

# CS-457

# Information Security

Module 6 Access Control

## a) Access Control at the Operating System

Based on Supplied Reading Material

# The 3 Types of Access Control Policies

## Discretionary Access Control (DAC)

- Based on Requestor's Identity + Access Rules
- Discretionary: Entity may grant others access to resources under its control

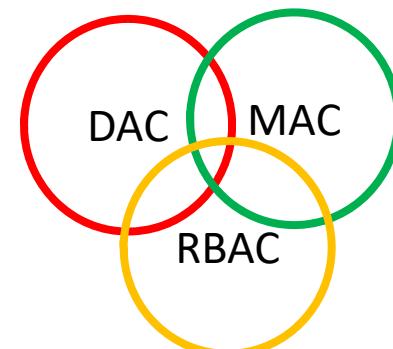
## Mandatory Access Control (MAC)

- Based on Resource Security Labels + Requestor's Security Clearance
- Mandatory: Entity may not simply grant access to others

## Role-Based Access Control (RBAC)

- Based on requestor's role + access rules
  - which role has what access right

May be implemented as stand alone, or as a mix



# Access Control: Subjects vs Objects

## Subjects (Requesters)

- Process (on behalf of some user)
- Device

## Objects (Resources being accessed)

- Data
- Memory
- CPU time
- Other Subjects treated as objects

# The Concept of an Access Control Matrix

- Each entry specifies the set of access “*rights*” a subject has on some object, e.g. a file

$$A[i,j] = \{ right_1, right_2, \dots, right_n \}$$

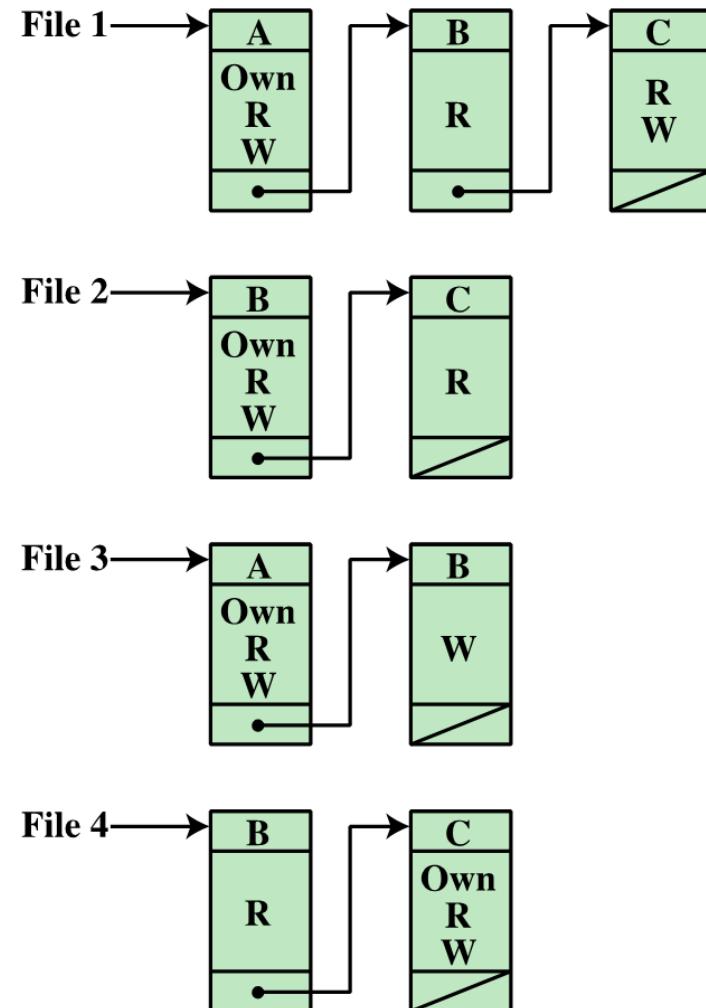
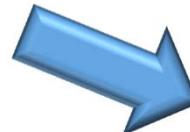
	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

