# Lab7: Network Driver

2351882 王小萌

Tongji University,2025 Summer

代码仓库链接：https://github.com/wxmxmw/Tongji-University-xv6-labs-2021.git



# 1 Networking

## 1.1 实验目的

通过为网络接口卡（NIC）编写一个 xv6 设备驱动程序，来理解网络通信的核心机制。具体步骤包括创建以太网驱动程序、处理 ARP 请求和响应、实现 IP 数据包的发送和接收、处理 ICMP Echo 请求和响应、 实现 UDP 数据报的发送和接收，以及编写一个简单的 DHCP 客户端从 DHCP 服务器获取 IP 地址。通过这个实验，可以深入理解网络协议栈的工作原理。

## 1.2 实验步骤

1. 实现 kernel/e1000.c 中的 e1000_transmit() 函数以发送以太网数据帧到网卡。
   具体实现过程如下：
   1. 读取发送尾指针对应的寄存器 regs[E1000_TDT]，获取软件可以写入的位置，即发送队列的 索引 tail。
   2. 获取对应的发送描述符 desc。
   3. 检查尾指针指向的描述符的状态 status 是否包含 E1000_TXD_STAT_DD 标志位。如果未设置该标志位，说明数据尚未传输完毕，返回失败。
   4. 释放描述符对应的缓冲区（如果存在）。
   5. 更新描述符的 addr 字段为数据帧缓冲区的头部 m->head，length 字段为数据帧的长度 m >len。
   6. 更新描述符的 cmd 字段，设置 E1000_TXD_CMD_EOP 和 E1000_TXD_CMD_RS 标志位。
   7. 将数据帧缓冲区 m 记录到 tx_mbufs 中，以便后续释放。
   8. 使用 __sync_synchronize() 设置内存屏障，确保描述符更新完成后再更新尾指针。
   9. 更新发送尾指针 regs[E1000_TDT]。

```
95  int
96  e1000_transmit(struct mbuf *m)
97  {
98    uint32 tail;
99    struct tx_desc *desc;
100   acquire(&e1000_lock);
101   tail = regs[E1000_TDT];
102   desc = &tx_ring[tail];
103   //check the ring
104   if((desc->status & E1000_TXD_STST_DD)==0){
105     release(&e1000_lock);
106     return -1;
107   }
108   //free the last mbuf
109   if(tx_mbufs[tail]){
110     mbuffree(tx_mbufs[tail]);
111   }
112   //fill in
113   desc->addr=(uint64)m->head;
114   desc->length=m->len;
115   desc->cmd=E1000_TXD_CMD_EOP | E1000_TXD_CMD_RS;
116   tx_mbufs[tail] = m;
117
118   //a barrier to prevent reorder of instructions
119   __syns_synchronize();
120
121   regs[E1000_TDT]=(tail+1)%TX_RING_SIZE;
122   release(&e1000_lock);
123
124   return 0;
125 }
126
```

