Organización del Computador II

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

Trabajo Práctico III

System Programming - Zombie Defense

Grupo: Mini oso polar

Integrante	LU	Correo electrónico
Ansaldi, Nicolas	128/14	nansaldi611@gmail.com
Olveira, Paulo	464/12	paulo.olveira@gmail.com
Suárez, Romina Julieta	182/14	romi_de_munro@hotmail.com

Índice

1. Introducción

El objetivo de este trabajo practico fue la construccion de un sistema minimo para poder correr 16 tareas en paralelo, involucradas al juego "Zombie Defense"las cuales tienen como objetivo llegar al otro lado del mapa, generando puntos y, al mismo tiempo, prueban el sistema de desalojo para las tareas que irrumpan en las reglas previstas por el sistema.

Para tal trabajo, nuestra herramienta disponible fue el simulador de sitema Bochs. Las tareas a correr son los zombies, y las violaciones de sistema, provocan que ese zombie se deje de mover, y que se desaloje la tarea que esta corriendo.

Figura 1: Zombie mago, copyright x2

2. Desarrollo

2.1. Ejercicio 1 : GDT

2.1.1. GDT

Completamos los descriptores de la GDT utilizando las estructuras brindadas por la catedra, según el enunciado , de la siguiente manera :

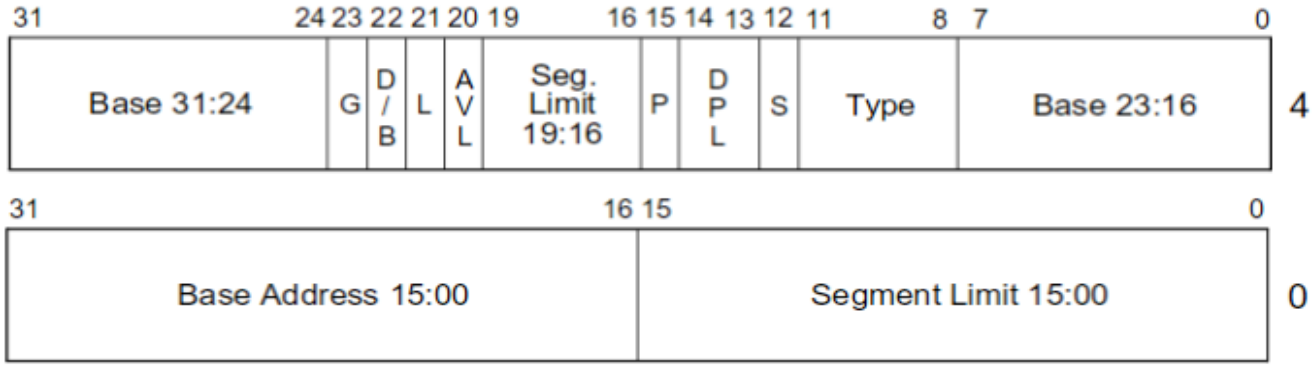
- El segmento nulo , por indicaciones del manual de intel
- Un segmento de código nivel 0
- Un segmento de datos nivel 0
- Un segmento de código nivel 3
- Un segmento de datos nivel 3
- El segmento de Video
- Las entradas restantes disponibles para los descriptores de TSS de las Tareas (la tarea incial, IDLE y 16 zombies)

Como los segmentos de memoria siguen un esquema de identity mapping , los segmentos estan completados de la siguenta manera (salvo por el tipo y el nivel de privilegio) :

- Los bits de S, P, d/b y G activos
- El DPL y el Tipo Acorde Al del Segmento
- La Base en 0x00000000
- El limite en 0x06FFFF
- Los bits de avl y l no activos

El descriptor del segmento de video se completa de la siguiente forma :

- Los bits de S, P, d/b activos
- El DPL de nivel 0 y el Tipo (0x02)
- La Base en 0x000b0000
- El limite en 0x06FFFF
- Los bits de g , avl y l no activos



