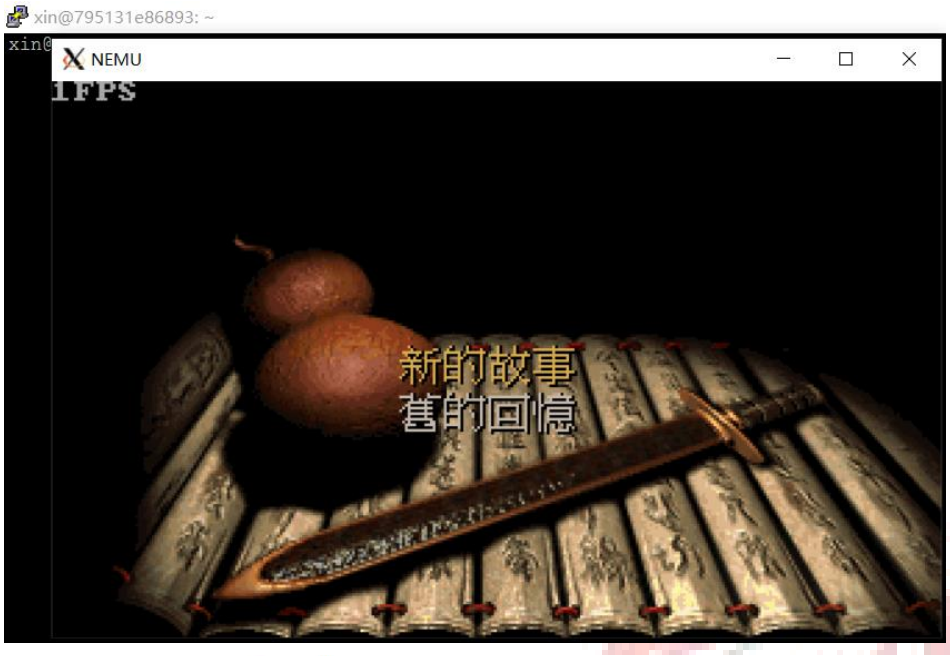
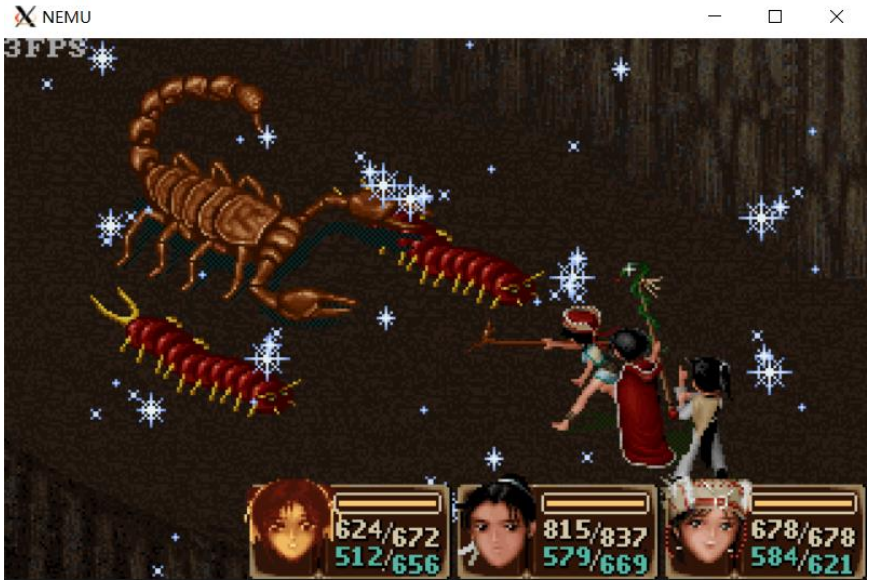
PA4 实验报告

实验进度：PA 完结, 跑出仙剑, 人生圆满.





计算机科学与技术

151220127 吴昕

2017/1/8

必答题

游戏是如何工作的 在享受打字小游戏的乐趣的同时, 思考一下, 游戏究竟是如何工作的? 具体来说, 时钟中断到来/某个键被按下之后, 直到游戏逻辑更新(更新屏幕/寻找击中的字符)的这段时间, 计算机硬件(NEMU 模拟出的 CPU 和设备)和软件(kernel 和游戏代码)如何相互协助来支持游戏的运行? 这个问题涉及到硬件中断的处理过程, 你需要选择一种中断源(时钟/键盘), 并结合代码, 深入计算机层面来回答这个问题(能多详细就多详细).

选择键盘中断为例：

1. (uint32_t)vecx 可以直接取到 vecx()函数的地址, 建立函数中的关联关系后, CPU 要找哪一个异常, 就能找得到处理异常的函数. set_trap()将每一个异常的编号与处理异常的函数程序对应起来.

首先注册 keyboard_event, 通过 add_irq_handle()增加系统调用, 使得系统能够使用该函数来实现信号处理.

