Grupo ARCOS
Universidad Carlos III de Madrid

Lesson 1
Introduction

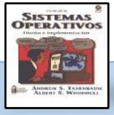
Operating systems design
Degree in Computer Science and Engineering

Recommended materials

Base

1. Carretero 2007:
   1. Cap. 2

Recommended



1. Tanenbaum 2006:
   1. Cap.1
2. Stallings 2005:
   1. Parte uno. Transfondo.
3. Silberschatz 2006:
   1. Cap.1

Contents

1. What an Operating System is.
   1. Definition, main functionalities and features

1. Operating system structure.
   1. Main goals, structure and asynchronous execution.
   2. Kernel and modules

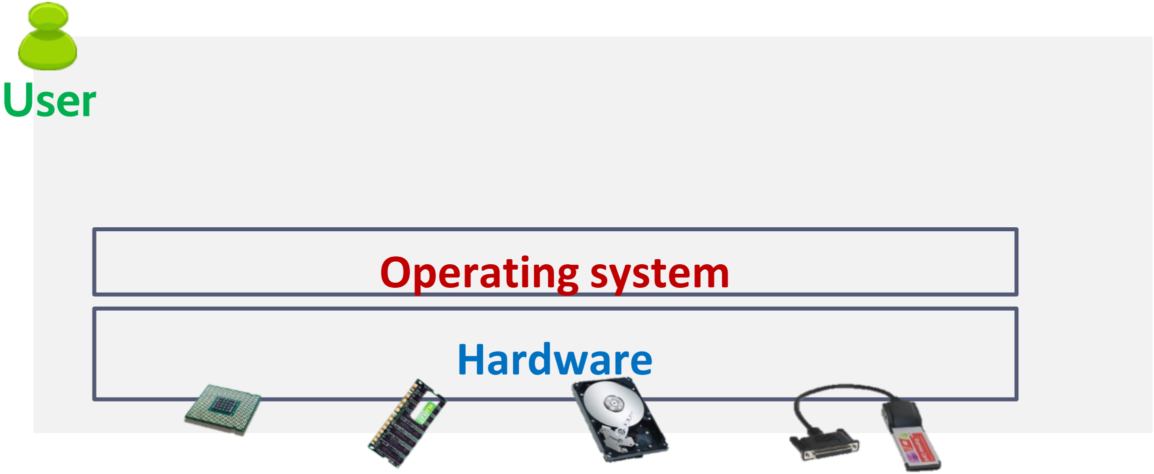
Contents

1. What an Operating System is.
   1. **Definition**, main functionalities and features

1. Operating system structure.
   1. Main goals, structure and asynchronous execution.
   2. Kernel and modules

What is an Operating System?

- Operating system: software designed to communicate users and hardware and to manage the available resources efficiently.



What is an Operating System?

- Operating system: software designed to communicate users and hardware and to manage the available resources efficiently.

Contents

1. What an Operating System is.
   1. Definition, **main functionalities** and features

1. Operating system structure.
   1. Main goals, structure and asynchronous execution.
   2. Kernel and modules

Operating system functionalities

User

Operating system

- User interface
- Resource manager:
  - CPU, memory, etc.
- Extended machine:
  - Services, programming interface, etc.

Hardware

Operating system functionalities

User

Operating system

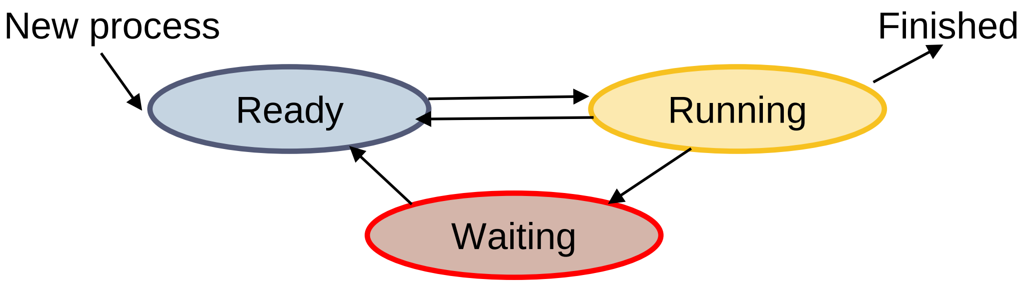
- User interface
- Resource manager:
  - CPU, memory, etc.
- Extended machine:
  - Services, programming interface, etc.

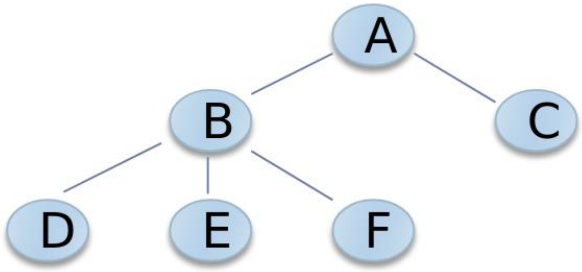
Hardware

Fundamental abstractions

Processes

- Processes, process table, process tree
- Basic image, scheduling, signals
- Users and group identifications
- User interface (*shell*)



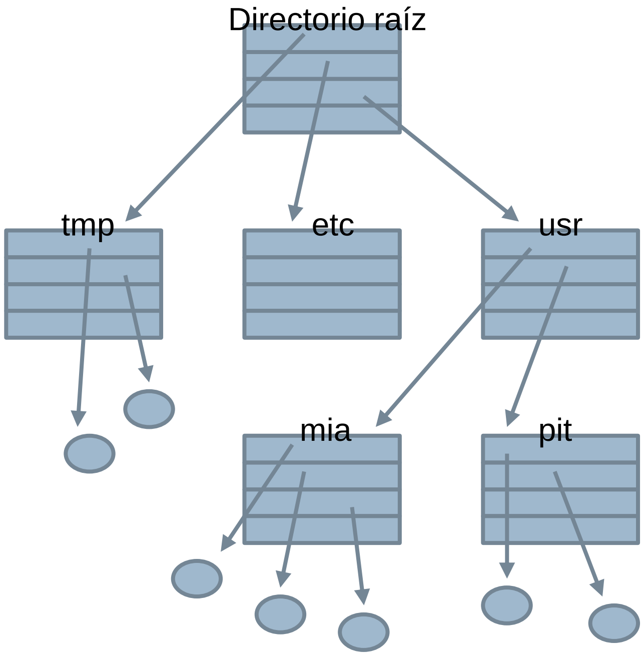


Process tree

https://www.microsoft.com/resources/sharedsource/windowsacademic/curriculumresourcekit.mspx

Fundamental abstractions
Files

- Files and directories
- Path, working directory and root.
- Protection
- File descriptors
- Special files:
  - I/O Devices
  - Pipes
- Standard input/ourput/error.



Operating system functionalities

User

Operating system

- User interface
- Resource manager:
  - CPU, memory, etc.
- Extended machine:
  - Services, programming interface, etc.

Hardware

Resource management

- Processing management
  - Scheduling
  - Priorities, multi-user
- Memory management
  - Memory assignement among processes with protection and sharing.
- Storage management – File systems
  - Offers an unified logical vision for users and programs that is independent of the physical storage.
- Device management
  - Hide away the hardware dependencies
  - Provide support for concurrent accesses

Operating system functionalities

User

Operating system

- User interface
- Resource manager:
  - CPU, memory, etc.
- Extended machine:
  - Services, programming interface, etc.

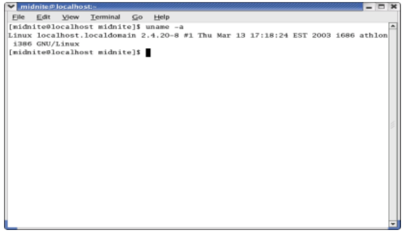
Hardware

User interface

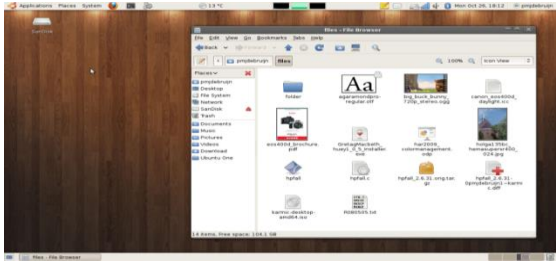
- **Programming interface**:
  - System calls.

ret = close (filedesc) ;

- **User interface**:
  - command-line interface or CLI



  - Graphic Interface o GUI



Operating system functionalities

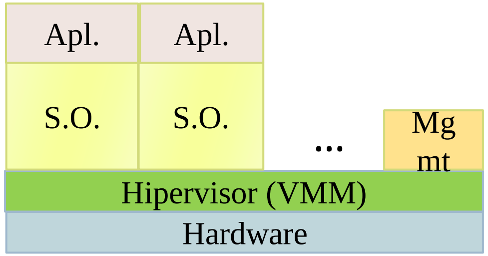
User

Operating system

- User interface
- Resource manager:
  - CPU, memory, etc.
- Extended machine:
  - Services, programming interface, etc.

Hardware

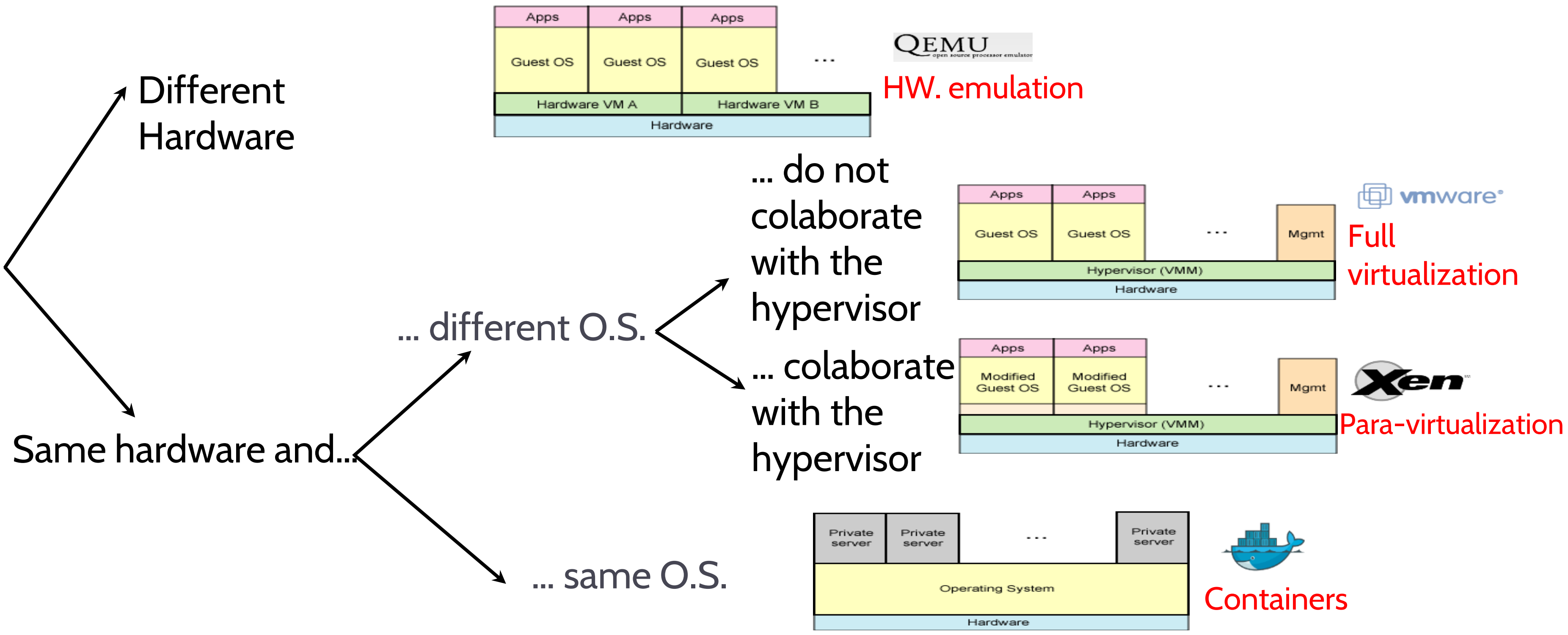
Virtual machines



- An operating system virtualize part of the hardware elements; Why not virtualize all of them?
- IBM used this idea on their mainframes since 70s.
- An hipervisor virtualize the whole computer, allowing the execution of multiple operating systems at the same time.
- Virtualization:
  - [+] offers an excelent system isolating among systems and reduces costs thanks to the flexible resource allocation.
  - [-] overheads

