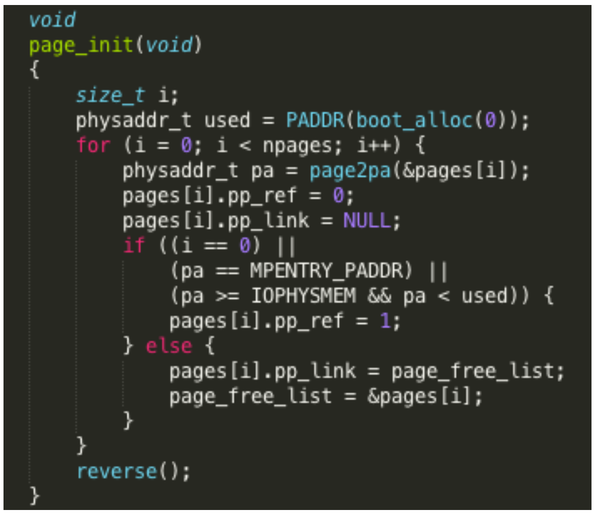
Lab 4 实验报告

姓名：李浩宇

学号：5130809070

Exercise 1. Read boot_aps() and mp_main() in kern/init.c, and the assembly code in kern/mpentry.S. Make sure you understand the control flow transfer during the bootstrap of APs. Then modify your implementation of page_init() in kern/pmap.c to avoid adding the page at MPENTRY_PADDR to the free list, so that we can safely copy and run AP bootstrap code at that physical address.

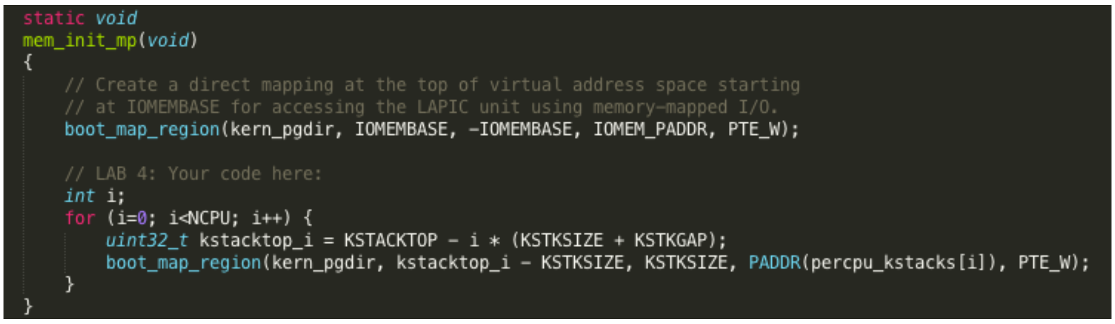
在 kern/pmap.c 原有的 page_init()基础上新增一行判断即可：



```
void
page_init(void)
{
    size_t i;
    physaddr_t used = PADDR(boot_alloc(0));
    for (i = 0; i < npages; i++) {
        physaddr_t pa = page2pa(&pages[i]);
        pages[i].pp_ref = 0;
        pages[i].pp_link = NULL;
        if ((i == 0) ||
            (pa == MPENTRY_PADDR) ||
            (pa >= IOPHYSMEM && pa < used)) {
            pages[i].pp_ref = 1;
        } else {
            pages[i].pp_link = page_free_list;
            page_free_list = &pages[i];
        }
    }
    reverse();
}
```

Exercise 2. Modify mem_init_mp() (in kern/pmap.c) to map per-CPU stacks starting at KSTACKTOP, as shown in our revised inc/memlayout.h. The size of each stack is KSTKSIZE bytes plus KSTKGAP bytes of unmapped guard pages.

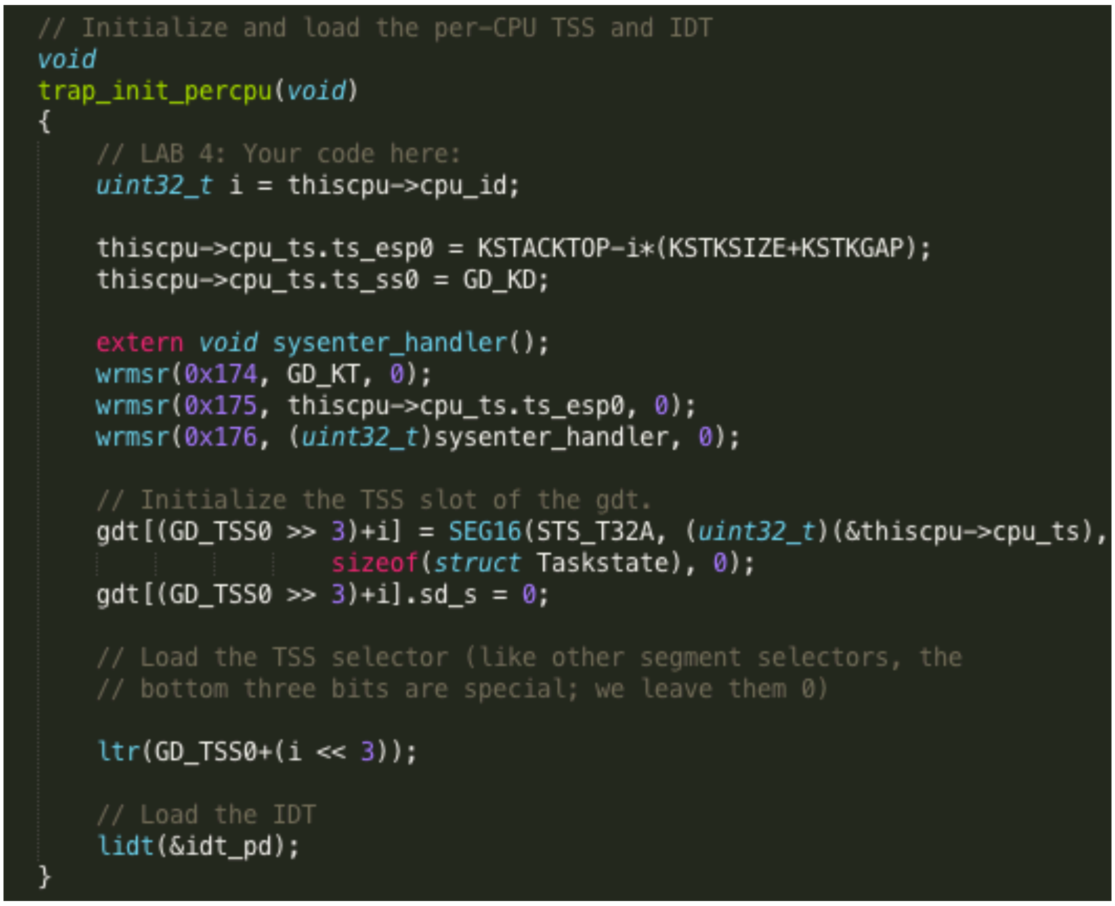
按照注释中的提示，为每个 CPU 分配栈空间：



```
static void
mem_init_mp(void)
{
    // Create a direct mapping at the top of virtual address space starting
    // at IOMEMBASE for accessing the LAPIC unit using memory-mapped I/O.
    boot_map_region(kern_pgdir, IOMEMBASE, -IOMEMBASE, IOMEM_PADDR, PTE_W);

    // LAB 4: Your code here:
    int i;
    for (i=0; i<NCPU; i++) {
        uint32_t kstacktop_i = KSTACKTOP - i * (KSTKSIZE + KSTKGAP);
        boot_map_region(kern_pgdir, kstacktop_i - KSTKSIZE, KSTKSIZE, PADDR(percpu_kstacks[i]), PTE_W);
    }
}
```

Exercise 3. The code in trap_init_percpu() (kern/trap.c) initializes the TSS and TSS descriptor for the BSP. It worked in Lab 3, but is incorrect when running on other CPUs. Change the code so that it can work on all CPUs.

设置好 cpu_ts 的相关变量：



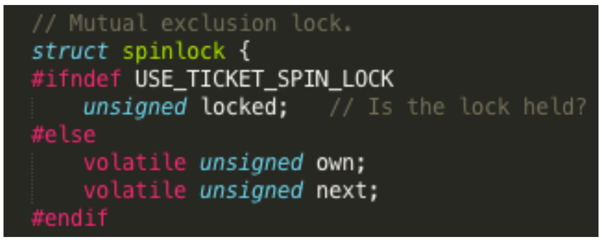
```c
// Initialize and load the per-CPU TSS and IDT
void
trap_init_percpu(void)
{
    // LAB 4: Your code here:
    uint32_t i = thiscpu->cpu_id;

    thiscpu->cpu_ts.ts_esp0 = KSTACKTOP-i*(KSTKSIZE+KSTKGAP);
    thiscpu->cpu_ts.ts_ss0 = GD_KD;

    extern void sysenter_handler();
    wrmsr(0x174, GD_KT, 0);
    wrmsr(0x175, thiscpu->cpu_ts.ts_esp0, 0);
    wrmsr(0x176, (uint32_t)sysenter_handler, 0);

    // Initialize the TSS slot of the gdt.
    gdt[(GD_TSS0 >> 3)+i] = SEG16(STS_T32A, (uint32_t)(&thiscpu->cpu_ts),
                    sizeof(struct Taskstate), 0);
    gdt[(GD_TSS0 >> 3)+i].sd_s = 0;

    // Load the TSS selector (like other segment selectors, the
    // bottom three bits are special; we leave them 0)

    ltr(GD_TSS0+(i << 3));

    // Load the IDT
    lidt(&idt_pd);
}
```

Exercise 4. Apply the big kernel lock as described above, by calling lock_kernel() and unlock_kernel() at the proper locations.

