DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

COMPUTER ARCHITECTURES

Protection

Budapest, 2025. 03. 26.

Prepared by: **Gábor Horváth**, ghorvath@hit.bme.hu

MŰEGYETEM 1782

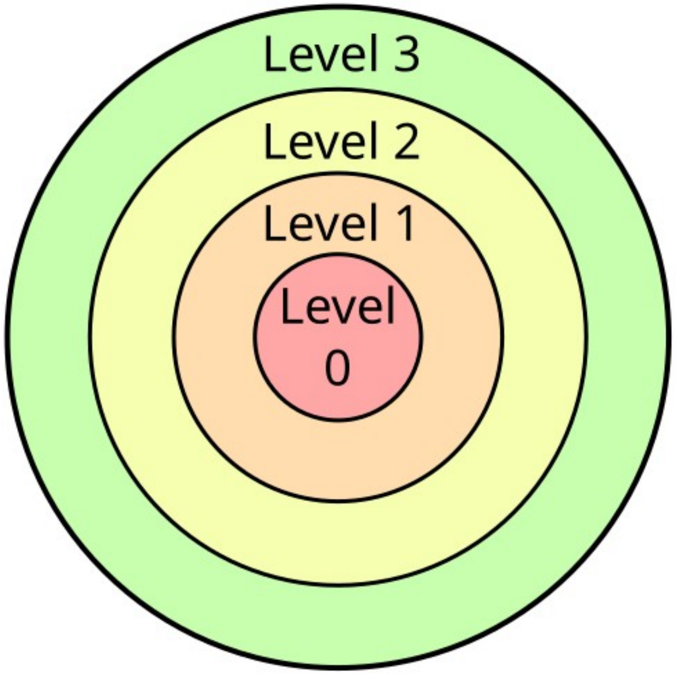
PURPOSE OF PROTECTION

- Objective:
  - **Ensuring the operation and integrity of the system**
- Even in the presence of faulty or malicious
  - Tasks
  - Hardware

  in the system!
- Protection = Restriction
  - Restricting Tasks:
    - Must not access data of other tasks or the operating system
    - Must not call private functions of other tasks or the operating system
    - Must not communicate with I/O devices bypassing the OS
  - Restricting Hardware:
    - Must not corrupt memory using DMA

PRIVILEGE LEVELS

- Privilege level = The amount of permissions a task has
  - =0: The highest possible privilege
  - The higher the number, the fewer the privileges
  - At least 2 levels are required (e.g., x86 has 4 levels)
- What does it affect?
  - The instructions a task can use
  - The memory areas a task can access
  - The functions a task can call
  - The I/O devices a task can use

