## LAB6

这个实验开始进入实现网络系统，看介绍说这个Lab比前面的都要难，感觉还是有点慌的。

QEMU的用户模式有一个网络栈和虚拟E1000网卡用于这个实验。

QEMU有一个虚拟路由器，IP为10.0.2.2，而JOS对应会被赋予一个虚拟IP为10.0.2.15，这些在net/ns.h中都有。

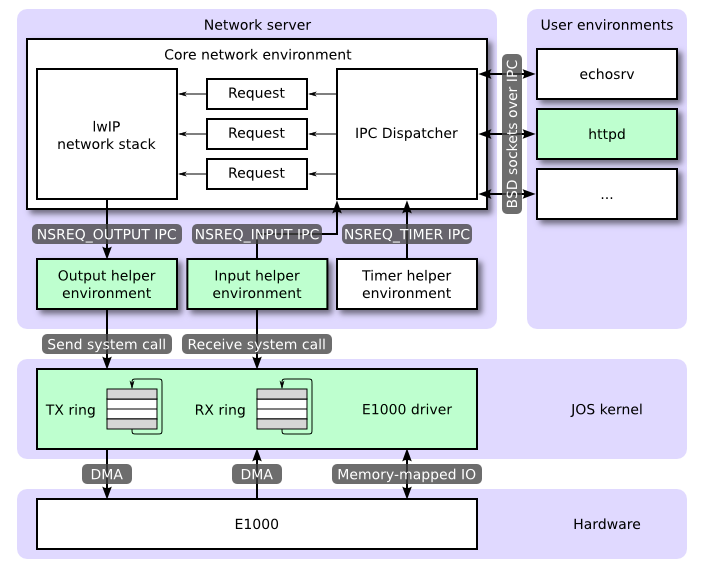


使用tcpdump -XXnr qemu.pcap 来获取捕获的数据包的ASCII码

使用make E1000\_DEBUG=tx,txerr ....能显示E1000虚拟网卡的调试信息

本lab实现的是lwIP，一个简易的TCP/IP协议栈

本实验中需要实现下图中绿色部分的内容。



这里的ring就表示环形缓冲区

这里的内核网络环境就类似于lab5的文件系统，也是一个服务器进程，对用户服务。lib/nsipc.c)

里面有一些用户可以使用的工具来向网络服务器发送请求。一般用户是用lib/sockets.c中的基于文件描述符的接口来访问的。

对于每个IPC，dispatcher都会创建一个新的线程来处理，避免整个系统的阻塞。

lwIP相当于实现了整个协议栈，能够把用户传来的数据直接变成最后网卡要传的数据。所以这个实验不需要实现网络中间那些协议栈了。

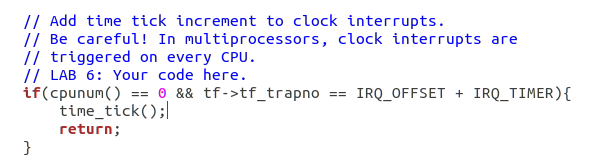
lwIP会使用NSREQ\_OUTPUT这个IPC，并且把包数据作为附加页传给Output模块。

Input模块则负责从网卡接受数据，取出内核态并使用NSREQ\_INPUT这个IPC交给上层网络协议栈

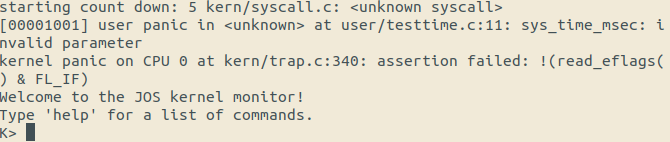
#### Exercise 1

Add a call to time\_tick for every clock interrupt in kern/trap.c. Implement sys\_time\_msec and add it to syscall in kern/syscall.c so that user space has access to the time.