2. Wait_intr 调用了 hlt 指令, cpu 进入休眠, 在每一个时钟周期当中, 执行的过程中每执行一个指令, 查看一次中断引脚, 看是否有硬件中断到来.

3. 有硬件中断到来时通过优先判定电路获得优先响应的中断序号.

4. kernel/src/irq/do_irq.c 里定义了一系列处理异常程序的指令. asm_do_irq 中的 push al 第二阶段, 保存现场. add $8 是因为 push 0 和 push 0x80.

执行 asm_do_irq 跳转到键盘中断处理程序当中去(irq_handle), 因为 irq==0x80, 转入系统调用. 最终调用 keyboard_event 来实现中断的处理.

5. 从硬件接口 in_byte 获得输入的键盘的码字, 调用 press_key 函数来实现游戏对按键的反应.

6. 中断处理完之后,就跳回到 irq_handle 函数, 继续处理下一个中断, 一直继续下去直到处理完所有的中断, 回到游戏主循环.

7. 在游戏的主循环当中, 嵌套循环调用 update_keypress 来检查按键情况.

8. 最终找到用户按下的对应键, 并进行反馈. 至此, 从敲击键盘到键盘逻辑被更新全部完成.

蓝框选答题

重新设置 GDT

为什么要重新设置 GDT? 尝试把 init_cond() 函数中的 init_segment() 注释掉, 编译后重新运行, 你会发现运行出错, 你知道为什么吗?

```
(nemu) c
.PDE not present!
```

报错页目录缺失. 指令在 int $0x80 前停止. 在 raise_intr()函数里面调用了 lnaddr_read(), 而 start.S 里面的 GDT 已经无法适用, 若注释掉 main.c 中的 init_segment(), 那么无法对页目录及页进行初始化, 会报错页目录和页缺失.

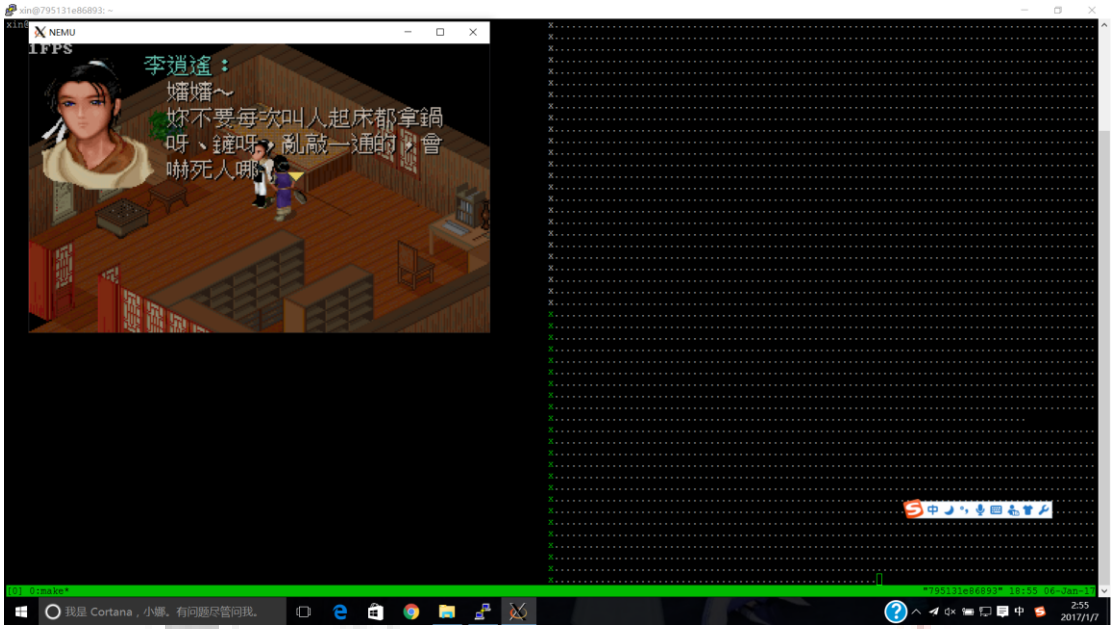
诡异的代码

do_irq.S 中有一行 pushl %esp 的代码, 乍看之下其行为十分诡异, 你能结合前后的代码理解它的行为吗? Hint: 不用想太多, 其实都是你学过的知识.

