

# Scheduler de tareas

Programación de Sistemas Operativos

David Alejandro González Márquez

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

# Introducción

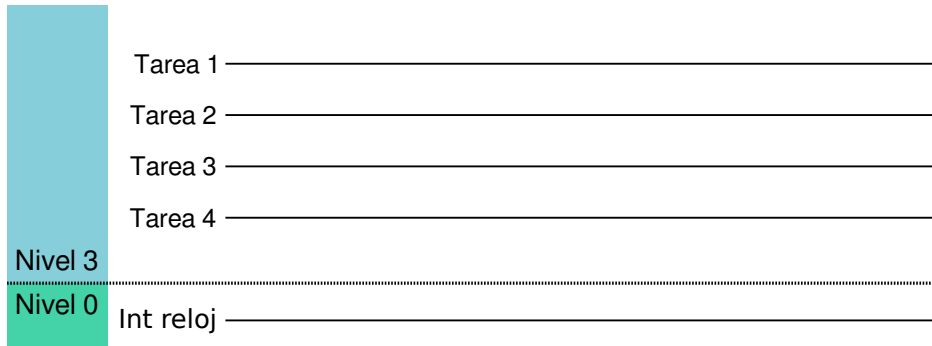
# Introducción

- Un *Scheduler* es un conjunto de rutinas de código que permiten **intercambiar** tareas en algún orden dado.

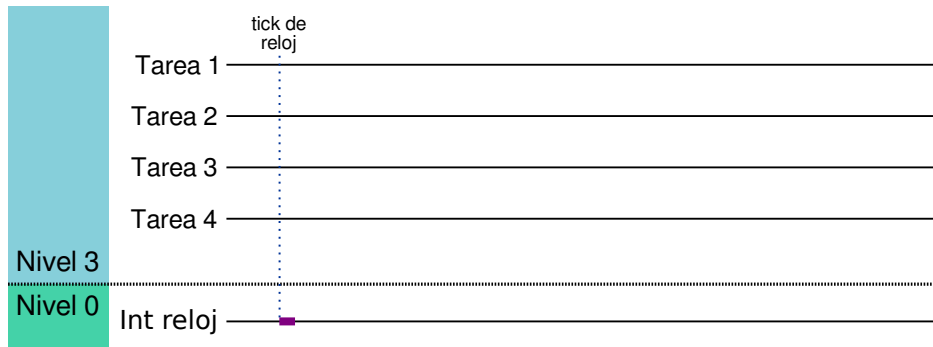
# Introducción

- Un *Scheduler* es un conjunto de rutinas de código que permiten **intercambiar** tareas en algún orden dado.
- En la materia nos vamos a limitar a *Schedulers Round-robin*, donde cada tarea tiene su turno en orden.
- La operatoria del *Scheduler* se va a limitar a la rutina de código ejecutada en la **interrupción de reloj**.

## Scheduler: Operatoria

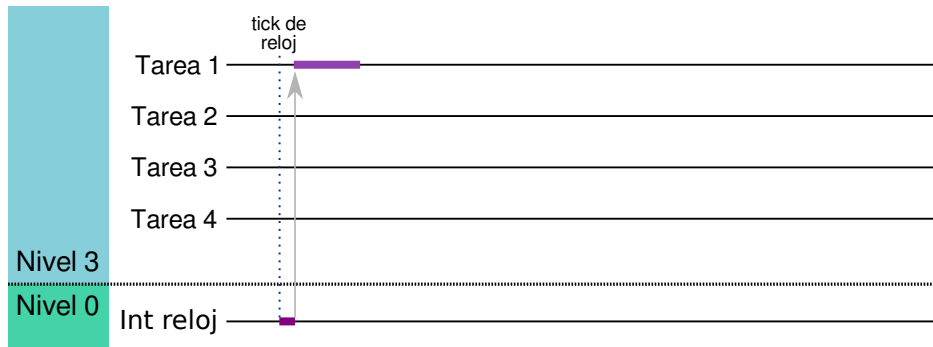


## Scheduler: Operatoria



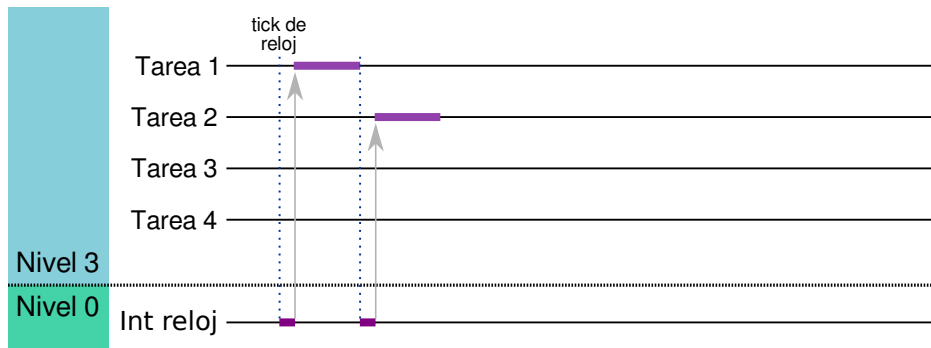
1. Inicialmente cae una interrupción de reloj, y realiza un intercambio de tareas.

## Scheduler: Operatoria



1. Inicialmente cae una interrupción de reloj, y realiza un intercambio de tareas.
2. Se salta al contexto de ejecución de la tarea desde la interrupción de reloj.

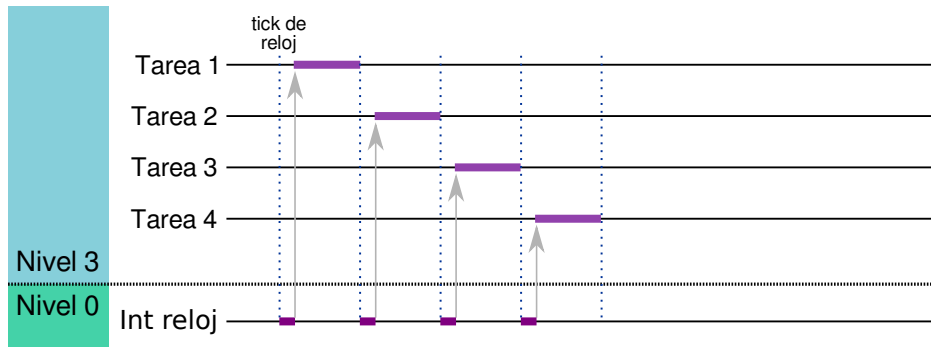
## Scheduler: Operatoria



1. Inicialmente cae una interrupción de reloj, y realiza un intercambio de tareas.
2. Se salta al contexto de ejecución de la tarea desde la interrupción de reloj.
3. Continúa el proceso hasta recorrer todas las tareas en el sistema.

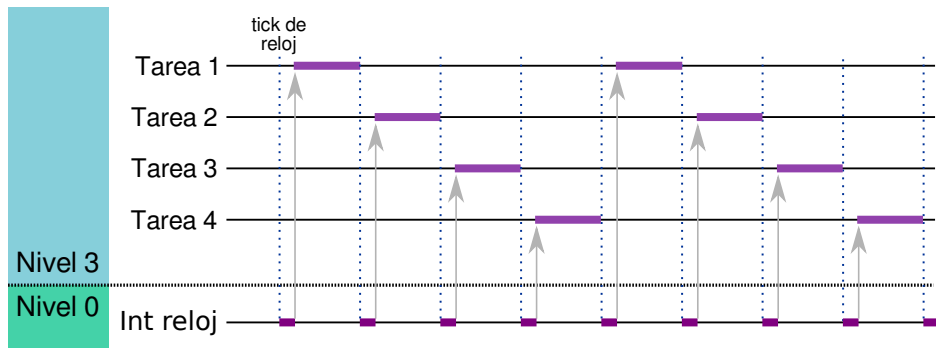


## Scheduler: Operatoria



1. Inicialmente cae una interrupción de reloj, y realiza un intercambio de tareas.
2. Se salta al contexto de ejecución de la tarea desde la interrupción de reloj.
3. Continúa el proceso hasta recorrer todas las tareas en el sistema.