http://www-128.ibm.com/developerworks/library/l-linuxvirt/index.html

Virtual machines



Contents

1. What an Operating System is.
   1. Definition, main functionalities and **features**

1. Operating system structure.
   1. Main goals, structure and asynchronous execution.
   2. Kernel and modules

Main features

- Portable
- Adaptative
- Multidisciplinary
- Complex
- Sensitive

Portability

Supercomputer
Unix, Linux, …

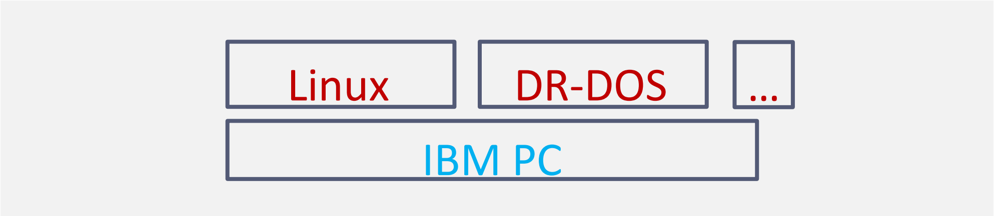
Mainframe
OS/360, z/OS, …

Minicomputers y PC
Unix, MacOs, Windows, …

Embedded
VxWorks, QNX, LynxOS, Android, iOS, Windows Embedded, …

1) Portability

- Same hardware, different O.S.: IBM PC



- Same O.S., different hardware: Unix

2) Adaptive to changes

- New user requirements:
  - Voice recognition, multitouch, etc.
- Hardware evolution:
  - Controllers for new devices
  - Multicore systems, virtualization, etc.
- Integrate solutions for different environments:
  - Batch processing, multiprogramming, shared CPU time, etc.
  - Multiuser, cooperative work, etc.
  - Distributed systems, network services, etc.

3) Multidisciplinary software

- Integrates works from different areas:
  User interface, system software, artificial intelligence, security, software engineering, etc.

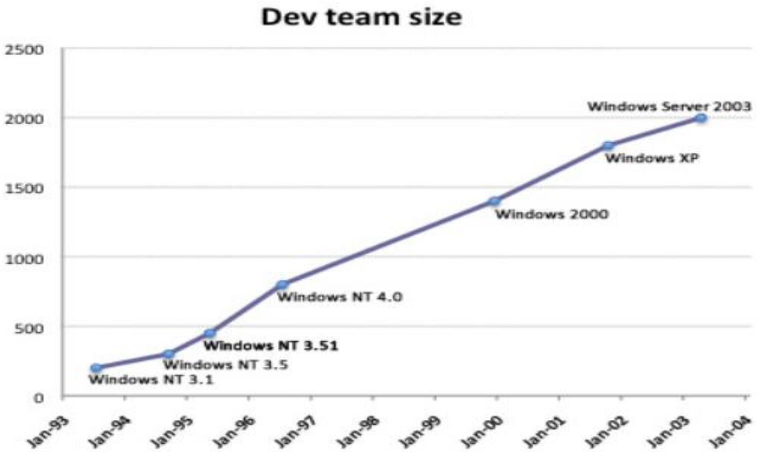
http://work-at-home-data-entry.com/wp-content/uploads/2014/10/Work-from-home-team-group-of-workers-icon.png

4) Complex software

- Many lines of code.
- Many working groups.

4) Complex software

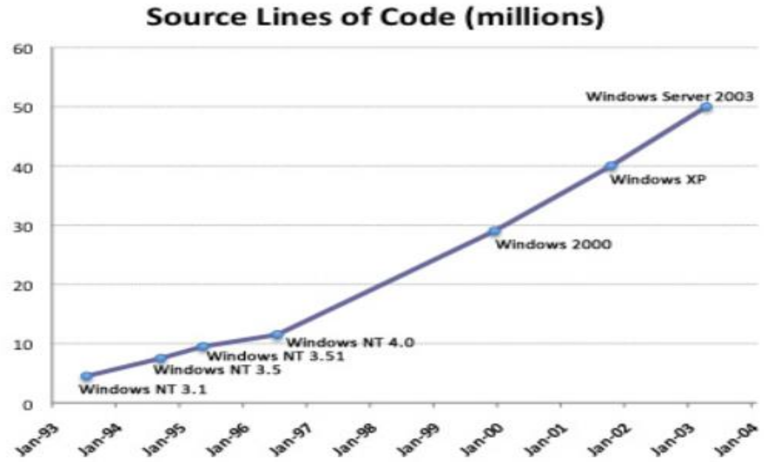
- Many lines of code.
- Many working groups.



http://www.zdnet.co.uk/reviews/desktop-os/2010/11/20/a-quarter-century-of-windows-40090900/5/#top

4) Complex software

- Many lines of code.
- Many working groups.



http://www.zdnet.co.uk/reviews/desktop-os/2010/11/20/a-quarter-century-of-windows-40090900/5/#top

5) Sensitive software

- An error in a driver (software in charge of managing a device) may block the entire system.
- It may work with data that should be carefully treated to not expose the information to not legitimate users nor lose them.

Contents

1. What an Operating System is.
   1. Definition, main functionalities and features

1. Operating system structure.
   1. **Main goals**, structure and asynchronous execution.
   2. Kernel and modules

Goals of an operating system design

- Performance and efficiency.
  - Low overheads, efficient resource usage
- Stability: robustness and resilence
  - Uptime, aceptable degradationde, reliability and integrity
- Capacity: flexibility and compatibility
- Security and protection
  - Protection among users
  - Security
- Portability
- Clarity
- Extensibility

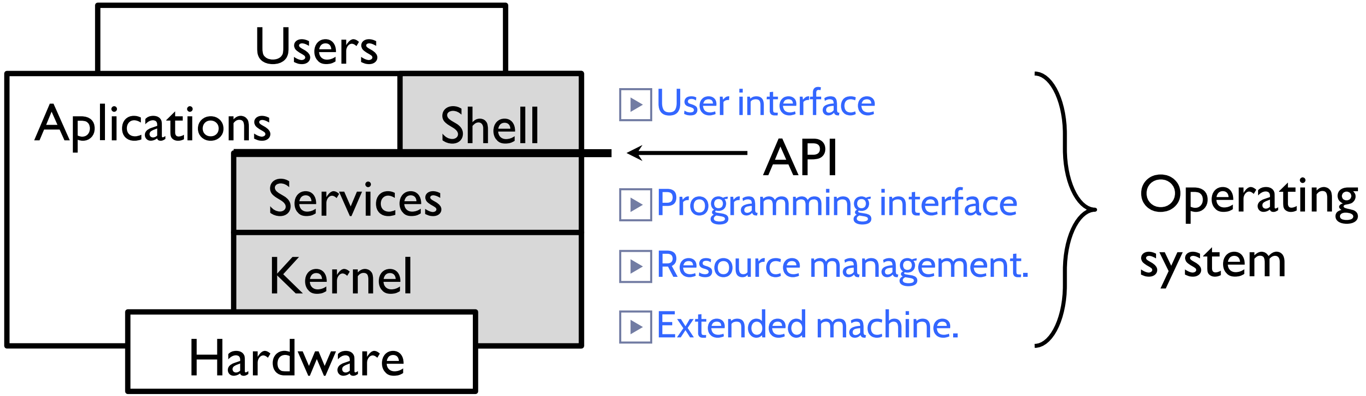
http://www.cc.gatech.edu/~pwh/

Contents

1. What an Operating System is.
   1. Definition, main functionalities and features

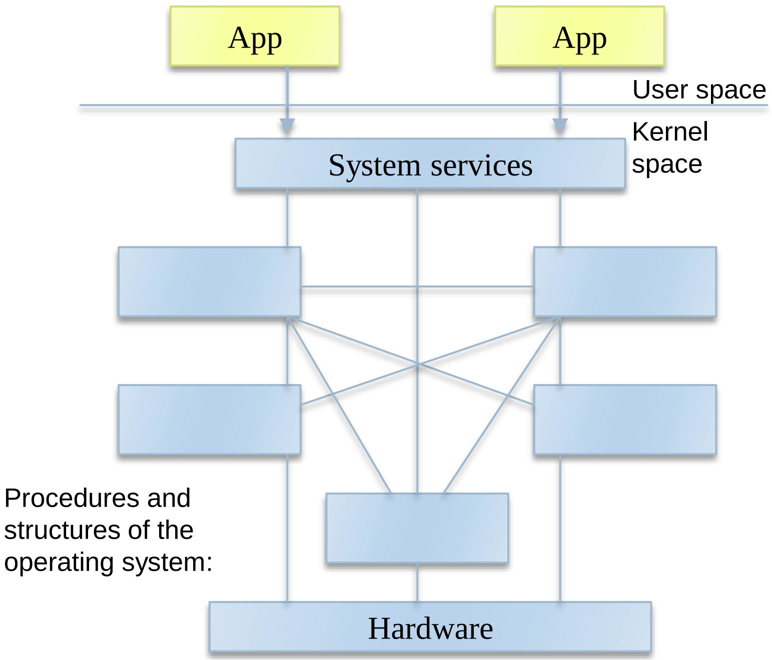
1. Operating system structure.
   1. Main goals, **structure** and asynchronous execution.
   2. Kernel and modules

Operating system structure



Operating system structure
Monolithic (macrokernel)

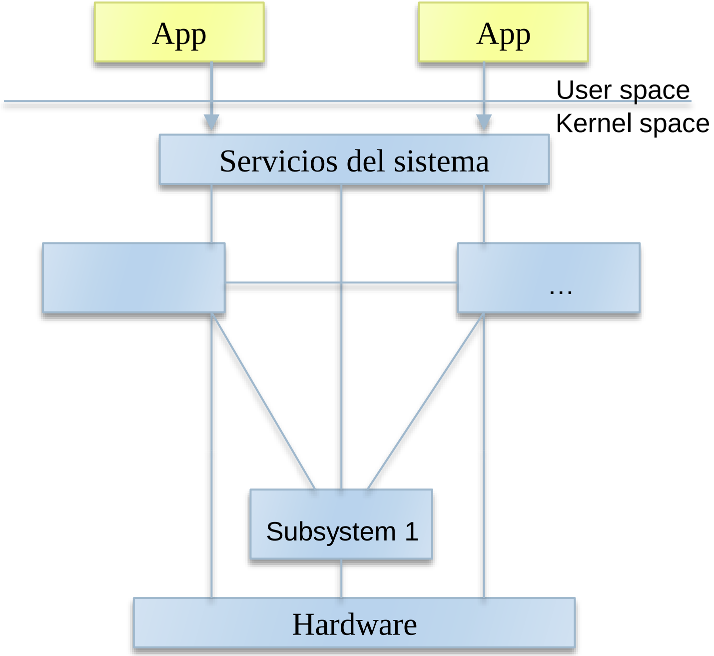
- monolothic system.
- Unstructured.
- Every point can access to any variable or fuction of other kernel part.
- [I] Poor maintanability, error sensitive.



