PERSONAL - Grundlagen: Betriebssysteme und Systemsoftware: Begriffe

- **EDITOR'S NOTE**: Dieses Dokument erhält nur übersichtliche Definitionen für manche fachchinesische Begriffe. Insbesondere werden hier Algorithmen und Übungsschemas **nicht** behandelt. Für die genaueren Funktionsweisen oder weitere Definitionen empfehle ich stattdessen die Vorlesungsfolien und Tutorblätter.

Einführung in GBS

- **Rechensystem / Computer**: offene, dynamische, technische Systeme mit der Fähigkeit, **Informationen zu speichern und zu verarbeiten** und mit anderen Systemen zu **kommunizieren**
- **offenes System**: System, welches mit seiner Umgebung via Schnittstellen extern kommunizieren kann
  - **Black-Box-Sicht**: Komponentensicht, innere Implementierung von außen **verborgen**
  - **White-Box-Sicht**: Komponentensicht, innere Implementierung von außen **sichtbar**
- **dynamisches System**: System bestehend aus **aktiven und passiven Komponenten**, dessen Zustand sich konstant ändert
  - **aktive Komponenten**: Komponenten, die eine **Zustandsänderung** bewirken (z.B. CPU)
  - **passive Komponenten**: Hilfsmittel der aktiven Komponenten (z.B. RAM)
- **Betriebssystem (Operating System, OS)**: Software, das die **Ausführung von System- und Benutzeranwendungen überwacht** und eine **Schnittstelle zu den Hardwarekomponenten des Systems** bereitstellt
  - **General Purpose OS**: komplexe Betriebssysteme, die eine **Vielzahl an Prozessen** verarbeiten und auf **verschiedener Hardware** laufen kann (z.B. Linux, MacOS, Windows)
  - **Special Purpose OS**: einfache, eingeschränkte Betriebssysteme, die **nur für einen bestimmten Zweck** und für bestimmte Hardware entwickelt wurden
- **Betriebsmodus**: Unterteilung der Berechtigungen zw. Benutzer und Betriebssystem zum Schutz der Daten vor Fehlern und Angriffen
  - **Benutzermodus (User Mode / Space)**: begrenzter Modus für Benutzeranwendungen; kein direkter Hardwarezugriff, keine privilegierten Befehle, kein / nur lesender Zugriff auf Systemcode bzw. -daten, nur virtuelle Adressen erlaubt
  - **Systemmodus (Kernel Mode / Space)**: privilegierter Modus; OS-Kern läuft im Kernel Mode
- **Systemaufruf (System Call, Syscall)**: Schnittstelle, damit Benutzeranwendungen bestimmte **OS-Dienste** nutzen können; ähneln Funktionsaufrufe

Betriebssystemarchitekturen

- **Monolithisches System**: OS-Kern beinhaltet mehrere Funktionen und befindet sich konstant im **Hauptspeicher**
- **Mikrokern-Architektur**: Unterteilung in **Mikrokern** (nur Basismechanismen, Kernel Mode) und **Systemdienste** (weitere Subsysteme, User Mode)

Betriebsarten

- **Stapelverarbeitung (Batch Processing)**: eine bereitgestellte Menge an Daten bzw. ein Programm wird *vollständig, automatisch und meist ohne Nutzerinteraktion verarbeitet*
- **Transaktionsbetrieb (Transaction System)**: Aufträge werden in Form von *Transaktionen* verarbeitet (insb. Datenbanken), erfüllt *ACID-Kriterien* (Atomicity, Consistency, Isolation, Durability)

- **Dialogbetrieb (Time Sharing)**: Gegenteil von Batch Processing, erfordert *Interaktion zwischen Benutzer und Programm / OS über ein (G)UI*
- **Echtzeitsysteme (Real Time System)**: Systeme, die an sie gestellte *Echtzeitanforderungen* erfüllen müssen
  - **hard deadline**: Reaktionszeit darf nicht überschritten werden (z.B. Robotik)
  - **soft deadline**: Reaktionszeit darf einigermaßen überschritten werden (z.B. Multimedia)

Prozess- und Prozessorverwaltung

- **CPU-limitiert**: ein Prozess würde schneller laufen, wenn die **CPU** schneller wäre (z.B. Programm, das komplexe mathematische Rechnungen macht, hängt von der Stärke der CPU ab)
- **I/O-limitiert**: ein Prozess würde schneller laufen, wenn das **I/O-Subsystem** schneller wäre (z.B. Programm, das aus einer Datei liest, hängt von der Dateigröße ab)
- **Prozess**: ein **Programm in Ausführung**, gruppiert und verwaltet Ressourcen
  - **sequentieller Prozess**: besitzt nur 1 Thread
  - **multithreaded Prozess**: besitzt mehrere Threads mit je einem Befehlszähler, Stack und weitere Register
- **Thread**: definiert einen **Kontrollfluss / Ausführungsstrang** in der Abarbeitung eines Programms
  - **Thread-Kontext**: Program Counter, Stack zur Verwaltung von Unterprogrammaufrufen, Registerwerte
  - **User-Level-Threads**: Realisierung von Threads mit Threadbibliotheken innerhalb eines Prozesses, Verwaltung der Threads durch Laufzeitsystem mit eigenem Thread-Scheduler, je eine Threadtabelle für jeden Prozess, Unterbrechung eines Threads führt zur Unterbrechung des Prozesses, *OS kennt User-Level-Threads nicht!*
  - **Kernel-Level-Threads**: Realisierung und Verwaltung von Threads durch OS-Kern via Traps, Syscalls und OS-Scheduler, eine Threadtabelle für alle Threads, Unterbrechung eines Threads führt nicht zur Unterbrechung des Prozesses
  - **Hybride Threads**: OS verwaltet Kernel-Threads, n Threads im User-Space werden auf m Threads im Kernel-Space abgebildet
- **Process Control Block (PCB)**: Datenstruktur, **enthält Informationen für die Prozessverwaltung** (Register, PID, PPID, PGID...), Speicherverwaltung (Code, Data und Stack) und Dateiverwaltung (Root-Verzeichnis, File Deskriptoren der geöffneten Dateien, UID, GID) eines Prozesses
  - **Prozesstabelle**: verkettete Liste von PCBs
- **Run Queue**: Warteschlange für rechenwillige Prozesse
- **Wait Queue**: Warteschlange für wartende Prozesse
- **Zombie**: Kind-Prozess, welches **vor dem Eltern-Prozess terminiert** und noch in der Prozesstabelle vorhanden ist

