

Bootloader

Monitores: Matheus Deodato <mdo2>
 Lucas Mendes <Imm3>
 Eneri D' Angelis <emd>

Slide por: Matheus Deodato <mdo2>

Bootloader, o que é?

- É o primeiro programa a ser carregado e executado;
- Possui a responsabilidade de organizar e levantar o sistema operacional;
- É executado em modo real;
- Possui limitação de tamanho (512 bytes);
- Baseado em interrupções da BIOS;
- Pode ser dividido em várias etapas (possibilitando usar mais que 512 bytes).

Projeto do Bootloader

O projeto consiste em desenvolver um bootloader de 2 estágios, que ao final, chame um kernel feito em assembly, tudo desenvolvido em modo real.

Projeto do Bootloader

Primeiro Estágio

A BIOS carrega o primeiro setor do disco (se ele tiver a assinatura de boot 0xAA55 no final do setor), no endereço linear de memória 0x7C00, e pula para esse endereço. Como um setor no disco geralmente tem 512 bytes, muitas vezes dividi-se o bootloader em mais de um estágio, de modo que a BIOS carregue o primeiro estágio, e este carregue o próximo estágio.

Resumo do primeiro estágio

Escrever um código que carregue o segundo estágio do bootloader do disco para a memória, escolhendo qualquer endereço livre da memória para carregar o mesmo.

Projeto do Bootloader

Segundo Estágio

Em um sistema real o segundo estágio de um bootloader geralmente carrega algumas estruturas que serão úteis ao kernel na memória, carrega o kernel na memória, configura o ambiente adequado ao kernel (passar para o modo protegido, por exemplo), e passa o controle para o kernel.

Projeto do Bootloader

O que será cobrado:

- Entender e saber explicar todo o processo de boot;
- Entender o funcionamento do código assembly;
- Entender o conceito de Segment:Offset;
- Entender o funcionamento e utilização dos registradores de segmento;
- Um kernel bem implementado (e criativo).

Resumo do primeiro estágio

Segundo Estágio a ser implementado:

Escrever um código que imprima na tela tarefas comuns do segundo estágio (mesmo que na prática elas não são implementadas), exemplo:

- Loading structures for the kernel...
- Setting up protected mode...
- Loading kernel in memory...
- Running kernel...

Esse código deve também carregar o kernel na memória e passar o controle para ele.

Projeto do Bootloader

Kernel

Em um S.O. moderno, o kernel possui um código gigantesco (10 milhões de linhas >>), e oferece várias funcionalidades, como:

- Gerenciamento de recursos (memória, processador, etc);
- Gerar uma camada de abstração para o software de usuário.

Resumo do Kernel

Kernel

Nessa parte vocês devem exercitar sua criatividade, escolhendo apresentações, movimentos ou jogos para essa tela de abertura. Qualquer trabalho extra que o grupo faça será levado em consideração na hora da avaliação.

OBS: É obrigatório o uso da interrupção 13h, modo vídeo VGA. Utilizem com bom senso, da forma que quiserem e com criatividade.

Memory Map

> 1 MB

0xFFFFF

HIGH MEMORY

0xEFFFF

BIOS

65536 b

0xA0000

Video Memory, Memory
Mapped I/O, etc

327679 b

0x9FC00

Extended
BIOS Data Area

1024 b

0x07E00

Free Memory

622080 b

0x07C00

Bootsector Area

512 b

0x00500

Free Memory

30464 b

0x00400

BIOS Data Area

256 b

0x00000

Interrupt Vector Table

1024 b

Boot 1

```
org 0x7c00
jmp 0x0000:start
start:
    xor ax, ax
    mov ds, ax
    mov es, ax
    mov ax, 0x50 ;0x50<<1 = 0x500 (início de boot2.asm)
    mov es, ax
    xor bx, bx ;posição = es<<1+bx
    jmp reset
reset:
    mov ah, 00h ;reseta o controlador de disco
    mov dl, 0 ;floppy disk
    int 13h
    jc reset ;se o acesso falhar, tenta novamente
    jmp load
```

```
load:
    mov ah, 02h ;lê um setor do disco
    mov al, 1 ;quantidade de setores ocupados pelo boot2
    mov ch, 0 ;track 0
    mov cl, 2 ;sector 2
    mov dh, 0 ;head 0
    mov dl, 0 ;drive 0
    int 13h

    jc load ;se o acesso falhar, tenta novamente

    jmp 0x500 ;pula para o setor de endereço
               ;0x500 (start do boot2)

times 510-($-$$) db 0 ;512 bytes
dw 0xaa55 ;assinatura
```

Boot 2

```
org 0x500
jmp 0x0000:start
start:
```

```
    xor ax, ax
    mov ds, ax
    mov es, ax
```

```
    mov ax, 0x7e0 ;0x7e0<<1 = 0x7e00 (início de kernel.asm)
    mov es, ax
    xor bx, bx    ;posição es<<1+bx
```

```
    jmp reset
```

```
reset:
```

```
    mov ah, 00h ;reseta o controlador de disco
    mov dl, 0   ;floppy disk
    int 13h
```

```
    jc reset    ;se o acesso falhar, tenta novamente
    jmp load
```

```
load:
```

```
    mov ah, 02h ;lê um setor do disco
    mov al, 20  ;quantidade de setores ocupados pelo kernel
    mov ch, 0   ;track 0
    mov cl, 3   ;sector 3
    mov dh, 0   ;head 0
    mov dl, 0   ;drive 0
    int 13h
```

```
    jc load     ;se o acesso falhar, tenta novamente
```

```
    jmp 0x7e00 ;pula para o setor de endereço 0x7e00 (start do
boot2)
```

Kernel

```
org 0x7e00  
jmp 0x0000:start
```

```
start:  
    xor ax, ax  
    mov ds, ax  
    mov es, ax
```

```
done:  
    jmp $
```

Links

<https://www.ibm.com/developerworks/linux/library/l-lpic1-v3-101-2/>

Materials de apoio:

<https://drive.google.com/file/d/0B4LizlLkBUcKb0pXbm5VWTBpUTA/view?usp=sharing>

https://drive.google.com/file/d/0B_lmKsSTCSfOcThwbzZiX01qUGs/view?usp=sharing