

Seudo-Scheduler de Tareas

Organización del Computador II

Leandro Ezequiel Barrios

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

2018-06-05

Como viene la mano,

Modo Real

Modo Protegido

Paginación

Interrupciones

Tareas

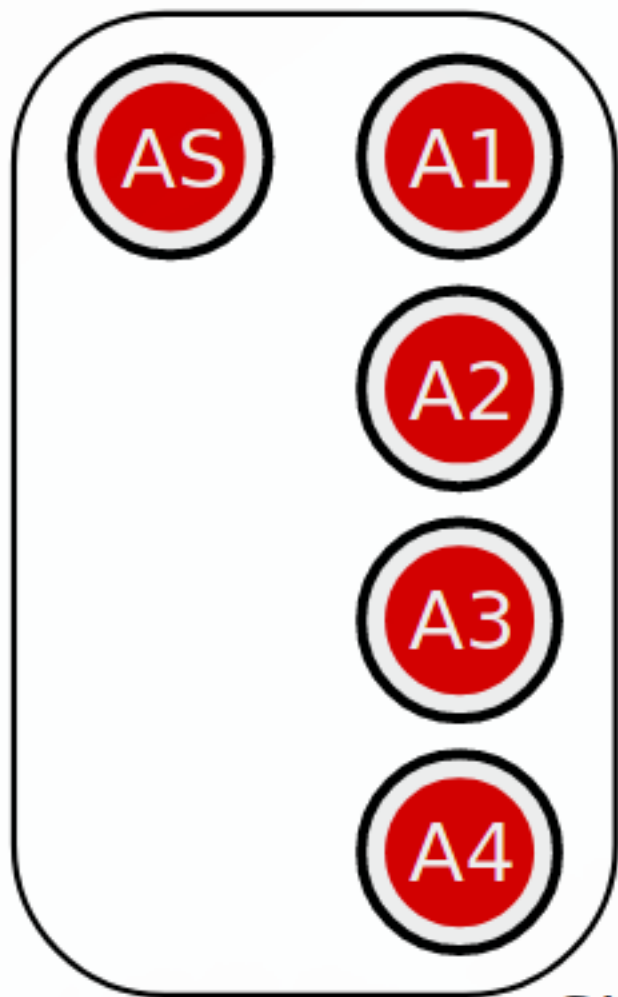
- crear un mapa de memoria
- mapear una página
- desmapear una página
- cargar una entrada de IDT
- escribir una rutina de interrupción
- rellenar las TSS
- cargar los descriptores de TSS en la GDT
- saltar a una tarea (jmp)

