

# Organizacion del Computador II

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

## Trabajo Práctico N°3

**Grupo enifra**

Integrante	LU	Correo electrónico
Serio, Franco	215/15	francoagustinserio@gmail.com

**Reservado para la cátedra**

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

# Índice

<b>1. Introduccion</b>	<b>3</b>
<b>2. Modo Real</b>	<b>4</b>
<b>3. Modo Protegido</b>	<b>5</b>
3.1. Segmentacion . . . . .	5
3.2. Paginado . . . . .	5
3.3. Manejo de Interrupciones . . . . .	8
3.4. Tareas . . . . .	9
3.5. Scheduler . . . . .	13

# 1. Introduccion

Este trabajo práctico *Tierra Pirata* consiste en aplicar los conceptos de System Programming vistos en clase.

Construimos un sistema mínimo que permite correr hasta 16 tareas concurrentes a nivel de usuario.

El sistema es capaz de capturar cualquier problema que puedan generar las tareas y tomar las acciones necesarias para quitar a la tarea del sistema.

Algunas tareas podrán ser cargadas en el sistema de manera dinamica por medio de uso del teclado.

Con la realizacion de este trabajo, vamos a utilizar los mecanismos que posee el procesador para la programación desde el punto de vista del sistema operativo enfocados en dos aspectos: el sistema de protección y la ejecución concurrente de tareas.

El informe consta de varias secciones, en las cuales se explica lo realizado para poder hacer funcionar el sistema.

## 2. Modo Real

Cuando encendemos nuestra computadora, la misma inicia en *Modo Real*. En este modo tenemos una memoria disponible de 1mb, no disponemos de privilegios de ningun tipo, y debemos alinear la memoria a 16 bits para poder operar. Para manejar los distintos tipos de interrupciones, disponemos de rutinas de atencion a las mismas. Podemos acceder a instrucciones de cualquier tipo. Estando en Modo real, lo unico que vamos a hacer, es preparar el sistema para poder pasar a *Modo Protegido*, y realizar todo lo que necesitamos que haga nuestro sistema. Para nuestro sistema, definimos el código de *Modo Real* en el archivo *kernel.asm*:

```

BITS 16
start:
    ; Deshabilitar interrupciones
    cli

    ; Cambiar modo de video a 80 X 50
    mov ax, 0003h
    int 10h ; set mode 03h
    xor bx, bx
    mov ax, 1112h
    int 10h ; load 8x8 font

    ; Imprimir mensaje de bienvenida
    imprimir_texto_mr iniciando_mr_msg, iniciando_mr_len, 0x07, 0, 0
    ; Habilitar A20
    call habilitar_A20
    ; Cargar la GDT
    lgdt [GDT_DESC]

    ; Setear el bit PE del registro CR0
    mov eax, cr0
    or eax, 1
    mov cr0, eax

    ; Saltar a modo protegido
    jmp 0x58:mp

```

Mediante este set de instrucciones estamos preparando nuestro sistema para poder pasar a *Modo Protegido*. Dentro del codigo, estamos habilitando A20 que nos da accesos a direcciones superiores a los  $2^{20}$  bits. Cargamos la *GDT* (Global Descriptor Table). La *GDT* es una tabla ubicada en memoria que define los siguientes descriptores:

- Descriptores de segmento de memoria
- Descriptor de Task State Segment (TSS)

El primer descriptor de la tabla siempre es *nulo*. También al principio tenemos otros 4 segmentos. 1 de código nivel kernel, 1 de código nivel usuario, 1 de datos nivel usuario y 1 de datos nivel kernel. Los segmentos de user tienen dpl 3 y los de sistema 0. A su vez, el límite de los segmentos de la gdt se saca porque en el TP dice que se direccionan los primeros 500MB de memoria. Eso significa que tenemos que usar granularidad de a 4KB de lo contrario no llegamos. El limite se calcula de hacer  $(500\text{MB} / 4\text{KB}) - 1$ . Esto es porque granularity como dijimos está seteado. Luego seteamos el bit *PE* del registro *CR0*.

Una vez activado, vamos a saltar a *Modo Protegido*. Este salto lo realizamos utilizando un far jump al descriptor de segmento de código con nivel de privilegio de sistema. Es el jmp que realizamos en la última línea.