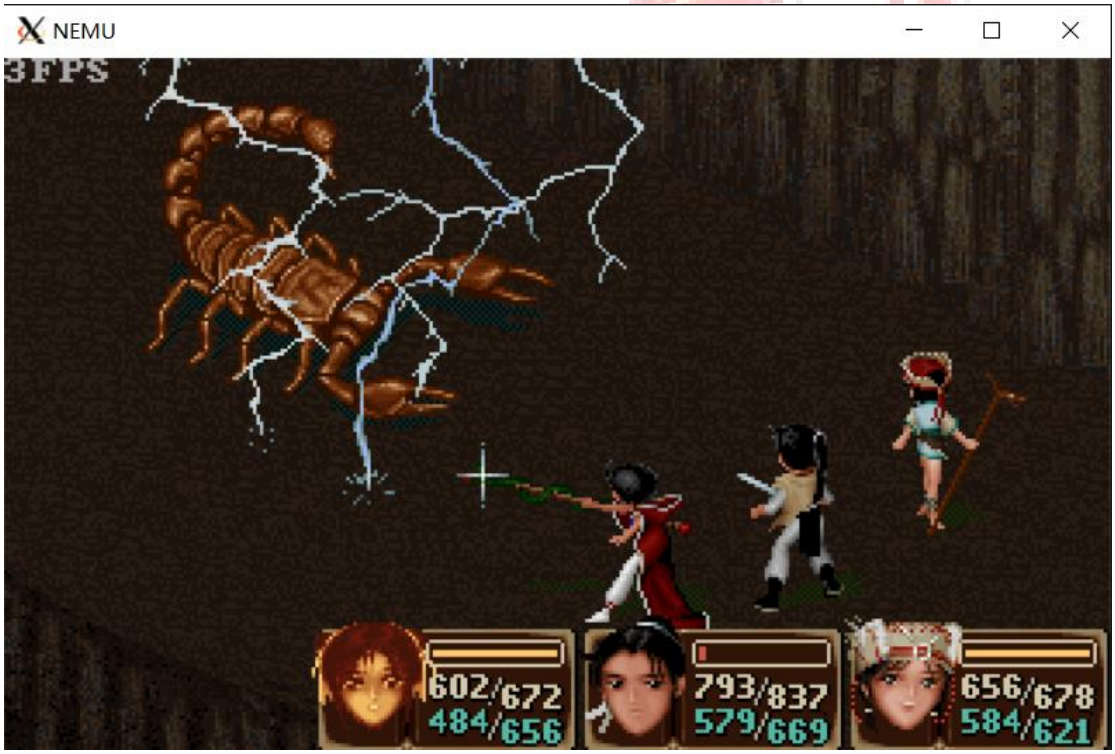
把当前的 %esp 压栈, 作为 old_esp.

实验成果

新的故事



旧的回忆



退出游戏

```
xin@795131e86893:~/ics2016$ make run
objcopy -S -O binary obj/kernel/kernel entry
obj/nemu/nemu obj/game/game
c
Welcome to NEMU!
The executable is obj/game/game.
For help, type "help"
(nemu) c
[kernel/src/main.c,73,init_cond] {kernel} Hello, NEMU world!
[game/src/common/main.c,23,main] {game} game start!
[game/src/nemu-pal/main.c,115,PAL_Init] {game} VIDEO_Init success
[game/src/nemu-pal/global/global.c,69,PAL_InitGlobals] {game} loading fbp.mkf
[game/src/nemu-pal/global/global.c,71,PAL_InitGlobals] {game} loading mgo.mkf
[game/src/nemu-pal/global/global.c,73,PAL_InitGlobals] {game} loading ball.mkf
[game/src/nemu-pal/global/global.c,75,PAL_InitGlobals] {game} loading data.mkf
[game/src/nemu-pal/global/global.c,77,PAL_InitGlobals] {game} loading f.mkf
[game/src/nemu-pal/global/global.c,79,PAL_InitGlobals] {game} loading fire.mkf
[game/src/nemu-pal/global/global.c,81,PAL_InitGlobals] {game} loading rgm.mkf
[game/src/nemu-pal/global/global.c,83,PAL_InitGlobals] {game} loading sss.mkf
[game/src/nemu-pal/global/global.c,86,PAL_InitGlobals] {game} loading desc.dat
[game/src/nemu-pal/main.c,123,PAL_Init] {game} PAL_InitGolbals success
[game/src/nemu-pal/main.c,130,PAL_Init] {game} PAL_InitFont success
[game/src/nemu-pal/main.c,137,PAL_Init] {game} PAL_InitUI success
[game/src/nemu-pal/main.c,144,PAL_Init] {game} PAL_InitText success
[game/src/nemu-pal/main.c,147,PAL_Init] {game} PAL_InitInput success
[game/src/nemu-pal/main.c,149,PAL_Init] {game} PAL_InitResources success
[game/src/nemu-pal/main.c,187,PAL_Shutdown] {game} SOUND_CloseAudio success
[game/src/nemu-pal/main.c,189,PAL_Shutdown] {game} PAL_FreeFont success
[game/src/nemu-pal/main.c,191,PAL_Shutdown] {game} PAL_FreeResources success
[game/src/nemu-pal/main.c,193,PAL_Shutdown] {game} PAL_FreeGlobals success
[game/src/nemu-pal/main.c,195,PAL_Shutdown] {game} PAL_FreeUI success
[game/src/nemu-pal/main.c,197,PAL_Shutdown] {game} PAL_FreeText success
[game/src/nemu-pal/main.c,199,PAL_Shutdown] {game} PAL_ShutdownInput success
[game/src/nemu-pal/main.c,201,PAL_Shutdown] {game} VIDEO_Shutdown success
[game/src/nemu-pal/main.c,204,PAL_Shutdown] {game} UTIL_CloseLog success
[game/src/nemu-pal/main.c,207,PAL_Shutdown] {game} SDL_Quit success
nemu: HIT GOOD TRAP at eip = 0x08064fcb

(nemu) q
```

代码心路历程

为了写 sys_write()函数, 我们有必要将步骤倒过来看, 即从 nemu_trap 理解 sys_wirte().