**Subjects** (vertical axis) and **Objects** (horizontal axis) are indicated by red arrows.

- For a large number of subjects and objects, the matrix will be sparse, i.e. most entries will be empty.
- Instead of storing it as a matrix, use:
  - Access Control Lists
  - Capability Lists

# The Access Control List: *Object-Focused*

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

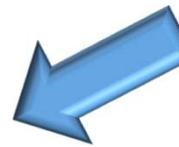
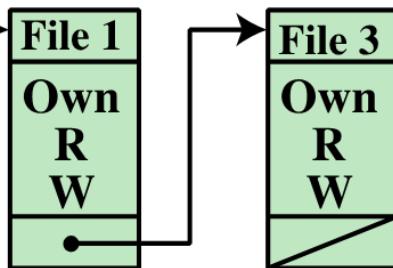


- For each object (e.g. a file), create a list of all subjects who have any type of access “right” on this object (file)

# The Capability List: *Subject-Focused*

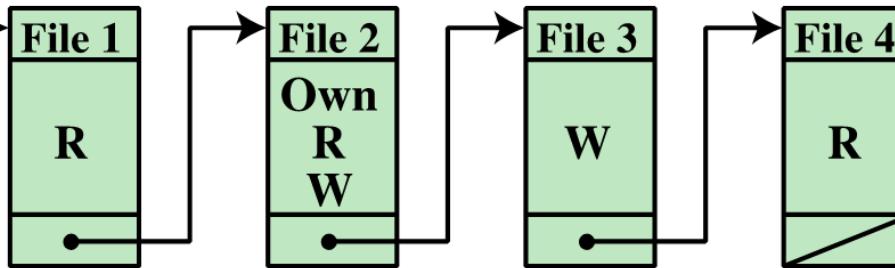
- For each subject (e.g. a user), create a list of all objects (e.g. files) on which it may have any type of access “right”

User A →

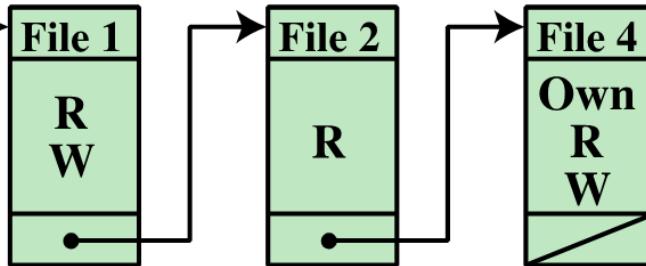


	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

User B →



User C →



# DAC: The Extended Access Control Matrix

- Subjects are also treated as objects:
  - There is a column for each subject just like any other object

		OBJECTS								
		subjects			files		processes		disk drives	
		S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>	F <sub>1</sub>	F <sub>2</sub>	P <sub>1</sub>	P <sub>2</sub>	D <sub>1</sub>	D <sub>2</sub>
SUBJECTS	S <sub>1</sub>	control	owner	owner control	read*	read owner	wakeup	wakeup	seek	owner
	S <sub>2</sub>		control		write*	execute			owner	seek *
	S <sub>3</sub>			control		write	stop			

