5140379041
马嘉诚

Lab：
完成了对多 CPU 和多线程的支持，包括多 CPU 的启动，多线程的简单调度，lock，时钟中断，以及 IPC 和 user mode handler 等。在编码工程中大部分时间用于 debug，找到若干个来自 lab1，lab2 和 lab3 的 bug。
在第一个 CPU 进入保护模式后，建立若干个 Idle 的 Env，同时叫醒其它 CPU，所有 CPU 在进入第一次 yield 时，去 Env list 中选取能够被执行的，进入 User 模式。当一个 Env 进入 kernel 时，会请求 big kernel lock，因此在同一时刻只有一个 Env 运行在 kernel 中。在用户态程序手动进行 yield 系统调用时，kernel 会选择另一个可用的 Env 放入 User 模式中执行。总共有两种方法能进入 kernel，分别为 sysenter 何 trap。trap 会在进入 ring0 时保存调用栈，而 sysenter 需要手动保存一个相同的结构体，因为 yield 可能会通过 yield 的方式出 kernel。
Fork 会通过 user mode handler 的方式实现。实现一个 user mode 的 page fault handler，以及 map 和 alloc page 的系统调用，将 fork 出的 child 和 parent 先设置成写保护，当发生 page fault 时，手动 map 和复制相应的 page。
时钟终端通过 irq0 进入 kernel，每次时钟中断都会默认发生一次 yield。
IPC 采用了简单的模型，发送者不断尝试发送，直到发现接收者。由发送者将 message 写入接受者的数据结构，而接受者在发出接收请求后就将自己置为 not runnable，等待被发送者叫醒。

Qusetions：
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? Hint: recall the differences between the link address and the load address that we have discussed in Lab 1.
MPBOOTTPHYS 将高地址转换为物理地址。在运行到此处时分段还没开启，如果不转换为物理地址会无法跳转到 start32

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.
进入 kernel 时会 push trapframe，出 kernel 时会 pop trapframe，但在 pop 时，存在一个小的时间片，此时 lock 已经释放，但是 pop 还没有完成。此时，另一个 core 上的程序可以进入 kernel，如果使用共享的 stack，则会破坏 trapframe。

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?
Kernel 地址在所有的 page table 中的 mapping 都是一样的。所以，只要 e 在 kernel 地址中，则在切换 cr3 前后均能用同一个 virtual address 访问到。

Challenge：
Extend your kernel so that not only page faults, but *all* types of processor exceptions that code running in user space can generate, can be redirected to a user-mode exception handler. Write user-mode test programs to test user-mode handling of various exceptions such as divide-by-zero, general protection fault, and illegal opcode.
在 kern/syscall.c 和 inc/syscall.c 中定义 sys_env_set_user_fault_handler(envid_t envid, int faultid, void* func)，传入 envid，fault 类型和函数指针，更新 Env 中对应的项。在 lib 中，增加了类似于_pgfault_upcall, _pgfault_handler 等的定义，并且给出了定义相应的 user mode handler 的接口。在 user 文件夹中给出了 demo：divzerofault.c，该程序会注册一个 handler，并在发生除 0 错时打印一条信息，并修改全局变量。