- **Waisen (Orphans)**: Eltern-Prozess **terminiert vor Kind-Prozess**; Kind-Prozess wird vom Init-Prozess (PID 1) adoptiert
- **Daemon**: Prozess, der im Hintergrund läuft und bestimmte Dienste zur Verfügung stellt; kann aus Waisenprozess stammen

Prozessverwaltung

- **Scheduler**: wählt den nächsten auszuführenden Prozess aus der Run Queue
- **Dispatcher**: führt den Kontext- und Zustandswechsel um
- **Time Sharing**: eine Ressource wird von mehreren Prozessen abwechselnd benutzt
- **Space Sharing**: abhängige Prozesse werden Ressourcen zugeweisen und gemeinsam gescheduled
- **Work Stealing**: Prozesse werden aus einer lokalen Warteschlange in eine andere verschoben
- **Cache-Reuse**: wurde der Prozess bereits einmal auf der CPU ausgeführt, kann er vielleicht auf noch im Cache vorhandene Daten zurückgreifen
- **Non-Uniform Memory Access (NUMA)**: Zugriffszeit auf Arbeitsspeicher ist nicht für alle CPUs identisch, Prozesse sollten möglichst nahe an dem von ihnen benötigten Speicher ausgeführt werden
- **Big-Little-Architecture**: Prozessorarchitektur von ARM, kombiniert **kleine, schwächere, energieeffiziente Cores** mit **stärkeren Cores mit hohem Stromverbrauch**
- **Ziele**:
  - **Fairness**: alle Systeme, gebe jedem Prozess einen fairen Anteil an CPU-Nutzung
  - **Balance**: alle Systeme, nutze alle Teile des Systems (CPU, I/O) möglichst effektiv
  - **Durchsatz**: Batchsysteme, maximiere Anzahl an Aufträge pro Zeit
  - **Ausführungszeit**: Batchsysteme, minimiere Ausführungszeit
  - **CPU-Belegung**: Batchsysteme, belege CPU konstant
  - **Antwortzeit**: interaktive Systeme, minimiere Antwortzeit
  - **Proportionalität**: interaktive Systeme, berücksichtige Erwartungshaltung des Benutzers
  - **Deadlines einhalten**: Echtzeitsysteme, vermeide Datenverlust
  - **Vorhersagbarkeit**: Echtzeitsysteme, vermeide Qualitätsverlust
- **Scheduling-Klassen**:
  - **nicht-unterbrechend (non-preemptive)**: Prozesse werden so lange ausgeführt, bis sie blockieren oder freiwillig die CPU abgeben
  - **unterbrechend (preemptive)**: Prozesse können u.U. unterbrochen werden
- **Scheduling-Strategien**:
  - **First Come First Serve (FCFS)**: non-preemptive, Batchsysteme, wähle Prozesse in der Reihenfolge, in der sie angefordert werden

    - **Shortest Job First (SJF)**: non-preemptive, Batchsysteme, wähle Prozess mit kürzester Rechenzeit
    - **Shortest Remaining Time Next (SRTN)**: preemptive, Batchsysteme, unterbreche beim Eintreffen eines neuen Prozesses und wähle Prozess mit kürzester verbleibenden Rechenzeit
    - **Round Robin (RR)**: preemptive, interaktive Systeme, wechselartiges Scheduling mit Hilfe eines Zeitquantums
    - **Priority Scheduling**: preemptive, interaktive Systeme, jedem Prozess wird eine Priorität (statisch oder dynamisch) zugewiesen, anhand welcher der nächste auszuführende Prozess ausgewählt wird
    - **Earliest Deadline First (EDF)**: non-preemptive oder preemptive, Echtzeitsysteme, self-explanatory
- **Multi-Level-Scheduling**:
    - **Short-Term-Scheduler**: wählt den nächsten auszuführenden Prozess aus der Run-Queue
    - **Medium-Term-Scheduler**: entscheidet, ob ein Prozess in den Speicher eingelagert werden soll
    - **Long-Term-Scheduler**: erzeugt einen guten Prozessmix aus I/O- und rechenintensiven Prozessen

Synchronisation

- **Kausale Beziehung**: Interaktion zwischen zwei Prozessen, **Ursache-Wirkung-Prinzip**
- **Kommunikation**: Interaktion zwischen zwei Prozessen, wobei diese **gegenseitig Nachrichten austauschen**
- **Koordinierung**: Interaktion zwischen zwei Prozessen, einem **Auftraggeber** und einem **Auftragnehmer** (also Client-Server)
- **Konkurrenz**: Interaktion zwischen zwei Prozessen, wo beide **um gemeinsame Ressourcen konkurrieren**
- **Race Condition**: zwei oder mehrere Prozesse greifen (lesend oder schreibend) auf einer gemeinsamen Ressource zu, so dass das Ergebnis **von der Reihenfolge der Prozessausführung abhängt**
- **Kritischer Abschnitt**: Teil des Programmcodes, wo mehrere Prozesse auf gemeinsame exklusiv nutzbare Ressourcen zugreifen könnten $\rightarrow$ erfordert Synchronisation
- **Livelock**: so wie ein Deadlock, aber die Programme warten nicht, sondern arbeiten weiter (quasi exzessive Höflichkeit...)
- **Starvation**: ein Prozess wartet unendlich lange, dass seine Ressourcenanfrage erfüllt wird
    - **alternativ**: ein Prozess bekommt keine CPU-Zeit zugeteilt, obwohl er rechenwillig ist

