Operating Systems Security

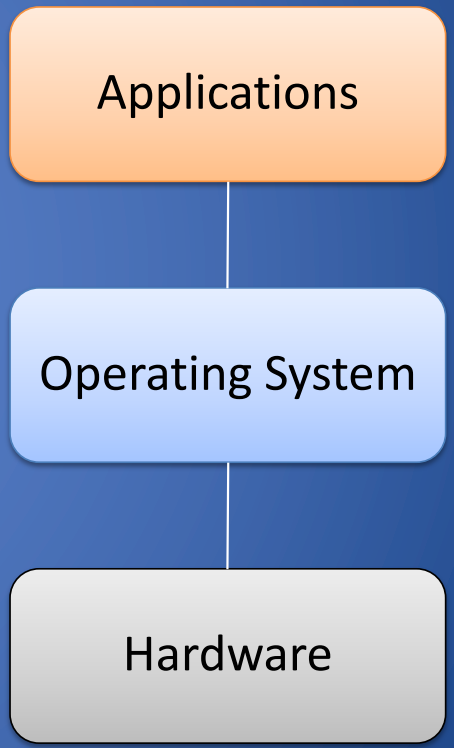
CS 1660: Introduction to Computer Systems Security

What is an Operating System?

How applications (and you) interface with the hardware

Provides abstractions for using:

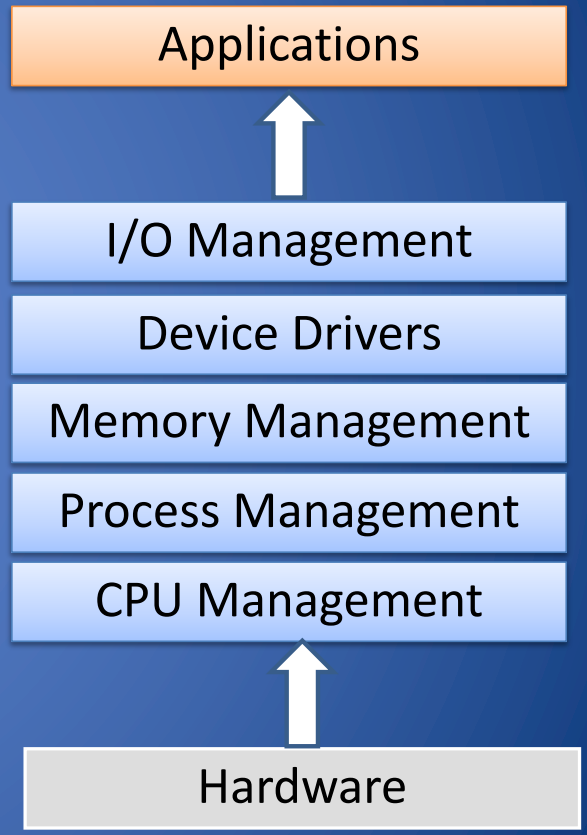
- CPU
- Memory
- Files / folders
- Windows
- Network
- Cursor
- Application

