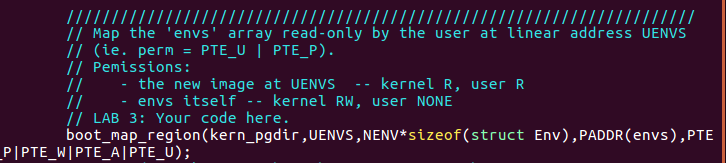
6/11/2019 5:15:35 PM

今天遇到了一个问题：

libmain() then calls umain, which, in the case of the hello program, is in user/hello.c. Note that after printing "hello, world", it tries to access thisenv->env\_id. This is why it faulted earlier. Now that you've initialized thisenv properly, it should not fault. If it still faults, you probably haven't mapped the UENVS area user-readable (back in Part A in pmap.c; this is the first time we've actually used the UENVS area).

就是遇到了这个问题，自己花了几个小时排查了一下：



问题最终出现在这里：将PTE\_U加上之后就可以了。这里自己没有理解envs和UENVS两个实际上指向同一个数据的变量为何要设置不同的权限。其实这就是内存保护的作用

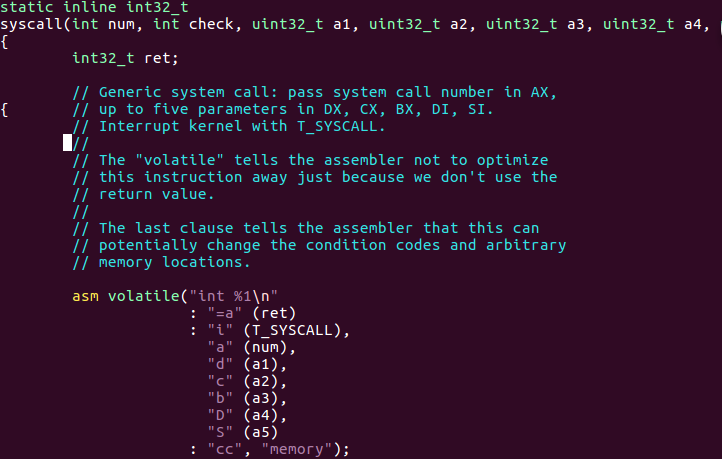
实际上envs的物理地址是在0x001bf000位置,内核状态下其地址被映射到0xf01bf000位置，在用户状态下envs的真实地址又被映射到0xeec00000。实际上是同一个位置。

而且根据env->env\_pgdir可以知道我们申请的所有的用户环境都是具有与内核同样的视野的，但是其访问状态受到限制。

// UVPT maps the env's own page table read-only.

// Permissions: kernel R, user R

e->env\_pgdir[PDX(UVPT)] = PADDR(e->env\_pgdir) | PTE\_P | PTE\_U;

1. 关于自己在Lab3不懂得点：  
   

就是这里面的汇编的真正意思

自己理解的大致意思是：lib中的系统调用实际上就是一个包装好的壳子，间接调用内核底层的系统调用。只是这段嵌入式汇编表达的意思不太理解（关于内联式的汇编语法：<http://www.delorie.com/djgpp/doc/brennan/brennan_att_inline_djgpp.html>）

