History and classification of Operating Systems, HW environment

Tamás Kovácsházy, PhD

1st topic, Introduction

Méréstechnika és Információs Rendszerek Tanszék

Operating system

- Definition (Wikipedia)
  - An operating system (OS) is an **interface between hardware and user** which is responsible for the **management and coordination of activities** and the **sharing of the resources** of a computer, **that acts as a host for computing applications** run on the machine.
  - Operating systems **offer a number of services to application programs and users**. Applications access these services through **application programming interfaces (APIs)** or system calls.
- Is it a good definition?
  - More or less OK for client and server operating systems.
    - Practically there is no good definition: Microsoft v. USA legal proceedings
  - There are new terminologies in the definition . (they will be introduced later)
  - What kind of operating systems do exist?
    - Too many, let's start with the history of operating systems…

Early operating systems

- Evolution of Hardware defines the process
  - Early computers had “wired” programming
    - One task can be executed in a time
    - Changing task was very time consuming (rewiring)
  - The optimization of resource use by done by human operators
    - Selection of tasks and the order of execution (based on e.g. priority)
    - Human, machine and other resources are allocated to the task
    - Execution attempt
    - Evaluation of results
    - Rewiring
    - Repeat the process as long as the results are OK

Early BATCH systems

- Batch system:
  - Programms are written on paper using early programming languages (early Fortran dialects)
  - Putting the program on punch cards
  - Punch card set is submitted to the operators of the computer
  - Operators run the job
  - Results (and errors) are printed
  - Fully on-line peripheral operation
- Jobs with similar resource use are grouped together to reduce the number of repeated tasks
- On-line peripheral operations substituted by off-line operations (I/O processor is introduced), faster execution but complexity grows
- Resident monitor schedules the jobs
  - One is finished the next one started automatically by the computer

Buffered processing

- I/O processors appear on the market
  - Standard abstract interfaces
  - Logical I/O peripherals appear
- Buffering
  - To optimize the connection between I/O peripherals and the CPU
  - Input –> CPU –> Output overlap
  - Finding programming errors is even hard even in this situation
    - The programmer has no on-line access to the computer
    - Results and errors are receive only after program execution

Spooling

- RAM memory chips appear on the market
- Capacity and speed of random access memory (RAM) grow
  - More than one task executed virtually
  - One main program and I/O tasks are involved
  - All running virtually in a parallel manner
    - Spooling (Simultaneous peripherial operation on-line)
    - Tasks can be even more interleaved.
    - Results: Steps toward multiprogramming

Multiprogramming

- Even bigger capacity RAMs with higher speeds makes possible new advances
  - Tasks are not necessarily processed in FIFO manner (human scheduling more)
  - Optimization possibilities
  - Job pool
    - The aim is to reach 100% CPU, but other factors play their role
    - CPU scheduling : Which task can run? :
      - Resource utilization is an opened question (CPU, memory, permanent store, peripherials) to be solved in real-time.
      - The response of the on-line connection has mixed properties
  - Tasks run as long as it does not finished or I/O occurs.

The end of the 1960s

- Minicomputers appear (e.g. PDP)
  - Small groups (department) have access to computers
    - More people start to use computers
    - Number of humans handled by a computer decreases
    - Programmers have on-line access to computers
  - MULTICS, and UNIX later
  - C and some other modern programming languages
  - First steps toward the Internet (ARPANET, information sharing)
  - Fast development cycle
  - First attempts to control physical processes with computers
    - The first embedded systems are introduced to the market

Time sharing systems

- Time sharing or multitasking
- On-line users need short response time
  - Multiple people use the same computer (n*10-100 people share one computer)
  - The type something than wait for the output, interactive users
    - The machine should not be in idle state (Utilize it!)
    - Response time should be acceptable (n*10 ms)
  - Task run virtually in one time (parallel), in reality they get time slices of the processor
    - They run after one another, timed by a periodic timer
    - The timer interrupts the running task, and runs the next one
  - A batch system runs in the background
    - It utilizes the remaining time slices not used by interactive tasks
- Classic UNIX OS is designed around this model

Personal computers

- From the middle 1970s
  - A user gets her/his own computer due to the advances of technology
  - IBM PC
    - x86 CPU architecture
    - memory + HDD
    - Character or graphics terminal (X-windows, Windows)
    - Keyboard and later mouse
    - Soundcard
    - Network interface cards (LAN later Internet)
- Steps towards distributed systems
- New requirement: Userfriendlines

Distributed systems

- Decentralization
  - Distribution of function in space
  - Advantages and disadvantages: Security, resiliency , scalability, reliability, easy or hard developement
    - It is very hard to develop such systems (complexity)
    - We are moving towards it (cloud computing, etc.)
- No time to speak about it in this subject
- Next step is mobile systems, we will not have to speak about that also