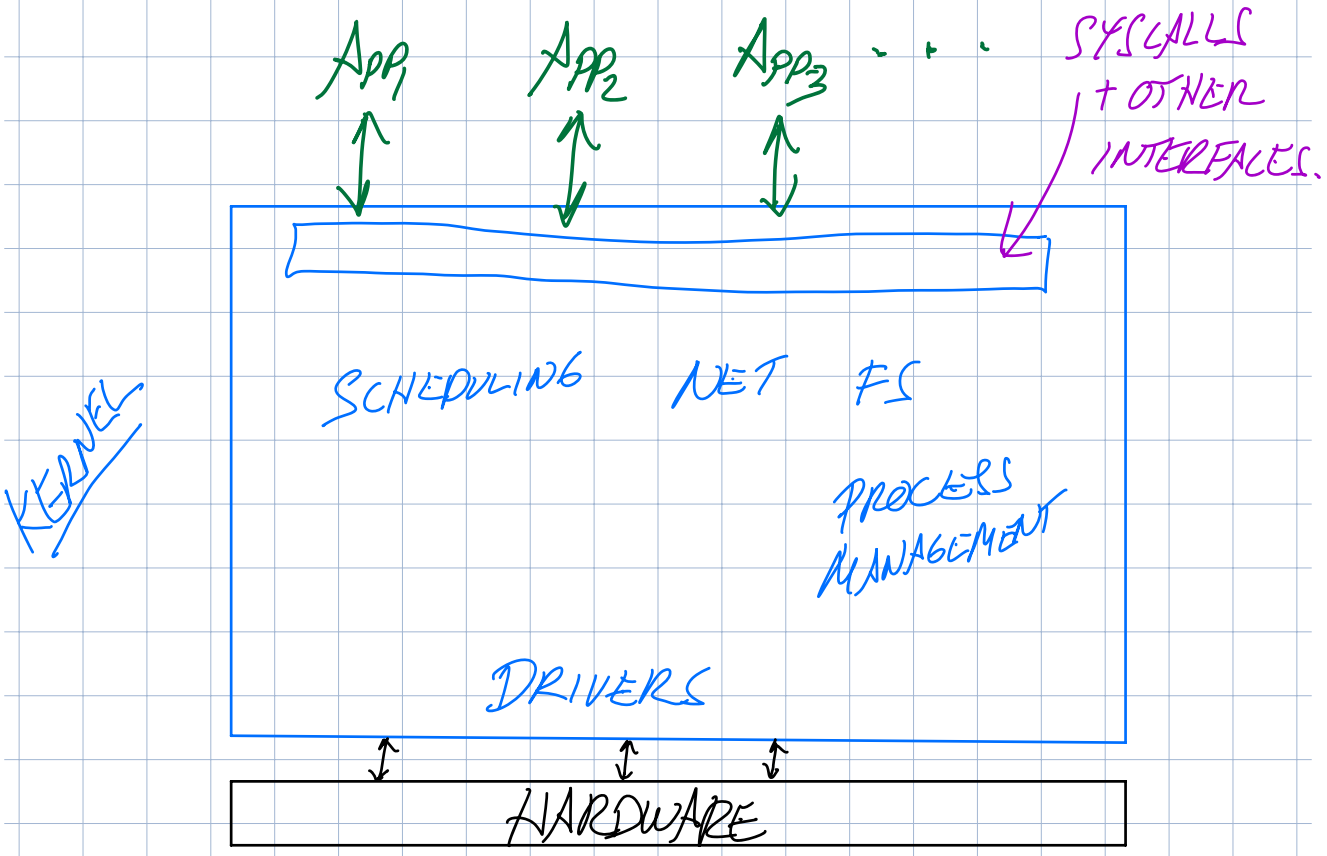


Operating System Layers

The OS has many layers of abstraction:

- The kernel: core of the OS, responsible for controlling hardware, resource access
  - Various subsystems (memory management, networking, storage, ...)
- Execution modes:
  - user mode: access to resources mediated by the kernel
  - kernel mode: full and direct access to resources




