Sistemas de computación 2021

Práctico

Modo protegido

Docente:

Solinas, Miguel

Jorge, Javier

Alumna:

Severini Montanari, Alejo

Vega Cuevas, Silvia Jimena

UNC-FCEFyN 2021

Redes de computadoras 2021

Teórico

PGP

Docente:

Britos, Jose Daniel

Alumna:

Vega Cuevas, Silvia Jimena

UNC-FCEFyN 2021

Redes de computadoras 2021

Teórico

PGP

Docente:

Britos, Jose Daniel

Alumna:

Vega Cuevas, Silvia Jimena

UNC-FCEFyN 2021

Redes de computadoras 2021

Teórico

PGP

Docente:

Britos, Jose Daniel

Alumna:

Vega Cuevas, Silvia Jimena

UNC-FCEFyN 2021

Redes de computadoras 2021

Teórico

PGP

Docente:

Britos, Jose Daniel

Alumna:

Vega Cuevas, Silvia Jimena

UNC-FCEFyN 2021

Redes de computadoras 2021

Teórico

PGP

Docente:

Britos, Jose Daniel

Alumna:

Vega Cuevas, Silvia Jimena

UNC-FCEFyN 2021

Desafío: UEFI y coreboot

1- ¿Qué es UEFI? ¿cómo puedo usarlo? Mencionar además una función a la que podría llamar usando esa dinámica.

UEFI (Unifed extensible firmware interface) es una interfaz de firmware utilizada para arrancar el SO. La principal diferencia con BIOS es que en vez de usar MBR este utiliza la tabla de particiones GUID (GPT). Otras diferencias son que UEFI ejecuta instrucciones en 32 o 64 bits, posee un arranque más rápido, interfaz gráfica y soporta más de 2TB de disco (que es lo máximo que tolera la BIOS). También utiliza un mayor número de funciones incluyendo funciones de seguridad como por ejemplo “UEFI secure boot” que evita el ingreso de sistemas no autenticados en el proceso de booteado.

Para utilizar esta funcionalidad se necesita acceder a la BIOS y en la pestaña de configuración de booteo seleccionar UEFI en vez de legacy.
De esta forma se podrá acceder al “UEFI boot mode”, donde UEFI guarda toda la información para la inicialización en un archivo .efi que se guarda en una partición especial llamada ESP (EFI System partition). Durante el procedimiento POST, UEFI escanea todos

los dispositivos de almacenamiento booteables que están conectados al sistema para una tabla de partición GUID válida.

2- ¿Menciona casos de bugs de UEFI que puedan ser explotados?

Una de las razones por las cuales se inventó UEFI fue para reducir la cantidad de rootkits que se utilizaban para explotar la BIOS. Sin embargo, el primer rootkit de UEFI fue expuesto en 2015. Este fue llamado “HT rkloader” realizado por Hacking team, que es una agencia que vende programas con capacidades de intrusión ofensiva y espionaje a distintos gobiernos.

El Hacking Team se aseguraba que su RCS (el sistema que utilizaban para manejar software remotamente) seguiría en el sistema aunque la víctima formateara el disco duro, lo cambiara, instalará el sistema operativo, etc

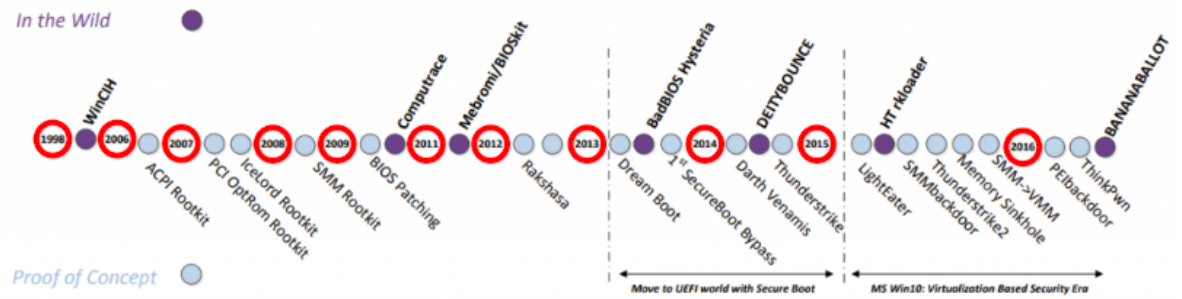


Figura 1: Historia de rootkits para BIOS presentada en “UEFI Firmware Rootkits: Myths and Reality”