在提示位置加锁即可。

Exercise 4.1. Implement ticket spinlock in kern/spinlock.c. You can define a macro USE_TICKET_SPIN_LOCK at the beginning of kern/spinlock.h to make it work.

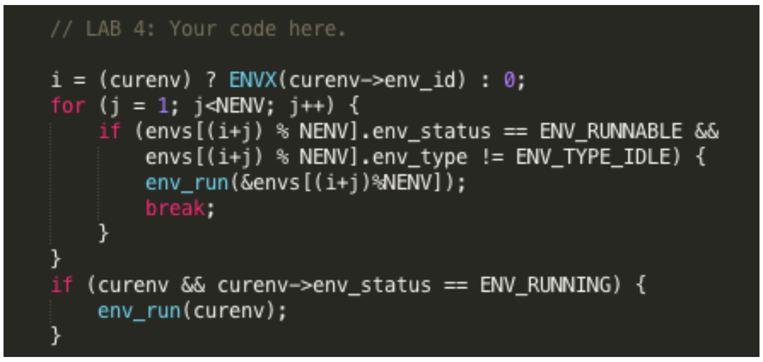
理解了 spinlock 之后，其实现非常地 simple and straightforward，但在这里我却遇到了整个 lab 中最难的一个 bug，如下图所示，需要在 spinlock.h 中将 own 和 next 改为 volatile 的变量，否则编译器会在等锁的 while 循环中，将其优化至寄存器中，从而当锁已被释放时无法察觉，产生“死锁”的现象。



```c
// Mutual exclusion lock.
struct spinlock {
#ifndef USE_TICKET_SPIN_LOCK
    unsigned locked;    // Is the lock held?
#else
    volatile unsigned own;
    volatile unsigned next;
#endif
```

Exercise 5. Implement round-robin scheduling in sched_yield() as described above.

按照 round-robin 遍历 envs 数组即可：

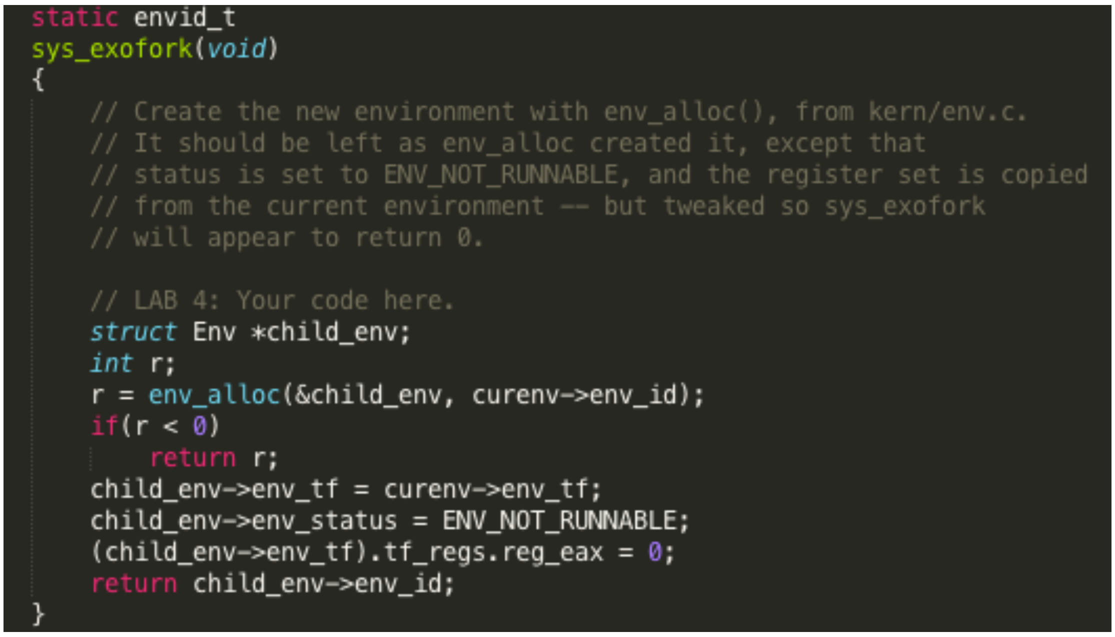


```c
    // LAB 4: Your code here.

    i = (curenv) ? ENVX(curenv->env_id) : 0;
    for (j = 1; j<NENV; j++) {
        if (envs[(i+j) % NENV].env_status == ENV_RUNNABLE &&
            envs[(i+j) % NENV].env_type != ENV_TYPE_IDLE) {
            env_run(&envs[(i+j)%NENV]);
            break;
        }
    }
    if (curenv && curenv->env_status == ENV_RUNNING) {
        env_run(curenv);
    }
```

Exercise 6. Implement the system calls described above in kern/syscall.c.

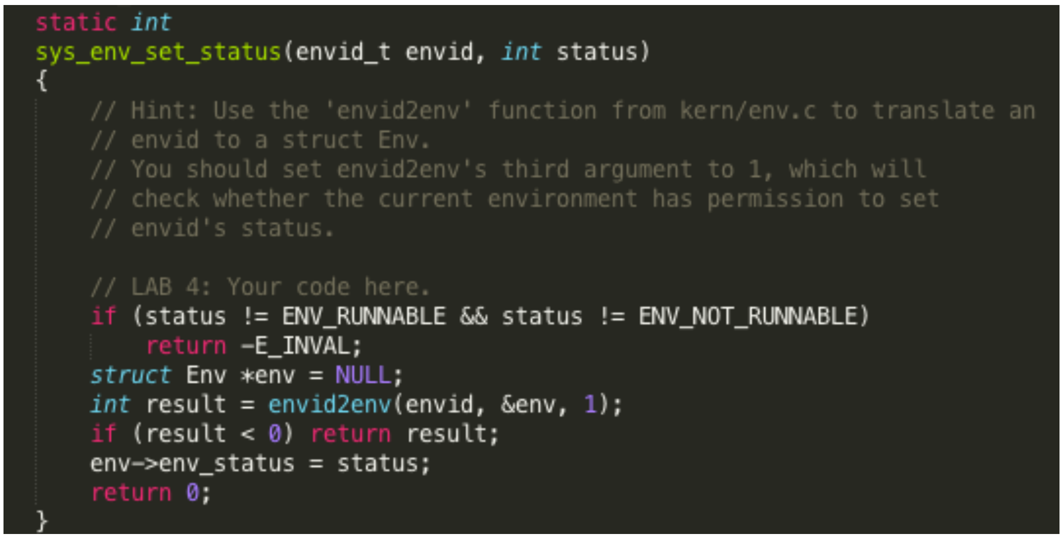
这个练习需要实现五个 fork 相关的 syscall，注释和文档中描述得都非常详细，实现起来没有什么困难，具体代码如下：

