

Operating Systems

Prepared by Dr. Alaa Eldeen Sayed Ahmed

IT263 / CYB510
Virtualization & Protection

Lecture 10



- **Virtualization**
 - Overview
 - Benefits and Features
 - Types of Virtual Machines and Their Implementations
- **Protection**
 - Goals of Protection
 - Protection Rings
 - Domain of Protection
 - Access Matrix
 - Access Control methods

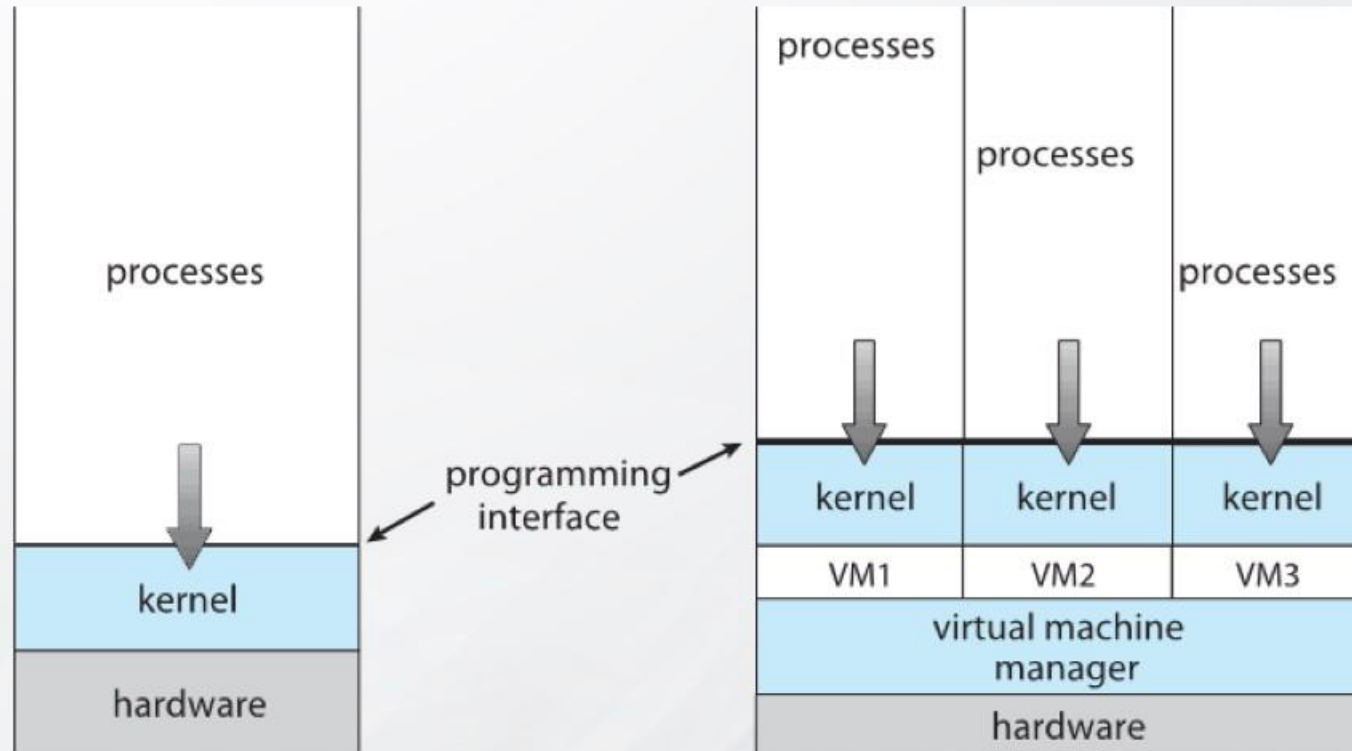
- Operating system **Virtualization** refers to the use of software to allow system hardware to run multiple instances of different operating systems concurrently.
- This will allow users to run different applications requiring different operating systems on one machine.
- These multiple operating systems do not interfere with each other or the different applications. They are isolated from each other.

- **Virtual Machine** is one such method of achieving virtualization.
- Emulates full system hardware in software to create one or more virtual machine instances on a single hardware instance
- This give an illusion that each of these operating system runs on different hardware machines.