注释里提示说，多核系统中每个核都有独立的时钟。那这里tick是个全局变量，所有核都共享的，所以应该只有一个核的时钟能触发它？

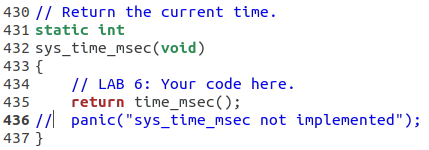


不知道为什么，好像没法测make INIT\_CFLAGS=-DTEST\_NO\_NS run-testtime



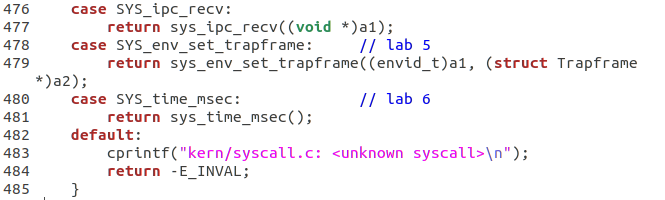
刚上来就有问题，去看看user/testtime.c里面有啥。

原来是还需要实现sys\_time\_msec函数。

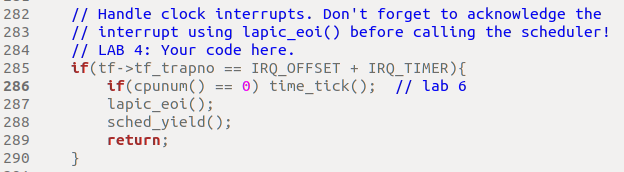


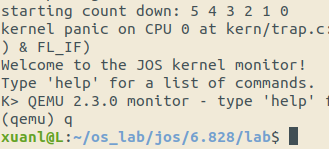
别忘了还要把这个加到系统里，要不然switch找不到还是报错。

加了之后没有panic了，但是时钟不走，这个不知道咋整。。。



最后把那一段整合到之前LAB 4的代码里，就可以了，之前因为执行完lab4就不往后执行了，所以后面的代码写了没用





#### Exercise 2

Browse Intel's [Software Developer's Manual](https://pdos.csail.mit.edu/6.828/2016/readings/hardware/8254x_GBe_SDM.pdf) for the E1000. This manual covers several closely related Ethernet controllers. QEMU emulates the 82540EM.

You should skim over chapter 2 now to get a feel for the device. To write your driver, you'll need to be familiar with chapters 3 and 14, as well as 4.1 (though not 4.1's subsections). You'll also need to use chapter 13 as reference. The other chapters mostly cover components of the E1000 that your driver won't have to interact with. Don't worry about the details right now; just get a feel for how the document is structured so you can find things later.

While reading the manual, keep in mind that the E1000 is a sophisticated device with many advanced features. A working E1000 driver only needs a fraction of the features and interfaces that the NIC provides. Think carefully about the easiest way to interface with the card. We strongly recommend that you get a basic driver working before taking advantage of the advanced features.

这个Exercise主要是阅读Intel手册中关于E1000网卡的部分，主要看第2、3、4.1和13、14章。

这文档感觉很难看懂，太多术语都不知道，一堆缩写，基本没法看。

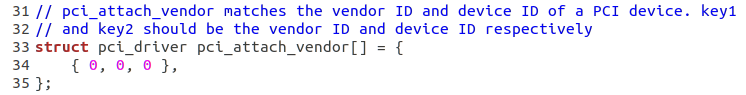
在kern/pci.c中，实现了PCI总线相关的操作，

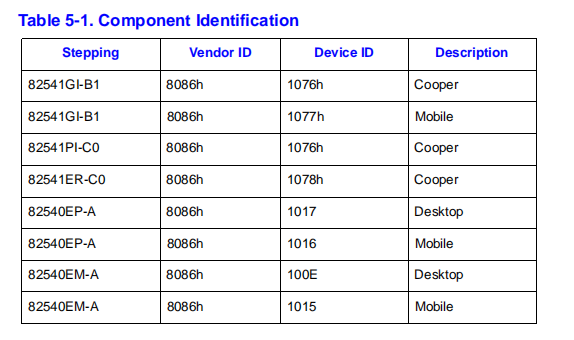
#### Exercise 3

先在init.c中找到了Lab 6的描述，然后看一下pci\_init函数干了什么



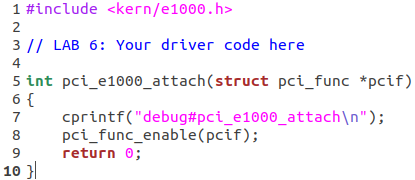
需要在下面这个数组中增加一项





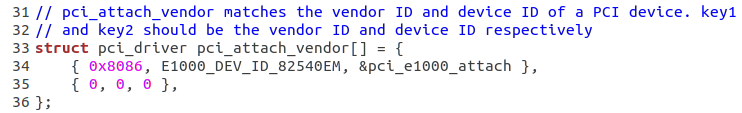
在上表中找到82540EM-A设备的ID是8086:100E，记录一下，可以加入verdor数组中。

首先在e1000.c中加入这个新函数

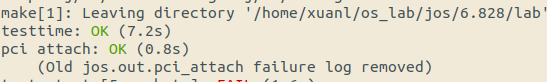


记得在e1000.h中也加上

最后在pic.c里面数组里加上



这样就可以了，上面这个常量是自己在e1000.h里面加的，即0x100e



#### Exercise 4

In your attach function, create a virtual memory mapping for the E1000's BAR 0 by calling mmio\_map\_region (which you wrote in lab 4 to support memory-mapping the LAPIC).

