Organización del Computador II
TP2

16 de junio de 2015

Integrante	LU	Correo electrónico
Federico Beuter	827/13	federicobeuter@gmail.com
Juan Rinaudo	864/13	jangamesdev@gmail.com
Mauro Cherubini	835/13	cheru.mf@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción

Aca va la introduccion

2. Ejercicio 1

Para la resolucion de este ejercicio modificamos `kernel.asm`, `gdt.c`, `gdt.h`, `defines.h` y `screen.c`.

2.1. a

Comenzamos desde `C` creando los defines de los indices de los descriptores de segmento en la gdt en `defines.h`, como los primeros 8 indices se consideran utilizados vamos a enumerar nustros descriptores desde 8 (Como indica el subindice a).
`GDT_NIVEL0_CODIGO 8`
`GDT_NIVEL0_DATOS 9`
`GDT_NIVEL3_CODIGO 10`
`GDT_NIVEL3_DATOS 11`

Luego en el archivo `gdt.c` agregamos al array `gdt` los 4 descriptores, con su respectivo nivel (0 para los descriptores del kernel y 3 para los de usuario) y tipo (`0xA` para los de codigo y `0x2` para los de datos), su base la posicionamos al principio de la memoria, con un limite de 500MB seteando granularity (`G = 1`) y el valor de limite (`0x1F400`), marcandolo como presente (`P = 1`) y de 32 bits (`D/B = 1`) y no de sistema (`S = 1`).

2.2. b

Para este subindice pasamos a `kernel.asm` donde habilitamos **A20** (utilizando la funcion brindada por la catedra `habilitar_A20`) y luego cargamos la **GDT** usando `LGDT` y un puntero al descriptor de **GDT** `GDT_DESC` (Estructura brindada por la catedra y definida en `gdt.h`). A continuacion habilitamos la proteccion (`PE = 1`. Protected Mode Enable) en **CR0** pasandolo a `EAX` haciendo un `OR` y moviendolo nuevamente a `CR0`. Ahora para pasar a modo protegido hacemos un salto al descriptor de nivel 0 de codigo (`0x40`) en la **GDT** a travez de un `JMP FAR`.

Una vez en modo protegido seteamos los selectores de segmento de datos y de stack de nivel 0, y finalizamos posicionando la pila (`EBP` y `ESP`) en `0x27000`.

2.3. c

Para este punto volvemos a `defines.h` y definimos un quinto indice para el descriptor de la memoria de la pantalla.
`GDT_PANTALLA 12`

Luego en `gdt.c` agregamos al array `gdt` un descriptor que comienza en `0xB8000` (Memoria de video) con un limite de 4KB, de nivel 0, con granularidad y el resto de los bits igual que los descriptores anteriores.

2.4. d

Para este punto vamos a `screen.c` y completamos las siguientes funciones auxiliares:
`screen_inicializar`: Limpia la pantalla y luego llama a `screen_pintar_puntajes`.
`screen_pintar_rect`: Pinta un rectangulo de un color en la pantalla, comienza de un punto (x, y) y tiene un tamaño relativo (width, height).
`screen_pintar_puntajes`: Pinta los puntajes igual que la figura 3 del enunciado.

3. Ejercicio 2

4. Ejercicio 3

Para resolver este ejercicio se modicara `screen.c`, `screen.h`, `mmu.c`, `mmu.h`, `kernel.asm` y `defines.h`.

4.1. a

Para comenzar vamos a modificar `screen.c` y `screen.h`, creamos la funcion en `C` `screen_refrescar` que se encargara de limpiar la pantalla y dibujar el fondo usando `screen_inicializar` y luego escribir la informacion del juego (Los datos de las tareas de ambos jugadores y los puntajes) usando la funcion `print` (Brindada por la catedra).

4.2. b

Definimos `DIR_PAGINAS_KERNEL` como `0x27000` en `defines.h`, luego vamos a `mmu.h` y creamos `mmu_inicializar_dir_kernel`, esta funcion crea en la posicion de memoria definida anteriormente el directorio de tablas de paginas del kernel y en la siguiente pagina (`0x28000`) la primera tabla de paginas del kernel, la memoria sera mapeada con identity mapping de `0x0` a `0x3FFFFF`.