Operating system structure
subsystems

- Monolithic system comprised by logic subsistems that provides well defined interfaces as entry points.
- The subsystems groups related procedures and structures.

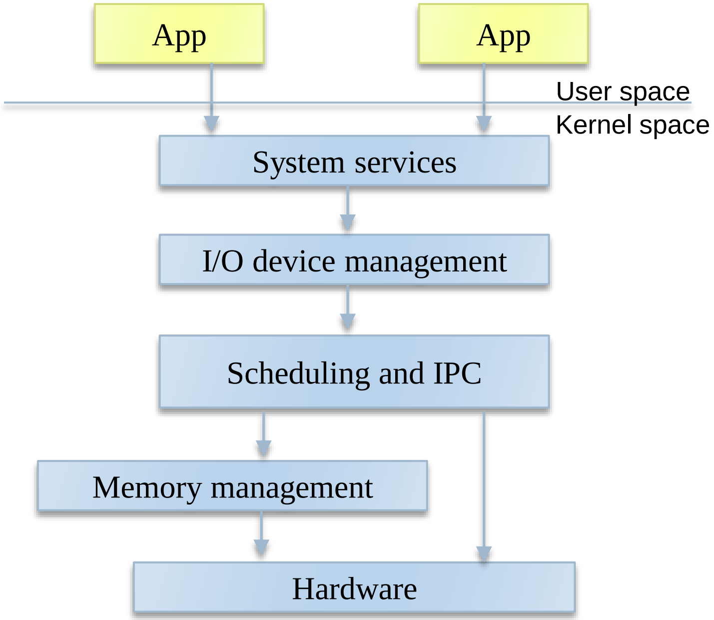
- e.g: Linux



Operating system structure
Layered

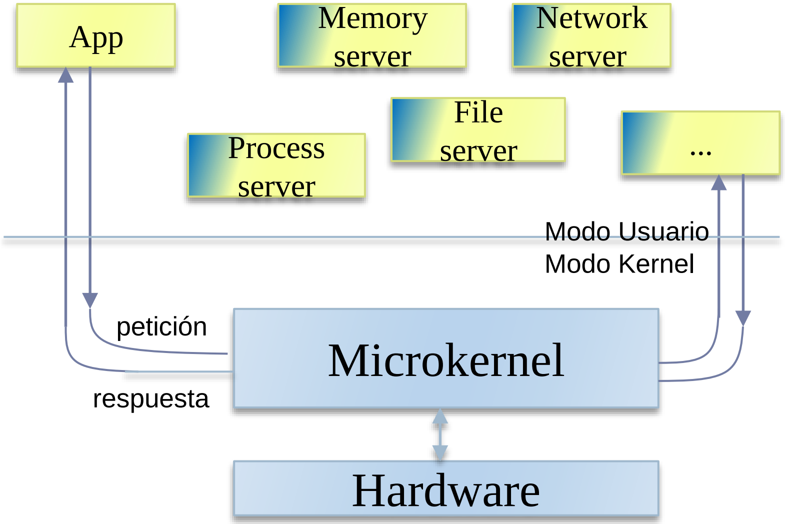
- Structured in logic layers.
- Each layer only provide access to lower layers.

- e.g:
  - THE (Dijkstra)
  - Multics, this operating system added the privilege rings.



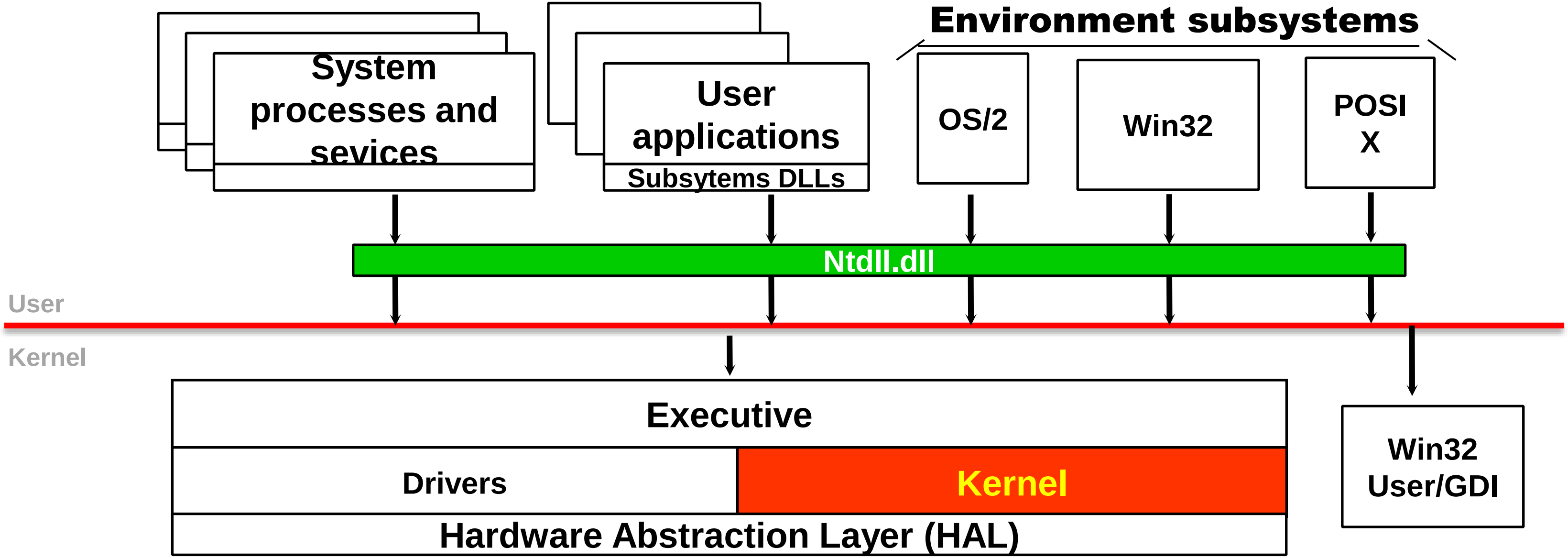
Operating system structure
Microkernel

- The main componentes are executed outside the kenel space.
- microkernel:
  - Scheduling and process management.
  - Basic virtual memory management.
  - Basic communication among processes.
- e.g:
  - Match, QNX, Minix, L4, etc.



Operating system structure

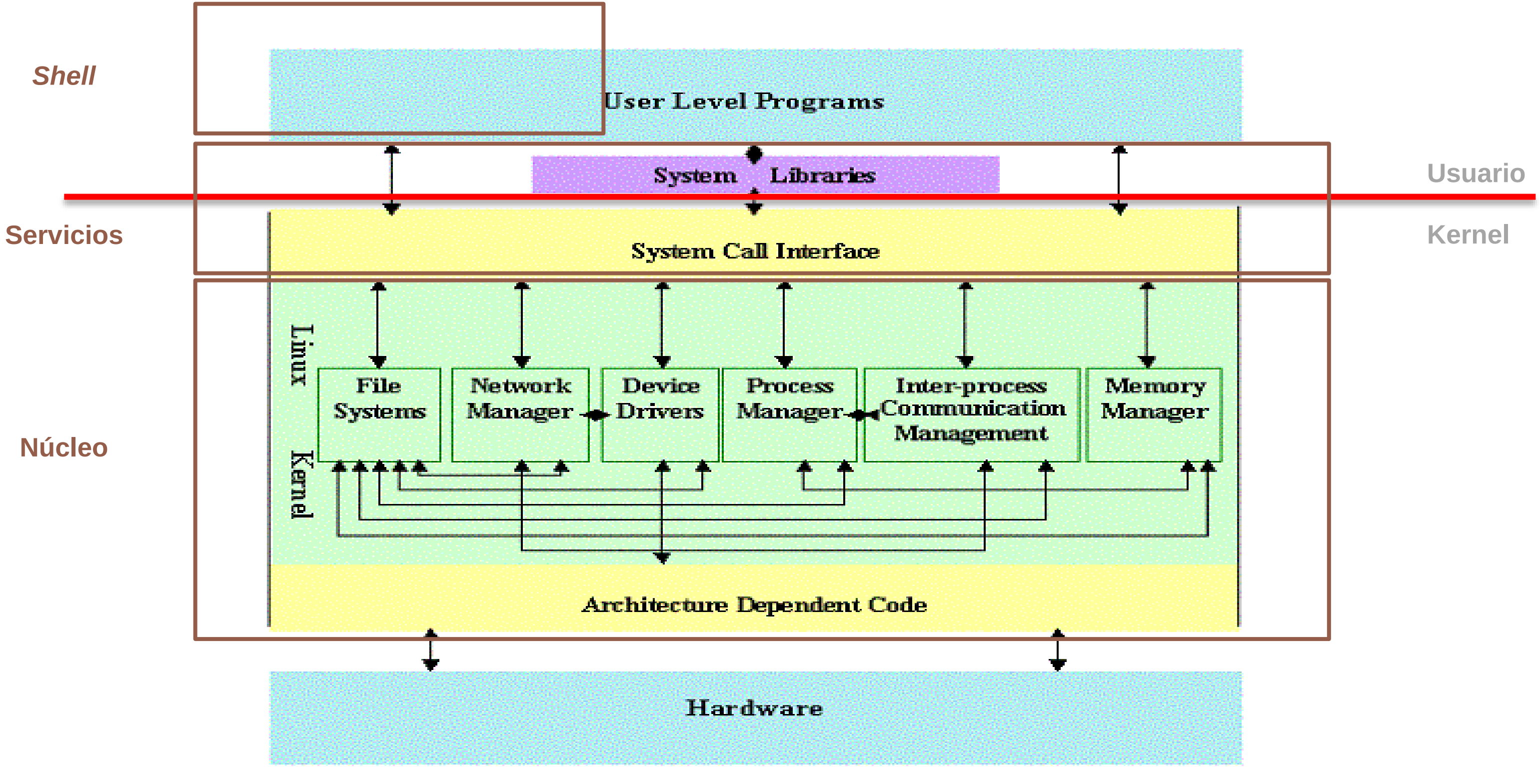
Windows 2000 (simplified)



