ulm university universität ulm

Grundlagen der Betriebssysteme

[CS2100]

Sommersemester 2014

Heiko Falk

Institut für Eingebettete Systeme/Echtzeitsysteme
Ingenieurwissenschaften und Informatik
Universität Ulm

ulm university universität ulm

Kapitel 8

Rechteverwaltung

Inhalte der Vorlesung

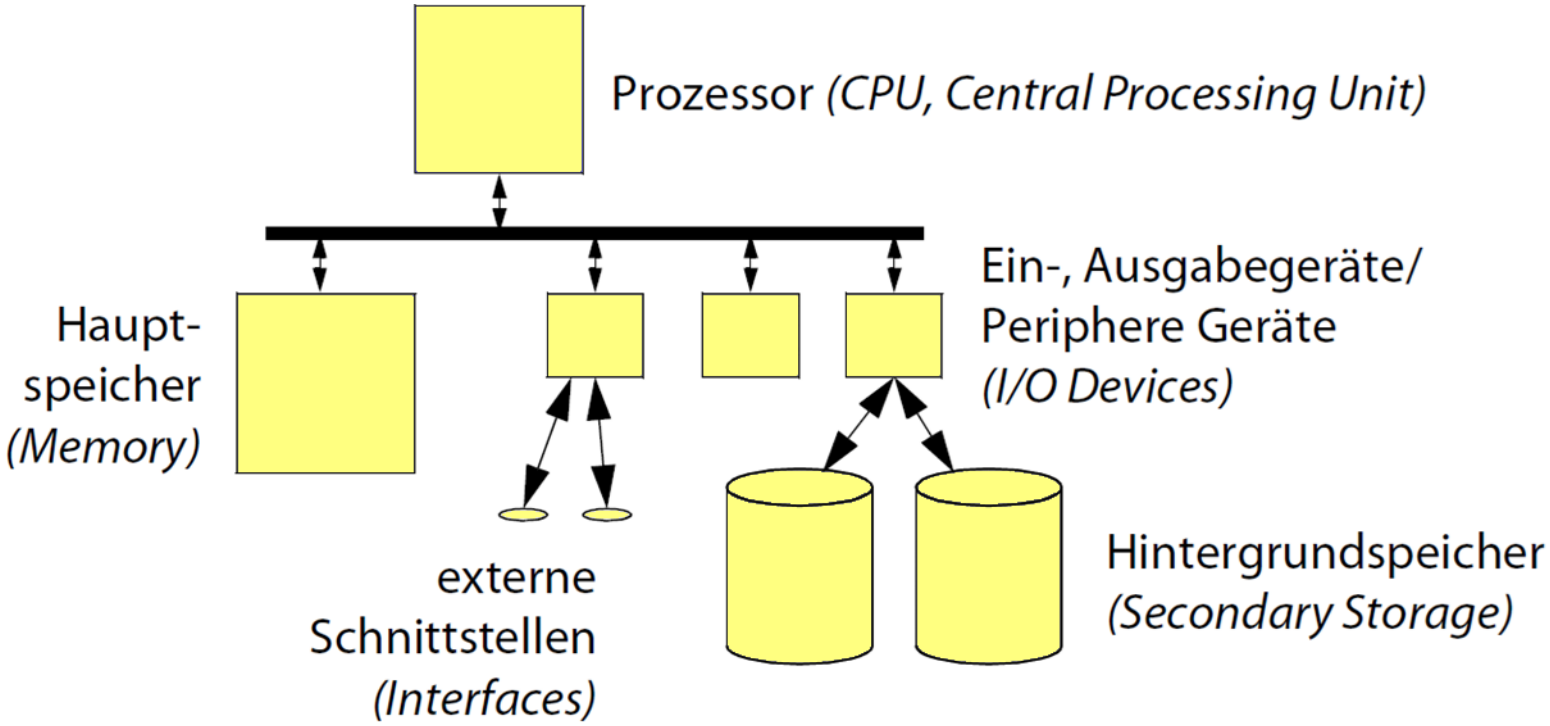
Inhalte des Kapitels

8. Rechteverwaltung

- Einleitung
  - Was ist ein Recht?
- Benutzer- und Gruppenverwaltung
- Anmeldung am System
  - Anmeldedaten
  - Login-Prozess
- Speicherung von Rechten in Filesystemen
  - ACLs unter NT- und POSIX-Systemen
- Übertragung von Privilegien
- Schutz vor schädlicher/fehlerhafter Software

Einordnung

Betroffene physikalische Ressourcen



An der Rechteverwaltung sind alle Komponenten eines Rechners und zusätzlich der Benutzer beteiligt

Was ist ein Recht?

„Sie haben das Recht, zu schweigen.“

Ihr habt auch das Recht auf freie Meinungsäußerung!

Beispiele für Rechte

Blick über den Tellerrand
- Freie Meinungsäußerung
- Demokratische Wahlen

Aus dem Uni-Leben bekannt
- Vorlesung halten (ggfs. delegierbar an Übungsleiter)
- Prüfung abnehmen (i.d.R. nicht übertragbar)
- Im Büro staubsaugen
- Umbauten im Serverraum vornehmen

Konzentration auf Betriebssysteme
- Mehrprozesssysteme
- Mehrbenutzersysteme

Beispiele für Rechte in Betriebssystemen (1)

„Meta“-Rechte

- Rechte vergeben, ändern und entziehen
  - Pauschal
  - Nur für bestimmte Rechte

Anwendungsebene

- Formular in PDF- oder ODF-Textdokument ausfüllen / Felder ändern
- Installierte Software
  - ausführen,
  - deinstallieren,
  - upgraden
- Spielen

Beispiele für Rechte in Betriebssystemen (2)

Low-Level im System

- Direkten Speicherzugriff durchführen
- *Stack*-Größe anpassen
- Echte Adressen sehen
  - Virtuelle Speicherverwaltung
  - Blöcke im Dateisystem
  - Speicheradressen von Java-Objekten

Prozesskontrolle

- Liste der Prozesse anzeigen
- Prozess anhalten oder weiterlaufen lassen
- Prozess starten/beenden, Kind-Prozess erzeugen

Beispiele für Rechte in Betriebssystemen (3)

Zugriff auf Ressourcen

- CPU
- Arbeitsspeicher
- Hintergrundspeicher (Festplatte)
- Netzwerkzugriff und –Bandbreite
- E/A-Bandbreite
- Externe Geräte
- Rechner selbst: Neu starten, abschalten
- *Power Management:* System/Geräte schlafen legen oder aufwecken

Was ist ein Recht?

