# OPERATING SYSTEMS

## I/O Management, System Protection and Security

**Venkatesh Prasad**
Department of Computer Science

# OPERATING SYSTEMS

## System Protection – Implementation of Access Matrix, Access control, Access rights

**Venkatesh Prasad**
Department of Computer Science

# OPERATING SYSTEMS

## Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

# Implementation of Access Matrix

- Generally, a sparse matrix (i.e. most of the entries will be empty)
- **Option 1 – Global table**
  - Store ordered triples **<domain, object, rights-set>** in table
  - A requested operation M on object $O_j$ within domain $D_i$ -> search table for a triple $< D_i, O_j, R_k >$
    - with $M \in R_k$
    - If triple found, operation is allowed to continue; otherwise an exception or condition is raised
  - But table could be large -> won't fit in main memory
    - Virtual memory techniques can be used for managing this table
  - Difficult to group objects
    - consider an object that all domains can read, this object must have a separate entry in every domain)

# Implementation of Access Matrix (Cont.)

## Option 2 – Access lists for objects

- Each **column** implemented as an access list for one object i.e. specifying user names and the types of access allowed for each user (empty entries can be discarded)
- Resulting per-object list consists of ordered pairs **<domain, rights-set>** defining all domains with non-empty set of access rights for the object
- Easily extended to contain default set -> If M ∈ default set, also allow access
  - For efficiency, check the default set first and then search the access list

# Implementation of Access Matrix (Cont.)

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

# Implementation of Access Matrix (Cont.)

## Option 3 – Capability list for domains

- Instead of object-based (i.e column wise), list is domain based (i.e row wise)
- **Capability list** for domain is list of objects together with operations allowed on them
- Object represented by its name or address, called a **capability**
- Execute operation M on object $O_j$, process requests operation and specifies capability as parameter
  - **Possession of capability** means access is allowed
- Capability list associated with domain but never directly accessible to a process executing in that domain
  - Rather, protected object, maintained by OS and accessed by the user indirectly
  - Like a "secure pointer"
  - Idea can be extended up to the application level

# Implementation of Access Matrix (Cont.)



## Option 4 – Lock-key

- Compromise between access lists and capability lists
- Each object has a list of unique bit patterns, called **locks**
- Each domain has a list of unique bit patterns called **keys** (managed by the OS)
- Process in a domain can only access object if domain has a key that matches one of the locks

## Comparison of Implementations

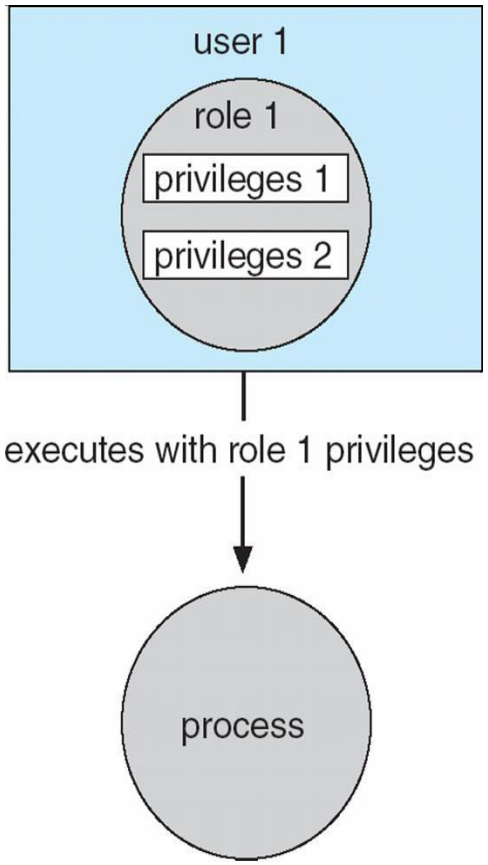
- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Access rights for a particular domain is non-localized, so difficult to determine the set of access rights for each domain
    - Every access to an object must be checked
      - Many objects and access rights -> slow (i.e not suitable for large system with long access lists)
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

## Comparison of Implementations (Cont.)

- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file recorded in a new entry in a file table (file table maintained by the OS such as UNIX and protection is ensured)

# Access Control

- Protection can be applied to non-file resources
- Oracle Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege
  - ***Privilege*** is right to execute system call or use an option (ex: write access for a file) within a system call
  - Can be assigned to processes
  - Users assigned ***roles*** granting access to privileges and programs
    - Enable role via password to gain its privileges
  - Similar to access matrix



## Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - **Immediate vs. delayed** (i.e. when revocation will occur)
  - **Selective vs. general** (i.e. select group of users or all the users)
  - **Partial vs. total** (i.e. subset of the rights or all the rights)
  - **Temporary vs. permanent** (can access right be revoked and obtained later?)
- Access List – Delete access rights from access list
  - **Simple** – search access list and remove entry, revocation is easy
  - **Immediate**, **general** or **selective**, **total** or **partial**, **permanent** or **temporary**

## Revocation of Access Rights (Cont.)

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete from each domain, with reacquire and denial if revoked by a process
  - **Back-pointers** – set of pointers from each object to all capabilities of that object , follow these pointers for revocation (adopted in Multics)
  - **Indirection** – capability points to global table entry which in turn points to object – delete entry from global table, selective revocation not allowed
  - **Keys** – unique bit pattern associated with a capability, generated when capability is created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key (with a new value)
    - Policy decision of who can create and modify keys – object owner or others?


# THANK YOU

**Venkatesh Prasad**
Department of Computer Science Engineering

**venkateshprasad@pes.edu**