

L 05: Memory

Fragen

- Bedeutung: Shared All / Shared Nothing ?
- Was ist: N+M Topologien Cluster ?
- Problem der unterschiedlichen Disk-IDs im Cluster
- Was ist das PxFs?
- Verfügbarkeit globaler IP Adressen erhöhen?
- Welche Methoden hat eine Cluster Resource Group?
- Was ist SMF?
- Was macht man bei mehr als 200 km Abstand?

Memory Ebenen

Welche gibt es?

Memory Ebenen

CPU

- Register
- Cache (ggf: L2 Cache, L3 Cache)
- Spezial-Cache (TLB, Write-Cache, Prefetch-Cache)

Hauptspeicher (primär)

Platten (sekundär)

- ggf. Cache im Platten Controller

Band (tertiär)

- teilweise auch optische Medien

Memory-Sicht User Prozess

- User Prozess sieht (genau) einen Adressraum
- Elemente des Adressraums
 - Programmcode (*text*)
 - Vorinitialisierte Daten (*data*)
 - Uninitialisierte Daten (*bss*)
 - Stack
 - Segmente
 - Memory mapped Dateien
 - Gesharte Bibliotheken (*shared libraries*)
 - Gesharte Datensegmente (*shared segments*)

Adressierung im User

Verwendete Adressierungen

– Realer Speicher

- Adresse im Programm \leftrightarrow Adresse im Hauptspeicher
- Ggf Relokation (für Multi Programming)
 - statisch (Modifikation beim Laden)
 - dynamisch (*base* und *limit* Register)

– Virtueller Speicher

- Adressraum in Seiten aufgeteilt
 - Adresse := (Seitennummer, Adresse in Seite)
 - Umsetztabelle: Seite in Adressraum \rightarrow Seite im Hauptspeicher

– Objektorientierter Speicher

- wie bei Java, CPU/Interpreter organisiert Zugriff

Swapping

vollständige Auslagerung eines Segments /
Adressraums

- notwendig bei realem Speicher
- dauert lange
- Platzverwaltung bei dynamischer Relokation

Paging

Auslagerung einzelner Seiten nach Bedarf

- Steuerung durch Programm explizit (*overlay*)
- Automatisch nach Bedarf (*demand paging*)

Strukturen

- Prozess → Segment
- Prozess → Adressraum → Segment-Liste
- Segment → Liste der Seiten

(Leider häufig synonym als *swapping* bezeichnet)

File Input/Output

- Traditionelle I/O
 - open, read, write, close
 - Buffercache-Verwaltung (separat)
 - Buffercache vs. Seiten im Paging
 - Heute: Realisierung via Memory Mapping im OS
- Memory Mapping
 - Datei wird in eigenes Segment gelegt
 - ggf. teilweise, wenn Platz zu klein
 - Verwaltung einfacher, da Paging genutzt wird

Laden eines Programms

Traditionell:

- Programm ist im Lade-Format
- Anpassung Adressen beim Laden
- Auflösung offener Referenzen
 - weitere Programmteile (z.B.: .o-Dateien)
 - Bibliotheken
 - Anordnung und Relokation

Aktuell:

- *demand paged executable*

Demand Paged Executable

Dateiformat (zb. *COFF*, *elf*, *dwarf*, ...)

- Metadaten
 - Länge
 - Name, Typ der Objekte (Funktionen, Variablen)
 - Ggf Debug-Info
- Relokatibles Programm
 - direkt nutzbar im Hauptspeicher
- weitere Programmteile: dynamisches Nachladen
- Bibliotheken: gleiches Format

Start eines Programmes in Unix (Linux)

`fork() + exec()`

`fork()`: Duplizieren Prozess

- Statt Kopie aller Seiten:
nur Pagetable wird dupliziert und auf r/o gesetzt
- Erstes Schreiben in r/w Segment:
Seite wird kopiert (COW = copy on write)
- Dadurch: schnelles `fork()`

`exec()`: Ersetzen Programm im Adressraum

Shared Segmente

- Segmente sind in mehreren Adressräumen eingetragen
 - Rechte analog zu Dateien
 - rwx für user/group/others
- Nutzung
 - Mehrfach benutzte Programme
 - Mehrfach benutzte Bibliotheken
 - Gemeinsame Datensegmente

