# Lab4

王根国
1120379047
wanggenguo@yahoo.com

## Questions：

1. Compare kern/mpentry.S side by side with boot/boot.S. Bearing in mind that kern/mpentry.S is compiled and linked to run above KERNBASE just like everything else in the kernel, what is the purpose of macro MPBOOTPHYS? Why is it necessary in kern/mpentry.S but not in boot/boot.S? In other words, what could go wrong if it were omitted in kern/mpentry.S?
   答：
   因为 boot.S 的 link address 和 load address 是一致的，不需要转换。
   而 mpentry.S 的 link addres 和 load address 是不一致的，在设置好地址转换前，需要自己手动转换地址，所以就需要 MPBOOTPYHS，不然就会跳转到不对的位置了。
2. It seems that using the big kernel lock guarantees that only one CPU can run the kernel code at a time. Why do we still need separate kernel stacks for each CPU? Describe a scenario in which using a shared kernel stack will go wrong, even with the protection of the big kernel lock.
   答：
   因为在跳跃 lock_kernel 前，已经使用了一些 stack 了，比如说 sysenter 的时候，会保持一些信息到 stack 里面，还有 trap 的时候，cpu 自动会保存信息到 stack 里面，如果使用同样的 stack，那样这些内容就有 data race 了，会出错。
3. In your implementation of env_run() you should have called lcr3(). Before and after the call to lcr3(), your code makes references (at least it should) to the variable e, the argument to env_run. Upon loading the %cr3 register, the addressing context used by the MMU is instantly changed. But a virtual address (namely e) has meaning relative to a given address context--the address context specifies the physical address to which the virtual address maps. Why can the pointer e be dereferenced both before and after the addressing switch?
   答：
   因为对于每个 environment，envs 的地址空间是一样的，都 map 到了 UENVS

## Challenge

我实现的是把大锁改成小锁。
主要的思路是：保护那些 cpu 共享的内存或内容，比如 pages，envs，console drives，以及一些逻辑状态。
最重要的是细心。
经过测试，小锁版本是可以正常工作的。

如果不幸发生错误，请联系我，使用大锁版本。

# 遇到的问题和解决方法：

1. 在开启多 cpu 的时候，没有给每个 cpu 设置 sysenter 的配置。因为在前一个 lab 里面，我把 wrmsr 放在了 trap 函数里面了。做这个 lab 的时候根本忘了有这件事。
   于是开启多 cpu 后跑，基本都是 General Protect 这个错误。
   在经过不断调试后，把错误定位到了 sysenter，然后发现自己没有给每个 cpu 设置 wrmsr。
2. 在做 copy on write 的时候，先 map page 到 parent，然后在 map 到 child。于是怎么也过不了。后来机缘巧合，改了一下顺序，竟然过了。经过仔细分析，如果先 map 到 parent，那么在 map 到 child 前，由于有 stack 的访问，那个 page 会立即从 COW 变成 W，然后在 map 到 child。
   这是，parent 那里是 W 权限，child 那里是 COW。于是 parent 可以尽情的修改内存，导致 child 读到的 stack 内容是错误的了。
3. 在处理 timer interruption 的时候，遇到 timer 不再产生的问题。没有思路了，只能狂调试。最后把错误源锁定在 sysenter。查看了 intel 的手册，发现 sysenter 会清掉 IF，而 sysexit 不会把它恢复。我的第一个想法是用 pushf 和 popf 再用户态恢复 IF。然后发现没用。接着又是调试啊调试，发现 popf 的内容和 pushf 是的值不一样。查手册后发现，用户态 popf 不能修改 IF 的值。于是我把这个放到了 sysexit 前。又遇到了问题：由于在内核态开启了 IF，就遇到了内核态 timer interruption 了。经过对其的单独处理后，程序能正常工作了。后来觉得这个解决方法太丑陋，想到 env_run 是个很好的返回用户态并同时设 IF 的过程，于是我用 env_run(curenv)代替 sysexit 来进行返回用户态的工作。