2. 实现 kernel/e1000.c 中的 e1000_recv() 函数以接收数据到内核。具体实现过程如下：

   1. 读取接收尾指针寄存器 regs[E1000_RDT] 并加 1 取余，获取软件可读取的位置，即接收且未处理的数据帧的描述符索引 tail。

   2. 获取对应的接收描述符 desc。

   3. 检查描述符的状态 status 是否包含 E1000_RXD_STAT_DD 标志位，确定数据帧已被硬件处理完毕，可以由内核解封装。

   4. 将 rx_mbufs[tail]中的数据帧长度记录到描述符的 length 字段，并调用 net_rx() 将数据传递给网络栈进行解封装。

   5. 调用 mbufalloc() 分配新的接收缓冲区替换发送给网络栈的缓冲区，并更新描述符的 addr 字段指向新的缓冲区。

   6. 清空描述符的状态 status 字段。

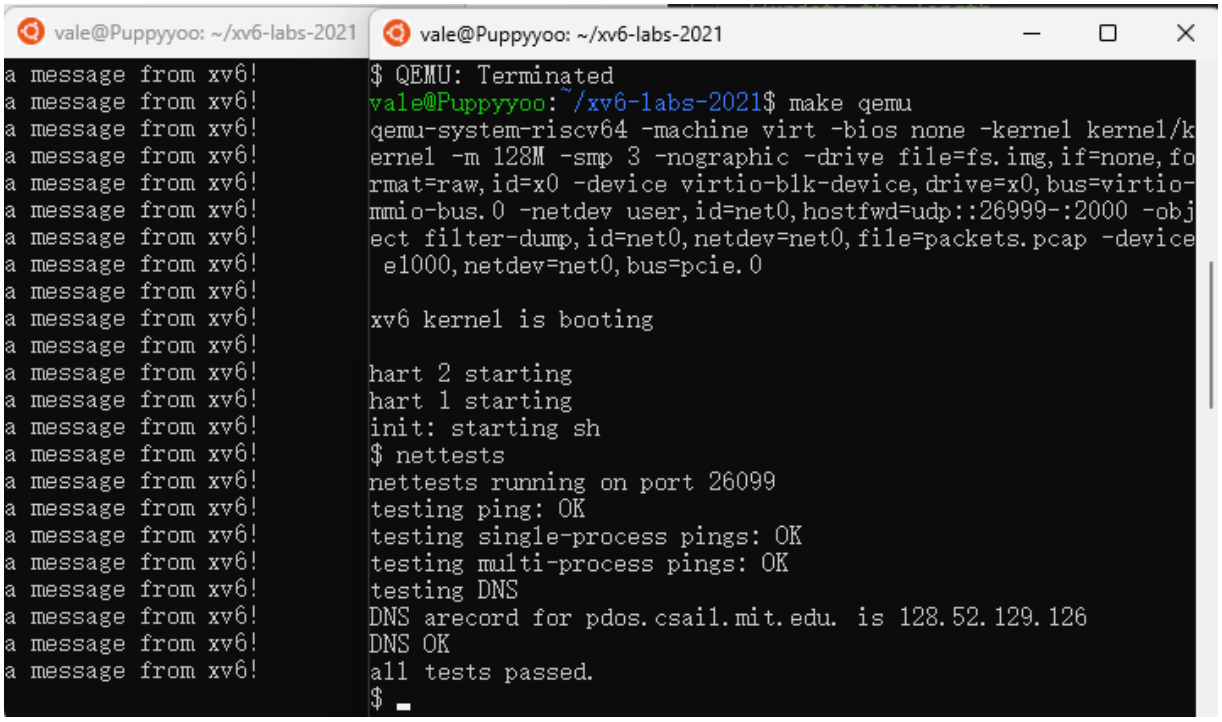
7. 继续检查下一个描述符，直到当前描述符的 DD 标志位未被设置，说明数据尚未被硬件处理完毕。
8. 更新接收尾指针 regs[E1000_RDT] 指向最后一个已处理的描述符。
9. 无需加锁，因为接收函数仅在中断处理函数 e1000_intr() 中调用，不会出现并发问题。发送和接收数据结构独立不会共享资源。

```
static void
e1000_recv(void)
{
  int tail = (regs[E1000_RDT]+1)%RX_RING_SIZE;
  struct rx_desc *desc=&rx_ring[tail];

  while((desc->status&E1000_RXD_STAT_DD)){
    if(desc->length > MBUF_SIZE){
      panic("e1000 len");
    }
    //update the length
    rx_mbufs[tail]->len = desc->length;
    //deliver the mbuf to the stack
    net_rx(rx_mbuf[tail]);
    //allocate a new buf to replace the above one
    rx_mbufs[tail] = mbufalloc(0);
    if(!rx_mbufs[tail]){
      panic("e1000 no mmbufs");
    }
    desc->addr= (uint64)rx_mbufs[tail]->head;
    desc->status =0;

    tail = (tail+1)%RX_RING_SIZE;
    desc = &rx_ring[tail];
  }
  regs[E1000_RDT]=(tail-1)%RX_RING_SIZE;
}
```

3. 在终端执行执行 make server 启动服务端。

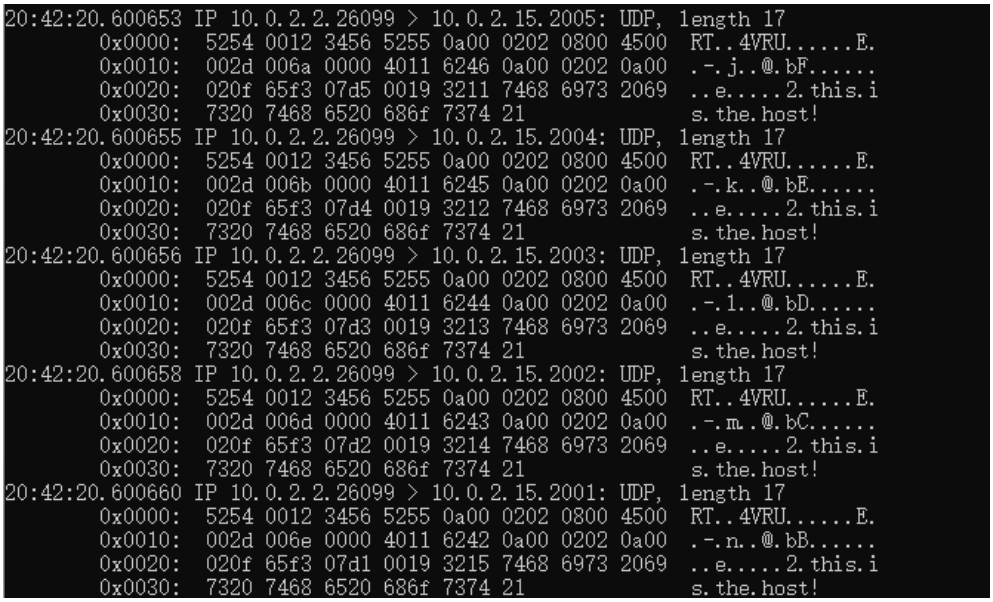
```
vale@Puppyyoo:~/xv6-labs-2021$ make server
python3 server.py 26099
listening on localhost port 26099
```

