Lab: networking

Git链接：[Jeery1/xv6-6.S081 at net](https://github.com/Jeery1/xv6-6.S081/tree/net)

**实验目的：**xv6 操作系统的 Networking 实验的主要目的是让学生理解和实现一个简单的网络协议栈，特别是以太网(Ethernet)、互联网协议(IP)和用户数据报协议(UDP)的基本功能。通过这个实验，学生将：

理解网络协议栈的基本架构：学习网络协议的分层结构，包括链路层、网络层和传输层。

实现网络设备驱动：编写代码与模拟的以太网卡(E1000)交互，处理数据包的接收和发送。

实现网络协议处理：

以太网帧(Ethernet frame)的处理

IP数据包的处理(包括校验和计算)

UDP数据包的处理

理解数据包传输流程：从应用程序通过协议栈向下到网卡，以及从网卡通过协议栈向上到应用程序的完整路径。

实现简单的网络功能：如通过UDP协议发送和接收数据包。

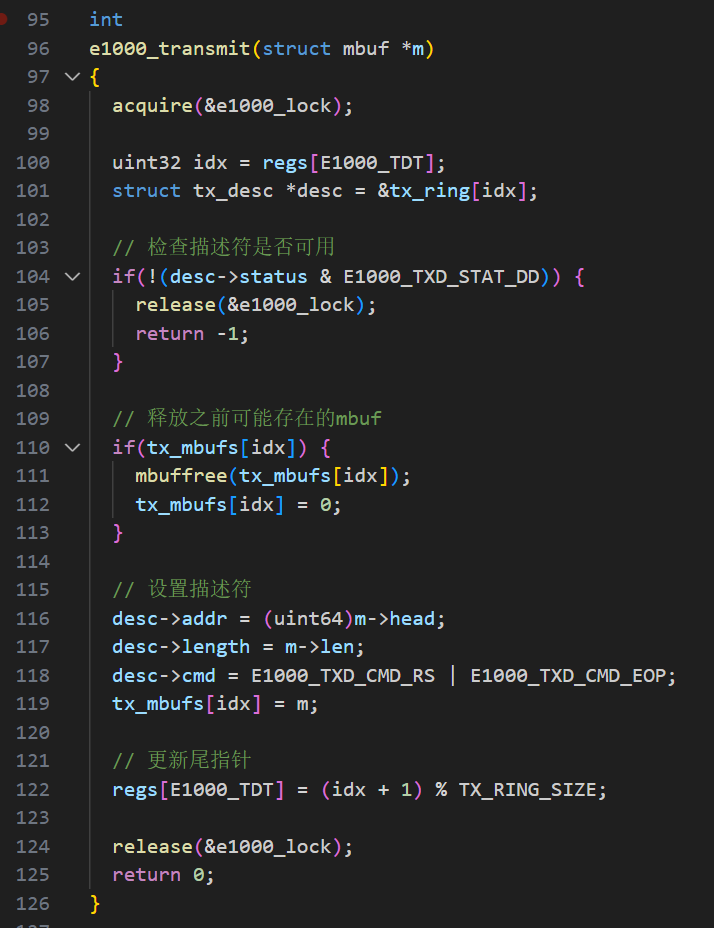
学习缓冲区管理：如何处理网络数据包的缓冲和内存管理。

理解中断处理：网络设备如何使用中断来通知操作系统有新数据包到达。

这个实验通常要求学生扩展xv6内核，添加网络支持，最终能够通过网络与其他实例通信。通过实践，学生可以深入理解操作系统如何与网络硬件交互，以及网络协议栈的实现原理。

**实验步骤：**

E1000网卡使用DMA环形缓冲区来传输数据包，分为发送(TX)和接收(RX)两个队列。我们需要实现e1000\_transmit()和e1000\_recv()两个函数。



e1000\_transmit()函数负责将数据包通过 E1000 网卡发送出去。

该函数接收一个 mbuf 结构体指针，表示要发送的网络数据包。首先立即获取 e1000\_lock 自旋锁，保护对共享资源（发送队列）的访问。然后获取当前发送位置，检查描述符可用性，如果当前描述符的 DD 位未被设置，说明之前的发送操作还未完成，没有可用的描述符，释放锁并返回 -1 表示发送失败。如果描述符可用，则释放之前可能存在的 mbuf避免内存泄漏。然后设置新的发送描述符，addr：设置数据包在内存中的物理地址（mbuf 的 head 指针）

