, spalte,…)] AS
SELECT…FROM…WHERE;
```

Ein View wird nicht interpretiert, sondern als Abfrage gespeichert. SELECT ist auch mit Views möglich. Bind

vereinigt die View-Definition mit der gewünschten View-Abfrage und erhält so eine kombinierte Abfrage.

```
DROP VIEW viewname;
```

Nachteile von Views

Nicht alle Views-Operationen sind immer erfolgreich. Ist ein Feld z.B. als NOT-NULL definiert, der View zeigt aber dieses Feld gar nicht an, dann kann auch kein Datensatz per View-INSERT eingefügt werden.

Vorteile von Views

Derselbe Datenbestand kann für verschiedene Benutzer verschiedene Aspekte zeigen.

(9) Kapitel III: Das relationale Modell

Das relationale Datenbankmodell stellt die Mindestanforderungen an ein modernes Datenbanksystem dar.

Mathematische Begriffe

Relation:
: Tabelle (Menge von Mengen)

Tupel:
: Datensatz (Zeile)

Attribut:
: Feld eines Datensatzes (Spalte)

Kardinalität:
: Anzahl Datensätze

Grad:
: Anzahl Felder je Datensatz

Primärer Schlüssel:
: Eindeutige Identifikation eines Datensatzes in einer Tabelle

Domain:
: Wertebereich eines Feldes.

Domains

Nur Werte derselben Domain sind vergleichbar (das vermeidet Fehler).

Relationen

Eine Relation besteht aus:

- Heading
  (Menge von Feldern mit deren Domains)
  ➔ { (F1:D1) , (F2:D2) , … }
- Body
  (Menge von Tupeln aus Feldern und Werten)
  ➔ { (F1:wert1) , (F2:wert2) , … }

Eigenschaften einer Relation

- Es gibt keine doppelten Tupel
  (aufgrund der mathematischen Definition)
- Tupel sind ungeordnet
  (aufgrund der mathematischen Definition)
- Felder sind ungeordnet
- Feldwerte sind atomar
  (jedes Feld hat nur einen Wert)
- Relationen sind endlich
  (sonst kann man sie nicht speichern)

Normalisierung

Höhergradige (Grad > 1) Relationen enthalten nicht nur Werte sondern auch wiederum Relationen. In Tabellen sind nur atomare Werte in den Feldern erlaubt. Es wird dabei für jede mögliche Kombination von Werten eine einzelne Zeile in der Tabelle erzeugt (wie beim ausmultiplizieren).

Arten der Relationen

- Tabellen
  (base relations)
- Views
  (virtual relations)
- Snapshots
  (Views, die eine eigene Tabelle bilden)
- Abfrage-Ergebnisse
  (query results)

Relationale Datenbanken

Relationale Datenbank:
: Eine Datenbank, die sich dem Anwender als eine Sammlung von zeitlich-veränderlichen normalisierten Relationen darstellt.

(10) Integritätsregeln

Integritätsregeln stellen sicher, dass Felder einen sinnvollen Inhalt haben. Es ist z.B. nicht möglich, von einer Lagermenge „1“ den Wert „3“ abzuziehen.

Primärschlüssel

Primärer Schlüssel:
: Eindeutige Identifikation eines Datensatzes (Tupels).

Jede Relation hat einen Primärschlüssel. (Notfalls wird er aus der Zusammenfassung aller Feldinhalte gebildet.)

Entity-Integritätsregel

Kein Primärschlüssels eines Datensatzes darf NULL enthalten.

[…]

Fremdschlüssel (*foreign keys*)

Fremdschlüssel einer Tabelle entsprechen Primärschlüsseln einer anderen Tabelle, wenn diese beiden Tabellen inhaltlich zusammenhängen.

Referenz-Integritätsregel

Die Datenbank darf keine Fremdschlüssel enthalten, die nicht zum Primärschlüssel der korrespondierenden Datenbank passen.

Fremdschlüssel-Regeln

…

(11) Relationale Algebra

Relationale Algebra:

> Sammlung von Operationen, die aus einer oder zwei Relationen eine neue Relation erzeugen.

Operationen:

- RESTRICT
  (wählt ganze Tupel nach Kriterien aus – gleiche Funktionsweise wie SELECT)
- PROJECT
  (extrahiert einzelne Spalte aus allen Tupeln)
- PRODUCT
  (alle möglichen Kombinationen von Tupeln)
- UNION
  (Vereinigung der Tupel)
- INTERSECT
  (Schnittmenge der Tupel)
- DIFFERENCE
  (Differenzmenge der Tupel)
- JOIN
  (Fügt Tupel mit gleichen Feldern nach Kriterien zusammen)
- DIVIDE
  (Auswahl der Felder, die in einer binären Relation und einer unären vorhanden sind)

[Syntax für relationale Algebra]

IV Programmiersprachen

(Kapitel 1, 4, 5, 6, 7 und 8)

(1) Einleitung

Programmiersprache

Programmiersprache:
: Ein notationelles System zur Beschreibung von Berechnungen in einer durch Maschinen und Menschen lesbaren Form.

Berechnungen beziehen sich dabei auf den Begriff der „Berechenbarkeit“ durch eine Turing-Maschine. Nach der Churchschen These kann eine Turing-Maschine jede denkbare Berechnung ausführen.

Maschinenlesbarkeit bedeutet, dass die Sprache eine einfache compilierbare (kontextfreie) Struktur hat.

Menschenlesbarkeit bezieht sich auf die Ähnlichkeit zwischen Programmiersprachen und natürlichen Sprachen. Die Programmiersprache muss von den Details eines Rechensystems abstrahieren.

Algorithmus:
: Eine präzise endliche Beschreibung eines allgemeinen Verfahrens unter Verwendung elementarerer Ausführungsschritte.

Abstraktionen

Datenabstraktion:
: Abstraktion von Eigenschaften der Daten.
(➔Datentypen)

Kontrollabstraktion:
: Abstraktion von Eigenschaften des Kontrollflusses.
(➔Schleifen, Sprunganweisungen)

Datenabstraktionen

Fundamentale Abstraktionen:
: Abstraktion von der internen Darstellung allgemeiner Datenwerte.
(➔Speicherung von ganzen Zahlen als Zweierkomplement im Speicher.)
Variablen: Namen der Speicherplätze
Datentyp: Art der Daten

Strukturierte Abstraktionen:
: Datenstruktur: Abstraktion von Ansammlungen zusammengehöriger Datenwerte

Typdeklaration: Definition neuer Datenstrukturen aus bekannten Datenstrukturen (strukturierte Typen)

Modulabstraktionen:
: Methoden zum Arbeiten mit einem Datentyp werden gekapselt, damit die Änderungen am Datentyp zentral gehalten werden können.

Kontrollabstraktionen

Fundamentale Abstraktionen:
: Anweisungen einer Sprache
(➔Zuweisungen, Sprunganweisungen)

Strukturierte Abstraktionen:
: Zusammenfassung mehrere Anweisungen (Blöcke, Prozeduren), die eventuell nur bei Erfüllung bestimmter Bedingungen ausgeführt werden (if, switch). Steuerung des Kontrollflusses.

Modulabstraktionen:
: Logisch zusammengehörende Anweisungen werden zentralisiert, damit man sie leichter wiederverwenden und warten kann.

Berechnungsparadigmen

Imperative Programmierung:
: Sequentielle Ausführung von Befehlen.
(➔BASIC, C, FORTRAN…)

Funktionale Programmierung:
: Grundmechanismus ist die Anwendung von Funktionen im mathematischen Sinn. Hier gibt es keine Variablen. Schleifen werden durch Rekursion realisiert. Funktionen werden verschachtelt.
(➔Lisp, hat aber Variablen)

Logische Programmierung:
: Einsatz von symbolischer Logik. Es werden Eigenschaften von Ergebnissen beschrieben. Die Kontrolle und Auswertung übernimmt das System.
(➔Prolog)

Sprachdefinition

Syntax:
: Beschreibung der Struktur einer Sprache (Art und Weise, wie Teile der Sprache kombiniert werden können).
(➔if ➔ then ➔ else)

Die Syntax wird unter Verwendung kontextfreier Grammatiken angegeben (➔EBNF).

Semantik:
: Bedeutung der einzelnen Anweisungen.

Sprachübersetzung

Interpreter:
: Übersetzer, der das Programm direkt ausführt.

Compiler:
: Übersetzt das Programm in eine für das System direkt ausführbare Datei.

Beim Übersetzen wird der Quellcode zunächst lexikalisch analysiert (*Scanner*). Dabei werden die Zeichen in Token übersetzt.

Bei der syntaktischen Analyse (*Parser*) wird die Struktur der Folge von Token überprüft.

Die semantische Analyse erzeugt das Zielprogramm, indem es die Bedeutung der Anweisungen ermittelt.

