

Betriebssysteme - Das ultimative Cheat Sheet

Basierend auf Kurs 01670 - FernUniversität in Hagen

1 KE 1: Einführung & Grundlagen

1.1 Was ist ein Betriebssystem (BS)?

Definition: Menge von Programmen, die es ermöglichen, den Rechner zu betreiben und Anwendungsprogramme auf ihm auszuführen.

Zwei Hauptsichten:

- **Abstrakte/Virtuelle Maschine:** Verbirgt Hardware-Komplexität, bietet einfache Schnittstelle (API)
- **Ressourcen-Manager:** Verwaltet & verteilt Ressourcen (CPU, Speicher, Geräte) fair und effizient

1.2 Aufgaben eines Betriebssystems

Klassische Aufgaben:

- **Gerätesteuerung:** Verbergen der Hardware-Besonderheiten, Anbieten von Diensten
- **Schutz:** Speicherschutz, Zugriffsschutz zwischen Benutzern
- **Fehlerbehandlung:** Division durch 0, illegale Adressen, Hardware-Defekte
- **Mehrprogrammbetrieb:** Parallele Ausführung mehrerer Programme
- **Prozess-Synchronisation/-Kommunikation:** Nachrichtenaustausch, Synchronisation
- **Ressourcenverwaltung:** CPU, E/A-Geräte, Hauptspeicher, Kommunikationsverbindungen
- **Kommandosprache:** Textuelle/grafische Schnittstelle zum System
- **Administration:** Datensicherung, Systemkonfiguration, Leistungsüberwachung

1.3 Systemarchitektur & Ebenenmodell

Ebenenmodell (von unten nach oben):

1. **Digitale Logikebene:** Gatter, Boole'sche Funktionen
2. **Mikroprogramm-Ebene:** Mikrobefehle, Mikroprogramme
3. **Konventionelle Maschinenebene:** Maschinenbefehle des Prozessors
4. **Betriebssystem-Ebene:** Systemaufrufe erweitern Maschinenebene
5. **Assembler-Sprachen:** Lesbare Namen für Maschinenbefehle
6. **Höhere Programmiersprachen:** Hardware-unabhängig

Betriebssystem-Komponenten:

- **Kern (Kernel):** Programme, die immer im Hauptspeicher sind
- **Standard-Bibliotheken:** Häufig benötigte Funktionen
- **Dienstprogramme (Utilities):** Administration, Textverarbeitung

1.4 Hardware-Grundlagen

Unterbrechungen (Interrupts):

- **Hardware-Interrupts:** Asynchrone Signale von Geräten (E/A-Abschluss)
- **Software-Interrupts (Traps):** Synchron durch Programmfehler oder Systemaufrufe
- **Ablauf:** Signal → CPU unterbricht Programm → Unterbrechungsroutine → Programmfortsetzung
- **Unterbrechungsvektor:** Tabelle mit Adressen der Unterbrechungsroutinen

Speicherschutz:

- **Grenzregister:** Trennt Benutzer- und Betriebssystem-Bereich
- **Zweck:** Schutz des BS vor fehlerhaften/bösartigen Programmen

System- und Benutzermodus:

- **Benutzermodus (User Mode):** Eingeschränkte Befehle, Speicherschutz aktiv
- **Systemmodus (Kernel Mode):** Alle Befehle erlaubt, Speicherschutz deaktiviert
- **Supervisor Call (SVC):** Kontrollierter Übergang User→Kernel für Systemaufrufe

1.5 Mehrprogrammbetrieb

Motivation:

- **Auslastungsverbesserung:** CPU arbeitet während E/A-Wartezeiten anderer Prozesse
- **Parallelität:** Mehrere Benutzer/Programme gleichzeitig
- **Virtueller Prozessor:** Jeder Prozess hat Eindruck einer eigenen CPU

nen CPU

Zeitscheiben:

- **Zeitgeber (Timer):** Hardware-Komponente für regelmäßige Unterbrechungen
- **Zeitscheibenablauf:** Unterbrechung nach Ablauf der zugeteilten Zeit
- **Prozesswechsel:** Umschaltung zwischen Prozessen

1.6 Betriebsarten

Interaktiver Betrieb (Dialog):

- Sofortige Programmausführung, direkte Benutzer-Programmkommunikation
- Time-Sharing: Mehrbenutzer-Dialogsysteme
- Optimierungsziel: Kurze Antwortzeiten

Stapelbetrieb (Batch):

- Jobs werden in Warteschlange eingereiht, keine direkte Kommunikation
- Optimierungsziel: Maximale Ressourcenauslastung
- Höhere Durchsatzraten, längere Wartezeiten akzeptabel

