System-Level Programming

24 Operating Systems

Peter Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

Summer Term 2025

http://sys.cs.fau.de/lehre/ss25

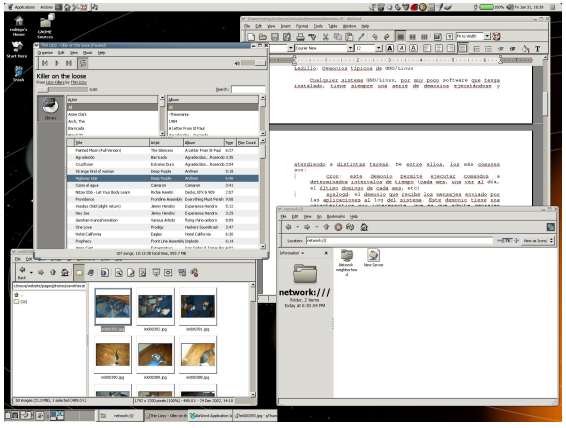
Definition “Operating System”

- DIN 44300
  - “... the programs of a digital computing system which, together with the properties of the computing system, form the basis of the possible operating modes of the digital computing system and that particularly control and monitor the execution of programs”
- Andrew S. Tannenbaum
  - “... a *software layer* ..., that manages all parts of a system and provides the user with an interface or virtual machine that is easier to understand and program [than the bare hardware].”
- Conclusion:
  - Software for managing and *virtualizing* the hardware components (i.e., resources)
  - Program for *controlling & monitoring* other programs

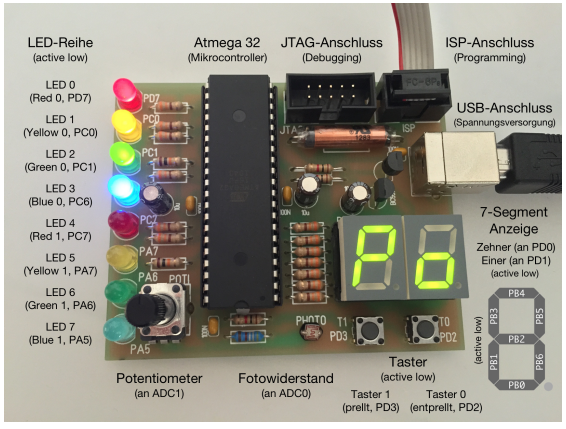
Overview

Previous lectures:

- **One** program that
- controls its environment,
- **alone**,
- started **during boot**,
- with **hardware accesses**.



Source: www.wikipedia.org



Now:

- **Multiple** programs that
- control their environment,
- **concurrently**,
- started/stopped **dynamically**,
- via **defined I/O functions**.

Multitasking

If more than one application exists on a system (“multitasking”),

- the applications have to coordinate
  - who and when can access the/one *CPU*,
  - who can use which *memory* areas,
  - who can use which part of the *disk/hard drive*,
  - who is allowed to display which part on the *screen*,
  - ...

  Since no application can decide on its own, for example, which areas in the memory are still unused, **shared methods and state variables** are required.
- It has to be ensured that
  - all applications meet the agreements
    (even those, that are (un)intentionally programmed erroneously!)
- **Hardware extensions** have to restrict access to unauthorized memory areas or I/O devices.

Operating System Kernel

- **Shared methods and state variables**
  - **Operating-system kernel** (also kernel or system kernel)
- **Hardware extensions**
  - **Levels of privilege**: “rings”
  - **Memory protection**: memory-management unit (MMU)