sys_exofork



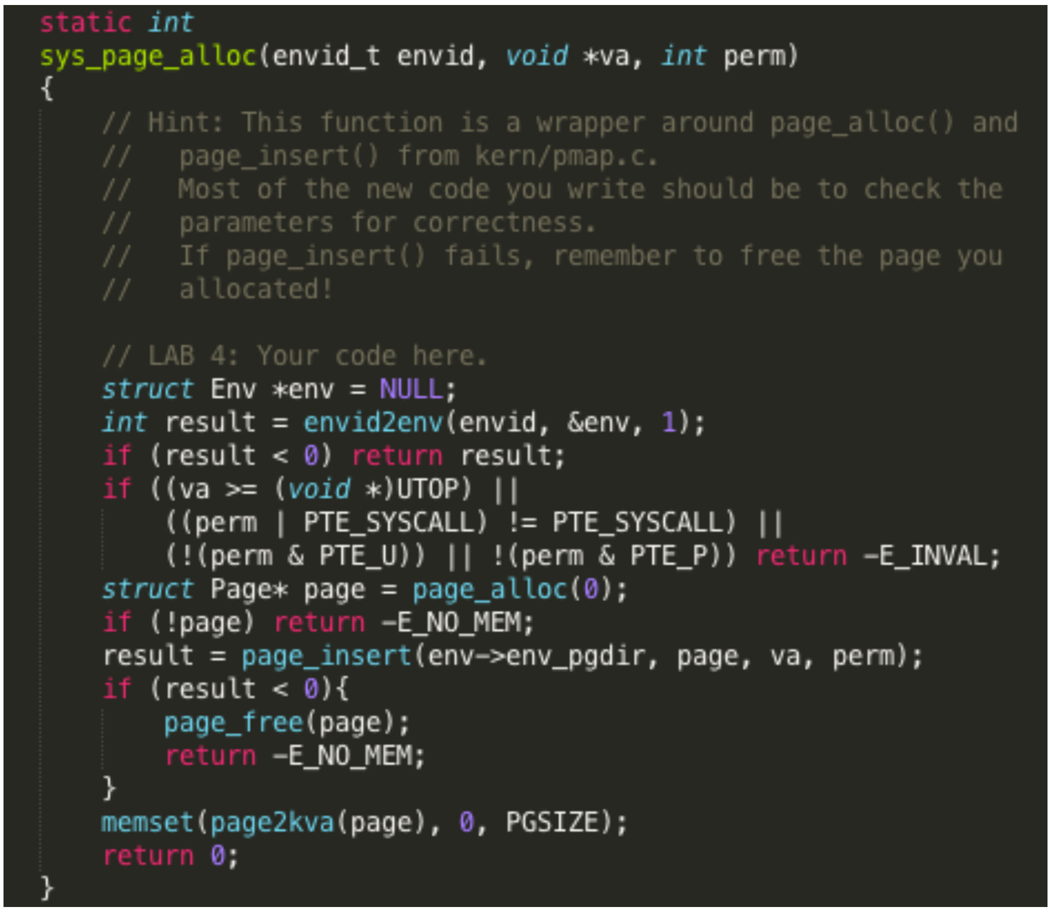
```c
static envid_t
sys_exofork(void)
{
    // Create the new environment with env_alloc(), from kern/env.c.
    // It should be left as env_alloc created it, except that
    // status is set to ENV_NOT_RUNNABLE, and the register set is copied
    // from the current environment -- but tweaked so sys_exofork
    // will appear to return 0.

    // LAB 4: Your code here.
    struct Env *child_env;
    int r;
    r = env_alloc(&child_env, curenv->env_id);
    if(r < 0)
        return r;
    child_env->env_tf = curenv->env_tf;
    child_env->env_status = ENV_NOT_RUNNABLE;
    (child_env->env_tf).tf_regs.reg_eax = 0;
    return child_env->env_id;
}
```

sys_env_set_status



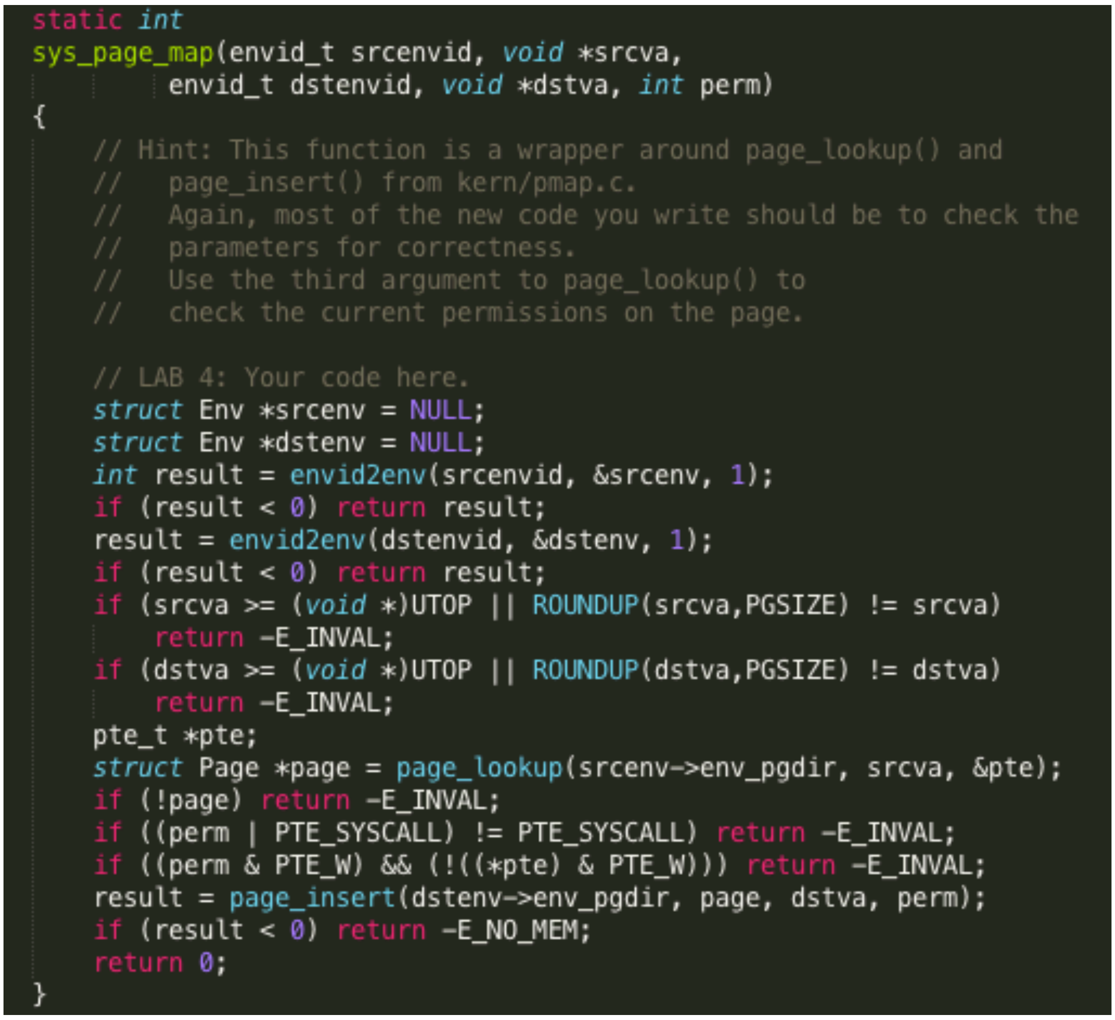
```c
static int
sys_env_set_status(envid_t envid, int status)
{
    // Hint: Use the 'envid2env' function from kern/env.c to translate an
    // envid to a struct Env.
    // You should set envid2env's third argument to 1, which will
    // check whether the current environment has permission to set
    // envid's status.

    // LAB 4: Your code here.
    if (status != ENV_RUNNABLE && status != ENV_NOT_RUNNABLE)
        return -E_INVAL;
    struct Env *env = NULL;
    int result = envid2env(envid, &env, 1);
    if (result < 0) return result;
    env->env_status = status;
    return 0;
}
```

sys_page_alloc



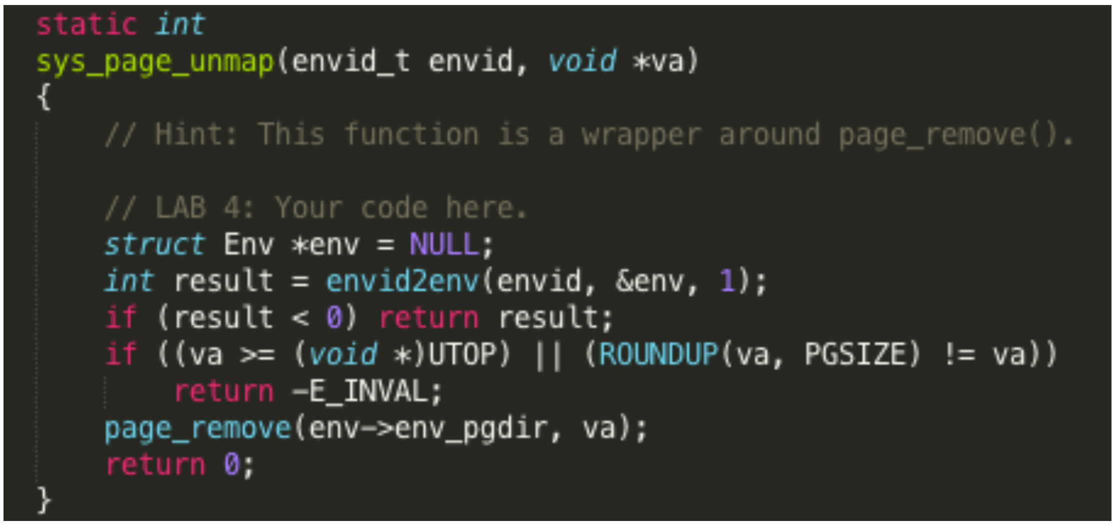
```c
static int
sys_page_alloc(envid_t envid, void *va, int perm)
{
    // Hint: This function is a wrapper around page_alloc() and
    //   page_insert() from kern/pmap.c.
    //   Most of the new code you write should be to check the
    //   parameters for correctness.
    //   If page_insert() fails, remember to free the page you
    //   allocated!

    // LAB 4: Your code here.
    struct Env *env = NULL;
    int result = envid2env(envid, &env, 1);
    if (result < 0) return result;
    if ((va >= (void *)UTOP) ||
        ((perm | PTE_SYSCALL) != PTE_SYSCALL) ||
        (!(perm & PTE_U)) || !(perm & PTE_P)) return -E_INVAL;
    struct Page* page = page_alloc(0);
    if (!page) return -E_NO_MEM;
    result = page_insert(env->env_pgdir, page, va, perm);
    if (result < 0){
        page_free(page);
        return -E_NO_MEM;
    }
    memset(page2kva(page), 0, PGSIZE);
    return 0;
}
```

