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

Trabajo Práctico Nro 3
System Pro1gramming - Batalla Bytal

Organización del Computador II

Segundo Cuatrimestre de 2013

Grupo: **Frambuesa a la Crema**

Apellido y Nombre	LU	E-mail
Ignacio, Truffat	387/10	el_truffa@hotmail.com
Lasso, Nicolás	763/10	lasso.nico@gmail.com
Rodríguez, Agustín	120/10	agustinrodriguez90@hotmail.com

Índice

1. Introducción

En el siguiente informe se describe el código del Trabajo Práctico Nro 3 entregado. Para la realización de este informe, se separó en ejercicios, describiendo los temas de la materia incluídos en el TP: configuración de la GDT, pasaje a modo protegido, configuración de la IDT, paginación, TSS y la organización del scheduller.

2. Desarrollo y Resultados

2.1. Ejercicio 1. GDT

2.1.1. Global Descriptor Table

Como ya sabemos, el procesador comienza en lo que se llama "modo real", que direcciona a 1 MB de memoria y no existen niveles de protección ni privilegios.
Por eso necesitamos que el procesador pase a "modo protegido", para direccionar a más memoria y manejar niveles de protección. El kernel se encargará de hacer esto.
Antes de iniciar en modo protegido, es imprescindible tener bien configurado la Tabla de Descriptores Globales, la cual es una tabla que contiene descriptores de segmento, con la finalidad de definir características de varias áreas de la memoria.
En el enunciado se piden 4 segmentos que deben direccionar a 1.75 GB: 2 para código de nivel 0 y 3 respectivamente, y 2 para datos, de nivel 0 y 3 también.
La estructura de un descriptor de segmento es la siguiente:

- 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

Para definir los segmentos que nos requieren, los items importantes son:
BASE. Ubicación del byte 0 del segmento en el espacio de direcciones lineales. El valor en los 4 segmentos es 0x00000000.
P. Present. Indica si el segmento está presente en la memoria. El valor en los 4 segmentos es 0x01.
DPL. Descriptor de privilegios. Según el enunciado, 2 segmentos llevan el valor 0x00, y los otros 2 0x03.
G. Granularity. Debe estar en 0x01 para que las unidades de base y limit se interpreten de 4-KBytes.
Limit. Tamaño del segmento. Va para los 4 segmentos lo mismo:
Tenemos que direccionar a 1.75 GB, que son 1792 MB, que equivalen a 1835008 KB.
Como G vale 0x01, las unidades deben representarse de 4 KB, por eso dividimos por 4.
$\frac{1835008}{4} = 458752$
Pero como la memoria empieza desde el 0, debe ser un número menos: 458751
$458751 = 0x6FFFF$ Type. Indica si es un segmento de código o de datos. Para los 2 de código ponemos el valor de 0x08, indicando que son "Execute only". Mientras que para los 2 de datos ponemos el valor de 0x02, indicando que son de Read/Write.

También debemos colocar un segmento que describa el área de la pantalla en la memoria. Sabemos que empieza en la dirección 0x000B8000, con un tamaño de 0x0F9F.

Una vez que tenemos configurada la gdt, guardamos su ubicacion en una variable gdt_desc. Para que luego la instrucción lgdt pueda cargar la gdt.
Ya podemos pasar a modo protegido, poniendo en 1 el bit menos significativo del registro CR0, que indica "Protected Envirnoment".

2.2. Ejercicio 2 y 5. IDT

2.2.1. Interrupt Descriptor Table

A través de la IDT, definimos dónde está el código de las interrupciones que manejaremos.
La estructura de una entrada en la IDT está definida en idt.h y en el idt.c cargamos todas las entradas. Por medio de una macro cargamos las primeras 20 interrupciones del procesador, que van desde la división por 0 hasta la interrupción SIMD.
Luego llenamos el resto de la tabla con entradas de interrupciones inválidas, siendo un total de 256 entradas. Algunas de estas se definirán de nuevo como entradas de interrupciones de reloj, teclado, servicios y banderas.
En isr.asm se encuentra el código donde atendemos estas interrupciones. Por ahora de la 0 a la 19 sólo se imprime el código de error en pantalla.

La estructura de una entrada de la idt, definida en idt.h, es la siguiente:
offset_0_15: primeros 16 bits del offset al entry point, que atenderá la interrupción
segsel: selector de segmento de codigo de la gdt
attr: atributos de la entrada: Present, DPL, D
offset_16_31: segundos 16 bits del offset al entry point.

