Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Part 2: Process Management

Unit 3

Process concept and implementation

fSO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Goals and bibliography

- **Goals**
  - To define **process** concept
  - To analyze the differences between **sequential** and **concurrent** execution
  - To define the **process states** as well as the causes of the transitions between them
  - To study the **basic structures to support processes** in the OS
  - To analyze the **context switch** mechanism
- **Bibliography**
  - “Operating System Concepts” Silberschatz 8ª Ed
  - “Sistemas operativos: una visión aplicada” Carretero 2º Ed

Content

- **Previous concepts**
- Executable files
- Process concept
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- Exercises

Previous concepts

- In order to understand the process concept some terms have to be clarified, particularly the differences between the following pairs of concepts:
  - Program / Process
  - Sequential execution / Concurrent execucion

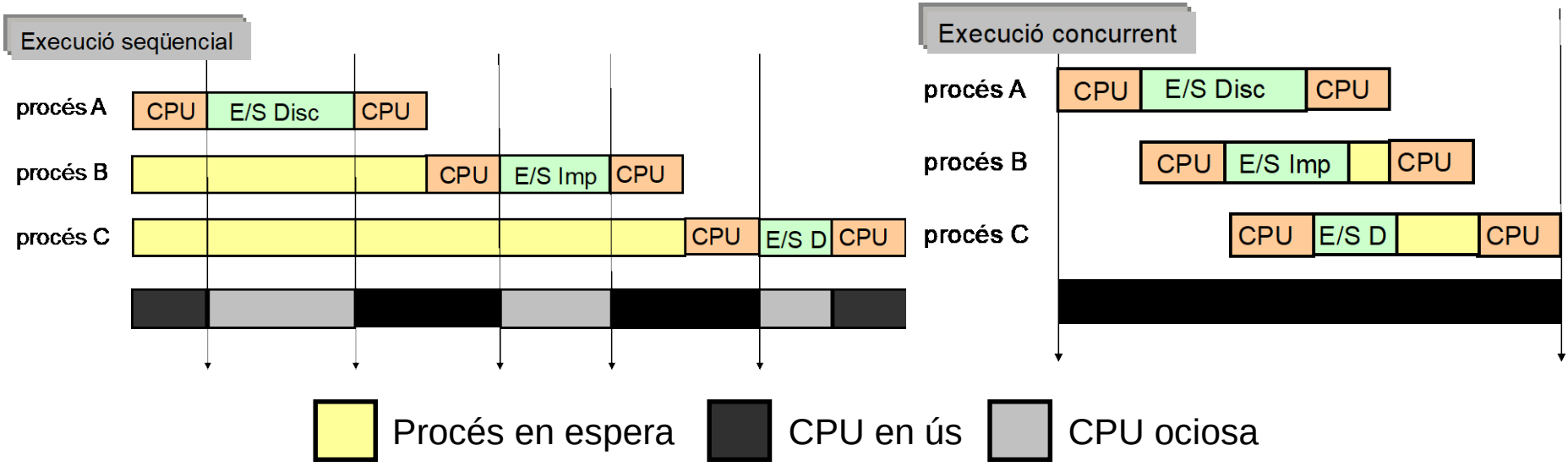
Program vs. Process

- **Program:**
  - **File** in executable format generated by a compilation of source code followed by a linking operation.
  - **Passive entity** it doesn't change along time.
- **Process:** a certain program running:
  - Working unit for the OS
  - Resource consumer
  - Every activity happening on a computer is related to a process
  - **Abstract entity belonging to the OS** that allows modeling computer activity

A process is an alive entity that experiments changes while it exists

Previous concepts

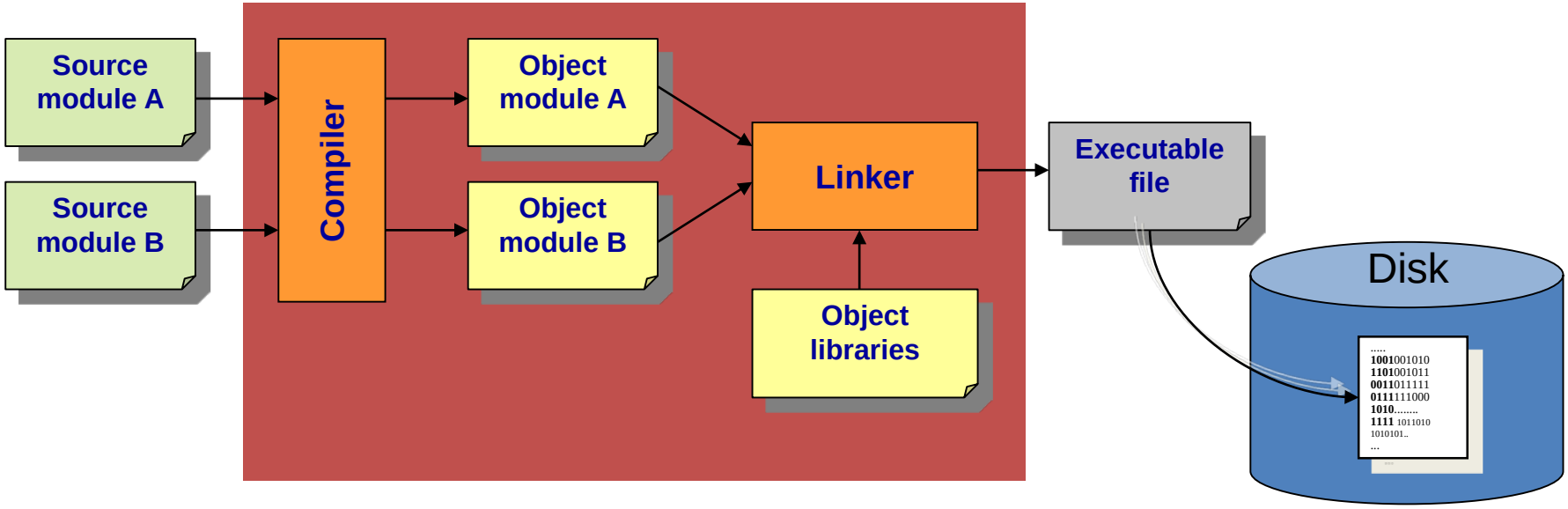
- **Sequential execution**: when the CPU is used by a single process from the beginning till the end of its execution
- **Concurrent execution**: when the CPU is used alternatively by several processes during their lifetime
  - The most direct benefit of concurrent execution is the increase in CPU utilization, since processes are not constantly demanding CPU because they alternate CPU with I/O bursts



