2nd Slide Set Operating Systems

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences
(1971–2014: Fachhochschule Frankfurt am Main)
Faculty of Computer Science and Engineering
christianbaun@fb2.fra-uas.de

Learning Objectives of this Slide Set

Operating systems can be classified according to different criteria

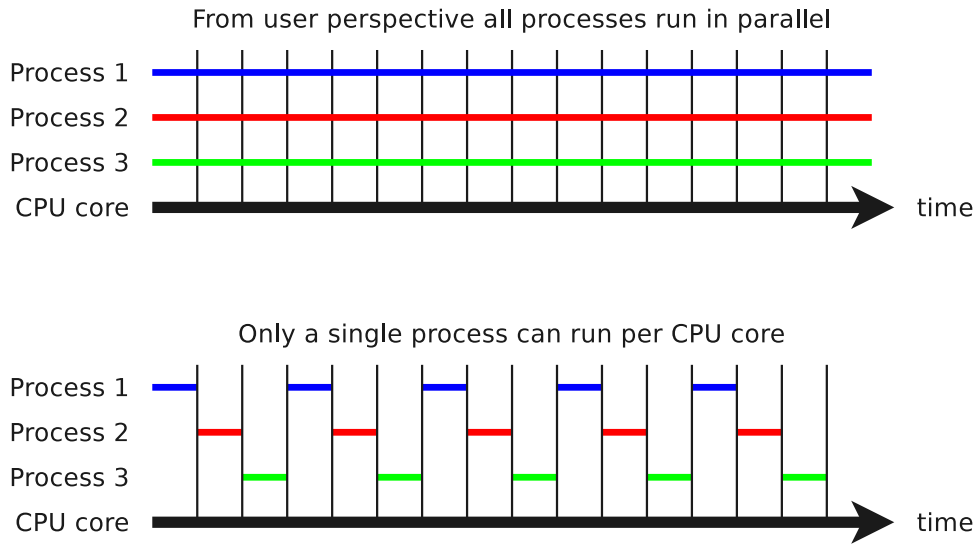
The most important distinction criteria are discussed in this slide set

- At the end of this slide set You know/understand...
  - the difference between **singletasking** and **multitasking**
  - the difference between **single-user** and **multi-user**
  - the reason for the **memory address length**
  - what **real-time operating systems** are
  - what **distributed operating systems** are
  - the different **kernel architectures**
    - **Monolithic kernel**
    - **Microkernel**
    - **Hybrid kernel**
  - the **structure (layers)** of operating systems

Exercise sheet 2 repeats the contents of this slide set which are relevant for these learning objectives

Singletasking and Multitasking

- **Singletasking**
  - At any given moment, only a single program 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 **quasi-parallel**



Task, process, job,. . .

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

Why Multitasking?

Wir already know...

- With **multitasking**, multiple processes are executed concurrently
- The processes are activated alternately at short intervals
  $\Longrightarrow$ For this reason, **the execution appears simultaneous**
- Drawback: Switching from one process to another one causes **overhead**

- Processes often need to wait for external events
  - External events may be user inputs, input/output operations of peripheral devices, or simply waiting for a message from another program
  - With multi-tasking, processes, which wait for incoming E-mails, successful database operations, data written to the HDD or something similar can be placed in the background
    - This way, other **processes can be executed sooner**
- The program switching, which is required to implement the quasi-parallel execution, causes overhead
  - **The overhead is rather small compared with the speedup**

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 (as fair as possible)
    - Users must be identified (via passwords)
    - Access to data/processes of other users must be prevented

	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

- Desktop/Workstation versions of Windows NT/XP/Vista/7/8/10 are only **half multi-user operating systems**
  - Different users can work with the system only one after the other, but the data and processes of the different users are protected from each other

8/16/32/64 bit Operating Systems

- The bit number indicates the **memory address length**, with which the operating system works internally
  - The number of memory units, an operating system can address, is limited by the address space, which is limited by the address bus $\Longrightarrow$ slide set 3
- **8 bit operating systems** can address 2^8 memory units
  - e.g. GEOS, Atari DOS, Contiki
- **16 bit operating systems** can address 2^{16} memory units
  - e.g. MS-DOS, Windows 3.x, OS/2 1.x

Bill Gates (1989)

„We will never make a 32-bit operating system.“

- **32 bit operating systems** can address 2^{32} 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** can address 2^{64} memory units
  - e.g. Linux (64 bit), Windows 7/8 (64 bit), MacOS X (64 bit)