App1
App2
App3
SYSCALLS
+ OTHER
INTERFACES.
KERNEL
SCHEDULING
NET
FS
PROCESS
MANAGEMENT
DRIVERS
HARDWARE

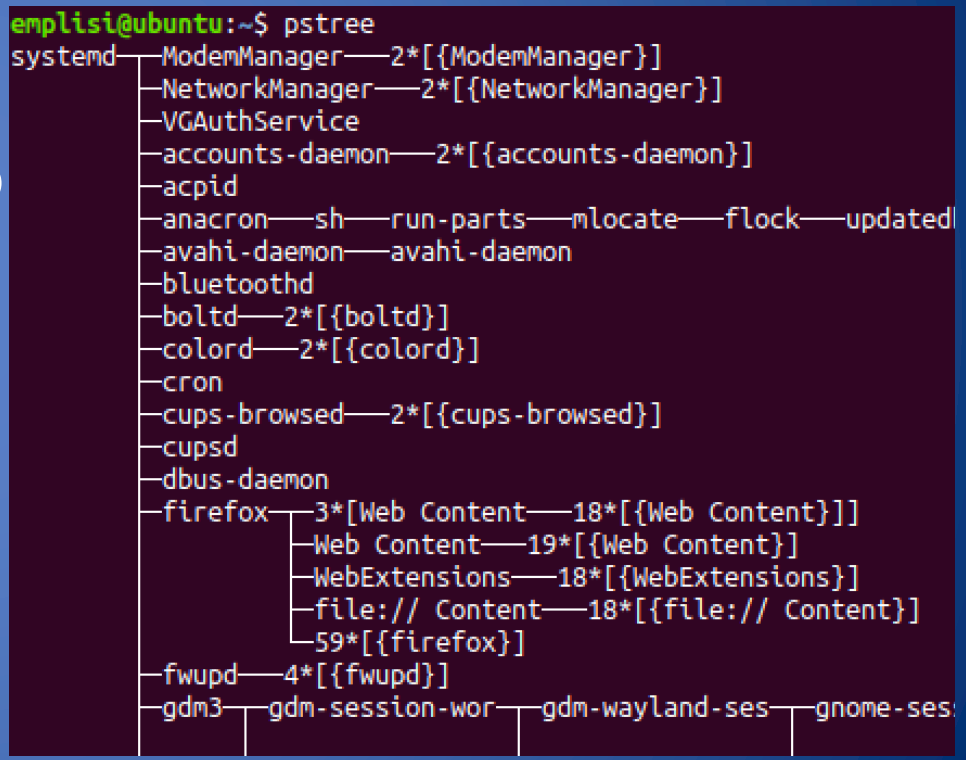
Processes

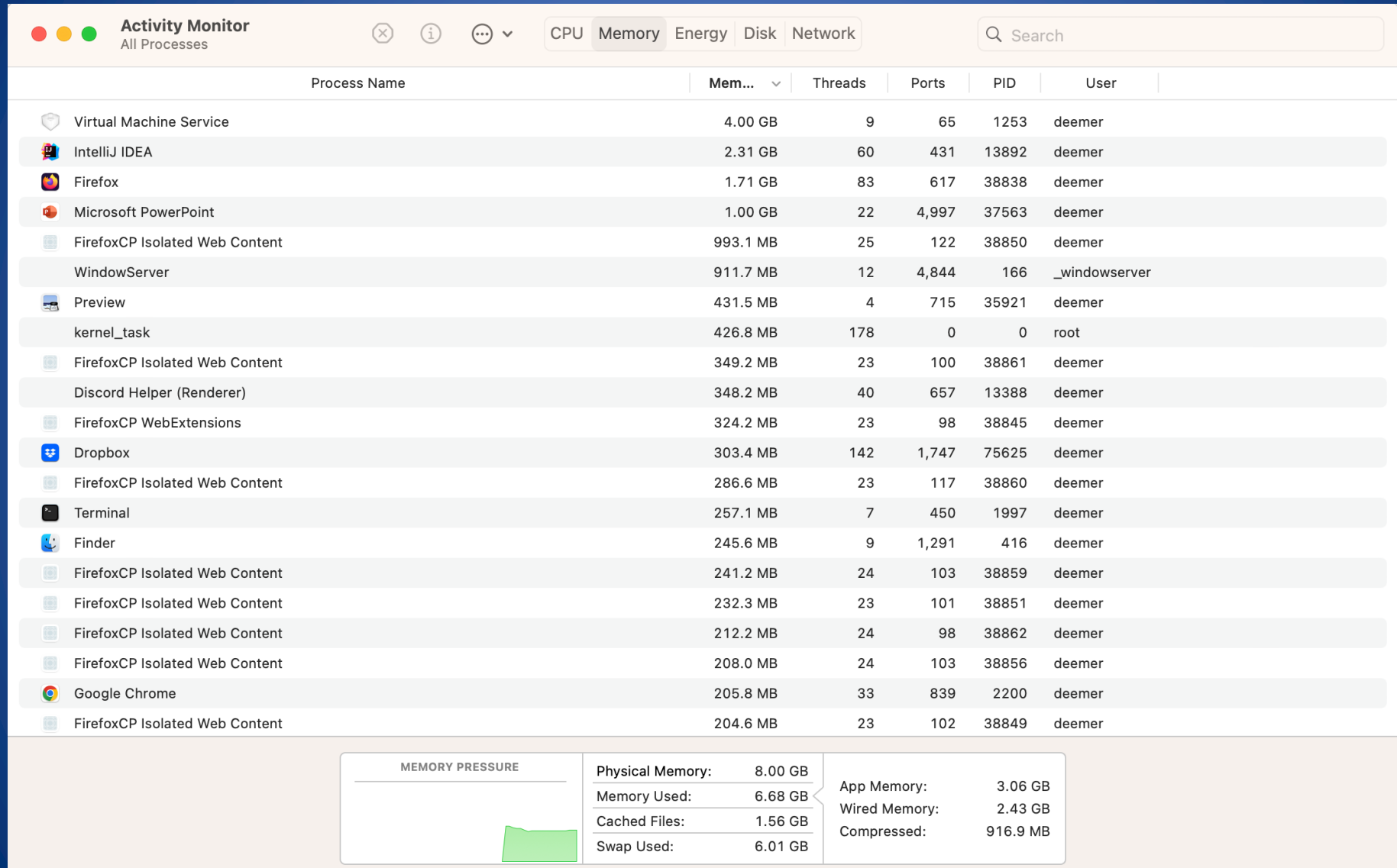
- The kernel manages applications as processes (or threads)
- An application can have multiple processes
- Every process has
  - Process ID (PID)
  - Effective user
- Kernel provides
  - Separate address space from other processes
  - Time/resource sharing between other running processes

Processes as a Tree

- Node: process
- Edge: connect a process (child) to the one that started it (parent)
- Child process inherits some context from parent process

Linux: PID 1 is the first process, which starts others at boot time




Activity Monitor
All Processes
CPU
Memory
Energy
Disk
Network
Search
Process Name
Mem...
Threads
Ports
PID
User
Virtual Machine Service 4.00 GB 9 65 1253 deemer
IntelliJ IDEA 2.31 GB 60 431 13892 deemer
Firefox 1.71 GB 83 617 38838 deemer
Microsoft PowerPoint 1.00 GB 22 4,997 37563 deemer
FirefoxCP Isolated Web Content 993.1 MB 25 122 38850 deemer
WindowServer 911.7 MB 12 4,844 166 _windowserver
Preview 431.5 MB 4 715 35921 deemer
kernel_task 426.8 MB 178 0 0 root
FirefoxCP Isolated Web Content 349.2 MB 23 100 38861 deemer
Discord Helper (Renderer) 348.2 MB 40 657 13388 deemer
FirefoxCP WebExtensions 324.2 MB 23 98 38845 deemer
Dropbox 303.4 MB 142 1,747 75625 deemer
FirefoxCP Isolated Web Content 286.6 MB 23 117 38860 deemer
Terminal 257.1 MB 7 450 1997 deemer
Finder 245.6 MB 9 1,291 416 deemer
FirefoxCP Isolated Web Content 241.2 MB 24 103 38859 deemer
FirefoxCP Isolated Web Content 232.3 MB 23 101 38851 deemer
FirefoxCP Isolated Web Content 212.2 MB 24 98 38862 deemer
FirefoxCP Isolated Web Content 208.0 MB 24 103 38856 deemer
Google Chrome 205.8 MB 33 839 2200 deemer
FirefoxCP Isolated Web Content 204.6 MB 23 102 38849 deemer
MEMORY PRESSURE
Physical Memory: 8.00 GB
Memory Used: 6.68 GB
Cached Files: 1.56 GB
Swap Used: 6.01 GB
App Memory: 3.06 GB
Wired Memory: 2.43 GB
Compressed: 916.9 MB

