

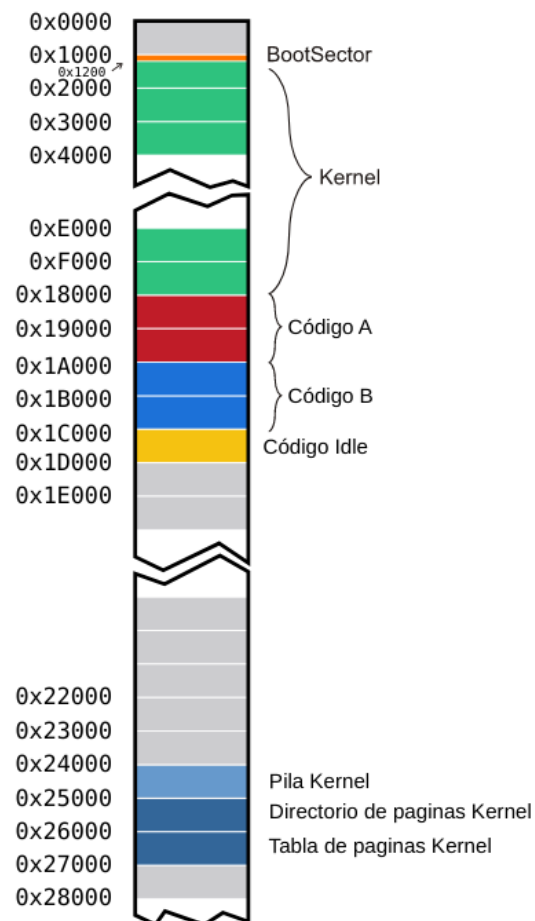
Organización del Computador II
System Programming
Taller 4: Tareas