Multiprocessor system

- Homogenous (identical) processors
  - E.g. multiple CPU cores, multiple identical CPU, multiple CPU and multiple CPU core in one physical CPU (AMD Opteron, Intel Xeon)
- Heterogeneous (different) processors
  - E.g. a multicore CPU plus GPGPU (CUDA, OpenCL) or FPGA based accelerator
- Manycore CPU-k (e.g. Intel experiments, PS3 Cell)
- How this monsters can be programmed efficiently?
  - We do not touch these issues, other subjects may later
  - It is also an active research area

What kind of OSs do exist?

- Application specific approach
  - Client, server, mainframe operating systems (IT infrastructure)
    - Multipe execution units, may distributed, Grid, Cloud, Supercomputers
  - Embedded operating systems
  - Mobile operating systems
- Capability specific approach
  - Generic operating systems
  - Real-time operating systems (bounded response time)
  - High availability operating systems (reliability, availability, redundancy)
  - Configurable operating systems (functions can be selected)
- Capability and application are quite interrelated...
  - E.g. Linux is everywhere
  - Microsoft has products in nearly all market segments
  - Large number of specialized OS manufacturers
    - Wikipedia: 45 commercial OS manufacturer,nearly all of them has multiple differentiated products, plus open source OSs
    - How many Linux distributions do we have? Are they different OSs? I do not know the correct answer, the OS kernel is the same, but the applications and configuration are different.

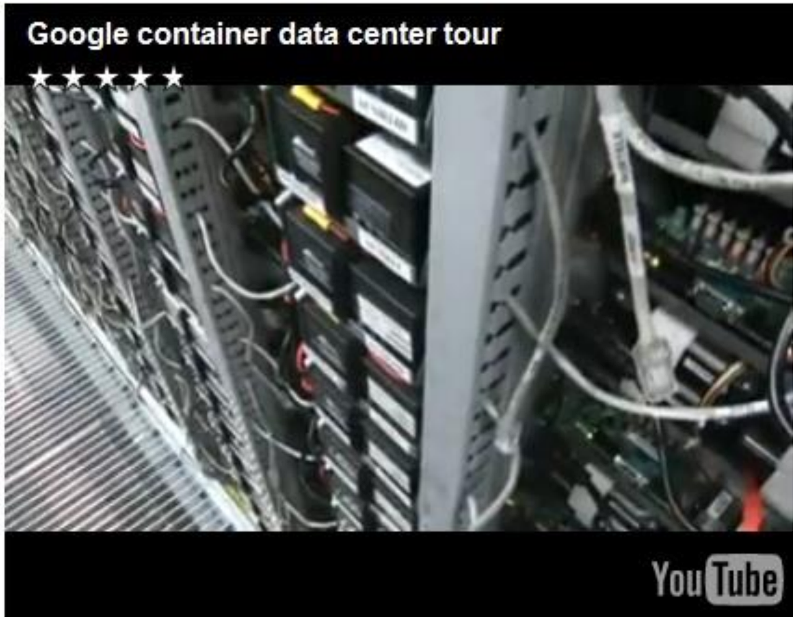
Client, server and mainframe OSs

- Client OS is clear for everybody… (or at least I hope)
- Multiprocessor server/mainframe
  - 8-64(256) CPU, n*10/100 Gbyte RAM
- High availability
  - Redundancy
  - Parts can be change while running
- Can be partitioned
- HW support for CPU, memory, and peripherals virtualization

IBM System Z10

Sun Fire X4600

Datacenters (grid, cloud, etc.)



- n*10.000 server
- n*100 TB memory
- Huge storage space
- Massively parallel tasks (WEB search)
- Task length vary drastically 20-50 msec or days
- Google, Microsoft, Facebook, YouTube

Embedded system 1.

- Embedded systems are specialized computer based systems **designed for a specific task**
  - Most cases to do this specific task they are in an intensive information exchange with the environment
    - They sense (by sensors) certain parameters of the environment and they influence the environment by actuators
    - There is a machine-environment interface: Sensors, Actuators, Communication interface
    - User interface for human operators is also present
- PCs can be used in embedded systems
  - It must have a dedicated task, it is not HW or SW specific property, but application specific
  - We may also use Windows or Linux in embedded systems!
    - They are not designed for that…
    - In non-demanding applications they may be a cheap and easy solution.

Embedded system 2.