Real-Time Operating Systems

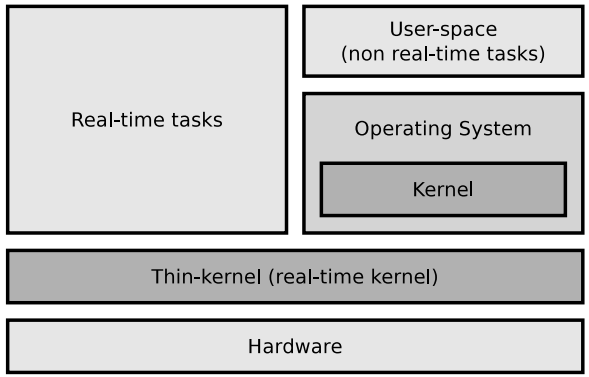
- Are multitasking operating systems with additional real-time functions for the compliance of time conditions
- Essential criteria of real-time operating systems:
  - **Response time**
  - Meet **deadlines**
- Different priorities are taken into account so that important processes are executed within certain time limits
- 2 types of real-time operating systems exist:
  - **Hard real-time operating systems**
  - **Soft real-time operating systems**
- Modern desktop operating systems can **guarantee** soft real-time behavior for processes with high priority
  - Because of the unpredictable time behavior due to swapping, hardware interrupts, etc. **hard real-time behavior** cannot be guaranteed

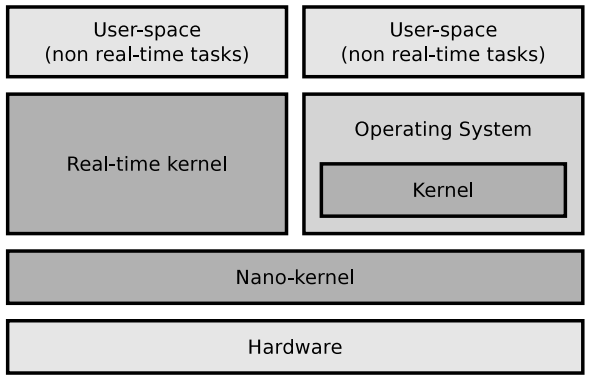
Hard and Soft Real-Time Operating Systems

- **Hard real-time operating systems**
  - Deadlines must be strictly met
  - 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

Architectures of Real-Time Operating Systems

- **Thin kernel**
  - The operating system kernel itself runs as a process with lowest priority
  - The real-time kernel does the scheduling
  - Real-time processes have the highest priority
    $\Longrightarrow$ minimum reaction time (latency)
- **Nano kernel**
  - In addition to the real-time kernel, any number of kernels of other operating systems may be executed
- **Pico kernel, Femto kernel, Atto kernel**
  - Marketing buzz-words of vendors of real-time operating systems to emphasize the smallness of their real-time kernel





Source: Tim Jones (2008). Anatomy of real-time Linux architectures
`http://www.ibm.com/developerworks/library/l-real-time-linux/`

Application Areas of Real-Time Operating Systems

- Typical application areas of real-time operating systems:
  - Cell phones
  - Industrial monitoring systems
  - Robots
- Examples of real-time operating systems:
  - QNX
  - VxWorks
  - LynxOS
  - RTLinux
  - Symbian (outdated)
  - Windows CE (outdated)