You'll want to record the location of this mapping in a variable so you can later access the registers you just mapped. Take a look at the lapic variable in kern/lapic.c for an example of one way to do this. If you do use a pointer to the device register mapping, be sure to declare it volatile; otherwise, the compiler is allowed to cache values and reorder accesses to this memory.

To test your mapping, try printing out the device status register (section 13.4.2). This is a 4 byte register that starts at byte 8 of the register space. You should get 0x80080783, which indicates a full duplex link is up at 1000 MB/s, among other things.

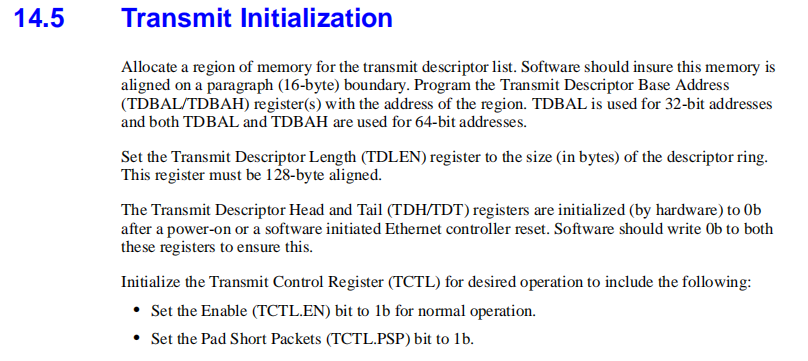
这里需要把一部分内存空间映射为E1000的寄存器地址，这样对这片内存的访问就其实访问的是设备的寄存器。

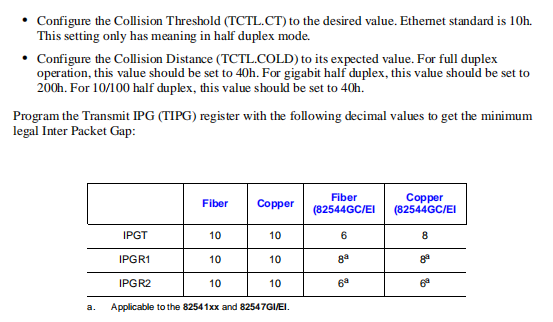
注意pcif->reg\_base存的是物理地址，所以需要使用mmio\_map\_region映射为虚拟地址，所以虚拟地址是这个函数的返回值。



