DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP3

Organización del computador II

Pencylvester

Integrante	LU	Correo electrónico
Alonso Tomás	396/16	tomasalonso96@gmail.com
Gaggero Damián	318/16	damiangaggero@gmail.com
Grosso Alejandro	016/16	alejandrogrosso@opmbx.org

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Índice

1. Introducción

2. Desarrollo

2.1. Ejercicio 1

A lo largo del TP se usan constantes (*#defines*), como por ejemplo en este ejercicio, donde usamos el índice de cada entrada de la **GDT**, declarados en el archivo *defines.h*.

Para completar cada entrada se usa la estructura dada por la cátedra sin ninguna modificación.

Se definió un segmento de datos especial para la memoria de video (*GDT_VIDEO*), el cual se usó solamente en este inciso y se comentó posteriormente, por ende no se agrega a la GDT.

Luego, se carga la GDT y se salta a modo protegido tal como se vió en las clases y los ejemplos.

Se cargan todos los segmentos con la modalidad **flat**, todos los segmentos apuntando a la misma memoria, y se apunta la pila del kernel a la dirección 0×27000, especificada por el enunciado.

Ahora se carga el segmento especial de video y con código assembler se recorre por medio de 2 ciclos anidados todas las posiciones del segmento que comienzan en la posición $0 \times B8000$, es decir, se recorre cada posición de la pantalla, pintándola de gris.

De aquí en más se deshabilitó este segmento especial de video, se accede a la memoria de video por los segmentos de datos y la inicializacion de la pantalla se realiza en el *ejercicio 3* con funciones en C.

2.2. Ejercicio 2

Para inicializar las entradas de la **IDT** se utilizo la macro *IDT_ENTRY* propuesta con una ligera modificacion, se decidió agregar a la misma un parámetro adicional, que permita especificar el atributo de la entrada, definidos previamente. Estas son **TRAP**, la cual indica una trap gate usada para las excepciones, **INTERRUPT**, usada para interrupciones que solo pueden ser llamadas por código de privilegio 0, y **USER_INTERRUPT**, que puede ser llamada por código de usuario también.

```
1  #define IDT_ENTRY(numero, attribute)
2    idt[numero].offset_0_15 = (unsigned short)
3                  ((unsigned int)(&_isr ## numero) & (unsigned int) 0xFFFF);
4    idt[numero].segsel = (unsigned short) (GDT_IDX_ROOT_CODE << 3);
5    idt[numero].attr = (unsigned short) attribute;
6    idt[numero].offset_16_31 = (unsigned short) (
7                  (unsigned int)(&_isr ## numero) >> 16 & (unsigned int) 0xFFFF);
8
9  #define TRAP      0b1000111100000000
10 #define INTERRUPT 0b1000111000000000
11 #define USER_INTERRUPT 0b1110111000000000
```

Para la creación de las funciones de las excepciones de procesador se utiliza otra macro, *ISR*, esta macro fue adaptada en este ejercicio para que mostrara un mensaje de error de forma rudimentaria.

```
1  %macro ISR 2
2  global _isr%1
3    msg%1 db %2, 0
4    msg%1_len equ    $ - msg%1
5  _isr%1:
6    mov eax, %1
7    imprimir_texto_mp msg%1, msg%1_len, 0x7, 0, 0
8    iret
9  %endmacro
```

Cuando se produce una excepción, el descriptor llama a su función asignada (*_isrX*), la cual, dependiendo del número de interrupción se elige el mensaje que se imprime por pantalla, esta funcionalidad

será aprovechada más adelante ya que son similares en cuanto a código todas las excepciones en este tp.

El funcionamiento de la versión final de la macro se detalla mejor en la parte correspondiente del ejercicio 7, cuando se implementa el verdadero funcionamiento de las rutinas de atención de estas excepciones.

Una vez inicializada la IDT en el kernel con la macro, como se indica en el enunciado, se probó a continuación el funcionamiento de varias excepciones. Esto funcionó correctamente y luego fue comentado para seguir con el trabajo practico.

