Betriebssysteme Test 1

Fabian Stebler & Jan Fässler

2. Semester (FS 2012)

If you use this documentation for a exam, you should offer a beer to the authors!

Inhaltsverzeichnis

1 Einleitung

1.1 Anforderungen an ein Betriebssystem

- Start des Systems
- Laden und Unterbrechen von Programmen
- Methoden fur die Interprozesskommunikation
- Verwaltung der Prozessorzeit
- Verwaltung des primaren und sekundaren Speicherplatzes fur das Betriebssystem und seine Anwendungen
- Verwaltung der angeschlossenen Gerate, Netzwerke etc.
- Schutz des Systemkerns und seiner Ressourcen vor nicht intendierter Benutzung
- Benutzerfuhrung, Rollen & Rechte
- Einheitliche Schnittstelle fur die System- & Anwendungs- programmierung
- Ereignisprotokollierung

1.2 Definition

Ein Betriebssystem ist die Software die die Verwendung eines Computers ermöglicht. Es verwaltet Betriebsmittel wie den Speicher, die I/O-Geräte, usw.

1.3 Bestandteile

Betriebssysteme bestehen in der Regel aus einem Betriebssystemkern (englisch: Kernel), der die Hardware des Computers verwaltet, sowie grundlegenden Programmen, die dem Start des Betriebssystems und dessen Konfiguration dienen.
Zu den Komponenten zählen:

- Boot-Loader
- Gerätetreiber
- Systemdienste
- Programmbibliotheken
- Dienstprogramme
- Anwendungen

1.4 Varianten von Betriebssystemen

- Einbenutzer- und Mehrbenutzersysteme
- Einzelprogramm- und Mehrprogrammsysteme
- Stapelverarbeitungs- und Dialogsysteme

Betriebssysteme finden sich in fast allen Computern: als Echtzeitbetriebssysteme auf Prozessrechnern, auf normalen PCs und als Mehrprozessorsysteme auf Hosts und Grossrechnern.

1.5 Geschichte

Mechanische Rechenmaschienen wurden mit der Zeit mit Lochstreifen versehen und somit konnte von einer Art Betriebssystem gesprochen werden. Später wurden die mechanischen Teile durch die Röhrentechnologie und anschliessend durch Transistoren ersetzt (ca.1947).

- 1955 Erfindung Mikroprogrammierung
- 1964 Erstes modellreihenübergreifendes BS
- 1969 Beginn Arbeit an UNIX
- 1972-1974 Umschreiben UNIX in C (portabilität)
- 1980-1990 Popularitätssteigerung bei Heimcomputern
- 1981 Entwicklung erste graphische Oberfläche. Apple kauft sich mit Aktien ein, kreirt MAC und MAC OS. Verliert aber aufgrund der Experimentierfreudigkeit Marktanteile an Windows.
- 1991 Linus Torvalds beginnt mit der Entwicklung des LINUX-Kernels. (Start Open Source Bewegung)
- Microsoft entwickelte MS-DOS weiter und liefert MS-Windows 95 Mitte der 90er Jahre aus. (Tabellenkalkulation)
- Im PC-Desktop-Bereich tobte ein eigentlicher Glaubenskrieg zwischen Microsoft und Apple.
- IBM und andere zogen sicher immer mehr in den Midrange/Mainframe-Markt zurück.

1.6 Lessons learned

- Die Grundkonzepte haben sich stark angenähert.
- Kompatibilität wird (oft zähneknirschend) bereitgestellt.
- Entscheide für/gegen ein Betriebssystem (bes. im Privat- bereich) haben teilweise weltanschauliche Hintergründe.
- Der Quellcode ist kein Geheimnis und kein Marktvorteil mehr.
- Partizipative Entwicklung durch Communities hat ein grosses Markt- und Sparpotential.
- Die Positionen scheinen bezogen, der Markt wächst immer noch stark genug, um den etablierten Anbietern Wachstum zu ermöglichen.
- Die Wertschöpfung hat sich verlagert:
  - Hardware zu Betriebssystem
  - GUI zu Applikationen
  - Daten zu Business Intelligence

