DM510: Introduction and Overview

Lars Rohwedder

Disclaimer

These slides contain (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 system: prerequisites and grand tour

Vote for your favourite operating system

https://etc.ch/8fFX

- Microsoft Windows
- macOS
- Linux (any distribution)
- Android
- iOS
- chromeOS

Your teacher

poll: https://etc.ch/8fFX

Lars Rohwedder

Short Bio

- Since Oct. 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: https://etc.ch/8fFX

Lars Rohwedder

Short Bio

- Since Oct. 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)

Teaching Assistant

Eva Agerbo Rindom is handling the exercise sessions in both sections

Course goals

poll: https://etc.ch/8fFX

- Understand what an operation system does
- How it provides these services
- How to use an operating system's services
- How to modify/program an operating system's services

Course goals

poll: https://etc.ch/8fFX

- Understand what an operation system does
- How it provides these services
- How to use an operating system's services
- How to modify/program an operating system's services

Placement within your curriculum

- Natural continuation of DM548: Computer architecture
- Apart from DM548 probably lowest abstraction level among all CS courses

Course goals

poll: https://etc.ch/8fFX

- Understand what an operation system does
- How it provides these services
- How to use an operating system's services
- How to modify/program an operating system's services

Placement within your curriculum

- Natural continuation of DM548: Computer architecture
- Apart from DM548 probably lowest abstraction level among all CS courses

Practical skills from this course

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

Course organization

poll: https://etc.ch/8fFX

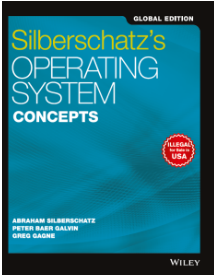
- 17 lectures between 03-02-2025 and 05-05-2025
- 12 exercise sessions to facilitate content from lecture, exercise sheet to complete before each session
- some backup dates in case of sickness, etc.
- 3 programming projects (in teams of 2):
  - from 06-02-2025 to 04-03-2025
  - from 04-03-2025 to 08-04-2025
  - from 08-04-2025 to 20-05-2025

Assessment

- 80% of grade comes from written exam during exam period
  - only content from textbook (next slide), consult slides and exercises for narrowing
  - exception: one exercise on contents of programming projects
- 20% comes from programming projects

Resources

poll: https://etc.ch/8fFX



Textbook

- Lectures based on different chapters from book
- In stock at adademic books
- Not strict requirement, but can be helpful next to the lectures

Additional resources

- https://larsrohwedder.com/teaching/dm510-25 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

Definition of Operating System

Components of an operating system

Boundaries of operating system are unclear.

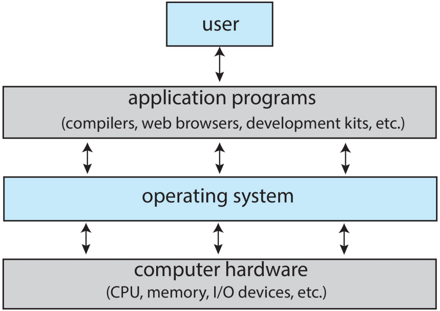
Shipped in a typical OS

- 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).
- ...

Role of operating system

User view

- Execute user programs
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner

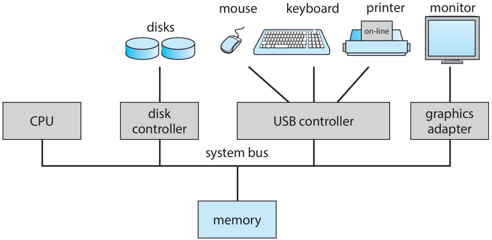


System view

- kernel program must run at all times
- User programs interact with hardware only through kernel, which acts as:
  resource allocator: decide who gets which hardware resources
  control program: prevent errors and improper use

