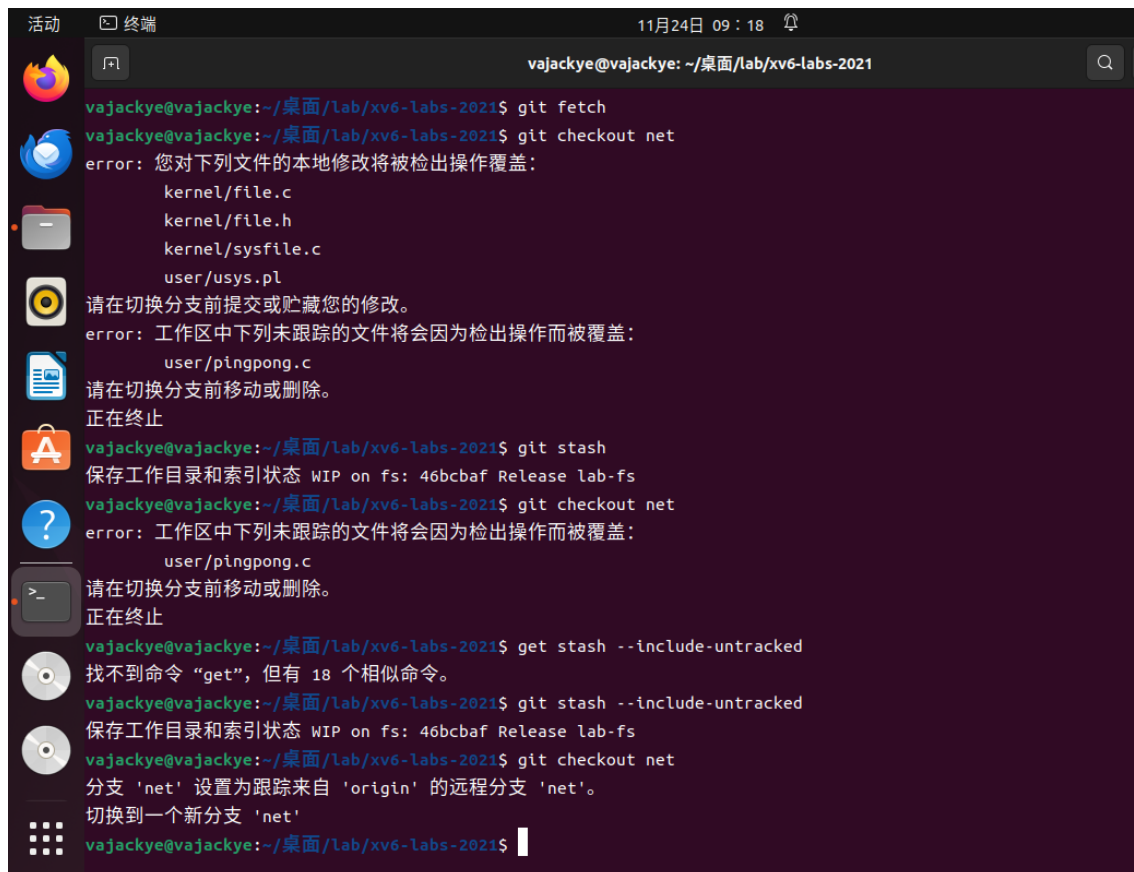


Lab5: I/O设备与网卡驱动

- 前置操作：先切换分支到lab5

```
git fetch
git checkout net
make clean
```



The terminal window shows the following sequence of commands and outputs:

```
vajackye@vajackye: ~/桌面/lab/xv6-labs-2021
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ git fetch
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ git checkout net
error: 您对下列文件的本地修改将被检出操作覆盖:
    kernel/file.c
    kernel/file.h
    kernel/sysfile.c
    user/usys.pl
请在切换分支前提交或贮藏您的修改。
error: 工作区中下列未跟踪的文件将会因为检出操作而被覆盖:
    user/pingpong.c
请在切换分支前移动或删除。
正在终止
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ git stash
保存工作目录和索引状态 WIP on fs: 46bcbafe Release lab-fs
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ git checkout net
error: 工作区中下列未跟踪的文件将会因为检出操作而被覆盖:
    user/pingpong.c
请在切换分支前移动或删除。
正在终止
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ get stash --include-untracked
找不到命令 "get", 但有 18 个相似命令。
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ git stash --include-untracked
保存工作目录和索引状态 WIP on fs: 46bcbafe Release lab-fs
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ git checkout net
分支 'net' 设置为跟踪来自 'origin' 的远程分支 'net'。
切换到一个新分支 'net'
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$
```