- Previous concepts
- **Executable files**
- Process concept
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- Exercises

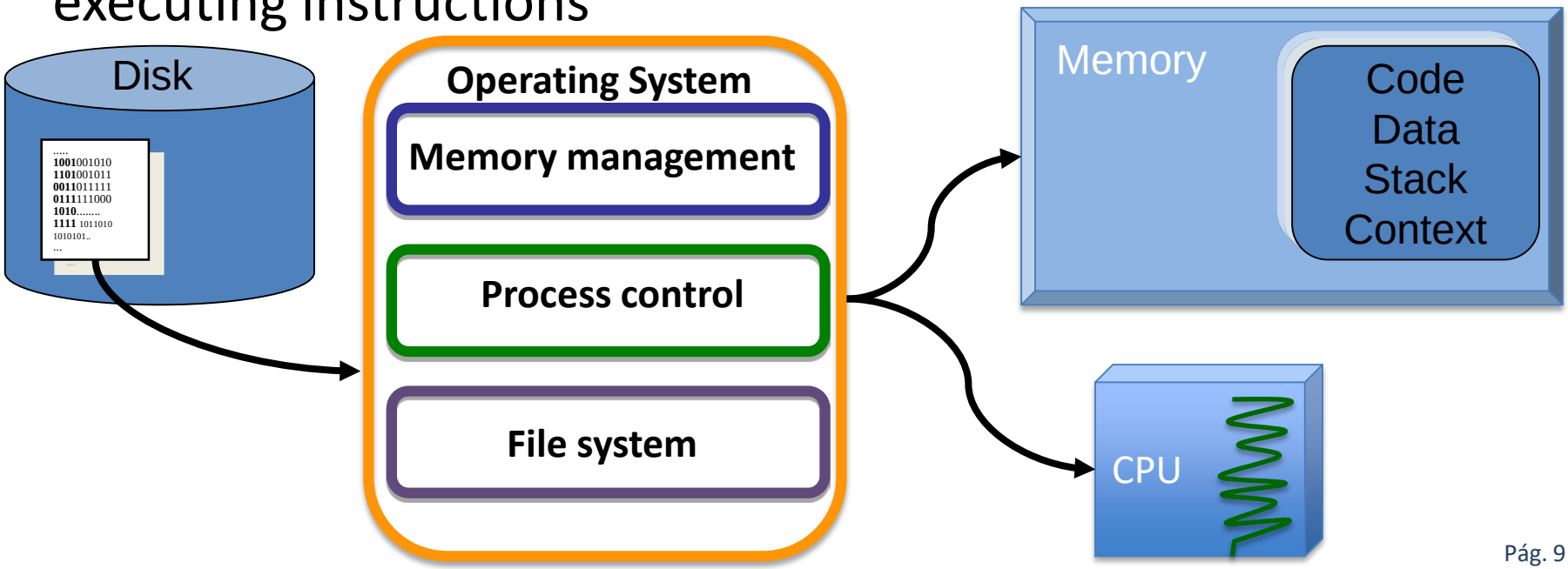
Executable files

- Steps to get an executable file:
  - The first step is **to compile** the **source code** to get **object code**
  - Later it is **linked** to system or other user **library code** -> this allows incorporating **external code**
  - The final result is an **executable file**



