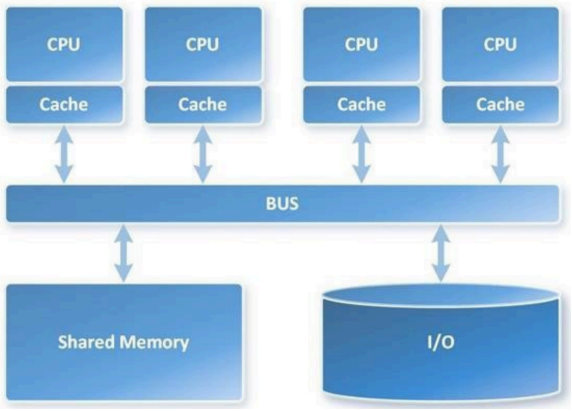
[05] ITEO Summary - Rechner, Virtualisierung, Filesysteme

Symmetrisches Multiprozessing (SMP) / Symmetrische Multiprozessoren (SMP)

Was ist symmetrisches Multiprozessing?

- Ein Standalone Computer mit zwei oder mehr gleichartigen Prozessoren mit vergleichbaren Möglichkeiten
- Prozessoren teilen sich das gleiche Main-Memnory und sich über einen Bus oder andere interne Verbindungen zusammengeschaltet
- Prozessoren teilen sich die I/O Devices
- Alle Prozessoren können die gleichen Funktionen ausführen (daher die Bezeichnung “symmetrisch)
- Das System wird durch ein integriertes OS kontrolliert, Dieses stellt die Interaktion zwischen Prozessoren und deren Programmen auf dem Level von Job-, Task-, File und Daten-Elementen her

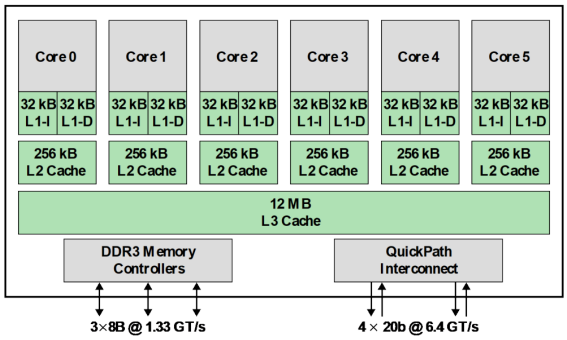
SMP Organisation



Multicore Computer

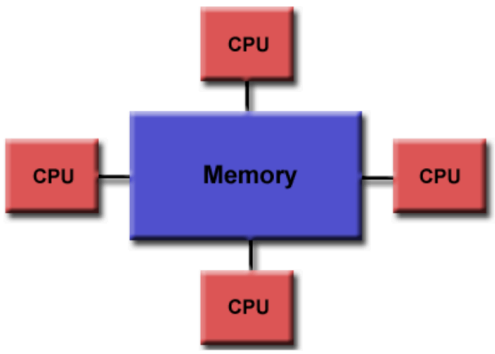
- Auch bekannt als **Chip Multi Processing**
- Kombiniert 2 oder mehr Prozessoren (Cores) auf einem Silicon Die (auf dem Silizium Chip).
- Jeder Core beinhaltet alle Komponenten von einem unabhängigen Prozessor
- Zusätzlich beinhalten Multicore Chips L2 Cache und unter Umständen sogar L3 Caches

Multicore System: Intel Core i7-990X



Unified Memory Architecture (UMA)

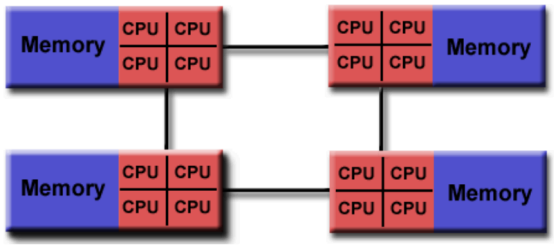
- Alle Prozessoren (CPU) haben direkten Zugriff auf alle Speicherstellen (M)
- einfacher und schneller Kommunikationsmechanismus
- Zusätzlich möglich: Private-Memory, mehrere lokale Addressräume



Shared Memory Architecture (NUMA)

Non-Uniform Memory Access

- meistens werden 2 oder mehr SMPs physikalisch verbunden
- Shared-Memory: ein globaler linearer Addressraum
- Ein SMP kann auf das Memory der anderen SMPs zugreifen
- Nicht alle Prozessoren haben die gleiche Zugriffszeit
- ⇒ Wenn Cache Kohärenz unterhalten wird, wäre es **CC-NUMA**



Memory Stalls — Wartezeiten der CPU

Wartende Applikation — Typische Anz. Zyklen — Memory Stalls (%)

Transactional Database — 3-6 Zyklen — > 75%

Web Server — 1,2 - 2,5 Zyklen — ~50%

Entscheidungs-unterstützende DB — 1-1,5 — ~10-50%

Wartezeiten CPU

- ca 50% der Prozessor Zeit ist Warten auf das Memory

Wie lange wartet ein CPU?

- Bei einer CPU mit 3.3 GHz Taktrate
- Wenn ein CPU Zyklus eine Sekunde betragen würde (real: 0.3 nanosekunden), würde das Daten laden aus Cache L1 3 Sekunden dauern, als L3 Cache 45 Sekunden, aus DRAM 6 Minuten. Das Warten auf Disk Reads bräuchte Monate.

Pipelined Multiprocessing

Bei pipelined Multiprocessing wird das genutzt was frei ist, so kann an verschiedenen Orten gleichzeitig gearbeitet werden. Ein Task blockiert nur eine bestimmte Ressource.

Simultanes Multi Threading (1)

SMT erhöht d ie Zahl der Transistoren um 10%

CMP (**C**hip-**M**ulti-**P**rocessing) erhöht die Zahl der Transistoren um > 50%

Superskalare Architektur

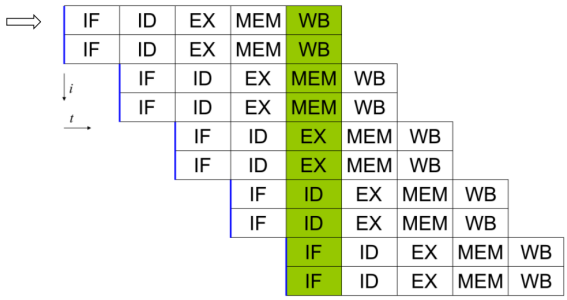
SSC, SuperScalarComputing ist eine Grundfunktion in modernen CPUs, um möglichst effizient zu werden.