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

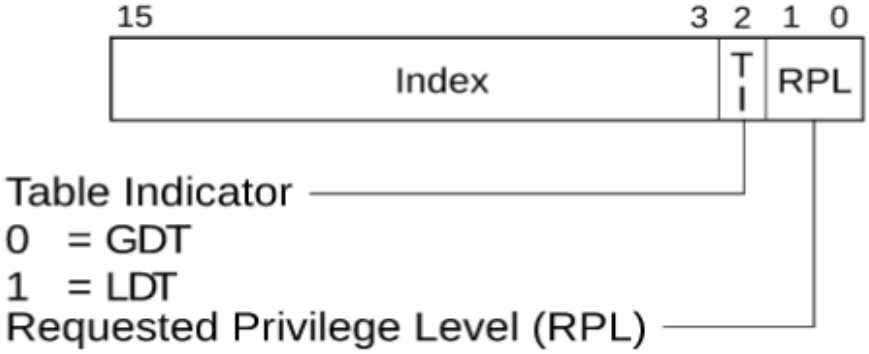


Figura 2: Imagen de Manual

Seteamos la pila y los registros de segmentos:

```
call habilitar_A20

    ; cargar la GDT
    lgdt [GDT_DESC]

    ; setear el bit PE del registro CR0
    mov eax , cr0
    or eax , 0x1
    mov cr0 , eax

    ; pasar a modo protegido
    jmp 8<<3:modo_protegido
    modo_protegido:
    BITS 32

    ; acomodar los segmentos
    xor eax , eax
    mov ax, 9<<3
    mov ds, ax
    mov es, ax
    mov gs, ax
    mov ss, ax
    mov ax, 12<<3
    mov fs, ax

    ; setear la pila
    mov ebp , 0x27000
    mov esp , 0x27000
```

Figura 3: Codigo para pasar a modo protegido

2.2. Ejercicio 2

2.2.1. IDT

Para completar la IDT , usamos las estructuras y la macro brindada por la catedra de manera que las entradas de las excepciones del procesador se completan de la siguiente forma:

- Son Interrupt Gates por lo que el Selector de Segmento es el de Datos de Nivel 0
- El Dpl para las interrupciones de procesador es de nivel 0
- Los Bit de P y D estan Activos
- El offset será la dirección en donde comienza el Handler de la interrupción

Para cargar la IDT usamos la instrucción lidt y cargamos la dirección del descriptor de la idt . Para las rutinas de excepcion también utilizamos una macro brindada por la catedra , para generar rutinas genericas necesarias para que impriman por pantalla la excepción . Para imprimir por pantalla que excepción es , utilizamos funciones auxiliares .

