Lab7：Networking

该实验由Your Job组成。

1. Your Job

实验目的

编写一个在xv6操作系统中用于网络接口卡（network interface card, NIC）的设备驱动程序。通过这个实验，你将学习如何初始化并操作一个虚拟的网络设备，以及如何处理网络通信，从而深入理解操作系统中设备驱动程序的工作原理。

实验步骤

1. 在kernel/e1000.c中实现e1000\_teansmit()函数，它是一个网络驱动程序中用于将数据包发送（transmit）的函数。

将以 struct mbuf \*m 传递的数据包发送到 e1000 网络设备。它会将数据包的内容写入一个空闲的发送描述符，然后将描述符指针后移，以便进行下一个数据包的发送。函数会在发送之前进行一些检查，确保发送队列不会溢出，并在发送完成后释放对应的 mbuf 内存。

其中 regs 是 e1000 网络设备的寄存器地址，tx\_ring 是用于发送的描述符环，tx\_mbufs 是指向 mbuf 的指针数组，acquire 和 release 是用于获取和释放锁的函数。

int e1000\_transmit(struct mbuf \*m)

{

  acquire(&e1000\_lock);

  // 获取 e1000\_lock 锁，这个锁用于同步对 e1000 网络设备的访问，以防止并发访问带来的问题。

  uint32 next\_index = regs[E1000\_TDT];

  // 从寄存器中读取当前可用的 TX（发送）描述符环的下一个索引。

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

  {

    // 检查当前下一个描述符的状态是否为 "E1000\_TXD\_STAT\_DD"（表示描述符是否可用）。

    // 如果不可用，则说明发送队列已满，无法继续发送新的数据包，所以需要释放锁并返回-1，表示发送失败。

    release(&e1000\_lock);

    return -1;

  }

  if (tx\_mbufs[next\_index])

    mbuffree(tx\_mbufs[next\_index]);

  // 检查下一个描述符的 mbuf 指针是否非空，如果非空，则释放之前可能存储在该描述符中的 mbuf。

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

  tx\_ring[next\_index].length = (uint16)m->len;

  // 将 mbuf 中的数据包内容的头部地址和长度存储到下一个描述符中，以便将数据包发送。

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

  // 设置下一个描述符的命令字段。其中 EOP 表示该描述符为一个完整数据包的结束描述符，

  // RS 表示发送时将报告状态（Report Status），即在数据包发送完成后触发中断。

  tx\_mbufs[next\_index] = m;

  // 将当前 mbuf 存储到下一个描述符对应的缓冲区中，以便在发送完成后能够释放 mbuf。

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

  // 更新寄存器中的 TDT（Transmit Descriptor Tail）指针，使其指向下一个可用的描述符。

  // 这样做后，该描述符就准备好发送数据包了。

  release(&e1000\_lock);

  // 释放 e1000\_lock 锁，允许其他线程再次访问 e1000 网络设备。

  return 0;

  // 返回0表示数据包发送成功。

}

1. 实现e1000\_transmit()函数。它是一个网络驱动程序中用于接收数据包的函数。

从 e1000 网络设备接收数据包。它会循环检查接收描述符环中的每个描述符，如果发现有新的数据包到达（E1000\_RXD\_STAT\_DD 状态为真），则会将数据包的内容读取到 mbuf 中，并将 mbuf 传递给网络栈的处理函数（net\_rx()）。然后，该函数会为下一个接收到的数据包准备新的接收描述符，并更新 RDT 指针，以便 e1000 网络设备知道从哪里开始接收新的数据包。

在处理接收到的数据包时，函数还进行了一些安全性检查，例如检查数据包长度是否超过 mbuf 的最大大小，如果超过则触发 panic。同时，函数会将接收描述符的状态字段清零，以便 e1000 网络设备知道该描述符已被处理完毕，可以用于接收新的数据包。

static void e1000\_recv(void)

{

  uint32 next\_index = (regs[E1000\_RDT] + 1) % RX\_RING\_SIZE;

  // 从寄存器中读取当前可用的 RX（接收）描述符环的下一个索引。

  while (rx\_ring[next\_index].status & E1000\_RXD\_STAT\_DD)

