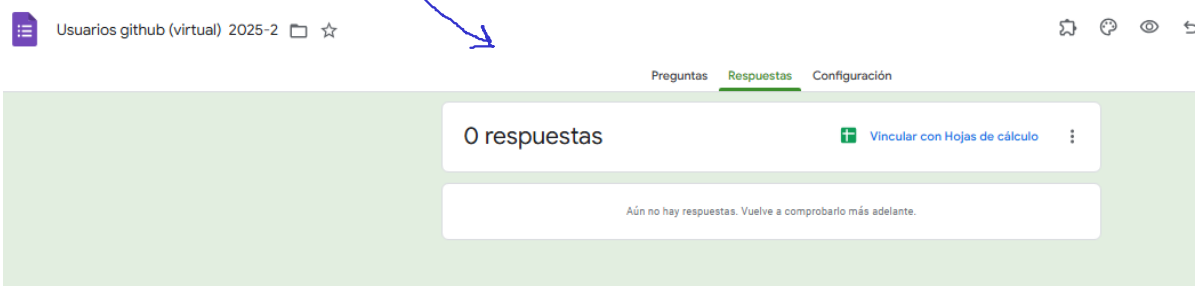
26/08/2025 - Sistemas Operativos (Vde@)

1. Sobre los avisos

Debate	Comenzado por	Último mensaje ↑	Réplicas
Bienvenida	HENRY ALBERTO ... 11 ago 2025	HENRY ALBERTO ... 11 ago 2025	0
Pendientes	HENRY ALBERTO ... 13 ago 2025	HENRY ALBERTO ... 13 ago 2025	0
Apuntes clase 2	HENRY ALBERTO ... 14 ago 2025	HENRY ALBERTO ... 14 ago 2025	0
Red Hat Academy	HENRY ALBERTO ... 18 ago 2025	HENRY ALBERTO ... 18 ago 2025	0
Apuntes clase 3	HENRY ALBERTO ... 19 ago 2025	HENRY ALBERTO ... 19 ago 2025	0
Problemas con el equipo	HENRY ALBERTO ... 21 ago 2025	HENRY ALBERTO ... 21 ago 2025	0
Quiz 2 disponible	HENRY ALBERTO ... 23 ago 2025	HENRY ALBERTO ... 23 ago 2025	0

Ojo

* Tener esto listo para el dia del laboratorio.

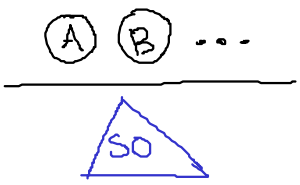


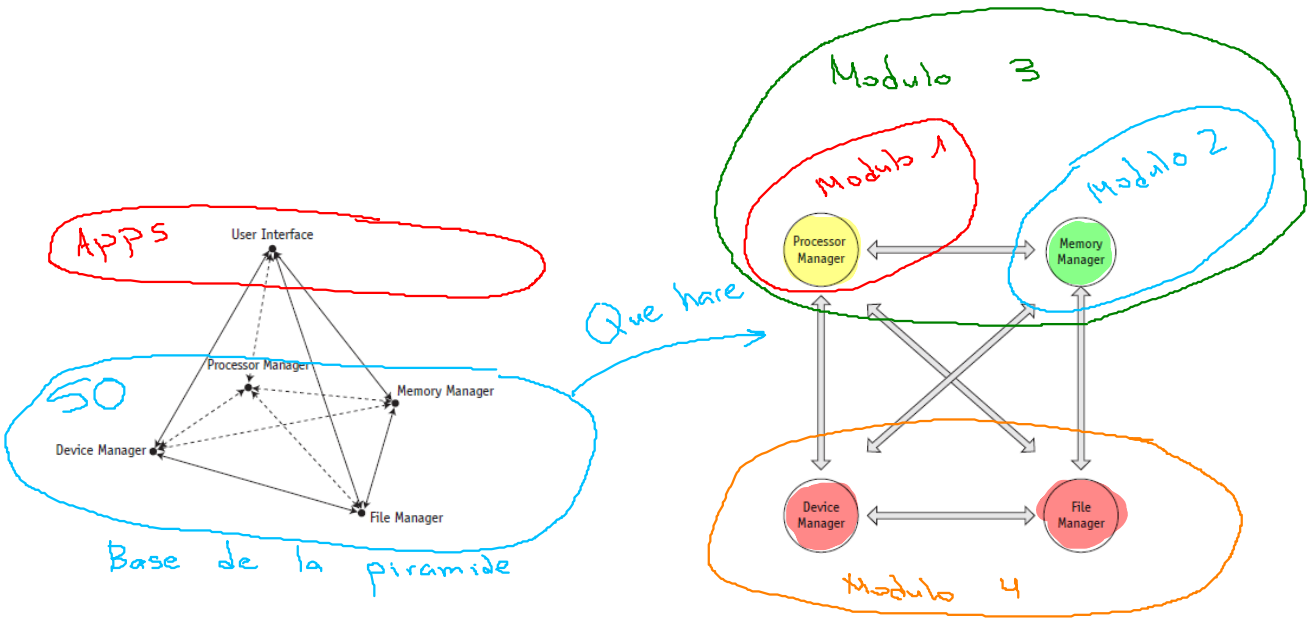
2. Repaso con imagenes

a. Definicion de Sistema Operativo

App: software normal

SO: Software especial

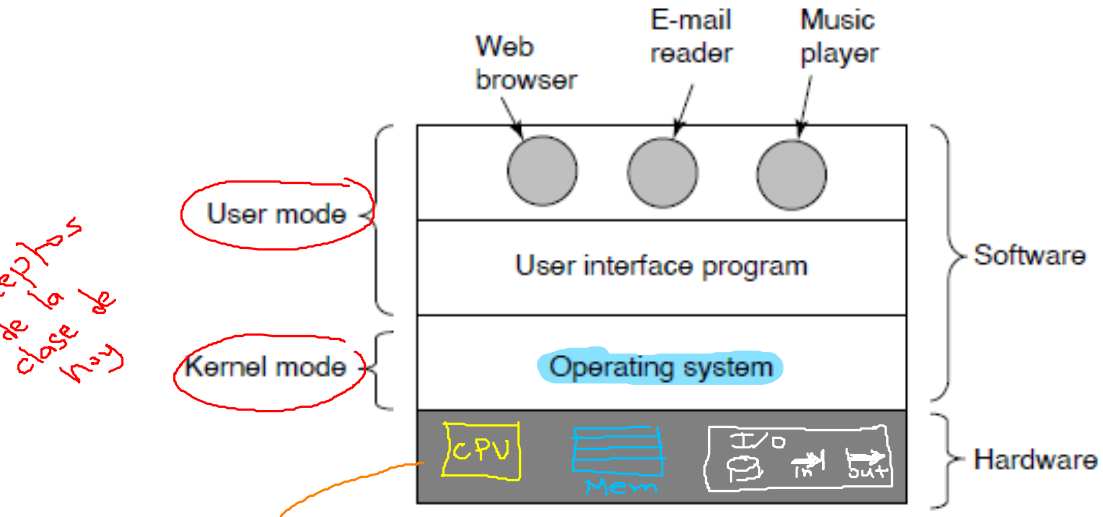




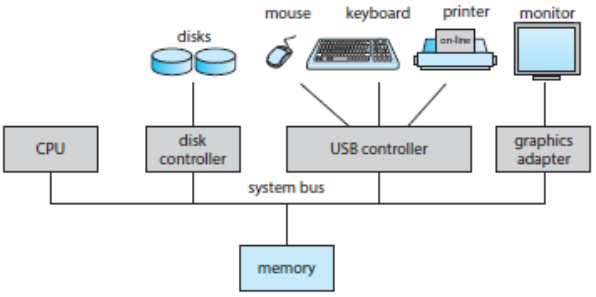
b. Interacción Software – Hardware

- Capas

Conceptos de la clase de hoy

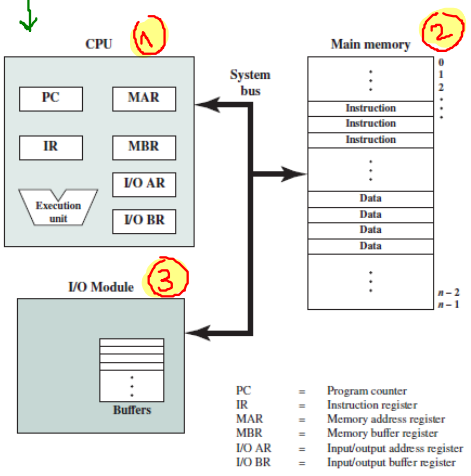


Hw. administrado por el Sistema Operativo



A typical PC computer system.