## Scheduler: Operatoria



1. Inicialmente cae una interrupción de reloj, y realiza un intercambio de tareas.
2. Se salta al contexto de ejecución de la tarea desde la interrupción de reloj.
3. Continúa el proceso hasta recorrer todas las tareas en el sistema.
- ∞. Luego comienza nuevamente a recorrer desde la primera tarea.

## Scheduler: GDT, TSS y datos

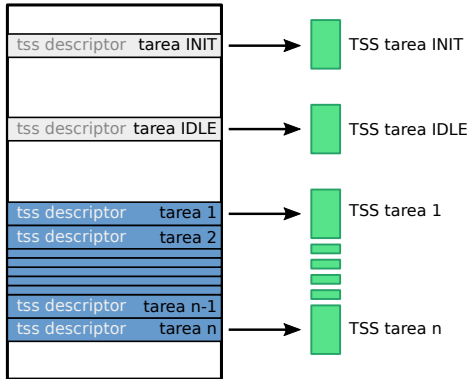
### GDT

tss descriptor    tarea INIT
tss descriptor    tarea IDLE
tss descriptor    tarea 1
tss descriptor    tarea 2
tss descriptor    tarea n-1
tss descriptor    tarea n

- Cada tarea tendrá su propia entrada en la GDT.

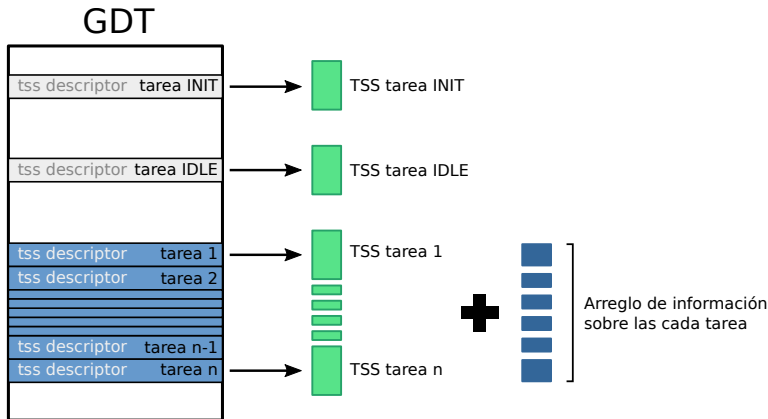
## Scheduler: GDT, TSS y datos

### GDT



- Cada tarea tendrá su propia entrada en la GDT.
- Cada selector de segmento de TSS, apuntará a una TSS.

## Scheduler: GDT, TSS y datos



- Cada tarea tendrá su propia entrada en la GDT.
- Cada selector de segmento de TSS, apuntará a una TSS.
- Además se contará con un arreglo de información configurable sobre cada tarea.

## Scheduler: Intercambio de tareas

- Para saltar a una tarea cualquiera es necesario poder **modificar** el selector de segmento.

## Scheduler: Intercambio de tareas

- Para saltar a una tarea cualquiera es necesario poder **modificar** el selector de segmento.
- Usando: `jmp <selector>:0`. No es posible, ya que `<selector>` es fijo.

## Scheduler: Intercambio de tareas

- Para saltar a una tarea cualquiera es necesario poder **modificar** el selector de segmento.
- Usando: `jmp <selector>:0`. No es posible, ya que `<selector>` es fijo.
- Entonces, se debe usar memoria para indicar el selector de segmento para el intercambio.



## Scheduler: Intercambio de tareas

- Para saltar a una tarea cualquiera es necesario poder **modificar** el selector de segmento.
- Usando: `jmp <selector>:0`. No es posible, ya que `<selector>` es fijo.
- Entonces, se debe usar memoria para indicar el selector de segmento para el intercambio.

Se define en algún lugar del código la siguiente estructura:

```
offset:  dd 0  
selector: dw 0
```

La estructura definida se puede ver como una dirección lógica de 48 bits en *little endian*

## Scheduler: Intercambio de tareas

- Para saltar a una tarea cualquiera es necesario poder **modificar** el selector de segmento.
- Usando: `jmp <selector>:0`. No es posible, ya que `<selector>` es fijo.
- Entonces, se debe usar memoria para indicar el selector de segmento para el intercambio.

Se define en algún lugar del código la siguiente estructura:

```
offset:  dd 0  
selector: dw 0
```

La estructura definida se puede ver como una dirección lógica de 48 bits en *little endian*

En la rutina se utiliza de la siguiente forma:

```
...  
mov [selector], ax ; se carga el selector de segmento  
jmp far [offset]   ; se salta a la direccion logica definida  
...
```

# Scheduler: Código

## Rutina de atención de interrupciones para el reloj

```
global _isr32
_isr32:
    pushad

    call pic_finish1

    call sched_nextTask

    str cx
    cmp ax, cx
    je .fin

    mov [selector], ax
    jmp far [offset]

.fin:

    popad
    iret
```

## Scheduler: Código

### Rutina de atención de interrupciones para el reloj

```
global _isr32
_isr32:
    pushad

    call pic_finish1    ; indicar al pic que la interrupcion fue atendida

    call sched_nextTask

    str cx
    cmp ax, cx
    je .fin

    mov [selector], ax
    jmp far [offset]

.fin:

    popad
    iret
```

## Scheduler: Código

### Rutina de atención de interrupciones para el reloj

```
global _isr32
_isr32:
    pushad

    call pic_finish1    ; indicar al pic que la interrupcion fue atendida

    call sched_nextTask ; obtener indice de la proxima tarea a ejecutar

    str cx
    cmp ax, cx
    je .fin

    mov [selector], ax
    jmp far [offset]

.fin:

    popad
    iret
```

# Scheduler: Código

## Rutina de atención de interrupciones para el reloj

```
global _isr32
_isr32:
    pushad

    call pic_finish1    ; indicar al pic que la interrupcion fue atendida

    call sched_nextTask ; obtener indice de la proxima tarea a ejecutar

    str cx              ; compara con la tarea actual y salta solo si es diferente
    cmp ax, cx
    je .fin

    mov [selector], ax
    jmp far [offset]

.fin:

    popad
    iret
```

# Scheduler: Código

## Rutina de atención de interrupciones para el reloj

```
global _isr32
_isr32:
    pushad

    call pic_finish1    ; indicar al pic que la interrupcion fue atendida

    call sched_nextTask ; obtener indice de la proxima tarea a ejecutar

    str cx              ; compara con la tarea actual y salta solo si es diferente
    cmp ax, cx
    je .fin

    mov [selector], ax  ; carga el selector de segmento de la tarea a saltar
    jmp far [offset]

.fin:

    popad
    iret
```

# Scheduler: Código

## Rutina de atención de interrupciones para el reloj

```
global _isr32
_isr32:
    pushad

    call pic_finish1    ; indicar al pic que la interrupcion fue atendida

    call sched_nextTask ; obtener indice de la proxima tarea a ejecutar

    str cx              ; compara con la tarea actual y salta solo si es diferente
    cmp ax, cx
    je .fin

    mov [selector], ax  ; carga el selector de segmento de la tarea a saltar
    jmp far [offset]    ; intercambio de tareas

.fin:

    popad
    iret
```



