Organización del Computador II

Trabajo Práctico 2

Emilio Almansi
ealmansi@gmail.com

Álvaro Machicado
rednaxela.007@hotmail.com

Miguel Duarte
miguelfeliped@gmail.com

2do cuatrimestre de 2013

Resumen

Un breve abstract sobre como arruinar tu vida en 24 hs, cortesía de Bochs.

Índice

1. Introducción **2**

2. Modo real y modo protegido **3**

3. Global Descriptor Table (GDT) **4**

4. Paginación **5**

5. Interrupciones **6**

6. Manejo de tareas **7**

7. Manejo de tareas **8**

8. Scheduler **9**

9. Conclusiones **10**

1. Introducción

2. Modo real y modo protegido

En la GDT hay que poner los descriptores de segmento y los descriptores de TSS para cada cada tarea y para cada bandera.

La misma está representada como un arreglo "*gdt_entry*" declarado de manera global en C. Las *gdt_entry* son structs de 4 bytes que poseen un campo cara cada atributo de una entrada de gdt.

Los descriptores de segmento fueron cargados de manera estática en tiempo de compilación. Lo mismo con el descriptor de la IDT. Esto fue posible porque se conocen de antemano todos los valores necesarios para completar los descriptores.

A la hora de cargar los descriptores de TSS nos encontramos con la siguiente dificultad: Los descriptores de TSS fueron declarados como una variable global en código C. Por lo tanto en tiempo de compilación no se sabe en que dirección van a ser cargados. Por este motivo se cargan de manera dinámica mediante una función que se llama desde kernel.asm. La función sencillamente crea una entrada más en el arreglo que representa de *gdt_entry* con los atributos adecuados.

3. Global Descriptor Table (GDT)

En la GDT hay que poner los descriptores de segmento y los descriptores de TSS para cada cada tarea y para cada bandera.

La misma está representada como un arreglo "*gdt_entry*" declarado de manera global en C. Las *gdt_entry* son structs de 4 bytes que poseen un campo cara cada atributo de una entrada de gdt.

Los descriptores de segmento fueron cargados de manera estática en tiempo de compilación. Lo mismo con el descriptor de la IDT. Esto fue posible porque se conocen de antemano todos los valores necesarios para completar los descriptores.

A la hora de cargar los descriptores de TSS nos encontramos con la siguiente dificultad: Los descriptores de TSS fueron declarados como una variable global en código C. Por lo tanto en tiempo de compilación no se sabe en que dirección van a ser cargados. Por este motivo se cargan de manera dinámica mediante una función que se llama desde kernel.asm. La función sencillamente crea una entrada más en el arreglo que representa de *gdt_entry* con los atributos adecuados.

4. Paginación

5. Interrupciones

6. Manejo de tareas

El manejo de tareas tiene varias aristas. Por un lado está la gestión "física" de las tareas. Es decir como se cambia entre el contexto de una y el contexto de otra. Este punto se manejo mediante el mecanismo automático que provee la arquitectura intel.

Para esto hubo que crear tss para cada tarea. Además por cuestiones de facilitar la gestión la tareas se decidió crear una tss extra para cada bandera. De esta forma cada navio tiene 2 tss, una para correr su código de acción y otra para correr su código de bandera. Además hay 2 tss extra, una para la tarea idle y otra auxiliar, para poder realizar el primer salto sin inconvenientes.

Para poder primer cambio de contexto (JMP FAR) sobre esta tss auxiliar previamente se setea el TR. Este seteo está hardcodeado porque el descriptor de la tss auxiliar siemre se guarda exactamente en el mismo índice de la gdt.

Todos los cambios de texto se hacen mediante un JMP FAR. En ningún momento se usa ningún otro mecanismo. El problema que surge con esto es que las banderas no tienen un código cíclico. Es decir, las banderas ejecutan una porción de código que termina, al terminar su contexto queda estático en la posición en la uqe terminó, por lo tanto se se llama a esa bandera otra vez en ese mismo contexto sin hacer nada la bandera va a ejecutar cosas indebidas. Para subsanar ese inconveniente siempre antes de saltar una bandera se vuelve a inicializar su tss, de esta forma nos aseguramos de que el contexto siempre sea el contexto inicial.

7. Protección

Las tareas corren en el anillo de seguridad 3. El kernel en anillo 0. A nivel paginación las tareas son usuario y el kernel es supervisor.

Las tareas no pueden acceder a memoria del kernel bajo ninguna circinstancia. Esto está asegurado en el mapa de memoria. El mismo si bien incluye las tablasde memoria del kernel las incluye con un privilegios mayor, por lo tanto si una tarea intenta acceder a esas páginas cae un page fault que termina en el desalojo de esa tarea.

El kernel está programado para no tener que incurrir en ninguna excepción especial, lo tanto el código que se ejecuta cuando ocurre una excepción es siempre el mismo. Desalojar a esa tarea y saltar a la tarea iddle. Esto asegura que toda tarea que realice una acción ilegal que produzca uan excepción (dividir entre cero, seg fault, page fault, etc) será desalojada.

Además hay un par de cuestiones extra. Por ejemplo, una bandera no puede llamar a una int0x50. Si hace esto inmediatamente es desalojada.

8. Scheduler

9. Conclusiones