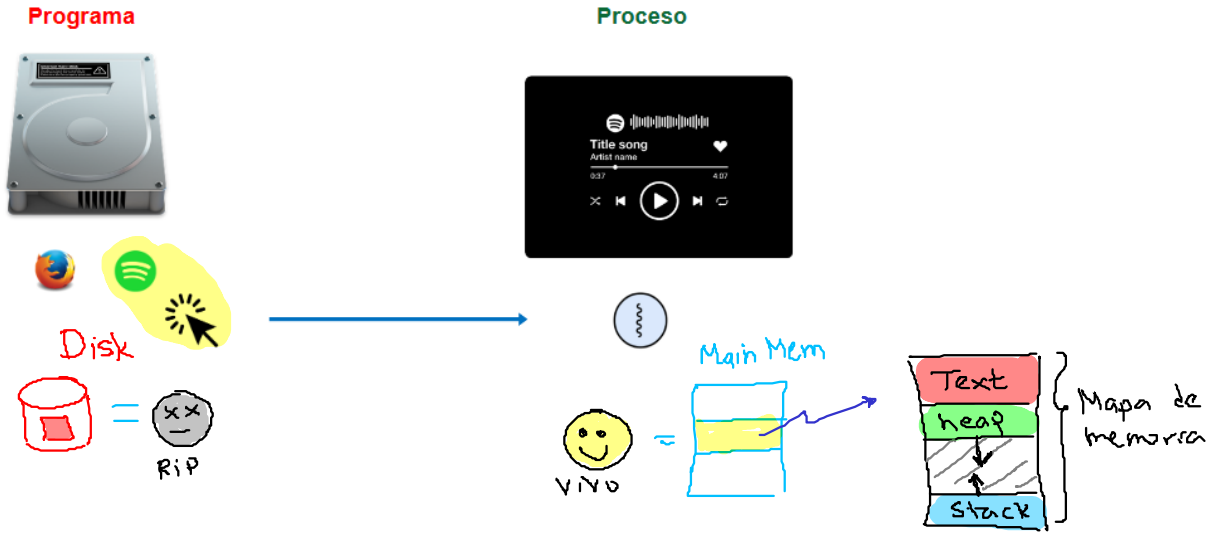
Representación Resumida del Hw como bloques



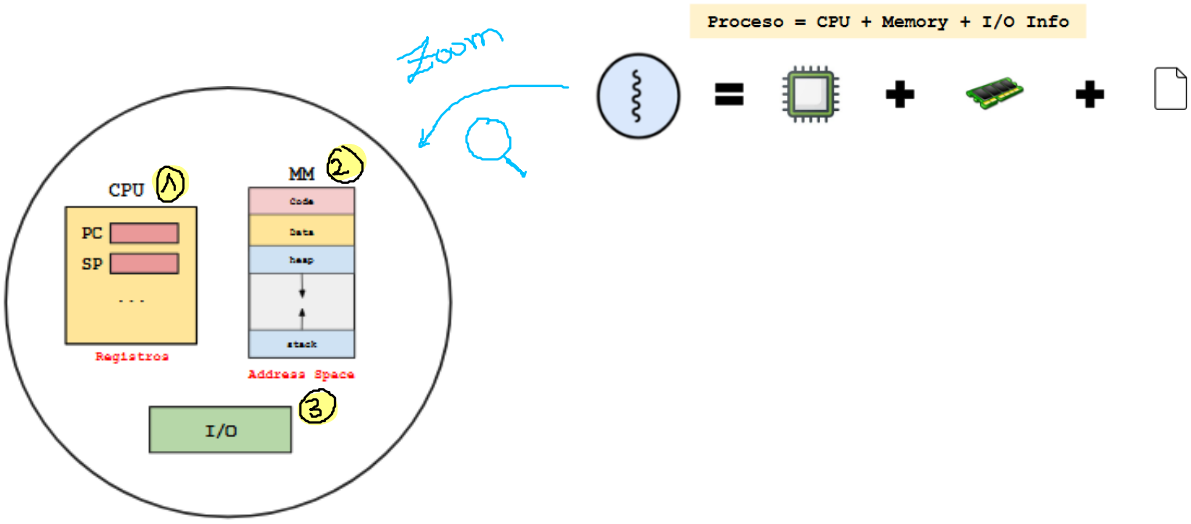
Computer Components: Top-Level View

c. Proceso

* Distincion importante : Programa ≠ Proceso



Abstracción: El Proceso



* Estados de un proceso: De alguna manera un proceso es como un ser vivo.

Modelo de cinco estados (Silverchatz, Stallings)

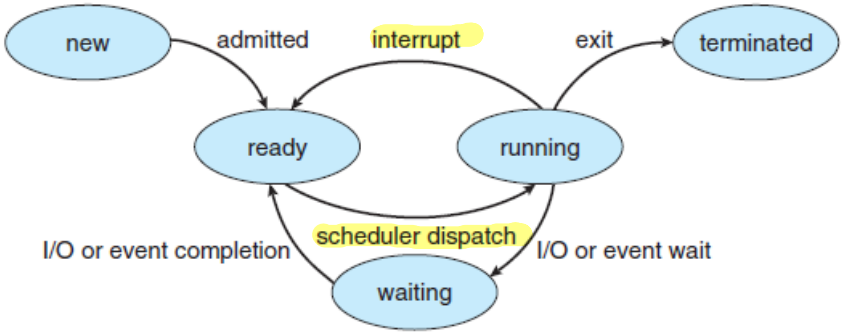
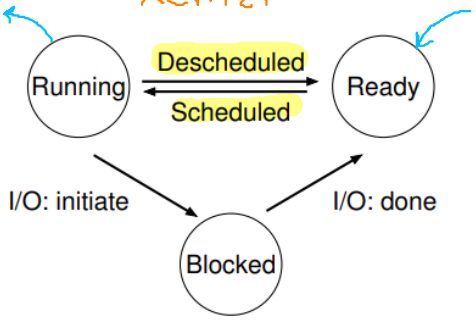


Diagram of process state.

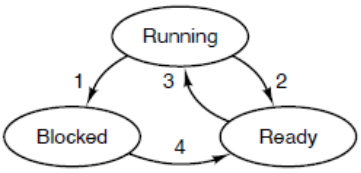
Modelo de tres estados

Remzi



Process: State Transitions

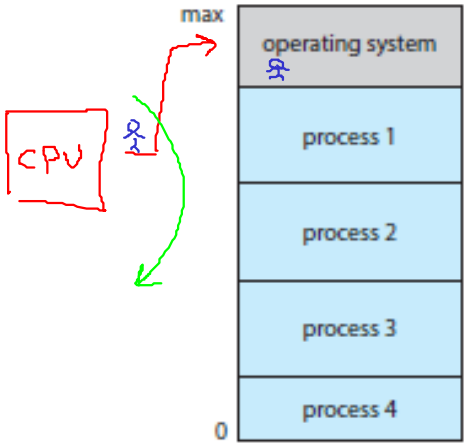
Tanembaum



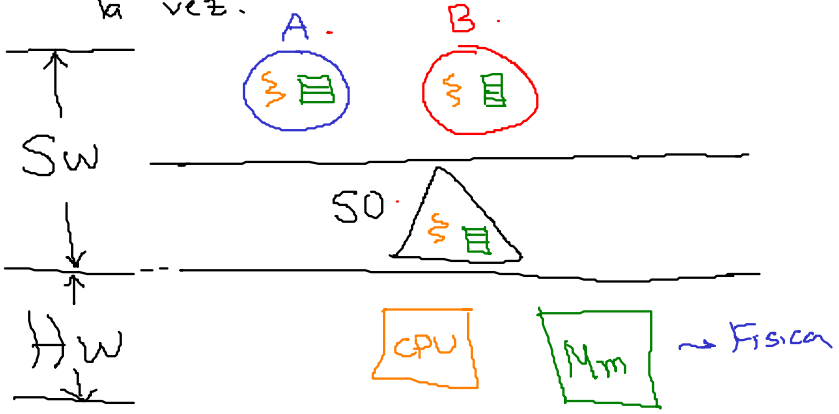
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

A process can be in running, blocked, or ready state. Transitions between these states are as shown.

* Proceso y Virtualización

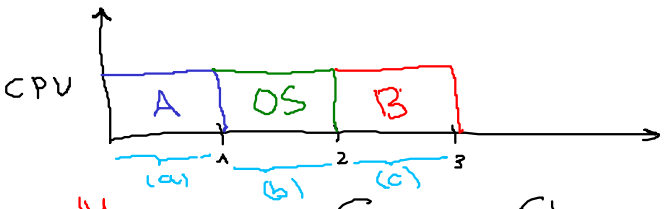


Multiprogramación: Ilusión de que varios programas se ejecutan a la vez.

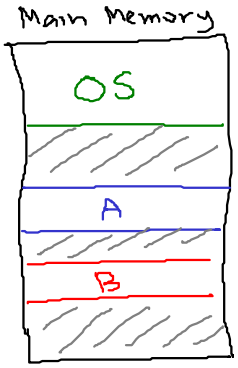


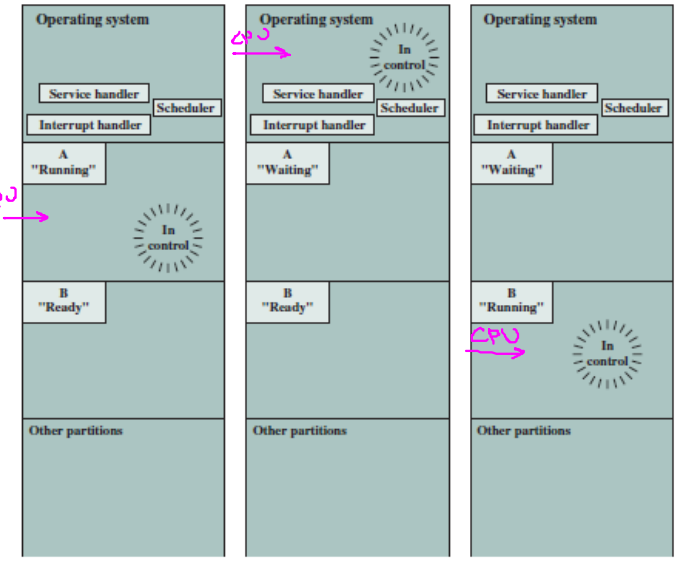
Memory layout for a multiprogramming system.