  {

    // 循环检查下一个描述符的状态是否为 "E1000\_RXD\_STAT\_DD"（表示描述符中有新的数据包到达）。

    if (rx\_ring[next\_index].length > MBUF\_SIZE)

    {

      panic("MBUF\_SIZE OVERFLOW!");

    }

    // 检查数据包的长度是否超过了 mbuf 的最大大小（MBUF\_SIZE）。如果超过了，就触发 panic（内核恐慌）。

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

    // 将接收到的数据包长度存储到对应的 mbuf 中

    net\_rx(rx\_mbufs[next\_index]);

    // 调用 net\_rx() 函数，将 mbuf 传递给网络栈处理，以进行后续的数据包处理和解析

    rx\_mbufs[next\_index] = mbufalloc(0);

    // 分配一个新的 mbuf，并将其存储到接收描述符的缓冲区中，以准备接收下一个数据包。

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

    // 将新的 mbuf 的头部地址存储到接收描述符中，以便接收数据包时能够正确写入数据。

    rx\_ring[next\_index].status = 0;

    // 将接收描述符的状态字段清零，表示该描述符已被处理完毕，可以用于接收新的数据包。

    next\_index = (next\_index + 1) % RX\_RING\_SIZE;

    // 更新下一个可用接收描述符的索引，准备处理下一个接收到的数据包。

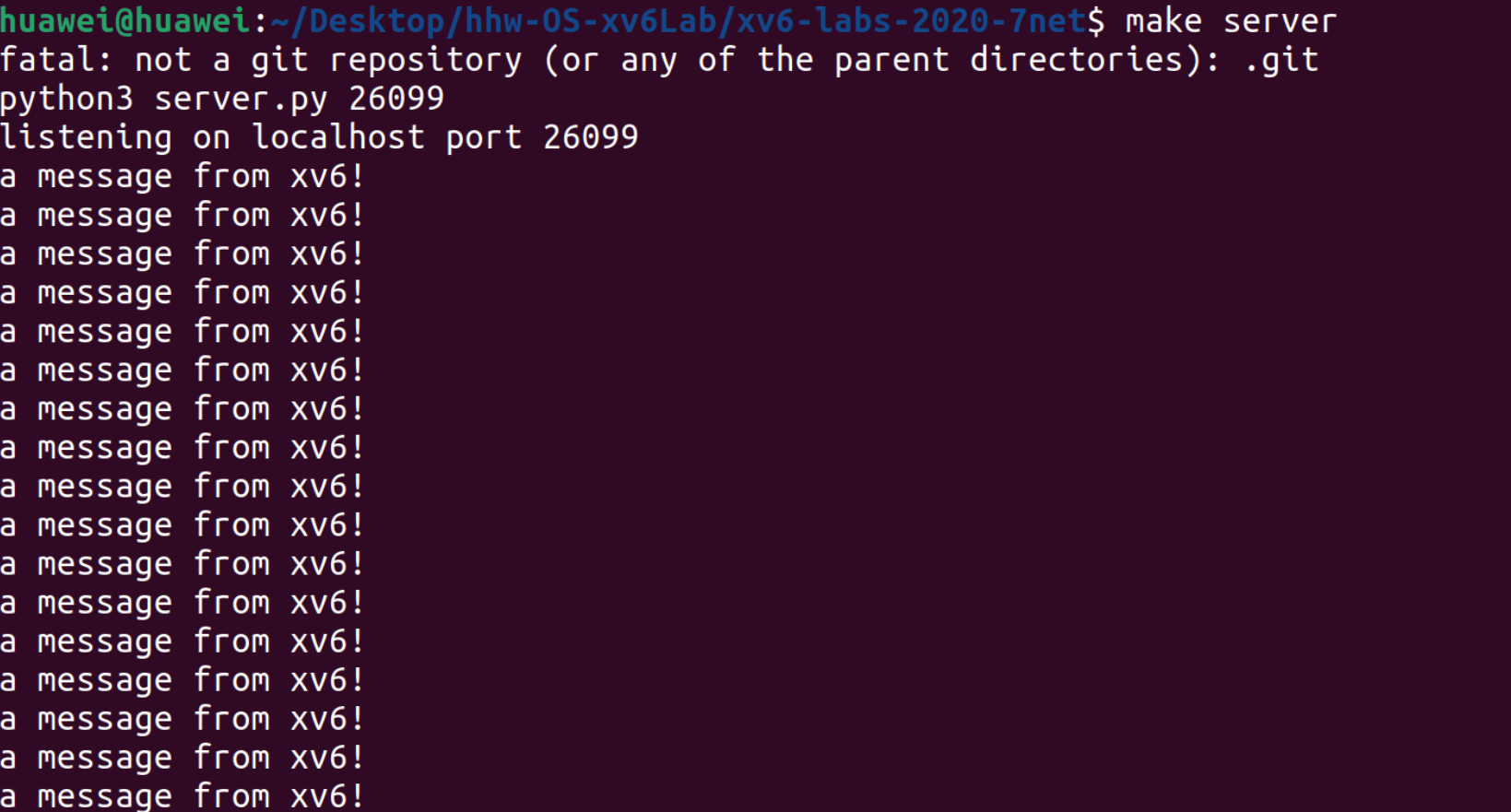
  }

  regs[E1000\_RDT] = (next\_index - 1) % RX\_RING\_SIZE;

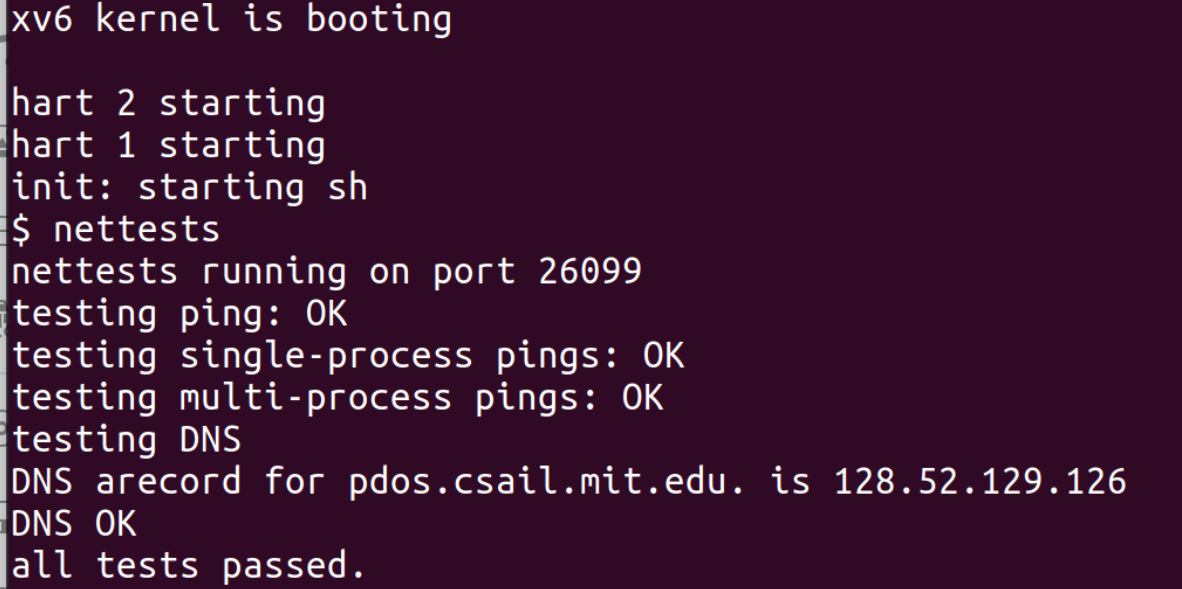
  // 更新寄存器中的 RDT（Receive Descriptor Tail）指针，使其指向最后一个处理过的接收描述符。