- Superscalar auch Instruction Level Parallel genannt

Superscalar Pipeline

Beispiel mit 2 Instruktionen pro Prozessor-Zyklus

- Ein SSC Prozessor erledigt mehr als eine Instruktion während eines Clock-Cycles, indem er mehrere Instruktionen auf mehrere redundante Funktionseinheiten im Prozessor legt.
- Eine funktionale Einheit ist nicht eine CPU sondern eine Ausführungsressource innerhalb einer einzigen CPU, wie z.B. arithmetic logic unit, bit shifter, multiplier



IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, i = Instruction Number,
MEM = Memory access, WB = Register write back, t = Clock Cycle

Leistungsverbesserung

Die Leistungsverbesserung ist nicht linear zu den Ressourcen. Also doppelt so viel Prozessoren ist nicht = doppelte Geschwindigkeit:

1. **Gesetz von Amdahl**: ... ist nicht linear, da nicht alle Teile eines Programms parallelisiert werden können. Es gibt immer einen seriellen Anteil.
2. **Overhead durch Parallelisierung:** Mit Zunahme der Prozessoren steigt der Kommunikations- und Synchronisationsaufwand zwischen Prozessoren. Dieser Overhead kann die Leistungsgewinne reduzieren
3. **Ressourcenkonflikte**: Wenn eine Ressource besetzt wird, kann ein Prozessor nicht mehr weiterarbeiten und muss warten.

Formeln

Einfache Leistungsmasse

$$V(n) = \frac{T(1)}{T(n)}, E(n) = \frac{V(n)}{n}$$

wobei: $V(n)$ = Leistungsverbesserung, $E(n)$ = Effizienz, $T(1)$ = Ausführungszeit für einen Prozessor, $T(n)$ Ausführungszeit für n Prozessoren, n = Anz. Prozessoren

Amdahls Gesetz

$$T = a + \frac{1-a}{n}$$

wobei:
T = Zeit für Programm, a = nicht parallelisierbarer Anteil, n = Prozessoren

Cache Probleme (1) - Write through cache

- Bei dieser Strategie werden Änderungen gleichzeitig sowohl im Cache als auch im Hauptspeicher vorgenommen.
- Der Vorteil dieser Methode ist die einfache Implementierung und die Datenintegrität, da der Hauptspeicher immer aktuell gehalten wird.
- Der Nachteil ist jedoch, dass Schreibvorgänge verlangsamt werden können, da auf den langsameren Hauptspeicher gewartet werden muss, was zu einem Leistungsengpass führen kann.

Cache Probleme (2) - Write back cache

- Hierbei werden Änderungen zunächst nur im Cache gespeichert und erst später in den Hauptspeicher zurückgeschrieben, normalerweise dann, wenn der geänderte Cache-Eintrag ersetzt werden muss.
- Der Vorteil ist, dass Schreibvorgänge schneller sind, da sie nur im schnellen Cache stattfinden und nicht sofort auf den Hauptspeicher warten müssen.
- Ein Nachteil ist die Komplexität der Verwaltung, da sicherzustellen ist, dass keine Daten verloren gehen, zum Beispiel bei einem Stromausfall. Ausserdem kann es zu Inkonsistenzen zwischen den Daten im Cache und im Hauptspeicher kommen.

Cachekohärenz Lösungen

Grundsätzlich 2 Varianten:

1. Cachekohärenzprotokolle
   Es werden Snoopy und Directory-Protokolle unterschieden.
2. Verwendung eines gemeinsamen Chaches:
   Grosser Hardwareaufwand, Sequentialisierung der Speicherzugriffe

Protokolle (Algorithmen)

Snoopingbasiert (Snoop)

Da alle Zugriffe über das gleiche Medium verlaufen (Bus, Switch), können die Cachecontroller beobachten und erkennen, welche Blöcke sie selbst gespeichert haben und diese anpassen.

Verzeichnisbasiert (Directory)

Zentrale Liste mit Status aller Cache Blöcke. Ab 64 Prozessoren oder Cores werden ausschliesslich Directory

basierende Protokolle eingesetzt, da die Bandbreite des Busses nicht ausreichend skaliert.

Snoopy

- Snooping-Protokolle werden häufig in Systemen mit einem gemeinsamen Bus verwendet. Jeder Cache, der an den Bus angeschlossen ist, "lauscht" (engl. "snoop") auf dem Bus nach Transaktionen, die seine Inhalte beeinflussen könnten.
- Wenn ein Prozessor einen Schreibvorgang ausführt, überprüfen alle anderen Caches, ob sie eine Kopie der Daten haben. Wenn ja, invalidieren oder aktualisieren sie ihre Kopie, je nach Kohärenzprotokoll (z.B. MESI).
- Dies sorgt dafür, dass alle Prozessoren eine konsistente Sicht auf die Daten haben. Allerdings kann dieser Ansatz bei einer großen Anzahl von Prozessoren skalenbedingte Probleme aufweisen, da der Bus zum Flaschenhals wird.

Directory

- Directory-basierte Protokolle vermeiden die Skalierungsprobleme von Snooping, indem sie eine zentrale Datenstruktur verwenden, das Directory, das den Status jeder Cache-Zeile im System verfolgt.
- Wenn ein Cache seine Daten aktualisieren möchte, sendet er eine Anfrage an das Directory. Das Directory verwaltet die Kohärenz, indem es entscheidet, welche Caches invalide gemacht werden müssen oder Updates erhalten.
- Diese Methode skaliert besser mit der Anzahl der Prozessoren, da sie nicht auf einen gemeinsamen Bus angewiesen ist. Allerdings kann sie komplexer in der Implementierung sein, weil das Directory eine zusätzliche Abstraktionsschicht darstellt und selbst zu einem Engpass werden kann.

Verbindungsnetzwerke

hohe Leistung → viele Leitungen

niedrige Kosten → wenig Leitungen

Klassifikationskritieren

- Topologie

Kennwerte in Verbindungsnetzen

Viele Begriffe kommen aus der Graphentheorie.

- Inzidenz
- Durchmesser

    - Wie sind die Prozessoren miteinander verbunden (gemeinsames Medium vs geschaltete Verbindung)?
    - Statisch oder dynamisch veränderbar (Verbindungen invariant vs. auf Anforderung etabliert).
- Routing
    - Wie werden die Nachrichten verteilt (store and forward, wormhole, cut-through, circuit switching, packet switching)?
    - Welcher Algorithmus zur Pfadbestimmung wird verwendet (deterministisch, off-line, adaptiv, randomized)?