CPU: Time Sharing

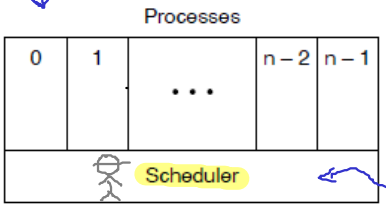


Memoria: Space Sharing





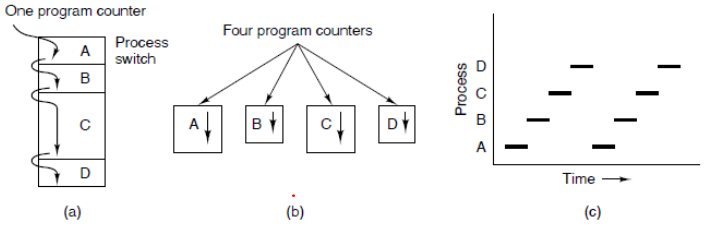
Scheduling Example



Parte del SO

The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

* Ilusión . vs. Realidad

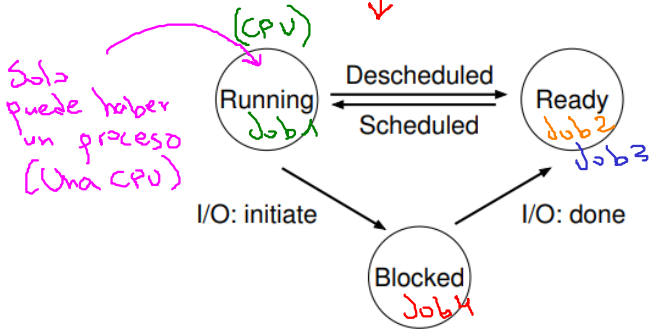


(a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

CPU

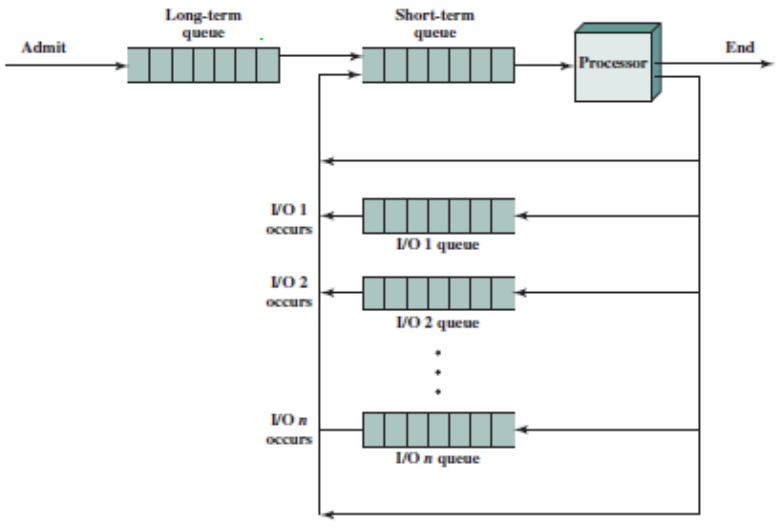
Modelo de 3 estados

Solo puede haber un proceso (Una CPU)



Process: State Transitions

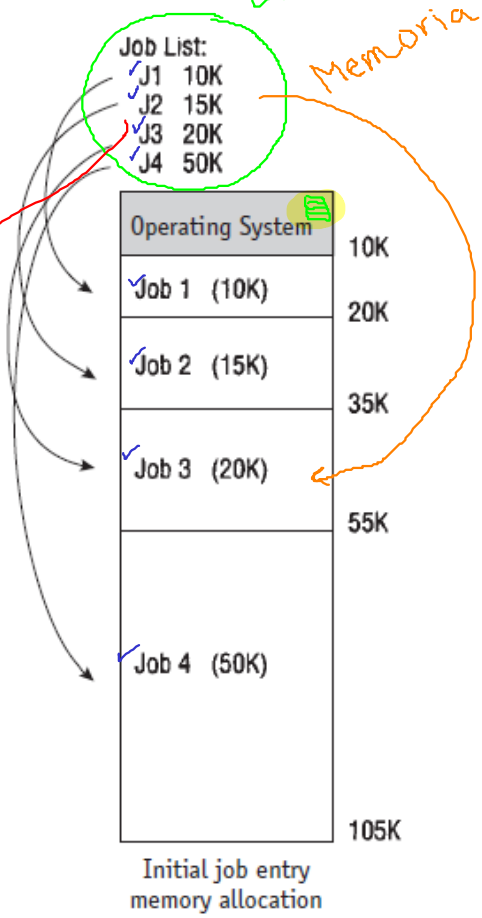
Como se implementa este modelo (Aprox).



Queuing Diagram Representation of Processor Scheduling

Task List

Memoria

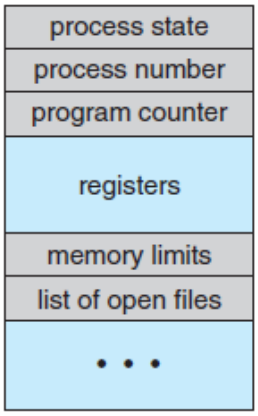


Initial job entry memory allocation

* Estructuras de datos asociadas

- Process control Block (PCB)

Zoom ?

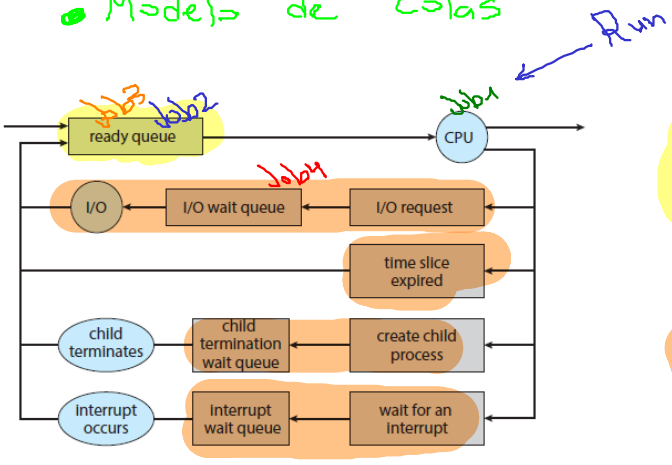


Process control block (PCB).

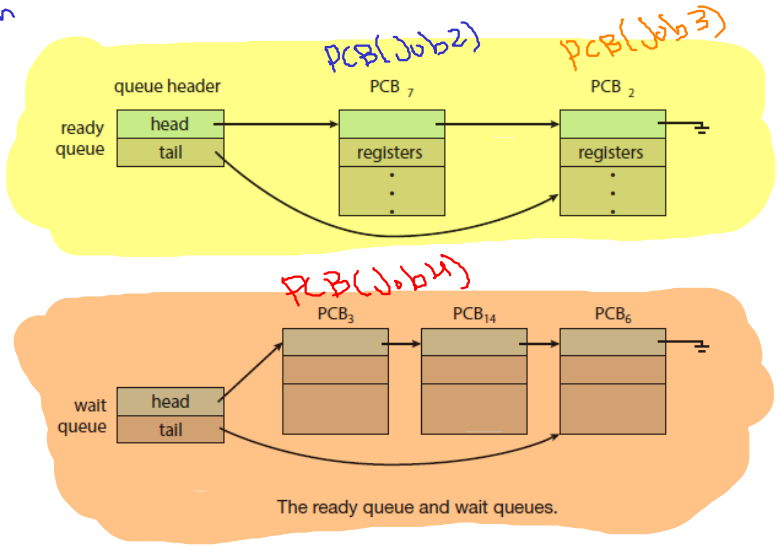
Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Some of the fields of a typical process-table entry.

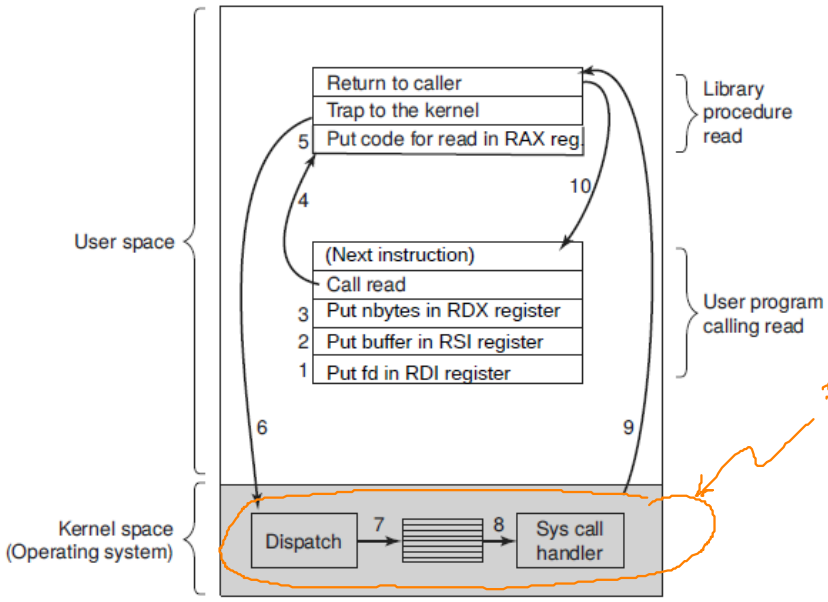
- Modelo de Colas



Queueing-diagram representation of process scheduling.