- Virtual machine implementation involves several components.
 - **Host** – underlying hardware system
 - **Virtual machine manager (VMM)** or **hypervisor** – creates and runs virtual machines by providing interface that is *identical* to the host
 - (Except in the case of paravirtualization)
 - **Guest** – process provided with virtual copy of the host
 - Usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

Lecture 10

System Model



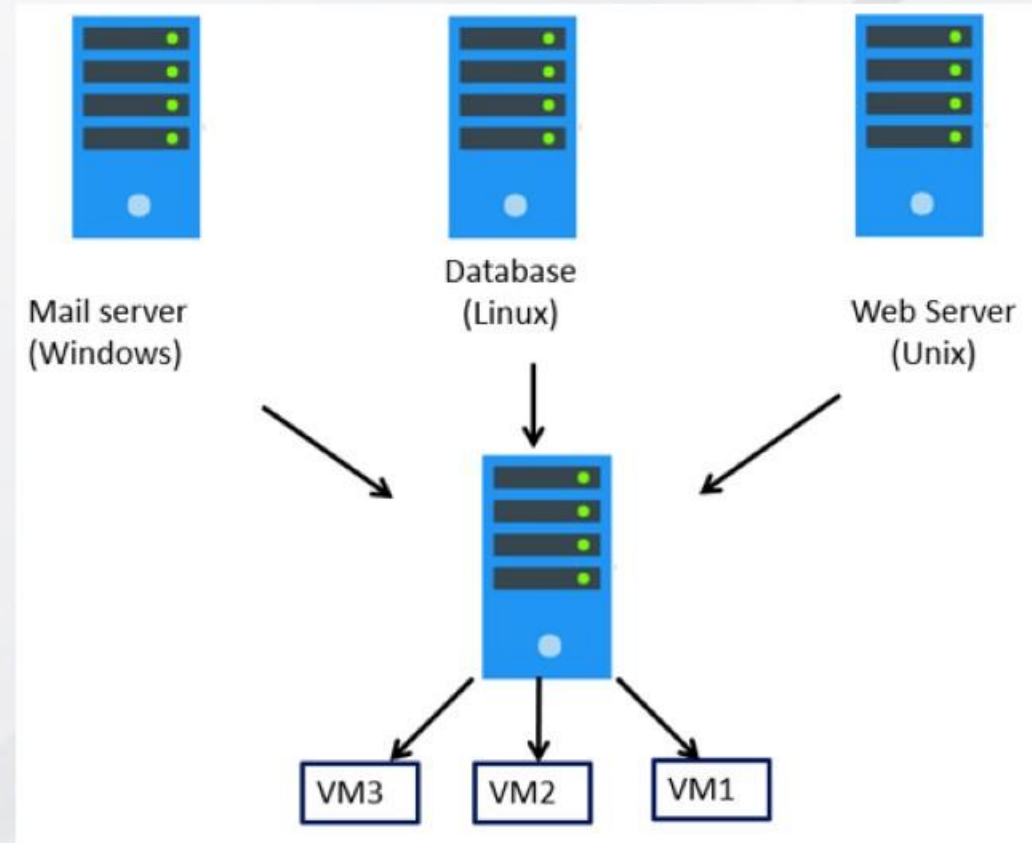
Non-virtual machine

Virtual machine

Lecture 10

System Model

Virtualization is used to improve IT throughput and costs by using physical resources as a pool from which virtual resources can be allocated.



- Vary greatly, with options including:
 - **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
 - IBM LPARs and Oracle LDOMs are examples
 - **Type 1 hypervisors** - Operating-system-like software built to provide virtualization Including VMware ESX, Joyent SmartOS, and Citrix XenServer

- **Type 1 hypervisors** - Also includes general-purpose operating systems that provide standard functions as well as VMM functions
 - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
- **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating
 - Including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

- Other variations include:
 - **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
 - **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
 - Used by Oracle Java and Microsoft.Net

- Host system protected from VMs, VMs protected from each other
 - I.e. A virus less likely to spread
 - Sharing is provided though via shared file system volume, network communication
- Freeze, **suspend**, running VM
 - Then can move or copy somewhere else and **resume**
 - Snapshot of a given state, able to restore back to that state
 - Some VMMs allow multiple snapshots per VM
 - **Clone** by creating copy and running both original and copy
- Great for OS research, better system development efficiency
- Run multiple, different OSes on a single machine
 - **Consolidation**, app dev, ...