Recht in der Informatik

Rechte beschreiben, ob und wie Subjekte (Benutzer, Programme, ...) Operationen auf Objekten (Ressourcen) ausführen dürfen.

Kategorisierung

Es gibt Rechte auf verschiedenen Gebieten mit unterschiedlich feiner Granularität.

Rechte aus verschiedener Sicht

Benutzersicht

„Ich darf etwas nicht.“

Zu wenige Rechte: Starke Einschränkungen in Benutzbarkeit

Administratorsicht

„Ich darf alles (und will gar nicht alles dürfen).“

Zu viele Rechte: Gefahr versehentlicher Fehler, Löschungen etc.

Später: Wie können Rechte reduziert/eingeschränkt werden?

Betriebssystem(programmierer)sicht

„Wie können die vorgegebenen oder vom Administrator/Benutzer eingestellten Rechte durchgesetzt werden?“

Rechteverwaltung im Betriebssystem (1)

Wieso im Betriebssystem verankert?

– Auswirkung auf alle Benutzer und Programme am Rechner

Schutz von Informationen

– Vor unerlaubter/ungewollter Einsichtnahme oder Manipulation
– Vor versehentlicher oder absichtlicher Änderung

Sinnvollerweise ergänzt durch

– Netzwerk-Design und –*Policy*
  – Wie ist das Netzwerk aufgebaut?
  – Welche Daten werden von wo nach wo auf welche Art übertragen?
– *Backup*-Plan
– Ggfs. Verträge
– Türschlösser

Rechteverwaltung im Betriebssystem (2)

Rechte pro Benutzer

– Das Betriebssystem kennt Benutzer als Subjekte, die Rechte haben können

Rechte pro Ressource

– Das Betriebssystem kennt Ressourcen wie Dateien, Geräte, Schnittstellen etc. als Objekte, für die Rechte existieren

Rechte pro Operation

– Das Betriebssystem kennt Operationen, die auf Objekten durchgeführt werden und auch reglementiert werden können

Benutzer- und Gruppenverwaltung (1)

Wozu Benutzerverwaltung?

- Mehrere Benutzer an einem Rechner
- Somit nicht mehr für jeden Benutzer ein eigener Rechner nötig
- Jeder bekommt sein eigenes Arbeitsumfeld:
  - Keine manuellen Umstellungen zwischen Benutzerwechseln
  - Bevorzugte Befehle, Tastenkürzel, Hintergrundbild etc. verfügbar
  - Eigene E-Mail-Adresse und –Umgebung,
  - ...
- Vertraulichkeit der Daten der Benutzer sicherstellen

Benutzer- und Gruppenverwaltung (2)

Wozu Gruppenverwaltung?

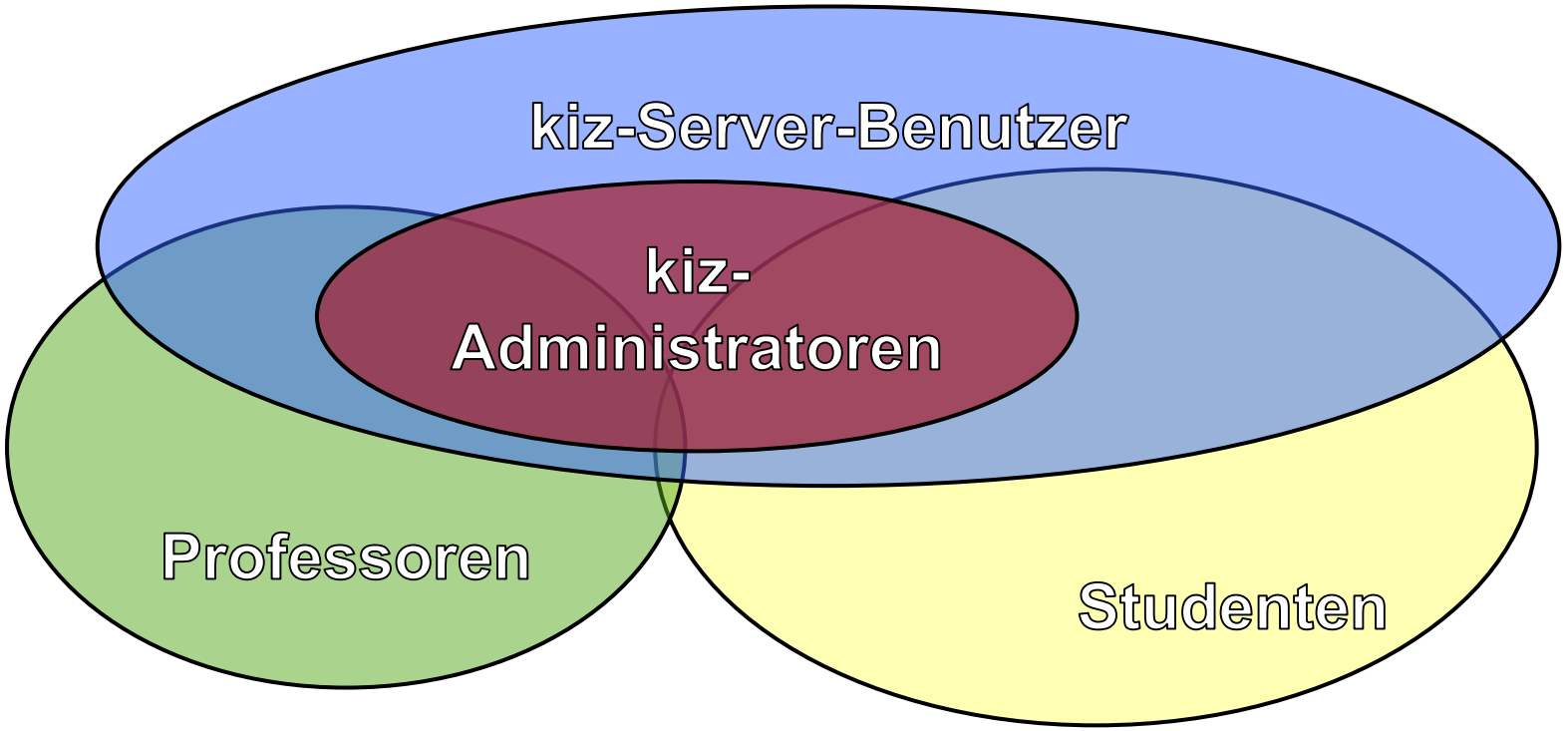
- Abbildung von Gruppierungen aus dem Leben,
  z.B. Institute, Arbeitsgruppen, Freundeskreise, ...
- Einstellungen für viele Benutzer auf einmal
  - Weniger Administrationsaufwand
  - Besserer Überblick
  - Weniger Potenzial für Fehler

