ARCOS Group
Computer Science and Engineering Department
Universidad Carlos III de Madrid

Lesson 3a
process, devices, drivers, and extended services

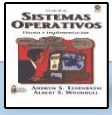
Operating System Design
Degree in Computer Science and Engineering, Double Degree CS&E + BA

Recommended readings

Base


1. Carretero 2007:
   1. Cap.7

Recommended



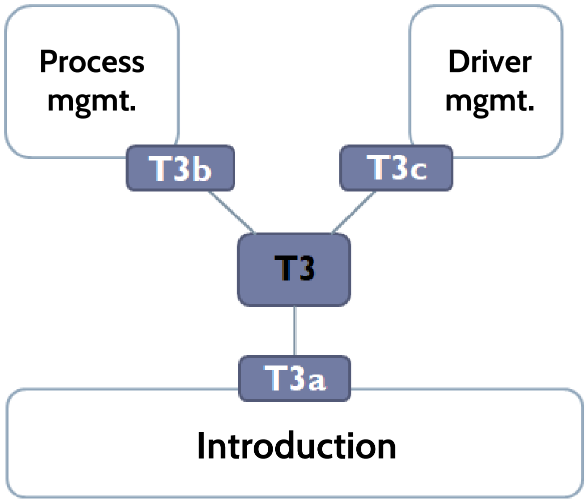
1. Tanenbaum 2006(en):
   1. Cap.3
1. Stallings 2005(en):
   1. Parte tres
1. Silberschatz 2006:
   1. Cap. Sistemas Module

To remember…

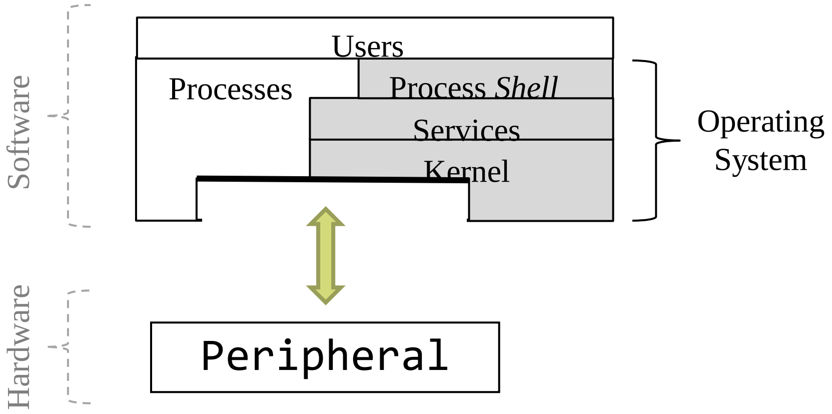
1. To prepare and review the class explanations.
   - Study the bibliography material: only slides are not enough.
   - Ask your doubts.

1. To exercise skills and abilities.
   - Solve as much exercises as possible.
   - Perform the guided laboratories progressively.
   - Build laboratories progressively.

General context…

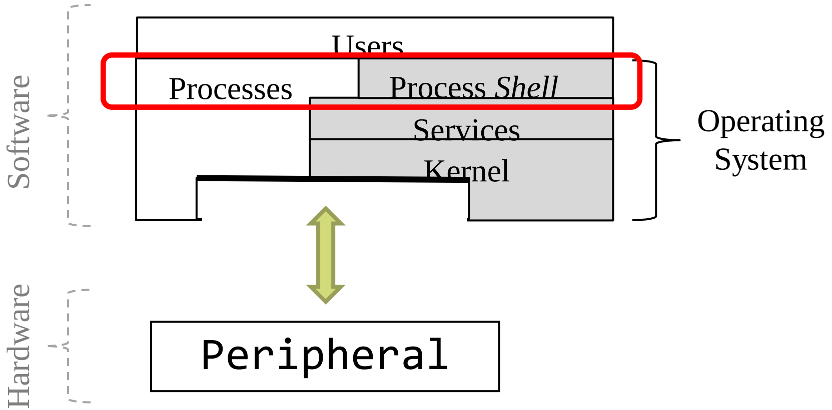


Overview



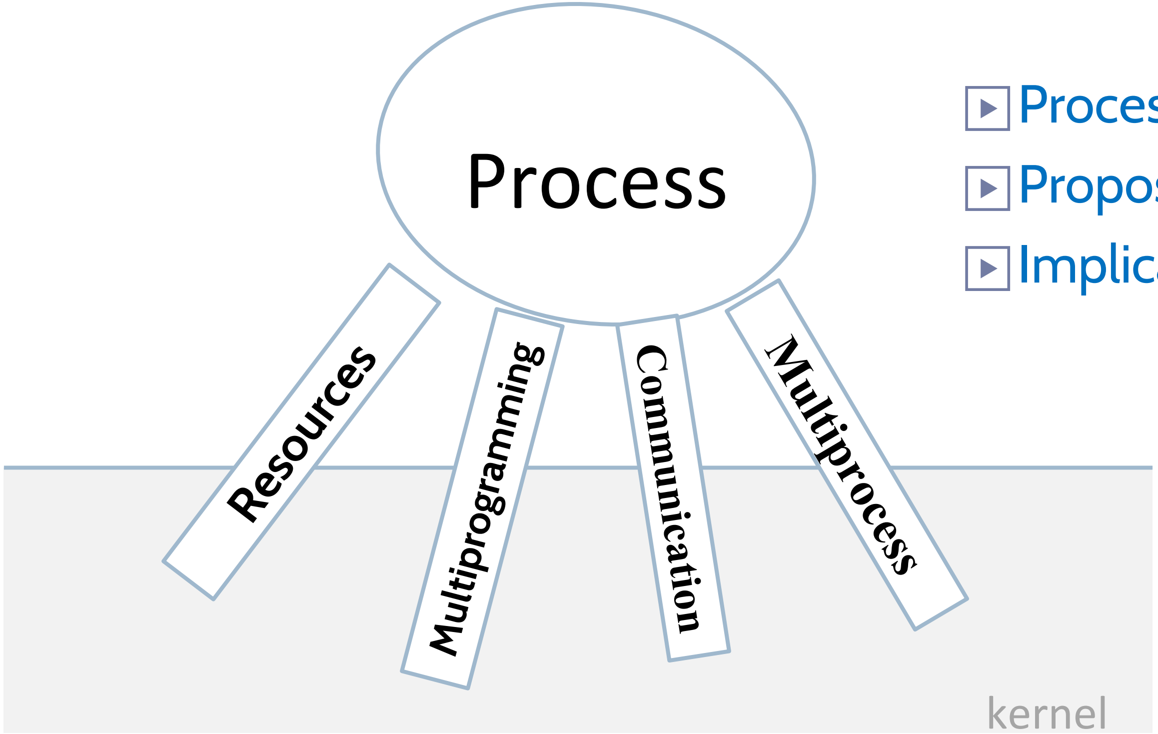
- Processes
- Peripheral

Overview



- **Processes**
- Peripheral

Introduction



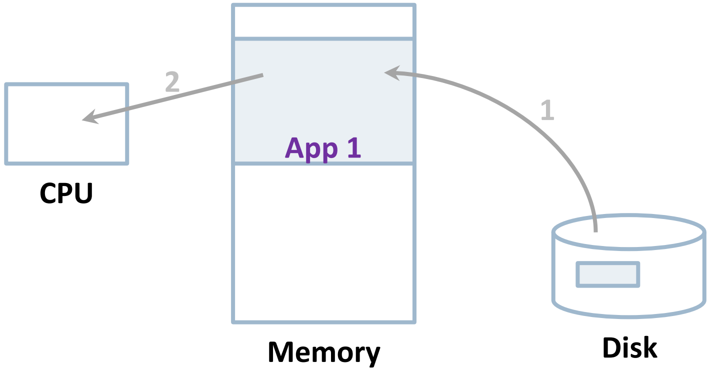
- Process concept
- Proposed model
- Implications in the O.S.

Introduction

Process

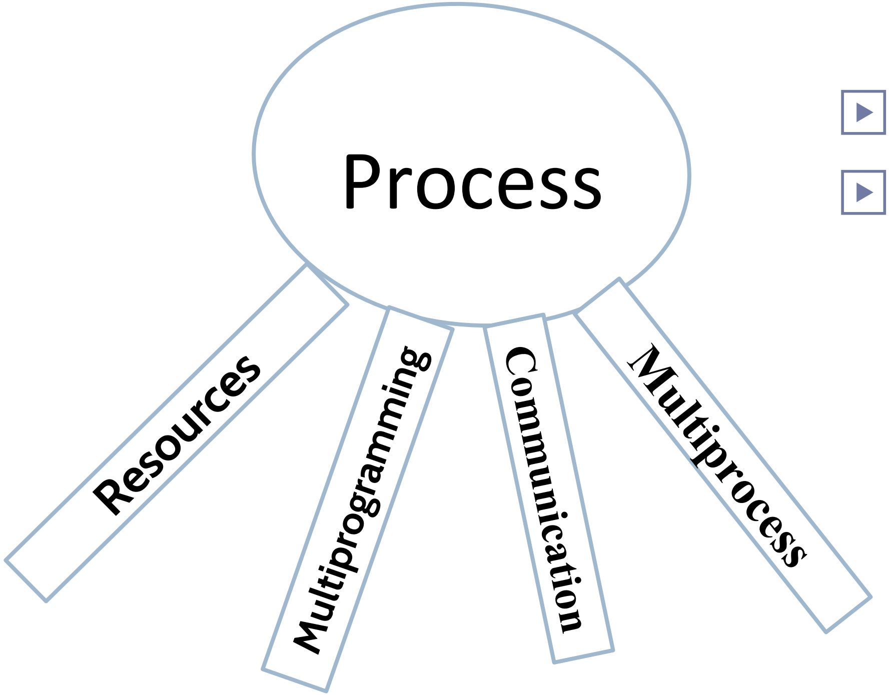
- **Process concept**

Process concept



- Process
  - Programm in execution
  - Processing unit managed by the Operating System (O.S.)

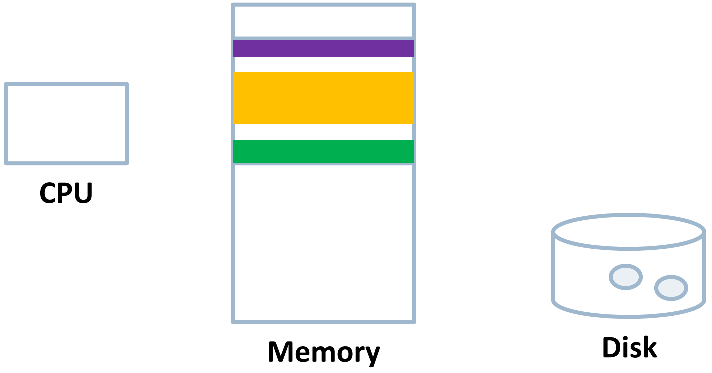
Introduction



- Process concept
- **Proposed model**

Proposed model

- **resource**
- multiprogramming
  - isolation/sharing
  - process hierarchy
- multitasking
- multiprocess

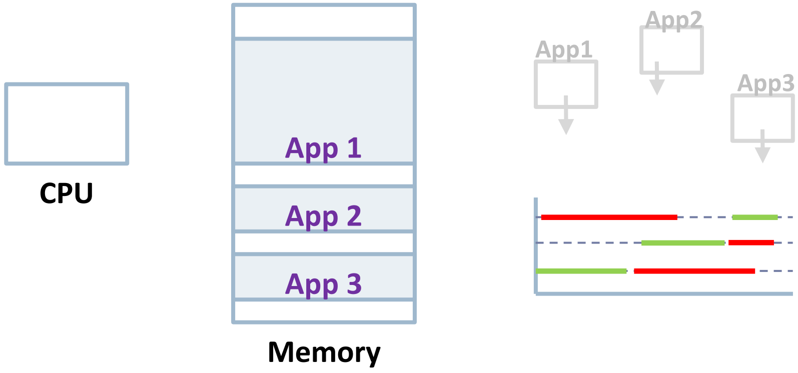


Associated resources

- Areas of memory
  - At least: code, data, and stack
- Open files
- Signals

Proposed model

- resource
- **multiprogramming**
  - isolation/sharing
  - process hierarchy
- multitasking
- multiprocess

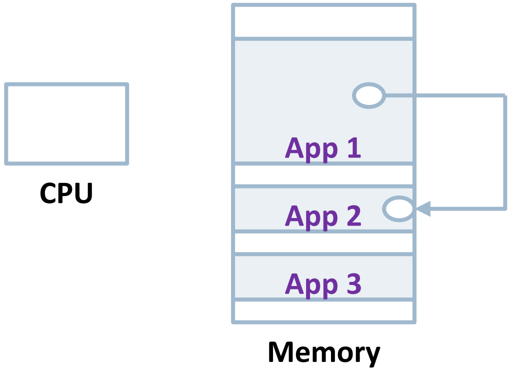


Multiprogramming

- Several applications loaded in main memory
- If one blocks because request some slow I/O then another is executed until this new one get blocket too
  - Voluntary Context Switching (V.C.S.)
- Efficiency in the use of the processor.
- Degree of multiprogramming = number of applications loaded in main memory

Proposed model

- resource
- multiprogramming
  - **isolation/sharing**
  - process hierarchy
- multitasking
- multiprocess

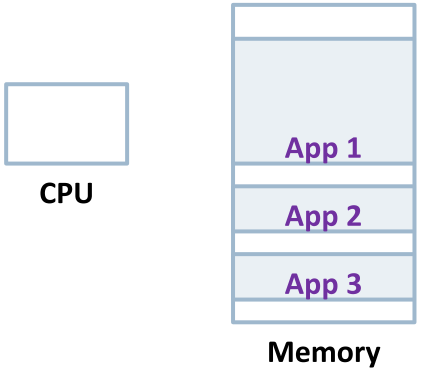


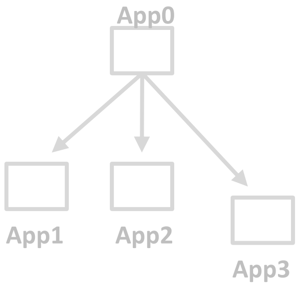
Isolation / Sharing

- Private address space per application, but
- Possibility of communicating data between two applications
  - Message passing
  - Sharing memory