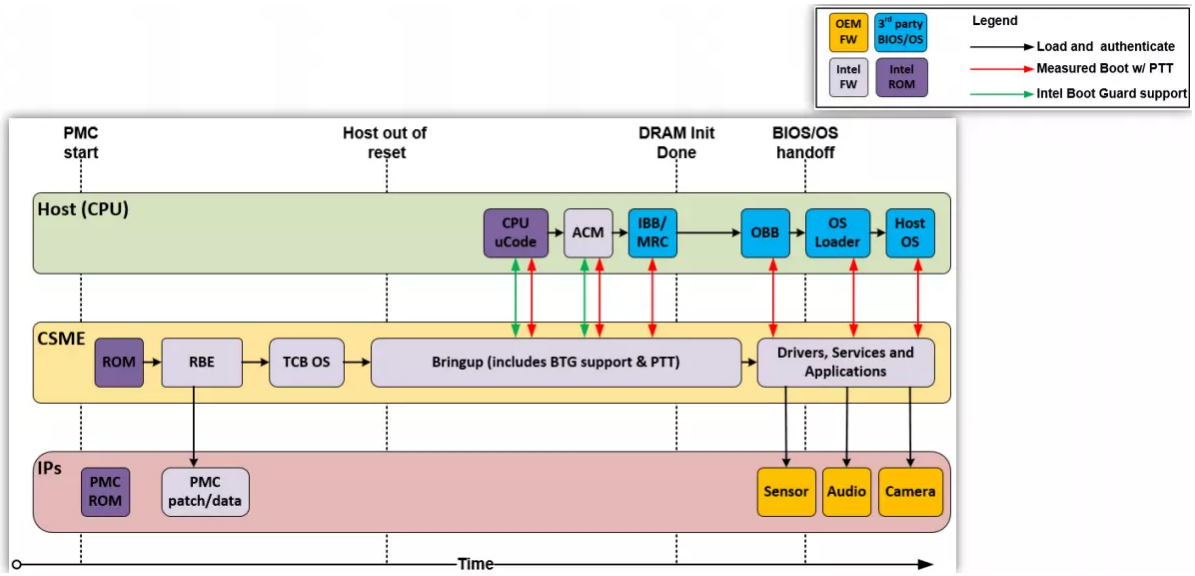
3- ¿Qué es Converged Security and Management Engine (CSME)

CSME es un subsistema embebido y un dispositivo PCIe (Peripheral Component Interconect Express) diseñado para actuar como un controlador de seguridad en el PCH(Platform Controller Hub).

Engloba tres módulos:

- Management Engine (ME)
- Servicios de plataforma de servidor (SPS)
- Trusted execution engine (TXE)

Su objetivo es implementar un ambiente de ejecución aislado y protegido desde el host de los principales procesos de ejecución de software como BIOS, OS y otras aplicaciones.



CSME puede acceder a un número limitado de interfaces como GPIO y LAN para poder realizar sus operaciones regulares.

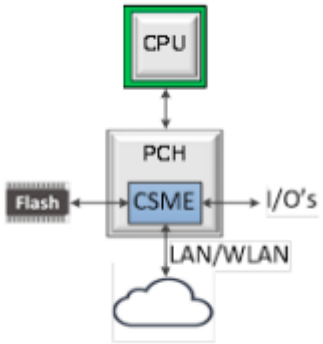


Figure 1 - Intel CSME in the system

Roles principales de CSME:

Chasis

- Arranque seguro de la plataforma.
- Overclocking.
- Carga del código en PCH.

Seguridad

- Ejecución aislada y confiable de los servicios de seguridad.

Manejabilidad

- Gestión de la plataforma en una red fuera de línea.

