oslab1 实验报告

姚荣春 151110079

1. gcc 版本及联系方式

Ubuntu 4.9.3

15895873878@163.com

2. 实验心得

实验非常有趣，收获不少。以下是几个有趣的经历。

(1)在写 mbr 的时候，mbr 的大小超过了 512B。后来发现使用-O1 优化就能够使大小在 512B 以内。

(2)在写 printk 的时候，没有考虑到 int 负数比正数多一个的情况，最后和测试样例不一样才发现。之前遇到负数直接取反为正数加一个符号后按位取模输出。但是这样处理 0x10000000 的时候，会输出乱码。因为负数模 10 是负数。解决：如果是负数，用 unsigned int 来放其转化后的正数。这样就不会出现乱码。

(3)一开始写的贪吃蛇，蛇一下能跳 2-3 步，十分的不规律。但是加上输出蛇位置的调试信息后就能正常的一下走一步。结果发现是实现方法过于简陋导致的，如图：

```
while(1)
{
        game_init();
        while(1)
        {
                direction=last_key_code();
                if(get_time()%200==0)
                {
                init();
                asm volatile("cli");
                win_or_lose=do_move();
```

时钟的频率是 1000，没有打印调试信息的时候 1 毫秒足够刷屏好几次，因此会造成奇怪现象。解决：暴力的让时间加一，如下图：图中的 time_pop 函数强制把游戏时间+1，虽然这样很不好，但是在 lab1 中没有内核，我觉得还是一个不错的权宜之计。

```
while(1)
{
        direction=last_key_code();
        if(get_time()%200==0)
        {
        init();
        asm volatile("cli");
        win_or_lose=do_move();
        if(win_or_lose!=1) break;// game over o
        asm volatile("sti");
        draw_string(game_name,0,0,4);
        draw_mark();
        draw_whole_snake();
        display_all();
        time_pop();
        }
}
```

3.游戏设计思路

（1）使用一个 last_key_code 来储存蛇移动方向，有趣的是，要注意当前方向相反的按键要无效处理，比如，last_key_code 是 a 的时候，按 d 无效。

（2）使用一个结构体数组来储存蛇的每一个节点的位置。

（3）使用伪随机来产生食物的位置，只要食物刷新出来的位置被占用就重新随机生成。蛇吃掉食物后生成新的节点，每个节点颜色后推。

（4）每个循环 do_move 一次来刷新蛇的位置。do_move 函数结尾判断是否吃自己或者撞墙。游戏结束，继续，或者胜利会使 do_move 返回不同的值。主循环通过判断 do_move 的返回值来决定是否跳出

循环。

4.游戏控制介绍

awsd 控制蛇移动的方向。蛇撞边界或者撞到自己算输。右上角累计的积分到 50 就算获胜。game over 之后按 r 重新开始。

5.makefile 使用介绍

make run 编译运行

make clean 清除.o 等文件

make run 之后运行 terminal.sh(需要加权限)连接 qemu