Proposed model

- resource
- multiprogramming
  - isolation/sharing
  - **process hierarchy**
- multitasking
- multiprocess



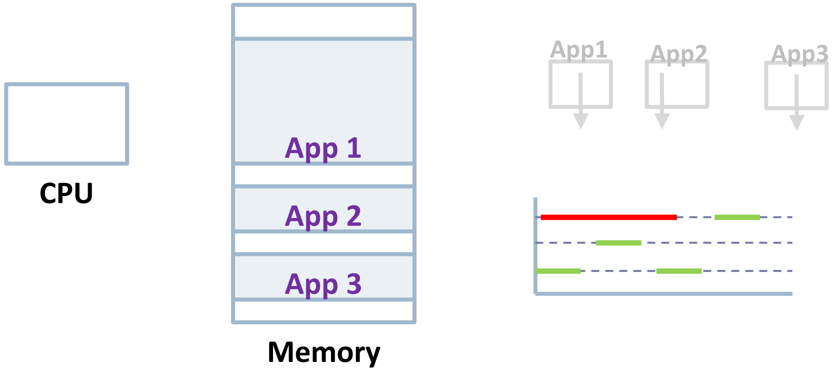


Process hierarchy

- Create process
  - As a copy of another existing process
  - From a application on disk
  - As boot process
- Group of processes that share the same treatment

Proposed model

- resource
- multiprogramming
  - isolation/sharing
  - process hierarchy
- **multitasking**
- multiprocess

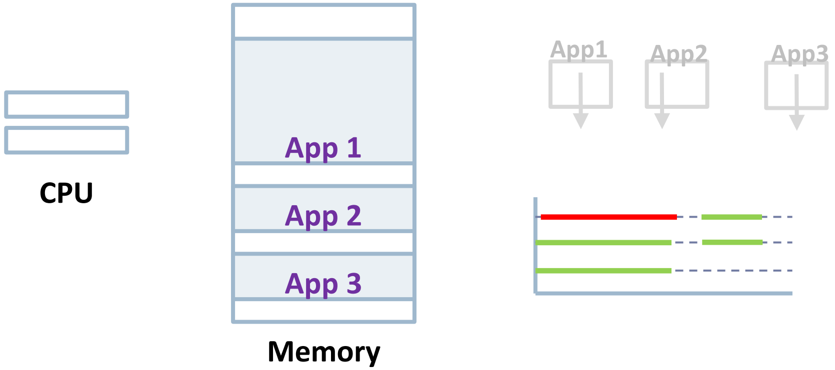


Multitasking

- Each process is executed a quantum of time (E.g .: 5 ms), and the turn is rotated to execute another ready processes
  - Involuntary Context Switching (I.C.S.)
- Sharing the use of the processor
  - It seems that everything is running at the same time

Proposed model

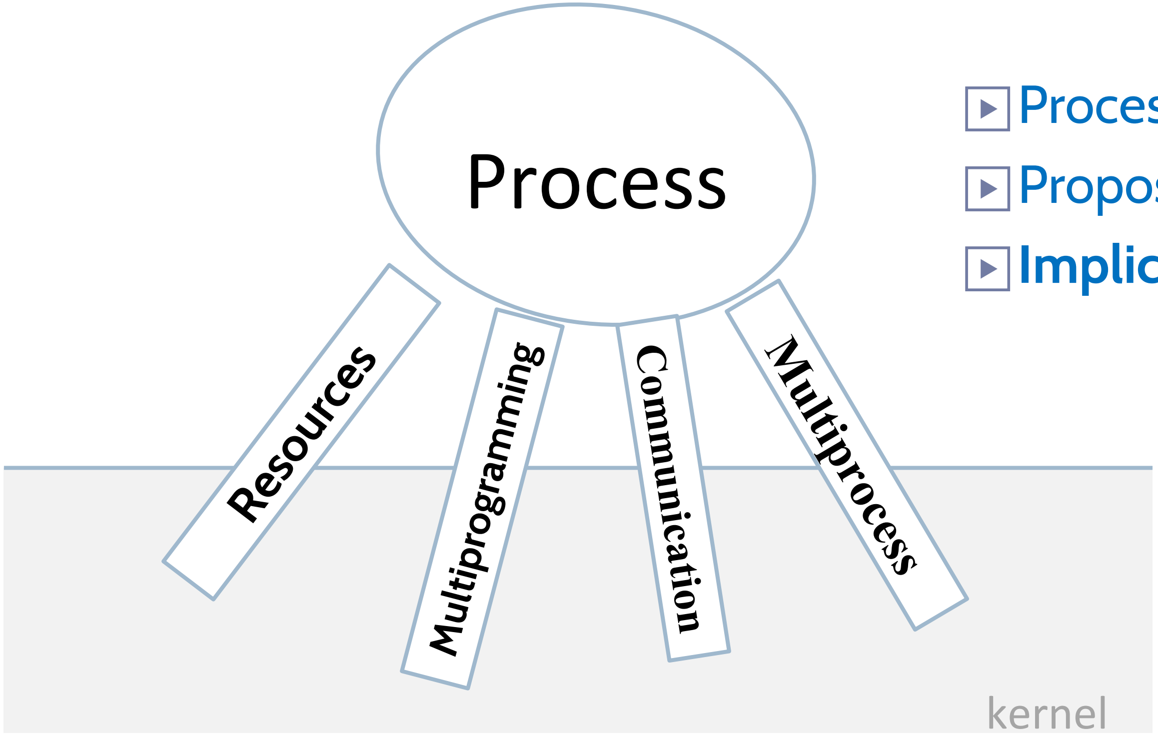
- resource
- multiprogramming
  - isolation/sharing
  - process hierarchy
- multitasking
- **multiprocess**



Multiprocess

- Several processors are available (multicore / multiprocessor)
- In addition to the distribution of each CPU (multitasking), there is real parallelism between several tasks (as many as processors)
  - It usually uses a scheduler and data structures per processor, with some load balancing mechanism

Introduction



- Process concept
- Proposed model
- **Implications in the O.S.**

Implications in the operating system

1. Data structures

Requirements	Information (in data structures)
Resources	• Areas of memory (code, data and stack) • Open files • Activated signals
Multiprogramming	• Execution state • Context: CPU registers… • Process list

o Insolation / Sharing	• Message passing
	    • Cola de mensajes de recepción
	• Memory compartida
	    • Zones, locks and conditions

o Hierarchy of processes	• Family relationship • Related sets of processes • Processes from the same session
Multitasking	• Quantum restante • Priority
Multiprocess	• Affinity

Implications in the operating system

1. Data structures

Implications in the operating system

2. Functions: internal management

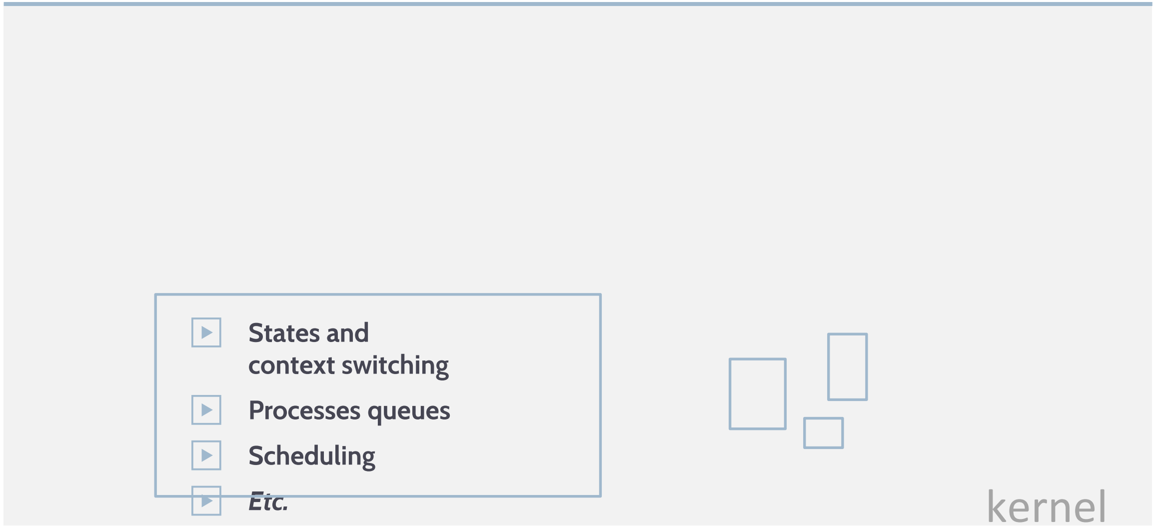
Requirements	Information (in data structures)	Functions (Internals, services, and API)
Resources	• Areas of memory (code, data and stack) • Open files • Activated signals	• Several internal functions • Several service function for memory, files, etc.
Multiprogramming	• Execution state • Context: CPU registers… • Process list	• Hw./Sw. int. from devices • Scheduler • Create/Destroy/Schedule process

○ Insolation / Sharing	• Message passing
	  • Cola de mensajes de recepción
	• Memory compartida
	  • Zones, locks and conditions
		• Send/Receive message and management of the message queue
		• API for concurrency control (access to data structures)

○ Hierarchy of processes	• Family relationship • Related sets of processes • Processes from the same session	• Clonar/Cambiar imagen de proceso • Associate process and leader selection
Multitasking	• Quantum restante • Priority	• Hw./Sw. int. from clock device • Scheduler • Create/Destroy/Schedule process
Multiprocess	• Affinity	• Hw./Sw. int. from clock device • Scheduler • Create/Destroy/Schedule process

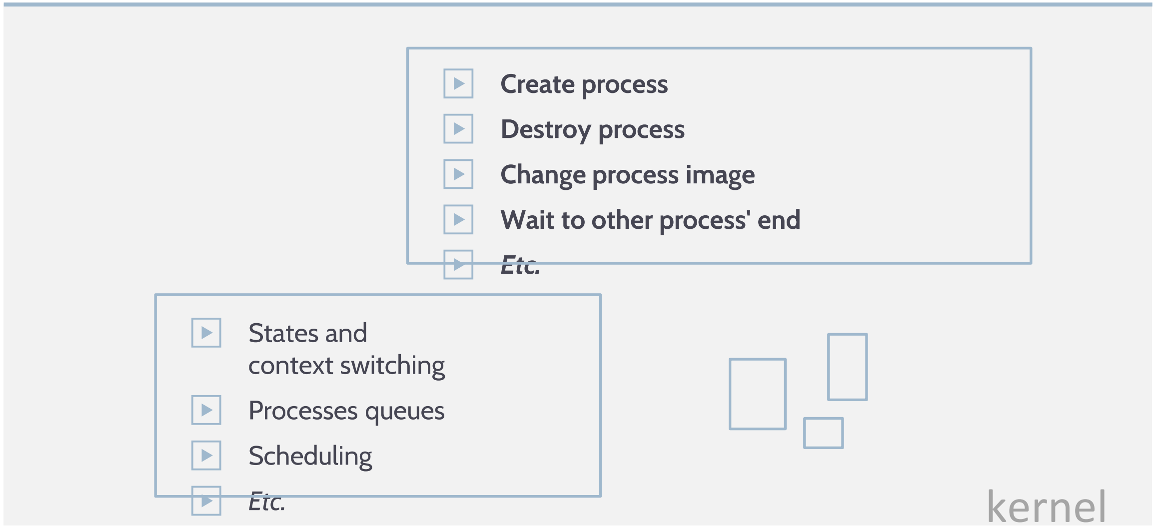
Implications in the operating system

2. Functions: internal management



Implications in the operating system

3. Functions: services



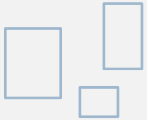
Implications in the operating system

3. Functions: service API

- fork, exit, exec, wait, ...
- *pthread_create, pthread...*

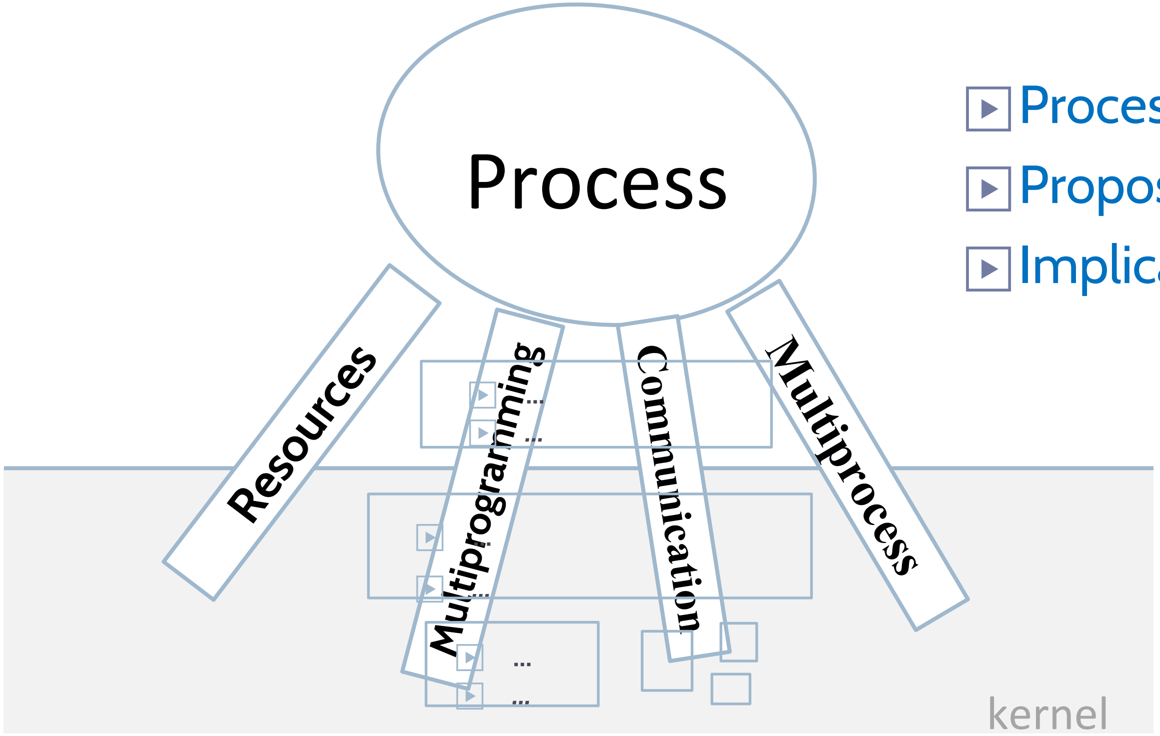
- Create process
- Destroy process
- Change process image
- Wait to other process' end
- *Etc.*