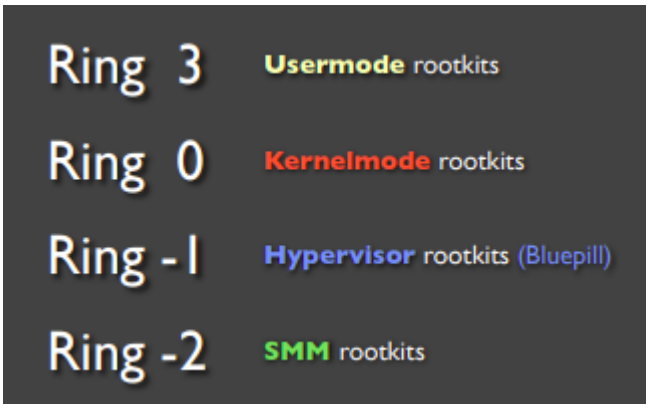
4- ¿Qué es Intel Management Engine BIOS Extension (Intel MEBx).?

Intel Management Engine (ME) carga su código desde la memoria flash de la placa base antes de que se haya cargado el SO.

Algunas funcionalidades:

- Acceso a la memoria sin que la CPU se entere.
- Ejecuta un servidor TCP/IP en la interfaz de red.
- Gestiona la entrada y salida de paquete de datos que se saltan medidas de seguridad como firewall de nuestro sistema.
- Funciona totalmente independiente del SO.
- Su consumo de energía es independiente del procesador.

Seguridad



Algunos expertos en seguridad lo califican como un “anillo de seguridad -3”. Está protegido por un cifrado RSA 2048 lo que lo debería hacer inviolable.

5- ¿Qué es coreboot ? ¿Qué productos lo incorporan ?¿Cuáles son las ventajas de su utilización?

Coreboot (anteriormente llamado LinuxBIOS) es un proyecto dirigido a reemplazar el firmware no libre de los BIOS propietarios.

Ventajas de la utilización:

- Security: Coreboot viene con una base segura de computación que reduce la superficie de ataques generales. También soporta un proceso de booteo seguro llamado VBOOT2.
- Safety: la arquitectura de coreboot fue diseñada para tener un proceso de actualización irrompible. Actualizar el firmware no debería ser más peligroso que instalar una simple app en el smartphone.
- Performance: Coreboot está diseñado para un booteo rápido.

Linker

1- ¿Qué es un linker? ¿qué hace ?

Un linker es un programa que toma uno o más objetos generados por un compilador o un ensamblador y los combina dentro de un mismo ejecutable. Esto consta de dos tareas fundamentales, la resolución de símbolos y la relocation.

2- ¿Qué es la dirección que aparece en el script del linker?¿Por qué es necesaria ?

La dirección que aparece es a partir de dónde se va a copiar el código que se trae del disco. Es necesaria ya que el linker calcula las direcciones de los símbolos a partir de esa dirección.

3- Compare la salida de objdump con hd, verifique donde fue colocado el programa dentro de la imagen.

```
➜ hello_world objdump -f -S main.o

main.o:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000


Disassembly of section .text:

0000000000000000 <loop-0x5>:
.code16
    mov $msg, %si
   0:   be 00 00 b4 0e          mov    $0xeb40000,%esi

0000000000000005 <loop>:
    mov $0x0e, %ah
```

Utilizando la opción -f se puede ver que la dirección de comienzo del programa está en 0x00h, corroborando el siguiente esquema presentado en clases.

Address		Description	Size (bytes)
Hex	Dec		
$+000_{hex}$	+0	Bootstrap code area	446
$+1BE_{hex}$	+446	Partition entry №1	16

Además al utilizar la opción hexdump (hd) se ve que la dirección de comienzo del programa también es 0x00.

```
→ hello_world hd main.img
00000000  be 0f 7c b4 0e ac 08 c0  74 04 cd 10 eb f7 f4 68  |..|.....t......h|
00000010  65 6c 6c 6f 20 77 6f 72  6c 64 00 66 2e 0f 1f 84  |ello world.f....|
00000020  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000030  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000040  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
00000050  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
00000060  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
00000070  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000080  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000090  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
000000a0  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
000000b0  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
000000c0  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
000000d0  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
000000e0  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
000000f0  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
00000100  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
00000110  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000120  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000130  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
00000140  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
00000150  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
00000160  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000170  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000180  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
00000190  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
000001a0  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
000001b0  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
000001c0  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
000001d0  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
000001e0  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
000001f0  00 66 2e 0f 1f 84 00 00  00 00 00 0f 1f 00 55 aa  |.f............U.|
00000200
```