### 3. Modo Protegido

El *Modo Protegido*, es un poco mas completo que el *Modo Real*, tenemos 4GB de memoria disponible, 4 niveles de protección de memoria y rutinas de atención con privilegios.

#### 3.1. Segmentacion

Nos encargamos de setear los offset de los descriptores de los selectores de segmentos. Luego seteamos la pila del kernel en la dirección 0x27000

```

BITS 32
mp:
    ; Establecer selectores de segmentos
    xor eax, eax
    mov ax, 1001000b
    mov ds, ax
    mov ss, ax
    mov es, ax
    mov gs, ax
    mov ax, 1100000b
    mov fs, ax
    ; Establecer la base de la pila
    mov esp, 0x27000
    mov ebp, 0x27000

```

Luego de esto imprimimos en pantalla un mensaje de bienvenida a nuestro sistema e inicializamos la pantalla y el juego.

```

    ; Imprimir mensaje de bienvenida
    imprimir_texto_mp iniciando_mp_msg, iniciando_mp_len, 0x07, 2, 0
    call game_inicializar
    call screen_inicializar
    call screen_pintar_nombre
    call screen_pintar_puntajes

```

Las funciones *screen\_inicializar*, *screen\_pintar\_nombre* estan definidas en **screen.h** y desarrolladas en **screen.c**. Estos métodos se encargan de pintar la pantalla, según lo pedido en el enunciado. La función *game\_inicializar* esta definida en **game.c** que inicializa los 2 jugadores y el reloj de cada pirata.

#### 3.2. Paginado

Habilitamos la paginación en nuestro sistema de la siguiente manera:

```

    ; Cargar directorio de paginas
    ; Inicializar el directorio de paginas
    call mmu_inicializar
    call mmu_inicializar_dir_kernel
    xor eax, eax
    mov eax, 0x27000
    mov cr3, eax
    ; Habilitar paginacion
    xor eax, eax
    mov eax, cr0
    or eax, 0x80000000
    mov cr0, eax

```

Para mapear y unmapear definimos métodos en el archivo *mmu.c*. Para mapear:

```

void mmu_mapear_pagina(unsigned int virtual, unsigned int cr3,

```

```

unsigned int fisica, unsigned char read_write, unsigned char user_supervisor)
    page_directory_entry* pd = (page_directory_entry*)(cr3);
    unsigned int indiceDirectory = virtual >> 22;
    unsigned int indiceTable = (virtual >> 12) << 10;
    page_directory_entry pde = pd[indiceDirectory];
    if (pde.present == 1)
        page_table_entry* pt = (page_table_entry*)((pde.base_address << 12) >> 12);
        page_table_entry pte = pt[indiceTable];
        if (pte.present == 1)
            pte.user_supervisor = user_supervisor;
            pte.read_write = read_write;
            pte.base_address = (fisica >> 12);
        else
            pte.present = 1;
            pte.user_supervisor = user_supervisor;
            pte.read_write = read_write;
            pte.base_address = (fisica >> 12);

    else
        unsigned int proxima_pag = mmu_proxima_pagina_fisica_libre();
        pde.present = 1;
        pde.read_write = read_write;
        pde.user_supervisor = user_supervisor;
        pde.base_address = proxima_pag >> 12;
        page_table_entry* pt = (page_table_entry*)(proxima_pag);
        pt[indiceTable].present = 1;
        pt[indiceTable].user_supervisor = user_supervisor;
        pt[indiceTable].read_write = read_write;
        pt[indiceTable].base_address = (fisica >> 12);

    tlbflush();

```

Finalmente, para unmapear una pagina, lo hacemos de la siguiente manera:

```

void mmu_unmapear_pagina(unsigned int virtual, unsigned int cr3)
    page_directory_entry* pd = (page_directory_entry*)(cr3);
    unsigned int indiceDirectory = virtual >> 22;
    unsigned int indiceTable = (virtual >> 12) << 10;
    page_directory_entry pde = pd[indiceDirectory];
    page_table_entry* pt = (page_table_entry*)(pde.base_address << 12);
    pt[indiceTable].present = 0;
    int i = 0;
    int estanTodasEnNotPresent = 1;
    while (i < 1024 && estanTodasEnNotPresent == 1)
        page_table_entry* pt = (page_table_entry*)(pde.base_address << 12);
        page_table_entry pte = pt[i];
        if (pte.present == 1)
            estanTodasEnNotPresent = 0;

        i++;

    if (estanTodasEnNotPresent == 1)
        pde.present = 0;

    tlbflush();