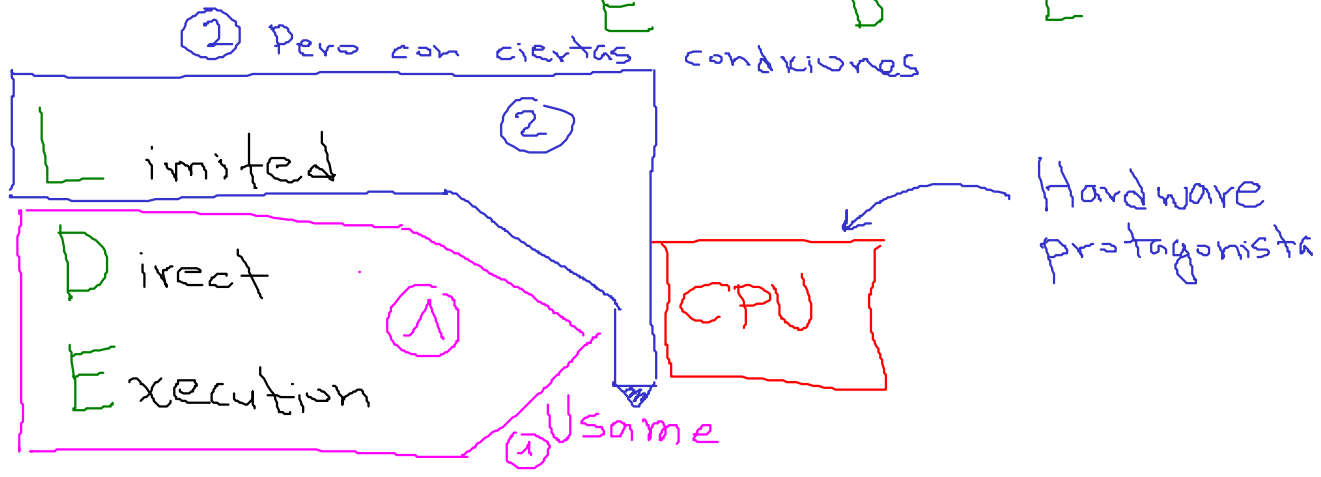
The ready queue and wait queues.



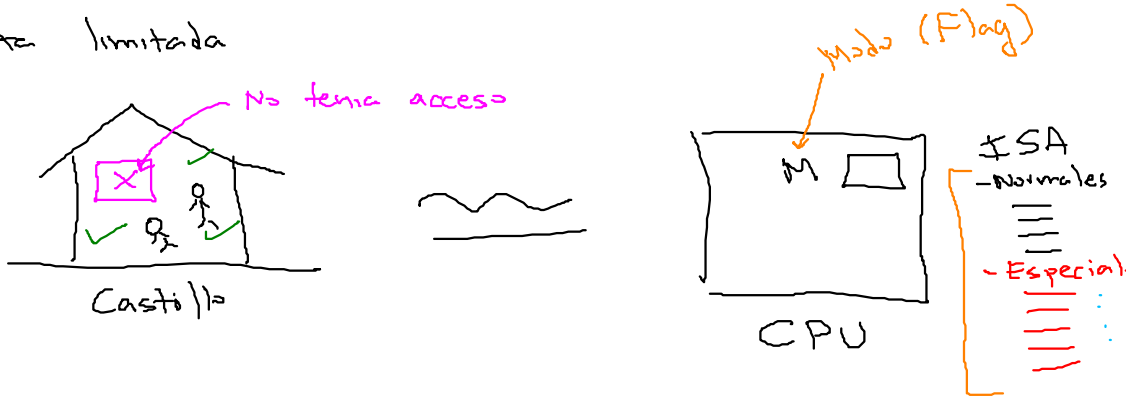
Estas estructuras de datos estan en espacio kernel.

3. Tema de Hoy (Ejecución directa limitada)

E D L



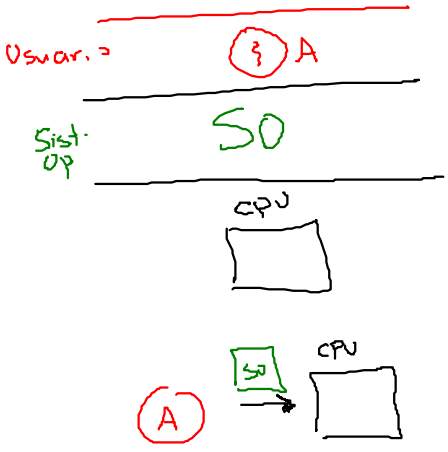
Ejecución directa limitada

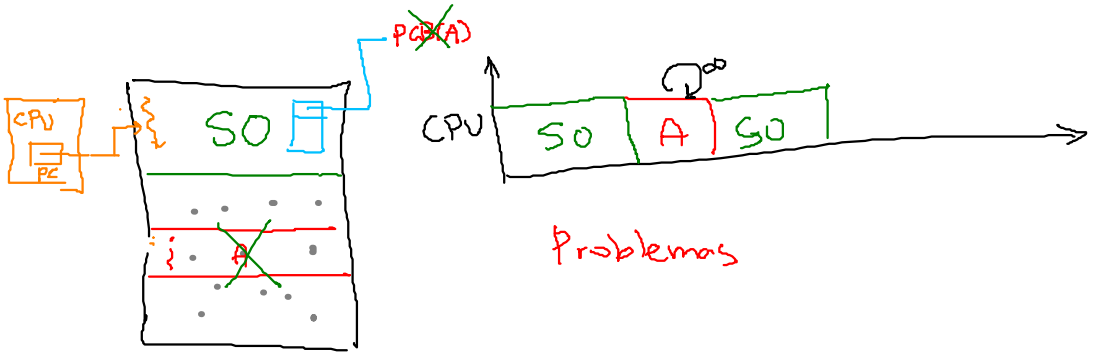


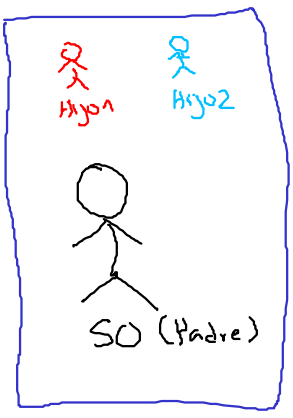
Como se usa la CPU entre los diferentes proceso?

1 Ejecución Directa

Sistema Operativo	Proceso de usuario
1. Crea una entrada en la **lista de procesos** 2. Asigna memoria al proceso 3. Carga el programa a memoria 4. Inicializa pila (*stack*) con `argc/argv` 5. Reinicia registros de CPU 6. Ejecuta llamada a `main()`	
	7. Ejecuta `main()` 8. Ejecuta `return` de `main()` ∞
9. Libera la memoria del proceso 10. Elimina la entrada de la **lista de procesos**	

