Introduction and Overview

DM510 Operating Systems

Lars Rohwedder

Windows macOS Android iOS

Disclaimer

These slides contain (heavily modified) content and media from the official Operating System Concepts slides:
https://www.os-book.com/OS10/slide-dir/index.html

Today's lecture

- Organization of course
- Brief introduction to operating systems

Vote for the best operating system

forms.office.com/e/jChPB7FLg6

- Microsoft Windows
- macOS
- Linux (any desktop distribution)
- Android
- iOS
- Windows Mobile/Phone
- chromeOS

Your teacher

poll: forms.office.com/e/jChPB7FLg6

Lars Rohwedder

Short Bio

- Since 2024: Associate prof. at IMADA, SDU.
- 2022-2024: Assistant prof. at Maastricht University
- 2019-2021: Post-Doc at EPFL
- 2019: PhD from University of Kiel
- Nowadays research focus on algorithms
- Focus on systems during early career: Research assistant at Oracle, 2013-2014, and VMWare, 2015 (both San Francisco Bay Area)

Your teacher

poll: forms.office.com/e/jChPB7FLg6

Lars Rohwedder

Short Bio

- Since 2024: Associate prof. at IMADA, SDU.
- 2022-2024: Assistant prof. at Maastricht University
- 2019-2021: Post-Doc at EPFL
- 2019: PhD from University of Kiel
- Nowadays research focus on algorithms
- Focus on systems during early career: Research assistant at Oracle, 2013-2014, and VMWare, 2015 (both San Francisco Bay Area)

Tutors

Carl-Gustav Øboe Rasmussen and Bastian Graver Blohm

Course goals

Understand

- what an operation system does
- How it does this
- How to use it
- How to program it

Course goals

Understand

- what an operation system does
- How it does this
- How to use it
- How to program it

Placement within your curriculum

- Natural continuation of DM548: Computer architecture
- Along with DM548 and DM546 (Compiler Construction) closest to hardware among CS courses

Course goals

poll: forms.office.com/e/jChPB7FLg6

Understand

- what an operation system does
- How it does this
- How to use it
- How to program it

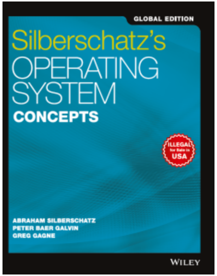
Placement within your curriculum

- Natural continuation of **DM548: Computer architecture**
- Along with DM548 and DM546 (Compiler Construction) closest to hardware among CS courses

Practical skills from this course

- improve your C and (low level) systems programming skills
- improve your Linux skills
- (for high level programming) understand and solve performance issues

Resources



Textbook

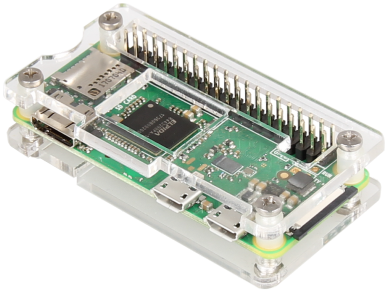
- Lectures based on different chapters from book
- Digital or in paper from academic books
- Not strict requirement, but highly recommended

Additional resources

- https://larsrohwedder.com/teaching/dm510-26 for everything you need (link also on itslearning)
- Online sources for Linux specific documentation (relevant for programming exercises): see course website
- Explanations of tools used throughout course: see course website

Course organization

poll: forms.office.com/e/jChPB7FLg6



Week structure:

- lecture on Monday - exercise sheet given out
- lecture on Wednesday (every other week)
- tutorial on Thursday - prepare exercise sheet before

Programming projects:

- 3 parts, each roughly 1 month
- In teams of 2
- based on Raspberry Pi Zero mini-computer

Assessment

- 80% of grade comes from written exam on
  - content from textbook, which also appears in slides and exercises
  - Linux and C, as in exercises and programming project
  - more information later
- 20% comes from programming projects

What is an Operating System?

