Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
  - Benutzer und Benutzergruppen
  - Dateien / Files
  - Ordner / Directories
  - Prozesse
  - Benutzeroberfläche
  - Virtualisierte Ressourcen
- Rechnernetzwerke

Betriebssysteme

- Der ”Makler“ zwischen
  - Hardware einerseits und
  - Anwendungsprogrammen und Endbenutzern andererseits
- Kontrolliert die Datenverwaltung, Benutzerverwaltung, die I/O–Schnittstellen sowie den Ablauf aller Prozesse.

Beachte:

- Die diversen Betriebssysteme sehen auf den ersten Blick sehr unterschiedlich aus.
- Die Unterschiede sind auf den zweiten Blick aber nicht mehr so groß.
- Im folgenden lehnen wir uns trotzdem an eine konkrete Familie von Betriebssystemen an: *UNIX*.

Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
  - Benutzer und Benutzergruppen
  - Dateien / Files
  - Ordner / Directories
  - Prozesse
  - Benutzeroberfläche
  - Virtualisierte Ressourcen
- Rechnernetzwerke

Benutzer und Benutzergruppen

- Jeder Arbeitsbereich ist einem virtuellen *Benutzer* (*User*) zugeordnet.
  - Ein Benutzer erhält Rechnerzugang über die Einrichtung eines Accounts und die Weitergabe des zugehörigen Passwortes an diese Funktionseinheit.
- In einer *Group* sind mehrere User zu einer als Ganzes mit einem Gruppennamen ansprechbaren Einheit zusammengefasst.
  - Groups dürfen sich beliebig überlappen.
  - Jeder User gehört genau einer der Gruppen, denen er angehört, besonders fest zu.
  - Diese Gruppe nennt man die *Primärgruppe* des Users.
  - Alle anderen Gruppen, denen der User angehört, nennt man seine *Sekundärgruppen*.

UNIX-Attribute von Usern

- Benutzerkennung (*Login–Name*).
- Momentanes Passwort.
- *Home Directory*: der zugeordnete Arbeitsbereich
- Die *Primärgruppe*.
- Beliebig viele *Sekundärgruppen* (auch keine einzige möglich).

Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
  - Benutzer und Benutzergruppen
  - ➔ Dateien / Files
  - Ordner / Directories
  - Prozesse
  - Benutzeroberfläche
  - Virtualisierte Ressourcen
- Rechnernetzwerke

Dateien / Files

- Alle Daten sind in UNIX in Form von *Files* (*Dateien*) abgelegt.

- Programme, Mailboxes, WWW–Seiten, Bilder, Filme, Tonspuren etc. sind alles Files.

- Beispiel:
  Mail–Reader sind nichts anderes als Editoren für Files, deren Inhalt nach bestimmten Konventionen formatiert sind:

  Eine Sequenz von Zeilenblöcken (eben den E-Mails), die jeweils aus einem (nach strengen Regeln formatierten) *Header* (Kopf) und dem eigentlichen Inhalt der E-Mail (*Body*) bestehen.

Beispiel: Mail-Datei

Header

```
From mw@allgemeineinformatik.de
Date: Fri, 26 Sep 2003 14:08:52 +0200
Subject: Testmail fuer das Skript
From: Markus Weimer <mw@allgemeineinformatik.de>
To: Markus Weimer <mw@allgemeineinformatik.de>
Message-ID: <BB99F8F4.1FE7>
Content-type: text/plain; charset="ISO-8859-1"
Content-Transfer-Encoding: 8bit
```

Body

```
Hallo,

dies ist eine Testmail fuer
das Skript in Allgemeine Informatik.

Viel Spass in der Vorlesung,

Markus Weimer
```

Reguläre Ausdrücke

- Engl. Name für reguläre Ausdrücke: *Regular Expressions*.

Mit dem UNIX–Kommando grep kann man in Dateien (*Files*) nach Zeichenketten (*Strings*) suchen.
Das Kommando ”grep“ steht für *global regular expression print.*

Beispiel: Alle Zeilen mit Zeichenkette ”class“ in einem File namens ”Trial.java“ erhält man mit dem Kommando

```
grep ¨class¨ Trial.java
```

- Man kann auch ”unscharfe“ Anfragen stellen, zum Beispiel

```
grep ¨cl.ss¨ Trial.java
grep ¨clas*¨ Trial.java
grep ¨c[j-m]ass¨ Trial.java
grep ¨c[:lower]ass¨ Trial.java
```

Beispiele

- ”`cl.ss`“:

  Alle fünfbuchstabigen Worte mit „`cl`“ vorne und “`ss`“ hinten.
  (Auch mit einem Großbuchstaben, einer Ziffer oder einem Sonderzeichen als drittem Zeichen.)

- ”`clas*`“:

  ”`clas`“ und ”`class`“ und ”`classs`“ und ”`classss`“ und ”`classsss`“ und ...
  aber auch ”`cla`“

- ”`c[j-m]ass`“:

  ”`cjass`“ und ”`ckass`“ und ”`class`“ und ”`cmass`“.

- ”`c[:lower]ass`“:

  ”`caass`“ und ”`cbass`“ und ”`ccass`“ und ... und ”`cxass`“ und ”`cyass`“ und `czass`“.

Allgemeine Regeln

- Eine Zeichenfolge ohne die Zeichen ”.“, ”*“ und ”[“ (und noch ein paar weitere, hier nicht erwähnte) steht einfach für sich selbst.
- Ein ”.“ in einem Wort steht für jedes beliebige Zeichen.
- Ein ”*“ in einem Wort, aber nicht am Anfang des Wortes, steht für beliebig viele Vorkommen des unmittelbar vorhergehenden Zeichens.

  (Am Wortanfang steht ”*“ für sich selbst.)
- Die Zeichen innerhalb eines Paares von eckigen Klammern ”[...]“ haben besondere Bedeutung (hier nicht näher ausgeführt, siehe die Beispiele auf der letzten Folie).
- Usw. → `man grep`

`man` steht für „Manual“.
Damit erhalten sie eine genaue Beschreibung eines UNIX-Befehls.

Verwendung von „\“

- ”\“ vor einem Sonderzeichen (wie bspw.”*“) nimmt dem Sonderzeichen die Sonderbedeutung.

→ Mit Text `cl\*ss` wird gezielt nach ”cl*ss“ gesucht.

- ”\“ vor einem anderen Zeichen wird verschluckt:

→ `cl\ass` steht für das einzelne Wort ”class“.

- Dem Zeichen ”\“ selbst nimmt man genauso mit einem vorangestellten ”\“ die Sonderbedeutung.

→ `cl\\ss` steht f ür das einzelne Wort ”cl\ss“.

- *Insgesamt*: Mit einigen solcher Sonderregeln und Sonderzeichen (hier nicht näher ausgeführt) wird eine sehr große Ausdrucksmächtigkeit erreicht.

UNIX-Attribute von Files

- Besitzer (ein User).
- Besitzende Gruppe
  (die primäre oder eine sekundäre Group des besitzenden Users).

  Drei *Zeitstempel* (*Time Stamps*):
  - *Access Time*: Zeitpunkt des letzten lesenden oder schreibenden Zugriffs (der spätere von beiden).
  - *Modification Time*: Zeitpunkt des letzten schreibenden Zugriffs.
  - *Status Change Time*: Zeitpunkt der letzten Änderung der Attribute des Files.
- Zugriffsrechte
- ...

Zugriffsrechte für Files

- Einteilung 1: lesen, schreiben, ausführen.
- Einteilung 2: Besitzer (*Owner*), besitzende Gruppe, Rest der Welt.

→ Es gibt insgesamt 3x3 = 9 Freiheitsgrade, um einzelnen Benutzerkreisen Rechte zu geben bzw. zu nehmen.

Ausführrecht:

- Ist zwar für jedes File definiert,
- ist aber sinnlos für Files, die nicht Programme o.ä. sind.
- *Bedeutung*:
  - Das File darf als Programm vom betreffenden Benutzerkreis ausgeführt werden.

