Arquitectura de Sistemas

Práctica 2: Sector de arranque y controlador gráfico

Gustavo Romero López

Updated: 14 de marzo de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

Objetivos

- Creación de un **sector de arranque** con un **controlador de vídeo básico**.
- Partiendo desde el más sencillo posible iremos añadiendo capacidades (pocas).
- Utilizaremos **as**, **ld** y **qemu**.
- Proceso incremental...
  - El más **simple** posible.
  - Añadiremos **eficiencia energética**.
  - Imprimir un mensaje a través de la **BIOS**.
  - Imprimir un mensaje directamente en la **memoria de video**.

El proceso de arranque de un PC (1)

Nada más encender un ordenador se ejecuta un programa especial denominado sistema básico de entrada/salida o **BIOS**, del inglés "*Basic Input/Output System*".

Este programa es almacenado en una memoria no volátil para evitar que se borre al apagar el ordenador. La función de la BIOS es inicializar todo el hardware del ordenador, desde los registros de la CPU hasta los contenidos de la memoria, pasando por los controladores de dispositivos.

Una vez hecho esto, una parte del mismo llamada **gestor de arranque**, busca el SO, lo carga en memoria y lo ejecuta. Para esto debe localizar donde se encuentra el núcleo del SO.

El proceso de arranque de un PC (2)

La BIOS busca la información de arranque en el registro principal de arranque o MBR, del inglés "*Master Boot Record*", o simplemente **sector de arranque**.

Es un sector de **512 bytes** al principio del disco duro que contiene una secuencia de instrucciones necesaria para cargar el sistema operativo y una tabla donde están definidas las particiones del disco duro. También el primer sector de cada partición, en la arquitectura del PC, tiene la misión de arrancar el sistema operativo.

Normalmente el MBR lo único que hace es ejecutar el sector de arranque de la partición marcada como arrancable.

Objetivos

- Nuestra misión en estas prácticas será escribir un **sector de arranque**.
- Necesitaremos:
  - un ensamblador: **gas**.
  - un enlazador: **ld**.
  - una máquina virtual: **qemu**, que emula un PC y así no tener que reiniciar el ordenador cada vez que queramos probar un nuevo sector de arranque, y de camino nos ahorra problemas al no tener que modificar el MBR de nuestro ordenador.

Detalles importantes (1)

- Al arrancar el PC funciona en **modo real** y sólo acepta código de **16 bits**. ¿Cómo se le indica al gas que genere código de 16 bits? $\Longrightarrow$ `.code16`
- La BIOS carga el sector de arranque en la dirección **0x7C00**. ¿Cómo hacemos que un programa se ejecute en la dirección 0x7C00? $\Longrightarrow$ el ensamblador no puede hacerlo sólo y necesitamos el enlazador $\Longrightarrow$ `-Ttext 0x7C00`
- Para empezar nos conformaremos con que nuestro sector de arranque no haga nada y deje **colgado** el ordenador. ¿Cómo conseguir esto? $\Longrightarrow$ bucle infinito: "`jmp .`" o "`aqui: jmp aqui`"

Detalles importantes (2)

- La BIOS reconocerá el sector de arranque si ocupa `512` bytes y está correctamente "**firmado**". Esto quiere decir que su última palabra debe ser `0xAA55`. ¿Cómo conseguir esto? $\Longrightarrow$ `.org 510 .word 0xAA55`
- Recientes versiones de las herramientas GNU añaden la sección ".note.gnu.property". Podemos eliminarla con `strip` aun a riesgo de perder información necesaria.

```
strip --keep-symbol=_start
      --remove-section=.note.gnu.property $@
```

- El ejecutable debe tener **formato binario**. ¿Cómo se le indica a as que use dicho formato? $\Longrightarrow$ El ensamblador no puede hacerlo directamente y requiere la ayuda del enlazador: $\Longrightarrow$ `--oformat binary`

Detalles importantes (3)

- Todo lo anterior podemos hacerlo con la ayuda del enlazador mediante un fichero de configuración:

linker.ld

```
ENTRY(_start)
OUTPUT_FORMAT("binary")
SECTIONS
{
    . = 0x7C00;
    .text : { *(.text); }
    /DISCARD/ : { *(.note.gnu.property); }
}
```

makefile I

```
.ONESHELL:

ASM = $(wildcard *.s)
ATT = $(BIN:.bin=.att)
BIN = $(OBJ:.o=.bin)
OBJ = $(ASM:.s=.o)

all: curses | $(ATT)

clean: kill
    -rm -fv $(ATT) $(BIN) $(OBJ) core.* *~

curses: $(BIN)
    gnome-terminal --geometry 80x25 -- qemu-system-i386 -display
    ↪  curses -drive file=$<,format=raw -k es -serial mon:stdio &
```