Statische Eigenschaften:
: Eigenschaften, die zur Compile-Zeit festliegen

Dynamische Eigenschaften:
: Eigenschaften, die erst zur Laufzeit festgelegt werden

Sprachentwurf

Wichtige Ziele beim Entwurf von Programmiersprachen:

- Lesbarkeit
  (durch Mensch und Maschine)
- Komplexitätskontrolle
  (Abstraktionen zur besseren Übersicht in großen Programmen)

(2) Syntax

Lexikalische Struktur von Programmiersprachen

Token:
: „Wörter“ einer Programmiersprache

Reservierte Wörter:
: Schlüsselwörter wie „begin“, „if“ oder „goto“ (dürfen nicht als Bezeichner verwendet werden)

Konstanten/Literale:
: Numerische (42) und String-Konstanten („hello world“)

Sonderzeichen:
: : := +

Bezeichner:
: x3, result_code

Beim Scanning des Quelltextes werden immer die längsten möglichen Zeichenketten zu einem Token zusammengefasst und dann dem Parser übergeben. Token-Begrenzer sind z.B. Leerzeichen oder Zeilenenden.

Kontextfreie Grammatiken und Backus-Naur-Form (BNF)

BNFen sind äquivalent zu *kontextfreien*[2] *Grammatiken* und werden eingesetzt als:

- BNF
- EBNF (erweiterte BNF)
- Syntaxdiagramm

Formale Sprachen (so auch Programmiersprachen) können von Grammatiken erzeugt werden. Aus der Startvariable wird jedes gültige Wort der Sprache abgeleitet. Bei der BNF haben die Produktionen (Grammatikregeln) die Form:

<Nonterminal> ::= Terminal1 | Terminal2

oder

<Nonterminal> ::= <Nonterminal>

„::=“ und „|“ sind *Metasymbole*.

Deklaration einer Zahl in BNF:

<Zahl> ::= <Zahl> <Ziffer>
<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5…

Ableitungsbäume und abstrakte Syntax

Syntaxgesteuerte Semantik:
: Die Syntax hängt eng mit der Semantik zusammen. Bei der Syntax „a+b“ lässt sich auf eine Addition schließen.

Ableitungsbaum:
: Grafische Darstellung des Ableitungsvorgangs als Baum.

Syntaxbaum:
: Ableitungsbaum ohne Nonterminale.

Mehrdeutigkeit, Assoziativität und Vorrang

Mehrdeutige Grammatik:
: Es sind mehrere Ableitungsbäume möglich.
(➔a+b*c ; wie soll geklammert werden?)

linksrekursive Regel:
: <Ausdruck> ::= <Ausdruck> + <Term>
Es wird also nach links assoziiert.

EBNFs und Syntaxdiagramme

Zusätzlich erlaubte Regeln in der EBNF:
: <Ausdruck> ::= <Term> [<Term>]
(der letzte Term ist optional)

[2] Ein Nonterminal kann jederzeit und unabhängig vom „Kontext“ durch die rechte Seite einer Produktion ersetzt werden.

<Ausdruck> ::= <Term> {<Term>}
(der letzte Term kommt beliebig oft vor)

Deklaration einer Zahl in EBNF:
<Zahl> ::= <Ziffer> {<Ziffer>}
<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5…

Syntaxdiagramm:
Darstellung der möglichen Ableitungen als Graph. Quadrate sind Nonterminale und Kreise sind Terminalsymbole.

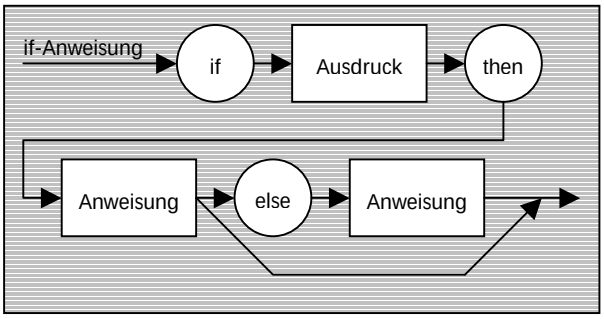


Abbildung 1: Syntaxdiagramm

Verfahren und Werkzeuge zur Syntax-Überprüfung

Grammatik als BNF beschreibt die erlaubten Tokenfolgen. Der Parser erkennt, ob eine Zeichenkette in der Grammatik gültig ist oder nicht.

Bottom-Up-Parser

Versucht, die gegebene Zeichenkette in den rechten Seiten der Produktionen zu finden und in ein Nonterminal umzuwandeln.

Top-Down-Parser

Ausgehend von einem Nonterminal versucht der Parser, die Zeichenkette aus den Produktionen zu erzeugen.

Rekursiv-absteigendes Parsen / prädikatives Parsen

Jedes Nonterminal wird in eine Prozedur umgewandelt, deren Aktionen sich auf die rechten Seiten der Regeln beziehen. Die Prozeduren rufen sich gegenseitig rekursiv auf (daher „rekursiv-absteigend“). Der Parser untersucht in diesen Routinen den Eingabestrom der Token auf Gültigkeit (➔*Einzelsymbol-Lookahead*). Diese Parser, die aufgrund des jeweils nächsten Tokens sofort Entscheidungen treffen, heißen *prädikative Parser*.

Probleme:

- Linksrekursionen wie
  <Ausdruck> ::= <Ausdruck> + <Term>
  würden zu Endlosschleifen im Parser führen, da sich die Prozedur immer wieder selbst aufruft. Um diese Linksrekursionen zu entfernen, werden diese Ausdrücke vor dem Parsen ersetzt durch
  <Ausdruck> ::= <Term> {<Term>}

  Rechtsrekursionen führen zu keinen Problemen.
- Die rechten Seiten der Produktionen müssen präfixfrei sein, damit sie in eine deterministische Prozedur umgewandelt werden können. Notfalls werden optionale Ausdrücke ([und]) verwendet (➔if <Bedingung> then <Anweisung> [else <Anweisung>]). Es wäre sonst möglich, eine Form mit und eine Form ohne „else“-Teil als gültig zu definieren. Dieser Vorgang heißt *Linksfaktorisierung*.

Die Erkennung passiert nur durch Einzelsymbol-Lookahead. Damit prädikative Parser bei Alternativen (<Ausdruck> ::= a | b | c) richtig entscheiden können, müssen die First-Mengen eines Ausdrucks disjunkt sein.

First-Funktion:
First(String) ergibt das erste Token des Strings. First wird solange rekursiv abgearbeitet, bis nur noch Terminale in der Ergebnismenge enthalten sind.

Außerdem dürfen keine Token, die am Anfang eines optionalen Teils stehen, dahinter noch einmal auftauchen. Dies ist gegeben, wenn die Follow-Mengen eines optionalen Ausdrucks disjunkt sind.

Follow-Funktion:
Follow(String) ergibt die Menge der Token, die potentiell hinter String stehen können.

Lexik - Syntax - Semantik

Es ist möglich, in einer Programmiersprache eigene Token lexikalisch mittels *regulären Ausdrücken* zu definieren.

Reservierte Wörter:
Unveränderliche Befehle und Konstrukte einer Sprache. („while“, „else“…)

Vordefinierte Bezeichner:
Bezeichner, die vom Benutzer geändert werden könnten, was aber nicht sehr nett wäre. („true“, „false“…)

(3) Semantik

Spezifikation von Semantik durch:

- Referenz-Handbuch
- Übersetzer, der die Sprache definiert
- formale Definition

Attribute, Bindung und semantische Funktionen

Bindung:
Assoziation eines Namens (➔Variable, Prozedur) mit Attributen.
„const n=5“ weist n die Attribute „Konstante“ und „Wert ist 5“ zu.
Statisches Attribut: Compile-Zeit oder Link-Zeit oder Ladezeit
Dynamisches Attribut: Laufzeit

Deklarationen, Blöcke und Gültigkeitsbereich

Explizite Deklaration:
Festlegung von Attributen im Quelltext.

Implizite Deklaration:
Festlegung von Attributen durch die Sprache (TAG$ ist ein String in BASIC).

Globale Deklarationen:
Allgemeiner Sichtbarkeitsbereich im Programm, wenn nichts anderes angegeben.

Lokale Deklarationen:
Überlagerung globaler Deklarationen.

Gültigkeitsbereich einer Deklaration:
Abschnitt des Programms, in dem die durchgeführten Bindungen erhalten bleiben.
(➔blockstrukturierte Sprache)

Symboltabelle

Symboltabelle:
Vom Compiler verwaltete Abbildung von Namen auf statische Attribute.

Dynamischer Gültigkeitsbereich:
Speicherzuteilung für Variablen wird zur Laufzeit festgelegt (wenn die Kontrolle auf eine Variable trifft).

Statischer Gültigkeitsbereich:
Speicherzuteilung für Variablen wird zur Compilierzeit festgelegt.

Speicherzuteilung, Lebensdauer, Umgebung

Umgebung:
Bindung von Namen an Speicheradressen.

Speicher:
Bindung von Speicheradressen an Werte.

Beim Interpreter sind Symboltabelle und Umgebung identisch.

Globalen Variablen wird der Speicher statisch zugeteilt. Speicher für lokale Variablen wird nach Bedarf (➔beim Erreichen der Deklaration in einer Prozedur) angefordert.

Aktivierung [Prozedur]:
Aufruf einer Prozedur. Der Bereich des zugeteilten Speichers für lokale Variable heißt *Aktivierungssegment*.

Lebensdauer:
Dauer der Speicherzuteilung einer Variable in der Umgebung.

Dereferenzierung:
Auflösen eines Zeigers in seinen eigentlichen Wert.

Casts:
Zuweisung eines Datentyps zu einer Variablen (➔bei dynamischer Belegung von Speicher nötig).