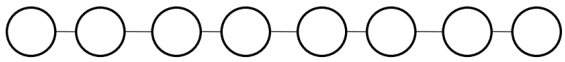
- Grad
- Bisektionsbreite
- Symmetrie
- Skalierbarkeit
- Konnektivität

Berechnung der benötigten Verbindungen:

$\text{Verbindungen} = n \cdot (n-1)$, wobei n = Anzahl Prozessoren

Lineares Feld [Verbindungsnetz]

Das lineare Feld ist das einfachste Verbindungsnetzwerk



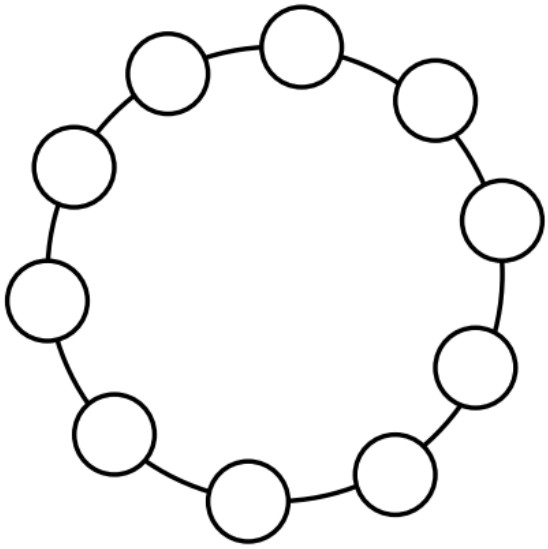
Es ist aber asymetrisch.

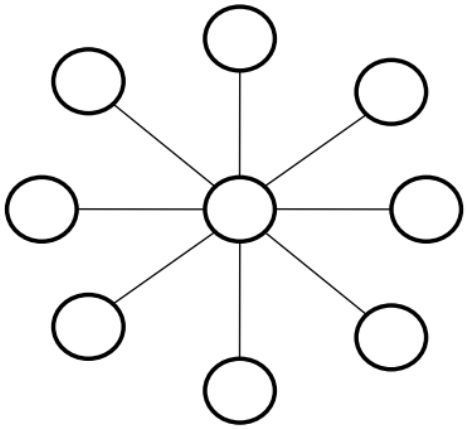
Stern [Verbindungsnetz]

Der Stern is ein asymetrisches Netzwerk mit einem Zentrumsknoten.

Ring [Verbindungsnetz]

Der Ring ist ein symetrisches Netzwerk. Im Gegensatz zum linearen Feld können die Verbindungen im Ring uni- oder bidirektional ausgelegt sein.





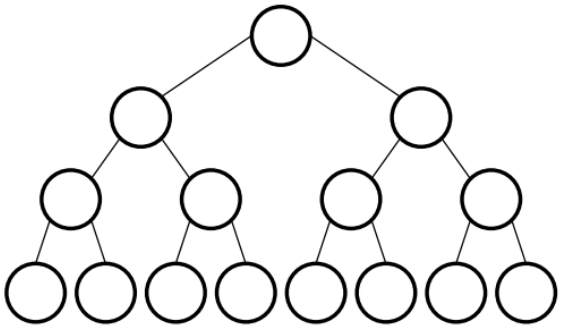
Ring Interconnect

Der Bus besteht z.B. aus 4 unabhängigen Ringen:

Data Ring, Request Ring, Acknowledge Ring, Snoop Ring

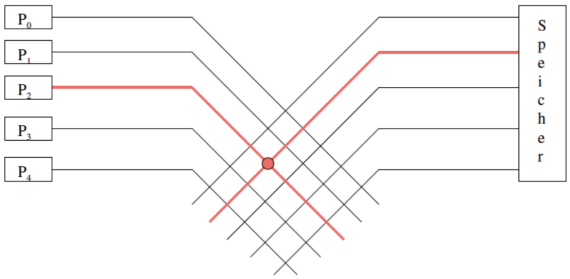
Baum [Verbindungsnetz]

Ein Baum ist ein asymetrisches Netzwerk.

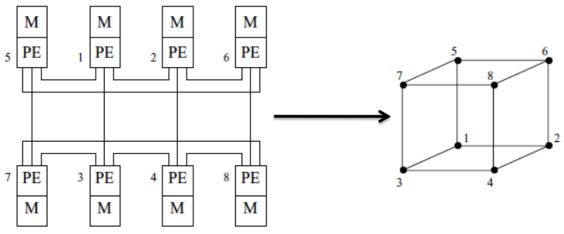


Koppelnetze

Ein anderer Ansatz wäre ein geschaltetes Netzwerk oder Schaltnetzwerk

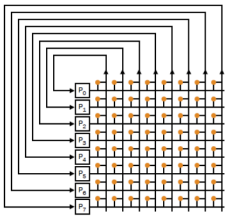


Hypercube [Koppelnetz]

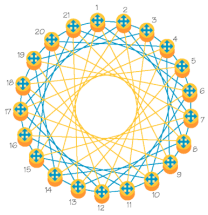


Der Hypercube wird bei einigen modernen Rechnerarchitekturen angewandt.

Crossbar Switch [Koppelnetz]



8 x 8 Crossbar Switch

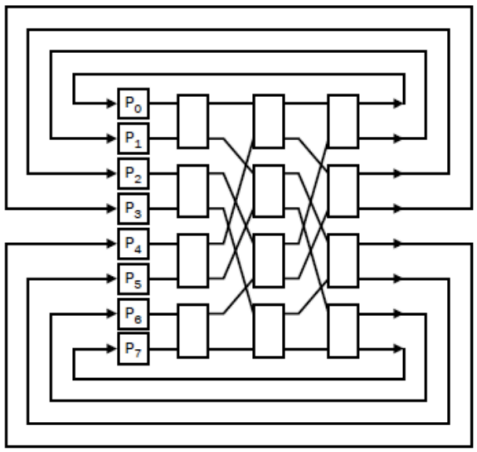


Grad der Knoten: z.B.: 3 Ringe = 2R

Jeder Punkt ist mit jedem Punkt verbunden über eine konstante Anzahl Hops (Weg über einen anderen Knoten).

Crossbar Switches sind nicht blockierend.

Omega Netzwerk [Koppelnetz]



Omega Netzwerke sind blockierend.

Skalierbare Applikationen

- Skalierbare Algorithmen
- Locking — verwenden von fein granularen Locking Mechanismen pro Objekt
- Verwenden von Worker Thread Pools
- Ausführen von Skalier-Tests
- Beobachten der skalierbaren Applikation
- Identifizieren der orte, in welchen wait-time involviert sind
- Identifizieren von heissen synchronization locks
- Identifizieren der nicht skalierbaren Algorithmen

Prozess Elemente (Prozess Kontroll Block)

- **Identifier**: Einzigartiger Prozess Identifier der die Prozesse unterscheidbar macht
- **State**: Der Status in dem sich der Prozess befindet
- **Priority**: Prioritäts Level relativ zu anderen Prozessen
- **Program counter**: die Adresse von der nächsten auszuführenden Instruktion

Prozess Kontroll Block

- Wird verwendet, um laufende Prozesse zu unterbrechen und um sie später wieder laufen lassen zu können
- Wird erzeugt und verwaltet durch das OS
- Schlüsselbaustein, der das gleichzeitige Existieren mehrerer Prozeesse unterstützt und ermöglicht

