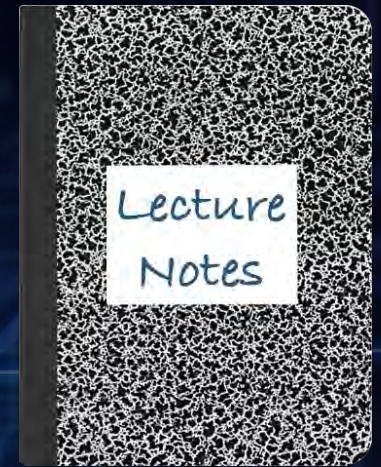


CS 419: Computer Security

Week 6: Part 1
Access Control

Paul Krzyzanowski



© 2024 Paul Krzyzanowski. No part of this content may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.

Protection is essential to security

- **Protection**
 - The **mechanism** that provides controlled access of resources to processes
 - A protection mechanism **enforces** security policies
- **Protection includes:**
 - User privileges: access rights to files, devices, and other system resources
 - Resource scheduling & allocation
 - Process scheduling & memory allocation – Which processes get priority?
 - Quotas (sometimes) – set limits on disk space, CPU, network, memory, ...
- **And relies on**
 - Mechanisms for user accounts & user authentication – identify who we're dealing with
 - Policies – defining who should be allowed do what
 - Auditing: generate audit logs for certain events

Co-located resources

- **Earliest computers – 1945+**
 - Single-user batch processing – no shared resources
 - No need for access control – access control was physical
- **Then ... batch processing ... but no shared storage – 1950s**
 - Per-process allocation of tape drives, printers, punched card machines, ...
- **Later ... shared storage & timesharing systems – 1960s-now**
 - Multiple users share the same computer
 - User accounts & access control important
- **Even later ... PCs – 1974 to now**
 - Back to single-user systems (mostly)
 - ... but software & media became less trusted by the 1990s
- **Now: networked PCs + mobile devices + IoT devices + ...**
 - Shared access: cloud computing, file servers, university systems
 - Need to enforce **access control**

Access control

- Ensure that authorized users can do what they are permitted to do ... *and no more*
- Real world
 - Keys, badges, guards, policies
- Computer world
 - Hardware
 - Operating systems
 - Web servers, databases & other multi-access software
 - Policies



Goals

- **The OS gives us access to resources on a computer:**
 - CPU
 - Memory
 - Files & devices
 - Network
- **We need to:**
 - Protect the operating system from applications
 - Protect applications from each other
 - Allow the OS to stay in control
 - Restrict what users can do

**The OS and hardware are the
fundamental parts of the Trusted Computing Base (TCB)**

Regaining control: hardware timer

- **OS kernel requests timer interrupts**
- **One of several timer devices:**
 - Programmable Interval Timer (PIT)
 - High Precision Event Timer (HPET)
 - or Advanced Programmable Interrupt Controller (APIC timer, one per CPU)
- **Most current Intel Linux systems use APIC**
- **Applications cannot disable this**

