国科大操作系统研讨课任务书

Project 5

版本：6.0

一、实验说明

通过之前的实验，我们的操作系统已经能够具备任务的调度、例外的处理，并且在用户态运行的进程已经可以通过系统调用接口去调用内核的代码，也可以用 shell 输入命令启动线程，本次任务需要实现网卡驱动。我们需要实现的网卡驱动框架如图 1 所示：

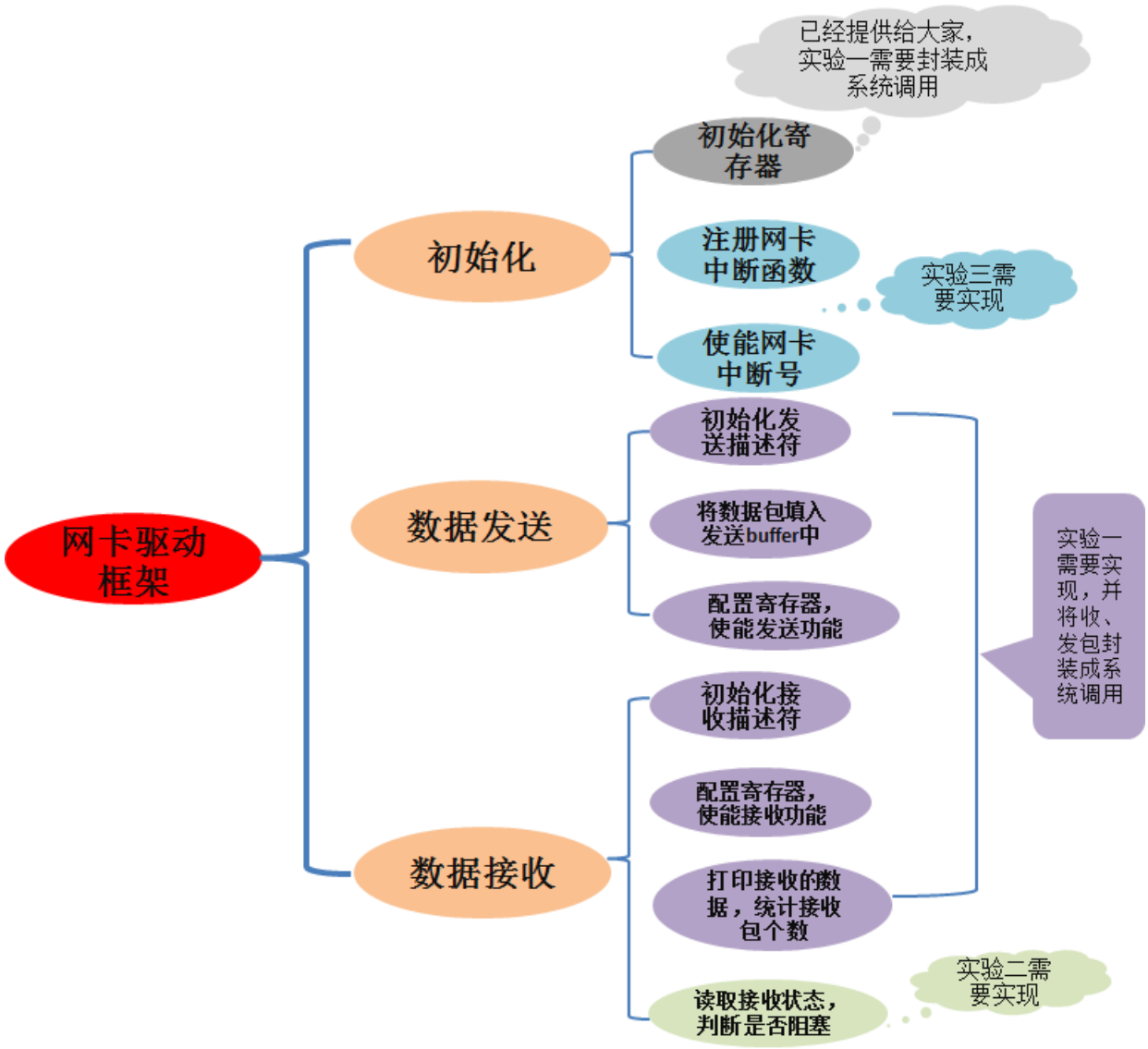

图 1 网卡驱动框架图

我们通过三次实验可以逐步完成一个简单的网卡驱动，本次的实验内容如下：

实验一：系统调用实现的收发数据包

1. 学习使用 DMA 描述符和配置 MAC 控制器中的寄存器；
2. 不涉及中断，实现数据包的发送和接收；
3. 将网卡初始化、数据发送、数据接收分别封装成系统调用。

实验二：有阻塞的网卡收发包

1. 在实验一的基础上实现有阻塞的网卡发包，即当没有数据包到达网卡或到达的数据包不满 64 个时，接收包线程被阻塞，当有数据包到达网卡时，接收包线程在时钟中断中被唤醒。

实验三：添加网卡中断，实现有网卡中断的多线程收发包。

1. 理解龙芯开发版的四级中断，修改 project2 已经完成的例外处理，实现网卡中断函数；
2. 在实验二的基础上实现有网卡中断的收发包，即当没有数据包到达网卡或到达的数据包不满 64 个时，接收包线程被阻塞，当有数据包到达网卡时，接收包线程在网卡中断中被唤醒。

bonus：利用我们提供的 pktRxTx 程序发送本地文件到板子的内存上，测试网卡的接收速度要到 1Mbit/s 以上。

希望大家可以通过本次实验完善自己操作系统的代码。

和 Project2、3、4 一样，我们提供了一个初始的代码框架 start_code，里面会给出一些要实现的基本函数，同学们可以参考它，在它的基础上完成本实验。注意：请将本次 project 的 start-code 增加到同学们自己实现好的 project4 中，进而继续增加网卡驱动的功能，Makefile 文件可以直接替换使用本次 project 的 Makefile，也可以阅读理解 Makefile 后，根据自己增加的 c 文件自行增加 SRC_TEST5 部分。当然，同学们也可以通过合理的设计，改变现有的代码框架，也许你能设计出更好的网卡驱动。

除了本任务书之外，同学们可以参考《Loongson1C300_user_manual_v1.3》中的内容，任务书中第 11 章详细介绍了 MAC 控制器的相关寄存器，任务书中摘录了手册中大部分相关的内容，但是依然推荐同学们在学习和调试的过程多参考手册。同时，同学们也需要结合理论课学到的知识，设计出网卡驱动。从下一章开始，我们将按任务的顺序，详细介绍同学们应该完成的功能。

二、网卡驱动

之前我们已经实现了操作系统中的时钟中断、实现了进程间的通信，内存管理等，已经一步步构建了一个操作系统。此外，驱动程序在系统中的所占的地位也十分重要，一般当操作系统安装完毕后，首要的便是安装硬件设备的驱动程序。本次实验我们来实现一个能收发数据包的网卡驱动。

2.1 开发板 MAC 控制器的介绍

我们使用的龙芯 1C 开发板集成了 1 个 MAC 控制器。MAC 内部集成独有的 DMA 控制器，专门配合 MAC 数据传输；该 DMA 控制器不能被开发板中其他模块使用，MAC 也无法使用开发板中其他的 DMA。MAC 控制器寄存器包括 MAC 寄存器组部分和 DMA 寄存器组部分。MAC 寄存器组的起始地址是 0xbfe10000，一共有 462 个寄存器；DMA 寄存器组的起始地址是 0xbfe11000，一共有 22 个寄存器。我们在对某个寄存器访问时，可以用**寄存器组的起始地址+寄存器的偏移量**的方式进行访问。

2.2 IP/UDP 协议包

在本次网卡驱动实验中，我们给同学们提供的发送数据包结构为：以太网帧头（12 字节）+IP 数据报头（20 字节）+ UDP 数据报头（8 字节）+数据包（984 字节，其中有 48 字节有效数据，其他字节为 0）。我们的板卡是小端，所以在一个 uint32_t 的 4 字节数据内，需要将每个字节逆序。

```
uint32_t buffer[PSIZE] = {0xffffffff, 0x5500ffff, 0xf77db57b, 0x00450008, 0x0000d400, 0x11ff0040,
0xa8c073d8, 0x00e00101, 0xe914fb00, 0x0004e914, 0x0000, 0x005e0001, 0x2300fb00, 0x84b7f28b,
0x00450008, 0x0000d400, 0x11ff0040, 0xa8c073d8, 0x00e00101, 0xe914fb00, 0x0801e914, 0x0000};
```

buffer[PSIZE]数组中，前 12 字节是以太网帧头，其中前 6 字节为目的物理地址（ff:ff:ff:ff:ff:ff），后 6 字节为源物理地址（00:55:7b:b5:7d:f7）；以太网帧头后接着 20 字节的 IP 数据报头；IP 数据报头后接着 8 字节的 UDP 数据报头；UDP 数据报头后接着 984 字节，48 字节有效数据。

在接下来的实验中，**请大家使用我们提供的数据包作为发包的数据。**如果对网络协议包感兴趣的同学可以自行修改提供的数据包，不过需要确保修改后的数据包能发出且被测试软件捕获。