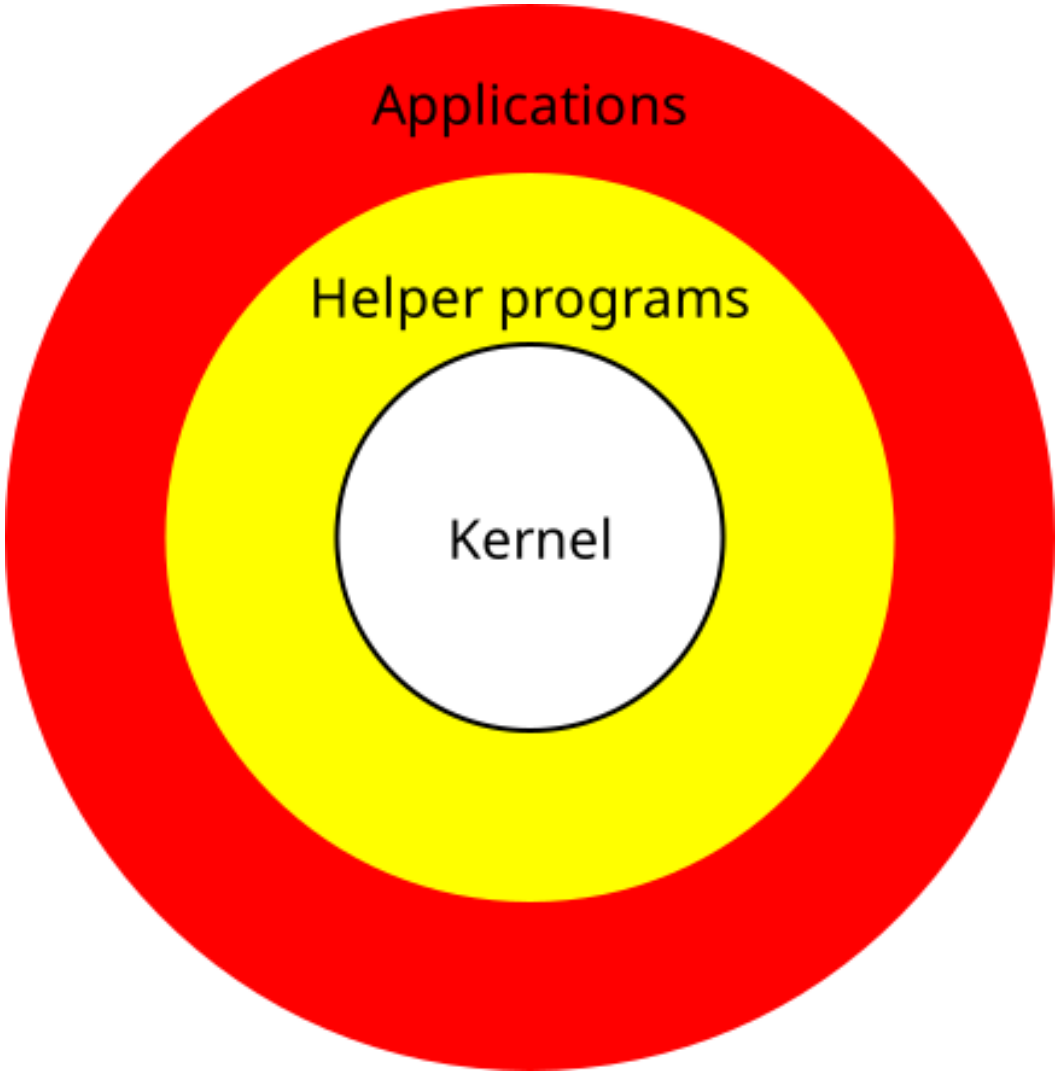
Layer Model

Definition “**operating system**”:

Operating system:
Kernel *and* auxiliary programs

or (depending on perspective)

Operating system:
Only the kernel



Levels of Privilege

- Unprivileged layers (“application layer”, “user layer”, “user ring”)
  - may execute “normal” CPU instructions
  - may access its assigned memory areas
  - may call OS functions
- Privileged layer (“system layer”, “kernel layer”, “ring 0”)
  - may execute all CPU instructions
  - may access every memory area
  - may reconfigure the memory protection
  - may access I/O devices