- **Templating** – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
- **Live migration** – move a running VM from one host to another!
 - No interruption of user access
- All those features taken together -> **cloud computing**
 - Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops

- Old idea, under many names by HW manufacturers
 - “partitions”, “domains”
 - A HW features are implemented by firmware
 - OS need to do nothing special, VMM is in firmware
 - Smaller feature set than other types
 - Each guest has dedicated HW
- I/O is a challenge as difficult to have enough devices, controllers to dedicate to each guest
- Sometimes VMM implements a **control partition** running daemons (background service) that other guests communicate with for shared I/O
- Can provide virtualization-within-virtualization (guest itself can be a VMM with guests)
 - Other types have difficulty doing this

Lecture 10

Types of VMs – Type 0 Hypervisor

	Guest	Guest	Guest		Guest	Guest
Guest 1	Guest 2			Guest 3	Guest 4	
CPUs memory	CPUs memory			CPUs memory	CPUs memory	
Hypervisor (in firmware)						I/O

- Commonly found in company datacenters
 - In a sense becoming “datacenter operating systems”
 - Datacenter managers control and manage OSes in new, sophisticated ways by controlling the Type 1 hypervisor
 - Consolidation of multiple OSes and apps onto less HW
 - Move guests between systems to balance performance
 - Snapshots and cloning

- Special purpose operating systems that run natively on HW
 - Rather than providing system call interface, create run and manage guest OSes
 - Can run on Type 0 hypervisors but not on other Type 1s
 - Run in kernel mode
 - Guests generally don't know they are running in a VM
 - Also provide other traditional OS services like CPU and memory management

- Another variation is a general purpose OS that also provides VMM functionality
 - RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris
 - Perform normal duties as well as VMM duties
 - Typically, less feature rich than dedicated Type 1 hypervisors
- In many ways, treat guests OSes as just another process
 - Albeit with special handling when guest tries to execute special instructions

- Less interesting from an OS perspective
 - Very little OS involvement in virtualization
 - VMM is simply another process, run and managed by host
 - Even the host doesn't know they are a VMM running guests
 - Tend to have poorer overall performance because can't take advantage of some HW features
 - But also a benefit because require no changes to host OS
 - Student could have Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

- Does not fit the definition of virtualization – VMM not presenting an exact duplication of underlying hardware
 - But still useful!
 - VMM provides services that guest must be modified to use
 - Leads to increased performance
 - Less needed as hardware support for VMs grows

- Xen, leader in Para-virtualized space, adds several techniques
 - For example,
 - clean and simple device abstractions
 - Good communication between guest and VMM about device I/O
 - Each device has circular buffer shared by guest and VMM via shared memory

- Memory management does not include nested page tables
 - Each guest has own read-only tables
 - Guest uses **hypercall** (call to hypervisor) when page-table changes needed **(Similar to system calls which provide to users to communicate with kernel in traditional operating systems)**
- Guest OS had to be modified to use run on para-virtualized VMM

- This is not-really-virtualization but using same techniques, providing similar features
- Programming language is designed to run within custom-built virtualized environment
 - For example Oracle Java has many features that depend on running in **Java Virtual Machine (JVM)**

- In this case **virtualization** is defined as providing APIs that define a set of features made available to a language and programs written in that language to provide an improved execution environment
- JVM compiled to run on many systems (including some smart phones even)
- Programs written in Java run in the JVM no matter the underlying system
- Similar to **interpreted languages**

Protection in Operating systems

- In any protection model, computer consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so