2.3 DMA 简介

DMA 是指外部设备不通过 CPU 而直接与系统内存交换数据的接口技术，要把外设的数据读入内存或把内存的数据传送到外设，一般都要通过 CPU 控制完成，如 CPU 程序查询或中断方式。利用中断进行数据传送，可以大大提高 CPU 的利用率。

但是采用中断传送有它的缺点，对于一个高速 I/O 设备，以及批量交换数据的情况，只能采用 DMA 方式，才能解决效率和速度问题。DMA 在外设与内存间直接进行数据交换，而不通过 CPU，这样数据传送的速度就取决于存储器和外设的工作速度。本次实验就使用了 DMA 技术。

2.4 DMA 描述符

DMA 描述符是 MAC 驱动和硬件的交互接口，如图 2 所示，DMA 描述符记录了数据包的内存地址和传输状态，一共有 16 字节，分别为 DES0、DES1、DES2、DES3。描述符可以自由选择分别以环式(ring mode)或者链式(chain mode)相连，以供 MAC 使用。**实验中建议大家采用环式的链表来连接描述符，即每一个描述符的 buffer2 address 中填入下一个描述符的地址，最后一个描述符的 buffer2 address 中填入第一个描述符的地址。**

需要注意的是描述符的地址必须保证按照所连接的系统总线位宽对齐，同时保证与系统字节序相同(默认小尾端)，

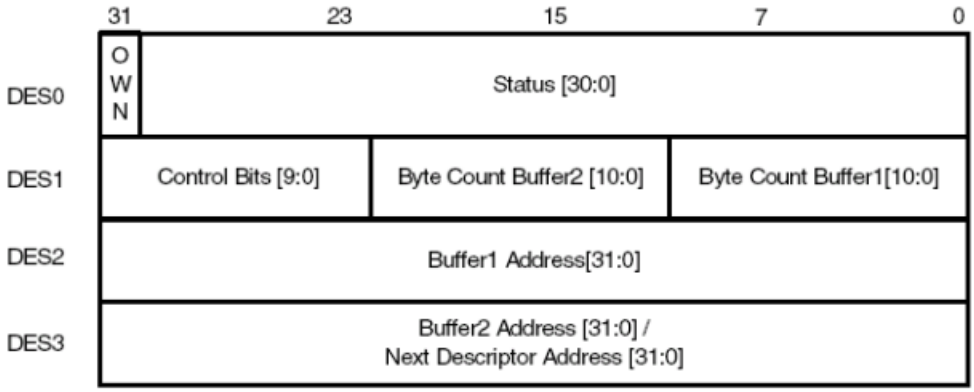

图 2 描述符结构图

具体的，DMA 描述符分为发送描述符(Tx Desciptor)和接收描述符(Rx Descriptor)两种数据结构。下面将详细介绍 DMA 接收描述符和 DMA 发送描述符。

2.3.1 DMA 接收描述符

GMAC 子系统在工作模式下需要至少两个接收描述符才能够正常的接收一个网络数据包。其内部的接收模块在处理一个网络数据包时，总是在同时尝试获取下一个接收描述符。每一个网络数据包被称为一个帧(frame)。

接收描述符的结构如图 3 所示，其中 RDES0 包括了当前接收帧状态、长度以及该描述符的所有情况(主机或 DMA 拥有)，RDES0 的具体每个位域的含义如图 4 所示。在初始化接收描述符时可以将 RDES0 赋值为 0，在开始接收数据（即对 Receive Poll Demand Register 写任意值）时需要将 RDES0 赋值为 0x80000000，即将 OWN 位置 1，表示此接收描述符被 DMA 拥有，此后当 OWN 位变成 0 时，说明此接收描述符已经接收到数据，可以读取接收的数据包。

RDES1 如图 5 所示，记录了描述符所指向的 buffer 大小，以及描述符的组织格式(环形或链型)。**当 RDES3 中存储的是下一个描述符的地址时，即链式，需要将 RDES1 的第 24 位置 1。**如果这个描述符是环形描述符链表中的最后一个，需要将 RDES1 的第 25 位置 1。**建议大家采用环形链表的形式连接描述符。buffer1 大小填在 RDES1 的 0-10 的位域中。**

RDES2 域记录了数据接收 buffer1 的物理地址。**注意，分配的 buffer1 需要进行清零初始化。**

RDES3 域记录了数据接收 buffer2 的物理地址。当 RDES1 的第 24 位设置成 1 时，RDES3 内存储的是下一个接收描述符的物理地址。

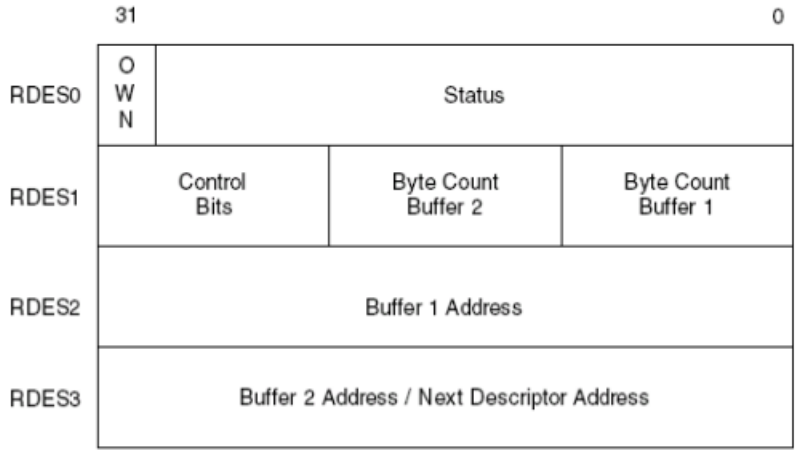

图 3 接收描述符结构图

RDES0 位	位	含义
OWN 所有模式	31	该位为 1 时表示描述符当前属于 DMA 控制，0 表示属于主机控制。当 DMA 模块完成一次传输时，会将该位主动清 0
AFM: Destination Address Filter Fai 目标地址过滤错误 l	30	当该位为 1 时，表示当前数据帧目标地址不符合 GMAC 内部的帧目标地址过滤器
FR: Frame length 帧长度	29:16	表示接收当前帧的长度,当 ES 位为 0 时有效
ES: Error Summary 总体错误信息	15	指示当前帧是否出错，其值为 RDES[0]、RDES[1]、RDES[3]、RDES[4]、 RDES[6]、 RDES[7、RDES[11]、 RDES[14]各位作或运算(OR)的结果
DE: Descriptor Error 描述符错误	14	当该位为 1 时表示，当前描述符所指向的 buffer 与帧不相符或者 OWN 为 0(主机控制)
SAF: Source Address Filter Fail 源地址过滤错误	13	当该位为 1 时，表示当前数据帧的源地址不符合 GMAC 内部的帧源地址过滤器
LE: Length Error 长度错误	12	当该位为 1 时，表示当前接收帧长度与默认长度不符。当 Frame Type 位为 1 且 CRC Error 位为 0 时有效
OE: Over Flow Error 溢出错误	11	当该位为 1 时，表示接收该帧时 GMAC 内部 RxFIFO 溢出
VLAN: VLAN Tag VLAN 标志	10	当该位为 1 时，表示该帧的类型为 VLAN
FS: First Descriptor 第一个描述符	9	当该位为 1 时，表示当前描述符所指向的 buffer 为当前接收帧的第一个保存 buffer
LS: Last Descriptor 最后一个描述符	8	当该位为 1 时，表示当前描述符所指向的 buffer 为当前接收帧的最后一个保存 buffer
IPC Checksum Error/Giant Frame 校验错误/超长帧	7	当该位为 1 时,如果 IPC 校验功能启用则表示当前帧的 IPv4 头校验值与帧内部校验域的值不相符。如果未启用则表示当前帧为一个超长帧(长度大于 1518 字节)
	6	当该位为 1 时，表示在半双工模

LC: late collision 后期冲突		式下，当前帧接收时发生了一个后期冲突
FT: Frame Type 帧类型	5	当该位为 1 时，表示当前帧为一个以太网格式帧，为 0 时表示当前帧为一个 IEEE802.3 格式帧
RWT: Receive Watchdog Timeout	4	当该位为 1 时，表示当前时钟值超过了接收模块看门狗电路时钟的值，既接收帧超时
RE: Receive Error 接收错误	3	当该位为 1 时，表示接收当前帧时内部模块出错。内部信号 rxer 置 1 且 rxdv 置 1
DE: Dribble bit Error 奇数位错误	2	当该位为 1 时，表示接收帧长度不是整数，即总长度为奇数位，该位只有在 mii 模式下有效
CE: CRC Error 接收 CRC 校验错误	1	当该位为 1 时，表示接收当前帧时内部 CRC 校验出错。该位只有在 last descriptor(RDES0[8])为 1 时有效
RX MAC: Checksum/payload Checksum Error 接受校验/负载校验错误	0	当该位为 1 时，表示接收当前帧时内部 RX MAC 寄存器组 1-15 中存在一个匹配当前帧目的地址。为 0 时表示 RX MAC 寄存器组 0 匹配接受帧目的地址。如果 Full Checksum Offload Engine 启用时，为 1 表示该帧 TCP/UDP/ICMP 校验错误。该位为 1 时也可能表示当前帧实际接受长度与帧内部记载长度不相符