根据讲义, 在 sys_write()中使用内联汇编来“陷入”到 NEMU 中：asm volatile(“.byte 0xd6” : : “a”(2), “c”(buf), “d”(len));
这是条无优化汇编语句, 执行 opcode 为 0xd6 的指令, 而 i386 中本无此指令, 被 nemu 设为 nemu_trap. 那么这条汇编语句的作用就是转入执行 nemu_trap. 输入 eax = 2, 我们再回过头来看 nemu_trap 指令.
nemu_trap 中如果%eax==2, 则输出%ecx 为首地址的%edx 字节的内容.
sys_write 的原型是这样的：sys_write(unsigned int fd, const char * buf, size_t count)
查阅资料知道 fd 对应的是寄存器中的 ebx. 那么我们对几个参数进行解析, 并通过对 nemu_trap 的功能分析, 将参数和寄存器建立联系：fd 是文件描述符(cpu.ebx)；buf 是缓冲区(cpu.ecx), 存放写入或者读出的数据； count 对应的是字节数(cpu.edx).
所以在 sys_write 函数中, 我们只要提取 buf 和 len, 然后调用汇编, 最后返回给 trapframe 中的 eax 即可. 并且根据 KISS 原则, 先不考虑 fd != 1 && fd != 2 的情况, 一律 assert 或者 panic.

bug 及 debug

PDE 缺失错误

1. mov_a2moffs 指令实现错误, instr_fetch(cpu.eip + 1, 4)改为 instr_fetch(eip + 1, 4)即可.
2. lgdt 指令中原本是 dram_read(). 但是 kernel 里面内存为高地址 0xc0...时显然会超出物理内存范围, 改为 swaddr_read()即可.

hello-inline-asm 实现输出却报错 PDE 缺失

可以知道指令在 nemu_trap 处实现是没有问题的, 那么问题就出在这之后. 执行指令到报错的前一条语句, infor 一下发现 eip 值为 0x80480a3. 然而根据反汇编代码（截图如下）, 是没有以 0x80480a3 开头的指令的.

```
08048089 <main>:
 8048089:       55                      push   %ebp
 804808a:       89 e5                   mov    %esp,%ebp
 804808c:       b8 04 00 00 00          mov    $0x4,%eax
 8048091:       bb 01 00 00 00          mov    $0x1,%ebx
 8048096:       b9 a9 80 04 08          mov    $0x80480a9,%ecx
 804809b:       ba 0e 00 00 00          mov    $0xe,%edx
 80480a0:       cd 80                   int    $0x80
 80480a2:       b8 00 00 00 00          mov    $0x0,%eax
 80480a7:       5d                      pop    %ebp
 80480a8:       c3                      ret
```

猜想 1: 在加载完 kernal 回到用户程序的时候 eip 传的是错误值, 也就是 loader()返回值错误. debug 时便在 loader()函数中 return entry 语句前 set_bp(). 由于返回值是存放在 eax 中的, 那么 infor 一下查看寄存器的值, 发现 eax = 0x8048089, 正是 main 函数的地址呀, 那么 loader()实现应该是没问题的. 猜想 1 错误.

猜想 2: 由于报错 PDE 缺失的时候 eip 的值为 0x80480a3, 和 int $0x80 语句的下一条指令起始 0x80480a2 非常接近, 而且 int 又是新增语句, 那么应当是在指令实现时对 cpu.eip 的赋值出问题了, 查看指令代码中, 有 cpu.eip += 2, 改为 cpu.eip += 1, HIT GOOD TRAP !

归根到底还是指令实现没有完善, 以后一定要加倍小心.

typing game

x11 之后开始有了窗口, 也出现了花屏, 着实兴奋好久. 可是好景不长, 马上就出现了报错. 如图 1 图 2, 有时候是缺页, 页码位置还很奇怪(如图中所示 5401), 再来是奇怪的断言.

```
call.c,46,do_syscall] {kernel} eax = 0 here
ebx=c0127190, (ebx)=5401eip = 5401, addr = 5401, PTE not present!
```

```
(nemu) c
.[kernel/src/main.c,73,init_cond] {kernel} Hello, NEMU world!
.....................[kernel/src/irq/irq_handle.c,22,add_irq_handle] {kerne
stem panic: Assertion failed: handle_count <= NR_IRQ_HANDLE
nemu: HIT BAD TRAP at eip = 0xc010081c
```