Como viene la mano,

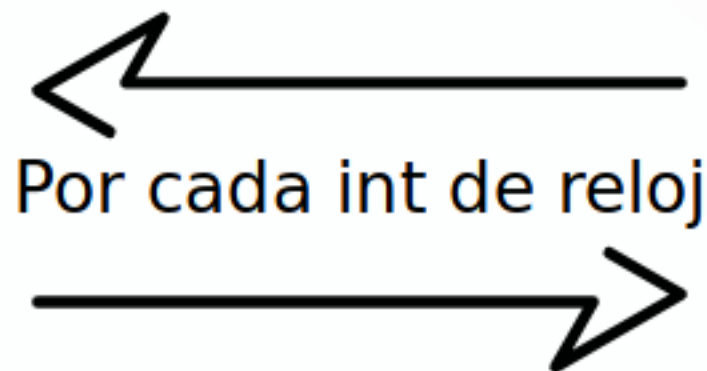
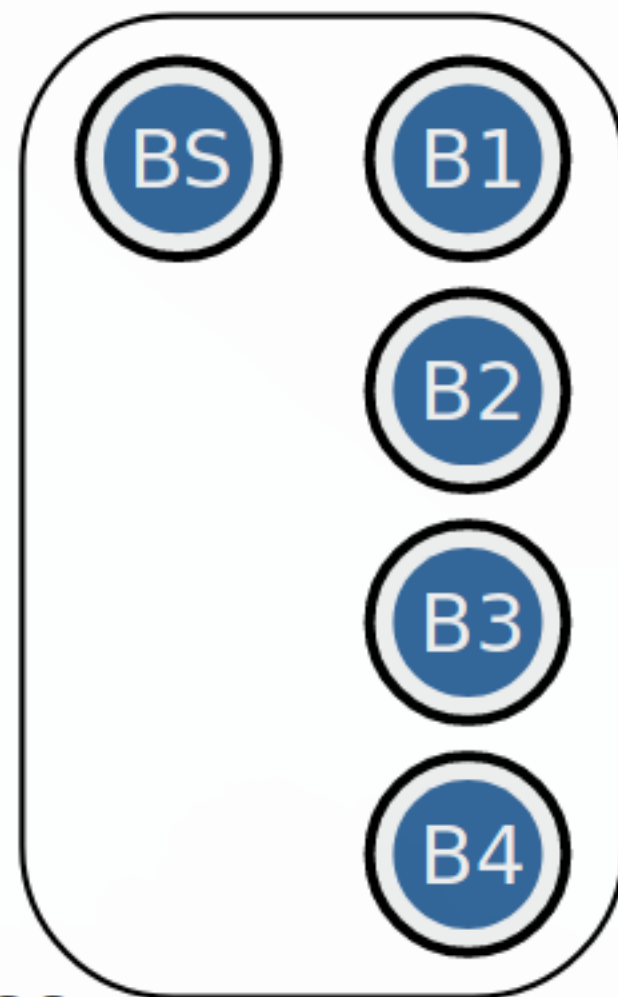
- crear un mapa de memoria
 - `mmu_inicializar(), mmu_inicializar_dir_kernel(), mmu_inicializarDirTarea(), ...`
- mapear una página
 - `void mmu mapearPagina(unsigned int virtual, unsigned int cr3, unsigned int fisica)`
- desmapear una página
 - `void mmu unmapearPagina(unsigned int virtual, unsigned int cr3)`
- cargar una entrada de IDT
 - `void idt_inicializar()`
- escribir una rutina de interrupción
 - `void _isrN() ...`
- rellenar las TSS
 - `void tss_inicializar()`
- cargar los descriptores de TSS en la GDT
 - (hacer en GDT)
- saltar a una tarea (jmp)
 - `offset: dd 0`
`selector: dw 0`
`...`
`mov [selector], ax`
`jmp far [offset]`