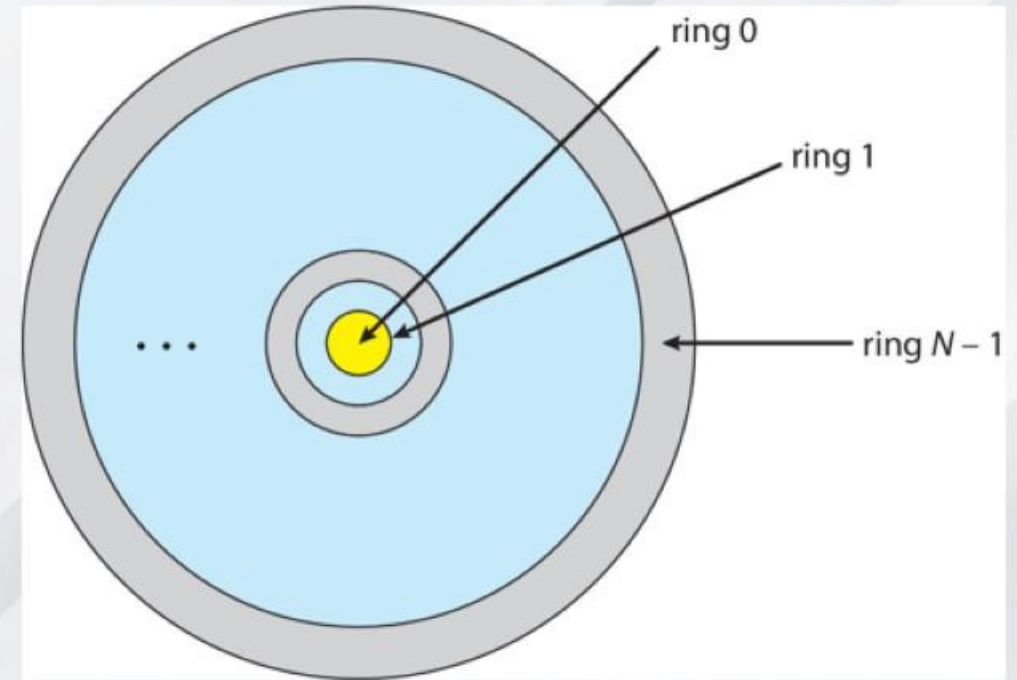
- Guiding principle – An operating system following the **principle of least privilege**
 - Programs, users and systems should be given just enough **privileges** to perform their tasks
 - So, failure or compromise of a component does the minimum damage and allows the minimum damage to be done.

- For Example:
- **In UNIX OS**— Normal user should not run as root.
 - root user can execute privileged commands.
 - Most users respect that to avoid accidental delete operation for which there is no corresponding undelete.
- Now consider that rather than human error, damage may result from malicious attack (Virus ,....etc).
 - **The principle of least privilege** would give the system a chance to mitigate the attack
 - if malicious code cannot obtain root privileges, then there is a chance that adequately defined permissions may block all, or at least some, of the damaging operations.
 - Permissions can act like an immune system at the operating-system level.

- The main component of modern operating systems is the kernel, which manages access to system resources and hardware.
- The kernel, is a trusted and privileged component and therefore must run with a higher level of privileges than user processes.
- This privilege separation requires hardware support
- A popular model of privilege separation is called **protection rings**.

Protection Rings

- In this model, execution is defined as a set of concentric rings, with ring i providing a subset of the functionality of ring j for any $j < i$.
- The innermost ring, ring 0, thus provides the full set of privileges.



