Lab 6 实验报告

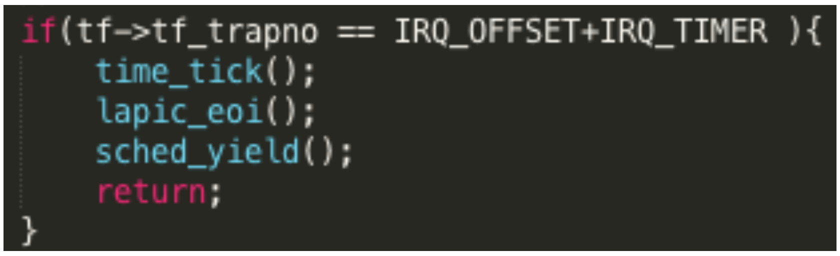
姓名：李浩宇

学号：5130809070

Exercise 1. Add a call to time_tick for every clock interrupt in kern/trap.c. Implement sys_time_msec and add it to syscall in kern/syscall.c so that user space has access to the time.

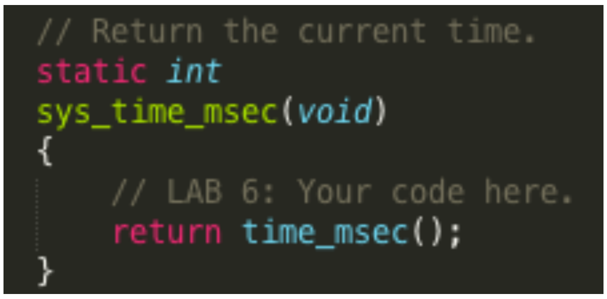
根据题意，在 kern/trap.c 的 trap_dispatch()函数中，找到处理时钟中断的代码块，在其中增加一处 time_tick()的调用即可。同时 sys_time_msec 的实现也十分简单，只要调用返回 time_msec()就好。

trap_dispatch():



```
if(tf->tf_trapno == IRQ_OFFSET+IRQ_TIMER ){
    time_tick();
    lapic_eoi();
    sched_yield();
    return;
}
```

sys_time_msec():

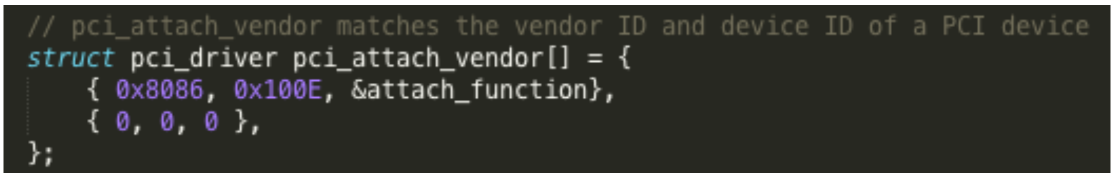


```
// Return the current time.
static int
sys_time_msec(void)
{
    // LAB 6: Your code here.
    return time_msec();
}
```

Exercise 3. Implement an attach function to initialize the E1000.

在这里，attach function 还不需要处理寄存器的初始化，只需要简单调用 pci_func_enable()，同时在 pci attach vendor 中将其注册即可。

kern/pci.c:



```
// pci_attach_vendor matches the vendor ID and device ID of a PCI device
struct pci_driver pci_attach_vendor[] = {
    { 0x8086, 0x100E, &attach_function},
    { 0, 0, 0 },
};
```

Exercise 4. In your attach function, create a virtual memory mapping for the E1000's BAR 0.