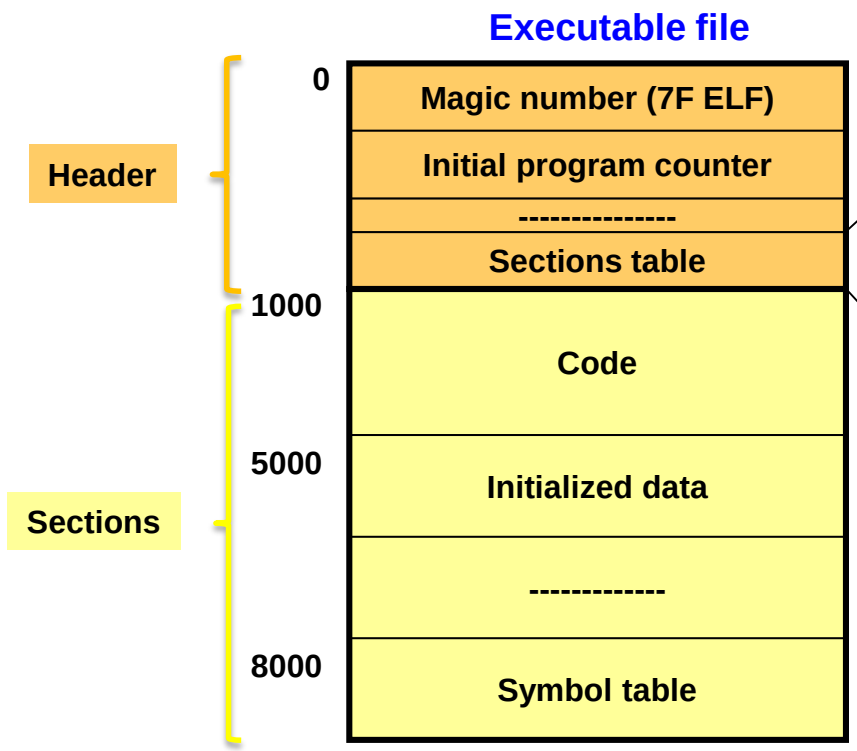
Executable files

- An executable is a file whose structure is well known by the operating system because it contains:
  - The code to be executed
  - Initialized data
  - Library functions (static linking)
- With this information the OS can allocate the necessary resources for execution. Load data, code etc. and start executing instructions



The executable ELF

- **Executable and Linking Format**
  - File format for executables, object code, libraries and memory dumps
    - Widely used in Unix, Linux, Solaris and BSD
    - Extensions: .o, .so, .elf



	Offset	Size
Code	1000	4000
I.V. Data	5000	1000
Not I.V. data	------	500
----------	------	-----
# symbols	8000	1000

Other formats:

- Windows NT: PE/COFF (.exe, .dll)
- Mac OSX: Mach-O

Content

- Previous concepts
- Executable files
- **Process concept**
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- Exercises

Process concept

- In order to define a process there are three aspects to be considered:
  - **Atributtes** or features that define a process and allow to manage them
  - **Behaviour**, as being an active entity it will move between states so these states and their transitions have to be defined
  - **Operations** that can be done with processes

- **Process attributes**

  Resources and features owned by a process and kept by the OS

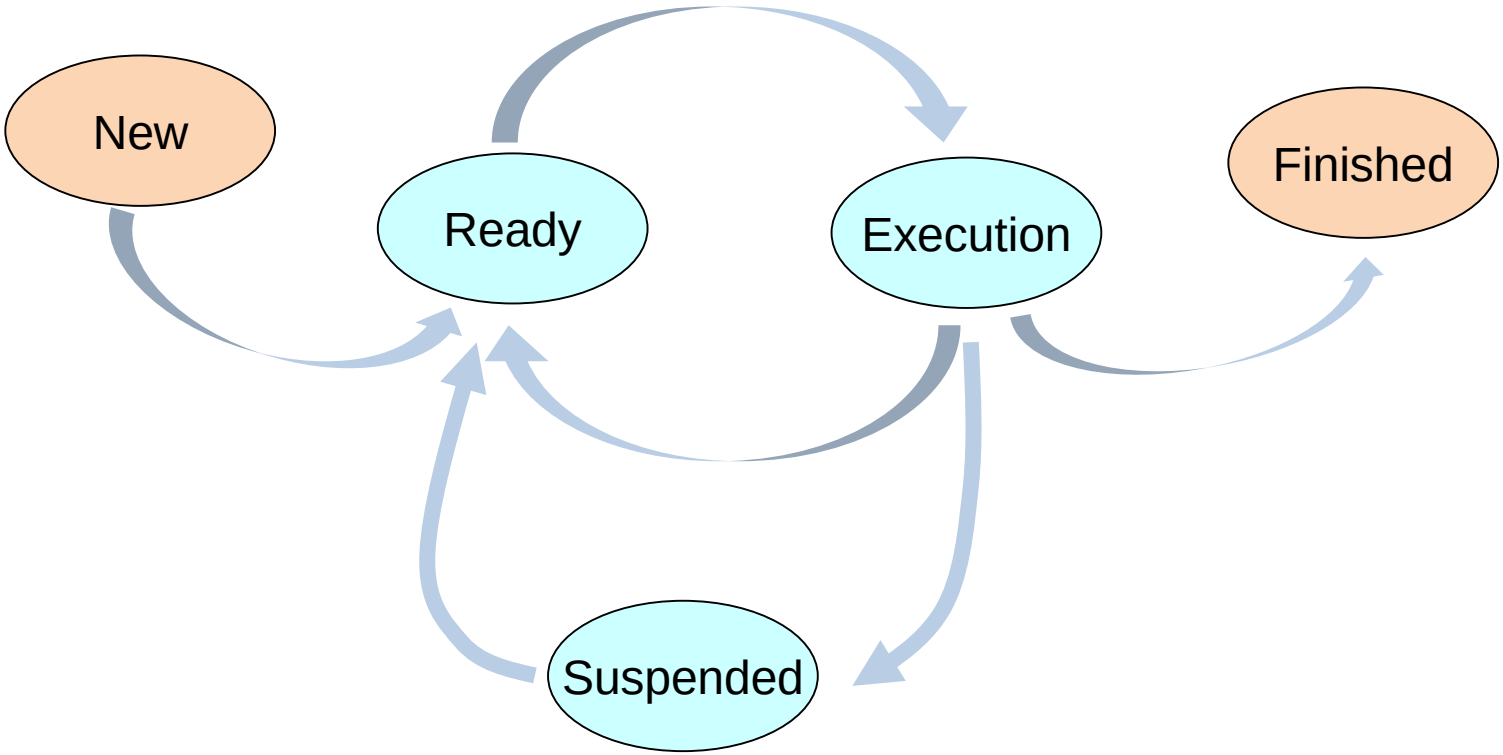
  - **Identity**
    - Process identifier (PID)
  - **Runtime environment**
    - Working directory
    - Opened file descriptors
  - **State**
    - Process state
    - Machine context (program counter, stack pointer, general porpuse register values)
    - Etc.
  - **Memory**
    - Memory addresses for code, data and stack
  - **Scheduling**
    - CPU consumption time
    - Priority
  - **Monitoring**