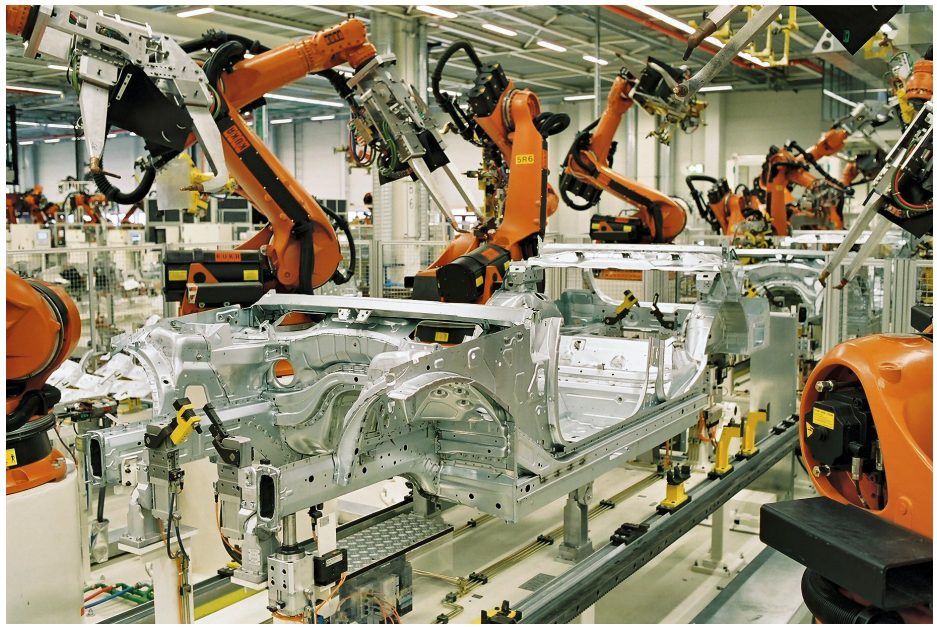


Image source: BMW Werk Leipzig (CC-BY-SA 2.0)

Have some Fun with the QNX Demo Disc from 1999...

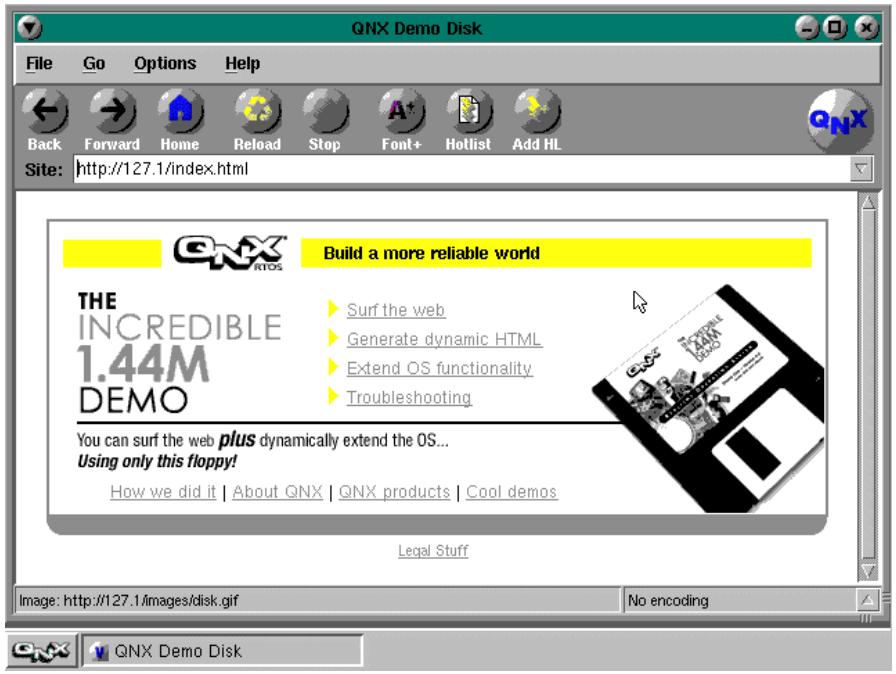


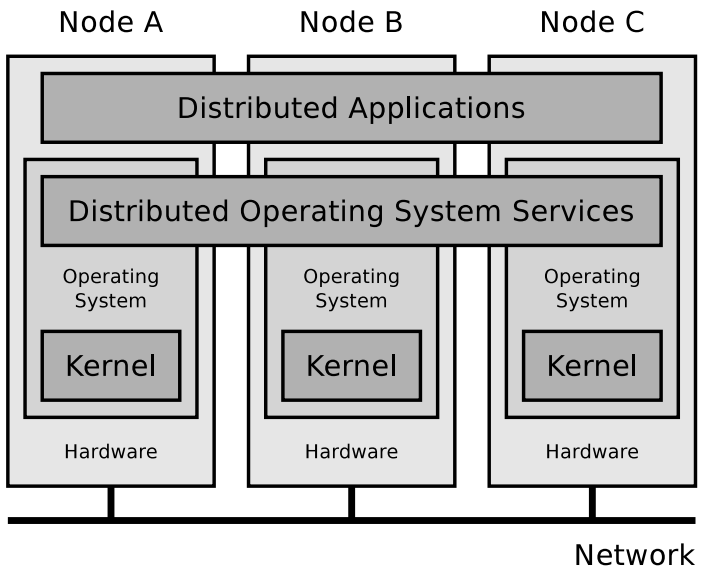
Image source:
`http://toastytech.com/guis/qnxdemo.html`

Impressive video of the demo disc:
`https://www.youtube.com/watch?v=K_VlI6IBEJ0`

Distributed Operating Systems

- Distributed system
- 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

- The principle of distributed operating systems is dead!
- 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 (1/3)