- States and context switching
- Processes queues
- Scheduling
- *Etc.*



kernel

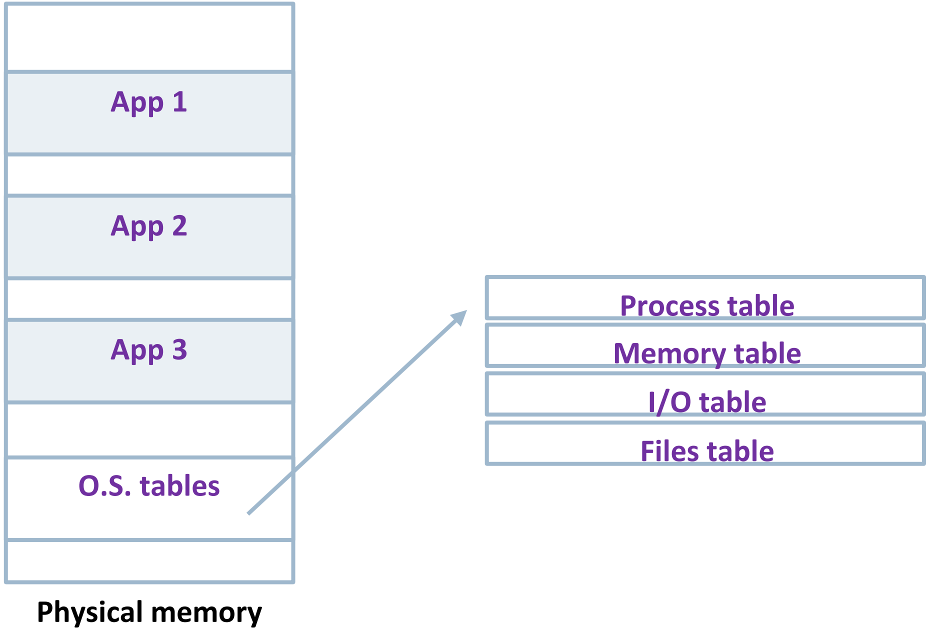
Introduction
summary



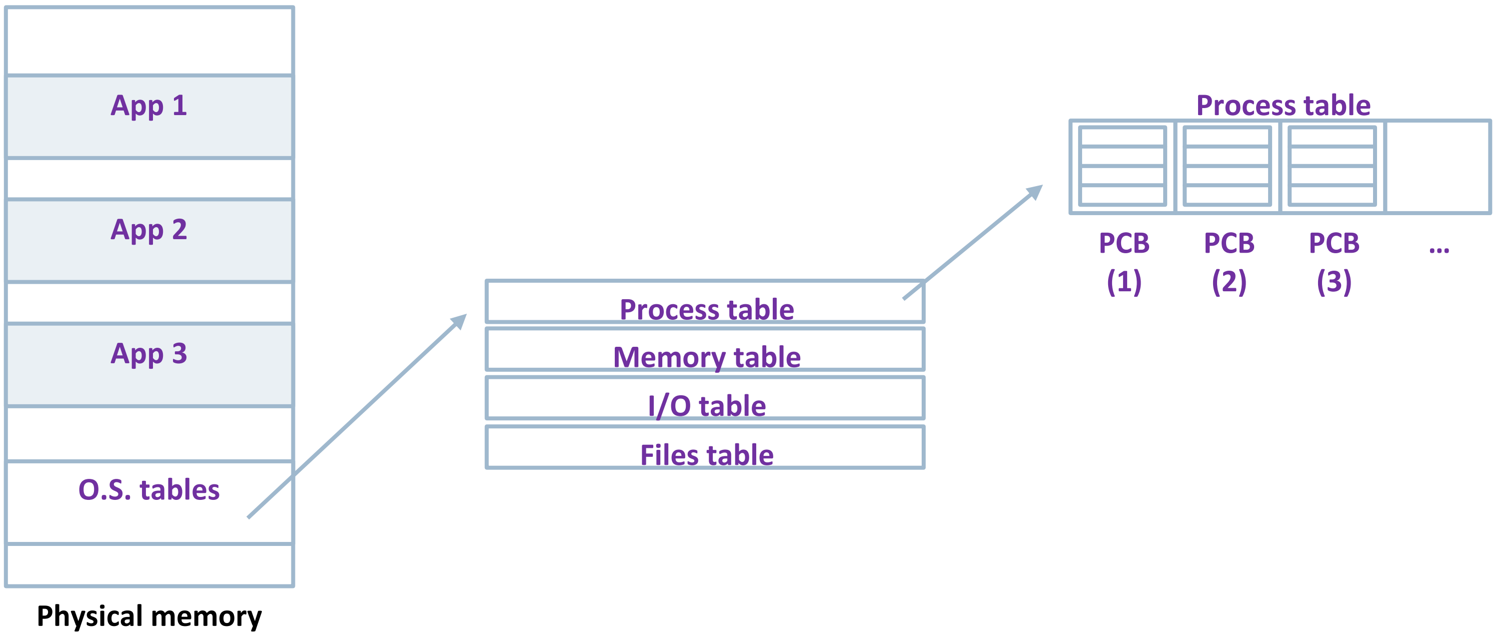
- Process concept
- Proposed model
- Implications in the O.S.

Main data structures

Information in the operating system



Information associated with a process



Alejandro Calderón Mateos

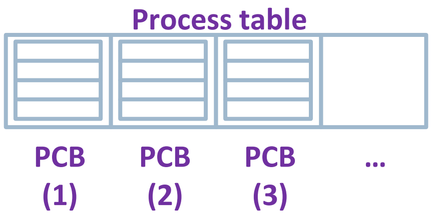
PCB: Process Table unit

- Process management

State
- General purpose registers
- Program counter
- State register
- Stack pointer

Id.
- Process identification
- Father process
- Process group

Mgmr.
- Priority
- Scheduler params
- Signals
- Timestamp when execution started
- Time of CPU used
- Time to next alarm



- *Process Control Block* (PCB / PCB)
  - Data structure with all related information needed for the management of a particular process
  - Manifestation of a process in the kernel
- *Thread Control Block* (TCB / BCT)
  - Similar to PCB for each thread in the process

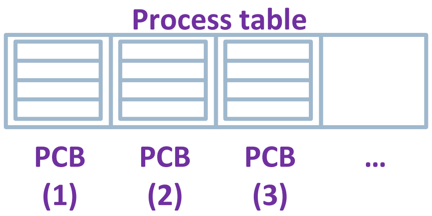
PCB: Process Table unit

- Process management

State
- General purpose registers
- Program counter
- State register
- Stack pointer

Id.
- Process identification
- Father process
- Process group

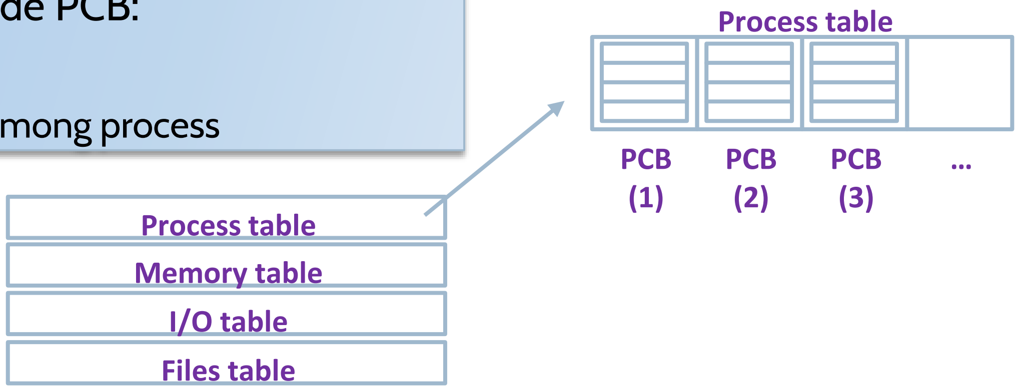
Mgmr.
- Priority
- Scheduler params
- Signals
- Timestamp when execution started
- Time of CPU used
- Time to next alarm



- *Process Identification* (PID)
  - Identification used by users
  - Use to be a positive number of 16 bits (32767) dynamically assigned, reused not immediately
- *Address of process descriptor* (APD)
  - Identification within the kernel
  - Use to be a translation PID -> APD (E.g.: hash)

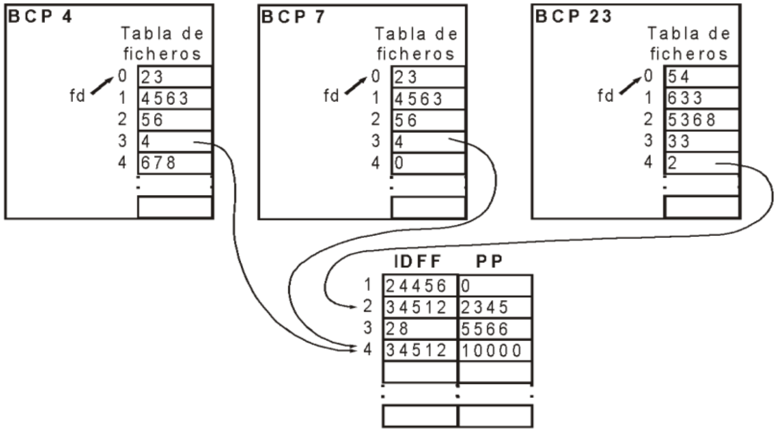
Where?: information of a process

- The information of a process in on its PCB…
- But some Information is outside PCB:
  - Because better efficiency
  - In order to share information among process



- Examples:
  - Table of memory segments and pages
  - Table of file placeholder
  - List of requests to devices

Where?: information of a process



- Table of file position pointer (seek pointer):
  - Describe the read/write position of open files.
  - In order to share the state of the file among process, this part has to be external to PCB.
  - The PCB contains the index of the element in the table that contains the information of the open file: the i-node, and the seek position.

Process information

summary

Tabla de procesos

- Process management
  - State
    - General purpose registers
    - Program counter
    - State register
    - Stack pointer
  - Id.
    - Process identification
    - Father process
    - Process group
  - Mgmr.
    - Priority
    - Scheduler params
    - Signals
    - Timestamp when execution started
    - Time of CPU used
    - Time to next alarm

PCB

Tabla de ficheros

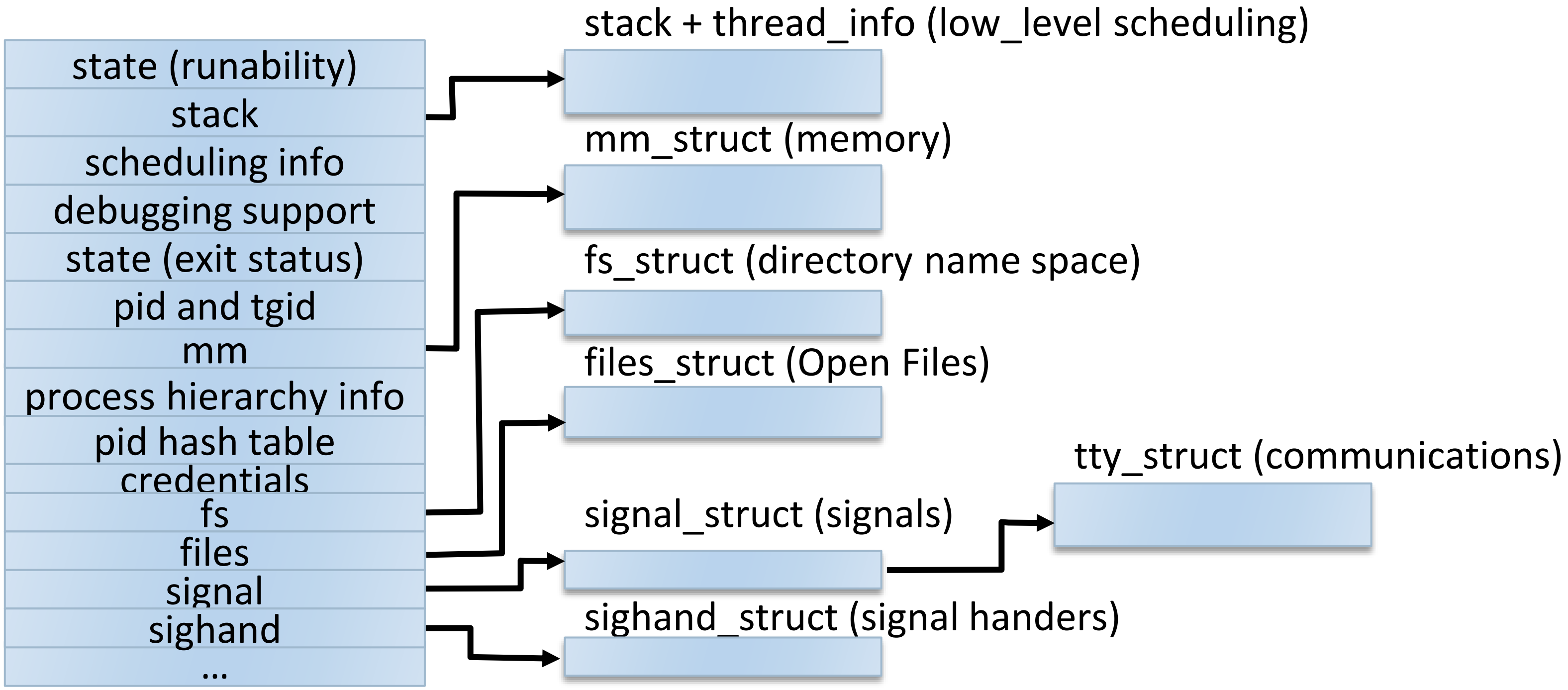
- File management
  - Root directory
  - Work directory
  - File descriptors
  - User identification
  - Group identification

Tabla de memoria

- Memory management
  - Pointer to the code segment
  - Pointer to the data segment
  - Pointer to the stack segment