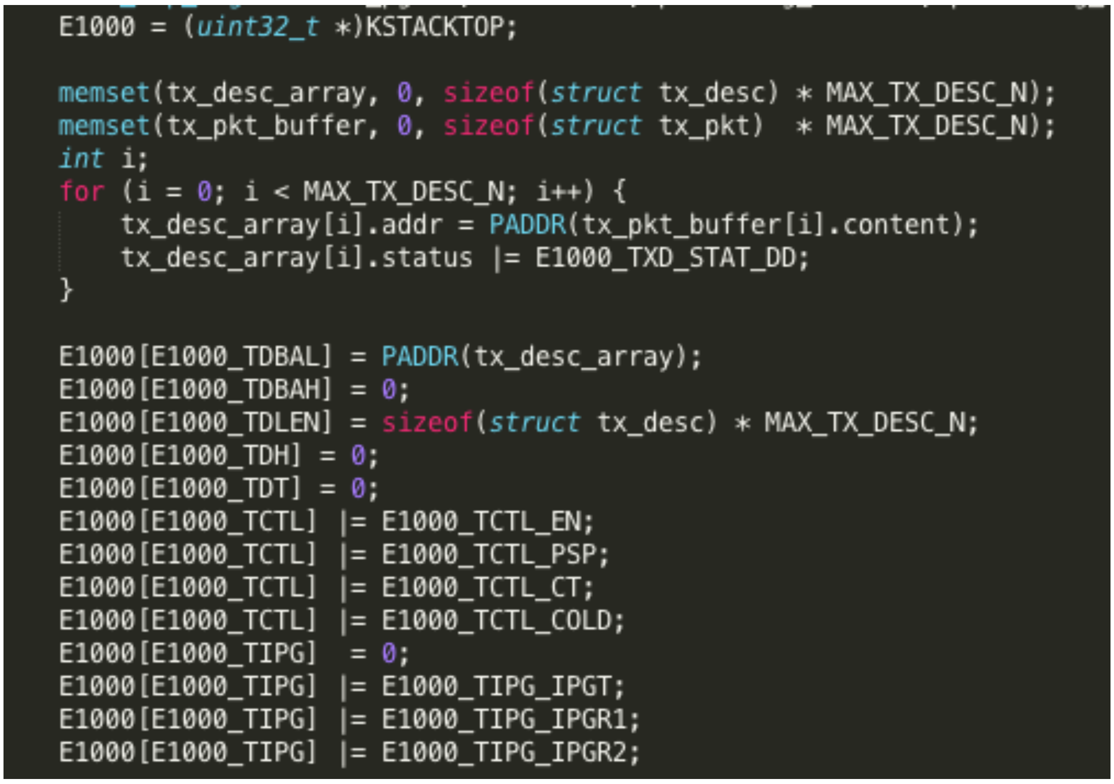
按照文档要求，将 E1000 所需要的内存映射至 KSTACKTOP 和 KERNBASE 之

间，利用 lab 2 中实现过的 boot_map_region 可以很容易实现这一点。

Exercise 5. Perform the initialization steps described in section 14.5 (but not its subsections).

到这一步需要在之前的 attach function 中初始化发包相关的数据结构和寄存器，依照文档一步一步走下来即可。

attach_function():

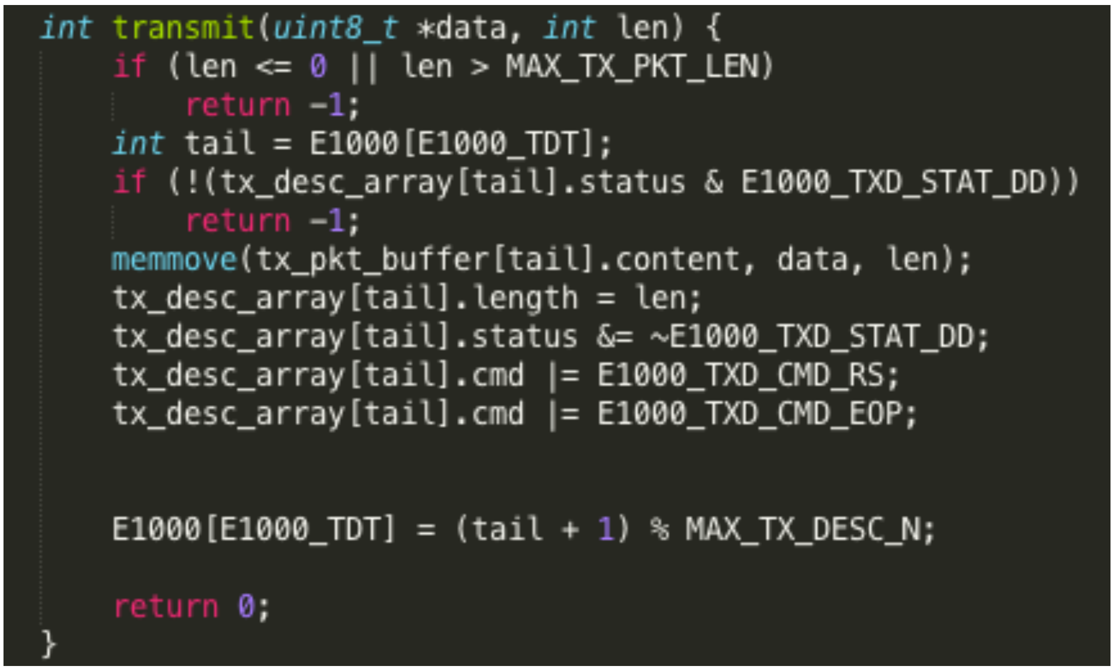


```c
    E1000 = (uint32_t *)KSTACKTOP;

    memset(tx_desc_array, 0, sizeof(struct tx_desc) * MAX_TX_DESC_N);
    memset(tx_pkt_buffer, 0, sizeof(struct tx_pkt)  * MAX_TX_DESC_N);
    int i;
    for (i = 0; i < MAX_TX_DESC_N; i++) {
        tx_desc_array[i].addr = PADDR(tx_pkt_buffer[i].content);
        tx_desc_array[i].status |= E1000_TXD_STAT_DD;
    }

    E1000[E1000_TDBAL] = PADDR(tx_desc_array);
    E1000[E1000_TDBAH] = 0;
    E1000[E1000_TDLEN] = sizeof(struct tx_desc) * MAX_TX_DESC_N;
    E1000[E1000_TDH] = 0;
    E1000[E1000_TDT] = 0;
    E1000[E1000_TCTL] |= E1000_TCTL_EN;
    E1000[E1000_TCTL] |= E1000_TCTL_PSP;
    E1000[E1000_TCTL] |= E1000_TCTL_CT;
    E1000[E1000_TCTL] |= E1000_TCTL_COLD;
    E1000[E1000_TIPG]  = 0;
    E1000[E1000_TIPG] |= E1000_TIPG_IPGT;
    E1000[E1000_TIPG] |= E1000_TIPG_IPGR1;
    E1000[E1000_TIPG] |= E1000_TIPG_IPGR2;
```