makefile II

```
debug: $(BIN) | kill
    qemu-system-i386 -drive file=$<,format=raw -k es -s -S &>
    ↪  /dev/null &
    pushd ..
    gdb -ix gdb_init_real_mode.txt \
        -ex 'set tdesc filename target.xml' \
        -ex 'target remote localhost:1234' \
        -ex 'hbr *0x7c00' -ex 'hbr *0x7c20'
    popd

kill:
    -killall -q qemu-system-i386 || true

qemu: $(BIN)
    qemu-system-i386 -drive file=$<,format=raw &> /dev/null &
```

makefile III

```
%.att: %.bin
    objdump -D -b binary -mi8086 -Maddr16,data16 -z $< > $@

%.bin: %.o
    ld -T../linker.ld $< -o $@

.PHONY: all clean debug kill qemu
```

El más sencillo: boot.s

```
.code16                 # código de 16 bits

.text                   # sección de código
    .globl _start       # punto de entrada

_start:
    jmp _start          # bucle infinito

    .org 510            # posición 510
    .word 0xAA55        # firma
```

Sugerencias

- Podemos comprobar que el sector de arranque va a ser reconocido mediante: "`file sector_de_arranque`".
- Podemos probar el sector de arranque escribiéndolo en un disco de 3,5" mediante la orden
  "`dd if=sector_de_arranque of=/dev/fda`" (como root).
- Podemos depurar el sector de arranque de forma remota a través del puerto serie:
  - lanzar `qemu` con la opción "`-s`" (`-gdb tcp::1234`).
  - ejecutar `gdb` y ejecutar: "`target remote 127.0.0.1:1234`"
- Podemos ver el código binario con algún visualizador hexadecimal como `hexdump` o `ghex`.
- Para desensamblar el sector de arranque necesitaremos...
  - `objdump -b binary -D -m i8086 sector_de_arranque`

Sector de arranque energéticamente **eficiente**: energy.s

- Modificar el sector de arranque anterior de forma que en lugar de ejecutar un bucle infinito deshabilite las interrupciones ("`cli`") y después detenga el procesador ("`hlt`").
- El fichero makefile proporcionado sólo funciona cuando hay un **único** fichero ensamblador por directorio, así que cree un directorio nuevo con `mkdir` para el nuevo sector de arranque.
- Comprobar que ahora `qemu` en efecto no utiliza el 100 % del tiempo del procesador mediante la orden **top**.

Imprimir un mensaje a través de la BIOS

- Vamos a modificar el sector de arranque anterior de forma que imprima un mensaje a través de la interrupción 0x10 de la BIOS. Dicha función requiere:
  - ah = 0x0e
  - al = carácter que deseamos imprimir
  - bh = 0
  - bl = color del carácter y del fondo
- Lista de colores: 0x00 negro, 0x01 azul, 0x02 verde, 0x03 cian, 0x04 rojo, 0x05 magenta, 0x06 marrón, 0x07 gris, 0x08 gris oscuro, 0x09 azul brillante, 0x0a verde brillante, 0x0b cian brillante, 0x0c rosa, 0x0d magenta brillante, 0x0e amarillo, 0x0f blanco.

Imprimir un mensaje a través de la memoria de video

- La memoria de video comienza en la **posición 0xb8000**.
- Está formada por **2000 palabras** = 25 filas por 80 columnas.
- Cada palabra está compuesta por un byte de **carácter** y otro de **color**.
- Los códigos de color son los mismos que hemos visto para la BIOS.

Limitaciones del ensamblador 8086 en modo real

- Registros de 16 bits:
  - Propósito general: AX, BX, CX, DX, DI, SI, SP, BP
  - Segmento: CS, DS, ES, SS
- Direccionamiento de 20 bits: DS:SI = DS × 16 + SI
- Modos de direccionamiento:

$$\begin{matrix}\texttt{CS}:\\ \texttt{DS}:\\ \texttt{SS}:\\ \texttt{ES}:\end{matrix}\left(\begin{bmatrix}\texttt{BX}\\ \texttt{BP}\end{bmatrix}+\begin{bmatrix}\texttt{SI}\\ \texttt{DI}\end{bmatrix}\right)+\text{displacement}$$

- Muchas limitaciones y peculiaridades en intrucciones:
  - Los registros de segmento sólo pueden cargarse desde AX
  - CX contador: `loop` y `rep cmps`/`lods`/`movs`/`scas`/`stos`
  - Parejas segmento:índice preestablecidas: DS:SI y ES:DI