Speicherklassen

- statische Speicherzuteilung
  (für globale Variablen)
- automatische Speicherzuteilung
  (für lokale Variablen in Prozeduren)
- dynamische Speicherzuteilung
  (bei manueller Belegung mittels *malloc*)

Variablen und Konstanten

Variablen

Objekte, dessen gespeicherter Wert sich während der Ausführung des Programms ändern kann.
Zu einer Variablen gehören:
- Speicherplatz
- Wert
- Attribute (Länge, Datentyp…)

l-Wert (*l-value*):
Speicherplatz einer Variablen

r-Wert (*r-value*):
Wert einer Variablen

In C ist z.B. &x die Speicheradresse der Variablen x und *pointer der Wert der Variablen, die an der Speicheradresse pointer gespeichert ist.

Zuweisung:
x:=y (Wert wird kopiert)

gemeinsame Zuweisung:
x:=y (x und y zeigen auf dieselbe Speicheradresse)

Konstanten

Konstante:
Objekt, das für die Dauer des Programms einen festen Wert hat.
(Konstanten belegen gewöhnlich keinen Speicherplatz. Der Compiler ersetzt jedes Auftreten der Konstanten direkt durch den Wert.)

Aliasnamen, hängende Referenzen und Garbage

Aliasnamen

Zwei Namen verweisen auf dasselbe Objekt. Ein Aliasname kann entstehen, wenn z.B. Zeiger statt der Werte zugewiesen werden (pointer2:=pointer1 zeigen dann beide auf *pointer1).

Seiteneffekte sind möglich, wenn sich unerwartet Werte verändern.

Hängende Referenzen

Zugriff auf ein Objekt nach seiner Lebensdauer.

Seiteneffekte sind möglich, wenn Speicher für ein Objekt freigegeben wurde und bereits anderweitig belegt wurde. Dann greift die Referenz auf fremde Daten zu.

Garbage

Garbage (*„Speichermüll“*) entsteht, wenn Speicher in einer Umgebung nicht wieder freigegeben wird.

Es kann zwar nicht zu hängenden Referenzen kommen, aber es wird Speicherplatz verschwendet.

Einige Systeme geben Speicher während der *garbage collection* automatisch wieder frei.
(➔dynamische Laufzeitumgebungen)

Auswertung von Ausdrücken

Ausdruck:
ergibt einen Wert

Anweisung:
haben Seiteneffekte und Rückgabewerte

Infix-Notation:
1+2
(➔C)

Präfix-Notation:
+(1 2)
(➔LISP)

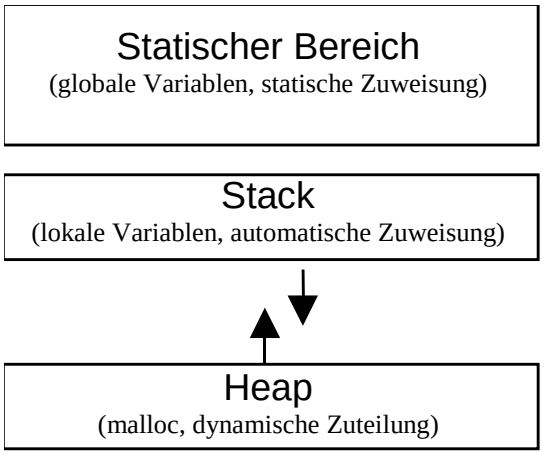
Postfix-Notation:
(1 2) +

Abgekürzte Auswertung:
„x and y and false“ wird abgebrochen, sobald x „false“ ist und die Berechnung nicht mehr „true“ ergeben kann.

(4) Datentypen

Datentyp:
eine Menge von Werten mit darauf definierten Operationen

Typdeklaration:
Definition eines neuen Datentyps aus bekannten Datentypen:
„type iarray=array[1..10] of integer“



Typüberprüfung:
Prozess während des Übersetzens, der die Typinformationen in einem Programm auf Konsistenz überprüft. (Kompatibilitätsregeln)

Typinferenzregel:
Gibt an, welchen Typ ein Ausdruck hat, in dem verschiedene Typen verknüpft werden
(➔int+real=real)

streng typisierte Sprache:
Alle Objekte der Sprache sind wohldefiniert und es gibt einen Satz von Regeln für Typäquivalenz und Typinferenz.
Alle Typfehler können zur Übersetzungszeit bestimmt werden.

Einfache Typen

Aus den in der Sprache *vordefinierten Typen* lassen sich andere gewünschte Typen herleiten.

Aufzählungstypen (Ordinaltypen) (type colors=(red,green,blue)) und *Unterbereichstypen* (type digit=0…9) sind *einfache Typen*. Sie ordnen jedem Wert einen Vorgänger und Nachfolger zu. (Realzahlen sind keine Ordinaltypen!)

Typkonstruktoren

Typkonstruktoren erstellen neue Datentypen auf Basis bekannter Datentypen.

Kartesisches Produkt

Kartesische Produkte sind Tupel aus verschiedenen Werten. Einen Typ „INTEGER x BOOLEAN“ kann man in den meisten Programmiersprachen als Record definieren.

```
TYPE IntBool = RECORD
        i : INTEGER;
        b : BOOLEAN;
```

Vereinigung

Vereinigungen von Mengen erlauben verschiedene Typen an einer Stelle. In C ist das über Unions gelöst.

```
typedef union
{
        int i;
        float r;
} IntOrFloat;
```

In Programmiersprachen ohne Unions werden Diskriminatoren eingesetzt (*diskriminierte Vereinigung*), um im Datentyp per Fallunterscheidung mit einem Flag festzulegen, welcher Datentyp gemeint ist.

Teilmengenbildung

Teilmengen von Datentypen schränken den Gültigkeitsbereich ein.

```
TYPE digit = INTEGER [0..9]
```

Teilbereiche von Arrays nennt man Slices.

Potenzmengenbildung

Mengen von Teilmengen.

```
TYPE digits = SET OF [0..9]
```

(hier wären die Bits gesetzt oder gelöscht je nachdem oder die Ziffer in der Menge ist oder nicht)

Funktionen

Ein *Funktionstyp* bildet eine mathematische Funktion über eine Funktionsprozedur nach.

Ein *Arraytyp* ist ein Array, das bei der Initialisierung mit Funktions- und Ergebniswerten vorbelegt wird (nur für Ordinaltypen).

```
TYPE digitToInt = ARRAY [0..9] OF INTEGER;
```

Zeiger

Zeiger beinhalten Speicheradressen, in denen Werte gespeichert sind. In C:

```
int *x;
*x = 20;
```

(Achtung! Hier wird kein Speicher allokiert!)

Bei verketteten Listen zeigt ein Pointer auf die nächste Struktur – die Struktur selbst wird nicht in den Typ mit aufgenommen.

Andere Typkonstruktoren

Strings: char string[30];

Nichttyp: void *pointer;

Typäquivalenz

Fragestellung: „Wann sind zwei Typen gleich?“

Strukturäquivalenz:
: Bei zwei Typen stehen dieselben Datentypen an derselben Stelle.

Namensäquivalenz:
: Bei der Typdefinition müssen beide Typen denselben Namen haben.
(In Algol müssen in einer Record-Definition die Namen der einzelnen Variablen identisch sein.)

Deklarationsäquivalenz:
: Zwei Typen lassen sich auf dieselbe Definition zurückführen. Eine Neudefinition gilt hier nicht.

Typüberprüfung

Dynamische Typüberprüfung:
: Interpreter und teilweise auch Compiler (LISP)

Statische Typüberprüfung:
: Compiler

Typinferenz:
: Der Ausdruck „Integer + Integer“ ergibt wieder einen Wert vom Typ Integer.

Typkompatibilität

Zwei Typen sind *kompatibel*, wenn man sie miteinander kombinieren kann. In Perl sind alle Typen kombinierbar.

Überlappende Typen

Unterbereichstypen können sich überlappen ([1..5] und [3..8]). Hier wird die Typprüfung bei einer Operation bis zur Laufzeit zurückgestellt.

Implizite Typen

Bei Konstanten entscheidet der Compiler über den passenden Typen aus dem Kontext.

(In Perl sind alle Typen implizit.)

Gemeinsam verwendete Operationen

Überladene Operatoren:
: Die Operation hat verschiedene Bedeutungen je nach Typ.
(‚+‘ gibt es für Real und Integer)

[Mehrfach typisierte Objekte]

Typumwandlung

Implizite Typumwandlung:
: Bei Verwendung verschiedener Typen in einer Operation entscheidet der Compiler über den Typ des Ausdrucks.

Bei ‚1+3.5‘ wird 1 zur Realzahl gemacht.

C ist nicht stark typisiert und würde einen Ausdruck mit maximaler (real) Genauigkeit berechnen und dann gerundet (int) zuweisen.

Casts (*explizite Typumwandlung*):
Explizite Angabe des gewünschten Typs.
➔ (int) 2.52

(5) Steuerung des Kontrollflusses

Kontrollstrukturen unterbrechen den sequentiellen Kontrollfluss, der Anweisungen nacheinander ablaufen lässt.

Bedingte Anweisungen

If-Anweisungen

if <Boolescher-Ausdruck> then <Anweisung>
[else <Anweisung>]

Der Boolesche Ausdruck wird auch Wächter genannt (bewachte If-Anweisung), weil er die Ausführung der Anweisungen steuert.