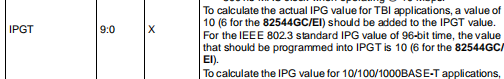
#### Exercise 5

按照手册上初始化



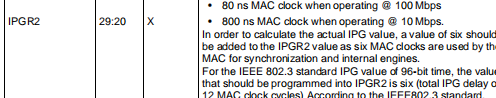


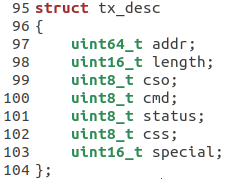
注意TIPG配置不能按照这里来，而要按照13章的来。

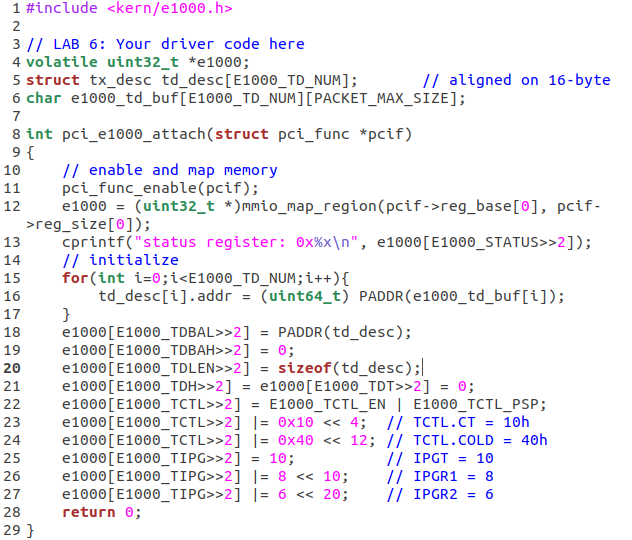


例如这里就讲了802.3标准下IPGT应该是10

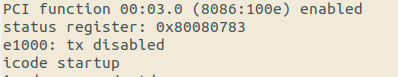
而后面两个分别是8和6



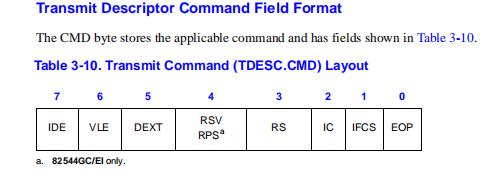




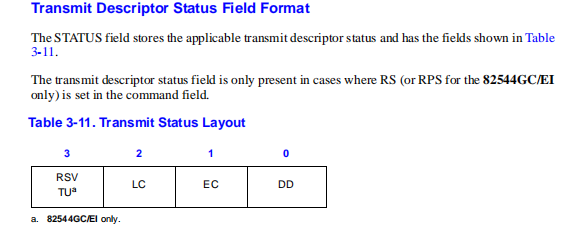
直接在内核静态预留好缓冲区空间，不使用动态分配了。



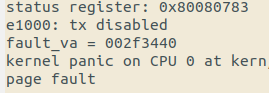
#### Exercise 6

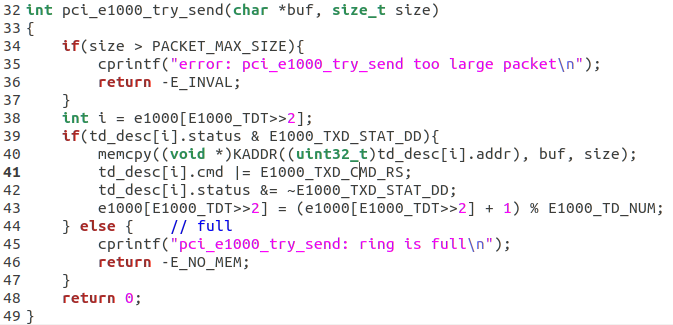


初始化的时候就要把DD位设置好

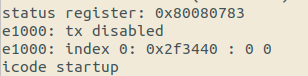


Pagefault，这个是虚拟地址和物理地址的问题。



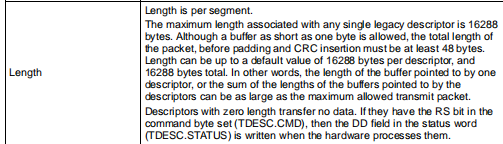


主要就是要注意，在描述符里的addr字段存的是物理地址，所以memcpy的时候要用KADDR转成虚拟地址才行。



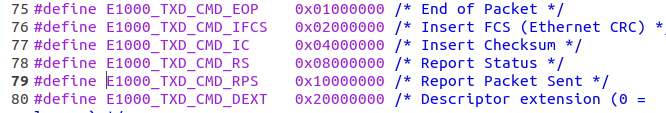
感觉长度为0，这不太对。

要把描述符里的CMD字段和Length都设置好，才能正常传输数据。

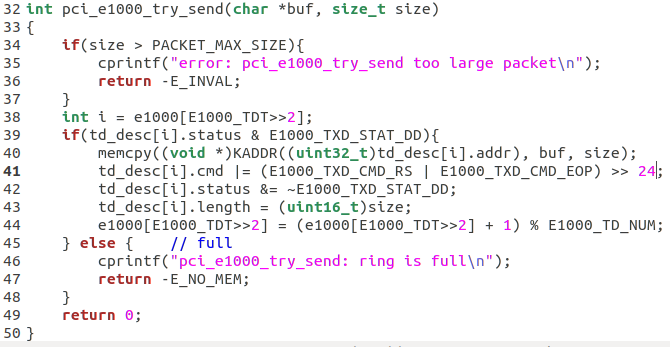


Length字段如果是0，那网卡就以为没有数据，就不会传输了。

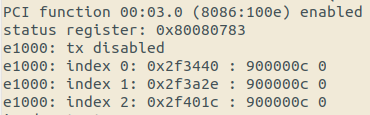
还有问题，注意一下常量

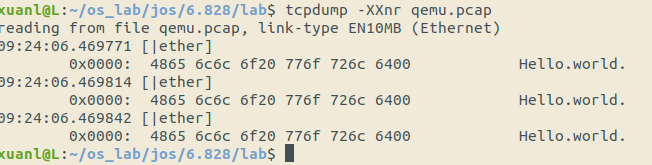


这些常量都是32位的，但是cmd字段是uint8\_t，所以不能直接用这些常量去赋值，会造成高位截断！



稍作修改，改成左移24位之后，就可以了。



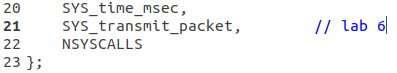


#### Exercise 7

Add a system call that lets you transmit packets from user space. The exact interface is up to you. Don't forget to check any pointers passed to the kernel from user space.