## Scheduler: Ejecución de la rutina del reloj

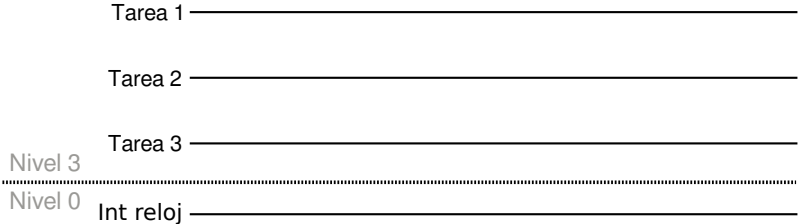
```
_isr32:
    pushad
    call pic_finish1
    call sched_nextTask
    str cx
    cmp ax, cx
    je .fin
    mov [selector], ax
    jmp far [offset]
.fin:
    popad
    iret
```

Int reloj

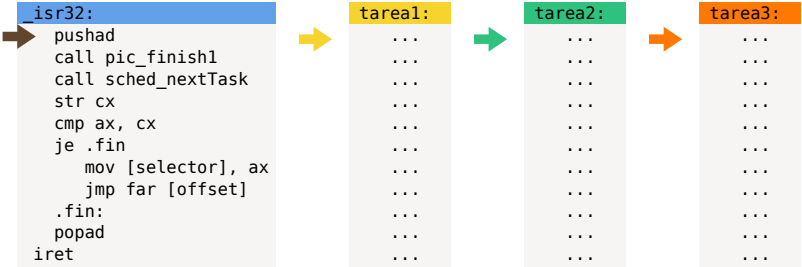
---

# Scheduler: Ejecución de la rutina del reloj

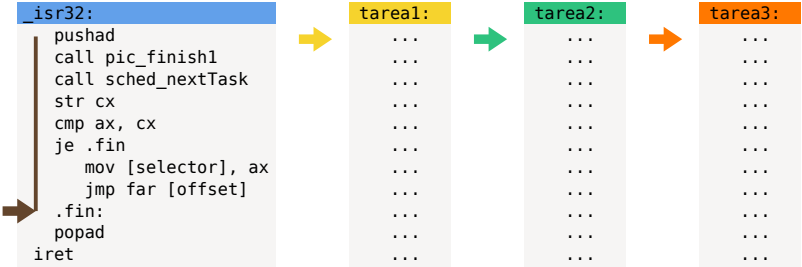
_isr32:	tarea1:	tarea2:	tarea3:
pushad	...	...	...
call pic_finish1	...	...	...
call sched_nextTask	...	...	...
str cx	...	...	...
cmp ax, cx	...	...	...
je .fin	...	...	...
mov [selector], ax	...	...	...
jmp far [offset]	...	...	...
.fin:	...	...	...
popad	...	...	...
iret	...	...	...



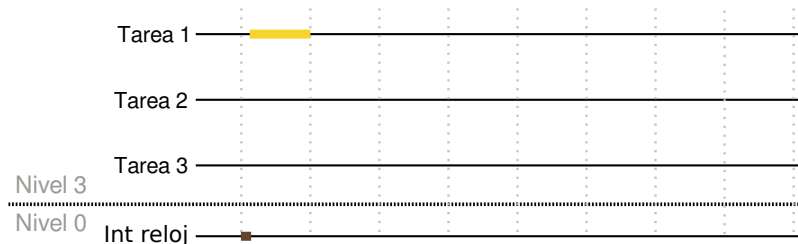
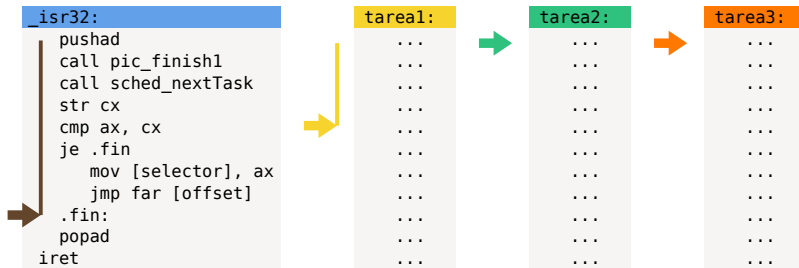
# Scheduler: Ejecución de la rutina del reloj



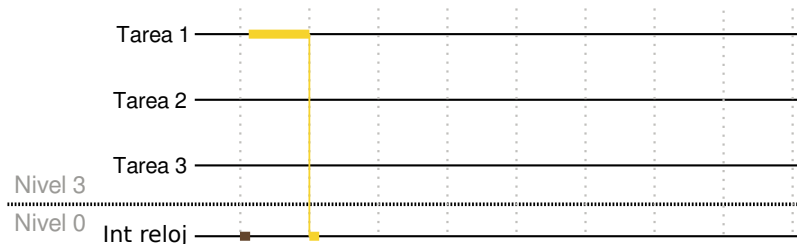
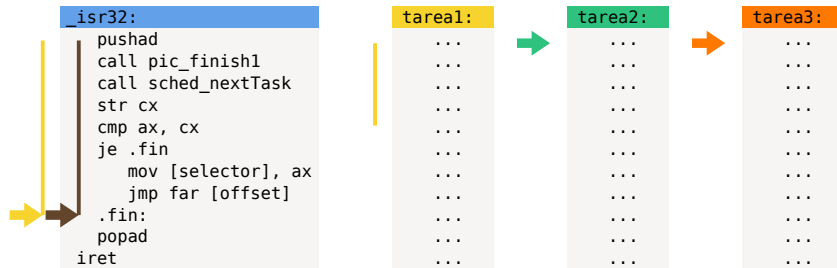
# Scheduler: Ejecución de la rutina del reloj



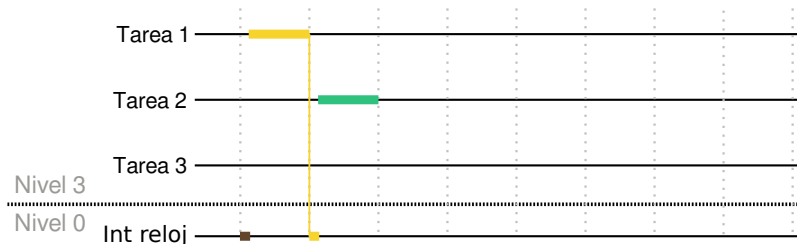
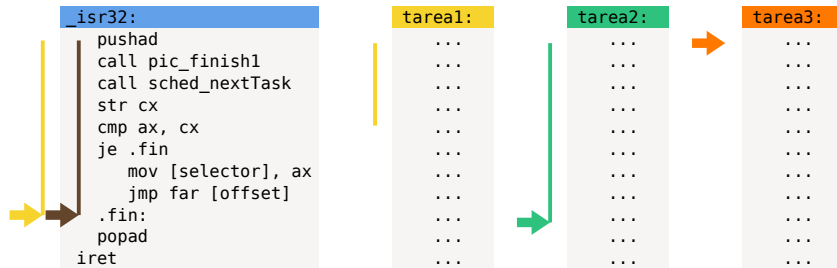
# Scheduler: Ejecución de la rutina del reloj



