仓库链接： https://github.com/zhaoyihui233/mit-6.S081-lab-2021/tree/main/lab7%20Networking

# 1 Uthread: switching between threads

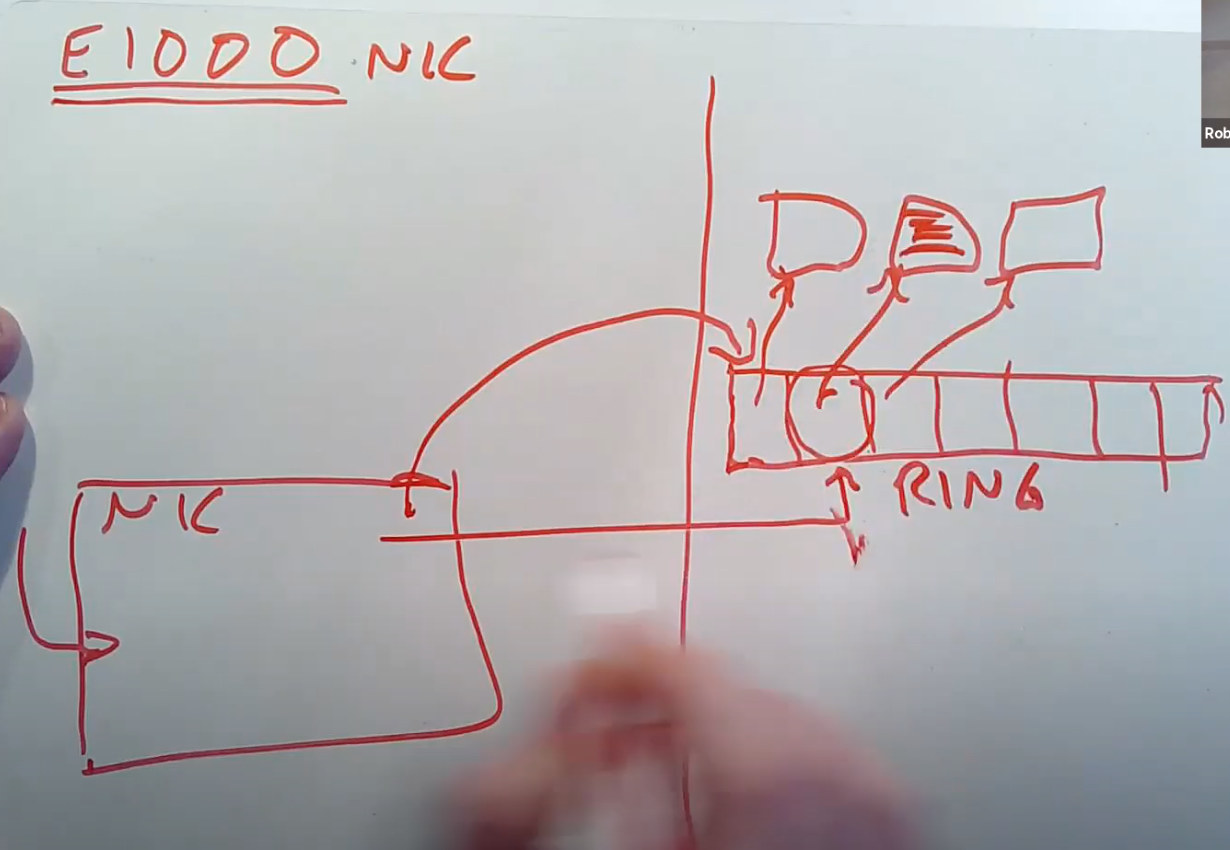
## 1.1实验内容

在本实验中，我们将使用一个叫做E1000的网络设备来处理网络通信。对于xv6（以及我们写的驱动），E1000看起来像一个真的连接到以太网LAN的硬件设备。事实上， E1000是被qemu模拟出来的，与LAN连接也是由qemu模拟的。在这个模拟的LAN中，xv6（访客）有一个IP地址10.0.2.15，qemu也给运行qemu的计算机提供了一个IP地址 10.0.2.2。当xv6用E1000发送一个数据包给 10.0.2.2，qemu将会把这个数据包传送到对应的应用。