View Processes in Linux

- ps: displays snapshot of running processes
  - ps -ef : show all processes
  - ps -u <username>: show processes for a user
- Top, htop: Fancyier list of processes
  - top -u <username>: filter by username
- kill <pid>: Send signal to a process (usually to terminate it)

```
metcalfe /u/lmeyerov % ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2005 ?        00:00:01 init [2]
root         2     1  0  2005 ?        00:00:00 [ksoftirqd/0]
root         3     1  0  2005 ?        00:00:00 [events/0]
root         4     3  0  2005 ?        00:00:00 [khelper]
root        31     3  0  2005 ?        00:00:00 [kblockd/0]
root       104     3  0  2005 ?        00:00:01 [pdflush]
root       105     3  0  2005 ?        00:00:00 [pdflush]
root       107     3  0  2005 ?        00:00:00 [aio/0]
root       106     1  0  2005 ?        00:00:03 [kswapd0]
root       694     1  0  2005 ?        00:00:00 [kseriod]
root       745     3  0  2005 ?        00:00:00 [ata/0]
root       750     1  0  2005 ?        00:00:00 [scsi_eh_2]
root       751     1  0  2005 ?        00:00:00 [scsi_eh_3]
root       769     1  0  2005 ?        00:00:02 [kjournald]
root      1157     1  0  2005 ?        00:00:00 [khubd]
root      1263     1  0  2005 ?        00:00:00 [kjournald]
root      1264     1  0  2005 ?        00:00:00 [kjournald]
root      1265     1  0  2005 ?        00:00:00 [kjournald]
root      1266     1  0  2005 ?        00:00:06 [kjournald]
root      1521     1  0  2005 ?        00:00:00 [khpsbpkt]
root      1584     1  0  2005 ?        00:00:00 [knodemgrd_0]
root      2813     1  0  2005 ?        00:00:00 dhclient -e -pf /var/run/dhclien
```

```
top - 14:23:25 up 41 days, 20:45, 10 users,  load average: 0.02, 0.12, 0.19
Tasks: 142 total,   1 running, 140 sleeping,   0 stopped,   1 zombie
Cpu(s):  3.7% us,  0.7% sy,  0.0% ni, 95.7% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:   1034436k total,   910324k used,   124112k free,   183132k buffers
Swap:  1048816k total,     4892k used,  1043924k free,   222260k cached
Which user (blank for all):
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4104 root       5 -10  180m  50m 6724 S  1.0  5.0   2:42.53 XFree86
13091 lmeyerov  15   0  132m  72m  26m S  1.0  7.2  17:22.59 mozilla-bin
21114 lmeyerov  15   0 36876  17m  13m S  1.0  1.7   0:01.38 ksnapshot
24555 lmeyerov  15   0 21128 7508 5460 S  0.7  0.7   0:00.79 artsd
21270 lmeyerov  17   0  2036 1116  860 R  0.7  0.1   0:00.02 top
    1 root      16   0  1504  528  468 S  0.0  0.1   0:01.32 init
    2 root      35  19     0    0    0 S  0.0  0.0   0:00.28 ksoftirqd/0
    3 root       5 -10     0    0    0 S  0.0  0.0   0:00.85 events/0
    4 root       6 -10     0    0    0 S  0.0  0.0   0:00.00 khelper
   31 root       5 -10     0    0    0 S  0.0  0.0   0:00.61 kblockd/0
  104 root      15   0     0    0    0 S  0.0  0.0   0:01.81 pdflush
  105 root      15   0     0    0    0 S  0.0  0.0   0:00.32 pdflush
  107 root      15 -10     0    0    0 S  0.0  0.0   0:00.00 aio/0
  106 root      15   0     0    0    0 S  0.0  0.0   0:03.65 kswapd0
  694 root      25   0     0    0    0 S  0.0  0.0   0:00.00 kseriod
  745 root       6 -10     0    0    0 S  0.0  0.0   0:00.00 ata/0
  750 root      18   0     0    0    0 S  0.0  0.0   0:00.00 scsi_eh_2
```

Process Management

- Each process has a context, which includes the user, parent process, and address space (for storing data and instructions)

- Access control policies determine which resources can be accessed by each process and which other processes it can control

System Calls (Syscalls)

- Primary way processes interact with kernel
- OS provides a "library" of syscalls for nearly all OS functions
  - Files: **`read`**, **`write`**, **`open`**, **`close`**, **`chmod`**, **`...`**
  - Process management: **`fork`**, **`clone`**, **`kill`**, **`...`**
  - Networking: **`socket`**, **`bind`**, **`connect`**, **`...`**

- On syscall, process "yields" to kernel, executes syscall in kernel mode

→ TRANSITION BETWEEN OS & KERNEL MODE.