http://technet.microsoft.com/en-us/library/cc750820.aspx

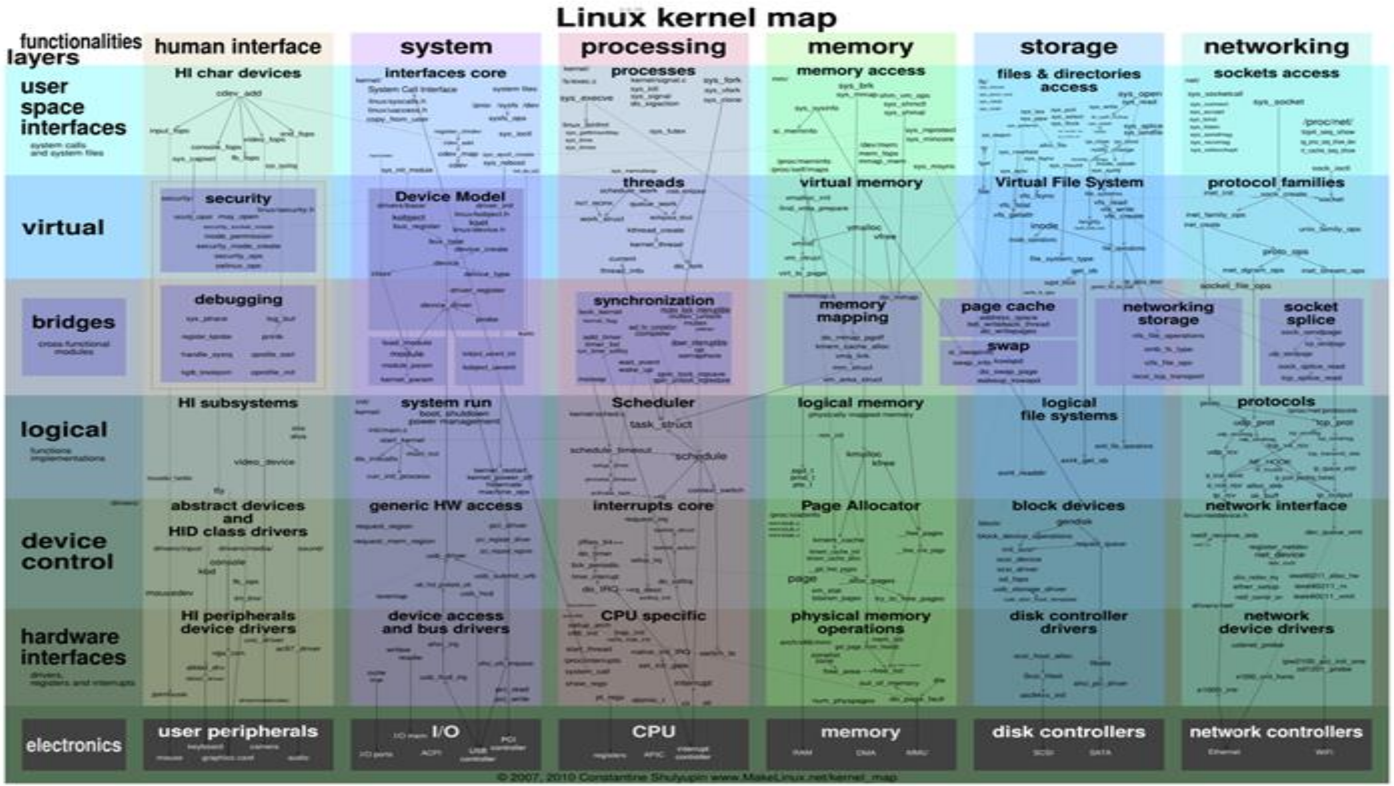
Example of a subsytem operating system

Linux (simplified)



Real O.S.

Linux ('less' simplified)



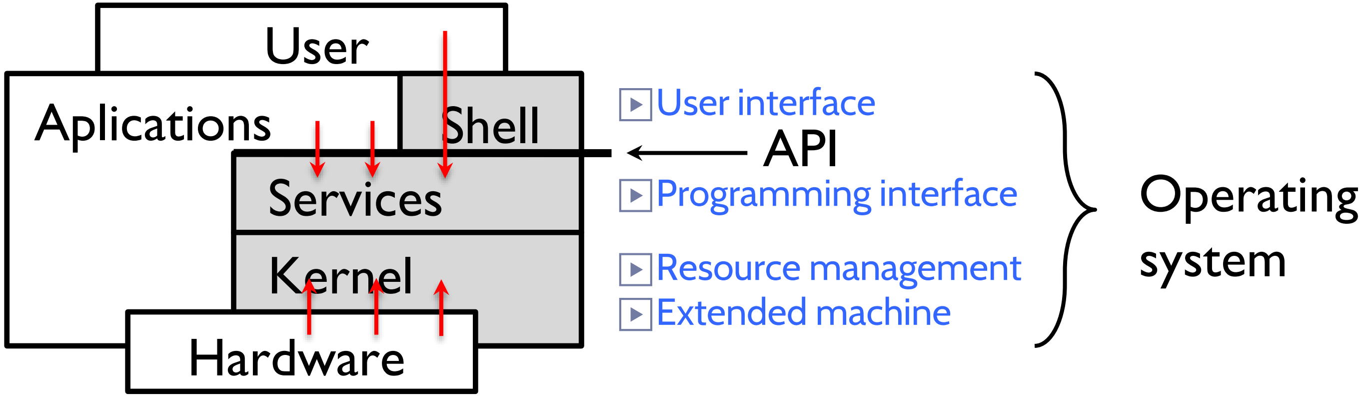
http://www.makelinux.net/kernel_map.shtml

Contents

1. What an Operating System is.
   1. Definition, main functionalities and features

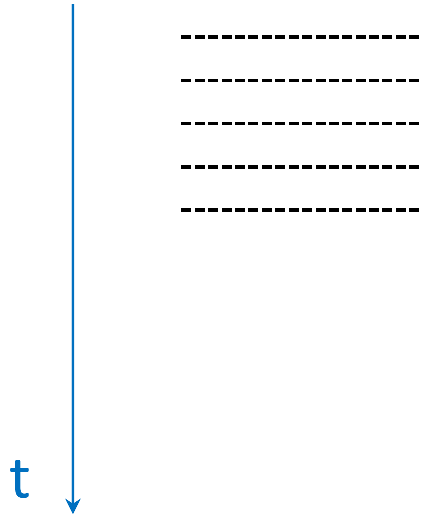
1. Operating system structure.
   1. Main goals, structure and **asynchronous execution**.
   2. Kernel and modules

Operating system structure



Asynchronous execution

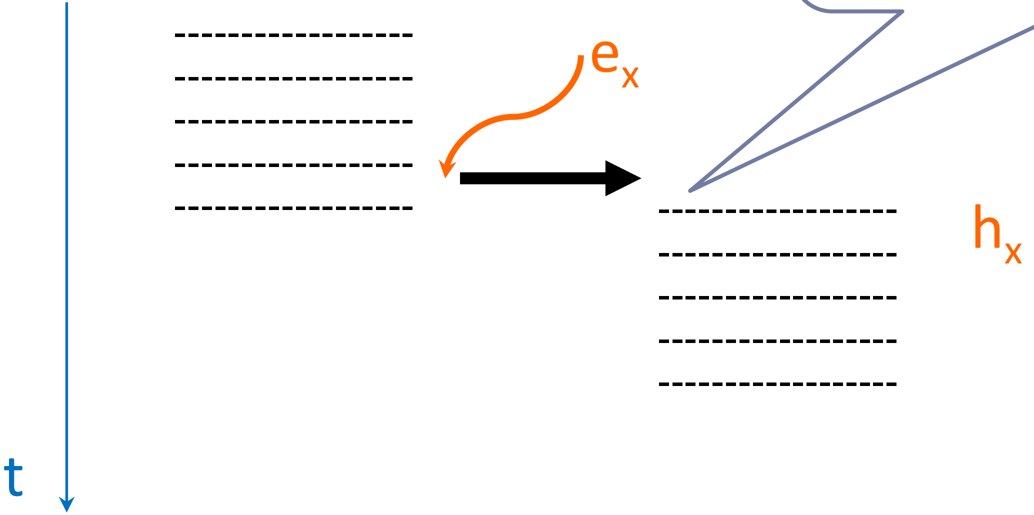
Execution (general)



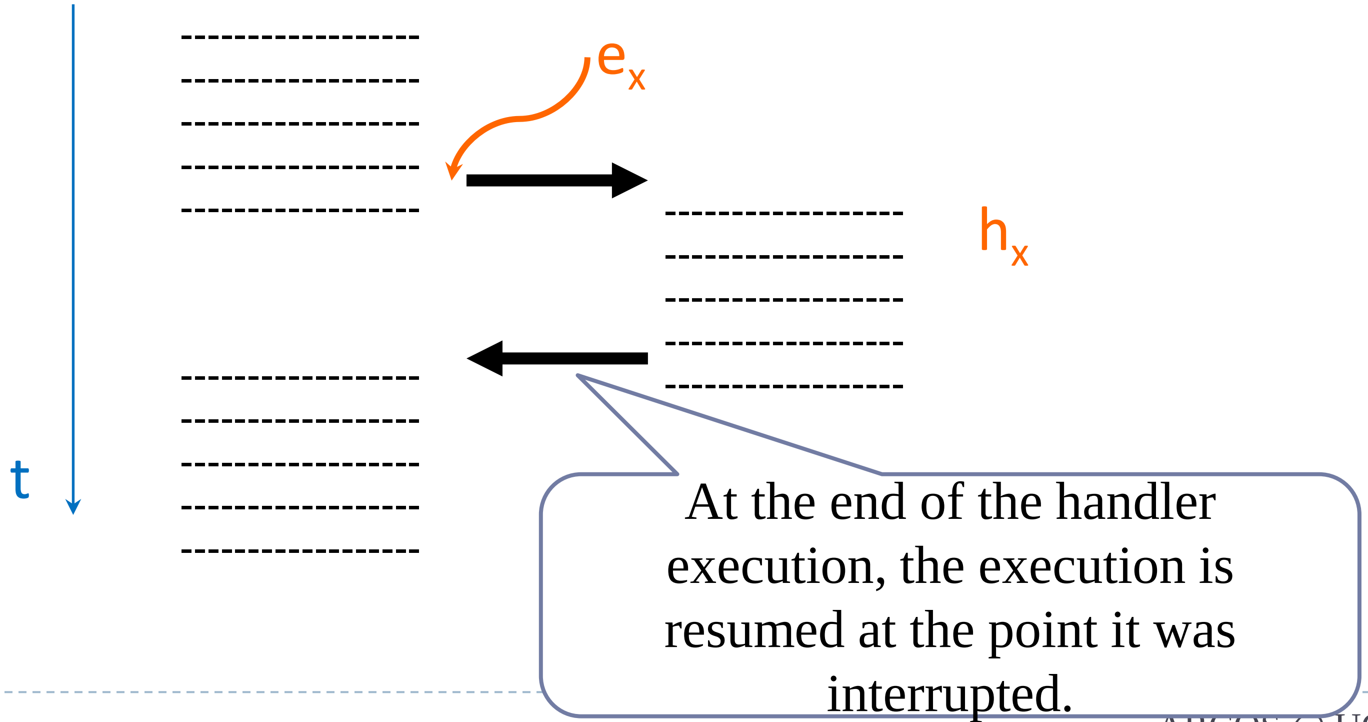
Asynchronous execution

execution (general)

When a new event happens (e_x) the corresponding handler is executed (h_x)



Asynchronous execution
Execution (general)



Asyncrhonous execution

Source code (general)

```
int main ( … )
{
     …
     On (event1, handler1) ;
     …
}
```

1) Asociate handler 1 to event 1

Asynchronous execution
Source code (general)

```
void handler1 ( … )
{
}

int main ( … )
{
    …
    On (event1, handler1) ;
    …
}
```

2) Implement the event handler function

1) Asociate handler 1 to event 1

Asynchronous execution

Source code (general)

```
int global1;
…

void handler1 ( … )
{
}

int main ( … )
{
     …
     On (event1, handler1) ;
     …
}
```

3) To communicate functions , we employ global variables.

