Operating Systems

Classification, Architecture, and Layering

Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

October 25, 2022

Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Agenda

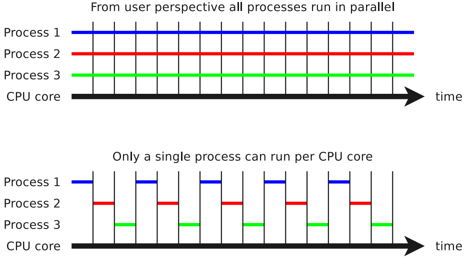
- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Singletasking and Multitasking

- **Singletasking**
  - At any given moment, only a single process is executed
  - Multiple started programs are executed one after the other

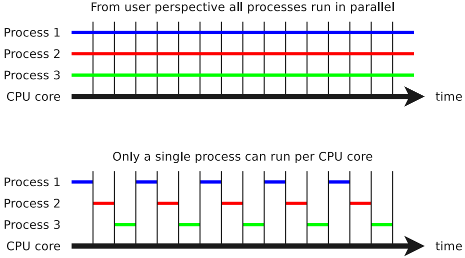
Singletasking and Multitasking

- **Singletasking**
  - At any given moment, only a single process is executed
  - Multiple started programs are executed one after the other
- **Multitasking**
  - Multiple programs can be executed at the same time (with multiple CPUs/Cores) or pseudo parallel



Singletasking and Multitasking

- **Singletasking**
  - At any given moment, only a single process is executed
  - Multiple started programs are executed one after the other
- **Multitasking**
  - Multiple programs can be executed at the same time (with multiple CPUs/Cores) or pseudo parallel



Task, process, job,. . .

The term **task** is equivalent to **process** or from the user's point of view **job**

Why Multitasking?

- Processes often need to wait for external events, for example...
  - user input,
  - input/output (I/O) operations of peripheral devices, or
  - information from another process.

Multitasking avoids blocking

With multitasking processes, waiting for, e.g., incoming E-mails, successful database operations, or data written into memory can yield the processor

The Cost of Multitasking

What is the drawback of this concept?

The Cost of Multitasking

What is the drawback of this concept?

Drawback

Switching from one process to another one causes **overhead**.
→ Dependent on the use case and the type of system this overhead may be negligible or significant

Single-user and Multi-user

- Single-User
  - The computer can only be used by single user at any point in time

Single-user and Multi-user

- Single-User
  - The computer can only be used by single user at any point in time
- Multi-User
  - Multiple users can work simultaneously with the computer
    - Users share the system resources (typically as fair as possible)
    - Users must authenticate themselves (e.g., via credentials)
    - Resources like data or process must be separated and access control is required

Single-user and Multi-user

- Single-User
  - The computer can only be used by single user at any point in time
- Multi-User
  - Multiple users can work simultaneously with the computer
    - Users share the system resources (typically as fair as possible)
    - Users must authenticate themselves (e.g., via credentials)
    - Resources like data or process must be separated and access control is required
- Examples

	Single-User	**Multi-User**
Singletasking	MS-DOS, Palm OS	—
Multitasking	OS/2, Windows 3x/95/98, BeOS, MacOS 8x/9x, AmigaOS, Risc OS	Linux/UNIX, MacOS X, Server editions of the Windows NT family

Many versions MS Windows (NT, XP, Vista, 7, 8, 10, 11) for desktop/workstation allow for separation of data and process, but not for *concurrent* use of the system between multiple users.

Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

8/16/32/64 bit Operating Systems

- Any operating system works with a fixed memory address length — specified in bits
- This limits the number of memory units which can be addressed by the OS
- The upper bound is given by the address bus of the computer architecture

Different Architectures

- **8 bit operating systems** $\equiv$ 256 memory units
  - e.g., GEOS, Atari DOS, Contiki
- **16 bit operating systems** $\equiv$ 65,536 memory units
  - e.g., MS-DOS, Windows 3.x, OS/2 1.x
- **32 bit operating systems** $\approx 4.294 * 10^9$ memory units
  - e.g., Windows 95/98/NT/Vista/7/8/10, OS/2 2/3/4, eComStation, Linux, BeOS, MacOS X (until 10.7)
- **64 bit operating systems** $\approx 18.446 * 10^{18}$ memory units
  - e.g., Linux (64 bit), Windows 7/8 (64 bit), MacOS X (64 bit)

Size and Scope


Source: Wikipedia (Jfreyre), CC BY-SA 3.0

- How Big is an Operating System?

Size and Scope

Source: Wikipedia (Jfreyre), CC BY-SA 3.0

- How Big is an Operating System?
- Which software does the OS comprise?

Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Real-Time Operating Systems (RTOS)

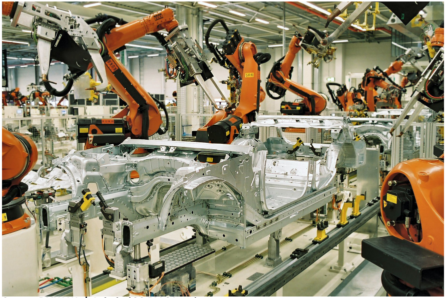
- An RTOS is a multitasking OS which can guarantee to meet certain deadlines
- Typically tasks can be assigned with different priorities
- The ability to meet the desired deadlines may still require precautions by the application developer
- 2 types of real-time operating systems exist:
  - Hard real-time operating systems
  - Soft real-time operating systems

Hard and Soft Real-Time Operating Systems

- Hard real-time operating systems
  - Deadlines are strict
  - Delays cannot be accepted under any circumstances
  - Delays lead to disastrous consequences and high cost
  - Results are useless if they are achieved too late
  - Application examples: Welding robot, reactor control, Anti-lock braking system (ABS), aircraft flight control, monitoring systems of an intensive care unit
- Soft real-time operating systems
  - Certain tolerances are allowed
  - Delays cause acceptable costs
  - Typical applications: Telephone system, parking ticket vending machine, ticket machine, multimedia applications such as audio/video on demand

Applications and Examples of RTOS

- Typical application areas of real-time operating systems:
  - Cell phones
  - Industrial monitoring systems
  - Robots
- Examples of real-time operating systems:
  - QNX
  - VxWorks
  - FreeRTOS
  - RTLinux
  - RIOT



Source: BMW Werk Leipzig (CC-BY-SA 2.0)

Real-Time on Phones

Why does a cell phone (or a smartphone) require an RTOS?

Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Embedded Operating Systems

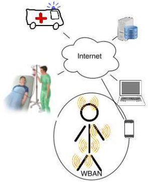
- An embedded system is a computer system with a dedicated function *embedded* in a larger system
- It typically runs without a (direct) human user and therefore often does not offer a user interface (UI)
- It offers typically less hardware resources than traditional desktop or server systems
- Subcategories
  - IoT OS
  - WSN OS
  - Router OS

Do not confuse

RTOS are often embedded OS, but not every embedded OS is an RTOS!

Applications and Examples of Embedded Operating Systems

Mobile Health



Building & Home Automation

Examples

- Embedded Linux
  - Yocto
  - Openmoko
  - Sailfish
  - OpenWRT
  - ...
- Android
- NetBSD
- Windows CE
- TinyOS
- Cisco OS
- NuttX
- ChibiOS
- Symbian

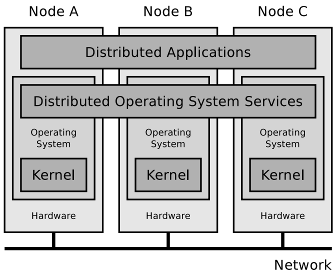
Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Distributed Operating Systems

- A distributed system allows the execution of a distributed application
- Requires networking support
- Controls processes on multiple computers of a cluster
- The individual computers remain transparently hidden from the users and their applications
  - The system appears as a single large computer
    - Single System Image principle

- No implementation of a distributed operating ever gained high practical relevancy
- However, during the development of some distributed operating systems some interesting technologies have been developed and applied for the first time
- Some of these technologies are still relevant today



Distributed Operating Systems (1/2)

- Amoeba
  - Mid-1980s to mid-1990s
  - Andrew S. Tanenbaum (Vrije Universiteit Amsterdam)
  - The programming language Python was developed for Amoeba

`http://www.cs.vu.nl/pub/amoeba/`

The Amoeba Distributed Operating System. *A. S. Tanenbaum, G. J. Sharp.*
`http://www.cs.vu.nl/pub/amoeba/Intro.pdf`

- Inferno
  - Based on the UNIX operating system Plan 9
  - Bell Laboratories
  - Applications are programmed in the programming language Limbo
    - Similar to Java, Limbo produces bytecode, which is executed by a virtual machine
  - Minimal hardware requirements
    - Requires only 1 MB of main memory

`http://www.vitanuova.com/inferno/index.html`

Distributed Operating Systems (2/2)