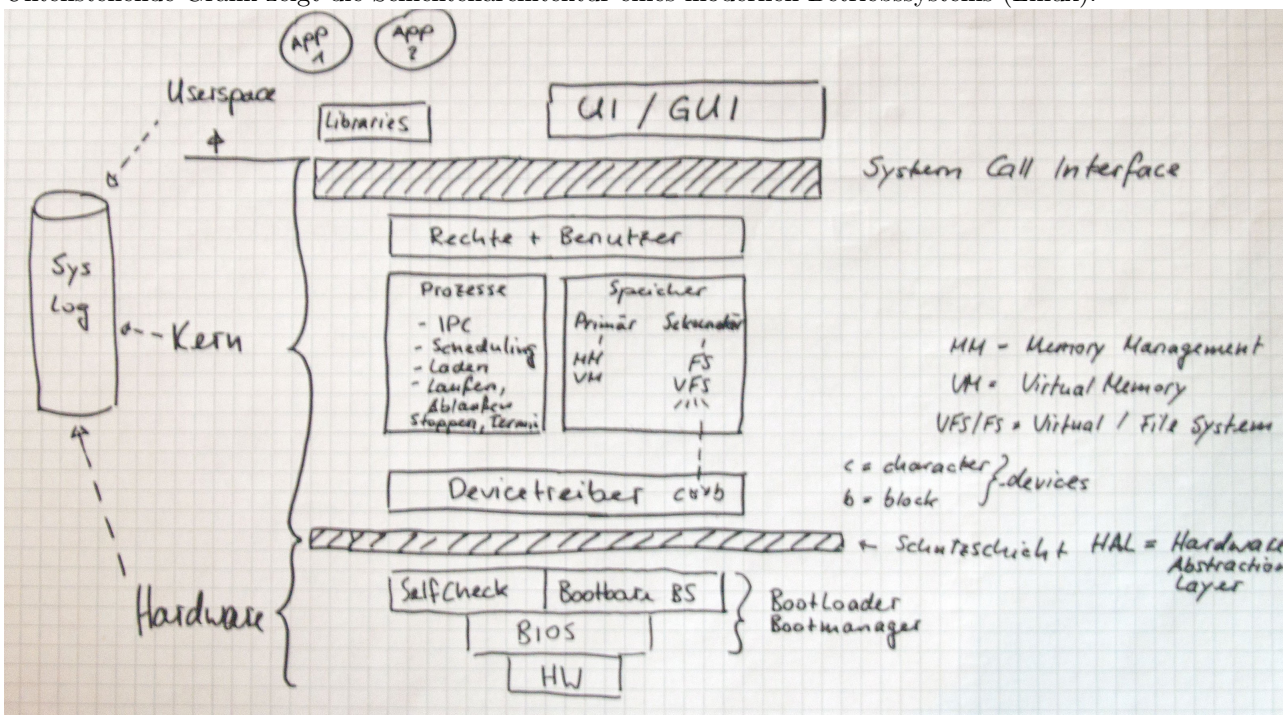
2 Blockstruktur eines Betriebssystems

2.1 Aufgaben des Betriebssystems

- Start des Systems
- Laden und Unterbrechen von Programmen (Laufzeitumgebung)
- Methoden für die Interprozesskommunikation
- Verwaltung der Prozessorzeit
- Verwaltung des primären und sekundären Speicherplatzes für das Betriebssystem und seine Anwendungen
- Verwaltung der angeschlossenen Geräte, Netzwerke etc.
- Schutz des Systemkerns und seiner Ressourcen vor nicht intendierter Benutzung
- Benutzerführung, Rollen und Rechte
- Einheitliche Schnittstelle für die System- und Anwendungsprogrammierung Ereignisprotokollierung

2.2 Grapische Darstellung

Untenstehende Grafik zeigt die Schichtenarchitektur eines modernen Betriebssystems (Linux).



2.3 Aufgabenteilung der Blöcke

2.3.1 Dateisystem

- Struktur des Dateisystems (Baum, Graph, flach, ...)
- Strukturelemente (Directories)
- Zugriffsrechte auf Directories und Dateien
- Anlage, Suche, Manipulation, Löschen von Dateien

- Verwaltung von Datenblöcken auf Speichermedien
- Kombination von Dateisystemen (mounting)
- Benutzerschnittstelle und Navigation
- Backup / Restore

2.3.2 Prozesssteuersystem

- Prozesse kreieren
- Prozesse starten
- Prozesse schedulen, Warteschlangen, Ressourcenverbrauch
- Prozesse stoppen / unterbrechen
- Prozesse terminieren (freiwillig / wegen Fehler)
- Prozesskommunikation (Prozess-Prozess und Kern-Prozess / Prozess-Kern)
- Zuordnung von Hauptspeicher und anderen geteilten Ressourcen
- Ein-/Auslagerung von Prozessen
- Prozesse und ihre Zustände anzeigen

2.3.3 System Call Interface

- Einzige Schnittstelle zwischen Kern und Benutzer
- Normierung der Syntax und Semantik (POSIX 1003)
- Parametrisierung und übergabe
- übergabe der Kontrolle $\rightarrowtail$ Betriebsmodi

2.3.4 Programmierung

- Wahl der Programmiersprache / Systempräferenz
- System-/Applikationsnahe Bibliotheksfunktionen
- Programmierumgebung (Editor, Compiler, Assembler, Linker, Loader, Debugger)
- Bundling in einer Applikation (z.B. Eclipse)

2.3.5 Benutzerschnittstelle

- Textuelle Basis-Schnittstelle mit Kommando-Interpreter (Shell) Konsole
- Programmierbarkeit (Scripting, Pipelining, I/O-Redirection) der Benutzerschnittstelle
- Graphische Benutzerschnittstelle (GUI) mit
- Abstraktion der unterliegenden Komplexität und Syntax für Nicht Systemspezialisten
- Austauschbarkeit der Shell und der Systembefehle (Applikationen)

2.3.6 I/O Management

Ein Betriebssystem muss auch die Hardware kontrollieren:

- Die Fähigkeiten der Hardware voll ausschöpfen.
- Die verschiedenen inhomogenen Komponenten zu einer Einheit formen.
- Die Hardware schützen vor unerlaubtem Zugriff.

Anm.: Peripheriegeräte sind meist unterschiedlich, sollten aber leicht in das System integrierbar sein.

2.3.7 File System als generelle Schnittstelle

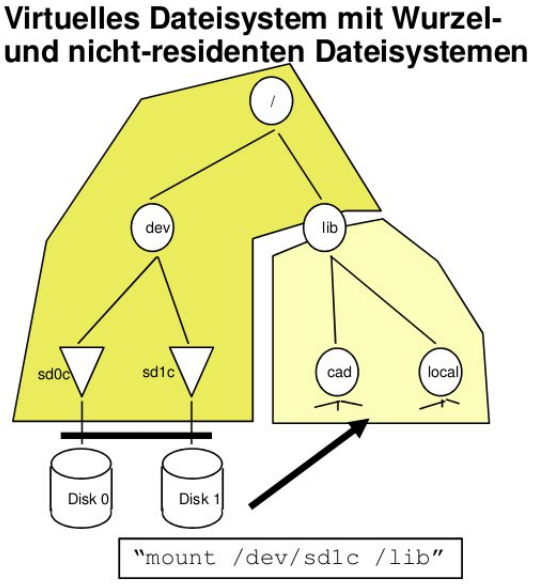
Die Idee von Unix war, dass File System für möglichst viele Subsysteme als Schnittstelle zu verwenden.