Exercise 6. Write a function to transmit a packet by checking that the next descriptor is free, copying the packet data into the next descriptor, and updating TDT.

接下来需要实现一个发包的函数，在这里我处理队列已满的方法是直接返回错误值(-1)，将问题交给上层协议处理。

transmit():



```c
int transmit(uint8_t *data, int len) {
    if (len <= 0 || len > MAX_TX_PKT_LEN)
        return -1;
    int tail = E1000[E1000_TDT];
    if (!(tx_desc_array[tail].status & E1000_TXD_STAT_DD))
        return -1;
    memmove(tx_pkt_buffer[tail].content, data, len);
    tx_desc_array[tail].length = len;
    tx_desc_array[tail].status &= ~E1000_TXD_STAT_DD;
    tx_desc_array[tail].cmd |= E1000_TXD_CMD_RS;
    tx_desc_array[tail].cmd |= E1000_TXD_CMD_EOP;


    E1000[E1000_TDT] = (tail + 1) % MAX_TX_DESC_N;

    return 0;
}
```

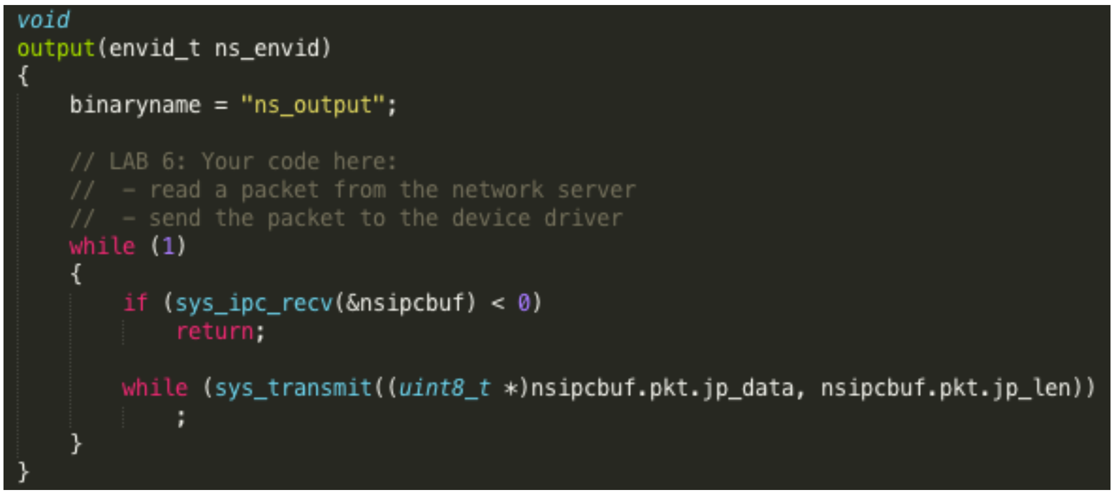
Exercise 7. Add a system call that lets you transmit packets from user space.