Ensures that the OS can always regain control

Timer interrupts ensure OS can take control periodically

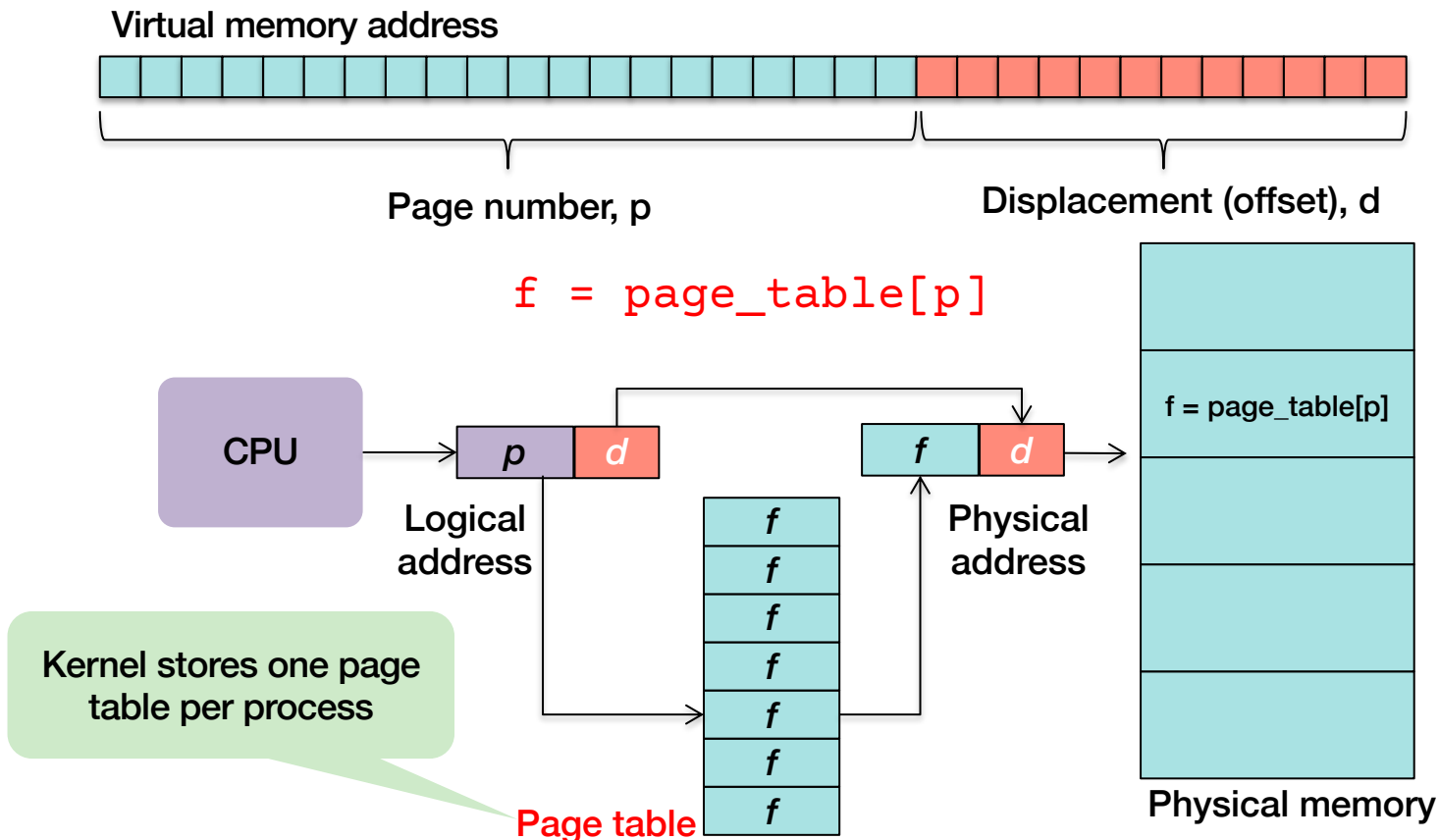
OS Process Scheduler

- Decides whether a process had enough CPU time, and it is time for another process to run
- Prioritizes threads
 - Based on user, user-defined priorities, interactivity, deadlines, “fairness”
 - One process should not adversely affect others
- Avoid **starvation**: ensure all processes will get a chance to run
 - This would be an **availability** attack