```

Durante el juego, debemos poder paginar de manera dinámica las direcciones de los piratas que se agregan, para

poder realizarlo utilizamos el siguiente método:

```

unsigned int mmu_proxima_pagina_fisica_libre()
    unsigned int pagina_libre = proxima_pagina_libre;
    proxima_pagina_libre += PAGE_SIZE;
    return pagina_libre;

unsigned int mmu_inicializar_dir_pirata(jugador_t * jugador, pirata_t * tarea){ //aca ya tengo la page
directory y table de la tarea.
    page_directory_entry * pd = (page_directory_entry*)mmu_proxima_pagina_fisica_libre();
    page_table_entry * pt = (page_table_entry*)mmu_proxima_pagina_fisica_libre();
    pd[0].present = 1;
    pd[0].read_write = 1;
    pd[0].user_supervisor = 0;
    pd[0].write_through = 0;
    pd[0].cache_disable = 0;
    pd[0].accessed = 0;
    pd[0].ignored = 0;
    pd[0].page_size = 0;
    pd[0].global = 0;
    unsigned int page_table_address = (unsigned int)pt;
    pd[0].base_address = page_table_address >> 12;
    for(int i = 0; i < 1024; i++){
        pt[i].present = 1;
        pt[i].read_write = 1;
        pt[i].user_supervisor = 0;
        pt[i].write_through = 0;
        pt[i].cache_disable = 0;
        pt[i].accessed = 0;
        pt[i].dirty = 0;
        pt[i].attribute_index = 0;
        pt[i].global = 0;
        pt[i].base_address = i;
    }
    unsigned int page_directory_address = (unsigned int)pd;
    if(jugador->index == JUGADOR_A){ //empiezo en la primera posicion //mapeo la 0x400000
        mmu_mapear_pagina((unsigned int)0x00400000, (page_directory_address), pos2mapFis(1, 2), 1, 1);
        breakpoint();
        if(tarea->tipo == explorador){
            for(int i = 0; i < 1024; i++){
                ((unsigned int*)((unsigned int)0x00400000))[i] = ((unsigned int*)((unsigned int)0x10000))[i];
            }
        }else{
            for(int i = 0; i < 1024; i++){
                ((unsigned int*)((unsigned int)0x00400000))[i] = ((unsigned int*)((unsigned int)0x11000))[i];
            }
        }
        breakpoint(); //mapeo donde estamos parados
        mmu_mapear_pagina(pos2mapVir(1, 2), (page_directory_address), pos2mapFis(1, 2), 1, 1);
        if(tarea->tipo == explorador){ //explorador
            for(int i = 0; i < 1024; i++){ ((unsigned int*)(pos2mapVir(1, 2)))[i] = ((unsigned int*)(unsigned
int)0x10000))[i];
            }
        }else{ //minero
            for(int i = 0; i < 1024; i++){ ((unsigned int*)(pos2mapVir(1, 2)))[i] = ((unsigned int*)(unsigned
int)0x11000))[i];
            }
        }
    }