System Services (Daemons)

- Packground process that performs common tasks for system
- Typically start at boot time
- Could run with higher permissions than users
- Typical services:
  - Remote SSH connections
  - Web servers
  - Logging

Authentication, Authorization, Accounting

AAA (recap)

- Authorization is the function of specifying access rights to resources (access control)
- To authorize => to define access policy

How do we do this in an OS?

Users

- Each process is associated with a user
- Specific users can have more privileges than regular users
  - Install or remove programs
  - Change rights of other users
  - Modify the configuration of the system
- Unix:
  - The root is a “super-user” with no restrictions

HOW TO CONTROL WHAT USERS CAN ACCESS?
⇒ AUTHORIZATION

HOW TO KNOW WHAT A USER IS DOING?
⇒ ACCOUNTING.

Discretionary Access Control (DAC)

- Users can protect what they own
  - The owner may grant access to others
  - The owner may define the type of access (read/write/execute) given to others
- DAC is the standard model used in operating systems
- Mandatory Access Control (MAC)
  - Multiple levels of security for users and documents (i.e. confidential, restricted, secret, top secret)
  - A user can create documents with just their level of security

General Principles

- Files and folders are managed by the operating system
- Applications, including shells, access files through an API
- Access control entry (ACE)
  - Allow/deny a certain type of access to a file/folder by user/group
- Access control list (ACL)
  - Collection of ACEs for a file/folder

Access Control Entries and Lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a sorted list of zero or more Access Control Entries (ACEs)
- An ACE refers specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group
- Examples of ACEs for folder "Bob's CS166 Grades"
  - Bob; Read; Allow
  - TAs; Read; Allow
  - TWD; Read, Write; Allow
  - Bob; Write; Deny
  - TAs; Write; Allow

WINDOWS

NEED TO "COMBINE" OR "COMPOSE" ENTRIES TO FIGURE OUT ACCESS

⇒ NEED TO "COMBINE" OR "COMPOSE" RULES TO FIGURE OUT ACL

Closed vs. Open Policy

"DEFAULT" RULES.

Closed policy

- Also called “default secure”

- Give Tom read access to “foo”
- Give Bob r/w access to “bar
- Tom: I would like to read “foo”
  - Access allowed
- Tom: I would like to read “bar”
  - Access denied

Open Policy

- Deny Tom read access to “foo”
- Deny Bob r/w access to “bar”
- Tom: I would like to read “foo”
  - Access denied
- Tom: I would like to read “bar”
  - Access allowed

Closed Policy with Negative Authorizations and Deny Priority

- Give Tom r/w access to “bar”
- Deny Tom write access to “bar”
- Tom: I would like to read “bar”
  - Access allowed
- Tom: I would like to write “bar”
  - Access denied
- Policy is used by Windows to manage access control to the file system

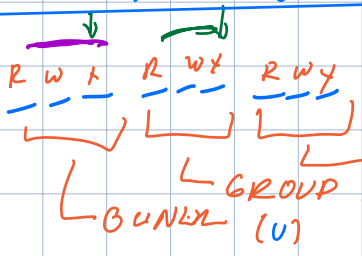
Role-Based Access Control

- Within an organization roles are created for various job functions
- The permissions to perform certain operations are assigned to specific roles
- Users are assigned particular role, with which they acquire the computer authorizations
- Users are not assigned permissions directly, but only acquire them through their role

U.S. Navy image in the public domain.

Access Control: File System

UNIX PERMISSIONS: AN OVERVIEW



A COUPLE OF OTHER BITS

- STICKY (T): ONLY OWNER CAN MOVE/RENAME
- SETUID
  - RUNS WITH PERMISSIONS OF OWNER, REGARDLESS OF WHO IS RUNNING IT.
  (SHOWS AS "S" IN OWNER'S X BIT
  ↑ "ELEVATING" PERMISSIONS
  ↓
- SETGID FOR FILES
  - RUNS WITH PERMISSIONS OF GROUP, REGARDLESS OF WHO IS RUNNING IT.
  - FOR DIRECTORY: NEW FILES MADE IN THIS DIRECTORY HAVE THIS GROUP.
  ("INHERITING" PERMISSIONS)

UNIX PERMISSIONS (DISCRETIONARY ACCESS CONTROL)