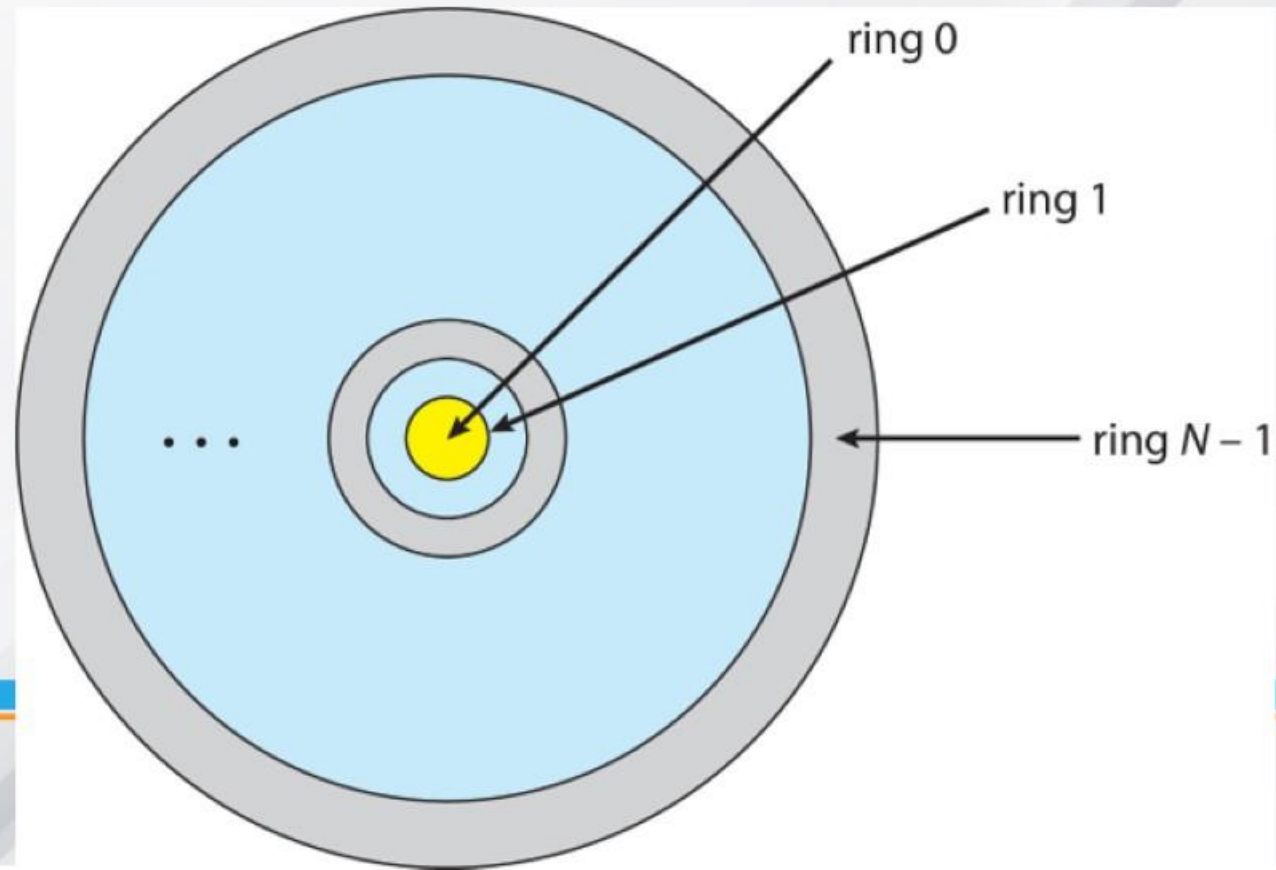
- When the system boots, it boots to the highest privilege level.
- Code at that level performs necessary initialization before dropping to a less privileged level.
- To return to a higher privilege level, code usually calls a special instruction, called gate.
- The syscall instruction (in Intel) is one example (Switch from user to kernel).

Domain of Protection

- Rings of protection separate functions into domains and order them hierarchically
- Computer can be treated as processes and objects
 - **Hardware objects** (such as devices) and **software objects** (such as files, programs, semaphores)
- Process should only have access to objects it currently requires to complete its task.

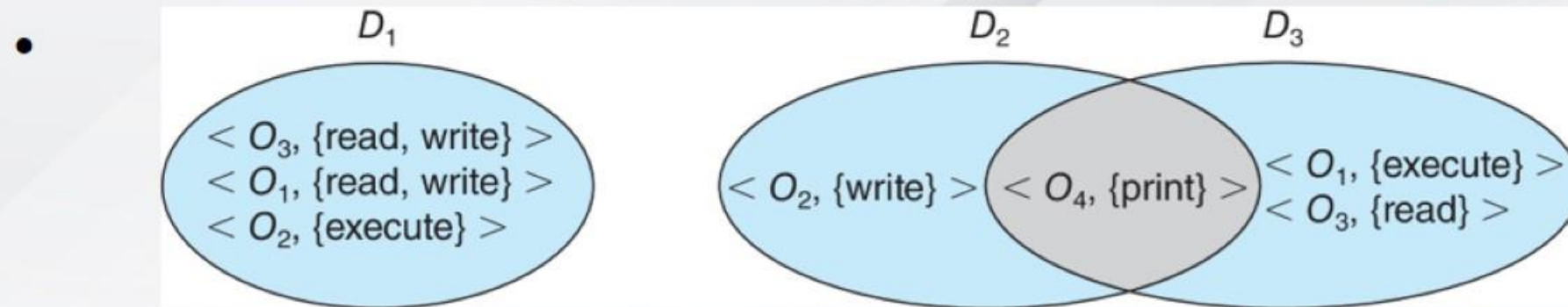
Protection Rings

- Let D_i and D_j be any two domain rings
- If $j < i \Rightarrow D_i \subseteq D_j$