length：设置数据包长度

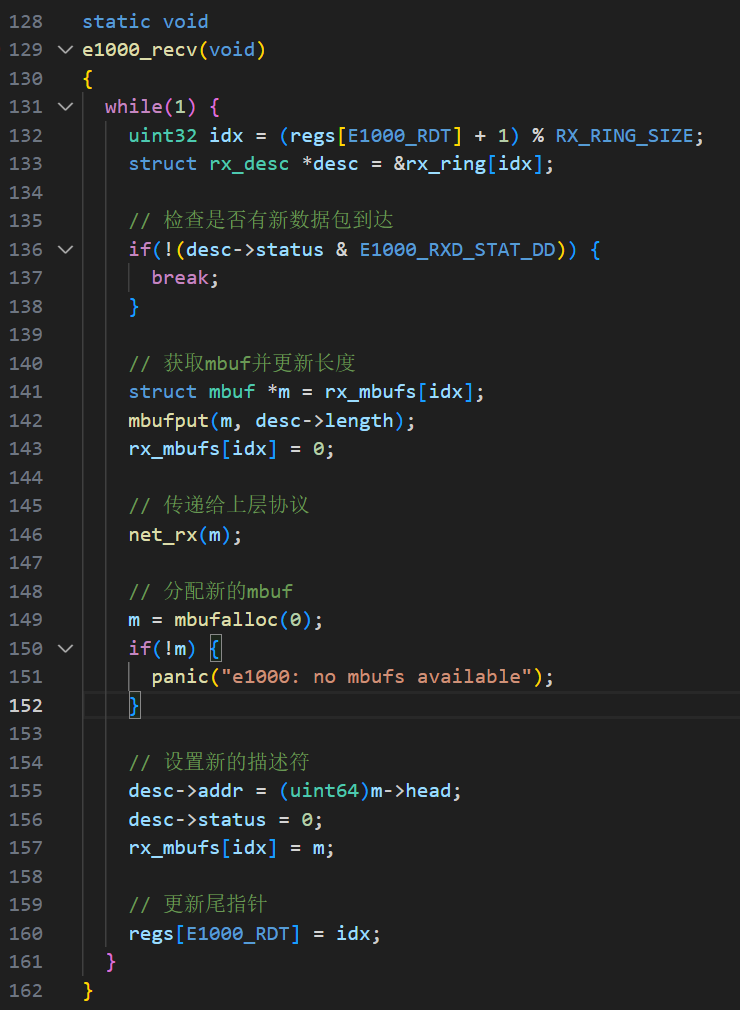
cmd：设置命令标志：

E1000\_TXD\_CMD\_RS：报告状态（Report Status）

E1000\_TXD\_CMD\_EOP：数据包结束（End Of Packet）

tx\_mbufs：保存 mbuf 指针供后续释放

然后，更新发送队列尾指针，通知硬件有新的数据包待发送，最后释放锁并返回。



e1000\_recv()函数负责从 E1000 网卡接收数据包。

while(1)：无限循环，持续处理接收到的数据包，首先获取下一个接收描述符索引。然后检查是否有新数据包到达，E1000\_RXD\_STAT\_DD是描述符完成位，如果 DD 位未设置，表示没有新数据包，退出处理循环。否则处理接收到的数据包，更新 mbuf 的数据长度，将描述符位置的 mbuf 指针置空。然后将数据包传递给上层协议栈，为描述符分配新的 mbuf，设置新的接收描述符。最后更新接收尾指针。