- **Memory pointers**: Enthält Pointer zu Programm Code und assoziierten Daten mit diesem Prozess inkl. Memory Blocks shared mit anderen Ressourcen.
- **Context data**: Das sind Daten, die momentan in den Prozessor Registern liegen, während der Prozess ausgeführt wird.
- **I/O Status Information**: Beinhaltet noch nicht ausgeführte I/O Anfragen, I/O Devices die diesem Prozess zugeordnet sind.
- **Accounting information:** Beinhaltet ev. Prozessor Zeit und verwendete Clock Cycles

Process Control Structures

- **Wissen des Betriebssystems**: Das OS muss den Standort des Prozesses und alle notwendigen Verwaltungsattribute kennen, wie z.B. Process-ID und Status.
- **Verwaltung und Kontrolle:** Um einen Prozess zu kontrollieren, sind spezifische Informationen notwendig, die dem Betriebssystem erlauben, das Prozess-Verhalten zu steuern und zu verwalten.
- **Prozessstandort**: Ein Prozess enthält ausführbare Programme und muss genügend Speicher für das Programm und seine Daten bereithalten. Ein Stack ist für die Ausführung von Prozeduraufrufen und Parameterübergabe erfoderlich.
- werden in verschiedenen Queues verwaltet

Prozess Ausführung: Memory Layout von 3 gleichzeitige Prozessen

Queuing Modell

- **Prozessauswahl:** Das OS muss ständig die komplette Prozess-Queue durchsuchen, um den geeigneten Kandidaten für die Ausführun gzu finden. Das kann hunderte oder tausende Prozesse umfassen, deshalb verwendet man mehrere Queues.
- **Event-Handling**: Wenn ein Event eintrifft, werden alle darauf wartenden Prozesse aus der entsprechenden Queue in die “ready Queue” verschoben, die bereit sind, ausgeführt zu werden.
- **Queue-Struktur**: Anstatt einfache FIFO-Queues zu verwenden, greift das OS auf doppelt verkettete Listen zurück, um die Prozess-Queues zu verwalten.
- **Ready und Blocked Queues**: Prozesse die im System laufen sollen, werden in die Ready Queue platziert. Wenn ein laufender Prozess pausiert wird, wird er je nach Situation in die Ready oder Blocked Queue verschoben
- **Effizienz und Priorisierung**: Für jedes Event spezifische Queues zu haben kann effizienter sein, vor allem in grossen Systemen. Dies

- **Prozessattribute**: Jeder Prozess hat eine Reihe von Attributen, die für die Kontrolle durch das OS notwendig sind. Diese umfassen das Prozess-Image, das aus dem Programm, den Daten, dem Stack und den Attributen besteht.
- **Speichermanagement**: Die Lokation des Prozess-Images hängt vom verwendeten Speichermanagement-Schema ab. Moderne Betriebssysteme nutzen Paging, wobei ein Teil des Prozesses im Hauptspeicher und der Rest auf der Festplatte sein kann. Das OS muss die Position jeder Seite des Prozess-Images verfolgen.

ermöglicht es, bei einem Event schnell alle wartenden Prozesse zu aktivieren. Das OS kann auch verschiedene Queues für unterschiedliche Prioritäten nutzen, um schnell den Prozess mit höchster Prio zu finden.

Prozess-Status-Wechsel

- Kontextspeicherung: Der Kontext des Prozessors wird gespeichert, einschliesslich des Program-Counters und anderer Register.
- Aktualisierung des PCB: Der Prozesskontrollblock (PCB) des laufenden Prozesses wird aktualisiert, einschliesslich Statusänderungen und Buchhaltungsinformationen.
- **Queue-Management**: Der PCB wird in die entsprechende Queue verschoben.
- **Prozessauswahl**: Ein anderer Prozess wird zur Ausführung ausgewählt und dessen PCB aktualisiert.

Modus Wechsel

- **Interrupt-Handling**: Bei Eintreffen eines Interrupts wechselt der Prozessor vom Benutzer- in den Kernelmodus, um den Interrupt zu behandeln.
- **Leichtgewichtigkeit**: Ein Moduswechsel kann ohne Prozess-Status-Wechsel durchgeführt werden und verursacht nur einen geringen Overhead

Prozess-Status-Wechsel vs. Modus Wechsel

- **Vergleich**: Ein Moduswechsel ist ein weniger komplexer Vorgang im Vergleich zum Prozess-Status-

- **Speicher-Management**: Die Speicherverwaltungsdatenstrukturen werden entsprechend aktualisiert.
- **Kontextrestaurierung**: Der Prozessorkontext wird wiederhergestellt, damit der Prozess an der Stelle fortgesetzt werden kann, an der er unterbrochen wurde

Wechsel, der eine vollständige Prozessumschaltung beinhaltet.

- **Overhead**: Während der Moduswechsel mit geringem Aufwand verbunden ist, erfordert der Prozess-Status-Wechsel substantielle Änderungen im Betriebssystemumfeld

Prozess erzeugen

Ein Prozess wird über den Kernel system call `fork()` erzeugt.

1. **Platzierung in der Prozesstabelle**: Das Betriebssystem alloziert einen freien Platz in der Prozesstabelle für den neuen Prozess.
2. **Prozess-ID Zuweisung**: Dem Kindprozess (Child Prozess) wird eine einzigartige Prozess-ID vergeben.
3. **Kopie des Prozess-Images**: Eine Kopie des Prozess-Images des Elternprozesses (Parent Prozess) wird angefertigt, ausgenommen der geteilten Speicherbereiche.
4. **File-Counter Inkrementierung**: Zähler für jedes vom Elternprozess genutzte oder besessene File werden inkrementiert, um zu kennzeichnen, dass nun ein weiterer Prozess auf diese Files zugreift.
5. **Zur "Ready to Run" Queue hinzufügen**: Der Kindprozess wird der "ready to run" Queue zugeordnet.

Unterbrechung der Ausführung eines Prozesses

- **Hardware Interrupt**: Externe Ereignisse, wie I/O-Geräte oder Timer, die asynchron auftreten.
- **Hardware Trap**: Synchron oder asynchron, im Zusammenhang mit dem aktuellen Prozess, z.B. Division durch Null oder ungültige Pointer.
- **Software-Initiated Trap**: Vom System genutzt, um Ereignisse zu erzwingen, wie Prozess-Re-Scheduling oder Netzwerkpaketverarbeitung

Interrupt Prioritäten

- **Prioritäten**: Von hoch bis niedrig, einschließlich Maschinenfehler, Zeitgeber, Disk, Netzwerk, Terminals und Software Interrupts.
- **Prozessumschaltung**: Kann jederzeit erfolgen, wenn das Betriebssystem die Kontrolle vom laufenden Prozess übernimmt

6. **Rückgabe der Prozess-ID**: Die ID-Nummer des Kindprozesses wird an den Elternprozess zurückgegeben, während der Kindprozess den Wert 0 erhält.