Switch to privileged layer by

- System calls or traps
- Interrupts
- Exceptions

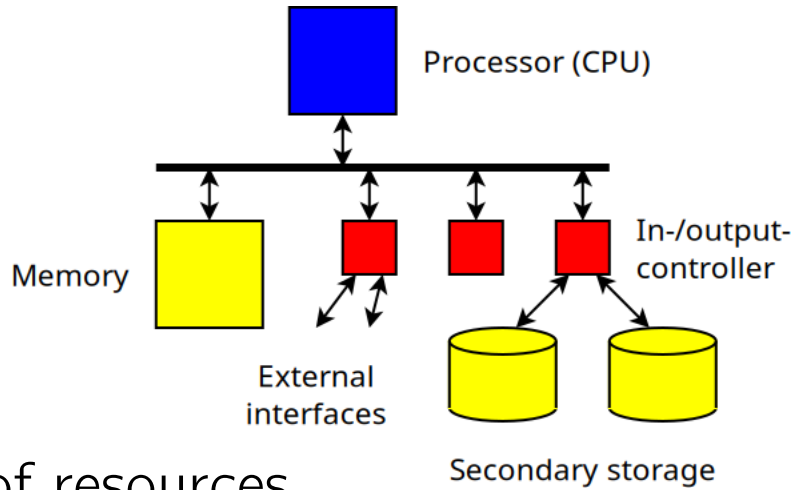
System Calls

Example: Application needs *more memory resources*, step by step:

- application calculates how much more memory is needed
- stores parameter in CPU registers
- switches in the kernel with a special CPU instruction,
  ($\Rightarrow$ from now on privileged!)
- reads parameters from the CPU register
- reserves more memory for itself
- reprograms the MMU
- stores the result in CPU registers
- switches back to the application layer with special CPU instructions,
  ($\Rightarrow$ from now on unprivileged again!)
- retrieves result from the CPU register

Management of Resources

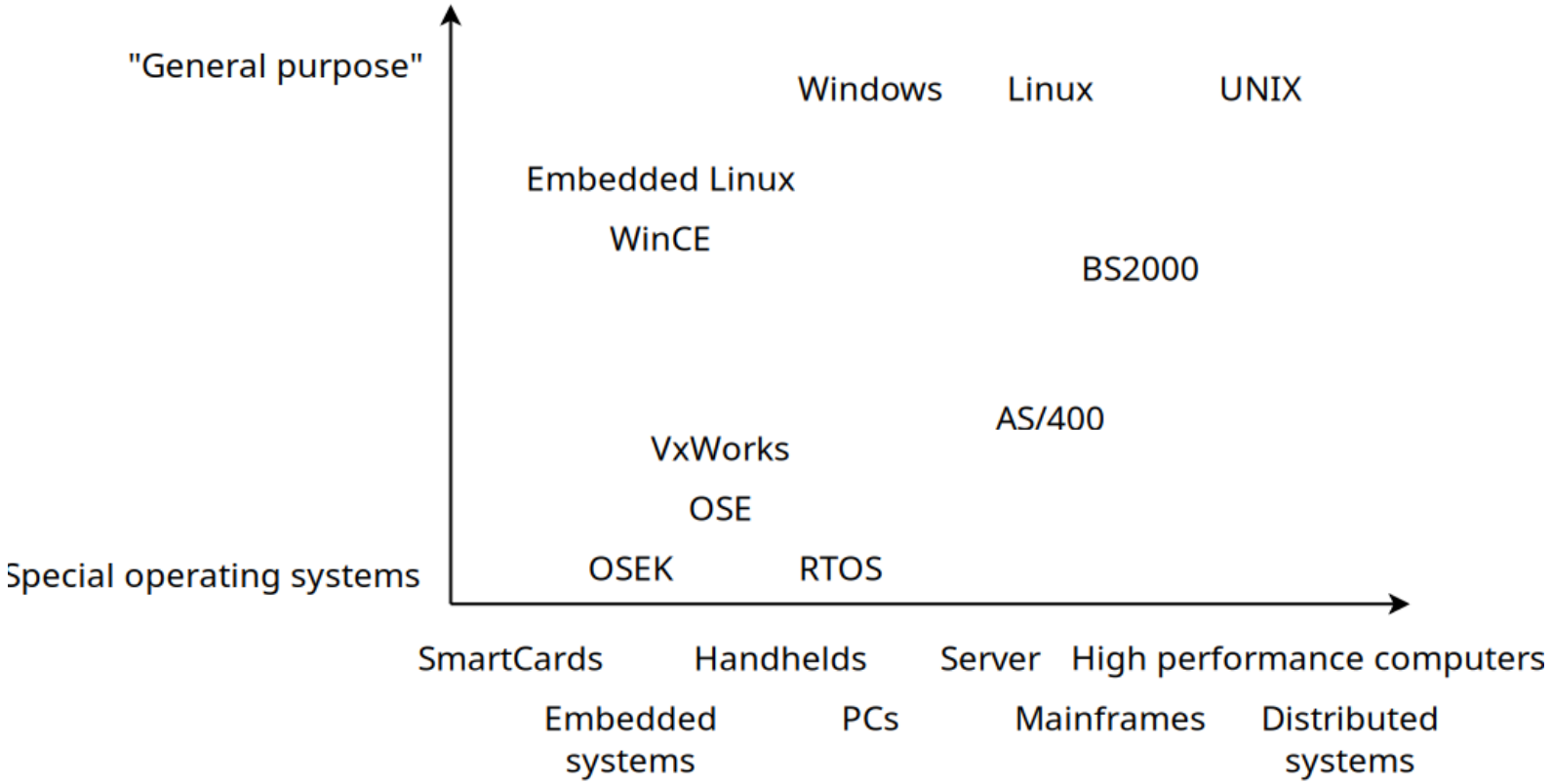
- Responsibilities of the operating-system kernel
  - **multiplexing** of resources for multiple users or applications
  - providing **protections**
  - providing **abstractions** for easier handling of resources



