INF-2201
04 – Operating System structures

John Markus Bjørndalen
2026

- How do we structure the operating system kernel?
- Monolithic
- Layered
- Microkernel
- Exokernels
- Virtual Machines

Monolithic systems

- Common structure for operating systems
- Everything in one "soup"
  - Everything has (in principle) access to everything else (runs in kernel context)
  - Can be efficient if you reduce the number of system calls / context switches
    - Subsystems often use other subsystems

Skype: "A monolithic operating system visualized as a pot of noodles with meat and vegetables."
Made with Image Creator from Designer. Powered by DALL·E 3.

Monolithic systems

Will usually have some structure
- Subsystems
- Loadable modules (ex: Linux)
- Some division necessary to manage and understand the system

Skype: "A monolithic operating system visualized as a pot of noodles with meat and vegetables."
Made with Image Creator from Designer. Powered by DALL·E 3.

Monolithic systems

One main issue:
- Crashes in one subsystem may bring down the rest or make the system unstable
  - Overwriting memory
  - Leaving locks or other structures in an incorrect state

Skype: "A monolithic operating system visualized as a pot of noodles with meat and vegetables."
Made with Image Creator from Designer. Powered by DALL·E 3.

Layered systems

- Idea: clean layers with identified responsibilities
  - Layers above build on abstractions further down
- Examples:
  - THE (mostly a design principle)
  - Multics (uses rings instead of "flat" layers)
- Question: is it easy to define the right layers in practice to provide modern functionality? What builds on what?

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure 1-25. Structure of the THE operating system.
Modern Operating Systems, Tanenbaum & Bos

Microkernel

Idea:

- Minimalistic kernel
  - As little as possible inside the kernel – less complexity and hopefully fewer bugs
  - Only deals with permissions, protection and a way of communicating
  - Some systems put the mechanism in the kernel and policy in user level processes
- Services implemented in user level processes, like:
  - File systems
  - Device drivers

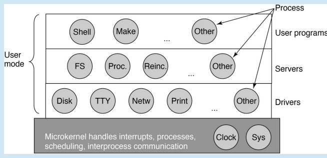


Figure 1-26. Simplified structure of the MINIX system.
Modern Operating Systems, Tanenbaum & Bos

Microkernel

- Communication between servers, drivers and user programs?
  - Message passing (Mach, Minix, ...)
- I/O?
  - Minix: send message to kernel

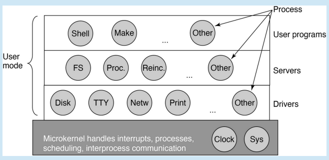


Figure 1-26. Simplified structure of the MINIX system.
Modern Operating Systems, Tanenbaum & Bos

Microkernel

- Examples:
  - L4
    - Formally verified version: seL4
  - QNX
    - Real time system
    - Also used in space
  - Mach
    - Also see XNU (for Apple users)
  - Fuchsia (Google) based on Zircon kernel
  - Minix

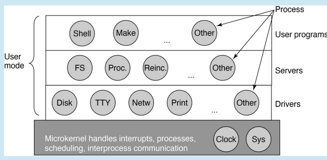


Figure 1-26. Simplified structure of the MINIX system.
Modern Operating Systems, Tanenbaum & Bos

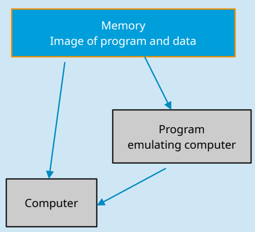
Exokernel

- Idea:
  - Tiny kernel
  - Don't provide abstractions that are not used by user
- Kernel partitions hardware
  - Higher level layers / library operating systems requests resources
  - Kernel assigns rights (if not used by others)

Virtual Machines

Idea

- Your computer is an implementation of an abstract idea (instruction set and architecture)
- We can implement the abstraction
  - As hardware
  - In software
  - As a combination of hardware and software



Virtual Machines

- The software idea 1: Java Virtual Machine (JVM)
  - Example of: interpret instructions and emulate instructions of an architecture.
  - Interpretation can be slow, but improved using binary translation (Just In Time (JIT) or pre-compiled)
  - Could also implement the instruction set in in hardware.

Virtual Machines

- The hardware idea 1: IBM CP/CMS (1968) and IBM VM/370 (1970)
  - Make it an isolated copy of the real machine so that the software running inside thinks it's running on the real computer
    - Traps to OS treated as traps inside the VM!
  - Can run multiple versions of operating systems (or even different operating systems) at the same time
  - Applications or customers can get slices of the computer
  - Requires hardware support
- The idea took a long time before it was supported on commodity PCs

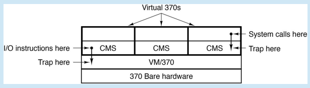


Figure 1-28. The structure of VM/370 with CMS
Modern Operating Systems, Tanenbaum & Bos

Virtual Machines

- The software idea 2: simulate real hardware
  - We used one previously in the course: bochs !
  - Interpret instructions and emulate instructions of an architecture.
  - More advanced uses binary translation (Just In Time (JIT) or pre-compiled)
- Still slower than running on the real hardware
  - But may be faster than old hardware. Ex: https://pistorm.github.io/

Virtual Machines

- The software idea 3: Type 2 hypervisors
  - PCs did not have VM hardware support, but OS kernels could give some support
    - Ex: Virtual Memory and shadowing of data to trap when modifying protected structures
  - Run a VM as a user level process, but use host CPU directly for user level instructions
  - Rewrite priviledged instructions or trap to host OS kernel to support priviledged instructions and modes
- Terms: Guest OS vs Host OS
- Getting closer, but still some performance gap

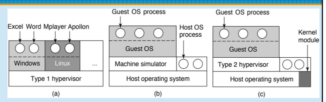


Figure 1-29. (a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor.
Modern Operating Systems, Tanenbaum & Bos

Virtual Machines

- The software/hardware idea: Type 1 hypervisors
  - Add VM hardware support (ex: Intel VT-x, AMD-V)
  - Separate protection mode orthogonal to protection rings
    - Lets host computer expose a full computer inside a VM
  - Coordinate and manage VMs using a Virtual Machine Monitor or Type 1 Hypervisor
- Example: KVM

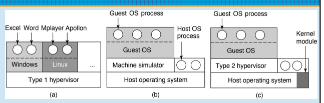


Figure 1-29. (a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor.
Modern Operating Systems, Tanenbaum & Bos

Containers

- Idea:
  - Share kernel and resources with the host
  - Restrict the resources available and visible inside the containers
  - Low overhead (just runs as normal processes on the host)
- Simple example: chroot on Linux
  - Changes the "root" of the file system to a folder inside the host computer
  - The process that is contained inside the chroot environment shares the kernel with the host, but can only see the files inside the chroot environtment.
    - Device files etc may not be visible (=> cannot access them)
    - Runs with the privileges of the user that the process was started with
  - Creates a "container" that limits access to the host system
- More advanced containers (LXC, Docker, ...):
  - Virtualize networks, devices and other resources in the operating system
  - More fine-grained restriction or exposing of resources
  - Tools for managing containers (move, copy, backup, configuration, ...)

Questions

- Why aren't we running everything that touches the internet in containers or VMs?
  - Browsers
  - LLMs ...
  -