D Autorisierung und Zugriffskontrolle

D.1 Autorisierung von Benutzern

- Sonderfall allgemeiner Schutzmechanismen
- Beispiel für einfache Realisierung: UNIX-Dateizugriffe
  - Benutzer ist UID zugeordnet
    - Korrektheit wird durch Authentifizierung sichergestellt
  - System kennt einzelne Benutzer und Gruppen
  - Für jede Datei kann die Berechtigung für Eigentümer, Gruppe, Rest festgelegt werden
    - rwx
- Zugriffskontrolle muss sicherstellen, dass jeder Zugriffsversuch kontrolliert wird und nur autorisierte Zugriffe möglich sind

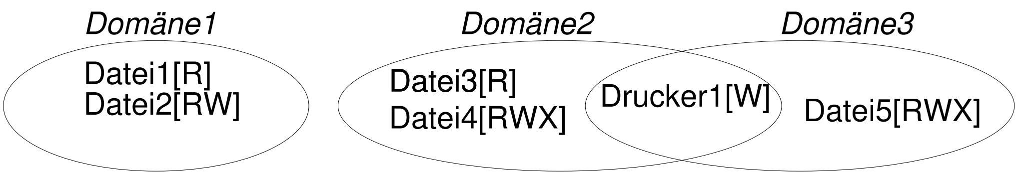
D.2 Speicherschutz

- Schutzräume für Anwendungen (Prozesse) und Betriebssystem (Systemkern)
- Schutz zwischen Systemkern und Anwendungen ist Basis für alle weiteren Zugriffskontrollkonzepte
  - Verwaltung in Datenstrukturen des Systemkerns
  - logische/virtuelle Adressräume gewährleisten Abgrenzung
    Programmadressen → MMU → Maschinenadressen
  - MMU-Datenstrukturen durch Systemkern-Adressraum geschützt
  - Kontrollierte Übergänge zwischen Betriebsmodi (*user / kernel mode*)
    → Trap
- Alternative: Schutz durch Programmiersprache (Typ-Konzept) und Laufzeitsystem
  - Beispiel: Java + Verifier + Virtuelle Maschine

D.3 Objektschutz

1 Schutzdomänen

- Domäne = Menge von (Objekt, Rechte)-Tupeln
  - Beispiele für Domänen
    - ein Benutzer
    - eine Gruppe
    - eine Rolle
- Recht = Menge von Operationen, die auf dem Objekt ausgeführt werden dürfen



1 Schutzdomänen (2)

- Ein Prozess ist zu jedem Zeitpunkt einer Schutzdomäne zugeordnet
- Wechsel der Schutzdomäne (am Beispiel UNIX)
  - exec einer Datei mit s-bit
  - setuid-Systemaufruf
- Verwaltung der Schutzdomänen im System
  - Schutzmatrix

Objekt:	Datei1	Datei2	Datei3	Datei4	Datei5	Drucker1
Domäne:1	R	RW				
2			R	RWX		W
3					RWX	W

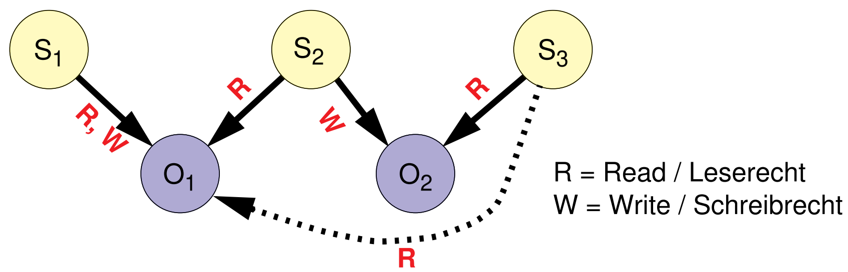
- Schutzmatrix in der Praxis nicht handhabbar
  - Information aus Spalten: Zugriffskontrolllisten
  - Information aus Zeilen: Capabilities