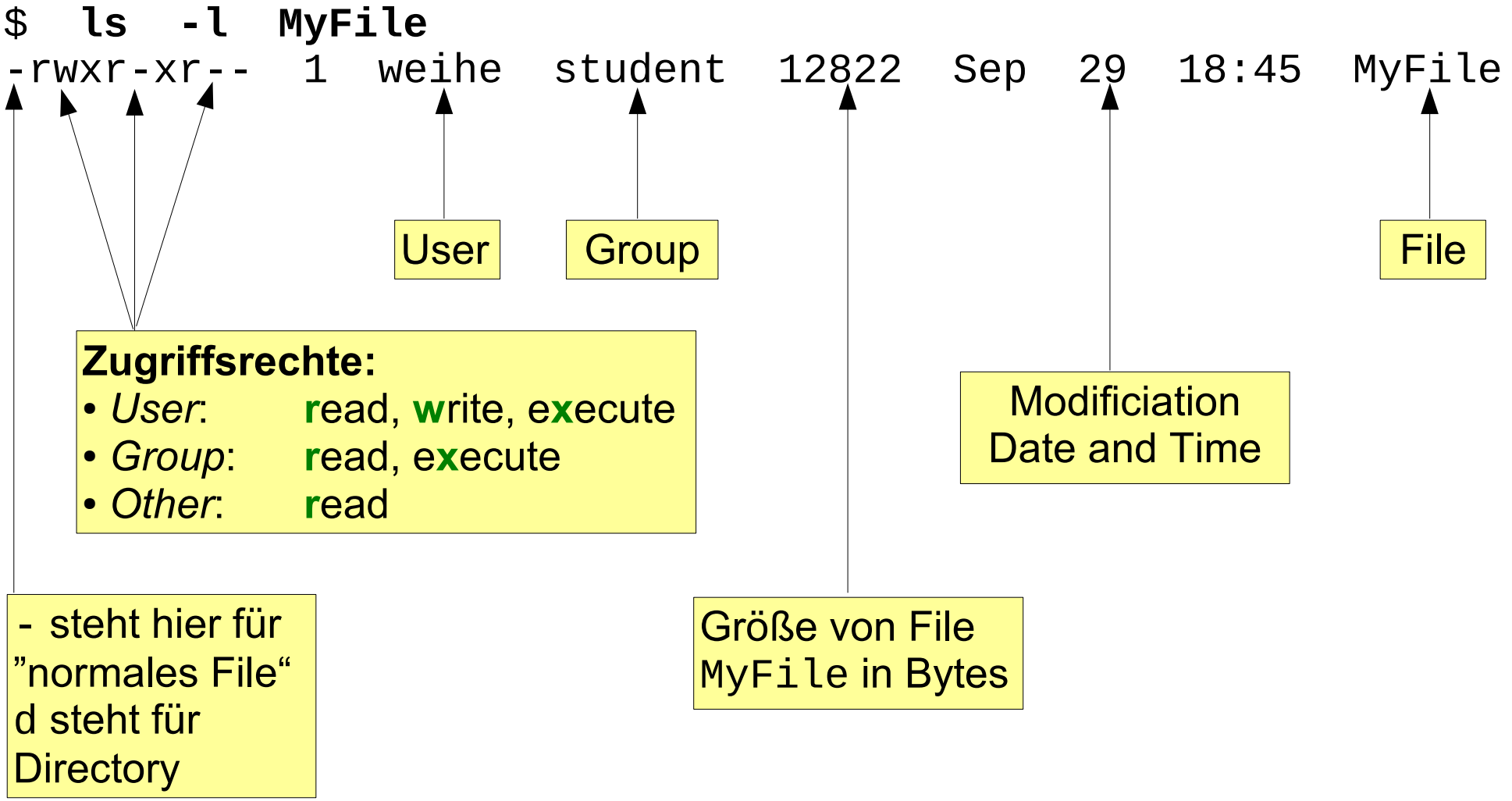
Anzeigen von File-Attributen

```
$  ls  -l  MyFile
-rwxr-xr--  1  weihe  student  12822  Sep  29  18:45  MyFile
```

Erläuterungen zu "`ls -l`" allgemein:

- Das Kommando "`ls`" ist in erster Linie dafür da, sich den Inhalt von Directories anzeigen zu lassen.
- Einige Optionen von "`ls`" sind dafür da, um festzulegen, was alles an Informationen für die einzelnen Files ausgegeben werden soll.
- Durch Option "`-l`" (=long) wird festgelegt, dass ein paar Standardinformationen ausgegeben werden, die in den meisten Fällen alle Information enthalten, die man sucht.
- Weitere Informationen finden Sie nach Eingabe von `man ls` bzw. `info ls` an einem UNIX-Rechner.

Anzeigen von File-Attributen



Erklärung Zugriffsrechte

- Ein ”r“=”read“,”w“=”write“ oder ”x“=”execute“ besagt, dass das jeweilige Recht gegeben ist.
- Steht statt dessen ein ”–“ an dieser Position, ist das jeweilige Recht statt dessen genommen.
- Zuerst kommen die drei Rechte des Owners, dann die drei Rechte der besitzenden Group, schließlich die drei Rechte für den Rest der Welt.
- In jedem der drei Tripel kommt immer zuerst Leserecht (”r“), dann Schreibrecht (”w“), schließlich Ausführrecht (”x“).

→ Die Anzeige ”rwxr-xr--“ besagt also, dass der User alle Rechte hat, die Group nur Lese- und Ausführrecht und der Rest der Welt nur Leserecht.

Ändern der Zugriffsrechte

- `chmod <options> <files>`
  - Die Optionen gibt man in Form einer Zeichenkette an, die aus folgenden Teilen besteht
    - `ugoa`: Änderungen für User, Group, Other, All
    - `+-`: Hinzufügen oder Wegnehmen von Rechten
    - `rwx`: Read, Write, Execute-Rechte
  - Beispiele:
    - `chmod a+r file`: Alle user erhalten Leserechte
    - `chmod u+rwx file`: Der Benutzer gibt sich alle Rechte
    - `chmod go-wx file`: Allen anderen nimmt er Schreib- und Ausführrecht
- `chgrp <new-group> <files>`
  - Ändern der Gruppe eines Files
- `chown <new-owner> <files>`
  - Ändern des Owners eines files (für Administrator / root)

Anmerkung: Nur der Besitzer eines Files darf Rechte ändern!

Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
  - Benutzer und Benutzergruppen
  - Dateien / Files
  - ➔ Ordner / Directories
  - Prozesse
  - Benutzeroberfläche
  - Virtualisierte Ressourcen
- Rechnernetzwerke

Ordner / Directories

- Files sind hierarchisch in *Verzeichnissen* (*Directories*) organisiert.
- *Hierarchisch* heißt: Zum Inhalt einer Directory können
  - ◊ nicht nur Files gehören,
  - ◊ sondern auch wieder Directories (die unmittelbaren *Subdirectories* der Directory).
- *Pfad*: Jedes File und jede Directory ist das Ende eines *Pfades* von Directories, der mit der ”Root–*Directory*“ beginnt.
- Der Name eines Files (oder einer Directory)
  - ◊ ist allgemein nicht aus sich heraus eindeutig,
  - ◊ aber zusammen mit dem Pfad ist er es.

Pfade

- Der `/` (”*Slash*“) wird als Trennsymbol verwendet:
  - zwischen dem Namen einer Directory und dem Namen einer ihrer unmittelbaren Subdirectories,
  - zwischen dem Namen einer Directory und dem Namen eines Files in dieser Directory.
- Der Slash leitet auch den Namen eines Pfades ein.
- Die Root–Directory heißt einfach nur `/`.
- *Sondernamen*:
  - `~weihe` steht für den absoluten Pfad der Home Directory von User ”weihe“.
  - `~` allein steht für den absoluten Pfad der eigenen Home Directory.

Relative Pfade

- Eine *Shell* (d.h. der Prozess in einem xterm–Fenster) hat zu jedem Zeitpunkt genau eine *Working Directory*.

  → Eigentlich jeder Prozess, siehe später

- Der Name eines Files oder einer Directory kann auch relativ zur Working Directory angegeben werden.

- Die Working Directory kann in der Shell immer durch ”.“ angesprochen werden.

- Die Directory, die in der Hierarchie genau eine Stufe über der Working Directory steht, wird durch ”..“ angesprochen.

UNIX-Attribute von Directories

- Praktisch dieselben Attribute wie bei Files und mit identischer Bedeutung.
- *Ausnahme*: Zugriffsrechte haben bei Directories eine eigene, aber (halbwegs) analoge Bedeutung.
- *Genauer*:
  - *Leserecht*: Man darf sich den Inhalt der Directory (also die darin unmittelbar enthaltenen Files und Subdirectories) mit Kommandos wie ”ls“ anzeigen lassen.
  - *Schreibrecht*: Man darf Files und Directories in die Directory hineinstellen oder daraus entfernen.
  - *Ausführrecht*: Man darf die Directory (oder eine Subdirectory) mit Kommandos wie ”cd“ zur Working Directory machen.