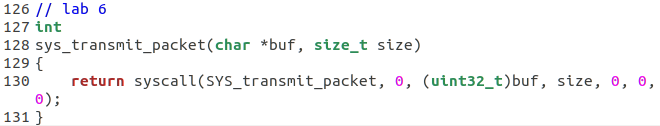
增加一个系统调用，这个还好吧。接口自行定义，那就定成char \*和size\_t好了

首先增加一个常量

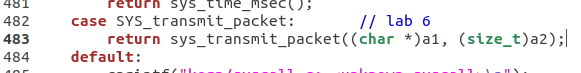


然后要注意kern里的sys函数得是static的，否则会和lib里的产生冲突。

在lib/syscall.c里加入下述函数



Switch-case中加入下面这项



Inc/lib.h里面也要加上一句



最后的实现就是直接调用上一个exercise写好的函数即可

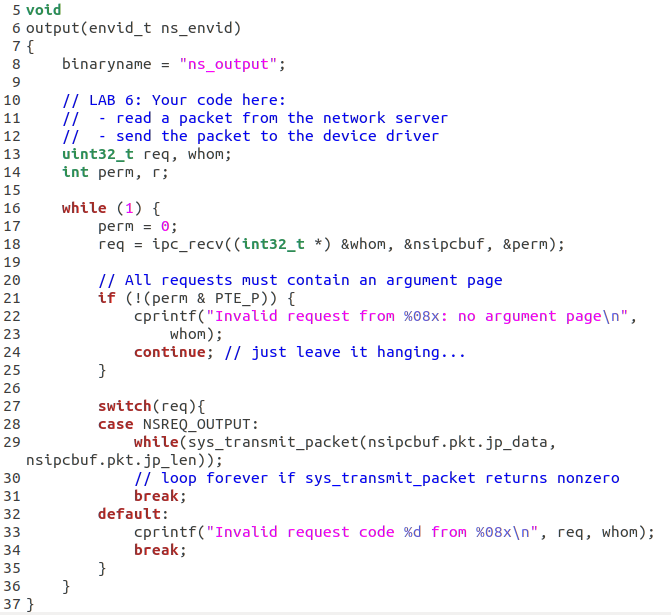


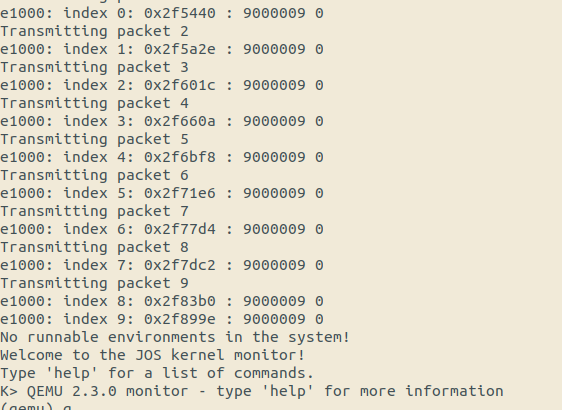
#### Exercise 8

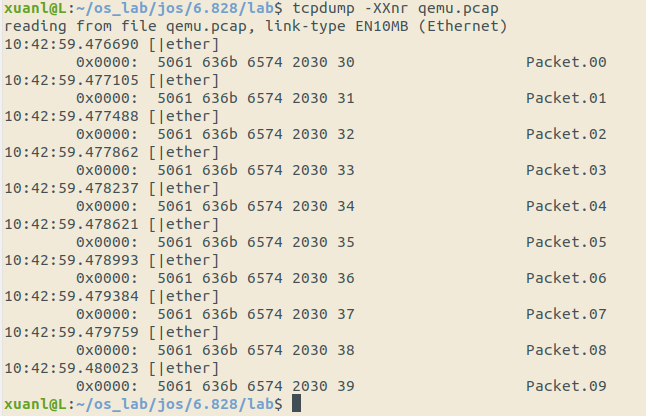
Implement net/output.c.

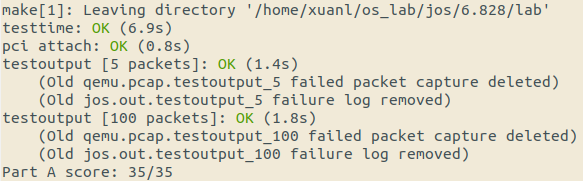
非常简单粗暴的要求。

仿照当时文件系统的写法即可







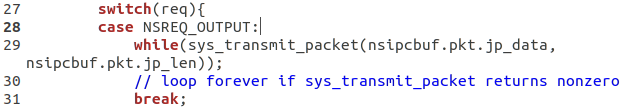


通过了Part A

#### Qusetion 1

How did you structure your transmit implementation? In particular, what do you do if the transmit ring is full?