Gruppen in der Praxis

- Mehrere Gruppen pro Benutzer sinnvoll, möglich und üblich
- Zuordnung von Benutzern zu „speziellen“ Gruppen wie bspw.
  - „Leute, die drucken dürfen“,
  - „Leute, die USB-Sticks nutzen dürfen“, etc.

Benutzer- und Gruppenverwaltung (3)

Beispiel



Anmeldedaten *(Credentials)* (1)

Erkennung des Benutzers durch das System

- Identifikation über Benutzername
- Beweis der Identität durch weiteres Kriterium, meist geheimes Passwort
- Alternativen zum Passwort: Chipkarte, Iris-Scan, Fingerabdruckscan, ...
- Unterscheidung der Verfahren:
  - Einen Gegenstand *haben*,
  - Etwas/jemand *sein*,
  - Etwas *wissen*
- Sicherer sind Kombinationen mehrerer Verfahren
- Benutzername + Authentifizierungskriterium = sog. *credentials*

Anmeldedaten *(Credentials)* (2)

Auswertung der Anmeldedaten
- Erfolgt durch den Anmeldeprozess

Speicherung im System
- In sicherer Datenbank
- Geschützt vor Manipulation und Auslesen
- Änderungen müssen möglich sein (z. B. möchte/muss Benutzer das Passwort regelmäßig ändern)
- Anmeldeprozess muss *Credentials* lesen können, um sie zu prüfen

Änderungen an Benutzer-/Gruppendatenbanken
- Benötigt erweiterte Rechte
- Bsp. Windows NT: GUI-Tool zur *User*-Verwaltung (in Systemsteuerung)
- Bsp. Kubuntu: KDE-Kontrollmodul *„User Management“*
- Bsp. *nix: **`passwd`**, **`chsh`**, **`usermod`**, ...

Anmeldedaten *(Credentials)* (3)

Beispiel: Speicherung in unixartigen Systemen

- Dateien **`/etc/group`**, **`/etc/passwd`**, **`/etc/shadow`**, **`/etc/gshadow`**
- Öffentlich lesbar: Gruppenzugehörigkeit, IDs, Heimatverzeichnis, vollständiger Name, ...
- Nur vom Administrator lesbar: Passwort(-Prüfsumme)
- **`/etc/passwd`**: (öffentlich)

```
root:x:0:0:root:/root:/bin/sh
nico:x:1000:1000:Nico:/home/nico:/bin/bash
```

- **`/etc/shadow`**: (geschützt)

```
root:$6$Az5Ne3iKJ.2B/…:15475:0:::::
postgres:!:15472::::::
nico:$6$Kjsd638/.JehD…:15458:0:90:5:::
```

Anmelden am System (1)

Der *Login*-Prozess

– Läuft mit erweiterten Rechten (mehr dazu später),
  denn er muss das Passwort aus der geschützten Datenbank lesen
– Wenn Passwort korrekt: Wechsel zu Umgebung des Benutzers
– Weitere Arbeit mit den Rechten des Benutzers (nicht mehr mit erweiterten Rechten!)