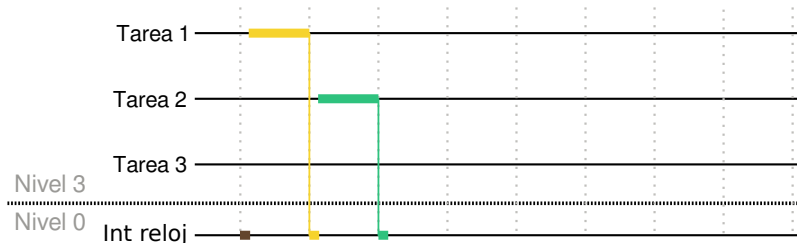
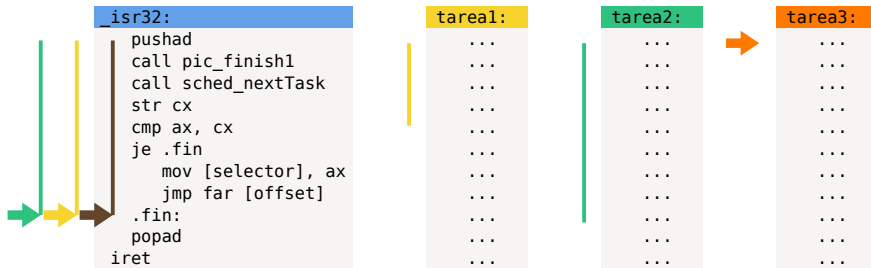
# Scheduler: Ejecución de la rutina del reloj



# Scheduler: Ejecución de la rutina del reloj

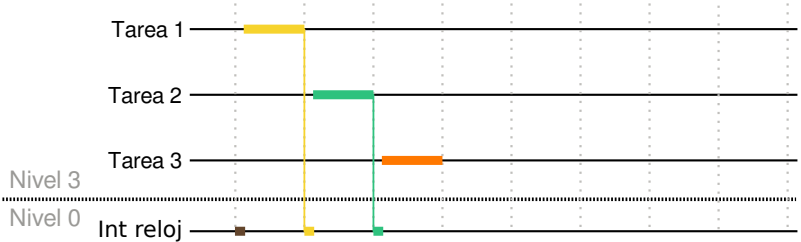
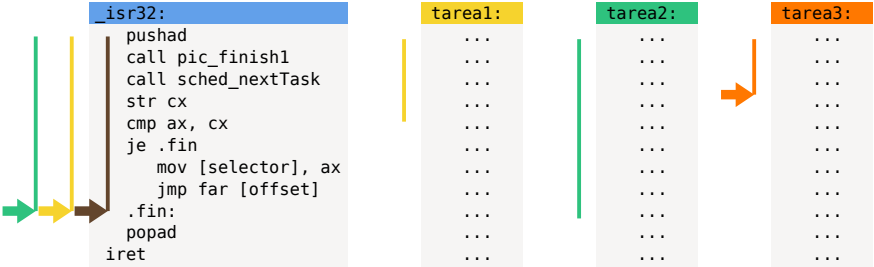


## Scheduler: Ejecución de la rutina del reloj

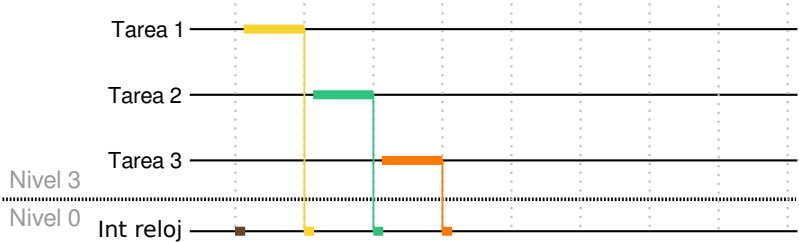
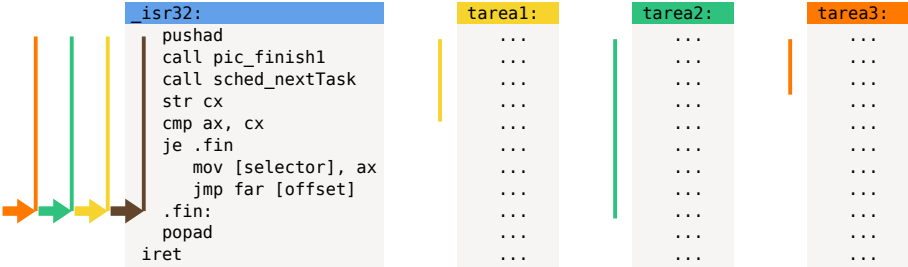




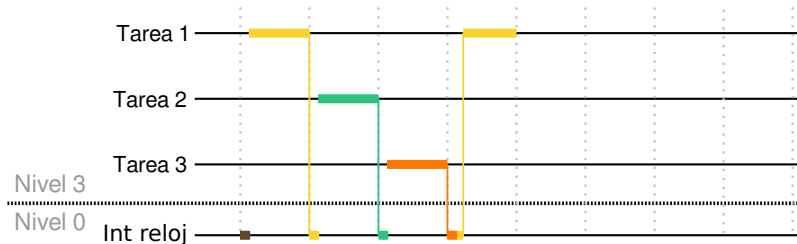
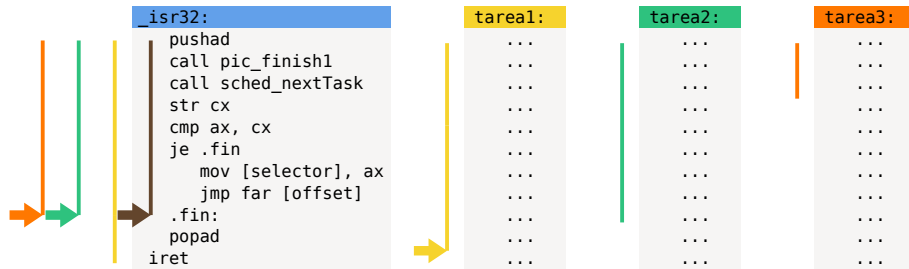
# Scheduler: Ejecución de la rutina del reloj



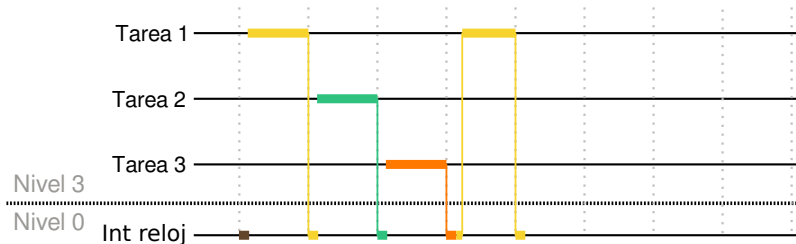
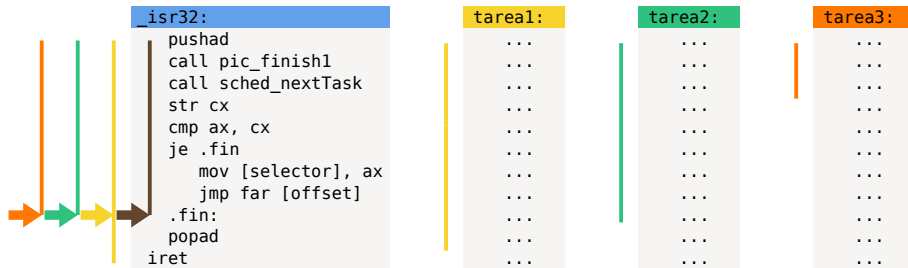
# Scheduler: Ejecución de la rutina del reloj



