Computer Labs
The Minix 3 Operating System
And Virtual Box

2º MIEIC

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

September 20, 2020

Goals

What is Minix 3?

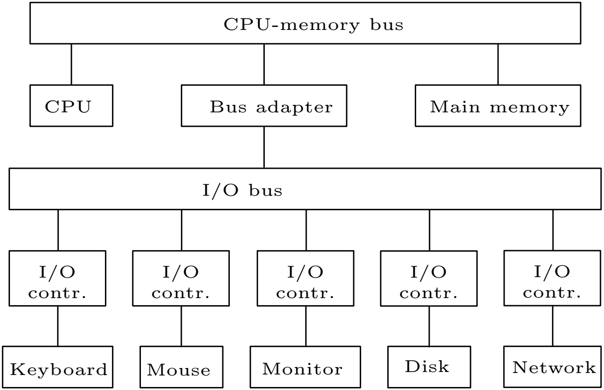
Why do we use Minix 3 in LCOM?

What is VirtualBox?

Why do we use VirtualBox in LCOM?

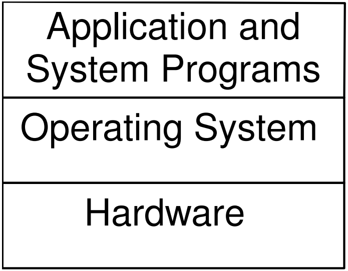
LCOM Labs

- One of the goals of LCOM is that you learn to use the programmatic interface of the most common PC I/O devices



Operating System

- In most modern computer systems, access to the HW is mediated by the operating system (OS)



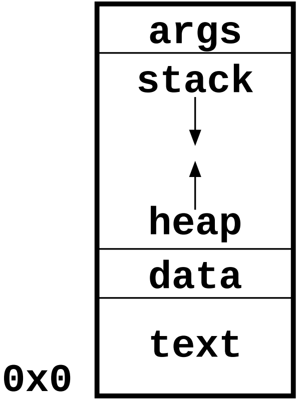
- I.e. user programs are not able to access directly the HW
  - Stability of operation
  - Security

Parenthesis: Program vs. Process

Program Piece of code, i.e. a set of instructions, that can be executed by a processor

Process OS abstraction of a program in execution

```
int main(int argc, char *argv[], char* envp[])}
```



`args` Arguments passed in the command line and environment variables

`stack` *Activation records* for invoked functions

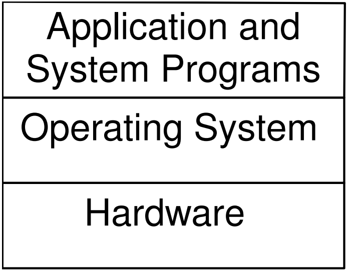
`heap` Memory region allocated dynamically with `malloc`.

`data` Memory region allocated statically by the compiller (e.g., a *string* “`Hello, World!`”)

`text` Program instructions

Operating System (corrected)

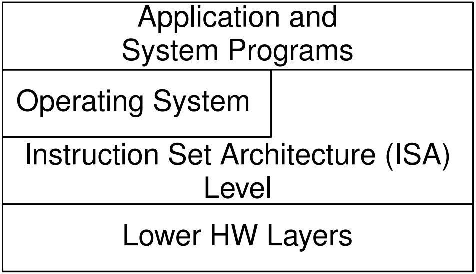
- In most modern computer systems, access to the HW is mediated by the operating system (OS)



- I.e. user **processes** are not able to access directly the HW, mostly for reasons of:

  Stability of operation
  Security

Access to the HW-level Interface

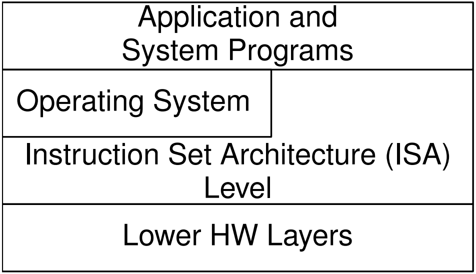


- Most of the HW interface, actually the processor instruction set, is still available to user processes
- However, a few instructions are not directly accessible to user processes
  - Thus preventing user processes from interfering with:

    Other processes most OSs are multi-process

    The OS which manages the HW resources
- Instead, the operating system offers its own “instructions”, which are known as **system calls.**

OS API: Its System Calls



Extends the ISA instructions with a set of “instructions”, system calls, that support concepts at a higher abstraction level

- OS system calls are too high-level for using directly the programmatic interface of I/O devices