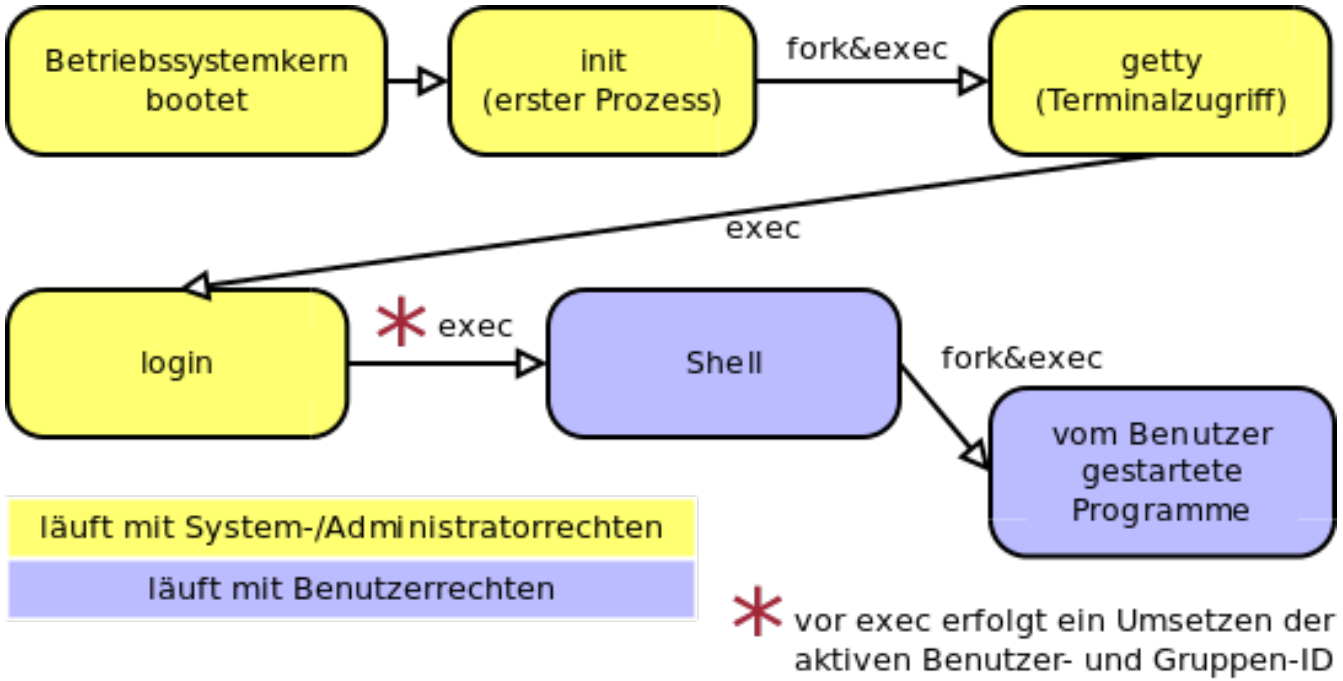
Code-Beispiel

– *Login*-Prozess von MINIX: **`login.c`** (Code vereinfacht)

```
password = getpass("Password:");
/* […] */
if (strcmp(crypt(password, cryptedpwd), cryptedpwd) != 0) {
    write(1, "Login incorrect\n", 16);
} else {
    /* richtiges Passwort, starte Shell für Benutzer */
}
```

Anmelden am System (2)

Ablauf des *Login*-Prozesses



- **`fork&exec`**: Startet einen Kind-Prozess
- **`exec`**: Ersetzt den aktuell laufenden Prozess durch einen anderen

Roter Faden

8. Rechteverwaltung

- Einleitung
  - Was ist ein Recht?
- Benutzer- und Gruppenverwaltung
- Anmeldung am System
  - Anmeldedaten
  - Login-Prozess
- Speicherung von Rechten in Filesystemen
- Übertragung von Privilegien
- Schutz vor schädlicher/fehlerhafter Software

