# 实验准备

在本实验中你将为网络接口卡（network interface card）写一个xv6设备驱动。

获取本实验xv6源码，并检出net分支：

git fetch、git checkout net、make clean

# 背景

在写代码前，你会发现：重温xv6 book的“第五章：中断和设备驱动”是有帮助的。

你将用一个被称作E1000的网络设备来处理网络通信。对xv6以及你写的驱动来说，E1000就像一个真正的硬件（连接到真实以太本地局域网Ethernet Local Area Net）。事实上，你的驱动将要通讯的E1000是由qemu仿真的，它连接到LAN上也是qemu仿真的。在这个仿真的LAN上，xv6（）有一个IP地址10.0.2.15。qemu也安排一个IP为10.0.2.2、运行qemu的计算机出现在LAN中。当xv6使用E1000发送一个包到10.0.2.2，qemu将包传递到对应计算机（正在运行qemu）应用上。

你将使用qemu的“用户模式网络栈”（user-mode network stack）。qemu的文档有更多有关用户模式栈的细节。<https://qemu-project.gitlab.io/qemu/system/net.html#using-the-user-mode-network-stack>。我们已经更新Makefile来启用qemu的用户模式网络栈，以及E1000网卡。

Makefile配置qemu记录所有输入、输出包到文件packets.cap，在你的实验目录。翻看这些记录，对于确认xv6正在传递、接收你期望的包是有帮助的。展示记录的包：

tcpdump -XXnr packets.pcap

我们已经添加许多文件到这个实验的xv6仓库。文件kernel/e1000.c包含E1000的初始化代码，以及传递、接收包的空函数（你需要填充）。kernel/e1000\_dev.h包含寄存器定义、E1000定义的标志位（在因特尔E1000中描述<https://pdos.csail.mit.edu/6.828/2020/readings/8254x_GBe_SDM.pdf）。kernel/net.c>和kernel/net.h包含一个简单的网络栈（实现了IP/UDP/ARP协议）。这些文件也包含灵活数据结构代码（持有包），称作mbuf。最终，kernel/pci.c包含代码：当xv6启动时，在PCI总线上查找E1000卡。

# 你的工作

你的工作是完成e1000\_transmit()和e1000\_recv()，都在kernel/e1000.c，因此驱动可以传递、接收包。当make grade提示你的方案通过所有测试时，就做对了。

当写代码时，你会参考E1000软件开发手册（[https://pdos.csail.mit.edu/6.828/2020/readings/8254x\_GBe\_SDM.pdf）。](https://pdos.csail.mit.edu/6.828/2020/readings/8254x_GBe_SDM.pdf）。kernel/net.c)下面章节可能会有帮助。

段2是特别重要的，给你整个设备的概览。

段3.2给出一个包接收的概览。

段3.3给出一个包传递的概览。

段13给出E1000使用寄存器的概览。

段14可以帮你理解我们提供的初始代码。

浏览E1000软件开发手册（https://pdos.csail.mit.edu/6.828/2020/readings/8254x\_GBe\_SDM.pdf）。这个手册包含几个与以太网十分相关的控制器。qemu模拟82540EM。浏览第二章了解设备。写驱动，你将需要熟悉第3、14章，以及4.1（并非4.1的子小节）。你将也需要使用章节13作为参考。其他章节包含了你的驱动无需交互的E1000组件。开始不用担心细节；只是了解文档结构，以便于之后可以找到。E1000有一些高级特性，绝大多数可以忽略。仅小部分基础特性需要用来完成本实验。

e1000.c中提供给你的e1000\_init()函数，配置E1000从RAM读取将要被传递的包，写接收到的包到RAM。这个技术称为DMA，直接内存访问，参考事实：E1000硬件直接从RAM写、读包。

因为bursts of packets可能会比驱动处理速度快，e1000\_init()给E1000提供多个buffers，E1000可以把包写到buffers。E1000需要这些buffers被描述为RAM中的一组描述符；每个描述符包含一个RAM中的地址，E1000可以把接收到的包写到其中。struct rx\_desc描述 描述符 格式。描述符数组被称为接收环或接受队列。这是一个环形：卡或驱动到达数组底部时，它转到起始处。e1000\_init()用mbufalloc()为E1000分配mbuf包buffers，用于DMA。也有一个传递环，驱动把需要E1000发送的包放到里面。e1000\_init()配置两个环的尺寸为RX\_RING\_SIZE和TX\_RING\_SIZE。

当net.c中的网络栈需要发送一个包时，调用e1000\_transmit()，用一个mbuf（持有要发送的包）。你的传递代码必须放置一个指针（指向包数据）到TX（transmit）环。struct tx\_desc描述 描述符格式。你将需要确保每个mbuf最终被释放，仅在E1000结束传递包之后（E1000在描述符中设置E1000\_TXD\_STAT\_DD位表示）。

当E1000接收来自以太网的每个包时，它首先DMA包到mbuf（被下个RX ring 描述符指向），然后生成一个中断。你的e1000\_recv()代码必须扫描RX ring，并调用net\_rx()传递每个新包的mbuf到网络栈（in net.c）。你将需要分配一个新mbuf，并放置它到描述符，因此当E1000再次到达RX环中该点时，它就会找到一个新的buffer，DMA一个新的包到该buffer。

除了读写RAM中的描述符环之外，你的驱动将需要与E1000内存映射控制寄存器交互，来检测何时接收到的数据包可用，并通知E1000：驱动已经用要发送的包填充到一些TX描述符。全局变量regs持有一个指针（指向E1000的第一个控制寄存器）；你的驱动可以通过像数组一样索引regs获取其他寄存器。你将需要特别使用索引E1000\_RDT和E1000\_TDT。