- **Amoeba**
  - Mid-1980s to mid-1990s
  - Andrew S. Tanenbaum (Free University of 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 (2/3)

- **Rainbow**
  - Universität Ulm
  - Concept of a common memory to implement an uniform address space, in which all computers in the cluster can store and access objects
    - For applications, it is transparent, on which computer in the cluster, objects are physically located
    - Applications can access desired objects via uniform addresses from any computer
    - If the object is physically located in the memory of a remote computer, Rainbow does the transmission and local deployment to the requesting computer in an automated and transparent way

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 (3/3)

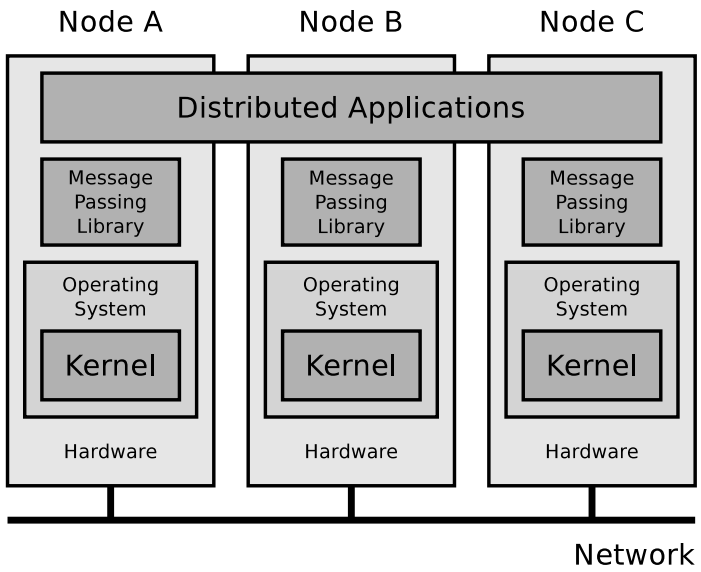
- **Sprite**
  - University of California, Berkeley (1984-1994)
  - Connects workstations in a way that they appear to users as a single time-shared system
  - The parallel version of `make`, called `pmake` was developed for Sprite

`http://www.stanford.edu/~ouster/cgi-bin/spriteRetrospective.php`

The Sprite Network Operating System. 1988. `http://www.research.ibm.com/people/f/fdouglis/papers/sprite.pdf`