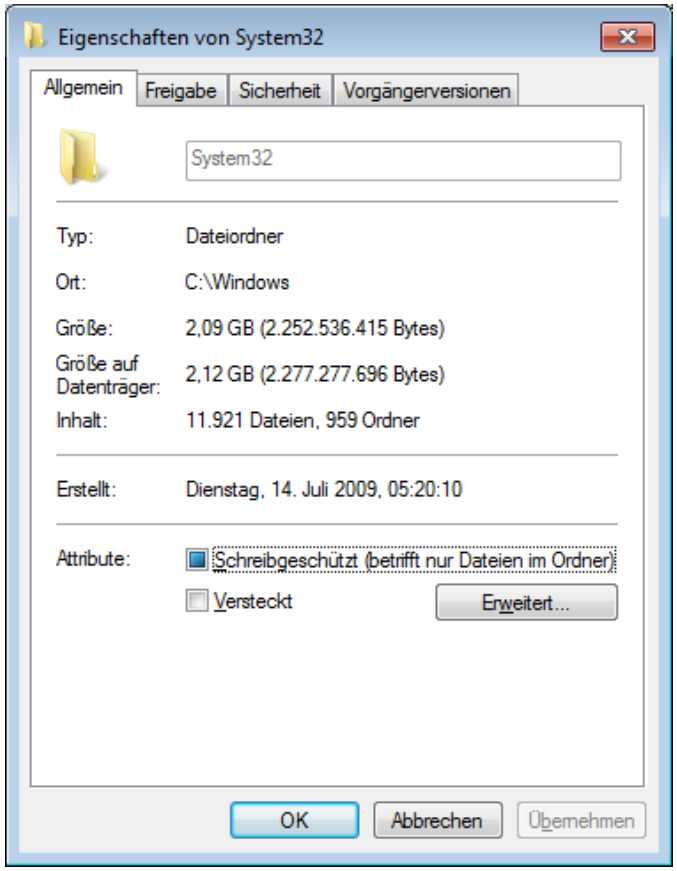
Wiederholung: Rechte in Filesystemen

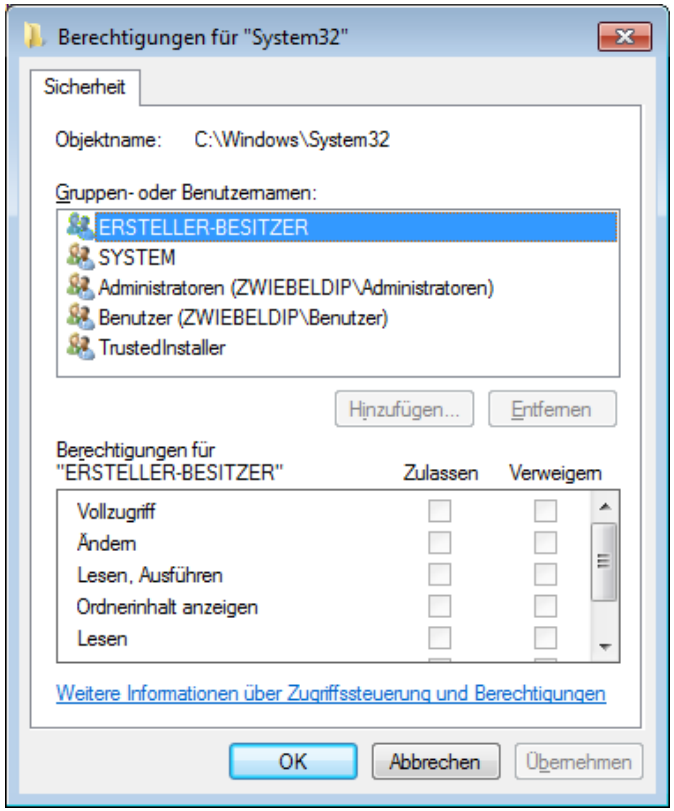
Rechte in Filesystemen

- „Rechte“ FAT
  - Systemdatei
  - Schreibschutz
  - Versteckt
- Rechte unter Windows NT und Nachfolgern
  - Alles ist Objekt
  - *Security Descriptors* für alle Objekte
  - ACLs an *Security Descriptors* gekoppelt
- Rechte unter Unix
  - Gesetzt für: Benutzer, Gruppe, andere („Rest der Welt“)
  - Jeweils möglich: Lesen *(read)*, schreiben *(write)*, ausführen *(execute)*
  - Jeweils 1 Bit; oft oktal angegeben,
    z.B. 0755 (rwxr-xr-x), 0640 (rw-r-----)

Rechte im Filesystem: NTFS (1)

Einstellung von herkömmlichen Filesystemrechten und ACLs





Rechte im Filesystem: NTFS (2)

Per ACLs zugewiesene Rechte

- „Benutzer“ **SYSTEM** (= Betriebssystem) kann explizit Rechte zugewiesen bekommen (normalerweise Vollzugriff auf alles)
- Administratoren dürfen in der Regel ebenfalls nicht alles, müssen auch ausdrücklich Rechte zugewiesen bekommen
- Beispiele für Rechte
  - Vollzugriff (kein eigenes Recht, sondern Kürzel für alle anderen)
  - Attribute lesen
  - Unterordner und Dateien löschen
  - Berechtigungen ändern
  - Besitz übernehmen
- Weit komplexeres und detaillierteres Rechtekonzept als unter Unix
- Vererbung der ACLs an niedrigere Ebenen im Verzeichnisbaum ist Standard, lässt sich abschalten

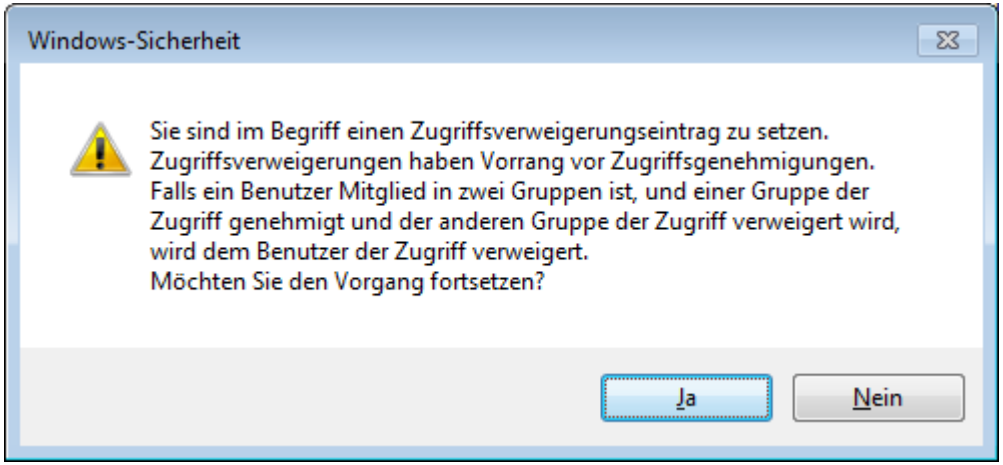
Rechte im Filesystem: NTFS (3)

Auswertung der ACLs

- Ist keine ACL vorhanden (Erlaubnis oder Verbot), ist der Zugriff *nicht* möglich
- Verbote überschreiben Erlaubnisse

  Achtung bei Verboten für ganze Gruppen, z. B.
  - Alice ist in der Gruppe Studenten, Bob nicht,
  - sie legt eine Datei an,
  - sie setzt ACLs: Alice & Bob dürfen lesen, Studenten ist alles verboten