Seudo-Scheduler de tareas

Jugador A



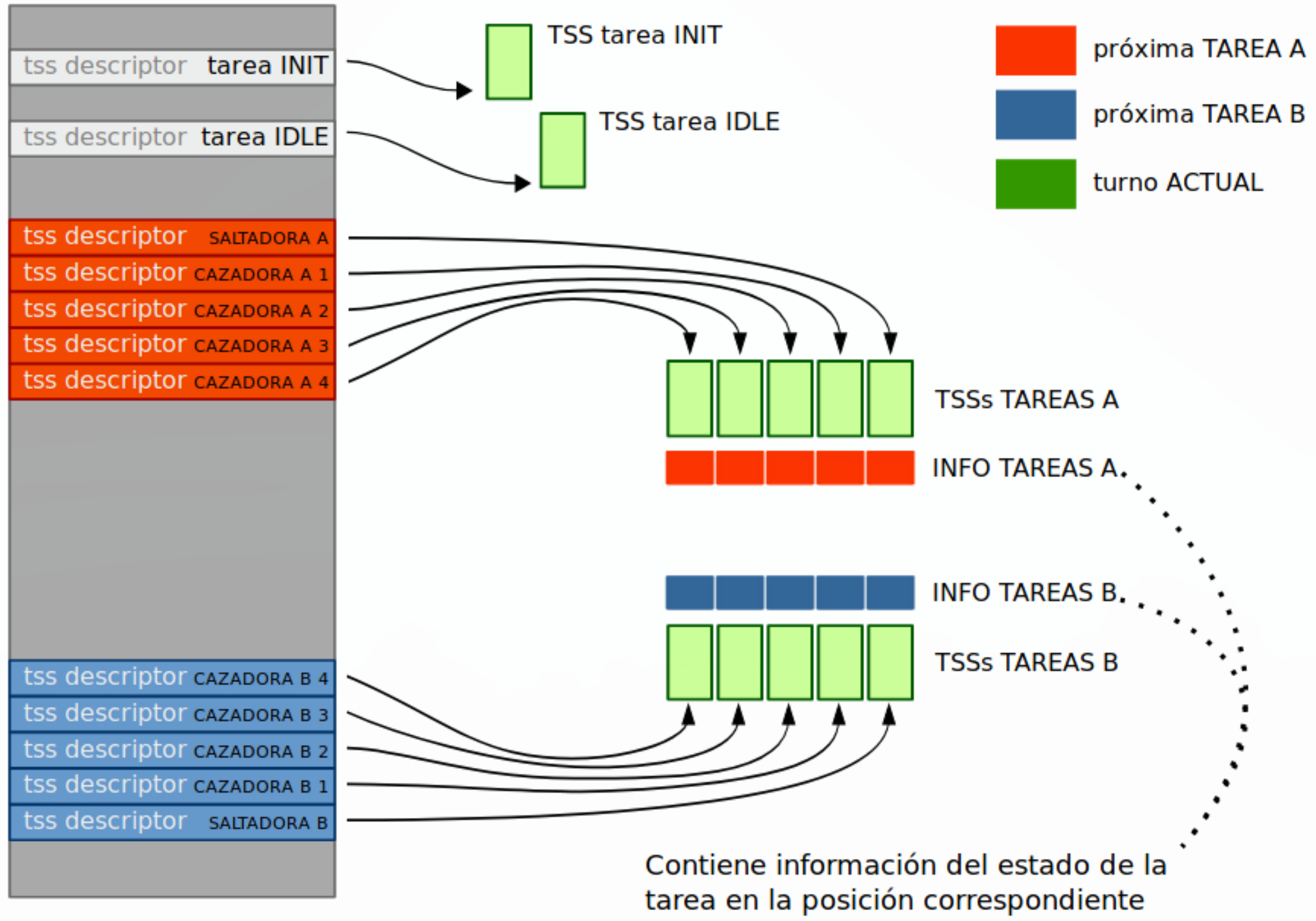
Jugador B



Si una tarea llamó al syscall 0x66
o produjo una excepción

Estructuras del Seudo-Scheduler

GDT



Estructuras del Juego



Jugador A



Jugador B

Contiene información de cada jugador: cantidad de vidas, estado de las tareas cazadoras, ...



INFO TAREA A




INFO TAREA B

Contiene información de la posición y tipo de cada tarea para cada jugador en el caso que exista.