Information of a process

Linux

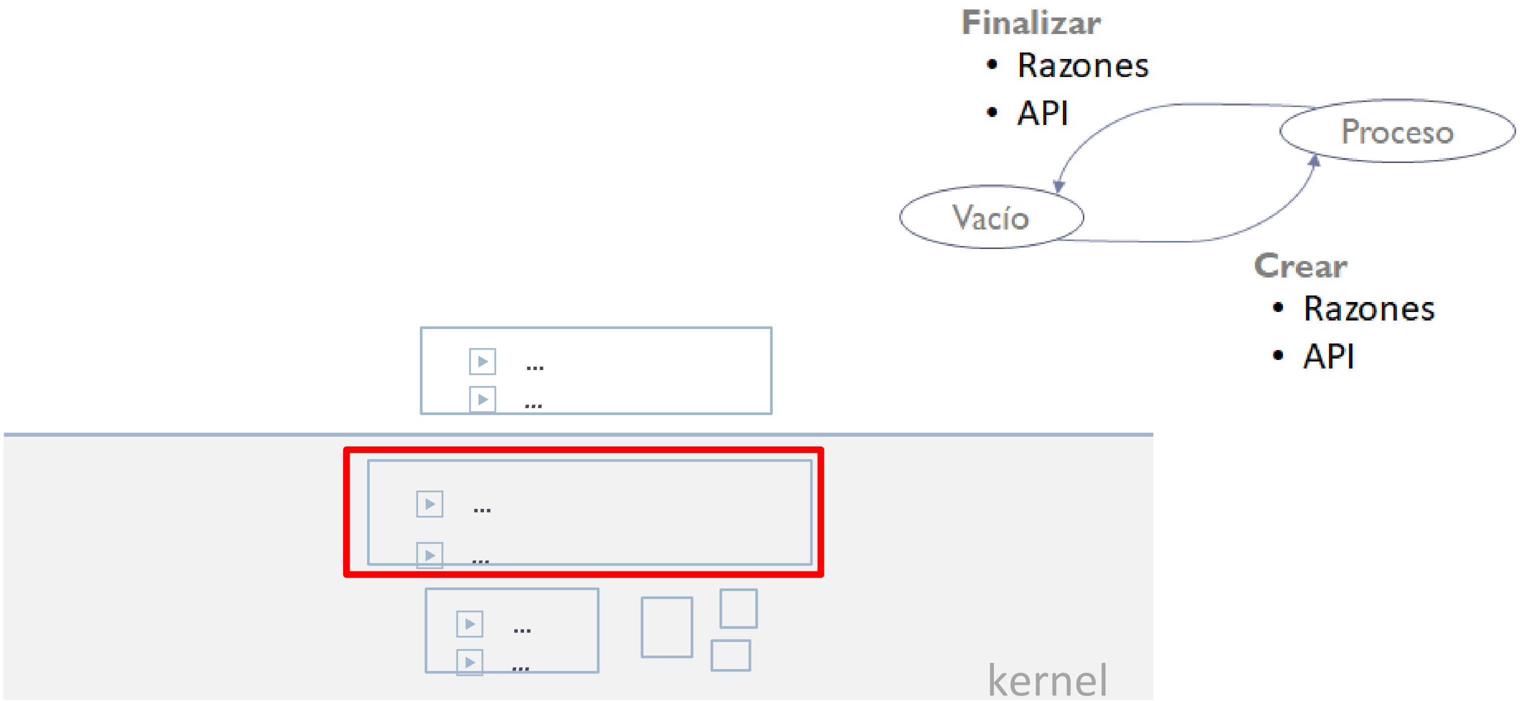


task_struct (Process Descriptor)

Understanding the Linux Kernel (O'Really)
http://www.eecs.harvard.edu/~cs161/assignments/sched.h.html

Operating System Services

Initialization and completion of processes.



Create process

- A process is created:
  - During system boot
    - Kernel threads + first process (E.g.: init, swapper, etc.)
  - When one process performs a system call to create another process:
    - When the operating system starts a new work
    - When an user starts a new application
    - When an running application needs a new process

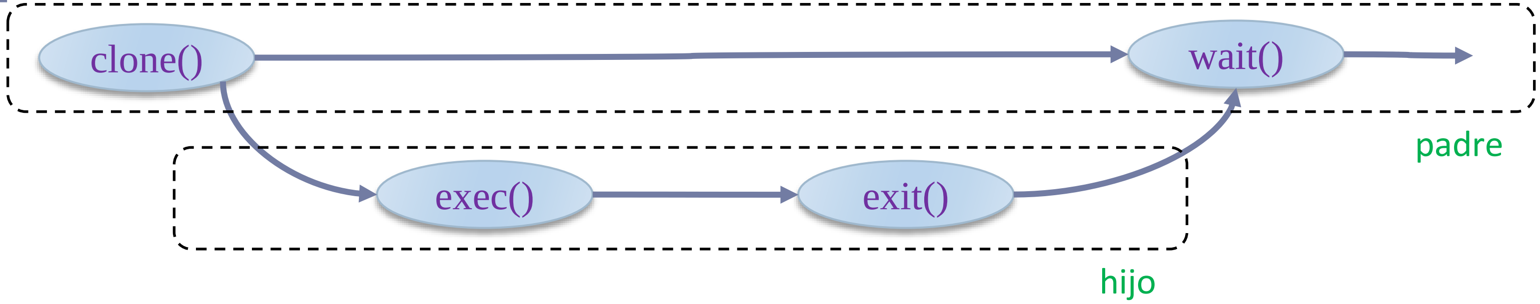
Destroy a process

- A process ends:
  - In a voluntary way:
    - Normal ending
    - Ending by error
  - In a non-voluntary way:
    - End by system (E.g.: exception, no available resources, etc.)
    - End by another process (E.g.: through a 'kill' system call)
    - End by user (E.g.: press Ctrl-C in the keyboard)

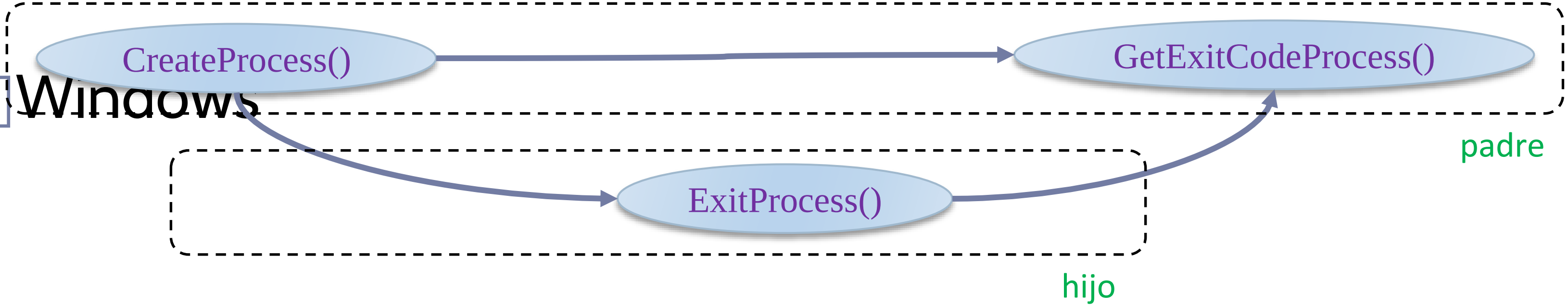
- In Unix/Linux signals are used as mechanism
- Signals can be captured and handled (but SIGKILL) to avoid some non-voluntary ways of ending

Creation and termination of processes
System calls

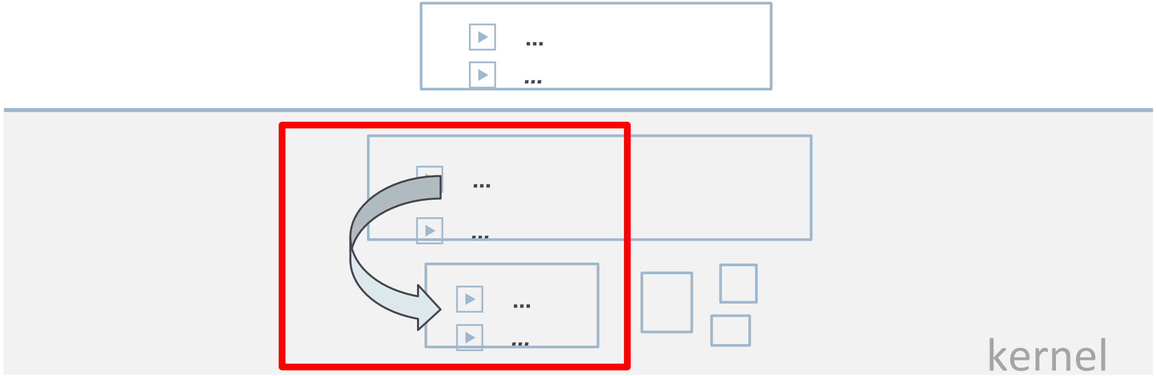
- Linux



- Windows



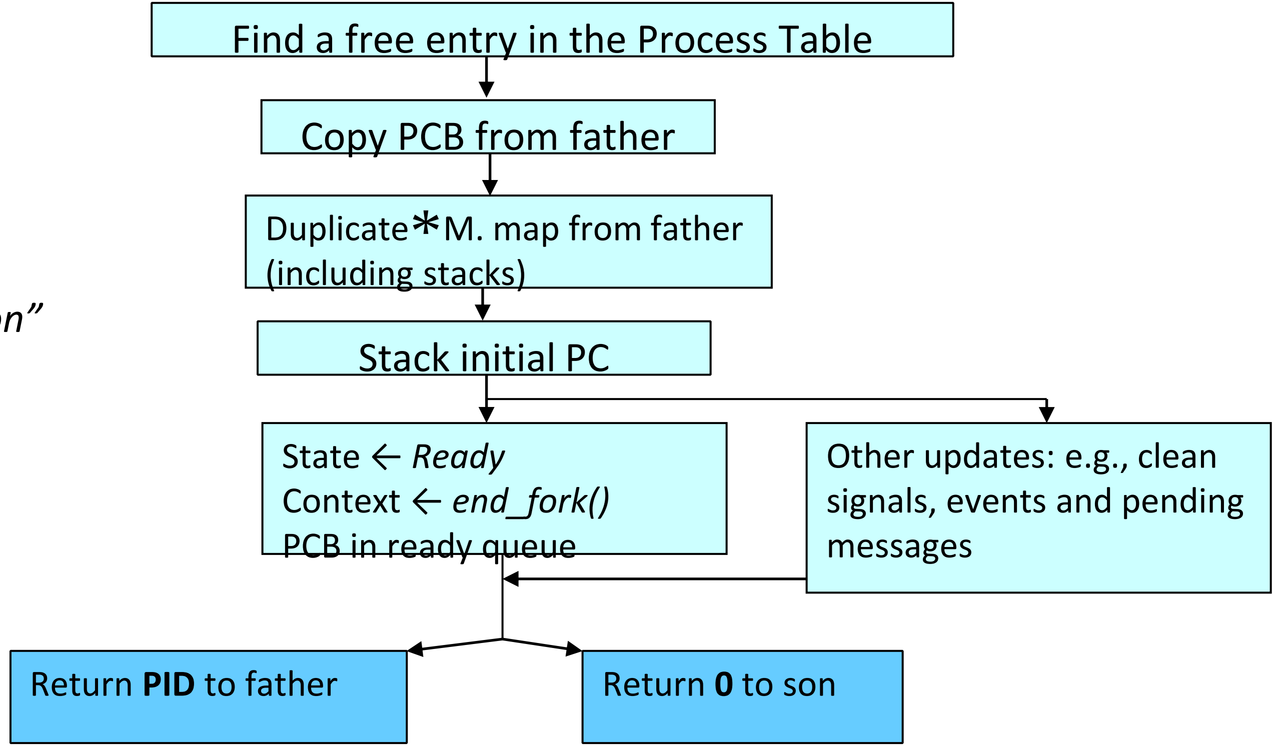
Operating System Services
Initialization and completion of processes.



Create process
Linux: clone

clone:

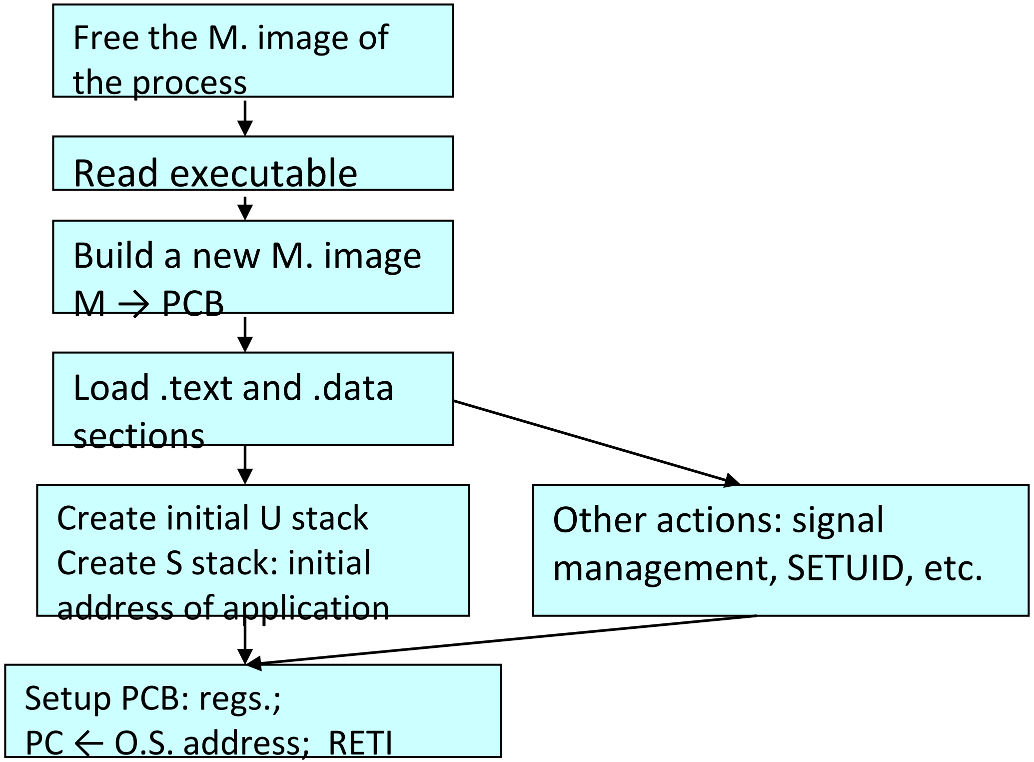
“Clone the father process and gives a new identity to the son”