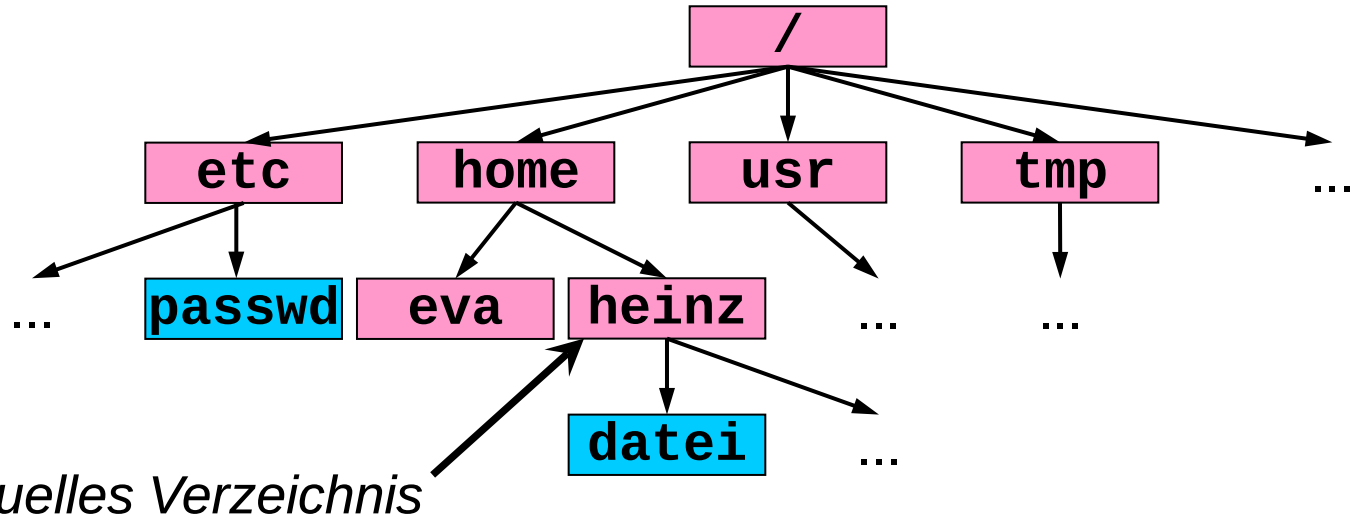
  ☞ Alice darf selbst nicht mehr lesen!



Rechte im Filesystem: *nix (1)

Zur Erinnerung

– Hierarchisches Filesystem



– Benutzer und Gruppen haben *User-IDs* und *Group-IDs*

– Weitere Filesysteme werden durch *Mounten* (Einhängen) in den Verzeichnisbaum verfügbar gemacht

Rechte im Filesystem: *nix (2)

„Klassische“ Rechte

- Zugriffsrechte für
  - Benutzer (eigentlich: Besitzer) der Datei,
  - eine Gruppe, und
  - alle anderen
- Rechte, die eine Datei haben kann:
  - Lesen / *read* (r)
  - Schreiben / *write* (w)
  - Ausführen / *execute* (x)
- **`root`** (genauer: Benutzer mit UID 0) darf alles:
  Alle Rechte-Prüfungen werden vom Betriebssystem umgangen

Rechte im Filesystem: *nix (3)

„Klassische“ Rechte: Beispiel

```
$ ls –l /home/alice
total 40
drwx------ 11 alice users  4096 Jun 14 14:05 Musik
-r--r--r-- 11 alice users   120 Jun 14 12:11 bsp.text
-rwxr-x--- 11 alice users 14828 Jun 11 00:42 test
-rw-r----- 11 alice users  2035 Jun 11 00:41 test.c
```

- **`Musik`** ist ein Verzeichnis, das Benutzer **`alice`** und Gruppe **`users`** gehört. Nur **`alice`** darf die darin enthaltenen Einträge lesen, neue Einträge anlegen oder existierende verändern und auch die *Inodes* der Einträge holen („in das Verzeichnis wechseln“)
- **`bsp.text`** kann von jedem gelesen, von niemandem geändert werden.
- **`test`** ist eine Datei, die von **`alice`** und allen Mitgliedern der Gruppe **`users`** gelesen und ausgeführt werden kann. **`alice`** darf das Programm zusätzlich noch verändern.

Rechte im Filesystem: *nix (4)

„Klassische“ Rechte: Bit ‚x‘ bei Verzeichnissen

- „In das Verzeichnis wechseln“
- Hat nichts mit dem Lesen der Verzeichniseinträge zu tun:

```
$ ls –ld /tmp/test
drw------- 2 alice users 4096 Jun 14 07:11 /tmp/test
$ ls –l /tmp/test
ls: cannot access test/welt: Permission denied
ls: cannot access test/hallo: Permission denied
total 0
-????????? ? ? ? ?            ? hallo
-????????? ? ? ? ?            ? welt
```

- Das Verzeichnis selbst kann von **`alice`** gelesen werden (Bit ‚**`r`**‘)
- Es ist nicht möglich, den *Inode*-Verweisen im Verzeichnis zu folgen
- Daher ist keine Anzeige von Rechten, Besitzer, Gruppe, Datum, Zeit der enthalten Dateien möglich

Rechte im Filesystem: *nix (5)

„Klassische“ Rechte: Spezielle Rechte-Bits

- suid *(set user ID)*
  - Übernehmen der *User-ID* des Datei-Eigentümers zur Laufzeit
  - Z. B. **`passwd`** mit suid-Bit und Eigentümer **`root`**, um Passwörter zu ändern (siehe später)
- sgid *(set group ID)*
  - Übernehmen der *Group-ID* der Gruppe der Datei zur Laufzeit
- „*sticky*“
  - Wenn für ein Verzeichnis gesetzt: Verhindert, dass Dateien darin durch jemand anderen außer dem Besitzer gelöscht werden
  - Üblich für **`/tmp`**, **`/var/tmp`**, **`/dev/shm`**

```
$ ls -ld /tmp
drwxrwxrwt 17 root root 12377 Jun 14 14:14 /tmp
```

  - Jeder darf Dateien in **`/tmp`** anlegen, aber nur der Besitzer löschen

Rechte im Filesystem: *nix (6)

Beispiel: Gruppenarbeit

```
$ groups alice
alice : users sopra42 students
$ ls –l /home/groups/sopra42
total 20
drwxrws---  3 admin sopra42  4096 Feb 21 00:54 .
drwxr-xr-x 56 root  root    86016 May 26 23:21 ..
drwxrwx---  2 alice sopra42  4096 Jun 14 11:09 dir1
-rw-rw----  1 bob   sopra42 24476 Jun 14 10:48 file1
-rwxrwx---  1 alice sopra42 96061 Jun 10 20:14 file2
```

– Alle Dateien und Verzeichnisse in **`/home/groups/sopra42`** gehören der Gruppe **`sopra42`** und sind von ihr les-/schreibbar

– Set-group-ID-Bit für das Verzeichnis:
Alle neu angelegten Dateien oder Verzeichnisse werden automatisch der Gruppe **`sopra42`** zugeordnet, obwohl die Standardgruppe der Benutzer **`users`** ist.