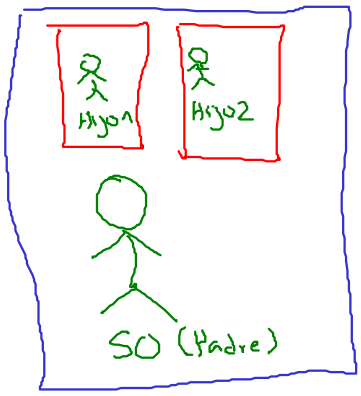






② Ejecución Directa Limitada

Restricciones



Modos

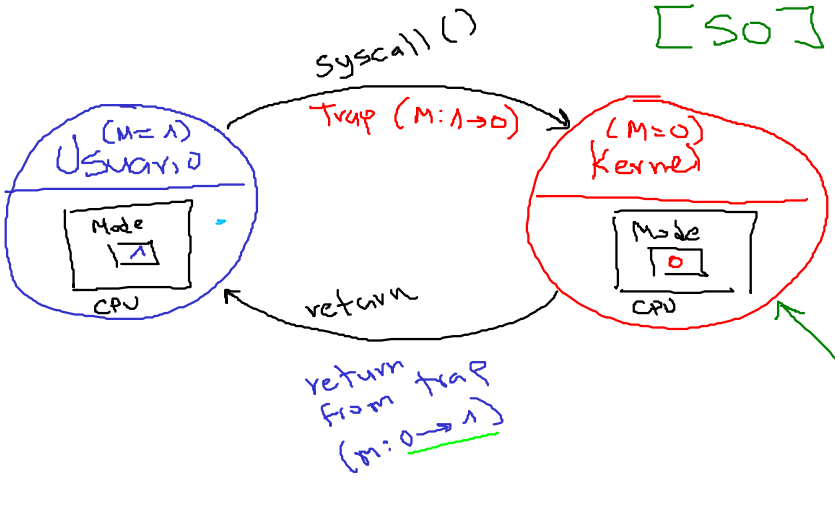
→ CPU { Modo: 0 → K ; Modo: 1 → U

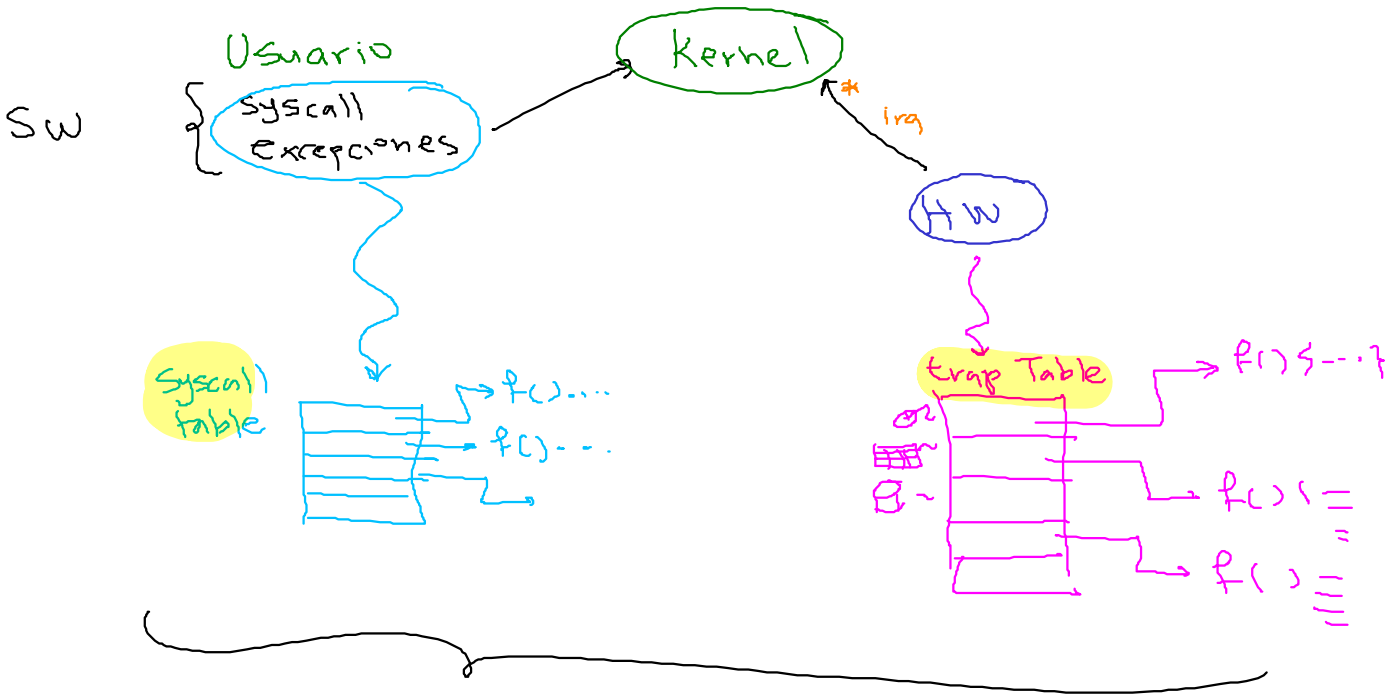
Kernel
- Acceso total
- HW
- Todas las instrucciones del ISA

[SO]

User
- Acceso Rest
- ~~HW~~
- Solo algunas instrucc del ISA

[Apps]





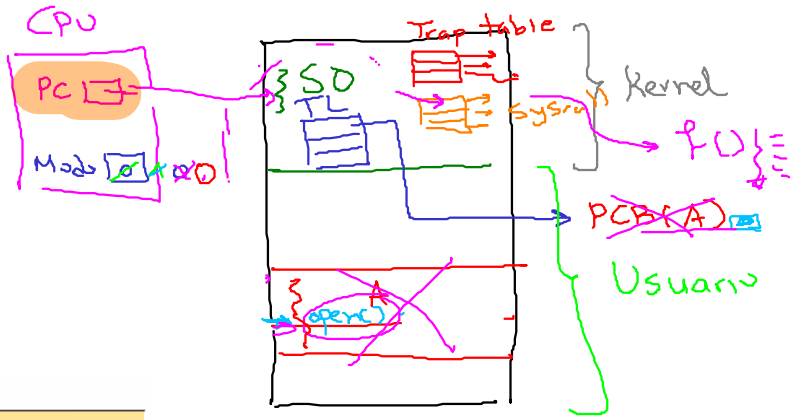
Estructuras del Kernel

1. Lista de procesos
2. Syscall table
3. trap table

Protocolo

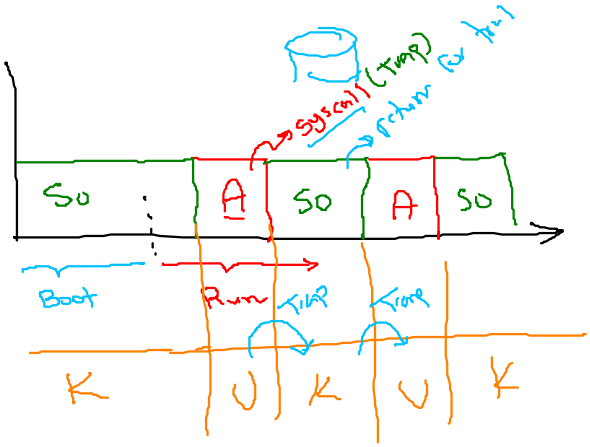
- SO @ Boot

SO @ Boot (Modo kernel)	Hardware
• Inicializa la trap table	• Almacena la dirección del Syscal handler • Ejecuta `return` de `main()`

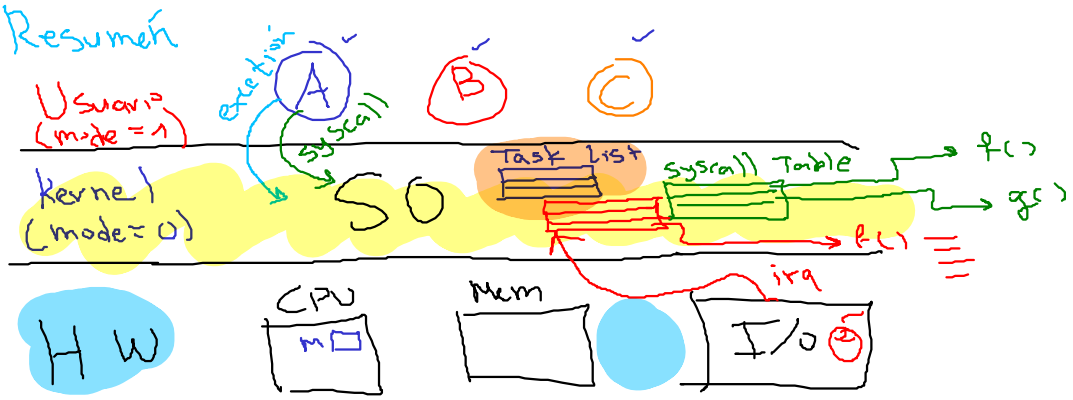


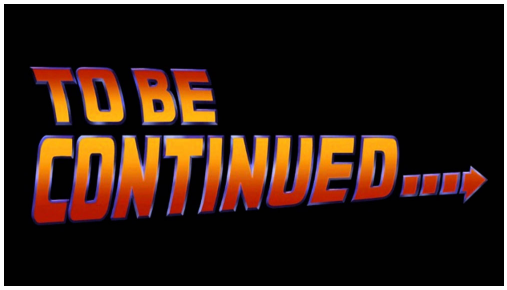
- SO @ Run

SO @ Run (Modo kernel)	Hardware (CPU)	Programa (Modo usuario)
• Crea entrada en la lista de procesos • Asigna memoria al proceso • Carga el programa a memoria • Inicializa pila (stack) con `argc/argv` • Almacena valores de registros en el `kernel stack` • `Return-from-trap`	• Inicializa registros desde el kernel stack • Pasa a modo usuario • Salta a `main()`	• Ejecuta `main()` • ... • Llamado al sistema (`trap`)



SO @ Run (Modo kernel)	Hardware	Programa (Modo usuario)
	• Almacena valores de registros en el kernel stack • Pasa a modo kernel • Salta al trap handler	
• Atiende el evento `trap` • Realiza el trabajo solicitado por la llamada al sistema • `Return-from-trap`		
	• Restaura valores de registros desde el kernel stack • Pasa a modo usuario • Salta a PC después de `trap`	
		• ... • return de `main()` • Llamado al sistema `exit` (`trap`)
• Libera la memoria del proceso • Elimina la entrada de la lista de procesos.		



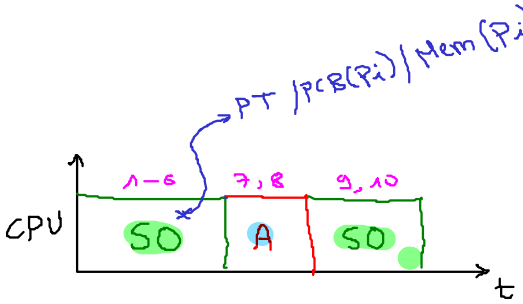


28/08/2025 - Sistemas Operativos (Vde@)

(3) Repaso clase anterior [Inicio del Repaso]

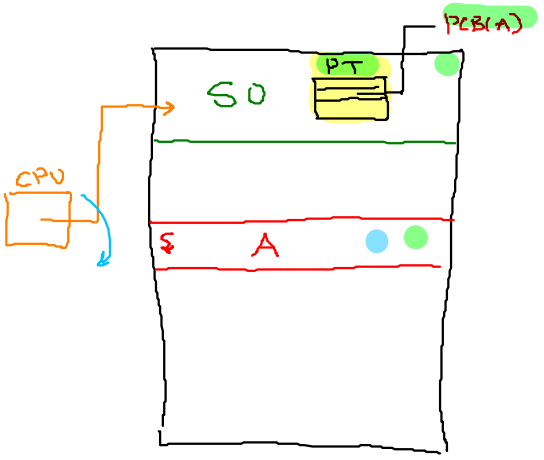
LDE - Ejecucion Directa Limitada

* Ejecucion directa: Todos usan la CPU

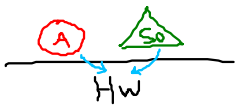


Protocolo

Sistema Operativo	Proceso de usuario
1. Crea una entrada en la **lista de procesos** 2. Asigna memoria al proceso 3. Carga el programa a memoria 4. Inicializa pila (*stack*) con `argc/argv` 5. Reinicia registros de CPU 6. Ejecuta llamada a `main()`	
	7. Ejecuta `main()` 8. Ejecuta `return` de `main()`
9. Libera la memoria del proceso 10. Elimina la entrada de la **lista de procesos**	



Problemas:



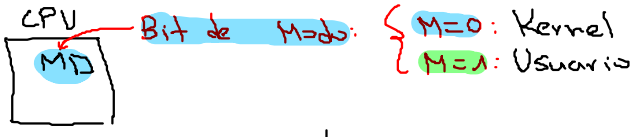
1. Operaciones restringidas

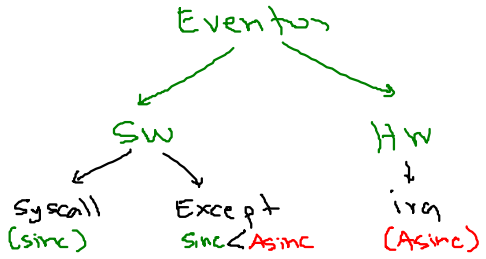
Desde aqui empezamos la clase

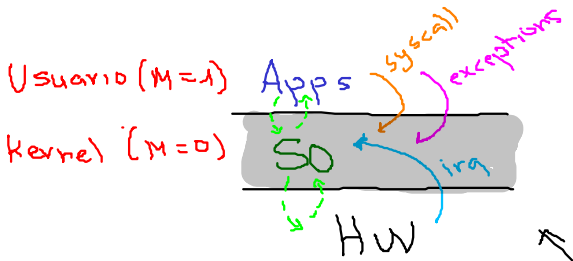
2. Como hacer el cambio de contexto

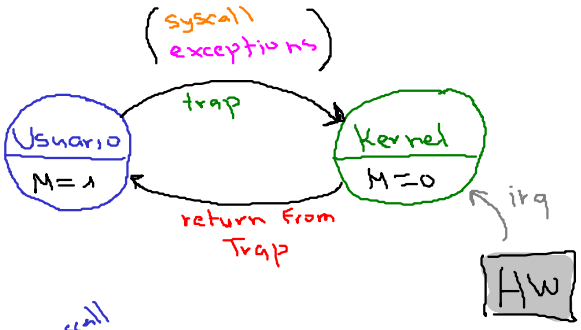
Como el SO vuelve a recuperar el control

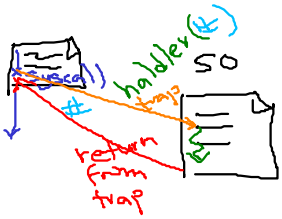
* Ejecucion Directa Limitada

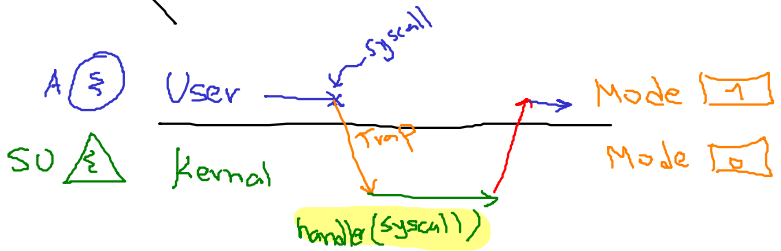






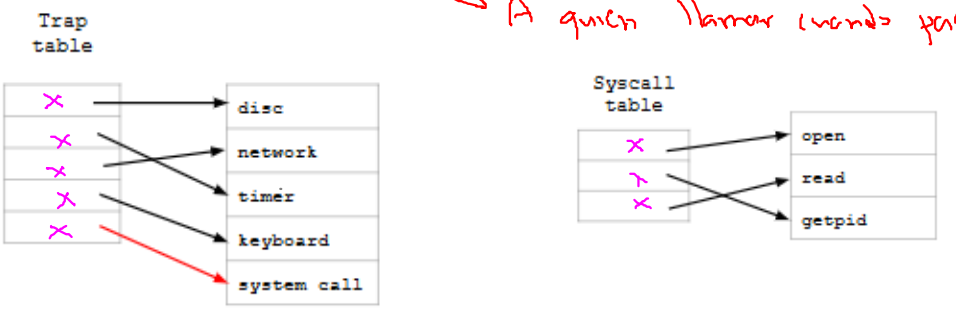




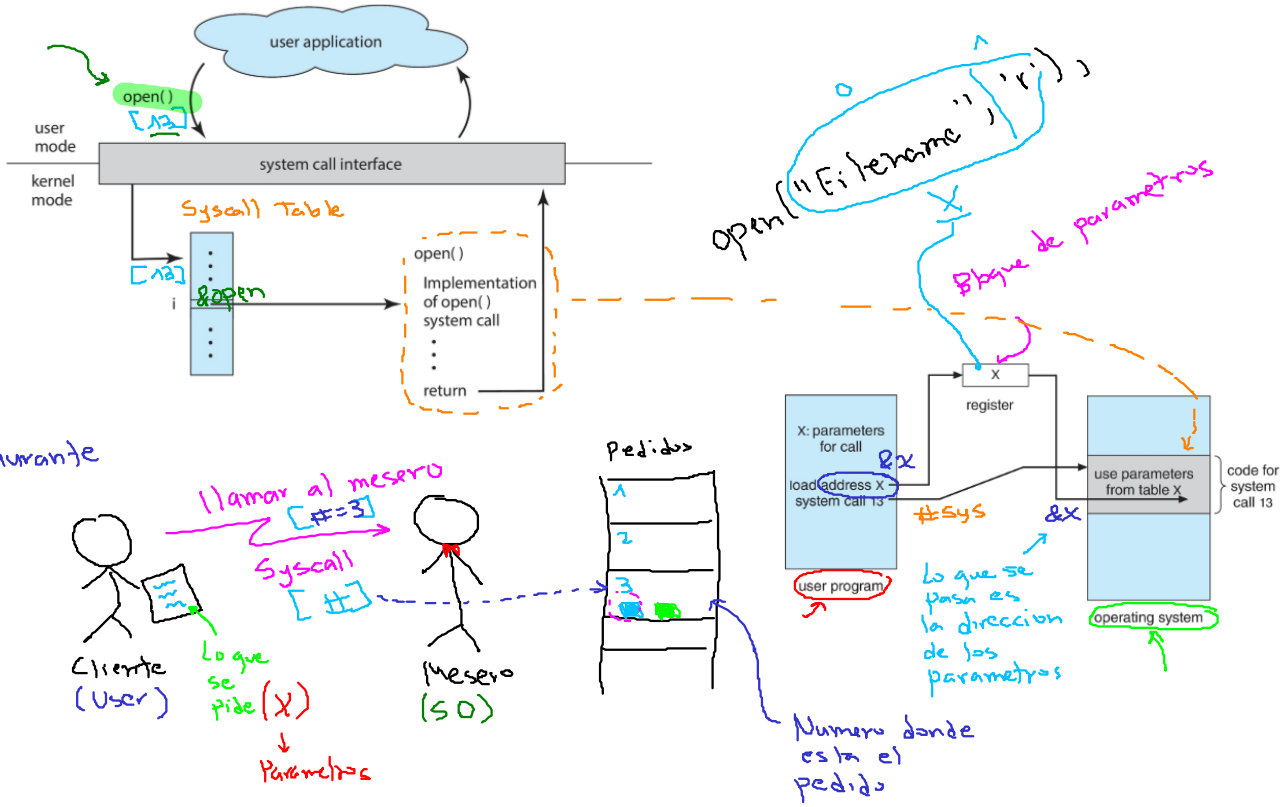


Necesitamos ademas unas nuevas estructuras de datos

→ A quien llamar cuando pasa algo

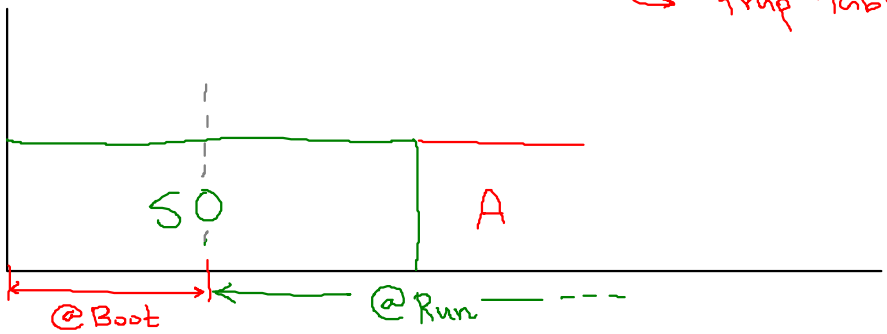


Llamadas de sistema y sistema operativo.