Makefile文件设置了qemu把所有的通讯数据包记录到到packets.pcap文件里。我们可以通过tcpdump -XXnr packets.pcap展示这些数据包的内容。

我们的工作是实现e1000\_transmit()和e1000\_revc()（它们都在kernel/e1000.c中），使得E1000能够传输和接收数据包。

我们需要理解E1000的工作流程。代码中有两个数组：rx\_ring和tx\_ring。在代码中它们长度都为16，它们的元素是内存某处的地址。在E1000接收包时，会根据内部一个指向rx\_ring元素的指针，将包直接拷贝到内存里。下图是接收包的示意图，其中的“RING”表示的就是rx\_ring。网卡（NIC）内部有指向rx\_ring的指针，每接收到一个包，指针都会后移一位，如果移到最后一个元素无法后移，则会回到数组的第一个元素，这也是rx\_ring被称为ring的原因。

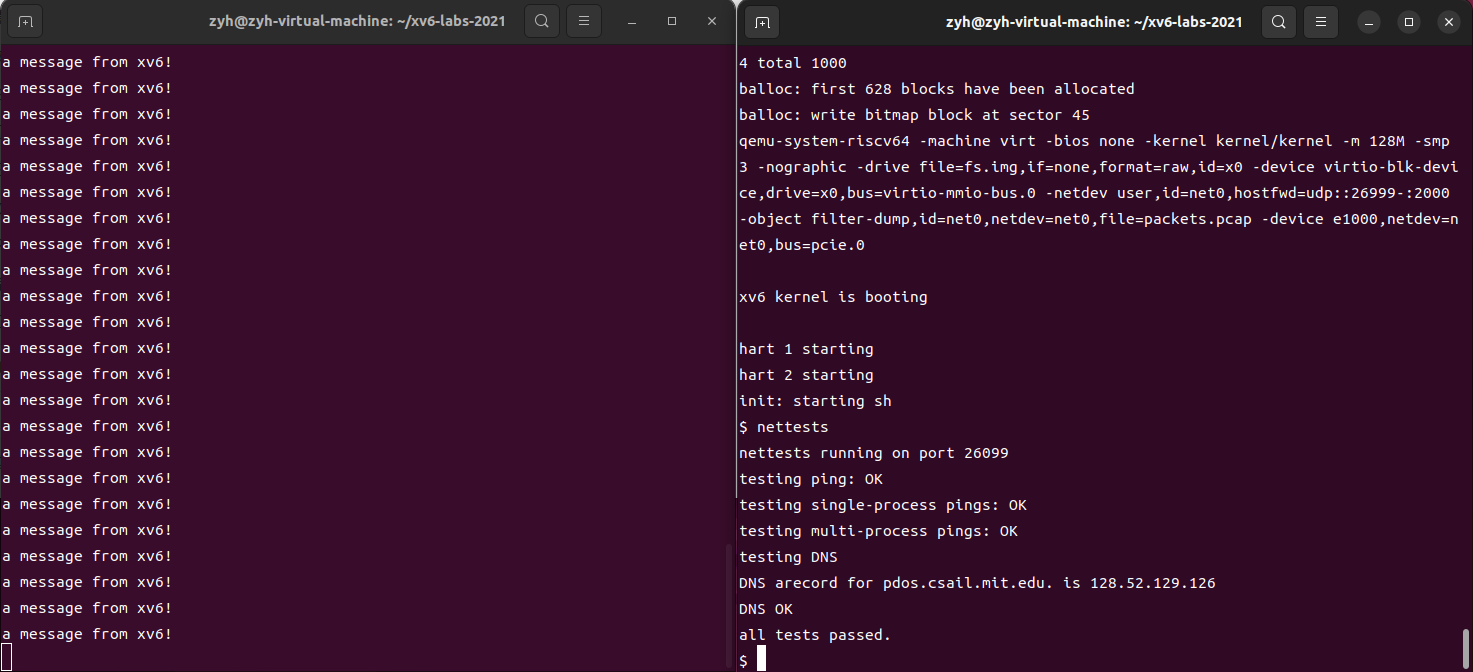


发送包的机制也是类似的。

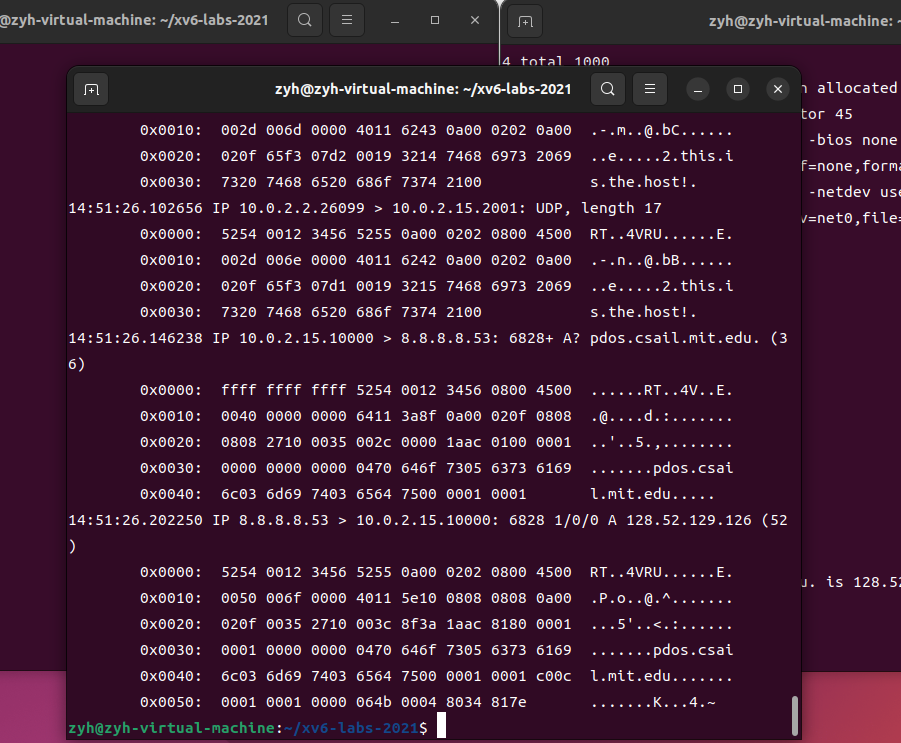
## 1.2代码位置与截图

## 1.3实验结果

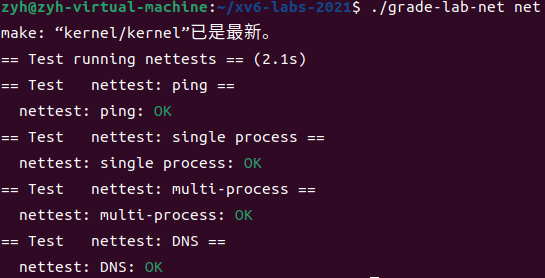
在一个终端里make server，然后在另一个终端里make qemu和nettests：