2.3. Ejercicio 3

En primer lugar se inicializa la pantalla con la función *screen_inicializar* hecha en C.

Esta función utiliza todas funciones de la catedra, entre ellas *screen_pintar_rect*, *screen_pintar_linea_-h*, *screen_pintar_puntajes*.

Los pasos a realizar por la función son, en primer lugar, pintar toda la pantalla de gris; luego se pinta una línea negra en la parte superior, donde se escribirá informacion, por el momento se escribe aquí el nombre del grupo, pero luego se escribirá la tecla pulsada y el modo *debug*. También se pintan los dos sectores de puntajes de los jugadores y se inicializan los relojes para cada pirata de cada jugador. Todo lo realizado es acorde a las imágenes del mapa sugerido por la cátedra.

Luego se implementó *mmu_inicializar_dir_kernel* que crea un *page directory* para el kernel a partir de la dirección 0×27000 como indica el enunciado. En este page directory se crea una primera *page table* que se ubica en la posición 0×28000, posición asignada directamente y dicho sea de paso, en este punto, al no estar paginación aún habilitada, pedir memoria libre no es una buena opción.

La *page table* se cicla con un ciclo y un incremento de 0 a 1024, asignando páginas sucesivas, de modo que el esquema de paginación sea **identity mapping**; y para que ocurra efectivamente, se asigna la page table en la primera entrada del page directory. De modo que queda mapeada toda la sección del kernel y su memoria libre. Como última aclaración, la sección del kernel corresponde al uso de la page table en su totalidad.

Finalmente se carga en el **cr3** del kernel la dirección del page directory y se habilita en el **cr0** el bit de paginación.

2.4. Ejercicio 4

Se iniciliza el seguimiento de las paginas libres con una variable global la cual comienza indicando el inicio del sector de memoria libre, con esta variable se indica donde se crea la proxima pagina pedida y luego se mueve una posicion hacia adelante.

Se implementaron las funciones mmu_mapear_pagina y mmu_unmapear_pagina, la primera accede al page directory indicado por cr3, chequea si la page table correspondiente a la direccion virtual que se pide mapear esta presente o no, si no lo esta, se crea, si lo esta, se apunta a la page table indicada. luego en esta tabla setea en la entrada correspondiente a la direccion virtual, los valores necesarios para mapear a la direccion fisica y se pone en presente esta pagina. Finalmente se limpia el cache de paginas accedidas.

La segunda, simplente setea a cero el bit de presente de la entrada correspondiente en la page table que estaba mapeando la direccion virtual indicada.

En la implementacion de la funcion mmu_inicializar_dir_pirata sucede lo siguiente:

Se crea un page directory pidiendo una pagina libre, se setea todo en cero para esta estructura, en la primera tabla se apunta a la tabla que mapea el kernel, la direccion 0x28000.

Luego se obtiene la direccion fisica en donde comienza el pirata, el puerto del jugador correspondiente y tambien se localiza con un puntero la posicion en memoria del codigo correspondiente al tipo de pirata. con el cr3 actual se mapea en la direccion 0x401000 el lugar donde va a ir localizado el codigo,

luego se copia efectivamente el codigo a esta direccion recien mapeada, se ponene manualmente en la pila los parametros necesarios por el codigo y una direccion de retorno 0 la cual no va a ser llamada nunca y finalmente se desmapea esta direccion 0x401000.

Se termina por mapear, en la direccion 0x40000 en el page directory creado al comienzo de la funcion, con la posicion en memoria a donde se copió el codigo de la tarea en los pasos anteriores.