sys_page_map



```c
static int
sys_page_map(envid_t srcenvid, void *srcva,
	     envid_t dstenvid, void *dstva, int perm)
{
	// Hint: This function is a wrapper around page_lookup() and
	//   page_insert() from kern/pmap.c.
	//   Again, most of the new code you write should be to check the
	//   parameters for correctness.
	//   Use the third argument to page_lookup() to
	//   check the current permissions on the page.

	// LAB 4: Your code here.
	struct Env *srcenv = NULL;
	struct Env *dstenv = NULL;
	int result = envid2env(srcenvid, &srcenv, 1);
	if (result < 0) return result;
	result = envid2env(dstenvid, &dstenv, 1);
	if (result < 0) return result;
	if (srcva >= (void *)UTOP || ROUNDUP(srcva,PGSIZE) != srcva)
		return -E_INVAL;
	if (dstva >= (void *)UTOP || ROUNDUP(dstva,PGSIZE) != dstva)
		return -E_INVAL;
	pte_t *pte;
	struct Page *page = page_lookup(srcenv->env_pgdir, srcva, &pte);
	if (!page) return -E_INVAL;
	if ((perm | PTE_SYSCALL) != PTE_SYSCALL) return -E_INVAL;
	if ((perm & PTE_W) && (!((*pte) & PTE_W))) return -E_INVAL;
	result = page_insert(dstenv->env_pgdir, page, dstva, perm);
	if (result < 0) return -E_NO_MEM;
	return 0;
}
```

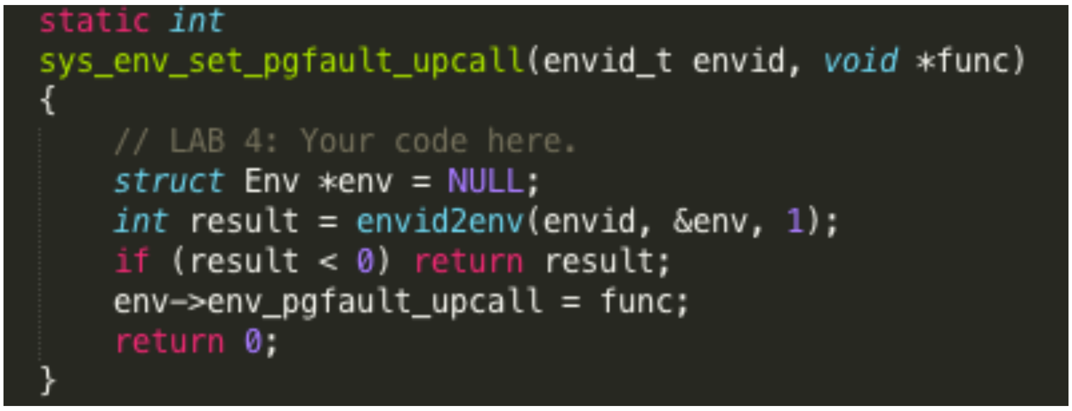
sys_page_unmap



```c
static int
sys_page_unmap(envid_t envid, void *va)
{
	// Hint: This function is a wrapper around page_remove().

	// LAB 4: Your code here.
	struct Env *env = NULL;
	int result = envid2env(envid, &env, 1);
	if (result < 0) return result;
	if ((va >= (void *)UTOP) || (ROUNDUP(va, PGSIZE) != va))
		return -E_INVAL;
	page_remove(env->env_pgdir, va);
	return 0;
}
```

Exercise 7. Implement the sys_env_set_pgfault_upcall system call.

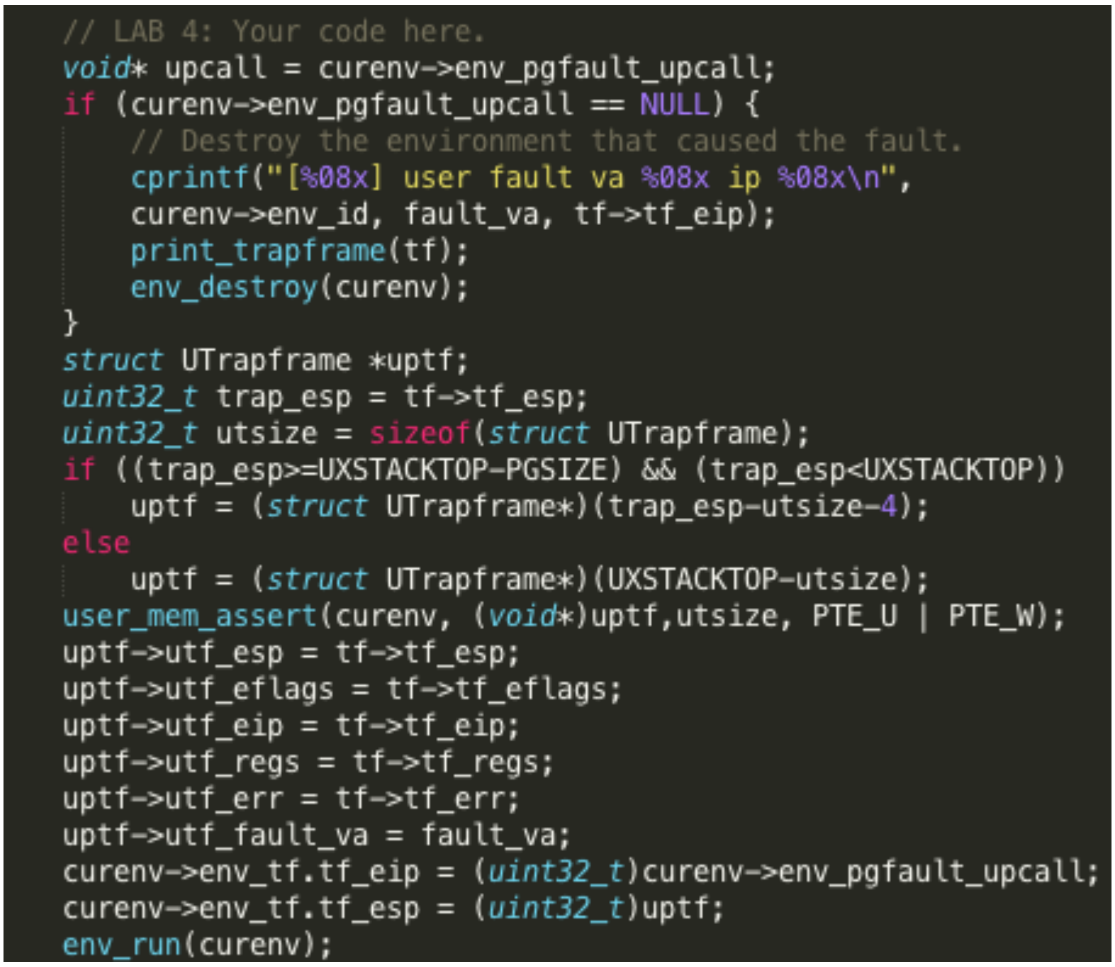
为进程设置自定义的 page fault upcall 函数：



```
static int
sys_env_set_pgfault_upcall(envid_t envid, void *func)
{
    // LAB 4: Your code here.
    struct Env *env = NULL;
    int result = envid2env(envid, &env, 1);
    if (result < 0) return result;
    env->env_pgfault_upcall = func;
    return 0;
}
```

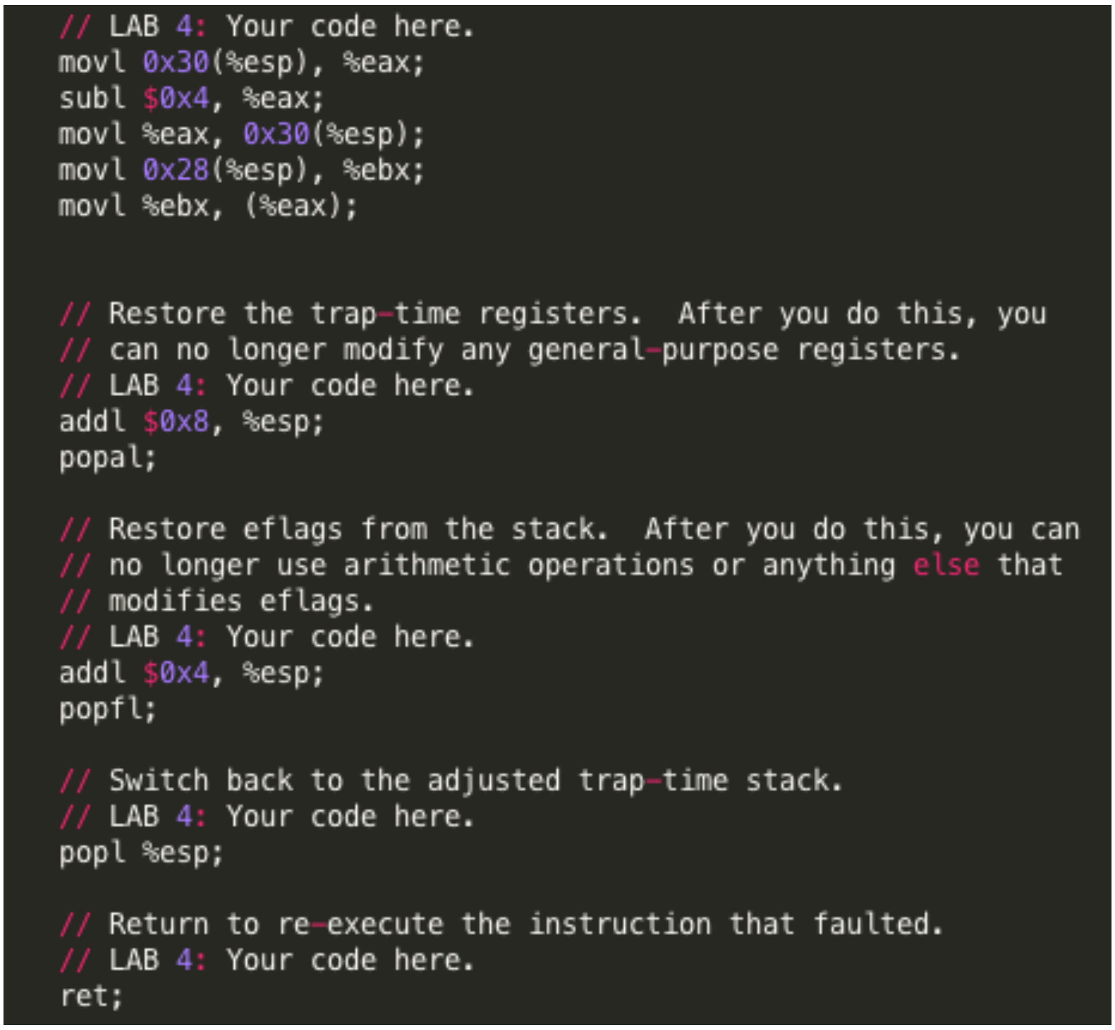
Exercise 8. Implement the code in page_fault_handler in kern/trap.c required to dispatch page faults to the user-mode handler.

