# Computer Labs: The Minix 3 Operating System

## 2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

September 30, 2011

# What is Minix 3

- Minix 3 is an operating system developed by Andrew Tanenbaum and its students at the Vrije Univ. of Amsterdam
  - Version 1 dates from the mid/late 1980's
  - Version 2 dates from the mid/late 1990's
  - Version 3 dates from the mid/late 2000's
- Linus Torvalds developed the first Linux kernel based on the first version of Minix.
  - Linux is now 20 years, and bears no resemblence to its ancestor

# What is an Operating System?

- An OS is a program that:
  - Manages the resources in a computer system;
  - Abstracts these resources, offering an interface that is more convenient to use.

# What is an OS?

- Actually an OS is not really a program. It comprises

  **Kernel** Which implements the OS services

  **Library** Which provides an API so that programs can use the OS services

  **Utilities** A set of “basic” programs, that allows a “user” to use the OS services.

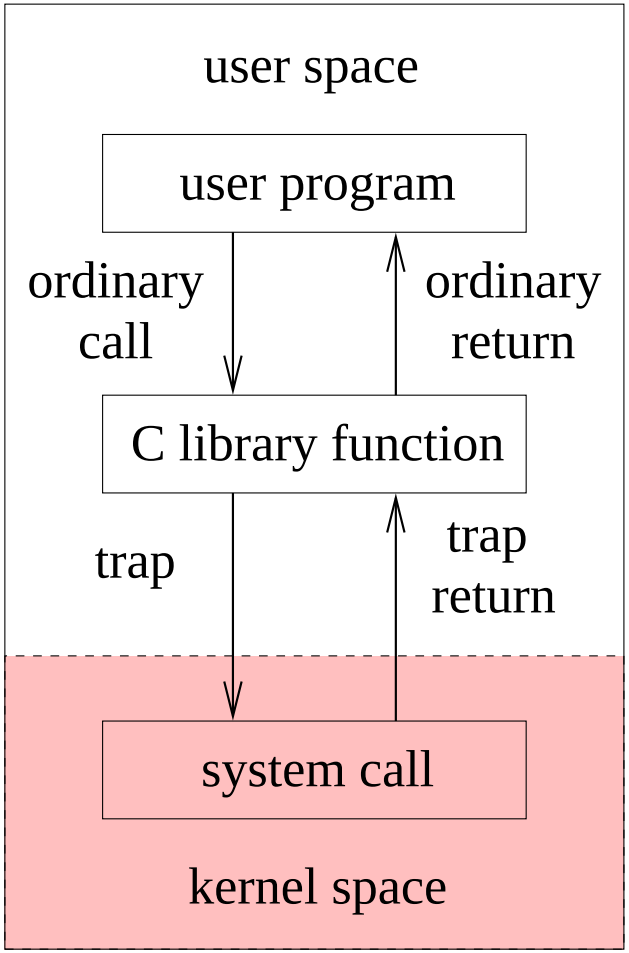
# Access to the Kernel Services

- The kernel is linked to an application almost like a library
- However, modern computer architectures provide mechanisms to ensure a separation between the applications and the kernel.
  - Most OS support multiple processes
    - Many of them associated to different users
  - Applications should not be allowed to access directly to kernel code and data
- These mechanisms are usually:
  - At least two privilege execution modes
    - Priviliged (kernel) vs. non-privileged (user)
    - Access to the computer resources depend on the current execution mode
  - A mechanism to change in a controlled way between these two execution modes