Eigenschaften paralleler Systeme

- **Determiniertheit**: bei **gleichen Bedingungen** entstehen die **gleichen Ergebnisse**

- **Störungsfreiheit**: unter **Einhaltung einer festen Ausführungsreihenfolge** ändert sich das Ergebnis nicht
- **Wechselseitiger Ausschluss (Mutual Exclusion)**: zu jedem Zeitpunkt greift **maximal ein Prozess** auf einer gemeinsamen Ressource
- **Verklemmungsfreiheit (No Deadlocks)**: es existieren **keine Deadlocks**
- **Kein Verhungern (No Starvation)**: es gibt keine Prozesse, deren Ausführung unendlich lange aufgeschoben wird, obwohl sich dieser nicht in einem Deadlock befindet

Synchronisationsmechanismen

- **Unterbrechungssperre**: deaktivieren von Interrupts
- **Atomare Maschinenbefehle**: `CMPXCHG`
- **Spinlock (Busy Waiting, Aktives Warten)**: blockierter Thread wartet in Schleife und prüft ständig Zustand des Locks
- **Passives Warten**: Thread schläft und wartet auf Benachrichtigung, um wieder aufgeweckt zu werden
- **Semaphore**: abstrakter Datentyp, enthält eine ganzzahlige Kontrollvariable (entsp. Anzahl der Prozesse, die gleichzeitig den kritischen Abschnitt betreten dürfen) und zwei darauf definierte Operationen (up, down)
- **Mutex**: ähnlich einem binären Semaphor, mit dem Unterschied, dass nur der Owner des Locks die Sperre auch freigeben kann

Bedingungen für Deadlocks

- **Mutual Exclusion**: die gemeinsam benutzten Ressourcen sind exklusiv nutzbar
- **Hold-And-Wait**: Prozesse, die *bereits Ressourcen zugeteilt haben*, belegen diese, auch wenn sie noch weitere Ressourcen anfordern
- **No Preemption**: zugeteilte Ressourcen können nicht entnommen werden
- **Circular Wait**: es gibt eine zyklische Kette von zwei oder mehreren Prozessen, die jeweils auf eine Ressource warten, die vom nächsten Prozess in der Kette belegt ist

Umgang mit Deadlocks

- **Ignorieren**: ...
- **Deadlock-Detection**: OS erkennt und beseitigt Deadlocks
- **Deadlock-Prevention**: OS setzt eine der vorherigen Deadlock-Bedingungen außer Kraft
- **Deadlock-Avoidance**: OS führt keine potentiell gefährliche Ressourcenzuteilung durch

Petrinetze

- **Stelle**: Zustände, Ressourcen werden als **Stellen** (Kreise) modelliert
- **Token(s)**: stellt dar, ob Zustand erfüllt / Ressourcen vorhanden sind (schwarze Punkte)

- **Transition**: Zustandsänderung (Rechtecke)
- **Vorbereich**: Ressourcen, die notwendig sind, damit eine Transition schaltet (linke Seite der Transition, eingehend)
- **Nachbereich**: Ergebnis der Zustandsänderung (rechte Seite der Transition, ausgehend)
- **boolesch**: für alle erreichbaren Zustände und Stellen gilt, dass in jeder Stelle des Zustands stets 0 oder 1 Token sind (impl. Stellenkapazitäten und Kantengewichte 1)
- **schaltbereit**: eine Transition kann schalten, d.h. die Vor- und Nachbedingungen sind erfüllt
  - **Vorbedingung**: alle Stellen, aus denen Token genommen werden sollen, haben genügend Tokens
  - **Nachbedingung**: alle Stellen, in die Token gelegt werden sollen, haben (aktuell!) genügend Platz
- **Verklemmung eines Zustandes**: es existiert **keine** Transition, die in diesem Zustand schalten kann; wird als Blatt im Erreichbarkeitsgraphen repräsentiert
- **Verklemmungsfreiheit**: es existiert kein erreichbar verklemmter Zustand
- **Lebendigkeit**: es existiert immer einen erreichbaren Zustand, so dass jede beliebige Transition schalten kann
  - *Lebendigkeit impliziert Verklemmungsfreiheit*
- **Verhungern einer Transition**: es existiert eine unendliche Schaltsequenz, in der eine Transition schaltbereit ist, aber nie schaltet; werden u.U. als Zyklen im Erreichbarkeitsgraphen repräsentiert
- **Fairness**: keine Transition verhungert

Interprozesskommunikation (IPC)

- **schmalbandige Kanäle**: Übertragen von **wenigen Bits** an Information (z.B. Signale)
- **breitbandige Kanäle**: Übertragen **größerer Datenmengen**
- **implizite Kommunikation**: Kommunikation über **gemeinsame Betriebsmittel** (z.B. gemeinsamer Speicher); keine direkte Unterstützung via OS
- **explizite Kommunikation**: Kommunikation durch explizites **Senden und Empfangen von Nachrichten**, bestehend aus **Header** (Managementinformationen) und **Payload** (eigentliche Nachricht); dedizierte Interaktion mit OS (`send`, `recv`)
- **synchrone Kommunikation**: blockierend, Prozess A wartet nach dem Senden einer Nachricht an B auf eine Antwort (Acknowledgement) von B
- **asynchrone Kommunikation**: nicht blockierend, Prozess A wartet *nicht* auf Antwort von B
- **Message Passing**: Nachrichtenaustausch zwischen Sender und Empfänger, wobei der Empfänger die Nachricht des Senders entsprechend interpretiert (meist Funktionsaufruf, Signale oder Datenpakete)
- **Datenstrom (Stream)**: kontinuierlicher **Datenfluss**, der fortlaufend beim Eintreten eines neuen Datensatzes (Bytestorms) verarbeitet wird