如果没有设置处理函数，则直接结束进程，否则检测异常栈，保存原来的信息，继续运行进程：



```
// LAB 4: Your code here.
void* upcall = curenv->env_pgfault_upcall;
if (curenv->env_pgfault_upcall == NULL) {
    // Destroy the environment that caused the fault.
    cprintf("[%08x] user fault va %08x ip %08x\n",
    curenv->env_id, fault_va, tf->tf_eip);
    print_trapframe(tf);
    env_destroy(curenv);
}
struct UTrapframe *uptf;
uint32_t trap_esp = tf->tf_esp;
uint32_t utsize = sizeof(struct UTrapframe);
if ((trap_esp>=UXSTACKTOP-PGSIZE) && (trap_esp<UXSTACKTOP))
    uptf = (struct UTrapframe*)(trap_esp-utsize-4);
else
    uptf = (struct UTrapframe*)(UXSTACKTOP-utsize);
user_mem_assert(curenv, (void*)uptf,utsize, PTE_U | PTE_W);
uptf->utf_esp = tf->tf_esp;
uptf->utf_eflags = tf->tf_eflags;
uptf->utf_eip = tf->tf_eip;
uptf->utf_regs = tf->tf_regs;
uptf->utf_err = tf->tf_err;
uptf->utf_fault_va = fault_va;
curenv->env_tf.tf_eip = (uint32_t)curenv->env_pgfault_upcall;
curenv->env_tf.tf_esp = (uint32_t)uptf;
env_run(curenv);
```

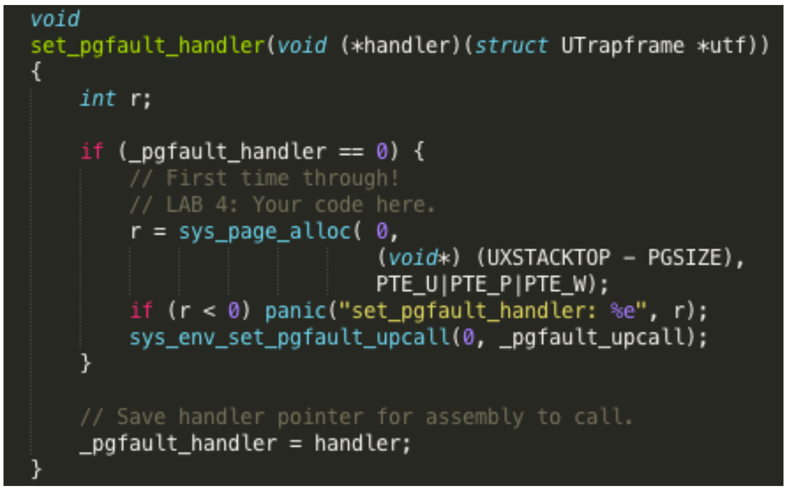
Exercise 9. Implement the _pgfault_upcall routine in lib/pfentry.S.



```
// LAB 4: Your code here.
movl 0x30(%esp), %eax;
subl $0x4, %eax;
movl %eax, 0x30(%esp);
movl 0x28(%esp), %ebx;
movl %ebx, (%eax);


// Restore the trap-time registers.  After you do this, you
// can no longer modify any general-purpose registers.
// LAB 4: Your code here.
addl $0x8, %esp;
popal;

// Restore eflags from the stack.  After you do this, you can
// no longer use arithmetic operations or anything else that
// modifies eflags.
// LAB 4: Your code here.
addl $0x4, %esp;
popfl;

// Switch back to the adjusted trap-time stack.
// LAB 4: Your code here.
popl %esp;

// Return to re-execute the instruction that faulted.
// LAB 4: Your code here.
ret;
```

Exercise 10. Finish set_pgfault_handler() in lib/pgfault.c.

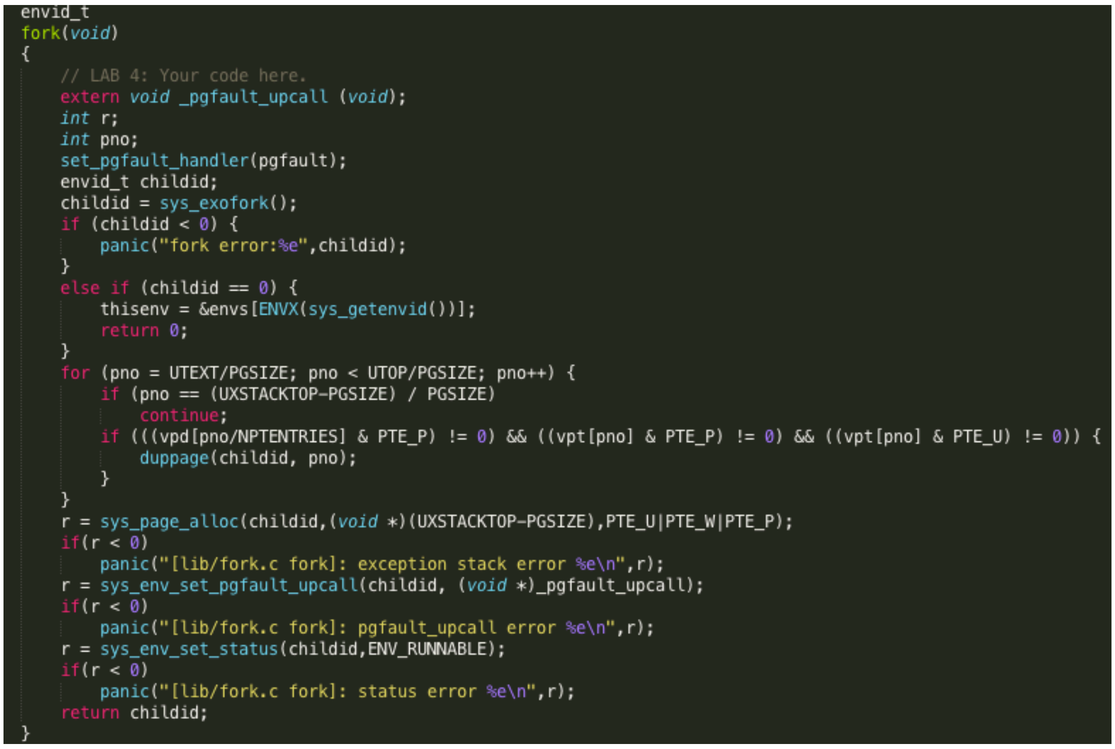
真正地设置 page fault 处理函数，首先需要检测一下_pgfault_handler 是否已经设置，如果没有则先进行空间的分配，然后调用写好的 sys_set_pgfault_upcall 进行设置：



```
void
set_pgfault_handler(void (*handler)(struct UTrapframe *utf))
{
    int r;

    if (_pgfault_handler == 0) {
        // First time through!
        // LAB 4: Your code here.
        r = sys_page_alloc( 0,
                            (void*) (UXSTACKTOP - PGSIZE),
                            PTE_U|PTE_P|PTE_W);
        if (r < 0) panic("set_pgfault_handler: %e", r);
        sys_env_set_pgfault_upcall(0, _pgfault_upcall);
    }

    // Save handler pointer for assembly to call.
    _pgfault_handler = handler;
}
```