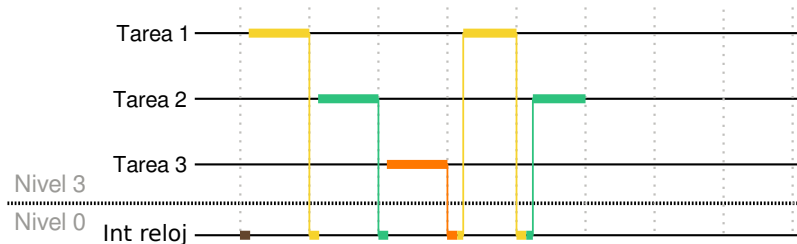
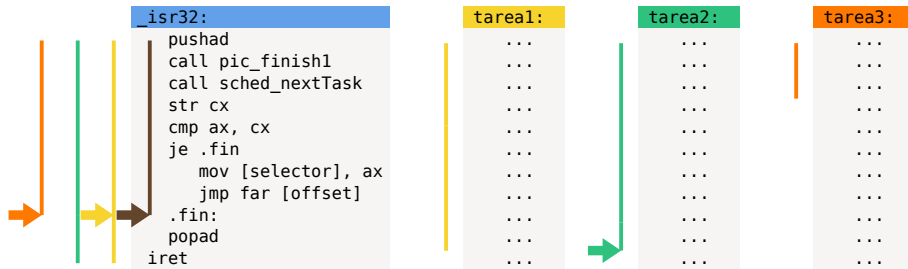
# Scheduler: Ejecución de la rutina del reloj



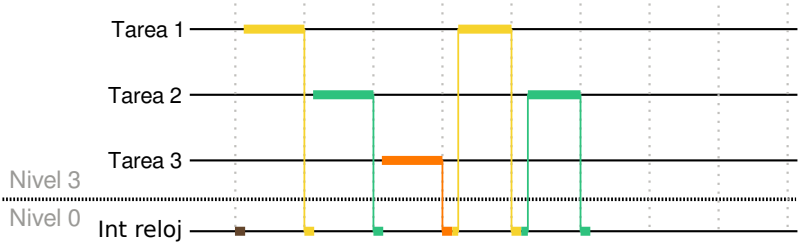
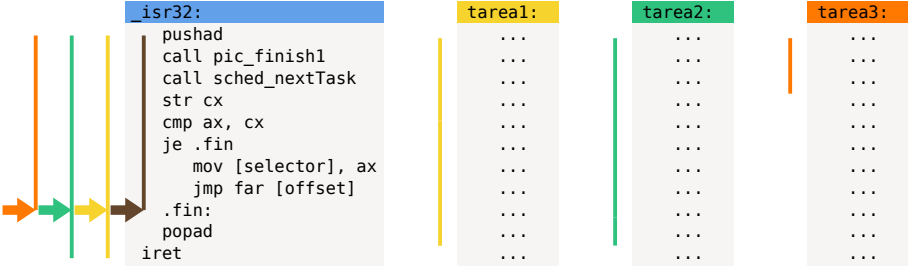
## Scheduler: Ejecución de la rutina del reloj



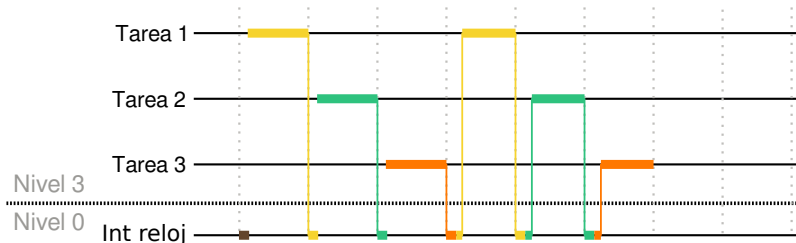
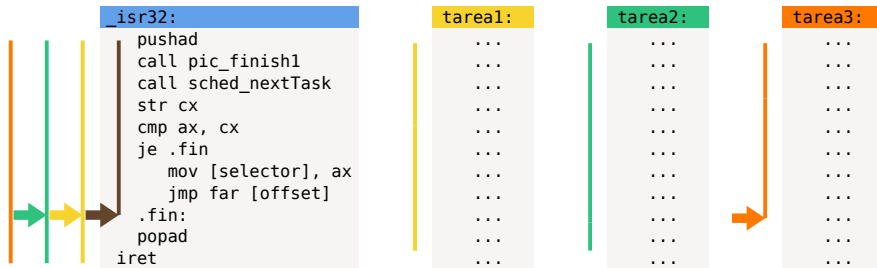
# Scheduler: Ejecución de la rutina del reloj



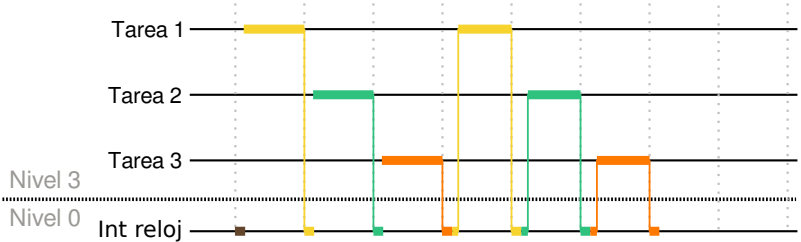
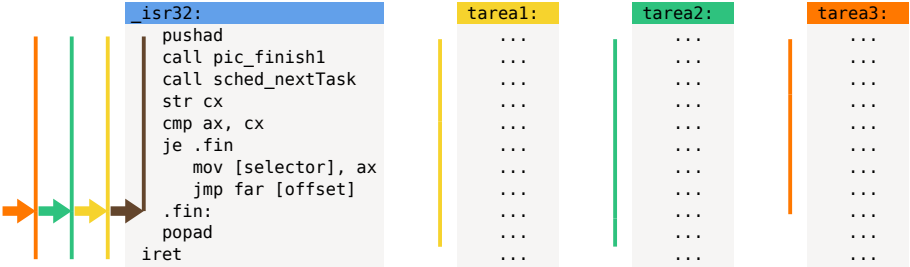
# Scheduler: Ejecución de la rutina del reloj



## Scheduler: Ejecución de la rutina del reloj



# Scheduler: Ejecución de la rutina del reloj





## Scheduler: Observaciones

## Scheduler: Observaciones

- El contexto de ejecución continua siendo el mismo hasta que es **desalojado**.

## Scheduler: Observaciones

- El contexto de ejecución continua siendo el mismo hasta que es **desalojado**.
- Las tareas desalojadas siempre quedan en la instrucción **siguiente** al jmp.

## Scheduler: Observaciones

- El contexto de ejecución continua siendo el mismo hasta que es **desalojado**.
- Las tareas desalojadas siempre quedan en la instrucción **siguiente** al jmp.
- El procesador siempre esta en **algún** contexto de ejecución.

## Scheduler: Observaciones

- El contexto de ejecución continua siendo el mismo hasta que es **desalojado**.
- Las tareas desalojadas siempre quedan en la instrucción **siguiente** al jmp.
- El procesador siempre esta en **algún** contexto de ejecución.
- ¿Y qué pasa en el cambio de privilegio?

## Scheduler: Observaciones

- El contexto de ejecución continua siendo el mismo hasta que es **desalojado**.
- Las tareas desalojadas siempre quedan en la instrucción **siguiente** al jmp.
- El procesador siempre esta en **algún** contexto de ejecución.
- ¿Y qué pasa en el cambio de privilegio?

### Cambio de pilas:

Desde nivel cero NO podemos usar la pila de nivel 3 para guardar el estado de retorno y variables locales. Por lo tanto se debe cambiar la base de pila.