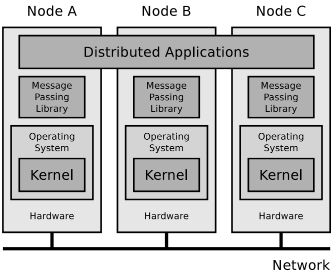
- Rainbow
  - Universität Ulm
  - Concept of a common memory to implement an uniform address space for all host in the distributed system
    - The physical location of any object remains *transparent*
    - Applications can access desired objects via uniform addresses from any computer
    - The OS takes care to transmit and deploy a requested object — if necessary — from a remote host

Rainbow OS: A distributed STM for in-memory data clusters. *Thilo Schmitt, Nico Kämmer, Patrick Schmidt, Alexander Weggerle, Steffen Gerhold, Peter Schulthess*. MIPRO 2011

Distributed Operating Systems – Situation Today

- The concept did not gain acceptance
  - Distributed operating systems never left research projects state
  - Established operating systems have never been replaced

- For developing cluster applications, libraries exist, which provide hardware-independent message passing
  - Message passing communication is based on message exchange
  - Popular message passing systems:
    - Message Passing Interface (MPI) $\Longrightarrow$ standard solution
    - Parallel Virtual Machine (PVM) $\Longrightarrow$ †



MPI tutorials

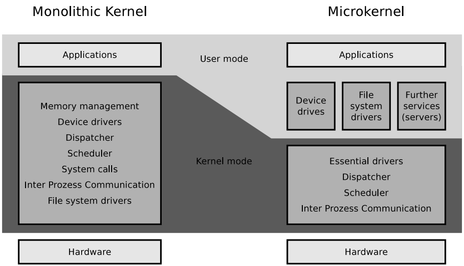
http://mpitutorial.com/tutorials/

Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Kernel Architectures

- The kernel. . .
  - contains the essential functions of the operating system and
  - runs with the highest privileges

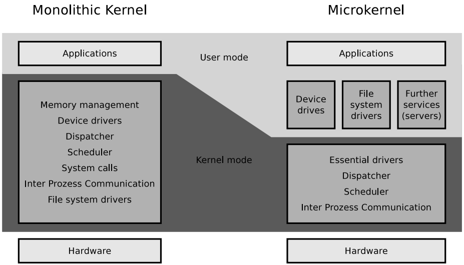


- Different kernel architectures describe which functions are in the kernel and which are outside the kernel as services
- Functions in the kernel, have full hardware access (kernel mode)
- Functions outside the kernel can only access their virtual memory (user mode)

Agenda

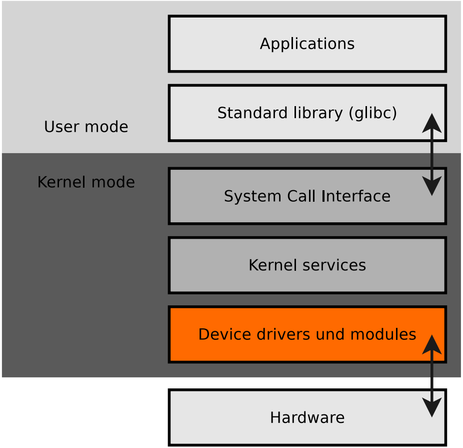
Monolithic Kernels (1/2)

- Contain functions for. . .
  - memory management
  - process management
  - interprocess communication
  - hardware management (drivers)
  - file systems



- **Advantages:**
  - Fewer context switching as with microkernels $\Longrightarrow$ better performance
  - Less complex interaction design
- **Drawbacks:**
  - Crashed kernel components can not be restarted separately and may cause the entire system to crash
  - Kernel extensions cause a high development effort, because for each compilation of the extension, the complete kernel need to be recompiled

Monolithic Kernels (2/2)



- Linux is the most popular modern operating system with a monolithic kernel

Do not confuse a modular kernel design with a microkernel

- It is possible to outsource drivers of the Linux kernel into modules
  - However, the modules are executed in *kernel mode* and not in the *user mode*

  ⇒ Therefore, the Linux kernel is a monolithic kernel

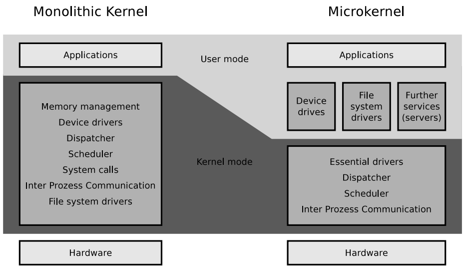
Examples of operating systems with monolithic kernels

Linux, BSD, MS-DOS, FreeDOS, Windows 95/98/ME, MacOS (until 8.6), OS/2

Agenda

Microkernels (1/2)

- The kernel contains only. . .
  - essential functions for memory management and process management
  - functions for process synchronization and interprocess communication (IPC)
  - essential hardware access (e.g., for system start)