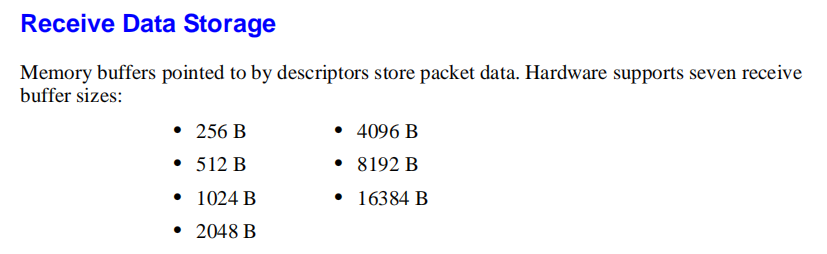
简单来说，就是如果传输失败，就一直尝试传输报文，直到成功。在此期间，output函数被阻塞（陷入死循环），而lwip也因为IPC没有被接收而阻塞。



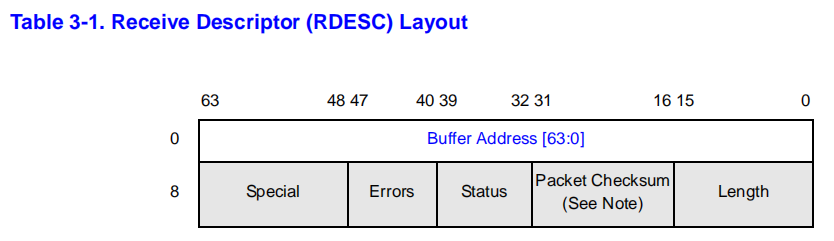
#### Exercise 9

Read section 3.2. You can ignore anything about interrupts and checksum offloading (you can return to these sections if you decide to use these features later), and you don't have to be concerned with the details of thresholds and how the card's internal caches work.

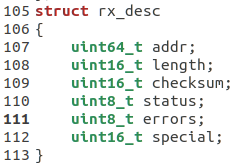
这个练习就是说要阅读一下3.2节，为后续练习作铺垫。

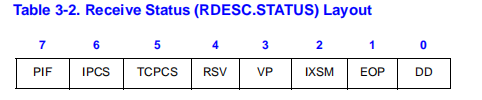


支持7种大小的缓冲区，这里保证支持最大的帧就好了（1518B，所以设置为2048B）



描述符的结构，这同样需要用C语言写出来。





Status里面主要用到的是最后两位（EOP和DD）

对于没有EOP置位的描述符来说，只有addr、DD和length字段是有效的。

#### Exercise 10

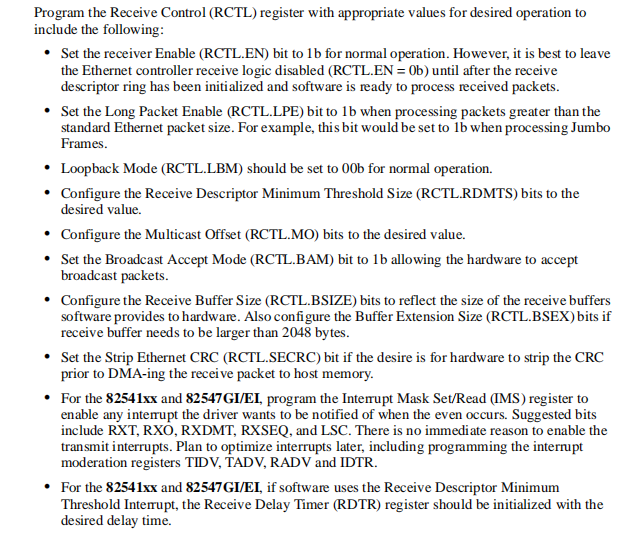
Set up the receive queue and configure the E1000 by following the process in section 14.4. You don't have to support "long packets" or multicast. For now, don't configure the card to use interrupts; you can change that later if you decide to use receive interrupts. Also, configure the E1000 to strip the Ethernet CRC, since the grade script expects it to be stripped.

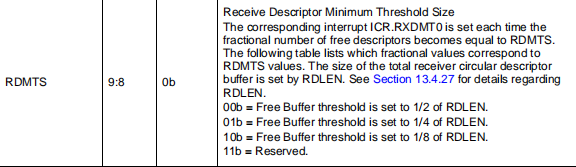
By default, the card will filter out all packets. You have to configure the Receive Address Registers (RAL and RAH) with the card's own MAC address in order to accept packets addressed to that card. You can simply hard-code QEMU's default MAC address of 52:54:00:12:34:56 (we already hard-code this in lwIP, so doing it here too doesn't make things any worse). Be very careful with the byte order; MAC addresses are written from lowest-order byte to highest-order byte, so 52:54:00:12 are the low-order 32 bits of the MAC address and 34:56 are the high-order 16 bits.