- Dateien, Directories
- Prozesssynchronisation (Lock Files, ...)
- Prozesskommunikation (Pipes, Sockets)
- Peripheriegeräte (Device Special Files)
- Kommunikationsprotokolle (TCP/IP, ...)
- Prozesse (/proc Dateisystem)

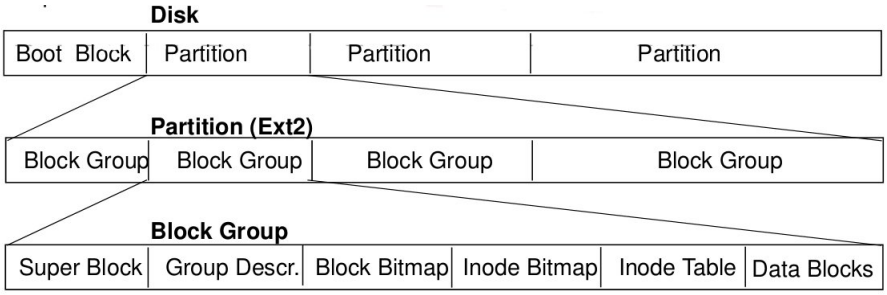
Anm.: Es bedingt einer zusätzlichen Abstraktionsschicht.

3 Filesystem

Die anschliessende Graphik zeigt ein virtuelles Dateisytem:



3.1 Partition auf der Disk



Superblockinhalt:

- Anzahl inodes und Datenblöcke
- Adresse des 1. Datenblocks
- Anzahl freie Blöcke und inodes
- Grösse eines Datenblocks
- Blöcke / inodes pro Gruppe
- Anzahl Bytes pro inode

3.2 Blockallokationen

3.2.1 Zusammenhängende Belegung

Vorteile:

- Einfachste aller Methoden
- Sehr schneller direkter Zugriff auf die Daten

- Für die Lokalisierung der Dateiblöcke brauchen wir nur Anfangsblock und Gröe der Datei zu wissen.
- Lese-Operationen können sehr effizient implementiert werden.
- Gute Fehlereingrenzung

Nachteile:

- Dynamische Dateigröen sind ein Problem
- Im Laufe der Zeit wird die Platte fragmentiert.
- Platz zu finden für neue Dateien ist ein Problem
- Verwaltung von freien Speicherplätzen notwendig
- Regelmäige Kompaktifizierung notwendig
- Platte hin- und zurück kopieren

3.2.2 Verlinkte Blöcke

Jede Datei wird als verkettete Liste von Plattenblöcken gespeichert. Nur die Plattenadresse des ersten und letzten Blocks wird in dem Verzeichniseintrag gespeichert.
Vorteile:

- Keine externe Fragmentierung
- Sequenzieller Zugriff ist kein Problem

Nachteile:

- Schlechter wahlfreier Zugriff auf Dateiinhalte
- Jeder Verweis verursacht einen neuen Plattenzugriff
- Overhead für das Speichern der Verkettung (Lösung: clusters aus mehren Blöcken)
- Erhöhter Aufwand bei Dateizugriffen
- Schlechte Fehlereingrenzung

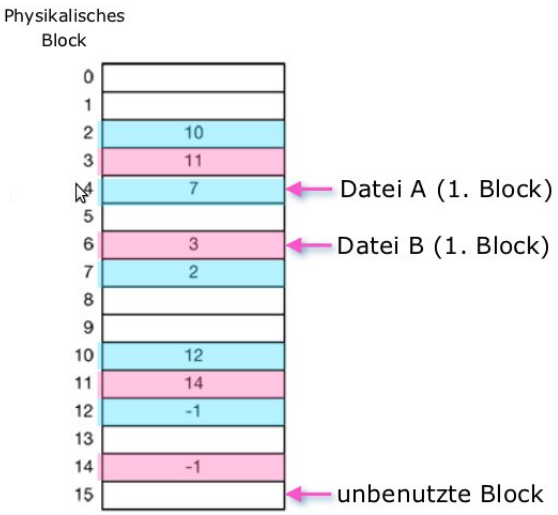
3.2.3 Filemap (Landkarte)

Vorteile:

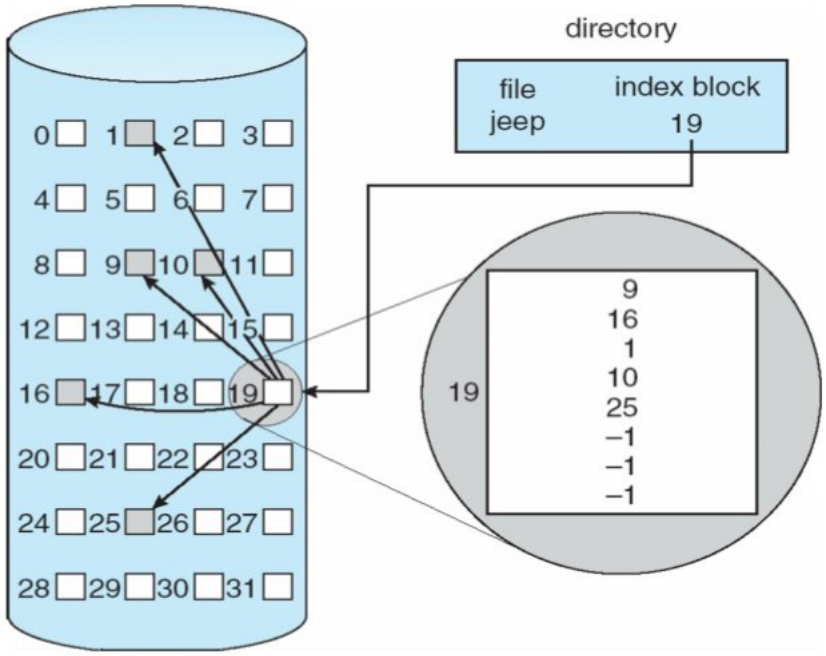
- Dateien können sehr leicht und effizient vergröert werden