# Kernel-level vs. User-level space

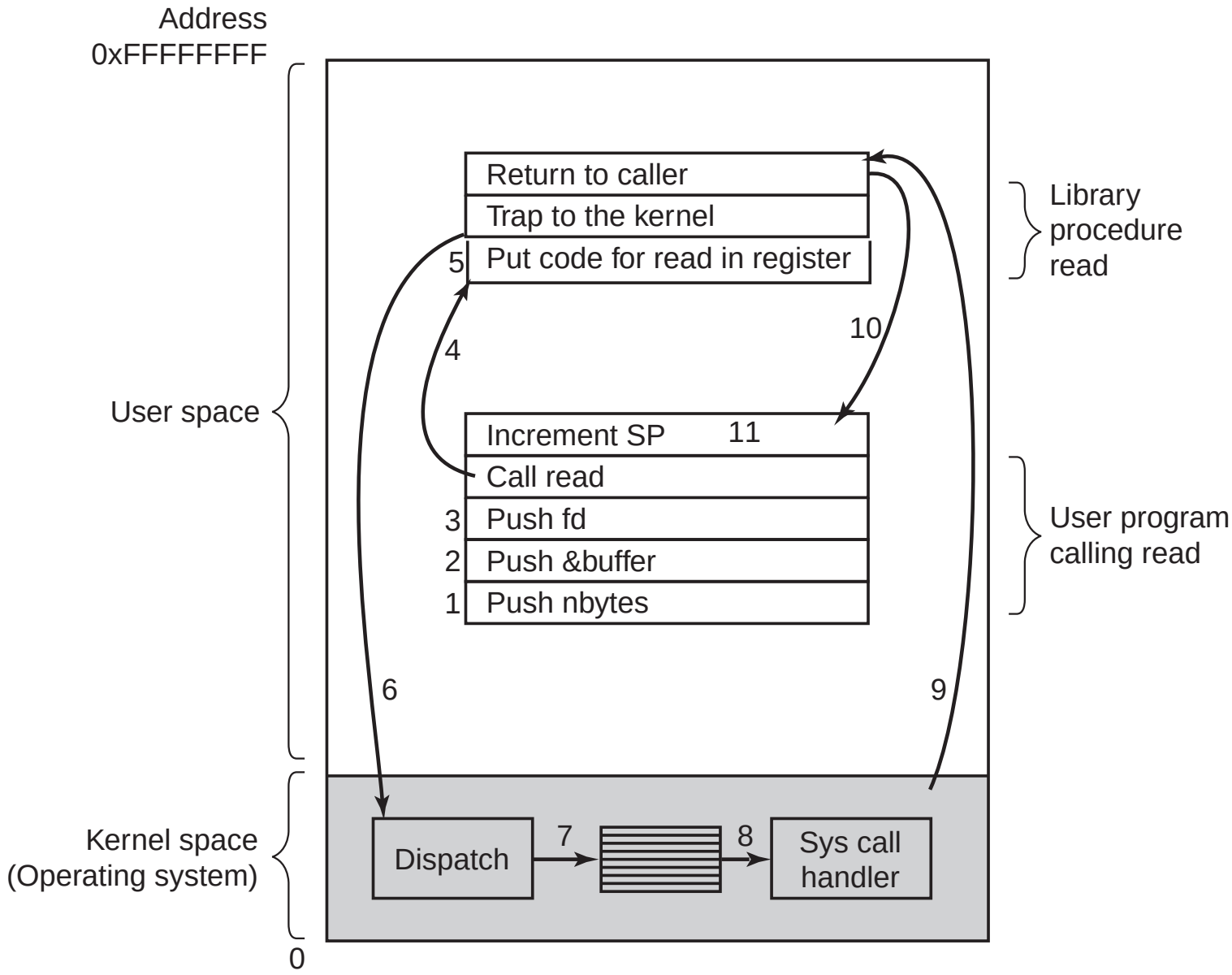
- This allows a process address space to be partitioned in user-level and kernel-level spaces
  - The kernel level address space can be accessed only when the processor executes in privileged mode
  - The kernel level address space is shared among all processes
- To allow for the access to kernel services, modern architectures provide special instructions to:
  - Switch to privileged execution mode;
  - Transfer execution control (jump) to specific locations in the kernel address space
- An example is the software interrupt instruction `INT` of the IA-32 architecture.
  - Many OSs use that instruction to implement **system calls**
  - The **system calls** are the OS API

# Implementação das Chamadas ao Sistema




- Usa instruções especiais oferecidas pelo HW (*call gates* ou *sw interrupts*, no caso da arquitectura IA32), que comutam automáticamente de nível de privilégio.
- Para o programador, é como se invocasse uma função da biblioteca de C.

# ssize_t read(int fd, void *buf, size_t count)



# Passos na Execução de `read()`

1, 2, 3 `push` dos argumentos para a *stack*;

4 chamada da função `read` da biblioteca C;

5 inicialização do registo com o # da chamada ao sistema;

6 mudança de modo de execução do CPU;

7 despacho para o *handler* apropriado;

8 execução do *handler*;

9 **possível** retorno para a função da biblioteca C;

10 retorno da função `read` da biblioteca C;

11 ajuste da *stack*.

# How is an OS/Kernel implemented?

**Monolithic** The whole kernel executes in a single address space

- Usually, the kernel is developed in a modular fashion
- However, there are no mechanisms that prevent one module from accessing the code, or even the data, of another module

**Micro-kernel** The kernel is implemented as a set of modules executing in its own address space

- A module cannot access directly data or even code of another module

# Monolithic Implementations

- Virtually all “main stream” OS use this architecture
- It is perceived as faster

# Minix 3: Micro-kernel Based

- It has a very small size kernel (about 6 K lines of code, most of it C)
- Most of the OS functionality is provided by a set of privileged user level processes:

  **Services** E.g. file system, process manager, VM server, Internet server, and the ressurection server.

  **Device Drivers** All, of them are user-level processes

**Issue** OS services and device drivers need to execute instructions that are allowed only in privileged mode

- But now, they are executed at user-level

# Kernel Calls

**Solution** The (micro-)kernel provides a set of kernel calls

- These calls allow privileged processes to execute operations that:
  - Can be executed only when running in privileged/kernel mode;
  - That are needed for them to carry out their tasks

**Examples** from Labs 1 and 2?

- `vm_map_phys()`
- `sys_int86()`

**Note** Kernel calls are (conceptually) different from system calls

- Any process can execute a system call
- Only privileged processes are allowed to execute a kernel call

However, they use the same basic mechanism:

- An instruction that switches to privileged execution mode

# Service/DD Privileges

## How can we specify the privileges of a process?

- Via the `/etc/system.conf`

```
service at_wini {
        io
                1f0:8           # Controller 0
                3f6             # Also controller 0
                170:8           # Controller 1
                376             # Also controller 1
                ;
        irq
                14              # Controller 0
                15              # Controller 1
                ;
        system
                UMAP            # 14
                IRQCTL          # 19
                DEVIO           # 21
                SDEVIO          # 22
                VDEVIO          # 23
                READBIO         # 35
                ;
        pci_class
```