为了测试你的驱动，在一个窗口中执行make server，另外一个窗口执行make qemu，然后在xv6执行nettests。nettests的首个测试尝试发送一个UDP包到host操作系统，发送到make server执行的程序。如果你没有完成实验，E1000驱动将不会真正发送包，且什么也不会发生。

当你完成本实验之后，E1000驱动将发送包，qemu将传递它到你的host电脑，make server将看到它，它将发送一个响应包，然后到E1000驱动，然后nettests将看到响应包。在host发送回复前，它发送一个ARP请求包给xv6来找到它的48位以太网地址，期望xv6对ARP做出响应。一旦你已经完成E1000驱动的工作，kernel/net.c将处理这。如果一切正常，nettests将打印testing ping: OK，make server将打印 a message from xv6！。

tcpdump -XXnr packets.pcap应该产生起始如下的输出：

reading from file packets.pcap, link-type EN10MB (Ethernet)

15:27:40.861988 IP 10.0.2.15.2000 > 10.0.2.2.25603: UDP, length 19

0x0000: ffff ffff ffff 5254 0012 3456 0800 4500 ......RT..4V..E.

0x0010: 002f 0000 0000 6411 3eae 0a00 020f 0a00 ./....d.>.......

0x0020: 0202 07d0 6403 001b 0000 6120 6d65 7373 ....d.....a.mess

0x0030: 6167 6520 6672 6f6d 2078 7636 21 age.from.xv6!

15:27:40.862370 ARP, Request who-has 10.0.2.15 tell 10.0.2.2, length 28

0x0000: ffff ffff ffff 5255 0a00 0202 0806 0001 ......RU........

0x0010: 0800 0604 0001 5255 0a00 0202 0a00 0202 ......RU........

0x0020: 0000 0000 0000 0a00 020f ..........

15:27:40.862844 ARP, Reply 10.0.2.15 is-at 52:54:00:12:34:56, length 28

0x0000: ffff ffff ffff 5254 0012 3456 0806 0001 ......RT..4V....

0x0010: 0800 0604 0002 5254 0012 3456 0a00 020f ......RT..4V....

0x0020: 5255 0a00 0202 0a00 0202 RU........

15:27:40.863036 IP 10.0.2.2.25603 > 10.0.2.15.2000: UDP, length 17

0x0000: 5254 0012 3456 5255 0a00 0202 0800 4500 RT..4VRU......E.

0x0010: 002d 0000 0000 4011 62b0 0a00 0202 0a00 .-....@.b.......

0x0020: 020f 6403 07d0 0019 3406 7468 6973 2069 ..d.....4.this.i

0x0030: 7320 7468 6520 686f 7374 21 s.the.host!

你的输出将看起来有点不同，但它应该包含字符串“ARP，Request”，“ARP，Reply”，“UDP”，“a.message.from.xv6”和“this.is.the.host”。

nettests执行一些其他测试，最终是一个DNS请求（通过真正的互联网发送到谷歌的一个名称服务器）。你应该确认你的代码通过所有这些测试，之后你将看到这个输出：

$ nettests

nettests running on port 25603

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.

你该确保make grade让你的方案通过。

# 提示

开始时，添加打印语句到e1000\_transmit()和e1000\_recv()，执行make server和nettests。你该从你的打印语句中看到：nettests生成一个对e1000\_transmit的调用。

一些实现e1000\_transmit的提示：

1. 首先向E1000索要TX环索引，在该索引上期待下个包，通过读E1000\_TDT控制寄存器。
2. 检测环是否溢出。如果E1000\_TXD\_STAT\_DD没有设置在由E1000\_TDT索引的描述符上，那么E1000没有结束之前的传递请求，因此返回一个错误。
3. 另外，使用mbuffree()来释放最后的mbuf（从那个描述符被传递）。
4. 填充描述符。m->head指向内存中包的内容，m->len为包的长度。设置必要的cmd标志位（看E1000手册中的section 3.3），暂存一个指针指向mbuf为了之后释放。
5. 最终，通过加1到E1000\_TDT，对TX\_RING\_SIZE取模运算，更新环位置。
6. 如果e1000\_transmit()成功地添加mbuf到环，返回0。失败（没有描述符可用于传递mbuf），返回-1，因此调用者知道释放该mbuf。

一些实现e1000\_recv的提示：

1. 首先向E1000索要环索引，下个等待接收的包位于该索引，通过获取E1000\_RDT控制寄存器的值，加1对RX\_RING\_SIZE取模。
2. 通过检查描述符中status部分的E1000\_RXD\_STAT\_DD位，检查是否新包可用。如果不可用，停止。
3. 更新mbuf的m->len到描述符中的length。使用net\_rx()传递mbuf到网络栈。
4. 使用mbufalloc()分配一个新mbuf来取代刚刚分配过的，分配给net\_rx()。让它的数据指针（m->head）放到描述符。清除描述符的状态位为0。
5. 最终，更新E1000\_RDT寄存器为环描述符最后处理的索引。
6. e1000\_init()用mbufs初始化RX环，你将需要了解他是如何做到这一点的，也许还需要借用代码。
7. 在一些点，已到达的包总数将超出环尺寸（16）；确保你的代码可用处理。

你将需要锁来处理这种可能：xv6可能不止一个进程使用E1000，也可能是：中断发生时正在内核线程中使用E1000。