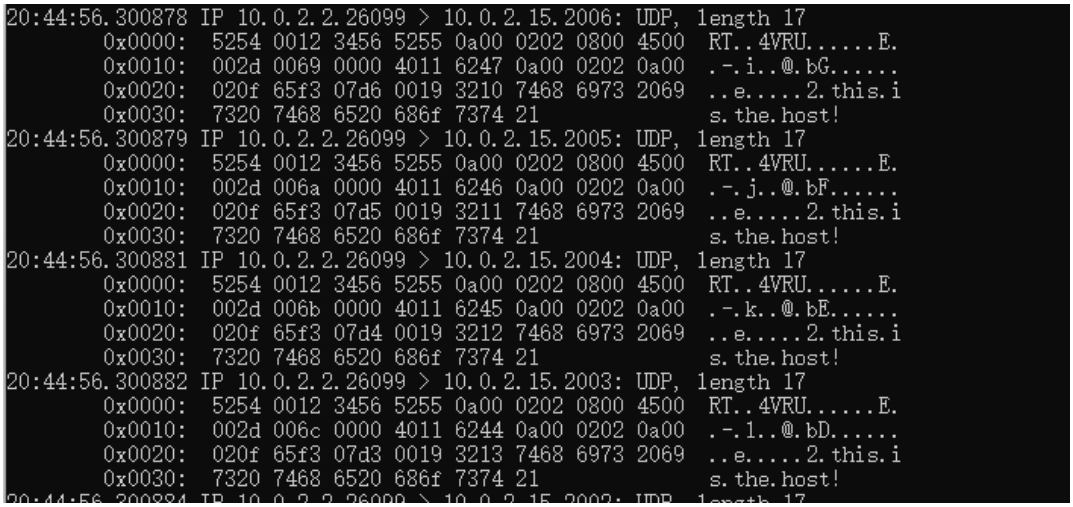
4. 另再启动一个终端。执行 make qemu 启动 xv6，然后执行 nettests 命令进行测试：




可以看到 message 发送成功，服务器接收成功。

5. 分别在两个终端执行 tcpdump -XXnr packets.pcap，可以查看捕获的报文:






## 1.3 实验中遇到的问题和解决办法

**问题 ：** 在处理接收数据时，不清楚如何判断数据帧是否已被硬件处理完毕。

**解决办法 ：** 通过检查描述符状态中的 E1000_RXD_STAT_DD 标志位，判断数据帧是否已被硬件处理完毕，确保只有已处理的数据帧才传递给网络栈。

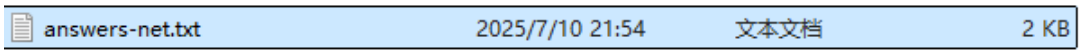
## 1.4 实验心得

通过本次实验，我们深入理解了网卡驱动程序的工作机制，尤其是发送和接收数据的处 理流程。通过实现和调试 e1000_transmit() 和 e1000_recv() 函数，我们掌握了如何管理网卡的 发送和接收队列，如何检查数据帧状态，如何进行缓冲区的分配和管理。

# 2 实验检验得分

1. 在实验目录下创建 time.txt，填写完成实验时间数。


2. 创建 answers-net.txt 文件,将程序运行结果填入。




3. 在终端中执行 make grade,得到 lab7 总分：

```
== Test running nettests ==
$ make qemu-gdb
(3.1s)
== 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
vale@Puppyyoo:~/xv6-labs-2021$
vale@Puppyyoo:~/xv6-labs-2021$
```

4. Lab7 代码提交

保存并提交到本地分支:

```
vale@Puppyyoo:~/xv6-labs-2021$ git add .
vale@Puppyyoo:~/xv6-labs-2021$ git commit -m"net finished"
[net 7d14122] net finished
 7 files changed, 110 insertions(+), 14 deletions(-)
 create mode 100644 answers-net.txt
 create mode 100644 answers-net.txt:Zone.Identifier
 create mode 100644 kernel/e1000.c:Zone.Identifier
 rewrite packets.pcap (100%)
 create mode 100644 time.txt
 create mode 100644 time.txt:Zone.Identifier
vale@Puppyyoo:~/xv6-labs-2021$ git status
On branch net
Your branch is ahead of 'origin/net' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
vale@Puppyyoo:~/xv6-labs-2021$
```