图 4 RDES0 位域图

RDES1 位	位	含义
Disable Intr in Completion 禁止完成后发中断	31	该位为 1 时表示该帧接收完成后将不会置起 STATUS 寄存器中 RI 位（CSR5[6]），这将会使得主机无法检测到该中断
Reserved 保留	30:26	
RER: Receive End of Ring 环型描述符结尾	25	该位为 1 时表示该描述符为环型描述符链表的最后一个，下一个描述符的地址为接收描述符链的基址
RCH: Second Address Chained 第二个 buffer 地址指向下个链式描述	24	该位为 1 时表示描述符中的第二个 buffer 地址指向的是下一个描述符的地址，为 0 时表示该地

符		址指向第二个 buffer 地址 当该位为 1 时， RDES1[21-11]的值将没有意义， RDES1[25]比 RDES1[24]具有更高优先级(代表环型而不是链型)
Reserved 保留	23:22	
RBS2: Receive Buffer Size 2 接收 buffer2 大小	21:11	该域表示数据 buffer2 的大小。根据系统总线的宽度 32/64/128，Buffer2 的大小应该为 4/8/16 的整数倍。如果不满足则会导致未知的结果。该域在 RDES1[24]为 0 时有效
RBS2: Receive Buffer Size 1 接收 buffer1 大小	10:0	该域表示数据 buffer1 的大小。根据系统总线的宽度 32/64/128， Buffer1 的大小应该为 4/8/16 的整数倍。如果不满足则会导致未知的结果。该域一直有效。如果该域值为 0，DMA 则会自动访问 buffer2 或者下一个接收描述符

图 5 RDES1 位域图

2.3.2 DMA 发送描述符

DMA 发送描述符如图 6 所示，发送描述符与接收描述符的格式基本相同。每个描述符的地址需要按照总线宽度(32/64/126 位)对齐。

TDES0 每个位域的含义如图 7 所示，包含了发送帧的状态和发送描述符的所属信息。与接收描述符相同，在初始化发送描述符时可以将 TDES0 赋值为 0，在开始发送数据（即对 Transmit Poll Demand Register 写任意值）时将 TDES0 赋值为 0x80000000，即将 OWN 位置 1，表示此发送描述符被 DMA 拥有，此后当 OWN 位变成 0 时，说明此发送描述符已经发送完数据。

TDES1 每个位域的含义如图 8 所示，包含了 buffer 大小以及其他一些控制位，如环型/链型连接的控制和状态位。**当 TDES3 中存储的是下一个描述符的地址时，即链式，需要将 TDES1 的第 24 位置 1。**如果这个描述符是环形描述符链表中的最后一个，需要将 TDES1 的第 25 位置 1.此外，还需要注意当前 buffer 是否是一帧数据的最后一段或第一段，是否需要完成时中断等。**buffer1 大小填在 TDES1 的 0-10 的位域中。**

TDES2 域记录了发送数据 buffer1 的物理地址。

TDES3 域记录了发送数据 buffer2 的物理地址。当位链式连接描述符时，TDES3 存放的是下一个发送描述符的物理地址。

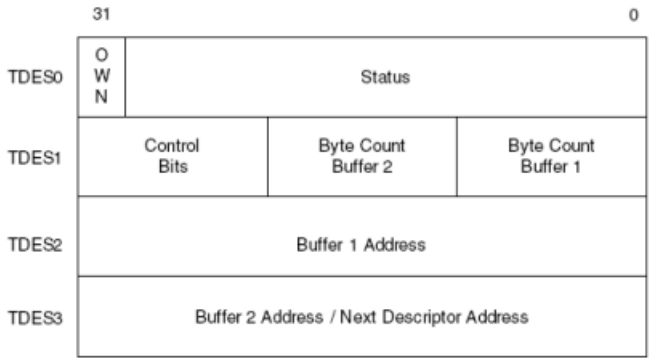

图 6 发送描述符结构图

TDES0 位	位	含义
OWN 所属模式	31	该位为 1 时表示描述符当前属于 DMA 控制， 0 表示属于主机控制。当 DMA 模块完成一次传输时，会将该位主动清 0
Reserved 保留	30:18	
TTSS: Tx Time Stamp Status 发送时间戳状态	17	当 IEEE1588 功能启用时，该位为 1 表示 TDES2 和 TDES3 中保存了该发送帧的时间戳信息。否则该位保留
IHE: IP Header Error IP 头错误	16	该位为 1 时表示内部校验模块发现该发送帧的 IP 头出错，并且不会对该域做任何修改
ES: Error Summary 总体错误信息	15	指示当前帧是否出错，其值为 TDES[1]、 TDES[2]、 TDES[8]、 TDES[9]、 TDES[10]、 TDES[11]、 TDES[13]、 TDES[14]各位作或运算(OR)的结果
JT: Jabber Timeout Jabber 超时	14	该位为 1 时表示 GMAC 发送模块遇到了 Jabber 超时
FF: Frame Flushed 帧刷新	13	该位为 1 时表示软件发出了一个刷新命令导致 DMA/MTL 将其内部的帧刷新掉
PCE: Payload Checksum Error 负载校验错误	12	该位为 1 时表示内部负载校验模块再向发送帧中插入校验数据时出错。当负载校验模块启用时，该位

		有效
LC: Loss of Carrier 载波丢失	11	该位为 1 时表示在发送该帧过程中载波丢失(gmii_crs 信号多个周期未置起)
NC: No Carrier 载波无效	10	该位为 1 时表示在发送过程中，PHY 的载波信号一直未置起
LC: Late Collision 后期冲突	9	当该位为 1 时表示在半双工模式下，当前帧接收时发生了一个后期冲突
EC: Excessive Collision 连续冲突	8	当该位为 1 时表示在发送当前帧的时候连续出现了 16 次冲突
VF: VLAN Frame VLAN 帧	7	该位为 1 时表示当前发送帧为一个 VLAN 帧
CC: Collision Count 冲突计数	6:3	该域表示当前帧在成功发送之前所遇到冲突次数的总数
ED: Excessive Deferral 连续 Deferral	2	该位为 1 时表示当前帧传输结束
UF: Underflow Error 溢出错误	1	该位为 1 时表示当前帧传输时发生了溢出错误，即数据传输 buffer 过小或不可用
DB: Deferred Bit 帧刷新	0	该位为 1 时表示此次发送被延迟，只有在半双工模式下有效

图 7 TDES0 位域结构图

TDES1 位	位	含义
IC: Interruption on Complete 完成时中断	31	该位为 1 时表示该帧接发送完成后将会置起 STATUS 寄存器中 TI 位(CSR5[0])
LS: Last Segment 最后段	30	该位为 1 时表示当前 buffer 包含的是一帧数据的最后一段(如果帧分为多个段)
FS: First Segment 第一段	29	该位为 1 时表示当前 buffer 包含的是一帧数据的第一段(如果帧分为多个段)
CIC: Checksum	28:27	该域控制内部模块是否在发送帧中

Insertion Control 校验数据填充控制		填充校验数据。值： 2'b00: 不填充校验数据 2'b01: 填充 IPV4 头校验数据 2'b10: 在伪头数据(pseudo-header)存在的情况下，填充 TCP/UDP/ICMP 全校验数据 2'b11: 总是填充 TCP/UDP/ICMP 的全校验数据
DC: Disable CRC 禁止 CRC 校验	26	该位为 1 时 GMAC 硬件不在每个发送帧的结尾添加 CRC 校验数据
TER: Transmit End of Ring 环形描述符结尾	25	该位为 1 时表示该描述符为环型描述符链表的最后一个，下一个描述符的地址为发送描述符链的基址
TCH: Second Address Chained 第二个 buffer 地址指向下个链式描述符	24	该位为 1 时表示描述符中的第二个 buffer 地址指向的是下一个描述符的地址，为 0 时表示该地址指向第二个 buffer 地址当该位为 1 时， TDES1[21-11]的值将没有意义， TDES1[25]比 TDES1[24]具有更高优先级(代表环型而不是链型)
DP: Disable Padding 禁止填充	23	该位为 1 时表示 GMAC 将不会对长度小于 64 字节的数据包进行空数据填充
TTSE: Transmit Time Stamp Enable 启用发送时间戳	22	该位为 1 时表示将启用内部模块计算 IEEE1588 硬件时间戳计算，在 TDES1[29]为 1 时有效
TBS2: Transmit Buffer Size 2 发送 buffer2 大小	21:11	该域表示数据 buffer2 的大小。当 TDES1[24]为 1 时，该域无效
TBS1: Transmit Buffer Size 1 发送 buffer1 大小	10:0	该域表示数据 buffer1 的大小。该域一直有效。如果该域值为 0， DMA 则会自动访问 buffer2 或者下一个接收描述符