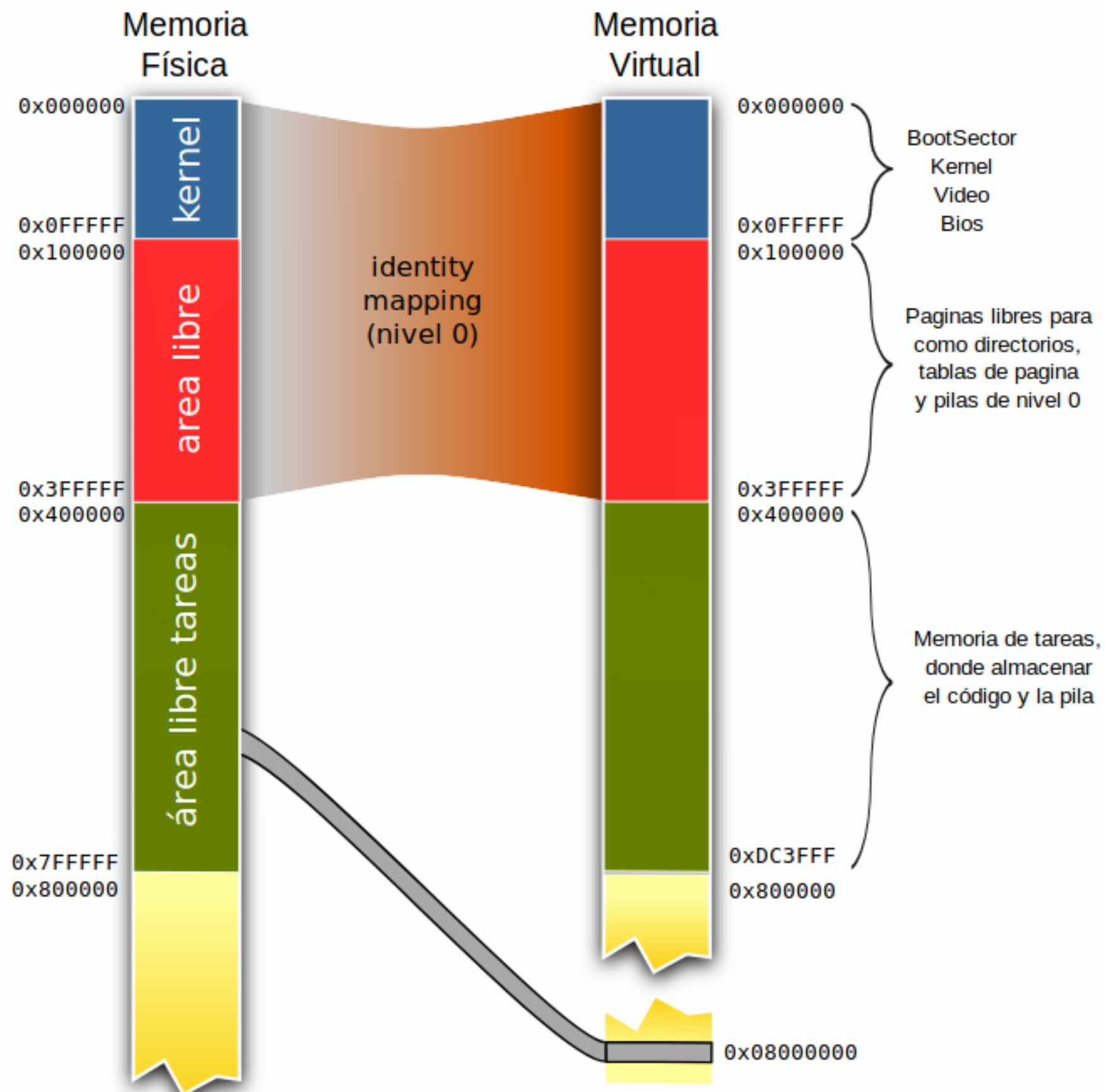
Completar una tss (PARA UNA TAREA)

31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
Reserved	GS		92
Reserved	FS		88
Reserved	DS		84
Reserved	SS		80
Reserved	CS		76
Reserved	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
Reserved	SS2		24
ESP2			20
Reserved	SS1		16
ESP1			12
Reserved	SS0		8
ESP0			4
Reserved	Previous Task Link		0