之后在kernel/sysnet.c中实现套接字相关功能。



这是处理接收到的 UDP 数据包的核心函数。

参数：

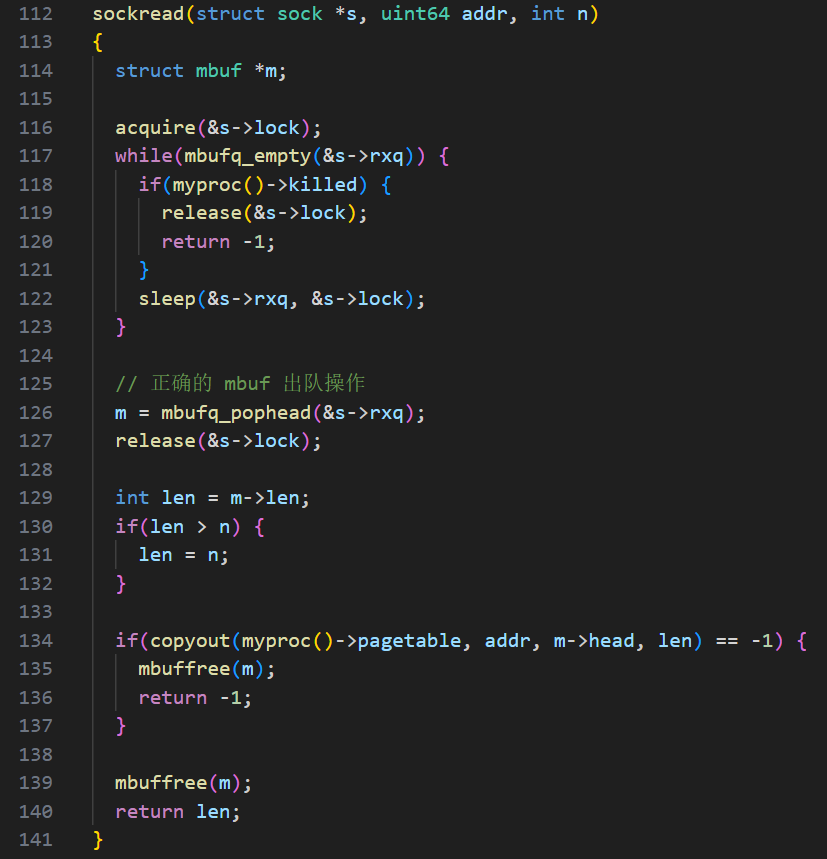
m: 指向接收到的网络数据包的 mbuf (内存缓冲区)指针

raddr: 远程IP地址(源地址)

lport: 本地端口号(目的端口)

rport: 远程端口号(源端口)

首先获取锁，然后遍历系统中所有的套接字，寻找匹配的UDP套接字。检查套接字是否匹配以下条件：远程IP地址(raddr)匹配、本地端口(lport)匹配、远程端口(rport)匹配，这实现了UDP连接的四元组匹配(本地IP+端口，远程IP+端口)，然后将接收到的数据包(mbuf)放入匹配套接字的接收队列(rxq)的尾部，唤醒可能在该套接字接收队列上休眠的进程，最后释放锁，完成临界区。



这是实现从套接字读取数据的核心函数。

参数：

s: 指向套接字结构的指针

addr: 用户空间目标缓冲区的地址

n: 请求读取的字节数

首先获取套接字锁，然后持续检查接收队列是否为空。在等待过程中检查当前进程是否被标记为终止，如果是，释放锁并返回错误。让当前进程休眠，等待接收队列有数据。等待结束后，mbufq\_pophead从接收队列(rxq)头部取出一个数据包(mbuf)，取出后立即释放套接字锁，减少锁持有时间，采用FIFO(先进先出)顺序处理数据包。然后获取数据包实际长度(m->len)，如果数据包长度大于用户请求的长度(n)，只拷贝请求的长度，确保不会拷贝超过用户缓冲区大小的数据。最后使用copyout将内核数据(m->head)拷贝到用户空间(addr)



