15-410

"...1969 > 1999?..."

Protection
Nov. 27, 2017

Dave Eckhardt

Dave O'Hallaron

Synchronization

Upcoming lectures

- Security, security
- Parallelism, parallelism
- …
- ...
- Transactions
- Device drivers
- Exam review

Attendance is probably in your best interest

Outline

Protection (OSC: Chapter 14)

- Protection vs. Security
- Domains (Unix, Multics)
- Access Matrix
  - Concept, Implementation
- Revocation – not really covered today (see text)

Mentioning EROS

[Later lectures: techniques and cracks]

Protection vs. Security

Textbook's distinction

- Protection happens inside a computer
  - Which parts may access which other parts (how)?
- Security considers *external threats*
  - Is the system's model intact or compromised?

Protection

Goals

- Prevent intentional attacks
- "Prove" *access policies* are always obeyed
- Detect bugs
  - "Wild pointer" example

Policy specifications

- System administrators
- Users - May want to add new privileges to system

Objects

Hardware

- Exclusive-use: printer, serial port, CD writer, ...
- Fluid aggregates: CPU, memory, disks, screen

Logical **objects**

- Files
- Processes
- TCP port 25
- Database tables

Operations

Depend on object!

- Disk: read_sector(), write_sector()
- CD-ROM: read_sector(...)
- TCP port: advertise(...)
- CPU
  - Conceptually: context_switch(...), <interrupt>
  - More sensibly: realtime_schedule(..., ...)

Access Control

Basic access control

- Your processes should access only “your stuff”
- Implemented by many systems

Access Control

Basic access control

- Your processes should access only “your stuff”
- Implemented by many systems

Principle of least privilege

- (text: “need-to-know”)
- `cc -c foo.c`
  - should read foo.c, stdio.h, ...
  - should write foo.o

Access Control

Basic access control

- Your processes should access only "your stuff"
- Implemented by many systems

Principle of least privilege

- (text: "need-to-know")
- `cc -c foo.c`
  - should read foo.c, stdio.h, ...
  - should write foo.o
  - *should not write ~/.cshrc*
- This is harder

Who Can Do What?

access right = (object, operations)

- /etc/passwd, r
- /etc/passwd, r/w

process → protection domain

- P0 → de0u, P1 → mowry, ...

protection domain → list of access rights

- de0u → (/etc/passwd, r), (/afs/andrew/usr/de0u/.cshrc, w)

Protection Domain Example

Domain 1

- /dev/null, read/write
- /usr/de0u/.cshrc, read/write
- /usr/mowry/.cshrc, read

Domain 2

- /dev/null, read/write
- /usr/mowry/.cshrc, read/write
- /usr/de0u/.cshrc, read

Using Protection Domains

Least privilege requires *domain changes*

- **Doing different jobs requires different privileges**
- **One printer daemon, N users**
  - **“Print each user's file with minimum necessary privileges...”**

Using Protection Domains

Least privilege requires *domain changes*

- Doing different jobs requires different privileges
- One printer daemon, N users
  - "Print each user's file with minimum necessary privileges..."

Two general approaches

- Hold constant the "process → domain" mapping
  - Requires domains to add and drop privileges
  - User "printer" gets & releases permission to read your file
- Hold constant the privileges of a domain
  - Processes *domain-switch* between high-privilege, low-privilege domains
  - Printer *process* opens file as you, opens printer as "printer"

Protection Domain Models

Three sample models

- Domain = user
- Domain = process
- Domain = procedure
- (other models are possible)

Domain = User

Object permissions depend on *who you are*

All processes you are running share privileges

Privilege adjustment?

- Log off, log on (i.e., domain switch)

Domain = Process

Resources managed by special processes

- Printer daemon, file server process, ...

Privilege adjustment?

- Objects cross domain boundaries via IPC
- “Please send these bytes to the printer”

```
/* concept only; pieces missing */
s = socket(AF_UNIX, SOCK_STREAM, 0);
connect(s, pserver, sizeof pserver);
mh->cmsg_type = SCM_RIGHTS;
mh->cmsg_len[0] = open(“/my/file”, 0, 0);
sendmsg(s, &mh, 0);
```

Domain = Procedure

Processor limits access at fine grain

- *Hardware protection on a per-variable* basis!