2) Implement the event handler function

Example of asynchronous execution

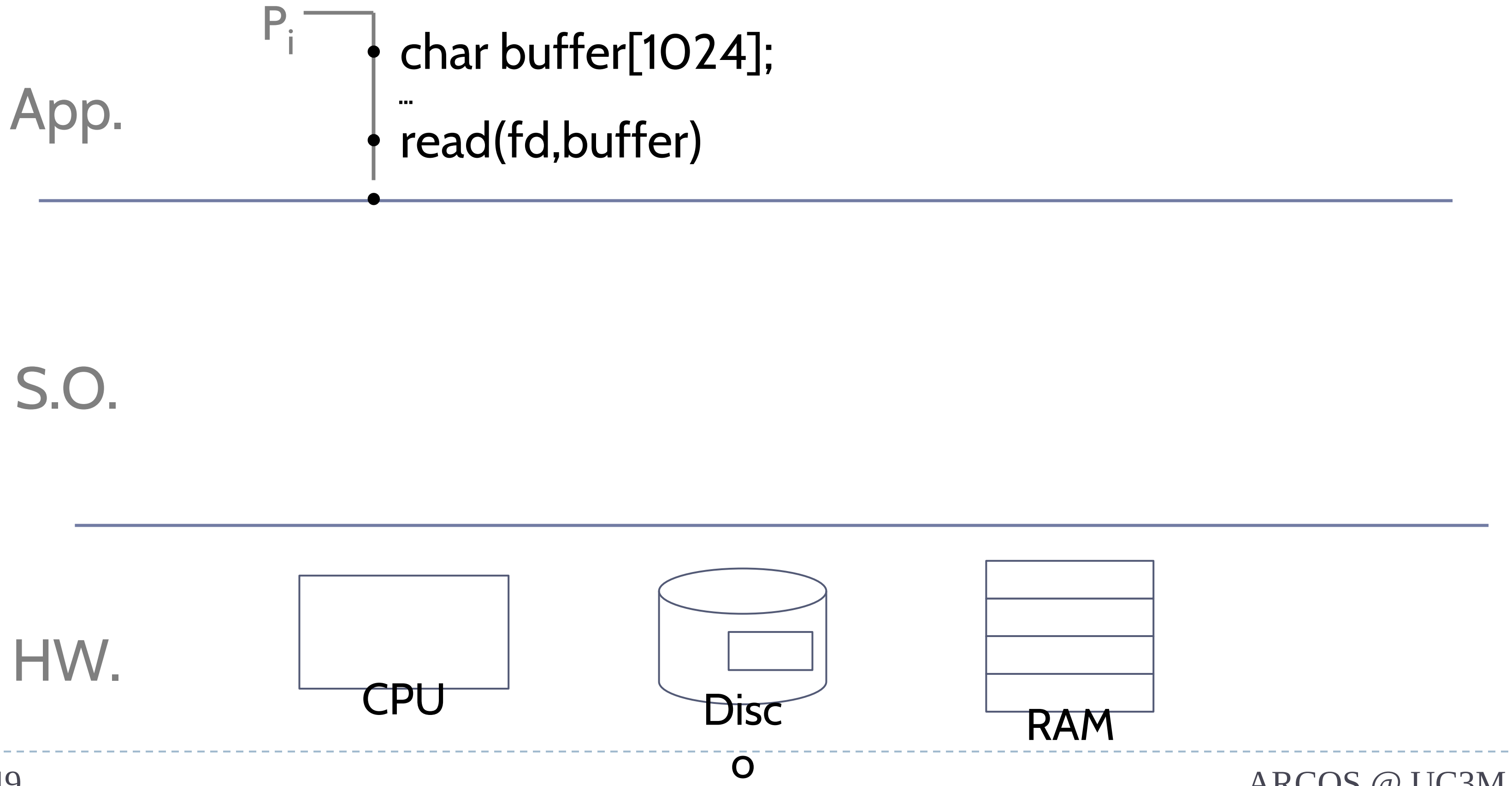
Signals

signal.h

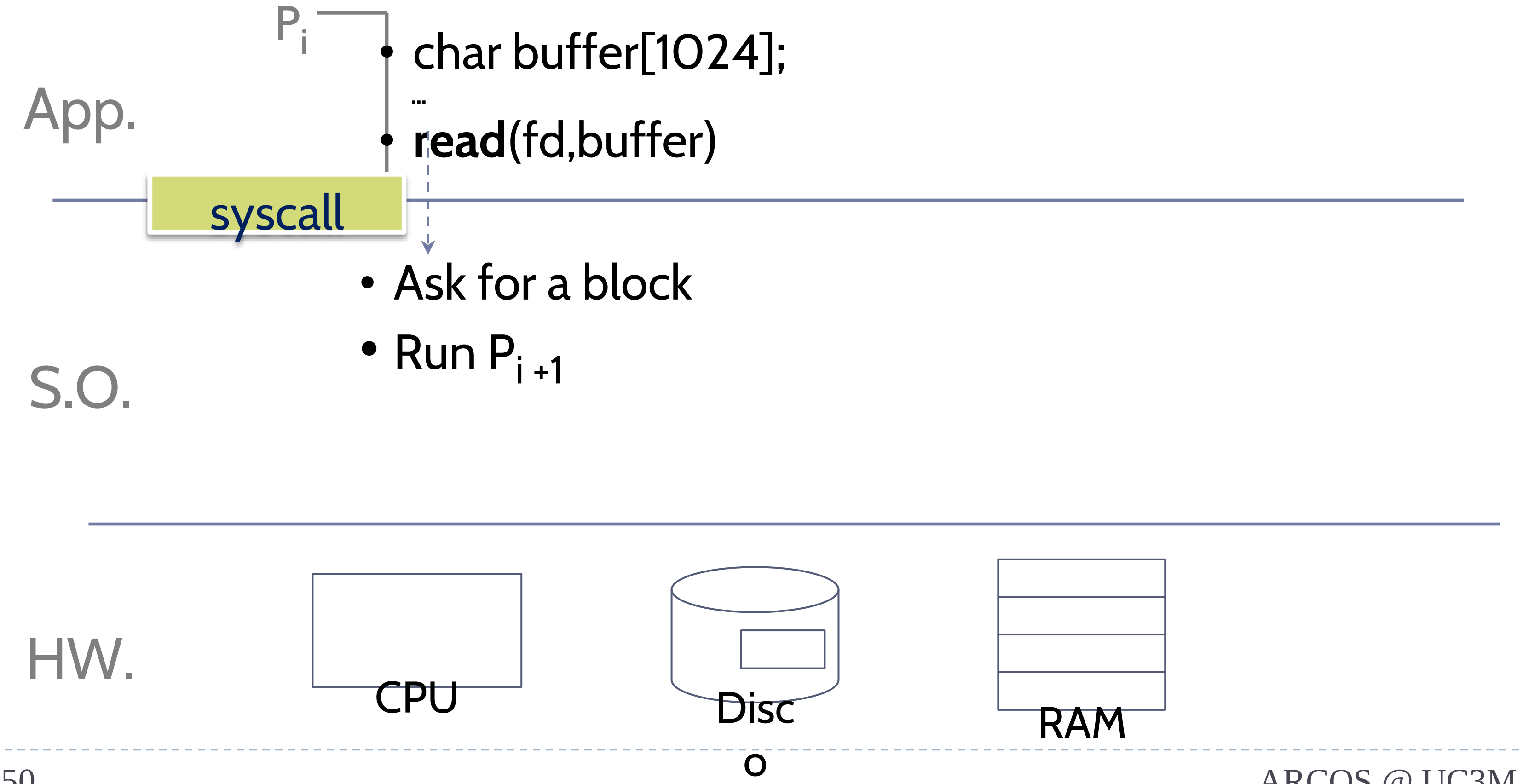
```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>

void sig_handler (int signo)
{
    if (signo == SIGINT)
        printf("received SIGINT\n");
}

int main(void)
{
     if (signal(SIGINT, sig_handler) == SIG_ERR)
         printf("\ncan't catch SIGINT\n");

      sleep(60); // simula un proceso largo.

     return 0;
}
```


http://www.thegeekstuff.com/2012/03/catch-signals-sample-c-code/

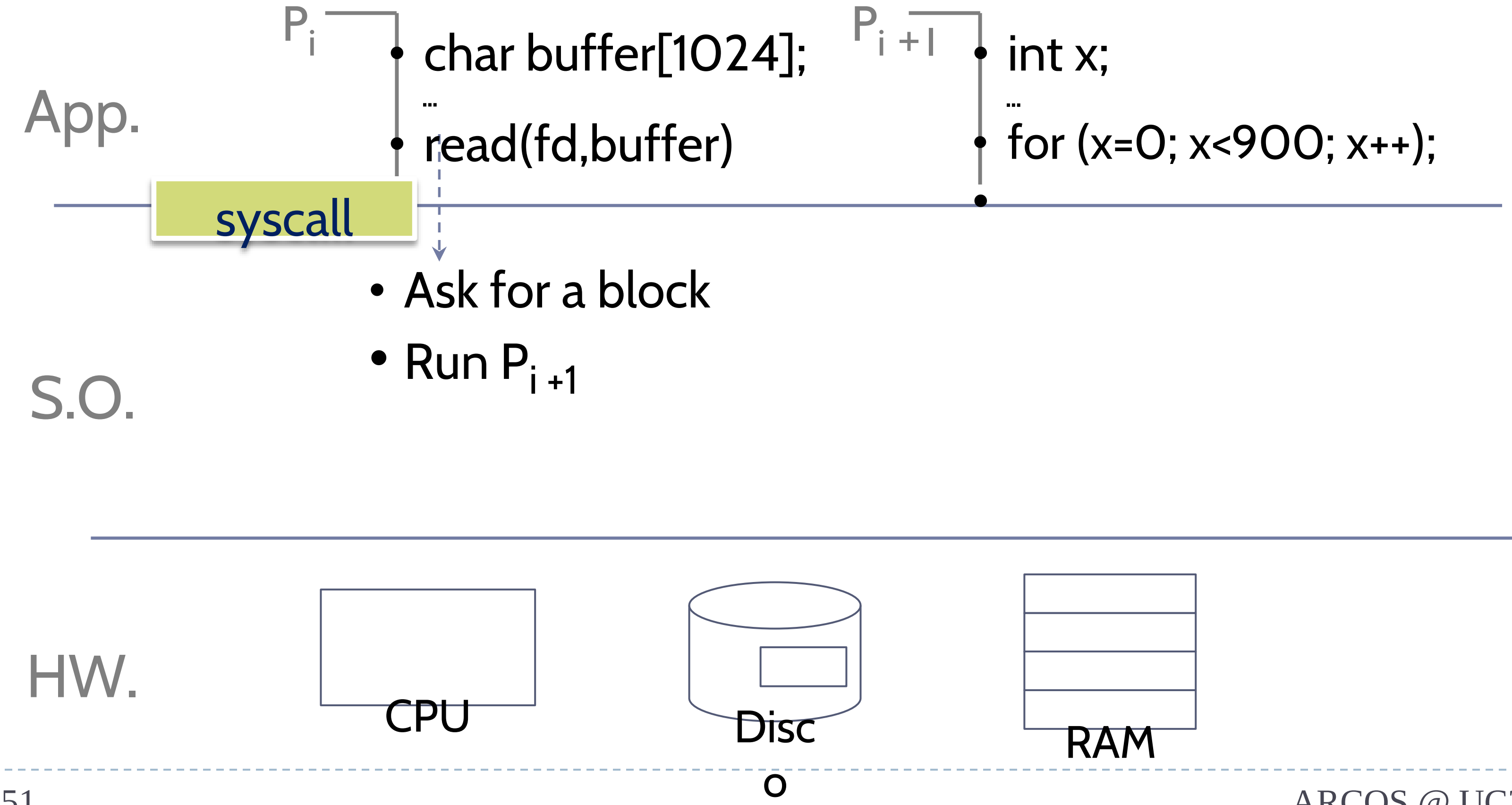
Asynchronous execution
Simplified example



