Taller de paginación

Organización del Computador II

Segundo Cuatrimestre 2021

En este taller vamos a inicializar y a habilitar los mecanismos de manejo de memoria de nuestro kernel.

Al corregir cada **checkpoint** todos los miembros del grupo deben estar presentes, salvo aquellas instancias en que puedan justificar su ausencia previamente. Si al finalizar la práctica de la materia algún miembro cuenta con mas de un 20 % de ausencia no justificada en la totalidad de los checkpoints se considerará que cursada como desaprobada.

Cada checkpoint se presenta a lx docente asignadx en cada breakout room durante el transcurso de la clase práctica actual o la siguiente, no hace falta entregar los archivos.

1. Organización de la memoria

Primero vamos a explicar cómo se encuentra el mapa de memoria física para comprender de qué modo inicializar las tablas de memoria. El primer MB de memoria física será organizado según indica la figura 1. En la misma se observa que a partir de la dirección `0x1200` se encuentra ubicado el *kernel*; inmediatamente después se ubica el código de las tareas A y B, y a continuación el código de la tarea Idle. El resto del mapa muestra el rango para la pila del kernel, desde `0x24000` a `0x25000` y a continuación la tabla y directorio de páginas donde inicializar paginación para el kernel. La parte derecha de la figura muestra la memoria a partir de la dirección `0xA0000`, donde se encuentra mapeada la memoria de vídeo y el código del `BIOS`.

Fuera del primer MB la memoria física se divide en: *kernel*, *área libre kernel* y *área libre tareas*

El área asociada al *kernel* corresponde al primer MB de memoria, el *área libre kernel* a los siguientes 3, y el *área libre tareas* a los siguientes 3.

2. Manejador de memoria

La administración del área libre de memoria se realizará a partir de una región de memoria que podemos comprender como un arreglo pre definido de páginas y dos contadores de páginas, uno para kernel y otro par usuarix, que indican cuál será la próxima página a emplear de una región para páginas de kernel de `0x100000` a `0x3FFFFF` y una región para páginas de usuarix de `0x400000` a `0x2FFFFFF`. Luego de cada pedido incrementamos contador correspondiente. Para el contexto de la materia no implelemntamos un mecanismo que permita liberar las páginas pedidas. Vamos a referirnos a este mecanismo como el **manejador de memoria**.

Las páginas del *área libre kernel* serán utilizadas para datos del kernel: directorios de páginas, tablas de páginas y pilas de nivel cero. Las páginas del *área libre tareas* serán utilizadas para datos de las tareas: stack y memoria compartida bajo demanda.

La memoria virtual de cada una de las tareas tiene mapeado el *kernel*, *área libre kernel* y *área libre tareas* con *identity mapping* en nivel 0.

El código de las tareas se encontrará a partir de la dirección virtual *0x08000000* y será mapeado como sólo lectura de nivel 3 a la dirección física correspondiente al código correspondiente. El stack será mapeado en la página siguiente, con permisos de lectura y escritura. La página física debe obtenerse del *área libre tareas*.

3. Ejercicios

a) ¿Cuántos niveles de privielgio podemos definir en las estructuras de paginación?

b) ¿Cómo se traduce una dirección lógica en una dirección física? ¿Cómo participan el selector de segmento, el registro de control `CR3`, el directorio y la tabla de páginas?

c) ¿Cuál es el efecto de los siguientes atributos en las entradas de la tabla de página?

