Lab 7 Networking

目录

[1. Networking (hard) 1](#_Toc174491389)

[1.1 实验目的 1](#_Toc174491390)

[1.2 实验步骤 1](#_Toc174491391)

[1.3 实验结果 4](#_Toc174491392)

[1.4 实验中遇到的问题和解决方法 4](#_Toc174491393)

[1.5 实验心得 5](#_Toc174491394)

[2. 测试结果 6](#_Toc174491395)

Networking (hard)

* 1. 实验目的
* 通过实现一个简单的以太网驱动,进一步理解操作系统内核网络栈和网卡驱动的工作机制
* 熟练掌握相关的数据结构和功能接口,获得编写设备驱动程序的实际经验
  1. 实验步骤
* kernel/e1000.c: 初始化E1000网卡，设置发送和接收队列，并处理数据包的发送和接收，通过中断机制来实现网络通信的实时处理。
* kernel/e1000\_dev.h: 通过定义硬件寄存器、控制位、描述符结构及其相关命令和状态，提供了与E1000网卡进行交互的基础接口和数据结构。
* kernel/net.c: 实现了基本的网络协议支持，包括ARP、IP和UDP协议的处理，并提供了mbuf缓冲区的管理功能。
* kernel/pci.c: 初始化PCI-Express总线上的E1000网卡。在QEMU虚拟机环境中，它通过扫描PCI设备，识别并配置E1000网卡的寄存器，并调用相应的初始化函数，从而使E1000网卡能够在xv6操作系统中正常工作。
* 完成 e1000\_transmit() 函数：

读取E1000\_TDT控制寄存器，获取发送队列的当前索引。

检查发送队列是否溢出，若E1000\_TXD\_STAT\_DD位未设置，则返回错误。

释放上次传输的 mbuf 。

填写发送描述符，设置必要的命令标志，并将 mbuf 的指针存储起来以便稍后释放。

更新E1000\_TDT寄存器，指向下一个可用的发送描述符。

int e1000\_transmit(struct mbuf \*m)

{

// 获取锁，确保多线程环境下的同步

acquire(&e1000\_lock);

// 读取E1000\_TDT控制寄存器，获取下一个可用的发送描述符索引

int index = regs[E1000\_TDT];

// 检查当前索引位置的发送描述符是否已完成发送（通过检查E1000\_TXD\_STAT\_DD位）

if ((tx\_ring[index].status & E1000\_TXD\_STAT\_DD) == 0) {

// 如果发送未完成，释放锁并返回错误

release(&e1000\_lock);

return -1;

}

// 如果当前索引位置有上一个发送的mbuf，释放它

if (tx\_mbufs[index])

mbuffree(tx\_mbufs[index]);

// 将当前要发送的mbuf保存到发送缓冲区数组中

tx\_mbufs[index] = m;

// 设置发送描述符的长度字段为当前mbuf的长度

tx\_ring[index].length = m->len;

// 设置发送描述符的地址字段为当前mbuf数据的首地址

tx\_ring[index].addr = (uint64)m->head;

// 设置发送描述符的命令字段，包括报告状态（E1000\_TXD\_CMD\_RS）和数据包结束（E1000\_TXD\_CMD\_EOP）

tx\_ring[index].cmd = E1000\_TXD\_CMD\_RS | E1000\_TXD\_CMD\_EOP;

// 更新E1000\_TDT控制寄存器，指向下一个可用的发送描述符索引（环状结构，取模运算）

regs[E1000\_TDT] = (index + 1) % TX\_RING\_SIZE;

// 释放锁

release(&e1000\_lock);

// 返回成功

return 0;

}

* 完成 e1000\_recv() 函数：

读取E1000\_RDT控制寄存器并加一，获取接收队列的当前索引。

检查接收队列中是否有新数据包，若E1000\_RXD\_STAT\_DD位未设置，则停止处理。

更新 mbuf 的长度，将数据包传递给网络堆栈（调用 net\_rx() 函数）。

分配新的 mbuf 并替换已处理的 mbuf ，清除描述符的状态位。

更新E1000\_RDT寄存器，指向上一个已处理的接收描述符。

static void e1000\_recv(void)

{

// 无限循环遍历接收描述符环，处理所有已接收的数据包

while (1) {

// 获取下一个接收描述符的索引，环状结构，取模运算

int index = (regs[E1000\_RDT] + 1) % RX\_RING\_SIZE;

// 检查当前描述符的状态位，判断是否有新的数据包

if ((rx\_ring[index].status & E1000\_RXD\_STAT\_DD) == 0) {

// 如果没有新的数据包，退出循环

return;

}

// 设置当前缓冲区的长度为接收描述符中记录的长度

rx\_mbufs[index]->len = rx\_ring[index].length;

// 将接收到的数据包传递给网络层处理

net\_rx(rx\_mbufs[index]);

// 分配新的mbuf缓冲区，以便接收新的数据包

rx\_mbufs[index] = mbufalloc(0);

// 检查分配是否成功

if (!rx\_mbufs[index])

panic("e1000\_recv: mbufalloc failed");

// 清除当前描述符的状态位

rx\_ring[index].status = 0;

// 更新描述符的地址为新分配的缓冲区地址

rx\_ring[index].addr = (uint64)rx\_mbufs[index]->head;

// 更新接收描述符尾指针，通知硬件描述符已处理

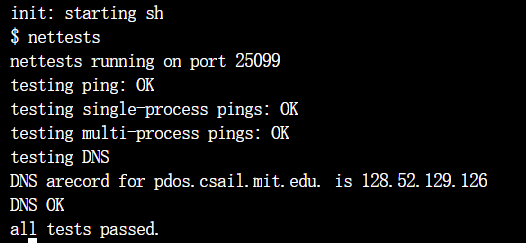
regs[E1000\_RDT] = index;

}

}

* 1. 实验结果

在一个窗口中运行 make server 。在另一个窗口中运行 make qemu ，然后在xv6中运行nettests ：



* 1. 实验中遇到的问题和解决方法
* 问题：

如何处理数据包的发送和接收的时序问题。

* 解决方法：

通过注意到E1000网卡是异步地发送和接收数据包的，我明白了需要使用锁来保护传输环和接收环的访问，以及使用内存屏障来防止指令重排。同时需要及时更新传输环和接收环的尾部索引，以及检查描述符的状态，来保证数据包的正确发送和接收。

* 1. 实验心得

实验让我深入学习了如何通过编写代码来初始化网卡、设置发送和接收队列，并处理数据包的发送和接收。通过实现 e1000\_transmit() 和 e1000\_recv() 函数，我理解了发送和接收数据包的全过程。这让我对网络驱动程序的工作机制有了更深入的认识，特别是在网络通信的实时处理方面如何通过中断机制实现高效的通信。

我学习了如何使用 kernel/e1000\_dev.h 中定义的硬件寄存器和描述符结构，与E1000网卡进行交互，掌握了设备驱动程序的基本编写方法。在实现 e1000\_recv() 时，我深入理解了网络协议栈的处理流程，并认识到网卡驱动程序需要与操作系统的网络协议栈紧密配合。我学会了如何在QEMU环境中初始化和配置PCI设备，成功识别并配置了E1000网卡，加深了对PCI设备管理的理解

测试结果