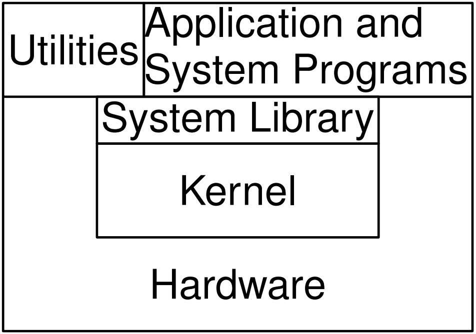
Hides some ISA instructions

- The HW provides mechanisms that ensure that applications cannot bypass the OS API

Issue The OS API (of main-stream OSs) do not allow us to directly access the programmatic interface of I/O devices

Parenthesis: OS vs. Kernel

- Usually, when we mention the OS we really mean the kernel
- An OS has several components



Kernel Which implements the OS services (system calls)

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

Parenthesis: Layered Structure

- Structure typically used to address complex problems
  - It allows us to think about the **what** without worrying about the **how** (this is usually called **abstraction**)
- This has several advantages

  Decomposition An "intractable" problem is decomposed in smaller problems that can be solved

  Modularity Facilitates adding new functionality or changing the implementation, as long as the **interfaces are preserved**
- Your project will be a somewhat complex piece of code
  - To structure it in several layers may be very important for your success

Other SW layers			
Video Driver	Keyboard Driver	Timer Driver	Mouse Driver

How is an OS/Kernel implemented?

Monolithic All OS services are implemented at kernel level by the kernel
- 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 Most OS services are implemented as modules that execute in their own address spaces
- A module cannot access directly data or even code of another module
- There is however the need for some functionality to be implemented at kernel level, but this is minimal (hence the name)

Monolithic Implementation

- Virtually all “main stream” OSs use this architecture
- It has lower overheads, and is faster

But is less reliable, because components are not isolated from each other

- If we used Linux or Windows instead of Minix, a bug in your program could just crash the entire system
  - This would make the development process very painful

Minix 3: Micro-kernel Based

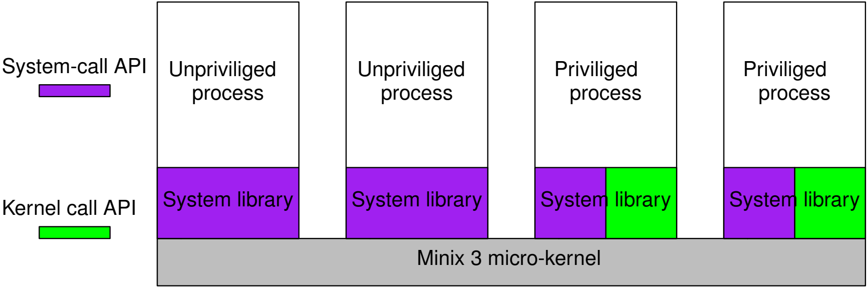
- It has a very small size kernel (about 6 K lines of code, most of it C)
- Most of the OS functionality is provided by a set of **privileged user-level** processes:

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

  Device Drivers All of them are user-level processes
- Minix 3 provides an API, which is known as **kernel-calls**, that allow privileged processes to execute instructions required by device-drivers
  - E.g. `sys_inb()`, which you'll use in Lab 2

  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

Minix 3: Non-Privileged vs. Privileged User Processes



Conclusion by using Minix 3, LCOM programs not running in kernel mode can use the programmatic interface of I/O devices

- The development process is much less painful

LCOM Lab Programs

- In LCOM, you'll use Minix 3 and its kernel-API to develop **privileged** programs:
  - Akin to device-drivers
    - They will access/control I/O devices
  - Different from device drivers. Your programs:
    - Will be self-contained

    Whereas each device driver:
    - Manages a class of I/O devices
    - Provides an interface so that other processes can access I/O devices of that class
- The use of Minix 3 simplifies the development
  - Your processes do not belong to the kernel
  - Their actions can be controlled

  Thus, bugs are much less harmful

VirtualBox (1/2)

Problem Direct access to the programmatic interface of a computer's I/O devices raises **security risks**. E.g.

- If you are able to directly access a HDD (or an SSD), you can have access to the data it stores
  - Worse, you can install malware that can, e.g., spy on users, even long after the last time you used that computer
- In FEUP, these risks are unacceptable

Solution Use a **virtual machine**, VirtualBox in the case of LCOM

VirtualBox (2/2)

What is VirtualBox? Is a (system) virtual machine, more specifically:

- It is a program that emulates the HW of a PC
  - VirtualBox runs as a privileged process in a computer system
- It can run (most) programs that run on a PC without any modification
  - Both the operating systems
  - And applications or user programs.

Why is this useful? When you run a program in Virtual Box you can only access emulated resources not the physical resources. E.g.:

- Access to an emulated HDD (or SDD) exposes only the data stored in that emulated HHD (usually a file in a physical HDD), not the data in the entire (physical) HDD of **host** computer, i.e. the computer that runs the VM