```

```

    mmu_unmapear_pagina(pos2mapVir(1,2), (page_directory_address));
    //se mapean las de alrededor para jugador 1
    mmu_mapear_pagina(pos2mapVir(2,1), (page_directory_address), pos2mapFis(2,1), 0, 1); //derecha
    mmu_mapear_pagina(pos2mapVir(2,2), (page_directory_address), pos2mapFis(2,2), 0, 1); //abajo-
derecha
    mmu_mapear_pagina(pos2mapVir(1,3), (page_directory_address), pos2mapFis(1,3), 0, 1); //abajo
    mmu_mapear_pagina(pos2mapVir(0,1), (page_directory_address), pos2mapFis(0,1), 0, 1); //izquierda
    mmu_mapear_pagina(pos2mapVir(0,2), (page_directory_address), pos2mapFis(0,2), 0, 1); //abajo-
izquierda
    mmu_mapear_pagina(pos2mapVir(1,1), (page_directory_address), pos2mapFis(1,1), 0, 1); //arriba
    mmu_mapear_pagina(pos2mapVir(0,3), (page_directory_address), pos2mapFis(0,3), 0, 1); //arriba-
izquierda
    mmu_mapear_pagina(pos2mapVir(2,3), (page_directory_address), pos2mapFis(2,3), 0, 1); //arriba-
derecha
} else { //empiezo en la ultima posicion
    breakpoint(); //mapeo la 0x400000
    mmu_mapear_pagina((unsigned int)0x00400000, (page_directory_address), pos2mapFis(1,2), 1, 1);
    if (tarea- > tipo == explorador) {
        for (inti = 0; i < 1024; i++) { ((unsigned int*)((unsigned int)0x00400000))[i] = ((unsigned
int*)((unsigned int)0x12000))[i];
        }
    } else {
        for (inti = 0; i < 1024; i++) {
            ((unsigned int*)((unsigned int)0x00400000))[i] = ((unsigned int*)((unsigned int)0x13000))[i];
        }
    }
    breakpoint(); //mapeo donde estamos parados
    mmu_mapear_pagina(pos2mapVir(78,43), (page_directory_address), pos2mapFis(78,43), 0, 1);
    if (tarea- > tipo == explorador) { //explorador
        for (inti = 0; i < 1024; i++) { ((unsigned int*)(pos2mapVir(78,43)))[i] = ((unsigned int*)((unsigned
int)0x12000))[i];
        }
    } else { //minero
        for (inti = 0; i < 1024; i++) {
            ((unsigned int*)(pos2mapVir(78,43)))[i] = ((unsigned int*)((unsigned int)0x13000))[i];
        }
    }
    mmu_unmapear_pagina(pos2mapVir(78,43), (page_directory_address));
    //al principio solo se mapean las paginas de la izquierda, arriba, arriba-izquierda para jug 2
    mmu_mapear_pagina(pos2mapVir(77,43), (page_directory_address), pos2mapFis(77,43), 0, 1); //izquierda
    mmu_mapear_pagina(pos2mapVir(77,42), (page_directory_address), pos2mapFis(77,42), 0, 1); //arriba-
izquierda
    mmu_mapear_pagina(pos2mapVir(78,42), (page_directory_address), pos2mapFis(78,42), 0, 1); //arriba
    mmu_mapear_pagina(pos2mapVir(79,43), (page_directory_address), pos2mapFis(79,43), 0, 1); //derecha
    mmu_mapear_pagina(pos2mapVir(79,42), (page_directory_address), pos2mapFis(79,42), 0, 1); //arriba
derecha
    mmu_mapear_pagina(pos2mapVir(78,44), (page_directory_address), pos2mapFis(78,44), 0, 1); //abajo
    mmu_mapear_pagina(pos2mapVir(79,44), (page_directory_address), pos2mapFis(79,44), 0, 1); //abajo
derecha
    mmu_mapear_pagina(pos2mapVir(77,44), (page_directory_address), pos2mapFis(77,44), 0, 1); //abajo
izquierda
}
breakpoint();
return page_directory_address;
}

```



### 3.3. Manejo de Interrupciones

Para poder atender los distintos tipos de interrupciones, definimos las tareas de atención en *isr.asm*. Definimos una rutina para atender las interrupciones del reloj:

```
global _isr32
_isr32:
    ; PRESERVAR REGISTROS
    pushad
    call fin_intr_pic1
    cmp byte [modoDebugPantalla], 1
    je .fin
    call sched_tick
    str cx
    cmp ax, cx
    je .fin
    mov [sched_tarea_selector], ax
    jmp far [sched_tarea_offset]
.fin:
    ; RESTAURAR REGISTROS
    popad
    iret
```

Luego, para atender las interrupciones correspondientes al teclado lo hacemos de la siguiente manera:

```
global _isr33
_isr33:
    pushad
    call fin_intr_pic1
    xor ax, ax
    in al, 0x60
    push eax
    call game_atender_teclado
    pop eax
    popad
    iret
```

El método *game\_atender\_teclado* está definido en nuestro archivo *game.c*. Este método lo que hace, es imprimir en el rincón derecho superior de la pantalla la tecla que se presionó. Cuenta con un switch, que evalúa el caso de cada tecla posible, y en base a esto imprime lo que corresponde.