Change process image
Linux: exec

exec:

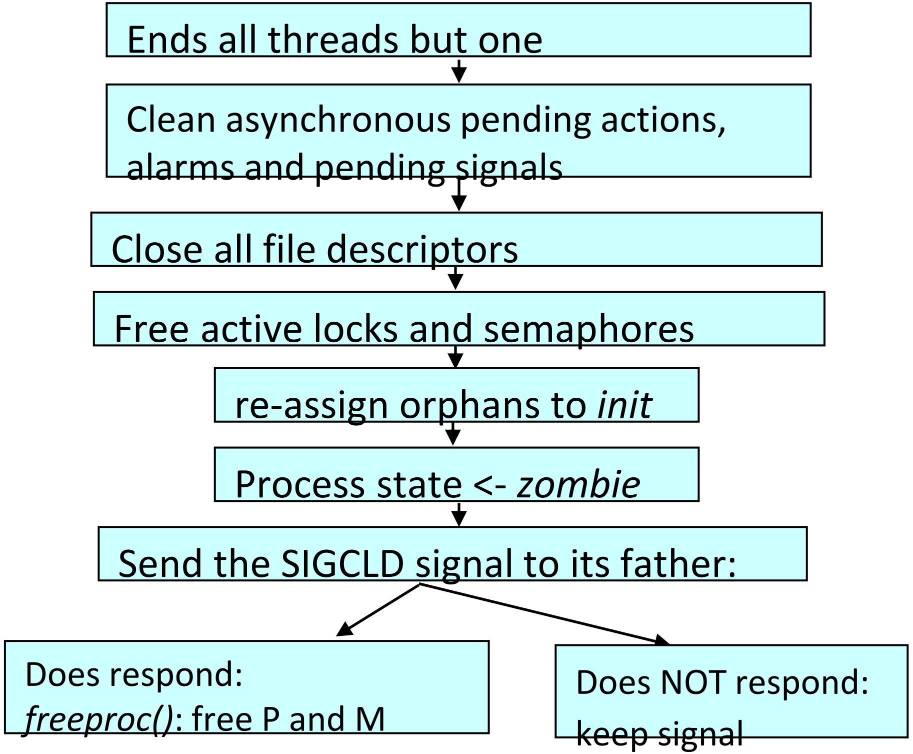
"Change the memory image of a process using as a previous one as 'container'"



Destroy process
Linux: exit

exit:

"It ends the execution of a process and release its associated resources"



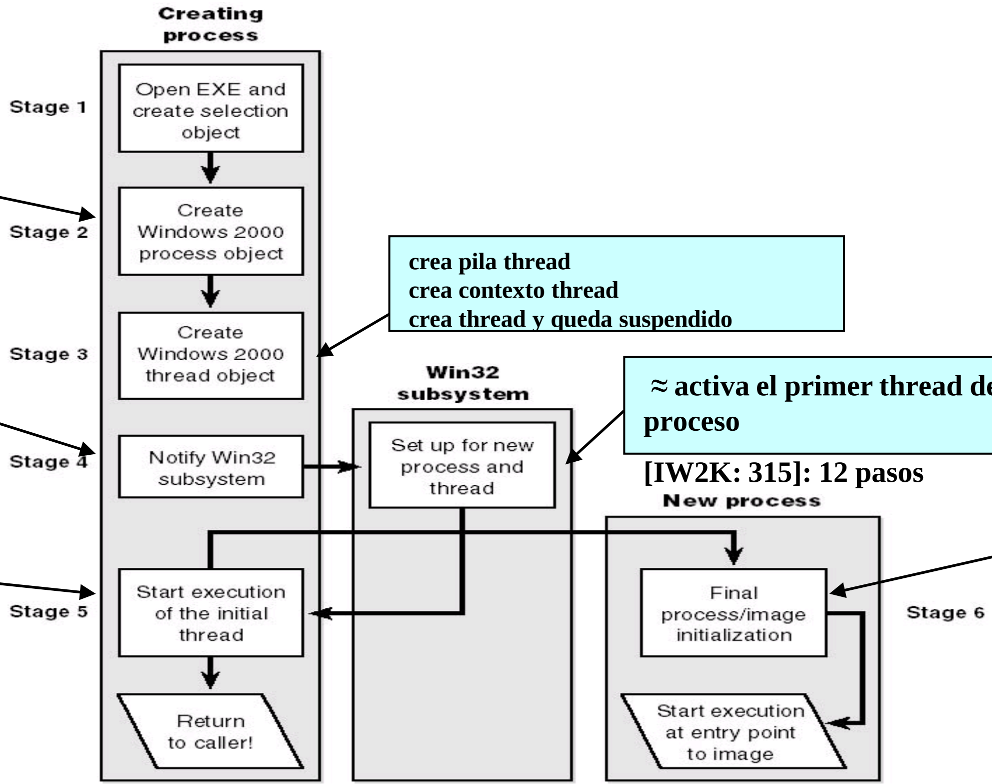
Create process

Windows: CreateProcess

crea EPROCESS
crea esp. M inicial proc.
crea KPROCESS
crea PEB

Kernel32.dll: mensaje a Win32: manejadores; flags; PID padre

thread inicial: comienza ejecución en contexto nuevo proceso



crea pila thread
crea contexto thread
crea thread y queda suspendido

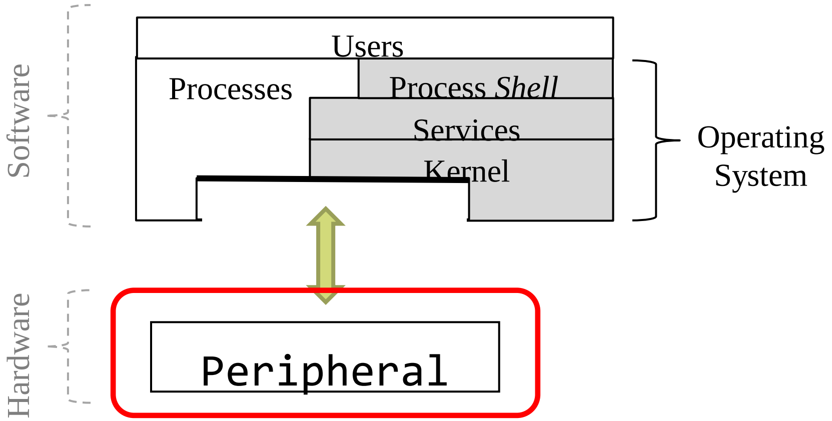
≈ activa el primer thread del proceso

[IW2K: 315]: 12 pasos

construye: contexto and stack inicial del thread:
-IRQL ← APC
-- encola APC: loader, heap manager, etc
--baja a IRQL ← 0: hace APC
--...
-- se pone en modo U

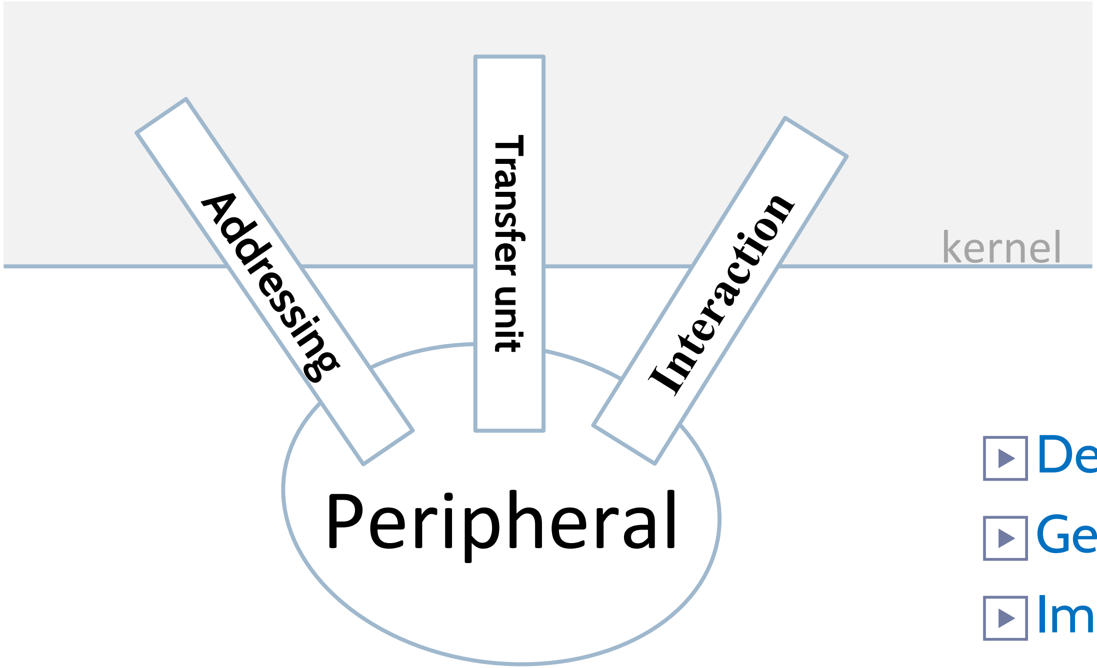
http://flylib.com/books/en/4.491.1.52/1/

Overview



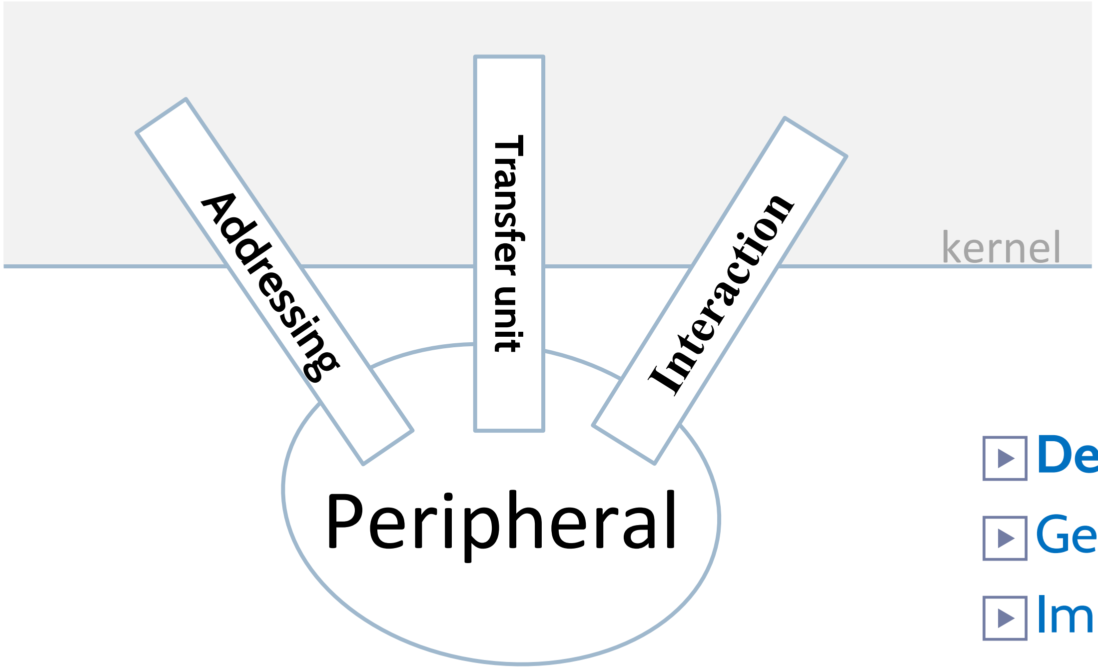
- Processes
- Peripheral

Introduction



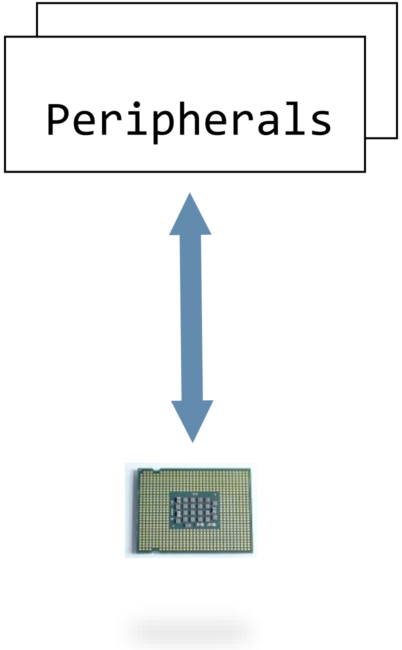
- Definition of Peripheral
- General structure
- Implications in the O.S.

Introduction



- **Definition of Peripheral**
- General structure
- Implications in the O.S.

Concept of peripheral

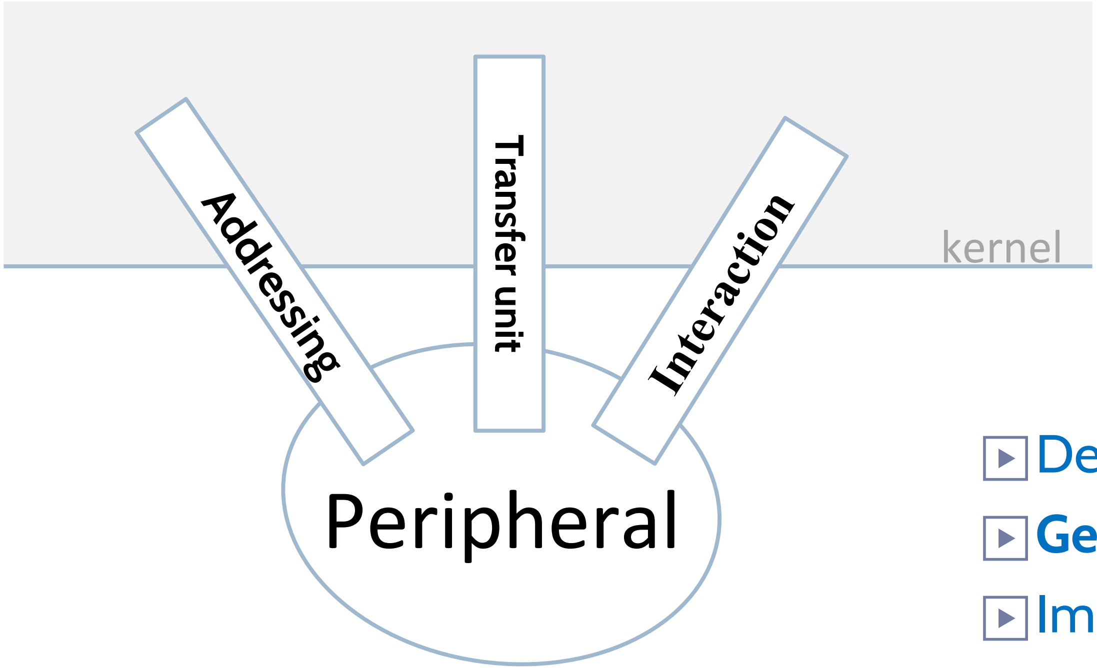


- **Peripheral:**
  - All external element connected to a CPU through the Input/Output (I/O) modules.
  - They let store information or communicate the computer with the exterior world.

