

Klassisches Unix

Download der Folien

<http://www.christians-software.de/uni>

(heute Abend)

von Christian Schmitz

Gliederung

1. Einführung
2. Geschichte
3. Systemaufbau
4. Dateisystem Schnittstelle
5. Dateisystem intern
6. Prozesse
7. Kommunikation zwischen Prozessen
8. Sockets

Gliederung

1. Einführung

2. Geschichte

3. Systemaufbau

4. Dateisystem Schnittstelle

5. Dateisystem intern

6. Prozesse

7. Kommunikation zwischen Prozessen

8. Sockets

Einführung

Unix ist seit 1978 bekannt geworden durch die 7. Edition von Unix entwickelt von der Firma Bell Telephone.

Der Umfang von Unix macht eine umfassende Beschreibung nicht möglich, daher werden nur einige Konzepte vorgestellt, die bis heute in aktuellen Unix Systemen verwendet werden.

Schon 1974 hatten Richie und Thompson die folgende Liste von Designvorgaben für Unix veröffentlicht:

1. Eine einheitliche Schnittstelle für Dateien, Geräte und Kommunikation zwischen Prozessen.
2. Ein hierarchisches Dateisystem mit Laufwerken, die dynamisch an- und abgemeldet werden können.
3. Die Möglichkeit für Prozesse andere Prozesse asynkron zu starten.
4. Eine für jeden Benutzer frei wählbare Benutzeroberfläche (Shell).
5. Hunderte von Programmen und Werkzeugen inklusive einem duzend Programmiersprachen.
6. Eine hohe Portierbarkeit durch die Verwendung der Programmiersprache C anstelle von Assembler Code.

Gliederung

1. Einführung
- 2. Geschichte**
3. Systemaufbau
4. Dateisystem Schnittstelle
5. Dateisystem intern
6. Prozesse
7. Kommunikation zwischen Prozessen
8. Sockets

Geschichte 1965 - 1969

Die Bell Telephone Laboratories starten gemeinsam mit der General Electric Company und dem Projekt MAC des Massachusetts Institute of Technology ihre Bemühungen das neue Betriebssystem Multics zu entwickeln.

Geschichte 1969 - 1971

- Multics erfüllt nicht die Anforderungen und Bell Telephone steigt aus dem Projekt aus.
- Einige Mitarbeiter von Multics entwickeln bei Bell Telephone Unix als Entwicklungssystem für PDP-7 Computer in Assembler.

Geschichte 1971 - 1973

- Unix erfüllte vieles was Multics versprach.
- Verwendung als Textprozessor
- Portierung auf PDP-11
- Portierung von Fortran auf Unix mit dem Ergebnis von B, einer Interpretersprache.
- Fortentwicklung von B zu C mit einem Compiler.

Geschichte 1973 - 1977

- Unix wird in C neu geschrieben.
- 25 Installationen und Unix Support Group bei AT&T.
- AT&T darf keine Computerprodukte verkaufen, aber für Universitäten wurde Unix kostenlos zur Verfügung gestellt.

Geschichte 1977 - 1982

- 500 Installationen, davon 125 an Universitäten.
- Viele verschiedene Varianten, auch kommerzielle mit Office Erweiterungen.
- Portierungen auf neue Plattformen:
interdata 8/32