Para atender las interrupciones de excepciones que describe el ejercicio 2 del TP, definimos una macro:

```
% macro ISR 1
global _isr %1
_ isr %1:
    mov eax, %1
    imprimir_texto_mp desc_ %1, desc_len_ %1, 0x07, 3, 0
    jmp $
```

De esta manera, podemos definir el mensaje correspondiente para cada interrupción, y utilizando la macro, no tenemos que repetir el código, ya que para todas las interrupciones va a trabajar de la misma manera. Va a imprimir en pantalla el nombre de la interrupción que acaba de ocurrir. Por ejemplo, cuando queremos identificar la interrupción de "Divide Error", definimos el mensaje de la siguiente manera:

```
desc_0 db 'Divide Error'
desc_len_0 equ $ - desc_0
```

De esta misma manera definimos todos los mensajes correspondientes a las interrupciones, en nuestro archivo *isr.asm*.

### 3.4. Tareas

Completamos las TSS utilizando código c, en el archivo *tss.c* Para completar la *idle* y la *inicial*:

```
void tss_inicializar(){  
  
    //tss_idle  
  
    tss_idle.ptl = 0;  
    tss_idle.unused0 = 0;  
    tss_idle.esp0 = 0;  
    tss_idle.ss0 = 0;  
    tss_idle.unused1 = 0;  
    tss_idle.esp1 = 0;  
    tss_idle.ss1 = 0;  
    tss_idle.unused2 = 0;  
    tss_idle.esp2 = 0;  
    tss_idle.ss2 = 0;  
    tss_idle.unused3 = 0;  
    tss_idle.cr3 = (unsigned int)0x27000;  
    tss_idle.eip = (unsigned int)0x16000;  
    tss_idle.eflags = (unsigned int)0x00202;  
    tss_idle.eax = 0;  
    tss_idle.ecx = 0;  
    tss_idle.edx = 0;  
    tss_idle.ebx = 0;  
    tss_idle.esp = (unsigned int)0x27000;  
    tss_idle.ebp = (unsigned int)0x27000;  
    tss_idle.esi = 0;  
    tss_idle.edi = 0;  
    tss_idle.es = (unsigned int)0x48;  
    tss_idle.unused4 = 0;  
    tss_idle.cs = (unsigned int)0x58;  
    tss_idle.unused5 = 0;  
    tss_idle.ss = (unsigned int)0x48;  
    tss_idle.unused6 = 0;  
    tss_idle.ds = (unsigned int)0x48;  
    tss_idle.unused7 = 0;  
    tss_idle.fs = (unsigned int)0x00060;  
    tss_idle.unused8 = 0;  
    tss_idle.gs = (unsigned int)0x48;  
    tss_idle.unused9 = 0;  
    tss_idle.ldt = 0;  
    tss_idle.unused10 = 0;  
    tss_idle.dtrap = 0;  
    tss_idle.iomap = 0;  
  
    //tss_inicial  
  
    tss_inicial.ptl = 0;  
    tss_inicial.unused0 = 0;  
    tss_inicial.esp0 = 0;  
    tss_inicial.ss0 = 0;  
    tss_inicial.unused1 = 0;  
    tss_inicial.esp1 = 0;  
    tss_inicial.ss1 = 0;  
    tss_inicial.unused2 = 0;  
    tss_inicial.esp2 = 0;  
}
```

```

tss_inicial.ss2 = 0;
tss_inicial.unused3 = 0;
tss_inicial.cr3 = 0;
tss_inicial.eip = 0;
tss_inicial.eflags = 0;
tss_inicial.eax = 0;
tss_inicial.ecx = 0;
tss_inicial.edx = 0;
tss_inicial.ebx = 0;
tss_inicial.esp = 0;
tss_inicial.ebp = 0;
tss_inicial.esi = 0;
tss_inicial.edi = 0;
tss_inicial.es = 0;
tss_inicial.unused4 = 0;
tss_inicial.cs = 0;
tss_inicial.unused5 = 0;
tss_inicial.ss = 0;
tss_inicial.unused6 = 0;
tss_inicial.ds = 0;
tss_inicial.unused7 = 0;
tss_inicial.fs = 0;
tss_inicial.unused8 = 0;
tss_inicial.gs = 0;
tss_inicial.unused9 = 0;
tss_inicial.ldt = 0;
tss_inicial.unused10 = 0;
tss_inicial.dtrap = 0;
tss_inicial.iomap = 0;
}