4- Grabar la imagen en un pendrive y probarla en una pc y subir una foto

```
alejo@alejo:~/UNC/5_Primer_Semestre/siscomp/mbr/hello_world_v2$ sh run
+ as -ggdb3 --32 -o entry.o entry.S
entry.S: Assembler messages:
entry.S: Warning: end of file not at end of a line; newline inserted
+ gcc -c -ggdb3 -m16 -ffreestanding -fno-PIE -nostartfiles -nostdlib -o main.o -
std=c99 main.c
+ ld -m elf_i386 -o main.elf -T linker.ld entry.o main.o
+ objcopy -O binary main.elf main.img
+ qemu-system-x86_64 -drive file=main.img,format=raw
```

```
QEMU
Machine  View
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8CB00+07ECCB00 CA00


Booting from Hard Disk...
hello world_
```

Como se puede ver en la imagen no aparece el hello world pero sí la función halt, no pudimos encontrar el problema o el por qué de esto.

Decidimos hacer otro intento siguiendo los pasos del siguiente repositorio: https://github.com/cirosantilli/x86-bare-metal-examples

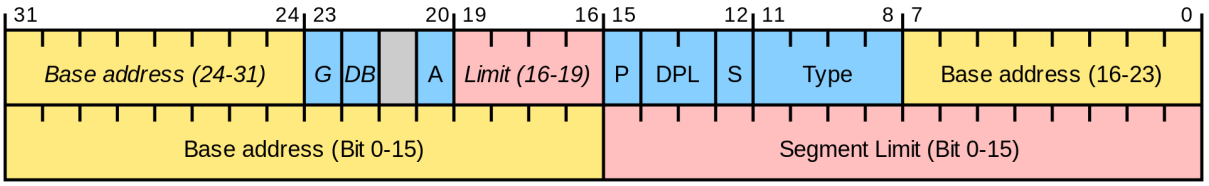
Siguiendo ese repositorio conseguimos que funcione de manera correcta:

5- ¿Para qué se utiliza la opción --oformat binary en el linker?

Es utilizado para especificar el formato del archivo binario resultante del linkeo de los objetos cuando se está linkeando más de un objeto.

Desafío final: Modo protegido

1- Crear un código assembler que pueda pasar a modo protegido (sin macros).

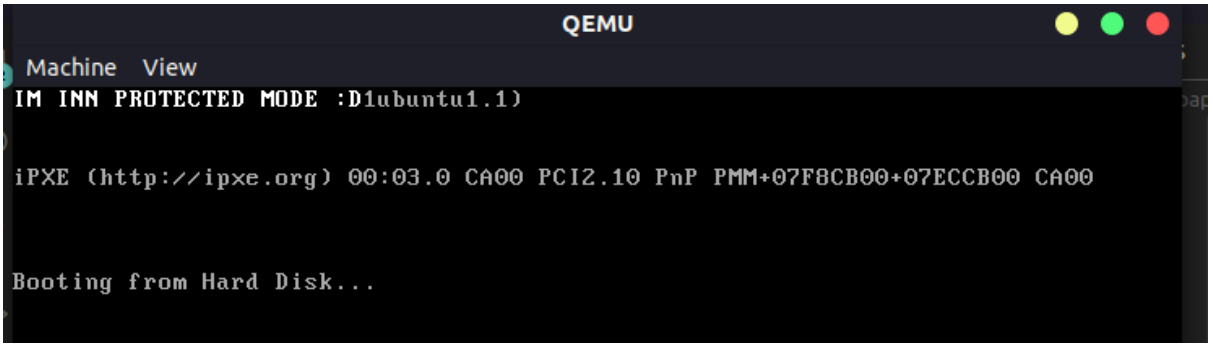



```
gdt_start:
gdt_null:
    .long 0x0
    .long 0x0

gdt_code:
    .word 0xffff
    .word 0x0
    .byte 0x0
    .byte 0b10011010
    .byte 0b11001111
    .byte 0x0

gdt_data:
    .word 0xffff
    .word 0x0
    .byte 0x0
    .byte 0b10010010
    .byte 0b11001111
    .byte 0x0
gdt_end:
```

