01

Def Betriebssystem:

- Programme + Eigenschaften einer Rechenanlage, insbesondere Abwicklung und Steuerung von Programmen.
- "Like a government", provide an environment within other programs can do useful work.
- Bindeglied Software - Hardware

Speicherhierarchie

Register - Cache - Hauptspeicher - SSD - Magnetische Disk - Optische Disk - Magnetbänder

Architektur

Einprozessorsysteme

- CPU <- Daten, Interrupt, IO -> Gerät
- CPU <- Instruktionen, Daten -> Speicher
- Gerät <- DMA -> Speicher

Mehrprozessorsysteme

Symmetrisch: Jeder Prozessor eigene Kopie von OS Assymetrisch: Zuoordnung von Tasks auf Prozessoren (Scheduling)

Clustersysteme

Sammlung von Rechnern, gemeinsamer Speicher + Verbindung, gegenseitige Überwachung

Sonstiges

Verteilte Systeme: Verteilung auf heterogene Rechnersysteme, lose Kupplung, Client/Server Cloud Computing: IaaS, public/private cloud, on-demand

Betriebssystemtypen

Stapelverarbeitung: Befehle auf Band, keine Interaktion Serversysteme: Service-sharing (Drucker, ...), Multi-user Multiprozessor: Anpassungen von Serversysteme für Multiprozessorsetup PC-Betriebssysteme: Häufig nur ein Nutzer, einfache Oberfläche, viele IO / HID Geräte Handheld Systeme: Teils

Realtime, ereignisgesteuert Embedded Systeme: Klein, effizient Sensorsysteme: Energieeffizient, ereignisgesteurt Echtzeitsysteme: Messgeräte, Roboter, Zeitschranken Smart-Card Systeme: ...

02

Systemkonponenten

Prozessverwaltung, Hauptspeicherverwaltung, IO Systeme, Kommunikationssysteme, Sekundärspeicherverwaltung, Schutz und Sicherheit, Resourcenverwaltung allgemein(CPU, Memory, Disk)

Dual-Mode

User-mode für Anwendungen, System-mode für privilegierte (IO disk, memory, ...) Instruktionen. Wechsel User -> System via trap oder interrupt Wechsel System -> User durch System (reset Modus-Bit)

Systembibliotheken

Bilden Interface zwischen Prozessen und OS, stellen Funktionen bereit die zB System-Calls enthalten.

Systemprogramme

Umgebung für Entwicklung und Ausführung, Arbeit, ... (ls, cp, gcc, ...)

Betriebssystemarchitektur

Monolithisch (Linux, Solaris, Windows): Grupperiung aber keine Modularisierung, jeder kann jeden aufrufen Geschichtet (OS X): A la TCP Mikrokerne: Auslagerung Teils des OS in Server (evtl User-Space) -> Kleiner Kernel Ereignisgesteuert: Klar VM: Klar

03

Prozess

Code, Stack, Heap Neu / Rechnend / Blockiert / Bereit / Beendet

Prozessleitblock

Zeiger / Zustand / Nummer / BfZähler / Register / ...

Verkettete PLBs bilden Warteschlange (doubly-linked-list)

Thread

Leichter Prozess, eigener Stack, teilt a) Code b) globale Daten c) OS Resourcen Task == Ansammlung von Threads. (Ein Prozess = Task + 1 Thread)

Zustände wie bei Prozessor.

Thread kann arbeiten während andere Threads der Task blockiert. Effiziente Kontextwechsel (100x schneller als bei Prozess) Aber: Scheduling auf Prozess-Stufe, d.h. Threads unsichtbar gegenüber OS - kein Schutz vor Starvation.

Bsp Browser mit Fenstern / Download Threads etc. Editor mit 1 Thread Formatierung, 1 Eingabe, ...

User Threads

Thread library, gleicher Adressraum, transparent gegen OS, müssen CPU freiwillig aufgeben

Kernel Threads

Von OS gemanaged, scheduling innerhalb Adressraum (z.B. bei blockierendem Thread)

Hybride Threads

Multiplexing von user threads auf kernel threads. Normalerweise N:M

Thread Pools

Spawnen von worker-Threads bei Start zwecks Vermeidung Overhead

Scheduling

Job Queue: Auf Massenspeicher abgelegte Prozesse (waiting to load) Ready Queue: Prozesse bereit im Hauptspeicher IO Queue: Prozesse die auf IO warten Ausgelagerte Prozesse: Swapped

Scheduler: Wählt Prozess.