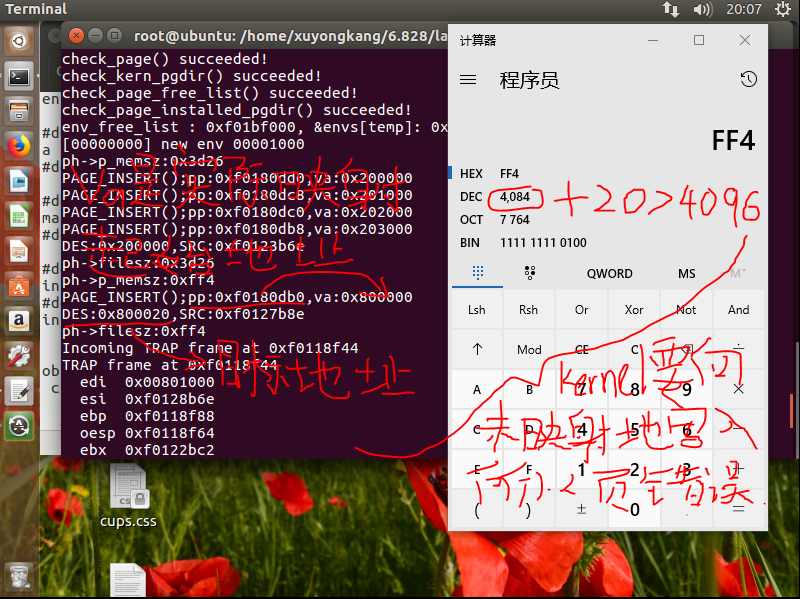
网上说jos中的系统调用是：int $0x30 。但是图中的汇编的int %1是什么意思？

最初的kernel环境是如何建立起来的？？为何会有kernel环境；

ENV\_CREATE()运行的时候就是属于内核建立起来的环境；（但是之前并没有搭建内核的环境啊！！

）

只有真正到env\_run(&envs[0])的时候才是真正建立起来了第一个用户环境;



1. 我们在这里探究代码运行在内核段的过程：
2. void
3. env\_init\_percpu(void)
4. {
5. lgdt(&gdt\_pd);
6. // The kernel never uses GS or FS, so we leave those set to
7. // the user data segment.
8. asm volatile("movw %%ax,%%gs" : : "a" (GD\_UD|3));
9. asm volatile("movw %%ax,%%fs" : : "a" (GD\_UD|3));
10. // The kernel does use ES, DS, and SS. We'll change between
11. // the kernel and user data segments as needed.
12. asm volatile("movw %%ax,%%es" : : "a" (GD\_KD));
13. asm volatile("movw %%ax,%%ds" : : "a" (GD\_KD));
14. asm volatile("movw %%ax,%%ss" : : "a" (GD\_KD));
15. // Load the kernel text segment into CS.
16. asm volatile("ljmp %0,$1f\n 1:\n" : : "i" (GD\_KT));
17. // For good measure, clear the local descriptor table (LDT),
18. // since we don't use it.
19. lldt(0);
20. }
21. asm volatile("ljmp %0,$1f\n 1:\n" : : "i" (GD\_KT));

关于这段代码的全部解释。

2. 关于GD\_KD GD\_KT各种段的研究内容。

3. 关于内联汇编中的一些基本语法asm volatile("ljmp %0,$1f\n 1:\n" : : "i" (GD\_KT));

I代表数字输入

AT&T语法

Intel语法。

asm volatile("int %1\n"

: "=a" (ret)

: "i" (T\_SYSCALL),

"a" (num),

"d" (a1),

"c" (a2),

"b" (a3),

"D" (a4),

"S" (a5)

: "cc", "memory");

上面一段inline assembly与asm volatile("ljmp %0,$1f\n 1:\n" : : "i" (GD\_KT));

Int %1 -> 按照顺序0指的是a而1指的是i(T\_SYSCALL) 因此也就重新触发了中断

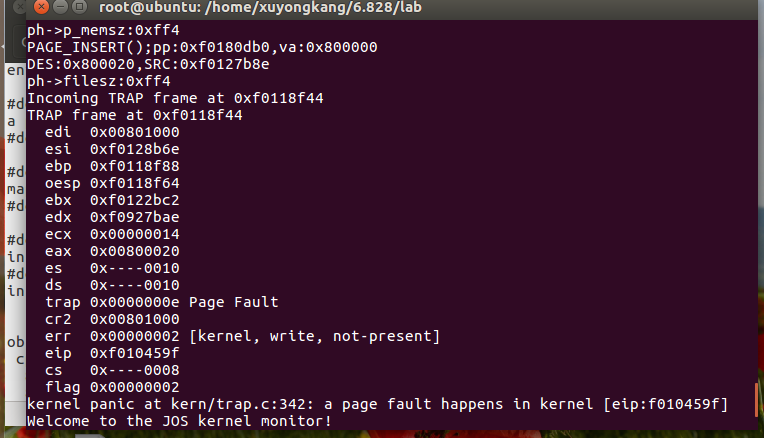
Ljmp %0 刚好0代表的就是一个数字：i(GD\_KT)