System Calls (Traps)

- **Definition**: System Calls sind software-initiierte Traps.
- **Ablauf**:
    1. Aufrufendes Programm legt Parameter auf den Stack.
    2. Sprung in eine Bibliotheksfunktion.
    3. Nummer des Systemaufrufs wird in ein Register geladen.
    4. Trap-Ausführung bewirkt einen Sprung zu einer festen Adresse im Kernel.
    5. Zuordnung der Nummer zu einem Funktionszeiger und Start des Systemaufrufs.
    6. Nach Beendigung des Systemaufrufs wird die Kontrolle an die Bibliotheksfunktion und dann an das Benutzerprogramm zurückgegeben, worauf das Benutzerprogramm weiterläuft

Prozess Kontext

Benutzer Kontext

- zugewiesener Adressraum und Daten

Hardware Kontext (multitasking relevant)

- Inhalte CPU Register
- weitere wichtige Infos wie z.B. Seitentabelle

System Kontext (Sicht OS)

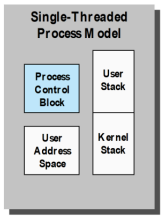
- Prozessnummer, geöffnete Dateien, Info Eltern und Kind Prozess, Prioritäten, etc.

Single und Multi Thread Prozess Modell

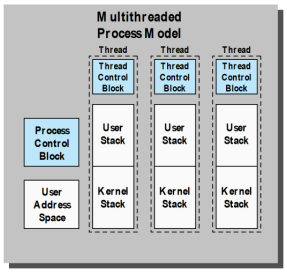
Vorteile Threads

- Es braucht weniger Zeit, um einen Thread zu erzeugen als einen Prozess

Ein Prozess



Ein Prozess mit 3 Threads



- Ein Wechsel zwischen Threads braucht weniger Zeit als einen Wechsel zwischen Prozessen
- Die Kommunkation zwischen Threads ist durch den gemeinsamen Prozess Kontext auf natürlcihe Weise gegeben.

Threads

- In einem modernen OS wird das Scheduling und Dispatching auf der Basis von Threads gemacht
- Die Status Informationen welche für die Ausführung eines Threads verwendet werden, sind in den Thread-Level Daten Strukturen abgelegt
- Einen Prozess zu beenden bedeutet die Unterbrechung aller Threads dieses Prozesses
- Einen Prozess zu terminieren, bedeutet das alle Threads im Prozess beendet werden.

Thread Kontext

- Ein Prozess besteht aus mind. 1 Thread
- Threads ovm gleichen Prozess teilen sich
  - Codesegment
  - Datensegment
  - Dateideskriptoren
- Jeder Thread hat einen eigenen Stack

Kernel-Level Threads (KLTs)

- **Thread Management**: Wird komplett vom Kernel ausgeführt, Anwendungen haben keine Thread-Verwaltungsfunktionen.
- **Vorteile von KLTs**: Der Kernel kann Threads eines Prozesses auf verschiedene Prozessoren verteilen und blockierte Threads durch andere ersetzen, was multithreaded Kernel-Prozeduren ermöglicht

- Jeder Thread hat seinen eigenen Code Pointer

Kernel Threads

- **Funktion**: Kernel-Threads sind die grundlegende Einheit, die zum Prozessor verlegt wird, und ihr Wechsel ist schnell.
- **Verwendung**: Sie werden für Kernel-Aufgaben wie Prozessausführung und Interrupt-Handling eingesetzt

System Calls im Gesamtkontext

API: Die Schnittstelle zwischen User-Prozessen und dem Kernel wird als Syscall oder "System Calls" bezeichnet

Zwei Level Thread Modell

- **Thread Erzeugung**: Obwohl das Verschieben von Threads schnell ist, sind Erstellung und Löschung ressourcenintensiv; jeder Thread benötigt einen eigenen Stack.
- **LWPs**: User-Threads werden auf Light Weight Processes (LWPs) abgebildet, und nur Threads auf LWPs sind aktiv

Thread Objekte

Kern Objekt: Für multithreaded Ausführung verwendet das Betriebssystem Kernel-Threads in Verbindung mit LWPs, was unabhängige Ausführung von User-Threads ermöglicht

Virtualisierte Umgebung

Hypervisor: Physische Hardware wird virtualisiert, um mehrere virtuelle Maschinen gleichzeitig auf einer physischen Hardware zu erstellen, was für die HW-Virtualisierung relevant ist

Server Virtualisierungsarten

- **HW Partitionierung**
  Ist nur noch auf High End Computern anzutreffen, ist teuer (IBM z und p Systems)
- **HW Virtualisierung (Hypervisor)**
  Physische Hardware wird virtualisiert, um mehrere virtuelle

Vorteile der (Hypervisor) Virtualisierung

- Workload Isolation
- Workload Consolidation
- Workload Migration
- Workload Embedding

Maschinen gleichzeitig auf einer physischen Hardware zu erstellen

- **OS Virtualisierung**
  Verwenden Kernel vom Host (Container). Sind light-weight.

Virtual Machine Monitor (VMM) (Hypervisor)

- **Funktion**: VMM schafft eine Umgebung, die der physischen Maschine entspricht, mit minimaler Geschwindigkeitsreduzierung für die darin laufenden Programme.
- **Aufbau**: VMM besteht aus einem Control Program mit Dispatcher, Allocator und Interpreter für privilegierte Instruktionen

Instruktionsklassifikation

- **Betriebsmodi**: Mindestens zwei Modi sind vorhanden - Supervisor (uneingeschränkt) und User Modus (eingeschränkt).
- **Instruktionstypen**: Privilegierte (sensitive) Instruktionen trappen im User Modus, während harmlose Instruktionen direkt ausgeführt werden

Emulation / Simulation

Emulation bezieht sich auf das Nachahmen der Funktionalität eines Systems oder Geräts, sodass es sich wie das emulierte System verhält. Dies wird häufig verwendet, um Software, die für eine bestimmte Hardware entwickelt wurde, auf anderer Hardware laufen zu lassen.

Simulation ist der Prozess der Nachbildung eines realen Prozesses oder Systems über die Zeit. In der Informatik wird sie verwendet, um das Verhalten eines Systems unter verschiedenen Bedingungen zu modellieren und zu analysieren.

Virtualisierung

Virtualität ist die Eigenschaft einer Sache, nicht in der Form zu existieren, in der sie zu existieren scheint, aber in ihrem Wesen oder ihrer Wirkung einer in dieser Form existierenden Sache zu gleichen.

