Lab4

516030910101 罗宇辰

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?

`MPBOOTPHYS`是用来计算gdt的物理地址的，因为此时AP的页表还没开启，所以必须使用物理地址

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.

当CPU0拿到锁正在使用kernel stack的时候，如果CPU1收到一个中断，CPU1就会在不拿锁的情况下向kernel stack push一些参数，就会影响CPU0对栈的使用

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?

因为`e`被存储在内核中，而不同的`env_pgdir`的kernel都是映射到同一位置的，所以切换页表不会对`e`产生影响

4.

Whenever the kernel switches from one environment to another, it must ensure the old environment's registers are saved so they can be restored properly later. Why? Where does this happen?

env的registers被存储在env_tf中，切换时在调用trap之前把old env的tf保存在栈上：

```
_alltraps:
    # Build trap frame
    pushl %ds    # tf_ds
    pushl %es    # tf_es
    pushal
    # Set up data segments
    movl $GD_KD, %eax
    movw %ax, %ds
    movw %ax, %es

    # Call trap(tf), where tf=%esp
    pushl %esp
    call trap
```

运行新的env的时候，把新env的tf从栈中pop出来：

```
void
env_pop_tf(struct Trapframe *tf)
{
    // Record the CPU we are running on for user-space debugging
    curenv->env_cpunum = cpunum();

    asm volatile(
        "\tmovl %0,%%esp\n"
        "\tpopal\n"
        "\tpopl %%es\n"
        "\tpopl %%ds\n"
        "\taddl $0x8,%%esp\n" /* skip tf_trapno and tf_errcode */
        "\tiret\n"
        : : "g" (tf) : "memory");
```

Challenge

> Why does `ipc_send` have to loop? Change the system call interface so it doesn't have to. Make sure you can handle multiple environments trying to send to one environment at the same time.

将`ipc_send`中的while循环改为if判断：

```
void
ipc_send(envid_t to_env, uint32_t val, void *pg, int perm)
{
    // LAB 4: Your code here.
    if (!pg)
        pg = (void*)UTOP;
    int r;
    // while((r = sys_ipc_try_send(to_env, val, pg, perm)) < 0) {
    //  if(r != -E_IPC_NOT_RECV)
    //      panic("sys_ipc_try_send %e", r);
    //  sys_yield();
    // }
    if((r = sys_ipc_try_send(to_env, val, pg, perm)) < 0) {
        if(r != -E_IPC_NOT_RECV)
            panic("sys_ipc_try_send %e", r);
        sys_yield();
    }
}
```

在`sys_ipc_try_send`中，之前当`e->env_ipc_recving`为0时返回了E_IPC_NOT_RECV错误码:

```
sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva, unsigned perm)
{
    // LAB 4: Your code here.
    struct Env *e;
    int r;
    if(envid2env(envid, &e, 0)  < 0)
        return -E_BAD_ENV;
    // if(!e->env_ipc_recving)   //注释掉之前的return
    //  return -E_IPC_NOT_RECV;
    ...
}
```

现在要实现message的pending，就要在`struct Env`中加入一些变量来保存传送的数据和参数:

```
struct Env {
    ...

    // Lab4 challenge
    struct PageInfo *env_pending_page; // received page
    uint32_t env_ipc_pending_value;     // Data value sent to us
    envid_t env_ipc_pending_to;     // envid of the receiver
    int env_ipc_pending_perm;       // Perm of page mapping received
};
```

在`sys_ipc_try_send`中加入对pending情况的处理:

```
static int
sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva, unsigned perm)
{
    // LAB 4: Your code here.
    ...
    if (e->env_ipc_recving){
        ... //env_ipc_recving为真时按照正常流程send
        return 0;
    }
    else{   //pending
        curenv->env_pending_page = NULL;
        if(srcva < (void*)UTOP ){
            if(PGOFF(srcva)){
                return -E_INVAL;
            }

            if(!(perm & PTE_U) || !(perm & PTE_P) || (perm & ~PTE_SYSCALL))
                return -E_INVAL;

            pte_t *pte;
            struct PageInfo *pp = page_lookup(curenv->env_pgdir, srcva, &pte);
            if(!pp)
                return -E_INVAL;
            if ((perm & PTE_W) && !(*pte & PTE_W))
                return -E_INVAL;

            curenv->env_pending_page = pp;  // 保存需要被map的page
        }

        curenv->env_ipc_pending_perm = perm;    // 保存perm
        curenv->env_ipc_pending_to = envid;     // 保存receiver的id，才能在recv中查找pending的
message
        curenv->env_ipc_pending_value = value;  // 保存value
        curenv->env_status = ENV_NOT_RUNNABLE;  // 把sender block住，直到recv成功
        sched_yield();  // 释放CPU
        return 0;
    }
}
```

在`sys_ipc_recv`中遍历所有env，查找是否有发送给当前env的被pending的message：

```
static int
sys_ipc_recv(void *dstva)
{
    // LAB 4: Your code here.
    ...
    for (int i = 0; i < NENV; i++) {
        struct Env* env = &envs[i];
        if (env->env_status == ENV_NOT_RUNNABLE && env->env_ipc_pending_to == curenv-
>env_id) { // 有env给自己发消息被pending了
            if (env->env_pending_page && dstva < (void*)UTOP) { // 需要map
                if (page_insert(curenv->env_pgdir, env->env_pending_page, dstva, env-
>env_ipc_pending_perm) < 0)
                    return -E_NO_MEM;
            }
            // 更新curenv的ipc参数
            curenv->env_ipc_recving = 0;
            curenv->env_ipc_from = env->env_id;
            curenv->env_ipc_value = env->env_ipc_pending_value;
            curenv->env_ipc_perm = env->env_ipc_pending_perm;
            // 恢复sender的status
            env->env_ipc_pending_to = -1;
            env->env_status = ENV_RUNNABLE;
            env->env_tf.tf_regs.reg_eax = 0;
            return 0;
        }
    }
    ...
}
```