512 bytes to boot

Pierre Marsais
`pim@lse.epita.fr`

December 12, 2017

Context

- K is a toy kernel used for ING1 kernel course
- We want to run it with as few memory as possible
- It is installed on a CD-ROM, with GRUB2 as bootloader

But...

- Under 3MB, GRUB refuses to run
- GRUB is bloatware

How to fix it ?

- Patch/tune GRUB ?
- Find another bootloader ?

How to fix it ?

- Patch/tune GRUB ? No
- Find another bootloader ?

How to fix it ?

- Patch/tune GRUB ? No
- Find another bootloader ? No

How to fix it ?

- Patch/tune GRUB ? No
- Find another bootloader ? No
- Write another bootloader ?

How to fix it ?

- Patch/tune GRUB ? No
- Find another bootloader ? No
- Write another bootloader ? Yes

Specifications

- We want to boot on BIOS powered machines
- We want to support the Multiboot specification (as much as possible)
- We need to load an ELF
- We need to boot from an CD-ROM/ISO filesystem
- We want to stay under 512 bytes

ELI5 BIOS boot process

- BIOS loads the first sector/512 bytes from the device at `0x7C00`, in real mode (16 bit), and jumps at this address
- bootloader does some magic
- We are in protected mode (32bit), mutliboot informations are loaded, and the kernel is loaded in memory from the filesystem

Multiboot

- Defines how a bootloader should boot a kernel
- Tells the kernel about the memory layout
- Kernel command line

Original plan

- Bootstrap a 32bit C runtime (easy)
- Write C (very easy)
- Wrap calls to BIOS in asm (easy-ish)

Original plan

- Bootstrap a 32bit C runtime (easy)
- Write C (very easy)
- Wrap calls to BIOS in asm (easy-ish)
- "It should take 1 day to run K" – me, about a month earlier

32bit runtime

- Set some segment selector
- Set a bit (`PE`) in a register (`cr0`)
- A20 ?
- That’s all

Calling BIOS

- BIOS has a mechanism to do some “high-level” stuff
  - Called with `int` instructions
  - Parameters are in registers
- Similar to x86 syscalls
- However, we NEED to be in 16 bit mode in order to call BIOS interrupts
- Multiple interrupts families
  - `int 0x13` for disk services
  - `int 0x15` for detecting memory

Back to the 16bit mode

- 2 ways (that I know of):
  - vm86
  - Really going back to 16 bit mode

How do we call any interrupt ?

- There is no instructions such as `int %eax`, the interrupt number must be hard-coded in the instruction
- `int XX` encoding is `cd XX`
- Should we do self-modifying code ?
- Linux does it, so why not

Reading from disc

- `int $0x13`, when `ah = 2` reads a sector (512 bytes)
- Use CHS (Cylinder, Head, Sector) addressing
- Simple formula: `lba = (C * nlh + H) * nls + S - 1`
  - `C`: Cylinder/Track, `H`: Head, `S`: Sector
  - `nlh`: head count, `nls`: sector count

Now, the easy part: C

- let's start easy: print a word in hexadecimal (about 20sloc)
- What could go wrong ?

Now, the easy part: C

- let's start easy: print a word in hexadecimal (about 20sloc)
- What could go wrong ?
- `collect2: error: ld returned 1 exit status`

Back to basics

- Forget C, I’m better than the compiler
- Let’s go full x86

Loading an ELF

- Check the magic
- Iterate over program headers
- `memcpy` + `memset` when `PT_LOAD`

Trivia time!

Trivia time!

Trivia time!

Trivia time!

- ‘memcpy‘ and ‘memset‘ in x86 ?

Fun with x86

	C	x86

memcpy	`for(u32 i = 0; i < sze; i++)`
	`.       dst[i] = src[i];`
		`rep movsb`

memset	`for(u32 i = 0; i < sze; i++)`
	`.       dst[i] = c;`
		`rep stosb`

- And they said that asm was verbose ?

Booting a CD-ROM

- Hard drives/Floppy disks are booted by copying the first sector
- ISO 9660 doesn't specify how to boot from a CD-ROM
- El Torito is an ISO 9660 extension for bootable CD-ROM

El Torito

- 2 operations modes:
  - Floppy disk/hard disk emulation
  - No emulation
- With emulation, an ISO driver isn't needed in the bootloader
- However, we need the Kernel on the "floppy"
- `xorriso` supports making bootable CD-ROM

Going FAT

- Currently we load the n first sectors after the bootloader
- Using a filesystem allows us to:
  - Know the kernel size at runtime
  - Have a fs when booting from the floppy
  - Be able to mount it
  - Have more than one file accessible to the bootloader (modules...)
  - Some tools complains when a disk image is a not FAT filesystem
- FAT16 seems like a good fit
- However, the first 62 bytes are taken by the FAT16 boot record

TODO

- Support FAT table in FAT driver
- Improve Multiboot support
- Support larger kernels ?
- Subdirectories in FAT16 driver ?
- Support El Torito no emulation mode ?

Optimizations ideas

- FAT16 driver in real mode
- Remove the BIOS call wrapper
- Optimize instructions here and there
- Single entry GDT

Layout

```
eb3c906d 6b66732e 66617400 02010100 01e00040 0bf00c00 12000200 00000000
00000000 00002911 8f702c4e 4f204e41 4d452020 20204641 54313620 2020fae8
2501b310 e80d0100 00bcf07b 000081ec 08020000 8d5c2408 5331c9b5 7c8a4110
6698660f af411666 03410e50 48e87400 0000595d 8b7d1cf7 df8da43d 00020000
f7dfea89 7c000018 008b461a f6260d7c 89c6a111 7c8b1e0b 7cc1eb05 31d2f7f3
89e501f0 01c84848 4889ebea b07c0800 5389f9c1 e9094151 5350e827 00000058
405b81c3 00020000 59e2ec66 b8440ebd 10000000 e8490000 005ae89b 000000bc
f07b0000 ffe031c9 b57c668b 491831d2 66f7f188 c588d141 4131d288 ee80e601
d0edb801 020000bd 13000000 e8110000 00c366b8 420ebd10 000000e8 02000000
ebfe87eb 881d4f7d 000055b3 20e82400 0000ea39 7d000018 00fb0f20 c3664b0f
22c3ea47 7d000031 dbe80a00 665bcd00 e81400fa b3106653 30ff8edb 8ec38ee3
8eeb8ed3 665bc30f 0116bf7d 0f01e583 cd010f01 f55d6a08 55cb813a 7f454c46
0f858cff ffff31c9 668b4a2c 89d3035a 1cc1e105 837c0be0 01751b51 8b740be4
01d68b7c 0be88b44 0bf48b4c 0bf029c8 f3a491f3 aa59c1e9 05e2d68b 4218c327
00bd7d00 00ffff00 00009acf 00ffff00 000092cf 00ffff00 00009a0f 00ffff00
0000920f 00000000 00000000 00000000 00000000 00000000 00000000 000055aa
```

- ● File format/magic/FAT16
- ● 32bit runtime
- ● FAT16 driver
- ● ELF loader
- ● GDT
- ● Free space

Questions ?

https://github.com/pimzero/bootload