Asynchronous execution
Simplified example

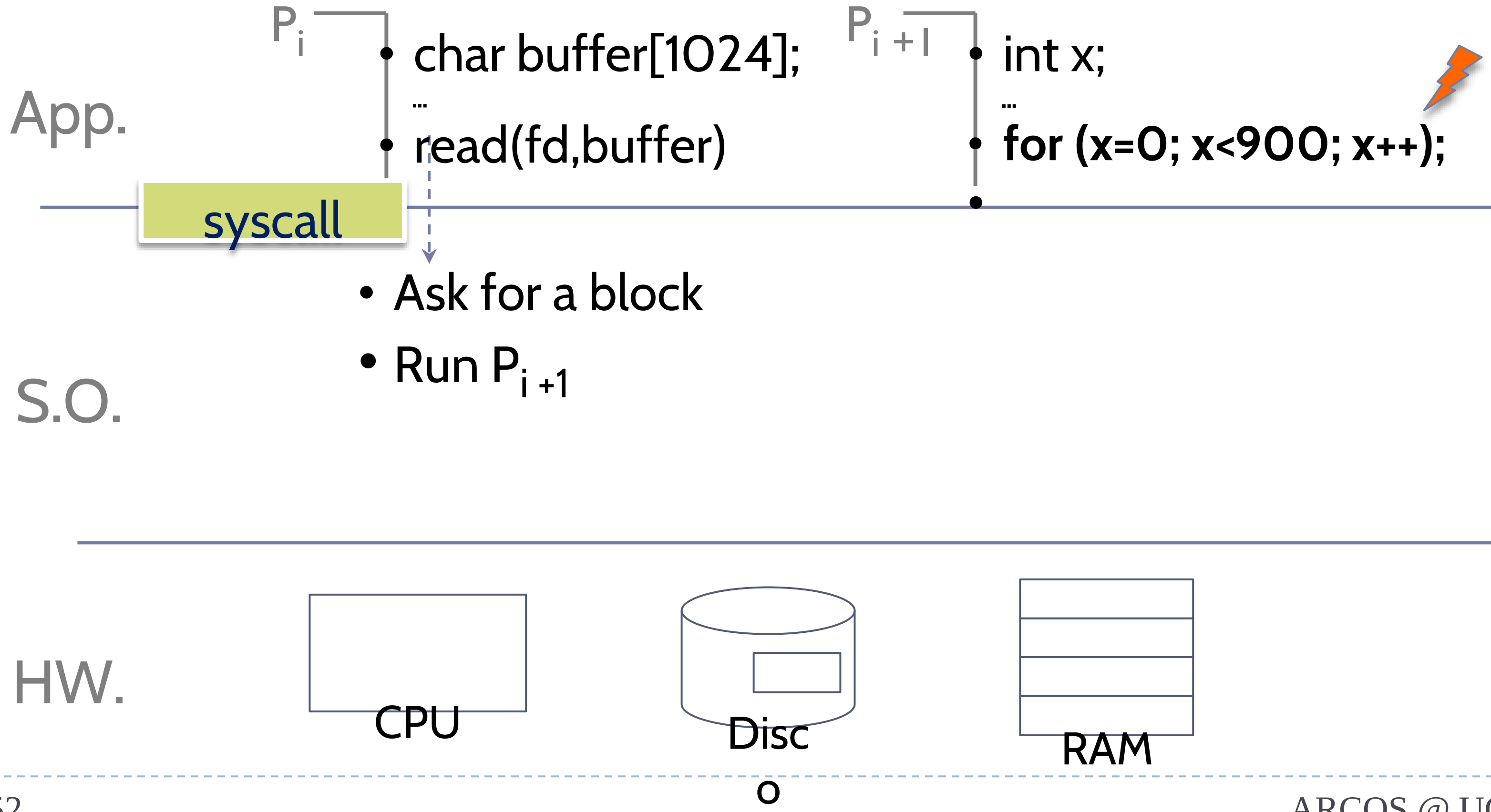


Asynchronous execution
Simplified example



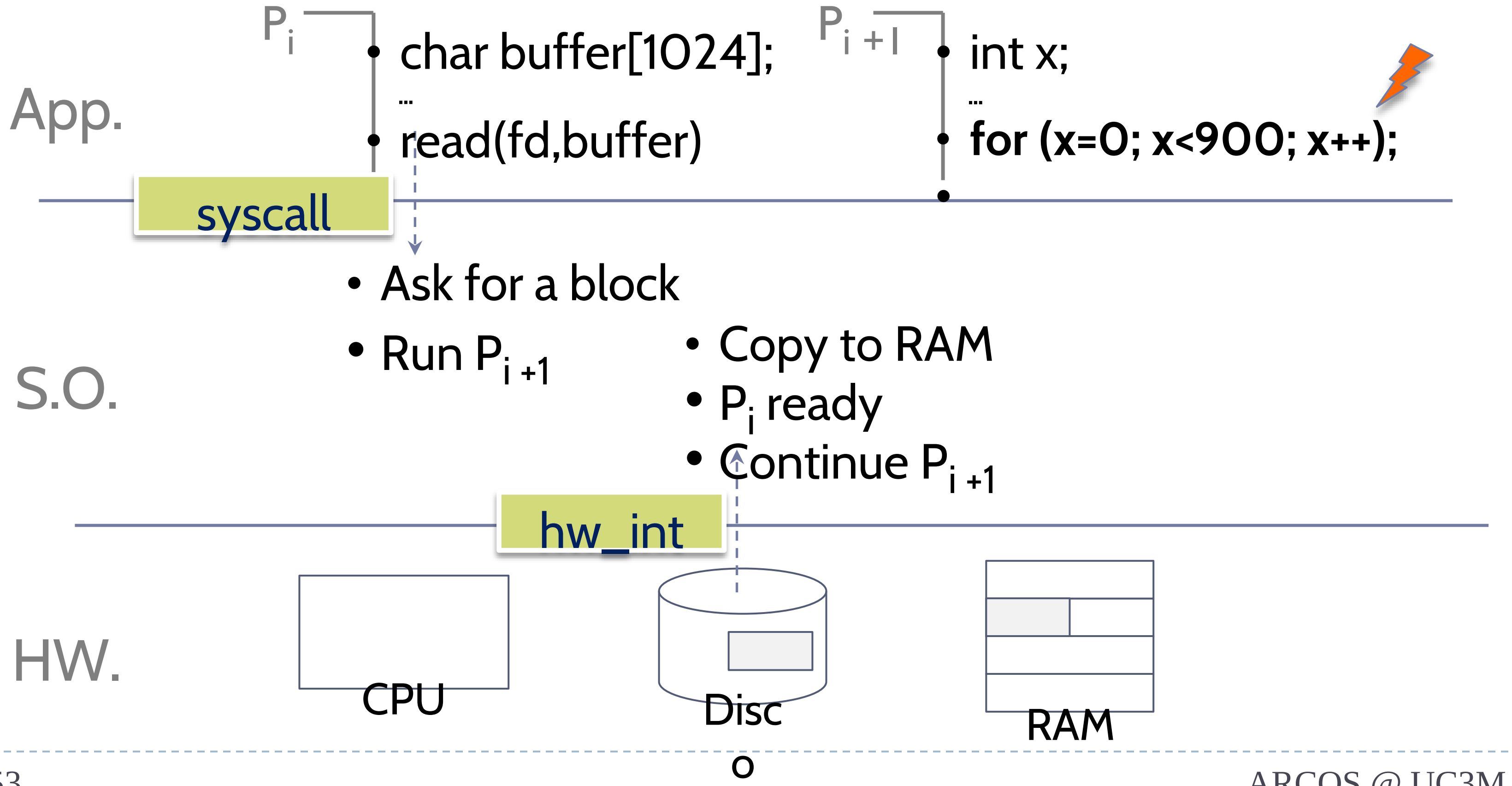
Asynchronous execution

Simplified exemple



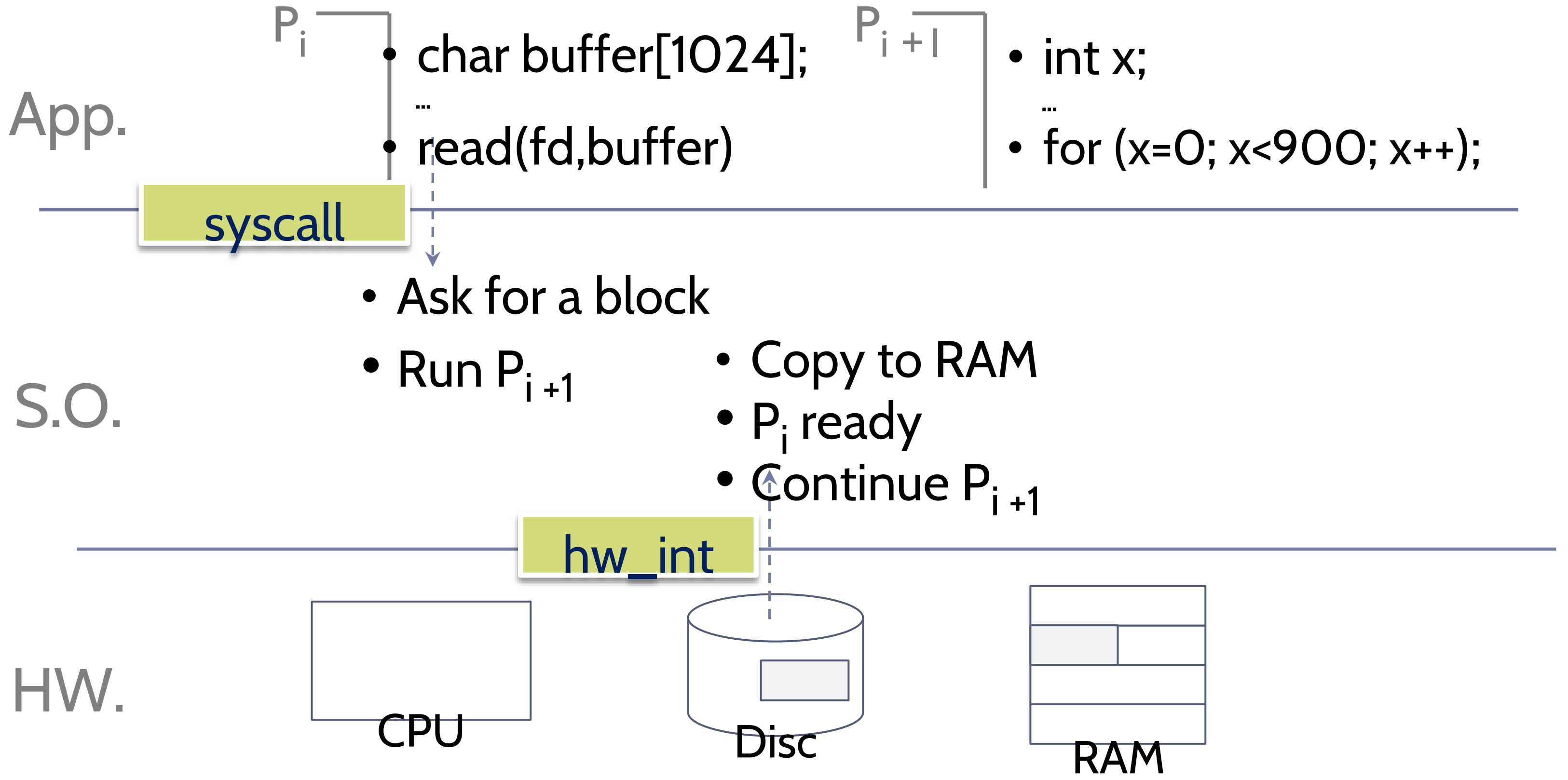
Asinchronous execution

Simplified example

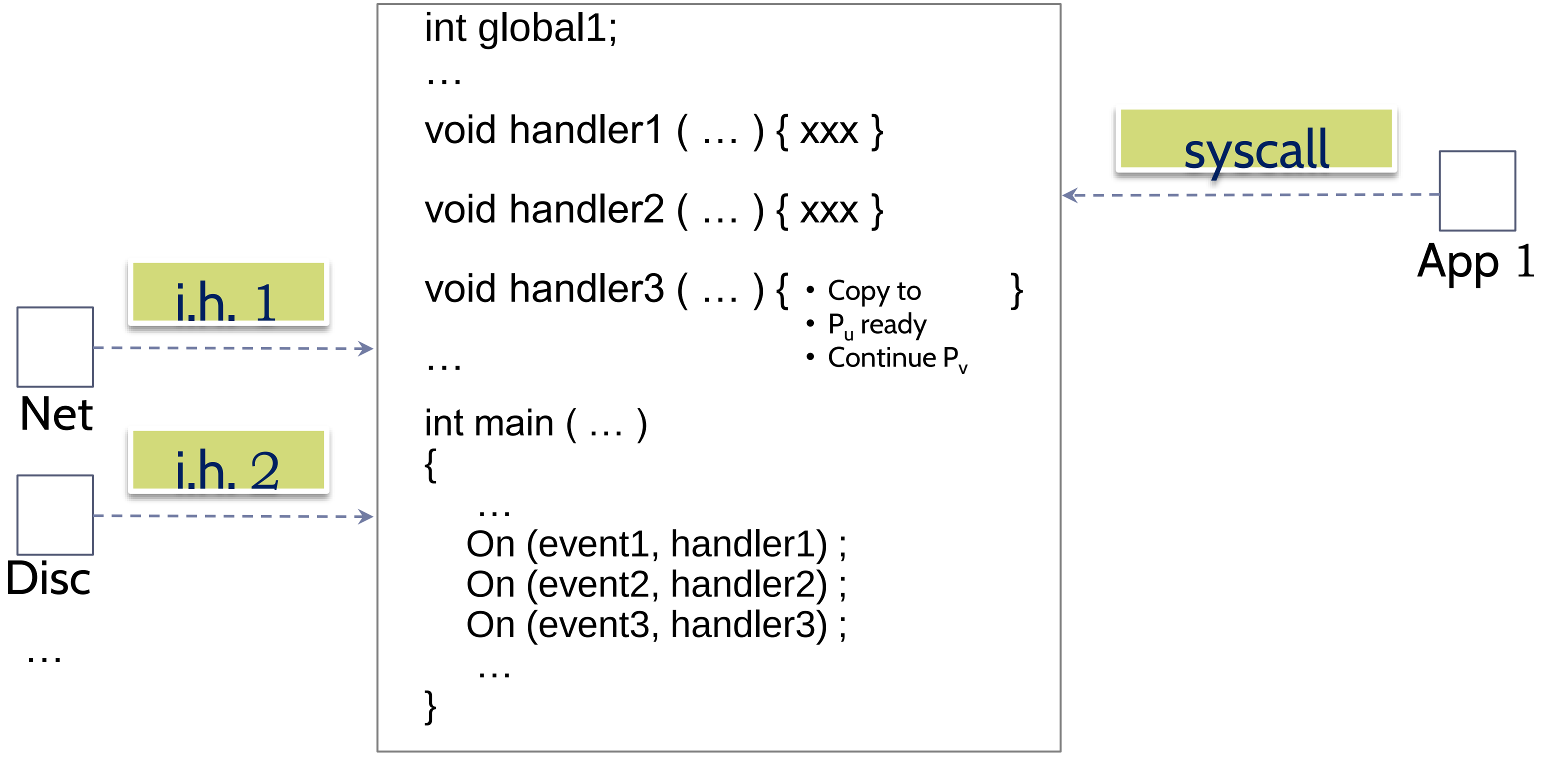


Asinchronous execution

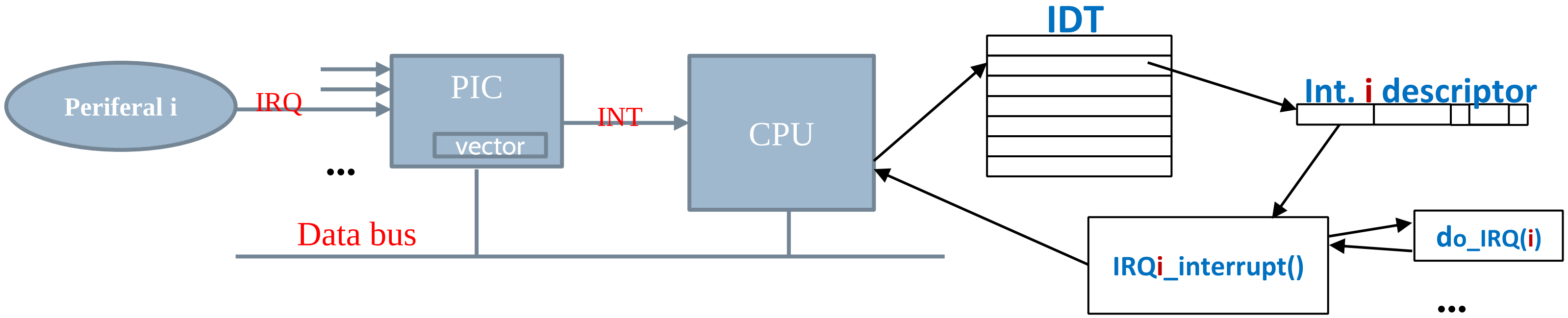
Simplified example



Operating system structure
Source code (general)

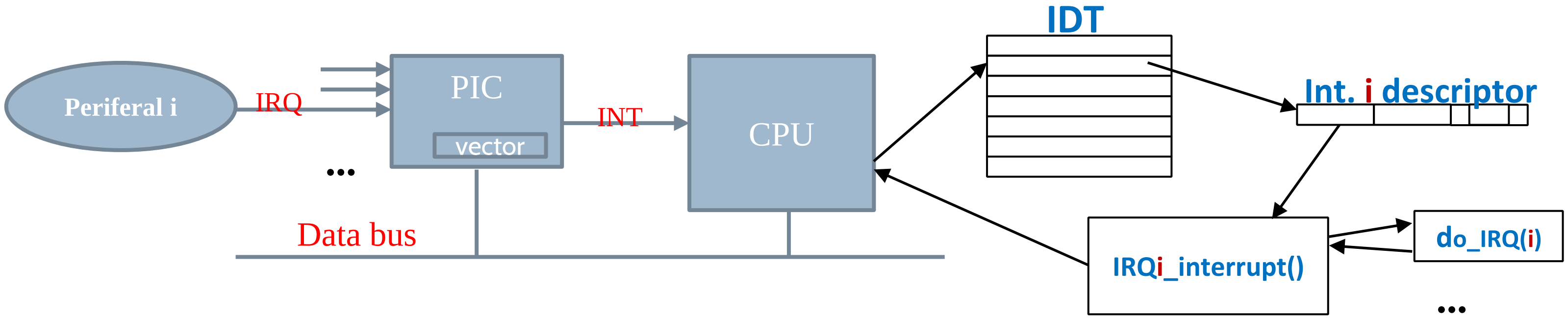


Hardware interrupts



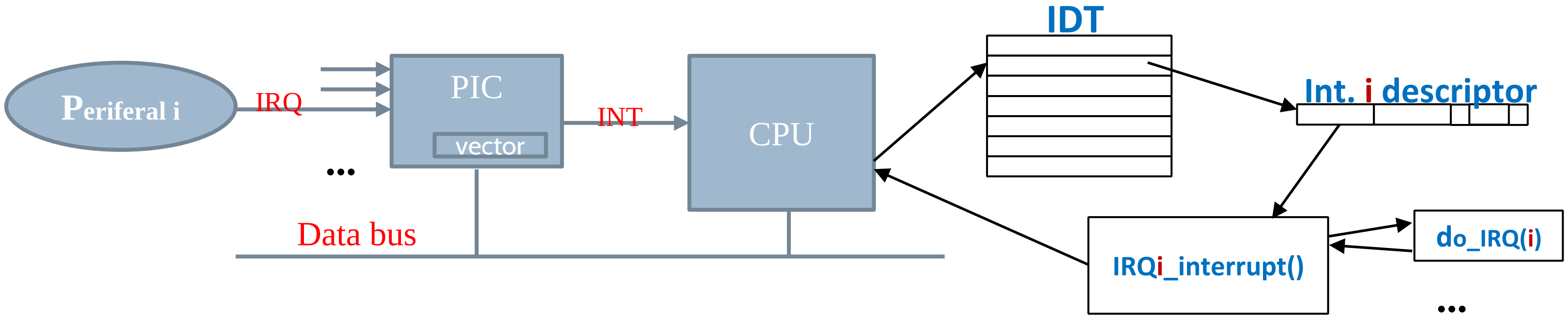
- Each periferal (able to generate an interrupt request) its associated to a given interrupt line or IRQ (*Interrupt ReQuest*)
- All lines are connected to a PIC (*Programmable Interrupt Controller*)
