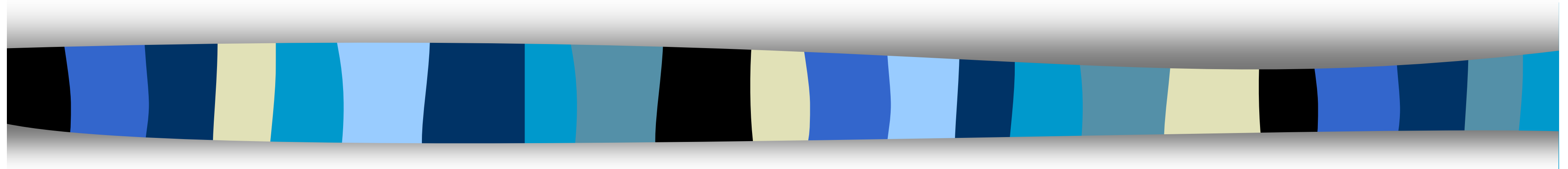
# Computer Security
# CS 426
## Lecture 9



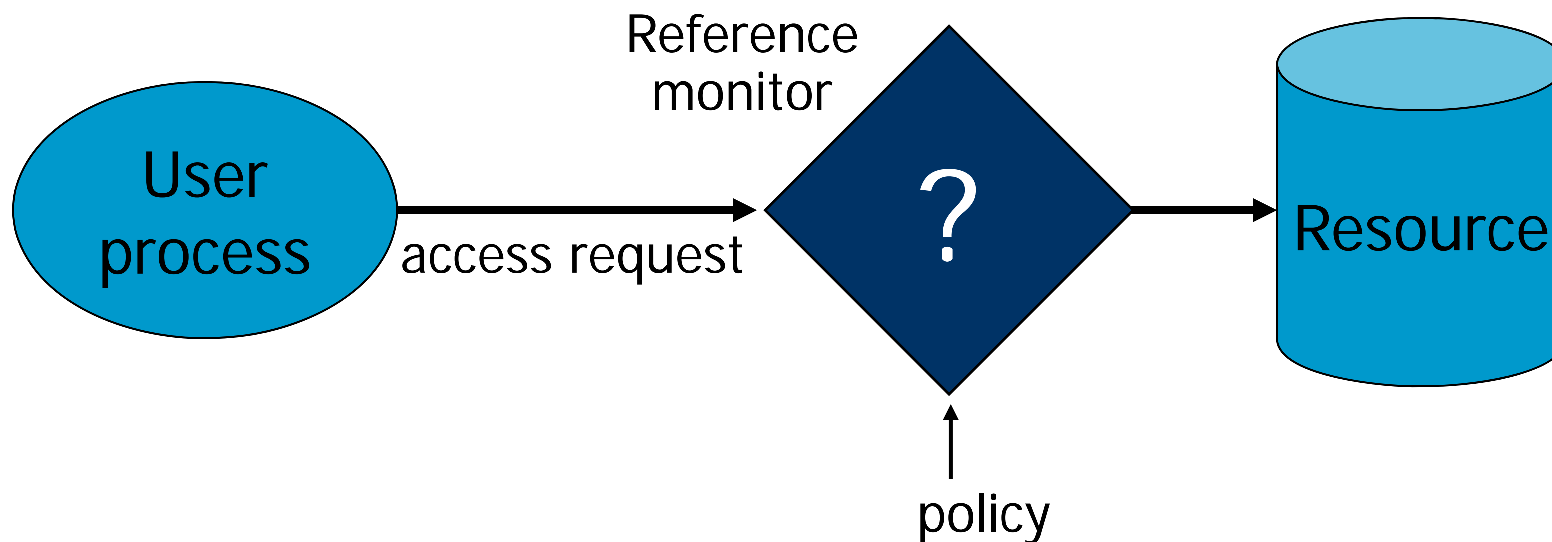
Unix Access Control

# Roadmap

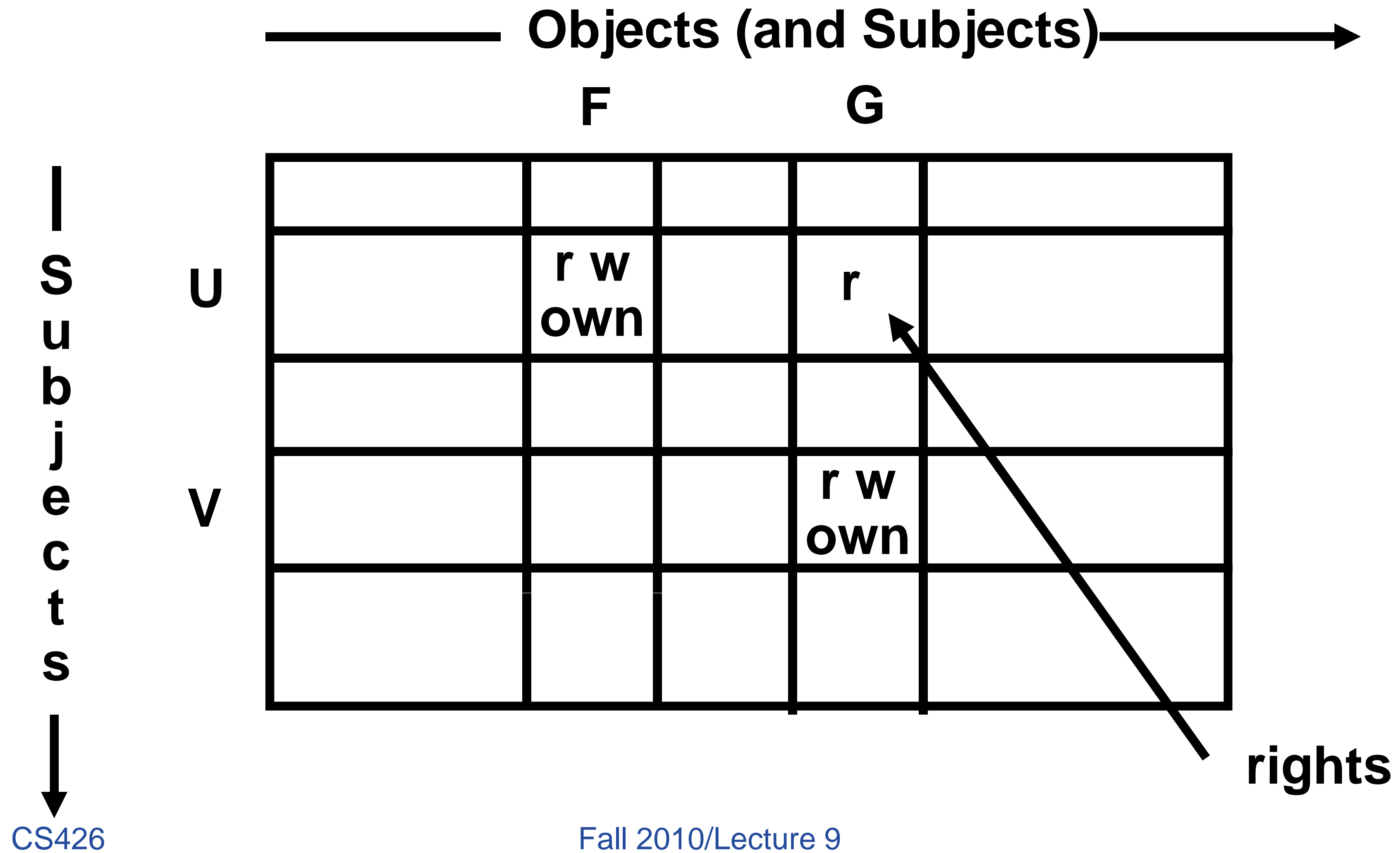
- Basic Concepts in Access Control & UNIX Access Control Overview
- Files in UNIX
- Processes in UNIX

# Access control

- A reference monitor mediates all access to resources
  - Tamper-proof:
  - Complete mediation: control **all** accesses to resources
  - Small enough to be analyzable



# ACCESS MATRIX MODEL



# ACCESS MATRIX MODEL

- Basic Abstractions
  - Subjects
  - Objects
  - Rights
- The rights in a cell specify the access of the subject (row) to the object (column)

# PRINCIPALS AND SUBJECTS

- A subject is a program (application) executing on behalf of some principal(s)
- A principal may at any time be idle, or have one or more subjects executing on its behalf