Se obtuvo el siguiente mensaje en pantalla:



2- ¿Cómo sería un programa que tenga dos descriptores de memoria diferentes, uno para cada segmento (código y datos) en espacios de memoria diferenciados?

Sería como el código que se encuentra en la imagen del punto anterior, donde tenemos dos entradas en la GDT, una para el segmento de código y otra para el segmento de datos. Estos descriptores deben ser distintos ya que el segmento de código no puede ser escrito, mientras que el de data si, y el segmento de data no puede ser ejecutado, mientras que el de código si. Por lo tanto tienen que ser segmentos distintos, en distintos espacios de memoria con sus propias características.

3- Cambiar los bits de acceso del segmento de datos para que sea de solo lectura, intentar escribir, ¿Qué sucede? ¿Qué debería suceder a continuación? (revisar el teórico) Verificarlo con gdb.

En teoría:

Según lo visto en la clase de modos de protección, para evitar el uso incorrecto de un segmento o una puerta se tiene en cuenta el tipo de los descriptores de segmento que viene determinado por el campo tipo y el flag S.

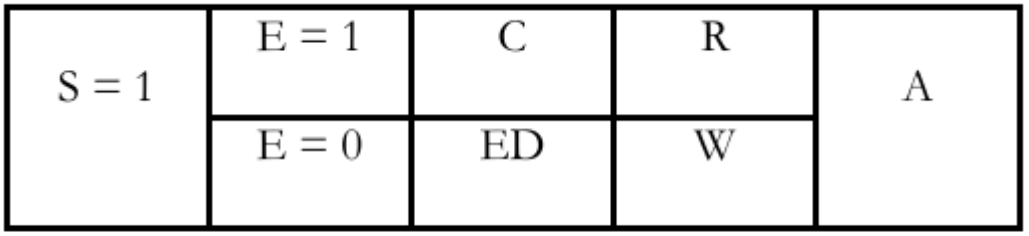
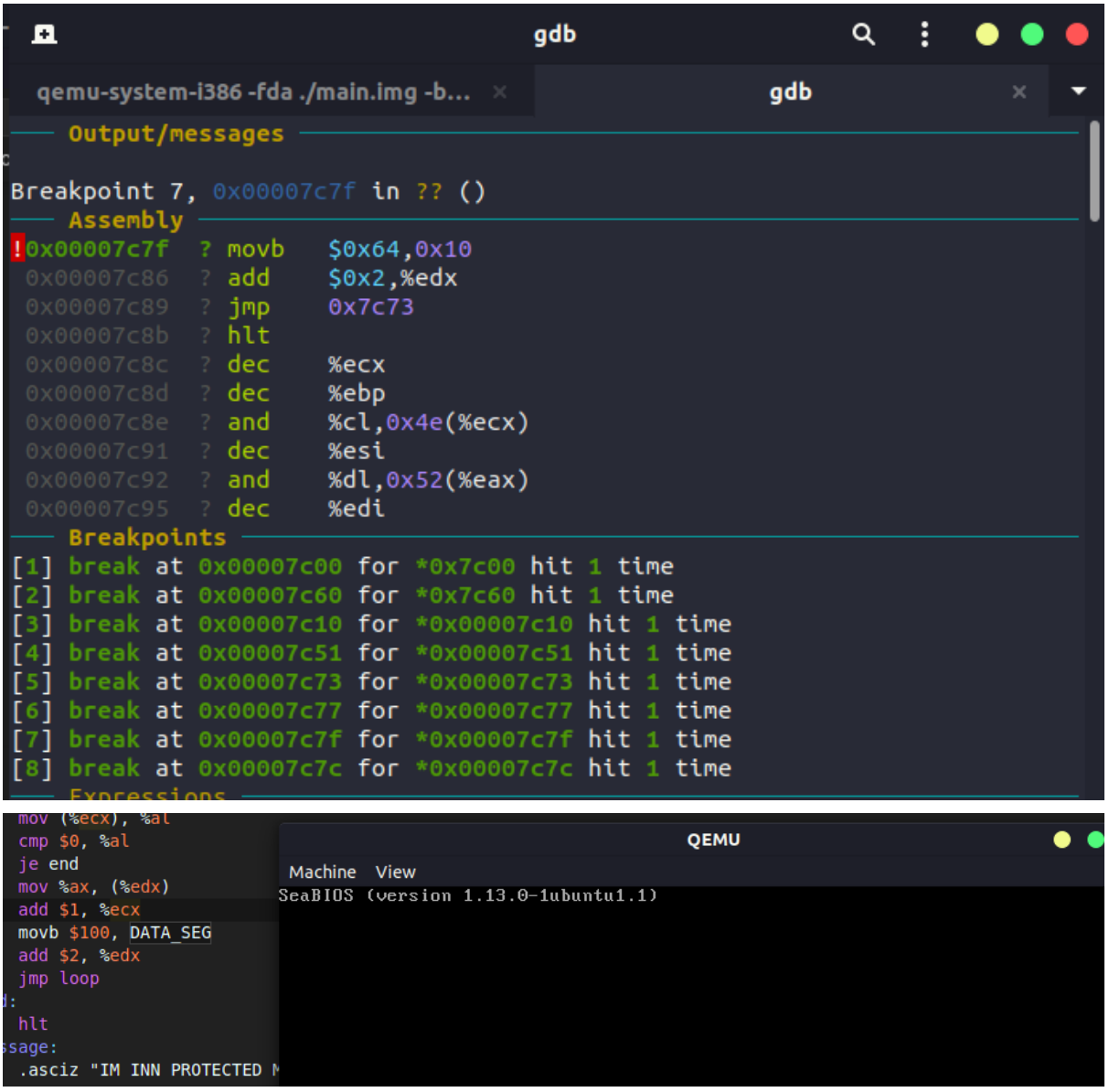