Übertragung von Privilegien (1)

Prozesstabelle

- Das Betriebssystem führt eine Tabelle aller aktiven Prozesse
- Jeder Prozess hat aktuelles Arbeitsverzeichnis und eine Benutzer- und Gruppen-ID (UID und GID)
- Operiert mit den Rechten des Benutzers und der Gruppe

ID-Konzept unter Unix

- *real* UID, *real* GID: IDs, unter denen der Prozess gestartet wurde
- *effective* UID, *effective* GID: IDs, gegen die geprüft wird, ob Zugriffe erlaubt sind
- In den meisten Fällen sind *real* und *effective* UIDs gleich (nämlich dann, wenn keine speziellen Rechte gegeben werden sollen)

Übertragung von Privilegien (2)

Änderungen der EUID

- Um die effektiven IDs zu ändern, gibt es Systemaufrufe
  - **`setuid(...) / setgid(...)`**
  - Setzen die effektiven IDs in der Prozesstabelle auf übergebenen Wert
  - Aufruf nur mit entsprechenden Rechten möglich (i. d. R. nur für Administratorbenutzer **`root`**)
- Alternative: Spezielle Bits im Filesystem
  - suid *(set user ID)*, sgid *(set group ID)*
  - Betriebssystem setzt die effektiven IDs in der Prozesstabelle bei Ausführung des Programms auf Besitzer bzw. Gruppe des Programms
  - Bits können von Besitzer der Datei gesetzt werden

Übertragung von Privilegien (3)

Beispiel: Passwort ändern

- Jeder Benutzer soll sein Passwort ändern können
- Dazu ist Schreibzugriff auf die Passwort-Datenbank (unter *nix: **`/etc/shadow`**) nötig
- Aber es soll nicht jeder die Passwörter anderer Benutzer überschreiben können

☞ Lösung: Programm, das dies mit Administratorrechten erledigt und nur Änderungen des einzelnen Benutzerpassworts durchführt

- Bsp. Ubuntu: Programm **`passwd`**:

```
$ ls –l /usr/bin/passwd
-rwsr-xr-x  1 root root 42824 Apr  9 04:32 /usr/bin/passwd
```

- Set-user-ID-Bit sorgt für Ausführung des Programms mit *effective* UID 0 (= **`root`**)

ACLs unter Linux (1)

***Access Control Lists* im Filesystem**

- Implementierung der POSIX-ACLs
- Speicherung in sog. *Extended Attributes* (*xattr*; Name-Wert-Paare) im Filesystem, kann jede Datei/Verzeichnis/Spezialdatei haben
- In Filesystem-Listing (ls-Befehl) Anzeige von klassischen Rechten für Besitzer, Gruppe, Rest; dahinter „**+**“:

```
$ ls –l /srv/www/web1
-rwxrwx---+ 2 root root  4096 May 18 14:16 /srv/www/web1
```

- Es gibt keine Negativ-Rechte, es können nur pro Benutzer oder Gruppe Rechte *eingeräumt* werden
- Jeder Benutzer kann die ACLs seiner Dateien selbst setzen (muss nicht durch **`root`** erfolgen), auch für Gruppen, denen er nicht angehört!

ACLs unter Linux (2)

Access Control Lists im Filesystem

– Setzen/Löschen der ACLs unter Linux über den Befehl **`setfacl`**:

```
$ setfacl –m user:alice:rwx /srv/www/web1
$ setfacl –m group:webusers:rwx /srv/www/web1
```

– Anzeige der ACLs unter Linux über den Befehl **`getfacl`**:

```
$ getfacl /srv/www/web1
user::rwx
user:alice:rwx
group::---
group:webusers:rwx
mask::rwx
other::---
```

ACLs unter Linux (3)

ACL-Maske (maximale Rechte)

- **`mask::rwx`**
- Angewendet (logisches UND) auf Benutzer-/Gruppen-ACL-Einträge vor Prüfung, ob Zugriff erlaubt ist
- Im Filesystem-Listing an Stelle der Gruppen-Rechte angezeigt

Vererbung der ACLs

- Es gibt Default-ACLs: **`default:user:alice:rwx`**, **`default:group::---`** usw.
- Wird eine neue Datei angelegt, werden Default-ACLs des enthaltenden Verzeichnisses übernommen
- Wird ein neues Verzeichnis angelegt, werden die Default-ACLs des enthaltenden Verzeichnisses als Default-ACLs des neuen gesetzt
- Nachträgliche Änderungen der ACLs sind dennoch möglich