Nachteile:

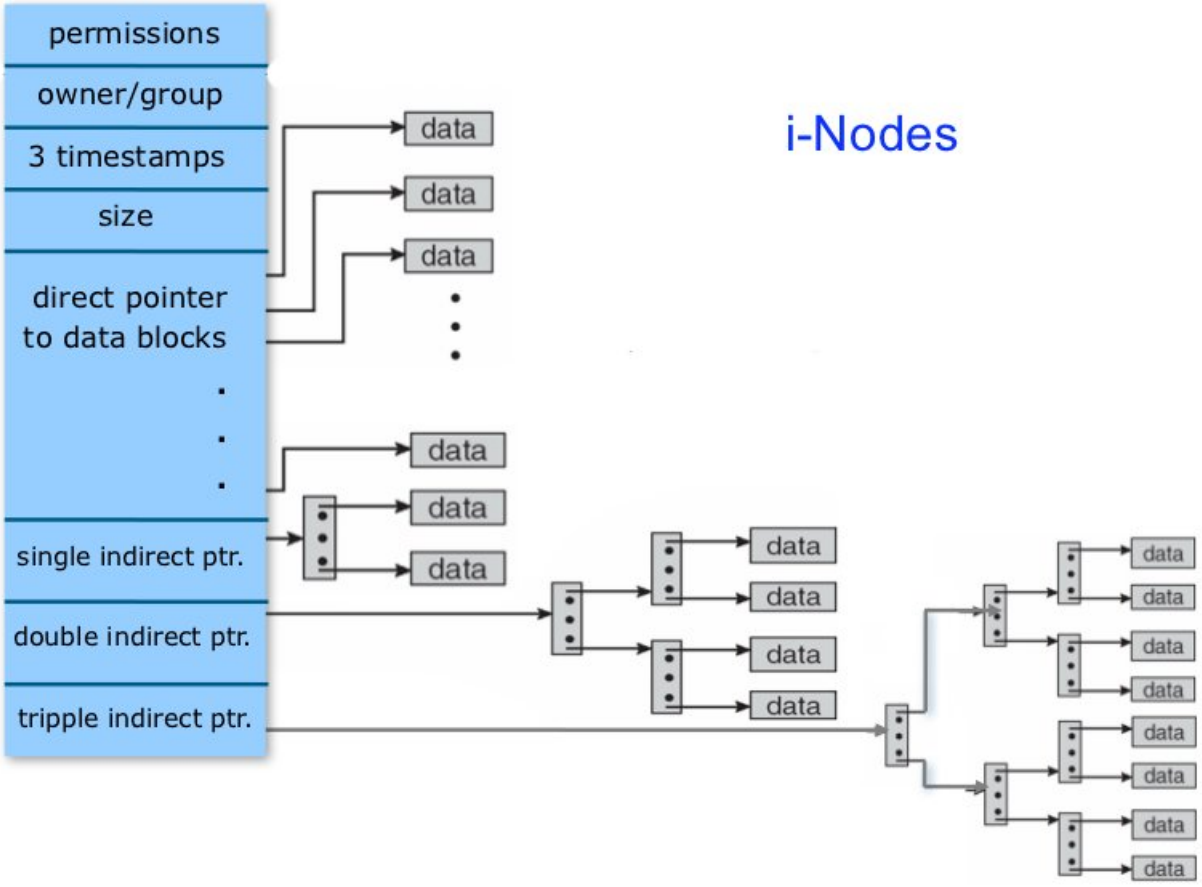
- Interne Fragmentierung
- schlecht für random accesses
- fehleranfällig



3.2.4 Index Allokation



3.3 Struktur von Inodes



3.3.1 Informatione Inode

Auf der Disk	Zusätzlich im Memory
• Inode Nummer • Anzahl hard links • Typ (-, l, d, b, c, p) • Rechte (-, r, w, x, s) • Besitzer • Gruppe • Grösse • Letzte access time • Letzte content change time • Letzte inode modification time • Datenblock Information	• Link auf die hash list • Link auf die inode list • Benutzerzähler • Gerätenummer • Device special file Indikator • Grösse eines Blocks • Anzahl Blöcke • Lock auf den inode • Mount point Indikator • Warteschlange wartender Prozesse • Locks auf die Datei • Hauptspeicher-Region (für Memory-mapped file I/O) • Belegte Seiten im Hauptspeicher

3.4 Designvorgaben für Filesystem

- Anzahl Disks und Disk Controller
- Verteilung auf Partitionen
- Blockgrösse pro Partition
  - Grosse Blöcke: schneller Zugriff auf Dateien (Datei-Durchschnittsgrösse < 4 kB)
  - Kleine Blöcke: weniger interne Fragmentierung
- Anzahl Dateien / Inodes pro Partition
  - Wenige Inodes: mehr Platz für Dateiblöcke
  - Viele Inodes: mehr Dateien pro Partition mgl.

4 Prozesse

4.1 Einleitung

4.1.1 Was ist ein Prozess?

Ein oder mehrere Programme (deterministische Sequenz von Instruktionen) werden auf einem oder mehreren physischen oder virtuellen Prozessoren ausgefuhrt. Zu jedem Zeitpunkt der Ausfuhrung verbunden mit einem computational state (aktuell verwendete Variablen etc. im Programm), externe Ressourcen wie Zustand der CPU, Register, Zeitnahme, usw. Ein Prozess wird durch das Betriebssystem strikt uberwacht und verwaltet

4.1.2 Anforderungen an ein Prozess-Steuersystem

- Prozesse kreieren
- Prozesse starten
- Prozesse schedulen, Warteschlangen, Ressourcenverbrauch
- Prozesse stoppen / unterbrechen
- Prozesse terminieren (freiwillig / wegen Fehler)
- Prozess-Signalisierung und -kommunikation
- Faire Zuordnung von Hauptspeicher und anderen geteilten Ressourcen
- Ein-/Auslagerung von Prozessen bei vollem Speicher
- Prozesse und ihre Zustande anzeigen

4.1.3 Unix-Prozess Segmente

- Text Segment (8 K)
- Daten Segment (32 K)
- Stack Segment (64 K)
- Shared Memory Segment
- Mapped File Segment

4.2 Kernel und User Mode

Ein Prozess hat mindestens (in Unix genau) zwei Ausfuhrungsmodi:

User Mode: Es wird der normale Programmcode ausgefuhrt.

