Lab - Networking 目录

1. networking (hard)

1.1 实验目的

- 编写xv6的网卡设备驱动程序，实现数据包的发送和接收

1.2 实验步骤

1. `qemu`模拟了硬件，并通过模拟出的E1000连接到`LAN`。在这个模拟的`LAN`上，`Guest`的IP地址为`10.0.2.15`，`Host`的IP地址为`10.0.2.2`。
2. 下面列出该实验部分文件及对应作用：
   - `kernel/e1000.c`：文件包含E1000的初始化代码以及用于发送和接收数据包的空函数
   - `kernel/e1000_dev.h`：包含E1000定义的寄存器和标志位的定义
   - `kernel/net.c`&`kernel/net.h`：QEMU中存在一个概念叫做“用户模式网络栈”（user-mode network stack），在这两个文件中包含一个实现IP（Internet Protocol）、UDP（User Datagram Protocol）和ARP（Address Resolution Protocol）协议的简单网络栈。这里将是处理请求的上层。同时，用于保存数据包的数据结构`mbuf`也位于其中。
   - `kernel/pci.c`：包含在xv6引导时在PCI总线上搜索E1000的代码
3. 对于实现`e1000_transmit()`，应当遵循如下步骤，详细注释在代码中提及：
   - 首先，通过读取`E1000_TDT`控制寄存器，从E1000获取等待下一个数据包的发送环缓冲区索引。
   - 检查环是否溢出。如果`E1000_TXD_STAT_DD`未在`E1000_TDT`索引的描述符中设置，则E1000尚未完成先前相应的传输请求，因此返回错误。
   - 若没有溢出，使用`mbuffree()`释放从该描述符传输的最后一个`mbuf`。
   - 然后在缓冲区相应位置填写描述符。`m->head`指向内存中数据包的内容，`m->len`是数据包的长度。设置相应的`cmd`标志，并保存指向`mbuf`的指针。
   - 最后，更新E1000_TDT控制寄存器，指向下一个数据包的发送环索引，通过将`E1000_TDT`加1再对`TX_RING_SIZE`取模来更新索引。
   - 如果`e1000_transmit()`成功地将`mbuf`添加到环中，则返回0。如果失败（例如没有可用的描述符来传输`mbuf`），则返回-1。

   实现代码如下：

```
int e1000_transmit(struct mbuf *m)
{
    // 对于包m，将其放入发送环中等待
    // tx_mbufs 发送缓冲区
    // tx_ring 发送环缓冲区
    // E1000_TDT 发送环索引
    // E1000_TXD_STAT_DD 发送环状态位，表示该数据包已经发送完毕

    acquire(&e1000_lock); // 锁

    int index = regs[E1000_TDT]; // 读取E1000_TDT控制寄存器，获取下一个数据包的TX环索引

    if ((tx_ring[index].status & E1000_TXD_STAT_DD) == 0) { // 仍在发送
        release(&e1000_lock);
        return -1;
    }

    if (tx_mbufs[index]) // 释放上一个数据包
```

```
        mbuffree(tx_mbufs[index]);

    tx_mbufs[index] = m;
    tx_ring[index].length = m->len;
    tx_ring[index].addr = (uint64)m->head;

    // E1000_TXD_CMD_RS表示报告状态位，表示当发送完该数据包时会产生一个中断报告状态
    // E1000_TXD_CMD_EOP表示结束位，表示这是该数据包的最后一个描述符。
    tx_ring[index].cmd = E1000_TXD_CMD_RS | E1000_TXD_CMD_EOP;

    // 更新E1000_TDT控制寄存器，指向下一个数据包的发送环索引
    // 因其是一个环状结构，故取模

    regs[E1000_TDT] = (index + 1) % TX_RING_SIZE;
    release(&e1000_lock);
    return 0;
}
```

