PartA

Excercise1. 试下来好像有两种方法行得通。一种是直接用 Env 的 Trapframe 的 padding 作为 flag。在 sysenter 进入时，设置 padding 为 1，这样可以通过判断来选择用 iret 还是 sysexit。还有一种是在 sysenter 进入内核后构造与 int 进入后相同的栈，这样好像不用判断，也可直接都使用 iret 返回（不知道是不是适用于所有情况）。

Excercise2. 本来想根据栈结构来取 lib/syscall 的参数的地址，这样前面的参数的地址知道了，后面参数也可以得到。但是因为 inline 所以栈结构不对，所以又把 inline 删掉了，牺牲了性能，但是代码比较简单。既然还能用省去传入 eip，那就试了一下，这个方法浅尝辄止，没有成功，等以后有空再试了。

What's the difference between fork in Linux and in JOS?

Linux 的 fork 基于 clone。传入一些参数。

JOS 则不然。

Linux 的 fork 尽量子进程先运行。Fork 则直接运行父程序。

why exo_fork wrapper function must be an inline function?

如果没inline。那么栈结构return后改变。不能保证copy时的正确性。

Why “INT” instruction is much easier than “sysenter” for exo_fork system call?

CPU 自动push %eip之类的寄存器。

万恶的测试文件：到了pagefault那里，即时是代码错误仍然能够通过。首先是缺页异常的栈的返回出错了，因为在处理完缺页异常后，测试用的用户程序没有调用栈，结果居然能够通过某些测试，这个还好很容易看出来。我因为前面贪省力，还没做copyonwrite时，常常用PTE_USER来代替PTE_W|PTE_U|PTE_P。结果写了copyonwrite的有关代码后，发现有些地方不能简单地用PTE_UER。于是稍作修改（前面的忘了自己用的是PTE_USER，所以没改），结果居然跑出来45分，前面的测试通过了，Fork怎么跑都跑不过。实在想不出错哪，后来才发现原来在前面的faultalloc等程序就运行不对了，但是通过了相关的测试，莫名其妙，浪费了很多时间。搞得我现在神经兮兮，没做完一个，都要亲自一个个运行前面的测试程序。

目前45分