1 Zugriffskontrolllisten (ACLs)

- Zugriffslisten an den Objekten (Dateien)
  - jedes Objekt hält eine Liste der Berechtigungen
    - welche Domänen haben welche Rechte
    - z. B. Unix-Datei: Inode enthält UID, GID und Rechte-Bits (primitiv)
- Zugriffslisten an den Subjekten (Prozesse)
  - jedes Subjekt hält eine Liste von Objekten und den Berechtigungen, die das Subjekt für das Objekt hat
  - ähnlich → Capabilities
- sichere Verwaltung
  - zentral: in Datenstrukturen des Systemkerns
  - dezentral
    - Objektverwaltung durch Serverprozesse
    - Aufteilung von Berechtigungs- und Zugriffskontrolle
    - Verwendung von kryptographischen Verfahren

2 Zugriffsausweise (Capabilities)

- Ein Benutzer (Subjekt) erhält eine Referenz auf ein Objekt
  - die Referenz enthält alle Rechte, die das Subjekt an dem Objekt besitzt
  - bei der Nutzung der Capability (Zugriff auf das Objekt) werden die Rechte überprüft



Subjekte und Objekte; Weitergabe einer Capability (O_1 von S_2 nach S_3)

2 Zugriffsausweise (Capabilities) (2)

★ Vorteile

- keine Speicherung von Rechten beim Objekt oder Subjekt nötig; Capability enthält Zugriffsrechte
- leichte Vergabe von individuellen Rechten
- einfache Weitergabe von Zugriffsrechten möglich

▲ Nachteile

- Weitergabe nicht kontrollierbar
- Rückruf von Zugriffsrechten nicht möglich
- Capability muss vor Fälschung und Verfälschung geschützt werden
  - durch kryptographische Mittel (Capability wird signiert und wird bei Manipulation automatisch zerstört)
  - durch Speicherschutz (Capability wird im Systemkern gehalten und Anwendung erhält nur einen Index darauf)

3 Capabilities und ACLs am Beispiel UNIX

■ Systemaufruf open

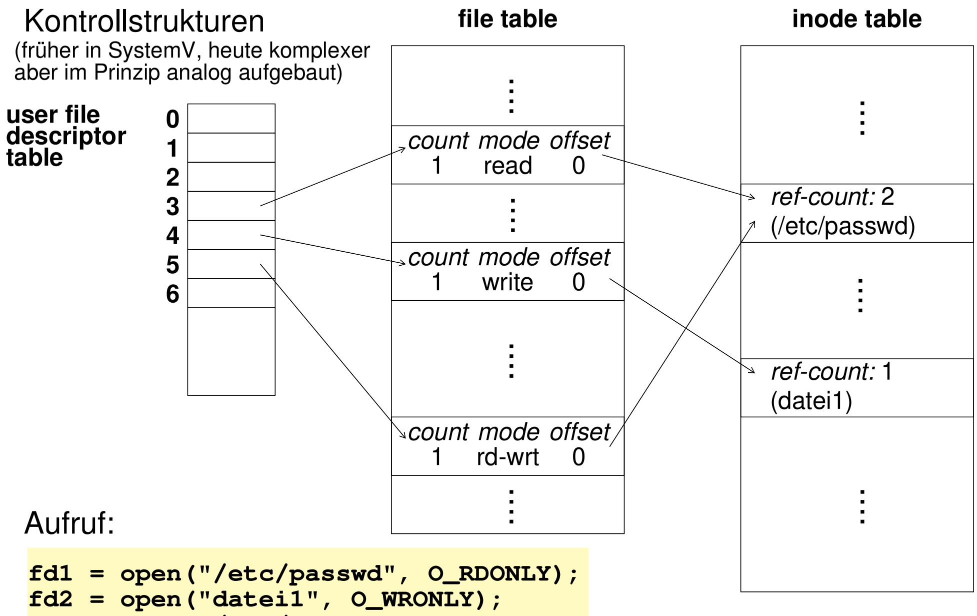
- überprüft Zugriffsrechte des Prozesses (ACL)
- erzeugt *file handle* Struktur im Systemkern
  - Zugriffsmode (read/write) wie bei open angegeben
  - Verweis of Inode-Struktur
- Verweis auf *file handle* wird in *open file table* im Prozesskontrollblock eingetragen
- open liefert Filedeskriptor zurück = Index in *open file table*