- **Operations on processes**
  - As happens with the attributes, the number and the type of operations that can be done on processes **depend on the OS considered**
  - Anyway, the following operations are always supported:
    - Creation
    - Communication
    - Waiting
    - Resource access
    - Ending

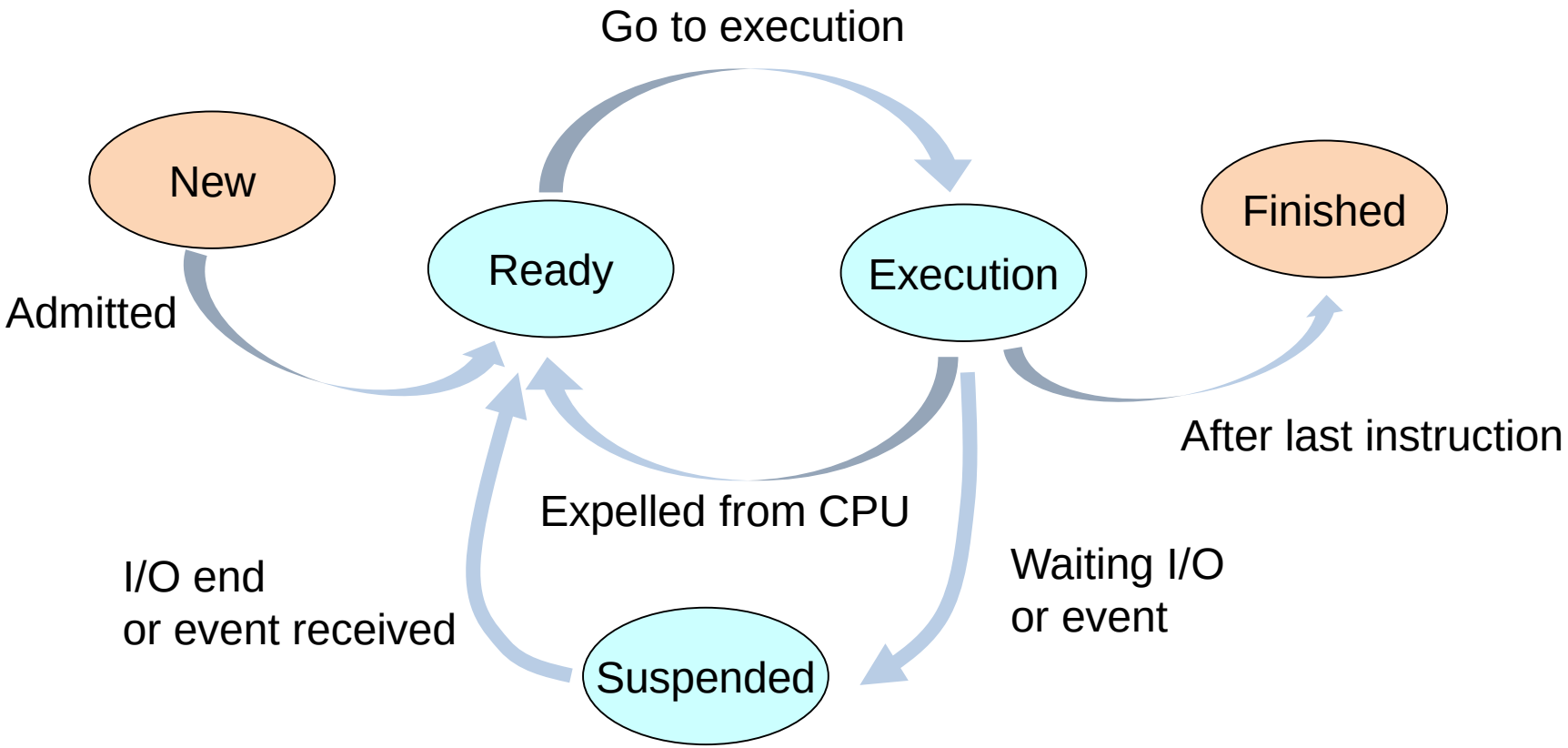
- Previous concepts
- Executable files
- Process concept
- **Process states**
- Process implementation: PCB
- System calls table for processes and signals
- Exercises

