Lab11 networking

1. networking(hard)

1）实验目的

> Your job is to complete `e1000_transmit()` and `e1000_recv()`, both in `kernel/e1000.c`, so that the driver can transmit and receive packets. You are done when `make grade` says your solution passes all the tests.
> 您的工作是在 `kernel/e1000.c` 中完成 `e1000_transmit` 和 `e1000_recv()`，以便驱动程序可以发送和接收数据包。当 `make grade` 表示你的解决方案通过了所有测试时，你就完成了。

2）实验步骤

编写代码

在 `kernel/e1000.c` 实现 `e1000_transmit()` 和 `e1000_recv()` 两个函数

```
int
e1000_transmit(struct mbuf *m)
{
  //
  // Your code here.
  //
  // the mbuf contains an ethernet frame; program it into
  // the TX descriptor ring so that the e1000 sends it. Stash
  // a pointer so that it can be freed after sending.
  //
  acquire(&e1000_lock);
  uint32 next_index = regs[E1000_TDT];
  if((tx_ring[next_index].status & E1000_TXD_STAT_DD) == 0){
      release(&e1000_lock);
      return -1;
  }
  if(tx_mbufs[next_index])
      mbuffree(tx_mbufs[next_index]);
  tx_ring[next_index].addr = (uint64)m->head;
  tx_ring[next_index].length = (uint16)m->len;
  tx_ring[next_index].cmd = E1000_TXD_CMD_EOP | E1000_TXD_CMD_RS;
  tx_mbufs[next_index] = m;
  regs[E1000_TDT] = (next_index+1)%TX_RING_SIZE;
  release(&e1000_lock);
  return 0;
}

static void
e1000_recv(void)
{
  //
  // Your code here.
  //
  // Check for packets that have arrived from the e1000
  // Create and deliver an mbuf for each packet (using net_rx()).
  //
  uint32 next_index = (regs[E1000_RDT]+1)%RX_RING_SIZE;
  while(rx_ring[next_index].status & E1000_RXD_STAT_DD){
    if(rx_ring[next_index].length>MBUF_SIZE){
        panic("MBUF_SIZE OVERFLOW!");
    }
    rx_mbufs[next_index]->len = rx_ring[next_index].length;
    net_rx(rx_mbufs[next_index]);
    rx_mbufs[next_index] =  mbufalloc(0);
    rx_ring[next_index].addr = (uint64)rx_mbufs[next_index]->head;
    rx_ring[next_index].status = 0;
    next_index = (next_index+1)%RX_RING_SIZE;
  }
  regs[E1000_RDT] = (next_index-1)%RX_RING_SIZE;
```

```
}
```

测试程序

```
root@LAPTOP-UER420HO:~/xv6-labs-2020# make server
python3 server.py 25099
listening on localhost port 25099


$ 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.
```

同时服务器也会接收数据

```
root@LAPTOP-UER420HO:~/xv6-labs-2020# make server
python3 server.py 25099
listening on localhost port 25099
a message from xv6!
a message from xv6!
a message from xv6!
...
```

键入 `Ctrl+c` 服务端终止运行，键入 `tcpdump -XXnr packets.pcap` 查看传输数据内容

```
root@LAPTOP-UER420HO:~/xv6-labs-2020# tcpdump -XXnr packets.pcap
...
10:11:21.110583 IP 10.0.2.15.10000 > 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.....
10:11:21.200944 IP 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 8ffc 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 0589 0004 8034 817e       ...........4.~
```

传输大量数据

回到开启xv6系统的终端，键入 `Ctrl+a` ， 松开，然后键入 `x` ， 退出xv6系统
退出xv6进行单元测试

```
root@LAPTOP-UER420HO:~/xv6-labs-2020# ./grade-lab-net
make: 'kernel/kernel' is up to date.
== Test running nettests == (2.9s)
== 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
```