Virtualisierung bezieht sich auf die Erstellung einer virtuellen (statt physischen) Version von etwas, wie z.B. ein Betriebssystem, Server, Speichergerät oder Netzwerkressourcen. Es ermöglicht die Ausführung mehrerer Betriebssysteme und Anwendungen auf einer einzelnen physischen Maschine.

Formale Definition eines Hypervisors / Anforderungen an physische und virtuelle Systeme

- **Gleichheit**: Programme in einer VM verhalten sich wie auf der physischen Maschine.
- **Effektivität**: Instruktionen sollen direkt auf dem physischen Prozessor ausgeführt werden, soweit möglich.
- **Ressourcenkontrolle**: VMM kontrolliert alle Ressourcen, Programme dürfen die Systemressourcen nicht beeinflussen

Theorem

- **VMM Aufbau**: Effektive VMMs können für Prozessorarchitekturen der dritten Generation gebaut werden, wenn kontrollkritische Instruktionen eine Untermenge der privilegierten Instruktionen sind.
- **Kontrollkritische Instruktionen**: Um einen VMM zu bauen, müssen alle Instruktionen, die das Funktionieren der VMM beeinflussen können, immer getrapped und vom VMM verarbeitet werden.
- **Nicht-privilegierte Instruktionen**: Diese müssen direkt auf dem Prozessor ausgeführt werden, um die Effizienz des VMM zu gewährleisten.

x86 Virtualisierbarkeit

- **Virtualisierungsgrundlage**: Gemäss Popek und Goldberg ist eine CPU virtualisierbar, wenn alle privilegierten Instruktionen in einem unprivilegierten Modus eine Exception erzeugen.
- **x86 Herausforderungen**: x86-Architekturen enthalten "Stealth Instructions", die nicht durch Traps abgefangen werden können, was die Virtualisierung erschwert.
- **Lösungen**: VMware's Binary Translation-Ansatz als Lösung für

Hypervisor Arten

- **Typ 1 Hypervisor**: Direkt auf der Hardware installiert (z.B. VMware ESXi, XEN, Hyper-V). Also **ohne OS auf der Hardware**! *HW → VMM → [VM → OS → Apps]'s*.
- **Typ 2 Hypervisor**: Auf einem Host-Betriebssystem installiert (z.B. VMware Workstation, KVM, Virtual Box)

Hypervisor-Platzierung und CPU Sicherheitsringe

CPL = Current Privilege Level

nicht abfangbare Instruktionen

Vollvirtualisierung mit Binary Translation

- **Funktionsweise**: Priviligierte und sensitive calls trappen automatisch zum Hypervisor ohne Binary Translation.
- MIT RING "-1"

Hardware-unterstützte Virtualisierung

- **Technologie**: Neuere CPUs bieten spezielle Modi und Erweiterungen zur Unterstützung der Virtualisierung, wodurch die Notwendigkeit für Techniken wie Binary Translation verringert wird.
- **Funktionalität**: Diese CPUs erlauben das automatische Weiterleiten von privilegierten und sensiblen Aufrufen an den Hypervisor und unterstützen die Verwaltung des Gast-Status über spezielle Strukturen

Binary Translation

Performance: Während die Binary Translation effektiv ist, verursacht sie einen Leistungs-Overhead im Vergleich zu nativen Systemaufrufen.

- **CPU Sicherheitsringe**: Die x86-Architektur verwendet vier verschiedene Privilegienlevel, bekannt als Ringe, wobei Ring 0 das höchste Privileg hat (für das Betriebssystem) und Ring 3 das niedrigste (für Benutzeranwendungen).
- **Hypervisor-Integration**: Ein Hypervisor muss so konzipiert sein, dass er unterhalb von Ring 0 agiert, um Kontroll- und Managementaufgaben für das Gast-Betriebssystem durchführen zu können, ohne dass dieses direkten Hardware-Zugriff benötigt

Paravirtualisierung

- **Kommunikation**: Paravirtualisierung ist eine Technik, bei der das Gast-Betriebssystem modifiziert wird, um direkt mit dem Hypervisor zu kommunizieren und somit die Leistung zu steigern.
- **Implementierung**: Das Gast-OS nutzt spezielle "Hypercalls" anstelle von herkömmlichen Systemaufrufen, um mit dem Hypervisor zu interagieren.

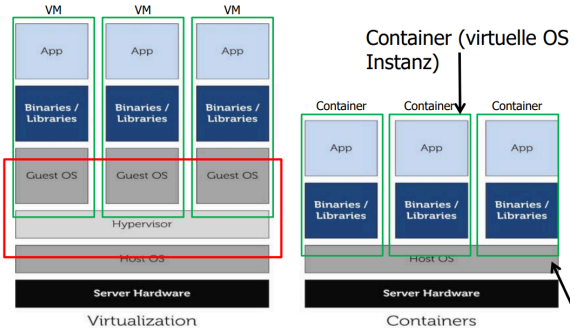
Speichervirtualisierung und -management

- **Memory Management**: Speichervirtualisierung ermöglicht es, dass mehrere Gast-Betriebssysteme auf einem

Hypervisor laufen, wobei jeder Gast eine eigene virtuelle Speicheransicht hat.

- **Ressourcenoptimierung**: Techniken wie Page Sharing und Compression werden verwendet, um Speicherplatz zu optimieren, während Ballooning genutzt wird, um Speicher dynamisch zwischen VMs neu zuzuweisen.

OS-Virtualisierung vs. HW-Virtualisierung Container vs. VMs



Provisionierung im Vergleich

- BareMetal ist heavy-weight (8-24h)
- VM's sind mid-weight (5-10min)
- Container sind light-weight (5-15s)

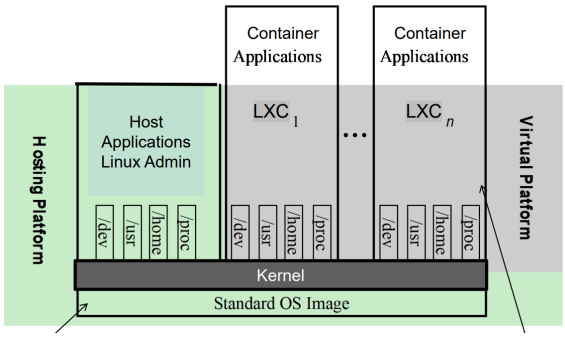
Container

- sind komplett isoliert von anderen Prozessen
- mehrere Container können dieselbe Betriebssysteminstanz verwenden. Das OS wird nicht mehrfach benötigt, sondern nur einmal auf dem Host.

Linux Container (LXC) Architektur

Operating System Virtualisierung

- verwendet keinen Hypervisor
- also kein "Trap and Emulate"
- braucht daher auch nur ganz wenige zusätzliche Prozessorzyklen



(z.B. mapping von UID zu vUID).