Process states

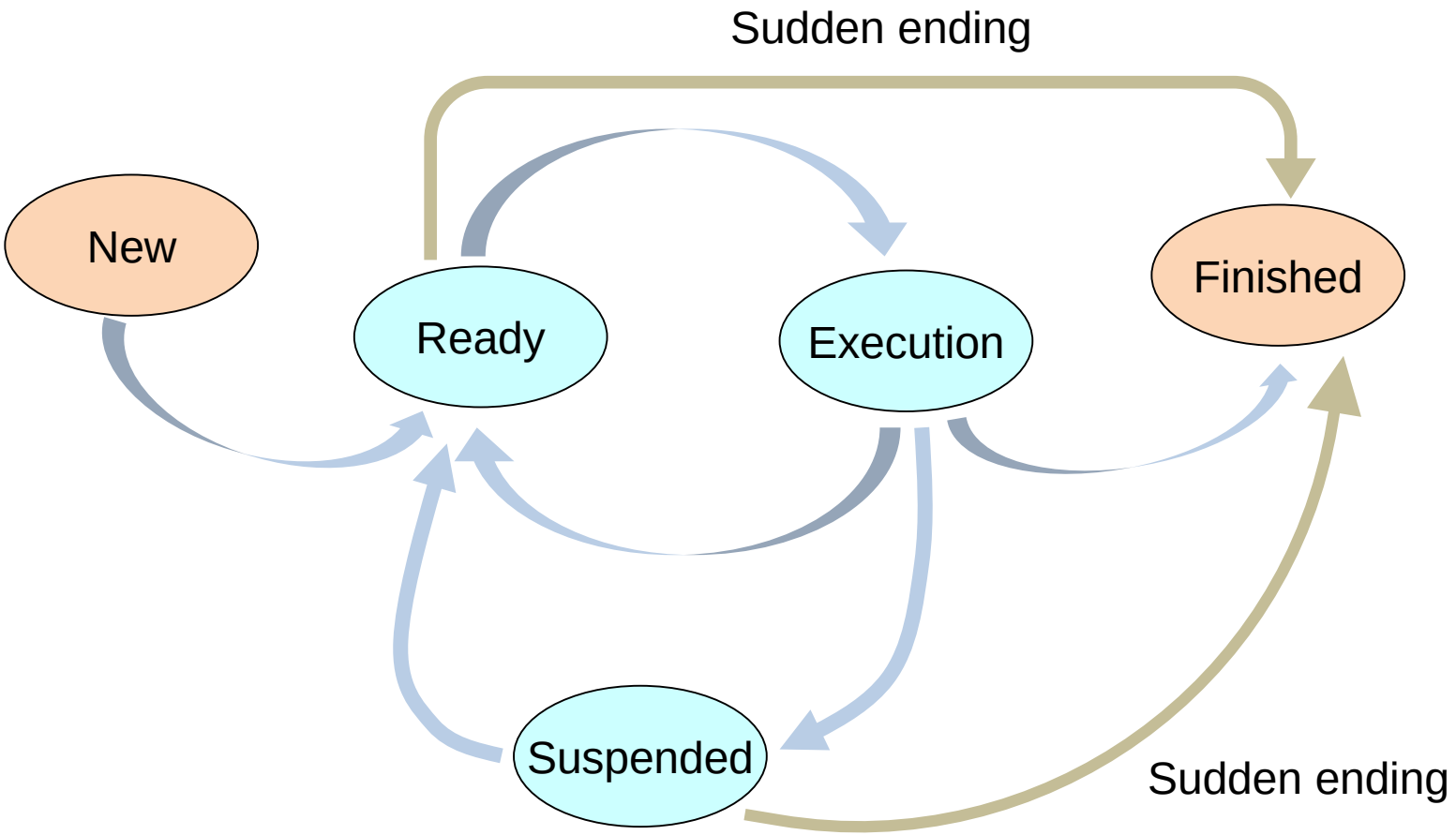
- **Processes** are **active entities** so they evolve during his lifetime through a **set of states**