图 8 TDES1 位域结构图

2.5 实验 1：系统调用实现的收发数据包

2.5.1 实验要求

本次实验不涉及中断，需要大家通过系统调用分别实现网卡的发包 sys_net_send（）和收包 sys_net_recv（），并

在能在 shell 中启动、运行、退出 phy_regs_task1（）、phy_regs_task2（）和 phy_regs_task3（）。

本次实验需要使用 wireshark（window 系统用 wireshark，Linux 和 mac 系统用 ***tcpdump***）软件进行抓包，查看网卡有没有把数据包发送出来，如图 9 所示，为发包成功的参考图，可以查看最后一个包的 Frame 值减去第一个包的 Frame 值，如果等于 256 则表示 wireshark256 收到了 256 个包：

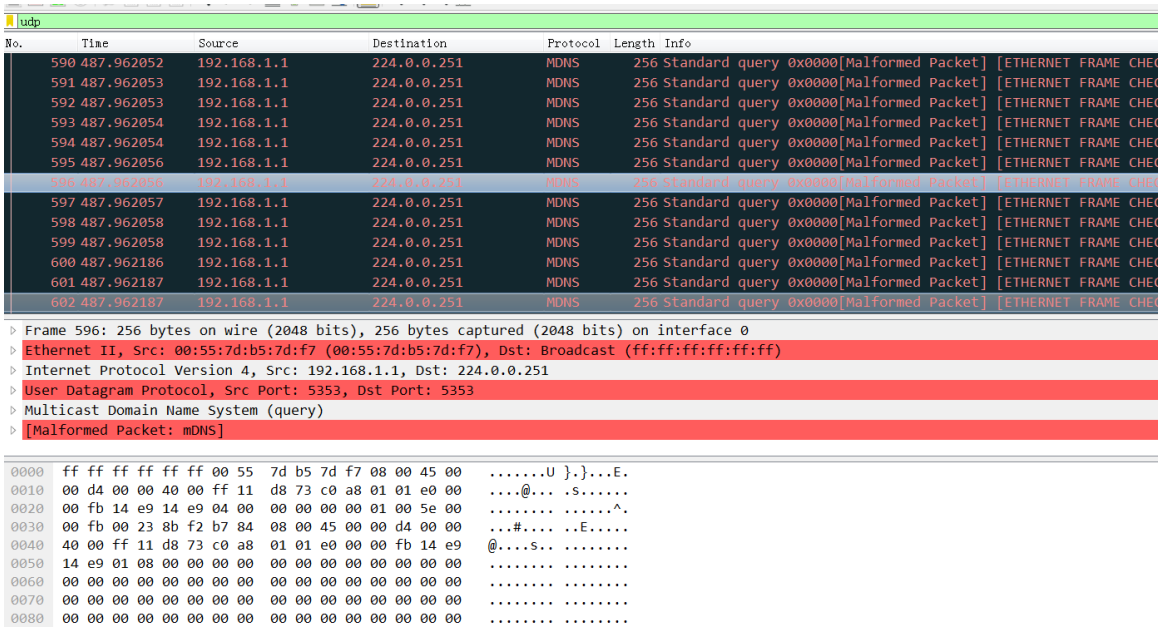

图 9 发送成功效果图（Windows 版）

在测试收包功能时，在 Windows 系统中大家需要使用我们提供的小程序 pktRxTx.exe。如图 10 所示，具体操作需要在 cmd 中进入 pktRxTx.exe 所在的目录，输入 pktRxTx.exe -m 2 命令，选择自己电脑的网卡，然后输入 send 60 即可发送 60 个数据包。小程序 pktRxTx.exe 的其他平台上的使用方法请参看提供的《pktRxTx 使用说明书》。

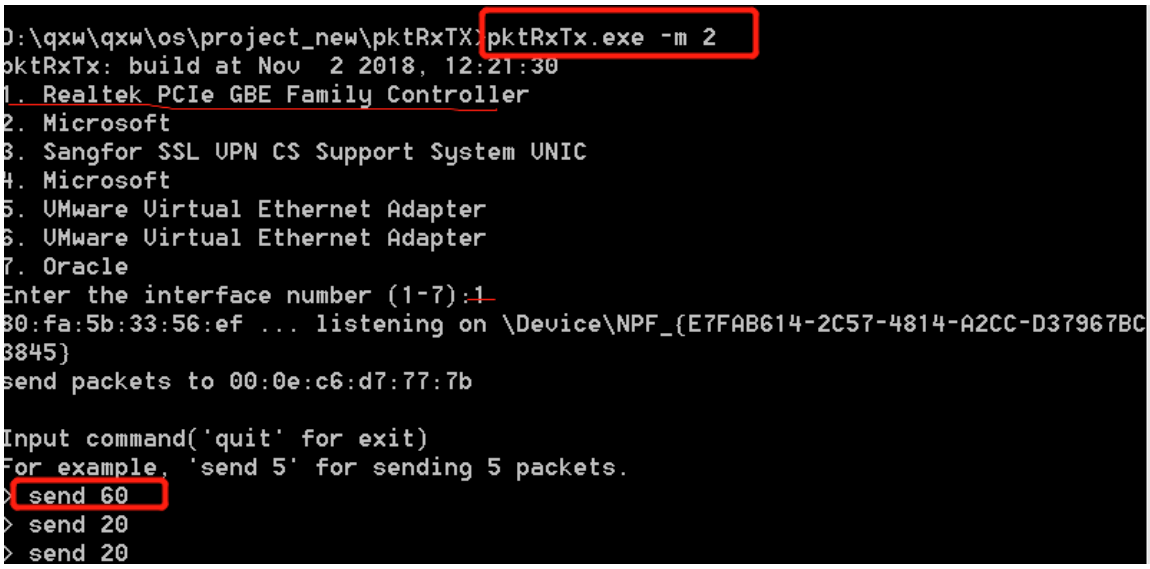

图 10 pktRxTx.exe 使用说明图（Windows 版）

网卡收包成功的打印效果图如图 11 所示，从图中可以看到当前接收描述符的 RDES0=0X15a0320，第 0~第 3 位为 0，第 6 位~第 7 位为 0，第 11 位~15 位为 0，第 30 位为 0，这表示当前接收到的数据没有出错，接收成功。

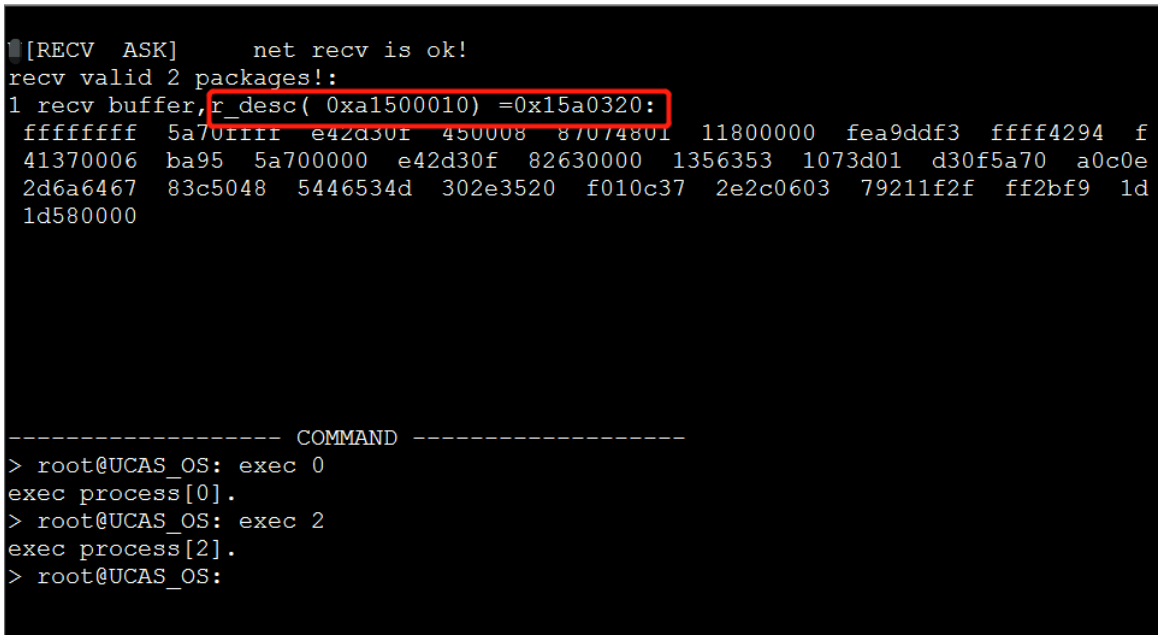

图 11 网卡收包成功的打印效果图

2.5.2 文件介绍

在 project2 的基础上，我们给大家的 startcode 中添加以下的文件：

	文件名	说明
1	test/test_net /test4.h	project3 的测试头文件
2	test/ test_net /test_regs1.c	测试任务一
3	test/ test_net /test_regs2.c	测试任务二
4	test/ test_net /test_regs3.c	测试任务三
5	device/mac/mac.c	网卡驱动相关代码，需补充其内容完成任务一、二、三
6	device/mac/mac.h	网卡驱动相关的头文件
7	Makefile	Makefile 文件