Kernel Mode: Ss werden Systemaufrufe ausgefuhrt oder Ausnahmen behandelt.

Der Ubergang erfolgt durch einen Systemaufruf durch das Programm, eine Ausnahmesituation (Fehler) oder durch asynchrone Events (Kom- munikation etc). Beide Modi haben separate Segmente und sind voneinander abgeschirmt.

4.2.1 Prozess-/Kontext- Wechsel

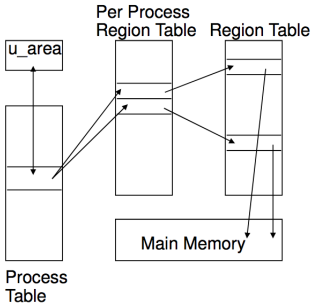
Wenn ein Prozess:

- warten muss (z.B. auf I/O oder einen Event),
- seine zugeordnete Laufzeit oder andere Ressourcen- grenzen erreicht bzw. uberschreitet,
- terminiert oder gestoppt wird,
- die CPU freiwillig abgibt

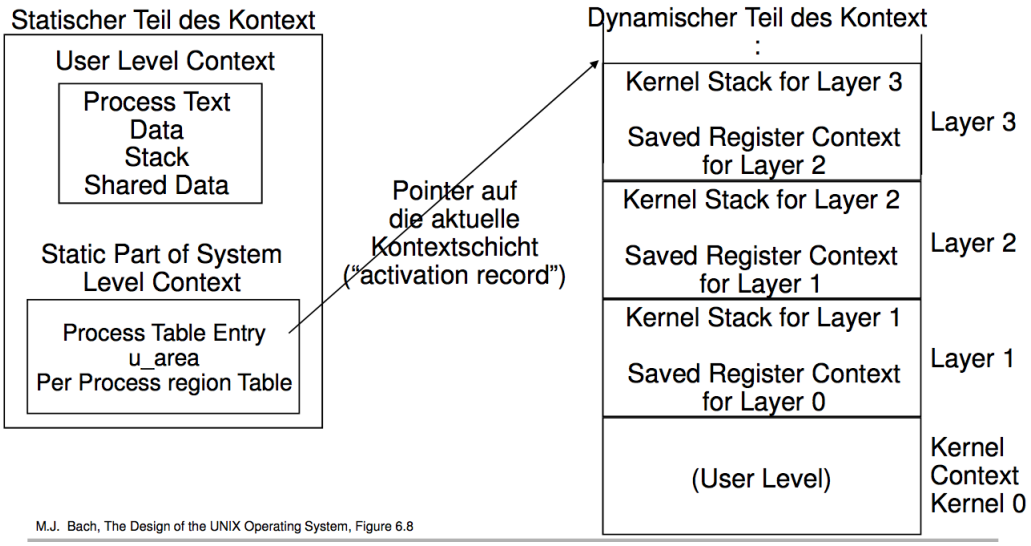
muss das Betriebssystem die CPU einen anderen ablaufbereiten Prozess zuteilen und diesen starten. Dies erfordert das Abspeichern des exakten Prozess- Zustandes und das spatere Restaurieren, wenn der Prozess wieder weiterlaufen soll.

4.2.2 Kernel-Datenstrukturen fur das Prozess-Management

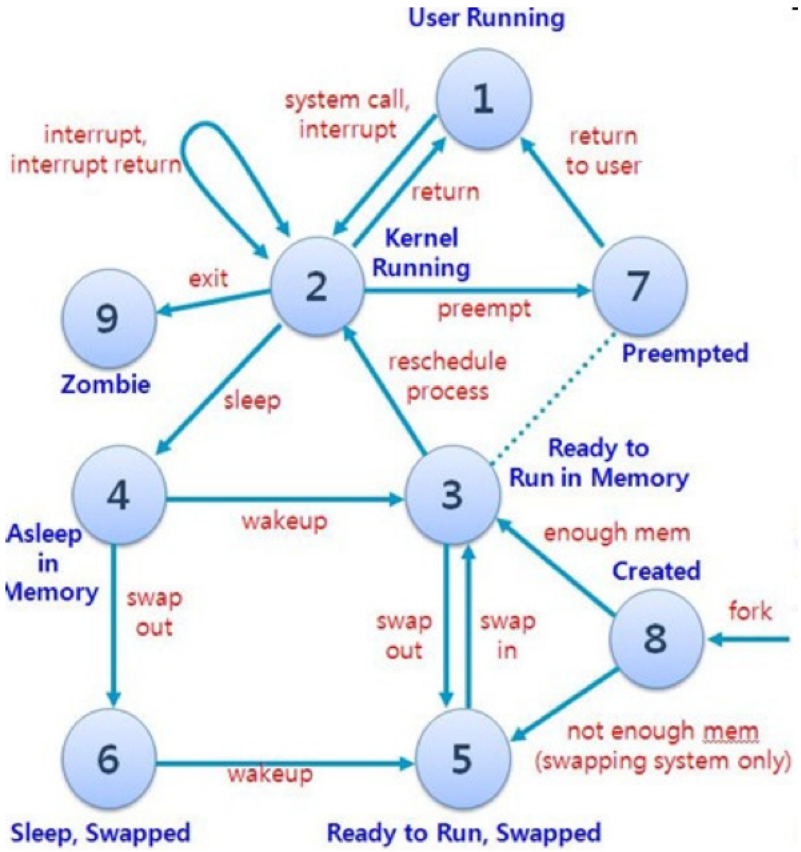
In alteren Unix-Varianten ist die Grosse der Prozesstabelle statisch (schnelle Indexierung, Lizenzierung uber Anzahl Prozesse / Benutzer).
Alternativ kann die Prozesstabelle eine verkettete Liste sein (variable Anzahl Prozesse, aber komplizierte Indexierung und Uberlauf-Gefahr).
Linux verwendet eine Mischform (dynamisch angelegte Prozesskon- trollblocke (PCB) in einer verketteten Liste mit einer statischen Hash- Tabelle fur die schnelle Suche).