Dangling-Else-Problem (in Pascal):
Bei geschachtelten If-Then-Else-Anweisungen ist der Else-Zweig mehrdeutig.
➔if a then if b then c else d
(Bei Pascal wird das else immer dem am nächsten stehenden If zugeordnet.)

Case-Anweisungen

Spezieller Fall der If-Anweisung, wobei der Wächter durch Ordinalwerte definiert wird.

In C heisst die Anweisung „switch“.

Schleifen und Variationen von while

In Schleifen wird eine Folge von Anweisungen nach gegebenen Kriterien wiederholt.

```
while bedingung do anweisung
```

(Anweisung wird nie ausgeführt, wenn die Bedingung falsch ist.)

```
repeat anweisung while bedingung
```

(Anweisung wird einmal ausgeführt, wenn die Bedingung falsch ist.)

```
loop; anweisung; exit if bedingung; end;
```

```
for i=0 to 10 do; anweisung; end;
```

Die GOTO-Kontroverse

GOTO-Anweisungen können den Quelltext verwüsten, erlauben aber auch einen einfachen Rücksprung aus einer tiefen Verschachtelung.

Prozeduren und Parameter

Eine Prozedur abstrahiert eine Gruppe von Operationen und wird bevorzugt eingesetzt, wenn Programmteile mehrfach verwendet werden.

Die Prozedur besteht aus:

- Name
- Parameterspezifikation
- Rumpf (durchzuführende Operationen)

Eine Funktion besteht zusätzlich aus:

- Return-Anweisung(en)

Prozedursemantik

Entsprechend der Umgebung werden lokale Objekte eines Blocks (hier einer Prozedur) in einem *Aktivierungssegment* gehalten. Andere (nicht-globale) Variablen müssen als Parameter an die Prozedur übergeben werden. Wird die Prozedur verlassen, werden die vorigen Gültigkeitsbereiche wiederhergestellt.

Mechanismen der Paramterübergabe

Call-By-Value:
Beim Aufruf werden die Parameter (falls es Ausdrücke sind) berechnet und als Konstanten übergeben. Die Prozedur arbeitet mit Kopien der Originalvariable.

Call-By-Reference:
Es wird die Speicheradresse der Variablen übergeben. Die Prozedur arbeitet direkt auf der Variablen.

[Call-By-Value-Result]

Call-By-Name:
Verzögerte Berechnung während der Laufzeit. Es wird der Name der Variablen als String an Unterroutinen übergeben.

Prozedurumgebungen, Aktivierungen und Speicherzuteilung

Statische Programmumgebungen (➔FORTRAN)

Alle Speicherzuteilungen passieren zur Ladezeit. Es gibt keine geschachtelten Prozeduren oder Rekursionen. Jede Prozedur bekommt ein Aktivierungssegment zugewiesen.

Jedes Aktivierungssegment besteht aus:

- Platz für lokale Variablen
- Platz für übergebene Parameter
- Rücksprungadresse
- Zwischenspeicher für Ausdrucksauswertung

Stack-basierte Laufzeitumgebungen (➔ALGOL, PASCAL, Modula-2, C)

Beim Betreten eines Blocks wird auf dem Stack eine neue Umgebung erzeugt und beim Verlassen wieder freigegeben.

Umgebungszeiger:
: Zeiger auf die aktuelle Umgebung.

Kontrollverbindung:
: Zeiger auf die übergeordnete Umgebung (um die Kontrolle zurückgeben zu können).

Zugriffsverbindung:
: Zeiger auf die globale Umgebung (für globale Variablen).

Zugriffsverkettung:
: Verfolgen mehrere Kontrollverbindungen, um auf Variablen übergeordneter Verbindungen zugreifen zu können.

Dynamisch berechnete Prozeduren und vollständig dynamische Umgebungen (➔LISP)

In einer vollständig dynamischen Umgebung werden Objekte automatisch vernichtet, wenn sie nicht weiter erreichbar ist. Das passiert über Referenzzähler oder Garbage Collection.

Ausnahmebehandlung

Bei schweren Fehlern während des Programmlaufs wird eine Ausnahme (*exception*) ausgelöst. Dann wird die passende Ausnahmebehandlungsroutine (*exception handler*) ausgeführt, die das Programm kontrolliert abbricht (*Terminationsmodell*) oder weiter ausführt (*Wiederaufnahmemodell*) und einen Fehler meldet.

Ausnahmen sind quasi „Software-Interrupts“.

(6) Abstrakte Datentypen (ADTs)

Ein abstrakter Datentyp ist ein benutzerdefinierter Datentyp (eine Struktur von mehreren Basistypen) und darauf definierten Operationen. Der Datentyp ist dann nur noch über diese Operationen zugreifbar (Kapselung). (Bei OO-Sprachen sind die Objekte nur über die definierten Methoden zugreifbar.)

Vorteile der Kapselung:

- Modifizierbarkeit
  (Die Implementation der Operationen kann verändert werden ohne den Rest des Programms zu beeinflussen.)
- Wiederverwendbarkeit
  (Die Implementation kann direkt in anderen Programmen verwendet werden.)
- Sicherheit
  (Details der Implementation sind vor dem Benutzer versteckt. ➔Information Hiding)

Kapselung:
: zentrale Definition des Typs

Information hiding:
: Verstecken von Details der Implementierung

Die algebraische Spezifikation abstrakter Datentypen

Die syntaktische Spezifikation gibt an, wie die Operationen mit diesem Datentyp aussehen:

summe: complex x real ➔ complex

Die semantische Spezifikation wird durch algebraische Axiome/Gleichungen aufgestellt:

realpart(x+y) = realpart(x) + realpart(y)

Konstruktor:
: Abbildung in einen Datentypen.

Desktruktor:
: Abbildung eines Datentypen in nichts.

Inspektor:
: Abbildung in einen anderen Datentypen (➔boolesch)

Parametrisierter Datentyp:
: Eine Variable (der Parameter) bleibt unspezifiziert und kann später durch jeden beliebigen Typen ersetzt werden.

Überladen und Polymorphismus

Überladene Operation:
: Unterschiedliche Operationen mit demselben Namen (➔Addition für Integerzahlen und für Realzahlen).

Polymorphe Operation:
: Operation mit beliebigen Datentypen (die nicht im Voraus bekannt sein müssen)

Parametrisierte ADTs

Ein Datentyp arbeitet mit (beliebigen) anderen Datentypen als Parametern. Ein Warteschlagen-Datentypen könnte „queue(element)“ erlauben, wobei „element“ eine beliebige Datenstruktur ist.

[Module und getrennte Übersetzung]

Probleme bei abstrakten Datentypmechanismen

Module sind keine Typen

Modula-2 kennt keine vollwertigen ADTs. Ein Modul exportiert den Typen und die Operationen. Es gibt keine Kapselung.

[Module sind statisch]

Inadäquate Typüberprüfung

Wenn man mit Zeigern statt mit Basistypen arbeitet, kann es zu Aliasnamen kommen, was die Module nicht abfangen können. Außerdem ist die Speicherverwaltung nicht klar (wann Speicher belegt oder freigegeben wird).

[Die Mathematik abstrakter Datentypen]

V Übungsfragen zu Betriebssystemen

Was sind die Hauptaufgaben des Betriebssystems?

- Virtuelle Maschine (Systemaufrufe zur Steuerung der Hardwarekomponenten)
- Ressourcenverwaltung und –koordination (Prozesse, Speicher, Dateien, I/O-Geräte)

Was unterscheidet Hardware- und Softwareinterrupts?

Hardwareinterrupts:
Prozess wird sofort verdrängt, Aufruf des Programms am Interruptvektor

Softwareinterrupts:
Kontrollfluss wird unterbrochen, Aufruf des Interrupt-Handlers innerhalb dieses Prozesses

Welche drei Zustände kann ein Prozess haben? Wann kommt es zu Zustandsübergängen?

Rechnend, bereit, blockiert…

Was ist Spooling?

Zugriff mehrerer Prozesse auf unteilbare Ressource über Spool-Verzeichnis, Daemon bearbeitet Spool-Dateien

Warum benutzt man Multitasking?

- Prozesse warten häufig auf I/O, während Sie die CPU nicht voll ausnutzen
- Mehrere Benutzer arbeiten auf einem System

Wie funktioniert das Scheduling? Was löst es aus?

Zeit-Interrupts der Hardware verdrängen den aktiven Prozess und aktivieren den Scheduler

Was ist ein virtueller Prozessor?

Gedachter Teil des realen Prozessors, der ausschließlich einem Prozess zur Verfügung steht.

Was ist die Prozesstabelle?

Liste aller Prozesse und der Information, die der Scheduler braucht, um den Prozess wieder zu reaktivieren (Register, Programmzähler, Stackpointer, belegte Ressourcen).

Was ist der Interrupt-Vektor?

Einsprungadresse, wenn ein Hardware-Interrupt durch ein Gerät ausgelöst wird. Jedes Gerät hat einen eigenen Vektor (Speicheradresse). Die Hardware verdrängt den Prozess und aktiviert den Interrupt-Handler.

Wie realisiert man Prozesskommunikation (IPC)?