Domain switch – *Inter-domain procedure call*

- nr = print(strlen(buf), buf);
- What is the “correct domain” for print()?
  - Access to OS's data structures
  - Permission to call OS's internal `putbytes()`
  - Permission to read user's `buf`

Domain = Procedure

Processor limits access at fine grain

- *Hardware protection on a per-variable* basis!

Domain switch – *Inter-domain procedure call*

- nr = print(strlen(buf), buf);
- What is the “correct domain” for print()?
  - Access to OS's data structures
  - Permission to call OS's internal `putbytes()`
  - Permission to read user's `buf`
- Ideally, correct domain automatically created by hardware
  - Common case: “user mode” vs. “kernel mode”
    - » Only a rough approximation of the right domain
    - » But simple for hardware to implement

Unix “setuid” concept

Assume Unix protection domain ≡ numeric user id

- Not the whole story! This overlooks:
  - Group id, group vector
  - Process group, controlling terminal
  - Superuser
- But let's pretend for today

Domain switch via *setuid executable*

- Special permission bit set with `chmod u+s file`
  - Meaning: exec() sets uid to executable file's owner
- Gatekeeper programs
  - “`lpr`” run by anybody can access `printer`'s queue files

Access Matrix Concept

Concept

- Formalization of “who can do what”

Basic idea

- Store all permissions in a matrix
  - One dimension is protection domains
  - Other dimension is objects
  - Entries are access rights

Access Matrix Concept

	File1	File2	File3	Printer
D1		rwxd	r	
D2	r		rwxd	w
D3	rwxd	rwxd	rwxd	w
D4	r	r	r	

Access Matrix Details

OS must still define process → domain mapping

OS must define, enforce domain-switching rules

- Ad-hoc approach
  - Special domain-switch rules (e.g., log off/on)
- Can encode domain-switch in access matrix!
  - Switching domains is a privilege like any other...
  - Add domain *columns* (domains are objects)
  - Add switch-to rights to domain objects
    - » "D2 processes can switch to D1 at will"
  - Subtle (dangerous)

Adding "Switch-Domain" Rights

	File1	File2	File3	D1
D1		rwxd	r	
D2	r		rwxd	s
D3	rwxd	rwxd	rwxd	
D4	r	r	r	

Updating the Matrix

Ad-hoc approach

- “System administrator” can update matrix

Matrix approach

- Add *copy rights* to objects
  - “Domain D1 may copy read rights for File2”
  - So D1 can give D2 the right to read File2

Adding Copy Rights

	File1	File2	File3
D1		rwxdR	r
D2	r		rwxd
D3	rwxd	rwxd	rwxd
D4	r	r	r

Adding Copy Rights

	File1	File2	File3
D1		rwxdR	r
D2	r	r	rwxd
D3	rwxd	rwxd	rwxd
D4	r	r	r

Updating the Matrix

Add *owner rights* to objects

- D1 has owner rights for O47
- D1 can modify the O47 column at will
  - Can add, delete rights to O47 from all other domains

Add *control rights* to domain objects

- D1 has control rights for D2
- D1 can modify D2's rights to any object
  - D1 may be teacher, parent, ...

Access Matrix Implementation

Implement matrix via matrix?

- Huge, messy, slow

Very **clumsy for...**

- "world readable file"
  - Need one entry per domain
  - Must fill rights in when creating new domain
- "private file"
  - Lots of blank squares
    - » Can Alice read the file? - No
    - » Can Bob read the file? - No
    - » ...

Two typical approaches – "ACL", "capabilities"

Access Control List

File1	
D1	
D2	r
D3	rwxd
D4	r

Access Control List (ACL)

List per matrix column (object)

- de0u, read; mowry, read+write

Naively, domain = user

AFS ACLs

- domain = user, user:group, system:anyuser, machine list (system:campushost)
- positive rights, negative rights
  - de0u:staff rlid
  - mdehesaa -rlid

Cool!

Access Control List (ACL)

List per matrix column (object)

- de0u, read; mowry, read+write

Naively, domain = user

AFS ACLs

- domain = user, user:group, system:anyuser, machine list (system:campushost)
- positive rights, negative rights
  - de0u:staff rlid
  - mdehesaa   -rlid

Doesn't really do *least privilege*

- Adding and deleting users is a heavy-weight operation
- System stores *many* privileges per user, permanently...