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) $\Longrightarrow$ †

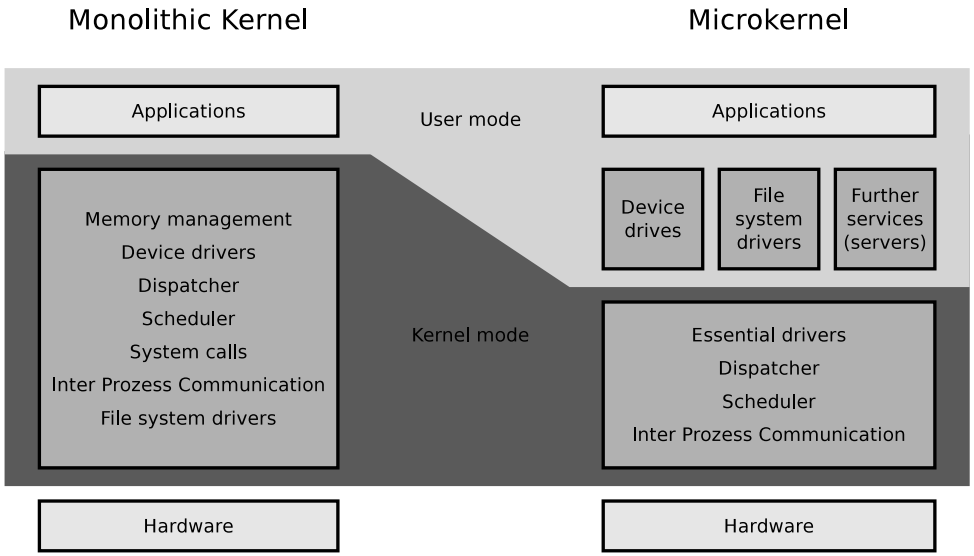


MPI tutorials

`http://mpitutorial.com/tutorials/`

Kernel Architectures

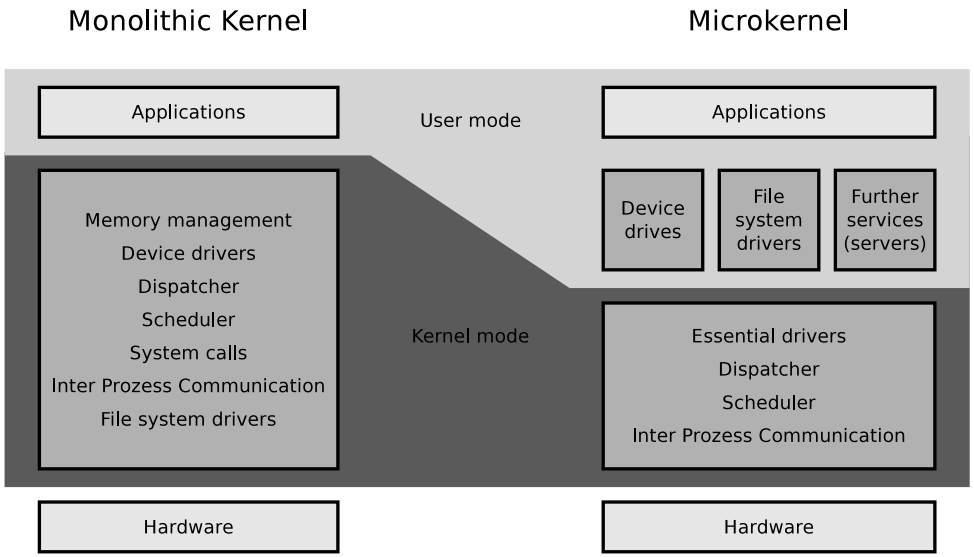
- The **kernel**...
  - contains the essential functions of the operating system and
  - is the interface to the hardware



- Different kernel architectures exist
  - They differ in which functions are included inside **in the kernel** and which functions are **outside the kernel** as services (servers)
- Functions in the kernel, have full hardware access (**kernel mode**)
- Functions outside the kernel can only access their virtual memory (**user mode**)
  $\Longrightarrow$ slide set 5

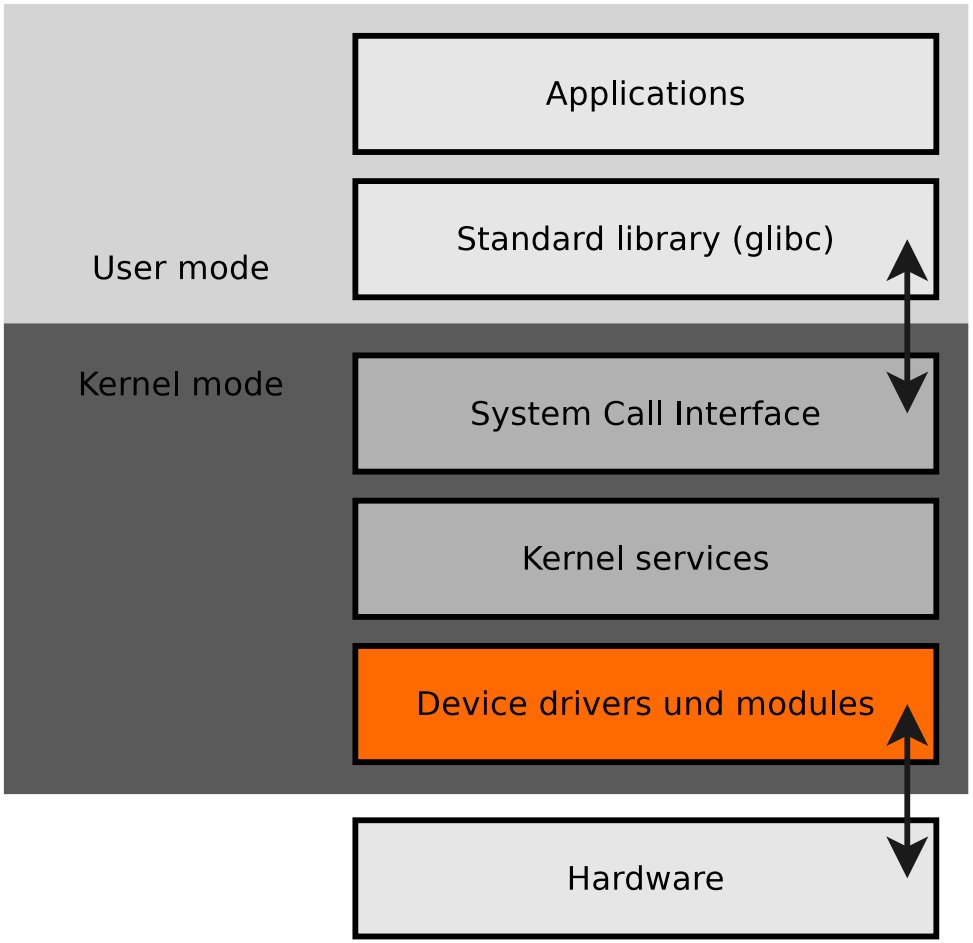
Monolithic Kernels (1/2)

