

操作系统 Lab1 实验报告

141220132 银琦 141220132@smail.nju.edu.cn

1. 保护模式下 32 位的寄存器的寻址空间就有 4GB, 那使用段选择子:偏移这样的寻址方式最大寻址空间能达到多大呢?给出计算的过程吗?

答: 其中 TI 用来指明全局描述符表 GDT 还是局部描述符表 LDT, RPL 表示请求特权级, 索引值为 13 位, 所以从这里看出, 在保护模式下最多可以表示 $2^{13}=8192$ 个段描述符, 而 TI 又分 GDT 和 LDT, 所以一共可以表示 $8192*2=16384$ 个段描述符, 每个段描述符可以指定一个具体的段信息, 所以一共可以表示 16384 个段。而图 1 看出, 段内偏移地址为 32 位值, 所以一个段最大可达 4GB, 这样 $16384*4GB=64TB$, 这就是所谓的 64TB 最大寻址能力, 也即逻辑地址/虚拟地址。

2. 将上面这段代码拷贝到我们的框架代码里后运行, 发现我们的屏幕输出了一行红色的"Hello world!", 但是此外我们还可能看到一些其他的文字, 有没有办法让屏幕中的其他内容消失呢?

```
mov    $3, %ax
```

```
int     $0x10
```

3. 实验步骤:

- 1) 将代码拷贝进框架, 输出"Hello, World!"。

2) 在 start.S 中定义 GDT 的数据结构，其中 asm.h 中已经定义了相关的宏。

3) 关中断

STI	;开中断, IF=1
CLI	;关中断, IF=0

4) 打开 A20 地址线。方法百度获得：（调用 BIOS 中断）

Enabling the A20 :

```
push ax
```

```
mov ax, 0x2401
```

```
int 0x15
```

```
pop ax
```

试想一下如果我们没有打开 A20 地址线，在保护模式下会出现什么问题？

A20 地址线并不是打开保护模式的关键，只是在保护模式下，不打开 A20 地址线，你将无法访问到所有的内存。

5) 加载 gdt 描述符

6) 置 cr0 的 PE 位为 1，不可以直接 `or $0x1, %cr0`，会报错：

```
start.S:14: Error: operand type mismatch for `or'
```