Capability List

	File1	File2	File3
D1		rwxdR	r

Capability Lists

Capability Lists

- List per matrix row (domain)
- Naively, domain = user
  - More typically, domain = process

Permit *least privilege*

- Domains can transfer & forget capabilities
  - Possible to create “just right” domains
    - » cc which can't write to .cshrc
- Bootstrapping problem
  - Who gets which rights at boot?
  - Who gets which rights at login?
  - Typical solution: store capability lists in files somehow

Mixed Approach

Permanently store ACL for each file

- Must fetch ACL from disk to access file
- ACL fetch & evaluation may be long, complicated

open() checks ACL, creates capability

- “Process 33 has read-only access to vnode #5894”
- Records access rights for this process
- Quick verification on each read(), write()
- Result: per-process fd table “caches” results of ACL checks

Internal Protection?

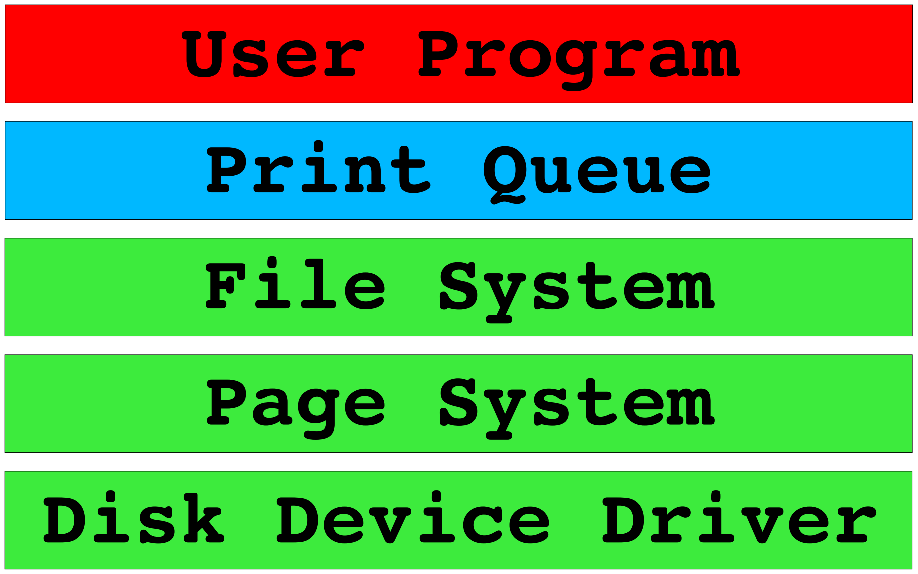
Understood so far:

- Which user process should be allowed to access what?
  - Job performed by OS
- How to protect OS code, data from user processes
  - Hardware user/kernel boundary

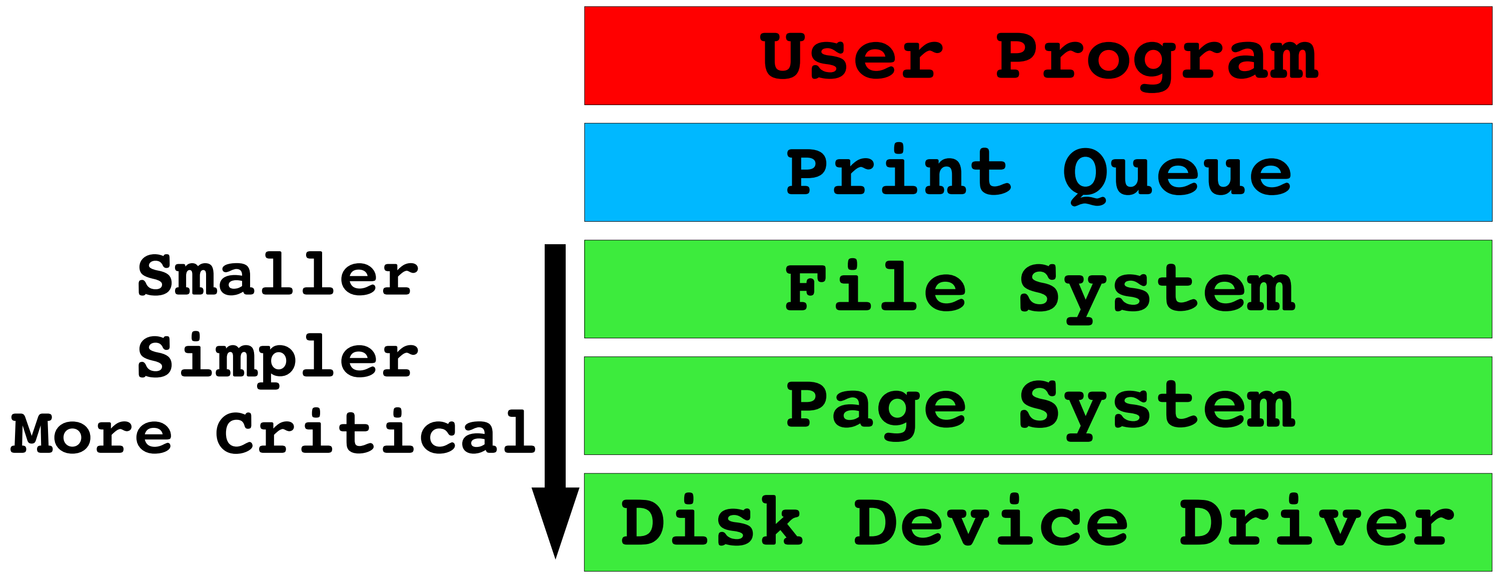
Can we do better?