Dispatcher: Gibt Kontrolle an Prozess. (Speichert / lädt basierend auf PLB). Overhead 1-1000 micro-s

04

Prozessausführung: CPU Burts + IO Bursts

Scheduler

Wählt aus Ready-Queue jenen, der laufen soll.

Entscheid bei Wechsel: - Running -> Blocked (non-preemptive) - Running -> Ready (preemptive) - Blocked -> Ready (preemptive) - Process terminates (non-preemptive)

Kriterien: Fairness, CPU Last, Durchsatz, Wartezeit (Ready-Queue), Verweilzeit (Lebenszeit), Realzeitverhalten, ...

Schätzung CPU Burst

z.B. exponentieller Mittelwert: `t_n+1 = a * t+n + (1 - a) * t+n`

FCFS

Nicht präemptiv. Trivial

Shortest Job First

Nicht präemptiv: Keine Verdrängung Präemptiv: Verdrängung

Priorität

Priorität basierend auf Speicherbedarf, EA, Wichtigkeit, ... Problem: Aushungern. Lösung: Aging, Priorität steigt mit Wartezeit Präemptiv oder nicht.

Round Robin

Gut für Time-Sharing Jeder Prozess erhält Zeitquantum. Nach Ablauf, Einreihung in Ready-Queue. Präemptiv

Multilevel

Mehrere Queues, jede Queue eigenes Scheduling, Aufteilung zwischen Queues mit Zeitscheiben

Multilevel Feedback

Mehrere Queues mit verschiedenen Zeitquanti. Präemptiv

Lotterie

Verlosung von Zeitquanti, Lose können zB von Client and Server gegeben werden

Garantiertes Scheduling

vorgesehene zeit = (aktuelle zeit - Erzeugungszeitpunkt) / Anzahl Prozesse
verbrauchte Zeit / vorgesehene Zeit = x Wähle Prozess mit kleinstem X
Präemptiv

Echtzeit

Planbar falls: Summe `(cpu_zeit_i / dauer_i) kleiner gleich 1`

Offline Scheduling

Scheduling vor Start. Voraussetzung: Periodische Aktivitäten

Earliest Deadline First

Prozess mit engster Frist selektiert Präemptiv / Nicht präemptiv

05

Prozessinteraktion

- Speicherbasiert (shared memory) -> synchronisation notwendig
- Nachrichtenbasiert (sync / async, ...): Via Mailbox oder direkt
  - Synchron: 0 Kapazität: Erfordert direkte Synchronisation
  - Async endlich: Sender wartet falls voll, Empfänger falls leer
  - Async unendlich: Keine Wartezeiten

Kritischer Abschnitt

Folge von Code-Anweisungen mit Zugriff auf gemeinsame Daten.
Anforderung: Wechselseitiger Ausschluss, Fortschritt, Begrenztes Warten

Semaphore

Zählende: Initialisiert auf Anzahl Resourcen. wait() zählt runter, signal() hoch.

Monitor

Sammlung von Prozeduren, Variablen, Datenstrukturen. Innerhalb Monitor

zu jedem Zeitpunkt nur ein aktiver Prozess.

06

Verklemmungen

A belegt Mittel das von B benötigt wird, B belegt Mittel das von A benötigt wird

Treten auf, wenn alle auftreten: - Wechselseitiger Ausschluss (Resource nur von einem Prozess nutzbar) - Halten und Warten: Prozess der Resource hält wartet auf andere - Keine Verdrängung: Resource nur frewillig freigebbar - Zirkluierendes Warten: Pi wartet auf Pi+1 mod n

Verhindern

Verhindern dass eine der vier Bedingungen zutrifft

Wechselseitiger Ausschluss: Nicht nötig für teilbare Resourcen Halten und Warten: Belegen aller Resourcen vor Ausführung / Abgabe aller Mittel bevor neue belegt Keine Verdrängung: Entzung von zugeweisenen Resourcen Zirkulierndes Warten: Anordnung

Vermeiden

Für jeden Zugriff entscheiden ob dadurch Verklemmung auftreten könnte

- Jeder Prozess beschreibt maximal verwendete Resourcen
- Anfrage erfüllt wenn in sicheren Zustand bleibt (keine Verklemmunge möglich)