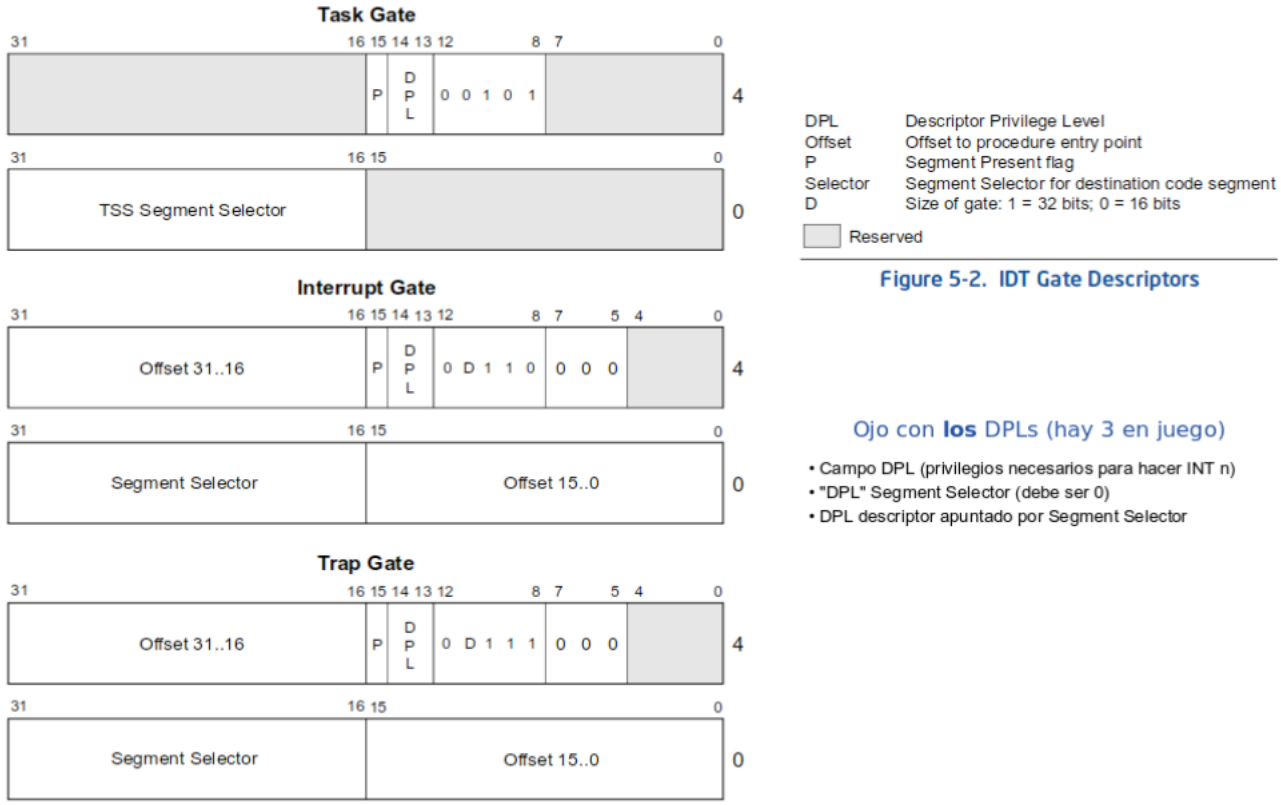


Figura 4: Imagen de Manual

2.3. Ejercicio 3

2.3.1. Pantalla

Antes de comenzar a organizar el mapeo de memoria con el que iba a funcionar el juego, primero hacía falta limpiar el *buffer* de video. El diseño que elegimos fue casi exacto al formato de las figuras en el enunciado.

Para ello, creamos funciones en el archivo *screen.c* cuyo proposito sería recorrer el buffer de 50 x 80 x 2 bytes, ya que cada caracter estaría compuesto por dos unidades, el primer byte donde estaría el caracter codificado en ASCII, y el segundo donde se dictaria el color a usar tanto para el caracter (los 4 bits menos significativos) como el color del fondo (los 4 mas significativos). Usando la logica de las funciones "print", y "print_hex", cuyo trabajo es recorrer el buffer con un puntero al struct ÇA ", al principio del mismo, modificando las coordenadas pasadas por parámetros, como si fueran indices en un arreglo de arreglos, entonces definimos funciones que modificarían de a pedazos porciones del buffer con un color definido, hasta tener el diseño completado.

Las zombies no se incluirian en el buffer hasta llegar a inicializar las tareas, asi que hasta el momento solo contariamos con un mapa vacio.

Estos cambios se guardarían en las posiciones de memoria correspondientes al Mapa. Al activar el modo Debug, y luego, eventualmente ocurrir una excepción, se copiara todo el buffer de vídeo, en un sector de memoria aparte, se modificara el sector actual con el estado de la tarea desalojada, y finalmente, cuando se presione y nuevamente, se copiara del sector guardado el vídeo del juego, y se seguirá corriendo con normalidad.