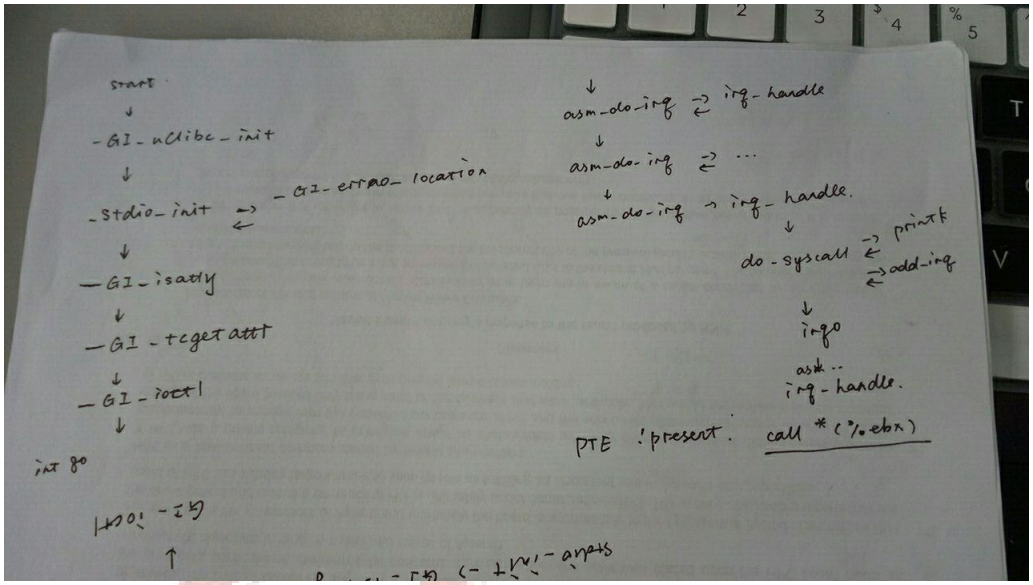
make test 中 hello-inline 和 hello 不过的原因是 fd13 和 14 没有实现, 找到 sysnum 文件查看 fd=13 和 fd=14 的系统调用如下, 是两个没有学过的莫名其妙的系统调用？为了保险起见我问了大神, 大神说没有用到, 好, 那就没有用到. 那就是我的指令又错了 TAT…

```
x#define __NR_time 13
x#define SYS_time __NR_time
x#undef __NR_mknod
x#define __NR_mknod 14
x#define SYS_mknod __NR_mknod
x#undef __NR_chmod
```

既然现在可以在 kernel 中 Log 了, 不妨用用来 debug. 我首先是在 kernel 的 init_cond 中用 Log 查看跑到哪一步了, 最后发现如图 3 所示语句前, Log 可以输出, 而在那之后就不行了. 而这条语句是用来跑进用户程序的. 那么出错应该在 game 里面.

```
x
x       /* Clear the test data we just written in the video memory. */
x       video_mapping_clear();
x#endif
x
x#ifdef IA32_PAGE
x       /* Set the %esp for user program, which is one of the
x        * convention of the "advanced" runtime environment. */
x       asm volatile("movl %0, %%esp" : : "i"(KOFFSET));
x#endif
x
x       /* Keep the `bt' command happy. */
x       asm volatile("movl $0, %ebp");
x       asm volatile("subl $16, %esp");
x
x       /* Here we go! */
x       Log("here!\n");
x       ((void(*)(void))eip)();
x       HIT_GOOD_TRAP;
x
x       panic("should not reach here");
x}
```

接着, 我在 log.txt 中顺藤摸瓜, 对照着 kernel.S 和 game.S 查看到底出错在什么函数, 图 4 是追溯历程.



之所以用到 kernel.S 是因为常常有中断指令会跑进内核, 结合 kernel 反汇编可以很好的跳过这些冗余部分.

我的 log.txt 中指令最后一条是 call far, 那么进入这个指令, 之前 pa2 的时候写的指令着实有点久远, 对照着 386 手册重新码了一遍. 其实很多语句的功能和原来的代码没有区别, 我只是换了一种写法. 然而奇迹般的…它…game start 了？！

```
.[kernel/src/main.c,73,init_cond] {kernel} Hello, NEMU world!
.....................[kernel/src/syscall/do_syscall.c,20,sys_ioctl] {kernel} 540
1

.[kernel/src/syscall/do_syscall.c,20,sys_ioctl] {kernel} 5401

[game/src/common/main.c,22,main] {game} game start!

exit
q
^C
quit
```

让我喝口咖啡冷静一下…可是！并没有讲义中说的什么字母游戏的界面啊！还是小黑框而且…死循环？！如图 6, 想尽各种办法退不出来. 让我再喝口咖啡冷静一下…

冷静之后我查看了一下 log.txt 的最后一条语句, 并找到这条语句所在函数, 发现它的下一条是 hlt. 原来是死在 hlt 的死循环里面了么？这个 bug 我是问了同学才知道, 我的判断条件错了. 之前写的是 `while(!(cpu.INTR && cpu.IF));` 改成 `while(!cpu.IF);`即可. 果然解决 hlt 之后不死循环了, 新的 bug 又出现了呢(生无可恋)…

```
.[kernel/src/main.c,73,init_cond] {kernel} Hello, NEMU world!
.....................[game/src/common/main.c,23,main] {game} game start!
.[game/include/device/video.h,16,draw_pixel] {game} f1680000 b0

[game/include/device/video.h,17,draw_pixel] {game} system panic: Assertion faile
d: x >= 0 && y >= 0 && x < SCR_HEIGHT && y < SCR_WIDTH
nemu: HIT BAD TRAP at eip = 0x0804834d
```

这个断言是 draw_pixel 中的传入参数 x 和 y 超过屏幕范围.