4.3 Sichern des Prozess Kontexts



4.4 Prozess Zustände



4.5 Prozess-bezogene System Calls

fork Erstellen eines neuen Prozesses durch Kopieren

exec AusfuhreneinesneuenProgrammsineiner vorhandenen Prozesshulle

Signal Stoppen eines Prozesses

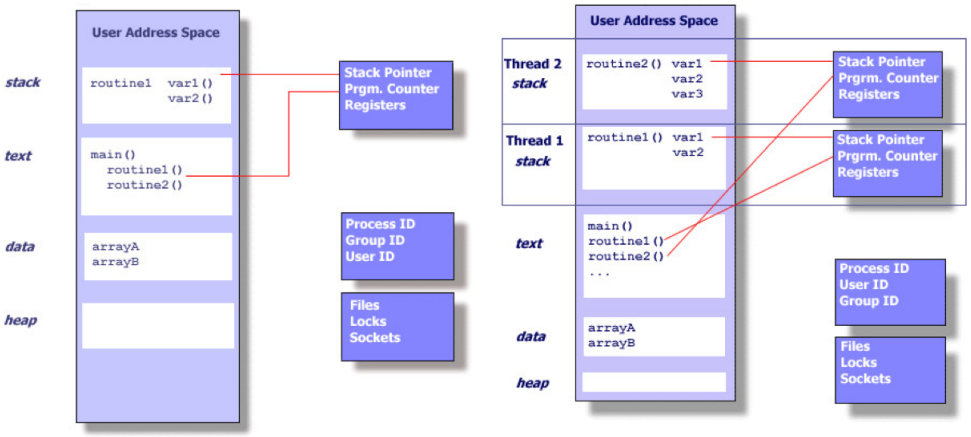
exit Terminieren eines Prozesses

wait Warten auf das Terminieren eines Prozesses, einfache Synchronisation

4.6 Prozesse versus Threads

Kontextwechsel sind eine schwere Operation mit viel Verarbeitungsaufwand durch den Kernel. Da viele Unix-Prozesse I/O-intensiv sind, verbringen sie die meiste Laufzeit mit Warten, dadurch erhoht sich die Anzahl von Kontextwechseln im System. Neuere Unix-/Linux-Systeme unterstutzen mehr als einen parallelen Ausfuhrungspfad innerhalb eines Prozesses (multi-threading). Es muss kein Kontextwechsel vorgenommen werden, um eine andere Aktivitat zu starten.
Aber:

- Das Scheduling muss innerhalb des Prozesses erfolgen,
- die Threads sind verwandt, d.h. ihr Code liegt innerhalb des gleichen Unix-Prozesses,
- der Programmierer muss fur die Datenintegritat selbst sorgen.



4.6.1 Basismodell fur Threads

- Kernel-Code (System Call Interface) oder Library?
- Ausfuhrungsmodelle
  - Master/Slave(s)
  - Cooperating Pool
  - Pipeline
  - Hybrid
- Besondere Anforderungen
  - Synchronisation
  - Betriebsmittel-Zuteilung (Scheduling)
  - Ausnahmebehandlung

4.6.2 Benutzungs- und Administrationssicht auf ein Prozess-Steuersystem

- Prozess-Erzeugung und Termination
- Identifikation
- Priorisierung
- Besitzer
- Ressourcenverbrauch
- Prozess-Ein-/Auslagerung
- Signalisierung
- Prozesskommunikation

5 Hauptspeicherverwaltung

5.1 Virtual Memory

- Hauptspeichererweiterung pro einzelnes System oder Prozess.
- Systematische Abstraktion für systemspezifische Overlay- Techniken
- Organisation des Hauptspeichers in gleich grosse, einheitlich adressierbare Einheiten (Seiten, pages)
- Benötigt hardware-unterstütze Abbildung zwischen physischen und virtuellen Adressen

5.2 Swapping

Swapper (Process 0) wird periodisch vom Kernel aufgerufen.

Swap Device $\longleftrightarrow$ Swapper $\longleftrightarrow$ Hauptspeicher

Swap Out (Hauptspeicher zu SwapDevice):

- Kein Platz im HS für weitere Prozesse, Prozess ruft aber fork auf $\Rightarrow$ fork swap
- Kein Platz im HS aber Prozess wächst, weil z.B. Stack wächst $\Rightarrow$ expansion swap
- Swap Device ausgelagerter wartender Prozess wird ready to run und wird vom Scheduler ausgeführt $\Rightarrow$ exchange swap

Swap In (SwapDevice zu Hauptspeicher):

- Nur ready to run Prozesse sind wählbar $\Rightarrow$
- Präferenz auf Prozesse die ¿ 2 sec. Ausgelagert waren

5.3 Demand Paging

Anforderung: ein einzelner Prozess soll grösser sein dürfen, als der verfügbare Hauptspeicher.

Voraussetzungen:

- Hardware unterstützt seitenorientieres Speichermanagement (1/2 4 kB / Seite)
- Wiederaufsetzbare CPU-Instruktionen (wenn Instruktionen über eine Seiten- grenze verlaufen)

Idee:
Nur die gerade verwendeten Teile des Codes werden geladen. Dies sind 10 bis 15 Prozent die im HS liegen (working set). Wird auf eine Seite zugegriffen die nicht geladen ist, so gibt es einen page fault und die Seite wird nachgeladen.