Directories als Files

```
$  cd  ~weihe
$  ls -l
...
drwx------ 2  weihe  algo  1024 Feb  1  1999  .elm
...
```

Erläuterung:

- Es ist kein Zufall, dass Directories dieselben Attribute wie Files haben und durch ”`ls`“ gleich behandelt werden.
- Directories sind in UNIX ebenfalls im Grunde nur Files, aber von anderer Art als ”normale“.
- *Kenntlichmachung* bei ”`ls -l`“: In der Zeile zu einer Directory steht ein ”d“ anstelle eines ”-“ als erstes Zeichen.

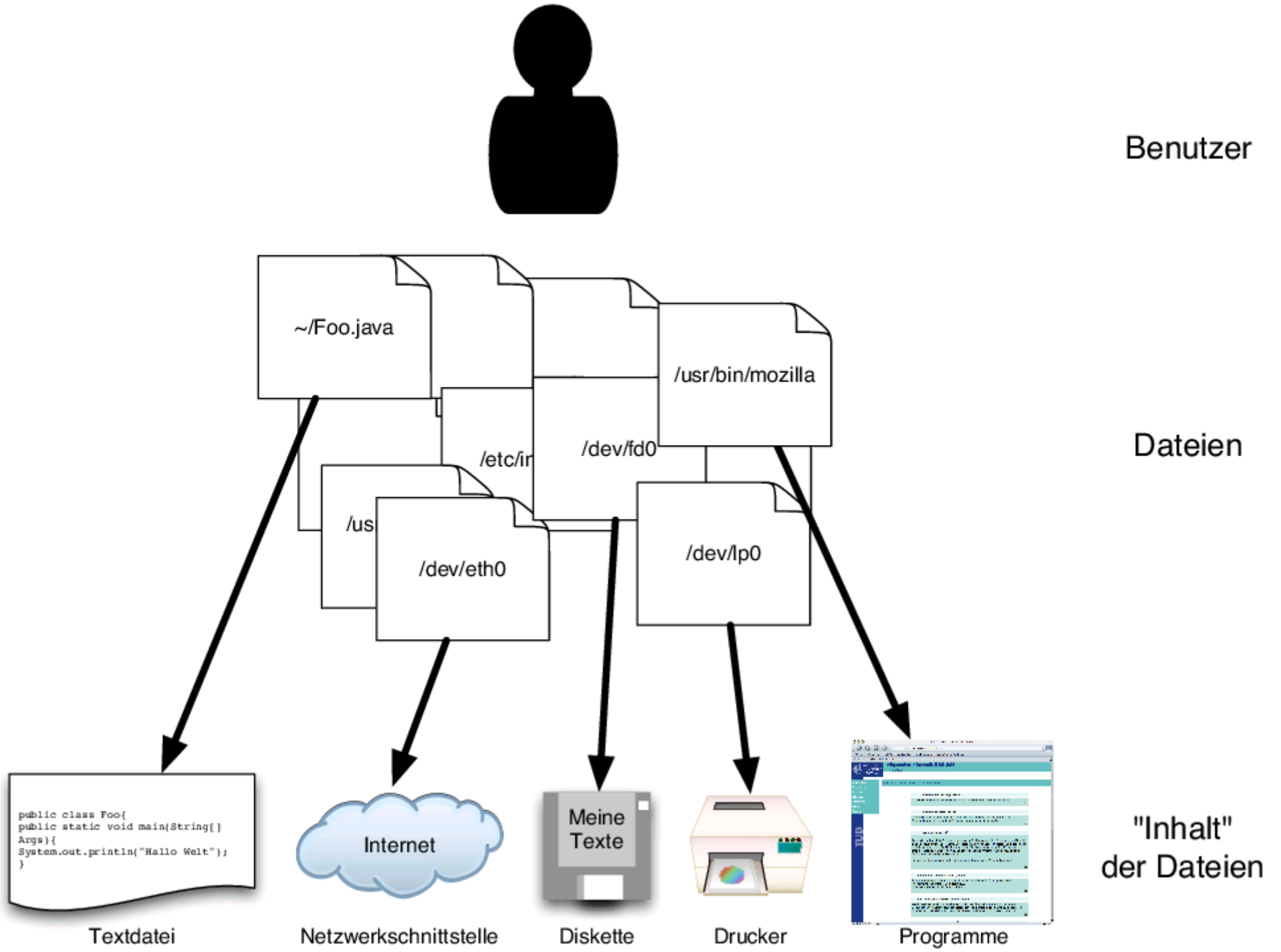
Abstrakte (Special) Files in UNIX

- Hinter der letzten Folie steht eine allgemeine UNIX–Philosophie: ”Alles ist File“.
- *Das heißt*: Das UNIX–Konzept ”File“ bündelt diverse Systemkonzepte, die
  - ◊ auf den ersten Blick überhaupt nichts miteinander zu tun zu haben scheinen,
  - ◊ auf den zweiten Blick aber so viel gemeinsam haben (z.B. Attribute), dass sie weitgehend gleich behandelt werden können.

→ Beispiel für das allgemeine Bestreben in der Informatik, Details soweit wie möglich ”wegzuabstrahieren“ und die ”reinen“ Konzepte herauszukristallieren.

Beispiele für Abstrakte Files

- Ein *Character Special File* ist die Repräsentation eines zeichenorientierten Ein-/Ausgabegeräts, mit der es ein UNIX–Nutzer zu tun bekommt, wenn er systemnah mit dem Gerät arbeiten will, zum Beispiel:
  - ◊ Von Tastatur lesen heißt, aus einem bestimmten Character Special File lesen, das in einer bestimmten Directory steht.
  - ◊ Auf den Bildschirm (xterm–Fenster) schreiben heißt, auf ein bestimmtes Character Special File in einer bestimmten Directory schreiben.
- Ein *Block Special File* repräsentiert ein Ein-/Ausgabegerät, das immer gleich ganze Blöcke fester, spezifischer Größe von Zeichen einliest und/oder ausgibt (z.B. Diskettenlaufwerk).
- Ein *Socket* ist ein File, in das ein Prozess hineinschreibt, und aus dem ein anderer Prozess herausliest.


Benutzer
~/Foo.java
/usr/bin/mozilla
/etc/ir
/dev/fd0
/us
/dev/eth0
/dev/lp0
Dateien
public class Foo{
public static void main(String[]
Args){
System.out.println("Hallo Welt");
}
Internet
Meine
Texte
"Inhalt"
der Dateien
Textdatei
Netzwerkschnittstelle
Diskette
Drucker
Programme

Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
  - Benutzer und Benutzergruppen
  - Dateien / Files
  - Ordner / Directories
  - ➔ Prozesse
  - Benutzeroberfläche
  - Virtualisierte Ressourcen
- Rechnernetzwerke

Prozesse

- Ein *Programm* ist eine Folge von Instruktionen in einer für den Computer ausführbaren Form.

  → Abgespeichert als ”normales“ File.

- Ein *Prozess* ist die Ausführung eines Programms.

  → Mehrere Prozesse können gleichzeitig, aber völlig unabhängig voneinander auf demselbem Rechner dasselbe Programm ausführen.

  → Genauso kann ein Benutzer gleichzeitig verschiedene Programme (i.e., Prozesse) ausführen lassen (haben Sie alle schon gemacht, wenn Sie verschiedene Fenster gleichzeitig offen hatten).

Prozessmanagement

- Ein Benutzer kann (fast) beliebig viele Prozesse gleichzeitig laufen lassen.
- Es können auch mehrere Benutzer zugleich auf demselben Computer (aber normalerweise natürlich über verschiedene Terminals) eingeloggt sein und Prozesse laufen lassen.
- Zugleich laufen noch zentrale Dienste für alle Benutzer wie z.B. der Mail–Server oder der Drucker-Spooler.
- *Aber*: Die meisten ”normalen“ Computer haben nur eine CPU (oder einige wenige).

Frage: Wie passt das zusammen?

Einfache Methode

- Das Betriebssystem teilt jedem Prozess reihum eine kurze Zeitspanne auf der CPU zu.
- Die Zeitspannen sind so kurz, dass der Benutzer am Terminal nicht mitbekommt, dass die Prozesse in Wirklichkeit nicht parallel, sondern *serialisiert* ablaufen.

  → Außer bei sehr hoher Auslastung, wenn es zu Engpässen kommt.
- Die CPU–Zeit wird dabei möglichst ”gerecht“ unter den Prozessen verteilt.
- Wartet ein Prozess auf ein Ereignis (z.B. Benutzereingabe), wird er bis zum Eintreten des Ereignisses aus der Warteschlange genommen.
- Prozesse sind mit unterschiedlichen *Prioritäten* ausgestattet.

  → Prozesse mit höherer Priorität werden öfter und/oder länger bedient.

