***JOS Lab3 文档***

杨健邦 515030910223

*本文档描述了本次LAB各个Exercise实现的方法或者我自己的理解 (只陈序一部分相对困难的Exercise)。至于对Question的解答以及以及Challenge的实现方法，请阅读answer-lab3.txt。*

## Part A: User Environments and Exception Handling

**Exercise 1.**

* 与pages数组类似，对envs数组分配空间以及初始化，也是使用bootalloc函数。此外，还要做从虚拟地址UENVS到这个envs数组的物理地址的映射，由于UENVS是UTOP以上的地址，envs所对应的物理页是永远不会被free掉的，因此也使用boot\_region\_map的方法。

**Exercise 2.**

* env\_init()：这个函数的主要目的是将envs数组里面的所有env成员加到env\_free\_list之中，这里与free\_page\_list有点类似，但是这个函数还要求初始化之后env\_free\_list的env顺序与envs数组中env的顺序要一致。因此遍历envs数组要从后往前遍历。
* env\_setup\_vm()：这个函数为新的env分配一个新的地址空间。这里使用了kern\_pgdir作为模板，只需要分配一个新的页，并且将kern\_pgdir的一级页表拷贝到这个页即可。
  + **为什么只需要拷贝一级页表，而二级页表不需要拷贝？**
    - 因为我们用来做模板的kern\_pgdir只映射了UTOP以上的空间，上面的映射对于每一个env来说都是一样的，只有UTOP以下的才会不一样。
    - 属于新的env的自己独有的二级页表还没有。
* load\_icode()：要注意使用memset以及memmove函数的时候，要把页表切换成env的页表，否则会产生page\_fault，因为kern\_pgdir这个页表中，是没有做这些用户态的映射的。
* 注意：在完成这个exercise之后，是会出现triple fault的情形，现象是不断重启。而网上文档说的出现triple fault会打印triple fault的消息并且退出是6.828 patched QEMU才会出现的，mit这个补丁版本的QEMU是做了特别修改，专门针对JOS的，但我们使用的只是普通的QEMU，因此出现triple fault的现象只是普通重启。

**Exercise 3.**

* IA32那本手册非常有用，讲得很详细，遇到不懂的只是可以阅读一下。完成本次lab需要了解GDT、IDT、TSS、trap gate和call gate的知识。

**Exercise 4.**

* 这里特别需要知道的是当触发了异常或者中断之后，CPU做了很多事情。

1. 由于异常和中断的处理是在RING0下处理的。因此CPU通过TSS切换当前env下RING0状态对应的栈(esp0: KSTACKTOP)以及更换段寄存器(cs0: GD\_KT)。
2. 切换了栈后，CPU已经自动把ss，esp，eflags，cs，eip，errno等信息push入栈，剩下的trapframe的信息是通过执行trap entry point的汇编代码来入栈。
3. 我们设置的trap gate中包含了trap entry point的偏移信息，CPU是通过这个偏移量来知道触发某个异常后应该执行什么代码的。

**Part B: Page Faults, Breakpoints Exceptions, and System Calls**

**Exercise 6.**

* sysenter/sysexit是与int 不同途径的产生系统调用的方法，前者是通过寄存器来传参的，而且是不走IDT trap gate那一条路径，因此需要重新设置某些东西。
* 首先要设置三个MSR寄存器，分别存了调用sysenter指令后，CS、ESP、EIP要改成成什么值，分别意味着系统调用使用的是什么代码段，系统调用要使用什么栈以及调用系统调用后第一句执行的指令在什么地址。
* 接下来要写汇编来将参数放到寄存器里面，以及将参数从寄存器中取出。那么，第一步使用的是asm内联汇编，要注意的是，lab3预先给的部分内敛汇编代码有一部分是多余的，已经在input/output list里面的寄存器是不需要再手动push和pop的，如果要了解其中的原理，可以去听听《编译原理》的课。
* 除此之外，还要注意的是，sysexit的时候，是通过哪个寄存器得值来找到返回地址和原来的栈地址的。

**Exercise 8.**

* sysbrk函数要注意防止用户将空间分配到UTOP以上。

**Exercise 9.**

* 对于单步调试，EFLAGS寄存器上有个特殊的位TF，只要将这个bit置上，每执行一条指令就会触发一个debug exception，trap到这个exception然后才打印相关的eip以及symbol信息。

**Exercise 12.**

* 这个Exercise是一个终极大BOSS，要搞懂这道练习要先去看网上的博客或者维基百科或者osdev或者IA32的文档，推荐去读IA32的文档，里面对于各个概念都讲得非常的清楚和全面。
* 做这道题之前，首先得了解GDT、LDT、IDT、task gate、trap gate、call gate、task state segment等等这些概念是什么、它们里面存的是什么信息、它们有什么联系、有什么区别、如何使用它们。
* 我们这道题是在GDT里面设置call gate，通过提升自己的等级来执行某段代码。经过测试后发现，修改GDT中的UT、UD和TSS0的对应位置变成call gate没有问题，因为系统只有执行某些操作的过程中，才会将GDT中的信息读到某些寄存器中（cs、ss以及tr），其它时候都是读缓存在这些寄存器的信息。