- Can we protect *parts* of the OS from other parts?

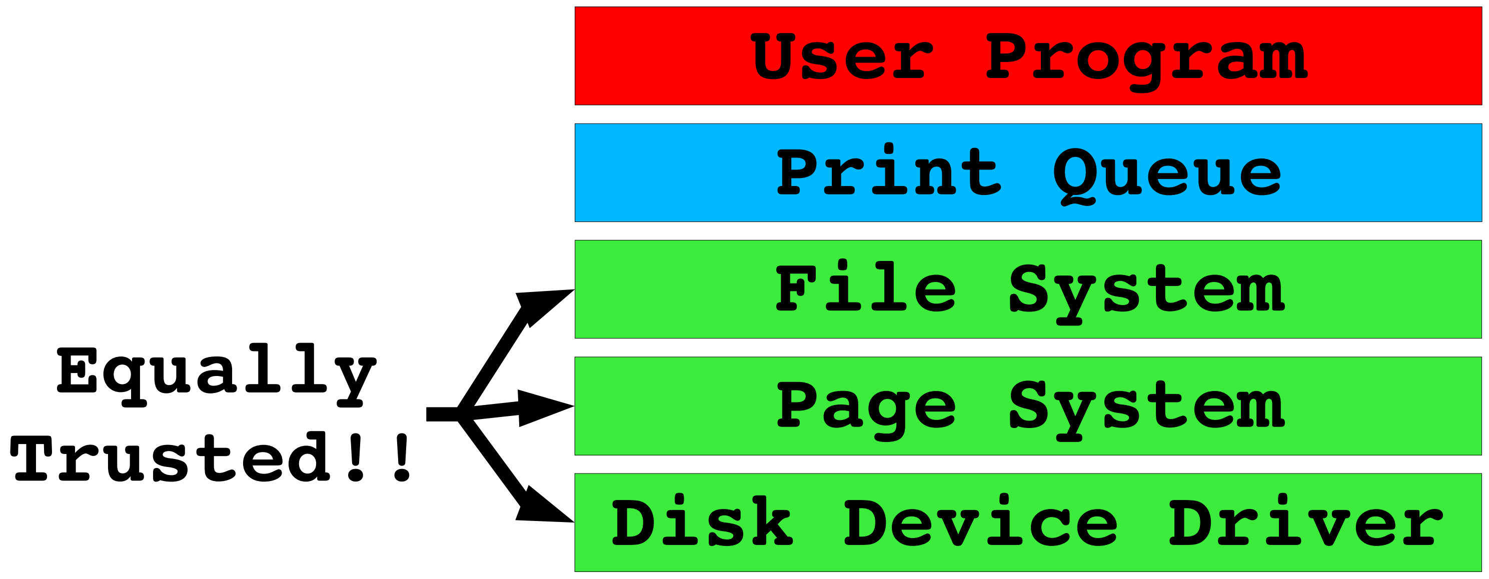
Traditional OS Layers



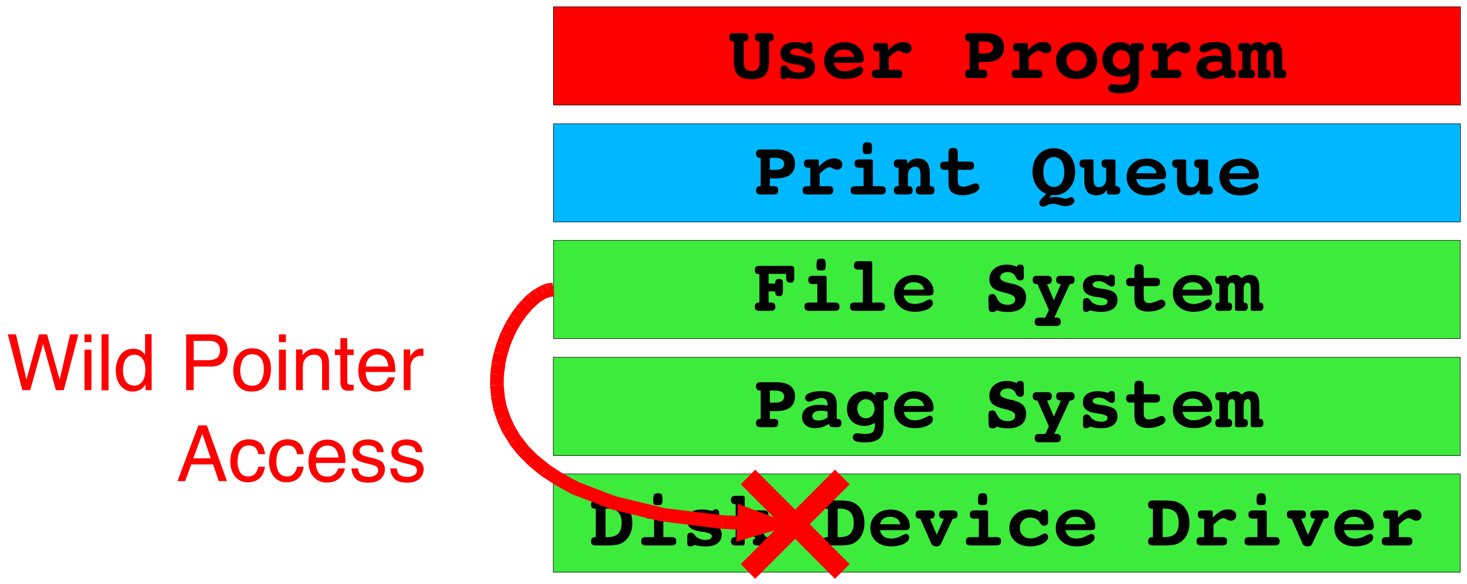
Traditional OS Layers



Traditional OS Layers



Traditional OS Layers



Multics

Multics =

- Multiplexed Information and Computing Service
- Plan: “information utility”
  - Mainframe per city

Designed to scale

- Many users, many programmers
- Protection seen as a key ingredient of reliability

Multics Approach

Trust *hierarchy*

Small "simple" very-trusted *kernel*

- Main job: access control
- Goal: "prove" it correct

Privilege layers (nested "rings")

- Ring 0 = kernel, "inside" every other ring
- Ring 1 = operating system core
- Ring 2 = operating system services
- ...
- Ring 7 = user programs