2.5.3 实验步骤

- 把 do_init_mac（）封装为系统调用 sys_init_mac（），把 do_net_recv（）封装为系统调用 sys_net_recv（），把 do_ net_send （）封装为系统调用 sys_ net_send（）；
- 实现初始化发送描述符 send_desc_init（）与接收描述符 recv_desc_init（），请大家分配接收描述符和发送描述符的内存区域，发送包的内存区域和接收包的内存区域并初始化分配的内存区域，大家可以在 unmapped 的地址空间中进行分配，比如 kseg1 空间。**建议大家采用环形链表的形式连接描述符**；
- 分别实现系统调用 sys_net_recv（）和 sys_net_send（），实现网卡的发送和接收数据包；
- 在 sys_net_recv（）中除了使能接收外，还需要实现轮询查看是否接收到数据包，只要收到 1 个包，就打印出来；
- 分别在 DMA 寄存器 4（Transmit Descriptor List Address Register，偏移为 0x10）和 DMA 寄存器 3（Receive Descriptor List Address Register，偏移为 0xC）中填入发送描述符和接收描述符的首物理地址。这个操作大家可以调用我们提供的 reg_write_32（）函数对寄存器赋值。
- 分别将 mac 第 0 寄存器的第 3 位和第 4 位设置为 1，这样可以分别使能 MAC 传输功能和接收功能

- 配置 DMA 第 6 寄存器、DMA 第 7 寄存器。这个操作在 do_net_send（）和 do_net_recv（）里已经帮大家实现了，请大家查看《Loongson1C300_user_manual_v1.3》手册，了解配置的每一位的含义。
- 在发送和接收前，每个描述符的 OWN 位需置 1，当开始发送和接收后，OWN 位从 1 变成 0 时（硬件自动置位），则此描述符已经完成的发送或接收，**可以以 OWN 为判断是否完成了发送或接收**；
- 每次发送前需要在 DMA 寄存器 1（Transmit Poll Demand Register）中写入任意值，发送 DMA 控制器将会读取寄存器 18 对应的描述符，这样就开始发送了一个数据包。
- 每次接收前需要在 DMA 寄存器 2（Receive Poll Demand Register）中写入任意值，接收 DMA 控制器将会读取寄存器 19 对应的描述符，这样当有数据包到达板卡时就会接收一个数据包。
- 使用 exec 指令启动 task1、task2、task3，关于任务的内容，阐述如下：

task1	初始化网卡驱动。
task2	网卡发送包线程。
task3	网卡接收包线程。

2.5.4 注意事项

想详细了解网卡驱动相关寄存器的同学可以详细阅读《Loongson1C300_user_manual_v1.3》的 11 章 MAC 控制器，在此章节中详细描述了网卡的 MAC 寄存器组和 DMA 寄存器组。

2.6 实验 2：有阻塞的网卡收包

2.6.1 实验要求

本次实验需要大家实现有阻塞的网卡收包。本实验与实验一的不同点就是在网卡的接收包线程中，在使能网卡的接收功能后，当没有数据包到达网卡或到达的数据包不满 64 个时，接收包线程被阻塞，当有数据包到达网卡时，接收包线程被唤醒，开始打印接收的数据。唤醒的位置判断在时钟中断中。

2.6.2 文件介绍

请参照实验一的文件介绍。

2.6.3 实验步骤

在实验一的基础上继续实现：

（1） 在 sys_net_recv（）中只需要实现使能网卡具有接收功能，不需要检查是否接收到数据包；

（2） 在 check_recv（）检查是否接收到数据包，如果接收到则打印出数据包的内容，如果没有接收到数据包，则阻塞接收线程；

（3） 实现 sys_wait_recv_package（）阻塞接收线程的系统调用；

（4） 在时钟中断中判断是否唤醒接收线程。

（5）使用 exec 指令启动 task1、task2、task3，关于任务的内容，阐述如下：

task1	初始化网卡驱动。
task2	网卡发送包线程，实现网卡发送我们提供的数据包
task3	网卡接收包线程，实现当没有数据包到达网卡时，接收包线程被阻塞，当有数据包到达网卡时，接收包线程被唤醒，开始打印接收的数据。**唤醒的位置判断在时钟中断中。**