**What are subjects in UNIX?**

**What are principals in UNIX?**

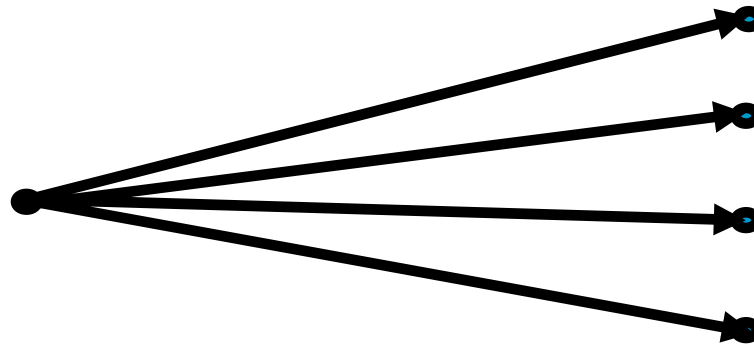
# OBJECTS

- An object is anything on which a subject can perform operations (mediated by rights)
- Usually objects are passive, for example:
  - File
  - Directory (or Folder)
  - Memory segment
- But, subjects can also be objects, with operations
  - kill
  - suspend
  - resume

# Basic Concepts of UNIX Access Control: Users, Groups, Files, Processes

- Each user account has a unique UID
  - The UID 0 means the super user (system admin)
- A user account belongs to multiple groups
- Subjects are processes
  - associated with uid/gid pairs, e.g., (euid, egid), (ruid, rgid), (suid, sgid)
- Objects are files

# USERS AND PRINCIPALS



**USERS**

**Real World User**

**PRINCIPALS**

**Unit of Access Control and Authorization**

the system authenticates the human user to a particular principal

# USERS AND PRINCIPALS

- There should be a one-to-many mapping from users to principals
  - a user may have many principals, but
  - each principal is associated with an unique user
- This ensures accountability of a user's actions

**What does the above imply in UNIX?**

# Roadmap

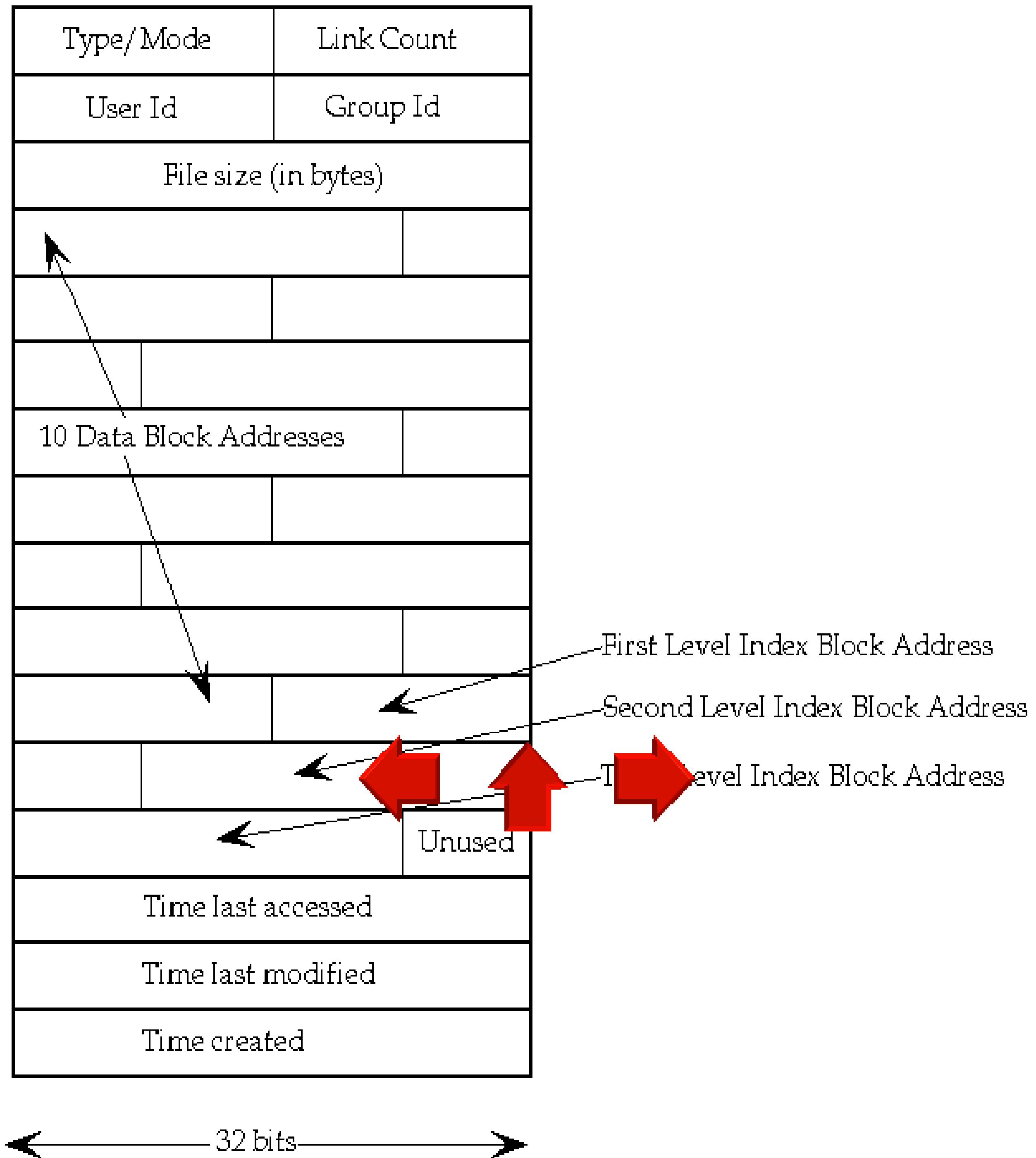
- Basic Concepts in Access Control & UNIX Access Control Overview
- Files in UNIX
- Processes in UNIX