## Scheduler: Observaciones

- El contexto de ejecución continua siendo el mismo hasta que es **desalojado**.
- Las tareas desalojadas siempre quedan en la instrucción **siguiente** al jmp.
- El procesador siempre esta en **algún** contexto de ejecución.
- ¿Y qué pasa en el cambio de privilegio?

### Cambio de pilas:

Desde nivel cero NO podemos usar la pila de nivel 3 para guardar el estado de retorno y variables locales. Por lo tanto se debe cambiar la base de pila. La nueva base de la pila se toma desde los campos SS0:ESP0 en la TSS.

## Scheduler: Observaciones

- El contexto de ejecución continua siendo el mismo hasta que es **desalojado**.
- Las tareas desalojadas siempre quedan en la instrucción **siguiente** al jmp.
- El procesador siempre esta en **algún** contexto de ejecución.
- ¿Y qué pasa en el cambio de privilegio?

### Cambio de pilas:

Desde nivel cero NO podemos usar la pila de nivel 3 para guardar el estado de retorno y variables locales. Por lo tanto se debe cambiar la base de pila. La nueva base de la pila se toma desde los campos SS0:ESP0 en la TSS. El estado de la pila de nivel 3 se guarda en la pila de nivel 0.



# Scheduler: Cambio de nivel de privilegio

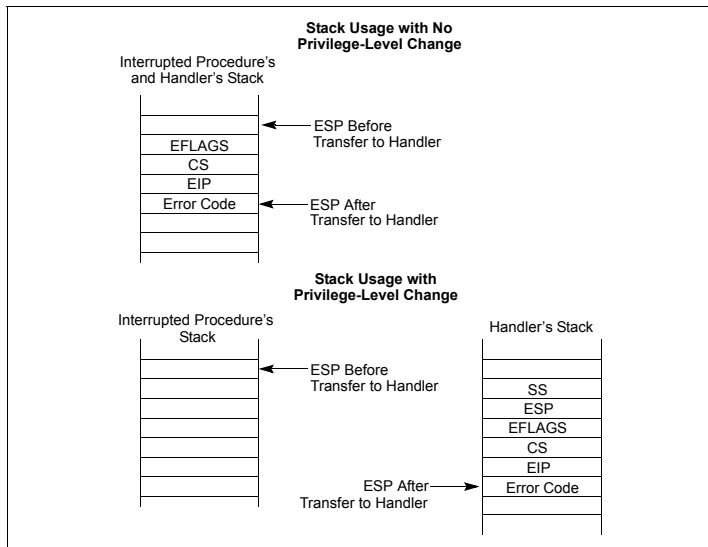


Figure 5-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

## Scheduler: Cambio de nivel de privilegio

```
...  
...  
int 0x80  
...  
...
```

Código de  
la tarea

```
_isr80:  
...  
...  
...  
iret
```

Código de  
la int 0x80

# Scheduler: Cambio de nivel de privilegio

```
...  
...  
int 0x80  
...  
...
```

Código de  
la tarea

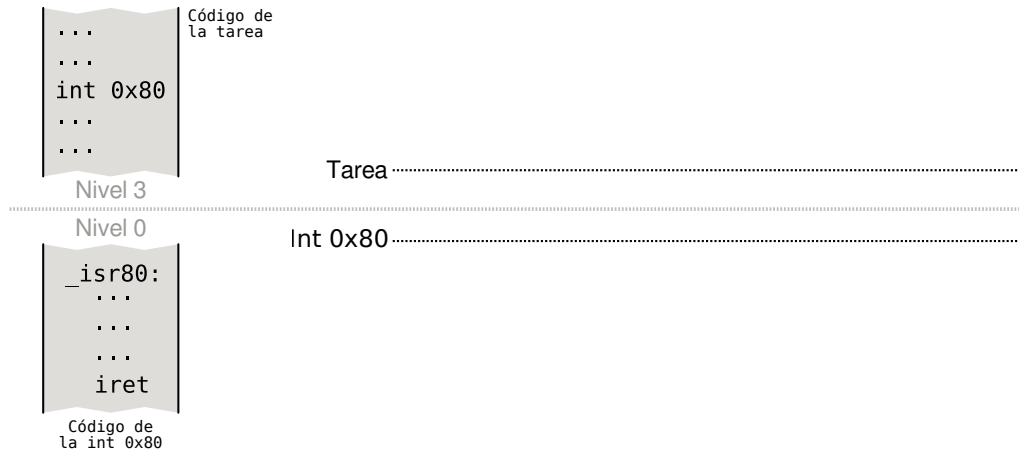
Nivel 3

Nivel 0

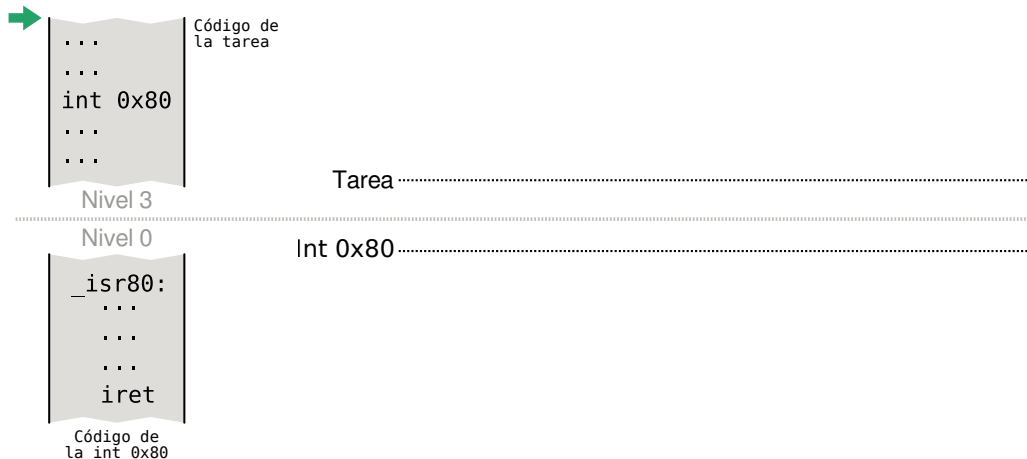
```
_isr80:  
...  
...  
...  
iret
```

Código de  
la int 0x80

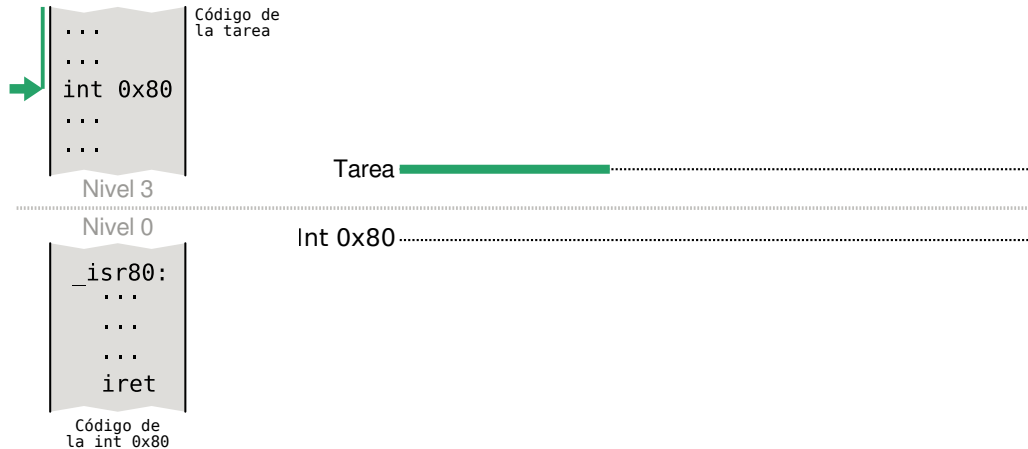
# Scheduler: Cambio de nivel de privilegio



