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。