Einprozessorsystem: gemeinsamer Speicher

Verteiltes System: Nachrichtenaustausch

Warum ist busy waiting schlechter als blockieren?

Blockieren erlaubt Ablauf anderer Programme. Busy waiting verbraucht CPU-Zeit.

Was sind zeitkritische Abläufe (race-conditions)?

Mehrere Prozesse arbeiten auf denselben Daten und durch ungünstiges Umschalten (Scheduling) der Prozesse kommt es zu falschen Annahmen, was zu falschen Endergebnissen führt (➔kritischer Bereich).

Wie wird Synchronisation realisiert?

Semaphore, Monitore, Nachrichtenaustausch…

Was ist das Erzeuger-Verbraucher-Problem und wie kann man es lösen?

Problem eines begrenzten Puffers, der vom Erzeuger gefüllt und vom Verbraucher geleert wird. Hauptproblem: ungeteilter Zugriff auf Füllungsvariable.

Lösungen:

- SLEEP/WAKEUP-Signalen
- Semaphore (full, empty, mutex)
- Monitoren (WAIT/SIGNAL)
- Nachrichtenaustausch (Mailbox, Rendezvous)

Was ist das Philosophenproblem?

5 Philophen, je 2 Gabeln, Semaphore, Verhungern, Fairness…

Was ist das Leser-Schreiber-Problem?

Modellierung von Datenbankzugriffen. Es darf nur geschrieben werden, wenn kein Prozess liest. Es dürfen aber mehrere Prozesse gleichzeitig lesen.

Was ist der Scheduler?

Teil des Betriebssystem, der mittels des Scheduling-Algorithmus über die Zuteilung von Rechenzeit entscheidet. Er wird über Hardware-Interrupts (je Tick) initiiert.

Welche Scheduling-Algorithmen gibt es?

Round-Robin, Zweistufiges Scheduling, Prioritätsscheduling, Garantiertes Scheduling, Shortest-Job-First

Was ist Round-Robin?

Scheduling-Algorithmus. Quantum. Fair.

Wann verwendet man mehrere Warteschlangen?

Wenn die Prozessumschaltung lange dauert (z.B. bei Swapping), weil längere Auträge mit der Zeit höhere Quanten erhalten.

Warum setzt man Shortest-Job-First praktisch nie ein?

Weil die Bedienzeit des Auftrags vor der Ausführung bekannt sein muss. Geht nur bei sich wiederholenden Stapelaufträgen.

Welchen Vorteil hat Shortest-Job-First?

Die mittlere Verweilzeit wird minimiert.

Was ist die Aufgabe des Speicherverwalters?

Verwaltung von freiem und belegtem Speicher. Paging.

Was ist der Unterschied zwischen Swapping und Paging?

Swapping: Ein-/Auslagern von Prozessen

Paging: Ein-/Auslagern von Speicherseiten (MMU)

Was ist das Relokationsproblem?

Zur Compilierzeit weiß der Linker noch nicht, in welchem Adressraum der Prozess später ausgeführt wird. Die Relokation sorgt zur Ladezeit für eine Modifikation des Programms, damit er am Zielort lauffähig ist.

Software-Relokation: Addieren der Startadresse

Hardware-Relokation: Basis-/Grenzregister, Segmentierung

Wie kann dafür gesorgt werden, dass ein Programm nicht auf fremde Speicherbereiche zugreift?

- Basisregister + Grenzregister der Hardware
- Segmentierung

Welche Speicherzuteilungsalgorithmen gibt es?

First Fit (einfach, schnell)

Next Fit (einfach, schnell)

Best Fit (langsam, kleine unbrauchbare Löcher)

Quick Fit (umständlich)

Was ist der Unterschied zwischen interner und externer Fragmentierung?

Intern:

Weniger Speicher benötigt als ein Seitenrahmen groß ist

Extern:

Verteilte Speicherlöcher (Summe des freien Speichers größer als das größte Speicherloch)

Was besagt bei der Speicherverwaltung die 50%-Regel?

Speicherlöcher können verschmolzen werden, Prozesse im Speicher aber nicht. Dadurch sind über die Zeit nur halb so viele Löcher wie Prozesse vorhanden.

Was ist virtueller Speicher gegenüber physikalischem Speicher?

Hauptspeicher ist Teil des adressierbaren virtuellen Speichers (virtueller Speicher = Hauptspeicher + Auslagerungsspeicher).

Was ist die MMU?

Memory-Management-Unit. Umwandlung virtueller Adressen in reale Adressen (obere Bits) oder Auslösen von Seitenfehlern. Befindet sich zwischen CPU und Speicherbus.

Was macht der Seitenersetzungsalgorithmus?

Ist kein Platz für das Einlagern einer Seite in den Hauptspeicher, so lagert er eine selten benötigte Seite wieder aus.

Was ist die Seitentabelle?

Liste der virtuellen Speicherseiten und wo sie sich befinden (Hauptspeicher oder Auslagerungsspeicher).

Warum ist eine einstufige Seitentabelle nicht effektiv?

Viele Speicherbereiche sind häufig nicht belegt, die Seitentabelle nimmt dann nur unnötig Platz in Anspruch. Bei einer mehrstufigen Seitentabelle ist nur mindestens die Tabelle der ersten Stufe im Speicher.

Was ist der Assoziativspeicher der MMU?

Cache für häufige MMU-Zugriffe auf dieselben Seiten.

Was macht der Seitenersetzungsalgorithmus?

Ist kein Platz zum Einlagern von Seiten, muss eine Seite ausgelagert werden. Welche, das entscheidet der Seitenersetzungsalgorithmus.

Welche Seitenersetzungsalgorithmen gibt es?

NRU, LRU, FIFO, Second Chance/Uhr

Was ist NRU?
(Wie kommt es zu Klasse-1-Seiten?)

NRU beachtet die R-(referenced) und M-(modified) Flags der Seiten. Je niedriger die Klasse, um so eher wird ausgelagert.

Jeder Zeit-Interrupt löscht die R-Flags. So werden Klasse-3-Seiten zu Klasse-1-Seiten.

Was ist Segmentierung? Welche Vorteile hat sie?

Zweidimensionale Adressierung. Veränderbare Größe, automatische Defragmentierung möglich, Schutzflags, gemeinsamer Zugriff (shared libraries).

Dateisystem

Welche Dateitypen gibt es?

Reguläre Dateien, Verzeichnisse, Spezialdateien (block/character), Links.

Vorteile von Dateien?

Persistente Speicherung, große Datenmengen, Zugriff durch mehrere Prozesse möglich

Wie kann auf Dateien zugegriffen werden?

Sequentiell (Magnetbänder), direkt (Festplatten).

Was sind speicherabgebildete Dateien?

Projektion einer Datei in den virtuellen Speicher. Größe kann sich nicht verändern.

Welche Arten der Allokation von Dateien gibt es?

Kontinuierliche, verkettete Listen, verkettete Listen mit Index (MS-DOS FAT), i-nodes (ufs)

Wie funktionieren i-nodes?

Erste i-node speichert Verweise auf Datenblöcke oder weitere i-nodes. UNIX verwendet mehrfach indirekte i-nodes (Zeiger auf Zeiger auf Zeiger auf Datenblöcke).

Haben Dateisysteme immer eine Baumstruktur?

Links führen zu Zyklen.

Was ist der Unterschied zwischen symbolischen Links und Hardlinks?

Symbolischer Link ist eine spezielle Datei, die auf eine andere verweist.

Beim Hardlink zeigt die i-node direkt auf die andere Datei. Referenzzähler.

Wie wird der belegte Speicher einer Festplatte verwaltet?

Verkettete Liste, Bitmap

Wie kann die Konsistenz der Dateisystems geprüft werden?

Anlegen einer Freiliste und einer Belegtliste. Jeder Block darf beim Durchlaufen aller Dateien nur in einer Liste auftauchen.

Wie funktioniert das Caching bei Festplatten?

Blöcke (oder ganze Spuren) werden im Speicher gehalten, falls erneut darauf zugegriffen wird, denn Speicherzugriffe sind viel schneller als Plattenzugriffe. Verwerfen von Cache-Einträgen ähnlich wie Paging (NRU-Algorithmus). Das Schreiben hat Priorität (evtl. Write-Through-Caching).

I/O

Welche zwei Arten von Geräten (Devices) gibt es?

Blockorientierte, Zeichenorientierte.

Wie wird ein Gerät angesprochen?

Befehle werden in die Register des Controllers geschrieben. Der Controller blendet sie in den Hauptspeicher ein.

Sobald die Operation beendet ist, signalisiert der Controller das mit einem Hardware-Interrupt.

Was ist DMA?

DMA (direct memory access) erlaubt einem Gerätecontroller das Schreiben von Daten direkt in den Hauptspeicher (ohne Zutun der CPU). Dem Gerät wird Anfang und Größe des Zielbereichs übergeben.

Gibt es bei DMA überhaupt eine Pufferung durch den Controller?

Ja, weil das Gerät die Daten erst komplett empfängt und dann erst in den Hauptspeicher kopiert.

Was ist synchroner/asynchroner Transfer bei Geräten?

Synchron: Gerät wird dediziert blockiert

Asynchron: Befehle werden in Register geschrieben

Was ist ein Gerätetreiber?