En las 4 tablas de paginas siguientes se mapean 4 tablas las cuales se crean cuando se inicia un jugador. Estas corresponden a indicar las direcciones del mapa que ya son conocidas por todos los piratas del jugador y se ven actualizadas con cada movimiento de los piratas exploradores. se hablara de como funcionan especificamente esta estructura de cuatro paginas al final del informe cuando se detalle el funcionamiento general del juego. (HABLAR DE TABLAS DE JUGADOR QUE MAPEAN LO DESCUBIERTO) Por ultimo se da por explorada la posicion en la que se crea el pirata en esta estructura de posiciones exploradas, en caso de que sea la primera vez, luego se devuelve el page directory del pirata.

2.5. Ejercicio 5

Se crean las 3 entradas en la IDT para las 3 interrupciones indicadas. En la interrupcion del reloj se deshabilitan las interrupciones de PIC y se realizaba lo pedido de igual manera que lo que recomienda el enunciado, se hablara en mayor profundida de esta y las otras interrupciones cuando se trate su implementacion final (HABLAR DE LAS 3 INTERRUPCIONES AL FINAL) En la interrupcion del teclado tambien se deshabilitan las interrupciones del PIC y luego se llama a la funcion game_atender_teclado, la cual en primer lugar solo se encargaba de imprimir en pantalla la tecla usando un switch y los distintos codigos de teclado, tambien se comentara en la funcion final de esto mas adelante.

Finalmente se agrega la interrupcion 0x46, la cual no hacia nada util y se explica su funcionamiento final más adelante cuando es correctamente implementada.

2.6. Ejercicio 6

Se agrega en la GDT un descriptor de TSS para la tarea incial, uno para la tarea idle, 8 para los piratas de jugador A y otras 8 para los piratas del jugador B.

En la funcion tss_inicializar se configura la TSS de la tarea idle como se pide, creando esta como una variable local en el codigo y modificando esta estructura con los valores correspondientes, la pila en la del kernel, el cr3 del kernel los eflags con valor 0x202 y el eip apuntando a la posicion 0x16000.

Finalmente en un ciclo se inicializan las bases y el bit de presente de todas las TSS de los piratas, que son inicializadas completamente luego, cuando se lanza un pirata con los valores necesarios de un pirata, pidiendo una pagina para la pila de nivel 0, seteando su pila al final de su pagina pero asumiendo que tiene los dos paramentros y la direccione de retorno en la pila y con el eip apuntando al principio de la pagina.

Se escribe el codigo necesario para cargar la tarea inicial como la actual e inmediatamente saltar a la tarea idle.

(ESTO ACA O EN EL EJERCICIO 7?) Modificamos la interrupcion 0x46 la cual ahora pone en la pila los parametros que se le pasan y llama a game_syscall_manejar. Esta le pide al scheduler cual es el jugador actual y el pirata actual, con una serie de ifs chequea cual fue el servicio pedido y llama a game_syscall_pirata_mover, game_syscall_cavar o game_syscall_pirata_posicion. si no es nungun caso destruye al pirata y devuelve un -1.

Cuando retorna a la interrupcion, esta acomoda la pila, recupera el resultado y chequea si se devolvio un -1, en este caso salta a la tarea idle, si no, se fija si se intento pedir la posicion, si fue asi, manualmente se pone en la pila en el registro guardado de la tarea la posicion actual de la misma y tambien se salta a la tarea idle.

2.7. Ejercicio 7

La estructura utilizada para el scheduler es la siguiente: Se tiene un uint para saber si el scheduler esta activo o no, una lista de uints con 17 posiciones las cuales representan los indices de la GDT que llevan al indice de la TSS de la respectiva tarea, las 8 primeras de jugador A, las segundas del jugador B y la ultima para la tarea idle. Hay un uint que indica el jugador actual (0 para A, 1 para B).

Una tupla de uints que indica en la primera posicion el indice del ultimo pirata ejecutado del Jugador A y en la otra el del ultimo pirata ejecutado del jugador B en las dos listas siguientes.

En estas dos listas se indica con un int si el pirata correspondiente al indice esta libre (0) o en ejecucion (1);

3. Resultados

4. Conclusión