Bankers

Geeignet für Resourcen mit mehrfachen Instanzen - Available[j] = k von j verfügbar - Max[i, j] = Pi nutzt maximal k von j - Allocation[i, j] = Pi nutzt aktuell k von j - Need[i, j] = Max[i, j] - Allocation[i, j] = Pi nutzt maximal k von j zustätzlich

Aufheben

Erlauben dass auftritt, dann Massnahme ergreiffen

07

Dynamisches Laden

- Aufgerufene Routine checkt ob aufgerufene Routine geladen. Wenn nicht: Loader lädt nach.
- Nützlich falls grosse Codesegmente selten benötigt
- User-Space Implementation möglich

Dynamisches Binden

- Stub ist Stv für aufgerufene Routine
- Prüft ob Routine geladen, lädt wenn nicht
- Ersetzt sich selbst mit Routine
- Bibliotheksupdates ohne Compilierung/Linkung
- Dynamic Link Library / SHared Library
- Teilen von Code-Segmenten
- Erfodert OS, da Wissen über geladene Routinen dort

Logische und Physikalische Adressen

- Physikalisch: 0 - n auf RAM
- Logisch: 0 - m in Prozessbereich.
- Logisch + Offset = Physikalisch
- Limit-Register: Logische Adresse kleiner Limit (0-basiert)

Hauptspeicherverwaltung

- Linked lists, zB Bitmaps (frei / belegt)
- Allocation: Schnell, wenig verschnitt, ...
- First Fit, Next fit (Fortsezten bei Ende letzter Suche), Best fit, Worst fit, Quick fit (Liste mit üblichen Löcheern)
- Buddy System: Blöcke sind Potenzen von 2. Kein externern, sondern nur interner, Verschnitt. Sehr schnell.

Paging

Nicht-zusammenhängender physikalischer Adressraum.

- Aufteilung physikalischer Speicher in Kacheln (Frames, 2*n Bytes)
- Logischer Speicher in Pages (Gleiche Grösse)
- Paging-Table mapped Pages auf Frames
- Internet Verschnitt
- Paging Table per Prozess
- Adresse: Page number p, Page offset 0, Frame nr k

Multi-level Paging

Problem: Seitentabelle wird gross. (2**32 Bytes, 4 KB Page, 2**20 Pages, 4 Bytes per table entry, 4MB Page table)

Lösung: Multi-level Paging. Logische Adresse = (p1, p2, 0), Lookup in

mehreren Nested Tables

Two-Level paging

Klar

Page table with hashes

Inverted page table

Nur Einträge für reale (belegte) Frames, Aber: Aufwändigere Suche

Segmentierung

- Compiler erstellt Segmente für lokale Variabeln, Stack, Prozedur A, Prozedur B, ...
- Jedes Segment mit eigenem Offset (+ Limit)
- Kombinierbar mit Paging

08

Virtueller Speicher

- Benötigte Teile eines Programms in Hauptspeicher, rest in Sekundärspeicher
- Grosser Speicherbereich
- Memory sharing
- Demand-Paging

Demand Paging

- Pager lagert nur benötigte Pages ein (Weniger IO, weniger Speicherbedarf)
- Memory Access: Laden der Page von Disk nach Memory if nicht geladen
- Zugriff auf Page in Memory
- Copy-on-write für bessere Performance

Paging Algorithmen

- FIFO: Trivial: Schlechte Performance
- LRU: Least recently used: Aufwändig
- Referenzbit (1 Bit: Second chance)
- Clock: 1 Ptr Auslagern, 1 Ptr zurücksetzen
- Second chance+: 2 Bit, 1 bit second chance, 1 bit unverändert / verändert

Paging

- Page buffering: Freihalten eines Pools an freien Pages: Schnellere Einlagerung
- Schreiben Pages auf Speicher wenn Idle -> Eventuell schnellere Auslagerung später
- Thrashing: Mehr mit Paging beschäftigt als mit Arbeit: Prozess mehr Kacheln zuordnen
- Überwachung Page Faults: Anpassung Speicherzuordnung des Prozesses

09

Disk-Anbindung

- Host-Anbindung: IDE, SATA, SCSI, ...
- Netz-Anbindung: LAN, RCPs, iSCSI
- SAN: Dediziertes Netz, spezielle Disk-Access Protokolle (Fibre-Channel, Infiniband, ...)

