

网络

YuanShen OS的网络模块参考了MITS6.081 2024的课程实验network driver。该实验要求实现e1000网卡驱动的收发功能，并实现IP报文和UDP报文的解析与校验，最终添加四个系统调用实现UDP报文的收发：

```
int bind(uint16);
int unbind(uint16);
int send(uint16, uint32, uint16, char *, uint32);
int recv(uint16, uint32*, uint16*, char *, uint32);
```

我们完成了实验要求的内容，在通过测试后将代码移植到YuanShen OS，使之也具备了与主机进程进行网络通信的能力。

注: 本文使用的所有图片均来自于手册PCI/PCI-X Family of Gigabit Ethernet Controllers Software Developer’s Manual。

1. e1000以太网卡驱动配置

配置e1000用到的寄存器映射如下文所示：

```
#define E1000_CTL      (0x00000/4)  /* Device Control Register - RW */
#define E1000_ICR      (0x000C0/4)  /* Interrupt Cause Read - R */
#define E1000_IMS      (0x000D0/4)  /* Interrupt Mask Set - RW */
#define E1000_RCTL      (0x00100/4)  /* RX Control - RW */
#define E1000_TCTL      (0x00400/4)  /* TX Control - RW */
#define E1000_TIPG      (0x00410/4)  /* TX Inter-packet gap -RW */
#define E1000_RDBAL     (0x02800/4)  /* RX Descriptor Base Address Low - RW
*/
#define E1000_RDTR      (0x02820/4)  /* RX Delay Timer */
#define E1000_RADV      (0x0282C/4)  /* RX Interrupt Absolute Delay Timer */
#define E1000_RDH      (0x02810/4)  /* RX Descriptor Head - RW */
#define E1000_RDT      (0x02818/4)  /* RX Descriptor Tail - RW */
#define E1000_RDLEN     (0x02808/4)  /* RX Descriptor Length - RW */
#define E1000_RSRPD     (0x02C00/4)  /* RX Small Packet Detect Interrupt */
#define E1000_TDBAL     (0x03800/4)  /* TX Descriptor Base Address Low - RW
*/
#define E1000_TDLEN     (0x03808/4)  /* TX Descriptor Length - RW */
#define E1000_TDH      (0x03810/4)  /* TX Descriptor Head - RW */
#define E1000_TDT      (0x03818/4)  /* TX Descriptor Tail - RW */
#define E1000_MTA      (0x05200/4)  /* Multicast Table Array - RW Array */
#define E1000_RA      (0x05400/4)  /* Receive Address - RW Array */
```

1.1 网卡初始化

参照手册PCI/PCI-X Family of Gigabit Ethernet Controllers Software Developer’s Manual的第14章可以了解e1000网卡的收发所需的初始化步骤,如下图所示：

14.4 Receive Initialization

Program the Receive Address Register(s) (RAL/RAH) with the desired Ethernet addresses. RAL[0]/RAH[0] should always be used to store the Individual Ethernet MAC address of the Ethernet controller. This can come from the EEPROM or from any other means (for example, on some machines, this comes from the system PROM not the EEPROM on the adapter port).

Initialize the MTA (Multicast Table Array) to 0b. Per software, entries can be added to this table as desired.

Program the Interrupt Mask Set/Read (IMS) register to enable any interrupt the software driver wants to be notified of when the event occurs. Suggested bits include RXT, RXO, RXDMT, RXSEQ, and LSC. There is no immediate reason to enable the transmit interrupts.

If software uses the Receive Descriptor Minimum Threshold Interrupt, the Receive Delay Timer (RDTR) register should be initialized with the desired delay time.

Allocate a region of memory for the receive descriptor list. Software should insure this memory is aligned on a paragraph (16-byte) boundary. Program the Receive Descriptor Base Address (RDBAL/RDBAH) register(s) with the address of the region. RDBAL is used for 32-bit addresses and both RDBAL and RDBAH are used for 64-bit addresses.

Set the Receive Descriptor Length (RDLEN) register to the size (in bytes) of the descriptor ring. This register must be 128-byte aligned.

The Receive Descriptor Head and Tail registers are initialized (by hardware) to 0b after a power-on or a software-initiated Ethernet controller reset. Receive buffers of appropriate size should be allocated and pointers to these buffers should be stored in the receive descriptor ring. Software initializes the Receive Descriptor Head (RDH) register and Receive Descriptor Tail (RDT) with the appropriate head and tail addresses. Head should point to the first valid receive descriptor in the descriptor ring and tail should point to one descriptor beyond the last valid descriptor in the descriptor ring.

