Lab7-Networking

实验目的

- 实现一个网络驱动程序，能够通过E1000网卡处理网络通信。
- 实现数据包的发送和接收功能
- 熟悉E1000的工作原理和操作寄存器
    - E1000是一个网卡设备，实验中会使用QEMU模拟这个设备并与xv6操作系统进行网络通信，通过模拟的E1000连接到`LAN`上，`Guest`的IP地址为`10.0.2.15`，`Host`的IP地址为`10.0.2.2`

实验步骤

1. `kernel/e1000.c`：初始化E1000网卡，设置发送和接收队列，并处理数据包的发送和接收，通过中断机制来实现网络通信的实时处理。

   `kernel/e1000_dev.h`：通过定义硬件寄存器、控制位、描述符结构及其相关命令和状态，提供了与E1000网卡进行交互的基础接口和数据结构。

   `kernel/net.c`：实现了基本的网络协议支持，包括ARP、IP和UDP协议的处理，并提供了mbuf缓冲区的管理功能。

   `kernel/pci.c`：初始化PCI-Express总线上的E1000网卡。在QEMU虚拟机环境中，它通过扫描PCI设备，识别并配置E1000网卡的寄存器，并调用相应的初始化函数，从而使E1000网卡能够在xv6操作系统中正常工作

2. 完成`e1000_transmit()`函数：
    - 读取E1000_TDT控制寄存器，获取发送队列的当前索引。
    - 检查发送队列是否溢出，若E1000_TXD_STAT_DD位未设置，则返回错误。
    - 释放上次传输的`mbuf`。
    - 填写发送描述符，设置必要的命令标志，并将`mbuf`的指针存储起来以便稍后释放。
    - 更新E1000_TDT寄存器，指向下一个可用的发送描述符。

```
int e1000_transmit(struct mbuf *m)
{
    // 获取锁，确保多线程环境下的同步
    acquire(&e1000_lock);

    // 读取E1000_TDT控制寄存器，获取下一个可用的发送描述符索引
    int index = regs[E1000_TDT];

    // 检查当前索引位置的发送描述符是否已完成发送（通过检查E1000_TXD_STAT_DD位）
    if ((tx_ring[index].status & E1000_TXD_STAT_DD) == 0) {
        // 如果发送未完成，释放锁并返回错误
        release(&e1000_lock);
        return -1;
    }

    // 如果当前索引位置有上一个发送的mbuf，释放它
    if (tx_mbufs[index])
        mbuffree(tx_mbufs[index]);

    // 将当前要发送的mbuf保存到发送缓冲区数组中
    tx_mbufs[index] = m;
    // 设置发送描述符的长度字段为当前mbuf的长度
    tx_ring[index].length = m->len;
    // 设置发送描述符的地址字段为当前mbuf数据的首地址
    tx_ring[index].addr = (uint64)m->head;

    // 设置发送描述符的命令字段，包括报告状态（E1000_TXD_CMD_RS）和数据包结束
（E1000_TXD_CMD_EOP）
    tx_ring[index].cmd = E1000_TXD_CMD_RS | E1000_TXD_CMD_EOP;

    // 更新E1000_TDT控制寄存器，指向下一个可用的发送描述符索引（环状结构，取模运算）
    regs[E1000_TDT] = (index + 1) % TX_RING_SIZE;

    // 释放锁
    release(&e1000_lock);

    // 返回成功
    return 0;
}
```

3. 完成`e1000_recv()`函数：

   - 读取E1000_RDT控制寄存器并加一，获取接收队列的当前索引。
   - 检查接收队列中是否有新数据包，若E1000_RXD_STAT_DD位未设置，则停止处理。
   - 更新`mbuf`的长度，将数据包传递给网络堆栈（调用`net_rx()`函数）。
   - 分配新的`mbuf`并替换已处理的`mbuf`，清除描述符的状态位。
   - 更新E1000_RDT寄存器，指向上一个已处理的接收描述符。

```
static void e1000_recv(void)
{
    // 无限循环遍历接收描述符环，处理所有已接收的数据包
    while (1) {

        // 获取下一个接收描述符的索引，环状结构，取模运算
        int index = (regs[E1000_RDT] + 1) % RX_RING_SIZE;

        // 检查当前描述符的状态位，判断是否有新的数据包
        if ((rx_ring[index].status & E1000_RXD_STAT_DD) == 0) {
            // 如果没有新的数据包，退出循环
            return;
        }

        // 设置当前缓冲区的长度为接收描述符中记录的长度
        rx_mbufs[index]->len = rx_ring[index].length;

        // 将接收到的数据包传递给网络层处理
        net_rx(rx_mbufs[index]);

        // 分配新的mbuf缓冲区，以便接收新的数据包
        rx_mbufs[index] = mbufalloc(0);
        // 检查分配是否成功
        if (!rx_mbufs[index])
            panic("e1000_recv: mbufalloc failed");

        // 清除当前描述符的状态位
        rx_ring[index].status = 0;

        // 更新描述符的地址为新分配的缓冲区地址
        rx_ring[index].addr = (uint64)rx_mbufs[index]->head;

        // 更新接收描述符尾指针，通知硬件描述符已处理
        regs[E1000_RDT] = index;
    }
```

