Programming Assignment 3

黄奕诚161220049

October 28, 2017

1 实验进度

完成所有内容，通过所有测试，成功运行仙剑奇侠传1.

2 必答题：文件读写的具体过程

2.1 游戏存档的读取

不妨按照从底层(系统架构)向顶层(用户端)的顺序，一步步分析仙剑奇侠传游戏存档读取的机理.

首先，nemu是一个具有cpu、存储器功能的模拟器，它是一个简单的冯诺依曼计算机系统，可以对于机器码进行译码、执行.

```
while (1) {
  从EIP指示的存储器位置取出指令;
  执行指令;
  更新EIP;
}
```

当接受到有关于游戏存档的机器指令时，nemu便会相应地译码、执行.那么它如何与操作系统联结起来，实现预期的功能呢？这时要考虑到nanos-lite与nemu之间的一个抽象层：AM.AM建立在计算机系统nemu之上，它封装了nemu的功能，并为操作系统提供了好的接口.游戏存档相当于一个异常(系统调用)，涉及到了am中的asye、trm功能，如下列的两个异常处理函数：

_RegSet* irq_handle(_RegSet *tf);
void _asye_init(_RegSet*(*h)(_Event, _RegSet*));

而真正地对“游戏存档”的处理是在操作系统层面上的.讲义中提到在navy-apps/apps/pal/src/global/global.c 的PAL_LoadGame() 中通过fread() 读取游戏存档，它实则是用户层面的一个请求，向操作系统发出了读取存档的需求，此时操作系统已经具备了文件系统，它可以在先前储存游戏进度的相应文件及

其偏移位置读出需要的存档，并进行加载，涉及到以下函数：

int fs_open(const char*, int, int);
ssize_t fs_read(int fd, void *buf, size_t len);

此时便能够完成存档的读取操作。
总得来说，仙剑奇侠传的源代码产生需求，并发送到的库函数libos，库函数libos将系统调用的指令发送给操作系统Nanos-lite，操作系统通过调用AM的接口，即建立在NEMU之上对需求(如文件读取)进行处理.

2.2 游戏屏幕的更新

总体工作原理与前一题“游戏存档读取”基本相同，但游戏屏幕的更新涉及到了nanos-lite中的下列函数：

void fb_write(const void *buf, off_t offset, size_t len);

它将新的屏幕内容写入fb，由此使得屏幕完成更新.首先hal.c中的函数redraw()调用了NDL_DrawRect，而这个NDL嫁接在我们自己实现的nanos-lite操作系统之上，于是便把任务交给了操作系统，得以进行下去.

3 问题&思索&心路历程

pa3-1

首先遇到的一个问题是：如何将门操作符中的offset域组合成目标地址.起初不是很理解GateDesc的结构，经过思索和对讲义的反复阅读，知道目标地址是由两部分构成，而每一部分都是由起始地址加上与NO有关的值构成，不需要通过GateDesc，仅需要掩码操作就能完成，于是经过尝试完成了这一部分。然而，我运行dummy时发生了地址溢出，通过log方法与diff-test缩小了debug范围，仍然但一直无法解决.后来无意中发现iret的执行函数忘记pop出先前push的内容(大概是脑抽了)，一改便成功达到效果了.可见写代码确实需要在头脑清醒的时候，而且debug时要考虑到距离上次成功运行所改写的所有函数，不能忽略.

pa3-2

这一阶段完成得相当轻松，然而却隐藏着最大的隐患，以至于pa3-3在运行仙剑时的bug调了大半天，后面会提到.

pa3-3

这部分的VGA和事件处理也很简单，在通过两个测试，运行仙剑奇侠传时，突然蹦出了一个

Assertion failed: screen-¿pitch == W, file src/hal/hal.c, line 204

nemu: HIT BAD TRAP at eip = 0x00100032

我首先去看了hal.c文件的assert位置，经过输出发现那里的screen-¿pitch的值为0而不是预期的320.我思考：是哪个函数改变了pitch的值？于是我在目录中查找了screen− >pitch的调用位置，并找到了好几处.其中初始化时它们都是320，在某些函数中被改变，然而改变的过程很晦涩，并不能看出是否是我的实现导致了这个问题.我首先作死地开了diff-test跑仙剑，当电脑滚烫地跑了十几分钟后，啥diff都没报，仍然Hit bad trap.此时我心情复杂(如果报了diff，那就找到了问题在pa2中，如果没报，虽然能确定问题不在pa2，可问题的位置就很迷了).随后，我感觉到screen-¿pitch很可能与我文件系统中读取屏幕大小与更新屏幕函数有关，但是改了很久代码仍然无法解决，在迷惘之际，我找到了已经跑出仙剑的hyf同学，他认真地看我代码看了很久，给出了一些改进建议，然而.....我仍然没有跑出仙剑.然后我跟他暴力地对比代码，仍然无法发现文件系统中的bug.....惆怅之余，我想了想还有什么地方没有查，噫，navy-apps用户层的那个nanos.c我好像还没看？看了一眼，这破堆区管理，我咋返回了一个更新后的end?????为什么不返回旧值?????改了一下，仙剑瞬间跑出.我：?????

4 蓝框思考题

4.1 对比异常与函数调用

异常处理需要保存的值：eax,ecx,edx,ebx,esp,ebp,esi,edi,eflags,cs,eip
函数调用需要保存的值：表示函数实参的相应寄存器、ebp、返回地址
可能是因为触发异常后，eip所指向的进行异常处理的指令在内核进行，对原来的各个通用寄存器及标志寄存器都有很明显的改变，所以全部要push进行保存，以免原先的数据丢失.而函数调用只涉及到少数几个与参数有关的寄存器，故不需要保存很多无关的寄存器。

4.2 不再神秘的秘籍

对于第2点，“不断使用乾坤一掷(钱必须多于五千文)用到财产低于五千文, 钱会暴增到上限, 如此一来就有用不完的钱了”，可能使用乾坤一掷的时候财产的下限默认为五千文，当少于五千文会溢出，如此变成了上限值.
对于1、3点我并没有想通.....