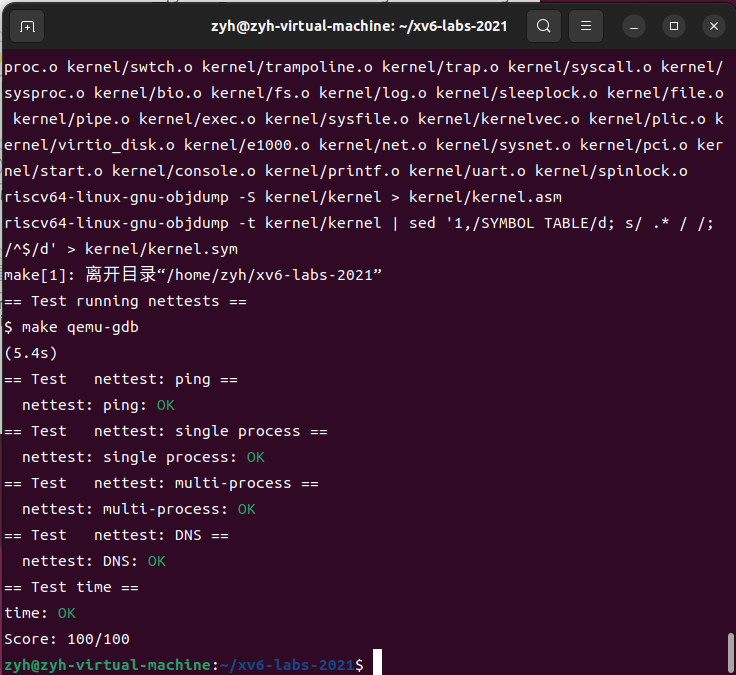
tcpdump -XXnr packets.pcap



./grade-lab-net net



make grade:



## 1.4代码实现

### 1.4.1 e1000\_transmit函数实现思路

提示已经基本点明了代码框架。

在发送包时，我们需要把要发送的包存入内存的某处，代码中表示为tx\_mbuf，同时相应的指向该处的tx\_ring指针也要更新。于是，e1000\_transmit函数的实现思路如下：

1、首先通过发送尾指针对应的寄存器regs[E1000\_TDT]获取到可以写入的位置, 也就是下面我们要放入发送包的发送队列tx\_ring的索引。

2、检查尾指针指向的描述符是否在状态中写入了 E1000\_TXD\_STAT\_DD标志位，若该标志位为1说明发送队列溢出了，这时需要报错。

3、检查尾指针执行的描述符对应数据帧的缓冲区是否被释放，若未被释放则调用mbuffree函数进行释放。

4、更新描述符，根据提示更新addr字段、length字段和cmd字段。addr 字段指向缓冲区地址m->head，length 字段记录包的长度m->len。更新尾指针指向的描述符的 cmd 字段，需要对E1000\_TXD\_CMD\_EOP和E1000\_TXD\_CMD\_RS两个标志位进行置位。EOF 标志位表示数据包的结束，因此需要被设置。RS标志位用于报告状态信息，只有设置了该字段，描述符的 status 字段才是有效的, 而网卡将数据包发送完成后会设置 DD 状态, 因此此处也需要对描述符设置 RS 标志位.

5、将包的缓冲数据m 记录到缓冲区队列mbuf中用于之后的释放。这里是为步骤 3 做准备的。数据帧的缓冲区 m 在因为包此时还未被网卡发送还不能被释放，因此需要将其记录到描述符对应的缓冲区队列中，当后续尾指针又指向该位置时再将其释放。

6、更新发送队列的尾指针。

需要注意的时，根据提示，由于e1000\_transmit函数上层由 sockwrite函数调用，可能有多个进程同时调用该函数，因此需要保证发送队列的指针的并发安全。考虑到e1000\_init函数中初始化的e1000\_lock互斥量，我们可以在e1000\_transmit函数里加上锁。

### 1.4.2 e1000\_recv函数实现思路

1、首先通过regs[E1000\_RDT]+1)%RX\_RING\_SIZE获取到软件可以读取的rx\_ring位置，也就是接收未被软件处理的第一个数据帧在接收队列rx\_ring的索引，该位置即软件需要解封装的数据帧的描述符。

2、与发送数据类似，需要检查数据帧状态的 E1000\_RXD\_STAT\_DD 标志位，以确定当前的数据帧已被网卡硬件处理完毕，否则则停止。

3、接收缓冲区待处理的数据帧，并更新描述符的信息。然后调用 net\_rx() 传递给网络栈进行解封装。

4、上面就算是已经接收了包，下面则需要为后续获取包做设置。调用 mbufalloc分配一个新的接收缓冲区替代原来的接收队列的缓冲区，并更新描述符的 addr 字段，指向新的缓冲区。

5、在实验指导中指出到达的数据包可能超过接收队列大小，需要针对此进行处理。这里是通过在 e1000\_recv添加循环，从而在一次中断触发后可以一直将可解封装的数据传递到网络栈，以避免队列中的可处理数据帧的堆积。

而关于锁的问题，与发送数据不同，接收数据时该函数只会被中断处理函数 e1000\_intr调用, 因此不会出现并发的情况；此外, 网卡的接收和发送的数据结构是独立的, 没有共享，因此无需加锁。