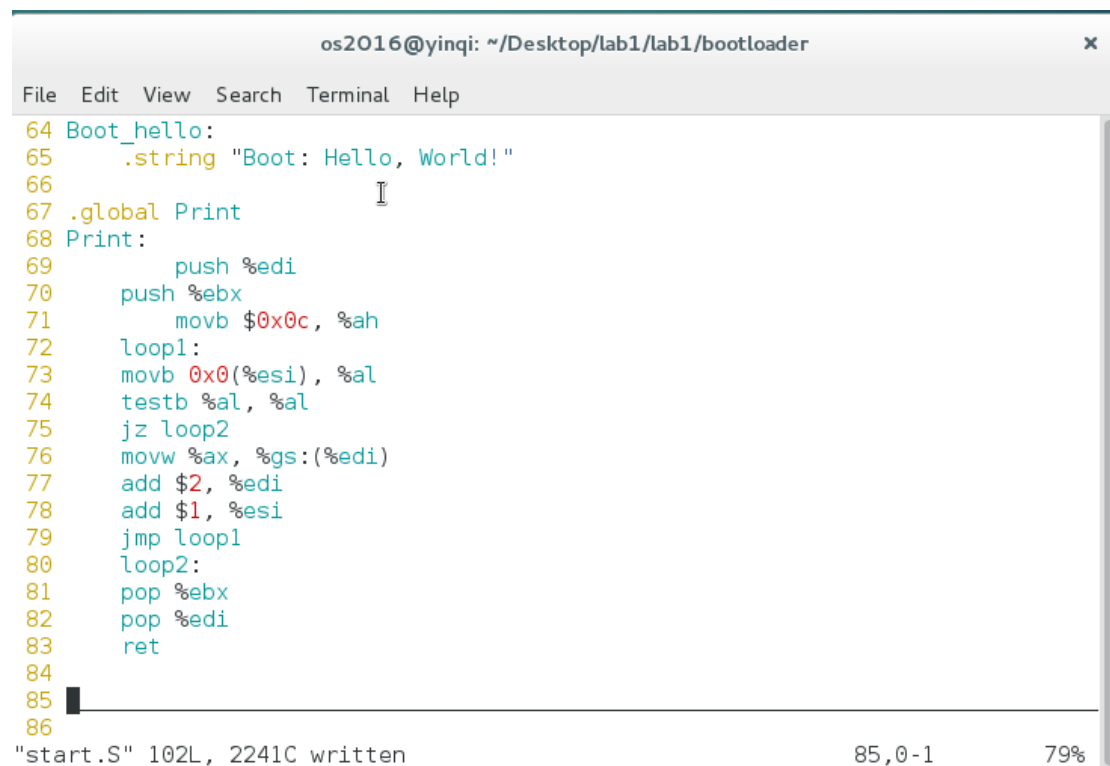
7) 跳转到保护模式：ljmp 的格式是：ljmp 段选择子，段内偏移

8) 加载磁盘

9) 写 print 函数，输出字符串。

4. 关键代码截图

1) 在 start.S 中的 print 函数



The screenshot shows a text editor window titled "os2016@yinqi: ~/Desktop/lab1/lab1/bootloader". The editor contains assembly code for a function named "Boot_hello:". The code defines a string "Boot: Hello, World!", sets up a global "Print" label, and implements a loop that prints the string character by character. The status bar at the bottom indicates "start.S" is 102 lines long, 2241 characters, and 79% of the file is shown.

```
64 Boot_hello:
65     .string "Boot: Hello, World!"
66
67     .global Print
68 Print:
69     push %edi
70     push %ebx
71     movb $0x0c, %ah
72 loop1:
73     movb 0x0(%esi), %al
74     testb %al, %al
75     jz loop2
76     movw %ax, %gs:(%edi)
77     add $2, %edi
78     add $1, %esi
79     jmp loop1
80 loop2:
81     pop %ebx
82     pop %edi
83     ret
84
85
86
```

"start.S" 102L, 2241C written 85,0-1 79%

2) 用 C 语言写的 myprint 函数

```
static void myprint(char *str, short* addr) {
// short* addr=0xb8000;
    int i;
    for(i=0;str[i]!='\0';i++){
        *addr=str[i]+0x0c00;
        addr++;
    }
}
```

3) 在 lib.h 中被 main 函数调用的 print 函数



The screenshot shows a text editor window titled "os2016@yinqi: ~/Desktop/lab1/lab1". The editor contains the implementation of the "print" function in lib.h. The function uses a while loop to print the string "Process:Hello world!" character by character. The status bar at the bottom shows a tilde symbol (~).

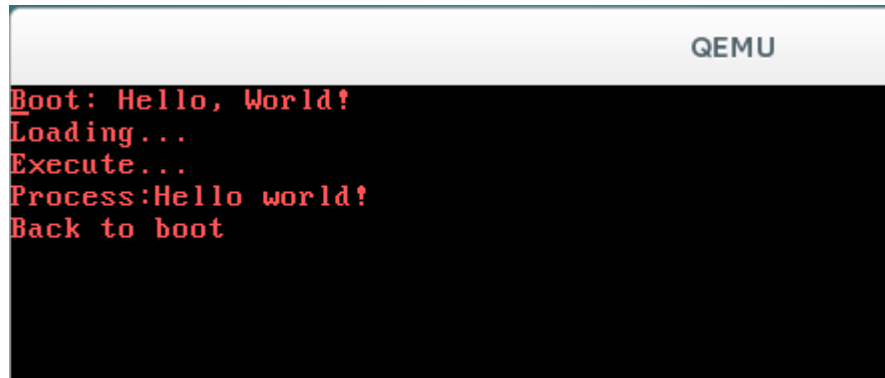
```
1 void print()
2 {
3     // while(1);
4     short* addr = 0xb81e0;
5     char str[] = "Process:Hello world!";
6     int i = 0;
7     for(i=0;str[i]!='\0';i++) {
8         *addr=str[i] + 0x0c00;
9         addr++;
10    }
11
12 }
```

~

4) 最终占用内存大小

```
OK: boot block is 509 bytes (max 510)
```

5) 运行成功截图



5. 一些细节

输出时，第一行的地址为 0xb8000，经过试验和计算，得第二行的地址为 0xb80a0，接下来依次为 0xb8140，0xb81e0，0xb8280。

输出字符串需要占很大的空间，所以要将代码精简，去掉多余的部分。

在函数前面加 `static` 后所占空间就大大减小（虽然不知道为什么，但是亲测有效）。

6. 心得体会

对我来说有三个难点，一是写 `print` 的汇编代码，如果另外写函数反汇编出来，则很容易超过 512，所以需要自己写；二是加载磁盘部分的代码不太会写，参考了上学期 `pa` 的框架代码....；三是对讲义的理解，一开始不知道怎么写 `print` 函数，后来在大神的帮助下理解了使用视频段的时候，就知道怎么写了。最终写完了对这块内容就有了

一个大概的结构与理解，对我帮助很大。