Peripheral classification (by usage)

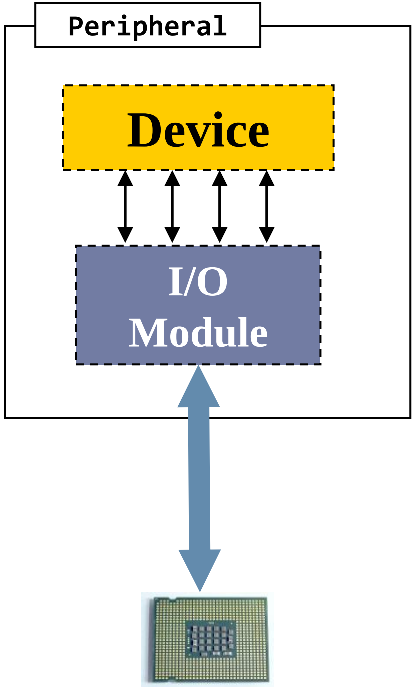
- **Communication:**
  - Human - machine
    - (Terminal) keyboard, mouse, …
    - (Printed) plotter, scanner, …
  - Machine - machine (Módem, …)
  - Physical environment - machine
    - (Read/accionamiento) x (analogic/digital)
- **Storage:**
  - Direct access (Disks, DVD, …)
  - Sequential access (Tapes)

Introduction



- Definition of Peripheral
- **General structure**
- Implications in the O.S.

General structure of a peripheral

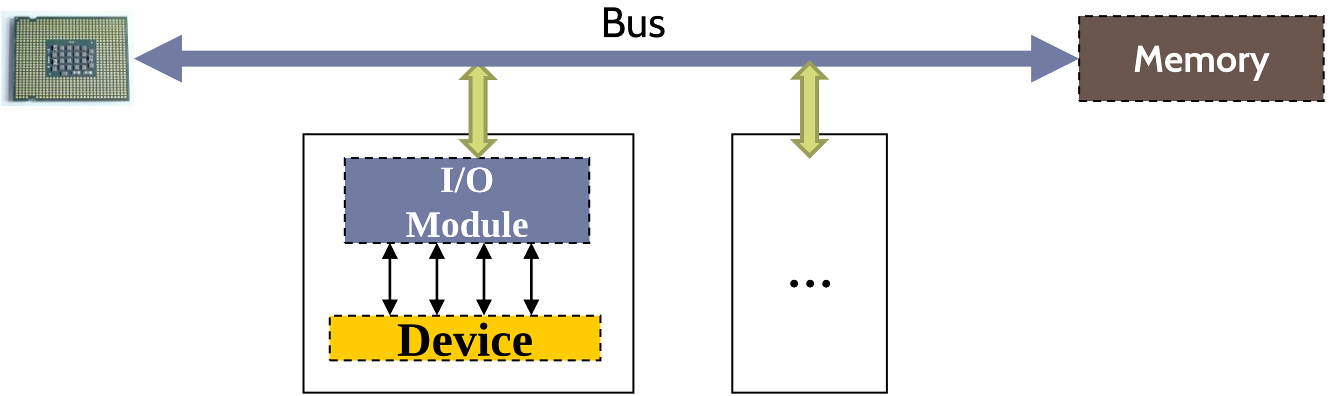


- Compound of:
  - Device
    - Hardware that interacts with the environment
  - I/O module
    - Also known as controller
    - Interface between the device and the CPU, which hides the particularities of this

Peripheral = Device + I/O module

I/O module
What are they

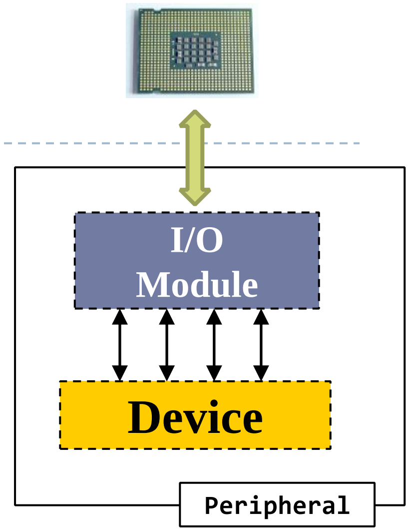
- The **I/O module** makes the connection between the CPU and the device.



I/O module

necessity

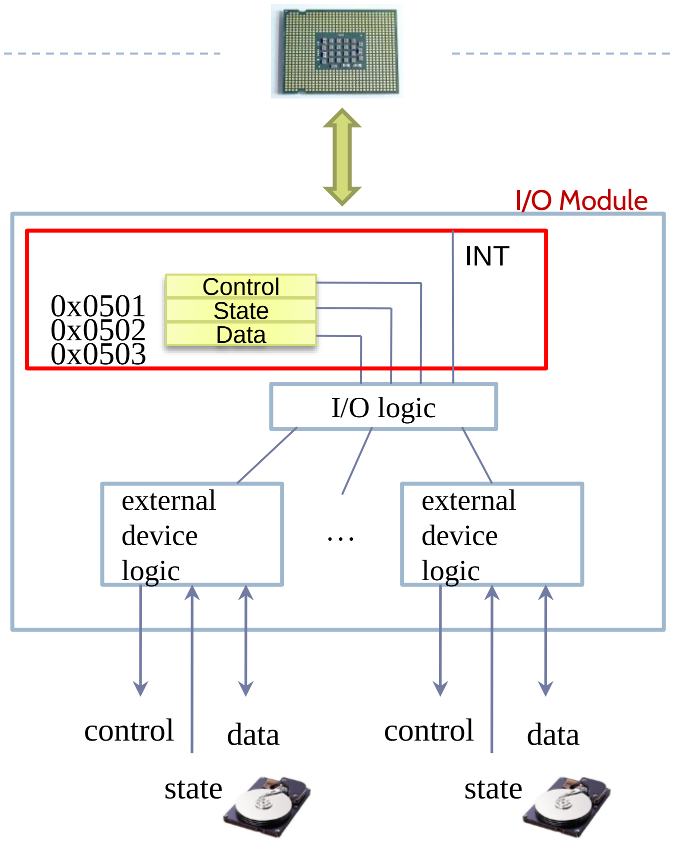
- There are necessary because:
  - Many types peripherals.
    - Peripherals use to be 'weird'
  - The data transfer speed of the peripherals use to be smaller than memory or processor (CPU).
    - Peripherals use to be 'slower'
  - Data formats and data sizes from the peripherals could be different to the ones used in the computer that are connected.



I/O module
structure: interface

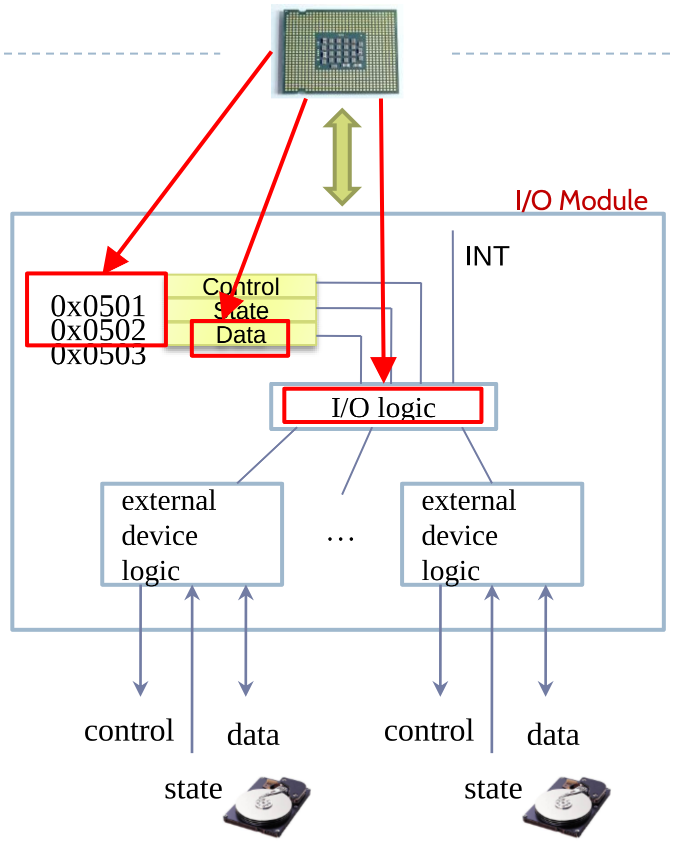
Interaction between CPU and I/O Module through:

- 3 **types** of registers:
  - Control register
    - Request for the peripheral
  - State register
    - Result of the last request performed
  - Data register
    - Interchange data between CPU/peripheral
- 1 **type** of interrupt line:
  - Notification interrupt

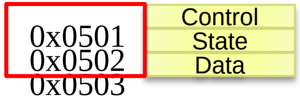


I/O module
characteristics to know

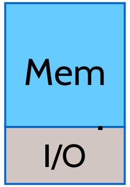
- Important aspects:
  - Addressing:
    - Memory-mapped, Port-mapped
  - Transfer unit:
    - Character, block
  - Interaction computador-controlador:
    - Direct, Interrupted, DMA



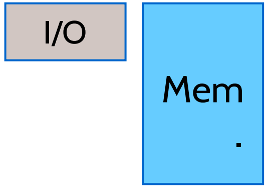
(1/3) Addressing Module



- Memory-mapped I/O
  - The I/O module registers are 'projected' into the main memory space and a memory area is used to associate address to I/O module + register of this module.
  - E.g.: int * rctrl = 0x105A ;
    (*rctl) = 1 ;



- Port-mapped I/O
  - With special assembler instructions (In / Out) you access the I/O module registers as special addresses (called ports).
  - E.g.: out(0x105A, 1) ;

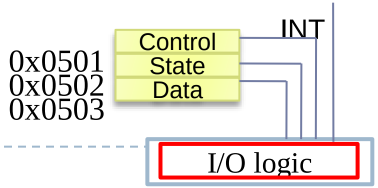


(2/3) Transfer unit

- **Block device:**
  - Unit: blocks of bytes
  - Access: sequential or direct
  - Actions: read, write, situarse, ...
  - Examples: tapes and disk

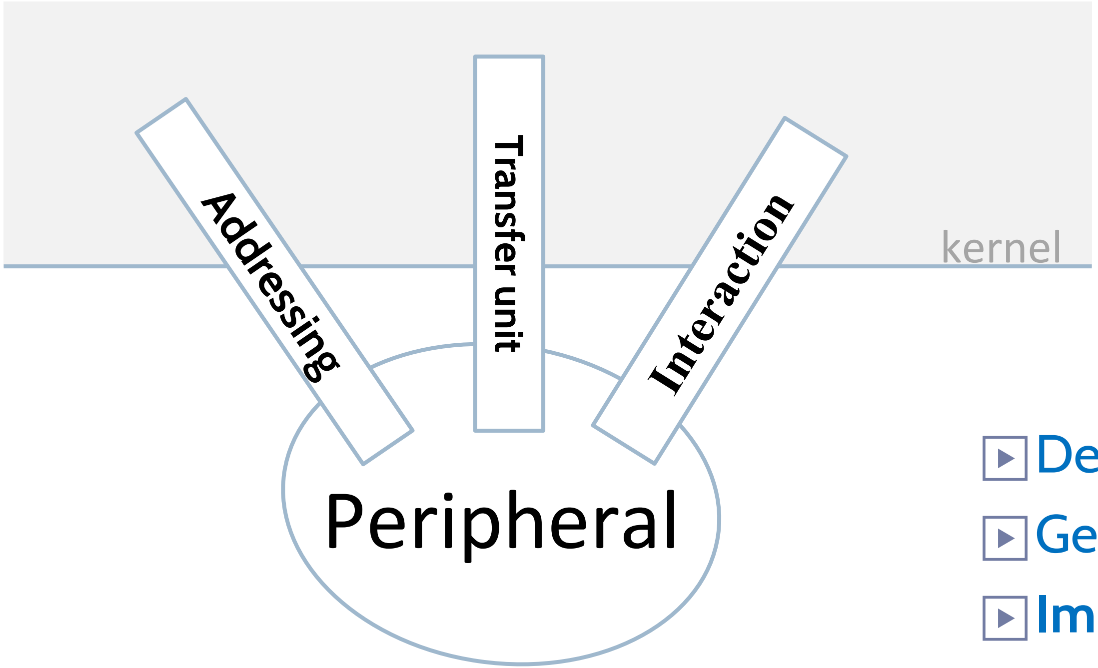
- **Character device:**
  - Unit: characters (ASCII, Unicode, etc.)
  - Access: sequential
  - Actions: `get`, `put`, ...
  - Example: terminals, printers, etc.

