DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

System Programming

Organización del Computador II
Segundo Cuatrimestre de 2016

Grupo: El Arquitecto

Integrante	LU	Correo electrónico
Freidin, Gregorio	433/15	gregoriofreidin@gmail.com
Taboh, Sebastián	185/13	sebi_282@hotmail.com
Romero, Lucía Inés	272/15	luciainesromero@hotmail.com

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. Ejercicio 1

El primer ejercicio consistió en dos elementos, el primero fue sobre llenar la Global Descriptor Table (GDT), con ciertos segmentos y el segundo, en pasar a modo protegido.

Inicializar la GDT

En esta parte lo que hicimos primero fue crear un arreglo de gdt_entry, en el archivo gdt.c de 41 entradas y luego completamos 6 de las mismas. En la primera posición del arreglo fue seteado el descriptor Nulo por restricciones de Intel. Y los otros 5 descriptores de segmento fueron completados empezando desde la posición 18, por restricciones del TP, hasta la 22 inclusive.

Los dos descriptores de segmento de las posiciones 18 y 19 fueron seteados como segmentos de código nivel 0 y 3 respectivamente. Empezando la base de ambos desde la posición 0x0000, y con un límite de 0x6ffff con el bit de Granularity activado respresentando así 1.75 GB.

1.75 GB son $\frac{7}{4} * 2^{30}B = 7 * 2^{28}B$. Como hay 20 bits del campo "Límite" no iban a ser suficientes para direccionar 1.75 GB como se buscaba, entonces por eso se puso un 1 en el bit de Granularidad para que el Límite se refiriera a bloques de 4 KB. Así, se obtiene $\frac{7 * 2^{28}B}{4KB} = 7 * 2^{16}$ y restándole 1 se consigue 0x6FFFF, el contenido que se puso en el campo "Límite".

Al type de ambos segmentos se les puso el valor de 0xA (Exectute/Read), y los atributos de Sistema y Presencia en 1.

Los dos descriptores de segmento de las posiciones 20 y 21 fueron seteados como segmentos de datos nivel 0 y 3 respectivamente. Empezando la base de ambos desde la posición 0x0000, y con un límite de 0x6ffff con el bit de Granularity activado. Al type de ambos segmentos se les puso el valor de 0x02 (Read/Write), y los atributos de Sistema y Presencia en 1.

Por último el descriptor seteado en la posición 22 del arreglo, fue colocado como un descriptor de video, con la base a partir de 0xb8000, y límite 0x0f9f, con el bit de Granularity en 0. A este segmento también se le asignó en los atributos de Sistema y Presencia el valor 1.

Pasar a Modo Protegido

Luego de cargar la GDT, creándola con los requisitos ya especificados y cargando el GDT register con **lgdt**, habilitamos la línea A20 para poder acceder a las posiciones mayores a 2^{20} y seteamos el bit de PE de CR0 en 1.

Así ya con todo el contexto armado, ejectuamos la instrucción **jmp** 18 << 3:mp para hacer un jmp far a modo protegido donde ponemos dentro del selector de segmento de código el valor 18 que es la posición dentro de nuestro arreglo gdt, donde está el desciptor de código nivel 0.

2. Ejercicio 2

Inicialización de la IDT

En este ejercicio el tema que se va a tratar es el de inicializar la IDT. Para ello, primero se declaró la estructura de la IDT en idt.c, como un arreglo de 256 entradas de idt_entry.

En este punto lo que hicimos fue simplemente llenar las primeras 31 entradas de este arreglo, con las interripuciones definidas por Intel. Para ello se utilizó una macro, ya definida, en la cual asignamos a todas las entradas por igual, como segmento de código en el cual se van a ejecutar, el numero 18 dentro de la GDT, por ser este de nivel 0. Y en los atributos, el valor de 0x8E00, lo cual especifica que cada excepción tiene el bit de presencia encendido, nivel de privilegio 0, y que es una interrupción de 32 bits. Y luego a cada entrada dentro del campo offset, le escribimos la dirección del handler de la misma.

Los handler de las interrupciones fueron declarados en isr.h e implementados en isr.asm. Donde a cada uno, simplemente se le pasó la tarea de imprimir por pantalla el número de interrpución que representa. Para ello en el área de memoria de la pantalla imipimimos el texto que según Intel representa dicha interrupción.