- ALL FILES/DIRS HAVE ONE OWNER, ONE GROUP BY DEFAULT
- ONLY OWNER CAN CHANGE PERMISSIONS

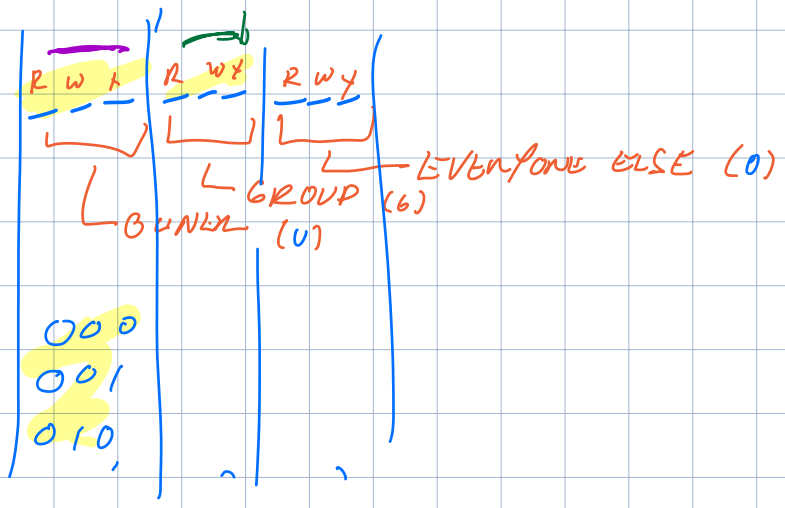
⇒ SOUND INFLEXIBLE? IT IS.

AN ALTERNATIVE MODEL: ACCESS CONTROL LIST

- DENY A: USER A : RW ←
- ALLOW
- ALLOW USER B : RWX
- ALLOW GROUP C : R
- DEFAULT: {R, RW, NONE, ...}

ROLE-BASED (RBAC)

- USED BY WINDOWS
- OPTIONAL ON LINUX



RWX	
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

CHMOD 644

6	4	4
110	100	100

RW- R-- R--

THE FOLLOWING SLIDES

HAVE MUCH MORE DETAIL THAN WE COVERED IN CLASS.

FEEL FREE TO USE THEM AS REFERENCE NOTES!

(WE'LL ALSO TALK ABOUT MORE EXAMPLES IN CLASS)

Linux vs. Windows

- Linux
  - Allow-only ACEs
  - Access to file depends on ACL of file and of all its ancestor folders
  - Start at root of file system
  - Traverse path of folders
  - Each folder must have execute (cd) permission
  - Different paths to same file not equivalent
  - File's ACL must allow requested access
- Windows
  - Allow and deny ACEs
  - By default, deny ACEs precede allow ones
  - Access to file depends only on file's ACL
  - ACLs of ancestors ignored when access is requested
  - Permissions set on a folder usually propagated to descendants (inheritance)
  - System keeps track of inherited ACE's

Linux File Access Control

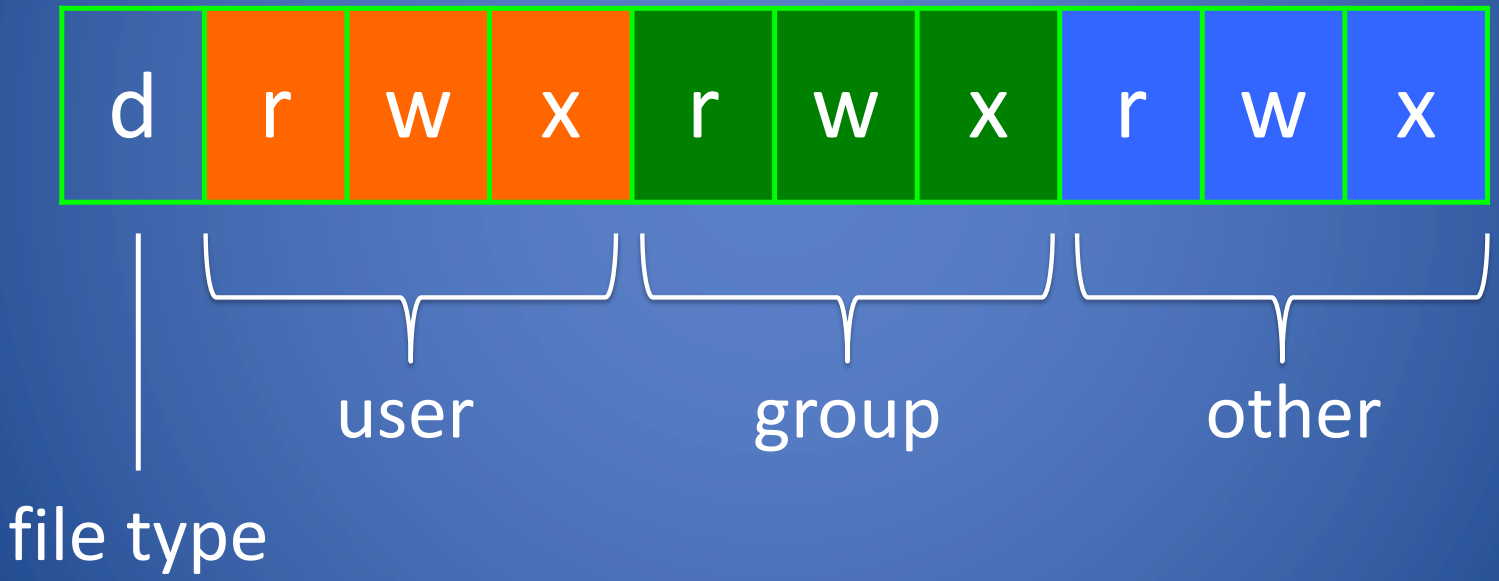
- File Access Control for:
  - Files
  - Directories
  - Therefore…
    - \dev\ : *devices*
    - \mnt\ : *mounted file systems*
    - What else? *Sockets, pipes, symbolic links*…

