



**DEPARTAMENTO
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

System Programming

Organización del Computador II
Segundo Cuatrimestre de 2020

Integrante	LU	Correo electrónico
Ivo Pajor	460/19	ivo_pajor@hotmail.com
Laureano Muñiz	498/19	lau2000m@hotmail.com
Luciana Gorosito	577/18	lugorosito0@gmail.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. Introducción	3
2. Desarrollo	3
2.1. Ejercicio 1	3
2.2. Ejercicio 2	4
2.3. Ejercicio 3	4
2.4. Ejercicio 4	4
2.5. Ejercicio 5	4
2.6. Ejercicio 6	4
2.7. Ejercicio 7	4
2.8. Ejercicio 8	4
2.9. Ejercicio 9	4

1. Introducción

El objetivo de este trabajo práctico es aplicar gradualmente los conceptos de *System Programming* vistos en las clases teóricas y prácticas, mediante la implementación de una serie de ejercicios que, inspirados en la serie *Rick y Morty*, en conjunto conformarán un kernel o un pequeño sistema operativo.

2. Desarrollo

A continuación detallamos las implementaciones de los ejercicios del trabajo práctico.

2.1. Ejercicio 1

Para la realización de este ejercicio analizamos las estructuras `gdt_entry_t` y `gdt_descriptor_t` definidas por la cátedra. En esta implementación, la Tabla de Descriptores Globales (GDT) es un arreglo de `gdt_entry_t` y su descriptor, que luego cargaremos en GDTR, es del tipo `gdt_descriptor_t`.

Siguiendo lo indicado en el primer ítem, definimos a partir del índice 10, 4 descriptores de segmento en la GDT utilizando la estructura `gdt_entry_t`, atendiendo a las propiedades particulares de cada segmento. Así definimos las entradas de dos segmentos de código de nivel 0 y 3 y de dos segmentos de datos, también de nivel 0 y 3. Puesto que estos segmentos deben direccionar los primeros 201 MB de memoria establecimos en todos el bit de G en 1, y definimos su base en 0x00000000 y su límite en 0x00C8FF. Además, como son segmentos de código o datos de 32 bits, los bit S y D/B se encuentran seteados en 1 y el bit L en 0. Los bits de DPL de cada uno están seteados en 0 o en 3 de acuerdo con el nivel de privilegio que le corresponda. Por último, los bits de tipo están seteados como 0xA en caso de tratarse de un segmento de código y como 0x2 en caso de tratarse de un segmento de datos. En todos, el bit P se setea en 1 para reflejar que los segmentos están presentes. En el caso del bit de AVL, como es un bit reservado lo seteamos en 0.

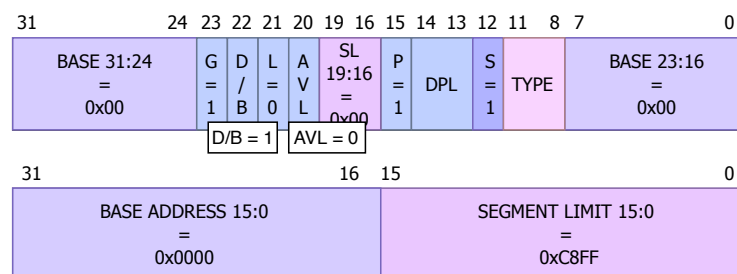


Figura 1: Esquema general de un descriptor de la GDT para los 4 segmentos.

Realizando lo pedido en el ítem b, pasamos a modo protegido. Para poder hacer esto modificamos el archivo `kernel.asm`, allí deshabilitamos las interrupciones, cargamos en el registro GDTR la estructura `gdt_descriptor_t` y modificamos el último bit del registro de control CR0, es decir, seteamos en 1 el bit de *Protection Enable*. Posteriormente, escribimos el código necesario para saltar efectivamente a modo protegido. Debido a que este salto se consigue haciendo un *far jump* a la próxima instrucción, designamos una etiqueta llamada **modo_protegido** a partir de la cual obtendremos el offset, mientras que como selector utilizamos el correspondiente al segmento de código de nivel 0. Además, seteamos la pila del kernel en la dirección 0x25000, es decir, en la base de la pila. Una vez que pasamos a modo protegido, cargamos correctamente los selectores de segmento, en los registros ds, es, fs, gs, ss, usando como registro auxiliar ax.

Seguidamente, definimos otra entrada en la GDT destinada a un segmento de vídeo de nivel 0 (DPL = 0). A diferencia de los cuatro segmentos anteriores, la base de este segmento es distinta de 0, por lo que los campos de la base fueron ser completados con el valor 0x000B8000. Como la pantalla es de 80x50 y como cada pixel ocupa 2 bytes, el límite de este segmento es 0x01F3F. Como este límite entra en 20 bits,

el bit de granularidad de este descriptor queda en 0. Además, al ser un segmento de datos de lectura y escritura el tipo es 0x02 y el bit de s queda seteado en 1. Como en las entradas anteriores, los bits de AVL y L quedan en 0, mientras que los bits de present y D/B quedan en 1.

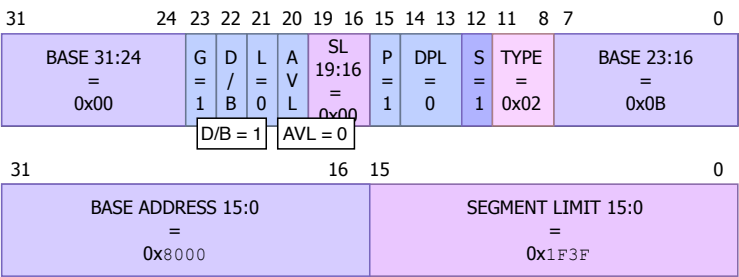


Figura 2: Esquema del descriptor de segmento de video.

Para finalizar este ejercicio, realizamos un rutina encargada de limpiar la pantalla, y pintar el área del mapa de color verde, junto con las barras de los jugadores Rick y Morty. Inicialmente, esta rutina fue implementada en ASM dentro del archivo *kernel.asm*, para corroborar el correcto funcionamiento del acceso al segmento de vídeo, pero luego fue reemplazada por una función en C, llamada “inicializar_pantalla” que se encuentra en el archivo *screen.c*.

- 2.2. Ejercicio 2
- 2.3. Ejercicio 3
- 2.4. Ejercicio 4
- 2.5. Ejercicio 5
- 2.6. Ejercicio 6
- 2.7. Ejercicio 7
- 2.8. Ejercicio 8
- 2.9. Ejercicio 9