这个函数实现了向套接字写入数据的功能 ，参数：

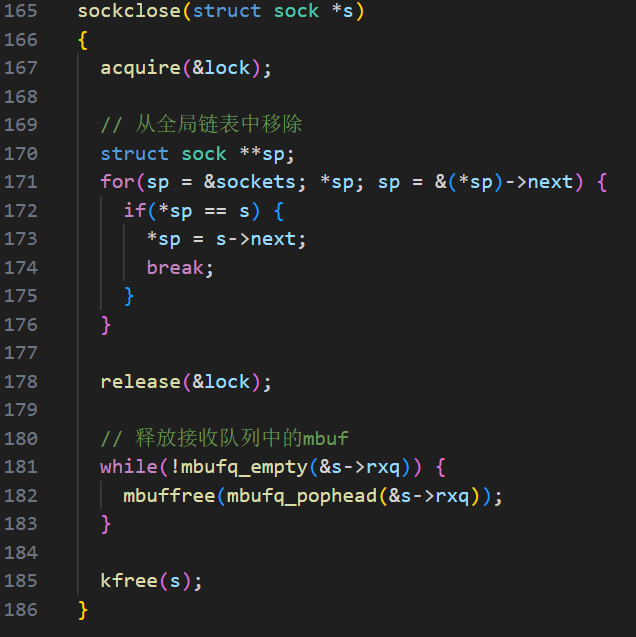
s: 指向套接字结构的指针

addr: 用户空间源缓冲区的地址

n: 要写入的字节数

返回值：成功写入的字节数或错误码(-1)

首先使用mbufalloc分配一个新的内存缓冲区，MBUF\_DEFAULT\_HEADROOM为数据预留头部空间(用于协议头)。然后准备缓冲区空间，使用copyin将用户空间数据拷贝到内核缓冲区，调用网络传输函数net\_tx\_udp发送UDP数据包。