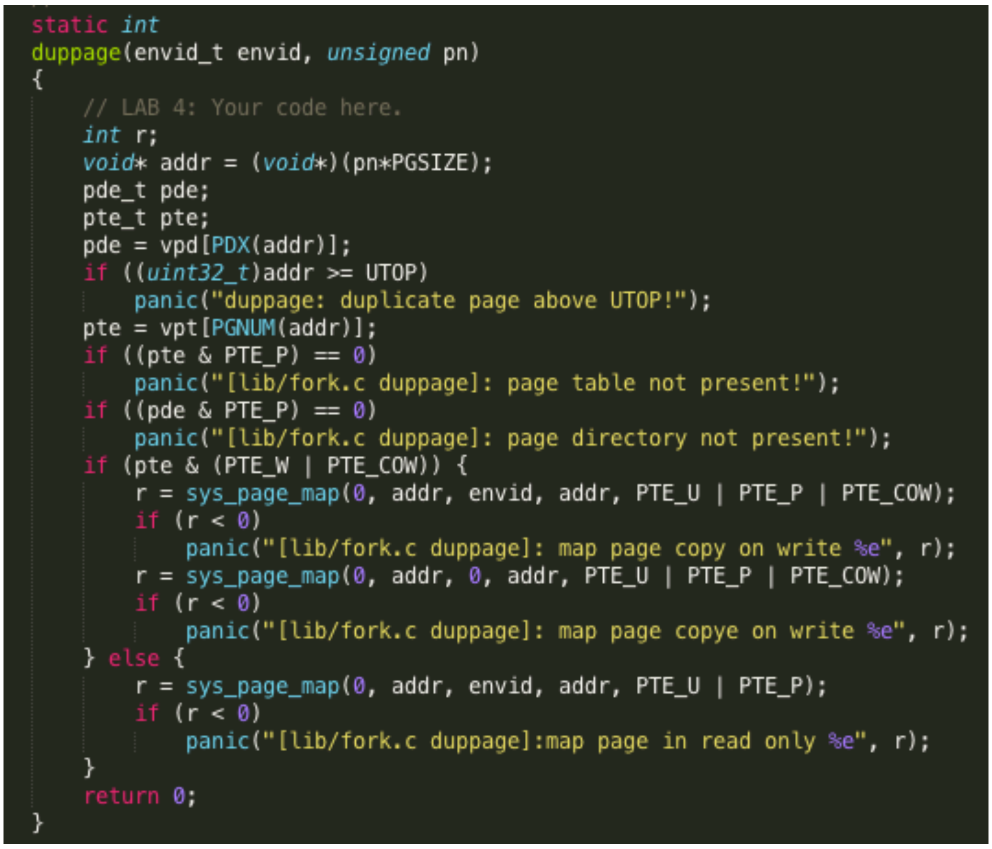
Exercise 11. Implement fork, duppage and pgfault in lib/fork.c.

fork:



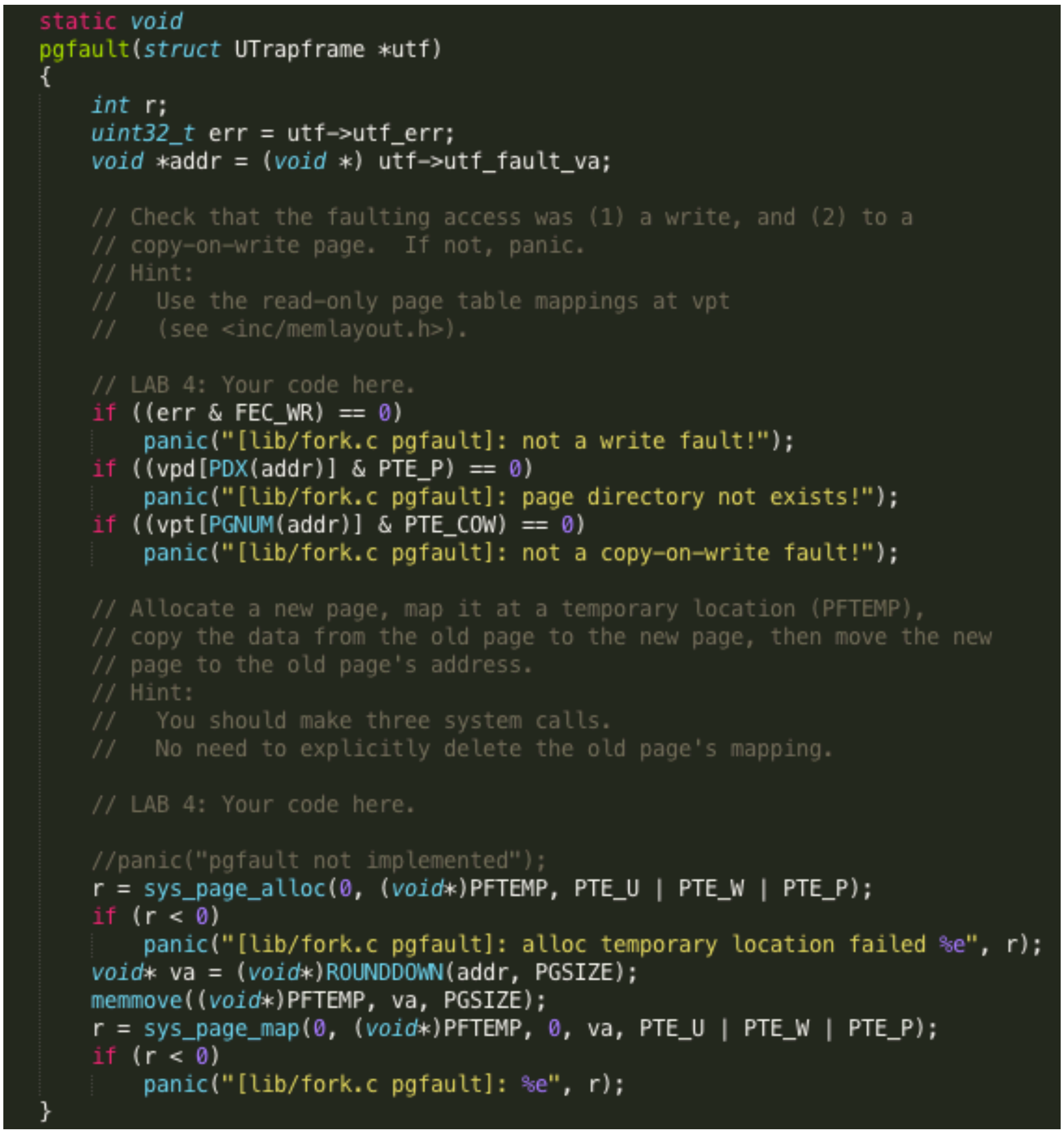
```c
envid_t
fork(void)
{
    // LAB 4: Your code here.
    extern void _pgfault_upcall (void);
    int r;
    int pno;
    set_pgfault_handler(pgfault);
    envid_t childid;
    childid = sys_exofork();
    if (childid < 0) {
        panic("fork error:%e",childid);
    }
    else if (childid == 0) {
        thisenv = &envs[ENVX(sys_getenvid())];
        return 0;
    }
    for (pno = UTEXT/PGSIZE; pno < UTOP/PGSIZE; pno++) {
        if (pno == (UXSTACKTOP-PGSIZE) / PGSIZE)
            continue;
        if (((vpd[pno/NPTENTRIES] & PTE_P) != 0) && ((vpt[pno] & PTE_P) != 0) && ((vpt[pno] & PTE_U) != 0)) {
            duppage(childid, pno);
        }
    }
    r = sys_page_alloc(childid,(void *)(UXSTACKTOP-PGSIZE),PTE_U|PTE_W|PTE_P);
    if(r < 0)
        panic("[lib/fork.c fork]: exception stack error %e\n",r);
    r = sys_env_set_pgfault_upcall(childid, (void *)_pgfault_upcall);
    if(r < 0)
        panic("[lib/fork.c fork]: pgfault_upcall error %e\n",r);
    r = sys_env_set_status(childid,ENV_RUNNABLE);
    if(r < 0)
        panic("[lib/fork.c fork]: status error %e\n",r);
    return childid;
}
```

duppage:



```c
static int
duppage(envid_t envid, unsigned pn)
{
    // LAB 4: Your code here.
    int r;
    void* addr = (void*)(pn*PGSIZE);
    pde_t pde;
    pte_t pte;
    pde = vpd[PDX(addr)];
    if ((uint32_t)addr >= UTOP)
        panic("duppage: duplicate page above UTOP!");
    pte = vpt[PGNUM(addr)];
    if ((pte & PTE_P) == 0)
        panic("[lib/fork.c duppage]: page table not present!");
    if ((pde & PTE_P) == 0)
        panic("[lib/fork.c duppage]: page directory not present!");
    if (pte & (PTE_W | PTE_COW)) {
        r = sys_page_map(0, addr, envid, addr, PTE_U | PTE_P | PTE_COW);
        if (r < 0)
            panic("[lib/fork.c duppage]: map page copy on write %e", r);
        r = sys_page_map(0, addr, 0, addr, PTE_U | PTE_P | PTE_COW);
        if (r < 0)
            panic("[lib/fork.c duppage]: map page copye on write %e", r);
    } else {
        r = sys_page_map(0, addr, envid, addr, PTE_U | PTE_P);
        if (r < 0)
            panic("[lib/fork.c duppage]:map page in read only %e", r);
    }
    return 0;
}
```

