Organización del Computador II
System Programming
Taller 2: Interrupciones

## El taller de hoy: Interrupciones

Durante la clase anterior, definimos la segmentación de la memoria, seteamos los valores iniciales de los registros de segmento, de los registros de la pila y mapeamos la pantalla a un segmento de video en memoria. También, modificamos el bit menos significativo del registro CR0 para hacer el pasaje a modo protegido.

En la clase de hoy, vamos a definir la tabla de interrupciones (IDT - Interrupt Descriptor Table) para que nuestro procesador pueda atender adecuadamente las interrupciones que provengan del software, dispositivos externos e internas.

En particular, vamos a implementar dos importantes rutinas de atención de interrupción: la correspondiente al reloj y la del teclado.

## Archivos provistos

A continuación les pasamos la lista de archivos que forman parte del taller de hoy junto con su descripción:

- **`Makefile`** - encargado de compilar y generar la imagen del floppy disk.
- **`bochsrc, bochsdbg`** - configuración para inicializar Bochs.
- **`diskette.img`** - la imagen del floppy que contiene el boot-sector preparado para cargar el kernel.
- **`kernel.asm`** - esquema básico del código para el kernel.
- **`defines.h, colors.h`** - constantes y definiciones.
- **`gdt.h, gdt.c`** - definición de la tabla de descriptores globales.
- **`screen.h, screen.c`** - rutinas para pintar la pantalla.
- **`a20.asm`** - rutinas para habilitar y deshabilitar A20.
- **`print.mac`** - macros útiles para imprimir por pantalla y transformar valores.
- **`idle.asm`** - código de la tarea Idle.
- **`syscalls.h`** - interfaz a utilizar desde las tareas para los llamados al sistema.
- **`kassert.h`** - rutinas para garantizar invariantes en el kernel.
- **`i386.h`** - funciones auxiliares para utilizar assembly desde C.
- **`pic.c, pic.h`** - funciones pic_enable, pic_disable, pic_finish1 y pic_reset.
- **`isr.h, isr.asm`** - rutinas de atención de interrupciones
- **`idt.h, idt.c`** - definición de la tabla de interrupciones

Los ejercicios que vamos a realizar hoy les van a pedir interpretar código de estos archivos y escribir código únicamente en los archivos **isr.h, isr.asm, idt.h** y **idt.c.**

## Ejercicios

### Primera parte: Definiendo la IDT

En esta parte, vamos a definir y cargar la IDT en memoria. Utilizaremos los descriptores de excepciones, interrupciones y system calls. Además, vamos a trabajar en las rutinas de atención de interrupción (*ISR: Interrupt Service Routine*), en particular, las necesarias para atender las excepciones. Estas rutinas, en principio, sólo van a imprimir en pantalla la excepción generada, junto con el estado de los registros del microprocesador.

Completen en grupo los siguientes puntos:

1. En el archivo `idt.h`, pueden encontrar la IDT definida como un arreglo de **idt_entry_t** declarado sólo una vez como **`idt`**. El descriptor de la IDT en el código se llama **IDT_DESC**.
   En el archivo `idt.c` pueden encontrar la definición de cada una de las entradas y la definición de la función **idt_init** que inicializa la IDT definiendo cada una de sus entradas usando la macro **IDT_ENTRY0**.
   a) Observen que la macro **IDT_ENTRY0** corresponde a cada entrada de la IDT de nivel 0 ¿A qué se refiere cada campo? ¿Qué valores toma el campo `offset`?
   Pueden ayudarse de la siguiente figura. Observen que los atributos son los que van desde el flag P al final de los bits reservados (bits 15 a 0 de la palabra de 32 bits superior).