4. 对于实现 `e1000_recv()`，应当遵循如下步骤，详细注释在代码中提及：
   - 首先通过提取 `E1000_RDT` 控制寄存器，将其索引加1并对 `RX_RING_SIZE` 取模，从而获得向E1000请求的下一个接收数据包所在的环索引。
   - 通过检查描述符 `status` 部分中的 `E1000_RXD_STAT_DD` 位来检查新数据包是否可用。如果不可用，则返回。
   - 若通过，将 `mbuf` 的 `m->len` 更新为包的长度，并使用 `net_rx()` 将 `mbuf` 传送到上层的网络栈。
   - 使用 `mbufalloc()` 分配一个新的 `mbuf`，以替换刚刚给 `net_rx()` 的 `mbuf`。将其数据指针（`m->head`）赋到描述符中并将描述符的状态位复位。
   - 最后，更新 `E1000_RDT` 寄存器，使其值为最后处理的环描述符的索引。

   实现代码如下：

```
static void e1000_recv(void)
{
    // 遍历发送环状缓冲区，把其中所有的Packet交由网络上层处理
    while (1) {

        int index = (regs[E1000_RDT] + 1) % RX_RING_SIZE;

        if ((rx_ring[index].status & E1000_RXD_STAT_DD) == 0) {
            // 已处理完
            return;
        }
        rx_mbufs[index]->len = rx_ring[index].length;

        // 向上传输
        net_rx(rx_mbufs[index]);

        // 置空
        rx_mbufs[index] = mbufalloc(0);
        rx_ring[index].status = 0;
        rx_ring[index].addr = (uint64)rx_mbufs[index]->head;
        regs[E1000_RDT] = index;
    }
}
```

5. 使用`tcpdump -XXnr packets.pcap`命令进行测试，得到的结果如下图所示：

```
07:59:43.400904 IP 10.0.2.2.26099 > 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 65f3 07d1 0019 3215 7468 6973 2069  ..e.....2.this.i
        0x0030:  7320 7468 6520 686f 7374 2100            s.the.host!.
07:59:43.418092 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.....
07:59:44.675655 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 8e7d 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 0708 0004 8034 817e       ..........4.~
```

6. 运行`nettests`测试，得到结果如下：

```
$ nettests
nettests running on port 26099
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.
```

7. 运行`grade`程序，最终结果如下：

```
== Test running nettests ==
$ make qemu-gdb
(3.8s)
== 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
```

1.3 实验中遇到的问题和解决方法

- **问题**：代码发生溢出，环缓冲区发生越界错误。

解决方法：在传输存入的过程中，放在环缓冲区中的数据包总数可能会超过环的大小（16），因此在使用时要将本质上是线性的数组在通过索引访问时对外呈现为一个环，即通过将`E1000_TDT`加1再对`TX_RING_SIZE`取模得到合理的索引值。

- **问题**：在运行`grade`时出现如下报错信息：

```
== Test time ==
time: FAIL
    Cannot read time.txt
Score: 99/100
make: *** [Makefile:337：grade] 错误 1
```

解决方法：程序无法打开`time.txt`，原因是没有足够权限，切换用户后再次运行即可。

- **问题**：难以预估的一些错误

解决方法：定位后发现错误发生在`kfree()`中，一开始的引用数为0，但在`kfree()`中执行减1命令后会发生溢出，从而变为`INTMAX-1`，无法释放物理页。

1.4 实验心得

首先，我了解了设备驱动程序的基本概念和工作原理。在实现`e1000_transmit()`和`e1000_recv()`函数时，我深入研究了E1000软件开发手册，特别是关于数据包接收和发送的部分。通过仔细阅读文档和代码，我逐步理解了驱动程序与硬件设备的交互方式，以及如何正确地初始化和配置E1000网卡。在编写代码的过程中，我遇到了一些问题，特别是在处理`mbuf`数据结构时，而要想解决还是得去阅读现有的文档和代码，更好理解其实现逻辑。同时我还初步学会了使用锁来确保多个进程或多个内核线程同时使用E1000时的数据一致性，多进程并发访问可能导致数据错误或冲突，而通过使用锁，能够确保了数据的正确传输和处理，从而提高了驱动程序的稳定性和可靠性。在成功完成实验后，我对操作系统内核和设备驱动程序有了更深入的理解，也更理解了中断和设备驱动。