3）实验中遇到的问题和解决方法

本次实验的代码量应该是所有实验中最少的，但也是相对难以理解的。因为没有学习过相关网络的知识，所以只能跟着提示和所查询的资料一步步实现

4）实验心得

遇到不了解的方面，我们要先去看看相关的概念才好入手，否则就算实验通过了也不知道自己所写的每一行代码是用来做什么的，在其中发挥什么样的功能。实际上本实验就是实现一个简陋的网卡驱动

发送函数

- 我们首先加锁，保护发送数据的时候不会有其他进程来干扰，然后从寄存器中获取**下一个可用的传输描述符**的索引，通过读取 `E1000_TDT` ，然后检查 `ring` 是否溢出。如果 `E1000_TXD_STAT_DD` 没有在 `E1000_TDT` 索引的描述符中被设置，则 `E1000` 还没有结束之前的传输请求，返回error
- 若已经设置，使用 `mbuffree()` 函数释放最后的（索引到的描述符指向的）已经被发送（但还没有释放）的 `mbuf` （如果使用过的话）
- 然后设置 `mbuf` 的传输数据的地址以及数据大小，已经传输描述符的命令字段
- 这里需要查询以下
  - `E1000_TXD_CMD_EOP` 表示该数据包是帧的结束
  - `E1000_TXD_CMD_RS` 表示发送后释放该描述符
- 然后将 `m` 存储到 `tx_mbuf` 数组中，以便在发送后可以释放
- 最后更新寄存器数组中下一个可用描述符的索引

接收函数

- 首先从寄存器中获取下一个可用的接收描述符的索引
- 利用 `while` 循环和 `E1000_RXD_STAT_DD` 判断是否有心得数据包准备好接收
- 然后将接收到的数据包的长度存储在对应的 `mbuf` 中
- 将接收到的数据包传输给网络层处理，用来实现解析数据包的内容等操作
- 接着给下一个接受描述符分配一个新的 `mbuf` ，以便接收下一个数据包
- 将新的mbuf的头部地址存储到接收描述符中，以便驱动程序可以将数据写入该地址
- 将接收描述符的状态字段清零，表示已经处理该数据包
- 更新下一个可用接收描述符的索引
- 更新寄存器中的接收描述符表的尾部索引

以太网

实验中提到的以太网是一种常见的局域网技术，用于在计算机和设备之间传输数据。它是最常用的有线局域网技术之一，广泛应用于家庭、办公室和企业网络中

以太网使用双绞线或光纤等物理媒介来传输数据。它基于一组标准化的通信协议和规范，其中最常见的是 `IEEE 802.3` 系列标准。这些标准定义了数据传输的格式、速率、错误检测和纠正等方面的规范，确保不同设备之间可以相互通信和交换数据

以太网的工作原理是通过发送和接收数据帧来实现。数据帧是数据传输的基本单位，包括源和目的地址、数据内容以及校验和等信息。计算机或设备通过网络交换机或集线器等网络设备连接到以太网，它们可以通过发送数据帧来与其他设备进行通信

数据包

数据包是在计算机网络中传输的基本单位。它是将数据按照特定的格式进行封装和分割后的一段信息。数据包通常包含了源地址、目的地址、控制信息、有效载荷等部分

在计算机网络中，数据通常被分割成较小的数据包进行传输。这样做的好处是可以提高网络的效率和可靠性。较大的数据可以被分割成多个数据包，每个数据包独立传输，到达目的地后再重新组装成完整的数据

数据包的封装和分割过程是在网络协议栈中的传输层和网络层完成的。在传输层，数据被封装成传输层协议的数据段或用户数据报。在网络层，数据段或用户数据报再被封装成网络层协议的数据包

数据包的封装和解封装过程中，会在数据包的头部添加一些控制信息，如源地址和目的地址，以便网络设备能够正确地将数据包传输到目的地。数据包还可以包含一些错误检测和纠正的校验和，以确保数据的完整性和准确性

在网络中，数据包通过网络设备（如路由器、交换机）在不同的网络节点之间进行转发和交换，最终到达目的地。在目的地，数据包被解封装，提取出有效载荷中的数据，并进行相应的处理和应用