xv6©-rev10

(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

The Boot Block (elective)

Carmi Merimovich

Tel-Aviv Academic College

September 18, 2017

(elective)

- The boot block is composed of two source files `bootasm.S` and `bootmain.c`.
- The ROM loads the boot block to address `0000:7C00`.
- Then the ROM jumps to address `0000:7C00`.
- The boot block must begin with machine instructions. No headers and the like.
- The code must begin with the assembly module.
- Hence the order of the modules in the `ld` invocation is important!

bootblock MAKE (elective)

```
bootblock: bootasm.S bootmain.c
   $(CC) $(CFLAGS) -fno-pic -O -nostdinc -I. -c \
                                             bootmain.c
   $(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c \
                                             bootasm.S
   $(LD) $(LDFLAGS) -N -e start -Ttext 0x7C00 \
                    -o bootblock.o bootasm.o bootmain.o
   $(OBJDUMP) -S bootblock.o > bootblock.asm
   $(OBJCOPY) -S -O binary -j .text bootblock.o  \
                                              bootblock
   ./sign.pl bootblock
```

(elective)

- The purpose of the `bootblock` is to load the xv6 kernel into memory.
- This code is (mainly) written in C.
- Alas, the process begins in 16-bit mode, hence:
- A small assembly routine is used to in order to switch to 32-bit mode.

bootasm.S steps (elective)

There are 5 steps in the execution of the assembly code:

1. Establish 16 bit environment.
2. Handle the A20 issue
3. Switch to protected mode.
4. Establish the 32 bit environment.
5. Jump to the C code in bootmain.c

bootasm.S step 1 (elective)

Establish access to the first 64KB of RAM.

```
.code16                              # Assemble for 16-bit
.globl start
start:
  cli                                # BIOS enabled interru

  # Zero data segment registers DS, ES, and SS.
  xorw    %ax,%ax                    # Set %ax to zero
  movw    %ax,%ds                    # -> Data Segment
  movw    %ax,%es                    # -> Extra Segment
  movw    %ax,%ss                    # -> Stack Segment
```

bootasm.S step 2 (elective)

```
seta20.1:
  inb     $0x64,%al                    # Wait for not bus
  testb   $0x2,%al
  jnz     seta20.1

  movb    $0xd1,%al                    # 0xd1 -> port 0x6
  outb    %al,$0x64

seta20.2:
  inb     $0x64,%al                    # Wait for not bus
  testb   $0x2,%al
  jnz     seta20.2

  movb    $0xdf,%al                    # 0xdf -> port 0x6
  outb    %al,$0x60
```

bootasm.S step 3 (elective)

```
  lgdt      gdtdesc
  movl      %cr0, %eax
  orl       $CR0_PE, %eax
  movl      %eax, %cr0

  ljmp      $(SEG_KCODE<<3), $start32
  .code32
start32:
    .
gdt:
  SEG_NULLASM                                   # null seg
  SEG_ASM(STA_X|STA_R, 0x0, 0xffffffff)         # code seg
  SEG_ASM(STA_W, 0x0, 0xffffffff)               # data seg
gdtdesc:
  .word     (gdtdesc - gdt - 1)                 # sizeof(g
  .long     gdt                                 # address
```

Gate descriptor (elective)

```
struct segdesc {
  uint lim_15_0 : 16;    // Low bits of segment limit
  uint base_15_0 : 16;   // Low bits of segment base a
  uint base_23_16 : 8;   // Middle bits of segment bas
  uint type : 4;         // Segment type (see STS_ con
  uint s : 1;            // 0 = system, 1 = applicatio
  uint dpl : 2;          // Descriptor Privilege Level
  uint p : 1;            // Present
  uint lim_19_16 : 4;    // High bits of segment limit
  uint avl : 1;          // Unused (available for soft
  uint rsv1 : 1;         // Reserved
  uint db : 1;           // 0 = 16-bit segment, 1 = 32
  uint g : 1;            // Granularity: limit scaled
  uint base_31_24 : 8;   // High bits of segment base
};
```

(elective)

```
#define SEG_ASM(type,base,lim)                   \
   .word (((lim) >> 12) & 0xffff),   \
                              ((base) & 0xffff); \
   .byte (((base) >> 16) & 0xff), (0x90 | (type)), \
        (0xC0 | (((lim) >> 28) & 0xf)), \
       (((base) >> 24) & 0xff)

#define STA_X     0x8       // Executable segment
#define STA_E     0x4       // Expand down (non-exec
#define STA_C     0x4       // Conforming code segme
#define STA_W     0x2       // Writeable (non-execut
#define STA_R     0x2       // Readable (executable
#define STA_A     0x1       // Accessed

#define SEG_NULLASM              \
           .word 0, 0;                  \
            byte 0  0  0  0
```

bootasm.S step 4 (elective)

Setup selectors so we have access to the first 4GB of memory.

```
movw    $(SEG_KDATA<<3), %ax     # Our data segment
movw    %ax, %ds                 # -> DS: Data Segm
movw    %ax, %es                 # -> ES: Extra Seg
movw    %ax, %ss                 # -> SS: Stack Seg
movw    $0, %ax                  # Zero segments no
movw    %ax, %fs                 # -> FS
movw    %ax, %gs                 # -> GS
```

bootasm.S step 5 (elective)

Jump to the C code.

```
movl    $start,%esp
call    bootmain
```