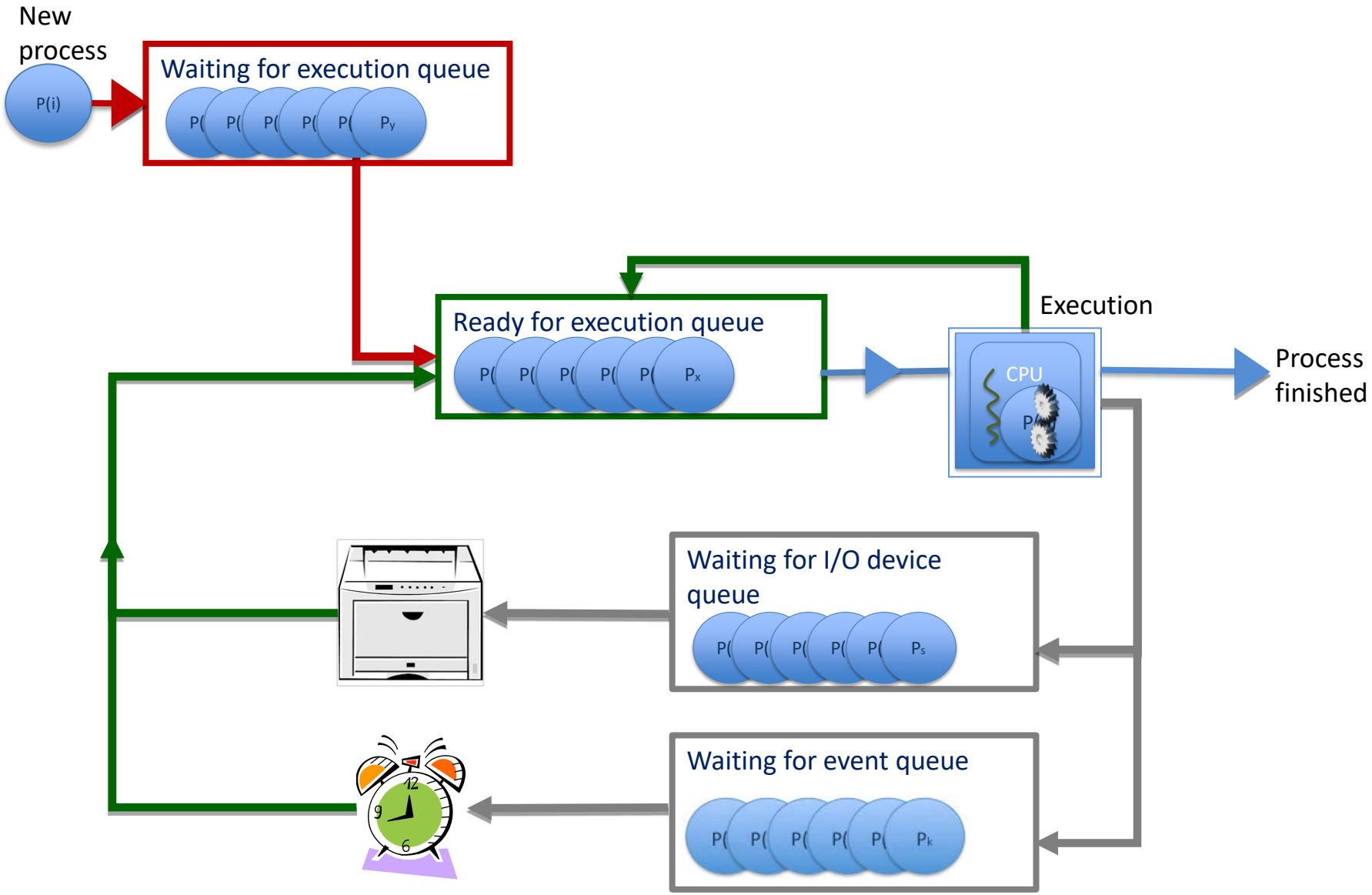
- **Common process state transitions**



- **Transitions due to anomalous behavior**



Process states

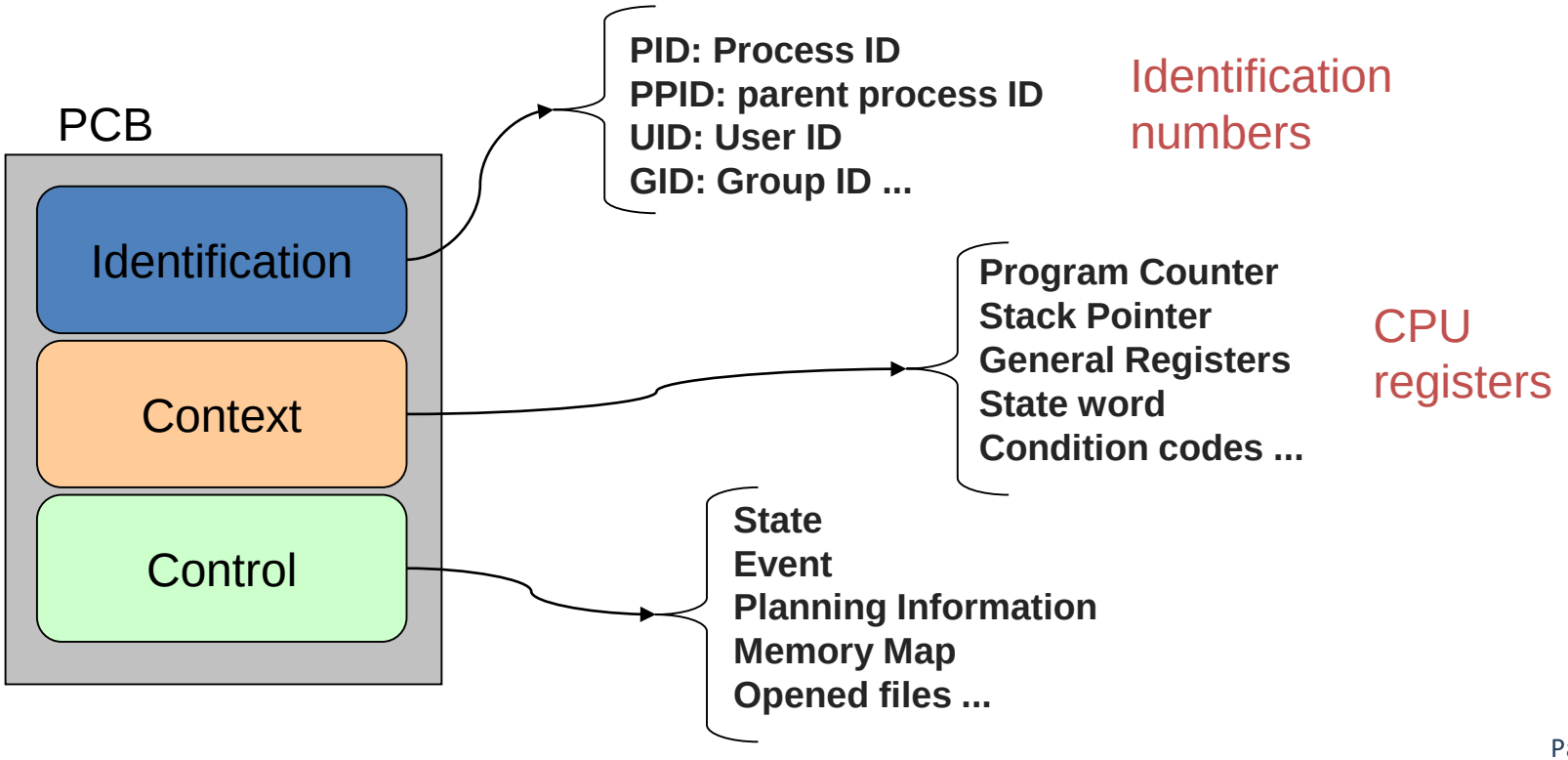


Content