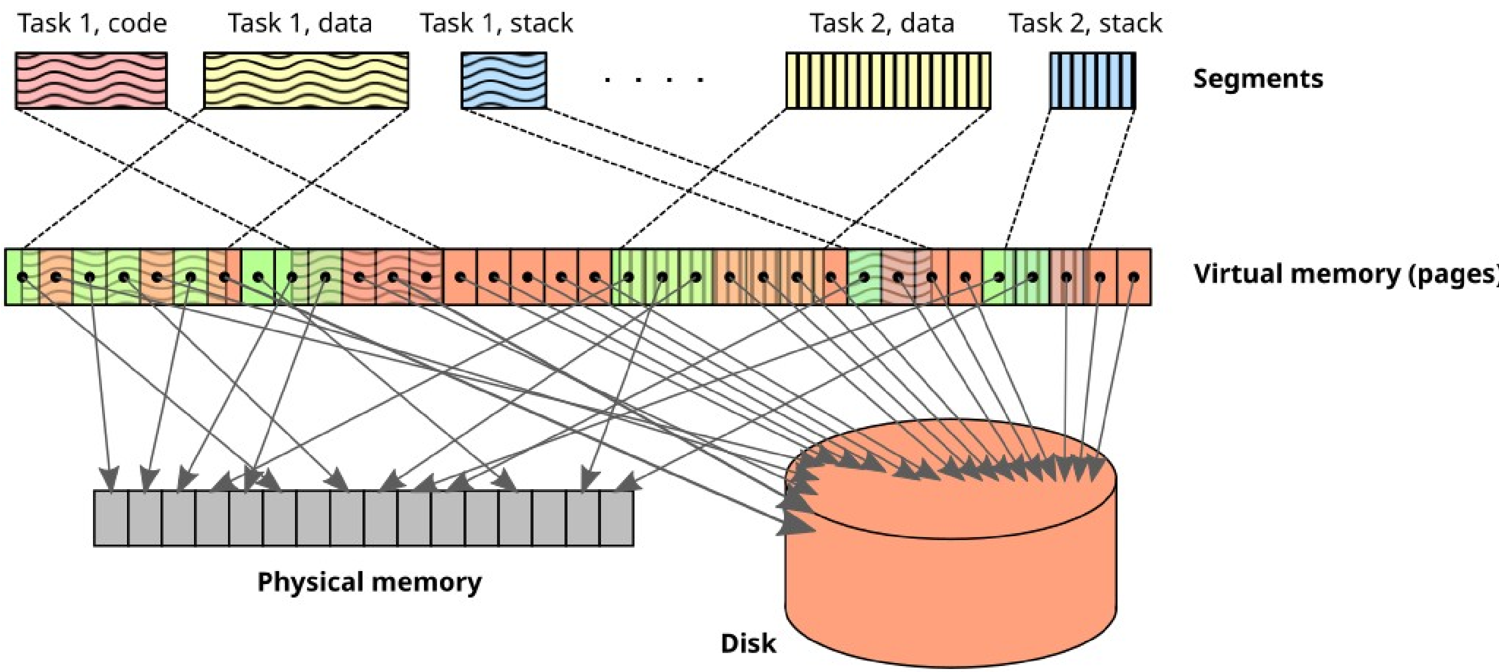


Memory protection

MEMORY PROTECTION WITH SEGMENTATION

- What is segmentation?
  - Logically grouped objects
  - In virtual memory:
    - Can start anywhere
    - Are continuous
    - Have variable lengths
- **Two-step address translation:**
  - Segment → Virtual memory (segment descriptor table)
  - Virtual memory → Physical memory (page table)

MEMORY PROTECTION WITH SEGMENTATION (CONTINUED)



- Within a segment: near addressing/jumping
- Across segments: far addressing/jumping
- x86 / 32-bit mode: 3 segments per task:
  - Code
  - Data
  - Stack
  - Not used in Windows/Linux
- x86 / 64-bit mode:
  - Practically eliminated

MEMORY PROTECTION WITH SEGMENTATION (CONTINUED)

- Fundamentals of Memory Protection:
  Address space separation

- Each task and the OS have separate segments
- Privilege levels:
  - Every task has one
  - Every segment has one (stored in the segment descriptor)
- Memory access protection rules:
  - Compare task PL ↔ segment PL
  - A task can only read/write equal or less privileged segments
  - Higher privileged segments are inaccessible
- If a rule is violated:
  - The OS takes control
  - The task is terminated