  - Currently, modern achitectures uses APIC (*Advanced Programmable Interrupt Controller*)
- The PIC is connected to the CPU by the pending interrupt line (INT)
- Both PIC and CPU are connected by data bus.

Hardware interrupts



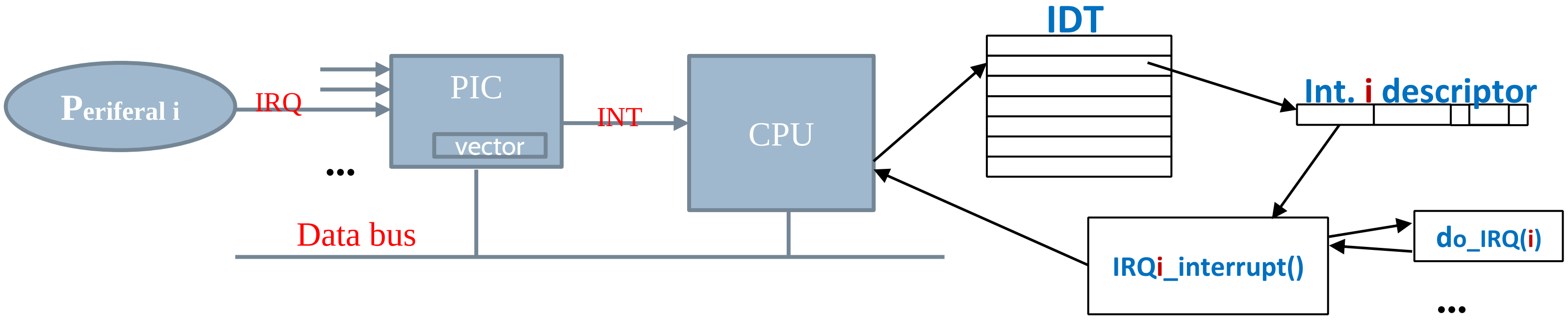
- PIC monitorizes the IRQ lines waiting for a signal.
- When a signal arrives:
  - Associate the corresponding IRQ to a value stored in a given PIC register (namely **vector**)
  - Signals the CPU through the pending interrupt line (INT)
  - The CPU read the vector register as an I/O port or memory address
  - The CPU writes in the PIC control register that it already accessed the vector
  - The PIC deactivate the pending interrupt line, clears the vector and start monitoring again...