2.3.2. MMU : Directorio y tabla de Páginas

Iniciamos el Directorio de Páginas y las tablas necesarias en momento de ejecución mediante una función auxiliar , iniciamos en la dirección 0x27000 el Directorio de páginas con sus las entradas necesarias para direc-

cionar de 0x00000000 a 0x003FFFFF con identity mapping . Para esto utilizamos estructuras auxiliares para poder setear las paginas de manera mas ordenada , y casteamos un puntero a la direcciń 0x27000 como puntero a un arreglo de entradas de Directorio de páginas. Haremos algo similar con las tablas de página . El directorio de Paginas queda con las siguientes entradas:

- La primera entrada (con indice 0) con la dirección base de la tabla de página apuntando a la primera tabla de paginas en la dirección 0x28000 (0x28) con los bit de present y write activos y con nivel de privilegios de usuario (bit u/s en 0).
- El resto de las entradas del directorio completadas con 0 , de esta manera serán inválidas.

Las Tablas de páginas estan completas de la siguiente manera :

- La primera Tabla esta completa y direcciona desde 0x00000 hasta 0x3fffff y cada entrada tiene el bit de present y el de write activos y con nivel de privelegios de usuario.
- El resto de las entradas esta en 0 para que no esten presentes.

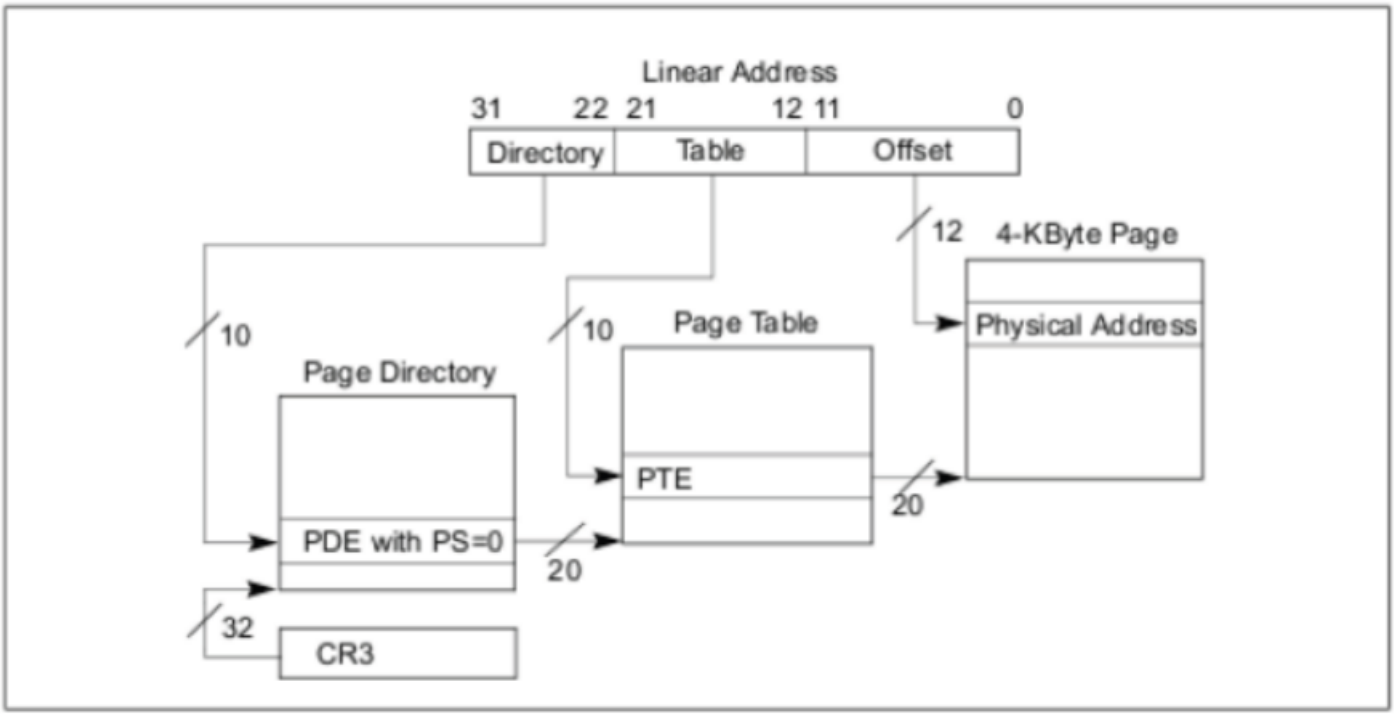


Figura 5

Para activar paginación hay que primero mover al registro cr3 la dirección base del directorio (0x27000) y activar el bit de paginación en el registro cr0 .

```
mov eax, cr0
or eax, 0x80000000
mov cr0, eax
```