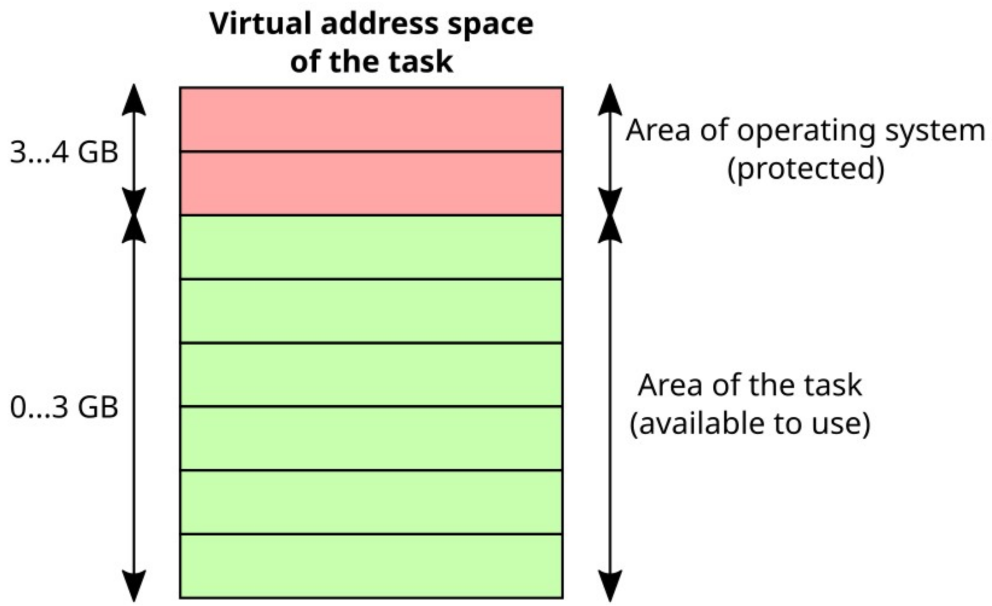
MEMORY PROTECTION WITH PAGING ONLY

- Task switching → Page table switching
- Each task gets a separate address space
  → Cannot access other tasks' memory!

- Issue: Some routines must always be available
  - Interrupt handlers for I/O devices
  - OS routines (system calls)
- Solution: Split the address space
  - Lower part: Task's space (switched on task switch)
  - Upper part: OS's space (constant)
- Issue: The protected area is visible to the task
  - How to prevent unauthorized access?
- Solution: New bit in page table entries:
  - **User/Supervisor bit**
    - =1: Only OS (privilege level 0) can access
    - =0: Accessible by anyone

MEMORY PROTECTION WITH PAGING ONLY

- Splitting the virtual address space:



- Windows 32 bit: 2 GB – 2 GB (optionally 3 GB – 1 GB)
- Windows 64 bit: user gets 8 TB
- Linux: kernel compilation parameter