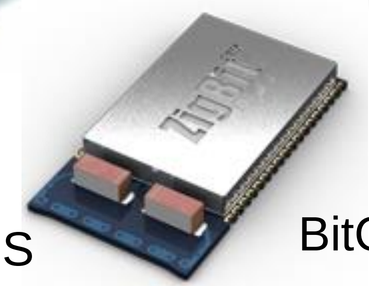
- Many embedded systems operate in a safety critical environment!
- In case of a system failure:
  - People may be injured or die
  - Wide scale property damage is possible
  - Non acceptable risk
    - We need to avoid it, design and implement the system very carefully
    - There is no 100% safety or security

Embedded applications 1.

Special certificates may be required:

- Road vehicles
- Railways
- Aerospace applications
- Military
- Health
- Industry and energy sector
- Etc.

- Real-time operation
- Reliability, security, availability
- No single OS can do all in one product



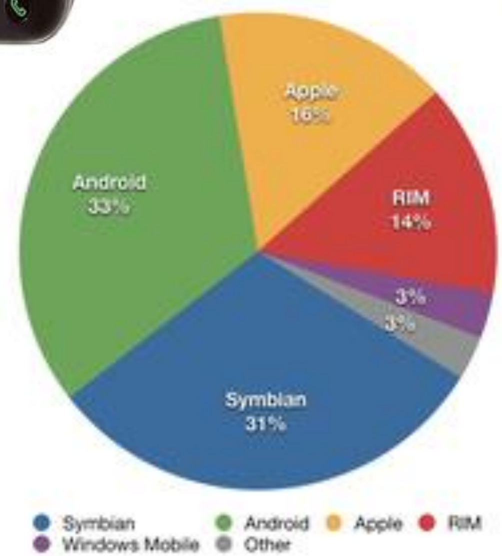
uC/OS

BitCloud

Embedded applications 2., Mobile OS

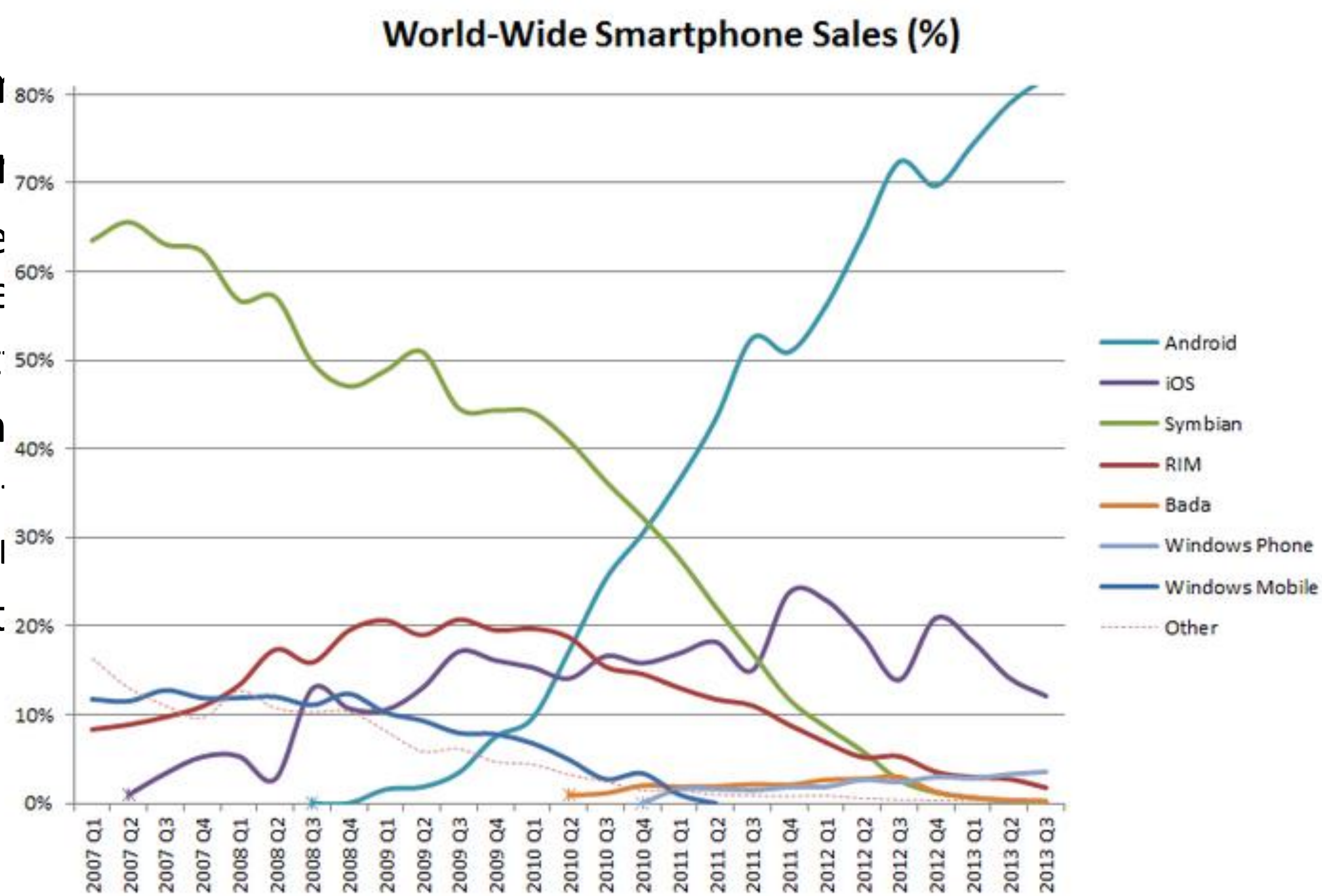
- Mobile embedded systems
- It is blurred with client operating systems
- Requirements
  - Special GUI, multitouch, sensor integration, etc.
  - Battery optimization
  - Limited resources
  - Partially real-time applications (communication related)
  - Heterogeneous HW architecture
    - User CPU
    - Communication DSP
    - Graphics accelerator
    - Multimedia coder/decoder