将上述函数注册成 system call，需要在 inc/syscall.h、inc/lib.h、lib/syscall.c 和 kern/syscall.c 中增加对应的入口。

Exercise 8. Implement net/output.c.

这里需要实现一个测试函数，逻辑比较简单，获取包然后发送即可，如果失败则是重试。

net/output.c

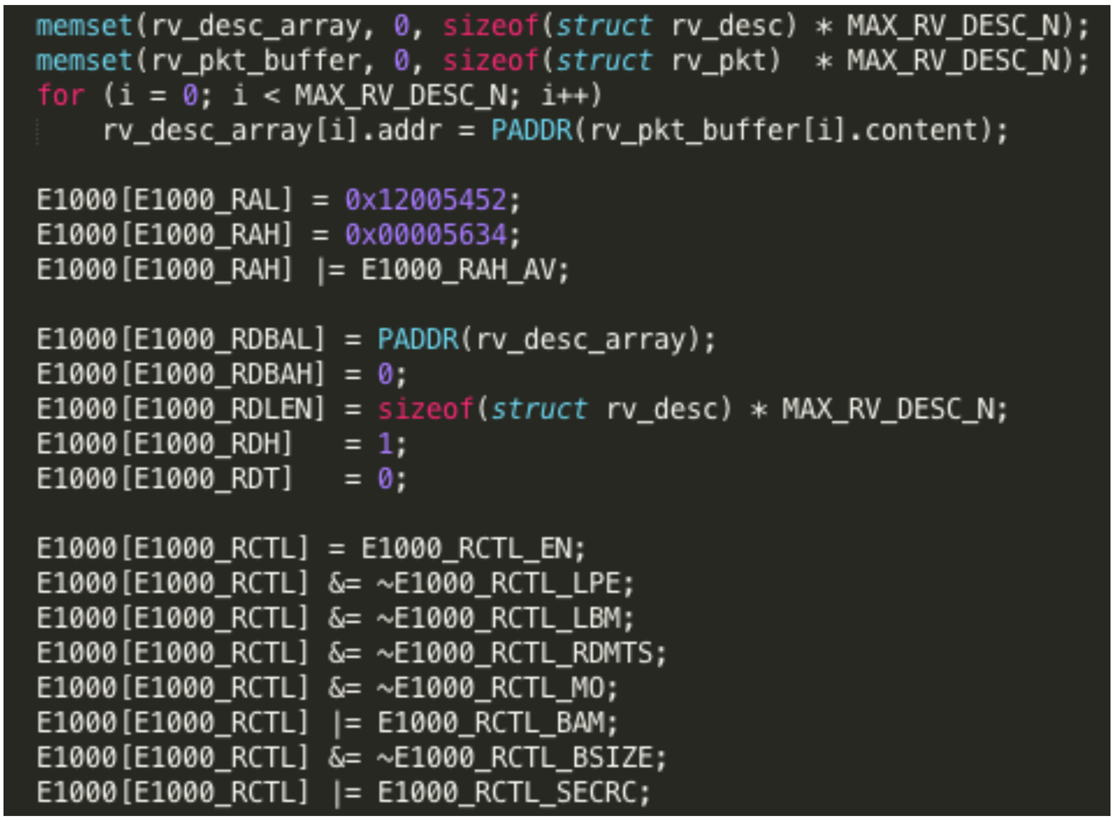


```c
void
output(envid_t ns_envid)
{
    binaryname = "ns_output";

    // LAB 6: Your code here:
    //  - read a packet from the network server
    //  - send the packet to the device driver
    while (1)
    {
        if (sys_ipc_recv(&nsipcbuf) < 0)
            return;

        while (sys_transmit((uint8_t *)nsipcbuf.pkt.jp_data, nsipcbuf.pkt.jp_len))
            ;
    }
}
```

Exercise 10. Set up the receive queue and configure the E1000 by following the process in section 14.4.