}

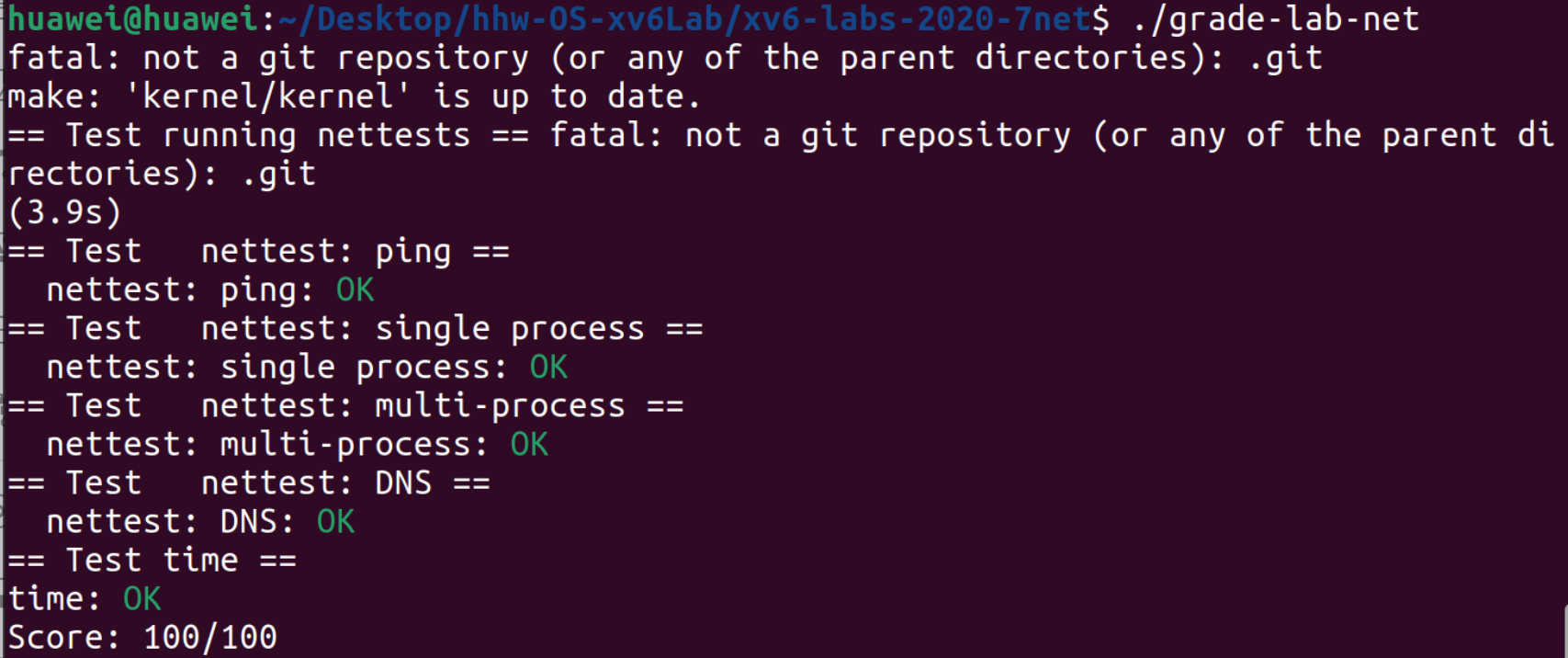
3.打开一个终端窗口执行make server



4.打开另一个窗口利用make qemu指令运行xv6，运行nettests以测试数据包的发送和接收功能：



5. 测试程序，退出xv6，在终端输入./grade-lab-net：



遇到的问题与心得

遇到的问题：

e1000\_recv的实现过程中有一句：重新分配一个新的mbuf，我理解出错，去重新创建了一个新的mbuf，根本上是因为我没理解到rx\_ring是被循环使用的，这个循环数组一直是用来写入，传递packet的

实验心得

本次实验看起来文档、字都很多，但实际的代码并不难，本次的提示就能作为伪代码，完全可以按着来写。难点应该在于理解整个收发包（recevice/transmit packet）的过程，cpu、网卡（ethernet）、RAM（buffer的存放处）这三者是如何通过xv6操作系统进行交互的。xv6得依靠network stack（网络栈）实现收发数据，即通过network stack收发packet。