Optimierung des demand paging durch

- Reference bit $\rightarrow$ ermöglicht nicht-lineares working set
- Age bit $\rightarrow$ verbleibende Zet einer Seite im working set

Zwei Aufgaben für das Paging-Subsystem:

- Seitenalterung und Auslagerung/Löschung genügend alter Seiten
- Bearbeitung von page faults

6 I/O, Peripherie

6.1 Anforderungen von Peripheriegeraten

- Ressourcenverwaltung:
  - Hauptspeicher fur Zwischenpufferung von Daten beim Transfer
  - CPU-Zeit fur das Behandeln asynchroner Events (z.B. Ankunft von Daten an der Netzwerkschnittstelle)
- Zugriffssteuerung und synchronisation:
  - Einheitliche Schnittstelle
  - Synchronisation von Zugriffen durch die Prozesse
  - Signalisierung
- Scheduling

6.2 Aufgaben und Funktionsweise des I/O-Subsystems

- Aufgabe: schneller und zuverlassiger Datentransfer zwischen Geraten (Disk, Drucker, Tastatur, Bildschirm, Maus etc.) und Prozessen, d.h. Datenstrukturen im Prozess-Adressraum (read und write Operationen)
- Design: Das I/O Subsystem besteht aus einer oberen Schicht, die Daten zwischen dem Benutzer- und dem Kernel-Adressraum bewegt, und einer unteren Schicht, die Daten zwischen dem Kernel-Adressraum und den Geraten bewegt.
- Standardisiertes I/O (Gerateunabhangigkeit bezuglich Programmierung und Benutzung)
- Optimiertes I/O (abhangig vom Gerat und dessen Eigenschaften, Durch- satz, Sicherheit usw.)
- Konsistenz trotz Unterbrechbarkeit der Operationen und Zugriffssicher- heit wenn Daten zwichen Kernel und Benutzer-Prozess bewegt werden
- Drei Typen von I/O: Datei-basiert, Zeichen-basiert, STREAM-basiert

6.3 Buffer-Cache

6.3.1 Der klassische Buffer Cache

- Ziel: Optimierung des Zugriffs auf block-orientierte gerate, maximale Menge Daten im Speicher behalten.
- Strategie 1: vorausschauendes Lesen (read ahead)
  - Vorteil: beschleunigt das sequentielle Lesen (z.B. einer Datei)
  - Risiko: potentielle Verschwendung von Hauptspeicher
- Strategie 2: verzogertes Schreiben (delayed write)
  - Vorteil: Bundeln von Daten in Blocke fur das Schreiben auf langsame Gerate (z.B. Disk)
  - Risiko:nacheinemerfolgreichenwrite()Systemaufrufsind die Daten noch nicht auf der Disk gespeichert (Verlustrisiko)
- Optimiert fur die Arbeit mit dummen Peripheriegeraten

6.3.2 Der neue Buffer Cache

Der neue Buffer Cache verwendet einen seitenorientierten Zugriff auf Dateien, ahnlich der Hauptspeicherverwaltung. Zu diesem Zweck unterstutzt der Kernel einen neuen Regionstyp (memory-mapped file), der den Inhalt einer Datei in Speicherseiten im Prozessadressraum abbildet. Der Zugriff erfolgt weiter via inodes und einen Offset der Kernel bildet diese Zugriffe intern auf die mmap- Strukturen ab (typischerweise 8096 aufeinanderfolgende Blocke von 8 kB Grosse).

6.4 Basisfunktionen des Datei I/O

Jeder Dateisystem-Typ definiert zwei spezifische Opera- tionen (getpage() und putpage()). Diese Funktionen werden fur den seiten-orientierten Daten- transfer zwischen dem Kernel-Adressraum und den Dateien im Dateisystem verwendet.ur die Organsiation des seiten-orientierten Transfers werden 5 Basisfunktionen im Kernel verwendet:

pageio_setup(): Pufferspeicher alloziieren

strategy(): Geratetreiberfunktion fur das Lesen/Schreiben uber den alten oder neuen Buffer Cache

biowait(): Warten auf das Ende eines synchronen Schreibvorgangs

biodone(): Aufwecken eines in biowait() schlafenden Prozesses (up-call durch den Geratetreiber)

pageio_done(): De-allokation der durch pageio_setup() gebrauchten Puffer

6.5 Einbindung und Verwaltung von Peripherie-Geraten

- Zentrales Element: Geratespezialdateien im /dev bzw. /devices Dateisystem als einheitliche Schnittstelle
- Major / Minor Device Number zur Identifikation
- Zugriffsrechte auf die Geratespezialdateien sind relevant
- Gerate in verschiedenen Betriebs-Modi: block- oder zeichen-weiser Zugriff
- Echte Gerate und Pseudo-Gerate (z.B. virtuelle Terminals, Netzwerkprotokolle oder /dev/null)

6.6 Geratetreiber

Geratetreiber sind die einzige Schnittstelle, uber die ein Prozess mit Geraten kommunizieren kann. Sie sind Teil des Kernel-Codes des Systems, und werden entweder statisch beim Systemstart oder zur Laufzeit (in Linux: insmod/rmmod) geladen. In Unix sind Geratetreiber Teil jedes Prozesses (uber den Kernel-Code) in anderen Betriebssystemen sind sie nur speziellen Kommunikationsprozessen zuganglich uber die die anderen Prozesse dann mit Geraten kommunizieren mussen.

6.6.1 Interrupt Behandlung

Routinen zur Interrupt-Behandlung sind sehr system- und hardware-spezifisch, es gibt nur wenige allgemeine Regeln fur das Design. In Unix werden Interrupts immer im Kontext des gerade laufenden Prozesses behandelt, auch wenn der Prozess den Interrupt nicht verursacht hat oder nicht davon profitiert. Der gerade laufende Prozess muss also seine Arbeit unterbrechen, den Kontext sichern, den Interrupt behandeln, den Kontext restaurieren und kann dann weiterarbeiten. Die Zeitstrafe fur das Behandeln von Interrupts verbleibt bei jedem Prozess.