Geschichte ab 1983

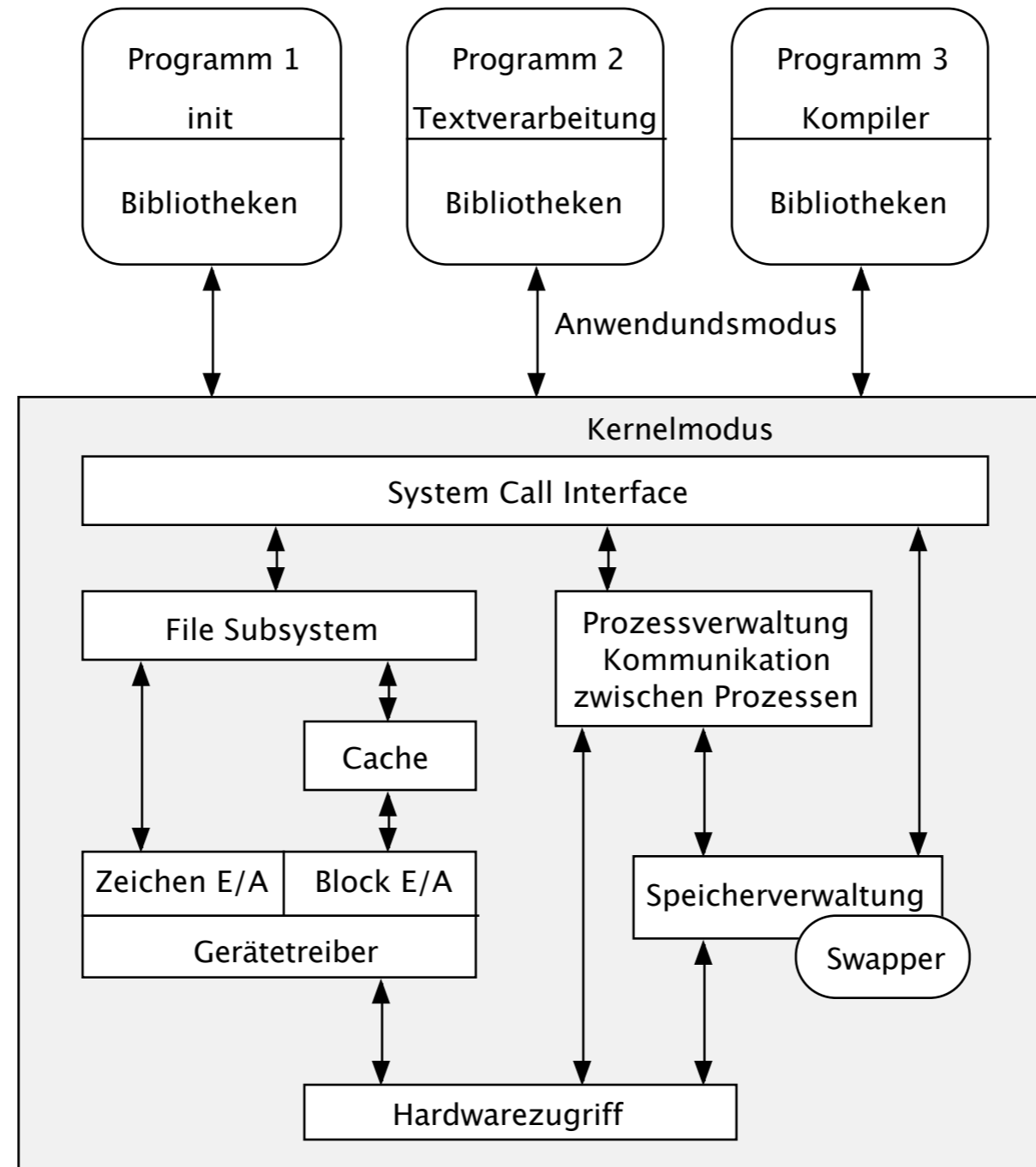
- Von AT&T kommerzielle Unix Version: System V
- Sun Solaris entsteht aus System V
- Paralell entstehen seit 1980 BSD Unix (Mac OS X, FreeBSD und OpenBSD) und Linux.

Gliederung

1. Einführung
2. Geschichte
- 3. Systemaufbau**
4. Dateisystem Schnittstelle
5. Dateisystem intern
6. Prozesse
7. Kommunikation zwischen Prozessen
8. Sockets

Systemaufbau

- Kernel Betriebssystem
- Mehrere Benutzer arbeiten an mehreren Terminals gleichzeitig.
- Kernel verwaltet und verteilt Ressourcen.
- Prozesse voneinander geschützt.



Unix Kernelmodule

Systemaufbau

- System Call Interface als Schnittstelle für Anwendungsprogramme.
- Dateisystem
- Dateicache
- Treibern für blockorientierte und zeichenbasierte Geräte.
- Speicherverwaltung mit Swapper
- Prozessverwaltung mit Kommunikation zwischen Prozessen.

System Call Interface

- Programme rufen Kernelfunktionen über das System Call Interface auf.
- Wechsel in den privilegierten Modus, der direkte Hardwarezugriffe erlaubt.
- Aufruf über Interrupt oder Trap.
(bei Linux Interrupt 80)
- Ausführung der Kernelfunktionen im Anwendungsprozess.

Kernel Einschränkungen

Das Kernel ist beim klassischen Unix nicht preemptiv. Es findet kein Prozesswechsel statt während eine Kernelfunktion läuft, sofern diese nicht auf ein E/A Event wartet.