- Enable a coordinated shared usage of resources, which can be classified:
  - active, **timely divisible** (processor)
  - passive, only **exclusively usable** (peripheral devices, e. g., printers, etc.)
  - passive, **spatially divisible** (memory, disk space, etc.)
- Support for recovering from errors (segmentation fault)

Classification of Operating Systems

- Different criteria for classification
  - target platform
  - intended use
  - functionality



Classification of Operating Systems (2)

- Compared to only a small number of “general purpose”-, mainframe- and high-performance computer operating systems, there exits a multitude of small and smallest specialized operating systems:

  C51, C166, C251, CMX RTOS, C-Smart/Raven, eCos, eRTOS, Embos, Ercos, Euros Plus, Hi Ross, Hynet-OS, LynxOS, MicroX/OS-II, Nucleus, OS-9, OSE, OSEK Flex, OSEK Turbo, OSEK Plus, OSEKtime, Pricise/MQX, Pricise/RTCS, proOSEK, SOS, PXROS, QNX, Realos, RTMOSxx, Real Time Architect, ThreadX, RTA, RTX51, RTX251, RTX166, RTXC, Softune, SSXS RTOS, VRTX, VxWorks, ...

  Usage: embedded systems, often real-time systems, more than 50% proprietary (in-house) solutions

- Alternative classification: by architecture

Architecture of Operating Systems

- Scope: tens of thousands or even *multiple millions of code lines*
  $\Rightarrow$ structuring required
- Various structural concepts
  - runtime libraries (minimal, mostly used for embedded systems)
  - monolithic systems
  - layered systems
  - microkernels (minimal kernels)
- Various protection concepts
  - no protection (µController)
  - protection of the operating system
  - protection of the operating system and between applications
  - fine-grained protection within application

Components of Operating Systems

- Memory management
  - Who is allowed to store information in memory?
- Process management
  - When is which task scheduled?
- File system
  - storage and protection of long-term data
- Inter-process communication (IPC)
  - communication between different applications or between executed parts (running in parallel) of an application
- Input/Output
  - communication with the “world outside” (user/computer)