LiveFDisk

Arquitectura del Computador

Guido De Luca
D - 3853/9

Tomás López
L - 2835/5

Florencia Rovere
R - 3961/6

23/06/2016

1 Introducción

El objetivo del trabajo es diseñar un programa que pueda ejecutarse previo al booteo del sistema operativo, cuya función es mostrar la tabla de particiones contenida en algún dispositivo de almacenamiento. La misma utilizará funciones de la BIOS para leer y escribir sectores del disco, y el acceso a la memoria de video para imprimir en pantalla.

Para lograr esto, utilizaremos QEMU como máquina virtual con arquitectura i386. El proceso de compilado y ejecución se detallará en otra sección del informe.

2 Contextualización teórica

2.1 Proceso de booteo

Antes de cargar el SO a memoria, la computadora sólo cuenta con un software encargado de controlar la entrada y salida de datos (BIOS: Basic Input/Output Software). El mismo consiste en una serie de rutinas cargadas a memoria en el momento que la computadora fue prendida, y provee una auto-detección de los dispositivos más esenciales (pantalla y teclado, por ejemplo).
Luego de que la BIOS complete algunos testeos de hardware, particularmente si la memoria principal está funcionando correctamente, deberá cargar en memoria nuestro **sector de booteo**. El sector de booteo es el primer sector físico del disco (es decir, 512 bytes, Cilindro 0, Cabezal 0, Sector 0), y debe llevar la signatura **0xaa55** al final para que sea reconocido como tal.
Una vez reconocido este sector, la BIOS lo carga en memoria en la dirección 0x7c00 y comienza a ejecutar el código allí detallado.

2.2 MBR

El sector de booteo previamente mencionado se conoce con el nombre de Master Boot Record. A continuación analizaremos en qué consiste. Consideraremos que nuestra MBR (con un tamaño de 512 bytes) está dividida en tres partes:

- Código ejecutable: son los primeros 446 bytes del sector.
- Tabla de particiones: contiene una descripción del disco. Puede tener hasta cuatro entradas, de las cuales al menos una debe ser una partición primaria activa para que se produzca correctamente el arranque. Cada entrada tiene un tamaño de 16 bytes, por lo que la tabla ocupa 64 bytes.
- Firma MBR: son los últimos dos bytes del sector: 0xaa55.

2.3 Tabla de particiones

Veamos un poco más de cerca la tabla de particiones. La siguiente tabla representa el corrimiento correspondiente a cada entrada.

Número de partición	**Offset**
Partición 1	0x1be
Partición 2	0x1ce
Partición 3	0x1de
Partición 4	0x1ee

E internamente, cada partición se organiza de la siguiente forma.

Elemento (Offset)	**Tamaño**	**Descripción**
0	byte	Indicador de booteo
1	byte	Primer cabezal
2	6 bits	Primer sector
3	10 bits	Primer Cilindro
4	byte	ID del sistema de archivos
5	byte	Último cabezal
6	6 bits	Último sector
7	10 bits	Último cilindro
8	uint32_t	Sector relativo
12	uint32_t	Número total de sectores

2.4 Tabla de interrupciones

Para el desarrollo de nuestro programa, vamos a utilizar determinadas rutinas de la BIOS. Dentro de cada interrupción, le pasamos dos bytes como argumento en el registro **ah** para explicitar qué rutina queremos realizar. A continuación listamos las interrupciones que usamos.

- INT 0x10: Servicios de video.
  - 0x00: Empezar modo de video.
  - 0x06: Limpiar la pantalla.
- INT 0x13: Servicios de disco de bajo nivel.
  - 0x02: Leer disco.
  - 0x03: Escribir disco.
  - 0x08: Leer parámetros generales del disco.
- INT 0x15: Misceláneas

  - 0x53: Control de energía

- INT 0x16: Provee rutinas que se comunican con el teclado.
  - 0x00: Lee un caracter del teclado.

3 Nuestro programa

3.1 Interpretación del código

Nuestro código consiste en dos partes: una cuya función es cargar en memoria el resto del programa (para no limitarnos a los 446 bytes de código de la MBR). Ahora, la nueva sección ya cargada en memoria, es la que se encargará de realizar la consigna del trabajo.

loader.s

Es el encargado de cargar el resto del programa en algún lugar de la memoria dónde haya espacio suficiente para almacenarlo (nosotros elegimos la dirección 0x8000). Utilizará la interrupción 0x13 con el fin de leer los dos sectores siguientes al sector de booteo, dónde se encontrará el código necesario para leer y manipular la tabla de particiones. Una vez cargado en memoria, setea el data segment y realiza un salto a la dirección mencionada.

fdisk.s

Una vez realizado el salto, se inicializará el stack y comenzará a ejecutarse este código. Lo primero que veremos será un menú que nos dejará elegir entre tres opciones: mostrar la tabla de particiones, mostrar los parámetros generales de la unidad o modificar el ID de la tercer partición.

- Opción 1: Mostrar la tabla de particiones
  Para leer la tabla, utilizaremos la interrupción 0x13 para leer el primer sector de la unidad de almacenamiento cuya tabla de particiones queremos conocer. La información se cargará en la dirección 0x9000. Luego, seteamos el registro **es** (extra segment) en esta dirección, y trabajando con los corrimientos mostrados para las entradas de cada partición, empezamos a interpretar cada una de ellas.

- Opción 2: Mostrar los parámetros de la unidad
  Nuevamente utilizaremos la interrupción 0x13, con otro código de función, que se encargará de leer los datos precisados. En este caso, los valores vuelven en registros y por lo tanto no necesitamos guardar nada en memoria.

- Opción 3: Modificar ID de la tercer partición
  Una vez que hayamos leído la tabla de particiones, esta opción nos permitirá modificar el filesystem de la tercer entrada entre alguna de las opciones que se mostrarán en pantalla. Para realizarlo, utilizaremos la misma interrupción, con el código de función correspondiente a escribir en disco.

3.2 Proceso de compilado

Escribimos dos scripts de bash con los comandos correspondientes, con el fin de facilitar el proceso de compilado y ejecución. Utilizaremos **make.sh** para compilar nuestros códigos de assembler y generar la imagen correspondiente. El compilador elegido es **NASM** y la imagen será creada con la herramienta **dd** donde los binarios estarán en sectores contiguos (los tres primeros sectores de la unidad).
Para la ejecución usamos **run.sh**, que ejecuta una máquina virtual (generada por QEMU) con los siguientes parámetros: como floppy tomará la imagen creada con make.sh, como disco rígido tomará una imagen que contiene una tabla de particiones cuyos datos conocemos y el orden de booteo comienza buscando en la unidad floppy.

4 Bibliografía

Writing a Simple Operating System - from scratch, Nick Blundell
http://wiki.osdev.org/
http://stackoverflow.com/
http://wikipedia.org/