# Organization of Objects

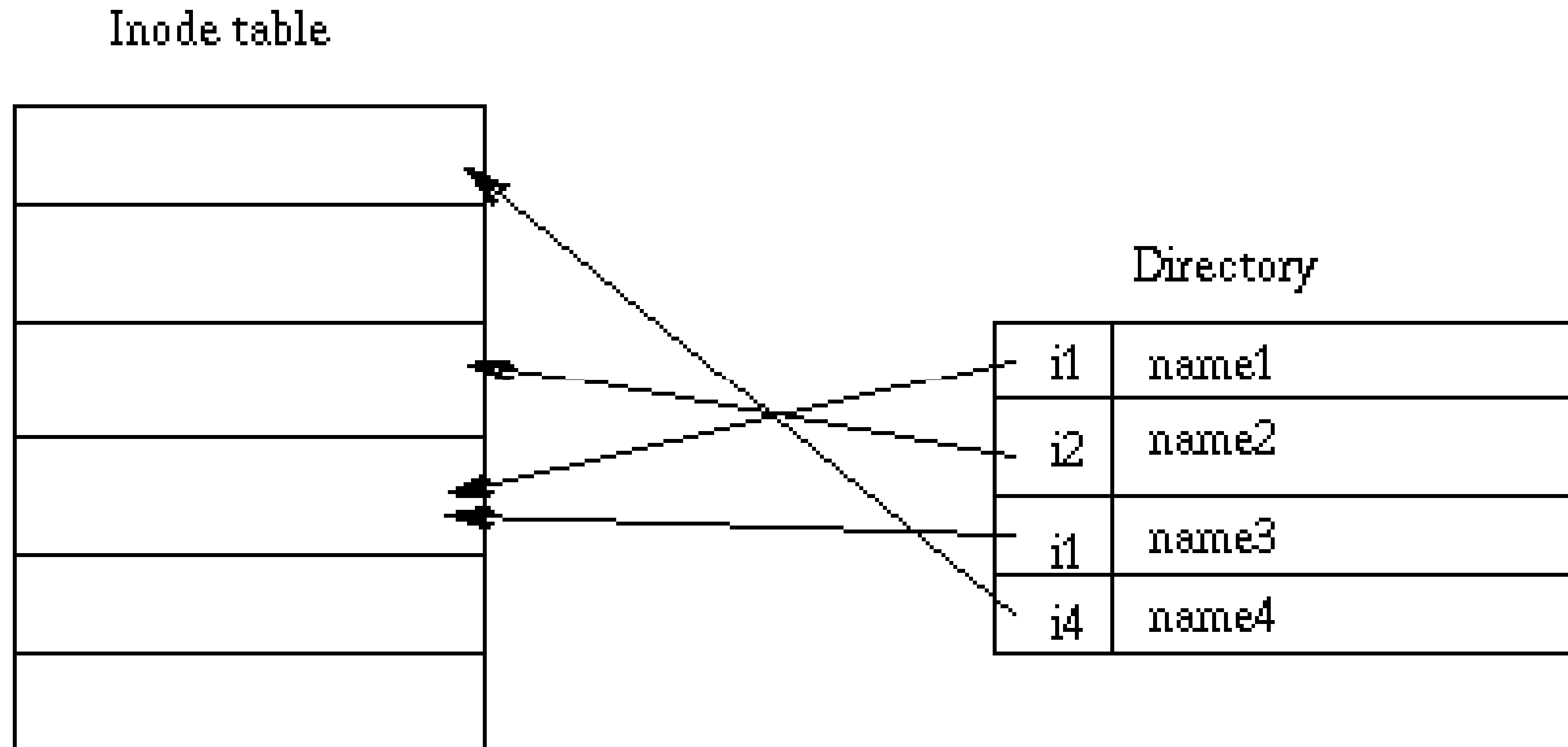
- Almost all objects are modeled as files
  - Files are arranged in a hierarchy
  - Files exist in directories
  - Directories are also one kind of files
- Each object has
  - owner
  - group
  - 12 permission bits
    - rwx for owner, rwx for group, and rwx for others
    - suid, sgid, sticky

