Trabajo Práctico 1 ∼ PSO

Liptak, Leandro
Reboratti, Patricio
Silvani, Damián

May 5, 2011

Documentación

Kernel Básico

`kernel` - Inicialización

`gdt` - Global Descriptor Table

`vga` - Pantalla

...

`idt` - Administrador de interrupciones

Como creímos (equivocadamente) que iba a resultar en código mantenible y facil de leer, definimos tres *wrappers* comunes para los *handlers* de interrupción definidos en la IDT: `ISR_NOERRCODE`, `ISR_ERRCODE` y `IRQ`.

Los primeros dos corresponden a un handler de excepción del procesador, y el tercero a un IRQ proveniente del PIC. La diferencia entre los dos primeros está dada en que hay ciertas excepciones en las que el procesador pushea un código de error a la pila antes de saltar al handler. Ambos finalmente hacen `jmp` a un handler común llamado `isr_common`.

Por otro lado `IRQ` salta a `irq_common`, previamente pusheando el número de IRQ y de la interrupción tal como está registrada en la IDT. Finalmente salta a `irq_common`.

Luego del salto, ambos pushean los registros de uso general y llaman a una función definida en C. Luego del `call` se acomoda la pila y se ejecuta finalmente `iret`.

Una vez en el mundo C, llamamos al handler definitivo registrado en un arreglo `interrupt_handlers` utilizando en número de interrupción obtenido de la pila. Aprovechando la pila que nos deja el procesador al principio del proceso de manejo de interrupción, y los otros valores que pusheamos antes, definimos una estructura en C llamada `registers_t` que contiene toda esta información, necesaria para cualquier handler que sea necesario registrar.

En el caso particular de las IRQs, antes de llamra al handler definitivo, se envía una serie de comandos a la PIC maestra y esclava para señalizar que la interrupción está siendo atendida.

El método `idt_register`, además de registrar en el arreglo `interrupt_handlers` el puntero al handler en C de la interrupción pedida, debe definir la entrada correspondiente en la IDT, con el puntero al handler en ASM definido por las macros mencionadas al principio. Es por eso que fue necesario un segundo arreglo, esta vez de punteros a los handlers en ASM.

Lo que al principio pareció ser una buena idea, terminó complicando la implementación, además de ser menos eficiente en espacio y tiempo. Para los dos arreglos de punteros de 256 entradas, se necesitan 2KB de espacio de kernel. En cuanto a la eficiencia, se realiza un `call` extra para todo handler de interrupción.

Surgió otro problema en la etapa de desarollo del módulo `loader`, dónde necesitamos que el handler del PIT sea rápido y tuviera una pila lo más chica posible, y el wrapper común nos dificultaba. Terminamos definiendo un `idt_register_asm` alternativo, tal como está especificado en la consigna del TP, que permitiera definir un handler en ASM, la cual es registrada en la IDT.

`debug` - Debug

Kernel funcional

`mm` - Manejador de memoria

`sched` - Scheduler

`loader` - Loader y administrador de tareas

`sem` - Semáforos de kernel