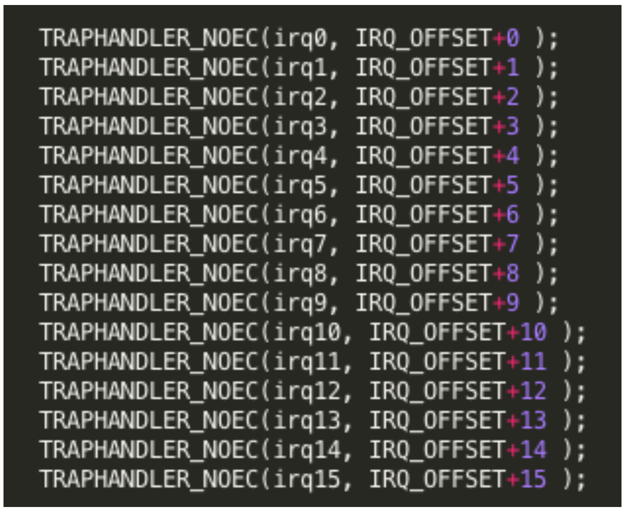
pgfault



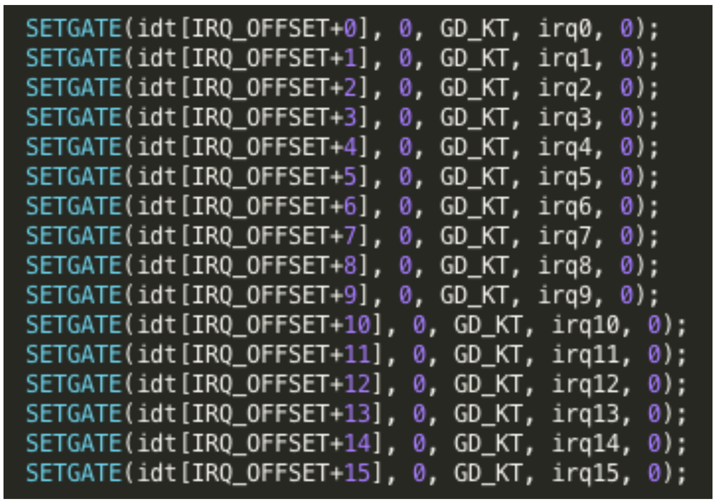
```c
static void
pgfault(struct UTrapframe *utf)
{
    int r;
    uint32_t err = utf->utf_err;
    void *addr = (void *) utf->utf_fault_va;

    // Check that the faulting access was (1) a write, and (2) to a
    // copy-on-write page.  If not, panic.
    // Hint:
    //   Use the read-only page table mappings at vpt
    //   (see <inc/memlayout.h>).

    // LAB 4: Your code here.
    if ((err & FEC_WR) == 0)
        panic("[lib/fork.c pgfault]: not a write fault!");
    if ((vpd[PDX(addr)] & PTE_P) == 0)
        panic("[lib/fork.c pgfault]: page directory not exists!");
    if ((vpt[PGNUM(addr)] & PTE_COW) == 0)
        panic("[lib/fork.c pgfault]: not a copy-on-write fault!");

    // Allocate a new page, map it at a temporary location (PFTEMP),
    // copy the data from the old page to the new page, then move the new
    // page to the old page's address.
    // Hint:
    //   You should make three system calls.
    //   No need to explicitly delete the old page's mapping.

    // LAB 4: Your code here.

    //panic("pgfault not implemented");
    r = sys_page_alloc(0, (void*)PFTEMP, PTE_U | PTE_W | PTE_P);
    if (r < 0)
        panic("[lib/fork.c pgfault]: alloc temporary location failed %e", r);
    void* va = (void*)ROUNDDOWN(addr, PGSIZE);
    memmove((void*)PFTEMP, va, PGSIZE);
    r = sys_page_map(0, (void*)PFTEMP, 0, va, PTE_U | PTE_W | PTE_P);
    if (r < 0)
        panic("[lib/fork.c pgfault]: %e", r);
}
```

Exercise 12. Modify kern/trapentry.S and kern/trap.c to initialize the appropriate entries in the IDT and provide handlers for IRQs 0 through 15. Then modify the code in env_alloc() in kern/env.c to ensure that user environmen

和 lab3 中添加中断的方式一样，在 kern/trapentry.S 和 kern/trap.c 中注册 IRQ：

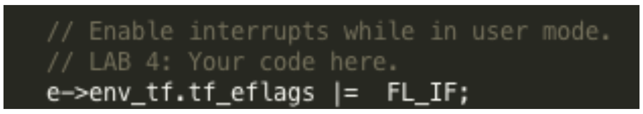


```
TRAPHANDLER_NOEC(irq0, IRQ_OFFSET+0 );
TRAPHANDLER_NOEC(irq1, IRQ_OFFSET+1 );
TRAPHANDLER_NOEC(irq2, IRQ_OFFSET+2 );
TRAPHANDLER_NOEC(irq3, IRQ_OFFSET+3 );
TRAPHANDLER_NOEC(irq4, IRQ_OFFSET+4 );
TRAPHANDLER_NOEC(irq5, IRQ_OFFSET+5 );
TRAPHANDLER_NOEC(irq6, IRQ_OFFSET+6 );
TRAPHANDLER_NOEC(irq7, IRQ_OFFSET+7 );
TRAPHANDLER_NOEC(irq8, IRQ_OFFSET+8 );
TRAPHANDLER_NOEC(irq9, IRQ_OFFSET+9 );
TRAPHANDLER_NOEC(irq10, IRQ_OFFSET+10 );
TRAPHANDLER_NOEC(irq11, IRQ_OFFSET+11 );
TRAPHANDLER_NOEC(irq12, IRQ_OFFSET+12 );
TRAPHANDLER_NOEC(irq13, IRQ_OFFSET+13 );
TRAPHANDLER_NOEC(irq14, IRQ_OFFSET+14 );
TRAPHANDLER_NOEC(irq15, IRQ_OFFSET+15 );
```



```
SETGATE(idt[IRQ_OFFSET+0], 0, GD_KT, irq0, 0);
SETGATE(idt[IRQ_OFFSET+1], 0, GD_KT, irq1, 0);
SETGATE(idt[IRQ_OFFSET+2], 0, GD_KT, irq2, 0);
SETGATE(idt[IRQ_OFFSET+3], 0, GD_KT, irq3, 0);
SETGATE(idt[IRQ_OFFSET+4], 0, GD_KT, irq4, 0);
SETGATE(idt[IRQ_OFFSET+5], 0, GD_KT, irq5, 0);
SETGATE(idt[IRQ_OFFSET+6], 0, GD_KT, irq6, 0);
SETGATE(idt[IRQ_OFFSET+7], 0, GD_KT, irq7, 0);
SETGATE(idt[IRQ_OFFSET+8], 0, GD_KT, irq8, 0);
SETGATE(idt[IRQ_OFFSET+9], 0, GD_KT, irq9, 0);
SETGATE(idt[IRQ_OFFSET+10], 0, GD_KT, irq10, 0);
SETGATE(idt[IRQ_OFFSET+11], 0, GD_KT, irq11, 0);
SETGATE(idt[IRQ_OFFSET+12], 0, GD_KT, irq12, 0);
SETGATE(idt[IRQ_OFFSET+13], 0, GD_KT, irq13, 0);
SETGATE(idt[IRQ_OFFSET+14], 0, GD_KT, irq14, 0);
SETGATE(idt[IRQ_OFFSET+15], 0, GD_KT, irq15, 0);
```