Smartphone market share 2010 Q3
Forrás: Canalys



Embedded applications 2., Mobile OS

- Mobile embedded systems
- It is bl
  systen
- Requi
  - Spe
    inte
  - Bat
  - Lim
  - Par
    (co
  - Het
    -
    -
    -
    -



Real-time systems

- A real-time systems reacts to outside events reaching the system in a given, application specific time, if this deadline is not met, the answer can be considered erroneous
  - E.g. Quiz in TV
- Real-time system types:
  - **Soft real-time: Deadline is met with a probability < 1, but the probability is > N.**
    - There are no catastrophic consequences, but …
    - The system may be late sometimes
    - Service Level Agreement
    - Not necessarily priority based!
  - **Hard real-time: Deadline is met with a probability = 1**
    - **If it does not meet the deadline, the system fails...**
    - There are catastrophic consequences not meeting the deadline
    - „**The system cannot be late**!”
- How we prove it?

Real-time system 2.

- The definition does not say about the length of the deadline!
  - The deadline is application specific
    - Think about a slow chemical/biological process, such as fermentation (n*60s deadline) or a car ESP or ABS (i.e. ms deadline)
- Real-time operating systems:
  - By architectural design it can execute certain functions of it in real-time (with strict deadlines).
    - For example, the interrupt latency has un upper bound
    - The applications must be designed for real-time operation, the real-time OS is a requirement, but not a guarantee for real-time operation of the whole system
    - Linux and Windows are not real-time
      - There are real-time extensions for them(RTLinux, Windows: eg. Ardence RTX)

HW architectures

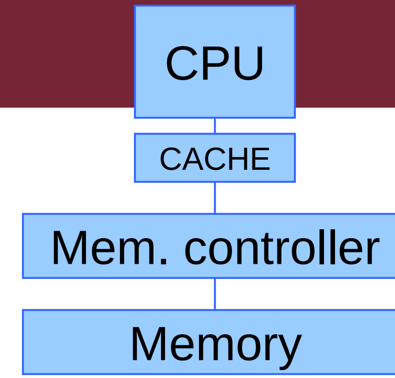
- There are lot of them, even for the x86 PC architecture…
- The operating system hides the differences:
  - A well-written application can run on a P3 PC (from approx. 2000), and on the newest multicore Core i7 PC
    - In the worst case it does not utilize more than one core.
  - Furthermore, after recompiling, it can run on ARM hardware on the same operating system.
    - If there is some HW specific in it, that must be changed, for example, some inline assembly code
  - However, we must know what happens inside the operating system and hardware

Méréstechnika és Információs Rendszerek Tanszék

Computer architectures

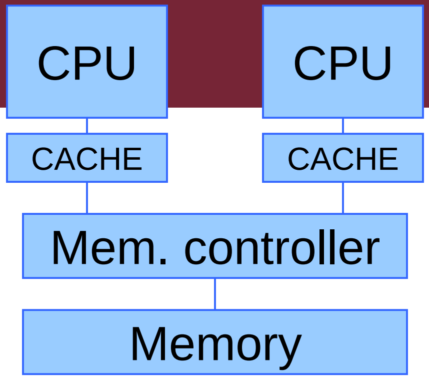
- The internal operation of the OS depends on:
  - The number and connection of processors in the system
  - The organization of memory in the system
  - The organization of other hardware in the system
    - How peripherals are connected?
- We will address only homogeneous multiprocessor system
  - There are identical processors in the system
- Types of systems we will talk about
  - Single CPU (Uniprocessor)
  - Symmetric multiprocessing (SMP)
  - Non-Uniform Memory Access (NUMA)