这一段inline assembly汇编进行比较

4. 自己是如何将错误定位到发生在memmove位置的，这个方法值得借鉴，能够调试接下来的代码？

### 2. 关于CPL, RPL, DPL

CPL是当前正在执行的代码所在的段的特权级，存在于CS寄存器的低两位(对CS来说，选择子的RPL=当前段的CPL)。RPL指的是进程对段访问的请求权限，是针对段选择子而言的，不是固定的。DPL则是在段描述符中存储的，规定了段的访问级别，是固定的。为什么需要RPL呢？因为同一时刻只能有一个CPL，而低权限的用户程序去调用内核的功能来访问一个目标段时，进入内核代码段时CPL 变成了内核的CPL，如果没有RPL，那么权限检查的时候就会用CPL，而这个CPL 权限比用户程序权限高，也就可能去访问需要高权限才能访问的数据，导致安全问题。（普通权限的用户进程借调用内核进程去访问高权限的敏感数据。）所以引入RPL，让它去代表访问权限，因此在检查CPL 的同时，也会检查RPL。一般来说如果RPL 的数字比CPL大(权限比CPL的低)，那么RPL会起决定性作用，这个权限检查是CPU硬件层面做的。

3.可以通过内核崩溃是寄存器中保存的地址而直接定位到错误位置，而不用一直写输出函数判断范围。

如图片中直接显示崩溃的eip:f010459f

root@ubuntu:/home/xuyongkang/6.828/lab# make qemu

+ cc kern/env.c

+ ld obj/kern/kernel

+ mk obj/kern/kernel.img

qemu-system-i386 -drive file=obj/kern/kernel.img,index=0,media=disk,format=raw -serial mon:stdio -gdb tcp::25000 -D qemu.log

VNC server running on 127.0.0.1:5900

6828 decimal is XXX15254 octal!

Physical memory: 131072K available, base = 640K, extended = 130432K

check\_page\_free\_list() succeeded!

check\_page\_alloc() succeeded!

check\_page() succeeded!

check\_kern\_pgdir() succeeded!

check\_page\_free\_list() succeeded!

check\_page\_installed\_pgdir() succeeded!

env\_free\_list : 0xf01bf000, &envs[temp]: 0xf01befa0

[00000000] new env 00001000

ph->p\_memsz:0x3d26

PAGE\_INSERT();pp:0xf0180dd0,va:0x200000

PAGE\_INSERT();pp:0xf0180dc8,va:0x201000

PAGE\_INSERT();pp:0xf0180dc0,va:0x202000

PAGE\_INSERT();pp:0xf0180db8,va:0x203000

DES:0x200000,SRC:0xf0123b6e

ph->filesz:0x3d26

ph->p\_memsz:0xff4

PAGE\_INSERT();pp:0xf0180db0,va:0x800000

DES:0x800020,SRC:0xf0127b8e

ph->filesz:0xff4

Incoming TRAP frame at 0xf0118f44

TRAP frame at 0xf0118f44

edi 0x00801000

esi 0xf0128b6e

ebp 0xf0118f88

oesp 0xf0118f64

ebx 0xf0122bc2

edx 0xf0927bae

ecx 0x00000014

eax 0x00800020

es 0x----0010

ds 0x----0010

trap 0x0000000e Page Fault

cr2 0x00801000

err 0x00000002 [kernel, write, not-present]

eip 0xf010459f

cs 0x----0008

flag 0x00000002

kernel panic at kern/trap.c:342: a page fault happens in kernel [eip:f010459f]

Welcome to the JOS kernel monitor!

Type 'help' for a list of commands.

here show code of myself

yourname is xuyongkang. HELLO\_WORLD

x 1,y 3,z 4

x 1,y 3,z 4

x,y,xK> Killed