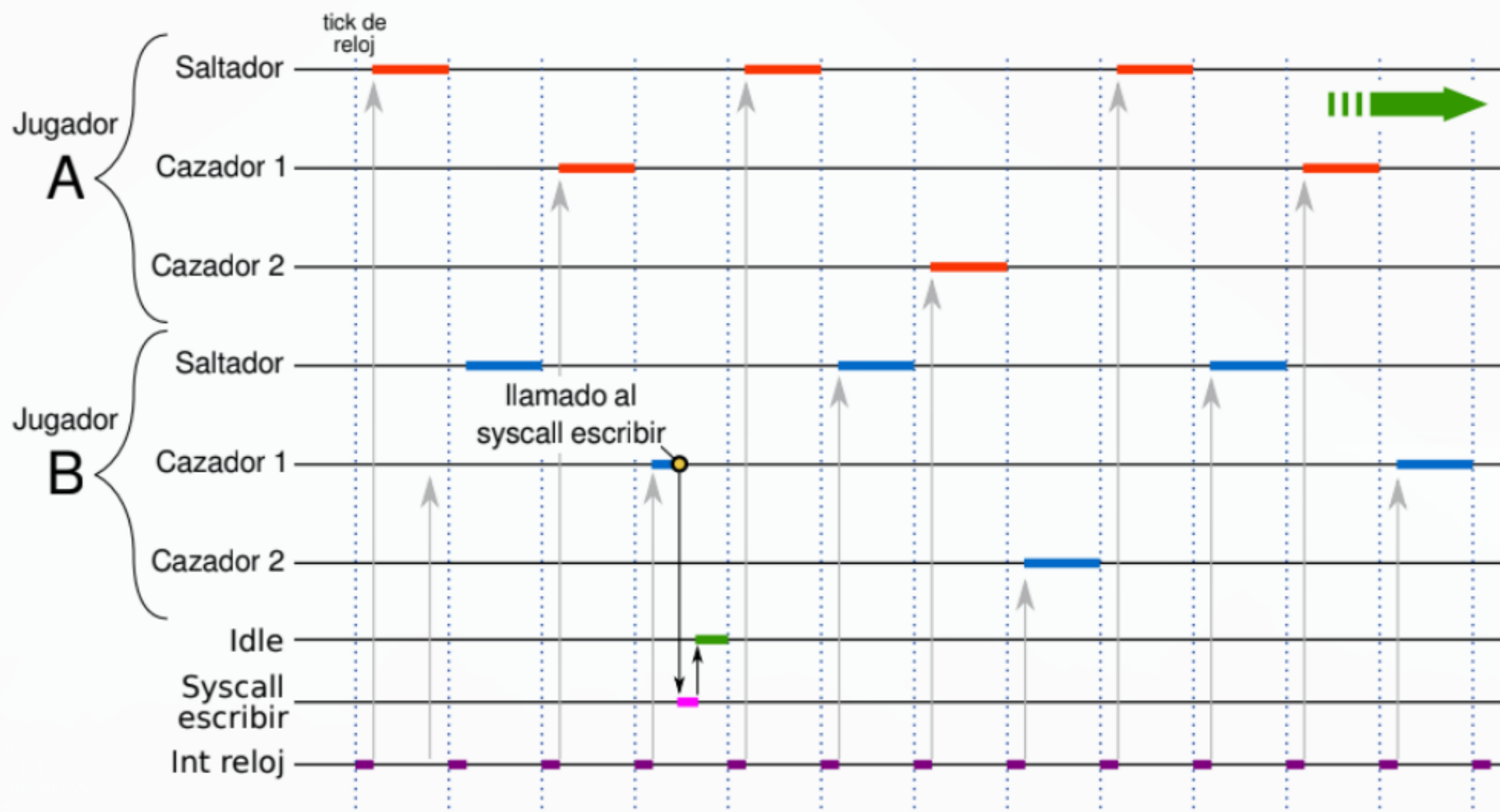
 Reserved bits. Set to 0.

- **cs** = selector de código
(¡cuidado con el nivel!)
- **es, gs ... ss** = selector de datos
(¡cuidado con el nivel!)
- **eax ... edx** = default...
- **esi ... edi** = default...
- **esp ... ebp** = puntero a la pila
- **eip** = inicio de una tarea
(ver ej 5)
- **eflags** = habilitar interrupciones
(¡ver manual / documentación!)
- **cr3** = nuevo mapa de memoria
- **esp0** = nueva página libre
(¡cuidado con el nivel!)
(¿apunta al principio o al final?)