4.3. c

Vamos a `kernel.asm`, llamamos a `inicializar_mmu` y despues a `mmu_inicializar_dir_kernel` para crear el directorio de tablas y las tablas. Luego muevo a **CR3** la posicion del directorio de tablas de paginas (`DIR_PAGINAS_KERNEL`). Luego activamos paginacion en **CR0** (**PG = 1**) (Paging) de **CR0**.

4.4. d

Para terminar en `screen.c/h` creamos la funcion `print_group` que usa print para escribir el nombre del grupo (Alineado a la derecha, tomando el tamaño del string y restandoselo al tamaño total de la pantalla) y la llamamos desde `kernel.asm`.

5. Ejercicio 4

Para resolver este ejercicio se modicara `mmu.c`, `mmu.h` y `kernel.asm`.

5.1. a

Comenzamos editando `mmu.c` y creando la funcion `mmu_inicializar` que se asigna la primera pagina de la memoria libre para guardar valores del sistema. Y en `0x100000` guarda un puntero a la siguiente pagina util.

5.2. b

Ahora escribimos `mmu_inicializar_dir_pirata` que se encargara de crear el directorio de pagina de las tareas, que pide dos paginas libres y crea el directorio de tablas en la primera y una tabla de paginas en la segunda, haciendo identity mapping de los primeros 500MB. Y luego mapeamos un sector de la memoria que no esta asignado a nada (`0x401000`) a la posicion del mapa correspondiente (`X:1,Y:1` si es el primer jugado y `X:78,Y:48` para el segundo jugador), copia el codigo de la tarea (Que se pasa como parametro a la funcion) para terminar desmapeamos la posicion mapeada anteriormente.

5.3. c

Para la tarea anterior es neceario crear las dos funcciones `mmu_mapear_pagina` y `mmu_unmapear_pagina` que con un **CR3**, una direccion virtual y una fisica en el caso de mapeo o una direccion virtual en el caso de desmapeo modifica el directorio de tablas y la tabla de paginas de la **CR3** pasada como parametro y mapea o desmapea la pagina.

5.4. d

Para este punto vamos a `kernel.asm` y creamos un segmento de codigo para testear las cosas echas. Llamamos a `mmu_inicializar_dir_pirata` y cambiamos el **CR3** del sistema con el que devuelve la funcion. Modificamos la memoria de video para cambiar el color de fondo del primer caracter de la pantalla. Esto luego fue eliminado del kernel.asm como pide el subindice.

```
1    PUSH <POS_CODIGO>
2    MOV EAX, CR3
3    PUSH EAX
4    PUSH <POS_PIRATA>
5    CALL mmu_inicializar_dir_pirata
6    SUB ESP, 12
7    MOV CR3, EAX
8    CALL print_group
```

6. Ejercicio 5

7. Ejercicio 6

Para resolver este ejercicio se modicara `tss.h`, `tss.c`, `kernel.asm` y `defines.h`.

7.1. a

Primero vamos a `defines.h` y definimos los indices de los descriptores de **TSS** en la **GDT**.

```
TSS_INICIAL 13
TSS_IDLE 14
```

7.2. b

Para este punto definimos `tss_inicializar` en `tss.h/c` para crear la tabla de `tss_idle`, con `EIP` en `0x16000`, con el mismo **CR3** que el kernel, la misma pila, y los descriptores de segmento de datos y codigo de nivel 0.

7.3. c

Definimos `completar_tabla_tss` en `tss.h/c` que toma una tabla **TSS**, un puntero a codigo de la tarea y un puntero a un lugar donde guardar la **CR3**.

7.4. d, e

Luego definimos la funcion `agregar_descriptor_tss` en `tss.h/c` que toma como parametro un indice de la **GDT** y un puntero a un puntero a una tabla tss y agrega en la **GDT** un descriptor de **TSS** en el indice. Luego modificamos `tss_inicializar` y usamos `agregar_descriptor_tss` para agregar el descriptor de `tss_inicial` y `tss_idle`.

7.5. f

Para saltar a la tarea idle vamos a modificar `kernel.asm`, primero limpiamos `EAX` ya que vamos a pasar el descriptor de **TSS** de `tss_inicial` a ese registro, movemos el indice del descriptor a `AX` (`0x68`) y usando `LTR` lo cargamos en el registro especial de tareas. Luego hacemos un `JMP FAR` a el indice del descriptor de tareas en la **GDT** de `tss_idle` (`0x70`).

7.6. g

7.7. h

8. Ejercicio 7

9. Ejercicio 8