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



- Advantages:
  - Fewer context switching as with microkernels $\Longrightarrow$ better performance
  - Grown stability
    - Microkernels are usually not more stable compared with monolithic kernels
- 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)



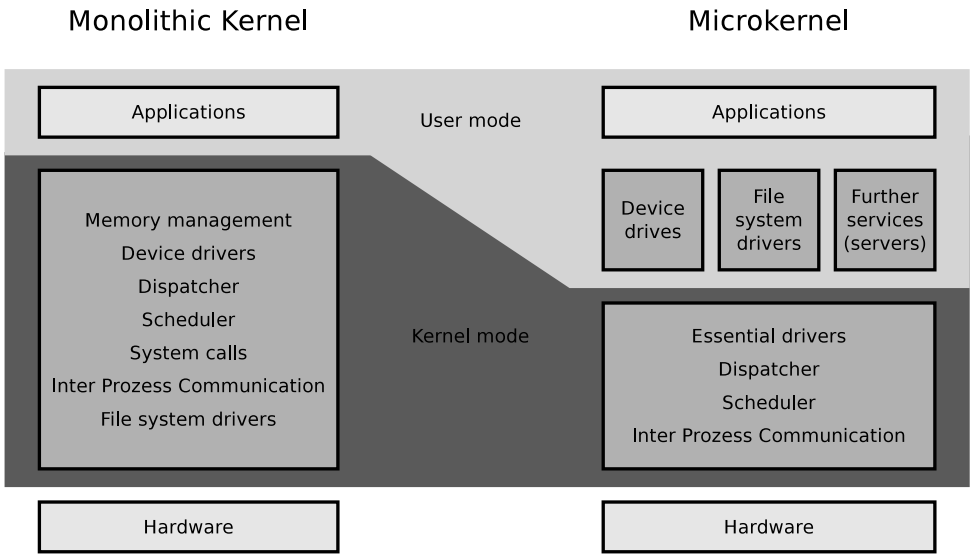
- Linux is the most popular modern operating system with a monolithic kernel
- It is possible to outsource drivers of the **Linux kernel** into modules
  - However, the modules are executed in *kernel mode* and not in the *user mode*
    - Therefore, the Linux kernel is a monolithic kernel

Examples of operating systems with monolithic kernels

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

Microkernels (1/2)

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



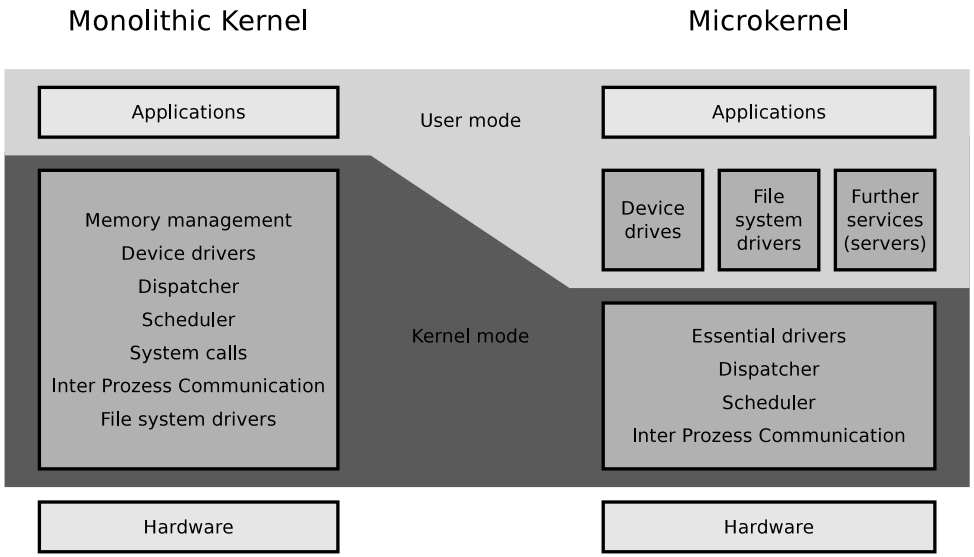
- Device drivers, file systems, and services (servers) are located outside the kernel and run equal to the user applications in user mode