Single CPU (Uniprocessor)



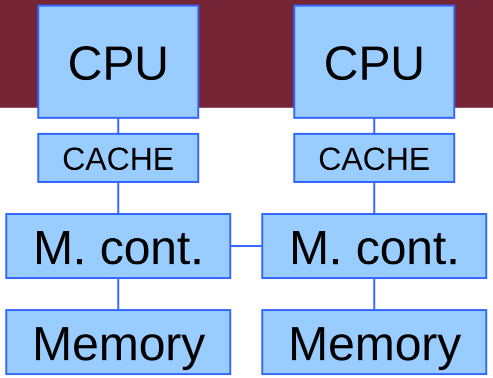
- Single CPU
  - It was the typical architecture for long time
  - In embedded systems it is the typical even now!
    - Microcontrollers (MCU) are single CPU typically today
    - They can reach higher performance by architectural changes and higher clocks
- DMA (Direct Memory Access) handles memory in parallel with the CPU!
  - Race condition between the DMA controller and the CPU
    - Input: DMA transfer ⇨ IT ⇨ CPU handles data
    - Output: DMA transfer, Peripheral handles data, IT, CPU removes data from memory
  - CACHE coherency problems may arise! Solutions:
    - The whole CACHE is invalidate if DMA transfer occur
      - Simple, but has catastrophic effects on performance
    - Memory locations handled by DMA not cached (CACHE controller, MMU)
    - CACHE coherent DMA (HW support required)

SMP



- Symmetric multiprocessing
  - Multiple, identical CPUs
    - Multiple CPUs or multiple CPU cores
    - Example: AMD Phenom, Intel C2D/C2Q, iX
  - Most cases with an architecture specific CACHE hierarchy is present
    - SMP is CACHE coherent most cases
  - Memory is connected to a controller
    - The whole memory is accessed with identical properties (bandwidth, latency, etc.) by all CPUs or cores
  - Muticore MCU
    - ARM15, ARM11 MPCore, ARM Cortex-A9 MPCore
    - The CPU core is cheap (small portion of the chip surface)
    - More and more SMP appears in embedded systems
  - To utilize multiple CPUs in the OS, the OS must support the SMP HW properly
    - Otherwise only one CPU is seen by the OS