Software, die ein bestimmtes Gerät ansteuert. Sie übernimmt das Schreiben von Werten in die Controller-Register. Muss der Treiber auf eine Eingabe warten, dann blockiert er sich selbst und wird durch den Interrupt-Handler des Gerätes wieder aktiviert.

Was ist Spooling?

Mehrere Prozesse müssen auf ein unteilbares Gerät zugreifen. Der Prozess legt eine Datei im Spool-Verzeichnis ab. Ein Daemon schickt diese Dateien nacheinander zum Gerät. (➔Drucker, UUCP)

Wie sind Festplatten aufgebaut?

Zylinder ➔ Spur ➔ Sektor

Welche Plattenarm-Scheduling-Algorithmen gibt es?

FCFS, SSF (shortest seek first), Fahrstuhl

Bei SSF und Fahrstuhl müssen die nachfolgenden Zugriffe bereits bekannt sein. Sonst nur FCFS möglich.

Was ist ein RAID?

RAID (redundant array of inexpensive disks) ist ein Stapel mehrerer Platten, bei denen der Ausfall einer einzelnen Platte durch fehlerkorrigierende Codes (➔Hamming) ausgeglichen werden können.

Was ist eine RAM-Disk?

Simulation eines Block-Gerätes im Hauptspeicher, auf das blockweise zugegriffen werden kann.

Wie funktioniert die Uhr-Hardware?

Ein Quarz-Oszillator dekrementiert einen Zähler. Bei 0 wird ein Tick ausgelöst und der Zähler neu gesetzt. Die Uhr löst Timer-Interrupts aus, die z.B. den Scheduler aktivieren.

Welche Arten von Terminals gibt es?

Serielle, Speicherbasierte (CRT).

Was ist der UART?

UART (universal asynchronous receiver transmitter) wandeln automatisch Bits zu 8-Bit-Zeichen um und puffern die Ein- und Ausgabe.

Deadlocks

Was ist ein Deadlock?

Mehrere Prozesse warten auf ein Ereignis, das nur ein anderer Prozess dieser Menge auslösen kann.

Deadlocks treten beim Zugriff auf unteilbare Ressourcen auf.

Was sind die 4 Deadlock-Bedingungen?

Wechselseitiger Ausschluss, Belegen-und-Warten, Ununterbrechbarkeit, Zyklisches Warten

Wie geht UNIX mit Deadlocks um?

Es unternimmt keine Maßnahmen gegen Deadlocks. Es liegt in der Verantwortung des Programmierers.

Wie kann man Deadlocks erkennen?

Modellierung (Prozesse = Kreise, Betriebsmittel = Quadrate), Zyklenerkennung

Wie kann man Deadlocks beheben?

Entziehbare Betriebsmittel:
entziehen

Exklusive Betriebsmittel:
Backtracking, Abbruch

Wie kann man Deadlocks verhindern?

Sichere Zustände (Bankiersalgorithmus), eine der vier Bedingungen verhindern, Zwei-Phasen-Belegung

Was ist der Bankiersalgorithmus?

Ein Bankier verleiht Geldbeträge an Gläubiger – insgesamt mehr als er zur Verfügung hat. Er muss sich die sicheren Zustände ausrechnen, ab wann er kein Geld mehr verleihen darf, damit er nicht in die Bredouille kommt.

VI Übungsfragen zu verteilten Systeme

Was ist ein Netzwerkbetriebssystem?

Ein Betriebssystem für autonome Rechner, die aber über bestimmte Dienste mit anderen Rechner kommunizieren können. Verschiedene Netzwerkbetriebssysteme können miteinander kooperieren (Solaris + Windows).

Was ist ein verteiltes System?

Ein Betriebssystem auf einer Menge von Rechnern, die sich dem Benutzer wie ein einzelnes System darstellt. Es gibt einen gemeinsamen Speicher und eine Menge gleichartiger Systemaufrufe. Häufig wird auf allen Systemen derselbe Kernel verwendet.

Was sind die Vorteile verteilter Systeme?

Wirtschaftlichkeit (langsame CPUs sind günstiger), Parallelverarbeitung (höhere Rechenleistung), Redundanz (Ausfallsicherheit), Kooperative Arbeit (Trend geht zu Netzanwendungen), Skalierbarkeit

Gibt es Nachteile in verteilten Systemen?

Verwaltung ist schwieriger (➔Semaphore).

Wie kann man Systeme klassifizieren?

SISD, SIMD, MIMD (Multiprozessor, Multicomputer).

Welche Arten von MIMD-Systemen gibt?

Multiprozessorsysteme (bus/switch)

Multicomputersysteme (bus/switch)

Was ist Koheränz?

Wenn mehrere Prozessoren auf einen gemeinsamen Speicher zugreifen, muss ein konsistentes Ergebnis vorliegen.

Was unterscheidet Multiprozessor- und Multicomputersysteme?

- Multiprozessor:
  enge Kopplung, gemeinsamer Speicher
- Multicomputer:
  lose Kopplung, eigener Speicher

Was unterscheidet Bus-Systeme von geswitchten Systemen?

Bus: jeder kann mithören, direkte Verbindung

Switch: mehrere Schaltungen für einen Weg

Was ist NFS?

NFS = Network-File-System. Exportieren/importieren von Dateibäumen aus oder in das eigene Dateisystem. Für den Benutzer transparent.

Was ist Transparenz?

Ein verteiltes System soll sich dem Benutzer wie ein zentrales System darstellen.

Orts-, Migrations-, Replikations-, (Nebenläufigkeitstransparenz)

Was ist der Unterschied zwischen verbindungsorientierten und verbindungslosen Protokollen?

Verbindungsorientiert: logische Verbindung

Verbindungslos: Weitergabe über Relais

Was ist das OSI-Modell?

OSI = Open Systems Interconnection

7 Schichten, WAN-Kommunikation

Was ist das Client/Server-Modell?

Schichten 1, 2 und 5 des OSI-Modells im LAN. Alle Schichten wären unnötiger Overhead.

Was sind blockierende/nicht blockierende Kommunikationsprimitive?

Blockierend: Sender wartet auf Zustellung

Nicht blockierend: Sender arbeitet weiter

Was sind puffernde/nicht puffernde Kommunikationsprimitive?

Puffernd: Mailbox im Zielsystem

Nicht puffernd: Receive und Send gleichzeitig

Was ist RPC?

RPC = Remote-Procedure-Call

Prozeduraufrufe sollen lokal wie entfernt gleich aussehen. Client- und Server-Stubs.

Wie werden Parameter bei RPC übergeben?

Marshalling vom Client-Stub. Eventuell Konvertierung in geeignetes Format des Zielsystems (*endians*).

Was ist dynamisches Binden?

Unter Angabe eines Dienstes und dessen Versionsnummer gibt der Server einen Dienst frei. Der Client fragt im Netz, ob es einen geeigneten Dienst für seine Anfrage gibt.

Was ist eine idempotente Anfrage?

Eine Anfrage, die ohne Nebenwirkungen mehrfach gestellt werden kann (➔make).

Was passiert bei RPC, wenn der Client oder Server plötzlich ausfallen?

Client: Ausrottung, Reinkarnation, Verfallszeitpunkt

Server: Mindestens/Höchstens-einmal-Semantik

Wie kann man eine Gruppe adressieren?

Multicast, Broadcast, Unicast, Prädikatadressierung.

Was ist Multicasting, Broadcasting, Unicasting?

Multicasting: Versand an Gruppe

Broadcasting: Versand an alle

Unicast: Versand an jeden einzeln

Was ist ein atomarer Broadcast?

Das System stellt sicher, dass jedes System in einer Gruppe die Nachricht erhält. Es kommt nicht vor, dass ein einzelnes System eine Nachricht nicht bekommt.

Was ist ein synchrones System?

Ein System ohne Nachrichtenlaufzeiten, in dem alle Nachrichten sequentiell eintreffen.

Was ist ISIS?

Eine Sammlung von Programmen, die die Entwicklung verteilter Anwendungen erleichtern soll. ISIS implementiert vor allem atomare Broadcasts.

Synchronisation in verteilten Systemen

Was ist eine logische Uhr?

Mehrere Uhren laufen synchron mit einer konstanten Abweichung von der realen Zeit.

Wie synchronisieren sich Uhren?

- Lamport (unlogische Sendezeitpunkte von Nachrichten korrigieren die Uhr)
- Cristian (DCF-Empfänger, Messung der Übertragungszeit im Netz)
- Berkeley (Abfrage von Zeit-Servern, Durchschnittsbildung und Broadcast der neuen Zeit)

Wie realisiert man wechselseitigen Ausschluss in verteilten Systemen?

Zentraler Koordinator, Verteilter Algorithmus (Broadcast), Token-Ring

Wie bestimmt man einen Koordinator in einem System?

Wahlalgorithmen: Bully, Ring

Was ist eine (atomare) Transaktion?