- Pipe (|): **unidirektionaler Strom** zwischen zwei Kommunikationspartnern durch einen **FIFO-Puffer**, kann *anonym* oder als virtuelle Datei (*named*) realisiert werden
- Port: logische, numerische Abstraktion eines Kommunikationsendpunktes; gehört eindeutig zu einem Prozess
- **passiver Prozess**: Prozess, der sich auf einer (oder mehreren) Portnummer(n) registriert und **auf eingehende Verbindungen** (TCP) bzw. Nachrichten (UDP) **wartet**
  - **TCP**: Transmission Control Protocol
  - **UDP**: User Datagram Protocol
- **aktiver Prozess**: Prozess, der **die Interaktion mit einem passiven Prozess initiiert** und dafür eine freie Portnummer vom OS bekommt
- **Socket**: logische **Verbindung** zwischen Kommunikationspartnern für **strombasierte Kommunikation**, meist **bidirektional** als Zuordnung von Ports (und evtl. auch IP-Adressen) realisiert
- **Server**: dedizierter Prozess, der ein Dienst zur Verfügung stellt, langlebig, hat meistens mehrere Worker-Threads für jeden einzelnen Auftrag
- **Client**: initiiert Aufträge an einen von einem Server angebotenen Dienst, kurzlebig, meistens auf einem verschiedenen Rechner als Server

Speicherverwaltung

- Adressraum: Speicherabstraktion, jeder Prozess erhält seinen eigenen isolierten Adressraum
  - **physischer Adressraum**: Menge aller Speicheradressen des Hauptspeichers, fortlaufend, beginnend bei 0 (Bytes)
  - **logischer / virtueller Adressraum**: Menge des adressierbaren Speichers, Abbildung des physischen Speichers
  - **Speicherabbildung (Memory Adress Translation)**: Abbildung phys. Speicheradresse $\rightarrow$ virt. Speicheradresse
- **direkte Adressierung**: 1-zu-1 Abbildung, jede logische Speicheradresse entspricht ihrer physischen Adresse
- **Basis-Adressierung**: jeder Prozess hat für seinen eigenen logischen Adressraum eine Basisadresse b_x
  - phys. Adresse = virt. Adresse + Basisadresse
- **Segmentadressierung**: Unterteilung des Adressraums in logische Segmente (z.B. Code, Data, Stack) mit **Segmentanfangsadresse**, **Segmentlänge** und **Zugriffsrechte**
  - Global Descriptor Table (GDT): enthält für jedes Segmentregister (Index) die Segmentanfangsadresse, Segmentlänge und Flags
- **Swapping**: Aus- / Einlagerung von **Prozessdaten** vom Hauptspeicher in Hintergrundspeicher **eines gesamten Prozesses**

- **Bitmap**: Datenstruktur zur Freispeicherverwaltung (Verwaltung der freien Speicherblöcke in RAM)
  - pro Block 1 Bit (1 wenn Block belegt, 0 sonst)
- **Verkettete Liste**: Datenstruktur zur Freispeicherverwaltung
  - enthält **Belegungszustand** des Blocks (P wenn belegt, F sonst), **Startadresse** des Bereichs, **Länge** des Bereichs (in Blöcke), **Zeiger** auf den nächsten Bereich (oder NULL)

Belegungsstrategien

- **First-Fit**: durchsuche Freelist vom **Anfang** an, belege den **ersten** Freibereich mit ausreichender Größe
- **Next-Fit**: durchsuche Freelist **beginnend beim als letztes belegten Bereich**, belege den **ersten** Freibereich mit ausreichender Größe
- **Best-Fit**: durchsuche **komplette** Freelist, belege Freibereich mit **minimalem** Verschnitt
- **Worst-Fit**: durchsuche **komplette** Freelist, belege Freibereich mit **maximalem** Verschnitt
- **Buddy-Algorithmus**: rekursive Halbierung der Freelist und Belegung des kleinstmöglichen Freibereichs, rekursive Zusammenführung zusammenhängender Bereiche (Buddies)
  - siehe hier für genauere Erklärung und Beispiel

Fragmentierung

- **Interne Fragmentierung / Verschnitt / Slack Space**: es wird mehr Speicher gefüllt als tatsächlich benötigt wird (Fragmentierung *innerhalb* eines Blocks)
- **Externe Fragmentierung**: Löcher im Speicherbereich; es existiert theoretisch genug Speicherplatz, kann aber nicht verschmolzen werden (Fragmentierung *außerhalb* eines Blocks, im gesamten Speicherbereich)

Paging

- **Paging**: Speicherverwaltungsmethode, Ein- / Auslagerung von **Seiten** vom Hauptspeicher in Hintergrundspeicher
  - **virtueller Speicher** wird in **Seiten** aufgeteilt
  - **physischer Speicher** wird in **Kacheln** aufgeteilt
- **Memory Management Unit (MMU)**: rechnet virtuelle Adressen in physischen Adressen um, lädt Page Table Einträge aus Cache / RAM / TLB und sendet phys. Adressen an Cache / RAM
- **Page Table (Seitentabelle)**: Datenstruktur, speichert Mappings von virtuellen auf physischen Adressen, 1 pro Prozess
  - **Page Table Entry**: enthält Mapping v.A $\rightarrow$ p.A. und einzelne Bits (P, U/S, R, M, XD)
- **virtuelle Adresse**: besteht aus **Seitennummer(n)** und **Offset**
  - **Seitennummer(n)**: **Index / Indices** in **Page Table(s)** (ein- / mehrstufig)
  - **Offset**: **Index** innerhalb der entsprechenden **Seite**