The E1000 only supports a specific set of receive buffer sizes (given in the description of RCTL.BSIZE in 13.4.22). If you make your receive packet buffers large enough and disable long packets, you won't have to worry about packets spanning multiple receive buffers. Also, remember that, just like for transmit, the receive queue and the packet buffers must be contiguous in physical memory.

You should use at least 128 receive descriptors

和之前是类似的，照着14.4写初始化函数。

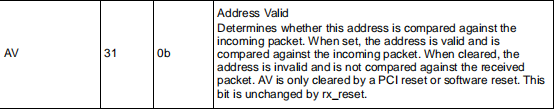




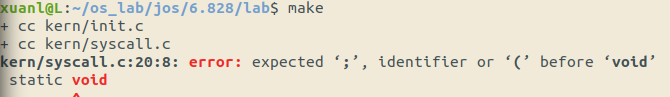
这里RDMTS这个字段，表示接受的数据填充满缓冲区多少的时候，产生一个中断让系统来处理。这两位如果是00b，那就表示填充到一半的时候产生中断。

BSIZE位为0就表示缓冲区大小为2048B，所以就不用管这两位了。

必须置位的是RCTL.SECRC，然后RCTL.BAM也可以置位了，能接受广播。

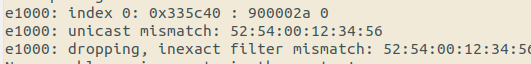


要注意，RAH寄存器的最高位需要置位，这样配置的MAC地址才是有效的。



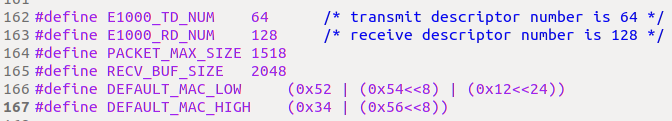
在其它文件里有奇怪的错误，检查后发现是因为inlcude 了e1000.h，而里面的结构体没有封号，这个也算是C语言经常会犯的错误了。

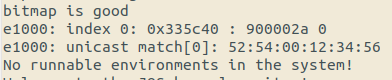
然后运行后，发现没有应该有的信息，原因在于RDH和RDT不能像TDH和TDT那样配置（都是0），而应该配置成不同。即配置成RDH=1，而RDT=0。因为看官网的信息可知，RDH是先写入数据，然后再移动指针，所以RDT和RDH相等的时候就表示满了，所以一开始不能配置成相等的情况。



然后出现了信息，但是unicase mismatch，说明MAC配置有问题。

原因在于，MAC地址是用16进制表示的，和IP地址是不同的！





这样就可以了。

#### Exercise 11

Write a function to receive a packet from the E1000 and expose it to user space by adding a system call. Make sure you handle the receive queue being empty.

暂时不考虑中断的实现，所以如果没有收到消息，也同样陷入死循环。

同样也是在syscall.h和lib.h里面加上相关声明，然后在syscall.c里面加上接口实现。最后在e1000.c里面给出最终的实现。

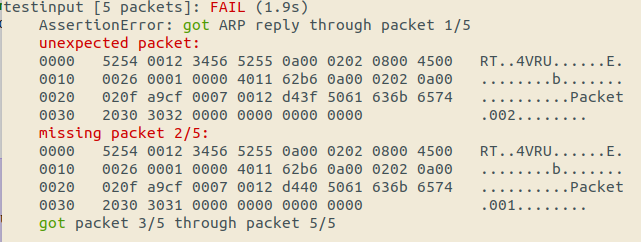
网络环境的进程号是多少呢？难道是input的参数ns\_envid？用output.c再测一下，发现应该就是这个意思。

还是要注意物理地址的问题

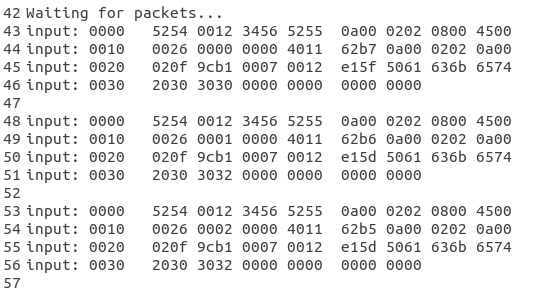
其实这个地方要注意，input函数需要动态分配一个页给nsipcbuf，因为它是IPC的发送方。

然后要注意，发送的IPC的附加页的地址肯定得页对齐。

但是过不了测试。



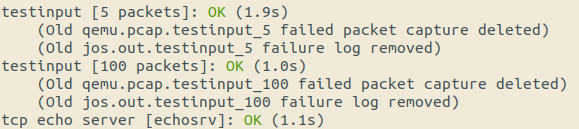
这里直接第一个就是2号包，可能还是原来那个地方，对head和tail的作用没理解。



可以看到，第一个包丢了，而第二个包却重复了。

那这应该是lwip来不及读取，就被下一个包覆盖了，所以要再加一个sys\_yield

这样就能通过测试了。



#### Qusetion 2

How did you structure your receive implementation? In particular, what do you do if the receive queue is empty and a user environment requests the next incoming packet?