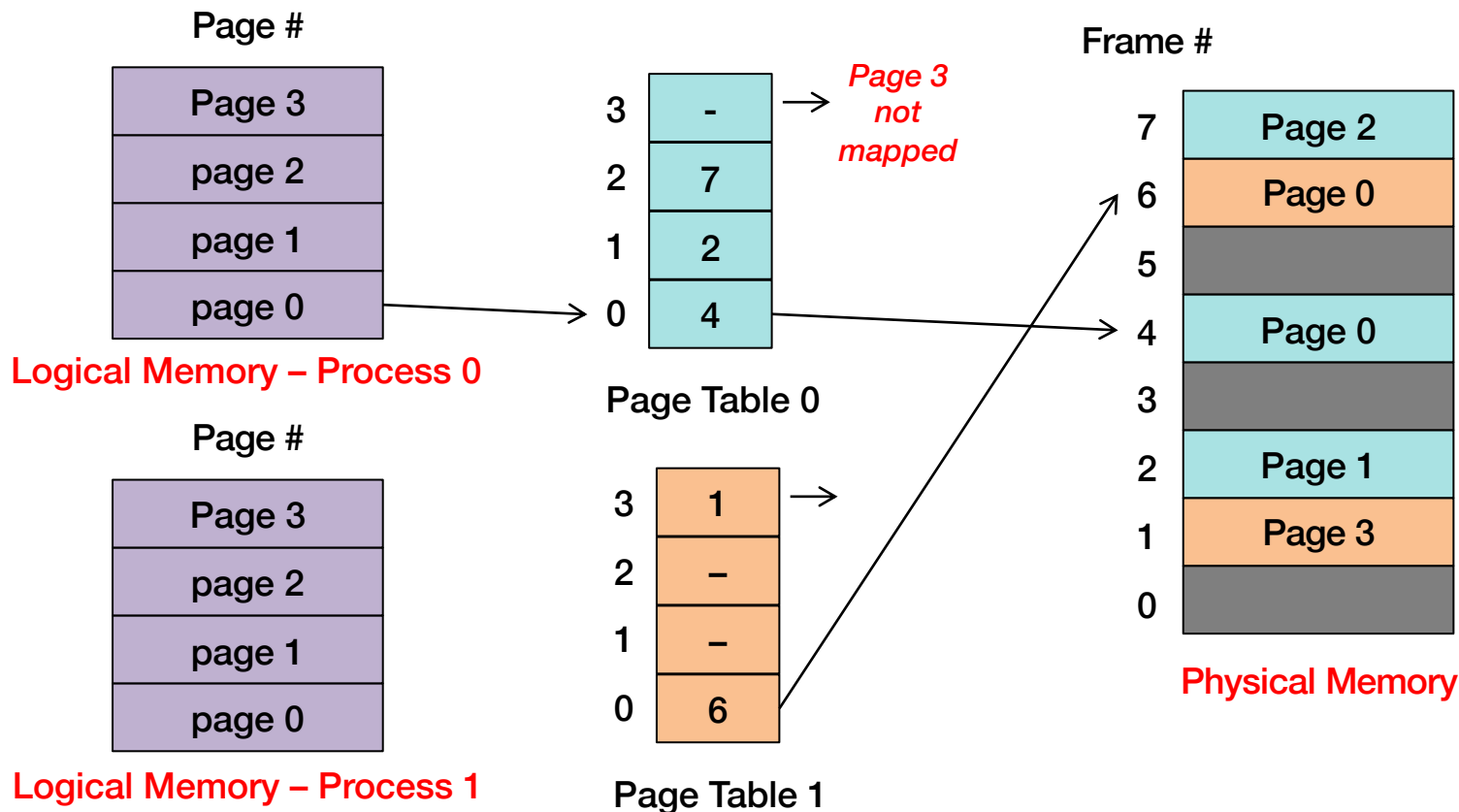
Memory Protection: Memory Management Unit

- All modern CPUs have a **Memory Management Unit (MMU)**
- OS provides each process with **virtual memory**
- Gives each process the illusion that it has the entire address space
- One process cannot see another process' address space
- **Enforce memory access rights**
 - Read-only (code)
 - Read-write (program's data)
 - Execute (code)
 - Unmapped

Page translation



Logical vs. physical views of memory



User & kernel mode

Kernel mode = privileged, system, or supervisor mode

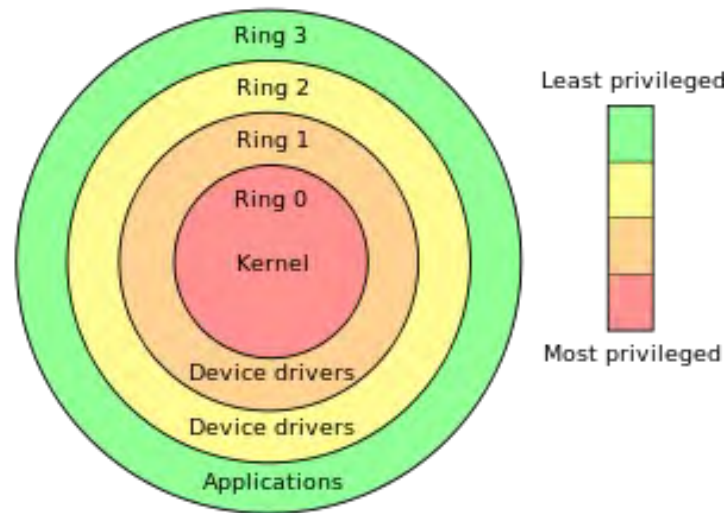
- Access restricted regions of memory
- Modify the memory management unit by changing the page table register
- Set timers
- Define interrupt vectors
- Halt the processor
- Etc.