Unix Permissions

- Standard for all UNIXes
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation
- To see permissions, use `ls -l`

```
jk@sphere:~/test$ ls -l
total 0
-rw-r-----  1 jk ugrad 0 2005-10-13 07:18 file1
-rwxrwxrwx  1 jk ugrad 0 2005-10-13 07:18 file2
```

Unix File Types and Basic Permissions



Permissions Examples (Regular Files)

-rw-r—r--	read/write for owner, read-only for everyone else
-rw-r-----	read/write for owner, read-only for group, forbidden to others
-rwx------	read/write/execute for owner, forbidden to everyone else
-r--r--r--	read-only to everyone, including owner
-rwxrwxrwx	read/write/execute to everyone

Permissions for Directories

- Permissions bits interpreted differently for directories
- *Read* bit allows listing names of files in directory, but not their properties like size and permissions
- *Write* bit allows creating and deleting files within the directory
- *Execute* bit allows entering the directory and getting properties of files in the directory
- Lines for directories in `ls -l` output begin with d, as below:

```
jk@sphere:~/test$ ls -l
Total 4
drwxr-xr-x  2 jk ugrad 4096 2005-10-13 07:37 dir1
-rw-r--r--  1 jk ugrad    0 2005-10-13 07:18 file1
```

Permissions Examples (Directories)

drwxr-xr-x	all can enter and list the directory, only owner can add/delete files
drwxrwx---	full access to owner and group, forbidden to others
drwx--x---	full access to owner, group can access known filenames in directory, forbidden to others
-rwxrwxrwx	full access to everyone

The /tmp Directory

- In Unix systems, directory /tmp is
  - Readable by any user
  - Writable by any user
  - Usually wiped on reboot
- Convenience
  - Place for temporary files used by applications
  - Files in /tmp are not subject to the user's space quota
- What could go wrong

Special Permission Bits

Three other important permission bits

- Set-user-ID (“suid” or “setuid”) bit
- Set-group-ID (“sgid” or “setgid”) bit
- Sticky bit

setuid bit: Set-user-ID

- On executable files, causes the program to run as file owner regardless of who runs it

- How to view: shown as s instead of x

  -rwsr-xr-x: setuid, executable by all

  -rwxr-xr-x: executable by all, but not setuid

  -rwSr--r--: setuid, but not executable - not useful

Setuid Programs

- Unix processes have two user IDs:
  - real user ID: user launching the process
  - effective user ID: user whose privileges are granted to the process
- An executable file can have the set-user-ID property (setuid) enabled
- If a user A executes setuid file owned by B, then the effective user ID of the process is B and not A

Setuid Programs

- System call setuid(uid) allows a process to change its effective user ID to uid
- Some programs that access system resources are owned by root and have the setuid bit set (setuid programs)
  - e.g., passwd and su
- Writing secure setuid programs is tricky because vulnerabilities may be exploited by malicious user actions

setgid bit: Set-group-ID

- On executable files, causes the program to run with the file's group, regardless of whether the user who runs it is in that group
- On directories, causes files created within the directory to have the same group as the directory, useful for directories shared by multiple users with different default groups

- How to view: replaces 7th character (x or -) with s (or S if not also executable)

  -rwxr-sr-x: setgid file, executable by all

  drwxrwsr-x: setgid directory; files within will have group of directory

  -rw-r-Sr--: setgid file, but not executable - not useful

Sticky Bit

- On directories, prevents users from deleting or renaming files they do not own
- Ignored for everything else
- In 10-character display, replaces 10th character (x or -) with t (or T if not also executable)

drwxrwxrwt: sticky bit set, full access for everyone

drwxrwx--T: sticky bit set, full access by user/group

drwxr--r-T: sticky, full owner access, others can read *(useless)*

Symbolic Link

- In Unix, a symbolic link (aka symlink) is a file that points to (stores the path of) another file
- A process accessing a symbolic link is transparently redirected to accessing the destination of the symbolic link
- Symbolic links can be chained, but not to form a cycle

- ln -s really_long_directory/even_longer_file_name myfile

Octal Notation

- Standard syntax is nice for simple cases, but bad for complex changes
  - Alternative is octal notation, i.e., three or four digits from 0 to 7
- Digits from left (most significant) to right(least significant):
  [special bits][user bits][group bits][other bits]
- Special bit digit =
  (4 if setuid) + (2 if setgid) + (1 if sticky)
- All other digits =
  (4 if readable) + (2 if writable) + (1 if executable)

Octal Notation Examples

644 or 0644	read/write for owner, read-only for everyone else
775 or 0775	read/write/execute for owner and group, read/execute for others
640 or 0640	read/write for owner, read-only for group, forbidden to others
2775	same as 775, plus setgid (useful for directories)
777 or 0777	read/write/execute to everyone *(dangerous!)*
1777	same as 777, plus sticky bit

Root

- “root” account is a super-user account, like Administrator on Windows
- Multiple roots possible
- File permissions do not restrict root
- This is *dangerous*, but necessary, and OK with good practices