■ Filedeskriptor + file handle = Capabiliy

- nicht manipulierbar
  - file handle und open file table liegen im Systemkern
  - nur Verweise auf erfolgreich geöffnete Dateien werden eingetragen
- Rechte (bei open angegebener mode) werden in file handle gehalten

3 Capabilities und ACLs am Beispiel UNIX (2)

- Kontrollstrukturen
  (früher in SystemV, heute komplexer aber im Prinzip analog aufgebaut)



- Aufruf:

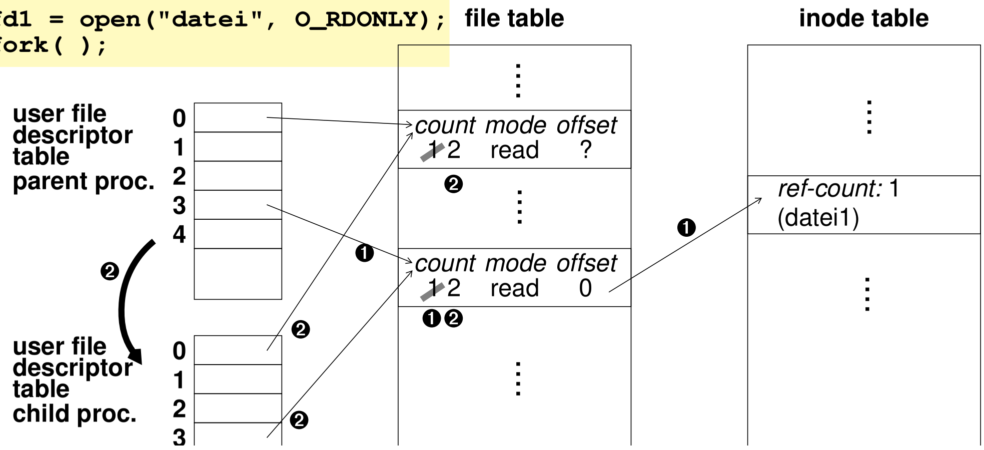
```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
fd3 = open("/etc/passwd, O_RDWR);

-> fd1 == 3, fd2 == 4, fd3 == 5
```


3 Capabilities und ACLs am Beispiel UNIX (3)

- Capabilities können weitergegeben werden
  - ➤ im Rahmen von fork — offene Datei wird vererbt

```
❶ fd1 = open("datei", O_RDONLY);
❷ fork( );
```



  - ➤ durch Versenden über socketpairs

- Empfangender Prozess benötigt dafür keine Zugriffsrechte (ACL)!

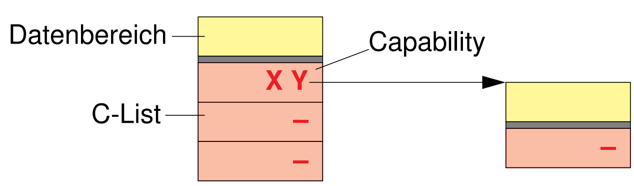
D.4 Konzeptionelle Grundlagen von Capabilities

1 Hydra

- Hydra ist ein Capability-basiertes Betriebssystem
  - ◆ entwickelt Mitte der 1970er Jahre an der Carnegie-Mellon University
  - ◆ lief auf einem speziellen Multiprozessor namens **C.mmp**
  - ◆ Ziel: Experimentierplattform für verschiedene Betriebssystemmechanismen
    - ➤ verschiedene Dateisysteme, Scheduler, ...
    - ➤ Policy-Mechanism-Separation:
      Mechanismen im Systemkern, Policies können ausgelagert werden
    - ➤ frühes Mikrokern-Konzept
    - ➤ Problem: Schutz der "Objekte" vor unberechtigten Zugriffen
  - ◆ Capability-Mechanismen sind integraler Bestandteil des Betriebssystems
  - ◆ Relevanz aus heutiger Sicht:
    - ➤ Hydra-Capabilities sind gut mit geschützten Objekt-Referenzen in einer objekt-orientierten Sprachumgebung (wie Java) vergleichbar