Anstelle von Semaphoren und Locks werden die Interrupts ausgeschaltet für uneingeschränkten Zugriff auf gemeinsam genutzten Kernelspeicher.

Gerätetreiber

- Zwei Arten von Geräten: blockorientierte Geräte (z.B. Datenträger bzw. Festplatten) und Zeichenbasierte Geräte (z.B. Tastatur und Bildschirmkonsole).
- Gerätetreiber weitgehend in C geschrieben.
- Blockorientierte Geräte sortieren Anfragen zur Verarbeitung in optimaler Reihenfolge.

Dateicache

- Zwischenspeicher für blockorientierte Geräte.
- Ab dem zweiten Zugriff auf einen Block kommen die Daten aus dem Cache.
- Zugriff auf den Cache ist schneller, daher wird der Zugriff insgesamt schneller.
- Daten werden später geschrieben während die Anwendung weiter arbeitet.

Dateicache - Probleme

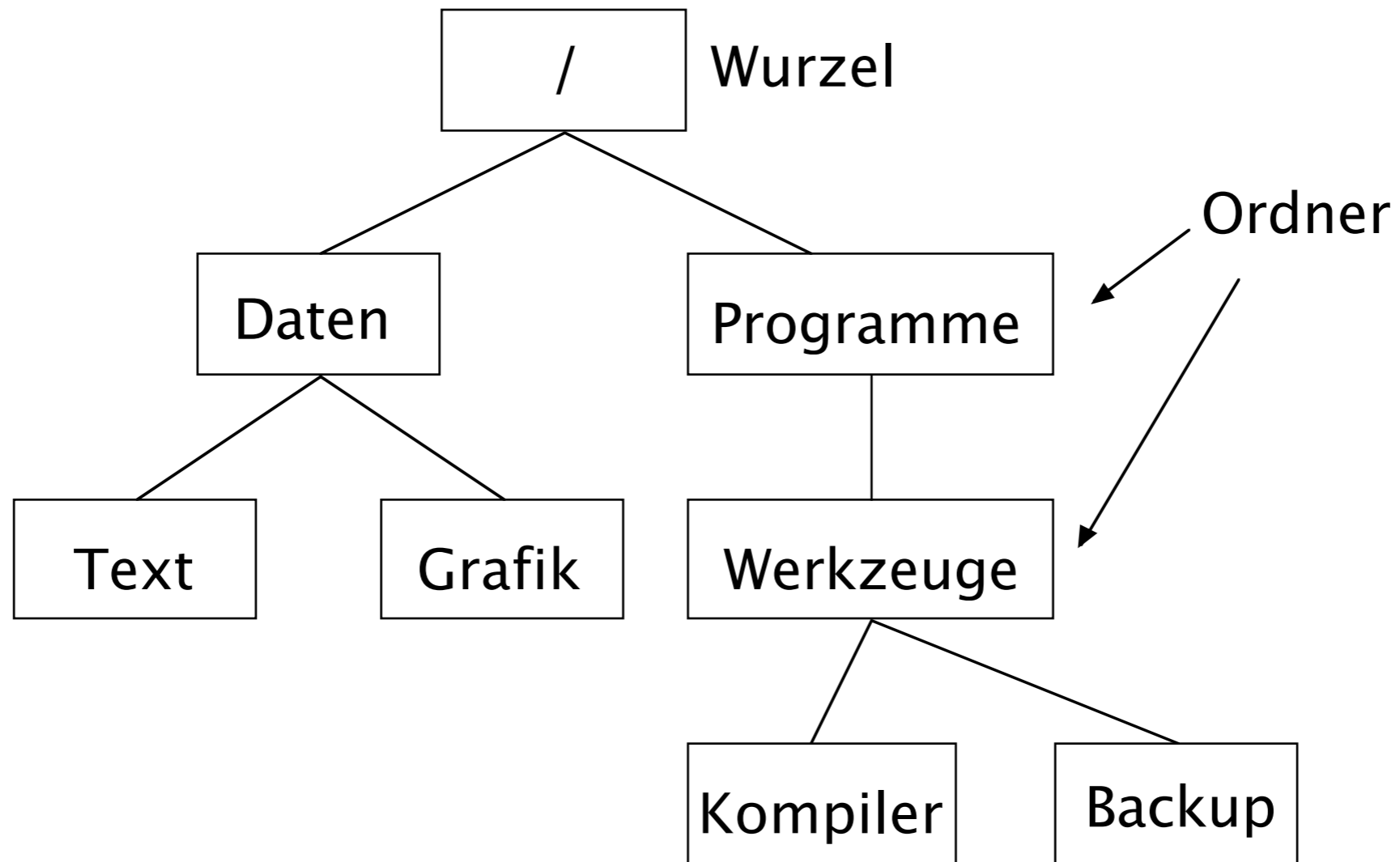
- Programm bekommen Erfolg gemeldet während Daten noch nicht geschrieben wurden.
- Programme können die Reihenfolge der Schreiboperationen nicht mehr vorherberechnen.
- Rückmeldung von Fehlern beim Schreiben werden spätestens beim Schliessen der Datei gemeldet.