Getting into kernel mode

- **Trap**: explicit instruction
 - Intel architecture: *INT* instruction (interrupt)
 - ARM architecture: *SWI* instruction (software interrupt)
 - System call instructions
- **Violation** (e.g., access unmapped memory, illegal instruction)
- Hardware **interrupt** (e.g., receipt of network data or timer)

Protection Rings

- All modern operating systems support two modes of operation: **user** & **kernel**
- Multics defined a ring structure with 6 different privilege levels – Intel inherited this
 - Each ring is protected from higher-numbered rings
 - Special call (**call gates**) to cross rings: jump to predefined locations
 - Most of the OS did not run in ring 0
- Intel x86, IA-32 and IA-64 support 4 rings
- Today's OSES only use
 - **Ring 0**: kernel
 - **Ring 3**: user
- Additional protection levels
 - **Ring -1**: Hypervisor (virtual machine monitor)
 - **Ring -2**: System Management Mode (SMM)
 - Low-level, high-priority tasks like power management, thermal monitoring



https://en.wikipedia.org/wiki/Protection_ring

Subjects, Principals, and Objects

Subject: the thing that needs to access resources

Principal: unique identity for a user

- Subjects may have multiple identities and be associated with a set of principals

User: a human (generally)

Object: the resource the subject may access

- Typically, files and devices – they do not perform operations

Subjects access objects: they perform actions on objects

Access control

- Define what operations subjects can perform on objects

Most operating systems control who can do what to each object
(permissions are associated with each object)

User authentication

Must be done before we can do access control

- **Establish user identity – determine the *subject***
 - Operating system privileges are granted based on user identity

Steps

1. **Get user credentials** (e.g., name, password)
2. **Authenticate user** by validating the credentials
 - Get user ID(s), group ID(s)
3. **Control access**: grant access to resources based on user/group IDs & policies

Domains of Protection

Domains of protection

- **Subjects (users running processes) interact with objects**
 - Process runs with the authority of the subject (user)
 - Objects include:
 - hardware (CPU, memory, I/O devices)
 - software: files, processes, semaphores, messages, signals
- **A process should be allowed to access only objects that it is authorized to access**
 - A process operates in a **protection domain**
 - It's part of the **context of the process**
 - Protection domain **defines the objects the process may access** and how it may access them

Modeling Protection: Access Control Matrix

Rows: **domains**
(subjects or groups of subjects)

Columns: **objects**

Each entry in the matrix represents an **access right** of a domain on an object

Objects

	F₀	F₁	Printer
D₀	read	read-write	print
D₁	read-write-execute	read	
D₂	read-execute		
D₃		read	print
D₄			print

*Subjects
domains of protection*

An Access Control Matrix is the primary abstraction for protection in computer security

We may need some more controls

- **Domain transfers**
 - Allow a process to run under another domain's permissions
- **Copy rights**
 - Allow a user to grant certain access rights for an object
- **Owner rights**
 - Identify a subject as the owner of an object
 - Can change access rights on that object for any domain
- **Domain control**
 - A process running in one domain can change any access rights for another domain

Access Control Matrix: Domain Transfers

Switching from one domain to another is a configurable policy

Domain transfers

Allow a process to run under another domain's permissions

Why? Log a user in – how would you run the first user's process?

objects

	F ₀	F ₁	Printer	D ₀	D ₁	D ₂	D ₃	D ₄
D ₀	read	read-write	print	-	switch	switch		
D ₁	read-write-execute	read			-			
D ₂	read-execute				switch			
D ₃		read	print					
D ₄			print					