Domain Structure

- Domain = set of access-rights
- Access-right = $\langle \textit{object-name}, \textit{rights-set} \rangle$
- where *rights-set* is a subset of all valid operations that can be performed on the object



- The general model of protection can be viewed abstractly as a matrix to control and manage permissions. This is called an **access matrix**.
- it's a table that shows what actions a domain of users can perform on specific objects within a system.
 - Rows represent domains
 - Columns represent objects
- **Access(i, j)** is the set of operations that a process executing in Domain_i can invoke on Object_j

- There are different types of rights the Objects(e.g files) can have. The most common ones are:
 - 1. Read-** This is a right given to a process in a domain that allows it to read the file.
 - 2. Write-** Process in the domain can be written into the file.
 - 2. Execute-** The process in the domain can execute the file.
 - 3. Print-** Process in the domain only has access to a printer.

Lecture 10

Access Matrix

- In this Example, There are **four domains** and **four objects**– three files(F_1 , F_2 , F_3) and one printer.
 - A process executing in D_1 can read files F_1 and F_3 .
 - A process executing in domain D_4 has same rights as D_1 but it can also write on files.
 - Printer can be accessed by only one process executing in domain D_2 .
 - A process executing in domain D_3 has the right to read file F_2 and execute file F_3 .

domain \ object				
	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

- The access matrix can implement policy decisions concerning protection.
- if a process in Domain D_i tries to do “op” on object O_j , then “op” must be in the access matrix
- User who creates object can define access column for that object
- To switch a process from one domain to another, **(switch operation)** is executing on an object (the domain).

Use of Access Matrix Cont.....

Access Matrix of Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

According to the above matrix, a process executing in domain D_2 can switch to domain D_3 and D_4 . A process executing in domain D_4 can switch to domain D_1 and process executing in domain D_1 can switch to domain D_2 .

- Allowing controlled change in the contents of the access-matrix entries requires three additional operations: copy, owner, and control.
- The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right.
- The copy right allows the access right to be copied only within the column (that is, for the object) for which the right is defined.
- In the following example, process executing in domain D_2 can copy the read operation into any entry associated with file F_2 .

Lecture 10

Access Matrix with *Copy* Rights

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Owner Rights

- The ***owner*** right adds the privilege of adding new rights or removing existing ones:

- If access(i, j) includes the owner right, then a process executing in domain D_i can add and remove any right in any entry in column j

Copy and Owner applicable to an object

object domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Lecture 10

Modified Access Matrix of Figure B

Control rights : A mechanism is also needed to change the entries in a row.

Control applicable to domain object

control right in access
(D2, D4). a process
executing in domain
D2 could modify
domain D4

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Discretionary and Mandatory Access Access Control (DAC Vs MAC)

- **Access control** regulates which users, applications, and devices can add, edit, view and delete resources in an organization's environment to protect sensitive data from misuse, theft, abuse, and other threats.
- Mandatory Access Control (MAC) and Discretionary Access Control (DAC) are two of the popular access control models in use.

Discretionary and Mandatory Access Access Control (DAC Vs MAC)

- The key difference among these two approaches is in how they provide access to users.
 - With **MAC**, admins creates a set of levels where each user is linked to a specific access level where that user can access all the resources that are not greater than his access level.
 - In contrast, each resource in **DAC** has a list of users who can access it. DAC provides access by identity of the user and not by permission level.

Mandatory Access Control (MAC)

- As a brief example of MAC, consider a simple set of Levels, ordered according to level of privilege:
 - “unclassified,” “secret,” and “top secret.”
- A user with “secret” level, will have access to “unclassified” and “secret” files, but not to “top secret” files.
- No user or processes will be aware of the existence of “top secret” files, the operating system filters them out of all file operations
- Same with User processes, “unclassified” process would not be able to see or perform IPC requests to a “secret” (or “top secret”) process.