Gliederung

1. Einführung
2. Geschichte
3. Systemaufbau
4. **Dateisystem Schnittstelle**
5. Dateisystem intern
6. Prozesse
7. Kommunikation zwischen Prozessen
8. Sockets

Dateisystem Schnittstelle

- Streng hierarchisch
- Dateien sind Byteströme,
keine Sammlungen von Datensätzen
- Ordner nehmen beliebig viele Dateien und
Unterordner auf.
- Wurzelordner heißt “/”.



Dateisystem

Ordner /dev

Spezielle Dateien für Geräte

- /dev/kram - Kernelspeicher
- /dev/console - Bildschirm
- /dev/tty - Tastatur
- /dev/modem - Modem
- /dev/diskos1 - 1. Partition der 1. Festplatte
- /dev/nul - Dummy
- /dev/random - Zufallszahlengenerator

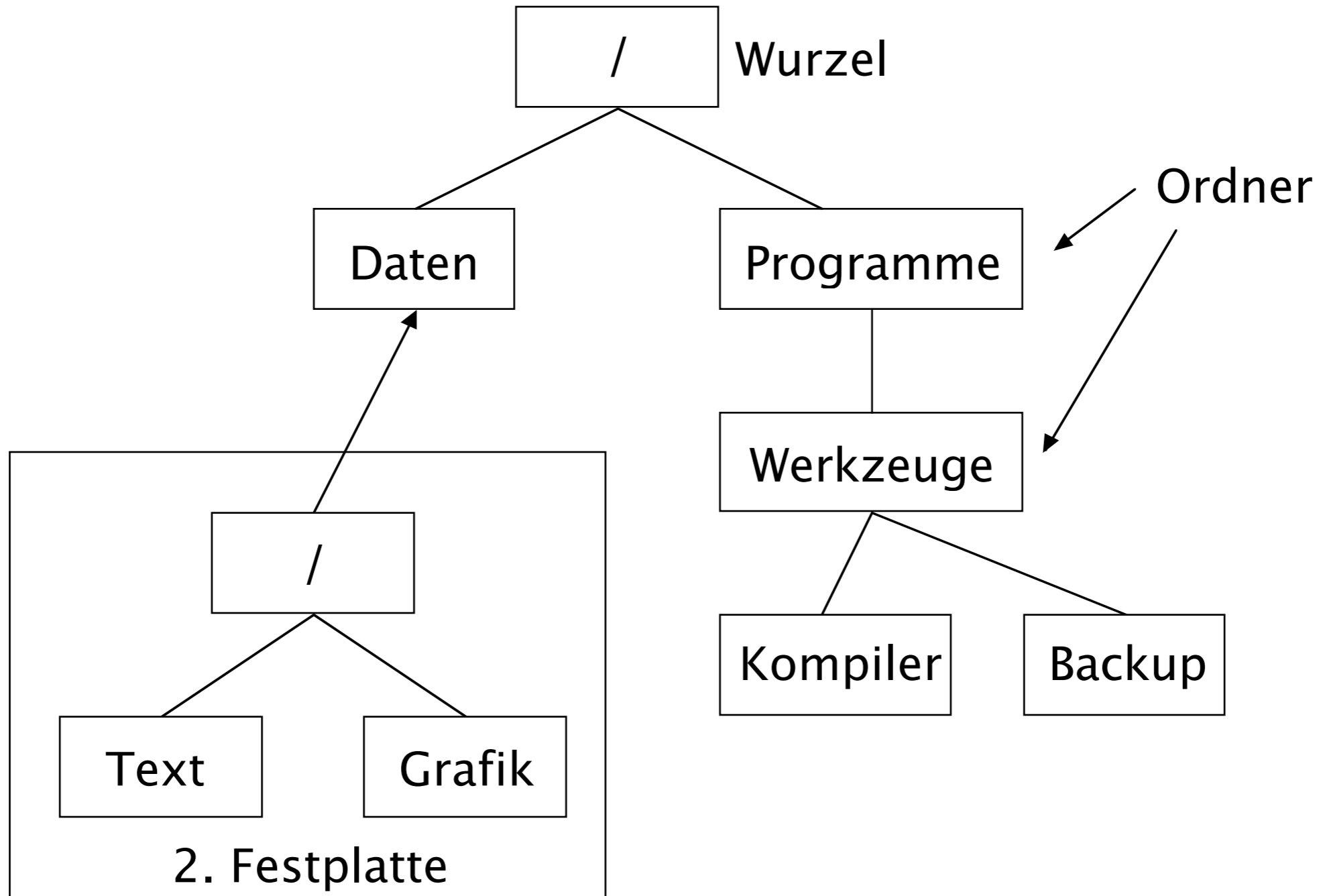
Funktionen für Ein- und Ausgabe

- `nr = open(Pfad, Modus) // Datei öffnen`
- `Fehlercode = close(nr) // Datei schliessen`
- `Anzahl Bytes = read(nr, Speicheradresse, Anzahl Bytes) // Lesen`
- `Anzahl Bytes = write(nr, Speicheradresse, Anzahl Bytes) // Schreiben`
- `Fehlercode = seek(nr, Neue Position) // Position ändern`