Multics Ring Architecture

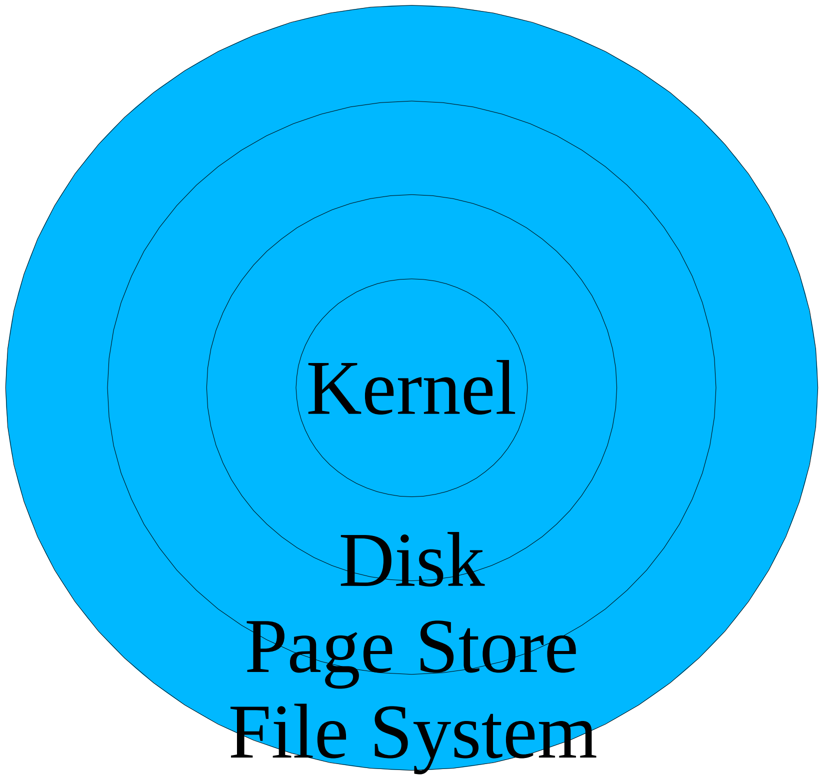
Segmented virtual address space

- “Print module” may contain
  - Entry points in a code segment
    - » `list_printers(), list_queue(), enqueue(), …`
  - Data segment
    - » List of printers, accounting data, queues
- Segment ≡ file (segments persist across reboots)
- VM permissions focus on segments, not pages

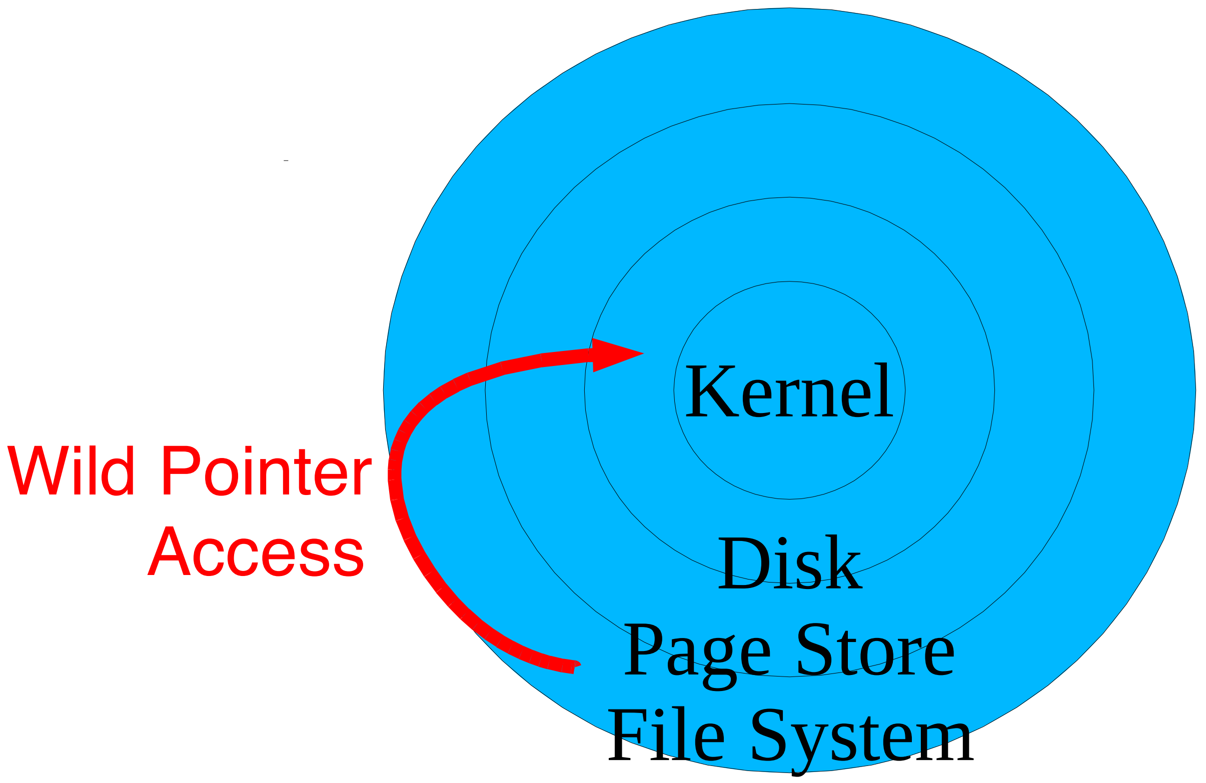
Access checked by hardware

- Which procedures can you call?
- Is access to that segment's data legal?