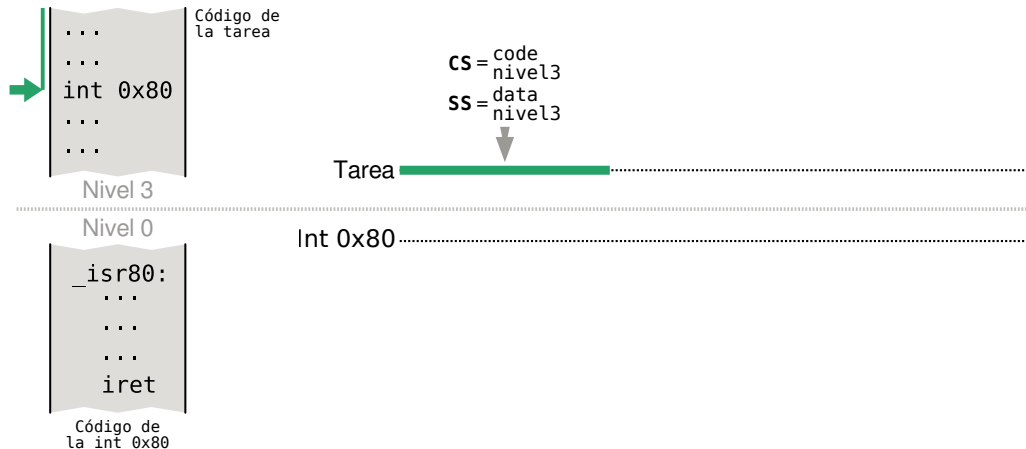
## Scheduler: Cambio de nivel de privilegio



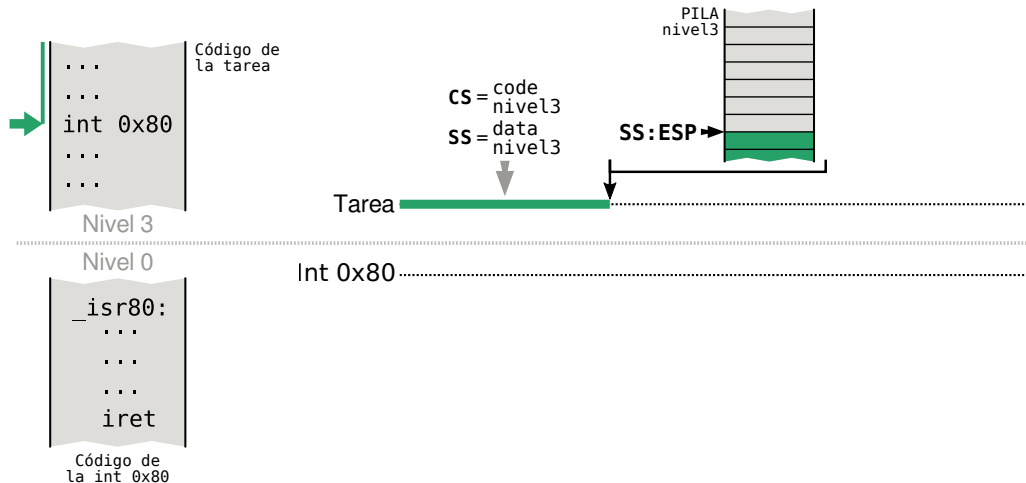
## Scheduler: Cambio de nivel de privilegio