Hardware interrupts



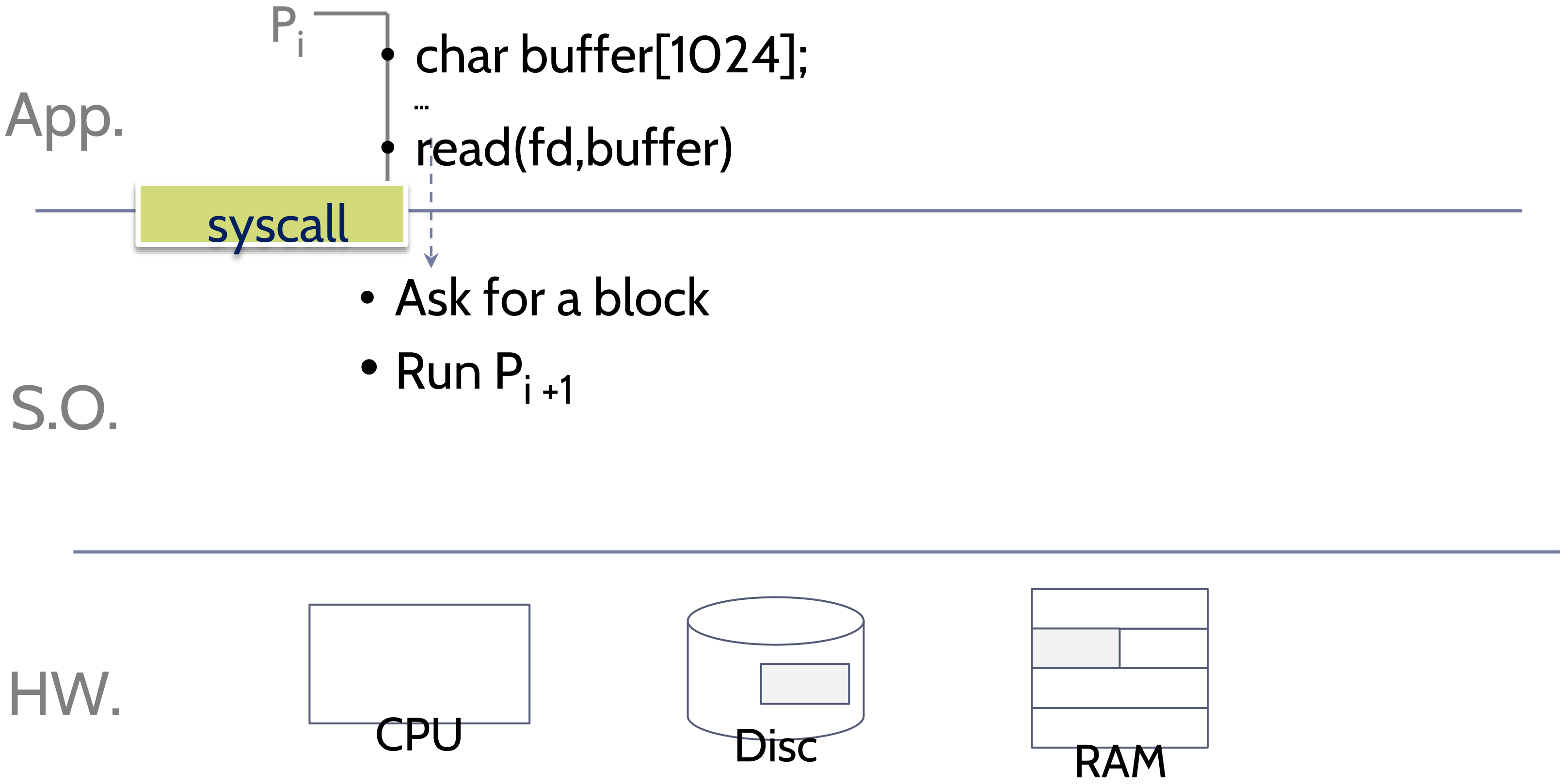
- PIC may allow disable the IRQ
  - In that case, the PIC does not signals the CPU of a given IRQ and are enqueued until they are enabled.
  - Disabling an interruption at the CPU level (mask/unmask) is different: the CPU ingnores the INT.
- Additionally, the PIC may have priority levels
  - Each IRQ is associated with a given priority level
  - If there are multiple IRQ, the PIC 'processes' those with the highest priority
  - If the PIC does not support priority level, it can be simulated by the operating system at software level.

Hardware interrupts



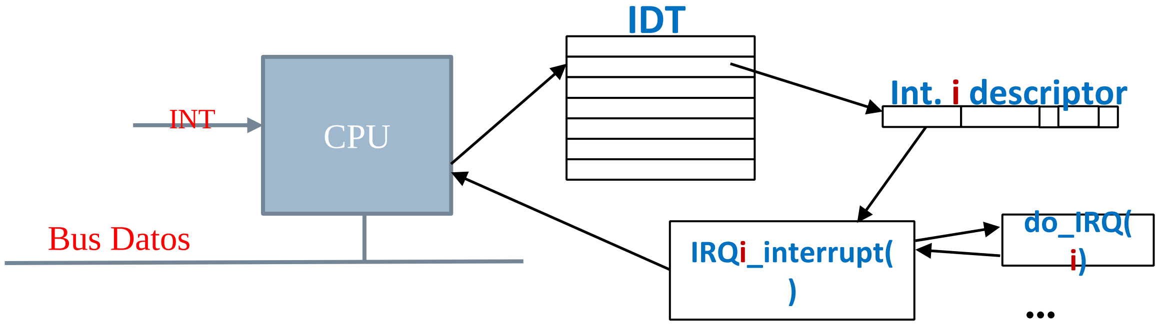
- The CPU receives the INT request
- Copies the vector through the data bus and notifies the PIC (ACK)
- Searches in the *Interrupt Descriptor Table* (IDT) for the associated function handler
- Stores the processor state at the stack, executes in privileged mode and runs the ISR
  - Multiple ISR (do_IRQ) may share the same interrupt
  - Multiple interrupt may share a generic handler function.
- Restore the state from the stack, and runs the RETI (goes to the previous mode and resume the execution)

System call



App.
P_i
char buffer[1024];
…
read(fd,buffer)
syscall
Ask for a block
Run P_{i+1}
S.O.
HW.
CPU
Disc
RAM

Llamada al sistema



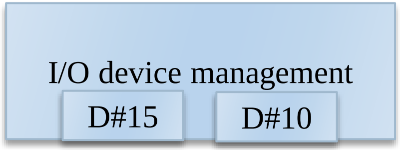
- There exists an assembly intruction to generate an interrupt by software
- Searches in the *Interrupt Descriptor Table* (IDT) for the associated function handler
- Stores the processor state at the stack, executes in privileged mode and runs the ISR
  - Multiple IST (do_IRQ) may share the same interrupt
  - Multiple interrupt may share a generic handler function.
- Restore the state from the stack, and runs the RETI (goes to the previous mode and resume the execution)

Contents

1. What an Operating System is.
   1. Definition, main functionalities and features

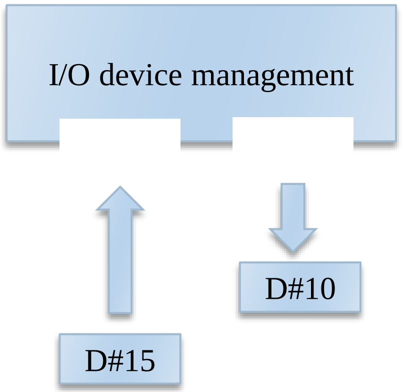
1. Operating system structure.
   1. Main goals, structure and asynchronous execution.
   2. **Kernel and modules**

Executables



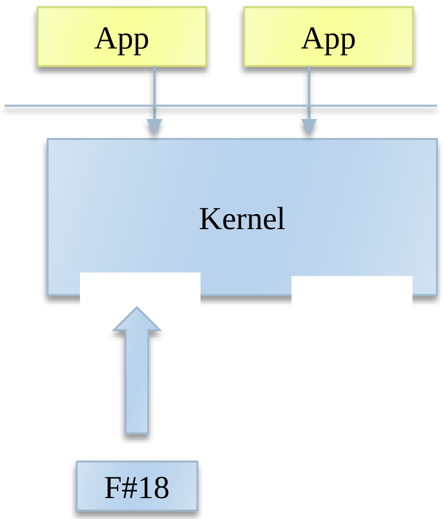
- Older kernels:
  - Included the code for all possible devices.
  - From time to time it was necessary to recompile the kernel to add support for new devices.
  - It was distributed as a set of **executables**.

Modules



- Were desinged to conditonally include device controllers (*drivers*)
  - They allow to dinamically include pre-compiled drivers.
  - Are distributed as **dynamic libraries** for the kernel (.so/.dll).
  - A given module may be downloaded when the device won't be used again.

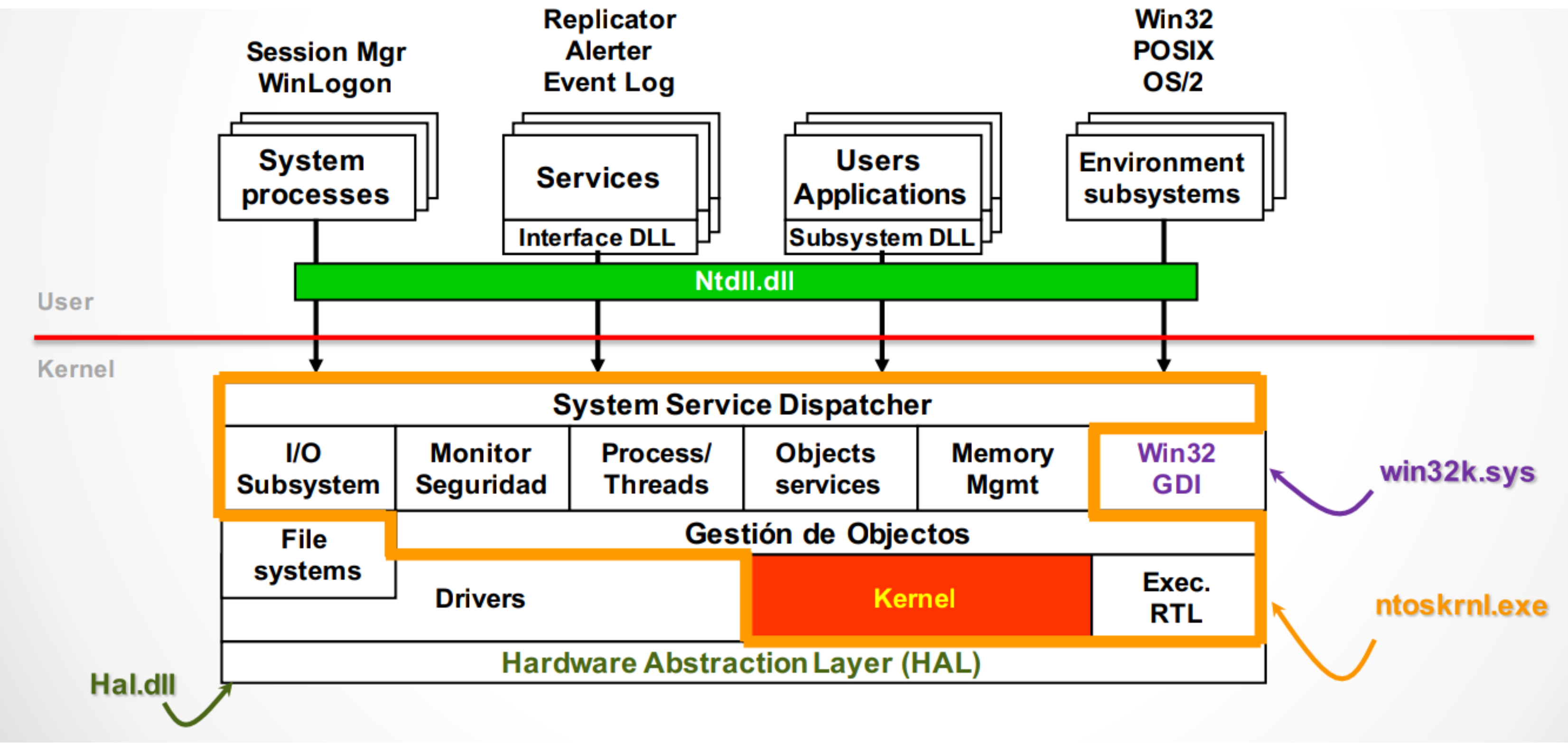
Módules



- Most of the current operating systems support modules:
  - Linux, Solaris, BSD, Windows, etc.
- Currently, the modules are not only used for drivers, but also to incorporate new functionalities:
  - Eg.: Linux kernel extensively uses modules for file systems, network protocols, system calls, etc.

http://www.cc.gatech.edu/~pwh/

Modules
Windows 2000



Grupo ARCOS
Universidad Carlos III de Madrid

Lesson 1
Introduction

Operating systems design
Degree in Computer Science and Engineering