```

Debemos agregar las *TSS* correspondientes a la tarea *idle* y la tarea *inicial* a nuestra *GDT*, para poder realizar esto, definimos un método llamado *tss\_agregar\_a\_gdt*. El método realiza lo siguiente:

```

void tss_agregar_a_gdt(){
    gdt[GDT_TSS_IDLE] = (gdt_entry)
        (unsigned short) 0x0067, /* limit[0:15] */
        (unsigned short) (int)(&tss_idle) & 0xFFFF, /* base[0:15] */
        (unsigned char) (int)((int)(&tss_idle) >> 16) & 0x00FF, /* base[23:16] */
        (unsigned char) 0x09, /* type */
        (unsigned char) 0x00, /* s */
        (unsigned char) 0x00, /* dpl */
        (unsigned char) 0x01, /* p */
        (unsigned char) 0x00, /* limit[16:19] */
        (unsigned char) 0x00, /* avl */
        (unsigned char) 0x00, /* l */
        (unsigned char) 0x00, /* db */
        (unsigned char) 0x00, /* g */
        (unsigned char) (int)(&tss_idle) >> 24, /* base[31:24] */
    ;
    gdt[GDT_TSS_INICIAL] = (gdt_entry)
        (unsigned short) 0x0067, /* limit[0:15] */
        (unsigned short) (int)(&tss_inicial) & 0xFFFF, /* base[0:15] */
        (unsigned char) (int)((int)(&tss_inicial) >> 16) & 0x00FF, /* base[23:16] */
        (unsigned char) 0x09, /* type */
        (unsigned char) 0x00, /* s */
        (unsigned char) 0x00, /* dpl */
        (unsigned char) 0x01, /* p */
        (unsigned char) 0x00, /* limit[16:19] */

```

```

        (unsigned char) 0x00, /* avl */
        (unsigned char) 0x00, /* l */
        (unsigned char) 0x00, /* db */
        (unsigned char) 0x00, /* g */
        (unsigned char) (int)(amp; tss_inicial) >> 24, /* base[31:24] */
    };
}

```

Luego necesitamos completar la TSS correspondiente a cada pirata, para esto realizamos el siguiente método:

```

void completarTssPirata(pirata_t tarea){
    unsigned int paginaParaPilaCero = mmu_proxima_pagina_fisica_libre() + 0x1000;

    tss* tss_pirata = (*(tarea.jugador)).index == JUGADOR_A ? &tss_jugadorA[tarea.id] : &tss_jugadorB[tarea.id];

    tss_pirata->eip = 0x00400000;
    tss_pirata->ptl = 0;
    tss_pirata->unused0 = 0;
    tss_pirata->esp0 = paginaParaPilaCero;
    tss_pirata->ss0 = (unsigned int)0x48;
    tss_pirata->unused1 = 0;
    tss_pirata->esp1 = 0;
    tss_pirata->ss1 = 0;
    tss_pirata->unused2 = 0;
    tss_pirata->esp2 = 0;
    tss_pirata->ss2 = 0;
    tss_pirata->unused3 = 0;
    tss_pirata->cr3 = mmu_inicializar_dir_pirata(tarea.jugador, &tarea);
    tss_pirata->eflags = (unsigned int)0x00202;
    tss_pirata->eax = 0;
    tss_pirata->ecx = 0;
    tss_pirata->edx = 0;
    tss_pirata->ebx = 0;
    tss_pirata->esp = 0x00401000 - 12;
    tss_pirata->ebp = 0x00401000 - 12;
    tss_pirata->esi = 0;
    tss_pirata->edi = 0;
    tss_pirata->es = (unsigned short)0x40 | 0x3;
    tss_pirata->unused4 = 0;
    tss_pirata->cs = (unsigned short)0x50 | 0x3;
    tss_pirata->unused5 = 0;
    tss_pirata->ss = (unsigned short)0x40 | 0x3;
    tss_pirata->unused6 = 0;
    tss_pirata->ds = (unsigned short)0x40 | 0x3;
    tss_pirata->unused7 = 0;
    tss_pirata->fs = (unsigned short)0x40 | 0x3;
    tss_pirata->unused8 = 0;
    tss_pirata->gs = (unsigned short)0x40 | 0x3;
    tss_pirata->unused9 = 0;
    tss_pirata->ldt = 0;
    tss_pirata->unused10 = 0;
    tss_pirata->dtrap = 0;
    tss_pirata->iomap = 0xFFFF;
}