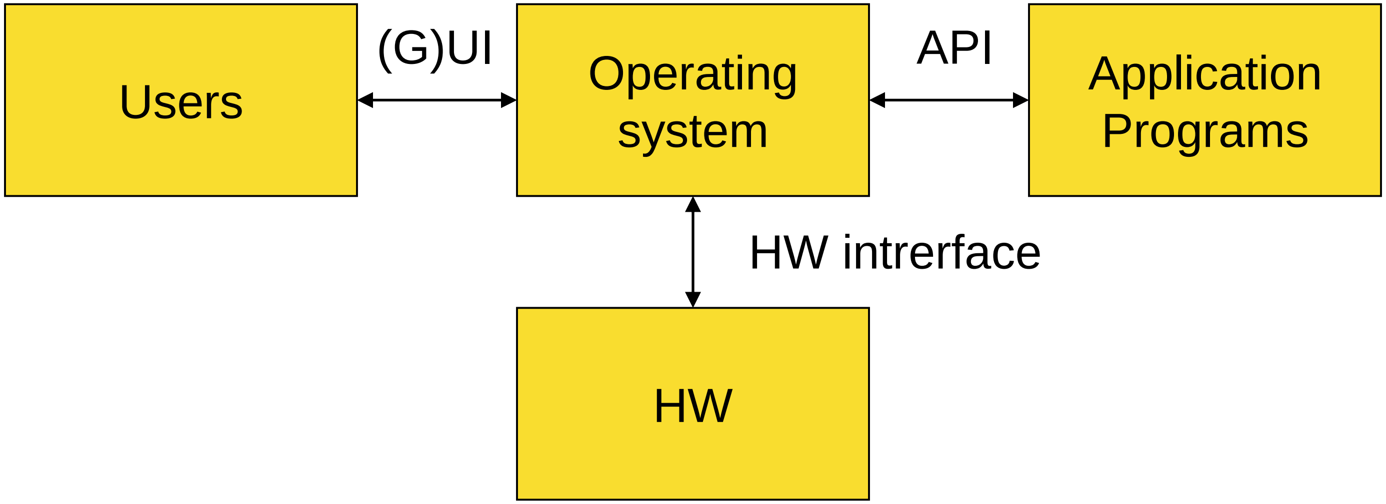
NUMA



Non-Uniform Memory Access

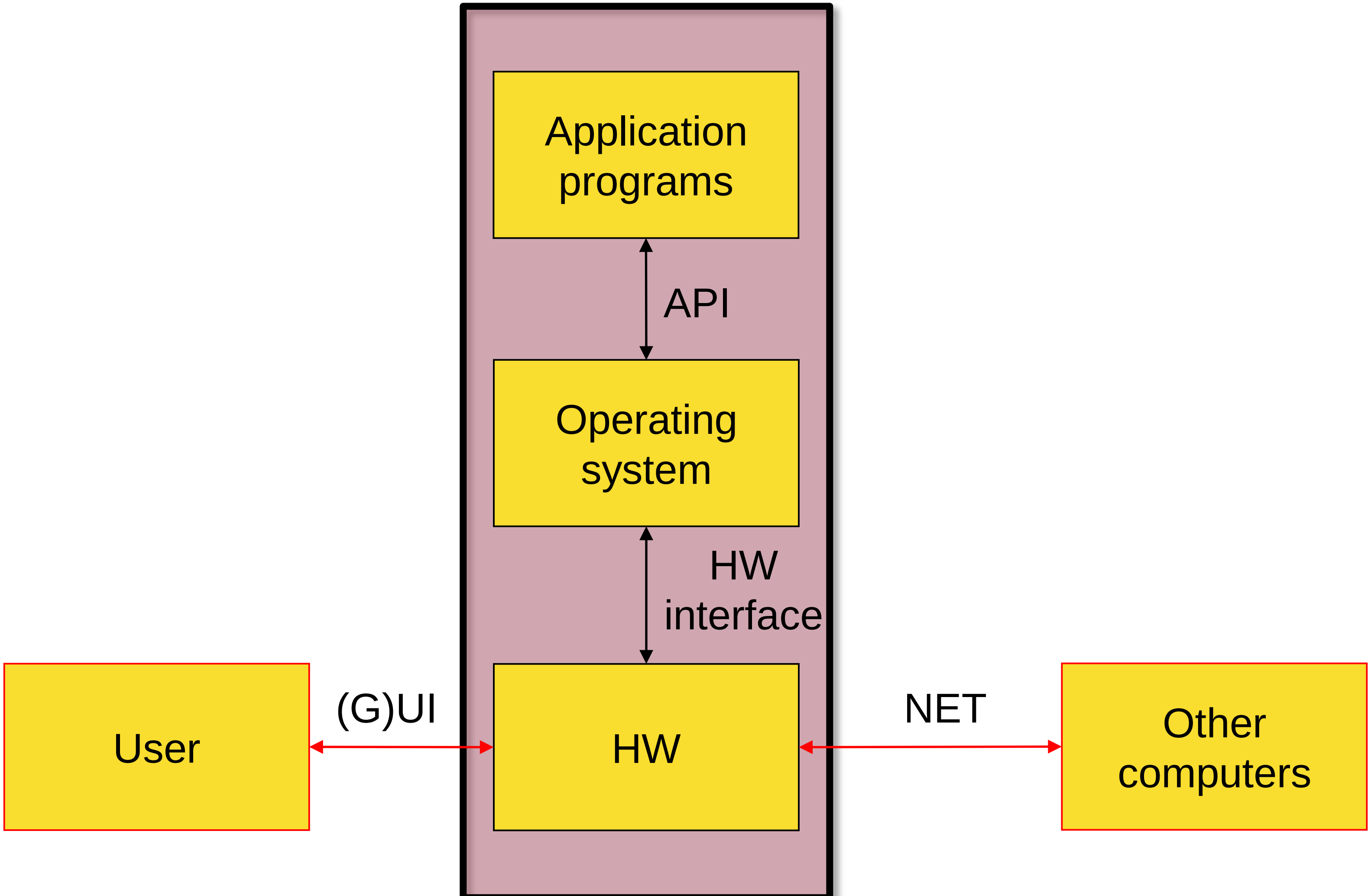
- Memory speed depends on the relative location of CPU core and memory location
- The physical memory is unified
  - Single and identical address space for all CPUs
- Cache coherency
  - CACHE coherent (ccNUMA)
  - No CACHE coherent
- Memory controllers are connected by a special communication interface
  - QPI for Intel, Hypertransport for AMD
- For example, multiple CPU AMD Opteron or Intel Core i7 based Xeon CPUs utilze ccNUMA architecture
  - Inside one chip the architecture SMP
- To utilize multiple CPUs in the OS, the OS must support the SMP HW properly
  - Otherwise only one CPU is seen by the OS

The operating system and its environment



- It must be emphasized that:
  - The user and the application programs cannot be in direct contact
  - HW and the application programs cannot be in direct contact
- Everything happens through the OS
  - For performance reasons there are some exceptions (Graphics), but OS control is there even in this case
  - There are some exceptions in embedded operating systems also

The operating system and its environment 2.