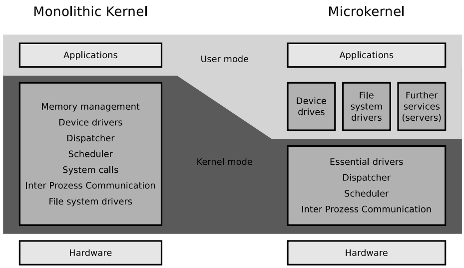
- Device drivers, file systems, and services are located outside the kernel and run equal to the user applications in user mode

Examples of operating systems with microkernels

AmigaOS, MorphOS, Tru64, QNX Neutrino, Symbian OS, GNU HURD (see slide 35), RIOT(?)

Microkernels (2/2)

- **Advantages:**
  - Components can be exchanged easily
  - Best stability and security in theory
    - Reason: Fewer functions run in kernel mode



- **Drawbacks:**
  - Slower because of more context switches
  - Development of a new (micro)kernel is a complex task

The success of the micro-kernel systems, which was forecasted in the early 1990s, did not happen
$\Longrightarrow$ Discussion of Linus Torvalds vs. Andrew S. Tanenbaum (1992) $\Longrightarrow$ see slide 33

Linus Torvalds vs. Andrew Tanenbaum (1992)

Image Source: unknown

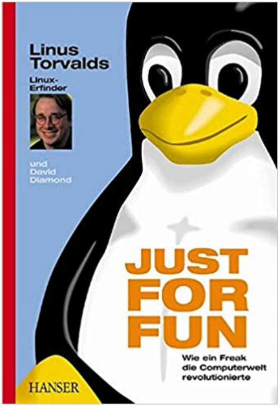
- August 26th 1991: Linus Torvalds announces the Linux project in the newsgroup `comp.os.minix`
  - September 17th 1991: First internal release (0.01)
  - October 5th 1991: First official release (0.02)
- 29. Januar 1992: Andrew S. Tanenbaum posts in the Newsgroup `comp.os.minix`: „**LINUX is obsolete**"
  - Linux has a monolithic kernel $\Longrightarrow$ step backwards
  - Linux is not portable, because it is optimized for the 80386 CPU and this architecture will soon be replaced by RISC CPUs (fail!)

This was followed by an intense and emotional several-day discussion about the advantages and drawbacks of monolithic kernel, microkernels, software portability and free software

A. Tanenbaum (30. January 1992): „*I still maintain the point that designing a monolithic kernel in 1991 is a fundamental error. Be thankful you are not my student. You would not get a high grade for such a design :-)*". Source: `http://www.oreilly.com/openbook/opensources/book/appa.html`

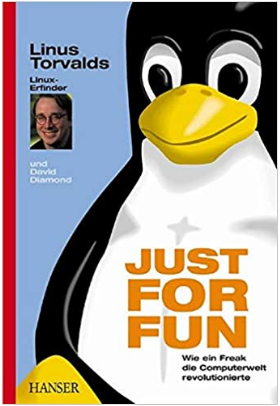
The success of an operating system does not only depend on its architectural design!

„Just for Fun“



- Why did Linus Torvalds begin to implement his own OS?

„Just for Fun“



- Why did Linus Torvalds begin to implement his own OS?
- Why has Linux become the Goto-OS for Internet services?

A sad Kernel Story – HURD

- 1984: Richard Stallman founds the GNU Project
- Objective: Develop a free Unix operating system
  $\Longrightarrow$ GNU HURD
- GNU HURD system consists of:
  - GNU Mach, the microkernel
  - File systems, protocols, servers (services), which run in user mode
  - GNU software, e.g., editors (GNU Emacs), compilers (GNU Compiler Collection (gcc)), shell (Bash),. . .
- GNU HURD is completed *so far*
  - The GNU software is almost completed since the early 1990s
  - Not all servers are completely implemented
- One component is still missing: The microkernel

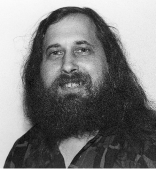


Image source: `stallman.org`

Wikipedia (CC-BY-SA-2.0)

Wikipedia (CC-BY-SA-3.0)

Agenda

- Classifications
  - Tasks and Users
  - Hardware Architectures
  - Real-Time Operating Systems
  - Embedded Operating Systems
  - Distributed Operating Systems
- Kernel Architectures
  - Monolithic Kernels
  - Microkernels
  - Hybrid Kernels
- Structure (Layers) of Operating Systems