- verwendet nur einen Kern (Solaris kennt zusätzlich die Kernel-Zones, was eine Ausnahme darstellt, weil so mehrere Kerne unabhängig verwendet werden können)

Was kann die OS Virtualisierung

- Stellt eine feingranulare Kontrolle der individuellen Prozesse und Applikationen zur Verfügung
- Erlaubt die Isolation von Applikationen
- Transparente Migration von Applikationen möglich, im Gegensatz zum Hypervisor, der nur ganze OS Instanzen migriert.

Was kann die OS Virtualisierung nicht

- Keine fremden OS möglich. Nur gleiche OS können virtualisiert werden. Beispiel: kein WIndows Programm möglich auf Linux Kern.
- Zonen müssen auf dem selben Patch Level laufen wie OS/Kern.
- Anforderungen an SysAdmin steigen (beim HyperV aber auch)
- Ressourcen Management muss eingeschaltet werden

Welche OS beherrschen OS Virtualisierung

- BSD — Jails (Idee entstan den)
- Solaris — Zonen (Begriff wurde geprägt)
- Linux — Container (LXC) - im mainstream Kern enthalten
- Microsoft — Container (ab Windows Server 2016)
  Kubernetes (ab Windows Server 2019)

Erweiterungen für Container-Virtualisierung

Es braucht mindestens 3 wichtige Erweiterungen:

- Filesystemgrenzen für Prozesse: chroot
- Ressourcen-Sharing zwischen Prozess Gruppen: cgroups (control groups)
- Namespaces: ns oder namespaces

Neben diesen hat der Linux Kern bereits alles, was es für die Unterstützung eines modernen Container Systems braucht.

Isolation (mit chroot)

chroot ist eine Operation, die das root Directory für den aktuell laufenden Prozess und dessen Kinder ändert / verschiebt und vom Rest der Maschien vollständig isoliert.

Ein Programm das in diesem modifizierten Umfeld läuft, kann auf keine Dateien und Kommandos mehr ausserhalb dieses Verzeichnis Baumes zugreifen.

Das modifizerte Umfeld nennt man **chroot jail**.

Ressourcen Management (mit cgroups)

cgroups sind ein Kernel Feature. Sie erlauben das Allozieren von Systemressourcen über eine benutzerdefinierte Gruppe von Tasks (Prozesse).

cgroups erlauben dem Administrator eine Kontrolle über die Zuordnung, Priorisierung, Verwaltung und Überwachung von Systemressourcen.

HW Ressourcen können so unter Benutzern und Tasks (Prozesen) afgeteilt werden.

Isolation im Namensraum (mit Namespaces)

- Namespaces abstrahieren eine globale System Ressource in der Art, dass der zum Namensraum gehörende Prozess meint, er besitze diese Ressource einzigartig.
- Änderungen an dieser globale Ressource sind nur sichtbar für die Mitglieder im gleichen Namensraum. Für andere Prozesse sind sie nicht sichtbar.
- Eine offensichtliche Verwendung von Namensräumen ist die Implementation von Contianern.

cgroups Prozess Modell

- cgroups ermöglichen das erstellen von mehreren Process-Tree's. Traditionell im Linux/Unix-Prozessmodell gibt es nur einen Process-Tree.
- **Mehrere Hierarchien:** Jede dieser Process-Subtree's kann an ein oder mehrere Subsysteme gebunden sien. Also gibt es mehrere Prozesshierarchien, wobei jede Hierarchie eine bestimmte Ressource oder Gruppe von Ressourcen verwaltet.
- **Subsysteme**: Ein Subsystem in cgroups repräsentiert eine einzelne

Namespaces sind bereits bekannt durch: Java, C#, XML, C++

Ressource wie CPU-Zeit, Arbeitsspeicher (Memory) und ähnliche Ressourcen. Linux verfügt über verschiedene solcher Subsysteme, die für die Feinsteuerung der Ressourcenverteilung auf die Prozesse genutzt werden können.

Docker und LXC Container

- Durch Docker wird das darunter liegende OS abstrahiert und damit neue Schnittstellen präsentiert
- Instanzen besitzen danach nur die benötigten zusätzlichen Runtimes aus dem eigenen OS (Image)
- Docker verwendet die Möglichkeit von LXC um die Applikation innerhalb eines Containers bereitzustellen
- Docker ist ein Software Layer über dem Linux (LXC) Container
- Container sind Prozess-Virtualisierungen. Erreicht durch Separierung der Prozesse in Namespaces und Zuweisung von Ressourcen zu Prozessen in diesen Namensräumen durch cgroups.

Docker vs. LXC

- LXC ist eine Container Technologie
- Docker ist eine Applikations-Engine basierend auf Containern
- Docker wird hauptsächlich verwendet für die Applikationsverteilung und bietet neue Schnittstellen an, wodurch auf unterschiedlichen System die selben Container mithilfe Docker gestaretet werden können.

Hier und in den Unterlagen wird das ZFS File System als Vertreter eines modernen Filesystems betrachtet.

Anforderungen an Moderne Filesysteme

- grosse Adressierbarkeit
- eingebauter Volume Manager
- umfangreiche Funktionalitäten
- immer konsistent und integer
- kein file system check mehr nötig nach Absturz
- kann mit “silent corruption” umgehen

Volumes

- Volumes fassen verteilte Speicherblöcke zu logischen Einheite, sog. Volumes zusammen
- Volumes werden in der Filesystemstruktur an geeigneten Stellen eingebunden (gemounted).
- Jedes Volume kann eigenständig verwaltet werden (resize, move, snapshot, ...)
- Speicher-Redundanzen können auch über Volumes folgen

Filesystem/Volume, I/O Stack

- **Block Device Interface**: Befehle werden als eine Serie von "Schreibe diesen Block, dann den nächsten Block, ..." Anweisungen ausgeführt. Bei einem Stromausfall besteht das Risiko eines Inkonsistenzproblems auf der Festplatte.
- **Workaround**: Als Lösung werden Journaling-Methoden verwendet, um die Konsistenz zu gewährleisten, die jedoch langsam und komplex sein können.
- **Block Device Interface (Volume)**: Jeder Block wird sofort auf die Festplatte geschrieben, um Spiegelungen (Mirroring) synchron zu halten. Bei einem Stromausfall ist

ZFS I/O Stack

- **Object-Based Transactions**: Statt einzelne Blöcke zu schreiben, werden Transaktionen als Änderungen an Objekten durchgeführt ("Mache diese 7 Änderungen an diesen 3 Objekten"). Diese Transaktionen sind atomar, das heisst, sie werden entweder vollständig oder gar nicht ausgeführt.
- **Transaction Group Commit**: Diese Transaktionen sind auch für ganze Gruppen von Änderungen atomar. Sie garantieren, dass die Festplatte immer konsistent ist, und ein Journaling ist nicht erforderlich.
- **Transaction Group Batch I/O**: Hier werden I/O-Anforderungen geplant,

ein Resynchronisieren erforderlich, und dieser Prozess ist synchron und kann langsam sein.

aggregiert und nach Bedarf ausgeführt. Im Falle eines Stromausfalls ist kein Resync erforderlich, und der Prozess kann mit der vollen Geschwindigkeit der Festplattenplatter (physisches Speichermedium der Festplatte) laufen.