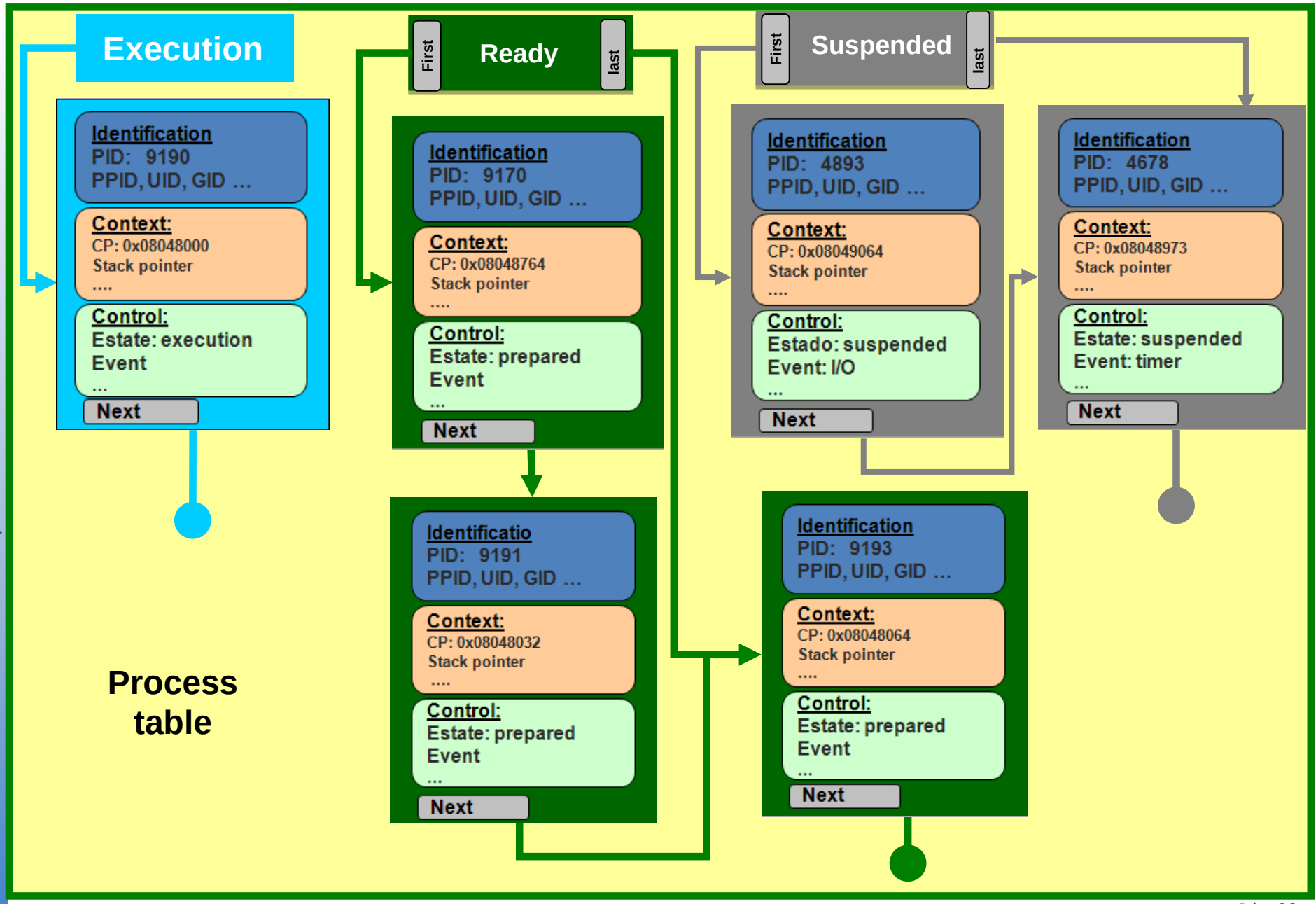
- Previous concepts
- Executable files
- Process concept
- Process states
- **Process implementation: PCB**
- System calls table for processes and signals
- Exercises