- **physische Adresse**: besteht aus **Frame-Nummer** und **Offset**
- **Translation Lookaside Buffer (TLB)**: kleiner Cache für die Abbildung von Seiten auf Kacheln
  - **TLB Hit**: Abbildung bereits in TLB, vermeidet Speicherzugriff
  - **TLB Miss**: Abbildung nicht in TLB, extra Speicherzugriff nötig
- **Seitenfehler (Page Fault, PF)**: tritt auf, wenn Seite der zugegriffenen virtuellen Adresse nicht im Hauptspeicher vorhanden ist (m.a.W. P-Bit *nicht* gesetzt)
  - **Page Fault Handler**: speichert Programmzustand und deaktiviert Interrupts
- **Working Set**: Menge der Seiten, die ein Prozess aktuell benötigt
  - **Prepaging**: halte Working Set für rechnende Prozesse bereit
- **Seitenflattern (Thrashing)**: Prozess verursacht häufig Page Faults

Seitenersetzungsstrategien

- **On-Demand-Paging**: lade Seite sobald auf sie zugegriffen wird und sie sich nicht im RAM befindet
- **Prefetching**: lade Seiten im Voraus
- **First-In-First-Out (FIFO)**: ersetze diejenige Seite, die am längsten im Speicher gehalten wurde
- **Clock-Algorithmus**: Verwalten der Seiten in einer zirkulären Liste (wie eine Uhr); Zeiger zeigt auf älteste Seite (genaueres Beispiel in Tutorübungen)
- **Least Recently Used (LRU)**: ersetze diejenige Seite, die am längsten nicht verwendet wurde
- **Not Frequently Used (NFU)**: ersetze diejenige Seite, die am wenigsten verwendet wurde
- **Aging**: bestimme, ob Zugriffe frisch sind oder nicht
- **Working-Set-Algorithmus**: ersetze nur Seiten, die sich nicht im aktuellen Working Set befinden

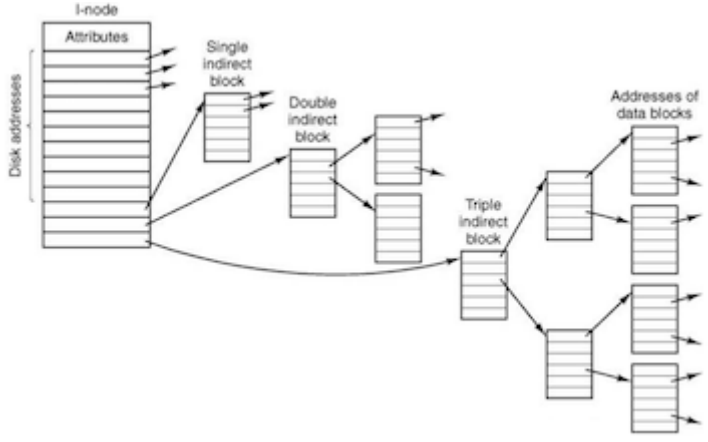
Dateisysteme

- **Festplatte**: Langzeitspeicher, lineare Sequenz von Blöcke fester Größe
- **Datenblock (Sektor)**: kleinste in einem Zugriff les- oder schreibbare Einheit einer Festplatte
- **Datei (File)**: logische Verwaltungseinheit, Abstraktion der Festplatte, speichert Daten dauerhaft
- **Dateisystem (File System)**: Datenstruktur zur Organisation und Verwaltung von Dateien
  - **NTFS**: Windows NT File System, für moderne Windows-Systeme
  - **Ext2/3/4**: Extended File System, für Linux-Kernel
- **Dateitypen**:
  - **Datei (regular file)**: Text- oder Binärdateien
  - **Verzeichnis (directory)**: Systemdatei zur Verwaltung und Strukturierung des File Systems
  - **Character Special File**: Schnittstelle für serielle I/O-Geräten (Terminals, Drucker...)
  - **Block Special File**: Schnittstelle für Massenspeicher (HDDs, SSD, USBs, CDs...)

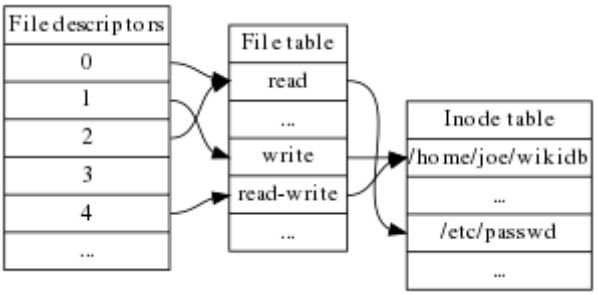
- **Sequential Access**: Prozess liest der Reihe nach Bytes einer Datei
- **Random Access**: Prozess liest Bytes einer Datei in einer beliebigen Reihenfolge
- `lseek`: Syscall, spezifiziert / modifiziert aktuelle Position in Datei
- **Single-Level Verzeichnisstrukturen**: nur ein Root-Verzeichnis für alle Dateien
- **Hierarchische Verzeichnissstukturen**: Gruppierung von Dateien in Unterverzeichnissen
- **Absoluter Pfadname**: gesamter Pfad einer Datei, ausgehend vom Root-Verzeichnis
- **Relativer Pfadname**: Pfad einer Datei ausgehend vom aktuellen Arbeitsverzeichnis
- **Current Working Directory (CWD)**: der Ordner, in dem wir uns gerade befinden
- `.`: aktuelles Arbeitsverzeichnis (CWD)
- `..`: übergeordnetes Verzeichnis des CWDs