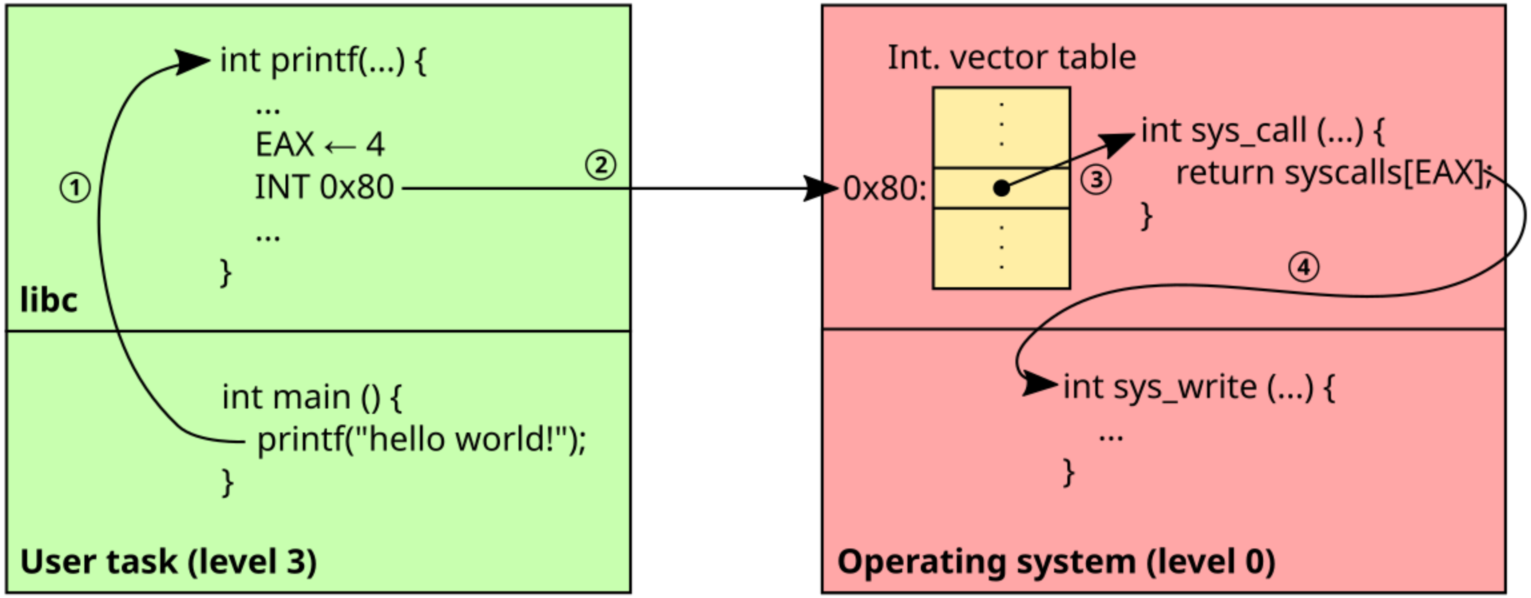
Control flow protection

SOFTWARE INTERRUPT-BASED CONTROL TRANSFER

- How does the OS handle I/O device events?
  - With **interrupts**
  - Triggered by hardware via a CPU pin
- How does the OS handle task requests?
  - With **interrupts**
  - Triggered by software via an instruction
- Interrupt Vector Table:
  - Stores the entry addresses of interrupt handlers
  - x86: 256 entries, ARM: 8 entries
  - All interrupts can be triggered by software!
  - x86: **`INT`** instruction, ARM: **`SWI`** instruction
- SW Interrupt vs. Function Call:
  - An interrupt can switch the CPU to PL=0!
- Idea:
  - Reserve some interrupts for OS calls!
  - The task triggers an interrupt to request an OS function
  - The interrupt switches PL=0 and accesses the OS function

SOFTWARE INTERRUPT-BASED CONTROL TRANSFER

- In practice:



- Old method:
  - Windows: Interrupt 0x2E, Linux: Interrupt 0x80
- Modern method:
  - sysenter/sysexit (same, but faster)

Protection and I/O devices

ACCESSING I/O DEVICES

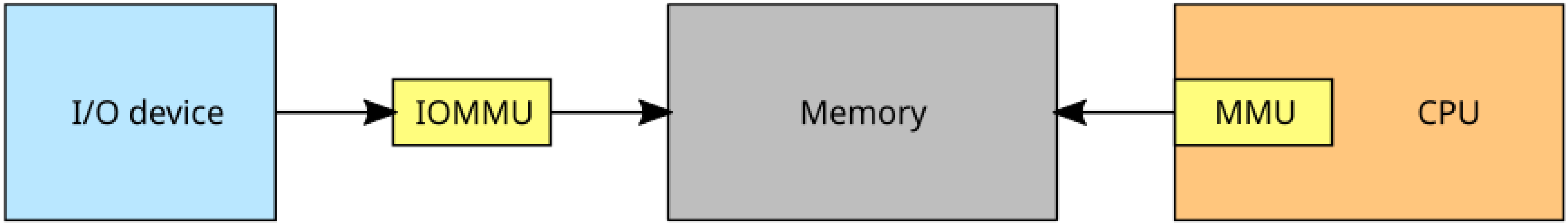
- Access to I/O devices must be restricted
  - A misconfigured I/O operation could corrupt memory via DMA
- For memory-mapped I/O handling:
  - OS sets the supervisor bit on that memory area
- For separate I/O instructions:
  - I/O operations can be made privileged

    → Harsh restriction: Task cannot issue I/O instructions
- Whitelist available I/O addresses

    → Fine control: Per-address access rules

RESTRICTING I/O DEVICES

- Protecting Memory from Tasks
  - Managed by the MMU
  - Page table entries store protection information
  - If a memory frame is not mapped, the task cannot access it
- Protecting Memory from I/O devices
  - By default: No protection!
  - Advanced systems use an **IOMMU**
  - I/O devices operate in virtual address space
  - Pages can have protection attributes



DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES

MŰEGYETEM 1782