Priorität von Prozessen

- Alle Prozesse eines normalen Users werden normalerweise mit einer bestimmten Standardpriorität gestartet.

→ Ungefähr ”mittlere“ Priorität.

- Unter UNIX ist das Kommando `nice` dafür da, einen Prozess mit anderer als der Standardpriorität zu starten.

- Abgesehen von den Administratoren darf jeder User die Priorität der von ihm gestarteten Prozesse aber immer nur verringern, nie erhöhen.

nice

```
nice  -n15  nedit myprogram.java
```

- Ein Programm namens ”`nedit`“ soll mit dem Argument ”`myprogram.java`“ gestartet werden.
- Durch vorangestelltes ”`nice`“ wird dieser Prozess nicht direkt, sondern indirekt durch das Programm ”`nice`“ aufgerufen.
- Durch Option ”`-n15`“ zu ”`nice`“ wird die Priorität von ”`nedit`“ um 15 *vermindert*.
- Man kann die Werte 1 ...19 hinter ”`-n`“ einsetzen.
- `nice` ohne Argumente gibt die momentane Priorität wieder.

→ Vergleiche Sie die Ausgabe von `nice` und `nice -n9 nice`

Why be nice?

Warum soll man auf Priorität verzichten?

- Umfangreichere Rechenprozesse können Stunden, Tage oder Wochen benötigen.
  - ◊ Auf ein bisschen Zeit mehr oder weniger kommt es dabei dann auch nicht an.
- Es wäre also nett (engl. ”nice“), wenn der Prozess wenig oder keine Rechenzeit beanspruc ht, während irgendein User interaktiv mit dem Rechner arbeitet.
- Das geht sehr gut durch eine geringere Priorität:
  - ◊ Solange andere Prozesse mit höherer Priorität laufen, bekommt dieser Prozess kaum Rechenzeit zugeteilt.
  - ◊ Ansonsten kann er den Rechner sehr stark für sich beanspruchen.
- Wann kommt der Prozess dann vorwärts:
  - ◊ Wenn praktisch niemand arbeitet (z.B. frühes Morgengrauen).
  - ◊ Aber durchaus auch zwischendurch, zum Beispiel wenn ”konkurrierende“ interaktive Prozesse auf Benutzereingaben warten und in dieser Zeit nichts tun.

Attribute von Prozessen

- Besitzer (ein User)
- Eindeutige Zahlenkennung (*Prozess–ID*)
- Bisher verbrauchte Rechenzeit
- Priorität
- Momentanes Working Directory
- Status des Prozesses:
  - *Running*: Arbeitet gerade (und verbraucht dabei Rechenzeit).
  - *Runnable*: In der Warteschlange.
  - *Sleeping*: Wartet auf einen Event.
  - *Zombie*: Prozess sollte eigentlich schon beendet sein, ist dem Betriebssystem aber aus dem Ruder gelaufen und läuft (mehr oder weniger unkontrollierbar) immer weiter.

Prozesse Starten

Stark vereinfacht:

- Nach dem Hochfahren (”Booten“) eines Computers startet zunächst einmal ein Hauptprozess.

- Jeder Prozess kann seinerseits (fast) beliebig neue Prozesse erzeugen (”Kind–Prozesse“).

  → Alle Prozesse entstehen also direkt oder indirekt aus dem Hauptprozess.

- Ein Elter–Prozess ”vererbt“ seinen Kind–Prozessen einige Attribute (z.B. Working Directory).

- Der Kind–Prozess kann diese Attribute auch durch eigene Setzungen überschreiben.

Shell

- Der Prozess, der in einem `xterm`–Fenster läuft.
- Dient in erster Linie der Kreation neuer Prozesse.

  → Durch Aufruf des zugrundeliegenden Programms als Kommando an die Shell.
- Die Shell ist der Elter–Prozess jedes Prozesses, der durch Kommandoaufruf über diese Shell gestartet wurde.
- Ein so gestarteter Prozess kann dann ebenfalls auf dieses xterm–Fenster Ausgaben schreiben.

Prozesse im Vordergrund oder im Hintergrund starten

- Ä*ußerliches Merkmal*: Man startet einen Prozess in der Shell ”im Hintergrund“, indem man an den Kommandoaufruf ein ”&“ anhängt:
  - `nedit MyFile`
  - `nedit MyFile &`
- *Technischer Unterschied*:
  - Wird der Prozess im Vordergrund gestartet, wird das `xterm`-Fenster dem gestarteten Prozess übergeben.
  - Ansonsten bleibt das `xterm`–Fenster im Besitz des Shell-Prozesses.
- *Konsequenz*: Die Eingaben, die man in einem `xterm`–Fenster mit Tastatur und Maus macht, gehen
  - im ersten Fall an den gestarteten Prozess,
  - im zweiten Fall an die Shell.

Working Directory des Prozesses

- Ein Prozess startet typischerweise mit der Working Directory seines Elter–Prozesses.
- Eine Shell startet normalerweise mit der Home Directory des Besitzers des Shell–Prozesses als Working Directory.
- Typische Regel bei Editoren:
  - ◊ Wenn der Editor mit einem Filenamen als Kommandozeilen-argument aufgerufen wird, wird die Directory dieses Files die Working Directory des Editor–Prozesses.
    Beispiel: `nedit /a/b.txt`
    Dann ist `/a/` das Working Directory des Editors.
  - ◊ Wenn der Editor ohne einen Filenamen aufgerufen wird, erbt er seine Working Directory vom Elter–Prozess (meist eine Shell).
- Ändern des Working Directories: mit „`cd <Pfad>`“

Daemons

- Ein *Daemon* (sprich: dih–men mit extrem kurzem ”e“) ist ein Prozess, der ununterbrochen über längere Zeitspannen existiert und ”im Hintergrund“ auf Arbeit wartet.
- Vor allem *Server*–Prozesse: Mail–Server,WWW–Server,...
- Beispiel Mail–Server (vereinfacht):
  - Mails von lokalen Usern und von außen werden nach strikten Formatierungsregeln zunächst in eindeutig definierten Adressbereichen abgelegt.
  - Wann immer der Mail–Server im Status *running* ist, fragt er den Inhalt dieser Bereiche ab.
  - Falls neue E-Mails da sind, werden sie vom Mail–Server bearbeitet und in die Mailboxes der jeweiligen Adressaten kopiert.

UNIX-Interprozesskommunikation I: Signale

Ein Prozess kann einem anderen Prozess ein *Signal* schicken.

- Mit Kommando ”`kill`“ schickt die Shell, in der das Kommando aufgerufen wurde, das spezifierte Signal an den Prozess, dessen Prozess–ID spezifiziert wurde.
- *Beispiele* für Signale: ”CONT“, ”HUP“, ”KILL“, ”STOP“, ”TERM“.
- *Vorgehensweise*:
  - ◊ Mit dem UNIX–Kommando ”`ps`“ lässt man sich Informationen zu den momentan auf dem Rechner laufenden Prozessen anzeigen.
  - ◊ Aus diesen Anzeigen sucht man die ID des Prozesses heraus, dem man das Signal schicken will.
  - ◊ Das Signal schickt man durch Aufruf von ”`kill`“ mit dem Signal als Option (Minus vor dem Signalnamen) und der Prozess–ID als Argument.

Beispiel

```
[bellman:/Skript]  weimer%  ps

PID  TT  STAT  TIME  COMMAND
469  std  Ss  0:00.10  -csh  (tcsh)
486  std  S   0:01.48  xdvi  folien  master.dvi
2311  p2  Ss+ 0:00.08  -csh  (tcsh)

[bellman:/Skript]  weimer%  kill  -HUP  486
```

Dies sendet das Signal ,,HUP" an den Prozess mit der Nummer „486", in diesem Fall läuft das Programm „xdvi" mit dieser Prozess-ID.

Signalbehandlung

- Für jedes mögliche Signal ist eine *Default*-Reaktion festgelegt, zum Beispiel
  - Abbruch des Prozesses bei ”*HUP*“ (*Hangup*), ”*KILL*“ und ”*TERM*“ (*Terminate*),
  - einschlafen bei ”*STOP*“,
  - wieder aufwachen bei ”*CONT*“ (*Continue*).
- Ein Programm kann auch so geschrieben sein, dass ein Prozess, der dieses Programm ausführt, auf einzelne Signale anders reagiert.

  → Wird hier nicht weiter ausgeführt.
- *Sinnvolles Beispiel*: Ein Editor reagiert auf ”*HUP*“, indem er die letzten, noch nicht abgespeicherten Modifikationen des Fileinhalts in einer Sicherheitskopie abspeichert — und sich dann selbst beendet.

