

## 操作系统 Lab2——系统调用

151220045 蒋雨霖

### 一. 实验目的:

本实验通过实现一个简单的应用程序，并在其中调用一个自定义实现的系统调用。实现系统调用库函数 `printf`，完善 `printf` 的格式化输出。

### 二. 实验思路

1. 首先仿照 LAB1 系统引导修改 `start.S` 文件，从实模式进入保护模式，与 LAB1 不同的是不用初始化 `GS` 寄存器，以及删掉 `GDT` 表中的视频段。
2. 之后修改 `bootldr` 中的 `boot.c`，仿照 PA 中 `loader` 利用调用 `elf` 实现加载内核至内存。跳转至内核 `kernel` 继续执行。
3. 内核初始化 `IDT` (`Interrupt Descriptor Table`，中断描述符表)：即中断号 `0xd` 和 `0x80` 其余中断号未加(不加可过)。初始化 `GDT`。然后在 `kvm.c` 中初始化 `tss`。
4. 内核加载用户程序至内存：实现函数 `loadUmain`，代码格式类似于实现 `boot.c` 的方法，从 201 扇区开始加载。需要注意的是需要判断 `ProgramHeader *ph`，`ph->type==1` 才进行 `memcpy` 和 `memset` 操作。对内核堆栈进行设置。之后在 `enterUserSpace` 函数中通过 `iret` 切换至用户空间：需 `push eflags`、`push cs`，最后 `push` 入口地址，执行用户程序。
5. 最后实现 `syscall.c` 中的 `syscall` 函数和 `printf` 函数格式化。

`Syscall` 函数中利用内嵌汇编保存 6 个参数至通用寄存器。

而实现 `prtinf` 格式化利用到了头文件 `stdarg.h`，通过对 `format` 的分析并利用 `switch case` 语句，将参数 `d`，`s`，`x`，`c` 情况下需要打印的内容存入一个总的数组当中，最后调用 `syscall` 实现打印内容到显存上。输出数组当中内容。注意由于输出屏幕大小的限制，需加入 '`\n`' 的判断，判断何时换行，以保证打印在正确的位置。

### 三. 实验效果截图:

```
QEMU
printf test begin...ntu-1.8.2-ubuntu1)
the answer should be:
#####
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. ~!@#/(^&*())_+'1234567890-=..... Now I will test your printf: 1 + 1 =
2, 123 * 456 = 56088
0, -1, -2147483648, -1412505855, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
#####
your answer:
=====
Hello, welcome ro OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. ~!@#/(^&*())_+'1234567890-=..... Now I will test your printf: 1 + 1 =
2, 123 * 456 = 56088
0, -1, -2147483648, -1412505855, -32768, 102030
0, FFFFFFFF, 80000000, ABCEDF01, FFFF8000, 18E8E
=====
Test end!!! Good luck!!!
```

#### 四. 实验心得及建议:

这次实验相比第一次复杂了许多,心得:多参考 PA 中的内容,来实现 lab2,深刻理解整个系统调用的过程。

建议:应该适当在在线网页及课程中多给出些提示,以及相对于 LAB1 的三周,LAB2 两周时间显得过短。