Trabajo Práctico 3
System Programming

Organización del Computador 2

2do. Cuatrimestre 2009

1. Objetivo

El tercer trabajo práctico de la materia consiste en un conjunto de ejercicios en los que se aplican los conceptos de *System Programming* vistos en las clases teóricas y prácticas de forma gradual. Se busca construir un sistema minimal que permita correr dos tareas concurrentemente. Los ejercicios se basan en la comprensión y utilización de los mecanismos que posee el procesador para la programación desde el punto de vista del sistema operativo.

2. Introducción

Para este trabajo se utilizará como entorno de pruebas el programa Bochs. El mismo permite simular una computadora compatible IBM-PC desde el inicio y realizar tareas de debugging. Todo el código provisto para la realización del presente trabajo está ideado para correr en Bochs de forma sencilla.

Una computadora al iniciar comienza con la ejecución del POST y el BIOS, el cual se encarga de reconocer el primer dispositivo de booteo. En este caso dispondremos de un Floppy Disk como dispositivo de booteo. En el primer sector del dispositivo se almacena el boot-sector. El BIOS se encarga de copiar a memoria los 512 bytes del sector, a partir de la dirección `0x7c00`. Luego se comienza a ejecutar el código a partir esta dirección. El boot-sector debe encontrar en el floppy el archivo `kernel.bin` y copiarlo a memoria. El kernel se copia a partir de la direccion `0x1200` y luego se ejecuta a partir de esta dirección. Es importante tener en cuenta que el código del boot-sector se encarga exclusivamente de copiar el kernel y dar el control al mismo, es decir, no cambia el modo del procesador.

El esquema para utilizar como punto de partida para el trabajo práctico consta de los siguientes archivos:

- `Makefile` - encargado de compilar y generar el floppy disk
- `bochsrc` - configuración para inicicializar Bochs
- `diskette.img` - la imagen del floppy que contiene el boot-sector preparado para cargar el kernel
- `kernel.asm` - esquema básico del código esperado para el kernel

- `GDT.H` y `GDT.C` - código donde definir la tabla de descriptores globales
- `TSS.H` y `TSS.C` - código donde definir entradas de TSS
- `IDT.H` y `IDT.C` - código donde definir entradas para la IDT y funciones asociadas, `idtFill` para completar entradas en la IDT y `handler` para imprimir un mensaje.
- `ISR.H` y `ISR.asm` - Definiciones de las rutinas para atender interrupciones (Interrupt Service Routines) y la definición de la función `next_clock`
- `A20.asm` - rutinas para deshabilitar y habilitar A20
- `macrosmodoprotegido.mac` y `macrosmodoreal.mac` - macros utiles para imprimir por pantalla y transformar valores
- `pintor.tsk` y `traductor.tsk` - binarios de las tareas a correr

Además se deben implementar las siguientes funciones para utilizar la controladora de interrupciones.

```
pic_reset:
    outb 0x20, 0x11
    outb 0x21, 0x20
    outb 0x21, 0x04
    outb 0x21, 0x01
    outb 0x21, 0xFF
    outb 0xA0, 0x11
    outb 0xA1, 0x28
    outb 0xA1, 0x02
    outb 0xA1, 0x01
    outb 0xA1, 0xFF

pic_enable:
    outb 0x21, 0x00
    outb 0xA1, 0x00

pic_disable:
    outb 0x21, 0xFF
    outb 0xA1, 0xFF

pic1_intr_end:
    outb 0x20, 0x20
    outb 0x20, 0x20
```

La función `pic_reset` se encarga de reiniciar y remapear la controladora. La función `pic_enable` y `pic_disable` se encargan de habilitar y deshabilitar las interrupciones respectivamente. Por último, la función `pic1_intr_end` es la encargada de indicar al controlador que la interrupción fue atendida.

3. El pintor y el traductor

El traductor y el pintor son los nombres de las dos tareas que se deben correr. Un cuento de vieja data menciona que antes de estandarizar la forma de escribir en pantalla, cada uno

hacia lo que quería. El pintor es un programa que escribe de forma muy extraña en memoria de video. Como lo escrito por el pintor no se puede entender, existe un programa que resuelve este problema llamado traductor. Éste se encarga de leer lo escrito por el pintor a partir de una dirección de memoria y traducirlo. Luego escribe el mensaje en otra posición de memoria como si se tratada de la memoria de video.