Dateisystem-Implementierung

- Partition: **logisch abgetrennte Datenblöcke** einer Festplatte mit je einem Dateisystem
- Master Boot Record: erster Sektor der Festplatte, enthält **MBR-Programm**
  - MBR-Programm (Startprogramm, Bootloader): lädt **Boot-Sektor** der *aktiven* Partition
- Partitionstabelle: Datenstruktur zur **Verwaltung der Partitionen** (Anfangs- und Endadressen), liegt direkt nach dem MBR
- Boot-Sektor / Boot-Block: erster Block einer Partition, enthält **notwendige Informationen zum Starten eines OS**
- **Superblock**: enthält Metadaten über das File System
- **Contiguous Allocation**: Datei wird als **zusammenhängende Folge von Blöcken** verwaltet
- **Linked List Allocation**: benutzte Blöcke einer Datei werden in einer Linked List verwaltet
  - File Allocation Table (FAT): Datenstruktur zur Verwaltung der **Zeiger** der Linked List innerhalb einer gemeinsamen Tabelle für **alle** Dateien, liegt im Hauptspeicher
- Index Node (i-node): Datenstruktur zur Repräsentierung einer Datei (Dateiattribute + Adressen der belegten Blöcke + Adressen von Zeigerblöcke), wird pro geöffnete Datei im Hauptspeicher geladen
  - **indirekte, doppelt indirekte und dreifach indirekte Blöcke**: Verzweigungsgrad des Zeigerblocks



- **File Descriptor Table** (pro Prozess): enthält Dateideskriptoren (hier Indizes) eines Prozesses, Einträge zeigen auf Einträge in der Open File Table
- **(Open) File Table**: enthält Informationen über alle gerade offenen Dateien (u.a. Reference Counter und Offset-Pointer), Einträge zeigen auf Einträge in inode-Tabelle
- **Inode Table**: Tabelle für im Hauptspeicher geladene inodes



- **Verzeichniseintrag (directory entry)**: enthält **Dateinamen, Adressinformationen** und **Verweis auf Dateiattribute**
- **Hard Link**: Datei, die auf dieselbe **Inode** wie eine andere Datei zeigt
- **Symbolic Link**: Datei mit Verweis auf verlinkte Datei durch **Angabe des Pfadnamens** (ähnlich zu einem Windows-Shortcut); unterschiedliche Inode wie ursprüngliche Datei!
- **Journaling**: **Protokollierung** aller Änderungen eines Dateisystems in einem **Journal** (Speicherbereich) vor ihrer Durchführung
- **Idempotente Operation**: Operation, die bei mehrmaliger Ausführung dasselbe Ergebnis liefert
- Virtual File System (VFS): Abstraktionsschicht; stellt Schnittstelle für Anwendungen bereit, um auf unterschiedliche Dateisysteme zuzugreifen
- **Buffer Cache**: Pufferplatz für **Datenblöcke** im Hauptspeicher
- **Read-Ahead**: lade in Kürze benötigte Datenblöcke in den Buffer Cache
- **Defragmentierung**: Neugruppierung der Blöcke auf der Festplatte

Ein- und Ausgabe (I/O)

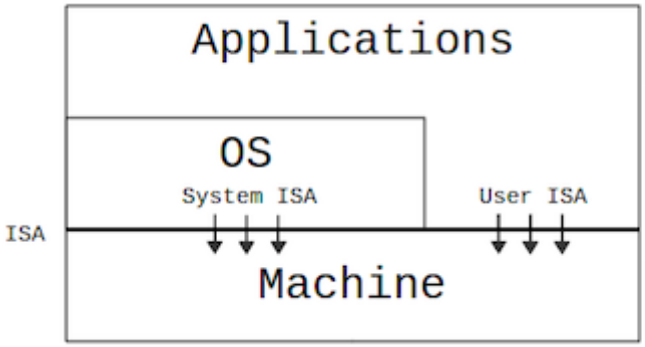
- Blockorientierte Geräte (block devices): Daten werden als **Blöcke** gesendet und empfangen, Inhalte adressierbar, Random Access möglich
- Zeichenorientierte Geräte (character devices): Daten werden als **Zeichenströme** (Byteströme) gesendet und empfangen, Inhalte *nicht* adressierbar
- Geräte-Steuereinheit (Device Controller): Hardwarekomponente, bietet **Hardwareschnittstelle** zwischen PC (OS) und I/O-Gerät an, besitzt **Status- und Steuerregister**, **Datenpuffer** und **Interruptleitung**
- Memory-Mapped-I/O (MMIO): die **Register** des Hardwarecontrollers werden in den **RAM-Adressraum** abgebildet, um Kommunikation mit üblichen **Speicheradressierung** (statt I/O-Befehlen) zu erlauben
- Port-Mapped-I/O (PMIO) oder Isolated I/O: Nutzung eines **separaten, kleineren, isolierten Adressraumes** für Datentransfere; Kommunikation geschieht via **I/O-Instruktionen** (z.B. `in`, `out`)