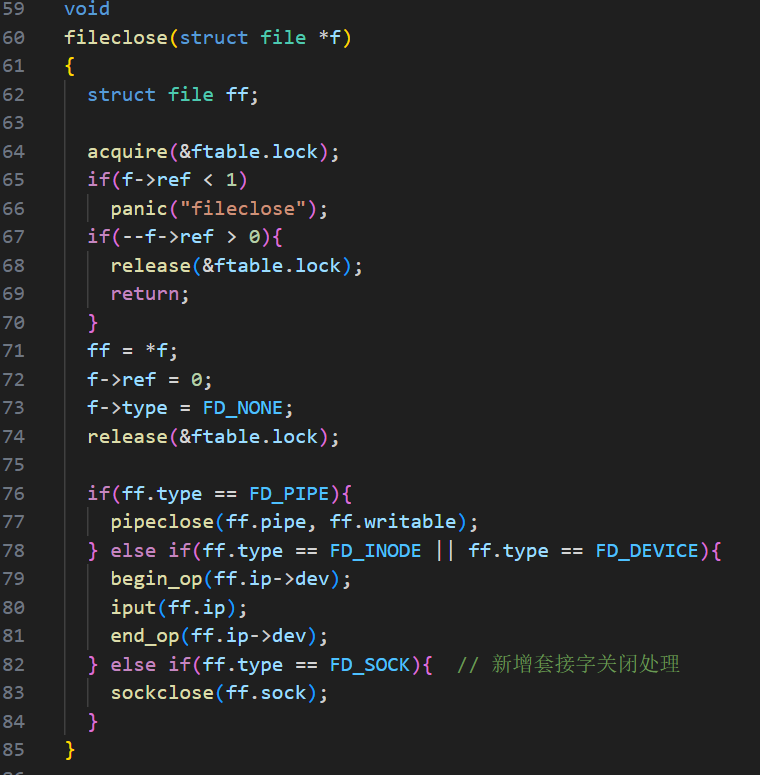


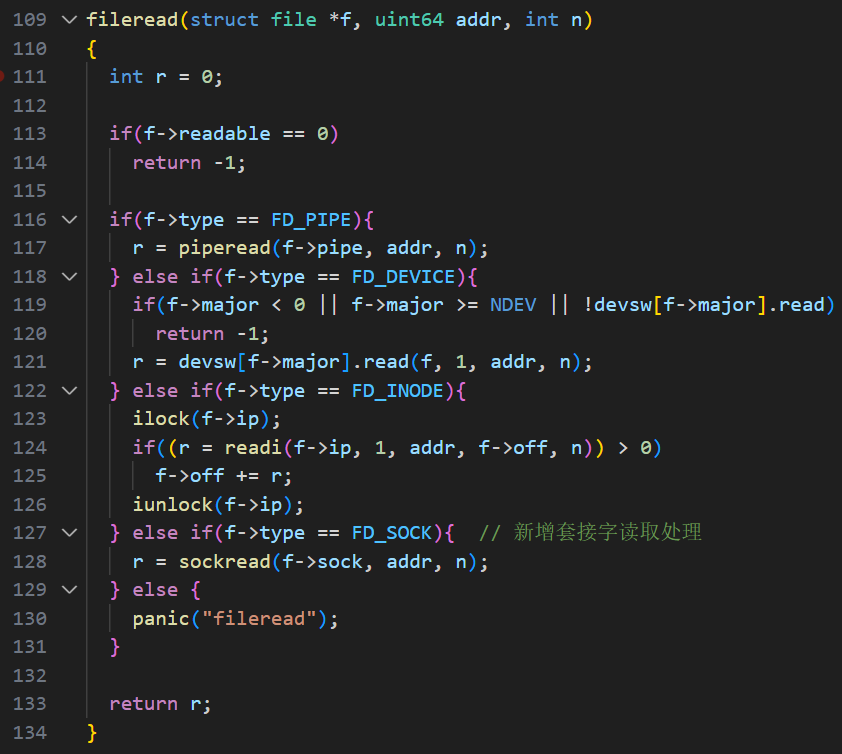
这个函数负责关闭套接字并释放相关资源

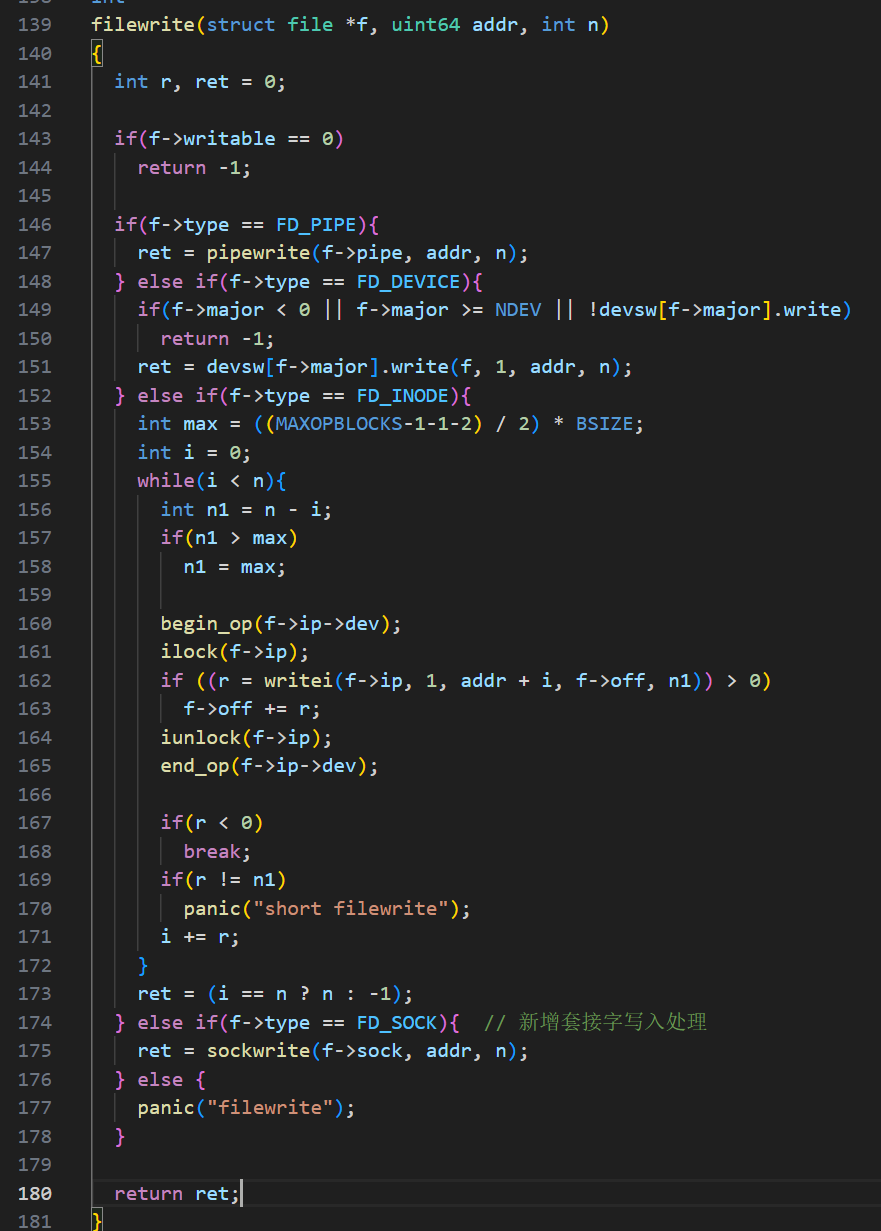
参数：s: 指向要关闭的套接字结构的指针

首先遍历全局套接字链表(sockets)，使用双指针技巧查找并移除目标套接字，修改链表指针绕过要关闭的套接字。循环处理直到接收队列(rxq)为空，从队列头部取出每个mbuf并释放内存，确保没有内存泄漏。