- `Fehlercode = create(Pfad) // Neue Datei erzeugen`

Demo

Mounten

- Mehr als ein Laufwerk:
 - Disketten,
 - CD/DVD,
 - freigegebene Ordner im Netzwerk
 - Ram Disk
 - Festplatten
 - Bandlaufwerke
- Laufwerke werden anstelle von Ordnern gemountet.



Dateisystem mit 2. Laufwerk

Navigation im Dateibaum

- Relative Pfade und Absolute Pfade.
- Absolute Pfade beginnen immer mit “/”.
- Für relative Pfade:
 - “.” aktueller Ordner
 - “..” übergeordneter Ordner
- Wechsel mit “chdir” (Systemaufruf) bzw. “cd” (Shell)

Navigation im Dateibaum

- Aktueller Arbeitsordner
(current working directory)
- Homeordner als initialer Arbeitsordner der Shell.
- Beispiel: `/users/cs`
- Bei Pfadangaben wird aus dem aktuellen Ordner und dem Pfad der absolute Pfad berechnet.
- Beispiel: `~/documents` → `/users/cs/documents`

Gliederung

1. Einführung
2. Geschichte
3. Systemaufbau
4. Dateisystem Schnittstelle
- 5. Dateisystem intern**
6. Prozesse
7. Kommunikation zwischen Prozessen
8. Sockets