里面提到的最为核心的概念就是transmit descriptor和receive descriptor。

为了接收以太网帧，我们需要初始化一个receive descriptor组成的环形描述符队列，如下图和下列代码所示：

```
memset(rx_ring, 0, sizeof(rx_ring));
for (i = 0; i < RX_RING_SIZE; i++) {
    rx_bufs[i] = kalloc();
    if (!rx_bufs[i])
        panic("e1000");
    rx_ring[i].addr = (uint64) rx_bufs[i];
}
```

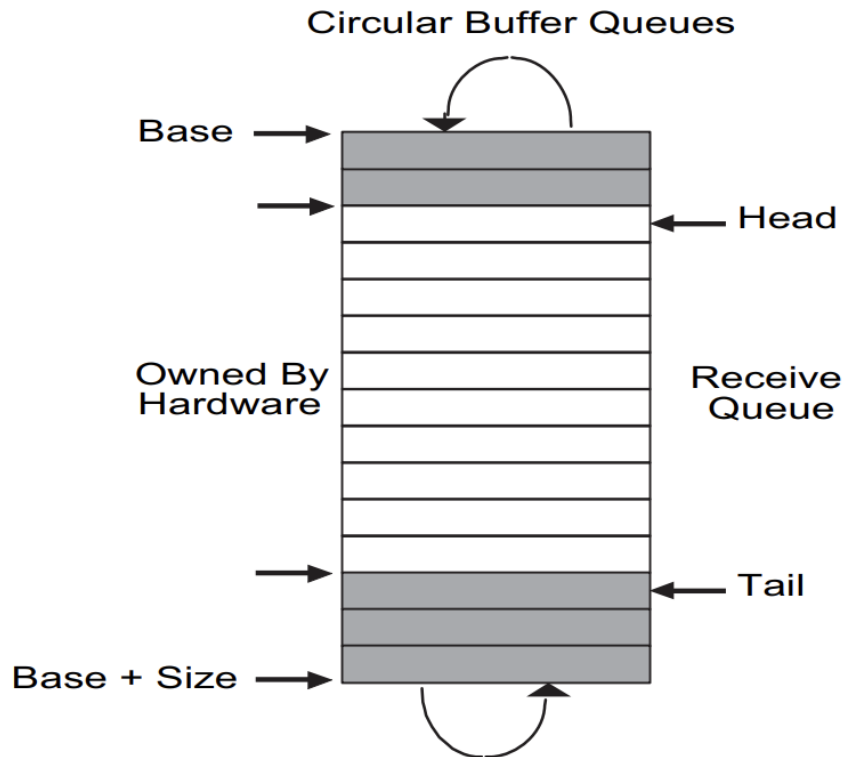


Figure 3-2. Receive Descriptor Ring Structure

我们需要完成的就是设置e1000 controller的各种配置寄存器，将RDBAL寄存器防止环形队列的首地址，然后设置队头和队尾的偏移量，最后设置环形队列的大小。

```
regs[E1000_RDBAL] = (uint64) rx_ring;
if(sizeof(rx_ring) % 128 != 0)
    panic("e1000");
regs[E1000_RDH] = 0;
regs[E1000_RDT] = RX_RING_SIZE - 1;
regs[E1000_RDLEN] = sizeof(rx_ring);
```

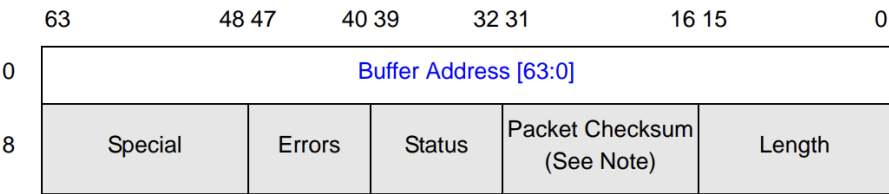
那么描述符里究竟有什么呢？根据数据手册，描述符主要维护一次收发的信息，包括以太网帧的长度，用户buffer的位置(e1000会用DMA的方式把数据传输到物理内存，具体的位置就是通过设置描述符的addr字段实现

的),如下图和下列代码所示:

3.2.3 Receive Descriptor Format

A receive descriptor is a data structure that contains the receive data buffer address and fields for hardware to store packet information. Table 3-1 lists where the shaded areas indicate fields that are modified by hardware upon packet reception.

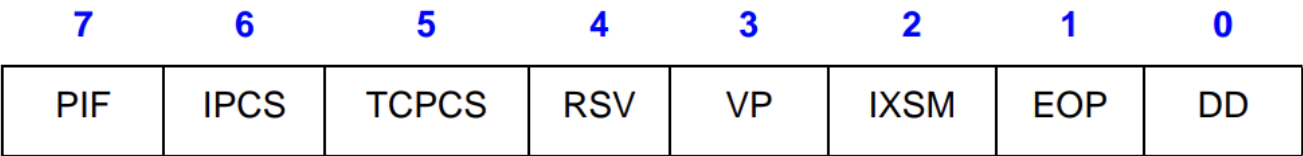
Table 3-1. Receive Descriptor (RDESC) Layout



```
struct rx_desc
{
    uint64 addr;          /* Address of the descriptor's data buffer */
    uint16 length;        /* Length of data DMAed into data buffer */
    uint16 csum;          /* Packet checksum */
    uint8 status;         /* Descriptor status */
    uint8 errors;         /* Descriptor Errors */
    uint16 special;
};
```

status里维护一些状态信息，如下图所示:

Table 3-2. Receive Status (RDESC.STATUS) Layout



这里我们要用到的是DD标志位，表示是否以太网帧已经在内存中，如下图所示:

DD (bit 0)	Descriptor Done Indicates whether hardware is done with the descriptor. When set along with EOP, the received packet is complete in main memory.
------------	---

当有以太网帧到达物理内存后，e1000会发出设备中断，我们使用中断方式调用e1000_recv(),将数据从e1000缓冲区移动到UDP端口对应的缓冲区,然后重新设置队尾指针,由下文代码所示:

```
static void
e1000_recv(void)
{
    //your code here
    while(1){
```

```

uint32 pos = (regs[E1000_RDT] + 1) % RX_RING_SIZE;
struct rx_desc *ptr = &rx_ring[pos];
if((ptr->status & E1000_RXD_STAT_DD) == 0){
    return;
}
net_rx((char*)ptr->addr, ptr->length);
ptr->addr = (uint64)kalloc();
ptr->status = (uint8)0;
regs[E1000_RDT] = pos;
}
}

```

接收的过程也是类似的，不与赘述。

2 UDP-IP协议

这个任务主要分为两个部分，分别为报文解析校验与搭建所需的内核数据结构。

2.1 报文解析

根据MIT的实验文档，我们首先实现了这个函数，它主要完成的就是报文解析，并把对应的udp报文解复用到对应的端口缓冲区。

```
void ip_rx(char *buf, int len)
```

报文解析所需的结构体如下文所示，这几个结构体分别用于解析以太网帧，IP数据报以及UDP报文：

```

// an Ethernet packet header (start of the packet).
struct eth {
    uint8  dhost[ETHADDR_LEN];
    uint8  shost[ETHADDR_LEN];
    uint16 type;
} __attribute__((packed));
// an IP packet header (comes after an Ethernet header).
struct ip {
    uint8  ip_vhl; // version <= 4 | header length >= 2
    uint8  ip_tos; // type of service
    uint16 ip_len; // total length, including this IP header
    uint16 ip_id;  // identification
    uint16 ip_off; // fragment offset field
    uint8  ip_ttl; // time to live
    uint8  ip_p;   // protocol
    uint16 ip_sum; // checksum, covers just IP header
    uint32 ip_src, ip_dst;
};

struct udp {
    uint16 sport; // source port
    uint16 dport; // destination port
}

```

```
uint16 ulen; // length, including udp header, not including IP header
uint16 sum;  // checksum
};
```

我们只需要利用这些结构体，在正确的地址处设置结构体指针，就可以操纵报文的对应字段。

```
struct eth *eth = (struct eth *)buf;
struct ip *ip = (struct ip*)(eth + 1);
struct udp *udp = (struct udp*)(ip + 1);
```

唯一值得注意的问题就是网络相关的数据通常都采用的是big-endianness存储，而qemu模拟的硬件是小端存储，因此需要对超过一个字节长度的字段做转化。

2.2 端口缓冲区与解复用

接下来我们还加入了端口缓冲区的部分，它其实是报文的一个队列容器，如下文所示:

```
struct udp_queue {
    struct udp_packet *packets[UDP_QUEUE_SIZE];
    int head;
    int tail;
    int count;
};

struct udp_port {
    struct udp_queue queue;
    int bound;
    int pid;
};
```

YuanShen OS拥有65536个UDP端口，每个端口能够缓存16个报文，多余的来不及处理的报文会被丢弃。我们回过来看我们实现的系统调用:

```
int bind(uint16);
int unbind(uint16);
```

它们的实现非常简单，只需要检查对应端口的bound字段即可，它记录了获取这个端口的pid。当进程被释放的时候，这个端口的绑定关系也应当被解除，因此我们应当在进程控制块内记录进程正在使用的所有端口号，并在进程终结后主动回收这些端口。

解复用的过程也极为简单，简单说起来就是先检查端口是否被占用，然后检查端口的缓冲区是否已经满了，最后再把报文给放入对应的队列里面。

测试

nettest测试

在项目目录键入python nettest.py rx测试UDP报文的接收能力。然后运行make qemu，发现报文接收成功，运行结果如下图所示：

```
.. .. ..va 0x00000003ffffff000 : idx 510 pte 0x0000000020385c07 pa 0x0000000080e0100
.. .. ..va 0x00000003ffffff000 : idx 511 pte 0x000000002000280b pa 0x000000008000a00
init: starting sh
$ arp_rx: received an ARP packet
ip_rx: received an IP packet
```

键入python nettest.py tx可以测试UDP报文的发送能力。运行make qemu后输入nettest tx后可以观察到主机上的nettest接收到了qemu的网卡发送而来的报文。

参考材料

E1000技术手册: Intel. (2009). 8254x Gigabit Ethernet 控制器软件开发手册.

https://pdos.csail.mit.edu/6.1810/2024/readings/8254x_GBe_SDM.pdf

网卡驱动实验: MIT. (2024). Lab: Networking. <https://pdos.csail.mit.edu/6.1810/2024/labs/net.html>