DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Tierra Pirata

Trabajo Práctico III - Programación de sistemas

29 de octubre de 2017 — Organización del Computador II

Grupo: Ariane 5

Integrante	LU	Correo electrónico
Greco, Luis	150/15	luifergreco@gmail.com
Hertzulis, Nicolás	811/15	nicohertzulis@gmail.com
Ramos, Ricardo	841/11	riki_german@yahoo.com.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Introducción

En este informe describimos la implementación de los fragmentos de código que completan el kernel provisto por la cátedra.

Un kernel se encarga de administrar los recursos del CPU tal que permite ejecutar múltiples tareas en simultáneo, garantizando que cada tarea disponga de la máxima cantidad de recursos posible.

El conjunto de registros y la arquitectura del CPU permiten definir niveles de privilegio de manera de ocultar el código del kernel de programas de usuario.

2. Desarrollo

2.1. Global Descriptor Table

2.1.1. Descriptores de código y datos

Antes de pasar a modo protegido creamos cuatro descriptores de segmento en la GDT. Como nos piden que dejemos lugar para 7 descriptores al comienzo de la tabla empezamos en índice 8, dejando libres índices desde 0 a 7. Los descriptores están dados por dos doble words (ver Figura 1), una doble word ocupa 32 bits, y se representan con estructuras llamadas `str_gdt_entry`, donde cada campo de la estructura representa subconjunto de bits de cada una de las dos doble words. En lo que sigue nos referiremos a los bits de las doble words.

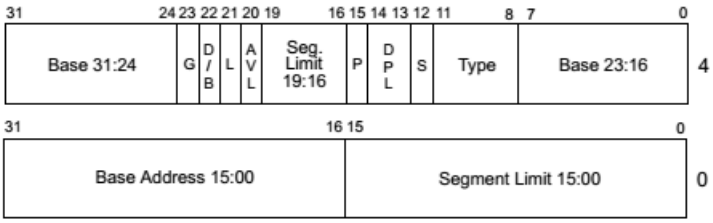

Figura 1: Descriptor de segmento (primera doble word abajo y segunda doble word arriba).

Los descriptores de código, en índice 8 y 10 de tabla, se completan tal que los bits de $BASE$, que son los bits 16 a 31 de primer doble word y bits 0 a 7 más bits 24 a 31 de segunda doble, queden todos en 0. Esto a causa de que debemos direccionar los primeros 500 megabytes de memoria. Luego el límite debe indicar 500 megas y por lo tanto debemos setear el bit G, granularidad, que es el bit número 23 de segunda doble word. A causa de esto el límite tiene rango desde 4 Kbytes a 4 Gbytes. Entonces despejando de ecuación $Limite$ * `0x400` + `0x3FF` = 500 $megas$ se obtiene valor de `0x7CFFF`. Escribimos en bits 0 a 15 de primer doble word, campo de límite, el valor `0xCFFF` y completamos el campo de límite escribiendo en los bits 16 a 19 de segunda doble word el valor `0x7`. Luego seteamos los bits de tipo, bits 8 a 11 en segunda doble word, de manera que indique que el descriptor es de código y se permite ejecución y lectura. Esto se hace llenando esos bits con `0xA`. Paso seguido seteamos bit S, bit 12 en segunda doble word, especificando que desciptor no es de sistema. Pasamos a bits 13 a 14 en segunda doble word, DPL, en donde establecemos privilegios. Acá llenamos con dos ceros en caso de descriptor de nivel 0 mientras que para descriptor de nivel 3 llenamos los bits con dos unos. Luego Seteamos bit P, número 15 en segunda doble word, tal que indique que el descriptor está presente. Bits AVL y L, bits 20 y 21 respectivamente de segunda doble word, se dejan en cero indicando no segmento de 64 bits. Finalmente bit D/B, bit 22 en segunda doble word, se setea indicando segmento de 32 bits.

En los descriptores de datos todos los bits menos los de tipo se completan exactamente como

completamos los de código, llenando los DPL con ceros para segmento de datos nivel 0 y con unos para el de nivel 3. Los bits de tipo se llenan con `0x2` tal que indican segmento de datos para lectura y escritura. Estos descriptores están en los índices 9 y 11 de tabla.

Para descriptor de segmento de pantalla debemos especificar como base `0xB8000` y para el límite necesitamos resolver (80 * 50 * 2) - 1, donde 80 es la cantidad de celdas, 1 celda == 2 bytes, en fila de pantalla y 50 es cantidad de celdas en columna. Es decir que el límite nos la cantidad de bytes que necesitamos para escribir en cualquier parte de la pantalla. Luego el valor del límite es `0x1F3F`. Este descriptor va en índice 12.
Comenzamos llenando los primeros bits de $BASE$, bits 16 a 31 de primer doble word, con `0x8000` (parte menos significativa de base) y continuamos llenando bits 0 a 7 de segunda doble word con `0x0B` y bits 24 a 31 de misma doble word en `0x00` (parte más significativa de base). Paso seguido llenamos bits 0 a 16 de primera doble word con `0x1F3F`, límite de segmento, y bits 16 a 19 de segunda doble word en 0. Luego llenamos bits de tipo con `0x2` indicando segmento de datos permitiendo lectura y escritura. Seguimos con bit S puesto en cero indicando que el segmento no es de sistema y bits de DPL en cero para indicar privilegio 0. El bit P es seteado para especificar segmento como presente y bits AVL y L en cero, el primero no es usado y el segundo indica no segmento de 64 bits. Finalizamos seteando bit D/B, para indicar operaciones en 32 bits, y bit G en 0 porque el límite es menor a 1 Mbyte.

2.1.2. Pasaje a modo protegido y pila de kernel

En `kernel.asm` cargamos el registro $GDTR$ con dirección de tabla, que es arreglo de `str_gdt_entry`, usando instrucción `lgdt`. Luego llamamos a función `habilitar_A20` y seteamos el bit PE del registro $CR0$. Esto se hace moviendo $CR0$ a registro general de 32 bits, entonces aplicamos `or` bit a bit entre el registro y valor 1 y movemos resultado a $CR0$. Siguiente instrucción es un salto a `0x40:modoprotegido` donde `0x40` es valor de selector de segmento tal que en bits 3 a 15 de selector (ver Figura 2) se escribe 8 en binario indicando índice 8 en la tabla. Por otra parte valores de bits 0 y 1, ambos en cero, de selector indican segmento con privilegio 0 y bit 2, también en cero, especifica en que tabla buscar, en este caso tabla global. Es decir buscamos segmento de código en la GDT y allí seguimos en las instrucciones bajo etiqueta `modoprotegido`.

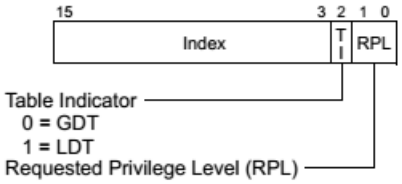

Figura 2: Selector de segmento.

El código siguiente está en 32 bits, incluyendo lo que está bajo etiqueta `modoprotegido`. Allí cargamos los selectores de segmento ds, es, ss, gs con `0x48` tal que bits 3 a 15 de `0x48` indican que cada selector indexe en descriptor número 9 de la GDT (bit 2 en cero) y con privilegio 0 (bits 0 y 1 en cero). Paso seguido cargamos selector de segmento fs con `0x60` tal que se escribe 12 en binario en bits 3 a 16 de selector, indicando índice 12, y los bits restantes especifican buscar en GDT con privilegio 0, es decir donde está el segmento de pantalla de kernel. Luego establecemos la base de la pila del kernel en dirección `0x27000`, cargando registro esp con esta dirección.