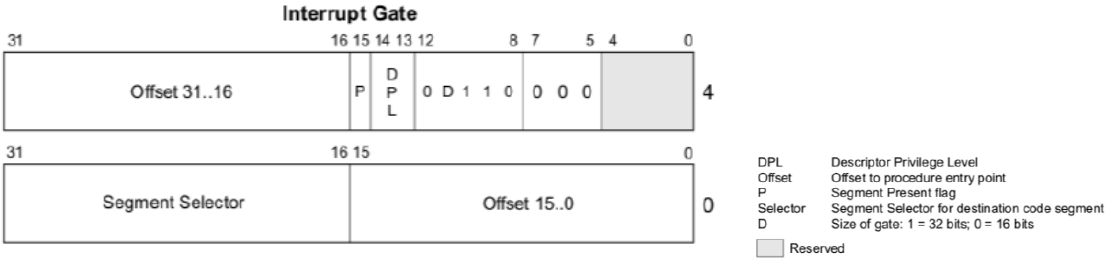


Figura 1: Entrada de la IDT - Interrupt Gate

   b) Completar los campos de Selector de Segmento (**segsel**) y los atributos(**attr**) de manera que al usarse la macro defina una *Interrupt Gate* de nivel 0. Para el Selector de Segmento, recuerden que la rutina de atención de interrupción es un código que corre en el nivel del kernel. ¿Cuál sería un selector de segmento apropiado acorde a los índices definidos en la GDT(**segsel**)? ¿Y el valor de los atributos usamos Gate Size de 32 bits (**attr**)?
   c) De manera similar, completar la macro **IDT_ENTRY3** para que defina interrupciones que puedan ser disparadas por código no privilegiado (nivel 3).

## Primer Checkpoint

2. Nos queda definir dos system calls. Estas son interrupciones de software que se van a poder usar con nivel de privilegio de usuario, en nuestro caso, nivel 3. Usar la macro **IDT_ENTRY3** para definir system calls con número 88 y 98.
3. Completen **kernel.asm** inicializando la IDT y usen LIDT para cargar la IDT en memoria
   Compilen y ejecuten con bochs. Pueden examinar la IDT con el comando: **idt info** (o **idt info i**, muestra la entrada iésima)

## Segundo Checkpoint

**<u>Segunda parte: Rutinas de Atención de Interrupción</u>**

El manejo de las interrupciones lo estaremos haciendo en la siguiente parte del taller.

4. Agregar a la IDT (`idt.c` - método `idt_init`) las entradas para la interrupción de reloj(32) y la de teclado(33). Dado que el reloj y el teclado son manejados por el kernel, estas deben ser de nivel 0.

5. Las rutinas de atención de interrupción son definidas en el archivo **`isr.asm`**. Cada una está definida usando la etiqueta _isr## donde ## es el número de la interrupción. Busquen en el archivo la rutina de atención de interrupción del reloj.
   Con el código provisto a continuación, completar la rutina asociada a la interrupción del reloj, para que por cada tick llame a la función **`nextClock`**. La misma se encarga de mostrar cada vez que se llame, la animación de un cursor rotando en la esquina inferior derecha de la pantalla. La función **`nextClock`** está definida en isr.asm.

   Este es el código la rutina de atención de interrupción del reloj que tienen que usar:

```
pushad
call pic_finish1
call next_clock
popad
iret
```

6. ¿Qué hacen `pushad` y `popad`? ¿Qué marca el `iret` y por qué no usamos `ret`?
7. Con el código provisto a continuación, completen la rutina de interrupción de teclado. Expliquen intuitivamente qué realiza la rutina de atención del teclado.

```
pushad
call pic_finish1
in al, 0x60

push eax
call printScanCode
add esp, 4

popad
```

8. Escribir las rutinas asociadas a las interrupciones 88 y 98, para que modifique el valor de eax por 0x58 y 0x62. Posteriormente este comportamiento va a ser modificado para atender cada uno de los servicios del sistema.
9. Habiliten las interrupciones con STI en **kernel.asm**. Luego, escriban un par de líneas que utilicen la instrucción **int** para probar alguna de las interrupciones de software que cargaron. Pueden usar un breakpoint después de la interrupción para inspeccionar los registros y verificar que se produjo el cambio de valor en eax.
10. Compilen y ejecuten con bochs. Verifiquen la ejecución de la rutina de atención del reloj y de las interrupciones 88, 98. Discutan por qué va rotando el reloj.

## Tercer Checkpoint

11. [Optativo] Ahora prestemos atención a los archivos **isr.h** e **isr.asm.** Acá están declaradas y definidas, respectivamente, las rutinas de servicio de cada interrupción. Están definidas por medio de 3 macros: ISRc, ISRNE, ISRE. En particular, ISRc define una rutina de interrupción genérica. Lo primero que hace es pushear el número de interrupción (argumento de la macro) en la primera instrucción. ¿Cómo queda el estado de la pila luego de esa instrucción? Completar todo lo marcado con “??” en el comentario que encabeza a la función. Nota: Consideren que la interrupción siempre apila un código de error.