Layered structure of operating systems

- Layered structure
  - Structure (design time efficiency) and run-time efficiency must be balanced
    - Layered approach is a must for extendibility also
  - A virtual machine is realized by the layers (upper interface)
    - High level function, virtual instruction set
  - On the lowest layer the real CPU and peripherals implement the real machine

Typical layers in OSs

- Layers
  - Kernel (implements the fundamental functionalities of the operating system)
    - Task and memory handling, security
  - Hardware specific layer (HAL and drivers, typically in a well-defined hierarchy)
    - HAL (Hardware Abstraction Layer) is a bridge between HW and the kernel
    - Handling of keyboard, mouse, graphics, sound, storage, network, etc.
  - System programs (subsystems to implement other functionalities of the OS)
    - Filesystem, high level network handling (TCP/IP), command shell, stb.
  - System call handler managing system calls coming from application programs
    - An API on various target languages compiled into the applications
    - This API maps API calls to system call
    - Changes privilege level to the kernel, and system call handler is executed there

Operating system architectures

- The OS is a complex software, its has its own architecture
- Monolithic kernel
  - All functionalities are compiled into a large executable (the OS)
  - Inflexible, any changes in HW or functionality can be added by recompiling the kernel
  - A failure of one component results the failure of the whole kernel
  - It is common in embedded systems (The HW does not change there)
- Modular kernel
  - Minimalistic kernel extended (or cut back) by loadable modules run-time
  - Flexible, can be adapted to changes in the HW or requirements
  - A failure of one component results the failure of the whole kernel
  - Linux kernel since version 2.x, Windows
- Microkernel
  - A kernel with minimalistic functions, services and drivers are attached to it using the client-server architecture
    - It uses 3 privilege levels (kernel, services and drivers, applications)
  - Needs more resources due to complex communication among components
  - Failure of a service, driver, or application program cannot influence the operation of the kernel
  - Mac OS X

Examples

- Linux
  - The basic architecture is a monolithic small kernel and loadable kernel modules
    - Modul handling commands: modprobe, insmod, lsmod, rmmod
  - Monolithic kernel can be built also
    - All services and drivers are compiled into the kernel
      - Module handling can be left out of the kernel
    - Inflexible, but in some applications flexibility is not required
      - No hardware or service changes required
      - Has some advantages in embedded systems (small size, faster)
- Apple OS X
  - Darwin
    - Mach 3.0 mikrokernel + FreeBSD (Berkeley Software Distribution) UNIX
    - Object-oriented framework

Accessing hardware

- Special CPU registers (CPU config)
- I/O ports accessed by I/O instructions
- Memory mapped I/O
  - Available bandwidth and latency can be drastically different compared to real memory
- DMA (Direct Memory Access)
  - DMA controller (HW specific), needs to be programmed
  - Can move information between peripherals and memory without the CPU
  - Faster but has some disadvantages (race condition in the CACHE)
- Interrupt
  - Interrupt controller (HW specific)
  - Can be disabled and enable
  - If the interrupt is enabled and the interrupt signal comes in, the CPU transfers the execution to the interrupt handler code
  - The details are HW specific

CPU privilege levels

- Hierarchic CPU privilege levels
  - First introduced in the middle of the 1960s (to support Multics and later UNIX)
  - The CPU needs to change in between them, it is done by executing a system call
- Nearly all modern generic processors support this functionality
  - Controls the access to CPU resources
    - Executable instructions
    - Access to CPU configuration registers
    - The possibility of I/O instruction execution is disabled
    - Access to memory locations may be restricted
  - E.g. x86 since 286/386, ARM Cortex Ax line, etc.
  - Microcontrollers do not support or have very limited support of privilege levels, e.g., Atmel AVR, ARM Cortex Mx line, stb.
- 2 or 4 privilege levels are implemented in modern processors
  - Typically 2 is used
    - User mode (real mode) –restricted access
    - Kernel mode (protected mode) – full access
    - In case of a microkernel a 3rd level may be also used for driver and services
      - It has limited access, more the in user mode, but less than in kernel mode

Memory Management Unit (MMU)