Figura 10.6. Formato del campo TIPO

Se sabe que el segmento de datos es imposible de ejecutar, por lo tanto la flag E es cero. Luego, si se pone W=0 y se intenta escribir en el segmento de datos el mecanismo de protección de segmento genera una excepción y detiene el flujo ejecución normal de la CPU.

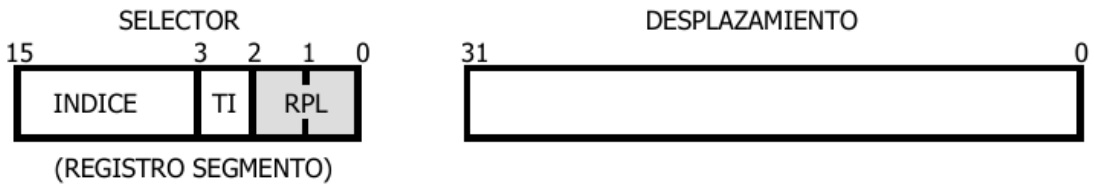
En la práctica:

Tras modificarse el bit W=0 en en el registro “Tipo” en el descriptor de datos, se escribe el valor constante 100 en la dirección de memoria del segmento de datos. A continuación se pudo ver que el programa se reiniciaba constantemente debido al error.



4- En modo protegido, ¿Con qué valor se cargan los registros de segmento? ¿Por qué?

En modo protegido los registros de segmento son llamados selectores.



Estos apuntan a estructuras de datos llamadas descriptores de segmento que contienen la información necesaria para acceder a una dirección de memoria física.

Elementos:

- Índice: son los 13 primeros bits más significativos. Le permiten redireccionar 8k de descriptores en total.
- TI: especifica si se apunta a un descriptor de la GDT o un descriptor de la LDT.
- RPL (Requested Privilege Level): Es el privilegio asociado al selector del segmento, puede tomar los valores 0, 1, 2 y 3, siendo 0 Root privilege y 3 User privilege.