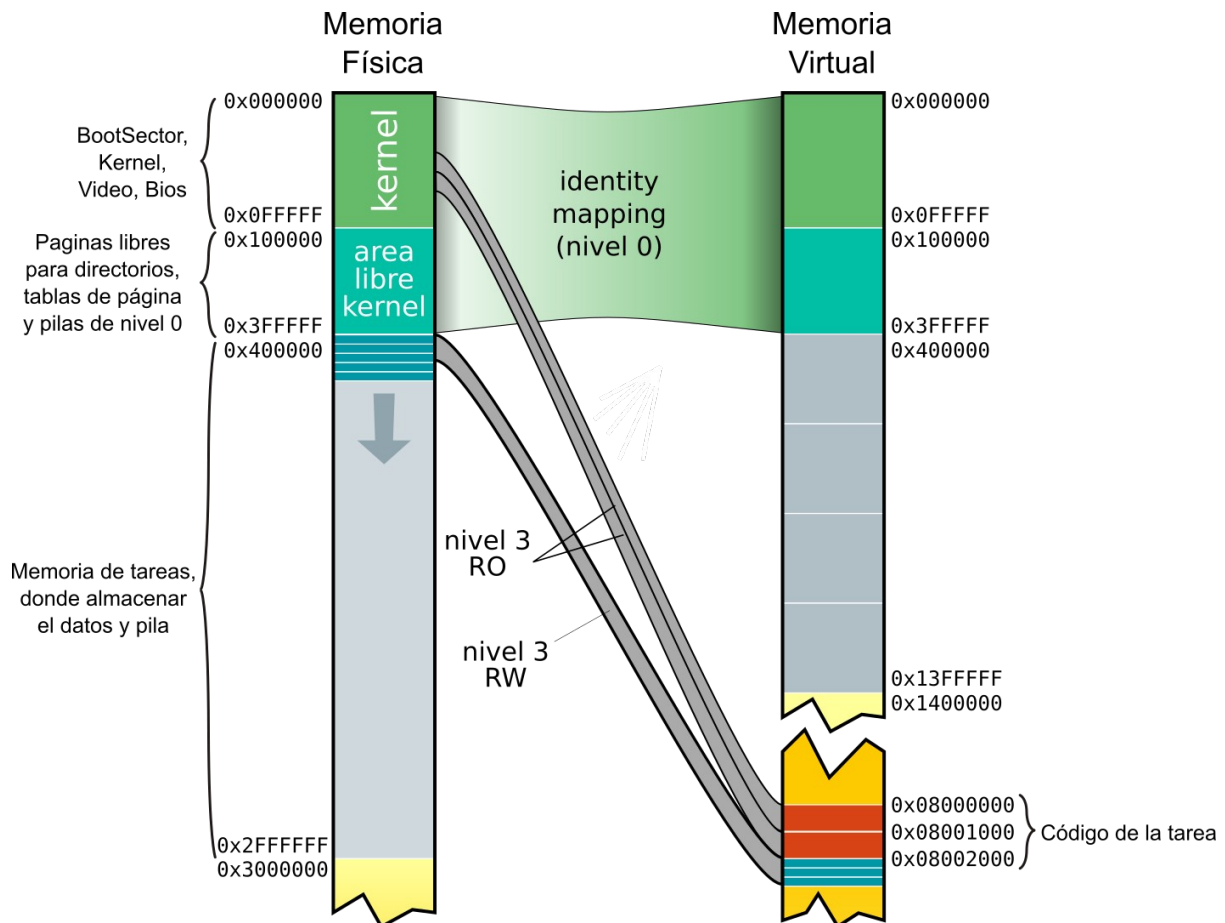
El taller de hoy: Tareas

Vamos a continuar trabajando con el kernel que estuvimos programando en los talleres anteriores. La idea es incorporar la posibilidad de ejecutar algunas tareas específicas. Para esto vamos a precisar:

- Definir las estructuras de las tareas disponibles para ser ejecutadas
- Tener un scheduler que determine la tarea que le toca ejecutarse en un período de tiempo y el mecanismo para el intercambio de tareas de la CPU
- Iniciar el kernel con una *tarea inicial* y tener una *tarea idle* para cuando no haya tareas en ejecución

Recordamos el mapeo de memoria con el que venimos trabajando. Las tareas que vamos a crear en este taller van a ser parte de esta organización de la memoria:





Archivos provistos

A continuación les pasamos la lista de archivos que forman parte del taller de hoy junto con su descripción:

- **Makefile** - encargado de compilar y generar la imagen del floppy disk.
- **idle.asm** - código de la tarea Idle.
- **syscall.h** - interfaz a utilizar desde las tareas para los llamados al sistema.
- **tss.h, tss.c** - definición de estructuras para el manejo de tareas
- **sched.h, sched.c** - scheduler de tareas del kernel
- **task.h, task.c, taskA.c, taskB.c** - código de las tareas

Ejercicios

1. Si queremos definir un sistema que utilice sólo dos tareas, ¿qué estructuras, cantidad de nuevas entradas por estructura y registros tenemos que configurar? Indicar el detalle de los bits de cada una, ¿qué formato tienen? y ¿dónde se encuentran almacenadas dichas estructuras?
2. ¿A qué llamamos cambio de contexto, cuándo se produce y qué efecto tiene sobre los registros del procesador? Expliquen en sus palabras que almacena el registro **TR** y cómo obtiene la información necesaria para ejecutar la tarea después de un cambio de contexto.
3. Al momento de realizar un cambio de contexto el procesador va almacenar el estado actual de

acuerdo al selector indicado en el registro **TR** y ha de restaurar aquel almacenado en la tss cuyo selector se asigna en el jmp far. ¿Qué consideraciones deberíamos tener para poder realizar el primer cambio de contexto? ¿Y cuáles cuando no tenemos tareas que ejecutar o se encuentran todas suspendidas?

4. ¿Qué hace el scheduler de un Sistema Operativo? ¿A qué nos referimos con que usa una política?

5. En un sistema de una única CPU, ¿cómo se hace para que lxs usuarixs vean todos los programas corriendo en simultáneo?

Primer Checkpoint

6. En **tss.c** se encuentran definidas las TSS de la Tarea Inicial e Idle. Ahora, vamos a agregar el TSS Descriptor correspondiente a estas tareas en la **GDT**.

- a) Observen qué hace el método: **tss_gdt_entry_for_task**
- b) Escriban el código del método **tss_init** de **tss.c** que agrega dos nuevas entradas a la **GDT** correspondientes al descriptor de TSS de la tarea Inicial e Idle.
- c) En **kernel.asm**, luego de habilitar paginación, agreguen una llamada a **tss_init** para que efectivamente estas entradas se agreguen a la **GDT**.
- d) Correr el bochs y usar **info gdt** para verificar que los descriptores de tss de la tarea Inicial e Idle esten efectivamente cargadas en la GDT

7. Como vimos, la primer tarea que va a ejecutar el procesador cuando arranque va a ser la **tarea Inicial**. Se encuentra definida en tss.c y que tiene todos sus campos en 0. Antes de que comience a ciclar infinitamente, completen lo necesario en **kernel.asm** para que cargue como la tarea inicial. Recuerden que tenemos que cargar el registro **TR** (Task Register) con la instrucción **LTR**, la primera vez.

Si obtienen un error, asegurense de haber proporcionado un selector de segmento para la tarea inicial. Un selector de segmento no es sólo el índice en la GDT sino que tiene algunos bits con privilegios y el table indicator.