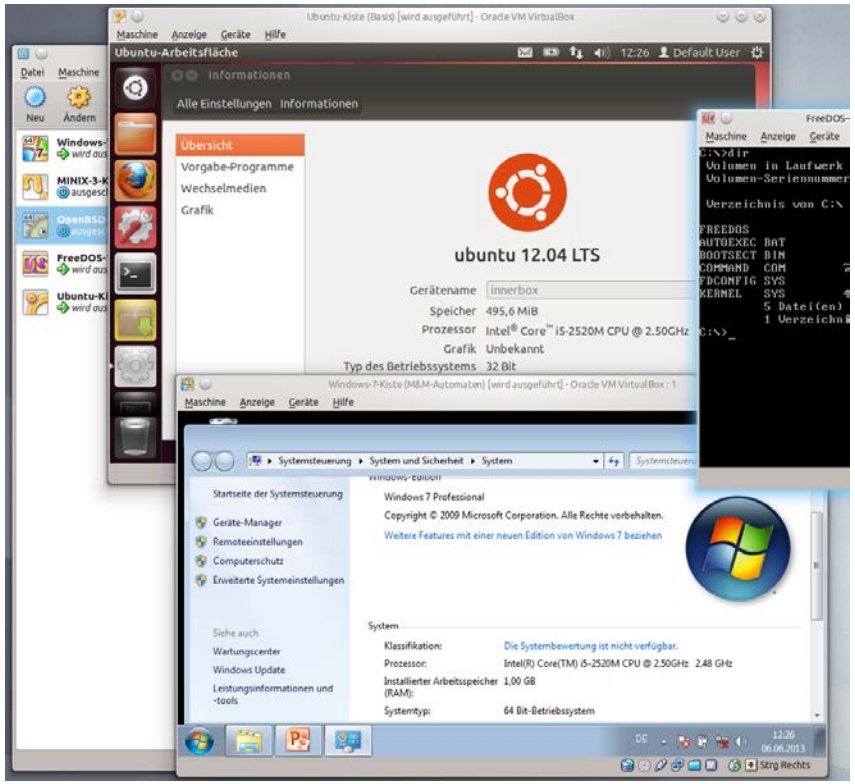
Schutz vor schädlicher/fehlerhafter Software

Speicherbereiche schützen

– Jeder Prozess bekommt seinen eigenen virtuellen Adressraum (☞ Kapitel 6, Folie 27ff.)

Prozesse isolieren

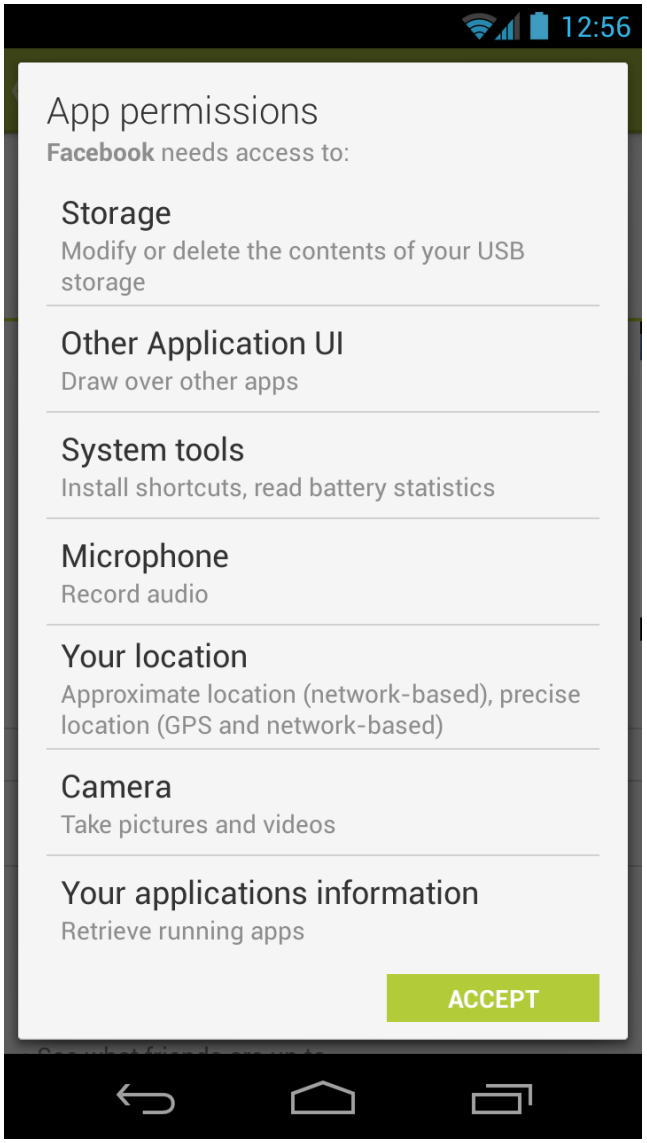
– *Application Sandboxing* (z. B. bei Android): Prozesse voneinander abgeschottet
– **`chroot`**: Trennung auf Filesystemebene – Neues Wurzelverzeichnis
– FreeBSD-Jails, OpenVZ, ...: Isoliertes Filesystem, Benutzer, Netzwerk, ...
– Virtuelle Maschine (z. B. VirtualBox): Software virtualisiert kompletten Computer, auf dem Betriebssystem und weitere Software laufen kann



Application Sandboxing

Beispiel: Android *Sandbox*

- *Sandbox* basiert auf klassischen Unix-Rechten
- Jeder Prozess läuft unter einem separaten Benutzer
  - Es gibt keine zwei Programme, die unter derselben Benutzerkennung ausgeführt werden
  - Prozess kann standardmäßig nur auf eigene Ressourcen zugreifen
- Prozess kann nach erweiterten Rechten fragen, um auf andere Ressourcen/Daten anderer Prozesse zuzugreifen:



Zusammenfassung

Rechteverwaltung
- Teil des Betriebssystems
- Auf verschiedenen Abstraktionsebenen

Benutzer und Gruppen
- Abbildung von Personen und Teams etc., so dass der Administrator des Systems Rechte zuweisen/verwalten kann
- Anmeldedaten *(Credentials)*

Bekannt machen am System
- *Login*-Prozess

Im Filesystem
- Verwendung von Benutzern, Gruppen, ACLs zur Einstellung von Zugriffsrechten für echte Dateien oder für angeschlossene Hardware über Spezialdateien

Vertiefung/Weiterführung der Thematik

Uni-Veranstaltungen

– Grundlagen der Rechnernetze
– Sicherheit in IT-Systemen

– Web Engineering
– Verteilte Betriebssysteme

– Kryptologie

Literatur

– Z. B. Barrett, Silverman, Byrnes. *Linux Security Cookbook*. O'Reilly, 2003.