（6）下图为实验成功的打印结果：

发送包时，如图 12、13 所示：

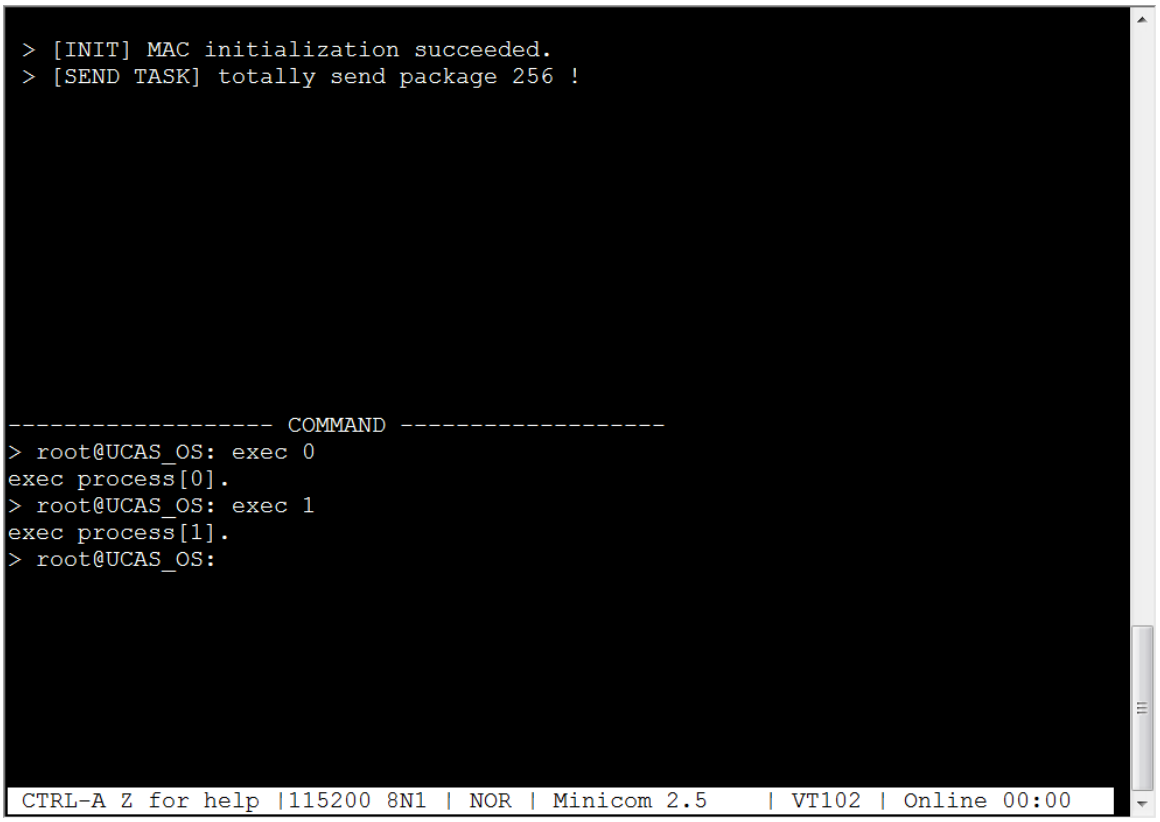

图 12 网卡发包成功的打印效果图

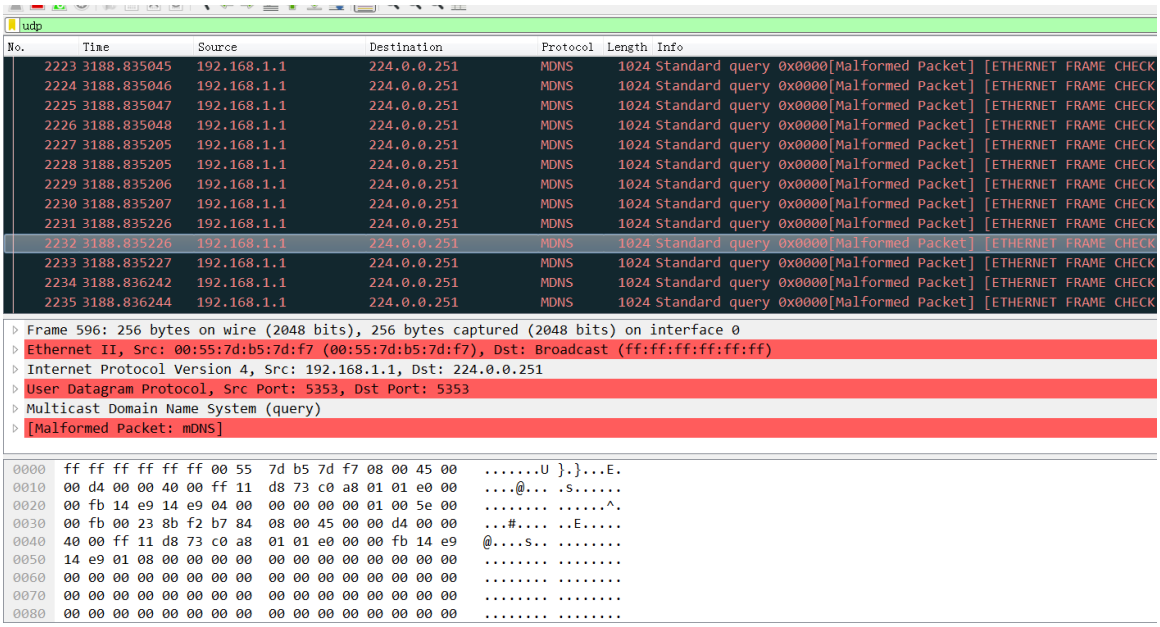

图 13 网卡发包成功的 wireshark 打印效果图

接收包时如图 14、15 所示：

当接收不够 64 个数据包被阻塞的情况如下图 14 所示，可以看到当前接收描述符的 RDES0=0X570320，第 0~第 3 位为 0，第 6 位~第 7 位为 0，第 11 位~15 位为 0，表示当前接收到的数据没有出错，接收成功。

```
 > [RECV TASK]print recv  buffer

 > [RECV TASK]still waiting receive 59th package.
58 recv buffer,r desc( 0xa15003a0) =0x570320:
 5e0001  5a70fb00  e42d30f  450008  a5044500  11ff0000  fea9c697  e04294  e914fb
00  3100e914  d61  d4d85ee
 1000000  5f0c0000  65656c73  72702d70  479786f  7064755f  636f6c05  6c61  2f010
00c  7077b74c  322e
 3039313a  4d0a0d30  203a4e41  64737322  69643a70  766f6373  d227265  3a584d0a
a0d3120  203a5453
 3a6e7275  6c616964  6c756d2d  63736974  6e656572  67726f2d  7265733a  65636976
 6169643a  d313a6c
 4553550a  47412d52  3a544e45  6f6f4720  20656c67  6f726843  372f656d  2e302e30
 38333533  3031312e
```

图 14 网卡收包成功的打印效果图

接收到 64 个数据包的情况如下图所示：

```
 > [INIT] MAC initialization succeeded.n .
 > [SEND TASK] totally send package 256 !
 > [RECV TASK]still waiting receive 50th package.
recv valid 64 packages!:xa15003f0) =0x4e0320:
                b77  ef56335b  450008  31d43c00  6ff0000  46dd  a0000  52b74400
 51c3  5400    22020200
 10500000  fb63d016  6e000000  616d726f  1111116c  11111111  11111111  f821111
11119137  70c00  2e
 f5a  27000000  a000c00  6e61696a  2d6a6467  10005048  e00  8003701  5446534d  3
02e3520  b30000
 8000600  17001800  27001100  b55a5060  27001100  1000100  3c000200  414d5c00
4c534c49  d313a6c
 425c544f  53574f52  c0045  493e0  4b524f57  554f5247  50  10000a03  7fe8000  49
4a0000  3  3230312e
 44474e41  50482d4a  8bdbf500  494a00cd  8bdbf500
> root@UCAS_OS: exec 0
exec process[0].
> root@UCAS_OS: exec 1
exec process[1].
> root@UCAS_OS: exec 2
exec process[2].
> root@UCAS_OS:

CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.5     | VT102 | Online 10:17
```

图 15 网卡收包成功的打印效果图

2.6.4 注意事项

请大家一次只编译一个网卡实验的.c 文件，防止编译错误。比如在实验以中 MAKEFILE 内 SRC_TEST5 = ./test/test_net/test_regs1.c，在实验二中 MAKEFILE 内 SRC_TEST5 = ./test/test_net/test_regs2.c。

2.7 龙芯的 4 级中断

在 project2 中我们已经介绍了龙芯开发板的 4 级例外，因为网卡驱动的收发涉及到网卡中断，在此我们回顾一下龙芯开发板的 4 级例外：

我们使用的通过这四级的判断，我们可以知道是哪种情况产生的中断。龙芯 1c 开发板的四级例外具体如下：

第一级:各种情况下例外的总入口，即上小节提到的一个固定例外处理的入口，每当 CPU 发现一个中断，都会将执行地址跳转到这个例外向量入口，这也是 MIPS 架构下所有例外的总入口；

第二级:各个例外的入口，如图 16 所示，通过 CP0 的 cause 寄存器中 ExcCode 位域来区分不同例外的入口，其中 CP0 的 cause 寄存器中 ExcCode=0 的异常为中断的总入口；

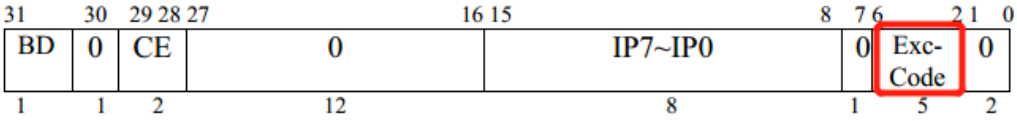

图 16 cause 寄存器的位域图

如图 17 所示，以下为不同 Exccode 对应的例外类型，可以看到，当 ExcCode 为 0 的时候，说明触发的例外为中断，我们需要完成的网卡中断就是一种 ExcCode 为 0 的中断：

例外代码	Mnemonic	描述
0	INT	中断
1	MOD	TLB 修改例外
2	TLBL	TLB 例外（读或者取指令）
3	TLBS	TLB 例外（存储）
4	ADEL	地址错误例外（读或者取指令）
5	ADES	地址错误例外（存储）
6	IBE	总线错误例外（取指令）
7	DBE	总线错误例外（数据引用：读或存储）
8	SYS	系统调用例外
9	BP	断点例外
10	RI	保留指令例外
11	CPU	协处理器不可用例外
12	OV	算术溢出例外
13	TR	陷阱例外
14	-	保留
15	FPE	浮点例外
16—22	-	保留
23	WATCH	WATCH 例外
24—30	-	保留
31	-	保留

图 17 Exccode 对应的例外类型

第三级:不同中断的各个入口，根据 IP7~IP0 的 8 位域（CP0_STATUS 寄存器中的）来区分，其中，**像网卡这种外设的中断是 IP3=1，其他 IP 为 0，外设中断需要判断到第四级才能确定是哪种外设引起的中断**；

第四级:每个外设中断的入口（龙芯 1c 将所有外设分为五组，对应有 5 组中断配置寄存器组，查看 5 组寄存器中各个 SR 寄存器中哪个位域是 1，就对应是哪一种外设产生的中断），具体可以参考《Loongson_1C300_user_1_3》5.3 中断配置寄存器。具体地，MAC 中断状态位是在 INT1_SR 寄存器中的第 3 位，如图 18 所示，所以 MAC 的中断号为 32+3，即 35。

INT1_SR	位	缺省值	描述
Gpio[105:96]	31:22	0	GPIO[105:96]作为中断输入，中断状态位
Reserved	21:20	0	
I2C_int0	19	0	I2C0 中断状态位
I2C_int1	18	0	I2C1 中断状态位
I2C_int2	17	0	I2C2 中断状态位
Reserved	16	0	
UART11_int	15	0	UART11 中断状态位
UART10_int	14	0	UART10 中断状态位
UART9_int	13	0	UART9 中断状态位
UART8_int	9	0	UART8 中断状态位
UART7_int	8	0	UART7 中断状态位
UART6_int	7	0	UART6 中断状态位
UART5_int	6	0	UART5 中断状态位
UART4_int	5	0	UART4 中断状态位
cam_int	4	0	CAMERA 中断状态位
mac_int	3	0	MAC 中断状态位
otg_int	2	0	OTG 中断状态位
ohci_int	1	0	USB_OHCI 中断状态位
ehci_int	0	0	USB_EHCI 中断状态位

图 18 INT1_SR 寄存器

即，网卡驱动的中断号是 35。通过这个中断号知道应该将第一组中断寄存器的 INT1_EN 中的第三位置 1（35/32=1，即第一组中断寄存器，35%32=3，即第三位），这样就可以使能网卡中断。

当去查找是哪种设备触发中断时，如果第一组中断寄存器的 INT1_SR 寄存器的第三位为 1，则此时是网卡中断。通过这四级例外的查找，我们可以知道到底是哪种情况触发的例外。

INT1_SR 寄存器的地址是 0xbfd01058，INT1_EN 中断使能寄存器的地址是 0xbfd0105c。在去使能网卡中断时，我们可以直接用 INT1_EN 的地址对 INT1_EN 的第三位进行赋值 1，也可以采用通过网卡中断号得到**中断配置寄存器组的偏移+中断配置寄存器组的首地址**来对 INT1_EN 进行赋值。中断配置寄存器组的首地址即 INT0_SR 的地址为 0xbfd01040。中断配置寄存器组 0 的内容如表 1 所示，其他 4 组中断配置寄存器和此类似：

表 1 第 0 组中断寄存器

寄存器名称	地址	读/写(R/W)	功能描述	复位值
INT0_SR	0xbfd0_1040	R	中断状态寄存器 0，为 1 表示对应的 bit 位产生中断，为 0 表示无中断	0x0
INT0_EN	0xbfd0_1044	R/W	中断使能寄存器 0，为 1 表示使能对应的 bit 位产生中断，为 0 表示禁止	0x0
INT0_SET	0xbfd0_1048	R/W	中断置位寄存器 0，为 1 表示中断置位，为 0 无效	0x0
INT0_CLR	0xbfd0_104C	R/W	中断清 0 寄存器 0，为 1 表示清除中断，为 0 无效	0x0
INT0_POL	0xbfd0_1050	R/W	中断极性选择寄存器 0，当 int0_edge 配置成电平触发时，为 1 表示高电平触发中断，为 0 表示低电平触发；当 int0_edge 配置成边沿触发时，为 1 表示上升沿触发中断，为 0 表示下降沿触发	0x0
INT0_EDGE	0xbfd0_1054	R/W	中断边沿选择寄存器 0，为 1 表示边沿触发 中断，为 0 表示电平触发	0x0

2.8 实验 3 有网卡中断的多线程收发包

2.8.1 实验要求

本实验与实验二的不同点就是：需要注册网卡中断函数，在网卡中断函数中判断是否唤醒接收包线程。网卡的中断号为 35。同样地，在网卡的接收包线程中，当没有数据包到达网卡或到达的数据包不满 64 个时，接收包线程被阻塞，当有数据包到达网卡时，接收包线程被唤醒，开始打印接收的数据。**唤醒的判断位置在网卡中断内。**发送数据包的功能和实验二相同。实验结果图与实验二相同。

2.8.2 文件介绍

请参照实验 1.