# UNIX inodes:

Each file corresponds to an inode



# UNIX Directories



# Basic Permissions Bits on Files (Non-directories)

- Read controls reading the content of a file
  - i.e., the read system call

- Write controls changing the content of a file
  - i.e., the write system call

- Execute controls loading the file in memory and execute
  - i.e., the execve system call

# Execution of a file

- Binary file vs. script file
- Having execute but not read, can one run a binary file?
- Having execute but not read, can one run a script file?
- Having read but not execute, can one run a script file?

# Permission Bits on Directories

- Read bit allows one to show file names in a directory
- The execution bit controls traversing a directory
  - does a lookup, allows one to find inode # from file name
  - chdir to a directory requires execution
- Write + execution control creating/deleting files in the directory
  - Deleting a file under a directory requires no permission on the file
- Accessing a file identified by a path name requires execution to all directories along the path

# The suid, sgid, sticky bits

| | suid | sgid | sticky bit |
|---|---|---|---|
| non-executable files | no effect | affect locking (unimportant for us) | not used anymore |
| executable files | change euid when executing the file | change egid when executing the file | not used anymore |
| directories | no effect | new files inherit group of the directory | only the owner of a file can delete |

# Some Examples

- What permissions are needed to access a file/directory?
  - read a file: /d1/d2/f3
  - write a file: /d1/d2/f3
  - delete a file: /d1/d2/f3
  - rename a file: from /d1/d2/f3 to /d1/d2/f4
  - …
- File/Directory Access Control is by System Calls
  - e.g., open(2), stat(2), read(2), write(2), chmod(2), opendir(2), readdir(2), readlink(2), chdir(2), …

# The Three sets of permission bits

- Intuition:
  - if the user is the owner of a file, then the r/w/x bits for owner apply
  - otherwise, if the user belongs to the group the file belongs to, then the r/w/x bits for group apply
  - otherwise, the r/w/x bits for others apply
- Can one implement negative authorization, i.e., only members of a particular group are not allowed to access a file?

# Other Issues On Objects in UNIX

- Accesses other than read/write/execute
  - Who can change the permission bits?
    - The owner can
  - Who can change the owner?
    - Only the superuser
- Rights not related to a file
  - Affecting another process
  - Operations such as shutting down the system, mounting a new file system, listening on a low port
    - traditionally reserved for the root user

# Roadmap

- Basic Concepts in Access Control & UNIX Access Control Overview
- Files in UNIX
- Processes in UNIX

# Subjects vs. Principals

- Access rights are specified for users (accounts)
- Accesses are performed by processes (subjects)
- The OS needs to know on which users' behalf a process is executing