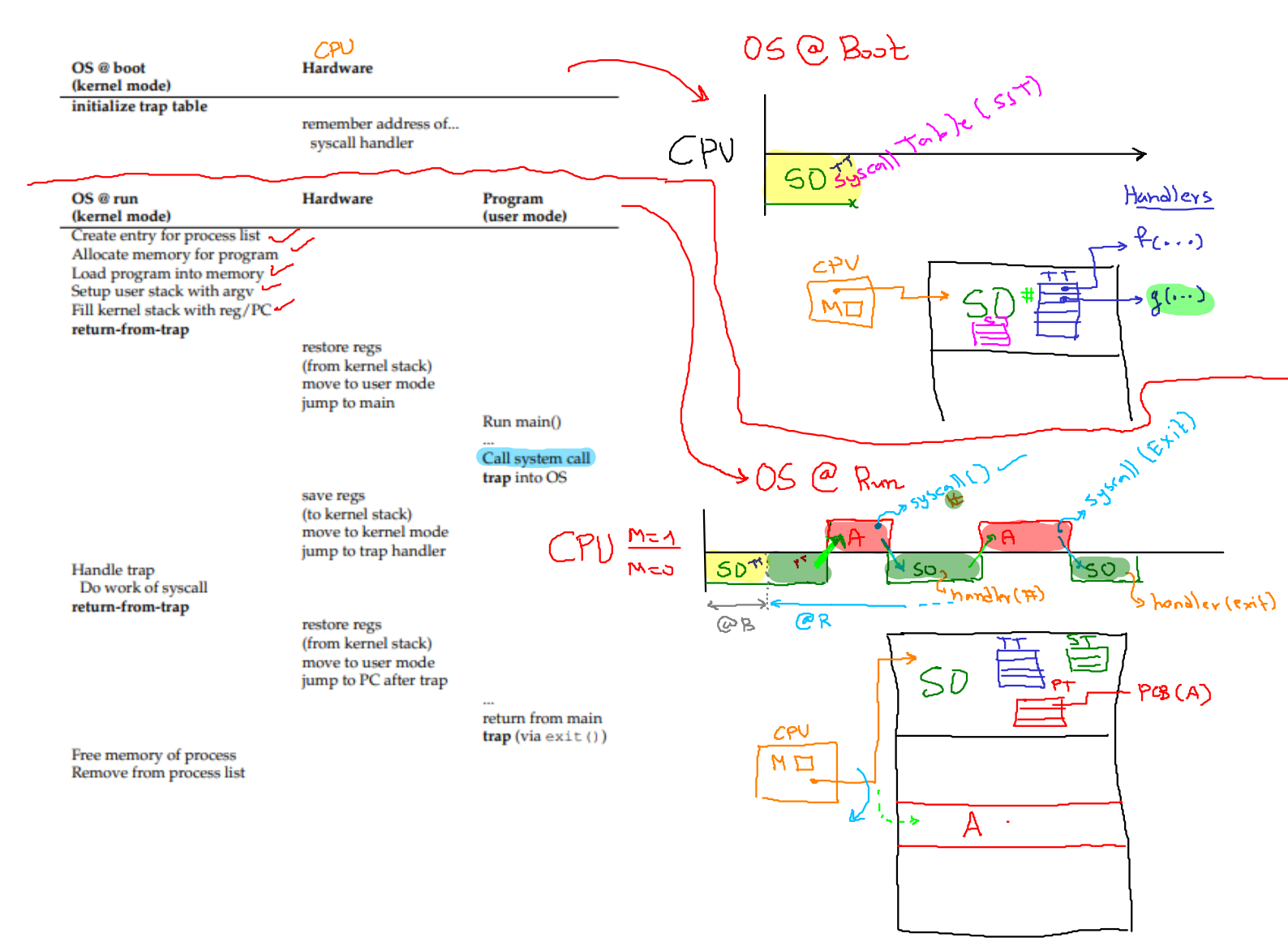
Protocolo de ejecucion directa Limitada

→ Syscall
→ trap table

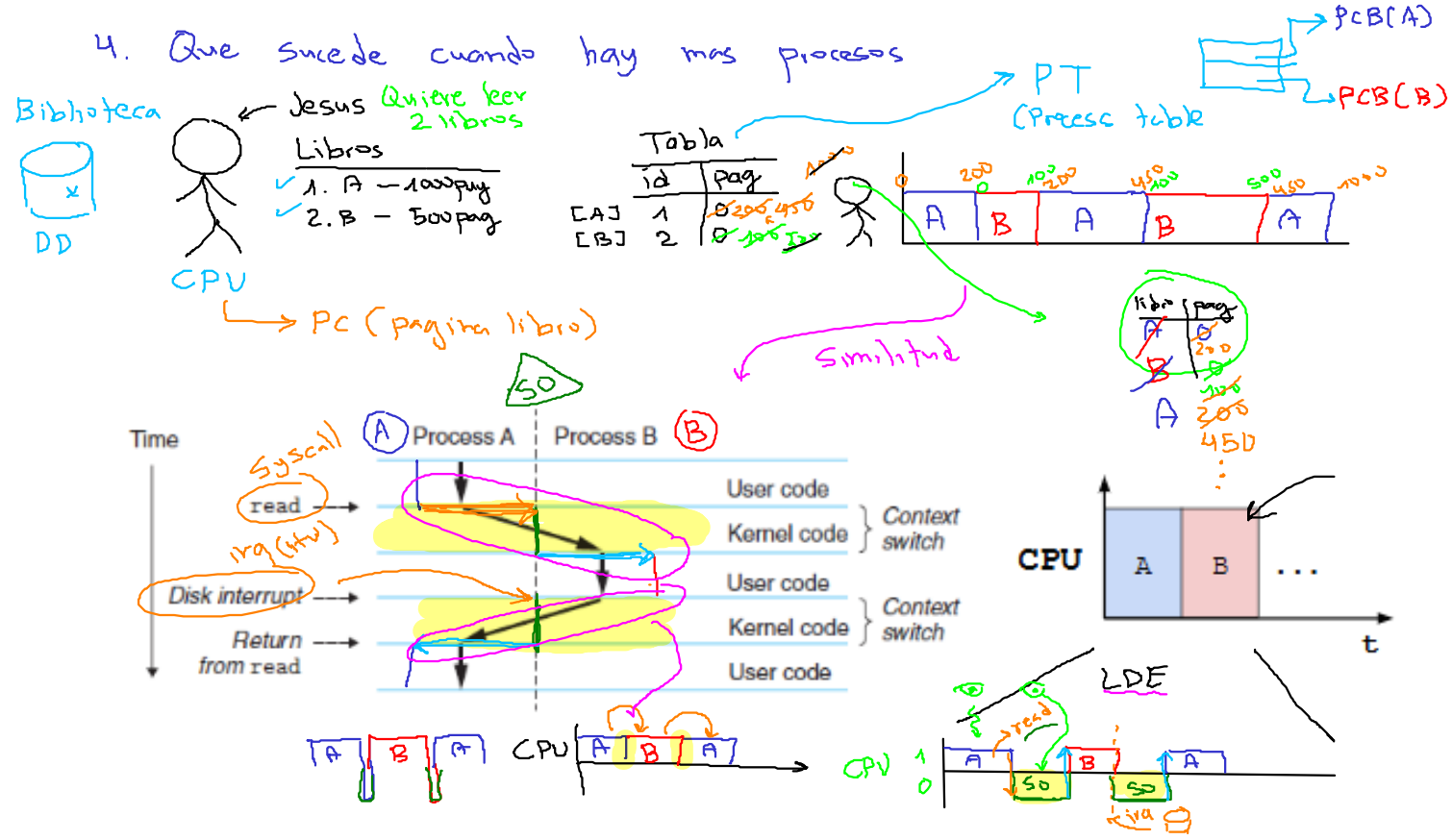


Ver pagina siguiente →

OS @ boot (kernel mode)	Hardware	Program (user mode)
initialize trap table		
	remember address of... syscall handler	
OS @ run (kernel mode)	**Hardware**	**Program (user mode)**
Create entry for process list		
Allocate memory for program		
Load program into memory		
Setup user stack with argv		
Fill kernel stack with reg/PC		
return-from-trap		
	restore regs (from kernel stack)	
	move to user mode	
	jump to main	
		Run main()
		...
		Call system call
		trap into OS
	save regs (to kernel stack)	
	move to kernel mode	
	jump to trap handler	
Handle trap		
Do work of syscall		
return-from-trap		
	restore regs (from kernel stack)	
	move to user mode	
	jump to PC after trap	
		...
		return from main
		trap (via `exit()`)
Free memory of process		
Remove from process list		



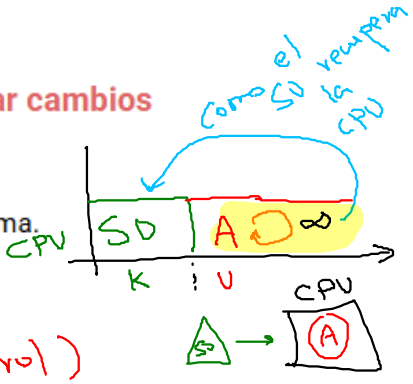
Fin del Repaso



¿Cómo hace el SO para retomar el control de la CPU y realizar cambios entre procesos?

Hay dos formas:

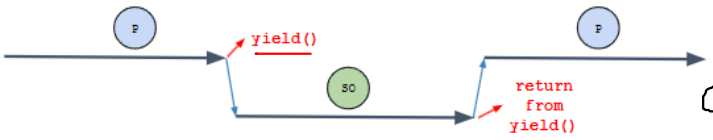
1. **Propuesta cooperativa**: Se cede el control mediante llamados a sistema.
2. **Propuesta no-cooperativa**: El SO toma el control.

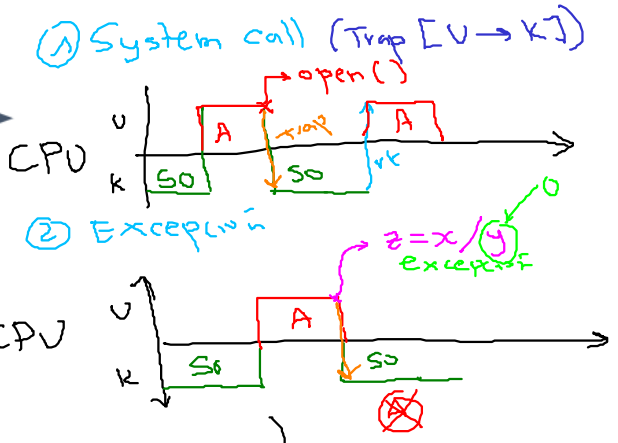


Propuesta cooperativa (no apropiativa)

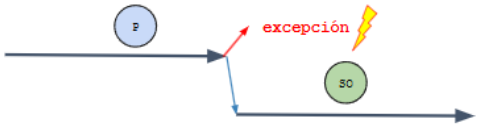
(La App cede el control)

- Los procesos liberan la CPU periódicamente
- El cambio de contexto se da empleando llamadas a sistema.
  - Empleo de la llamada a sistema `yield()`.