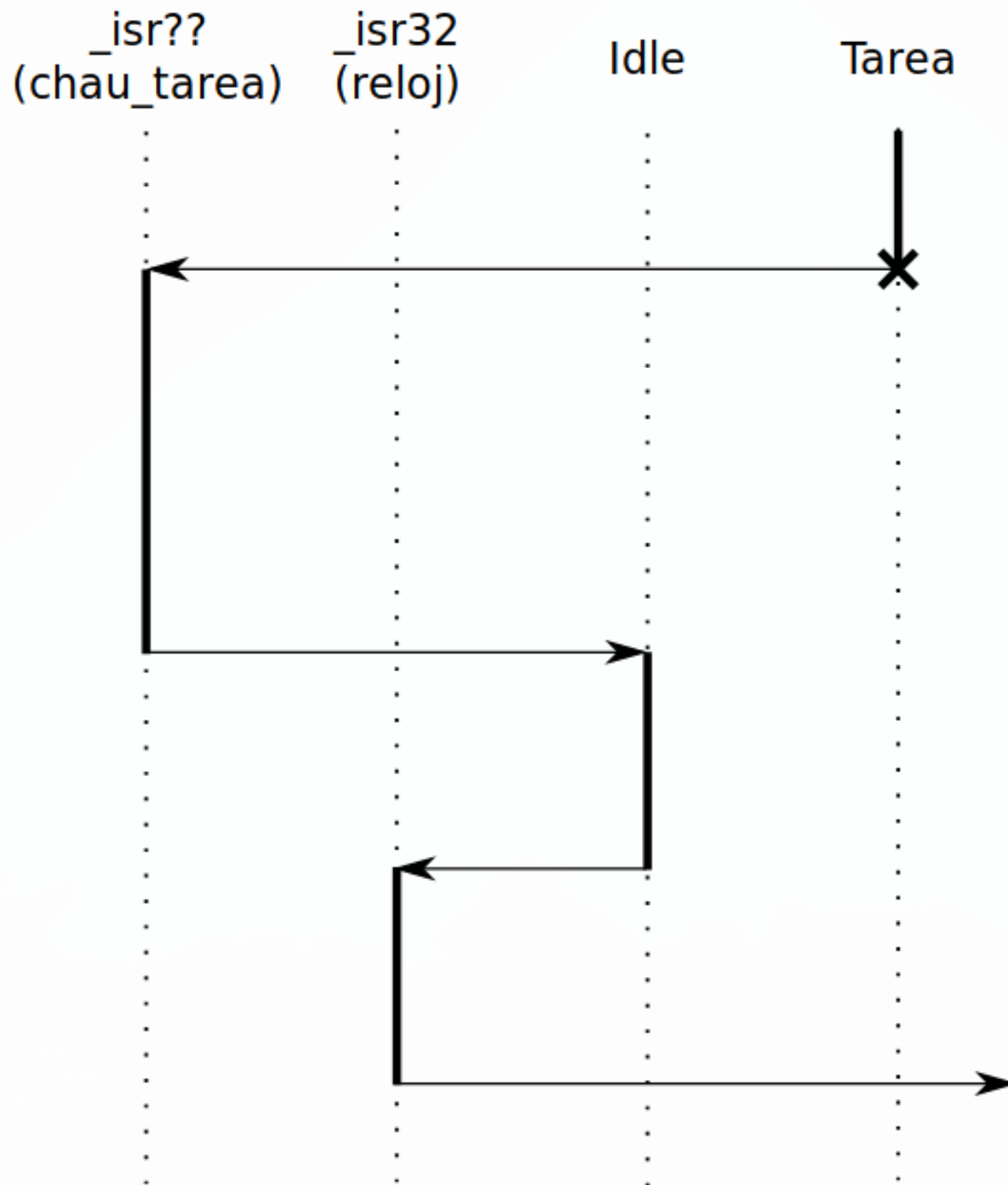
Construyendo el mapa de memoria de una tarea