Hybrid Kernels / Macrokernels

- Tradeoff between monolithic kernels and microkernels
  - They contain for performance reasons some components, which are never located inside microkernels
- It is not specified which additional components are located inside hybrid kernels
- Windows NT 4 indicates advantages and drawbacks of hybrid kernels
  - The kernel of Windows NT 4 contains the Graphics Device Interface
    - **Advantage:** Increased performance
    - **Drawback:** Buggy graphics drivers cause frequent crashes
- **Advantage:**
  - Better performance as with microkernels because fewer context switching
  - The stability is (theoretically) better as with monolithic kernels

Examples of operating systems with hybrid kernels

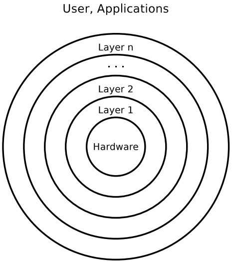
Windows NT family since NT 3.1, ReactOS, MacOS X, BeOS, ZETA, Haiku, Plan 9, DragonFly BSD

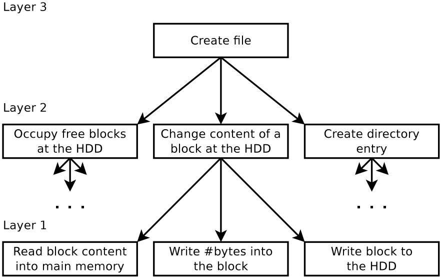
Agenda

Structure (Layers) of Operating Systems (1/2)

- Operating systems can be logically structured via layers
  - The layers surround each other
  - The layers contain from inside to outside ever more abstract functions
- The minimum is 3 layers:
  - The innermost layer contains the hardware-dependent parts of the operating system
    - This layer allows to (theoretically!) easily port operating systems to different computer architectures
  - The central layer contains basic input/output services (libraries and interfaces) for devices and data
  - The outermost layer contains the applications and the user interface
- Usually, operating systems are illustrated with more than 3 logical layers

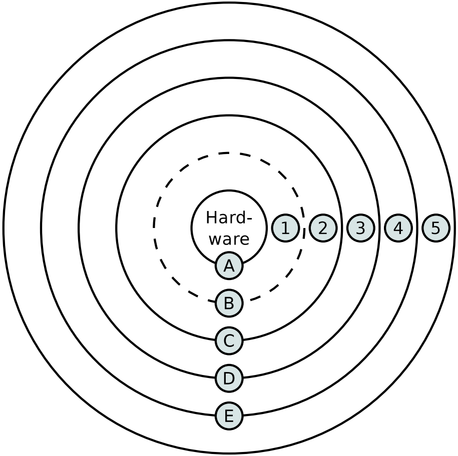
Structure (Layers) of Operating Systems (2/2)





- Each layer is similar with an abstract machine
- Layers communicate with neighboring layers via well-defined interfaces
- Layers can call functions of the next inside layer
- Layers provide functions to the next outside layer
- All functions (services), which are offered by a layer, and the rules, which must be observed, are called protocol

Layers of Linux/UNIX

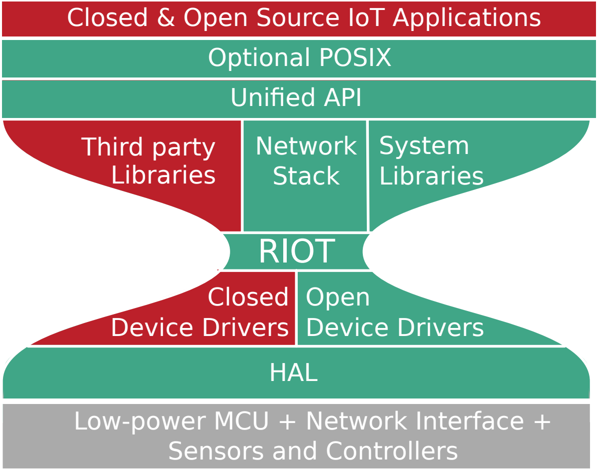


1. Kernel (hardware-dependent part)
2. Kernel (hardware-independent part)
3. Standard library (glibc)
4. Shell (bash), Applications
5. User

- A: Hardware interface
- B: Kernel-internal, hardware-independent interface
- C: System call interface
- D: Standard library interface (interface to glibc)
- E: User interface

In practice, the concept is not strictly followed all the time. User applications, can e.g., call wrapper function of the standard library glibc or directly call the system calls)

Layers of RIOT



You should now be able to answer the following questions:

- What are the differences between singletasking and multitasking or single-user and multi-user operation?
- How can operating systems be categorized with respect to their applications?
- What is the kernel of an OS and which different architectures exist?
- How can an OS be structured via layers and what is their purpose?