- `D`
- `A`
- `PCD`
- `PWT`
- `U/S`

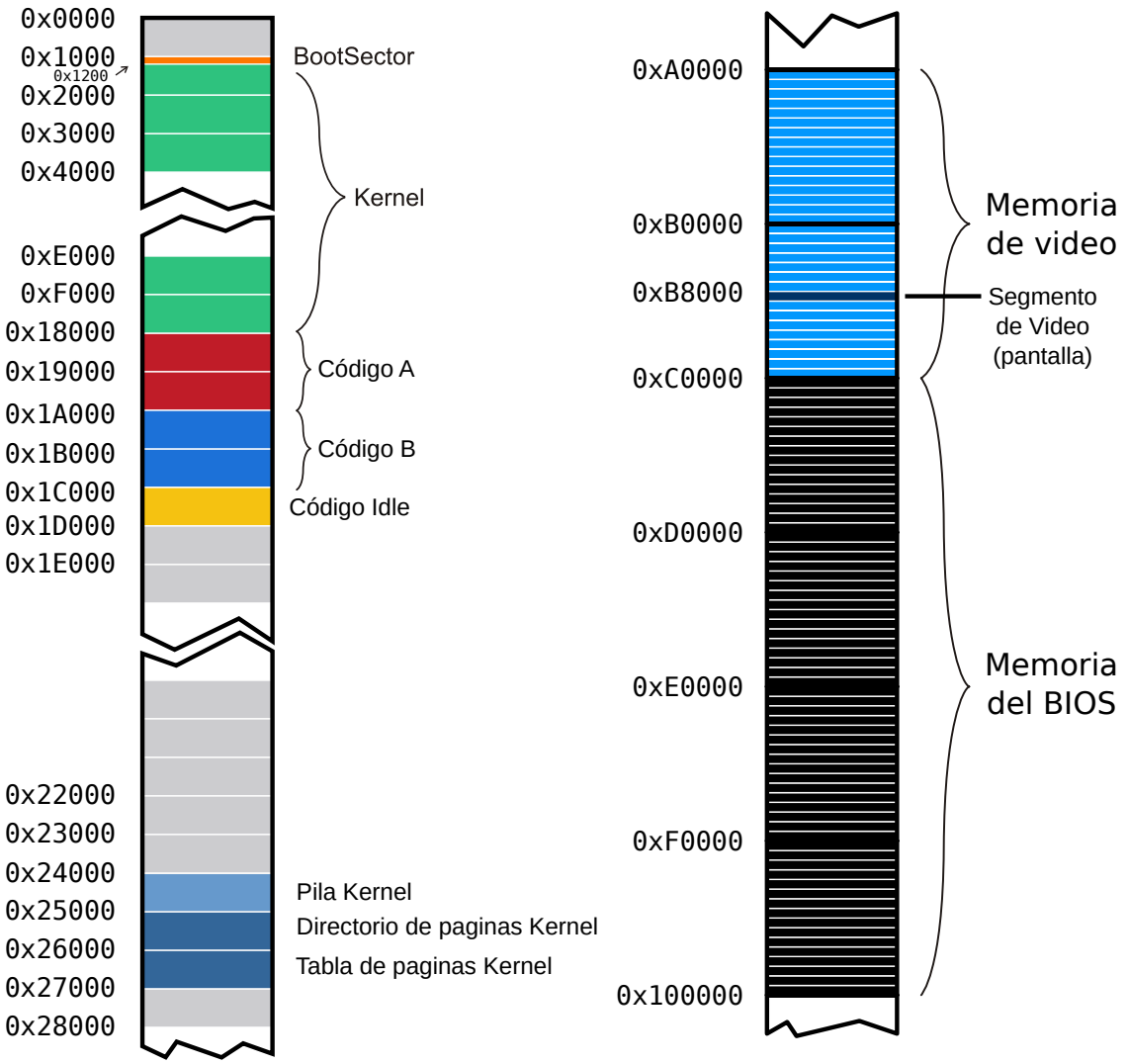

Figura 1: Mapa de la organización de la memoria física del *kernel*

- `R/W`
- `P`

d) Suponiendo que el código de la tarea ocupa dos páginas y utlizaremos una página para la pila de la tarea. ¿Cuántas páginas hace falta pedir al manejador de memoria para el directorio, tablas de páginas y la memoria de una tarea?

e) Observar el contenido de `defines.h` y `mmu.h` y explicar la función y motivación de los defines para:

- `VIRT_PAGE_OFFSET(X)` donde `X` es una dirección virtual.
- `VIRT_PAGE_TABLE(X)` donde `X` es una dirección virtual.
- `VIRT_PAGE_DIR(X)` donde `X` es una dirección virtual.
- `CR3_TO_PAGE_DIR(X)` donde `X` es el contenido del registro `CR3`.
- `MMU_ENTRY_PADDR(X)` donde `X` es una entrada de la tabla de páginas.

f) ¿Qué es el buffer auxiliar de traducción (translation lookaside buffer o **TLB**) y por qué es necesario purgarlo (`tlbflush`) al introducir modificaciones a nuestras estructuras de paginación (directorio, tabla de páginas o CR3)?

Checkpoint 1

a) Escriban el código indicado en el archivo `mmu.c` para completar la inicializaicón del directorio y tablas de páginas para el *kernel* en la función `mmu_init_kernel_dir`. Recuerden que las entradas del directorio y la tabla debe un mapeo por identidad (las direcciones lineales son iguales a las direcciones físicas) para el rango reservado para el kernel, de `0x00000000` a `0x003FFFFF`, como ilustra la figura 2. Esta función debe inicializar también el directorio de páginas en la dirección `0x25000` y las tablas de páginas según muestra la figura 1. ¿Cuántas entradas del directorio de página hacen falta?

