Arquitecturas de las Computadoras
TP5 - Acceso a Hardware

Introducción

Hasta ahora estuvimos trabajando como cliente del Sistema Operativo (SO) a través de su API. La responsabilidad del SO es administrar los recursos de la computadora. Estos recursos son los periféricos y la memoria. En esta guía vamos a hacer un acceso directo y empezar a trabajar de cómo administrarlos.

Para este trabajo contamos una imagen de diskette, con GRUB instalado, que luego de inicializar, va a cargar en memoria el *kernel* que nosotros programamos. GRUB inicializa la PC en modo protegido, con las interrupciones deshabilitadas y la GDT configurada en *modelo flat,* un registro de 4 GB de datos, otro de 4 GB de código, uno de 1MB de datos y otro de 1MB de datos. Los más pequeños son para poder correr aplicaciones de 16 bits.

Esta práctica cuenta con los siguientes elementos:

- `loader.asm`: recibe el control de GRUB luego que de la máquina se inició y se configuró.
- `main.c`: punto de entrada de C, no debe haber ninguna función de la biblioteca estándar, ya que donde esto va a correr, no está.
- `link.ld`: script de compilación para indicarle a *ld* donde tiene que separar las secciones y qué etiquetas definir
- `Makefile`: es un script sencillo para compilar y cargar el kernel en la imagen de diskette.
- `kernel.bin`: es el binario ejecutable que GRUB va a cargar en memoria.

El script de compilación hace uso del programa *mtools,* que es útil para modificar archivos de discos rígidos que tengan formato FAT. Para que el script funcione, deben crear el archivo (o agregar a éste) el archivo **.mtoolsrc** la siguiente línea (modificandola según corresponda) en el home.

```
drive z: file="/path/to/the/disk.img"
```

Luego, para correrla, pueden hacer uso de **qemu**, que es un emulador que puede simular varios entornos distintos

```
$> qemu-system-i386 -fda disk.img
```

Algo interesante que tiene **qemu** es la posibilidad de conectar el debugger de GCC para seguir el código y corregir errores.

```
$> qemu-system-i386 -s -S -fda disk.img
```

Y desde dónde queda el archivo `kernel.bin` (que debe estar compilado con símbolos de debug) es necesario correr

```
$> gdb kernel.bin
```

Luego, cuando este cargue,

```
(gdb) target remote localhost:1234
```

Luego se puede hacer un breakpoint a la función main y darle correr.

Si utilizan la VM de TinyCore, deben abrir una terminal y correr el siguiente comando:

```
sudo ln -s /usr/local/lib/libgcrypt.so /usr/local/lib/libgcrypt.so.11
```

Porque un Qemu busca la biblioteca libgcrypt versión 11, pero está instalada la 20.

Ejercicio 1

La pantalla es un periférico que está mapeado en memoria. En modo texto comienza *0xB8000.* Consta de 80 columnas y 25 filas, donde cada posición consta de dos componentes, una para el carácter a dibujar y su atributo, el color de fondo y el color de la letra.

Escriba en la pantalla la leyenda "Arquitecturas de las Computadoras", con blanco y letras verdes.

Ejercicio 2

Basado en el ejercicio anterior, escribir una función para escribir mensajes por la pantalla. Implemente un mini driver de video.

Ejercicio 3

El **Real Time Clock (RTC)** es el periférico que almacena la hora del sistema. Mientras la Placa Madre tenga pila, éste llevará la hora y es lo que permite conservar la hora luego de que la PC se apague y se le quite la energía.

El RTC permite leer y configurar la el año, el mes, el día, la hora, los minutos y los segundos por separado. También permite configurar una alarma de la misma forma. Para no consumir tantas direcciones de E/S, se accede de manera diferida. En el registro de E/S 70h, se elige lo que se quiere leer/escribir y en el registro 71h se hace la lectura. Por ejemplo, para obtener el los segundos, habría que ejecutar las siguientes líneas:

```
out 70h, 0
in ax, 71h
```

Implemente una función que informe por pantalla la hora actual del sistema. Para más información sobre el RTC puede acceder a la siguiente página:

Doc: http://stanislavs.org/helppc/cmos_ram.html

Ejercicio 4

Hay dos formas de obtener teclas del teclado. La primera que vamos a utilizar es por *pooling,* que consiste en consultar constantemente por la existencia en una tecla.

Lea la documentación del funcionamiento del teclado y escriba una función que se quede esperando que se haya presionado una tecla. Cuando esto suceda, mostrar en pantalla que tecla fue la que se presionó.

Doc: http://stanislavs.org/helppc/8042.html

Ejercicio 5

Ahora vamos a hacer uso de las interrupciones de Hardware. Antes que nada, es necesario configurar la IDT para que cuando una interrupción llegue al procesador, busque en la tabla cuál es el código a ejecutar.

Bajar el Archivo TP6-IDT.zip para ver su funcionamiento.

Compilar el sistema y analizar los elementos que son necesarios para configurar la IDT. Modifique la interrupción de Timer Tick para analizar el comportamiento de dicha interrupción. Implemente un mecanismo para que cada 5 segundos imprima un nuevo mensaje en la pantalla.

Ejercicio 6

Agregue un nuevo handler a la IDT para manejar las teclas del teclado mediante interrupciones. Las teclas deben ir apareciendo en la pantalla a medida que se escriban.

Ejercicio 7

Como parte de la implementación de un Sistema Operativo, es necesario proveer una API al usuario para que use funciones de éste. Agregue a la IDT la entrada 80h, configure un handler adecuado y provea el **Sistem Call write**, que funcione de acuerdo a la implementación que se ha visto en clase. Defina cómo fds:

- STDOUT (1) : la pantalla
- STDERR (2) : la pantalla, pero con texto rojo.

El objetivo es administrar y facilitar el acceso a los distintos recursos del sistema. La pantalla es uno.