Signalbehandlung (2)

- Einige wenige dieser Signale (z.B. Signal KILL) lassen sich von Prozessen grundsätzlich nicht abfangen.

  → Hierarchie zwischen ”weichen“ und ”harten“ Signalen zum Abbruch.

- Auf den abfangbaren Signalen zum Abbruch gibt es ebenfalls eine (rudimentäre) Hierarchie durch eine Konvention, welche Signale wie ”weich“ von Prozessen abgefangen werden sollten, z.B.:
  - HUP wird als ”weich“ interpretiert und von vielen Prozessen zum”Aufräumen“ abgefangen, bevor sich der Prozess dann von selbst beendet.
  - TERM wird als eher ”hartes“ Signal interpretiert und von vielen Prozessen gar nicht oder mit weniger weichen Reaktionen abgefangen.
- Beim Ausloggen sendet das Betriebssystem das Signal HUP zum Abbruch an alle Prozesse des Users

UNIX-Interprozesskommunikation II: Pipes

- Mit ”|“ läßt sich aus zwei angegebenen Prozessen ein dritter Prozess bilden, der die Ausgabe des ersten Prozesses als als Eingabe für den zweiten Prozeß verwendet.

- *Das heißt*:

  ◊ Die Ausgaben des ersten Prozesses, die sonst auf den Bildschirm geschrieben worden wären, werden statt dessen in einem Zwischenpuffer abgespeichert.

  ◊ Die Eingaben für den zweiten Prozess, die sonst von der Tastatur kämen, kommen nun aus diesem Zwischenpuffer.

Beispiele für Pipes

- `ls | wc -w`
  - ◊ ”`wc -w`“ zählt die Wörter in einem Text.
  - ◊ Diese Pipe zählt also die Files in der Working Directory.
- `ls -l | grep java`
  - ◊ ”`grep java`“ sucht aus einem Text alle Zeilen heraus, in denen die Zeichenkette ”`java`“ vorkommt.
  - ◊ Diese Pipe gibt die ”`ls -l`“–Zeilen aller Files in der Working Directory aus, deren Namen die Zeichenkette ”`java`“ enthält.
- `ps | grep nedit`
  - ◊ Diese Pipe zeigt also Informationen zu allen Prozessen an, die mit dem Programm ”`nedit`“ gestartet wurden.

UNIX-Interprozesskommunikation III: Sockets

- Ein ”Kommunikationskanal“ zwischen zwei Prozessen, über den der eine Prozess Daten vom anderen bekommen kann.
- Kann im Gegensatz zu einer Pipe im allgemeinen nicht beim Aufruf der beiden Programme in der Shell eingerichtet werden,

  sondern die beiden zugrundeliegenden Programme müssen schon so programmiert sein, dass die Prozesse sich eigenständig auf die Einrichtung eines Sockets verständigen.
- Auch Sockets sind eine spezielle Art von Files mit den entsprechenden Attributen.
- Sockets können auch zwischen Prozessen auf verschiedenen Computern eingerichtet werden.
- *Beispiel*: Programme wie ”`telnet`“, ”`rlogin`“ und ”`ssh`“ zum Einloggen auf anderen Computern basieren intern auf Socket–Paaren (ein Socket für jede Richtung der Kommunikation).

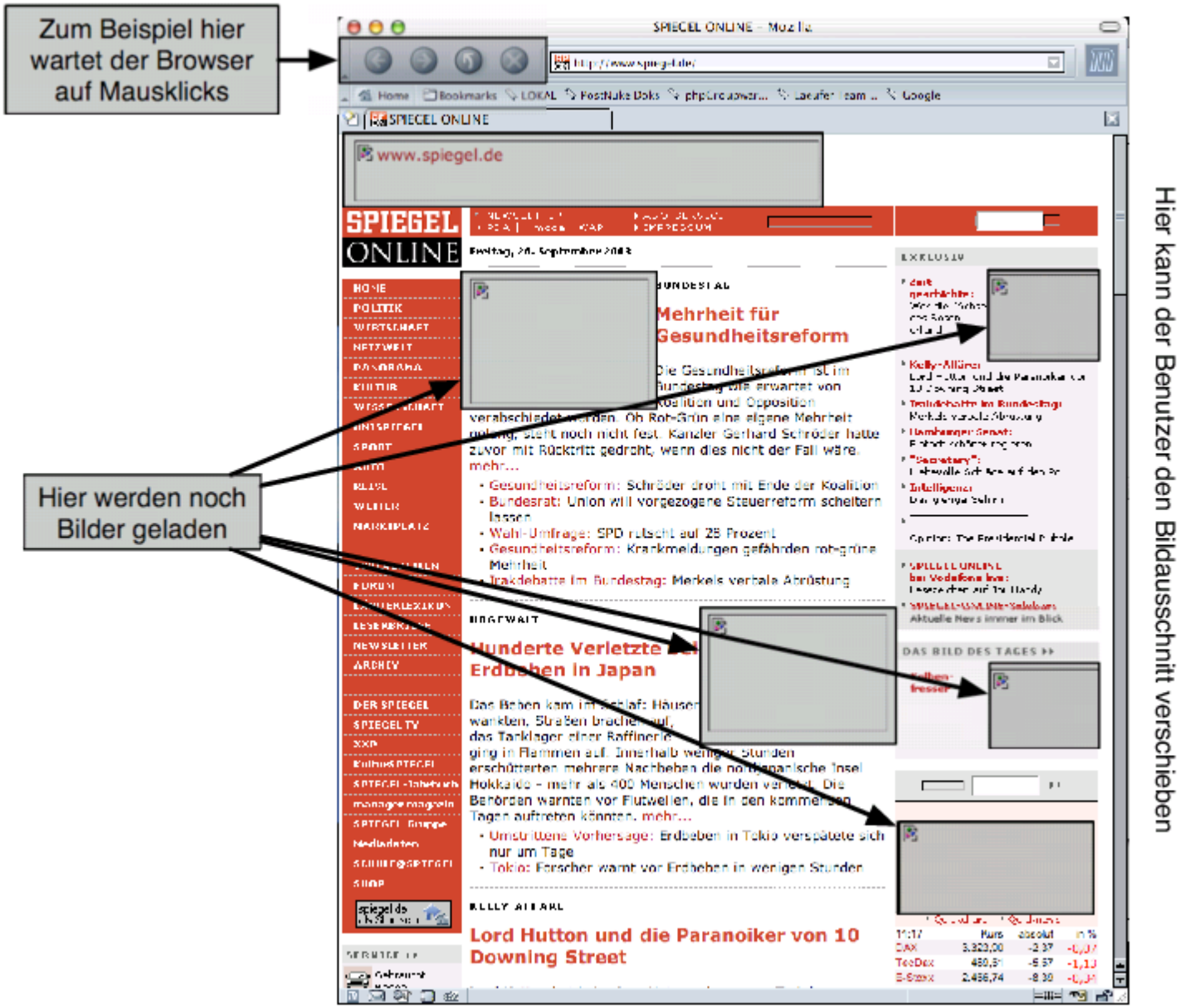
Parallele Prozesse

Ein Prozess erzeugt ein oder mehrere Kind–Prozesse für Teil- oder Sonderaufgaben.

Beispiel:

- Ein WWW–Browser kann offensichtlich mehrere Dinge gleichzeitig, zum Beispiel:
  - ◊ Auf Klick des Benutzers hin (im Balken rechts) den Ausschnitt aus einer längeren WWW–Seite hoch- und hinunterschieben (”scrollen“).
  - ◊ Die noch nicht fertig geladenen Bilder weiter laden.
  - ◊ Auf Mausklicks auf den Buttons warten.
  - ◊ Usw.
- Jede einzelne dieser Aktivitäten ist ein eigener, selbstständiger Prozess.
- Alle diese Prozesse sind aus dem Hauptprozess des WWW–Browsers erzeugt worden.

Beispiel


Zum Beispiel hier wartet der Browser auf Mausklicks
Hier werden noch Bilder geladen
Hier kann der Benutzer den Bildausschnitt verschieben
SPIEGEL ONLINE - Mozilla
www.spiegel.de
SPIEGEL ONLINE
Mehrheit für Gesundheitsreform
Erdbeben in Japan
Lord Hutton und die Paranoiker von 10 Downing Street

Verwandte Konzepte

- *Verteilte Prozesse*:

  Die Kind–Prozesse werden nicht auf demselben, sondern über Netzwerke auf anderen Computern erzeugt.