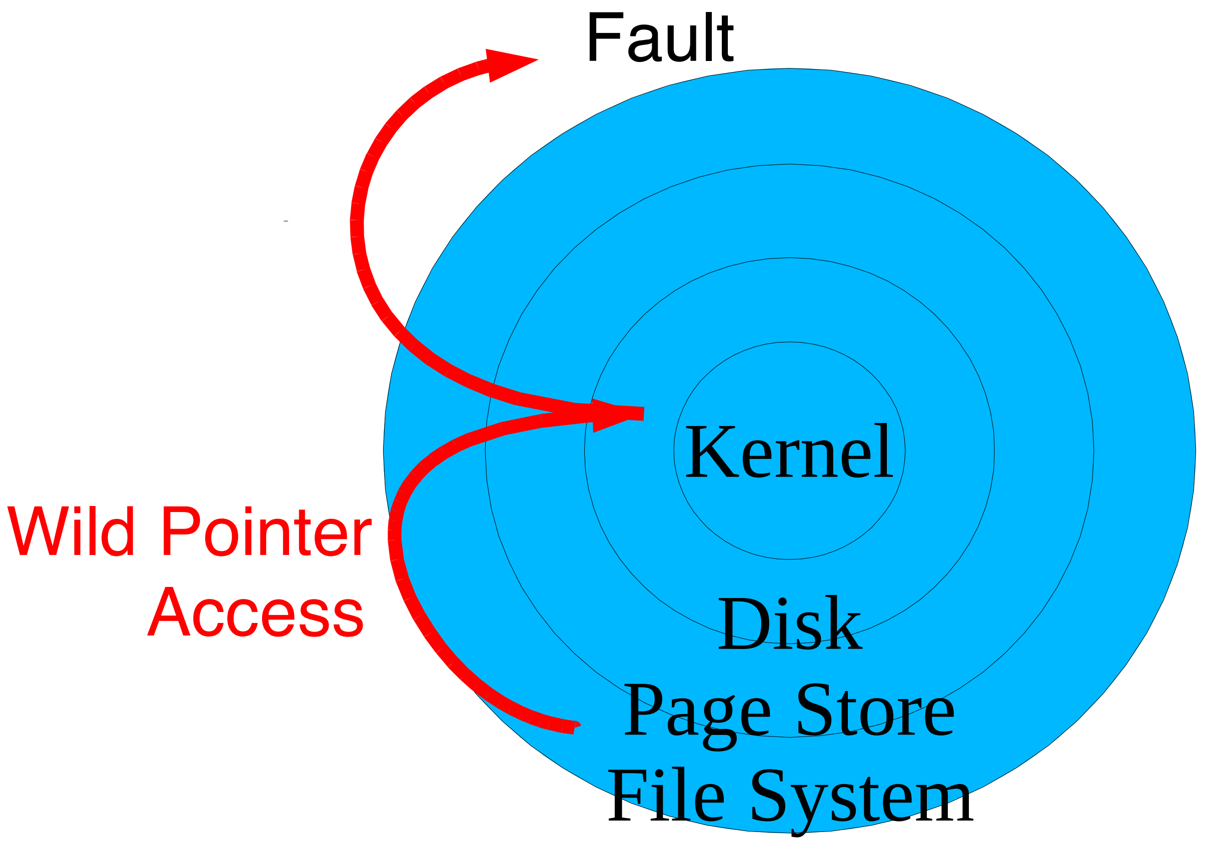
Multics Rings



Multics Rings



Multics Rings



Multics Domain Switching

CPU has *current ring number* register

- Current privilege level, [0..7]

Segment descriptors include

- “Traditional stuff”
  - Segment's limit (size)
  - Segment's base in physical memory
  - Access bits (read, write, execute)
- Ring number
- Access bracket [min, max]
  - Segment “appears in” ring min...ring max
- Entry limit - “you must be this tall to access this segment”
- List of gates (procedure entry points)

Multics Domain Switching

Every procedure call **is a potential domain switch**

Calling a procedure at current privilege level?