Hintergrundauführung:

- Programme laufen parallel zu interaktiven Prozessen
- Keine direkte Benutzerinteraktion während der Ausführung

Realzeitbetrieb:

- Harte Zeitgrenzen müssen eingehalten werden
- Zeitkritische Prozesse haben höchste Priorität
- Erfordert speziell konstruierten Betriebssystemkern

1.7 Systemstart (Bootstrap)

Ladevorgang:

- **Firmware/BIOS:** In ROM/EPROM gespeichert
- **Urlader (Bootstrap Loader):** Lädt Betriebssystem von Festplatte
- **Master Boot Record (MBR):** Enthält Startinformationen
- **Boot Manager:** Auswahl zwischen mehreren Betriebssystemen

1.8 Historisches Beispiel: CP/M

Komponenten:

- **BIOS:** Hardware-abhängige Gerätetreiber
- **BDOS:** Hardware-unabhängige Dateiverwaltung
- **CCP:** Kommandointerpreter (Shell)
- **TPA:** Transient Program Area (Benutzerbereich)

2 KE 2: Prozesse & Scheduling

2.1 Programm vs. Prozess

- **Programm:** Statische Formulierung eines Algorithmus (Programmtext)
- **Prozess:** Ablaufendes Programm inklusive aktueller Stand des Befehlszählers, Registerinhalte und Hauptspeicherbereich mit Variablenbelegungen

2.2 Prozessmerkmale

Prozesszustände:

- **erzeugt:** Datenstrukturen werden erstellt, Adressraum zugewiesen
- **bereit:** Rechenbereit, wartet auf Prozessorzuteilung
- **rechnend:** Prozessor ist zugeteilt, führt Anweisungen aus
- **blockiert:** Wartet auf Ereignis (z.B. E/A)
- **beendet:** Programmausführung ist beendet

Speicherbereich eines Prozesses:

- **Programmsegment:** Ausführbarer Code (ändert sich nicht)
- **Stacksegment:** Programmstack mit Aktivierungsblöcken
- **Datensegment:** Daten des Programms

Prozesskontrollblock (PCB):

- **Prozessidentifikation:** Eindeutige Prozess-ID
- **Prozessorstatus:** Programmzähler, alle Register
- **Prozesskontrollinformationen:** Zustand, Priorität, Speicherbereich, geöffnete Dateien, Buchhaltung, Besitzer

2.3 Zustandsübergänge & Prozesswechsel

Wichtige Übergänge:

- erzeugt → bereit (2): Ressourcen zugeteilt
- bereit → rechnend (3): Prozessor zugeteilt
- rechnend → blockiert (5): Warten auf Ereignis
- blockiert → bereit (6): Ereignis eingetreten
- rechnend → bereit (4): Zeitscheibe abgelaufen oder freiwillige Abgabe
- rechnend → beendet (7): Prozess terminiert

Dispatcher: Führt Prozesswechsel durch

- Sichert Prozessorzustand (Register, Programmzähler)
- Übergibt PCB an Scheduler
- Stellt Zustand des neuen Prozesses wieder her

Präemptiv vs. Nicht-präemptiv:

- **Nicht-präemptiv:** Nur Prozess selbst gibt Prozessor ab
- **Präemptiv:** Betriebssystem kann Prozessor entziehen (Timer-Interrupt)

2.4 Scheduling-Strategien

Qualitätsmaßstäbe:

- Prozessorauslastung, Durchlaufzeit, Durchsatz, Antwortzeit, Fairness
- CPU burst: Zeit, die Prozess den Prozessor am Stück behalten will

Nicht-präemptive Verfahren:

Verfahren	Vorteile	Nachteile
FCFS	Einfach, fair, geringer Verwaltungsaufwand	Kurze Prozesse warten lange (Convoy-Effekt)
SJF	Minimale mittlere Wartezeit	Verhungern langer Prozesse, Bedienzeit unbekannt
Priority	Wichtige Aufgaben schnell	Verhungern bei statischen Prioritäten

Präemptive Verfahren:

Verfahren	Vorteile	Nachteile
Round Robin	Sehr fair, gut für interaktive Systeme	Overhead durch Kontextwechsel
SRTF	Optimal für bekannte Zeiten	Verhungern, viele Unterbrechungen
Priority (präemptiv)	Flexible Prioritäten	Komplexer

Quantum-Wahl bei Round Robin:

- Zu klein: Hoher Verwaltungsaufwand
- Zu groß: Schlechte Antwortzeiten
- Optimal: Etwas größer als typische Interaktionszeit