- *Threads*:
  - ◊ Parallelprozesse werden innerhalb eines Prozesses nur *simuliert*
  - ◊ Die Kommunikation zwischen solchen simulierten Parallel-prozessen kann dann enger und effizienter gestaltet werden, als es vom Betriebssystem angeboten wird.
  - ◊ In einigen Programmiersprachen (z.B. Java) ist diese Möglichkeit von vornherein in die Sprache eingebaut, und sollte für parallel ablaufende Programmteile genutzt werden

Absicherung von Prozessen (I)

Gegen fehlerhaftes Ablaufen anderer Prozesse:

- Das Betriebssystem teilt jedem Prozess eine eigene ”Spielwiese“ im Speicher zu.
- Manche Programmiersprachen (z.B. C und C++) erlauben den direkten Zugriff auf Speicheradressen.

  → Prozesse können also aus Versehen Adressen in fremden Spielwiesen ansprechen.

  → Ein sehr häufiger Programmierfehler in Sprachen wie C und C++!
- Das Betriebssystem führt einen solchen Zugriff nicht aus, sondern schickt jedesmal statt dessen das Signal ”*SEGV*“ (*Segmentation Violation*) an den Prozess.
- Default–Reaktion: Beendigung (”Absturz“) des fälschlicherweise zugreifenden Prozesses.

Absicherung von Prozessen (II)

Gegen unbefugte Kontaktaufnahme anderer Prozesse:

- Signale werden durch das Betriebssystem in der Regel nur dann von einem Prozess an einen anderen wirklich weitergeleitet, wenn beide Prozesse demselben User gehören.
- Bei einer Pipe, die mit | auf der Shell eingerichtet wurde, stellt sich die Frage der Befugnis gar nicht erst: Beide Prozesse gehören demselben User.
- Ein Socket zwischen zwei Prozessen
  - ◊ kann zwar zwischen zwei Prozessen verschiedener User aufgemacht werden (so dass sich die Frage der Befugnis durchaus stellt),
  - ◊ kommt aber nur zustande, wenn die beiden zugrunde liegenden Programme so implementiert sind, dass sie beide der Öffnung eines Sockets zwischen ihnen ”zustimmen“.

Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
  - Benutzer und Benutzergruppen
  - Dateien / Files
  - Ordner / Directories
  - Prozesse
  - ➔ Benutzeroberfläche
  - Virtualisierte Ressourcen
- Rechnernetzwerke

Window-System

Zeichen- und graphikorientiert:

- Früher gab es nur zeichenorientierte Bildschirme und nur die Tastatur als interaktives Eingabegerät.
- Heutzutage sind Bildschirme graphikorientiert, und die Maus steht als zusätzliches Eingabegerät zur Verfügung.
- Neben der zeichenorientierten Ein-/Ausgabe über eine Shell (xterm–Fenster) kann ein Prozess auch eigene *Windows* (*Fenster*) zur Interaktion öffnen.

Window–System:

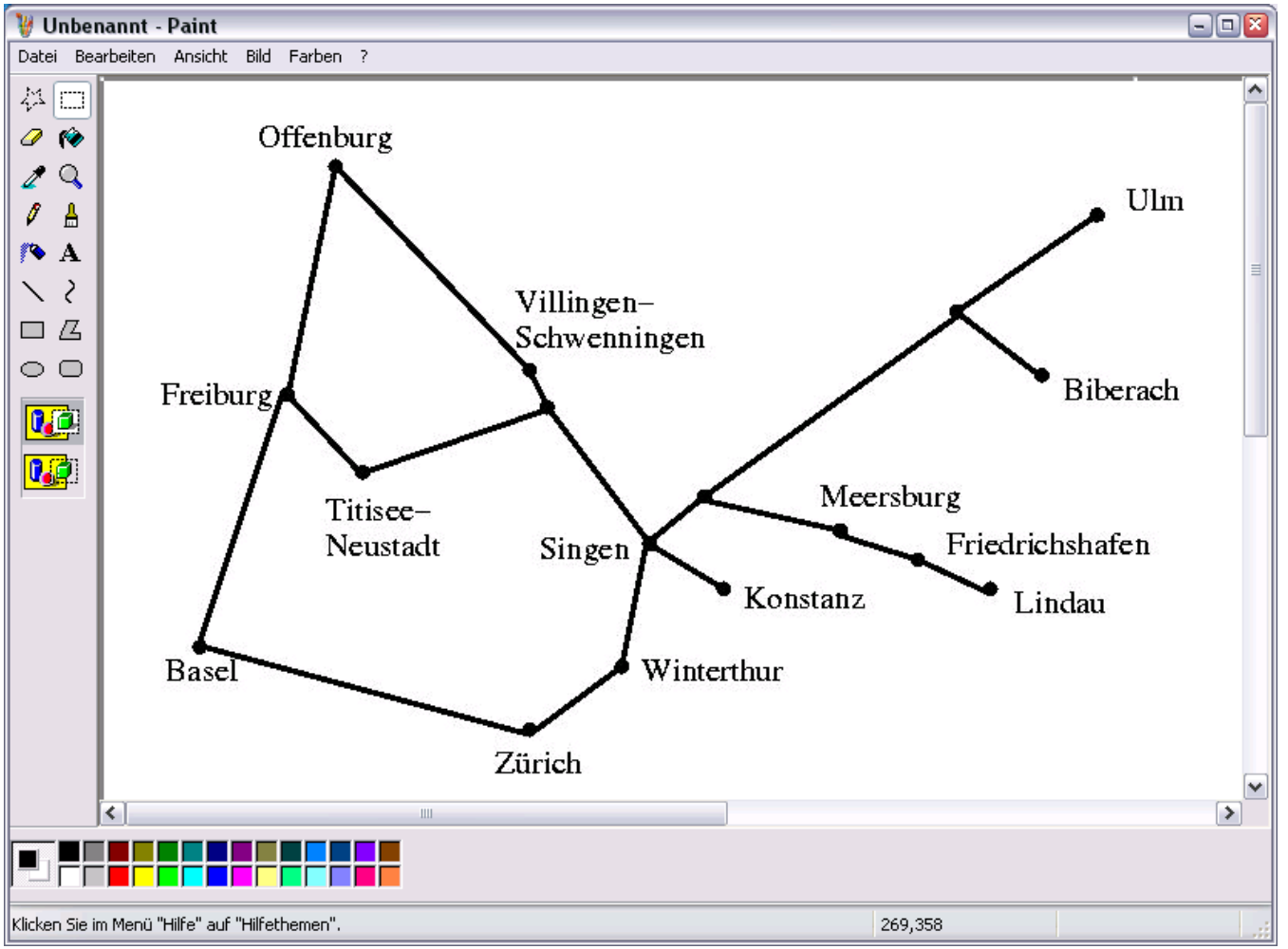
- Der Teil des Betriebssystems, der primär für diese neuen Möglichkeiten verantwortlich ist.
- Aus historischen Gründen ist das bei UNIX eine separate Einheit: das *X11 Window System*.

Window-Orientiertes Display

- Ähnlich wie bei Directories sind die Windows, die momentan auf dem Bildschirm sind, hierarchisch organisiert.
- Der Hintergrund des Bildschirms ist das oberste Window in der Hierarchie (*Root–Window*).
- Auf der nächsten Hierarchiestufe kommen die *Top–Level–Windows*:
  - Die Windows, die eine separate graphische Einheit bilden, die als Ganzes verschoben, vergrößert/verkleinert, iconifiziert etc. werden kann.
  - *Visuelles Kennzeichen*: Das sind genau die Windows mit den charakteristischen Rahmenelementen zum Verschieben, Vergrößern/Verkleinern, Iconifizieren etc.