6.6.2 Block-orientierte Geratetreiber

Block-orientierte Gerate erlauben random access auf Datenblocke (meist Disk-Partitionen zb das Mounten eines Dateisystems). WennDatenviadas/mntDateisystemgelesen oder geschrieben werden, mussen die entsprechenden Prozeduren fur das assoziierte Block Device aus der Block Device Switching Table verwendet werden (major device number). Die Hauptschnittstelle fur die Arbeit mit block-orientierten Geratetreibern ist die strategy() Funktion. Sie implementiert die read() und write() Operationen. Der Aufruf von strategy() geht durch den Buffer Cache und kann synchron (sleep) oder asynchron ausgefuhrt werden.
Achtung: nicht jeder Aufruf von strategy() fuhrt zum sofortigen Schreiben der Daten auf die Disk (Datenverlust!).

Die Raw I/O Schnittstelle
In diesem Modus werden der Buffer Cache und das Abbilden von Dateien auf virtuelle Speicheradressen (mmap) nicht verwendet (z.B. fur Datenbanken). Stattdessen werden die Daten direkt zwischen dem Prozess-Adressraum und dem on-board Speicher des Gerats transferiert (DMA). Dies vermeidet den Kopier- vorgang zwischen Prozess-Adressraum und Kernel- Adressraum. Das Lesen und Schreiben ist somit nicht optimiert, d.h. jeder Aufruf von read() oder write() resultiert in einem physischen Datentransfer uber den entsprechenden zeichen-orientierten Geratetreiber.

6.6.3 Zeichen-orientierte Geratetreiber

Bei der Verwendung von zeichen-orientierten Geratetreibern ist der Buffer Cache nicht involviert. Es wird zwischen STREAM- und nicht-STREAM- basierten zeichen-orientierten Geraten unterschieden (Flusskontrolle etc.). Die Eingabe kann im cooked mode (Verarbeitung von Spezialzeichen wie Backspace) oder raw mode erfolgen. Hauptnutzer von zeichen-orientierten Geraten sind Programme, die mit interaktiven, zeichenbasierten Geraten wie Terminals, Modems usw. arbeiten. Die wesentlichen Operationen sind read(), write(), ioctl(), poll(), und mmap().

- Fur die read() und write() Operationen gibt es geratespezifische Prozeduren in der Character Device Switching Table.
- Über die ioctl() Funktion konnen geratespezifische Eigenschaften (Ubertragungsgeschwindigkeit, Paritat, Puffergrossen etc.) abgefragt oder verandert werden.
- Mittelsderpoll()Funktionkanneinzeichen- orientiertes Gerat oder ein STREAM bezuglich Bereitschaft fur eine Eingabe- oder Ausgabe- Operation angefragt werden.
- Die Abfrage ist nicht blockierend, d.h. ein Prozess wird nicht blockiert, wenn das Gerat gerade nicht bereit ist.
- Mit dem Systemaufruf mmap() kann ein Prozess den Inhalt einer Datei in seinen virtuellen Adressraum kopieren und dort die Daten direkt manipulieren, anstatt read() und write() zu verwenden. Dies ist der gleiche Mechanismus wie im neuen Buffer Cache, jedoch ist er nun auch direkt fur die Progammierung zuganglich. Falls die Datei gerade von mehreren Prozessen benutzt wird, schreiben alle Prozesse auf die gleiche physische Speicherposition. Dermmap()Systemaufrufkannzudemaufdem zeichen-orientierten Geratespezialdatei eines Gerats benutzt werden, um direkten Zugriff auf den on-board Speicher des Gerats zu erhalten (z.B. auf den Frame Buffer einer Video-Karte)

7 Sheduling Strategien

Funktionsweise:

- Fairness / Regeleinhaltung bezüglich der Zuteilung von Betriebsmitteln an Prozesse gemäss definierter Kriterien
- Vermeidung von Starvation und Deadlocks

Einsatzgebiete:

- Echtzeitsysteme mit harten Garantien
- Möglichst unterbruchsfreie Batchverarbeitung
- Interaktives Mehrbenutzersystem

7.1 Scheduling in Unix

3 Prioritätsklassen:

- Realtime: Scheduling mit fixen Prioritäten
- System: Geschlossene Scheduling-Klasse für Systemprozesse
- Time-Shared: Für alle Benutzerprozesse

Prioritäts-basiertes Scheduling:

- Verminderte Prozesspriorität mit steigendem Ressourcenverbrauch
- Verminderung der Priorität gemäss Laufzeit

Scheduling-Strategien:

- Kleine, schnelle Prozesse präferieren (z.B. Shell)
- Ressourcen-intensiven Prozessen alle benötigten Ressourcen geben: Schnellere Beendigung und Freigabe und Beendigung im Zeitlimit

8 Systemüberwachung

Was kann alles sinnvollerweise überwacht werden:

- Betriebssystem Typ, Alter
- Laufzeitüberwachung
- Verschiedene Speicher, Prozesse CPU, RAM, Disks
- Auslastung/Trending CPU
- Netzwerkanschlüsse
- Autorisierter Anschluss an Netzwerk
- Peripheriegeräte,
- Backupüberwachung
- Software und Version, Lizenz
- Laufzeitüberwachung Disk
- Benutzer und Berechtigung

Anm.: Verschiedene Fachbegriffe rund um Systemüberwachung

- Incident (Einzelfall)
- Problem (Reihung)
- Desaster (Notfallplan ausführen!)
- Triage (Rollen, Verantwortlichkeiten, Beurteilungskriterien, Prozesse, Zeitfaktor, ...)