Figura 6: Codigo para activar paginación

2.4. Ejercicio 4

2.4.1. MMU : Directorios de Tareas

En la Memory Management Unit definimos la función "mmu_inicializar_dir_zombie", con el proposito de ser llamada una vez inicializada o lanzada una tarea. Ésta en su lugar, devolvería la dirección de memoria donde se encontraría la página del directorio de paginación, el cual se cargaria en el registro cr3 en cuanto se realizase un cambio de tarea.

Para no pisar el directorio de ninguna otra tarea, se definio la variable CANT_PAGINAS, el cual es no más que un contador que guarda la cantidad de paginas de 4KB que ya fueron reservadas a partir de la dirección 0x100000 que es el comienzo del Area Libre, area destinada para este uso.

En cuanto se supiera las direcciones fisicas que usase la tarea a inicializar, se llamaría a la función "mapear_nueva_pagina", que usando el directorio de paginas que se le asigno, crearía las entradas necesarias a nivel usuario, para tener virtualizadas las direcciones que precisase. Como mencionamos antes, el codigo de la tarea se copía entonces a estas direcciones fisicas.

2.4.2. MMU : Mapeo y Desmapeo

Para completar el punto anterior, fueron imprescindibles los usos de las funciones "mapear_nueva_pagina". Como el nombre lo dice, esta se encargará de tomar una dirección del directorio de paginas, una dirección lineal y una dirección fisica, y en base a estos parametros, crear la entrada necesaria en el directorio para que quede virtualizada la pagina de 4KB a donde pertenece la dirección fisica, con la lineal. También se usa un parametro adicional, el *user -sup*, que es un entero que define el bit que corresponde a usuario o supervisor de la entrada a crear.

Entonces, para realizar esto, primero se toman los 10 bits mas significativos de la dirección lineal para obtener el indice donde buscar en el directorio de páginas. Si la entrada obtenida es nula, entonces la definimos. Se pide una "proxima_pagina"para usar como tabla de paginas, y la dirección de la misma (sus 20 bits mas significativos) se guardan en la entrada del directorio que estamos modificando. Tambien para la nueva entrada se le define lo siguiente:

- El bit de presente se fija en 0x1.
- El bit de lectura/escritura también se fija en 0x1.
- El bit de usuario/supervisor se llena con el parametro con el que se llamo a "mapear_nueva_pagina".

El resto de los bits, como se hizo antes, se les asigna un 0.

Si la entrada en el directorio de páginas ya estaba definida, entonces continuamos la busqueda repitiendo el proceso previo, pero usando la dirección de la tabla de paginas, ubicada en la entrada que creamos/encontramos en el paso anterior. Solo que en este caso usamos los bits 12:21 de la dirección lineal como indice en la tabla, y la dirección fisica para completar la nueva entrada, si hace falta crearla.

Para desmapear estas paginas en el caso de que hiciera falta, esta definida la funcion ünmapear_pagina", que en base a una dirección lineal y a la dirección de un directorio de páginas, se encarga de buscar la entrada que le corresponde, y fija todos sus valores a 0x0, en el caso de que la entrada este definida.

2.5. Ejercicio 5

2.5.1. IDT : Interrupcion de Reloj

Para poder atender la interrupción del reloj, primero hubo que definir un interrupt gate en la idt para el indice 32, ya que este sería la señal que el PIC levantaría al querer enviar una interrupción de reloj. Esta tendria un CS de nivel 0, con el cual correría entonces el codigo de la rutina de atención que definimos para resolver el caso.