Process implementation : PCB

- A PCB (Process Context Block) is the **data structure** which supports the concept (abstraction) process
  - An OS is also a program, based on using **algorithms** and **data structures**
- PCB keeps process relevant information which changes during process lifetime -> each operating system has its own structure

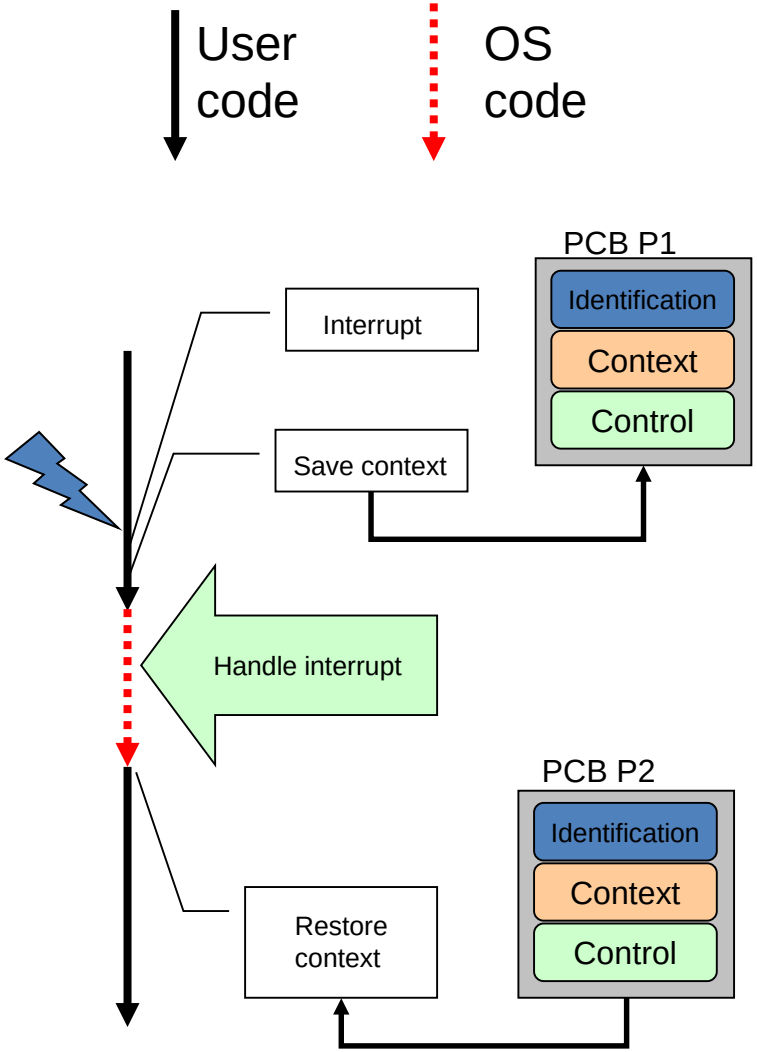


Process implementation : PCB



Process implementation: PCB

- **Context switch**
  - Mechanism that allows an OS to suspend the execution of the current process in order to start or resume the execution of another process.
    - This process is activated by an interrupt (i.e. clock interrupt)
  - What is done?
    1. The state information relevant to the running process (context) is saved
    2. The scheduler updates PCBs and queues
    3. The new process takes the CPU



Content

- Previous concepts
- Executable files
- Process concept
- Process state
- Process implementation: PCB
- **System calls table for processes and signals**
- Exercises

System calls table for processes and signals

	Processes
fork	Create a child process
exit	End process
wait	Wait for a process ending
exec	Execute a program
getpid	Get process attributes

	Signals
kill	Send signals
alarm	Generate an alarm (clock signal)
sigemptyset	Init a mask with no signals set
sigfillset	Init a mask with signals set
sigaddset	Append a signal to a signal set
sigdelset	Delete a signals in a signal set
sigismember	Check if a signals belongs to a signal set
sigprocmask	Check/Modify/Set a signal mask
sigaction	Capture/Manage a signal
sigsuspend	Wait signals capture

Content

- Previous concepts
- Executable files
- Process concept
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- **Exercises**

Exercise 1:

- In the following actions list, indicate from each one if it is shell code or OS code the one that performs it. Mark one, none or both options with a cross:

OS	Shell	
		Reading the command line and parse it
		Programming a device controller
		Providing a system calls interface
		Seleccting a process to assign it the CPU
		Performing a system call
		Providing a confortable user interface

Exercise 2

- Which state (new, ready, execution, suspended, finished) corresponds to every one os the following processes:

Process		State
P1	The CPU is executing intructions belowing to P1	
P2	P2 has ask for a disk access, but the disk is busy serving to process P3	
P3	P3 is doing disk access	
P4	P4 belongs to a user that has finishied all his/her jobs in a terminal and is loging out the session	
P5	P5 has a process identifier assigned and only its control tables are already created	
P6	P6 has all its SO control structures and its memory image stored in maim memory	

Exercise 3

- In the following command line:

```
$ cat f1 f2 f3 | grep start | wc -l >tracd
```

Indicate:

  - How many processes will be created during its execution in a UNIX system
  - What files has associated every command

Exercise 4

- Given the following code:

```
#include <stdio.h>
#include <sys/types.h>

int main(void)
{
   pid_t pid;
   int i;

   for (i=0; i<2; i++)
      pid=fork();
   return 0;
}
```

How many processes will be created along its execution?