同理 Exercise 5。

attach_function():

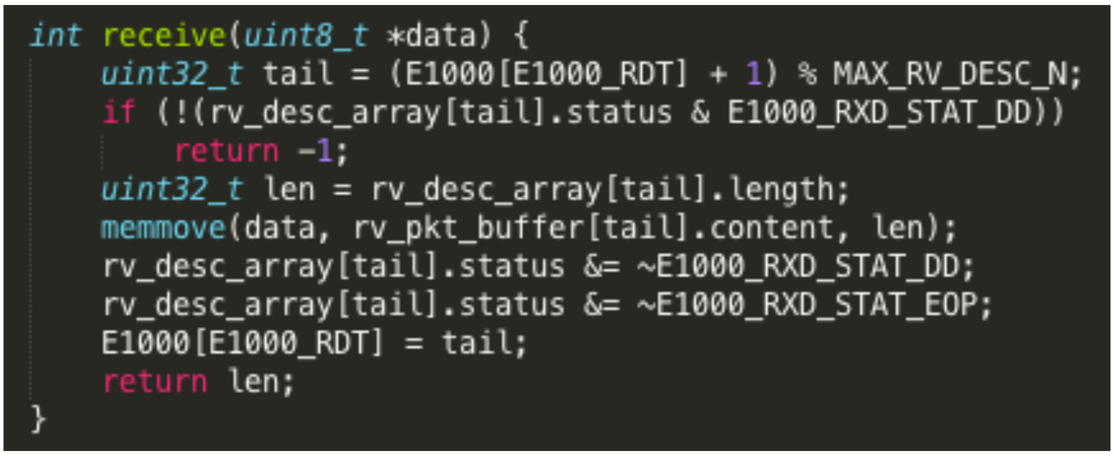


```
memset(rv_desc_array, 0, sizeof(struct rv_desc) * MAX_RV_DESC_N);
memset(rv_pkt_buffer, 0, sizeof(struct rv_pkt)  * MAX_RV_DESC_N);
for (i = 0; i < MAX_RV_DESC_N; i++)
    rv_desc_array[i].addr = PADDR(rv_pkt_buffer[i].content);

E1000[E1000_RAL] = 0x12005452;
E1000[E1000_RAH] = 0x00005634;
E1000[E1000_RAH] |= E1000_RAH_AV;

E1000[E1000_RDBAL] = PADDR(rv_desc_array);
E1000[E1000_RDBAH] = 0;
E1000[E1000_RDLEN] = sizeof(struct rv_desc) * MAX_RV_DESC_N;
E1000[E1000_RDH]   = 1;
E1000[E1000_RDT]   = 0;

E1000[E1000_RCTL] = E1000_RCTL_EN;
E1000[E1000_RCTL] &= ~E1000_RCTL_LPE;
E1000[E1000_RCTL] &= ~E1000_RCTL_LBM;
E1000[E1000_RCTL] &= ~E1000_RCTL_RDMTS;
E1000[E1000_RCTL] &= ~E1000_RCTL_MO;
E1000[E1000_RCTL] |= E1000_RCTL_BAM;
E1000[E1000_RCTL] &= ~E1000_RCTL_BSIZE;
E1000[E1000_RCTL] |= E1000_RCTL_SECRC;
```

Exercise 11. Write a function to receive a packet from the E1000 and expose it to user space by adding a system call.

同理 Exercise 6。

receive():