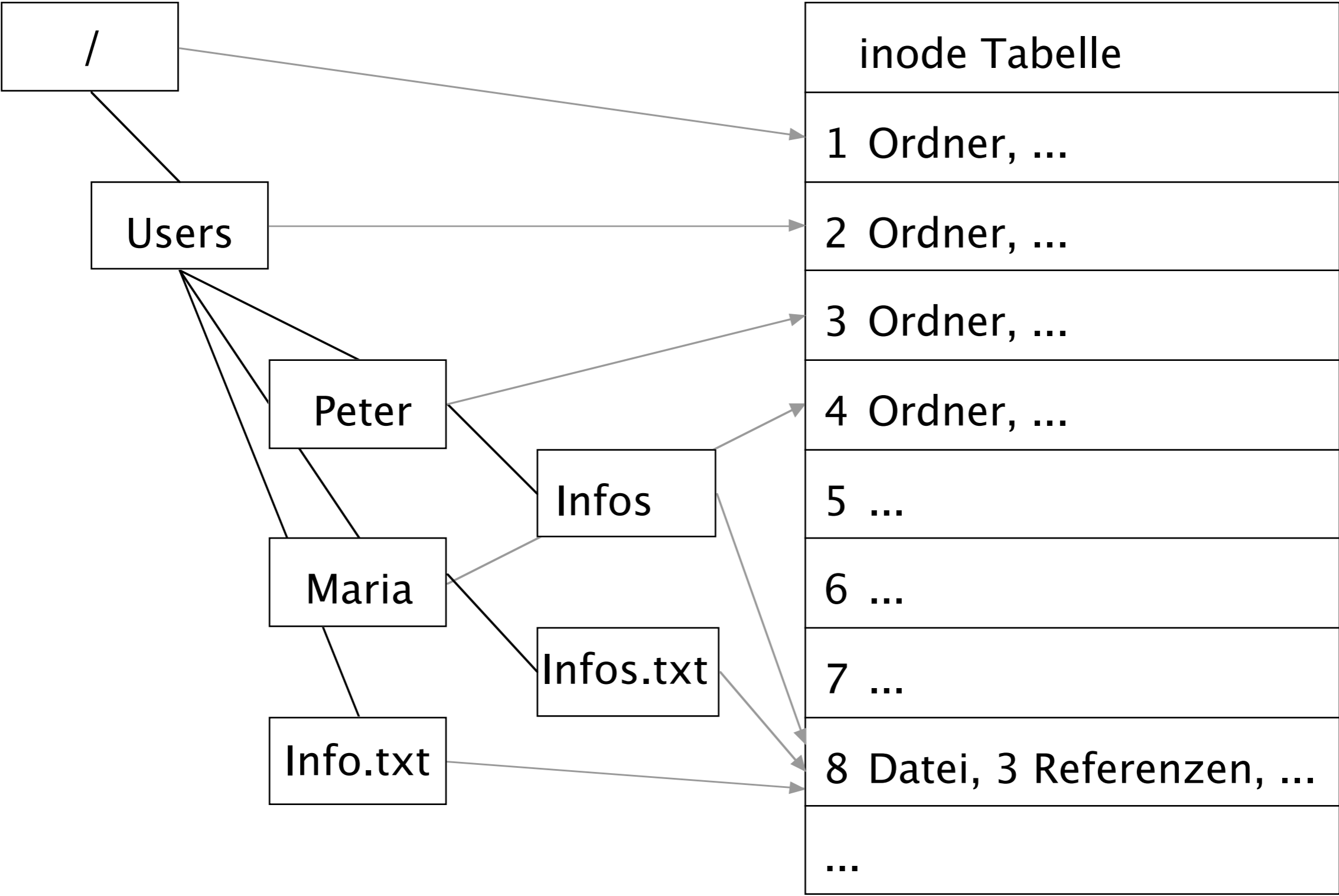
Dateisystem intern

- Jedes Unix hat sein eigenes Dateisystem
 - Linux: ext3
 - Unix: UFS
 - Mac OS X: HFS+
- neuere Systeme lesen auch fremde Dateisysteme

inodes

- Liste der Metadaten
- inode = ein Metadatensatz
- Gespeichert wird u.a. Zugriffsrechte, Dateigröße, belegte Blöcke und ein paar Flags.
- Entweder ist ein inode eine Datei, ein Ordner oder ein Gerät.

Dateihierarchie



Ordner

- Ordner sind speziell gekennzeichnete inodes.
- Inhalt ist eine Liste, die Dateinamen auf inodes abbildet.
- Ein inode kann in mehreren Ordnern enthalten sein.
- Daten zentral gespeichert, aber durchaus verschiedene Dateinamen.

Zugriffsrechte

- Jede Datei hat einen Besitzer und eine Gruppe.
- Zugriffsrechte sind festgelegt für Lesen, Schreiben und Ausführen.
- Verschiedene Zugriffsrechte für Besitzer, für die Benutzer der angegebenen Gruppe und alle anderen Benutzer.