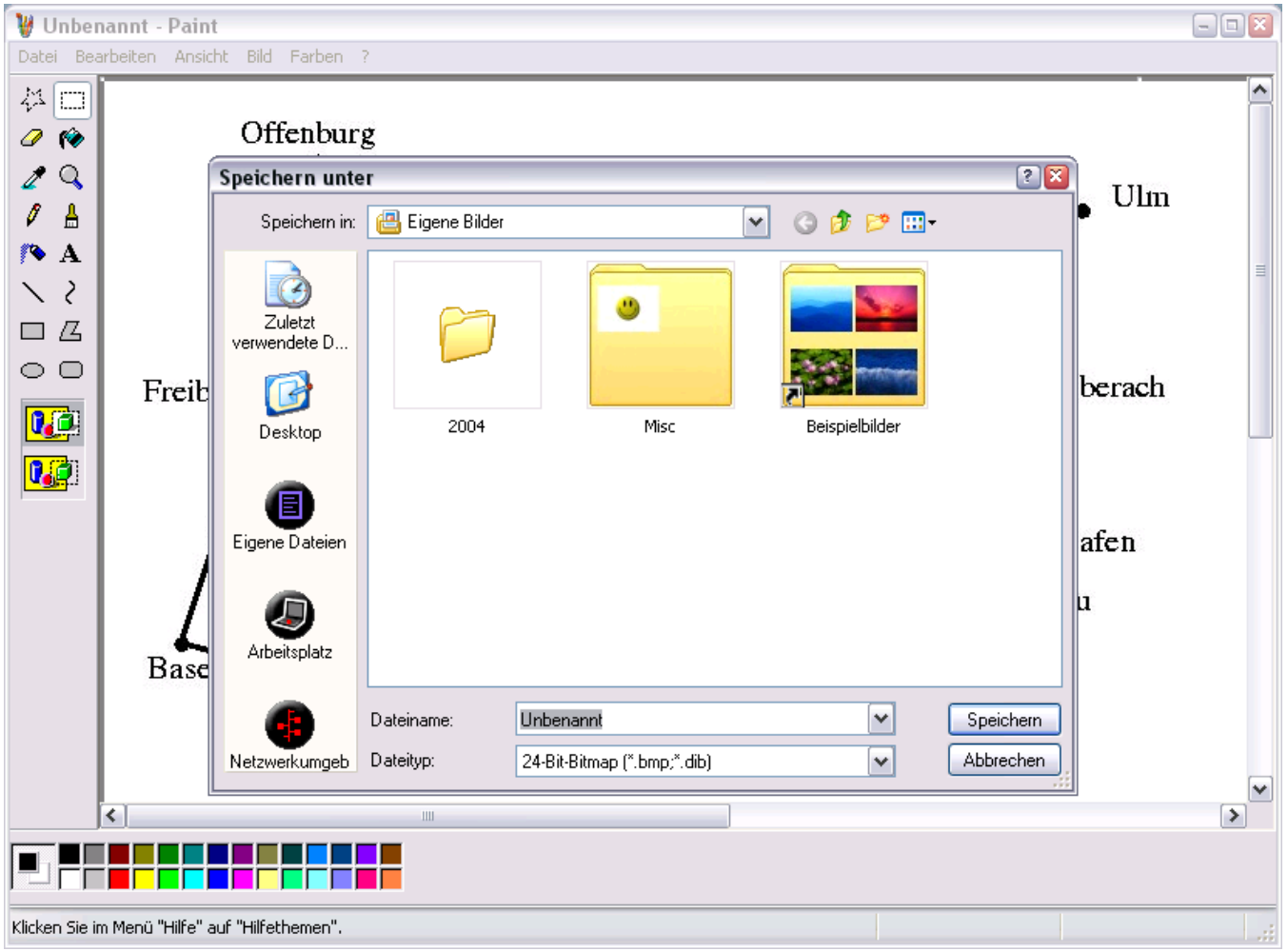
Weitere Fenster

- Sind in Top–Level–Windows als kleinere, abhängige Bestandteile integriert.
- *Beispiele*: Buttons, Menüs, Textfelder...
- Werden im dahinterstehenden Prozess als eigene Unterprozesse verwaltet.



Aber...

- Ein zusätzliches Fenster wie hier das Speicher–Menü beim Zeichenprogramm ”paint“ ist zwar vom selben Programm aufgemacht worden (in einem weiteren Prozess),
- ist aber trotzdem ein separates Top–Level–Window (Rahmen!).



Anzeige von Fenstern

- Für den Inhalt eines Windows ist der Prozess verantwortlich, der das Window geöffnet hat.
- Das Betriebssystem
  - ◊ verwaltet eine Reihung der Windows und
  - ◊ berechnet für jedes Window, welche seiner Regionen sich mit Windows höherer Reihungsnummer überlappen.

→ Diese Regionen des Windows werden nicht angezeigt.

→ Die Präsentation der Windows erweckt wie beabsichtigt den visuellen Eindruck, dass die Windows gemäß Reihung übereinandergelegt wurden.

Zuordnung von Benutzereingaben

- Zu jedem Zeitpunkt ist ein Bildschirmpunkt ausgezeichnet: der *Mouse Pointer* (*Mauszeiger*).
  - Kennzeichnung auf dem Bildschirm durch ein Icon (meist ein Pfeil).
  - Verschieben des Mouse Pointers auf dem Bildschirm: durch Verschieben der Maus.

- Der Mouse Pointer definiert das *aktive Window*:
  - das Window mit höchster Reihungsnummer unter allen Windows, die diesen Bildschirmpunkt enthalten.

  → Also das Window, das um den Mouse Pointer herum auf dem Bildschirm gezeigt wird.

- Der Prozess, zu dem das momentan aktive Window gehört, erhält alle Benutzereingaben (Tastatureingaben und Mausklicks).

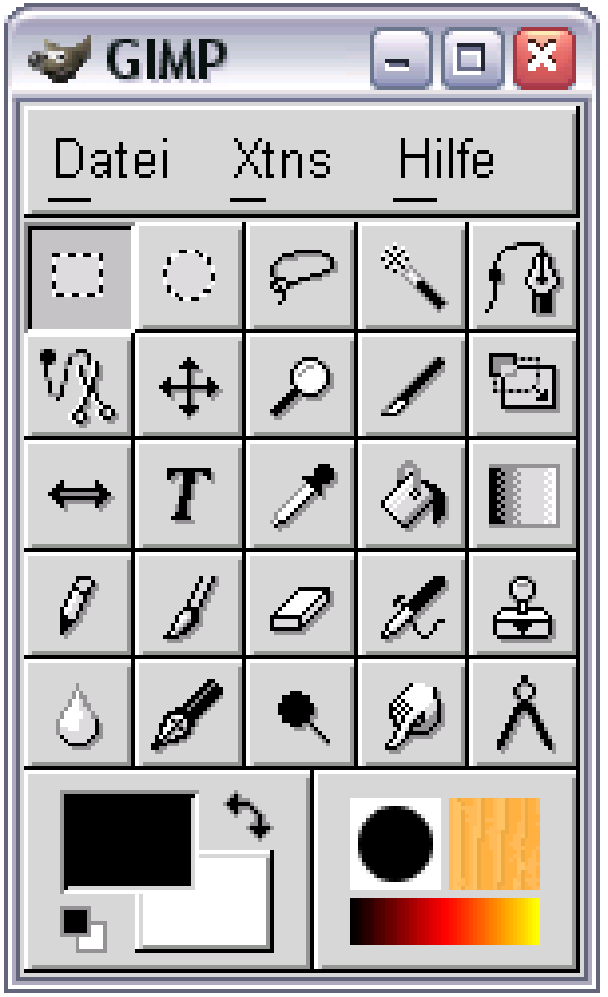
  - Dieser Prozess legt auch fest, durch welches Icon der Mouse Pointer dargestellt wird.

Window Manager

- Ein *Daemon,* der solange läuft, wie das Window System insgesamt läuft.
- Es kann immer nur ein Window–Manager–Prozess pro Einheit Bildschirm+Tastatur+Maus laufen.
- Vom Window Manager verwaltete Windows:
  - ◊ das Hintergrund–Window,
  - ◊ Hintergrund–Menüs,
  - ◊ die einzelnen Windows, aus denen sich der Rahmen jedes Top–Level–Windows zusammensetzt.

Rahmen eines Top-level Windows

- Damit ist der schmale Rahmen einschließlich Titelzeile gemeint.
- Dieser Rahmen setzt sich aus mehreren Windows zusammen:
  - ◊ die vier Seitenkanten des großen Windows,
  - ◊ die vier Ecken,
  - ◊ die Titelzeile,
  - ◊ die Icons in der Titelzeile.
- In jedem dieser kleinen Windows läuft ein eigener Unterprozess des Window Managers.



Operationen am Rahmen

- Benutzeraktionen mit der Maus zur Verschiebung, Größen-veränderung, Iconifizierung etc. von Top–Level–Windows werden vom Window Manager verarbeitet.

- Bei der Größenveränderung eines Windows schickt der Window Manager das Signal WINCH (*Window Change*) an den in diesem Fenster laufenden Prozess.

- Der Prozess, dem dieses Fenster gehört, kann dann sofort den Fensterinhalt an die neue Größe des Fensters anpassen.

Beispiel

Größenveränderung eines Fensters

- Der Prozess wird durch Signal WINCH in Kenntnis davon gesetzt, dass ”irgendetwas“ mit seinem Fenster passiert ist.
- Daraufhin fragt er die neuen Koordinaten des Fensters ab und berechnet damit einen neuen Fensterinhalt.
- Diesen neuen Inhalt lässt er dann ins Fenster zeichnen.
- Der Window Manager sorgt dann dafür, dass der neue Inhalt nur in den momentan sichtbaren Teilen des Fensters wirklich gezeichnet wird.

