同濟大學
TONGJI UNIVERSITY

MIT 6.S081 实验报告

Lab 7

2151767 薛树建

日期：2023 年 7 月 20 日

目录

Lab7：networking

1. Your Job (hard)

1) 实验目的

本实验目的是让我们编写网卡的驱动程序。我们主要是去完成 kernel/e1000.c 中完成 e1000_transmit()和 e1000_recv()两个函数。本实验代码量不大，关键是去理解收发包的过程。

Xv6 依靠网络栈实现收发数据，收包：当网络栈需要接受包时，网卡会直接访问内存，将接收到的 RAM 的数据(包的内容)写入 rx_ring 中，接着其会向 cpu 发出硬件中断，cpu 接收中断后从 rx_ring 中读取包传递到网络栈中。发包：当网络栈需要发送包时，会先把这个包存放在 tx_ring 中，最后通过网卡发送。

2) 实验步骤

本实验的提示部分基本上已经类似伪代码了，可用直接跟着提示完成实验。

a) 首先时去完成 e1000_transmit()函数，每次都从 TDT 寄存器中获取描述符的索引，判断当前包是否已完成发送，若没有则返回

```
index = regs[E1000_TDT];//读取E1000_TDT控制寄存器
tx_dp = &tx_ring[index];//访问tx_ring获取描述符指


if ((tx_dp->status & E1000_TXD_STAT_DD) == 0) {
  release(&e1000_lock);
  return -1;
}
```

若已完成发送，则释放该索引对应的缓冲区块(tx_mbufs)，并令该指针指向参数中的包，同时修改对应的文件描述符并更新 index(TDT 寄存器)。

```
tx_mbuf = tx_mbufs[index];//获取该指针对应的数据包缓冲区，并
if (tx_mbuf) {
  mbuffree(tx_mbuf);
}

tx_mbufs[index] = m;

tx_dp->addr = (uint64)m->head;
tx_dp->length = m->len;
tx_dp->cmd = E1000_TXD_CMD_RS | E1000_TXD_CMD_EOP;
//设置RS位来使完成发送的包会自动设置E1000_TXD_STAT_DD的值
//EOP标志包(packet)的结束

index = (index + 1) % TX_RING_SIZE;//加一取模来遍历tx环，同
regs[E1000_TDT] = index;
```

最后记得要释放对应的锁。

b) 接着是 e1000_recv 函数实现，先检查获取下一个索引的值，并检查是否可用

```
int rx_index = (regs[E1000_RDT] + 1) % RX_RING_SIZE;
if ((rx_ring[rx_index].status & E1000_RXD_STAT_DD) == 0)//检查是否可用
  return ;
```

我们修改相应的 rx_mbufs 的长度，并使用 net_rx 传递到网络栈。

```
//传送到网络栈并修改对应mbufs中的len
rx_mbufs[rx_index]->len = rx_ring[rx_index].length;
net_rx(rx_mbufs[rx_index]);
```

我们申请一块新的 mbufs，同时将其数据指针（m->head）添加到 rx_ring 描述符中。将描述符的状态位清除为零。

```
rx_mbufs[rx_index] = mbufalloc(0);
rx_ring[rx_index].addr = (uint64)rx_mbufs[rx_index]->head;
rx_ring[rx_index].status = 0;

//修改对应寄存器的值
regs[E1000_RDT] = rx_index;
```

c) 测试如下：

```
tjxsj@LAPTOP-ONAKFDL3:~/xv6-labs-2021$ make server
python3 server.py 26099
listening on localhost port 26099
a message from xv6!
a message from xv6!
a message from xv6!
a message from xv6!
a message from xv6!
a message from xv6!
a message from xv6!
```

```
init: starting sh
$ 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.
```

3) 实验中遇到的问题和解决方法

本实验需要注意的一个问题是，在 e1000_transmit 中需要添加锁，因为可能会出现多个线程访问的情况。但是在 e1000_recv 中不需要添加锁，在实验网站上说的是 e1000_recv 应该传送每一个包到网络栈中通过调用 net_rx 函数。

参考网络上的回答：因为 recv 过程发生在 OSI 协议的数据链路层，而在这个层次的 packet 接收是不区分进程。既然不区分进程接收，自然就可以一直接收，直到不能再接收为止。我们在写该函数时不需要考虑多线程的情况，可用使用 while 循环来保证接受所有的包。

4) 实验心得

在完成本实验之后，我对于网络驱动程序中的发送接收包的过程有了一定的了解，发送包的时候直接通过将该包放入 rx_ring 缓冲区中，最后通过网卡进行发送；在接受包的过程，程序将所有的包的接受在一个进程中完成。这样可以提高接收效率，减少 CPU 的干预。但是需要程序在缓冲区完之后尽快释放，以便接受新的数据包。

总结

通过完成本实验，我对操作系统和网络编程有了更深入的理解。这个实验不仅提高了我的编程能力，也拓宽了我的计算机科学知识。这对我今后在网络编程和系统设计中的应用都非常有帮助。

本实验总体评分如下：

```
tjxsj@LAPTOP-ONAKFDL3:~/xv6-labs-2021$ ./grade-lab-net
make: 'kernel/kernel' is up to date.
== Test running nettests == (2.7s)
== 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
```