Ein Prozess kündigt seine geplanten Aktionen an und fragt, ob alle anderen Prozesse damit einverstanden sind. Die Aktionen werden dann atomar durchgeführt. Verweigert ein anderer Prozess die Durchführung der Transaktion, so kann sie ohne Nebenwirkungen abgebrochen werden, weil der alte Zustand wiederhergestellt wird (privater Arbeitsbereich, Protokollierung der Schreibabsichten, 2-Phasen-Commit)

Serialisierbarkeit, Atomarität, Permanenz

Wie erkennt man Deadlocks in verteilten Systemen?

- Zentral (Koordinator)
- verteilt (Chandy-Misra-Haas)

Prozesse und Prozessoren

Was sind Threads?

Eigene Kontrollflüsse innerhalb eines Prozesses, die sich einen Adressraum mit anderen Threads des Prozesses teilen. Vereinfacht die IPC durch gemeinsamen Speicher. Erlaubt parallele Verarbeitung von Anfragen trotz blockierender Aufrufe.

Wozu benutzt man Threads?

Server-Prozess mit blockierenden Threads (➔File-Server)

Vereinfachter Softwareentwurf

Können Threads ohne Einschränkungen auf dem gemeinsamen Speicher arbeiten?

Zugriffe auf (für alle Threads) globale Variablen müssen über wechselseitigen Ausschluss beschränkt werden.

Hat ein Thread lokale Variablen?

Nur der Stack ist lokal für jeden Thread.

Was für Systemmodelle gibt es?

- Workstation
- Mitbenutzung von Workstations
- Prozessor-Pool

Wie kann man den Prozessen die Prozessoren zuteilen?

- Graphentheoretisch-deterministisch
- Zentral Up-Down
- Hierarchisch
- Verteilt heuristisch

Welche Zugriffsmöglichkeiten gibt es bei verteilten Dateisystemen?

Upload/Download (FTP), entfernter Zugriff (NFS)

VII Übungsfragen zu Datenbanken

(1) Kapitel I: Grundlegende Konzepte

Was ist eine Datenbank?

Ein computer-unterstütztes System, das große Datenmengen persistens speichern und auf Anfragen schnell nach gewünschten Kriterien wiedergeben kann.

Was ist das DBMS?

DBMS = Database Management System

- Verwaltung der Datenbanken (SQL, DDL, DML)
- Abstraktion von Hardware-Details (wie bei Programmiersprachen)
- Schnittstelle für Benutzeranfragen (➔SQL)

Welche Benutzer arbeiten mit dem DBMS?

- DA = Datenadministrator
- DBA = Datenbankadministrator
- Applikationsprogrammierer

Was sind „Entities" und „Relationships"?

Entity: Objekt, über das Daten gespeichert sind

Relationship: Beziehungen zwischen Entities

Warum benutzt man Datenbanken?

Kompakt, schnell, aktuell, sicher, shared.

Was bedeutet für Datenbanken „relational"?

Die einzelnen Datensätze (Zeilen) in einer Tabelle sind mathematisch gesehen Tupel. Eine Tabelle entspricht einer Relation.

Gibt es nicht-relationale Datenbanken?

Objektorientiert, hierarchisch (Baumstruktur).

(2) Architektur für ein Datenbanksystem

Was ist die ANSI/SPARC-Architektur?

Externe/Begriffliche/Interne Ebene.

Was ist DSL?

DSL = Data Sublanguage

Enthält die DDL (data definition language), DML (data manipulation language) und SQL (structures query language).

Was sind Frontends/Backends?

Frontend = Applikationen, die das DBMS nutzen

Backend = DBMS

Es kann mehrere Frontends und Backends geben, die über ein Netzwerk miteinander kommunizieren.

(3) Interne Ebene

Wie greift das System auf Datensätze zu?

1. DBMS fordert Datensatz anhand der RID (*record id*) vom Dateimanager an
2. Dateimanager fordert Datenblock vom Diskmanager an (➔Betriebssystem)
3. Diskmanager liest physikalischen Block von Platte (➔Betriebssystem)

Was ist Clustering?

Zusammengehörige Daten werden zusammenhängend auf der Platte gespeichert, um den Zugriff zu beschleunigen.

Was ist die Record ID (RID)?

Zweidimensionale Adressierung von Datensätzen (vgl. Segmentierung).

RID = Blocknummer + Byte-Offset

Wie kann man den Zugriff auf Datensätze beschleunigen?

Indizierung, B-Trees, Hashing, verkettete Zeiger.

Was ist Indizierung?

Wenn Tabellen öfter nach einem Feld abgefragt werden, kann ein Index (eine zusätzliche spezielle Tabelle) erstellt werden. Indices verweisen jeweils auf die RID in der eigentlichen Tabelle.

Vorteil: schnellerer Zugriff als naive sequentielle Suche

Was sind B-trees?

Sequenzmenge, Indexmenge, ausbalancierter Baum

Was ist Hashing?

Indizierung anhand einer Berechnungsvorschrift (Funktion).

Wie kann man Daten komprimieren?

Differentiell, Hierarchisch, Huffman.

(4) Kapitel II: Relationale Systeme und DB2

Was ist DB2?

Datenbanksystem von IBM, das SQL unterstützt (und von ANSI als Standard deklariert wurde).

Was ist SQL?

SQL = Structured Query Language

SQL kann definieren (DDL), manipulieren (DML) und abfragen

SQL kann interaktiv oder embedded benutzt werden.

Warum nennt man SQL „nicht-prozedural"?

Man spezifiziert, was man vom System verlangt, aber nicht im Detail, wie man es erhält.

(5) Datendefinition (DDL)

Welche SQL-Befehle gibt es in der DDL/DML?

DDL: CREATE TABLE, ALTER TABLE, DROP TABLE

DML: SELECT, UPDATE, DELETE FROM, INSERT INTO

(6) Datenmanipulation (DML)

Wie sieht der SELECT-Befehl aus?

SELECT felder FROM tabelle WHERE …;

(7) System-Katalog

Was ist der System-Katalog?

Tabelle aller Objekte einer Datenbank. Bei DB2 etwa 30 Objekte.

Tabellen/Views (SYSTABLES)

Spalten (SYSCOLUMNS)

Indizes (SYSINDEXES)

(8) Views

Was ist ein View?

Eine teilweise Ansicht einer Tabelle. Views lassen sich per SQL ansprechen.

Welchen Vorteil haben Views?

Verschiedene Benutzer sehen nur bestimmte Aspekte desselben Datenbestands.

Wie werden Views erzeugt?

CREATE VIEW AS SELECT…

Welche Probleme können mit Views auftreten?

Eine INSERT-Operation könnte schiefgehen, wenn ein Feld der zugrundeliegenden Tabelle als „NOT NULL" angegeben ist, aber das Feld im View gar nicht vorkommt.

(9) Kapitel III: Das relationale Modell

Was bedeuten die Begriffe Relation, Tupel, Attribut, Kardinalität und Grad in Bezug auf eine Datenbank?

Relation = Tabelle

Tupel = Zeile / Datensatz

Attribut = Feld

Kardinalität = Anzahl Zeilen / Datensätze

Grad = Anzahl Spalten / Felder

Warum verwendet man Domains?

Sie vermeiden den Vergleich von eigentlich nicht verwandten Werten. Domains entsprechen Datentypen.

Welche Eigenschaften hat eine Relation?

Tupel unsortiert, keine doppelten Tupel (unique).

Sind Tabellen und Relationen äquivalent?

Nicht direkt. Tabellen können nur endlich sein. Tabellen können nur atomare Werte enthalten (um eine Relation in eine Tabelle umzuformen, muss man die Relation normalisieren).

Was bedeutet normalisieren?

Höhergradige Relationen (Grad > 1), die statt atomaren Werten wiederum Relationen enthalten, werden zu Relationen mit Grad 1 umgeformt, indem jede mögliche Kombination eine Zeile der Tabelle bildet.

Welche Arten von Relationen gibt es in einer Datenbank?

Tabellen, Views, Snapshots, Abfrage-Ergebnisse

Was sind Snapshots?

Wie Views, bilden aber eine eigene Tabelle, die in vorgegebenen Intervallen erneuert wird.

Was ist eine relationale Datenbank?

Eine Datenbank, die sich dem Benutzer als eine Ansammlung von zeitlich-veränderlichen normalisierten Relationen darstellt.

(10) Integritätsregeln

Warum gibt es Integritätsregeln?

Sicherstellen von sinnvollen Werten als Feldinhalt eines Datensatzes. (➔Gewichte sind nie negativ)

Was ist ein Primärschlüssel?

Eindeutige Identifikation eines Datensatzes in einer Tabelle.

Hat jede Relation einen Primärschlüssel?

Ja. Wenn nicht angegeben, wird er aus allen Feldinhalten zusammengesetzt.

Ist ein Primärschlüssel ein Index?

Nein, der Index beschleunigt den Zugriff. Der Primärschlüssel dient der eindeutigen Identifikation eines Datensatzes.

Was besagt die Entity-Integritätsregel?

Der Primärschlüssel eines Datensatzes darf nicht leer sein.

Was sind Fremdschlüssel?

Zeiger auf Primärschlüssel einer anderen Datenbank. Sinnvoll, wenn zwei Datenbanken inhaltlich zusammenhängen.

Was besagt die Referenz-Integritätsregel?

Jeder Fremdschlüssel eines Datensatzes muss auf einen existierenden Primärschlüssel der korrespondierenden Datenbank zeigen.