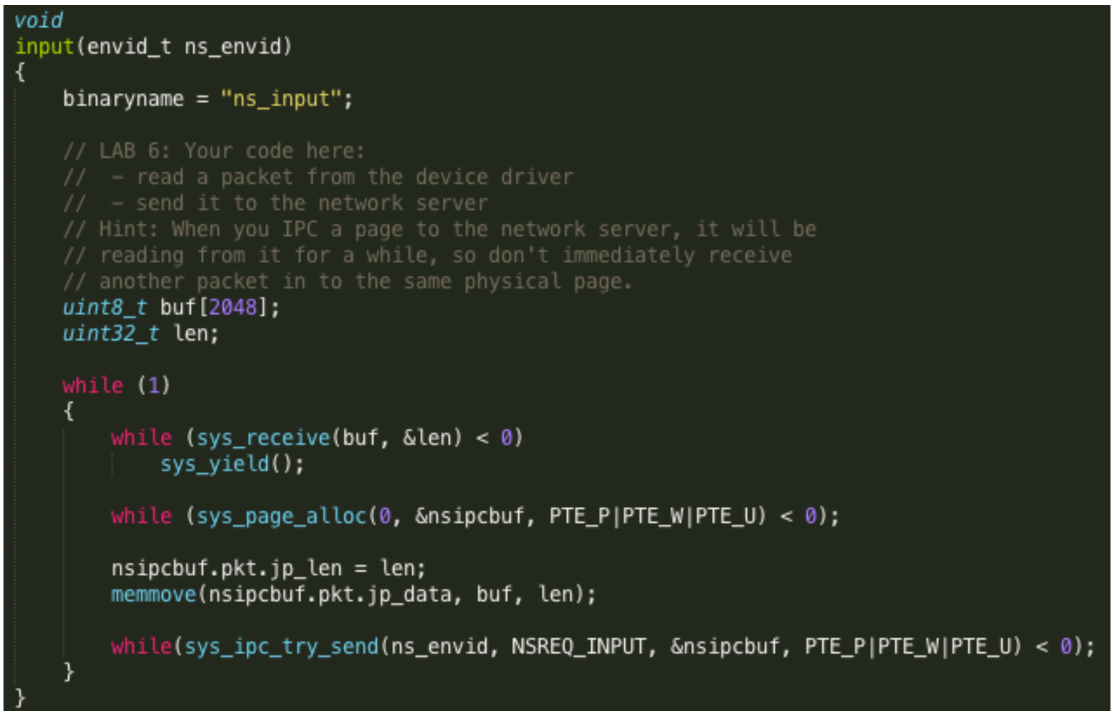


```
int receive(uint8_t *data) {
    uint32_t tail = (E1000[E1000_RDT] + 1) % MAX_RV_DESC_N;
    if (!(rv_desc_array[tail].status & E1000_RXD_STAT_DD))
        return -1;
    uint32_t len = rv_desc_array[tail].length;
    memmove(data, rv_pkt_buffer[tail].content, len);
    rv_desc_array[tail].status &= ~E1000_RXD_STAT_DD;
    rv_desc_array[tail].status &= ~E1000_RXD_STAT_EOP;
    E1000[E1000_RDT] = tail;
    return len;
}
```

Exercise 12. Implement net/input.c.

input.c 的实现比 output.c 要复杂一点，收到包之后需要额外调用一个 syscall 为 nsipcbuf 分配空间。

input.c:

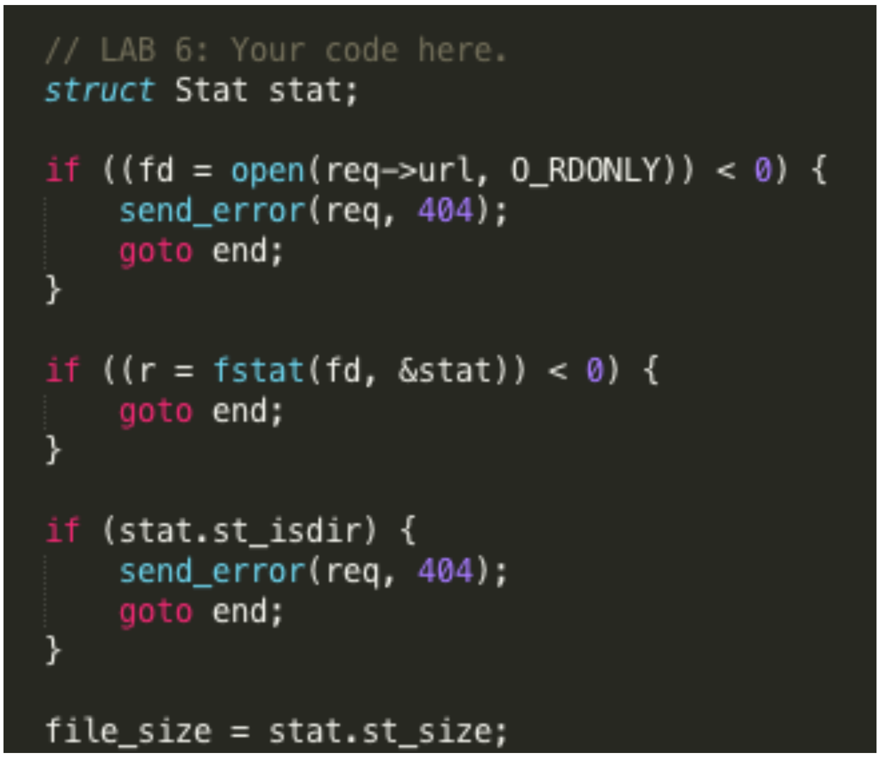


```c
void
input(envid_t ns_envid)
{
	binaryname = "ns_input";

	// LAB 6: Your code here:
	// 	- read a packet from the device driver
	//	- send it to the network server
	// Hint: When you IPC a page to the network server, it will be
	// reading from it for a while, so don't immediately receive
	// another packet in to the same physical page.
	uint8_t buf[2048];
	uint32_t len;

	while (1)
	{
		while (sys_receive(buf, &len) < 0)
			sys_yield();

		while (sys_page_alloc(0, &nsipcbuf, PTE_P|PTE_W|PTE_U) < 0);

		nsipcbuf.pkt.jp_len = len;
		memmove(nsipcbuf.pkt.jp_data, buf, len);

		while(sys_ipc_try_send(ns_envid, NSREQ_INPUT, &nsipcbuf, PTE_P|PTE_W|PTE_U) < 0);
	}
}
```