(3/3) Interaction with computer



- **Direct I/O**
  - CPU does all I/O: busy wait → transfer (*'polling'*)
- **Interrupted I/O**
  - CPU does not wait, only transfer data
- **DMA I/O (direct memory access)**
  - CPU neither wait, nor transfer, it is notified at the end of data transfers
    - I/O module is more sophisticated (cost more, better performance)
    - Try to reduce the overheat when transfering blocks of data

Introduction



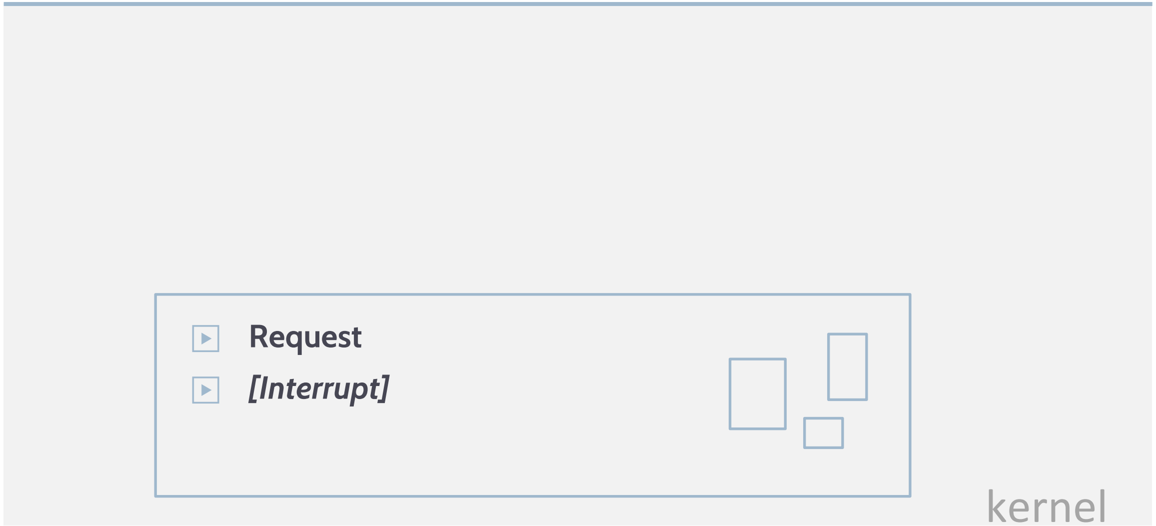
- Definition of Peripheral
- General structure
- **Implications in the O.S.**

Implications in the operating system

1. Data structures

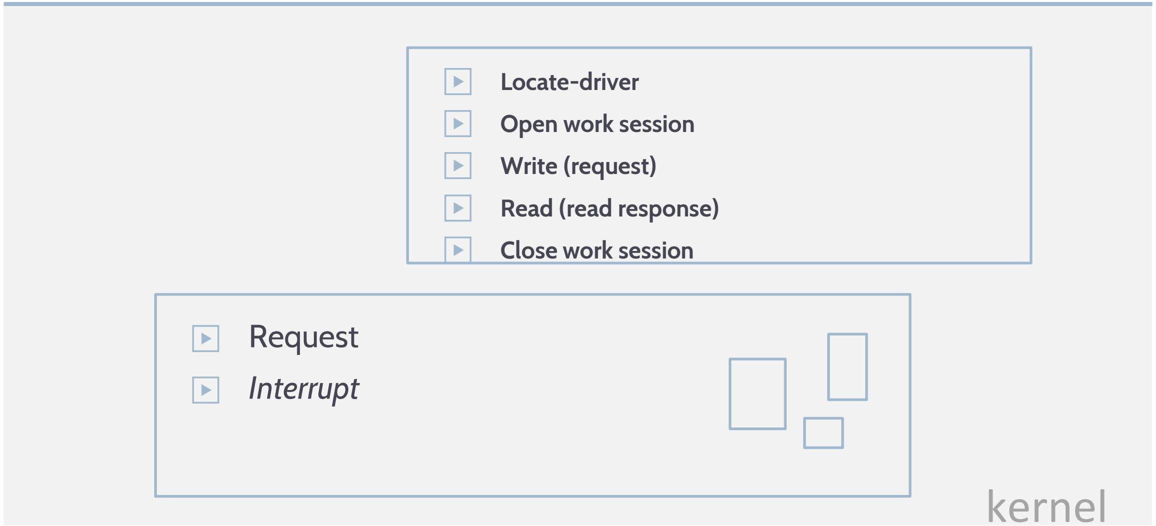
Implications in the operating system

2. Functions: internal management



Implications in the operating system

3. Functions: services



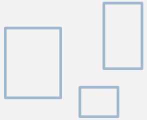
Implications in the operating system

3. **Functions: service API**

- ...

- Locate-driver
- Open work session
- Write (request)
- Read (read response)
- Close work session

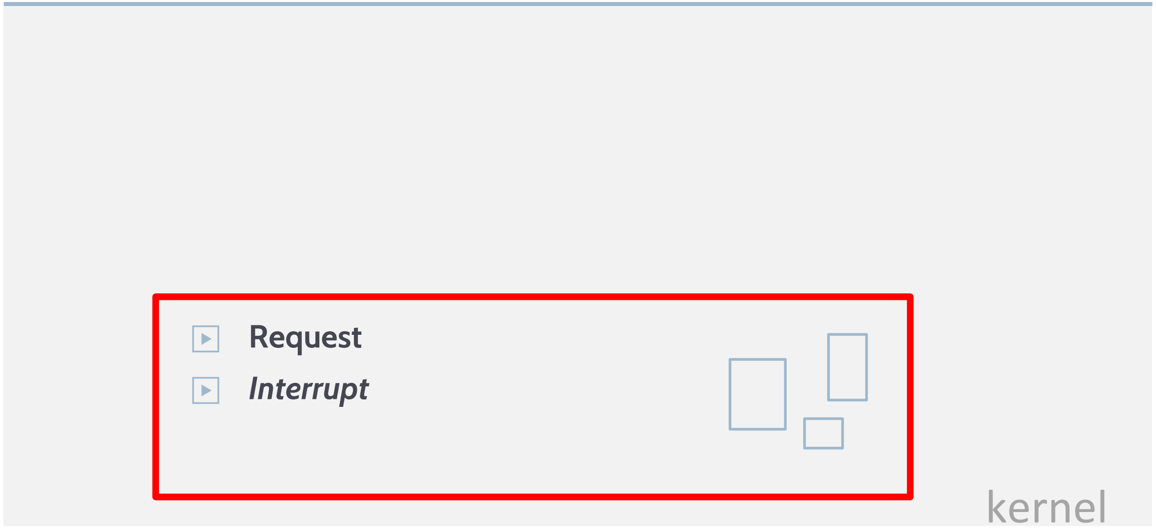
- Request
- *Interrupt*



kernel

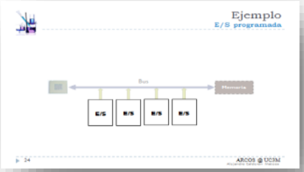
Implications in the operating system

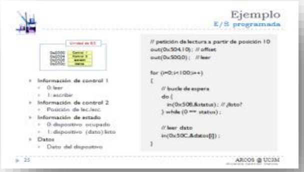
(1 + 2) Data structures + internal mgmt. functions = driver

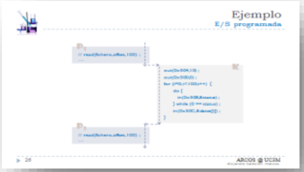


Impact in the Operating System of the device handling

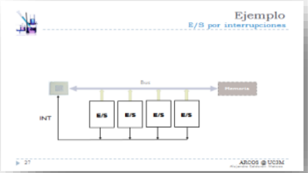
- **Direct I/O**

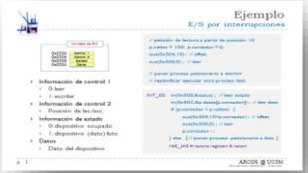


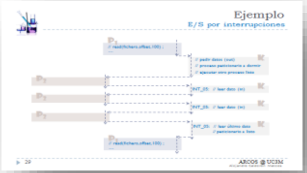




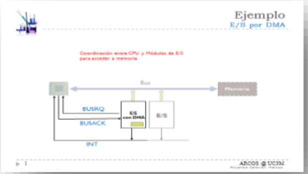
- **Interrupted I/O**

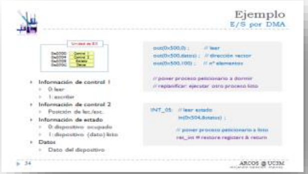


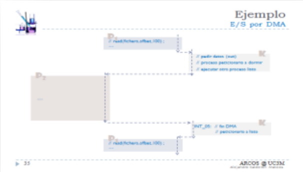




- **DMA I/O**

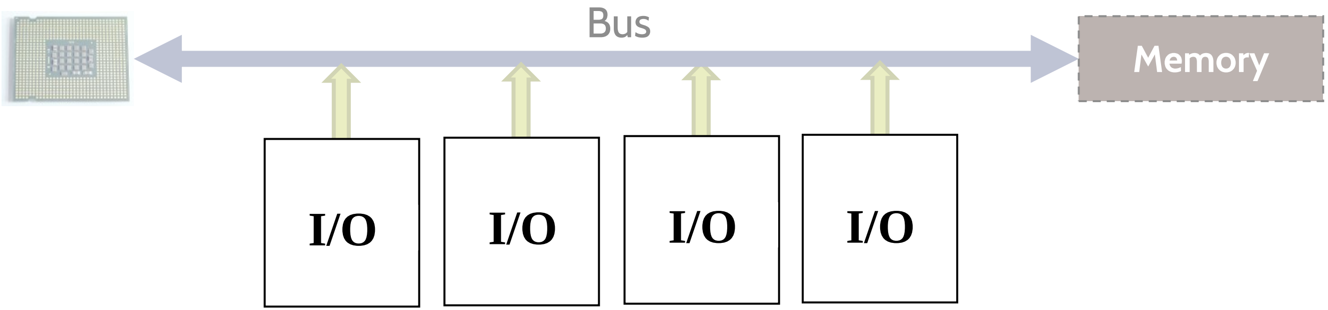




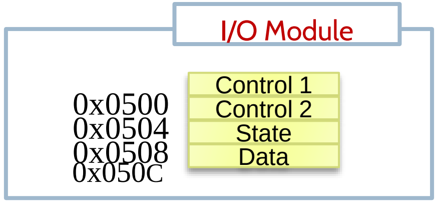


Example

Direct I/O



Example
Direct I/O



- Control 1 information
  - 0: read
  - 1: write
- State information
  - 0: busy device
  - 1: device ready (data available)
- Data
  - Data from device

```
request:
for (i=0; i<100;i++)
{
    // read request
    out(0x500, 0) ;

    // wait loop (busy wait)
    do {
        in(0x508, &(p.status)) ;  // ready?
    } while (0 == (p.status)) ;

    // read data
    in(0x50C, &(p.data[i])) ;
}
```

P1

```
// read(file,data,100) ;
...
```

k

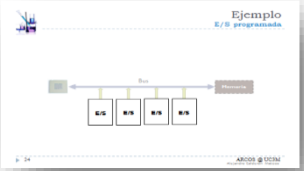
```
for (i=0; i<100;i++)  {
    out(0x500,0) ;
    do {
      in(0x508,&p.status) ;
    } while (0 == p.status) ;
    in(0x50C,&p.data[i]) ;
}
```

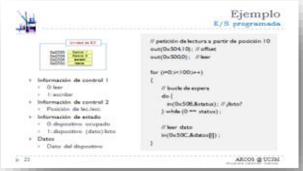
P1

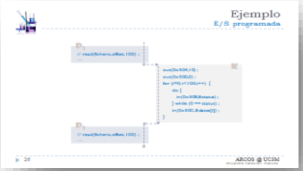
```
// read(file,data,100) ;
... // next instruction
```