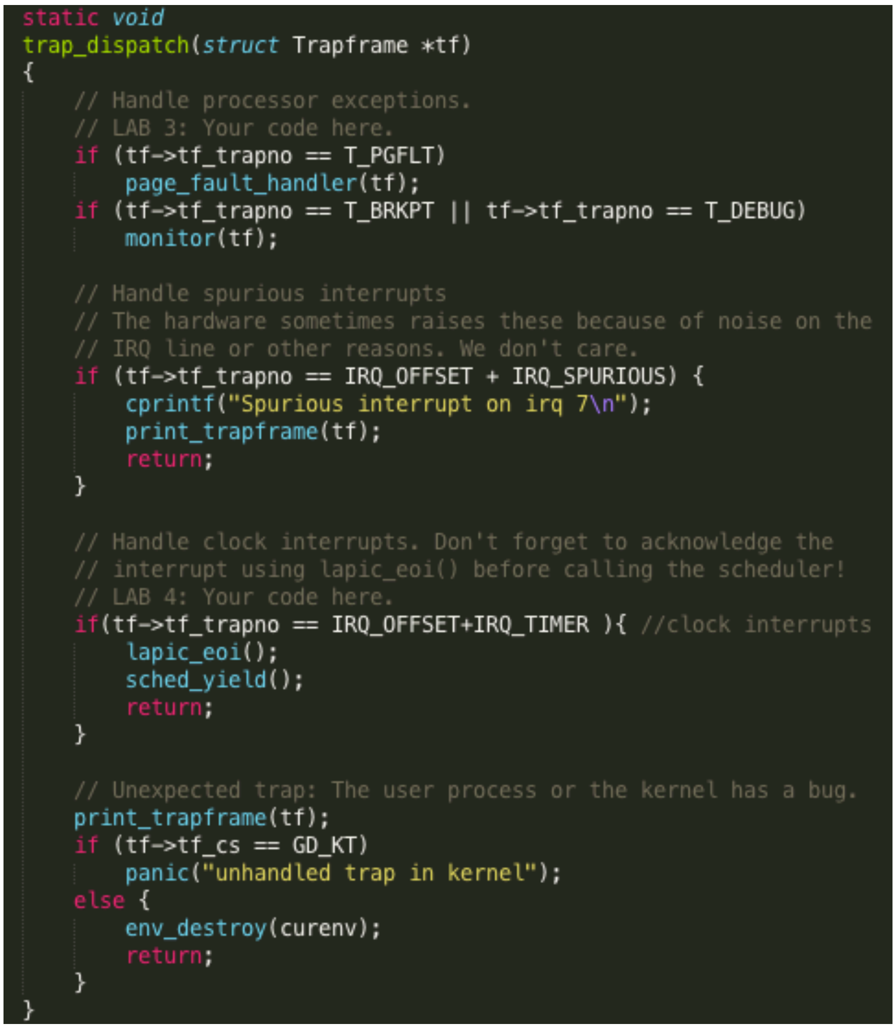
然后在 kern/env.c 中的 env_alloc()中为 env 打开终端开关：



```
// Enable interrupts while in user mode.
// LAB 4: Your code here.
e->env_tf.tf_eflags |=  FL_IF;
```

Exercise 13. Modify the kernel's trap_dispatch() function so that it calls sched_yield() to find and run a different environment whenever a clock interrupt takes place.

同原先判断 page fault、break point 中断一样，在 trap_dispatch()根据注释加上对 spurious interrupts 和 clock interrupts 的处理：

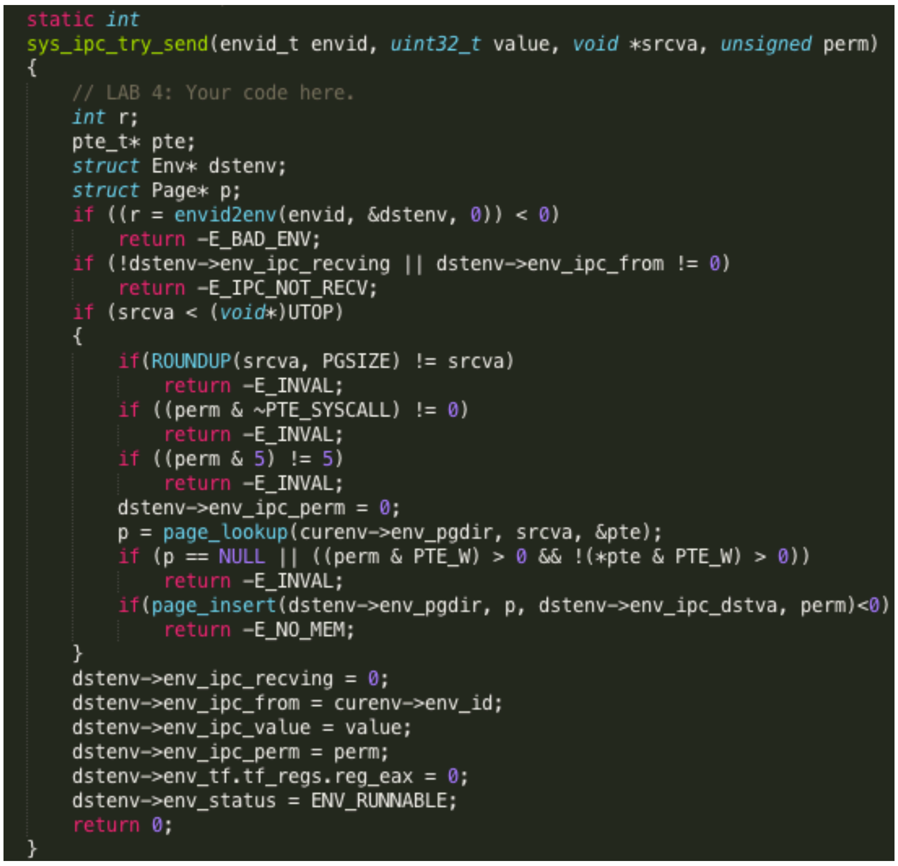


```c
static void
trap_dispatch(struct Trapframe *tf)
{
	// Handle processor exceptions.
	// LAB 3: Your code here.
	if (tf->tf_trapno == T_PGFLT)
		page_fault_handler(tf);
	if (tf->tf_trapno == T_BRKPT || tf->tf_trapno == T_DEBUG)
		monitor(tf);

	// Handle spurious interrupts
	// The hardware sometimes raises these because of noise on the
	// IRQ line or other reasons. We don't care.
	if (tf->tf_trapno == IRQ_OFFSET + IRQ_SPURIOUS) {
		cprintf("Spurious interrupt on irq 7\n");
		print_trapframe(tf);
		return;
	}

	// Handle clock interrupts. Don't forget to acknowledge the
	// interrupt using lapic_eoi() before calling the scheduler!
	// LAB 4: Your code here.
	if(tf->tf_trapno == IRQ_OFFSET+IRQ_TIMER ){ //clock interrupts
		lapic_eoi();
		sched_yield();
		return;
	}

	// Unexpected trap: The user process or the kernel has a bug.
	print_trapframe(tf);
	if (tf->tf_cs == GD_KT)
		panic("unhandled trap in kernel");
	else {
		env_destroy(curenv);
		return;
	}
}
```

Exercise 14. Implement sys_ipc_recv and sys_ipc_try_send in kern/syscall.c.

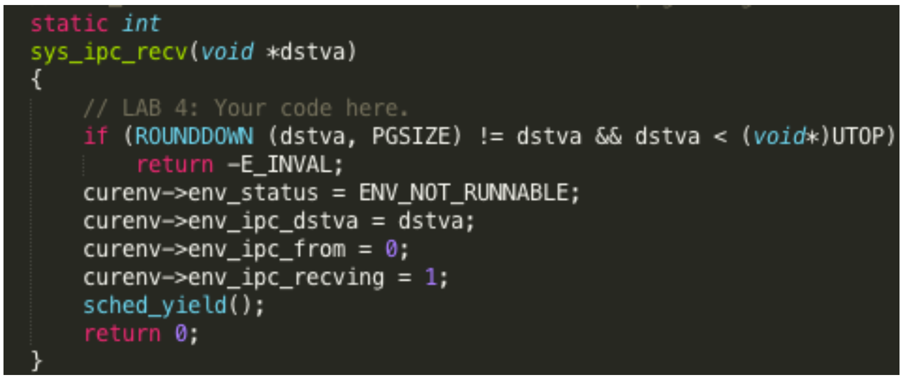
sys_ipc_try_send

这个练习的注释非常地详尽，直接按照注释的逻辑实现即可：



```
static int
sys_ipc_try_send(envid_t envid, uint32_t value, void *srcva, unsigned perm)
{
    // LAB 4: Your code here.
    int r;
    pte_t* pte;
    struct Env* dstenv;
    struct Page* p;
    if ((r = envid2env(envid, &dstenv, 0)) < 0)
        return -E_BAD_ENV;
    if (!dstenv->env_ipc_recving || dstenv->env_ipc_from != 0)
        return -E_IPC_NOT_RECV;
    if (srcva < (void*)UTOP)
    {
        if(ROUNDUP(srcva, PGSIZE) != srcva)
            return -E_INVAL;
        if ((perm & ~PTE_SYSCALL) != 0)
            return -E_INVAL;
        if ((perm & 5) != 5)
            return -E_INVAL;
        dstenv->env_ipc_perm = 0;
        p = page_lookup(curenv->env_pgdir, srcva, &pte);
        if (p == NULL || ((perm & PTE_W) > 0 && !(*pte & PTE_W) > 0))
            return -E_INVAL;
        if(page_insert(dstenv->env_pgdir, p, dstenv->env_ipc_dstva, perm)<0)
            return -E_NO_MEM;
    }
    dstenv->env_ipc_recving = 0;
    dstenv->env_ipc_from = curenv->env_id;
    dstenv->env_ipc_value = value;
    dstenv->env_ipc_perm = perm;
    dstenv->env_tf.tf_regs.reg_eax = 0;
    dstenv->env_status = ENV_RUNNABLE;
    return 0;
}
```

sys_ipc_recv



```
static int
sys_ipc_recv(void *dstva)
{
    // LAB 4: Your code here.
    if (ROUNDDOWN (dstva, PGSIZE) != dstva && dstva < (void*)UTOP)
        return -E_INVAL;
    curenv->env_status = ENV_NOT_RUNNABLE;
    curenv->env_ipc_dstva = dstva;
    curenv->env_ipc_from = 0;
    curenv->env_ipc_recving = 1;
    sched_yield();
    return 0;
}
```