Exercise 13. The web server is missing the code that deals with sending the contents of a file back to the client. Finish the web server by implementing send_file and send_data.

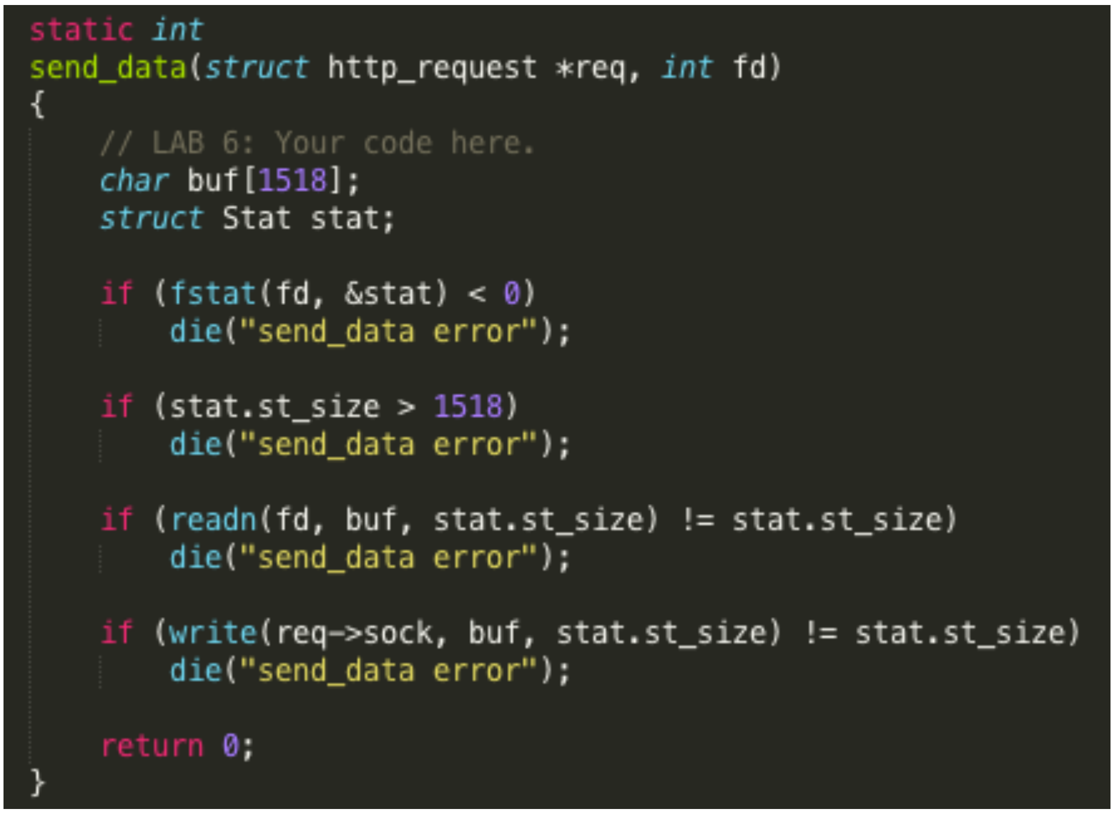
这个联系需要实现两个函数，其中 send_file()只需要检查一下文件是否存在以及是否为目录就可以了，注释很详细，而 send_data()也很简单。

send_file():



```c
// LAB 6: Your code here.
struct Stat stat;

if ((fd = open(req->url, O_RDONLY)) < 0) {
	send_error(req, 404);
	goto end;
}

if ((r = fstat(fd, &stat)) < 0) {
	goto end;
}

if (stat.st_isdir) {
	send_error(req, 404);
	goto end;
}

file_size = stat.st_size;
```

send_data():