Impact in the Operating System of the device handling

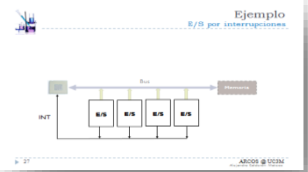
- **Direct I/O**

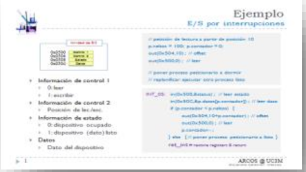


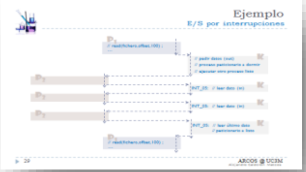




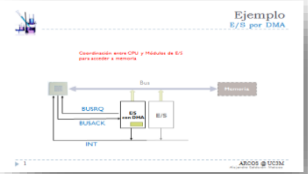
- **Interrupted I/O**

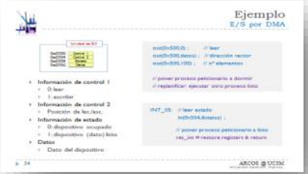


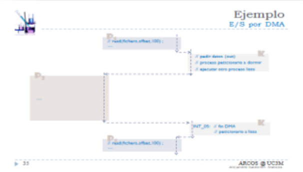




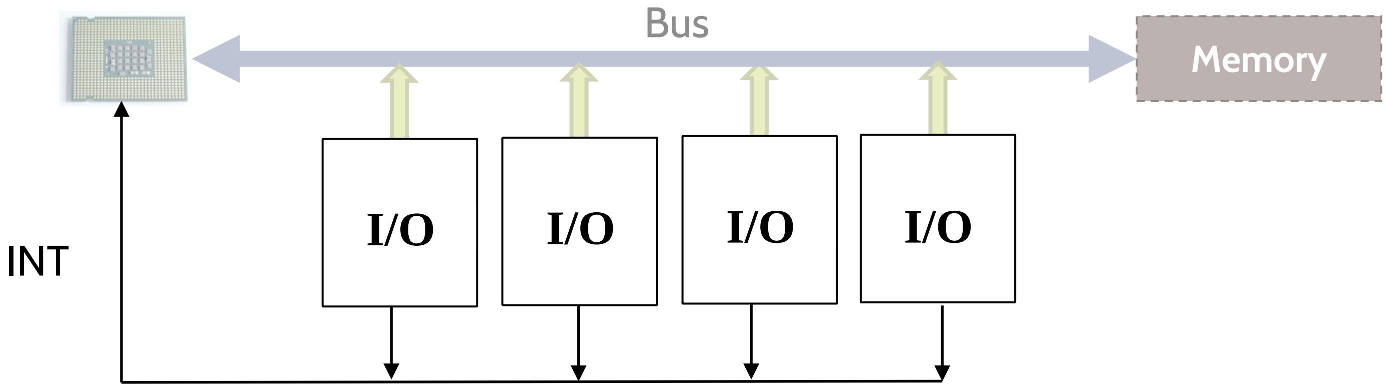
- **DMA I/O**





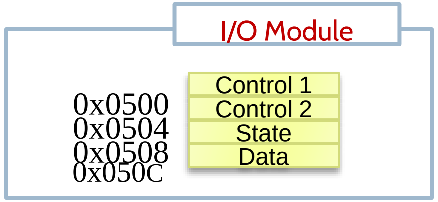


Example
Interrupted I/O



Example

Interrupted I/O



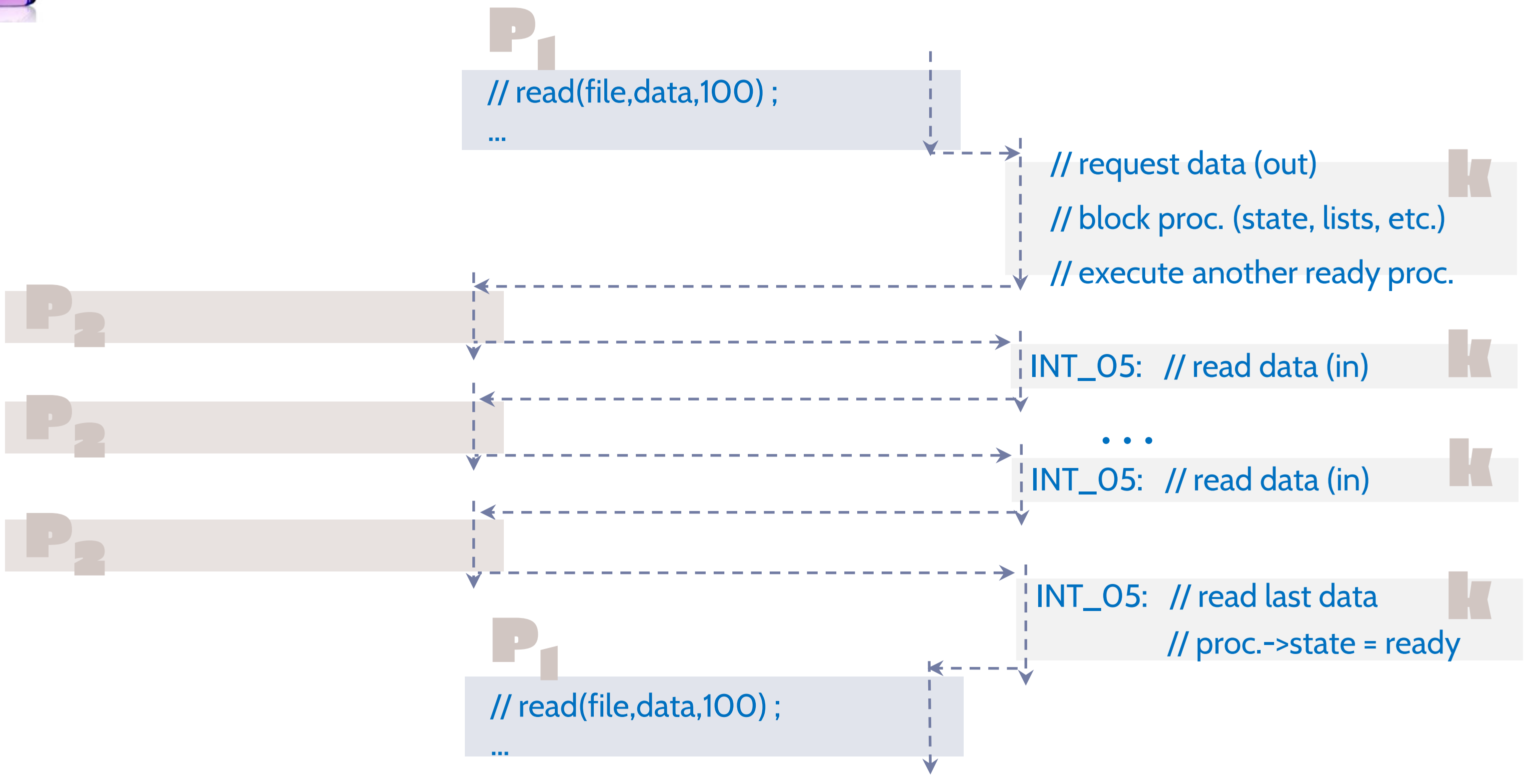
- Control 1 information
  - 0: read
  - 1: write
- State information
  - 0: busy device
  - 1: device ready (data available)
- Data
  - Data from device

```
request:
    // read request
     p.counter = 0;
     p.neltos = 100;
     out(0x500, 0) ;   // read
    // Voluntary context switching (V.C.S.)
```

```
INT_05:
    in(0x508, &(p.status)) ;                 // read state
    in(0x50C, &(p.data[p.counter])) ;        // read data
    if ((p.counter < p.neltos) && (p.status == OK)) {
        p.counter++ ;
        out(0x500, 0) ;  // read
    } else  { // process->state = ready  }
    ret_int # restore registers & return
```

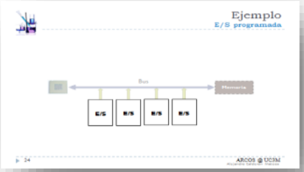
Example

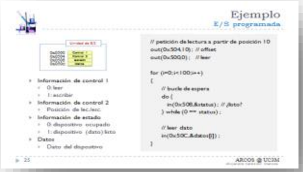
Interrupted I/O

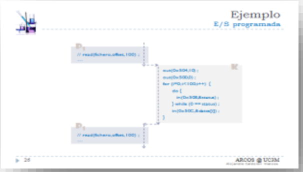


Impact in the Operating System of the device handling

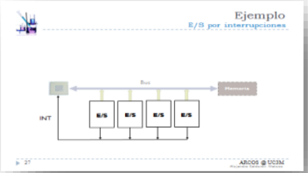
- **Direct I/O**

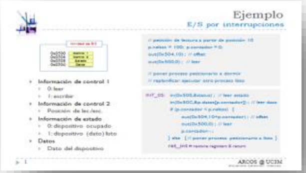


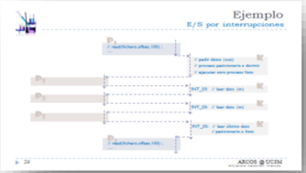




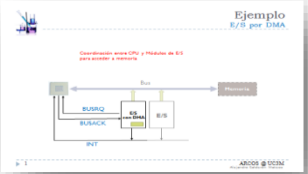
- **Interrupted I/O**

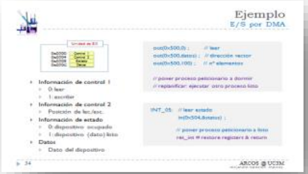


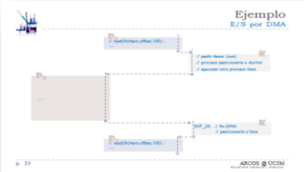




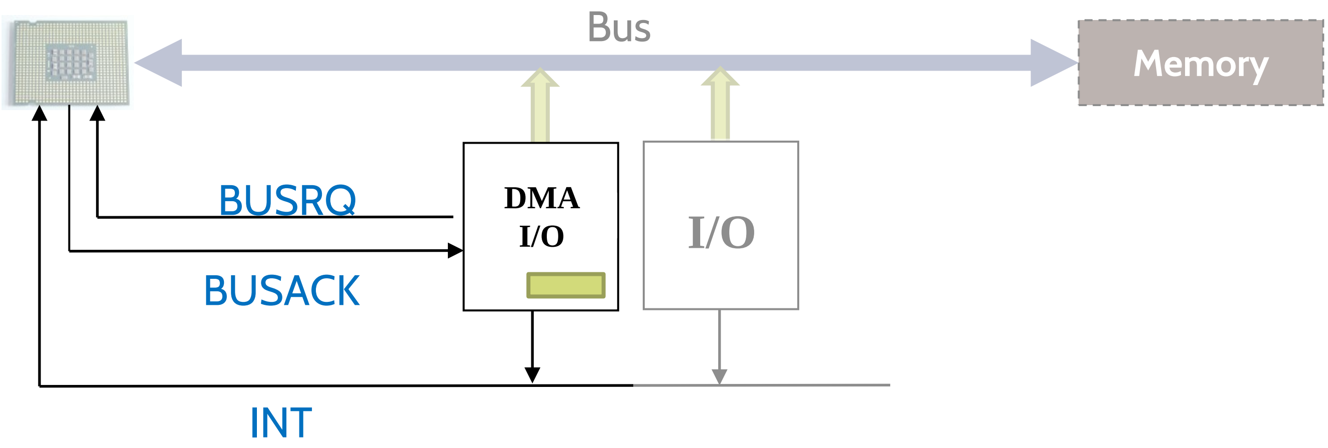
- **DMA I/O**



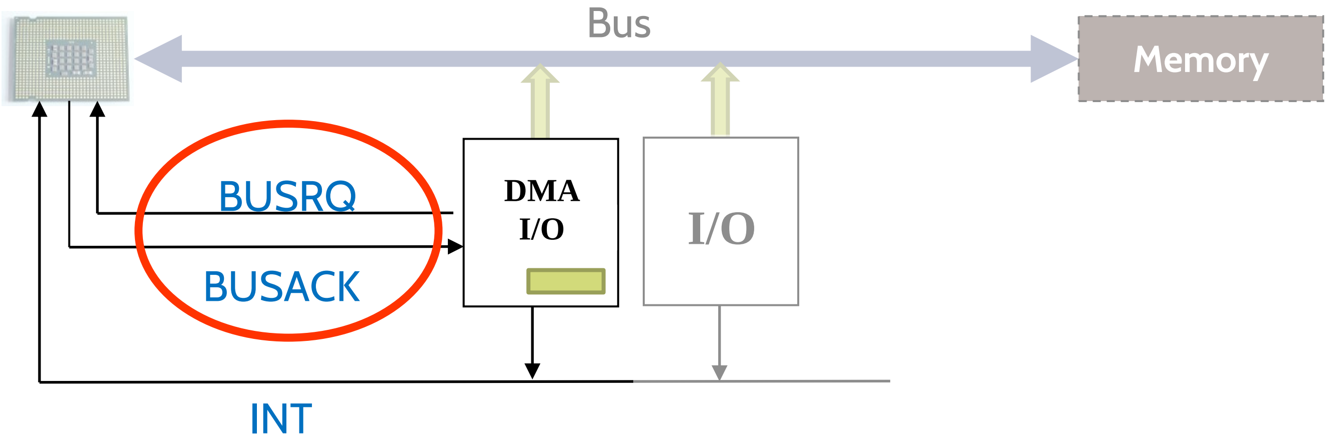




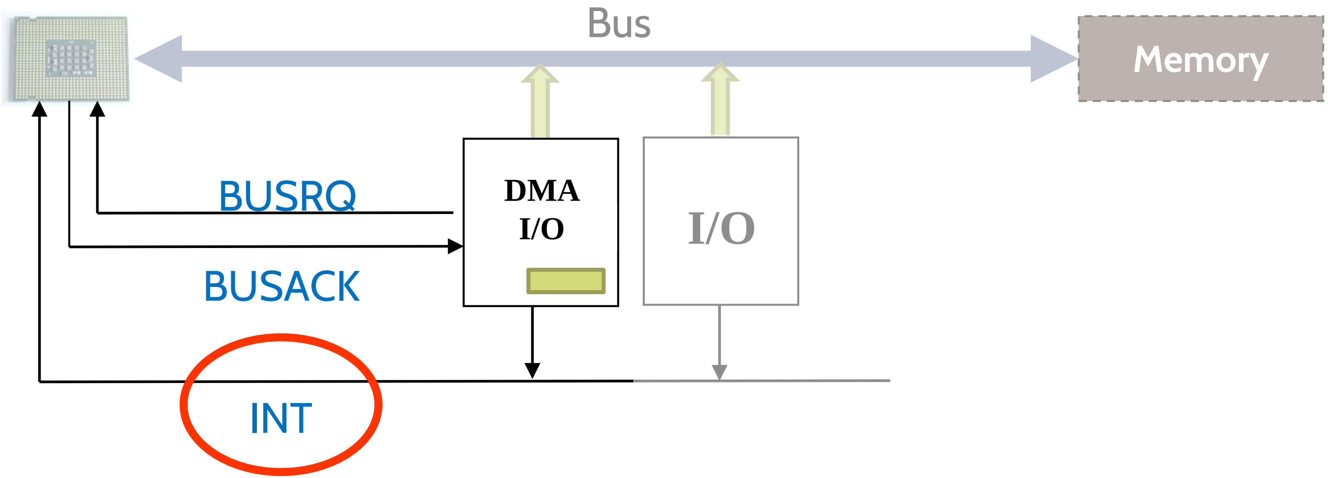
Coordination between CPU and I/O Modules
in order to access to memory



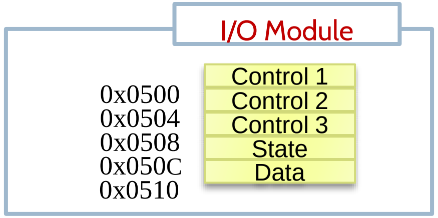
Each data transferred to memory implies:
- To ask permission for accessing memory (BUSRQ)
- To wait permission grant (BUSACK)
- To transfer to memory
- To disable request permission (BUSRQ)



Once all data has been transferred:
- Fire an interrupt (INT) to notify the CPU



Example
DMA I/O



- Control 1 information
  - 0: read, 1: write
- Control 2 information
  - Memory address.
- Control 3 information
  - Number of elements
- State information
  - 0: busy device
  - 1: device ready (data available)
- Data
  - Data from device

```
request:
// perform block request
   out(0x500,0) ;        // read
   out(0x504,p.data) ;  // vector address
   out(0x508,100) ;     // # eltos
 // Voluntary Context Switching (V.C.S.)
```

```
INT_05:   // read state y data
          in(0x50C, &status) ;

          if (p.status...

          // process->state = ready
          ret_int # restore registers & return
```

Example
DMA I/O

P1

// read(file,data,100) ;
...

k

// request data (out)
// block proc. (state, lists, etc.)
// execute another ready proc.

P2

...

k

INT_05: // DMA ends
// proc.->state = ready

P1

// read(file,data,100) ;
...

Main types of protocols

- Request -> individual response
  - Most devices
- Only request
  - E.g.: graphic card
  - Direct I/O (faster or real-time)
- Only response
  - E.g.: clock
  - Interrupted I/O (fire data without former request)
- Request -> shared response
  - E.g.: hard disk

ARCOS Group
Computer Science and Engineering Department
Universidad Carlos III de Madrid

Lesson 3a
process, devices, drivers, and extended services

Operating System Design
Degree in Computer Science and Engineering, Double Degree CS&E + BA