(11) Relationale Algebra

Was ist eine relationale Algebra?

Sammlung von Operationen, die aus einer oder zwei Relationen eine neue Relation erzeugen.

Teilmenge, Schnittmenge, Vereinigung, Differenzmenge, Projektion…

VIII Übungsfragen zu Programmiersprachen

(1) Einleitung

Was ist eine Programmiersprache?

Notationelles System zur Beschreibung von Berechnungen. Menschen-/Maschinenlesbar.

Was ist ein Algorithmus?

Präzise, endliche Beschreibung eines allgemeinen Verfahrens mittels elementarer Verarbeitungsschritte.

Wie kann man Programmiersprachen abstrahieren?

- Datenabstraktion (fundamental, strukturiert, modul)
- Kontrollabstraktion (fundamental, strukturiert, modul)

Was unterscheidet imperative, funktionale und logische Programmierung?

- Imperativ: sequentiell mit Sprunganweisungen
- Funktional: geschachtelte Funktionen, Schleifen nur durch Rekursion
- Logisch: Verarbeitung von Eigenschaften

Was ist Syntax und Semantik und wie beschreibt man sie?

Syntax: Beschreibung der Sprachstruktur (wie man Teile der Sprache kombinieren kann) (durch Grammatik)

Semantik: Bedeutung der Anweisungen (durch Referenzhandbuch, formale Definition oder Referenzimplementation eines Übersetzers)

Was unterscheidet Interpreter und Compiler?

Interpreter: direkte Ausführung des Quellcodes

Compiler: Übersetzung in Maschinensprache

Wie funktioniert ein Compiler?

Scanner: Tokenisieren (Analyse auf korrekte Lexik)

Parser: Analyse auf korrekte Syntax

Semantische Analyse: Erzeugung des Programms

(2) Syntax

Wie arbeitet die lexikalische Analyse?

Der Scanner fasst zusammenhängende Zeichenkette als Token zusammen.

Wie beschreibt man die Syntax von Programmiersprachen?

Programmiersprache = formale Typ-2-Sprache

BNF, EBNF, Syntaxdiagramm

<NT> ::= T | T <NT>

Was ist ein Ableitungsbaum bzw. Syntaxbaum?

Ableitungsbaum: graphische Darstellung eines Ableitungsvorgangs

Syntaxbaum: Ableitungsbaum ohne Nonterminale

Was ist eine mehrdeutige Grammatik?

Für eine Zeichenkette sind mehrere Ableitungsbäume möglich.

Was ist links-/rechts-rekursiv?

Linksrekursiv: es wird nach links assoziiert (die linke Seite wird länger)

Wie arbeitet der Parser?

- Bottom-Up-Parser
- Top-Down-Parser
- Prädikative Parser (rekursiv-absteigendes Parsen)

(3) Semantik

Was ist eine Bindung?

Assoziation eines Namens (➔Variable) mit Attributen (➔Inhalt der Speicherstelle).

Statische/dynamische Attribute.

Was ist die Symboltabelle?

Zuweisung von Namen auf Attribute
(Menge von Bindungen)

Was ist die Umgebung?

Zuweisung von Namen auf Speicherplätze

Was ist der Gültigkeitsbereich einer Deklaration?

Bereich des Programms, in dem die Bindungen dieser Deklaration erhalten bleiben (lokale Deklarationen können globale Deklarationen überlagern)

Statischer Gültigkeitsbereich (Compile-Zeit)

Dynamischer Gültigkeitsbereich (Laufzeit)

Was ist eine Aktivierung?

Aufruf einer Prozedur. Es wird ein Aktivierungssegment (Speicher für lokale Variablen) angelegt. Am Ende des Prozedur ist die Lebendauer der lokalen Objekte erreicht.

Was bedeutet Dereferenzieren?

Erhalten des Inhalts einer Speicherstelle anhand dessen Adresse.

Welche Arten der Speicherzuteilung gibt es?

Statische (globale Variablen / Speicher), automatische (lokale Variablen / Stacl), dynamische (manuelles malloc / Heap)

Was ist eine Variable/Konstante?

Variable: Ein Objekt, dessen assoziierter Wert sich ändern kann.

Konstante: Ein Objekt, dessen assoziierter Wert sich nicht ändern kann.

Was sind l-value und r-value?

l-value: Speicherplatz

r-value: Wert

Was sind Aliasnamen?

Zwei Zeiger zeigen fälschlicherweise auf dasselbe Objekt.

Was sind hängende Referenzen?

Zugriff auf ein Objekt nach seiner Lebensdauer (➔nach Rücksprung aus einer Unterroutine)

Was ist Garbage?

Garbage entsteht, wenn eine Umgebung seinen Speicher nicht explizit freigibt. Durch eine automatische Garbage Collection wird der Speicher bereinigt.
(➔vollständig dynamische Laufzeitumgebungen)

Wozu dienen Stack und Heap?

Stack: Rücksprungadressen, lokale Variablen

Heap: dynamische Speicherzuteilung (malloc)

(4) Datentypen

Was ist ein Datentyp?

Menge von Werten mit darauf definierten Operationen.

Was ist eine Deklaration?

Zuweisung eines Datentyps zu einer Variablen.

Was ist eine streng typisierte Sprache?

Alle Objekte der Sprache sind wohldefiniert. Alle Typfehler können zur Compile-Zeit gefunden werden.
(➔MODULA-2)

Schwach typisiert: LISP, BASIC, Perl

Was für Typen gibt es in Programmiersprachen?

Einfache (vordefinierte) Typen (Aufzählungstypen, Ordinaltypen, Unterbereichstypen) und durch Typkonstruktoren benutzerdefinierte Typen.

Welche Typkonstruktoren gibt es?

- Kartesisches Produkt (Record)
- Vereinigung (Union)
- Teilmenge (Slices)
- Potenzmenge (Set)
- Funktionen (vorbelegte Arrays)

Wann sind zwei Typen äquivalent?

- Strukturäquivalenz
  (gleiche Datentypen an jeder Stelle)
- Namensäquivalenz
  (gleiche Benennung des Typs)
- Deklarationsäquivalenz
  (zurückführen auf dieselbe Definition; Neudefinition nicht äquivalent)

Was ist die Typüberprüfung?

Prüfung auf korrekte Verwendung der Typen.

Statisch (Compile-Zeit) oder dynamisch (Laufzeit).

Was ist Typinferenz?

Ableitung von Typen aus Verknüpfung von Typen.
(➔int + int = int)

Wann sind Typen kompatibel?

Wenn man sie miteinander verknüpfen kann.

Was sind implizite Typen?

Compiler entscheidet über Typ aus dem Kontext

Was ist eine Typumwandlung (Cast)?

Explizit: Umwandlung in einen anderen Typ (➔(int) realzahl)

Implizit: Automatisches Anpassen des Typs, wenn Variablen verschiedener Typen kombiniert werden (int + real = real).

(5) Steuerung des Kontrollflusses

Was ist ein Kontrollfluss?

Sequentielles Ausführen hintereinander stehender Anweisungen, sofern nicht durch Sprunganweisungen durchbrochen.

Welche Anweisungen verändern den Kontrollfluss?

Bedingte (if/case), Schleifen (while/repeat), Sprung (goto), Prozeduren

Wie kann man Parameter übergeben?

Call-By-Value, Call-By-Reference, Call-By-Name

Was ist Rekursion?

Eine rekursive Prozedur ruft sich bis zum Erreichen eines Stop-Kriterium selbst auf, um einen Wert zu berechnen. (➔Fakultät)

Was ist eine statische Laufzeitumgebung?

Alle globalen und lokalen Variablen haben eine statische Umgebung. Unterprogramme dürfen sich nicht gegenseitig oder selbst aufrufen (keine Rekursion).

Was ist eine stack-basierte Laufzeitumgebung?

Wird ein Block betreten, dann wird auf dem Stack eine neue Umgebung für diesen Block angelegt.

Umgebungszeiger: →aktuelle Umgebung

Kontrollverbindung: →übergeordnete Umgebung

Zugriffsverbindung: →globale Umgebung

Was sind vollständig dynamische Umgebungen?

Objekte werden automatisch erzeugt und vernichtet. Garbage-Collection.

(6) Abstrakte Datentypen

Was ist ein abstrakter Datentyp (ADT)?

Ein benutzerdefinierter Datentyp, der eine Kapselung und semantische Spezifikation wie ein Basistyp hat. Vergleich mit OO-Sprachen, bei denen man nur Methoden auf Objekte anwenden kann.

Vorteile: Modifizierbarkeit, Wiederverwendbarkeit, Sicherheit

Wie spezifiziert man abstrakte Datentypen?

Syntax: Abbildung zwischen Mengen

Semantik: Axiome

Was sind parametrisierte abstrakte Datentypen?

Datentyp arbeitet mit Elementen, die nicht vorher spezifiert wurden (Queue kann beliebige Elemente aufnehmen).

Was ist Überladen bzw. Polymorphismus?

Überladen:

Operation verhält sich je nach benutztem Datentyp anders.

Polymorphismus:

Operation ist mit beliebigen Datentypen möglich.