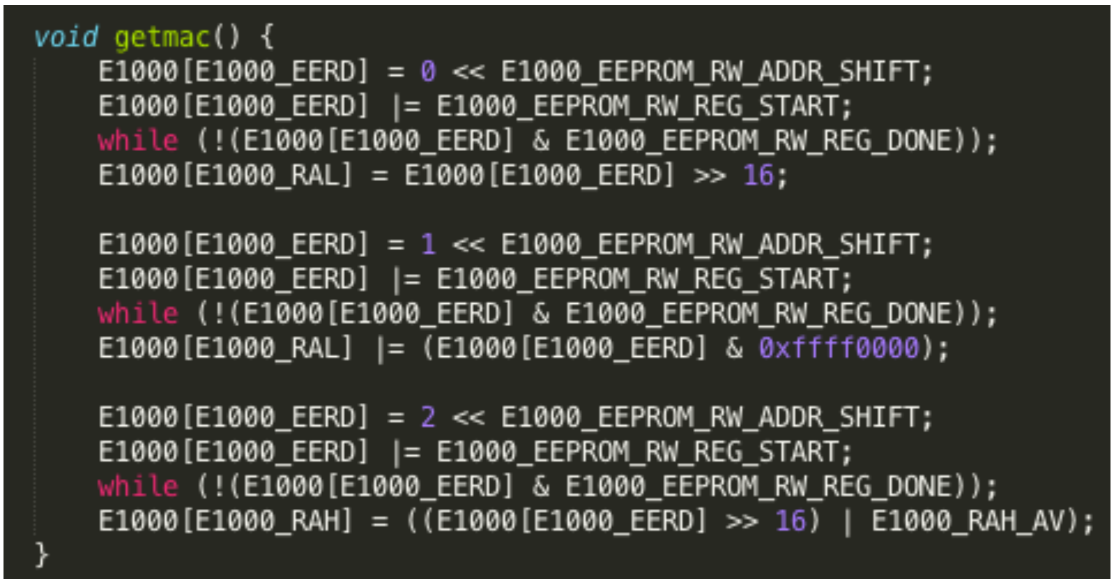


```c
static int
send_data(struct http_request *req, int fd)
{
    // LAB 6: Your code here.
    char buf[1518];
    struct Stat stat;

    if (fstat(fd, &stat) < 0)
        die("send_data error");

    if (stat.st_size > 1518)
        die("send_data error");

    if (readn(fd, buf, stat.st_size) != stat.st_size)
        die("send_data error");

    if (write(req->sock, buf, stat.st_size) != stat.st_size)
        die("send_data error");

    return 0;
}
```

Challenge! Read about the EEPROM in the developer's manual and write the code to load the E1000's MAC address out of the EEPROM. Currently, QEMU's default MAC address is hard-coded into both your receive initialization and lwIP. Fix your initialization to use the MAC address you read from the EEPROM, add a system call to pass the MAC address to lwIP, and modify lwIP to the MAC address read from the card. Test your change by configuring QEMU to use a different MAC address.

Challenge 我选择了优化 MAC 地址读取方式的这一个，实现从 EEPROM 动态读取而不是原来硬编码的方式，同之前一样仍然需要在 inc/syscall.h、inc/lib.h、lib/syscall.c 和 kern/syscall.c 中将其注册。

getmac():



```
void getmac() {
    E1000[E1000_EERD] = 0 << E1000_EEPROM_RW_ADDR_SHIFT;
    E1000[E1000_EERD] |= E1000_EEPROM_RW_REG_START;
    while (!(E1000[E1000_EERD] & E1000_EEPROM_RW_REG_DONE));
    E1000[E1000_RAL] = E1000[E1000_EERD] >> 16;

    E1000[E1000_EERD] = 1 << E1000_EEPROM_RW_ADDR_SHIFT;
    E1000[E1000_EERD] |= E1000_EEPROM_RW_REG_START;
    while (!(E1000[E1000_EERD] & E1000_EEPROM_RW_REG_DONE));
    E1000[E1000_RAL] |= (E1000[E1000_EERD] & 0xffff0000);

    E1000[E1000_EERD] = 2 << E1000_EEPROM_RW_ADDR_SHIFT;
    E1000[E1000_EERD] |= E1000_EEPROM_RW_REG_START;
    while (!(E1000[E1000_EERD] & E1000_EEPROM_RW_REG_DONE));
    E1000[E1000_RAH] = ((E1000[E1000_EERD] >> 16) | E1000_RAH_AV);
}
```