La tarea pintor escribe su inentendible mensaje a partir de la dirección `0xB8000`. Por otro lado, la tarea traductor lee a partir de la dirección `0x10000` y escribe a partir de la dirección `0x18000`.

Ambas tareas están en código binario y son incluidas por en `kernel.asm` utilizando la directiva del ensablador `incbin`. El kernel está escrito de forma de soportar el siguiente mapa de memoria física:

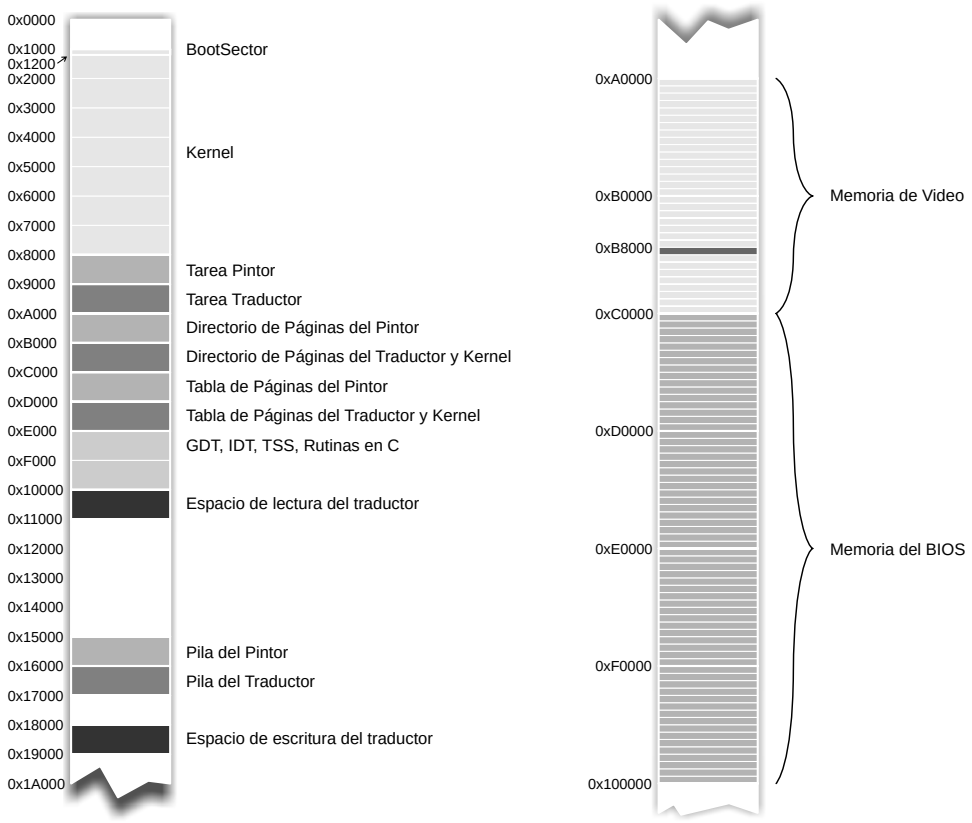


Figura 1: Mapa de memoria física

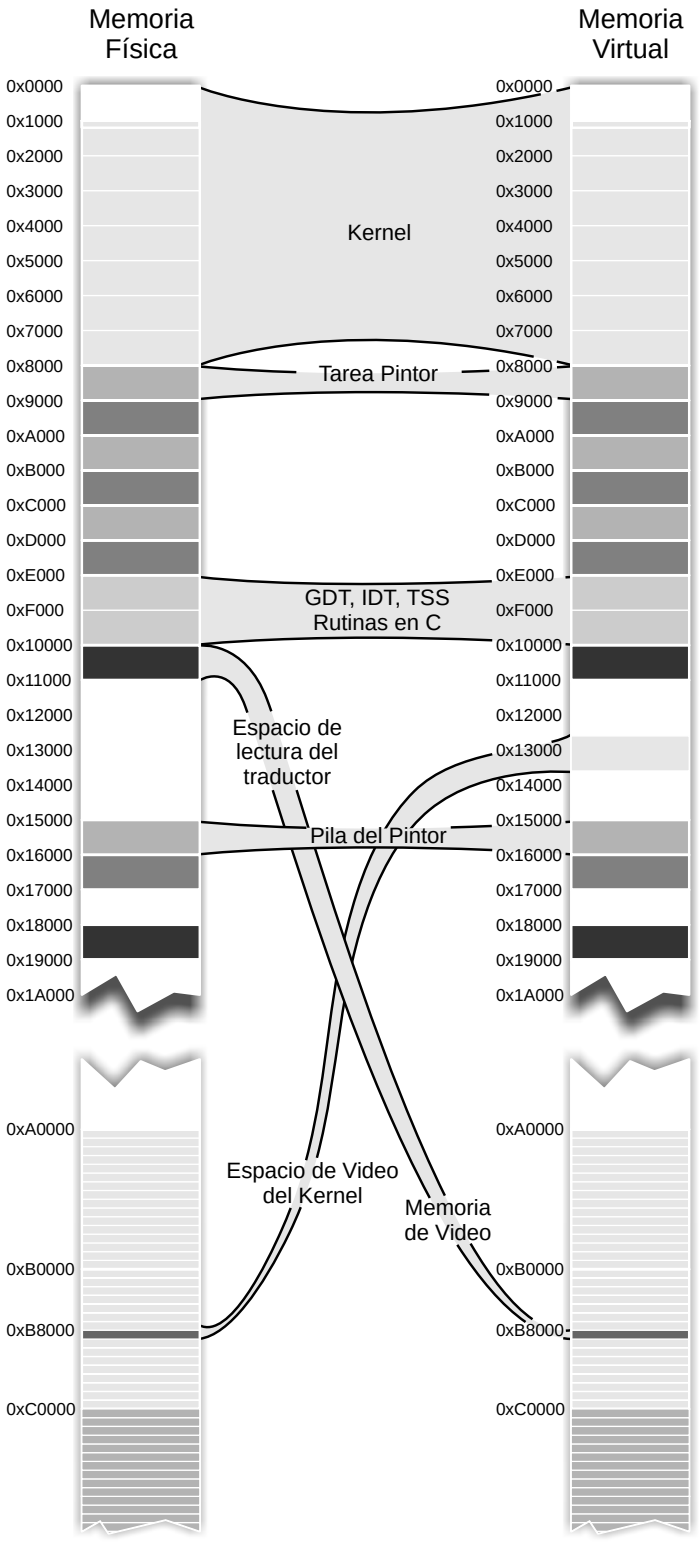


Figura 2: Mapa de memoria virtual para el proceso *pintor*

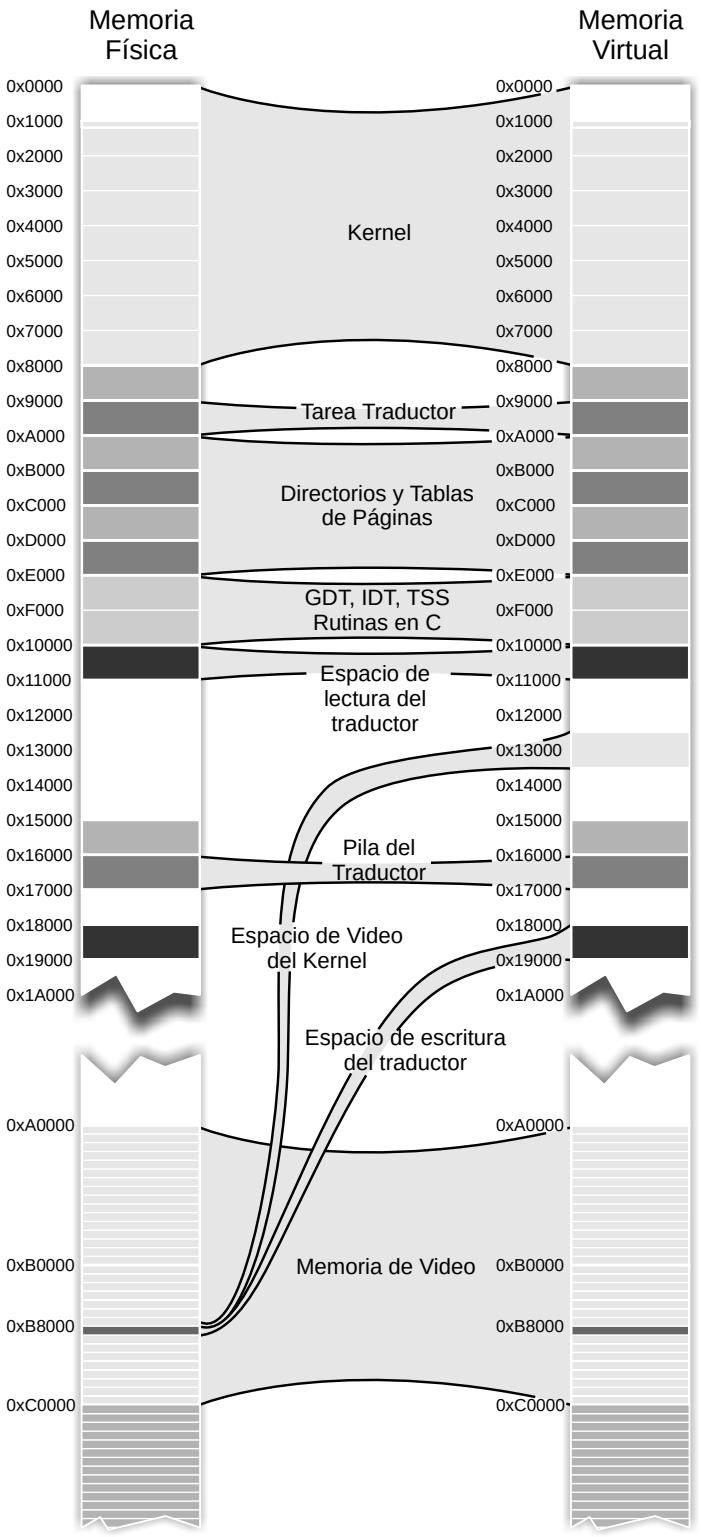


Figura 3: Mapa de memoria virtual para el proceso *Traductor*

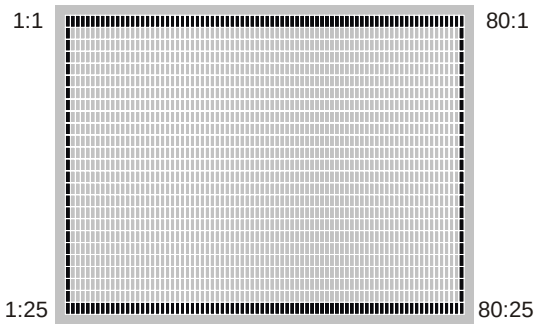
4. Ejercicios

4.1. Ejercicio 1

a) Completar la Tabla de Descriptores Globales (`GDT`) con 2 segmentos, uno para código y otro para datos de que ocupen todo el espacio de 4Gb. Ambos deben ser segmentos de nivel 0 y ocupar los índices 2 y 3 respectivamente. Además se debe completar el indice 4 con un segmento que direccione a la memoria de video.

b) Completar el código necesario para pasar a modo protegido.

c) Escribir una rutina que se encargue de limpiar la pantalla y pintar un marco. El marco consiste en escribir las posiciones de la pantalla comprendidas entre `1:1` a `80:1`, `80:1` a `80:25`, `1:1` a `1:25` y `1:25` a `80:25` con algún caracter a gusto. Para hacer esto se debe direccionar a memoria utilizar el segmento en el índice 4.



Nota: La `GDT` es un arreglo de `gdt_entry` declarado solo una vez como `gdt`. El descriptor de la `GDT` en el código se llama `GDT_DESC`.

4.2. Ejercicio 2

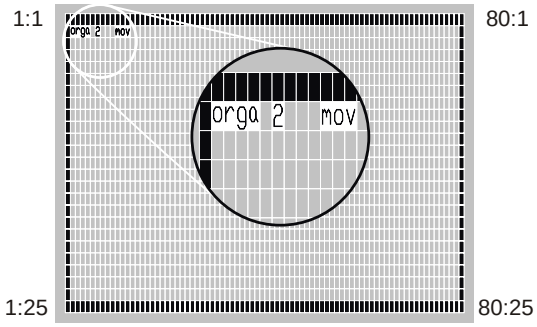
a) Escribir las rutinas encargadas de inicializar los directorios y tablas de paginas según indican los mapas de memoria. Para esto se deben generar dos directorios de paginas, uno para el pintor y otro para traductor y kernel, que utilizarán el mismo.

El directorio del pintor configura identity mapping sobre los rangos de direcciones, `0x0000` a `0x8FFF`, `0xE000` a `0xFFFF` y `0x15000` a `0x15FFF`. Además la página `0xB8000` de memoria virtual, debe ser mapeada a la dirección `0x10000` en memoria física. Al igual que la página `0x13000` de memoria virtual, debe ser mapeada a la dirección `0xB8000` en memoria física.

El directorio de la tarea traductor y kernel, debe configurar tambien identity mapping sobre los rangos de direcciones `0x0000` a `0x7FFF`, `0x9000` a `0x10FFF` y `0x16000` a `0x16FFF` y `0xA0000` a `0xBFFFF`. También se debe mapear la página `0x18000` de memoria virtual, a la dirección física `0xB8000`. Al igual que la página `0x13000` de memoria virtual, debe ser mapeada a la dirección `0xB8000` en memoria física.

En la figura 2 se muestra graficamente el mapa de memoria virtual para el proceso pintor. Para el proceso traductor se muestra el mapa en la figura 3.

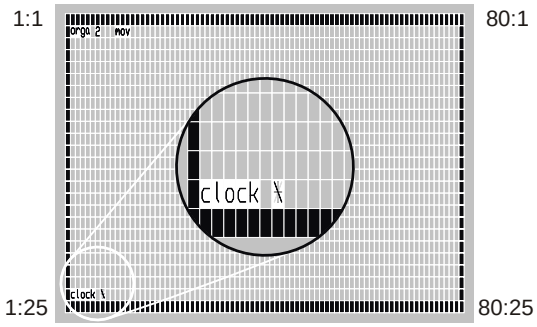
b) Escribir una rutina que muestre el nombre del grupo a partir de la posición `2:2` de la pantalla.



4.3. Ejercicio 3

a) Completar las entradas necesarias en la `IDT` para asociar una rutina a la interrupción del reloj y diferentes rutinas a todas las excepciones del procesador. Cada rutina de excepcion debe indicar en pantalla que problema se produjo e interrumpir la ejecución. Para escribir en pantalla, las rutinas deben hacerlo a travez de la dirección `0x13000` que está mapeada a memoria de video.

b) Escribir la rutina asociada a la interrupción del reloj, para que por cada tick llame a la función `next_clock`.



La función `next_clock` se encarga de mostrar, por cada vez que se llama, la animación de un cursor rotando. Esta función está definida en `ISR.asm`.

Nota: La `IDT` es un arreglo de `idt_entry` declarado solo una vez como `idt`. El descriptor de la `IDT` en el código se llama `IDT_DESC`. Para inicializar la `IDT` se debe invocar la función `idtFill()`.

4.4. Ejercicio 4

a) Completar en memoria las `TSS` correspondientes a las dos tareas, *pintor* y *traductor*. Esta información se encuentra en el archivo `TSS.C`.

b) Completar en la `GDT` las entradas de las `TSS`.

c) Escribir la rutina que intercambia las tareas por cada tick de reloj.

Nota: En `TSS.C` esta definido un arreglo llamado `tss` que contiene las estructuras `TSS`.

5. Entrega

La resolución de los ejercicios se debe realizar gradualmente. Dentro del archivo `kernel.asm` se encuentran comentarios (que muestra las funcionalidades que deben implementarse) para resolver cada ejercicio. También se deberán completar el resto de los archivos según corresponda.

Se deberá entregar un informe que describa detalladamente la implementación de cada uno de los fragmentos de código que fueron construidos para completar el kernel. En el caso que se requiera código adicional también debe estar descripto en el informe. Se deberán utilizar tanto pseudocódigos como esquemas gráficos, o cualquier otro recurso pertinente que permita explicar la resolución. Además se deberá entregar en soporte digital el código e informe. Incluyendo todos los archivos que hagan falta para compilar y correr el trabajo en Bochs.

La fecha de entrega de este trabajo es martes 24 de noviembre, en el horario de clase (de 17 a 22 hs).