Virtuelle Window Manager

Einige neuere Window–Manager–Programme bieten auch die Möglichkeit eines *virtuellen Bildschirms*:

- Das Hintergrund–Window ist wesentlich größer als der Bildschirm selbst.
- Es wird immer nur ein Ausschnitt der gesamten Szenerie auf dem Bildschirm gezeigt.
- Durch Verschieben der Maus, Drücken von Funktionstasten o.ä. wird der Ausschnitt, der auf dem Bildschirm gezeigt wird, über dem Hintergrund–Window verschoben.

Grundlagen der Informatik

- Logische und mathematische Grundlagen
- Digitale Daten
- Computerprogramme als Binärdaten
- Betriebssysteme
  - Benutzer und Benutzergruppen
  - Dateien / Files
  - Ordner / Directories
  - Prozesse
  - Benutzeroberfläche
  - ➔ Virtualisierte Ressourcen
- Rechnernetzwerke

Virtualisierte Ressourcen

Erinnerung:

- Jeder Prozess hat seine eigene Spielwiese.

  → Gemeinsame Ressourcen aller Prozesse (insb. der Computer-speicher) dürfen dem Prozess nicht in unmittelbarer, sondern nur in irgendwie ”virtueller“ Form zugänglich gemacht werden.

Allgemein:

Wir reden von virtualisierten Ressourcen, wenn das Betriebssystem

- den Zugriff von Prozessen auf eine Ressource strikt kontrolliert

- diese Ressource den Prozessen durch ”Wegabstrahieren“ technischer Details in einer einfacheren und/oder idealisierten Form präsentiert.

Beispiel: Virtueller Speicher

Problem: Die Adressierung einer Information im Speicher ist ziemlich kompliziert:

- Die Adresse auf der Festplatte wird durch Spurnummer, Offset in der Spur und ähnliche technische Details definiert.
- Zudem kommt noch der Fall hinzu, dass die Information schon irgendwo im Hauptspeicher (oder gar im Cache) ist und daher von dort anstatt von der Festplatte gelesen wird.
- Heutzutage werden die Daten häufig auch verteilt im Netzwerk gehalten.

Frage: Was heißt hier nun Virtualisierung?

Beispiel: Virtueller Speicher (2)

Virtuelle Sicht auf den Speicher:

- Prozesse ”sehen“ nur einen linearen, virtuellen Adressraum $[0...2^{n} - 1]$ wie im Von–Neumann–Modell, in dem jedem Maschinenwort eine einzelne, unveränderliche Zahl als Adresse zugeordnet ist.
- Bei jedem Speicherzugriff wird diese virtuelle Adresse ”unsichtbar“ auf die zugrunde liegende reale Adressierung umgerechnet.
- Das alles wird durch das Betriebssystem selbst geleistet, ohne dass die Prozesse davon irgendwie betroffen wären.

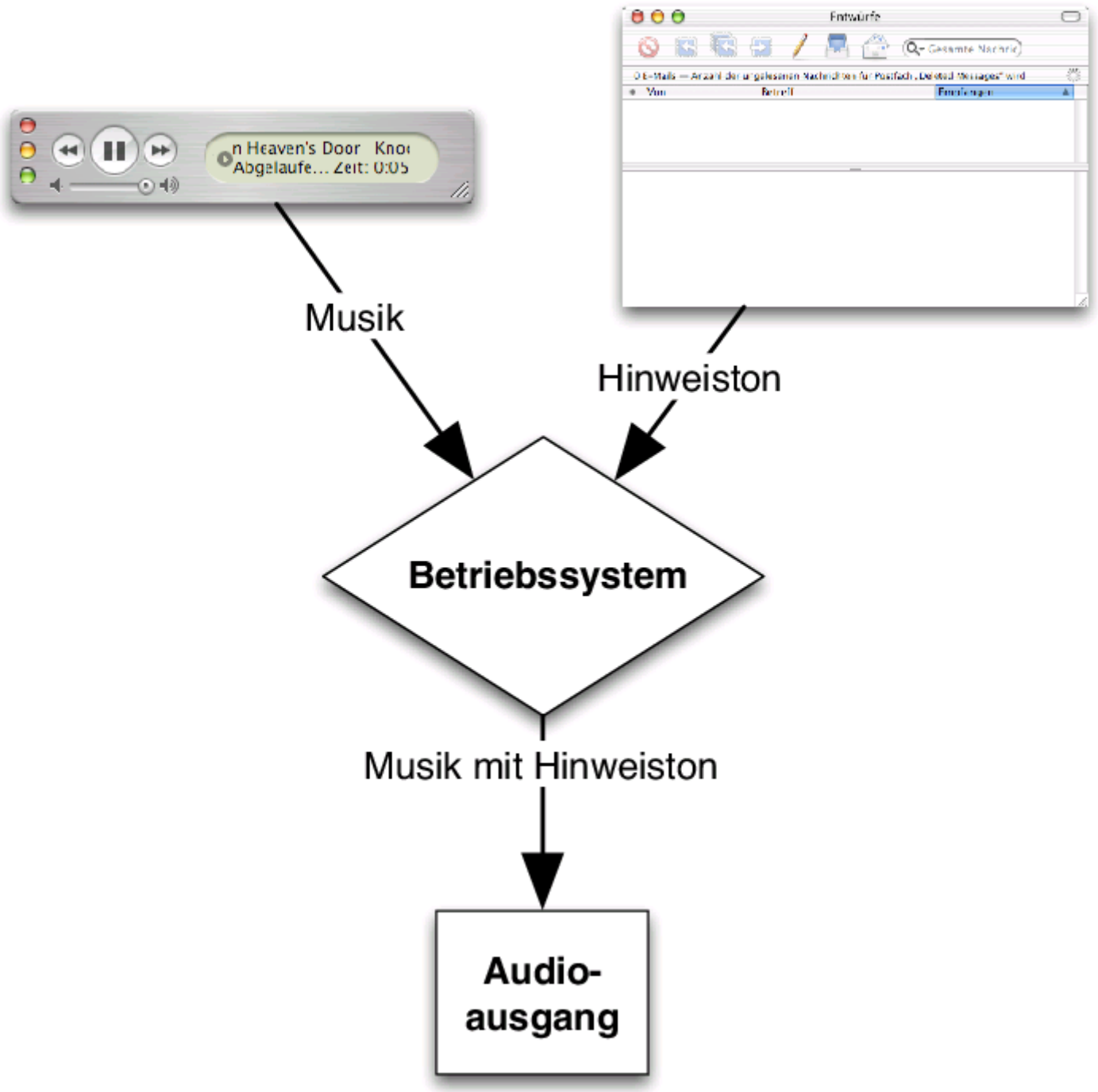
Beispiel: Tonausgabe

Problem:

- Bei normaler Nutzung eines Computers kommt es vor, dass mehr als ein Programm gleichzeitig Ton ausgeben will und soll.
  - Beispielsweise soll das E-Mail Programm akustisch auf neue Mail hinweisen können, während man eine CD oder DVD abspielt.
- Die meisten Computer verfügen aber nur über einen Audioausgang.

Lösung:

- Jedes Programm erhält seinen eigenen virtuellen Audioausgang mit Lautstärkeregler usw.
- Das Betriebssystem nimmt die Daten dieser virtuellen Audioausgänge und mischt sie anhand Ihrer individuellen Lautstärken.
- Das dabei entstandene Signal wird auf dem realen Audioausgang ausgegeben.


Musik
Hinweiston
Betriebssystem
Musik mit Hinweiston
Audio-
ausgang

Umgehung des Betriebssystems

- Es gibt auch Programme, die die Dienste und Virtualisierungsmechanismen des Betriebssystems umgehen und den direkten Speicherzugriff aus Effizienzgründen selbst organisieren.
- *Wichtigstes Beispiel*: Datenbanksysteme.

  → Siehe Veranstaltungen zu ”Datenbankmanagementsystemen“.
- Solche Programme sind natürlich nicht einfach so aufrufbar, sondern müssen auf besondere Art (am Betriebssystem vorbei) gestartet werden.
- Insbesondere wird dabei die Möglichkeit genutzt, dass die Festplatte in *Partitionen* zerlegt werden kann, die potentiell durch eigenständige Systemprozesse verwaltet werden können.

  → Wieder ”Spielwiese“, aber nun auf einem tieferen Abstraktionsniveau, unmittelbar auf der Speicherhardware.