Esta porción de codigo en ïsr.asm"se encarga de avisarle al PIC que atendió el pedido de interrupción. Luego llama a "proximo_reloj", que se encarga de mantener girando los relojes de todas las tareas vivas al igual que el reloj principal.

2.5.2. IDT : Interrupcion de Teclado

Con la interrupción del teclado, la logica fue la misma. Se crea una interrupt gate igual a la de reloj, pero con su offset apuntando a la rutina de la _isr33. Esta le alerta al PIC que ya atendió el pedido y prosigue con definir cual es el input que se ingresó por el buffer del teclado.

Dependiendo el caso, la rutina hace lo que la consigna solicita, siguiendo el esquema que utiliza el juego.

Si se presiona y se activa el modo debug, y a partir de entonces, toda excepción "pausaraël juego y mostrara por pantalla, los datos de la tarea que provoco la excepción, al momento de morir.

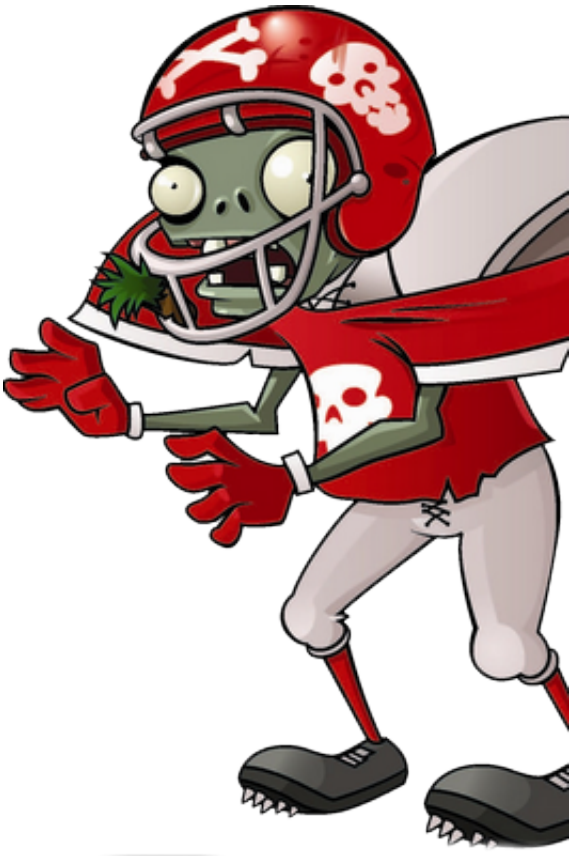


Figura 7: Lo que podria ser un zombie guerrero, copyright x3

2.6. Ejercicio 6

2.6.1. TSS

Para inicializar las TSS de las tareas utilizamos una funcion auxiliar *"tss_inicializar"* que inicializa los valores correspondientes de cada TSS . Esta función aprovecha la estructura tss y completa las tss declaradas previamente como variables globales, tarea_idle (las tss de la tarea idle) y los arreglos de tss de los zombies. La función completa la tss de la tarea idle de la siguiente manera :

- El cr3 es el mismo que el kernel , el esp y ebp apuntan al 0x27000
- El eip apunta a la dirección 0x16000 , donde la tarea esta compilada para ser ejecutada
- El esp y el ebp apuntan al 0x27000, dado que el ebp crece hacia el 0x27001 y el esp se mueve en direccion opuesta, al 0x26FFF
- Los registros de segmentos se inicializan con los mismo valores que los del kernel
- El resto de los valores de la tss se encuentran en 0

Las TSS de los zombies son completadas de la siguiente manera :

- El cr3 es la dirección a un directorio incializado con la funcion "mmu_inicializar_dir_zombie"que devuelve el puntero a un directorio inicializado con el mapeo necesario para la el funcionamiento de la tarea
- El esp de nivel 0 apunta a una página libre en el area libre y el ss0 contiene el selector correspondiente al segmento de datos de nivel 0
- El esp y ebp de nivel 3 comienzan en la diirección 0x8001000
- Los registros es ss ds fs gs son el selector correspondiente al segmento de datos de nivel 3 y el cs al de codigo de nivel 3
- El eip apunta al direrción 0x8000000

Todas las entradas inicializadas de las TSS tienen los eFlags en 0x202 y el iomap en 0xFFFF .