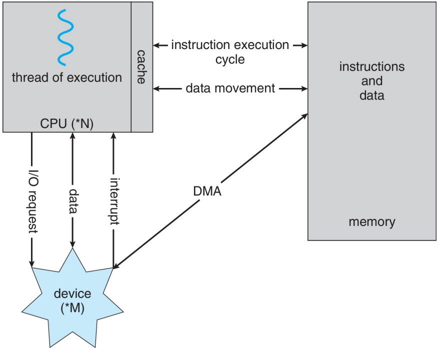
Hardware Support/Requirements

Typical components of a computer



- Various controllers (small processing units) for different devices (keyboard, GPU, network, etc.) execute concurrently
- Device controllers have local buffers of limited size

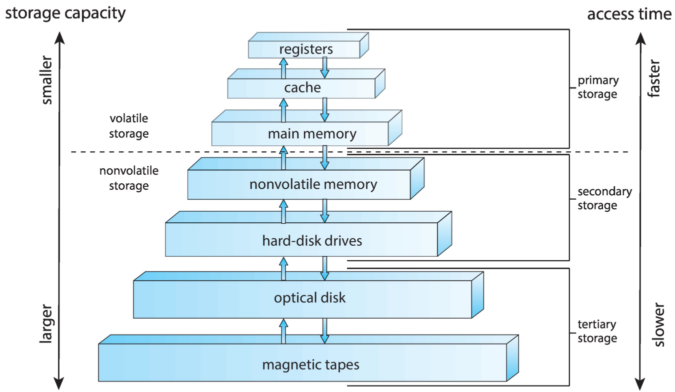
CPU architecture



Von Neumann Architecture

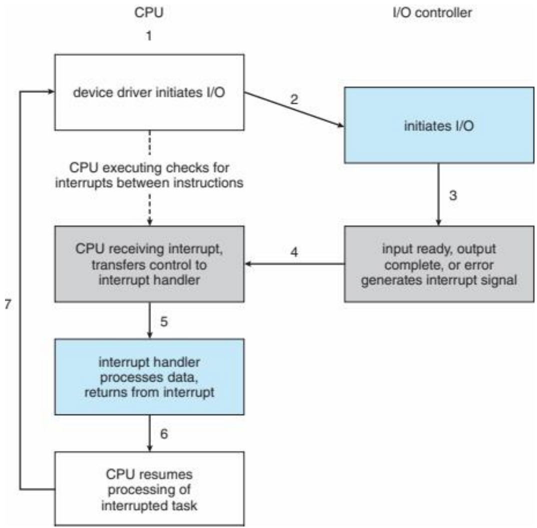
- Instructions and data are fetched from same main memory
- Interaction with devices via
  I/O request: from CPU to device
  interrupt: device notifies CPU of event
  data transfer: to and from local buffers
- For efficiency, devices can directly read and write to main memory via **direct memory access (DMA)**. CPU interaction only before and after transfer of a entire block of data

Storage



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

Interrupts



CPU has an interrupt bit that is checked at every instruction. If it is set/active, we interrupt process and execute an interrupt handler of the kernel. Afterwards we can resume interrupted process.

Interrupts (cont.)

- Interrupts can therefore transfer the control over CPU from user processes to kernel
- Transfer requires non-trivial 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 inconsistent.

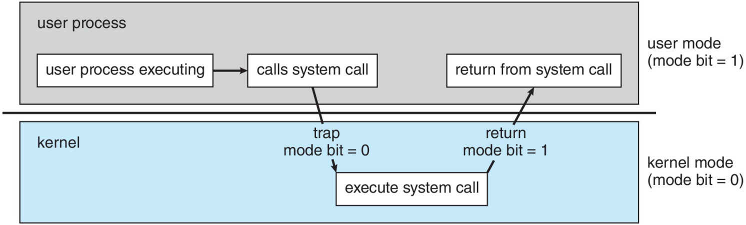
Interrupt table

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



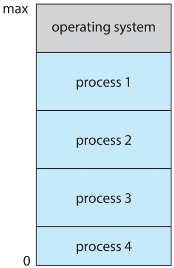
Multi-user, Multitasking, Parallelism

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



main memory

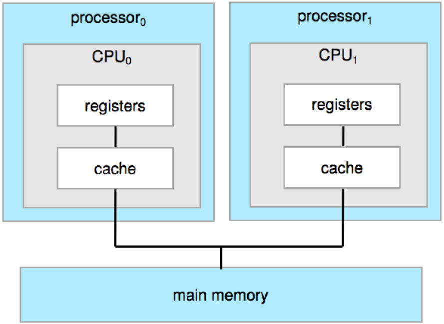
- Many processes (each with multiple threads) and users can be active at the same time
- Each one wants low response time (< 1 second) and fair share of resources
- Resource utilization should be high
- **Synchronization**: Concurrent access to shared resources/devices needs to be safe