Zugriffsrechte - Beispiele

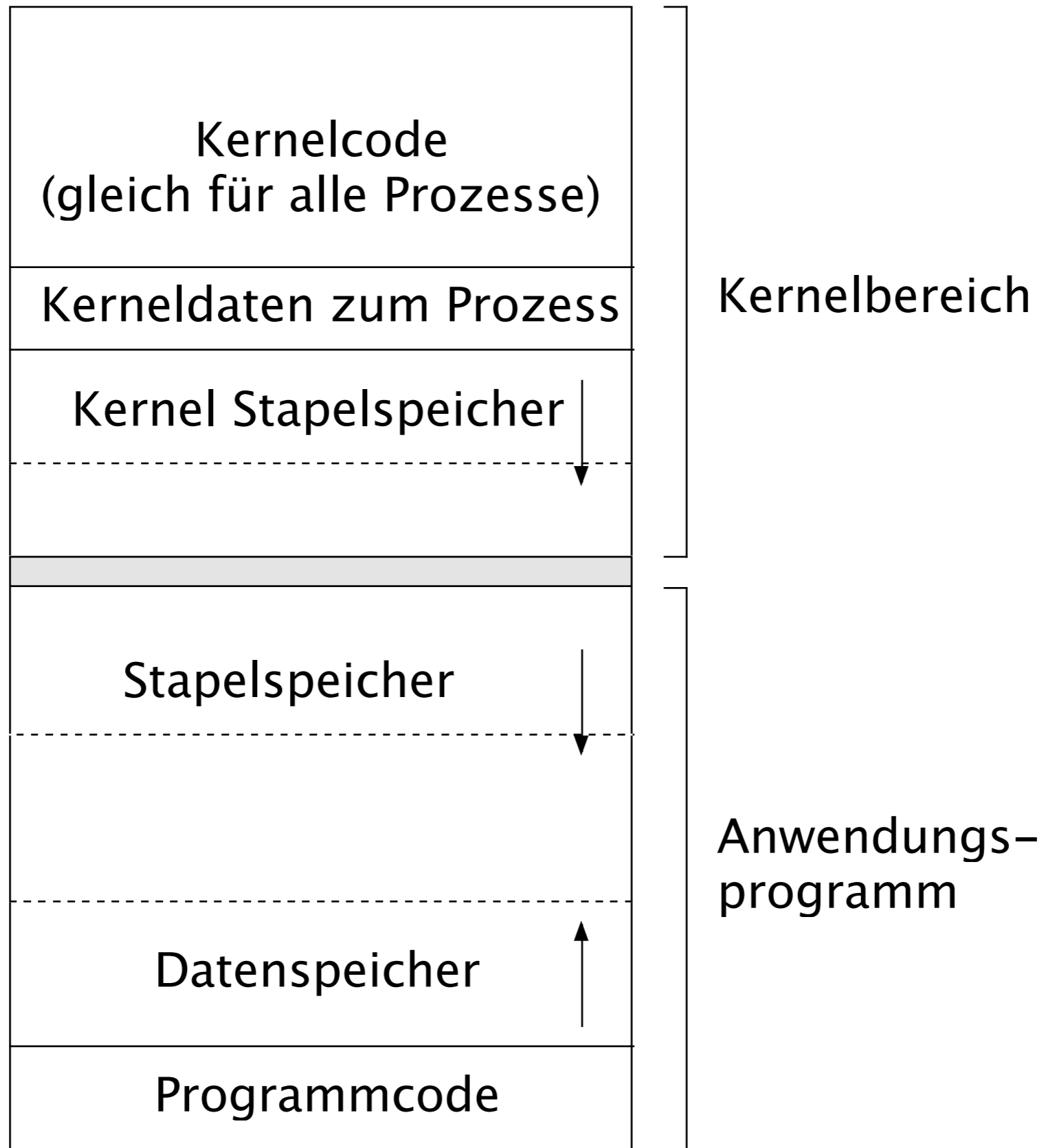
- `rwX rwX rwX` - alle dürfen alles
- `rwX r-- ---` - Besitzer darf lesen, schreiben und ausführen, Gruppe darf nur lesen und alle anderen sind ausgeschlossen.
- `rwX r-x r-x` - Nur der Besitzer darf schreiben.
- `r-s r-s r-x` - Programm darf von allen mit der Identität des Dateibesitzers ausgeführt werden.

Gliederung

1. Einführung
2. Geschichte
3. Systemaufbau
4. Dateisystem Schnittstelle
5. Dateisystem intern
- 6. Prozesse**
7. Kommunikation zwischen Prozessen
8. Sockets

Prozesse

- Die Unix Prozessverwaltung organisiert das Starten und Terminieren von Prozessen.
- Außerdem verteilt der Scheduler die CPU Zeit an alle Prozesse dynamisch.
- Geeignete Mechanismen verhindern, dass Anwendungen sich gegenseitig den Speicher überschreiben.



Prozess erzeugen

- `fork` erzeugt eine Kopie des laufenden Prozesses.
- `execve` lädt ein Programm aus einer Datei in den aktuellen Adressraum.
- Programme ausführen durch `fork` und `execve`.