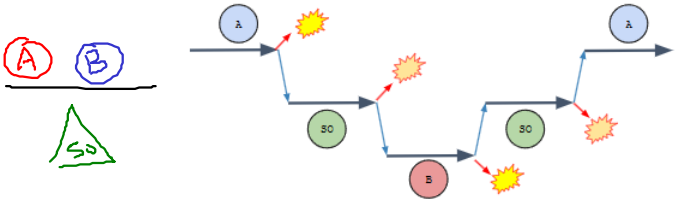


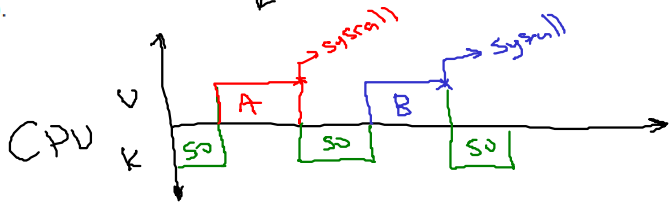


  - Operación ilegal: División por cero, acceso ilegal a memoria,etc.



- Cuando el proceso libera la CPU, es el sistema operativo quien decide el próximo proceso que la usara (Como se hace lo veremos luego).

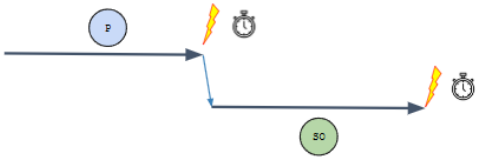


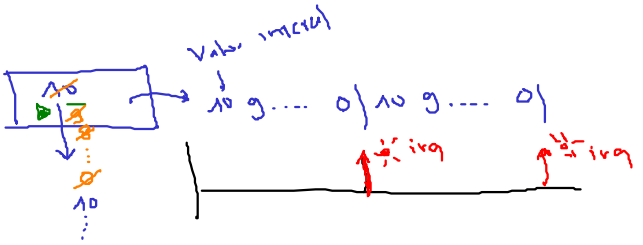


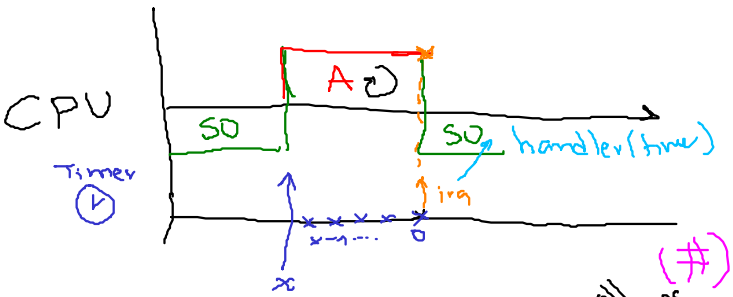
Propuesta no cooperativa (apropiativa)

(Hw) → Timer

- El sistema operativo es quien toma el control de la CPU mediante interrupciones por timer.
- El timer lanza interrupciones cada cierto tiempo (algunos milisegundos).

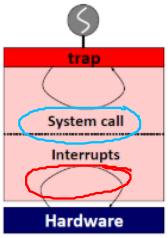


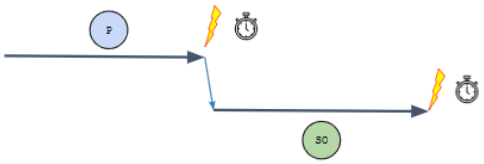


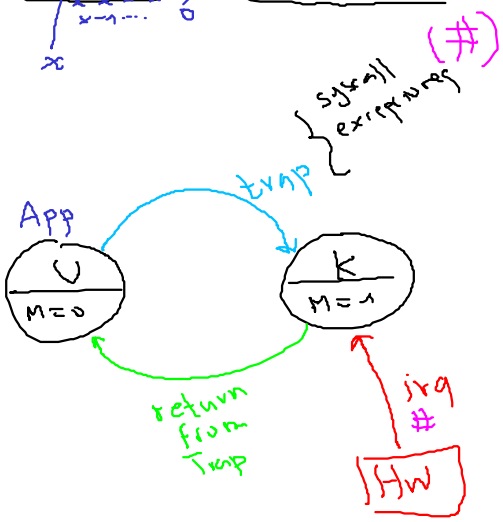


- Cuando se lanza una ejecución por timer:
  1. El proceso en ejecución es **suspendido**.
  2. Se almacena el estado del proceso.
  3. Una **rutina de atención** a la interrupción es ejecutada

- La interrupción del timer le da a la SO la habilidad de **retomar** el control de la CPU.

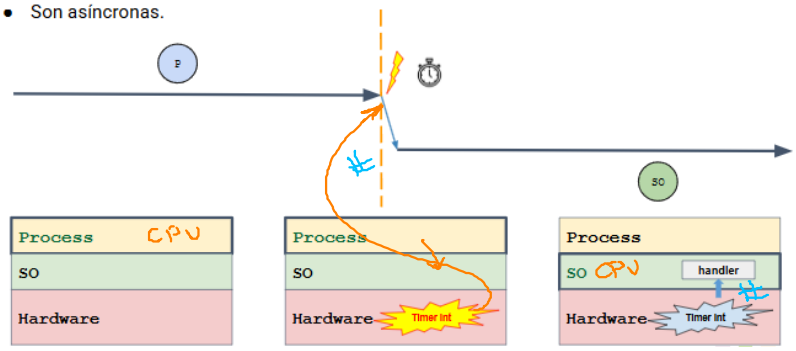




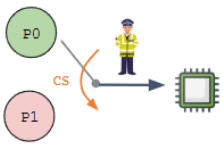


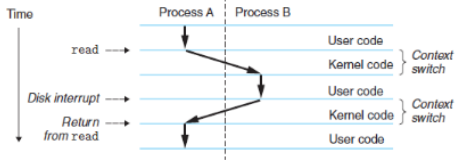
Sobre las interrupciones

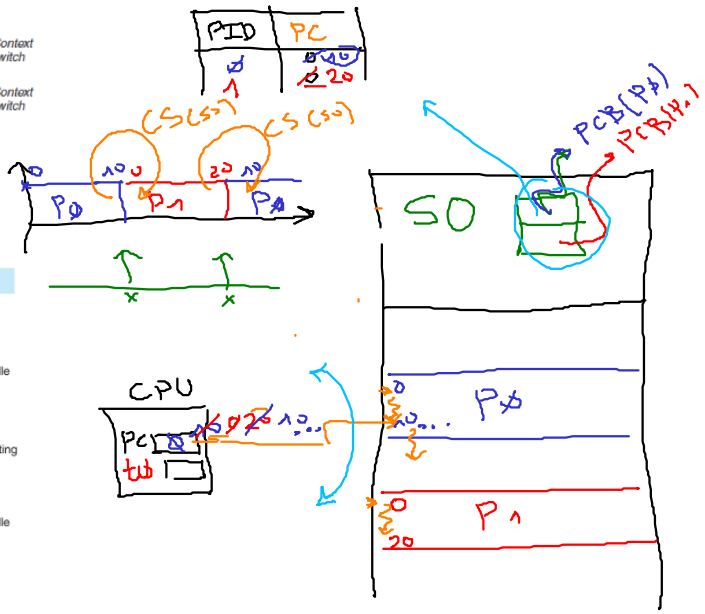
- Las interrupciones son generadas por hardware.
- Son asíncronas.



- Cuando hay una interrupción el scheduler tiene que tomar una decisión sobre el paso a seguir:
  1. Continuar con el proceso actual.
  2. Cambiar a un proceso diferente.
- Si la decisión es cambiar, el SO ejecuta un cambio de contexto (**context switch**).



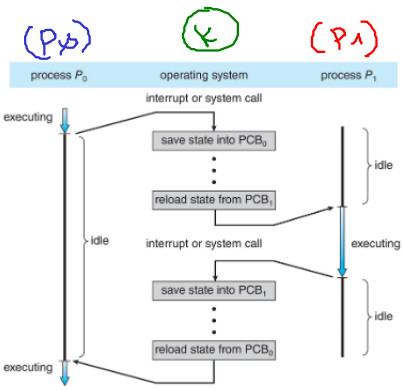




Cambio de contexto

Cuando hay un cambio de contexto pasa lo siguiente:

- Se almacena el estado del proceso en ejecución en el PCB, esto implica almacenar los valores de la CPU en el **kernel stack**:
  1. Registros de propósito general.
  2. PC.
  3. kernel Stack pointer.
- Se restauran los valores del proceso que se va a ejecutar desde el **kernel stack**.
- Se cambia al **kernel stack** del proceso que se va a ejecutar.



Protocolo LDE (Timer interrupt)

Limited Direct Execution Protocol (Timer Interrupt)

OS @ boot (kernel mode)	Hardware	Program (user mode)
initialize trap table		
	remember addresses of...	
	syscall handler	
	timer handler	
start interrupt timer		
	start timer	
	interrupt CPU in X ms	

OS @ run (kernel mode)	Hardware	Program (user mode)
		Process A
		...
	timer interrupt	
	save regs(A) → k-stack(A)	
	move to kernel mode	
	jump to trap handler	
Handle the trap		
Call `switch()` routine		
save regs(A) → proc_t(A)		
restore regs(B) ← proc_t(B)		
switch to k-stack(B)		
return-from-trap (into B)		
	restore regs(B) ← k-stack(B)	
	move to user mode	
	jump to B's PC	
		Process B
		...