8. Una vez que el procesador comenzó su ejecución en la **tarea Inicial**, le vamos a pedir que salte a la **tarea Idle** con un **JMP**. Para eso, completar en **kernel.asm** el código necesario para saltar intercambiando **TSS**, entre la tarea inicial y la tarea Idle.

9. Utilizando **info tss**, verifiquen el valor del **TR**. También, verifiquen los valores de los registros **CR3** con **creg** y registros de segmento **CS**, **DS**, **SS** con **sreg**. ¿Por qué hace falta tener definida la pila de nivel 0 en la tss?

10. En **tss.c**, completar la función **tss_create_user_task** para que inicialice una TSS con los datos correspondientes a una tarea cualquiera. La función recibe por parámetro la dirección del código de una tarea y será utilizada más adelante para crear tareas. El código de las tareas se encuentra a partir de la dirección 0x00018000 ocupando dos páginas de 4 KB cada una según indica la figura 1. Para la dirección de la pila se debe reservar una página física sin usar, la misma crecerá desde la base

de la pila. Tener en cuenta:

- a) Para el mapa de memoria se debe construir uno nuevo utilizando la función **mmu_init_task_dir** implementada en el taller anterior. Este método va a crear su directorio de página y tablas de nivel 3. Dejando en la dirección virtual **TASK_CODE_VIRTUAL** el código y en **TASK_STACK_BASE**, la pila.
- b) Además, tener en cuenta que cada tarea utilizará una pila distinta de nivel 0, para esto se debe pedir una nueva página del área libre de kernel a tal fin. Pueden usar el método: **mmu_next_free_kernel_page**.

Segundo Checkpoint

11. Estando definidas **sched_task_offset** y **sched_task_selector**:

```
sched_task_offset:    dd 0xFFFFFFFF
sched_task_selector:  dw 0xFFFF
```

Y siendo la siguiente es una implementación de una interrupción del reloj:

```
global _isr32

_isr32:
    pushad
    call pic_finish1

    call sched_next_task

    str cx
    cmp ax, cx
    je .fin

    mov word [sched_task_selector], ax
    jmp far [sched_task_offset]

.fin:
    popad
    iret
```

- a) Expliquen con sus palabras que se estaría ejecutando en cada tic del reloj línea por línea

12. Para este Taller la cátedra ha creado un scheduler que devuelve la próxima tarea a ejecutar. L

- a) Scheduler: Abran el archivo de texto **scheduler.readme** y agreguen el código provisto en los archivos indicados.
- b) Modifiquen **kernel.asm** para llamar a la función **sched_init** luego de iniciar la tss
- c) Aseguren de compilar, ejecutar **bochs** y ver que todo sigue funcionando correctamente.

d) En la línea de la RAI del reloj que dice:

```
jmp far [sched_task_offset]
```

¿De qué tamaño es el dato que estaría leyendo desde la memoria? ¿Qué indica cada uno de estos valores? ¿Tiene algún efecto el offset elegido?

e) ¿A dónde regresa la ejecución (eip) de una tarea cuando vuelve a ser puesta en ejecución?

13. En los archivos **sched.c** y **sched.h** se encuentran definidos los métodos necesarios para el Scheduler. Expliquen cómo funciona el scheduler, es decir, cómo decide cuál es la próxima tarea a ejecutar. Pueden encontrarlo en la función **sched_next_task**.

Tercer Checkpoint

14. Como parte de la inicialización del kernel, en **kernel.asm** se pide agregar una llamada a la función **task_init** de **task.c** que a su vez llama a **create_task**. Observe las siguientes líneas:

```
size_t gdt_id = (tipo == TASK_A? GDT_IDX_TASK_A_START :  
                GDT_IDX_TASK_B_START) + indice;  
gdt[gdt_id] = tss_gdt_entry_for_task(&tss_tasks[task_id]);  
sched_add_task(gdt_id << 3, task_id);
```

¿Qué está haciendo la función **tss_gdt_entry_for_task**? ¿Por qué motivo se realiza el desplazamiento a izquierda de **gdt_id** al pasarlo como parámetro de **sched_add_task**?

15. Ejecuten las tareas en bochs. Observen que hacen las tareas **taskA.c** y **taskB.c**. Pueden probar agregándoles código para que hagan distintas cosas.

16. ¿Por qué debemos dejar a las tareas ejecutando la instrucción **nop** indefinidamente luego de terminar de procesar?

Cuarto Checkpoint

17. [Opcional] Resuman con su equipo todas las estructuras vistas desde el Taller 1 al Taller 4. Escriban el funcionamiento general de segmentos, interrupciones, paginación y tareas en los procesadores Intel. ¿Cómo interactúan las estructuras? ¿Qué configuraciones son fundamentales a realizar? ¿Cómo son los niveles de privilegio y acceso a las estructuras?