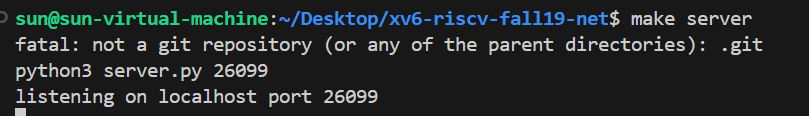
最后在kernel/file.c中添加套接字支持：



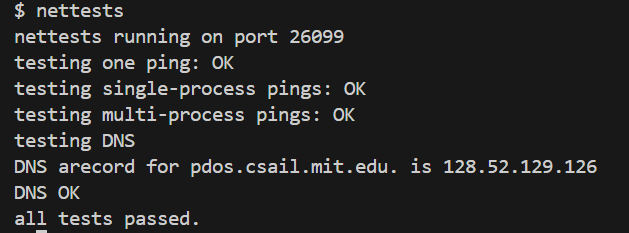




实验现象：



首先在一个终端启动make server监听窗口。



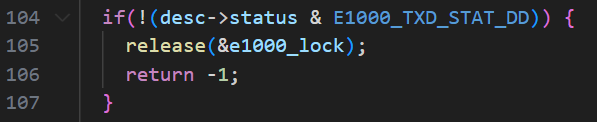
然后在另一个窗口启动nettests，测试通过。

**实验中遇到的问题和解决方法：**

问题描述：在实现e1000\_transmit()函数时，最初未正确检查描述符可用状态，导致当发送队列满时数据包丢失。测试make ping时发现部分数据包未能成功发送，通过tcpdump分析发现发送队列在连续发送16个包后停止工作。

解决方法：

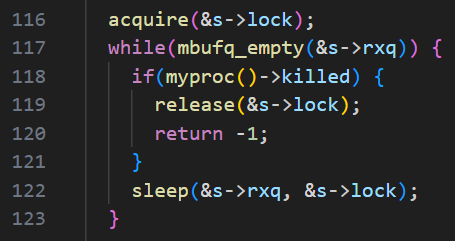
仔细阅读E1000手册第3.3节描述符状态机制。添加描述符状态检查逻辑：在调用处添加重试机制，当返回-1时延迟后重试发送。通过printf调试确认每次发送后TDT寄存器正确递增



问题描述：nettests测试中多进程ping测试时出现死锁，某些进程在sockread()中永久阻塞。分析发现当多个进程等待同一套接字时，wakeup()会唤醒所有进程，但只有一个能获取数据。

解决方法：

将等待逻辑改为while循环检查，添加进程终止检查。



**实验心得：**

通过本次网络实验，我对操作系统底层网络机制有了更深入的理解。在实现E1000网卡驱动时，最初对描述符状态位的理解不够准确，导致发送队列处理出现问题。通过查阅E1000手册，最终掌握了正确的状态检查方法，这让我对硬件交互有了更直观的认识。

在并发控制方面，实验让我切身体会到锁的重要性。多个进程同时访问套接字时，最初出现了链表损坏问题。通过引入全局锁和套接字独立锁的双重保护机制，最终解决了这个问题。这使我对操作系统的同步机制有了更实际的理解。

协议栈的实现让我认识到网络编程的复杂性。特别是在处理UDP校验和时，最初简单的置零处理导致通信失败。通过研究相关文档，最终实现了正确的校验和计算，这让我明白协议设计中的每个细节都很重要。

调试过程也让我获益良多。通过使用tcpdump分析数据包、添加调试打印等方法，我逐步解决了内存泄漏等问题。这些实践经验大大提升了我的系统调试能力。

总的来说，本次实验不仅巩固了我的系统编程基础，更重要的是培养了我分析解决复杂问题的能力。这种通过实践获得的知识，比单纯的理论学习更加深刻和实用。