Becoming Root

- su
  - Changes home directory, PATH, and shell to that of root, but doesn't touch most of environment and doesn't run login scripts
- sudo <command>
  - Run just one command as root
- su [-] <user>
  - Become another non-root user
  - Root does not require to enter password

Changing Permissions

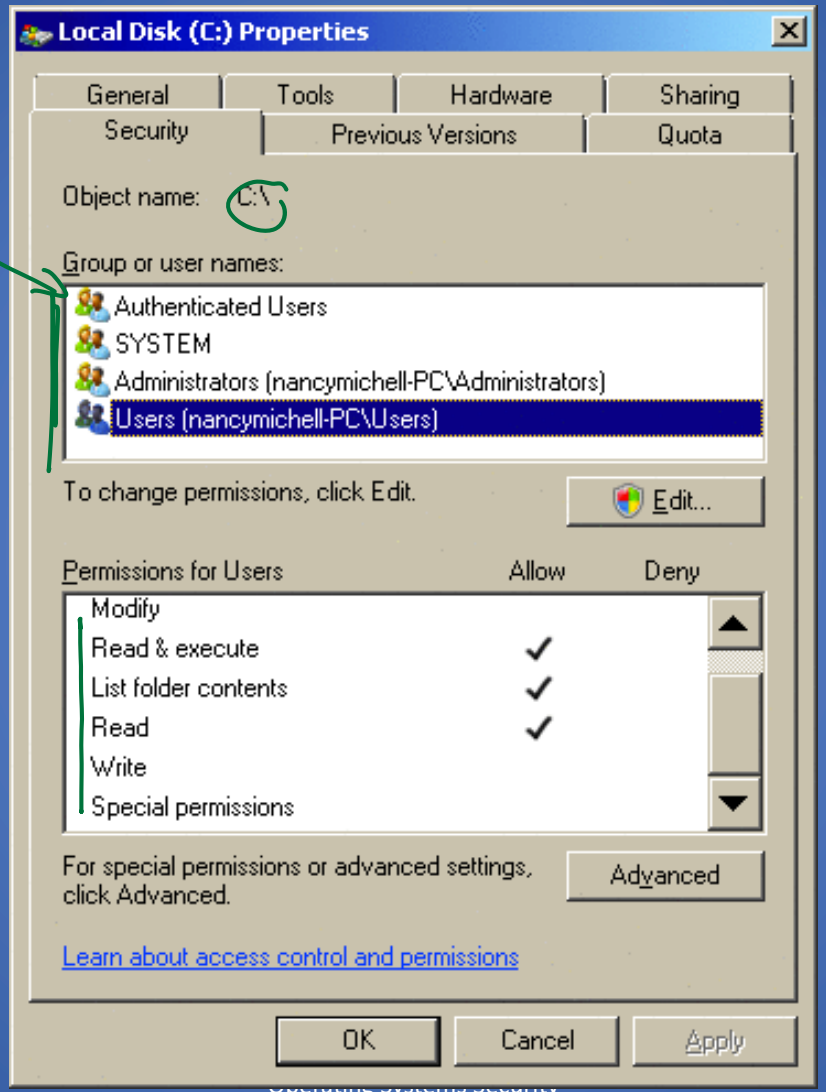
- Permissions are changed with chmod or through a GUI like Konqueror
- Only the file owner or root can change permissions
- If a user owns a file, the user can use chgrp to set its group to any group of which the user is a member
- root can change file ownership with chown (and can optionally change group in the same command)
- chown, chmod, and chgrp can take the -R option to recur through subdirectories

Examples of Changing Permissions

chown -R root dir1	Changes ownership of dir1 and everything within it to root
chmod g+w,o-rwx file1 file2	Adds group write permission to file1 and file2, denying all access to others
chmod -R g=rwX dir1	Adds group read/write permission to dir1 and everything within it, and group execute permission on files or directories where someone has execute permission
chgrp testgrp file1	Sets file1's group to testgrp, if the user is a member of that group
chmod u+s file1	Sets the setuid bit on file1. (Doesn't change execute bit.)

Limitations of Unix Permissions

- Unix permissions are not perfect
  - Groups are restrictive
  - Limitations on file creation
- Linux optionally uses POSIX ACLs
  - Builds on top of traditional Unix permissions
  - Several users and groups can be named in ACLs, each with different permissions
  - Allows for finer-grained access control
- Each ACL is of the form *type*:[*name*]:*rwx*
  - Setuid, setgid, and sticky bits are outside the ACL system


Local Disk (C:) Properties
General
Tools
Hardware
Sharing
Security
Previous Versions
Quota
Object name:
C:\
Group or user names:
Authenticated Users
SYSTEM
Administrators (nancymichell-PC\Administrators)
Users (nancymichell-PC\Users)
To change permissions, click Edit.
Edit...
Permissions for Users
Allow
Deny
Modify
Read & execute
List folder contents
Read
Write
Special permissions
For special permissions or advanced settings, click Advanced.
Advanced
Learn about access control and permissions
OK
Cancel
Apply

What We Have Learned

- What is an operating system
- Processes, users, services
- Access control models (DAC and RBAC)
- Setuid programs
- Dangers of symlinks, setuid, and shared directories
  - A demo if you are “Gone for Ten Seconds”