废了千辛万苦终于找到出错前调用 draw_pixel 的函数. 函数调用过程是这样的 main_loop->redraw_screen->draw_string->draw_character. 那么在 redraw_screen 里面 Log 一

下输出执行信息如下：

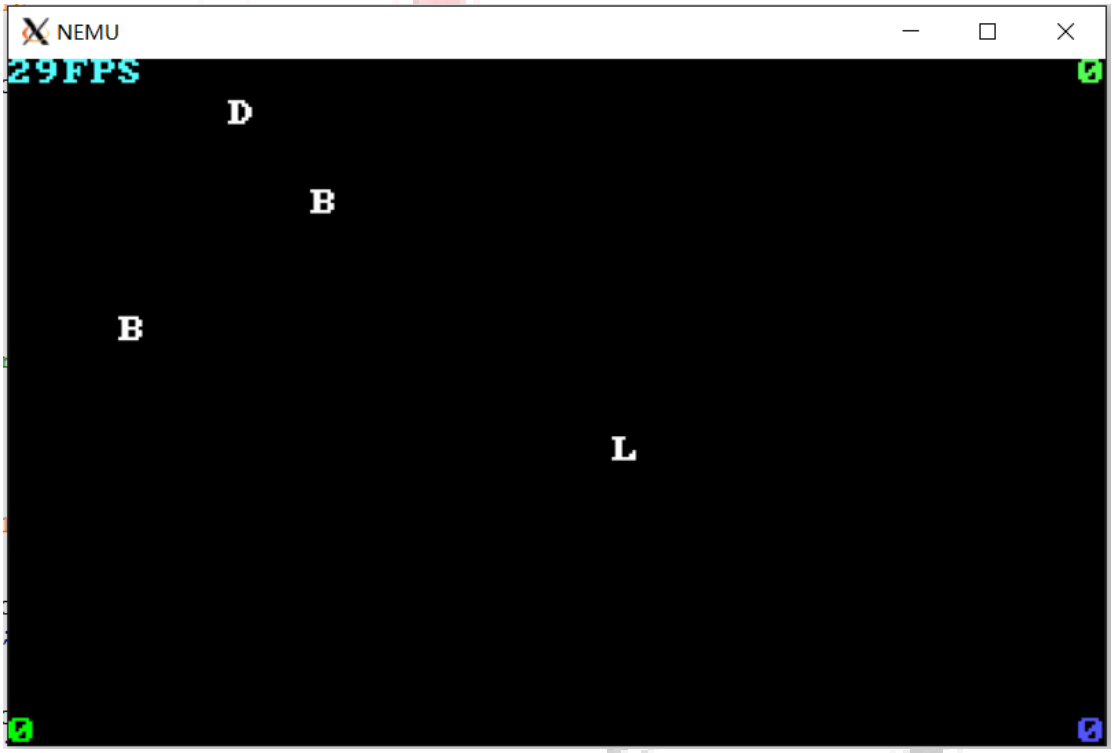
```
[game/src/typing/draw.c,28,redraw_screen] {game} here!

[game/src/typing/draw.c,30,redraw_screen] {game} buffer
[game/src/typing/draw.c,35,redraw_screen] {game} -4539809792 61800
.[game/src/common/device/video.c,29,draw_character] {game} x = -4539809792, y =
176
```

表明函数已完成 prepare_buffer, 在 draw_string 这里出了问题. 回想我的错误就是 x 超范围, 那么我只要在这里看看我的 x 究竟传的是什么.

```
draw_string(buf, F2int(it->x), it->y, 15);
```

F2int! 原来是这个东西！Log 一下 FLOAT(it->x)和 it->x, 分别是-4539809792 和 61800, 那么问题就锁定在 F2int 这边了. 果然还是浮点没实现好...

修改之后! 将将将将！



PA4

这里前前后后不知道 de 了多少 bug, 很多可能根本不是 bug 但是被我为了以防万一改写了很多次. 因为不知道什么原因 fread 的时候 fd 一直为 255. 后来(这里轻描淡写但是实际上真的经历了很长一段时间)才发现是 push 指令的错误. push_i_b 和 push_ib_v 是不一样的. 这个 bug 被 de 掉之后就一路通车跑到仙剑了.