Examples

Microsoft Windows

- 70% market share for desktop computers
- **closed-source and proprietary**
- not UNIX-based

MacOS

- Open-source kernel (based on BSD-Unix)
- Contains also **closed-source and proprietary** components
- Used almost exclusively with Apple hardware
- Basis for mobile operating system **iOS**

GNU/Linux

- GNU is a vast collection of free software projects initiated by Richard Stallman's Free Software Foundation
- Linux is Unix-like OS initially developed by Linus Torwalds, now contains contributions from thousands of volunteers
- Linux comes in different distributions (for example, Ubuntu, Linux Mint, Arch Linux,...) that share the Linux kernel, but different non-kernel software
- Most famous **free and open-source** operating system
- Basis for mobile operating system **Android**

Unix: standardization for APIs (known as **POSIX**), command-line, "philosophy", etc.

Components of an operating system

Shipped in a typical OS

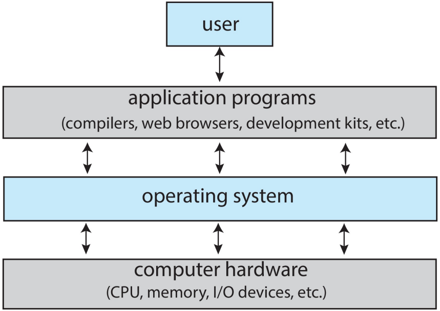
- bootloader
- kernel (main program of operating system)
- device drivers
- system programs: graphical user interface, terminal, file browser, device management, etc.
- application/user programs: PDF viewer, web browser, etc.
- middleware: APIs and software frameworks (e.g. python/java runtime).
- ...

It is debatable which of these components should be considered part of the operating system

Role of an operating system

User perspective

- Executes user programs
- Makes the computer system convenient to use
- Uses the computer hardware in an efficient manner

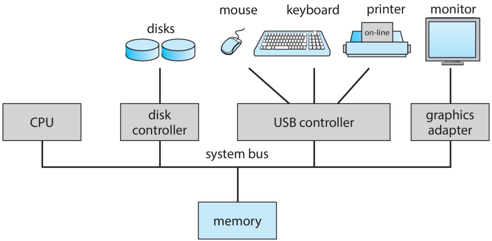


System perspective

- Provides **kernel**, the system's "main program" that is always running
- All interaction between user (program) and hardware goes through the kernel
- **resource allocator:** decide who gets which hardware resources
- **control program:** prevent errors and improper use

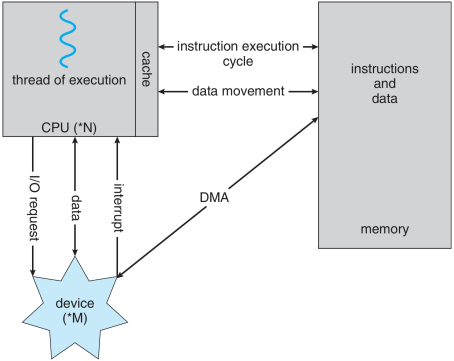
Hardware Fundamentals

Typical components of a computer



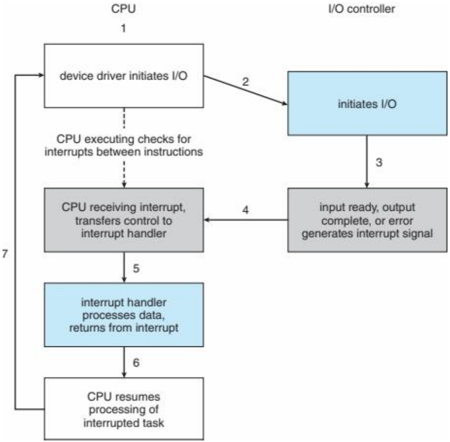
- Next to CPU, systems contain various device controllers (small processing units) that execute concurrently
- Device controllers have local buffers (memory) of limited size

CPU architecture and I/O



data transfer: between CPU registers and local buffers
direct memory access (DMA): For efficiency, devices can directly read and write to main memory. This requires CPU activity only before and after transfer of an entire block of data

Von Neumann Architecture: Instructions and data are fetched from same main memory



Interrupts

- CPU has interrupt bit in hardware that is checked before every instruction. If it is set/active, we interrupt the current process and execute an interrupt handler of the kernel. Afterwards we can resume interrupted process.
- Interrupts transfer the control over CPU from user processes to kernel
- Transfer involves context switch: Need to backup registers, program counter, etc. and restore them later
- Examples of hardware interrupts: I/O transfer finished, timer, keyboard input
- Sometimes software (CPU instructions) intentionally or unintentionally causes interrupts and gives control back to kernel. A software interrupt is called a **trap**[1]
- Everytime a user program makes a request to operating system, a **system call**, this is done by issuing a trap
- Examples of unintentional traps are errors (e.g., division by zero) or page faults

[1] terminology in literature is sometimes inconsistent.