Subjects
domains of protection

A process in D₀ can switch to running in domain D₁

Access Control Matrix: Delegation of Access

Copy rights: allow a user to grant certain rights to others

- Copy a specific access right on an object from one domain to another

objects

	F ₀	F ₁	Printer	D ₀	D ₁	D ₂	D ₃	D ₄
D ₀	read	read-write	print	-	switch	sw		
D ₁	read-write-execute	read*						
D ₂	read-execute				swtich	-		
D ₃		read	print					
D ₄			print					

Subjects
domains of protection

A process executing in D₁ can give a *read* right on F₁ to another domain

Access Control Matrix: Object Owner

Owner: allow new rights to be added or removed

Identify a subject as the owner of an object

Can change access rights on that object for any domain (column)

objects

	F ₀	F ₁	Printer	D ₀	D ₁	D ₂	D ₃	D ₄
<i>Subjects</i> <i>domains of protection</i>	D ₀	read owner	read-write	print	-	switch	swi	
	D ₁	read-write-execute	read*					
	D ₂	read-execute				swtich		
	D ₃		read	print				
	D ₄			print				

A process executing in D₀ owns F₀, so it can give a *read* right on F₀ to domain D₃ and remove the *execute* right from D₁

Access Matrix: Domain Control

- A process running in one domain can change any access rights for another domain
- Change entries in a row (all objects)

objects

	F ₀	F ₁	Printer	D ₀	D ₁	D ₂	D ₃	D ₄
D ₀	read owner	read- write	print	-	switch	switch		
D ₁	read- write- execute	read*			-			control
D ₂	read- execute				switch			
D ₃		read	print					
D ₄			print					

Subjects
domains of protection

A process executing in D₁ can modify any rights in domain D₄

This gets messy!

- **An access control matrix does not address everything we may want**
- **Processes execute with the rights of the user (domain)**
 - But sometimes they need extra privileges
 - Read configuration files
 - Read/write from/to a restricted device
 - Append to a queue
- **We don't want the user to be able to access these objects**
 - Adding domains to the row of objects is not sufficient
 - We may need a 3-D access control matrix: (subjects, objects, processes)
- **This gets messy!**
 - One solution is to give an executable file a **temporary domain transfer**
 - Assumption is this is a trusted application that can access these resources
 - When run, it assumes the privileges of another domain

Implementing an access matrix

A single table to store an access matrix is impractical

- Big size: # domains (users) × # objects (files)
- Objects may come and go frequently
- Lookup needs to be efficient

Implementing an access matrix

Access Control List

- Associate a column of the table with each object

objects

	F ₀	F ₁	F ₂	F ₃	F ₃	Printer
D ₀	read owner	read- write	read- execute	read		print
D ₁	read- write- execute	read	read- execute	read	write	
D ₂	read- execute		read- execute		write	
D ₃		read	read- execute			print
D ₄			read- execute		write	print

ACL for file F₀

Subjects
domains of protection

Implementing an access matrix

Capability List

- Associate a row of the table with each domain

objects

	F ₀	F ₁	F ₂	F ₃	F ₃	Printer
D ₀	read owner	read- write	read- execute	read		print
D ₁	read- write- execute	read	read- execute	read	write	
D ₂	read- execute		read- execute		write	
D ₃		read	read- execute			print
D ₄			read- execute			

Subjects
domains of protection

Capability list for domain D₁

Capability Lists

Capability list = list of objects together with the operations a specific subject can perform on the objects

- Each item in the list is a **capability**: the operations allowed on a specific object
 - Also known as a **ticket** or **access token**
- A process presents the capability to the OS along with a request
 - Possessing the capability means that access is allowed
- **The capability is a protected object**
 - A process cannot modify its capability list

Capability Lists

- **Advantages**
 - Run-time checking is more efficient
 - Delegating rights is easy
- **Disadvantages**
 - Creating or deleting files means updating all capability lists
 - Changing a file's permissions is hard
 - Hard to find all users that have access to a resource
 - Lists can be huge – the system might have millions of objects
- **Not used in mainstream systems in place of ACLs**
 - Limited implementations: Cambridge CAP, IBM AS/400, Google Fuchsia OS.
- **Capability lists are more commonly used for network services**
 - Used in single sign-on services and other authorization services such as OAuth and Kerberos (sort of)
 - **Access Tokens**
 - Identifies a user's identity and the access rights permitted on the requested service (not objects!)

The End