2.6.2. GDT : Tareas

Las entradas correspondientes a los descriptores de TSS , que habian sido previamente reservadas son inicializadas en el momento de ejecucion con una funcion auxiliar ïnicializar_gdt_tareas". Todas las entradas son completadas de la siguiente manera :

- El bit de s en 0 y el tipo de descriptor 0x09 (tss)
- El dpl de los descriptores son de nivel 0 para que solo el kernel pueda cambiar de tareas
- El bit de presente activo mientras que los bits de g d/b y l se encuentran en 0
- El limite es de 0x0067
- La base se completa de acuerdo al puntero de la tarea correspondiente

2.6.3. Tarea Inicial

Una vez que las TSS y las entradas de la GDT correspondientes estan inicializadas , se carga el selector de la tarea inicial en el registro ax y con la intstrución ltr se carga en el Task register . Luego se cambia de contexto a la tarea idle , mediante un jmp far .

2.7. Ejercicio 7

2.7.1. Scheduler: estructuras auxiliares

Para el funcionamiento del scheduler, se cuenta con una estructura auxiliar que guarda los selectores de la GDT correspondiente a los zombiess de las tareas. Esta estructura auxiliar es un simple arreglo de 8 structs que almacenan, ademas de los selectores, si esta activo o no para el scheduler, la posicion x e y del zombie, el tipo del zombie, que comprende 0 para el mago, 1 para el clerigo y 2 para el guerrero, la variable reloj, que sirve para printear el reloj de la tarea, y el codigo, que sirve para guardar la direccion del codigo de dicha tarea, del kernel .

2.7.2. Scheduler : *sched_proximo_indice*

El scheduler hace uso de una función que retorna el próximo indice, que devuelve el próximo selector de gdt de la tarea a ser ejecutada . Esto nos permite no escribir la logica del scheduler del trabajo en la rutina de atención de interrupciones en lenguaje ensamblador.

Primero verifica si debe buscar porjugador A o B, y luego, mediante un while, busca la proxima activa a correr, devolviendo 20 en caso de que no haya ninguna.

2.7.3. Scheduler: Interrupcion del Reloj

El scheduler utiliza la interrupción del reloj para hacer el cambio de tarea . Para esto se le modifico lo necesesario para lograr que el handler se comporte de la siguiente manera :

1. Se cambia de jugador por cada llamado a la interrupcion del reloj.
2. Se pide la siguiente tarea a correr, y mientras no sea la actual, se salta a ella.

2.7.4. Rutinas de excepciones

Sea el caso de que se genere una interrupcion, cualquiera de estas que sea de la 0 hasta la 20, la rutina de atención debe desalojar la tarea durante la cual se salto a dicha interrupción.

Para poder desalojar las tareas en las rutinas de excepciones del procesador, se utiliza una función auxiliar "print_tss", que llama a "sched_sacar", la cual elimina la tarea del scheduler. Luego print_tss imprime la excepción que mató a la tarea, pero solo la muestra si el modo debug esta activo .

2.7.5. Servicios del Sistema

El servicio de sistema se ejecuta a través de la interrupción 0x66, pasando previamente por el registro EAX el valor que corresponde a moverse en las 4 direcciones. Luego el Handler de la int 0x66 pushea este valor y llama a la función "game_mover_zombie"la cual se encarga de aumentar o decrementar la posición del zombie, y remapear según corresponda, finalmente copiando su código de donde estaba, antes de moverse. Y chequeando si hizo un punto a favor, o en contra.

Figura 8: Copyright x4, el famoso juego aun no ha implementado un este tipo de zombie, así que esto es lo que más se le acerca.

2.8. Ejercicio 8 (optativo)

2.8.1. Tarea Adicional: