

# 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

<b>1. Introducción</b>	<b>3</b>
<b>2. Ejercicio 1</b>	<b>4</b>
2.1. a . . . . .	4
2.2. b . . . . .	4
2.3. c . . . . .	4
2.4. d . . . . .	4
<b>3. Ejercicio 2</b>	<b>5</b>
<b>4. Ejercicio 3</b>	<b>6</b>
4.1. a . . . . .	6
4.2. b . . . . .	6
4.3. c . . . . .	6
4.4. d . . . . .	6
<b>5. Ejercicio 4</b>	<b>7</b>
5.1. a . . . . .	7
5.2. b . . . . .	7
5.3. c . . . . .	7
5.4. d . . . . .	7
<b>6. Ejercicio 5</b>	<b>8</b>
<b>7. Ejercicio 6</b>	<b>9</b>
7.1. a . . . . .	9
7.2. b . . . . .	9
7.3. c . . . . .	9
7.4. d, e . . . . .	9
7.5. f . . . . .	9
7.6. g . . . . .	9
7.7. h . . . . .	9
<b>8. Ejercicio 7</b>	<b>10</b>
<b>9. Ejercicio 8</b>	<b>11</b>

# 1. Introducción

Aca va la introduccion

## 2. Ejercicio 1

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

### 2.1. a

Comenzamos desde `C` creando las definiciones de las tablas de descriptors en `defines.h`, como los primeros 8 índices se consideran utilizadas vamos a enumerar nuestros descriptors desde 8 (Como indica el subíndice a), creando los siguientes defines.

```
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 descriptors, con su respectivo nivel (0 para los descriptors del kernel y 3 para los de usuario) y tipo, con su base en el principio de la memoria y su límite en 500MB, seteando granularidad (G) y el valor de límite (0x1F400), marcándolo como presente y de 32 bits (D/B) y no de sistema (S).

### 2.2. b

Para este subíndice pasamos a `kernel.asm` donde habilitamos **A20** (utilizando la función brindada por la cátedra `habilitar_A20`) y luego cargamos la **GDT** usando `LGDT` y el puntero al descriptor `GDT_DESC` (Estructura brindada por la cátedra y definida en `gdt.h`). A continuación activamos el bit PE (Protected Mode Enable) en **CR0** moviendo y hacemos el salto al descriptor de nivel 0 de código (0x40) en la **GDT** usando un `JMP FAR`. Una vez echo esto nos encontramos en modo protegido.

Ahora 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 índice para el descriptor de la memoria de la pantalla.  
`GDT_PANTALLA 12`

Y luego en `gdt.c` agregamos al array `gdt` un descriptor que comienza en 0xB8000 (Memoria de video) con un límite de 32KB, de nivel 0, sin granularidad y el resto de los bits igual que los descriptors anteriores.

### 2.4. d

Para este punto vamos a `screen.c` y completamos las funciones auxiliares `screen_inicializar` (Limpia la pantalla y escribe los puntajes), `screen_pintar_rect` (Pinta un rectángulo de un color en la pantalla), `screen_pintar_puntajes` (Pinta los puntajes igual que la figura 3 del enunciado).

### 3. Ejercicio 2

## 4. Ejercicio 3

Para resolver este ejercicio se modificara `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 sistema (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 de `DIR_PAGINAS_KERNEL` el directorio de tablas de paginas y en la siguiente pagina (`0x28000`) la primera tabla de paginas que posee las paginas de `0x0` a `0x3FFFFFF` con identity mapping.

### 4.3. c

Vamos a `kernel.asm` y llamamos el extern `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 para activar paginacion seteamos el bit **PG** (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) y la llamamos desde `kernel.asm`, con esto podemos confirmar que la pagina esta funcionando.

## 5. Ejercicio 4

Para resolver este ejercicio se modificara `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 necesario crear las dos funciones `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.

## 6. Ejercicio 5



## 7. Ejercicio 6

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

### 7.1. a

Primero vamos a `defines.h` y definimos los indices de los descriptors 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 descriptors 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