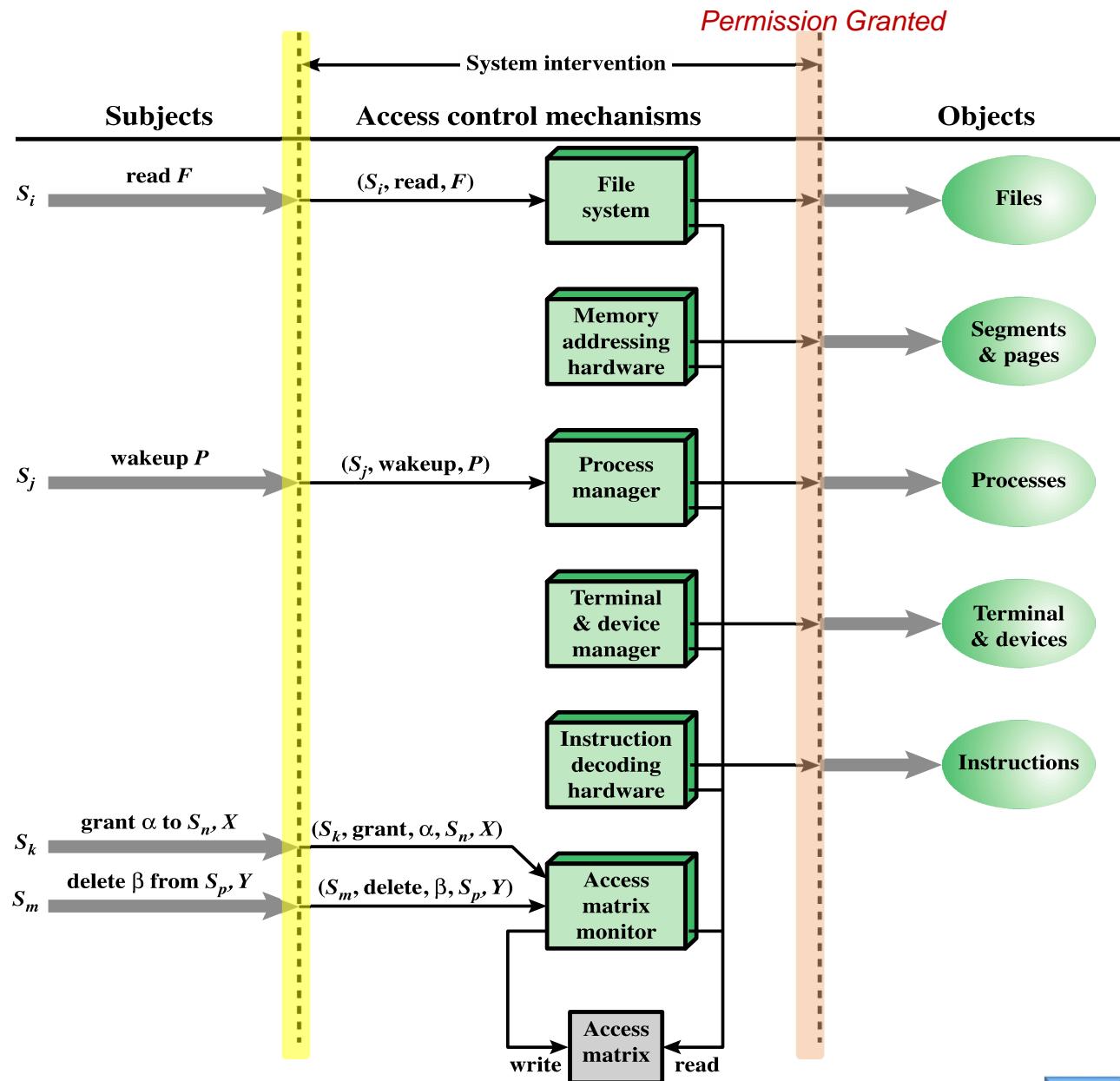
To authorize subject  $S_i$  attempting an access operation  $\alpha$  (e.g. *read*) on  $X$ , it must first be true that:

$$\alpha \in A[S_i, X]$$

*right\** means copy flag is set, so the subject can transfer that access right to other subjects

## DAC: Organization of the Access Control Function in the OS

For each type of objects,  
an interface module  
(highlighted in yellow)  
receives and reforms  
the request, then  
forwards it to the  
appropriate controller  
of that type of objects



# DAC: commands (as issued by $S_o$ ) and rules

(You may print this as Cheat Sheet)