- 先在 `kernel/e1000_dev.h`, 定义:

```
#define TX_RING_SIZE 16
#define RX_RING_SIZE 16

#define E1000_TCTL_EN 0x00000002
#define E1000_TCTL_PSP 0x00000008
#define E1000_TXD_STAT_DD 0x00000001

#define E1000_RCTL_EN 0x00000002
#define E1000_RCTL_BAM 0x00008000
#define E1000_RCTL_SZ_2048 0x00000000
#define E1000_RCTL_SECRC 0x04000000
```

- 阅读 `e1000_init()` 的初始化发送环和接收环的初始代码, 了解基本逻辑。
- 打开 `kernel/e1000.c` 文件, 开始填充用于发送包的 `e1000_transmit()` 函数:

```

int e1000_transmit(struct mbuf *m) {
    uint32 idx = regs[E1000_TDT]; // 获取发送环索引
    struct tx_desc *desc = &tx_ring[idx]; // 获取当前描述符

    // 检查发送环是否已满
    if (!(desc->status & E1000_TXD_STAT_DD)) {
        printf("e1000_transmit: TX ring full\n");
        return -1; // 无法发送数据包
    }

    // 如果有旧的 mbuf, 需要释放
    if (tx_mbufs[idx]) {
        mbuffree(tx_mbufs[idx]);
    }

    // 填写描述符内容
    desc->addr = (uint64)m->head; // 数据包起始地址
    desc->length = m->len; // 数据包长度
    desc->cmd = E1000_TXD_CMD_RS | E1000_TXD_CMD_EOP; // 设置命令标志位
    tx_mbufs[idx] = m; // 保存当前 mbuf

    regs[E1000_TDT] = (idx + 1) % TX_RING_SIZE;

    printf("e1000_transmit: Transmitted packet of length %d\n", m->len);
    return 0; // 发送成功
}

```

- 测试 `e1000_transmit()` 无误后, 继续在 `kernel/e1000.c` 内, 填充用于接收包的 `e1000_recv()` 函数:

```

void e1000_recv(void) {
    uint32 idx = (regs[E1000_RDT] + 1) % RX_RING_SIZE; // 获取接收环索引
    struct rx_desc *desc = &rx_ring[idx]; // 获取当前描述符

    while (desc->status & E1000_RXD_STAT_DD) { // 如果有数据包
        struct mbuf *m = rx_mbufs[idx]; // 获取当前 mbuf

        m->len = desc->length; // 更新数据长度
        net_rx(m); // 将数据传递给协议栈

        // 为接收环分配新的 mbuf
        struct mbuf *new_m = mbufalloc(0);
        if (!new_m) {
            panic("e1000_recv: mbufalloc failed");
        }

        desc->addr = (uint64)new_m->head; // 更新描述符地址
        desc->status = 0; // 清空状态位
        rx_mbufs[idx] = new_m; // 更新环中 mbuf

        idx = (idx + 1) % RX_RING_SIZE; // 更新索引
    }
}