```

Al igual que hicimos con la tarea *idle* y la tarea *inicial*, debemos agregar las tareas correspondientes a los piratas a nuestra *GDT*, para esto desarrollamos el método *tss\_agregar\_piratas\_a\_gdt*

```

void tss_agregar_piratas_a_gdt(jugador_t* j) {
    if (j->index == 0) {

```

```

    gdt[EMPIEZAN_TSS + proximaTareaA] = (gdt_entry) {
        (unsigned short) 0x0067, /* limit[0:15] */
        (unsigned short) (int)(&tss_jugadorA[jugadorA.piratas[proximaTareaA].index]) & 0xFFFF, /* base[0:15] */
        (unsigned char) (int)((int)(&tss_jugadorA[jugadorA.piratas[proximaTareaA].index]) >> 16) & 0x00FF,
        /* base[23:16] */
        (unsigned char) 0x09, /* type */
        (unsigned char) 0x00, /* s */
        (unsigned char) 0x03, /* dpl */
        (unsigned char) 0x01, /* p */
        (unsigned char) 0x00, /* limit[16:19] */
        (unsigned char) 0x00, /* avl */
        (unsigned char) 0x00, /* l */
        (unsigned char) 0x00, /* db */
        (unsigned char) 0x00, /* g */
        (unsigned char) (int)(&tss_jugadorA[jugadorA.piratas[proximaTareaA].index]) >> 24, /* base[31:24] */
    };
    completarTssPirata(jugadorA.piratas[proximaTareaA]);
} else {
    gdt[EMPIEZAN_TSS + 8 + proximaTareaB] = (gdt_entry) {
        (unsigned short) 0x0067, /* limit[0:15] */
        (unsigned short) (int)(&tss_jugadorB[jugadorB.piratas[proximaTareaB].index]) & 0xFFFF, /* base[0:15] */
        (unsigned char) (int)((int)(&tss_jugadorB[jugadorB.piratas[proximaTareaB].index]) >> 16) & 0x00FF,
        /* base[23:16] */
        (unsigned char) 0x09, /* type */
        (unsigned char) 0x00, /* s */
        (unsigned char) 0x03, /* dpl */
        (unsigned char) 0x01, /* p */
        (unsigned char) 0x00, /* limit[16:19] */
        (unsigned char) 0x00, /* avl */
        (unsigned char) 0x00, /* l */
        (unsigned char) 0x00, /* db */
        (unsigned char) 0x00, /* g */
        (unsigned char) (int)(&tss_jugadorB[jugadorB.piratas[proximaTareaB].id]) >> 24, /* base[31:24] */
    };
    completarTssPirata(jugadorB.piratas[proximaTareaB]);
}
}

```

### 3.5. Scheduler

Decidimos representar al *Scheduler* de la siguiente manera:

```

uint proximaTareaA; //indice 0-7
uint proximaTareaB; //indice 0-7
uchar turnoPirata; //0 A, 1 B
uchar estaEnIdle; // 0 NO, 1 SI
uint modoDebug;

```

Para inicializar estos valores, definimos el siguiente método:

```

void sched_inicializar(){
    turnoPirata = 0;
    proximaTareaA = 0;
    estaEnIdle = 1;
    proximaTareaB = 0;
}

```

```

    modoDebug = 0;
}

```

Para atender a la proxima tarea, vamos a reutilizar la función que cambia las tareas de acuerdo al tick del clock, utilizamos a la función:

```

unsigned int sched_proxima_a_ejecutar(){
    return sched_tick();
}