Rule	Command (by $S_o$ )	Authorization	Operation
R1	<b>transfer</b> $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ <b>to</b> $S, X$	' $\alpha^*$ ' in $A[S_o, X]$	<b>insert</b> $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ in $A[S, X]$
R2	<b>grant</b> $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ <b>to</b> $S, X$	'owner' in $A[S_o, X]$	<b>insert</b> $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ in $A[S, X]$
R3	<b>delete</b> $\alpha$ <b>from</b> $S, X$	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete $\alpha$ from $A[S, X]$
R4	$w \leftarrow \text{read } S, X$  Tell me what $S$ can do with $X$	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into $w$
R5	<b>create object</b> $X$	None	add column for $X$ to $A$ ; store 'owner' in $A[S_o, X]$
R6	<b>destroy object</b> $X$	'owner' in $A[S_o, X]$	delete column for $X$ from $A$
R7	<b>create subject</b> $S$	none	add row for $S$ to $A$ ; execute <b>create object</b> $S$ ; store 'control' in $A[S, S]$
R8	<b>destroy subject</b> $S$	'owner' in $A[S_o, S]$	delete row for $S$ from $A$ ; execute <b>destroy object</b> $S$

# DAC Homework Exercise

Starting from this access control matrix of one row and one column

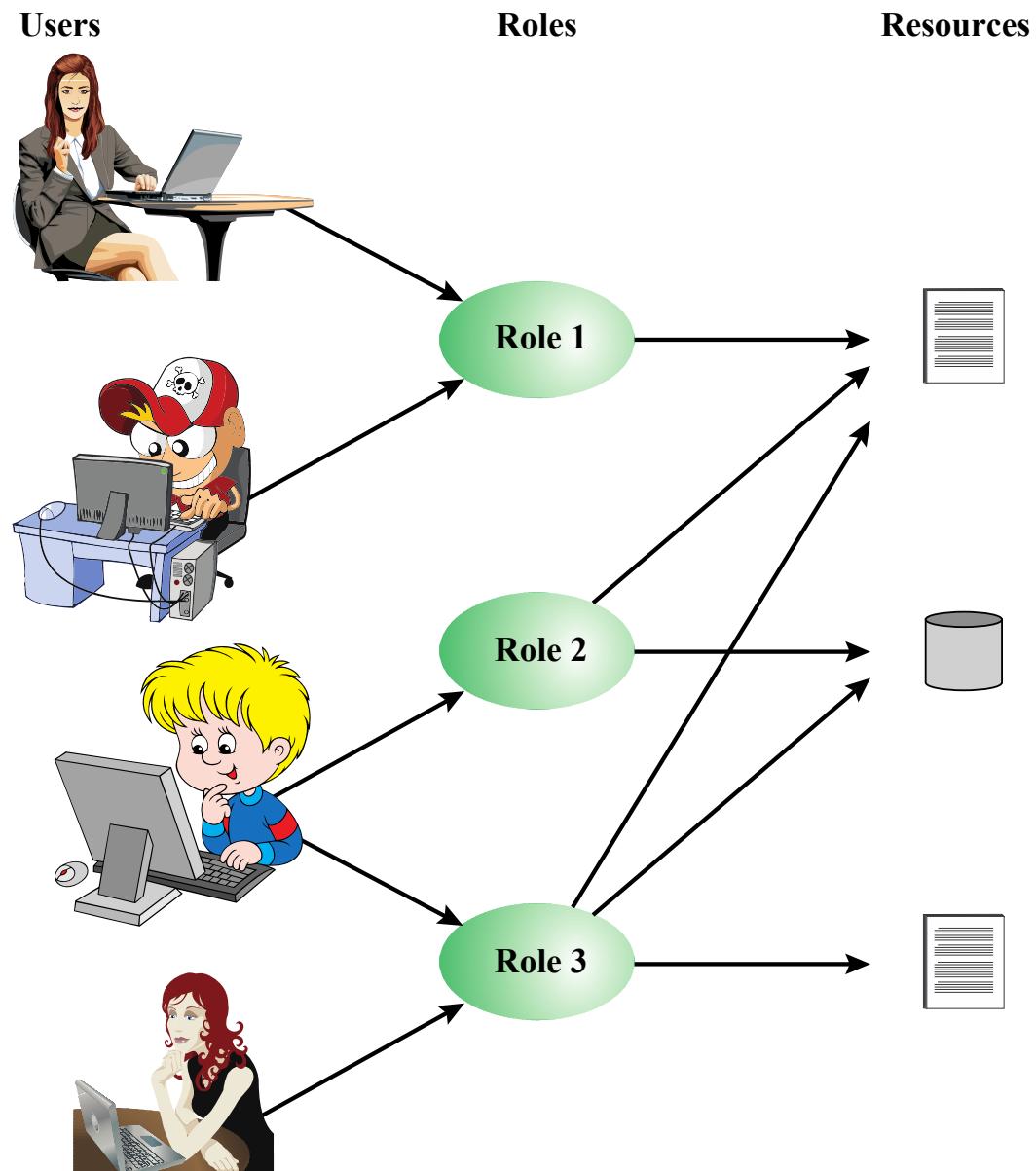


Show whether the following sequence of commands will be authorized (explain why/why not). If authorized, show the effect on the access control matrix.