# Scheduler: Cambio de nivel de privilegio

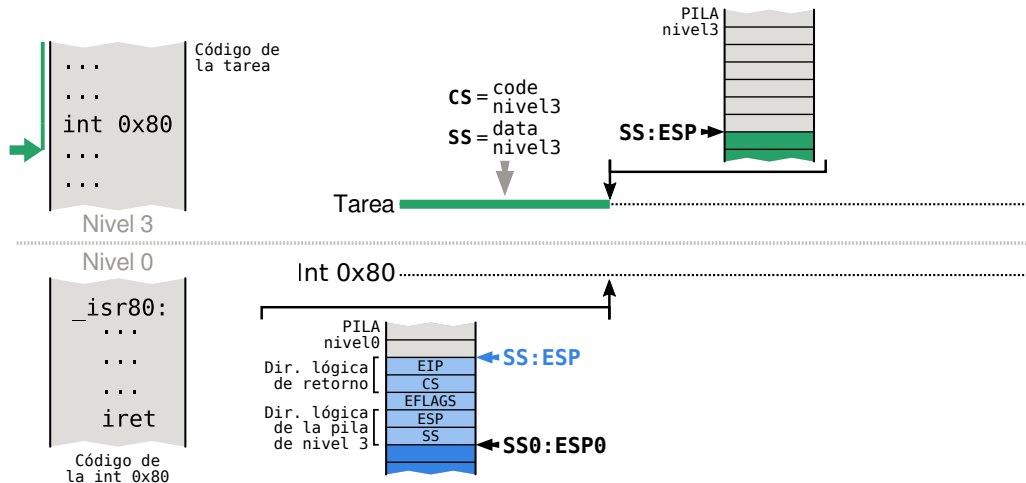


# Scheduler: Cambio de nivel de privilegio

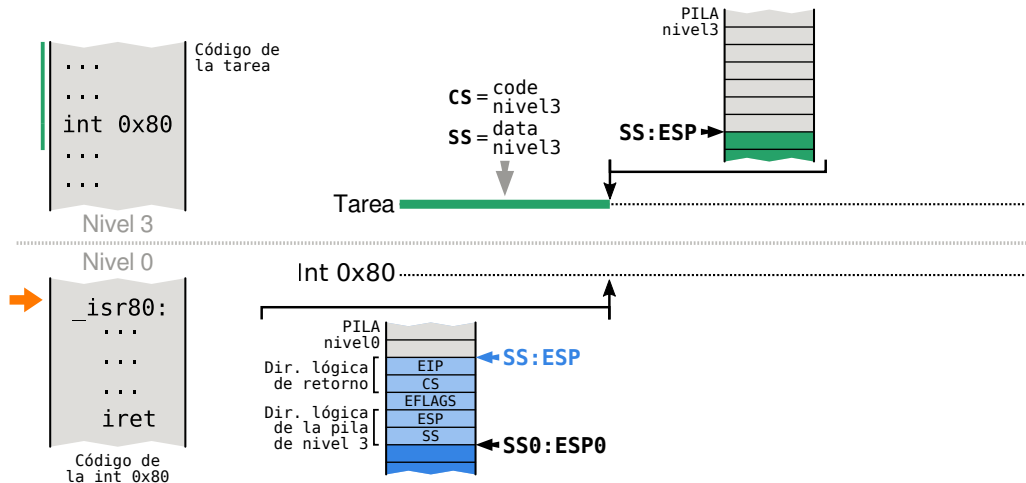




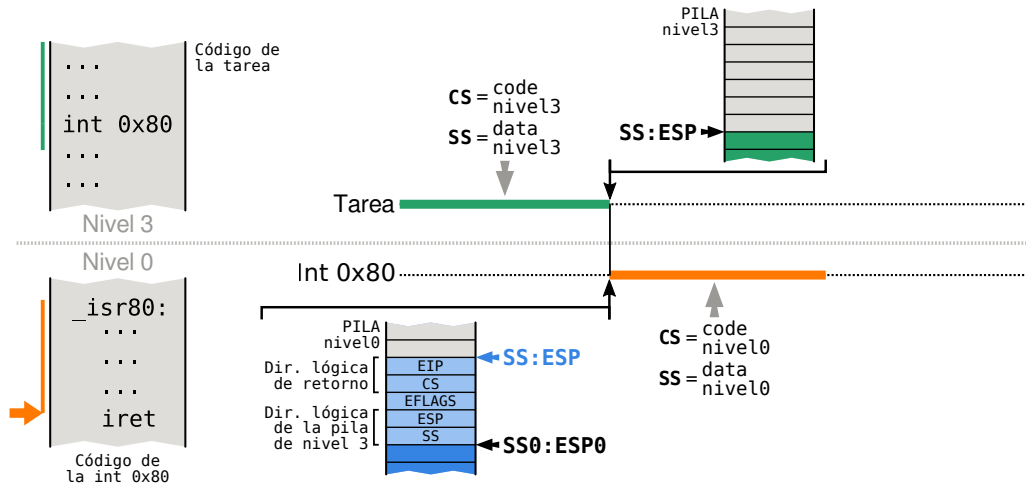
# Scheduler: Cambio de nivel de privilegio



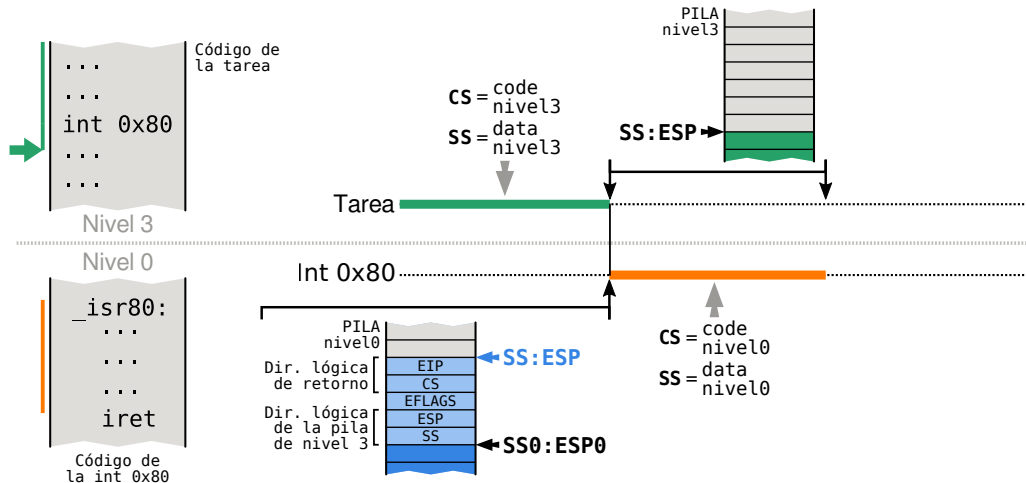
# Scheduler: Cambio de nivel de privilegio



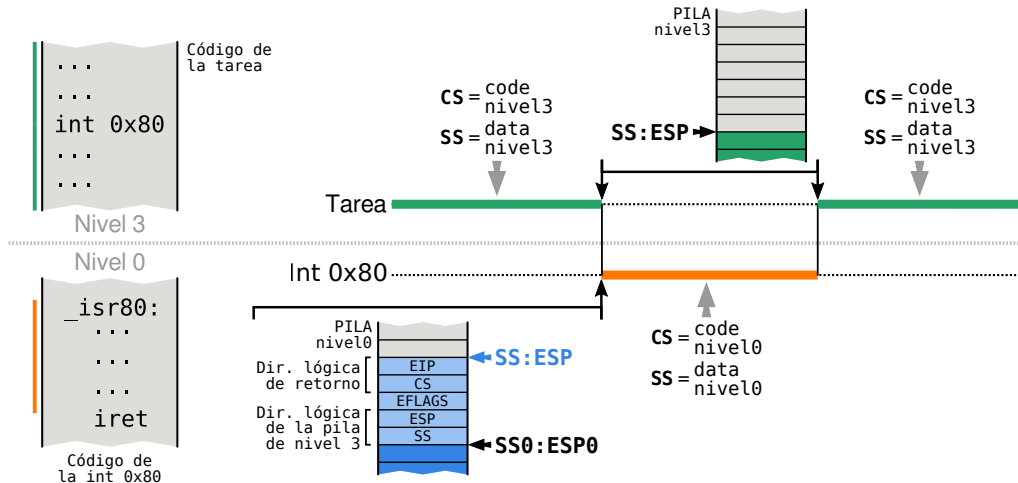
# Scheduler: Cambio de nivel de privilegio



# Scheduler: Cambio de nivel de privilegio



# Scheduler: Cambio de nivel de privilegio



# Scheduler: Observaciones

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.

Figure 7-2. 32-Bit Task-State Segment (TSS)

## Scheduler: Observaciones

- Siempre que hay un cambio de privilegio hay un cambio de pila.

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.

Figure 7-2. 32-Bit Task-State Segment (TSS)

## Scheduler: Observaciones

- Siempre que hay un cambio de privilegio hay un cambio de pila.
- Implica cambiar el SS y ESP por el correspondiente en su nivel de privilegio.

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.

Figure 7-2. 32-Bit Task-State Segment (TSS)



## Scheduler: Observaciones

- Siempre que hay un cambio de privilegio hay un cambio de pila.
- Implica cambiar el SS y ESP por el correspondiente en su nivel de privilegio.
- El resto de los segmentos de datos DS,  $\dots$ , ES no se modifican.

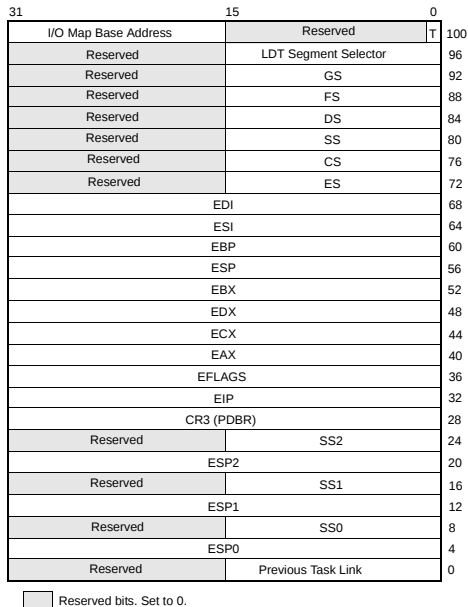


Figure 7-2. 32-Bit Task-State Segment (TSS)

## Bibliografía: Fuentes y material adicional

- Convenciones de llamados a función en x86:  
[https://en.wikipedia.org/wiki/X86\\_calling\\_conventions](https://en.wikipedia.org/wiki/X86_calling_conventions)
- Notas sobre System V ABI:  
[https://wiki.osdev.org/System\\_V\\_ABI](https://wiki.osdev.org/System_V_ABI)
- Documentación de NASM:  
<https://nasm.us/doc/>
- Artículo sobre el flag -pie:  
<https://eklitzke.org/position-independent-executables>
- Documentación de System V ABI:  
[https://uclibc.org/docs/psABI-x86\\_64.pdf](https://uclibc.org/docs/psABI-x86_64.pdf)
- Manuales de Intel:  
<https://software.intel.com/en-us/articles/intel-sdm>

# ¡Gracias!

Recuerden leer los comentarios al final de este video por aclaraciones o fe de erratas.