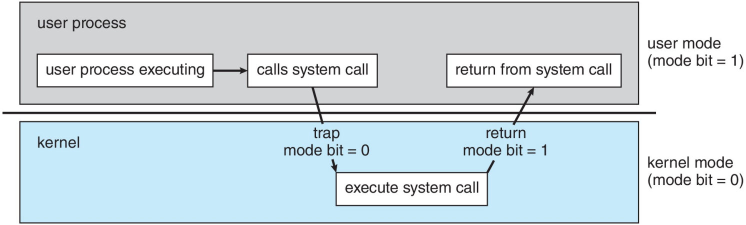
Interrupt Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detect overflow
5	bound range exception
⋮	
32-255	maskable interrupts

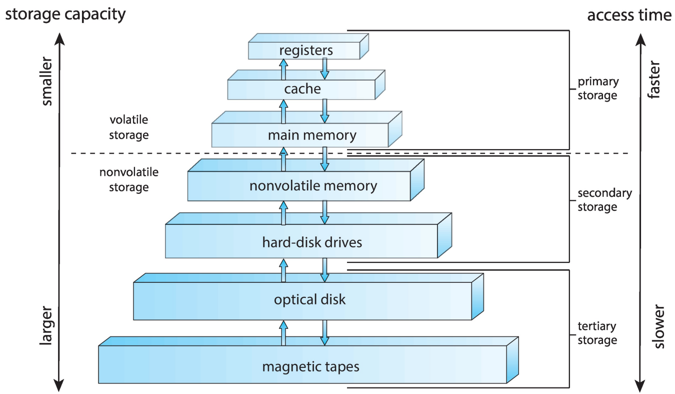
- Different types of interrupts specified in interrupt table
- Table used to jump to different handlers depending on type
- Some interrupt types can be **masked**: turned off by special instruction
- Prioritization can be necessary: decides when one interrupt can preempt other interrupt handler

Dual-mode Operation

- Each CPU core has a mode bit implemented in hardware that indicates **user mode** (= user process is active) or **kernel mode** (= kernel is active)
- Certain instructions are **privileged** and can only be executed when in kernel mode. Example: I/O requests to devices
- This is for protection against errors and malicious software
- Modern CPUs have more than two modes, for example for virtual machines or more fine-grained control over privileges



Storage



- > **volatile** storage is lost when computer turned off
- > kernel organizes non-volatile storage with file systems, etc.

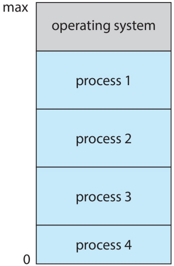
Multi-user, Multitasking, Parallelism

Users

- Operating system maintains persistent table of several **users**
- Users can belong to **groups**
- Processes and files are owned by specific users
- For security/protection: Access privileges per user/group and file/program. Example: Typical Linux program `apt` (package manager) can only be executed by superuser (root).

Note: not to be confused with privileged instructions (kernel/user mode)

Multitasking



sharing main memory

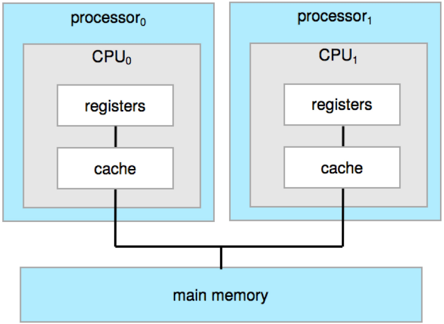
- Many processes (each with multiple threads) and users can be active at the same time, even on single core systems
- Each one wants low response time (< 1 second) and fair share of resources
- Resource utilization should be high
- **Synchronization**: Concurrent access to shared resources/devices needs to be safe