2.5 Kombinierte Strategien

Feedback Scheduling:

- Berücksichtigt Vergangenheit des Prozesses
- Aging: Priorität steigt mit Wartezeit → verhindert Verhungern
- Rechenzeitabhängig: Neue Prozesse hohe Priorität + kleine Zeitscheibe
- Bei Quantumverbrauch: Niedrigere Priorität + größere Zeitscheibe

Multiple Queues:

- Verschiedene Klassen (System-, Dialog-, Hintergrundprozesse)
- Jede Klasse eigene Warteschlange + Scheduler
- Prozessorzeit-Verteilung zwischen Klassen (z.B. 60%/30%/10%)

Linux-Scheduler:

- **O(1)-Scheduler:** Konstante Laufzeit, Prioritäten 0-139, aktive/abgelaufene Gruppen
- **CFS (Completely Fair):** Virtuelle Zeit pro Prozess, perfekte Fairness angestrebt

2.6 Threads (Leichtgewichtige Prozesse)

Konzept:

- Mehrere Ausführungspfade pro Prozess
- **Gemeinsam:** Code, Daten, geöffnete Dateien
- **Separat:** Programmzähler, Register, Stack

Realisierungen:

Benutzer-Threads	Kernel-Threads
+ Einfache Realisierung	+ Echte Parallelität auf Multiprozessoren
+ Schnelles Umschalten	+ Ein blockierender Thread stoppt nicht alle
- Blockierung stoppt alle	- Aufwändige Realisierung
- Keine echte Parallelität	- Langsameres Umschalten

Anwendungsgebiete:

- Mehrprozessor-Maschinen (Parallelisierung)
- Gerätetreiber (parallele Anfragen-Bearbeitung)
- Verteilte Systeme (Server mit mehreren Clients)

3 KE 3: Hauptspeicherverwaltung

3.1 Grundlagen

Logische vs. Physische Adressen:

- **Physische Adresse:** Reale Adresse einer Speicherzelle im RAM
- **Logische/Virtuelle Adresse:** Vom Programm erzeugte Adresse, hardware-unabhängig
- **MMU (Memory Management Unit):** Hardware zwischen CPU und Hauptspeicher für Adressumsetzung

Übersetzer, Binder und Lader:

- **Quellprogramm-Modul → Compiler → Bindemodul → Binder → Lademodul → Lader → Geladenes Programm**
- **Absolute Adressen:** Binder kennt logischen Adressraum bereits
- **Relative Adressen:** Lader addiert Startadresse zu relativen Adressen
- **Basisregister-Adressierung:** (Registernummer, Offset) für verschiebbare Programme
- **Dynamisches Binden:** Bindung zur Laufzeit, Module werden bei Bedarf nachgeladen

3.2 Einfache zusammenhängende Speicherzuweisung

Konzept: Ein Prozess = ein zusammenhängender Speicherbereich

- Betriebssystemkern ab Adresse 0
- Anwenderprogramm ab Adresse a
- **Speicherschutz:** Grenzregister verhindert Zugriff auf Adressen < a
- **Swapping:** Kompletter Prozess wird auf Sekundärspeicher ausgelagert

3.3 Mehrfache zusammenhängende Speicherzuweisung

MFT (Multiprogramming with Fixed Tasks):

- Feste Segmentgrößen beim Systemstart
- **Interne Fragmentierung:** Zugewiesener Speicher > Bedarf
- **Best-available-fit:** Kleinstes ausreichendes Segment wählen
- **Best-fit-only:** Nur Segmente verwenden, die nicht wesentlich größer sind

MVT (Multiprogramming with Variable Tasks):

- Variable Segmentgrößen je nach Bedarf
- **Externe Fragmentierung:** Viele kleine, nicht nutzbare Lücken
- **Lückenmanagement** erforderlich

Speicherplatzvergabestrategien:

- **First Fit:** Erste ausreichend große Lücke
- **Next Fit:** Wie First Fit, aber ab letzter Zuteilung suchen
- **Best Fit:** Kleinste ausreichend große Lücke (erzeugt kleine Restlücken)
- **Worst Fit:** Größte Lücke wählen (große Restlücke bleibt nutzbar)
- **Buddy-Verfahren:** Nur 2er-Potenzen, interne Fragmentierung, aber effiziente Verwaltung

Kompaktifizierung (Garbage Collection):

- Verschieben belegter Bereiche zur Zusammenlegung freier Bereiche
- Hoher Aufwand, daher selten verwendet

3.4 Nichtzusammenhängende Speicherzuweisung

Segmentierung:

- **Trennung von Programm und Daten** in separate Segmente
- Logische Adresse = (Segmentnummer, Relativadresse)
- **Basisregister pro Segment** für Adressumsetzung
- **Wiedereintrittsfähige Programme (Reentrant Code):** Ein Programmsegment für mehrere Prozesse
- **Shared Libraries:** Gemeinsam benutzte Bindemodule

3.5 Paging (Seitenorientierte Speicherverwaltung)

Grundkonzept:

- **Seiten (Pages):** Logischer Adressraum in gleichgroße Blöcke (typisch 4 KB)
- **Seitenrahmen (Page Frames):** Physischer Speicher in gleichgroße Blöcke
- **Seitentabelle:** Bildet Seiten auf Seitenrahmen ab
- Logische Adresse = (Seitennummer s , Offset d) mit $s = v \text{ div } p$, $d = v \text{ mod } p$

Adressumsetzung:

- Physische Adresse $r = ST[s] \times p + d$
- **TLB (Translation Lookaside Buffer):** Schneller Assoziativspeicher für häufig benutzte Seitentabellen-Einträge
- **TLB Hit:** Eintrag im TLB gefunden (schnell)
- **TLB Miss:** Zugriff auf Seitentabelle im Hauptspeicher (langsam)

Zusätzliche Funktionen:

- **Individueller Speicherschutz:** Protection-Bits pro Seite (read/write/execute)
- **Shared Memory:** Seiten in mehreren Adressräumen einblenden
- **Memory-Mapped Files:** Dateiseiten in virtuellen Speicher einblenden

Implementierung:

- **Mehrstufige Seitentabellen:** Bei großen Adressräumen (z.B. 32 Bit)
- **Seitenrahmentabelle:** Globale Tabelle über Zustand aller Frames

3.6 Virtueller Hauptspeicher

Demand Paging:

- **Konzept:** Seiten werden erst bei Bedarf geladen, nicht alle Seiten eines Prozesses sind im RAM
- **Seitenfehler (Page Fault):** Zugriff auf nicht geladene Seite → Trap → Einlagerung
- **Present-Bit:** Zeigt an, ob Seite im Hauptspeicher liegt
- **Dirty Bit:** Zeigt an, ob Seite seit Einlagerung verändert wurde
- **Swap-Bereich:** Reservierter Festplattenbereich für ausgelagerte Seiten

Kosten eines Seitenfehlers:

- Festplattenzugriff $\approx 10^5$ CPU-Instruktionen
- Seitenfehler dürfen nur sehr selten auftreten ($< 10^{-4}$)

3.7 Seitenauslagerungsstrategien

Optimale Strategie (theoretisch):

- Lagere Seite aus, die am weitesten in der Zukunft benutzt wird
- Nicht implementierbar (Hellssehen unmöglich)
- Dient als Vergleichsmaßstab

LRU (Least Recently Used):

- Lagere am längsten unbenutzte Seite aus
- Beste praktische Approximation der optimalen Strategie
- **Problem:** Aufwändige exakte Implementierung

Approximation von LRU:

- **Zugriffsbits (Referenced Bits):** Hardware setzt Bit bei jedem Zugriff
- **Schieberegister:** Sammelt Zugriffsmuster über mehrere Perioden
- **Second Chance:** FIFO + Zugriffsbits, gibt Seiten zweite Chance
- **Clock-Algorithmus:** Zirkuläre Liste + Uhrzeiger, effiziente Implementierung von Second Chance

Einfachere Strategien:

- **FIFO:** Längste Zeit im Speicher → Problem: kann auch aktive Seiten auslagern
- **Nachteile:** Belady-Anomalie möglich (mehr Frames → mehr Seitenfehler)

3.8 Zuweisungsstrategien

Lokalitäten:

- **Lokalitätsprinzip:** Programme greifen zeitlich/räumlich konzentriert auf Speicher zu
- **Lokalität:** Menge von Seiten, die über kurzen Zeitraum häufig benutzt werden
- Beispiele: Schleifen, sequentielle Datenverarbeitung, zusammenhängende Datenstrukturen

Arbeitsmengenstrategie (Working Set):

- **Arbeitsmenge:** Seiten, die in letzter Zeit benutzt wurden
- **Fenster:** Betrachteter Zeitraum (≈ 10.000 Zugriffe)
- **Strategie:** Jedem Prozess so viele Frames zuteilen, wie Arbeits-

menge groß ist

- Bei Seitenfehler: Weiteren Frame zuteilen
- Seite fällt aus Arbeitsmenge: Frame entziehen

Seitenfehler-Frequenz-Algorithmus (PFF):