Examples of operating systems with microkernels

AmigaOS, MorphOS, Tru64, QNX Neutrino, Symbian OS, GNU HURD (see slide 24)

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 23

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

Source: **MS Windows NT Kernel-mode User and GDI White Paper**. `https://technet.microsoft.com/library/cc750820.aspx`

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

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 future can not be predicted

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), 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

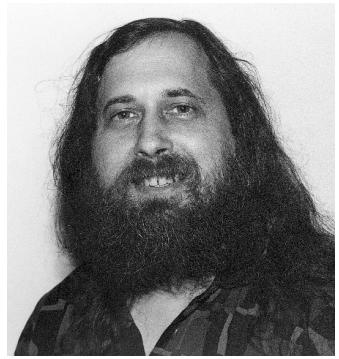
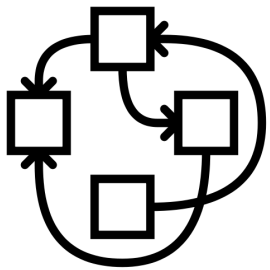


Image source: `stallman.org`

Wikipedia (CC-BY-SA-2.0)



Wikipedia (CC-BY-SA-3.0)

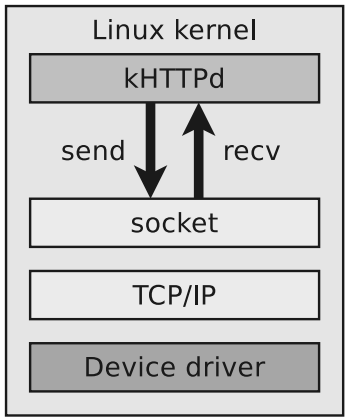
An extreme Kernel Story – kHTTPd

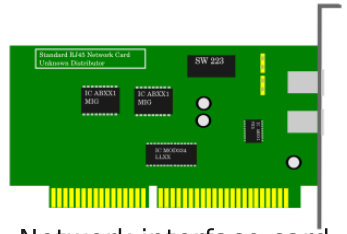
http://www.fenrus.demon.nl

- 1999: Arjan van de Ven develops the **kernel-based web server** kHTTPd for Linux kernel 2.4.x

The Design of kHTTPd: https://www.linux.it/~rubini/docs/khttpd/khttpd.html
Announce: kHTTPd 0.1.0: http://static.lwn.net/1999/0610/a/khttpd.html

- Advantage: Faster delivery of static(!) web pages
  - Less switching between user mode and kernel mode is required
- Drawback: Security risk
  - Complex software like a web server should not run in kernel mode
  - Bugs in the web server could cause system crashes or enable an attacker to takeover system control
- Linux kernel $\geq$ 2.6.x does not contain kHTTPd





Network interface card

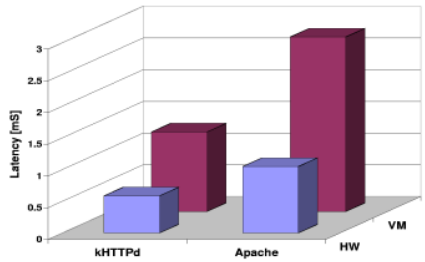
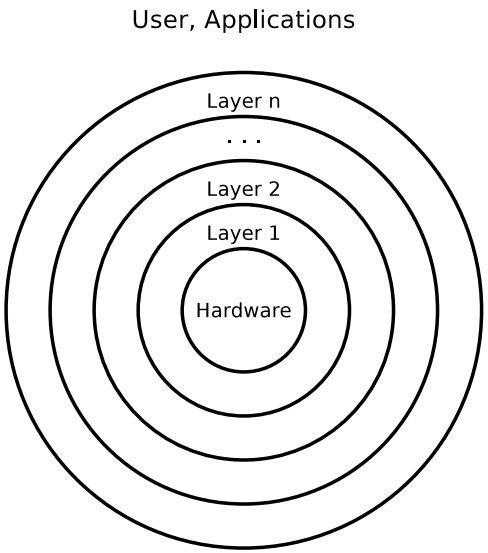