- Special HW in the CPU
- The functions of the MMU (all will be detailed later)
  - Maintaining the state of the memory
    - ID of the task that uses the memory
    - Access Control List (ACL)
    - CACHE control (e.g. DMA)
  - Mapping virtual memory to physical memory
    - Speeding up mapping (Translation lookaside buffer, TLB)
    - MMU state have some part which are task/context dependent
    - Pagefile or SWAP (HDD)
  - Protection of memory
    - Prohibited access to memory must be denied or at list signaled to the CPU
    - General Protection Fault (GPF) in older version of Windows
- We will talk about the MMU and memory handling later
- Linux, Windows, Windows CE (Windows Phone) requires a functional MMU for running these OSs

Interrupt

- Interrupt (IT) types
  - Hardware IT: a peripheral request handling from the CPU
    - External event influences the CPU
    - A peripheral may request services for multiple reasons
  - Exception: The CPU or the MMU has identified an event which needs specific software to run on the CPU
    - E.g., page fault, numeric overflow, divide by zero, privilege violation, etc.
    - A hardware signaled even, which comes from the CPU
  - Software IT: system call by executing a special instruction
    - A software influences the operation of the CPU through the IT mechanism
- Modern operating systems are interrupt driven.

Hardware interrupt example

- An external hardware requires immediate service
- Clock interrupt (exceptionally important)
  - ML1 (Who can remember it? The exchange students cannot.)
    - Fix frequency oscillator producing impulses
    - Programmable counter
    - After a predefined number of impulses its request a HW interrupt
  - This interrupt periodically runs the OS (scheduler)
  - The system clock is also derived from this source
  - Periodic or oneshot operation
    - The period is 1-20ms, typically 10 ms

System call

- What a "system call" is?
  - Starting point: The CPU runs an application program
  - The system call by the application program interrupts the CPU by a software interrupt, transfers the execution to the OS
  - A context switch happens
  - The OS does it work
  - Transfers the execution back to an application program (with a context switch)
- How a "system call" is executed?
  - Implementation specific...
- Consequences of the system call
  - It has a large overhead, consumes CPU time
    - The context must be save and restored 2 times
  - The number of system calls must be minimized
  - During the system call the CPU changes privilege levels 2 times

What the context is?

- CPU registers
  - PC, Status, Work, Segment, etc. registers
  - E.g. Linux does not save floating point registers while entering kernel mode
    - Consequence: Floating point arithmetics cannot be used in the kernel!
- MMU settings
  - Access to the memory of the running application program must be guaranteed
- Other application specific HW settings
- The most delicate task of is to determine what is needed to be saved and what is not necessary to be saved.

System call for programmers

- The programmer calls an API call from the application program
  - For example in case of C language (most OSs are written in C or C++), the API is:
    - Windows: windows.h
    - Linux: glibc
  - The API hides the details of the systems call from the programmer
    - It is a high level C wrapper around the system calls
  - The API implementation is compiled into the application (or bound to it run-time), runs in user mode, and executes the system call
  - The OS consists the system call handler, and the system call transfer the execution to it in kernel mode
  - There will be examples later…

I/O instructions

- Application programs cannot execute I/O instructions themselves (user mode)
- They initiate the execution of I/O operations by issuing system calls
- By issuing a system call the application program waits for finishing the system call
- Other programs may run during this (efficiency)
- The kernel executes the real low-level I/O instructions in kernel mode
- The peripheral signals the HW the by interrupt when the I/O is ready
- Due to the interrupt the system transfers to the OS, and the OS makes the decision what to do (e.g. it may run the application program waiting for the I/O)
- After the I/O the line of execution returns to the application program

Startup of an OS 1.

- Bootstrap process
- PC and servers
  - Init/RESET vector (CPU)
  - BIOS/EFI (firmware)
    - POST (Power on self test)
    - Search for HW and HW initialization
    - Determination of Boot the boot media
  - BOOT sector (HDD type storage)
  - 2nd level boot loader (GRUB, LILO, NTLDR)
  - OS is loaded into memory
    - HW reprogramming (device driver replaces the BIOS/EFI)
    - Privilege level change (transfer to kernel mode)
    - Initialization of other functions in kernel mode

MŰEGYETEM 1782

Startup of OS 2.

- Bootstrap process
  - Embedded system (PC does it on BIOS/EFI level)
    - OS image in ROM (ROM or flash, maybe compressed)
    - Can be run from ROM (Harvard architecture can do it only)
    - Can be copied to RAM (after decompression) and executed from there
- Stopping the machine (power off or hibernate, not standby, that is another issue)
  - Safe stopping must be done
    - Controlled saving of state during the process
    - Stopping or sleeping HW (low power mode)
  - Non-volatile storage must be left in a consistent state (HDD)