OSLAB实验报告

LAB1

调的最久的一集。使用的模式是buddy系统+slab。最开始不明白题意，以为那句“我们规定你使用的所有静态内存 (64-bit 模式下的代码、数据、bss) 不能超过 1 MiB”的意思就是要用malloc，而我发现在qemu-x64上用的是libc，malloc没有实现，自己写了个malloc，误打误撞地使用了heap中的地址，然后误打误撞地在qemu-x64环境下正确了。但是在native环境下，我发现不同cpu之间的操作似乎是独立的，也就是全局的buddy系统不起作用，请教了助教之后，发现native实现多线程似乎是用fork实现的，不同cpu都是fork出来的，malloc开出来的地址空间当然也是独立的（在native环境下malloc应该是一个未定义行为，但是OJ并不会识别）。当时还不知道怎么处理这个问题，于是我又修改了实现：在初始化时把地址空间平均分配给不同cpu的buddy系统，让每个cpu各玩各的，当然这在OJ上是过不了的。后来发现好像需要用heap规定的空间，因为只有这一部分空间在fork时是映射操作，也就是父子进程对heap内内存的修改共享。第二天上课的时候请教了jyy，发现我对锁的理解有点问题，spinlock.h中实现的互斥锁，并不是像课上讲的那样直观（拿cpu手上的值和lock的“变量”去交换），在代码中并没有找到这样的一个“变量”，而是类似“占位”，一个cpu把某个锁设成了1，就意味着锁被”占了“，直到这个cpu再把值设成0之前，其他cpu都会看到锁被”占了“。又改又调，到最后卡着时限通过，但还是有一点点瑕疵（我的buddy系统不会合并已经分配又释放的一页大小的内存，以此来提高分配4KB大小页的速度）。总之在一系列DDoS式提交过后终于过了，感谢jyy和助教不厌其烦的指导。

Lab2

关于内核线程调度和锁的实验。开始打代码之前先看了内核栈的相关内容，然后仔细读了一遍AM规约，就里面几个问题请教了助教之后，决定使用每个CPU各开一个RUNNALBE线程的链表来实现线程的调度。一是可以实现先进先出的调度，减少线程饥饿的情况，二是方便CPU之间的线程调度，减少CPU饥饿。由于要实现自旋锁和信号量的队列机制，所以一开始想的是只实现信号量，然后把自旋锁设置成value为1的信号量，实现一鱼两吃。但是再调试device时有奇怪的bug，就把自旋锁改成了jyy提供的版本，自带开关中断的push_off()和pop_off()。改完之后还是有bug，虚拟机不断神秘重启，然后返回了各种各样的AM Panic和assert fail。把push_off()和pop_off()改成直接开关中断的iset(false)和iset(true)，过后就奇怪的通过了。按理说不应该直接简单粗暴地开关中断，因为可能出现锁套锁地情况。助教的说法是框架代码提供的API cpu_current()没有被锁保护，所以会出现bug。希望是如此吧，不想再调再改了。