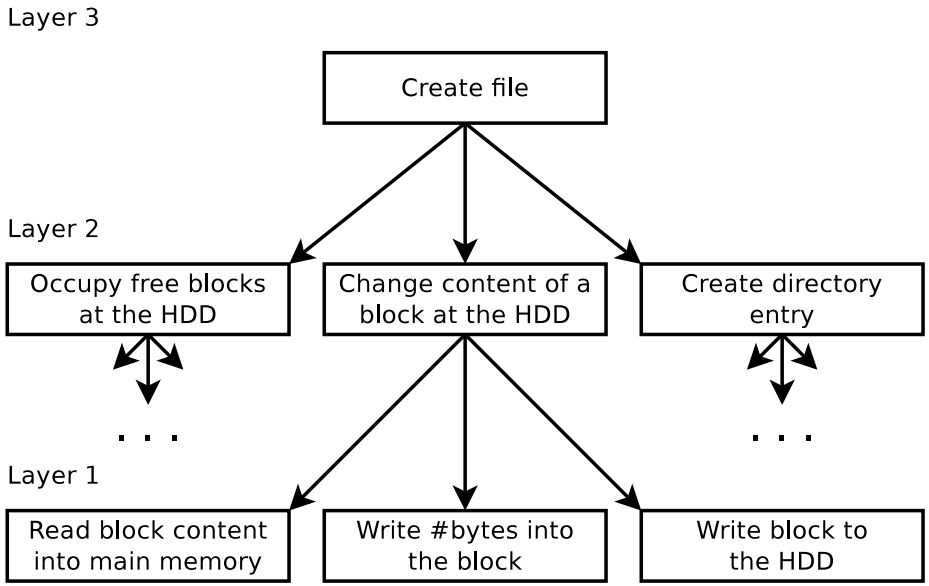
Image source:
Kernel Plugins: When A VM Is Too Much. *Ivan Ganev, Greg Eisenhauer, Karsten Schwan*. 2004

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

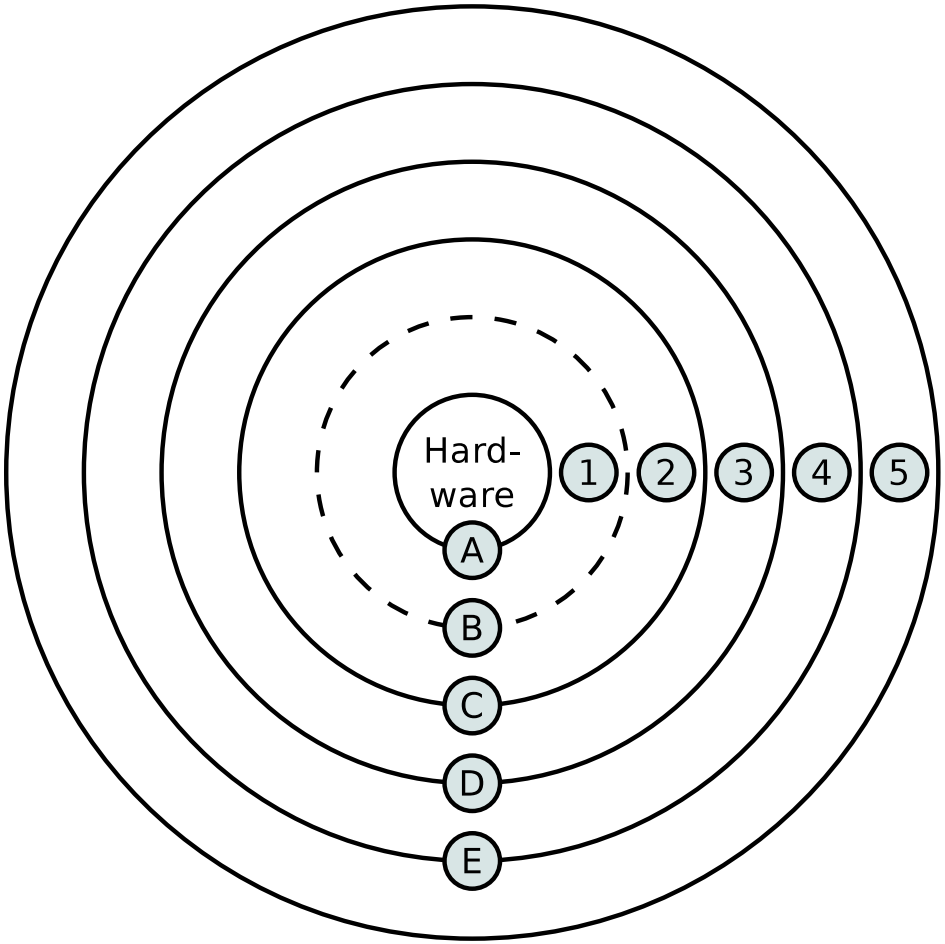
Structure (Layers) of Operating Systems (2/2)





- 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 of 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 ($\Longrightarrow$ see slide set 7)