

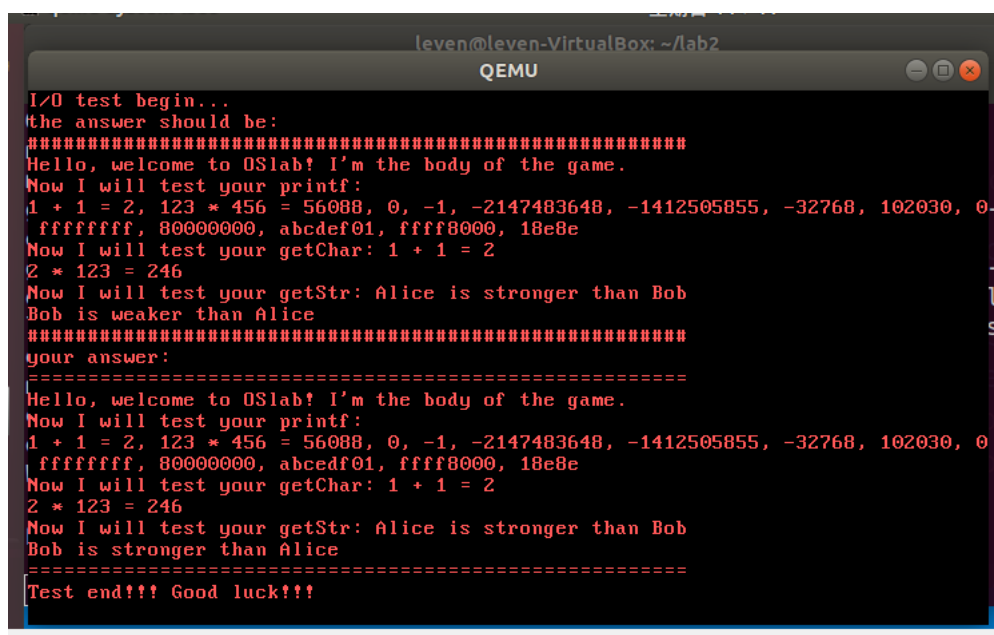
Lab2 实验报告

南京大学 人工智能学院

191300036 刘晔闻 450008668@qq.com

实验进度：我完成了所有内容（完整实现三个系统调用库函数 printf, getChar, getStr）。

实验结果：



```
leven@leven-VirtualBox: ~/lab2
QEMU

I/O test begin...
the answer should be:
#####
Hello, welcome to OSlab! I'm the body of the game.
Now I will test your printf:
1 + 1 = 2, 123 * 456 = 56088, 0, -1, -2147483648, -1412505855, -32768, 102030, 0-
ffffff, 80000000, abcdef01, ffff8000, 18e8e
Now I will test your getChar: 1 + 1 = 2
2 * 123 = 246
Now I will test your getStr: Alice is stronger than Bob
Bob is weaker than Alice
#####
your answer:
=====
Hello, welcome to OSlab! I'm the body of the game.
Now I will test your printf:
1 + 1 = 2, 123 * 456 = 56088, 0, -1, -2147483648, -1412505855, -32768, 102030, 0-
ffffff, 80000000, abcdef01, ffff8000, 18e8e
Now I will test your getChar: 1 + 1 = 2
2 * 123 = 246
Now I will test your getStr: Alice is stronger than Bob
Bob is stronger than Alice
=====
Test end!!! Good luck!!!
```

其中，通过 getChar 输入 “2”，通过 getStr 输入 “Bob”。

实验过程：

1. bootloader/boot.c: 填写 kMainEntry、phoff、offset
2. bootloader/start.s: 设 esp 寄存器
3. kernel/kernel/dolrq.s: 将 irqKeyboard 的中断向量号压入栈
4. kernel/kernel/idt.c: 初始化 interrupt gate 和 trap gate 和 IDT 表

5. kernel/kernel/irqHandle.c:

- (1) 在 irqHandle 中填写中断调用, 要注意在 -1 情况下直接 break, 否则会陷入 default 无法打印。
- (2) 填写 keyboardHandle (好像输入样例结果时用不到), 退格即是到前一格 (displayCol), 并更新光标; 回车即是到下一行 (displayRow) 的第一格 (displayCol), 并更新光标; 正常字符即是打印字符 (使用指南中给出的代码), 并更新光标 (下一格)。

以下这段代码可以将字符 `character` 显示在屏幕的 `displayRow` 行 `displayCol` 列

```
data = character | (0x0c << 8);  
pos = (80*displayRow+displayCol)*2;  
asm volatile("movw %0, (%1)":"r"(data), "r"(pos+0xb8000));
```

- (3) syscallprint 中完成光标维护和打印显存, 分为换行 (\n\r), 正常情况和超过 80*25 的屏幕大小 (要滚动屏幕) 三种情况, 主要是对 displayRow, displayCol 的处理。
- (4) 实现 getChar, 末尾字符通过 eax 寄存器传递 (tf->eax)。按照示例, 要求输入 2, 然后换行, 最后再打印一个 2, 并且不能影响到之后的输入 (还有 getStr), 所以要清空缓冲区。

```
uint32_t next_code=0; //qing kong huan chong qu  
while (next_code!=code+0x80)  
|   next_code=getKeyCode();
```

- (5) 实现 getStr (可以参考 syscallprint 和 getChar 的实现), 要求输入 Bob, 换行, 再打印 Bob。因为不止一个字符, 所以运用循环来输入并打印, 将循环条件设为回车键。还要求大小写, 所以设定一个变量 caoslock 判断是否应该大写。又因为 IO 速度太快, 导致输入的按键过多, 寄存器空间有限 (出现 unsupported size for integer register), 所以要设定不可打印的按键 (如 shift), 用静态数组 unable 存放 (只要去除一部分即可)。同样, 这里也要清空缓冲区。(多个字符, 要多循环) 关于清空缓冲区, 我的设想是将下一个定为非字符的即可 (大于 0x80)

```
uint32_t next_code=getKeyCode(); //qing kong huan chong qu
while (next_code!=code+0x80)
|   next_code=getKeyCode();
code=getKeyCode();
while (code==next_code)
|   code=getKeyCode();
```

6. kernel/kernel/kvm.c: 参照 bootloader 加载内核的方式加载 loadUMain。其中, 注意读磁盘函数 (readsect) 参数中的偏移量要比 bootloader 中加 200。(bootloader 中加载循环了 200 次)
7. kernel/main.c: 初始化
8. lib/syscall.c: 填写库函数 getChar, getStr 和 printf。其

中, getChar 和 getStr 使用文件中的 syscall() 函数, 填变量即可。而 printf 函数, 在 % 后判断是 %d, %x, %s, %c 的哪一种, 然后调用 dec2Str (%d), hex2Str (%x) 和 str2Str (%s) 三个函数, 要注意每次栈中的位置都要加 4, 即变量 paraList+=4, 并且索引不能超过最大值也不能小于 0 (要时刻更新)。

state 变量用处不大, 可以省去。

```
int i=0; // format index
char buffer[MAX_BUFFER_SIZE];
int count=0; // buffer index
int index=0; // parameter index
```

思考题:

1. ring3 的堆栈在哪里: ring3 的堆栈就是程序默认的堆栈, 所有 TSS 中没有 ring3 的堆栈信息。
2. 保存寄存器的旧值: 会产生不可恢复的错误。执行中断处理程序时可能会覆写寄存器的值。