1. ( root , create subject , Nancy)	9. ( root , grant write* , to Basma , F1)
2. ( root , create object , F1)	10. ( Basma , transfer write , to Nancy , F1)
3. ( root , read , F1)	11. ( root , write , F1)
4. ( root , grant read , to root , F1)	12. ( root , delete read , from Basma , F1)
5. ( root , read , F1)	13. ( root , grant control , to Nancy , Basma)
6. ( root , grant read , to Nancy , F1)	14. ( Basma , read , F1)
7. ( root , create subject , Basma)	15. ( Nancy , delete write , from Basma , F1)
8. ( Nancy , transfer read , to Basma , F1)	16. ( Nancy , destroy subject , Basma)

# Role-Based Access Control: RBAC

- ‘**Role**’ is a job function within some context (e.g. ‘manager’ in an organization)
- Access rights are assigned to roles (*many-to-many*)
- Users are given one or more roles (*many-to-many*)
  - Statically, i.e. permanently
  - Dynamically, i.e. when needed, then role is revoked

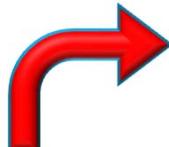


# RBAC Example

OBJECTS

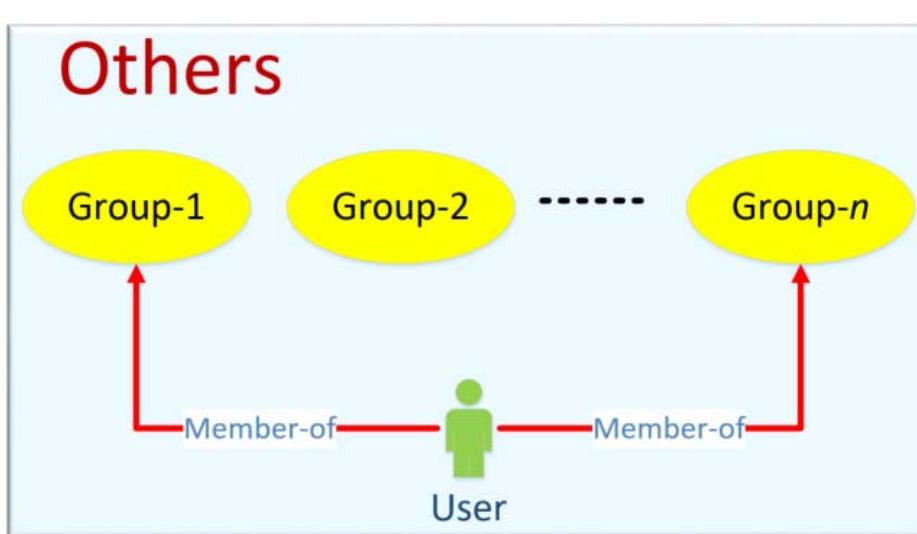
	R <sub>1</sub>	R <sub>2</sub>	R <sub>n</sub>	F <sub>1</sub>	F <sub>1</sub>	P <sub>1</sub>	P <sub>2</sub>	D <sub>1</sub>	D <sub>2</sub>
R <sub>1</sub>	control	owner	owner control	read*	read owner	wakeup	wakeup	seek	owner
R <sub>2</sub>		control		write*	execute			owner	seek *
•									
R <sub>n</sub>			control		write	stop			