1 Hydra (2)

- Objekte werden in Hydra durch Capabilities angesprochen und geschützt
  - ◆ Objekte haben einen Typ
    (z. B. Prozeduren, Prozesse/LNS, Semaphoren, Dateien etc.)
  - ◆ Capabilities haben entsprechenden Typ
  - ◆ benutzerdefinierte Typen sind möglich
  - ◆ generische Operationen für alle Typen, implementiert durch das Betriebssystem
  - ◆ Objekte besitzen eine Liste von Capabilities auf andere Objekte (genannt *C-List*)
  - ◆ Capabilities enthalten Rechte
  - ◆ Objekte besitzen einen Datenbereich (impl. durch geschütztes Segment)



1 Hydra (3)

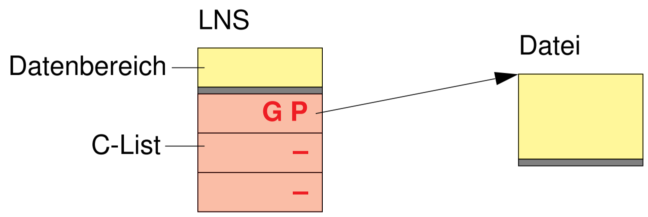
- Prozesse (Subjekte)
  - Prozesse besitzen einen aktuellen Kontext, den LNS *(Local name space)*
  - LNS ist ein Objekt
  - zum LNS gehört ein Aktivitätsträger (Thread)
  - Prozess kann nur auf Objekte zugreifen, die in der C-List seines LNS stehen (mehrstufige Zugriffe, z. B. auf die C-List eines Objekts, dessen Capabilities in der C-List des LNS steht, sind möglich → Pfad zur eigentlichen Capability)
- Capabilities
  - Prozesse können nur über Systemaufrufe ihre Capabilities bzw. ihre C-List bearbeiten
  - Capabilities können nicht gefälscht oder verfälscht werden
  - Betriebssystem kann sicheres Schutzkonzept basierend auf Capabilities implementieren

2 Datenzugriff in Hydra

- Operationen auf dem Datenbereich
  - *Getdata:* kopiere Abschnitt aus dem Datenbereich eines Objekts in den Datenbereich des LNS
  - *Putdata:* kopiere Abschnitt aus dem Datenbereich des LNS in den Datenbereich eines Objekts
  - *Adddata:* füge Daten zu dem Datenbereich eines Objekts hinzu
- Dazugehörige Rechte:
  - *GETRTS:* erlaubt den Aufruf von *Getdata*
  - *PUTRTS:* erlaubt den Aufruf von *Putdata*
  - *ADDRTS:* erlaubt den Aufruf von *Adddata*
- Rechte müssen in der Capability zum Objekt gesetzt sein

2 Datenzugriff in Hydra (2)

- Beispiel: Implementierung von Dateien
  - *Getdata* erlaubt das Lesen von Daten
  - *Putdata* erlaubt das Schreiben von Daten
  - *Adddata* erlaubt das Anhängen von Daten
- Entsprechende Rechte können pro Capability gesetzt werden



3 Zugriff auf Capabilities in Hydra

- Operationen auf der C-List eines Objekts:
  - *Load:* kopieren einer Capability aus der C-List eines Objekts in die C-List des LNS ($\triangleq$ Besorgen einer Objektreferenz von einem Nameserver)
  - *Store:* kopieren einer Capability aus der C-List des LNS in die C-List eines Objekts (dabei können Rechte maskiert werden)
  - *Append:* anfügen einer Capability in die C-List eines Objekts
  - *Delete:* löschen einer Capability aus der C-List eines Objekts
- Rechte:
  - *LOADRTS:* erlaubt Aufruf von *Load*
  - *STORTS:* erlaubt Aufruf von *Store*
  - *APPRTS:* erlaubt Aufruf von *Append*
  - *KILLRTS:* erlaubt Aufruf von *Delete*