其实本来应该还会有报缺页错, 因为我用的是 docker, 所以和 64 位虚拟机文件 sort 顺序不太一致, 这个问题是同学先发现的, 省去了我很多 debug 的功夫.

```
cat `find $(game_SRC_DIR)/nemu-pal/data -type f | sort -d` >> $@
```

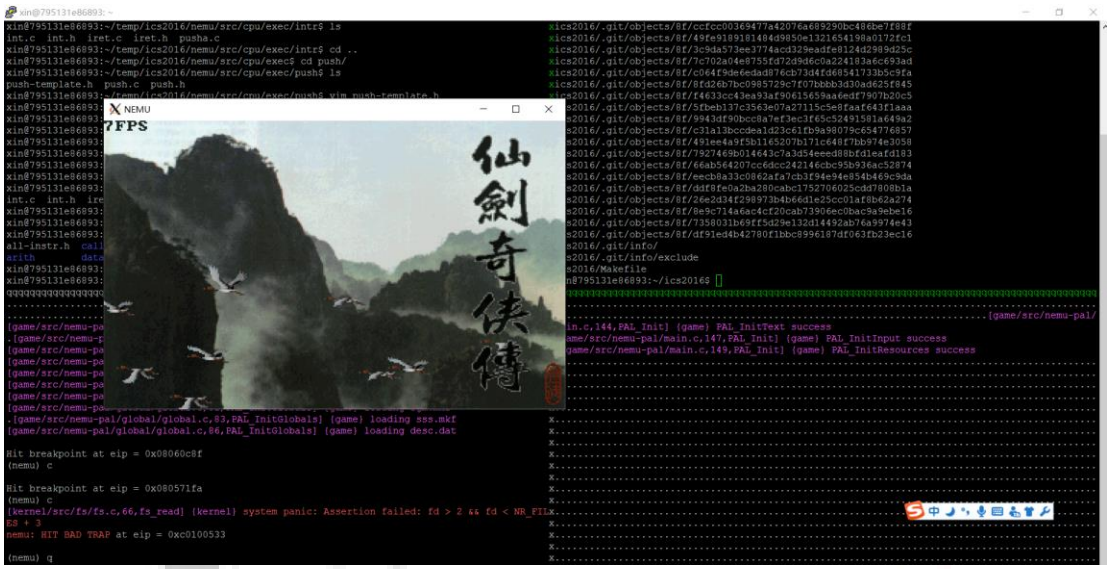
只要在 sort 后面加上-d 就可以免去缺页错.

声明事项

1. 2016/12/23 18:25 左右我的 docker 崩掉过一次, 按道理 git 应该是没问题的, 但是怕万一出事, 所以如有疑问可以找本人查询.
2. 仙剑跑出来之后再去看自己的代码, 几乎面目全非...其实 PA4 中间 de 到几近崩溃的时候几乎想要把 nemu 重新写一遍, 也确实重新写了一点...后来发现实在是太大的工程, 加上又在考试周, 实在坚持不了重写, 还是老老实实 debug, 所以 PA4 最终代码可能和前几次提交的代码不太一样, 重复率会较低, 但是确实是本人所写.
3. 后期 debug 的时候有时候 de 得越来越错所以中间会 git commit 备注一下, 因为每次 make run 都会 commit 一次, 为了查找记录方便, 我把几个 Makefile.part 以及 Makefile 中自动 commit 的命令注释掉了, 所以中间有一段时间 commit 得不太频繁, 甚至有一段时间我的分支不在 pa4 里, 而是在我读取的一个记录(是一串以 git log 里面 commit 后面的字符串命名的临时分支)里. 到后期 git log 也实在太麻烦, 我干脆将文件传到电脑里备份之后才将注释解除. commit 记录应该还是有的, 不确定是 make test 还是什么指令的时候还是会 commit, 只是不太频繁而已, 在此声明.

PA 心得

PA 跑出仙剑的时候, 是 1.7 日凌晨 3:04.
现在看那些截图, 感觉还是下面这张我最喜欢.



debug 的痕迹还在, 窗口还在 hit bad trap. 这张图好像让人相信, 下一秒就能跑出仙剑.
PA3 尾声的那堂 PA 课, 是倒数第二节 PA 课. 我以为我们还停留在 PA3, 没想到直接把 PA 剩下的内容全部讲完了. 听着感觉很轻松, 看着屏幕上仙剑跑出来的画面, 我热血沸腾. 马上能跑仙剑了.
哪那么容易呢.
这学期开始 PA 的时候, 我就是奔着仙剑去的. 我想, 我一定能跑出来的. 不知道是仙剑情结还是自己要强, 一半一半吧, 反正我就是要跑仙剑.
我自认不算聪明, 所以看着巨巨们超前完成, 我只能规规矩矩跟着进度来, 然后迷迷糊糊到了 PA4.
PA4 感觉比前几个阶段都漫长得多得多.
这也许不是最关键的一环, 却是最致命的一环. 前面代码的 bug 在这一关都会自食恶果. 这就是我这一关特别痛苦的原因.
恰逢期末, 助教们在忙着大作业, 不大好意思叨扰, 只能自己看 log, 再检查代码.
到了后期, 考试周, 因为没有课, 所以几乎是没日没夜的 de, 有时候 de 了一个通宵, 只要能过掉一个 bug, 哪怕迎来下一个, 也是开心的, 然后心满意足歇上一会儿, 再继续. 学长看我可怜私戳给我发了去年的助攻, 学姐也愿意给我她的代码, 但是由于框架不一样跑不起来. 终究还是只能靠自己.
其实就在不久前, 但是现在去看, 好像是很久很久以前的事儿了, 大概是跑完整个人终于可以放松下来, 心态也就不一样.
大概 4 天前吧, 我 de 得实在难受, 却又放不下手. 看着已经跑出来的大神巨巨, 想着别人能做到我肯定也能, 可是我为什么就是跑不出来. 然后就自己纠着, 越想越烦, 复习也没法好好复习, 总是挂念着, 看书看到一半莫名想到 PA 跑不出来跑不出来. 索性撂下书再 de 一会, 却实在没有头绪, 不知道究竟错哪儿了.