```

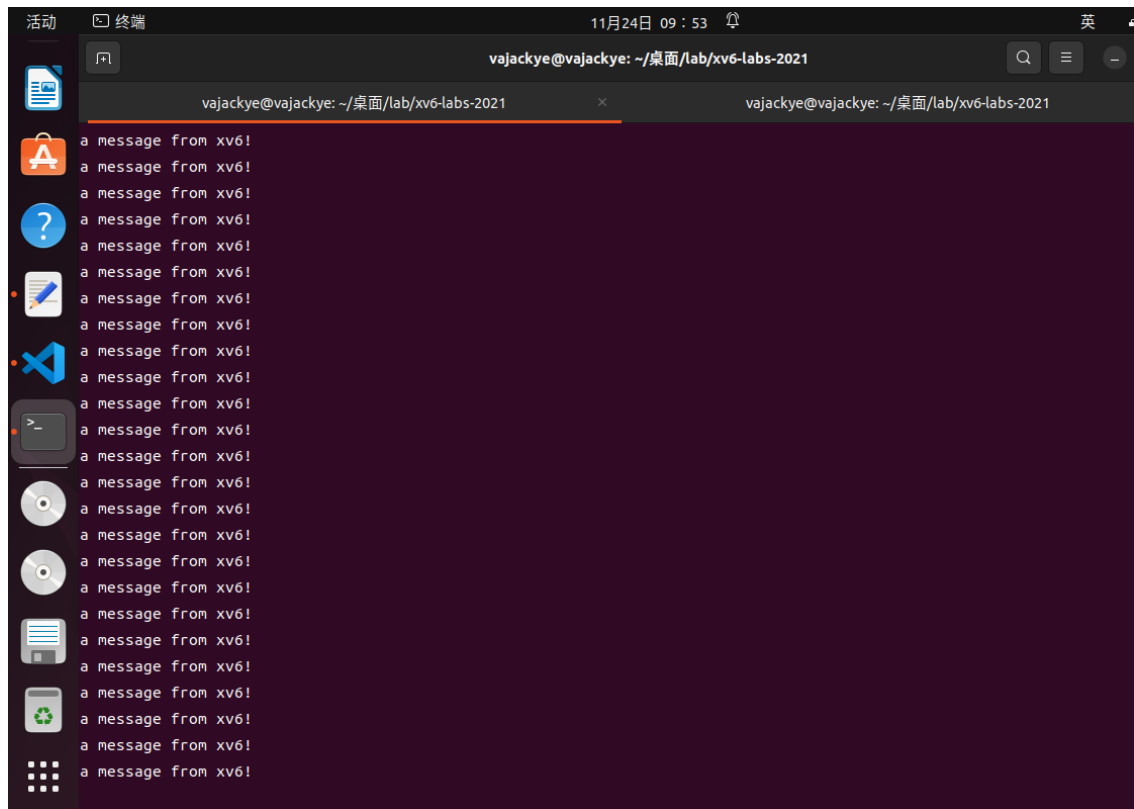
```
}

regs[E1000_RDT] = (idx - 1 + RX_RING_SIZE) % RX_RING_SIZE;
printf("e1000_recv: Processed packets in RX ring\n");
}
```

- 最后进行测试:

```
终端1:
    make server
终端2:
    make qemu
    nettests
```

- 测试结果如下, 发现测试成功:



```
QEMU: Terminated
vajackye@vajackye:~/桌面/lab/xv6-labs-2021$ make qemu
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -DSOL_NET -DLAB_NET -MD -mcpu=medany -ffrees
-fno-common -nostdlib -mno-relax -I. -fno-stack-protector -DNET_TESTS_PORT=26099 -fno-pie -no-pie -c -o kernel/e10
rnel/e1000.c
riscv64-linux-gnu-ld -z max-page-size=4096 -T kernel/kernel.ld -o kernel/kernel kernel/entry.o kernel/kalloc.o kerne
g.o kernel/main.o kernel/vm.o kernel/proc.o kernel/swtch.o kernel/trampoline.o kernel/trap.o kernel/syscall.o kernel
c.o kernel/bio.o kernel/fs.o kernel/log.o kernel/sleeplock.o kernel/file.o kernel/pipe.o kernel/exec.o kernel/sysfil
nel/kernelvec.o kernel/plic.o kernel/virtio_disk.o kernel/e1000.o kernel/net.o kernel/sysnet.o kernel/pci.o kernel/s
kernel/console.o kernel/printf.o kernel/uart.o kernel/spinlock.o
riscv64-linux-gnu-objdump -S kernel/kernel > kernel/kernel.asm
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel/kernel.sym
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -drive file=fs.img,if=n
mat=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -netdev user,id=net0,hostfwd=udp::26099-:2000
t filter-dump,id=net0,netdev=net0,file=packets.pcap -device e1000,netdev=net0,bus=pcie.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ nettests
nettests running on port 26099
testing ping: OK
testing single-process pings: OK
```

实验总结

这次实验是对xv6操作系统中网络驱动与网络协议栈的一个实践，是关于以太网卡（E1000）的驱动实现。通过本次实验，我理解了计算机网络通信的底层实现，尤其是硬件和操作系统如何协同工作来完成数据包的发送与接收。

在实现网卡驱动的过程中，我对发送环和接收环的工作机制有了一定理解。每个网络数据包的发送和接收都依赖于环形缓冲区的管理，而这些环缓冲区由网卡的硬件直接控制和管理。

在写代码过程中，我学会了如何通过对硬件寄存器的操作来控制网卡的行为。此外，我还了解了如何使用打印语句来检查网卡的数据传输状态，并通过调试信息确认网络通信是否正常。