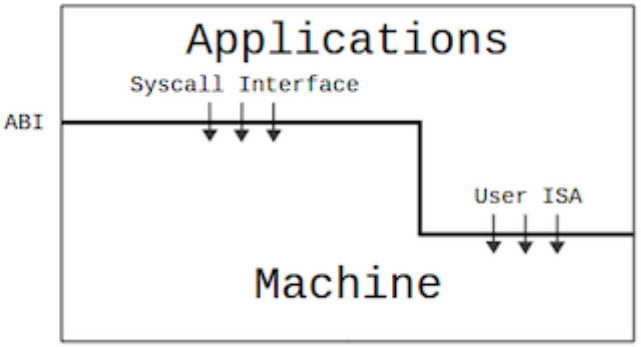
- **Programmed I/O**: **Datentransfermethode** zwischen CPU und I/O-Gerät, wo die CPU für jeden Datentransferschritt via CPU-Instruktion(en) **immer direkt beteiligt** ist (z.B. Netzwerkkarte ↔ CPU ↔ RAM)
- **Programmable Interrupt Controller (PIC)**: **Hardwarekomponente** (Controller), hilft CPU beim **Handling von IRQs** (Interrupt Requests)
  - **Interrupt Request (IRQ)**: **Hardwaresignal**, fordert **Interrupthandling** von CPU
  - **Interrupt-Handler**: **Codesegment**, das bei Auslösung eines Interrupts ausgeführt wird
  - **Interruptvektor**: Adresse, an die beim Auftreten eines Interrupts gesprungen wird (wo die Interruptroutine liegt)
- **Direct Memory Access (DMA)**: **Datentransfermethode**, wo Hardwarekomponenten **unabhängig von der CPU** mit I/O-Geräte Datentransfer durchführen können (z.B. Netzwerkkarte ↔ DMA-Controller ↔ RAM)
  - **DMA Controller (DMAC)**: Hardwarekomponente (Controller), regelt Datentransfer zwischen RAM und I/O-Gerät (von CPU initialisiert, sendet bei Abschluss Interrupt an CPU)
- **Gerätetreiber (Driver)**: **Software** zur **Steuerung von I/O-Geräten**, **Abstraktionsebene**, stellt **Schnittstelle** zwischen OS / Programme und I/O-Geräte bereit
- **Geräte-unabhängige Software**: einheitliche Schnittstelle zwischen Treibern und OS mit zusätzlicher Puffer-, Fehlerbehandlungsfunktionalität
- **Major Device Number**: Teil einer Gerätedatei-Inode; entsp. Gerätetreiber
- **Minor Device Number**: Teil einer Gerätedatei-Inode; entsp. Gerät, auf das zugegriffen werden soll
- **Spooling**: Datenaustauschmethode zwischen zwei Geräte, Aufträge werden via Spoolerdämon gepuffert und gequeued und vom I/O-Gerät entnommen
- **Hardware-Timer**: Hardwarekomponente, generiert Interrupt, wenn Timer abgelaufen ist
- **Line Discipline**: Bestandteil der Terminaltreibers; wie wird die Eingabe verarbeitet?
  - **Cooked Mode**: zeilenorientiert, Eingabe wird vorverarbeitet (preprocessed), bevor es gesendet wird
  - **Raw Mode**: zeichenorientiert, Eingabe wird ohne Vorverarbeitung weitergeschickt
- **Pseudoterminal**: Art der IPC, ermöglich asynchronen, bidirektionalen Datenaustausch

Virtualisierung

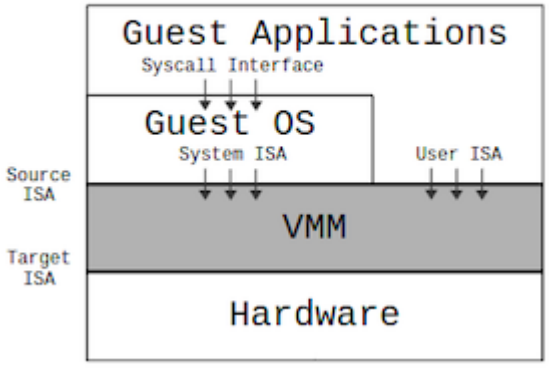
- **Abstraktionsschicht**: die Realisierung der Elemente unter dieser Schicht wird für einem Anwender "verborgen"
- **Schnittstellen (Interfaces)**: trennen Abstraktionsschichten voneinander, definieren **Softwaresicht auf Hardware**
- **Instruction Set Architecture (ISA)**: Interface zwischen OS und Hardware, definiert u.a. ASM-Instruktionen
  - **User-ISA**: verfügbarer Instruktionssatz für Benutzeranwendungen

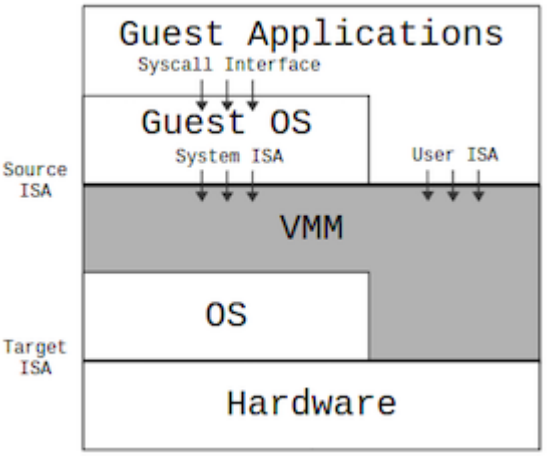
    - **System-ISA**: verfügbarer Instruktionssatz für OS
- **Application Binary Interface (ABI)**: Interface zwischen Anwendungen und OS, definiert u.a. wie auf Datenstrukturen auf Maschinencodeebene zugegriffen werden (z.B. Calling Convention...)
    - **Syscall Interface**: Interface, abstrahiert Implementierung der Syscalls durch bereitgestellte Funktionen





- **Laufzeitumgebung (Runtime Environment)**: alle Ressourcen, die ein Programm benötigt, um zu laufen
    - **virtuelle Laufzeitumgebung**: Abbilden von **virtuellen** auf **physischen** Ressourcen (m.a.W. Ressourcen, wie z.B. CPU, Speicher, I/O-Geräte, werden *virtuell* "emuliert", das virtualisierte System innerhalb einer VM betrachtet diese als echte *physische* Ressourcen)
- **Virtuelle Maschine**: virtualisiertes System
- **Virtual Machine Monitor / Hypervisor**: **Abstraktionsschicht** zwischen **Hardware** und **Guest OS / Guest Applications** (virtualisierte Software), bieten **virtuelle Laufzeitumgebung** (u.a. virtuelle ISA), virtualisiert **gesamte** physische Maschine
    - **Type 1 Hypervisor**: wird direkt auf *Hardware* ausgeführt
    - **Type 2 Hypervisor**: wird auf einem *Host OS* ausgeführt, nutzt Gerätetreiber des Host OS