Sharing of resources is mostly hidden from processes

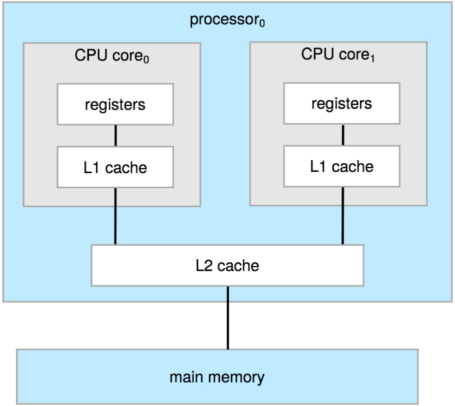
- CPU scheduler decides which process to run next
- Timers (and interrupts) to take control from process if execution too long
- Virtual memory ensures that processes do not see memory of other processes
- **Exception:** synchronization usually needs to be done explicitly

Physical parallelism

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



Multiprocessor



Multicore

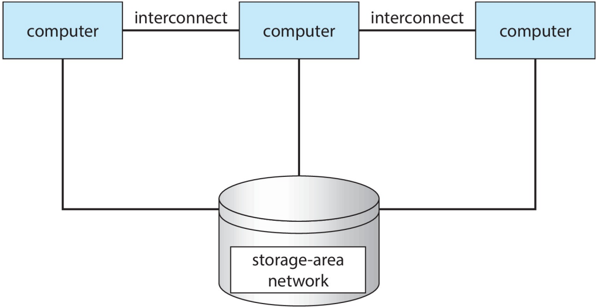
Special Computing System Environments

Computer cluster

Several computers linked through network that have a common purpose

Use-cases

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

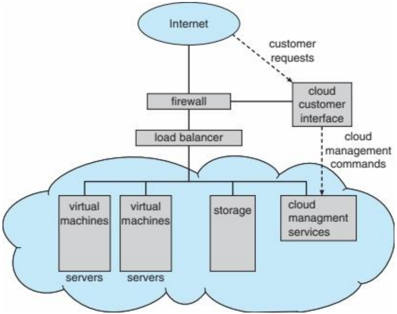


Source: Wikipedia

Cloud computing and virtual machines

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



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

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

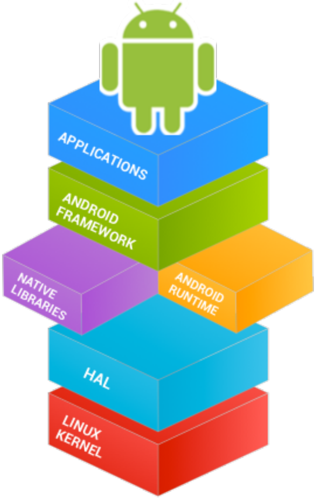
Often both coincide: car electronics, traffic lights, smart home, industrial robots . . .

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

Important Operating Systems

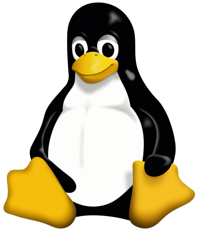
Free software

Unix/Unix-like operating systems

- implement operating system standardization for APIs (known as POSIX), command-line, “philosophy”, etc.
- Well known examples: Linux, BSD-Unix, MacOS
- Leads to some compability between systems

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
- Most famous free and open-source operating system
- Basis for mobile operating system Android



Proprietary software

Microsoft Windows

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

MacOS

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