# Process User ID Model in Modern UNIX Systems

- Each process has three user IDs
  - real user ID (ruid) — owner of the process
  - effective user ID (euid) — used in most access control decisions
  - saved user ID (suid)
- and three group IDs
  - real group ID
  - effective group ID
  - saved group ID

# Process User ID Model in Modern UNIX Systems

- When a process is created by *fork*
  - it inherits all three users IDs from its parent process
- When a process executes a file by *exec*
  - it keeps its three user IDs unless the set-user-ID bit of the file is set, in which case the effective uid and saved uid are assigned the user ID of the owner of the file
- A process may change the user ids via system calls

# The Need for suid/sgid Bits

- Some operations are not modeled as files and require user id = 0
  - halting the system
  - bind/listen on “privileged ports” (TCP/UDP ports below 1024)
  - non-root users need these privileges
- File level access control is not fine-grained enough
- System integrity requires more than controlling who can write, but also how it is written

# Security Problems of Programs with suid/sgid

- These programs are typically setuid root
- Violates the least privilege principle
  - every program and every user should operate using the least privilege necessary to complete the job
- Why violating least privilege is bad?
- How would an attacker exploit this problem?
- How to solve this problem?

# Changing effective user IDs

- A process that executes a set-uid program can drop its privilege; it can
  - drop privilege permanently
    - removes the privileged user id from all three user IDs
  - drop privilege temporarily
    - removes the privileged user ID from its effective uid but stores it in its saved uid, later the process may restore privilege by restoring privileged user ID in its effective uid

# Access Control in Early UNIX

- A process has two user IDs: real uid and effective uid and one system call setuid
- The system call setuid(id)
  – when euid is 0, setuid set both the ruid and the euid to the parameter
  – otherwise, the setuid could only set effective uid to real uid
    - Permanently drops privileges
- A process cannot temporarily drop privilege

Setuid Demystified, In USENIX Security ‘ 02

# System V

- Added saved uid & a new system call
- The system call seteuid
  - if euid is 0, seteuid could set euid to any user ID
  - otherwise, could set euid to ruid or suid
    - Setting to ruid temp. drops privilege
- The system call setuid is also changed
  - if euid is 0, setuid functions as seteuid
  - otherwise, setuid sets all three user IDs to real uid

# BSD

- Uses ruid & euid, change the system call from setuid to setreuid
  - if euid is 0, then the ruid and euid could be set to any user ID
  - otherwise, either the ruid or the euid could be set to value of the other one
    - enables a process to swap ruid & euid

# Modern UNIX

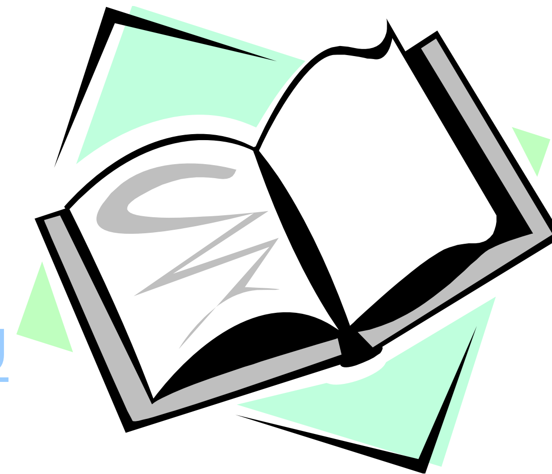
- System V & BSD affect each other, both implemented setuid, seteuid, setreuid, with different semantics
  - some modern UNIX introduced setresuid
- Things get messy, complicated, inconsistent, and buggy
  - POSIX standard, Solaris, FreeBSD, Linux

# Suggested Improved API

- Three method calls
  - drop_priv_temp
  - drop_priv_perm
  - restore_priv
- Lessons from this?
- Psychological acceptability principle
  - “human interface should be designed for ease of use”
  - the user’s mental image of his protection goals should match the mechanism

# Readings for This Lecture

- Wiki
  - Filesystem Permissions
- Other readings
  - UNIX File and Directory Permissions and Modes
    - http://www.hccfl.edu/pollock/AUnix1/FilePermissions.htm
  - Unix file permissions
    - http://www.unix.com/tips-tutorials/19060-unix-file-permissions.html



# Coming Attractions …

- Software vulnerabilities