- Just call it

Calling a more-privileged procedure?

- Call mechanism checks entry point is legal
- We enter more-privileged mode
- Called procedure can read & write all of our data

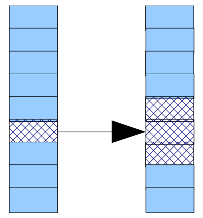
Calling a less-privileged procedure?

- We want to show it *some* of our data (procedure params)
- We don't want it to *modify* our data

Multics Domain Switching

min <= current-ring <= max

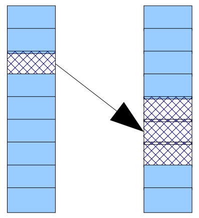
- We are executing in ring 3
- Procedure is “part of” rings 2..4
- Standard procedure call



Multics Domain Switching

current-ring > max

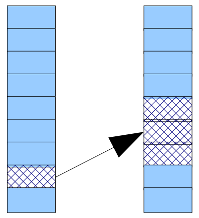
- Calling a more-privileged procedure
- It can do whatever it wants to us



Implementation

- Hardware traps to ring 0 permission-management kernel
- Ring 0 checks current-ring < entry-limit
  - User code may be forbidden to call ring 1 directly
- Ring 0 checks call address is a legal entry point
  - Less-privileged code can't jump into middle of a procedure
- Ring 0 sets current-ring to segment-ring
  - Privilege elevation – after consulting callee's rules
- Runs procedure call

Multics Domain Switching



current-ring < min

- Calling a less-privileged procedure

Implementation

- Trap to ring 0 permission-management kernel
- Ring 0 copies “privileged” procedure call parameters
  - Must be in low-privilege segment for callee to access
- Ring 0 sets current-ring to segment-ring
  - Privilege lowering – callee gets r/o access to carefully chosen privileged state
- Runs procedure call

Multics Ring Architecture

Does this look familiar?

- It should really remind you of something...

Benefits

- Core security policy small, centralized
- Damage limited vs. Unix “superuser”' model

Concerns

- *Hierarchy ≠ least privilege*
- Requires specific hardware
- Performance (maybe)

More About Multics

Back to the future

- Symmetric multiprocessing
- Hierarchical file system (access control lists)
- Memory-mapped files
- Hot-pluggable CPUs, memory, disks
- 1969!!!

Significant influence on Unix

- Ken Thompson was a Multics contributor

The One True OS

- In use 1968-2000
- www.multicians.org

Mentioning EROS

Text mentions Hydra, CAP

- Late 70's, early 80's
- Dead

EROS (“Extremely Reliable Operating System”)

- UPenn, Johns Hopkins
- Based on commercial GNOSIS/KeyKOS OS
- www.eros-os.org
- “Arguably less dead” (see below)

EROS Overview

“Pure capability” system

- “ACLs considered harmful”

“Pure principle system”

- Don't compromise principle for performance

Aggressive performance goal

- Domain switch ~100X procedure call

Unusual approach to capability-bootstrap problem

- *Persistent processes!*

Persistent Processes??

No such thing as reboot

Processes last "forever" (until exit)

OS kernel checkpoints system state to disk

- **Memory & registers defined as *cache of disk state***

Restart restores system state into hardware

"Login" *reconnects* you to your processes

EROS Objects

Disk pages

- capabilities: read/write, read-only

Capability nodes

- Arrays of capabilities

Numbers

- Protected capability ranges
  - “Disk pages 0...16384”

Process – executable node

EROS Revocation Stance

Really **revoking access is hard**

- The user could have copied the file

Don't give out real capabilities

- Give out proxy capabilities
- Then revoke however you wish

Verdict

- Not really satisfying
- Unclear there is a better answer
  - Palladium/“trusted computing” isn't clearly better

EROS Quick Start

http://www.eros-os.org/

- essays/
  - reliability/paper.html
  - capintro.html
  - wherefrom.html
  - ACLSvCaps.html

Current status

- EROS code base transitioned to CapROS.org
- Follow-on research project at Coyotos.org

Concept Summary

Object

- Operations

Domain

- Switching

Capabilities

- Revoking is hard, see text

"Protection" vs. "security"

- Protection is what our sysadmin *hopes* is happening...

Further reading?

- PLASH - "principle of least authority" shell for Linux