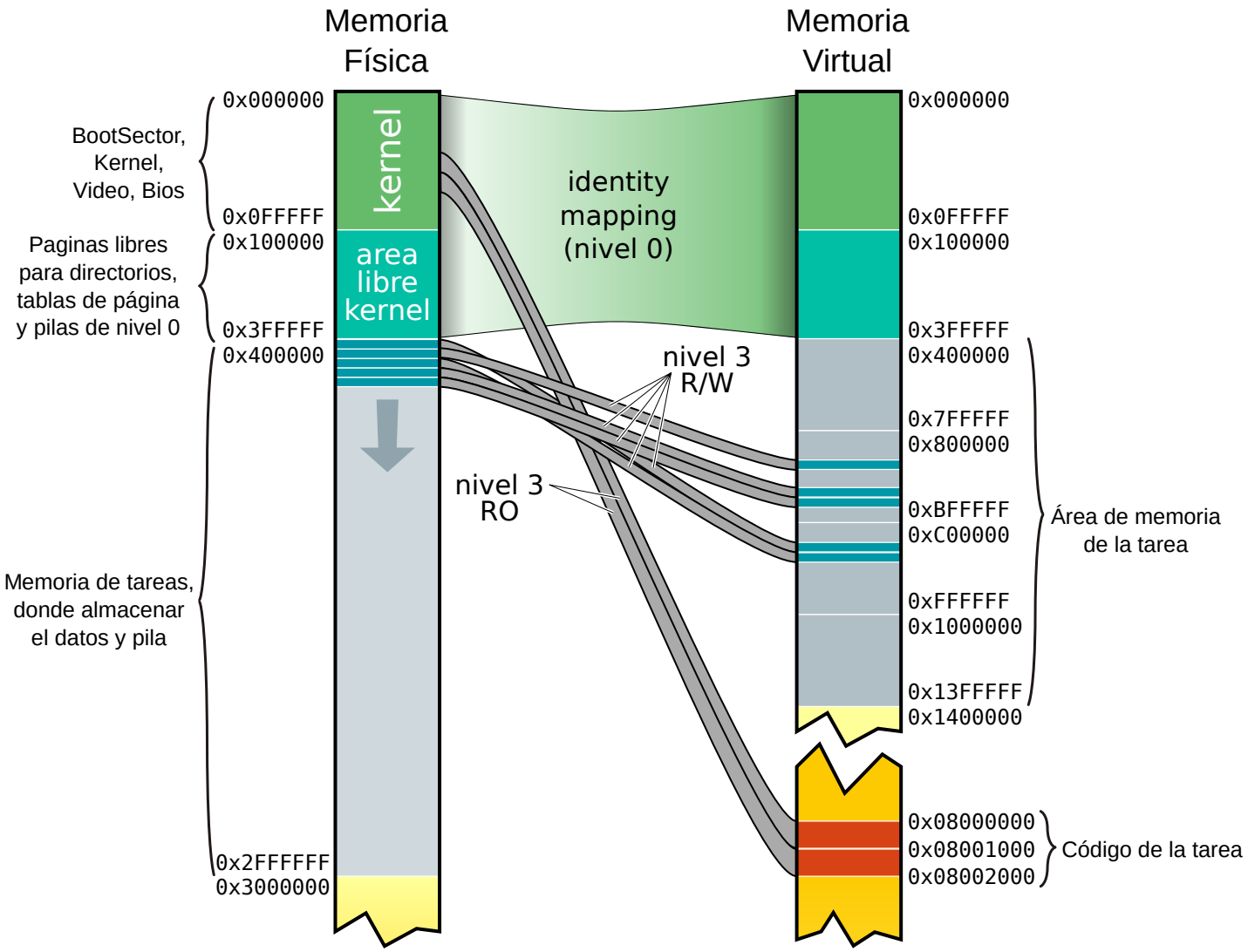

Figura 2: Mapa de memoria de la tarea

b) Completar el código necesario para activar paginación, recuerden que es necesario inicializar el registro `CR3` y activar el bit de `paging enable`(posición 31).

c) Introduzcan un breakpoint luego de activar paginación e impriman .

Checkpoint 2

a) Completen el código de la función `mmu_map_page`.

b) Observen el código de `copy_page`, ¿por qué es necesario mapear y desmapear las páginas de destino y fuente? ¿Qué función cumplen `SRC_VIRT_PAGE` y `DST_VIRT_PAGE`?

c) Completar la rutina (`mmu_init_task_dir`) encargada de inicializar un directorio de páginas y tablas de páginas para una tarea, respetando la figura 2. La rutina debe mapear las páginas de código como solo lectura, a partir de la dirección virtual `0x08000000`, y el stack como lectura-escritura con base en `0x08003000`. La memoria para la pila de la tarea debe salir del area de memoria de las tareas.

d) A modo de prueba, eliminar la instrucción que realiza el salto a `cargar_interrupciones` en `kernel.asm` para construir un mapa de memoria para tareas e intercambiarlo con el del *kernel*, luego cambiar el color del fondo del primer caracter de la pantalla y volver a la normalidad. Inspeccionar el mapa de memoria con el comando `info tab` en los breakpoints que se encuentran una vez que se asigna el `CR3` de la tarea y cuando se restituye el `CR3` del kernel.

Nota: Por construcción del *kernel*, las direcciones de los mapas de memoria (`page directory` y `page table`) están mapeadas con *identity mapping*.

En las funciones en donde se modifica el directorio o tabla de páginas, se debe llamar a la función `tlbflush` para que se invalide la *cache* de traducción de direcciones.

Con el comando `page vaddr` en bochs pueden ver información sobre cómo está mapeada la dirección virtual vaddr.

Checkpoint 3