```

Luego, para realizar lo mencionado anteriormente, correspondiente al tick del clock:

```

unsigned int sched_tick() {
    if (estaEnIdle == 1) {
        estaEnIdle = 0;
        if (turnoPirata == 0) {
            //turno jug A
            uint proxTarea = EMPIEZAN_TSS + proximaTareaA;
            game_tick(proxTarea);
            turnoPirata = 1;
            uchar noEncontreNinguna = 1;
            uchar todosMuertos = 0;
            int i = proximaTareaA + 1;
            while (noEncontreNinguna == 1 && todosMuertos == 0) {
                if (jugadorA.piratas[i].vivoMuerto == 1) {
                    //si esta vivo la pongo como la proxima tarea de A
                    proximaTareaA = jugadorA.piratas[i].index;
                    noEncontreNinguna = 0;
                }
                if (i == proximaTareaA) {
                    todosMuertos = 1;
                }
                i++;
                if (i == 8) {
                    i = 0;
                }
            }
            if (todosMuertos) { //salto a la idle
                return (13) << 3;
            }
            return (proxTarea << 3);
        } else {
            //turno jug B
            uint proxTarea = EMPIEZAN_TSS + 8 + proximaTareaB;
            game_tick(proxTarea);
            turnoPirata = 0;
            uchar noEncontreNinguna = 1;
            uchar todosMuertos = 0;
            int i = proximaTareaB + 1;
            while (noEncontreNinguna == 1 && todosMuertos == 0) {
                if (jugadorB.piratas[i].vivoMuerto == 1) { //si esta vivo la pongo como la proxima tarea de
                    B proximaTareaB = jugadorB.piratas[i].index; noEncontreNinguna = 0;
                }
                if (i == proximaTareaA) {
                    todosMuertos = 1;
                }
                i++;
                if (i == 8) {
                    i = 0;
                }
            }
        }
    }
}

```

```

    }
    if (todosMuertos) {
        //salto a la idle
        return (13) << 3;
    }
    return (proxTarea << 3);
}
} else {
    if (turnoPirata == 0) {
        //turno proximo es A
        uint proxTarea = EMPIEZAN_TSS + proximaTareaA;
        game_tick(proxTarea);
        turnoPirata = 1;
        uchar noEncontreNinguna = 1;
        uchar todosMuertos = 0;
        int i = proximaTareaA + 1;
        while (noEncontreNinguna == 1 && todosMuertos == 0) {
            if (jugadorA.piratas[i].vivoMuerto == 1) {
                //si esta vivo la pongo como la proxima tarea de A
                proximaTareaA = jugadorA.piratas[i].index;
                noEncontreNinguna = 0;
            }
            if (i == proximaTareaA) {
                todosMuertos = 1;
            }
            i++;
            if (i == 8) {
                i = 0;
            }
        }
        if (todosMuertos) { //salto a la idle
            return (13) << 3;
        }
        return (proxTarea) << 3;
    } else {
        //turno proximo es B
        uint proxTarea = EMPIEZAN_TSS + 8 + proximaTareaB;
        game_tick(proxTarea);
        turnoPirata = 0;
        uchar noEncontreNinguna = 1;
        uchar todosMuertos = 0;
        int i = proximaTareaB + 1;
        while (noEncontreNinguna == 1 && todosMuertos == 0) {
            if (jugadorB.piratas[i].vivoMuerto == 1) { //si esta vivo la pongo como la proxima tarea de
                proximaTareaB = jugadorB.piratas[i].index;
                noEncontreNinguna = 0;
            }
            if (i == proximaTareaA) {
                todosMuertos = 1;
            }
            i++;
            if (i == 8) {
                i = 0;
            }
        }
        if (todosMuertos) { //salto a la idle
            return (13) << 3;
        }
    }
}

```

```
        }  
        return (proxTarea) << 3;  
    }  
}
```

Por último, otros métodos que definimos en nuestro *sched.c*

```
void sched_intercambiar_por_idle(){  
    estaEnIdle = 1;  
}  
void sched_nointercambiar_por_idle(){  
    estaEnIdle = 0;  
}  
void sched_toggle_debug(){  
    if (modoDebug) {  
        modoDebug = FALSE;  
    } else {  
        modoDebug = TRUE;  
    }  
}
```