Disk Formatierung

- Low-level / physikalisch: Unterteilen einer Disk in Sektoren. (Header - Data - Trailer)
- Logisch: Partitionen, Filesystems

Partitionierung

- MBR: In Sektor 0
- Boot Block: Program das OS lädt
- Superblock: Datenträgeraufbau, Blockgrösse, ... Meta Informationen
- Freispeicherliste / Liste fehlender Blöcke

Fehlerhafte Blöcke

- Sector Sparing: Liste schlechter Blöcke, transparente Umleitung auf Reserveblock -> Verschlechtertes Disk-SCheduling
- Sector Slipping: Verschieben von Sektoren um eine Spur

RAID

- Schützt vor Hardwarefehlern, aber nicht Software (z.B fehlerhaftem Disk-Driver)
- Dort muss zB das FS (siehe ZFS) Abhilfe schaffen (Prüfsummen + Korrektur falls zB gespiegelt)

RAID 0

- Striping / Interleaving: Jede Disk hat Streifen der virtuellen Disk
- Block-level Striping: 1 Datei auf N Disks

RAID 1

- Mirroring

RAID 2

- Bit interleaving (7-bit hamming code)

RAID 3

- RAID 2 mit nur Parität (even or odd)

RAID 4

- Paritätsblöcke

RAID 5

- Verteilung & Parität

RAID 6

- Wie 5, aber mit mehr Redundanz

Dateisysteme

- Lesen & Schreiben
- Attribute per Datei (Namen, Ort, ...)
- Multi-user

Trivia

- mmap: Einbindung einer Datei als virtueller Speicher: Schreiben nach schliessen der Datei

10

Dateisysteme

Logisches Dateisystem

- Datei und Verzeichnisoperationen
- Verwalten von Dateien / Strukturen
- Schutzmechanismen

Organisationsmodul

- Übersetzung logische in physikalische Adresse
- Freispeicherverwaltung
- Festplattenmgmt

Basisdateisystem

- Kommandübergabe an I/O (Lese Disk 1, Zylinder 2, Spur 3, Sektor 4)
- Scheduling
- Caching

I/O Steuerung

- Interface zu Gerätetreiber, Interrupts

inode

- mode, link count, owner uid, gid, ...
- direct blocks (10) -> pointing to data blocks
- single indirect -> points to list of direct blocks
- double indirect -> points to list of single indirects
- triple indirect -> ....

File table

- Systemweite Tabelle mit offenen Files
- Tracking welche Prozesse welche Files
- Datei öffnen: FCB wird in file table kopiert, ausgabe Deskriptor für Zugriff

Verzeichnisse

- Liste mit Zeigern auf Dateiblöcke: Einfach, non-performant (Varianten: Bäume, sortierte Listen, ...)
- Hash-Tabelle: Berechnung Hash Wert aus Dateiname, dann Suche in Hash Bucket

Allokation

Zusammenhängend

- Einfache Implementierung

- Dateien können nicht wachsen
- Wahlfreier Zugriff
- Externer Verschnitt
- Allokation zB best, worst, ... fit

Verkettete

- Datei ist Liste von Blöcken
- Beliebige Anordnung
- Sequenzieller Zugriff, aber nicht wahlfrei
- Keine Verschwendung
- Bei beschädigtem Block ganze Datei weg
- Beispiel: FAT. Unbenutzte mit 0 markiert

Indizierte

- Alle Zeiger in Indexblock
- Wahlfreier Zugriff
- Kein Verschnitt
- Overhead durch Index Block

Freispeicherverwaltung

- Bitvektoren / Bitmaps (0: Block Frei, 1: Block Belegt)
- Linked List (effizienter da nur freie gespeichert)
- Gruppieren: Erste n freie Blöcke in einem Block grupieren (Schnell für grosse Mengen freier Platz finden)
- Zählen: Linked list, Zeiger und Anzahl folgender freier Blöcke (inklusive Block selber)
- Space Maps: (ZFS): Partition in Metaslabs, jeder 1 Space Map = Log aller Aktivitäten (Allokation + Freigabe), Aufbau als z.B Baum