- Misst Zeit t zwischen Seitenfehlern eines Prozesses
- Wenn $t \leq T$: Prozess braucht mehr Frames
- Wenn $t > T$: Alle Seiten mit R-Bit = 0 auslagern

3.9 Scheduling bei virtuellem Speicher

Thrashing (Seitenflattern):

- System produziert nur noch Seitenfehler, kaum produktive Arbeit
- **Ursache:** Summe aller Arbeitsmengen $>$ physischer Speicher
- **Lösung:** Weniger Prozesse parallel ausführen

Scheduling-Aspekte:

- Nur so viele Prozesse im Zustand "bereit", dass ihre Arbeitsmengen in den RAM passen
- Bei Speichermangel: Prozesse komplett stilllegen (alle Seiten auslagern)
- Vor Reaktivierung: Arbeitsmenge wieder einlagern (Prepaging)

Benutzergesteuerte Speicherverwaltung:

- Programme können Hinweise auf Zugriffsmuster geben
- Beispiel: Sequentielle Verarbeitung → Seiten schnell wieder auslagern
- Wichtige Datenstrukturen → bevorzugt im Speicher halten

4 KE 4: Synchronisation & Deadlocks

4.1 Problem: Race Conditions

In **kritischen Abschnitten**, wo mehrere Prozesse auf gemeinsame Ressourcen zugreifen. Benötigt **wechselseitigen Ausschluss**.

4.2 Mechanismen

Semaphore (Dijkstra): Zählervariable mit atomaren Operationen:

- **down() (P, wait):** Blockiert, wenn Zähler ≤ 0
- **up() (V, signal):** Weckt ggf. wartenden Prozess

Monitore: Hochsprachenkonstrukt, das Daten und Prozeduren kapselt und Mutex automatisch sicherstellt.

4.3 Deadlocks

Definition: Zyklisches Warten von Prozessen auf Ressourcen.

4 notwendige Bedingungen:

1. Wechselseitiger Ausschluss
2. Hold and Wait
3. Keine Unterbrechung
4. Zyklisches Warten

Umgang: Verhinderung, Vermeidung, Erkennung & Behebung, oder Ignorieren ("Vogel-Strauß-Algorithmus").

5 KE 5: Geräte- & Dateiverwaltung

5.1 Geräteverwaltung

- **Controller:** Hardware, die Geräte steuert
- **Gerätetreiber:** Software, die mit dem Controller kommuniziert
- **DMA (Direct Memory Access):** Ermöglicht Datentransfer zwischen Gerät und Speicher ohne CPU-Beteiligung

5.2 Dateisystem

- **Datei:** Abstraktion für permanenten Speicher
- **Verzeichnis:** Hierarchische Struktur zur Organisation
- **Dateizuordnung** (Wie werden Blöcke gespeichert?):
 - **FAT (File Allocation Table):** Verkettete Liste der Blöcke in zentraler Tabelle
 - **i-node (Index-Node):** Datenstruktur pro Datei mit Metadaten und Zeigern (direkt/indirekt) auf Datenblöcke (Standard in UNIX)

6 KE 6: Sicherheit

6.1 Ziele & Begriffe

- **Ziele:** Vertraulichkeit, Integrität, Verfügbarkeit
- **Subjekt:** Aktive Entität (Prozess, Benutzer)
- **Objekt:** Passive Entität (Datei, Gerät)

6.2 Kernmechanismen

Authentisierung: Wer bist du? (Passwort)

Autorisierung (Zugriffskontrolle): Was darfst du?

- **ACL (Access Control List):** Pro Objekt eine Liste mit Rechten für Subjekte
- **Capability:** Pro Subjekt eine Liste mit Tickets für Objekte

DAC (Discretionary): Besitzer legt Rechte fest

MAC (Mandatory): System erzwingt globale Regeln

- **Bell-LaPadula (Vertraulichkeit):** No Read Up, No Write Down"
- **Biba (Integrität):** No Read Down, No Write Up"

7 KE 7: Kommandosprachen

7.1 Kommandointerpreter (Shell)

- Schnittstelle zwischen Benutzer und BS
- Startet Prozesse (typisch: `fork()` & `exec()`)
- Verwaltet die Prozessumgebung (Variablen, offene Dateien)
- **I/O-Umlenkung:**
 - `>` leitet Ausgabe in Datei um
 - `<` liest Eingabe aus Datei
 - `|` (Pipe) leitet Ausgabe eines Prozesses als Eingabe an den nächsten weiter
- **Shell-Skripte:** Automatisierung von Kommandoabfolgen mit Variablen und Kontrollstrukturen