ZFS — Ein Merkle Tree

Ein Merkle Tree wird in der Literatur oft auch als HASH-Baum bezeichnet. HASH-Wrte der Kinder werden in den Eltern gespeichert.

- Jeder innere Knoten besitzt Hash von seinen Kind Knoten
- Verifikation eines Knotens in Zeit $O(log(n))$
- Merkle Signatur Schema braucht nur einen public Key für 2^n Knoten Signaturen

Konsistenzen von Daten

Um Konsistenz herzustellen oder wiederzuerlangen gibt es verschiedene Methoden und Anforderungen:

- `Fsck` oder `chkdsk`: angewendet auf grosse Volumes kann dies von einigen Minuten bis zu mehreren Tagen in Anspruch nehmen. In der Zwischenzeit herrscht Unsicherheit oder Pause.
- Zu jeder Zeit möchte man wissen, ob Daten korrupt sind. Nicht erst

Block Pointer [ZFS]

Ein Block Pointer in ZFS ist eine 128-Byte-Datenstruktur, die dazu dient, Datenblöcke auf einer Festplatte physisch zu lokalisieren, zu überprüfen und zu beschreiben. Jeder Block Pointer enthält Informationen über:

- **DVA (Data Virtual Address)**: Eine DVA ist das Äquivalent zu einer Blocknummer in anderen Dateisystemen und zeigt auf den Speicherort der Daten auf dem physischen Speichermedium.
- **Checksum**: Eine Prüfsumme zur Überprüfung der Datenintegrität. Sie gewährleistet, dass die Daten korrekt gelesen und geschrieben werden.
- **Ditto Blocks**: Kopien von Datenblöcken, die zur Erhöhung der Datenintegrität und Fehlertoleranz dienen.

RoW und CoW

RoW: Neue Schreibanfrage möchte die Daten in den alten Block schreiben. Diese werden aber in einen neuen Block

beim Verwenden der Daten. [Selbstheilendes ZFS]

- **CoW: “Copy on Write”** und **RoW “Redirect on Write”** werden angewandt, um jederzeit konsistent zu sein.

ZFS CoW eher ein RoW

Bei ZFS wird oft von CoW gesprochen, ist aber eher ein RoW.

- ZFS verwendet ein “reallocate or redirect on write”
- Blöcke mit aktiven Daten werden nie überschrieben
- Der Intent Log (ZIL) wird verwendet, wenn synchrone Schreib-Semantik verlangt wird.

3-2-1 Regel

3 Kopien auf 2 verschiedenen Medien

Selbstheilendes ZFS

In einem selbstheilenden ZFS werden korrupte Daten sofort bemerkt (durch die Hashes) und zerstörte/veränderte Daten nicht weiterverwendet. Der Lesevorgang erhält korrekte Daten und die defekten Daten werden automatisch korrigiert.

geschrieben (redirect). Der Originalblock enthält immer noch die alten Daten. Erst nach erfolgter Anpassung der Verkettungen dürfen diese wieder überschrieben werden.

CoW: Die alten Daten werden in einen neuen Speicherblock kopiert. Die Daten im alten Speicherblock werden danach mit den neuen Daten überschrieben. Die Kopie kann bestehen bleiben (Snapshot) oder erst nach abgeschlossener Anpassung der Verkettung wieder freigegeben werden.

Snapshots

Snapshots sind momentaufnahmen aus dem Filesystem, wobei die Speicherblöcke mit deren Metadaten in einem bestimmten Zustand eingefroren werden und jederzeit der “current pointer” wieder auf diese Bausteine verlinkt werden kann. Dies würde bezwecken, dass dann das Arbeiten von diesem Standpunkt aus wieder beginnt.

Snapshots ≠ Backups

Snapshots sind keine Backups. Backups speichern Daten auf einem anderen Drive.

Erasure Code

Erasure Code encodes k Data Disks in m coding Disks. Wenn m Disks ausfallen, werden deren Daten vom erasure Code wieder hergestellt.

Traiditionelle RAID 4 und RAID 5

- Mehrere Daten Disk und eine Parity Disk
- Ein kurzer Stromausfall oder eine andere Störung beim Schreiben zwischen Data und Parity resultiert in “silent data corruption” und wird evtl erst beim Recovern bemerkt.

RAID Z [RAID-Level im ZFS]

- Variable Block Grösse 512B - 128K
- Single und Double Parity
- Detektiert Silent Data Corruption
- Checksummen getriebene Kombinatorische Rekonstruktionen
- Schützt vor mehreren Block Ausfällen wenn nicht reduziert
- RAID-Z (RAID 5) verwendet 1 Parity Disk und verträgt den Ausfall von bis zu 1 Disk
- RAID-Z2 (RAID 6) verwendet 2 Parity Disks und verträgt den Ausfall von bis zu 2 Disks
- RAID-Z3 (Raid 7) verwendet 3 Parity Disks und verträgt den Ausfall von bis zu 3 Disks.

ZFS Skalierbarkeit

- Wahnsinnige Kapazität (bei 128 Bit-Worten)

Layout von Striping

Das Layout von Striping bezieht sich auf die Art und Weise, wie Daten auf mehrere Festplatten verteilt werden. Bei einem Striping-Layout werden Daten in Blöcken (oder Stripes) aufgeteilt und über eine Gruppe von Festplatten verteilt.

Bsp. bei einem System mit **vier Festplatten (n=4)**: Jeder Stripe besteht aus **drei Datenblöcken (k=3)** und **einem Kodierungsblock (m=1)**. Der Kodierungsblock wird für die Fehlerkorrektur verwendet und ermöglicht die Wiederherstellung der Daten, falls eine der Festplatten ausfällt.

Das Filesystem

- Kann mit allen Fehler-Klassen umgehen
    - Bit rot, Phantom writes, Misdirected read and writes, administrative errors
- Disk Scrubbing (nimeand kann gelöschte Daten lesen)
- Resilvering (rebuilding, reconstructing datas before use)
- Real time remote replication (zfs send, zfs receive)
- encryption
- Data deduplication

- Moore’s Law sagt: Wir brauchen das 65. Bit erst in 10-15 Jahren
- ZFS Kapazität: 256 Quadrillionen ZB (1 ZB = 1 Milliarde TB)
- Übersteigt die gesamte Speicherkapazität dieser Erde
- Die Kapazität von ZFS ist so ausgelegt, dass sie für immer ausreicht.