Sharing of resources is mostly hidden from processes

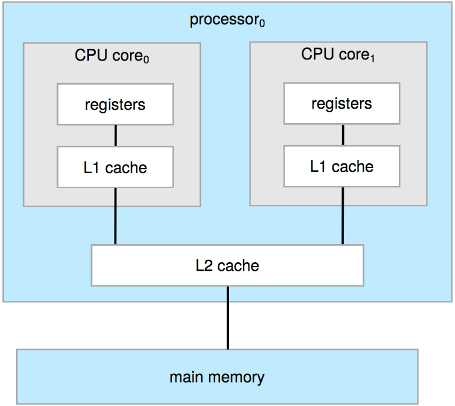
- CPU executes processes alternatingly, managed by CPU scheduler, but oblivious to processes
- Virtual memory ensures that processes do not see memory of other processes
- **Exception:** synchronization usually needs to be done explicitly

Physical parallelism

Modern computers have several CPUs or CPU cores, complicating CPU scheduling, synchronization, and caching



Multiprocessor



Multicore

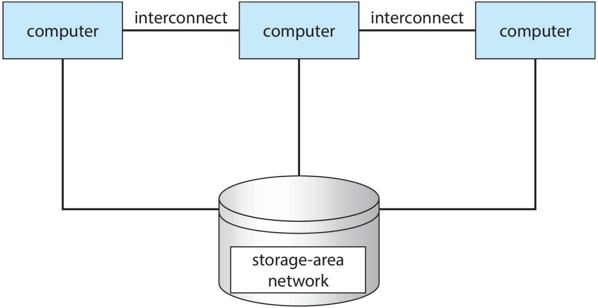
Special Computing System Environments

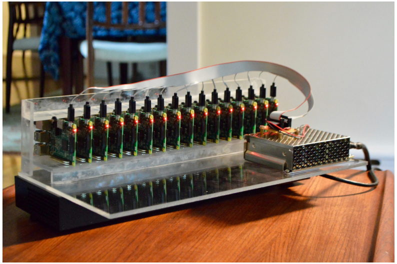
Computer cluster

Several computers linked through network that have a common purpose

Use-cases

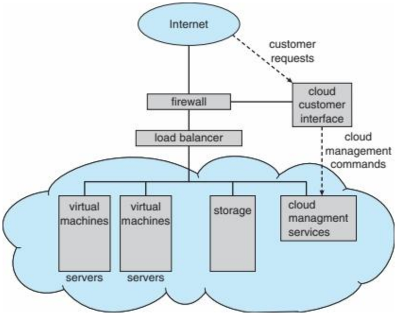
- High-availability (reliability)
- High-performance-computing (parallelism)





Source: `https://oroboro.com/compact-16-node-raspberry-pi-cluster/`

Cloud computing and virtual machines



Virtual machine

- An operation system runs inside another (different) host operation system
- Used to run otherwise incompatible programmes
- Used to “sandbox” applications (protect others from it)
- Sometimes specialized host OS is used, e.g. VMware ESX and Citrix XenServer

Cloud computing

- Services and computing is out-sourced to machines of cloud provider
- Via internet or other network
- Cloud provider runs many virtual machines (serving many customers) on each physical machines

Embedded systems and real-time systems

Embedded system

- Small device
- Often specialized functionality
- Limited resources (CPU, memory, UI,...)
- Simplified OS

Real-time system

- Responsiveness dominates other requirements
- Guarantees for worst-case response time needed. Much more important than average response time or efficient resource utilization

Source: Detroit news

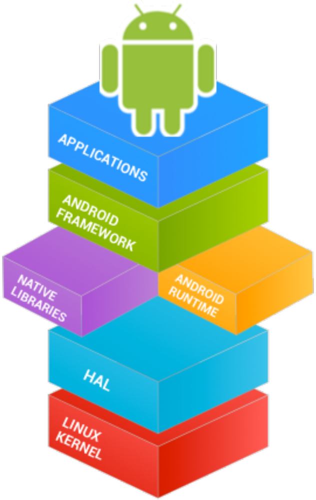
Often both coincide: car electronics, traffic lights, smart home, industrial robots ...

Mobile computing

Handheld devices

- Some overlap with embedded systems
- Limited battery makes energy saving a priority
- OS typically ships with a lot of middleware

Source: Wikipedia



Source: Wikipedia