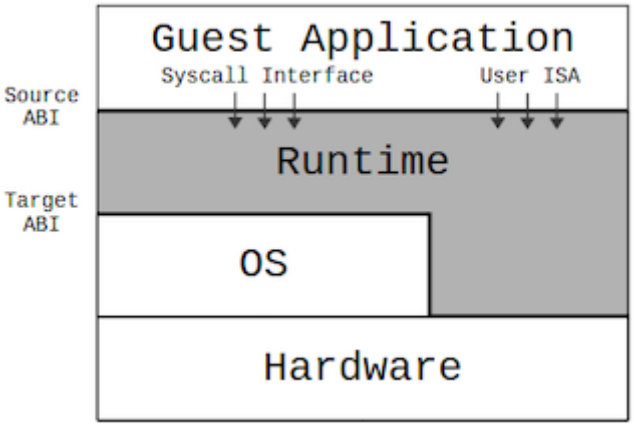
Komponentenweise Virtualisierung

- **CPU-Virtualisierung**:
    - **Paravirtualization**: *modifiziertes* Gast-OS, Syscalls des Gast-OS werden durch Hypercall-Interface via VMM virtualisiert

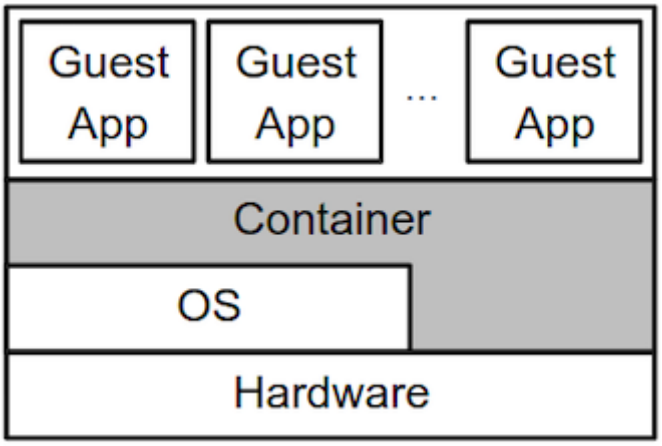
    - **Binary Translation**: unmodifiziertes Gast-OS, Hypervisor interpretiert Binärcode des Gast-OS
    - **Hardware-Assisted Virtualization**: unmodifiziertes Gast-OS, hardwareunterstützt durch Hardware Virtualization Extensions
- **Instruktionsklassen**:
    - **Privileged Instructions**: Instruktionen, die **nur im System Mode** ausgeführt werden dürfen
    - **Sensitive Instructions**: Instruktionen, die die **Systemkonfiguration ändern** (Control-Sensitive Instructions) oder Instruktionen, die **sich anders verhalten, als wenn sie im User-Mode** ausgeführt würden (Behavior-Sensitive Instructions)
    - **Innocuous Instructions**: alle anderen Instruktionen
- **Speicher-Virtualisierung**:
    - **Host-Physische Adressen (HPA)**: gesamter echter physischer Speicher im System
    - **Gast-Physische Adressen (GPA)**: der Speicher, der ein Gast OS sieht
    - **Gast-Virtueller Speicher / Gast-Virtuelle Adressen (GVA)**: wie üblicher virtueller Speicher (pro Prozess), nur innerhalb eines Gast OS
    - **Shadow Page Tables (SPT)**: erste Stufe der Adressabbildung, bilden **gast-virtuelle Adressen** auf **host-physische Adressen** ab, pro Prozess, vom Hypervisor verwaltet
    - **Second Level Address Translation (SLAT)**: zweite Stufe der Adressabbildung, bildet **gast-physische Adressen** auf **host-physische Adressen** ab
    - **Statische Speicherallokation**: Speichergröße des VMs bleibt konstant
    - **Dynamische Speicherallokation**: Speicher wird dynamisch reduziert / allokiert
    - **Ballooning (Balloon Driver)**: Softwarekomponente innerhalb eines Gast OS', teilt dem Hypervisor mit, ob mehr / weniger Speicher benötigt wird
- **I/O-Virtualisierung**:
    - **Full Virtualization / Emulation**: Hypervisor emuliert I/O-Geräte in Software
    - **Paravirtualization**: Virtualisierung durch **Split-Driver-Architektur**
        - **Backend-Treiber**: dedizierte, privilegierte VM greift direkt auf I/O-Geräte zu
        - **Frontend-Treiber**: Treiber innerhalb einer Gast-OS, kommuniziert mit Backend-Treiber
    - **Device Domains**: eine VM dient als Schnittstelle zu allen I/O-Geräten
    - **Direct I/O**: hardwareunterstütztes Multiplexing von I/O-Geräten mit dedizierter I/O-MMU
    - **Single Root I/O Virtualization (SR-IOV)**: I/O-Gerät unterstützt direkt Virtualisierung

Arten von Virtualisierung

- **Process Virtual Machine**: Laufzeitumgebung eines **einzelnen Prozesses** wird virtualisiert $\rightarrow$ Plattformunabhängigkeit



- **High-Level-Language VMs (HLL VMs)**: definieren eine plattformunabhängige **virtuelle ISA**
- **System Virtual Machine**: Laufzeitumgebung eines **gesamten Betriebssystems** (ganzes System) wird virtualisiert
- **OS-Level-Virtualization, Container**: Laufzeitumgebung **mehrerer Prozesse** wird virtualisiert, OS Kern wird *nicht* virtualisiert



- **Linux Namespaces**: Abstraktion / **Einschränkung der Sicht auf globalen Ressourcen** in einem Linux-System (UTS, IPC, Network, PID, User, Mount)
- **Linux Control Groups (cgroups)**: unterteilen Prozesse in **Prozessgruppen** und verteilen **bestimmte Systemressourcen für jede Prozessgruppe**
- **Secure Computing Mode (seccomp)**: Einschränkung der verfügbaren Syscalls für ein Prozess
  - **Strict**: nur `read()`, `write()`, `exit()` und `sigreturn()` verfügbar, sonst wird SIGKILL getriggert
  - **Filter**: steuert, welche Syscalls erlaubt sind
  - **Disabled**: alle Syscalls erlaubt

Summary by Flavius Schmidt, ge83pux, 2024.
https://home.in.tum.de/~scfl/
Images from "Modern Operating Systems" by Tanenbaum and Bos.