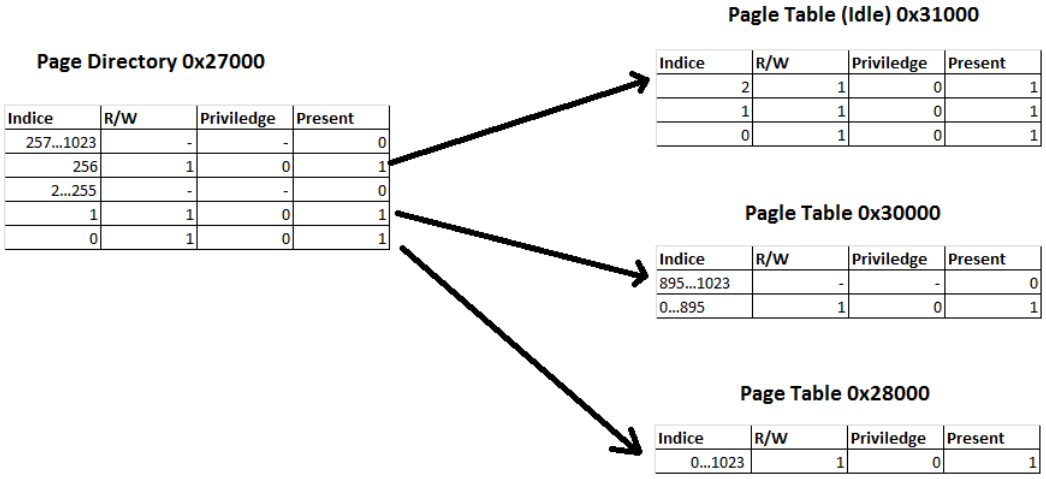
Todas las interrupciones, excepto las de servicios y banderas, tendrán en attr el siguiente valor:
0x8E00 = 1000 1110 0000 0000
Más detalladamente de izquierda a derecha:
1: Present
00: DPL = 0
0D110 000: D = 1 (Size of gate 32 bits), los demas bits caracterizan el tipo de interrupción que es 'Interrupt'.

Las interrupciones de servicios y de banderas, en attr tienen el valor 0xEE00, diferenciándose en el DPL que tiene en este caso el valor 3, ya que estas interrupciones serán llamadas por las tareas.

Indice	Descripcion	P	DPL	D
0...19	Ins del procesador	1	0	1
32	Clock	1	0	1
33	Teclado	1	0	1
80	Servicios	1	3	1
102	Banderas	1	3	1

2.3. Ejercicio 3. Paginación

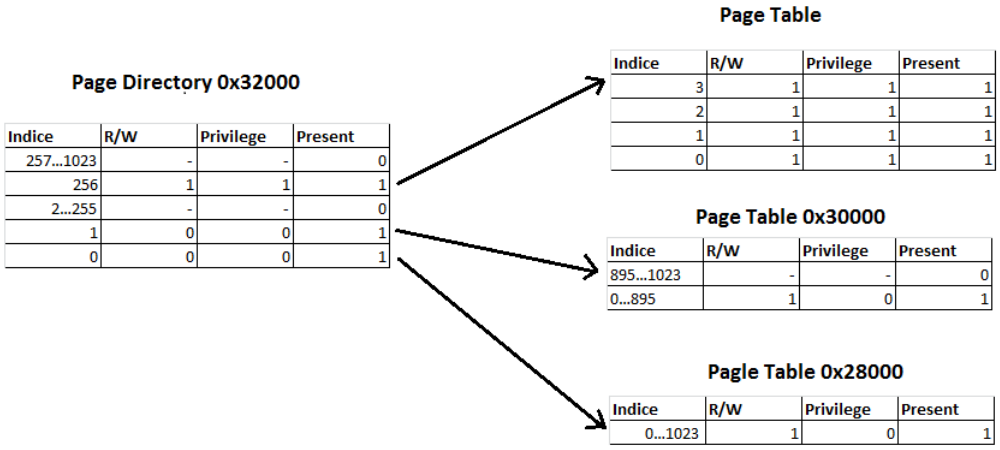
Debemos generar un directorio de páginas que mapee con identity maping para el kernel un total de 1920 páginas de 4 KB cada una, haciendo un total de 7,5 MB. Para eso cargamos un Page Directory en la dirección 0x27000, que contendrá 3 entradas que apuntan a 3 Directory Tables. Los primeros 2 Pages Tables son para las 1920 entradas que tenemos que mapear: 1024 para un Page Table, y 896 para el segundo. Definimos un tercer Page Table para la tarea Idle. El mapeo del directorio de páginas será de esta forma:



Luego de armar el directorio de páginas podemos habilitar la paginación. Para esto debemos prender el bit mas significativo del registro CRO:
mov eax, [TASK_PAG_DIR]
mov cr3, eax
mov eax, cr0
or eax, 0x80000000
mov cr0, eax

2.4. Ejercicio 4. Paginación de tareas

Para la paginación de tareas se inicializa un directorio de páginas con 3 entradas por tarea. Las primeras 2 mapean al kernel, y la tercera es para la tarea, la cual tiene 4 entradas de páginas de tabla: 2 para código que apuntan al mar, una para la pila de nivel 0 y otra para el ancla.



2.5. Ejercicio 6. TSS

Para que el procesador pueda despachar, ejecutar o suspender una tarea, es necesario salvar el estado de la misma. La arquitectura provee mecanismos para esto. El segmento de estado (TSS, Task State Segment), es el que se encarga de almacenar la información del estado de una tarea.
Una tarea está identificada por el selector de segmento de su TSS. Y a su vez la TSS es un segmento, por lo tanto debe estar descripto en la GDT junto con los descriptores de segmento de código y datos.
Tenemos 8 tareas y definimos un total de 18 TSS, uno para cada tarea, uno para cada bandera de tarea, uno para la tarea Idle, y otro lo dejamos en blanco para la tarea inicial donde se hace el primer salto.
Las entradas de tss idle y la que está en blanco tienen privilegio de kernel, mientras que las demás están configuradas con privilegios de usuario.
La tarea y la de su flag tienen TSS muy similares, exceptuando por cosas como el eip o donde empiezan las pilas, pero en el resto son iguales ya que comparten casi todo.

2.6. Ejercicio 7. Scheduller