2.8.3 实验步骤

本次实验请大家在已经完成的实验二的基础上实现：

在例外处理 irq.c 文件中添加：

- 在例外初始化时将 status 的 IM7~IM2 位置 1。
- 在中断处理函数 interrupt_helper（）中添加解析设备中断的功能，实现当 IP3=1 时检查第一组中断寄存器组的状态寄存器的第三位是否为 1，如果是 1 则跳入网卡中断函数；

在 mac.c 文件中：

- 网卡中断处理函数 irq_mac（），在 irq_mac（）中判断是否唤醒接收线程；
- 注册中断处理函数 register_irq_handler(int IRQn, irq_handler_t func)；
- 使能网卡驱动的中断号函数 irq_enable(int IRQn)，我们可以直接用 INT1_EN 的地址对 INT1_EN 的第三位进行赋值 1，也可以采用通过网卡中断号得到**中断配置寄存器组的偏移+中断配置寄存器组的首地址**来对 INT1_EN 进行赋值。中断配置寄存器组的首地址即 INT0_SR 的地址为 0xbfd01040。

2.8.4 注意事项

请大家思考一下在有网卡中断的情况下的例外处理流程和如何初始化例外。另外在模拟器 qemu 中，不能触发网卡中断，**所以本次实验的调试不能使用 qemu 调试，只能在板子上调试。**

2.9BONUS

2.9.1 实验要求

使用我们提供的 pktRxTx 程序发送本地文件到板子的内存上，测试网卡的接收速度要到 1Mbit/s 以上。

选择 pktRxTx 的-m 3 模式，在此模式中可以在一定时间内以 1.9MByte/s 速率向开发板发送数据包，如 test 30 表示在 30 秒内以 1.9MByte/s 的速率向开发板发送数据包。因此，我们可以通过控制发送时间来控制发给开发板的数据大小。

附录 1：如何在 qemu 中调试网卡驱动

监听发送包方法如下：

实现监听网卡发送出来的数据包的步骤需要在 termin 中依次输入（//后面是注释，不用输入）：

sudo apt-get install bridge-utils //安装 bridge-utils

sudo apt-get install uml-utilities //安装 uml-utilities

sudo brctl addbr br0 //加入网桥

sudo tunctl -t tap0 –u stu//让用户 stu 可以控制 tap0

sudo brctl addif br0 tap0

sudo ifconfig tap0 up

具体操作的截图如图 19 所示：

```
stu@stu-VirtualBox:~$ sudo apt-get install bridge-utils
Reading package lists... Done
Building dependency tree
Reading state information... Done
bridge-utils is already the newest version.
bridge-utils set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 375 not upgraded.
stu@stu-VirtualBox:~$ sudo apt-get install uml-utilities
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  user-mode-linux
The following NEW packages will be installed:
  uml-utilities
0 upgraded, 1 newly installed, 0 to remove and 375 not upgraded.
Need to get 60.1 kB of archives.
After this operation, 251 kB of additional disk space will be used.
Get:1 http://cn.archive.ubuntu.com/ubuntu/ precise/universe uml-utilities i386 20070815-1.3ubuntu1 [60.1 kB]
Fetched 60.1 kB in 2s (29.1 kB/s)
Selecting previously unselected package uml-utilities.
(Reading database ... 146920 files and directories currently installed.)
Unpacking uml-utilities (from .../uml-utilities_20070815-1.3ubuntu1_i386.deb) ...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
Setting up uml-utilities (20070815-1.3ubuntu1) ...
 * Starting User-mode networking switch uml_switch
stu@stu-VirtualBox:~$
```

```
stu@stu-VirtualBox:~$ sudo brctl addbr br0
[sudo] password for stu:
stu@stu-VirtualBox:~$ tunctl -t tap0 -u stu
TUNSETIFF: Operation not permitted
stu@stu-VirtualBox:~$ sudo tunctl -t tap0 -u stu
Set 'tap0' persistent and owned by uid 1000
stu@stu-VirtualBox:~$ sudo brctl addif br0 tap0
stu@stu-VirtualBox:~$ sudo ifconfig tap0 up
```

图 19 设置网桥示意图

如图 20 所示，在 QUME_Loongson 文件夹中的 run_pmon.sh 文件中添加：

-net tap,ifname=tap0,script=no,downscript=no –net nic

```
#SERIAL=2 ./qemu/bin/qemu-system-mipsel -M ls1c -vnc 127.0.0.1:0 -kernel ./bios/gzram -gdb tcp::500
10 -m 32 -usb -drive file=disk,id=a,if=none -device usb-storage,bus=usb-bus.1,drive=a  -serial stdi
o  "$@"

SERIAL=2 ./qemu/bin/qemu-system-mipsel -M ls1c -vnc 127.0.0.1:0 -kernel ./bios/gzram -gdb tcp::5001
0 -m 32 -usb -drive file=disk,id=a,if=none -device usb-storage,bus=usb-bus.1,drive=a  -serial stdio
-net tap,ifname=tap0,script=no,downscript=no -net nic  "$@"
~
~
~
~
~
~
~
```

图 20 修改 run_pmon.sh 文件示意图

我们可以使用 tcpdump 来对网卡发出的数据包进行截获，最后在 terminal 中输入 ***sudo tcpdump -i tap0 –n –xx*** 命令就可以监听模拟器中网卡发出的数据包了。

发包小程序的使用方法如下：

在 QEMU 中调试时，可以使用我们提供的发包小程序给模拟器中的网卡驱动发送一定数量的数据包，这样可以测试网卡的接收数据包功能。如图 21 所示，在编译发包小程序前，我们需要在 terminal 中输入以下命令：

sudo apt-get install libpcap

sudo apt-get install libpcap-dev

sudo apt-get install libnet-dev

```
stu@stu-VirtualBox:/mnt/shared$ sudo apt-get install libpcap
[sudo] password for stu:
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package libpcap
stu@stu-VirtualBox:/mnt/shared$ sudo apt-get install libpcap-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libpcap0.8-dev
The following NEW packages will be installed:
  libpcap-dev libpcap0.8-dev
0 upgraded, 2 newly installed, 0 to remove and 375 not upgraded.
Need to get 219 kB of archives.
After this operation, 548 kB of additional disk space will be used.
Do you want to continue [Y/n]? y
```

```
stu@stu-VirtualBox:/mnt/shared$ sudo apt-get install libnet-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libnet1-dev' instead of 'libnet-dev'
The following extra packages will be installed:
  libnet1
The following NEW packages will be installed:
  libnet1 libnet1-dev
0 upgraded, 2 newly installed, 0 to remove and 375 not upgraded.
Need to get 166 kB of archives.
After this operation, 599 kB of additional disk space will be used.
Do you want to continue [Y/n]? y
Get:1 http://cn.archive.ubuntu.com/ubuntu/ precise/main libnet1 i386 1.1.4-2.1 [51.4 kB]
Get:2 http://cn.archive.ubuntu.com/ubuntu/ precise/main libnet1-dev i386 1.1.4-2.1 [114 kB]
Fetched 166 kB in 5min 18s (520 B/s)
Selecting previously unselected package libnet1.
```

图 21 安装程序示意图

然后需要进入 pktRxTX 文件夹输入 make 指令进行编译，再输入 sudo ./pktRxTx -m 2 这样就可以发包了。

附录 2：Windows 下监测网卡收发包

监测发包：

安装完 wireshark 之后，打开 wireshark，选择与板子相连的端口，一般情况下是本地连接。双击本地连接，如图 22 所示：

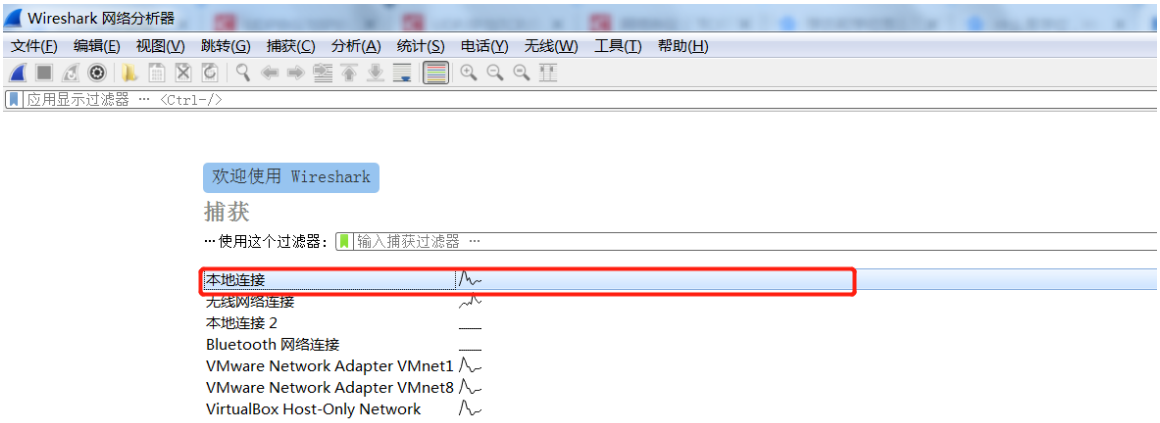

图 22 wireshark 使用示意图

在搜索框输入 udp，按 enter 键，则会显示捕捉到 udp 的报文。如图 23 所示，在测试发包时查看显示的 udp 报文即可。

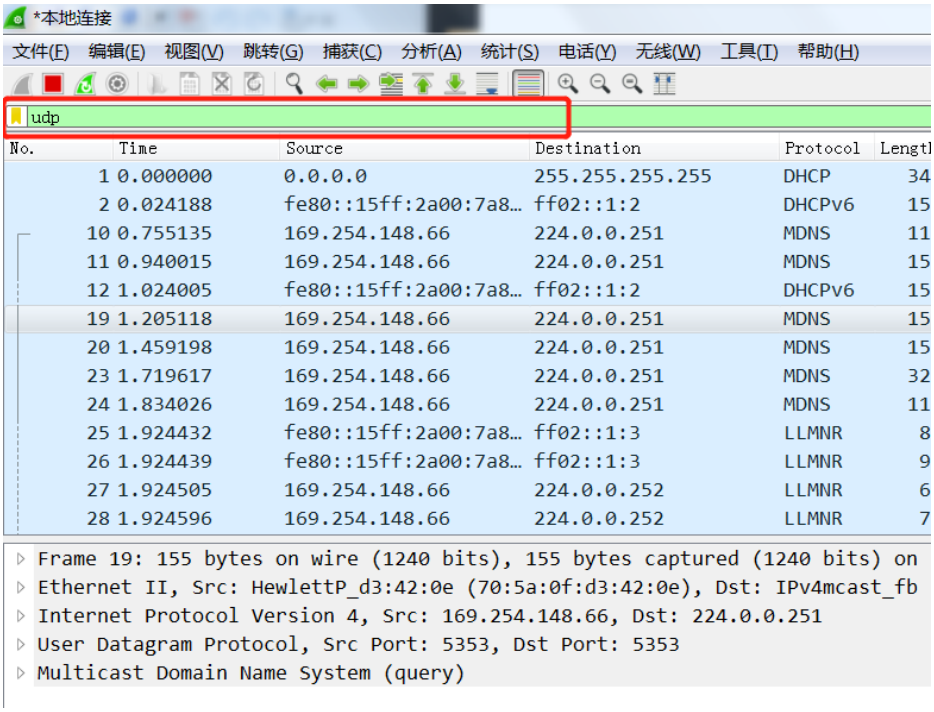

图 23 wireshark 使用示意图

收包详看《pktRxTx 使用说明》。

附录 3：在 mac/linux 中的收发包

在 mac 和 Linux 中我们可以使用 tcpdump 来对网卡发出的数据包进行截获。 tcpdump 可以将网络中传送的数据包的“头”完全截获下来提供分析。它支持针对网络层、协议、主机、网络或端口的过滤，并提供 and、or、not 等逻辑语句来帮助你去掉无用的信息。

在 mac 的 Linux 虚拟机上首先安装 tcpdump 包，在 terminal 中输入命令：yum install -y tcpdump

首先获得板子与虚拟机连接的网卡名，在我们电脑中显示的网卡端口名是 enx34298f709230，在你们的电脑中显示的名称可能不同，以你们电脑的显示为准。如图 23 所示，在 terminal 中输入命令:ifconfig

```
parallels@ubuntu:~$ ifconfig
enp0s5    Link encap:以太网  硬件地址 00:1c:42:fd:d5:c5
          inet 地址:10.211.55.7  广播:10.211.55.255  掩码:255.255.255.0
          inet6 地址: fe80::5074:b3aa:123e:a311/64 Scope:Link
          inet6 地址: fdb2:2c26:f4e4:0:e941:bd09:6171:f40f/64 Scope:Global
          inet6 地址: fdb2:2c26:f4e4:0:d77:42e:4d03:ba76/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:5949 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:2756 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:5476935 (5.4 MB)  发送字节:201424 (201.4 KB)

enx34298f709230 Link encap:以太网  硬件地址 34:29:8f:70:92:30
          inet6 地址: fe80::431e:cf7:df6f:7acb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:1280 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:312 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:1310720 (1.3 MB)  发送字节:46938 (46.9 KB)

lo        Link encap:本地环回
          inet 地址:127.0.0.1  掩码:255.0.0.0
          inet6 地址: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  跃点数:1
          接收数据包:589 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:589 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1
          接收字节:50556 (50.5 KB)  发送字节:50556 (50.5 KB)
```

图 23 查询网卡名示意图

如图 24 所示，在 terminal 中输入: **sudo tcpdump -i enx34298f709230 host 224.0.0.251** 命令就可以监听目的地址为 **224.0.0.251** 的数据包了。如果不加“-i enx34298f709230”是表示抓取所有的接口收到目的地址为 **224.0.0.251** 的数据包。

```
parallels@ubuntu:~$ sudo tcpdump -i enx34298f709230 host 224.0.0.251
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enx34298f709230, link-type EN10MB (Ethernet), capture size 262144 bytes
10:47:51.255255 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.255297 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.255302 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.255462 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.255466 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.255468 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.255756 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352491 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352506 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352509 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352639 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352644 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352645 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352900 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352908 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.352909 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353124 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353130 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353131 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353372 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353375 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353377 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353676 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353683 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353684 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353984 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353989 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.353990 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.354193 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.354199 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.354200 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
10:47:51.354408 IP 192.168.1.1.mdns > 224.0.0.251.mdns: 0+ [251a] [24064q] [35n] [35826au][|domain]
```

图 24 tcpdump 使用示意图

收包详看《pktRxTx 使用说明》。

附录 4：开发板与电脑连接方式

在网卡驱动的实验中，在运行网卡驱动时，需要大家用网线把开发板和电脑相连，具体如图所示：

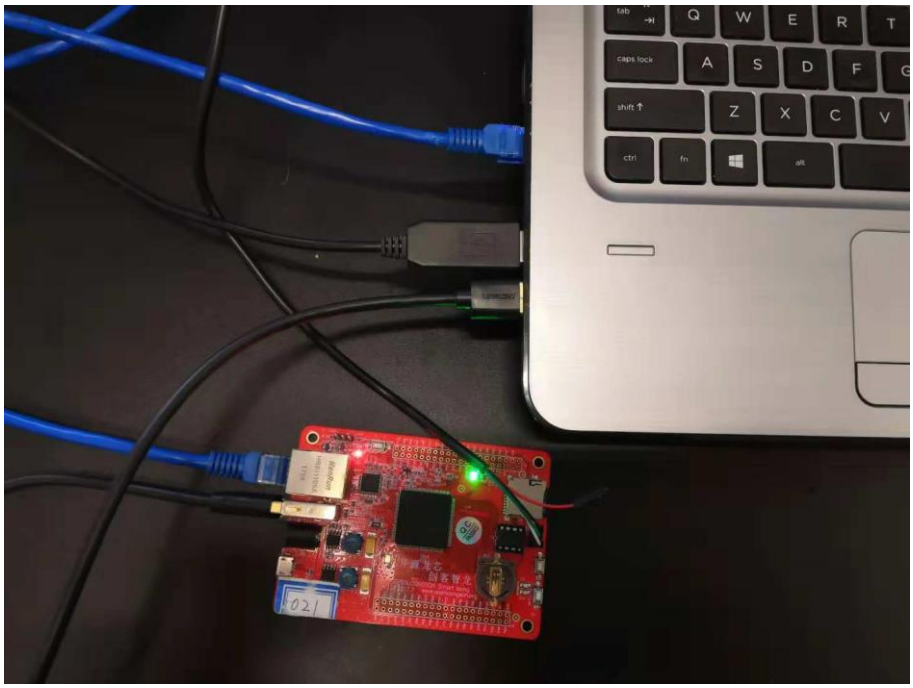

图 25 开发板与电脑连接示意图