我问了好多已经跑出来的学长学姐和同学后期有什么 bug, 是怎么 debug 的, 就想着那么多 bug 的原因哪怕有一个是我的都好. 大多的回答都是指令问题, 我就前前后后把所有指令看了 3 遍. 同学索性发了一份他的代码给我, 虽然仍旧存在 bug, 但是能进仙剑界面. 我也没头绪, 想着先过了这个 bug, 到下一个 bug 再说, 干脆用了最笨的方法, 对指令. 我就这样把我的指令一个个换过去, 看换了哪个进不去, 就是有问题的.

那时候学长安慰我说, “其实 PA3 完成分数就很高了, 其实打字小游戏出来就很不错了. ” “可是我就是要跑仙剑. ”“仙剑啊, 这个看缘分的. ”“向来情深, 奈何缘浅. ”

能到现在这一步, de 的应该都不是分数, 要么是情怀, 要么是要强. 我后面两种都是.

那天晚上照例等到阿姨来催才离开机房, 还下雨了那天. 我一边走一边抖, 一半冷的, 一半被自己气的, 吴昕啊你这人怎么这么不争气. 晚上回寝又 de 了一会, 照例没有头绪, 又自己跟自己较劲了一把. 躺在床上越想越不平, 还发了一篇特长的票圈, 票圈开头的基调大概是“唉我好累我想放弃了”, 但是不知道为什么写着写着文风就变成“大把青春都砸下去了现在让我放弃？见鬼去吧. ”第二天收到好多人的鼓励, 大多是让我早点睡.

我也不懂为什么都这么折磨了也还没想过放弃, 反正我就是要跑仙剑, 跑不出来我 PA 白写青春白搭一学期白活.

因为今天早上要考试, 所以近几天都在慢慢调生物钟, 所以尽量早睡了. 1.7 日那天, 大概是鬼迷心窍？反正没控制住自己的手又打开来 de 了一会, 最后的 bug 是 exec.c 里面, push 指令和 opcode 没有对好. de 掉之后, 一路顺风. 感情就这一个指令, 败坏我这么多日日夜夜. 但好值啊.

其实在之前想过好多次呀仙剑跑出来我会怎样, 好像高兴得尖叫和感动得流泪都不太像我, 但也不会像真的跑出来那样, 面无表情. 拜托, 那是凌晨 3 点还能有什么表情…

新的游戏, 罗刹鬼婆那一段是没法看的, 李逍遥的婶婶拿平底锅把他从梦中敲醒, 就这么在最快 7fps 的速度下看完了这段对话, 然后退出游戏, hit good trap.

和学长再聊 PA, “缘分终于到了啊. ”“多亏我死缠烂打. ”

终于可以安心去考试了.

好困啊, 上床睡觉. 又困又睡不着. 好像也不是特别兴奋. 唉又失眠了. 唉我的生物钟啊. 唉 8 号早上要怎么起来.

现在在写实验报告, 再去看 PA 讲义, 有些话现在看来, 感触特别深.

The machine is always right.

Every line of untested code is always wrong.

成长是一个痛苦的过程.

对这样的自己的不甘, 是改变自己的动力.

只要你坚持下来, 你就是非常了不起的! 你会看到成长的轨迹, 看到你正在告别过去的自己.

我们不是要讨好企业的毕业生, 而是要寻找改变世界的力量. --jyy

你在专业上的技不如人, 迟早有一天会找上来. –etone

总有一天会找上门来的 bug

特别鸣谢

感谢汪亮老师, 讲解非常到位, 如果没有 PA 课, 我对框架代码理解还存在不可逾越的困难. 在课后他也会耐心帮我看代码, 找 bug, 很多阶段性成果都是他帮助我达成的.
感谢刘锐意学长, 在后期最艰难的一段一直给我鼓励, 即使他自己有考试也有求必应.
感谢罗雯波同学, 给我很多理论指导, 帮助我理解框架代码, 帮我 de 掉一个很坑的 bug.
感谢谢旻晖同学, 借给我他的代码参考, 虽然他的代码仍旧存在 bug, 但是没有他的代码我也是没法这么快完成的.
感谢那些让我注意身体早点睡的朋友, 即使不在身边也能感觉得到你们的关怀.
感谢从来没说过放弃, 一腔热血, 执着又要强的自己.