其实还是死循环，如果没有收到数据包就阻塞，当然这是比较简单的实现方法。照理应该使用中断来做，比如缓冲区被填满一半的时候触发中断，然后再去处理，这会比较好。

#### Exercise 13

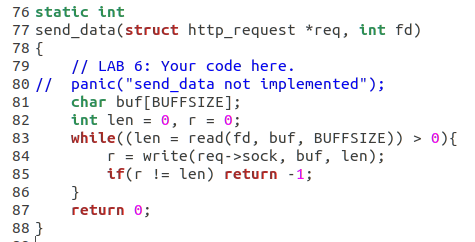
The web server is missing the code that deals with sending the contents of a file back to the client. Finish the web server by implementing send\_file and send\_data.

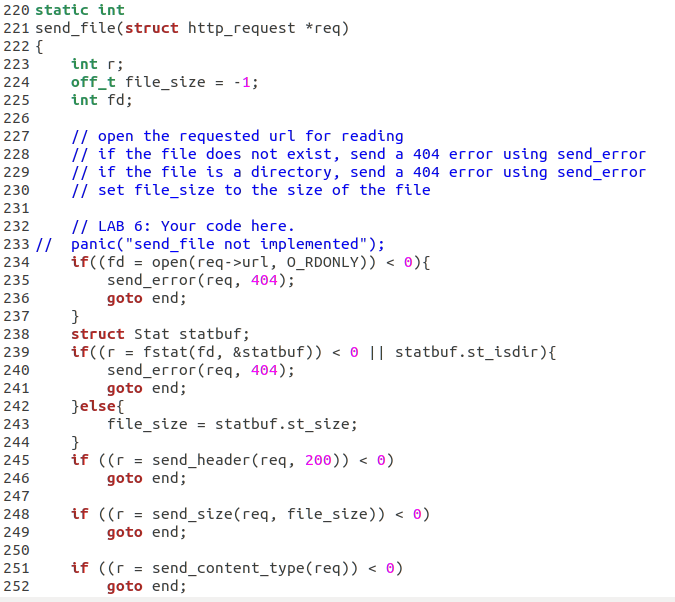
最后一个练习了，希望能顺利通过。

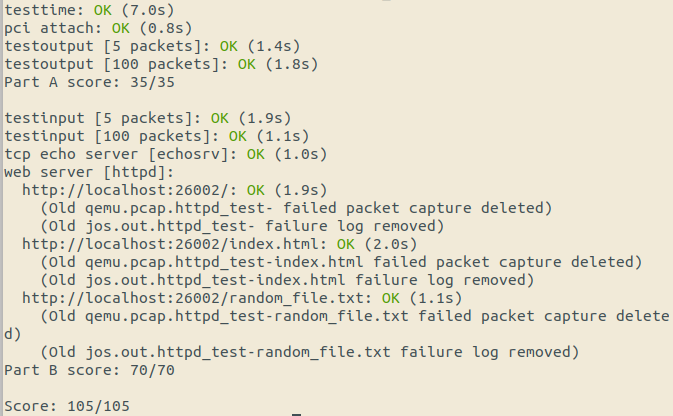
卡在了如何获取文件大小上

查了一下如何获取UNIX系统的文件大小，发现应该使用fstat函数。

这里稍微看一下这个文件里的其他函数，就知道应该用write函数来写socket，然后就比较好实现了。







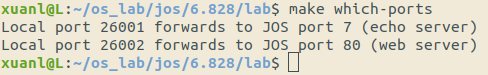
终于完成所有的实验了！

#### Qusetion 3

What does the web page served by JOS's web server say?



要注意使用make which-ports查看端口号



这里出现的26002才是端口号

所以在浏览器输入localhost:26002/index.html

即可看到上述网页

中间一个大标题This file came from JOS

下面有一行字从右向左飘过，Cheesy web page!

#### Qusetion 4

How long approximately did it take you to do this lab?

周日基本上做了一天吧，从上午9点开始，到晚上11点半才通过所有测试。

当然这其实还算快了，我感觉可能是因为熟练了之后就好了。之前的实验我基本都是要几天才能完成的。