4. 测试

在一个窗口中运行`make server`。在另一个窗口中运行`make qemu`，然后在xv6中运行`nettests`：

```
$ nettests
nettests running on port 25099
testing ping: OK
testing single-process pings: OK
testing multi-process pings: OK
testing DNS
DNS arecord for pdos.csail.mit.edu. is 128.52.129.126
DNS OK
all tests passed.
```

运行`make grade`：

```
== Test running nettests ==
$ make qemu-gdb
(4.3s)
== Test   nettest: ping ==
  nettest: ping: OK
== Test   nettest: single process ==
  nettest: single process: OK
== Test   nettest: multi-process ==
  nettest: multi-process: OK
== Test   nettest: DNS ==
  nettest: DNS: OK
== Test time ==
time: OK
Score: 100/100
```

运行`tcpdump -XXnr packets.pcap`：

```
06:42:21.901222 IP 10.0.2.2.25099 > 10.0.2.15.2003: UDP, length 17
        0x0000:  5254 0012 3456 5255 0a00 0202 0800 4500  RT..4VRU......E.
        0x0010:  002d 006c 0000 4011 6244 0a00 0202 0a00  .-.l..@.bD......
        0x0020:  020f 620b 07d3 0019 35fb 7468 6973 2069  ..b.....5.this.i
        0x0030:  7320 7468 6520 686f 7374 21              s.the.host!
06:42:21.901230 IP 10.0.2.2.25099 > 10.0.2.15.2002: UDP, length 17
        0x0000:  5254 0012 3456 5255 0a00 0202 0800 4500  RT..4VRU......E.
        0x0010:  002d 006d 0000 4011 6243 0a00 0202 0a00  .-.m..@.bC......
        0x0020:  020f 620b 07d2 0019 35fc 7468 6973 2069  ..b.....5.this.i
        0x0030:  7320 7468 6520 686f 7374 21              s.the.host!
06:42:21.901238 IP 10.0.2.2.25099 > 10.0.2.15.2001: UDP, length 17
        0x0000:  5254 0012 3456 5255 0a00 0202 0800 4500  RT..4VRU......E.
        0x0010:  002d 006e 0000 4011 6242 0a00 0202 0a00  .-.n..@.bB......
        0x0020:  020f 620b 07d1 0019 35fd 7468 6973 2069  ..b.....5.this.i
        0x0030:  7320 7468 6520 686f 7374 21              s.the.host!
06:42:21.919383 IP 10.0.2.15.10000 > 8.8.8.8.53: 6828+ A? pdos.csail.mit.edu. (36)
        0x0000:  ffff ffff ffff 5254 0012 3456 0800 4500  ......RT..4V..E.
        0x0010:  0040 0000 0000 6411 3a8f 0a00 020f 0808  .@....d.:.......
        0x0020:  0808 2710 0035 002c 0000 1aac 0100 0001  ..'..5.,........
        0x0030:  0000 0000 0000 0470 646f 7305 6373 6169  .......pdos.csai
        0x0040:  6c03 6d69 7403 6564 7500 0001 0001       l.mit.edu.....
06:42:22.000861 IP 8.8.8.8.53 > 10.0.2.15.10000: 6828 1/0/0 A 128.52.129.126 (52)
        0x0000:  5254 0012 3456 5255 0a00 0202 0800 4500  RT..4VRU......E.
        0x0010:  0050 006f 0000 4011 5e10 0808 0808 0a00  .P.o..@.^.......
        0x0020:  020f 0035 2710 003c 942a 1aac 8180 0001  ...5'..<.*......
        0x0030:  0001 0000 0000 0470 646f 7305 6373 6169  .......pdos.csai
        0x0040:  6c03 6d69 7403 6564 7500 0001 0001 c00c  l.mit.edu.......
        0x0050:  0001 0001 0000 015b 0004 8034 817e       .......[...4.~
```

实验中遇到的问题和解决办法

在实现e1000_recv()中，读取E1000_RDT控制寄存器并加一时要注意对RX_RING_SIZE取模，不然会溢出。这个函数负责处理接收的数据包，将它们从接收描述符环（RX Ring）中提取出来，要通过对 `RX_RING_SIZE` 取模以处理环状结构。

实验心得

在这次实验中，我深入了解了E1000网卡驱动程序的实现过程，特别是如何在xv6操作系统中实现网络通信。这不仅仅是一次编程实践，更是一次对底层硬件与操作系统交互机制的深刻探索。实现 `e1000_transmit()` 和 `e1000_recv()` 函数的过程中，我遇到了不少挑战。例如，如何确保发送和接收环的正确循环，如何处理并发访问问题，如何正确解析和组装网络数据包。这些问题让我反复调试代码，查阅资料，但每一次解决一个问题后的成就感，都是对我最大的鼓励。这次实验让我深刻体会到理论与实践结合的重要性。通过亲自动手实现网卡驱动程序，我不仅加深了对网络通信原理的理解，也提升了自己在系统编程方面的能力。