New Trends in Operating Systems

- Operating systems continue to evolve to meet modern computing needs.
- Recent Trends focus on :
 - Cloud-Native Operating Systems
 - AI-Enhanced
 - Security-Focused OS Innovations
 - Edge & IoT Operating Systems

Cloud-Native Operating Systems

- Designed specifically for cloud and distributed environments.
- Lightweight, modular architectures.
- Examples : container-optimized OS, micro-VMs and Flatcar Linux

- **[1]** A. B. Bitchebe and A. Balmau, “MaLV-OS: Rethinking the Operating System Architecture for Machine Learning in Virtualized Clouds,” *arXiv preprint arXiv:2508.03676*, 2025. [Online]. Available: <https://arxiv.org/abs/2508.03676>
- **[2]** M. Moore and A. Zenla, “Goldilocks Isolation: High-Performance VMs with Edera,” *arXiv preprint arXiv:2501.04580*, 2025. [Online]. Available: <https://arxiv.org/abs/2501.04580>
- **[3]** Google Cloud, “Container-Optimized OS: Features and Benefits,” Google Cloud Documentation, 2025. [Online]. Available: <https://cloud.google.com/container-optimized-os/docs/concepts/features-and-benefits>

AI-Enhanced OS

- OS increasingly integrates machine learning for optimization.
- Intelligent scheduling and resource prediction.
- Automated Anomaly Detection & Self-Healing

- [1] S. Bitchebe and O. Balmau, “MaLV-OS: Rethinking the Operating System Architecture for Machine Learning in Virtualized Clouds,” *arXiv preprint arXiv:2508.03676*, 2025.
- [2] S. Abbas, F. A. Abdi and S. Mehmood, “AI for Operating System Optimization: A Review of Intelligent Scheduling, Resource Allocation, and Security,” *Center for Management Science Research*, vol. 3, no. 4, pp. 1131–1148, 2025.
- [3] V. Shankar, K. B. Srivatsava and X. Li, “Enhancing Operating System Performance with AI: Optimized Scheduling and Resource Management,” *International Journal of Artificial Intelligence & Machine Learning*, vol. 4, no. 1, pp. 172–191, May 2025.

Security-Focused OS Innovations

- Zero-trust OS architecture.
- Sandboxing and hardware-based isolation.
- Stronger kernel-level protections and memory-safe programming languages (for example RUST).

Security-Focused OS Innovations

- W[1] Y. Peng, H. Tian, Z. Junyang et al., “Asterinas: A Linux ABI-Compatible, Rust-Based Framkernel OS with a Small and Sound TCB,” *arXiv preprint arXiv:2506.03876*, 2025. [arXiv](#)
- [2] Y. Guo, Z. Wang, W. Bai, Q. Zeng and K. Lu, “BULKHEAD: Secure, Scalable, and Efficient Kernel Compartmentalization with PKS,” *arXiv preprint arXiv:2409.09606*, 2024. [arXiv](#)
- [3] W. Liang, “The Role of Sandboxing and Isolation Techniques in Secure System Software,” *journal article*, 2025.

Security-Focused OS Innovations

- W[1] Y. Peng, H. Tian, Z. Junyang et al., “Asterinas: A Linux ABI-Compatible, Rust-Based Framkernel OS with a Small and Sound TCB,” *arXiv preprint arXiv:2506.03876*, 2025. [arXiv](#)
- [2] Y. Guo, Z. Wang, W. Bai, Q. Zeng and K. Lu, “BULKHEAD: Secure, Scalable, and Efficient Kernel Compartmentalization with PKS,” *arXiv preprint arXiv:2409.09606*, 2024. [arXiv](#)
- [3] W. Liang, “The Role of Sandboxing and Isolation Techniques in Secure System Software,” *journal article*, 2025.

- Ultra-light OS designed for small devices.
- Low-power consumption and real-time capabilities.
- Support for autonomous decision-making at the edge.

- [1] A. K. Singh, B. Sharma, and R. Ghosh, “TinyOS-Edge: A Lightweight Real-Time OS for IoT and Edge Devices,” *IEEE Internet of Things Journal*, vol. 12, no. 7, pp. 5432–5444, July 2025.
- [2] J. Zhang, L. Wang, P. Li and M. Chen, “Edge-Inspired Embedded OS for Low-Power Real-Time Systems,” *Proceedings of the 2024 IEEE International Conference on Embedded Systems & IoT (ICESI)*, pp. 109–116, Oct. 2024.

Overall View of Modern Trends in OSs

- These four trends illustrate how operating systems have evolved from simple programs that control hardware into systems that are:
 - **More intelligent**
 - **Distributed**
 - **More secure**
 - **More adaptive** to modern environments such as **Cloud, AI, and IoT**
- This makes taking a broader look at these trends an important step after studying the traditional Operating Systems curriculum.



Thank You