CPU Scheduling

- Dynamische Verteilung nach Bedarf
- Für jeden Prozess ist eine Priorität einstellbar.
- Prozesse, die Kernelfunktionen ausführen, bekommen mehr CPU Zeit, da das Kernel nicht preemptiv ist.
- Mehr CPU Zeit bekommen Prozesse, die einen Event zu verarbeiten haben.

Swapper

Wenn zu wenig physikalischer Arbeitsspeicher frei ist, kann der Swapper lange nicht benutzte Prozesse auslagern.

Moderne Swapper lagern Speicher in 4 KB Blöcken aus.

Zuerst unbenutzte Speicherblöcke bzw. Blöcke, die schon auf der Festplatte vorhanden sind.

Gliederung

1. Einführung
2. Geschichte
3. Systemaufbau
4. Dateisystem Schnittstelle
5. Dateisystem intern
6. Prozesse
7. **Kommunikation zwischen Prozessen**
8. Sockets

Kommunikation zwischen Prozessen

Möglichkeiten der Kommunikation

- Pipes (Rohre)
- Signale
- gemeinsam benutzte Dateien
- gemeinsam benutzte Speicherblöcke
- Sockets

Pipes

- Nur in eine Richtung.
- Synchrone Datenübertragung.
(Prozess wartet)
- Universell für Dateien und Konsole.
- System puffert zwischen den Prozessen.
- Nur lokal auf einem Computer.
- Beispiel: `cat "Bericht" | grep "London" | less`

Signale

- Signale sind asynchrone Events.
- Anwendungen können Eventhandler beim System anmelden.
- Verarbeitung nur am Ende einer Kernelfunktion.
- Nur ein Signal kann auf die Verarbeitung warten.

Wichtige Signale

- SIGCHILD meldet das Ende eines aufgerufenen Prozesses.
- SIGFPE meldet Division durch 0.
- SIGSEGV bei verbotenen Speicherzugriff.
- SIGKILL für Zwangsterminierung.
- Spezielle Signale für Debugger.

Kommunikation von Anwendungen

- Nur begrenzte Anzahl von Signalen.
- Signale gehen verloren oder werden lange nicht verarbeitet.
- Nur ein Signal kann zwischengespeichert werden.
- Pipes nur in eine Richtung, daher ungünstig.
- Pipes nur synkron. Prozesse werden blockiert.

Kommunikation im Kernel.

- Das Kernel benutzt Signale.
- Gemeinsam benutze Speicherbereiche.
Synkronisation der Zugriffe über
Deaktivierung der Interrupts.
- Schlafende Prozesse werden geweckt um auf
den Status von Ressourcen zu schauen.
(Warten auf freie Ressource)

Gliederung

1. Einführung
2. Geschichte
3. Systemaufbau
4. Dateisystem Schnittstelle
5. Dateisystem intern
6. Prozesse
7. Kommunikation zwischen Prozessen
- 8. Sockets**

Sockets

Für die Kommunikation zwischen Prozessen, die durchaus auf mehrere Computer verteilt sein können, kann man seit der BSD 4.1 Unix Version (ca. 1984) auf die sogenannten Sockets zurückgreifen.

Sockets sind ähnlich den Pipes Datenströme zwischen zwei Prozessen. Allerdings können Daten in beide Richtungen übertragen werden. Je nach Unix Version können verschiedene Protokolle und Adressierungsarten benutzt werden.

Sockets

- Die Socket Funktionen ergänzen die vorhandenen Funktionen zum Lesen und Schreiben.
- Synchrone Funktionen wie read & write.
- Socketspezifische Funktionen send & recv (TCP) und sendmsg & recvmsg (UDP).
- Optional auch asynchron mit Signal SIGIO.

Die Socket Funktionen

- `int socket(int family, int type, int protocol);`
- `int connect(int sockfd, struct sockaddr* serveraddress, int addresslength);`
- `int bind(int sockfd, struct sockaddr* address, int addresslength);`
- `int listen(int sockfd, int backlog);`
- `int accept(int sockfd, struct sockaddr* clientaddress, int addresslength);`

Demo

Zusammenfassung

Klassisches Unix ist recht einfach gehalten, da die ursprünglichen Rechner nur eine CPU hatten und kein preemptives Multitasking konnten.

Jedesmal wenn die Entscheidung zwischen Effizienz und Einfachheit stand, wurde der einfache Weg genommen.

Trotzdem ist Unix vor allem ein universelles und modular erweiterbares Betriebssystem.