Ejecutando tareas

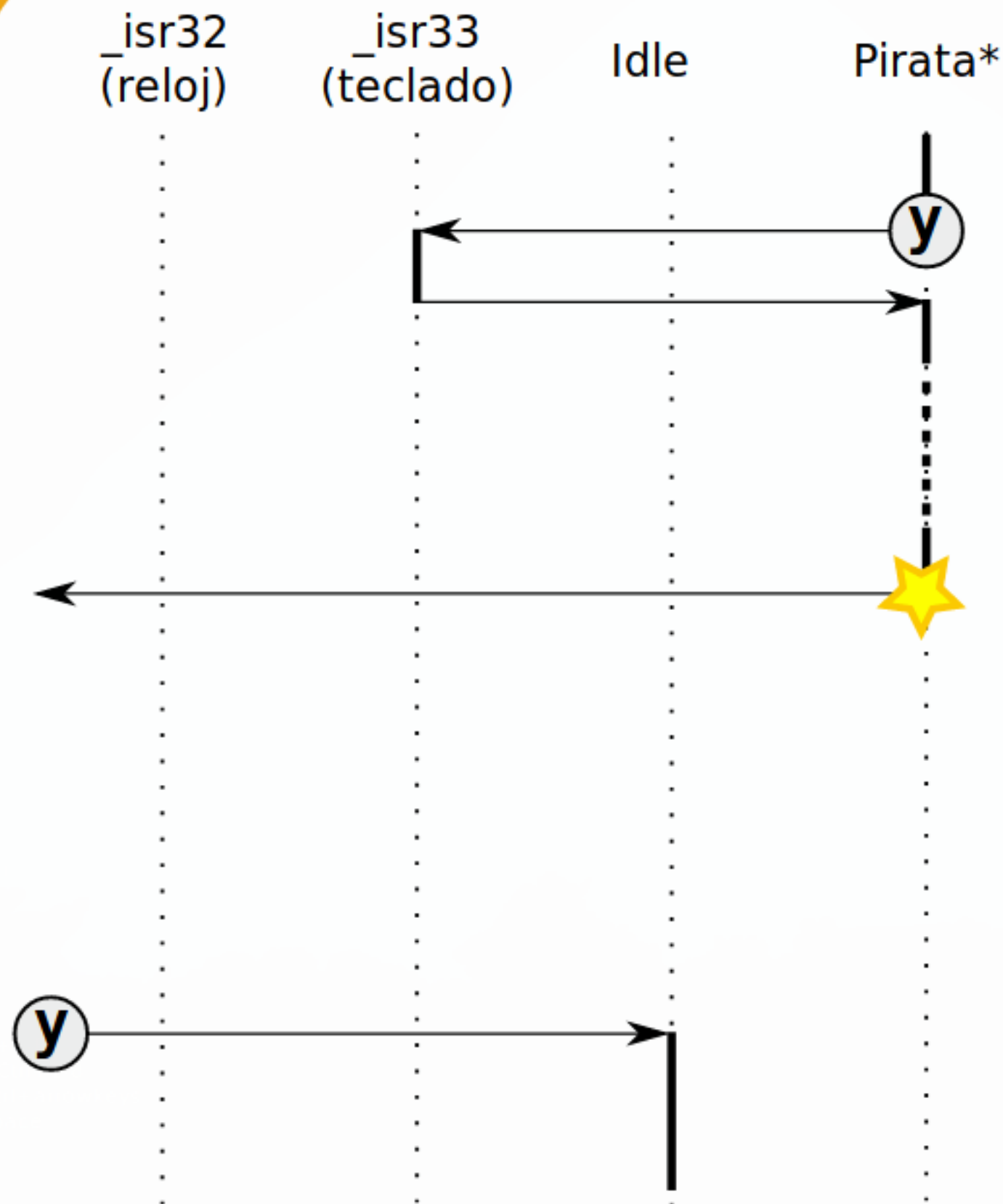


Ejecutando matar una tarea



- Se corre la tarea hasta que ejecuta FRUTA
- Es capturado por alguna excepción del procesador que imprime en pantalla el estado*, mata la tarea y lanza la tarea Idle
- La tarea Idle corre hasta que termina el quantum donde se ejecuta la siguiente tarea

Ejecutando pause de tareas



- Se presiona la tecla "y" y se activa/desactiva el modo debug

- Si alguna tarea produce una excepción, de estar en modo debug se detiene la ejecución mostrando en pantalla el estado

- Cuando se presiona la tecla "y" nuevamente, se reanuda la ejecución

Saltar a una tarea

```
offset: dd 0
selector: dw 0

.
.
mov [selector], ax
jmp far [offset]
.
.
```

¡OJO!

Buscar una tecla

```
.
.
xor eax, eax
in al, 60h
test al, 100000000b
jnz .fin
.
.
```

Reset del pic

```
call fin_intr_pic1
```

Rutina de atención de interrupciones para el reloj

```
global _isr32
```

```
_isr32:
```

```
    pushad
```

```
    call fin_intr_pic1
```

```
    call sched_tick
```

```
    str cx
```

```
    cmp ax, cx
```

```
    je .fin
```

```
        mov [selector], ax
```

```
        jmp far [offset]
```

```
.fin:
```

```
    popad
```

```
    iret
```

resetea el pic

llamar al scheduler

si el próximo es el mismo que el actual, no salta

salta a la proxima tarea y deja la actual guardada en su tss (eip apuntando a .fin)

asm para usar desde C (i386.h)

```
void lcr0(unsigned int val)  
unsigned int rcr0(void)
```

```
void lcr1(unsigned int val)  
unsigned int rcr1(void)
```

```
void lcr2(unsigned int val)  
unsigned int rcr2(void)
```

```
void lcr3(unsigned int val)  
unsigned int rcr3(void)
```

```
void lcr4(unsigned int val)  
unsigned int rcr4(void)
```

```
void tlbflush(void)
```

```
void ltr(unsigned short sel)  
unsigned short rtr(void)
```

```
void breakpoint(void)
```


¡Gracias!

¿Preguntas?