ROLES

U <sub>1</sub>	X								
U <sub>2</sub>	X								
U <sub>3</sub>		X			X				
U <sub>4</sub>					X				
U <sub>5</sub>					X				
U <sub>6</sub>						X			
•									
U <sub>m</sub>	X								

# Traditional UNIX File Access Control

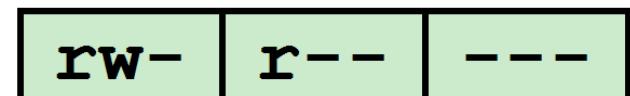


Q

chown: changes owner  
chgrp: changes group

*Highest relevant is applied*

Owner class      Group class      Other class



9 (of the 12)  
protection bits each  
file has

{  
user: :rw-  
group: :r--  
other: :---

0640 in Octal

110

100

000

# File vs. Directory Access Modes

File Access Modes	Directory Access Modes
<ul style="list-style-type: none"><li>• <b>Read</b> Grants the capability to read, i.e., view the contents of the file.</li><li>• <b>Write</b> Grants the capability to modify, or remove the content of the file.</li><li>• <b>Execute</b> User with execute permissions can run a file as a program.</li></ul>	<ul style="list-style-type: none"><li>• <b>Read</b> Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.</li><li>• <b>Write</b> Access means that the user can add or delete files from the directory.</li><li>• <b>Execute</b> A user must have <b>execute</b> access to the a directory in order to execute a program inside that directory.</li></ul>

`chmod`: changes protection bits

# Using the chmod command

```
$chmod o+wx,u-x,g = rx testfile  
$ls -l testfile  
-rw-r-xrwx 1 amrood users 1024 Nov 2 00:10 testfile
```

```
$ chmod 755 testfile  
$ls -l testfile  
-rwxr-xr-x 1 amrood users 1024 Nov 2 00:10 testfile  
$chmod 743 testfile  
$ls -l testfile  
-rwxr--wx 1 amrood users 1024 Nov 2 00:10 testfile  
$chmod 043 testfile  
$ls -l testfile  
----r---wx 1 amrood users 1024 Nov 2 00:10 testfile
```

# Traditional UNIX File Access Control: other protection bits

## SetUID bit

- *executable files*:
  - temporarily apply rights of file's creators to current executing user, if needed.
- *directory*:
  - ignored

## SetGIU bit

- *executable files*:
  - temporarily apply rights of file's group to current executing user, if needed.
- *directory*:
  - newly-created files will inherit group of this director

## Sticky bit

- *files*
  - outdated use
- *directory* (shared)
  - for each inside file, only the owner can rename, delete, or move this file

## Traditional UNIX File Access Control: *Super User*

- Unrestricted access to ALL files, system-wide.
- Carefully develop programs owned by “superuser”, especially if they have the SetUID bit on

Traditional Protection Bits (a.k.a. *minimal access control lists*) are adequate for small number of users and groups.

For larger systems, UNIX uses *extended Access Control Lists*

If interested, search for the many videos on YouTube on Unix ACLs