Paging Ablauf

User Adressraum enthält verschieden Segmente

- Zugriff (Lesen, Schreiben, Ausführen)
- CPU teilt die Adresse in
 - Virtuelle Seitenadresse (aka: Nummer der Seite)
 - Adresse in der Seite
- CPU prüft (ggf. mit MMU)
 - Ist der Seite eine Seite im Hauptspeicher zugeordnet?
 - Wenn nein: Paging-Interrupt
 - OS muss eine freie Seite besorgen und eintragen
 - Daten einlesen (wenn vorhanden)
- Virtuelle Seitenadresse umsetzen und benutzen

Seitentabelle

CPU Seitentabelle (*pagetable*)

- Zugriff erfordert Umsetzung virtuell → real
- Adressraum → Segment-Tabelle → Seiten-Tabelle
 - Größe wird durch Objekt-Größe bestimmt!
- Hierarchische Seiten-Tabellen
 - Mehrere Zugriffe auf Tabelle (Speicherzugriffe?)
- Invertierte Seitentabelle
 - 1 Eintrag pro Hauptspeicher Seite
 - z.B. als Hash-Tabelle oder B-Baum
 - Unterstützung durch TLB / TSB in der CPU / MMU

Page Replacement Algorithm (1)

OS braucht immer freie Seiten für Paging

- Bestimmter Teil immer frei gehalten

HW Unterstützung in CPU / MMU notwendig

- read-Bit Cache-Zeile wurde gelesen
- dirty-Bit Cache-Zeile wurde beschrieben
- Zugang zu diesen Bits per Software
- Weitere Bits für Algorithmen (ggf in Tabellen)
- Tabellen für Adressräume, Segmente, Seiten

Page Replacement Algorithm (2)

Verschiedene Algorithmen

- Optimal (leider meist nicht realisierbar)
- NRU Not Recently Used
- FIFO
- Second Chance
- Clock Replacement
- LRU
- ARC (innovative)

Adaptive Replacement Cache

Passt sich an reale Wahrscheinlichkeiten an

- Seiten immer wieder benutzt
- Seiten nur einmal benutzt
- Datenstrukturen
 - L1 Cache: T1: recent B1: recent ghosts
 - L2 Cache: T2: frequent B2: frequent ghosts
 - T1 + T2 bilden Cache B1 und B2: nur Adressen
- Implementierung
 - Neue Einträge → Seite geht nach T1
 - Hit in T1 → Seite geht nach T2
 - Hit in B1: T1 größer, T2 kleiner
 - Hit in B2: T2 größer, T1 kleiner

Schreiben der Daten

Pagetable und TLB: modified (dirty) Bit

- Schreiben auf die Seite ändert es auf 1

Implementierungen Schreiben:

- Scanner setzt dirty Seiten auf die Liste zum Schreiben
- Extra page daemon füllt die Liste (ohne Page Replacement Funktion: schneller)
- COW wird benutzt

Cache

Schnellzugriff auf Hauptspeicher

- Wenige Zyklen (vs. ~ 100 für Hauptspeicher)
- Ggf in Leveln organisiert
- Ggf Instruction/Data Trennung
- Organisation in Cache-Zeilen (16 – 256 Byte)
- Tags: Kontext, Adresse, read, dirty (written),
- Verschiedene Algorithmen
 - Voll assoziativ
 - Direct mapped
 - Set assoziativ

Cache Kohärenz

- snarfing (*to grab*, see Jargon)
 - alle Memory Controller beobachten alle CPU-Memory Interfaces...
- snooping
 - Spezial-Bus für Cache writes
- directory based
 - Zentrales Verzeichnis für Cache-Zeilen
- snooping & directory based (innovative)
 - In Untereinheiten (System boards): cache snooping
 - Zwischen Untereinheiten: directory based
 - verteilte Directories

Spezial Caches

- Translation Lookaside Buffer
 - Umsetzung virtueller Speicher-Adressen
 - Voll assoziativ und in CPU: relativ klein (... 512)
- Translation Lookaside Software Buffer
 - TLB Ergänzung in Software
- Write Cache (innovative)
 - Queue zum Schreiben aufs Memory
- Prefetch Cache (innovative)

Transactional Memory (innovative)

Synchronisation

- Instruktionen *test-and-set*, *compare-and-swap*
- Unterstützung durch HW notwendig
- Größere Synchronisation:
 - spinning locks, semaphore, message queues, ...

Transactional Memory

- Transaktion umfasst mehrere Befehle
- writes werden zurückgehalten (write cache)
- *commit* Kommando schreibt oder gibt Fehler