国科大操作系统研讨课任务书

RISC-V 版本

版本 *1.0*

目录

Project 5
设备驱动

1 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调度和例外处理的能力。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 输入命令启动进程，本次任务需要实现的网卡驱动框架如图P5-1所示。

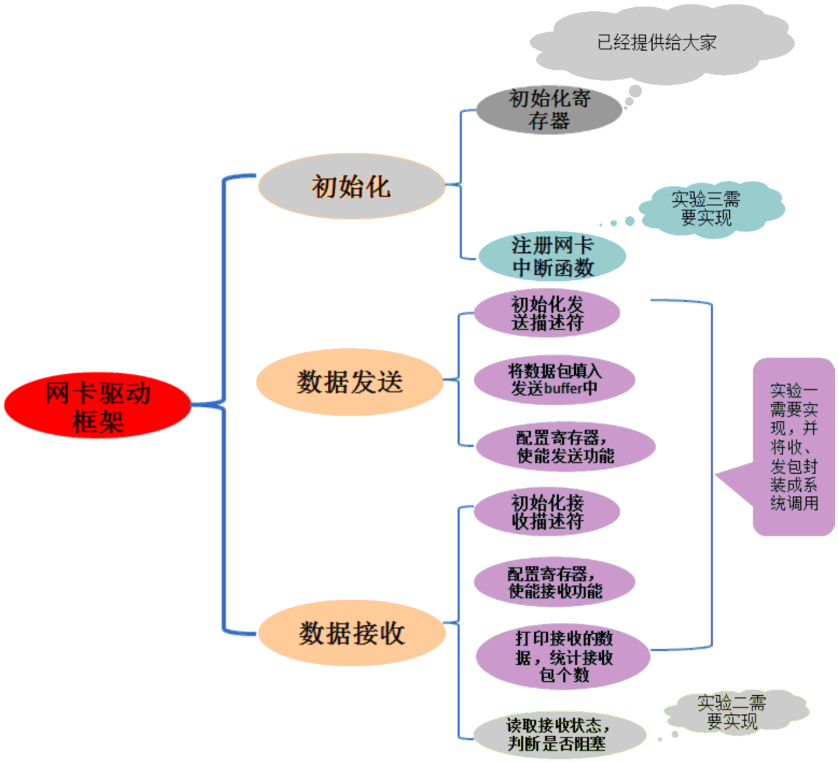

图 P5-1: 网卡驱动架构图

本次实验的各个任务如下：

任务一 实现轮询式的网卡收发包功能。

任务二 利用时钟中断实现由阻塞的网卡收发包功能。

任务三 添加网卡中断，实现有网卡中断的多进程收发包。

任务四 实现双端口监听功能，让两个监听不同端口的线程可以同时运行并接收到各自的网络包。

这四个任务中，S-core 的同学只需要完成任务一。A-core 的同学需要完成任务一和任务二，C-core 的同学需要完成全部四个任务。

和之前的实验一样，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 文件自行修改。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。

在前面的实验中，我们实践过了通过设置控制寄存器等方式来触发某些功能，比如虚存的功能。在本次实验中，由于 DMA 需要提供的信息很多，所以与以往直接写寄存器的方式不同，控制 DMA 需要将必要的信息以一个约定好的数据结构放置在内存中，再将相应的内存地址给到 DMA。DMA 会自动读取其中的内容来执行操作。这有一点像虚存实验中，我们将页表的结构建立好，CPU 会根据 satp 中存放的基地址自动查页表内容做地址翻译。网卡实验中，需要将 DMA 需要的信息以约定好的数据结构提供给 DMA，然后触发 DMA 的功能。DMA 就会自动读取数据结构中的内容，并执行发送/接收操作。在网卡实验中，我们需要构建的数据结构就是 DMA 的发送和接收描述符。

为了便于大家理解代码，这里对于网卡的一些原理做一些简述。板子上和网络相关的芯片有两个 **MAC** 和 **PHY**。MAC 实现的是 tcp/ip 中的数据链路层。PHY 是物理层。我们通过控制 MAC 控制器来实现收发包。MAC 控制器包含了一个 DMA，DMA 会自动把数据放入到指定的内存地址，或者将指定位置的内容发送出去。为此，我们需要设置一组描述符，描述我们希望发送的数据的起始地址、长度等信息，之后告知 DMA 描述符所在的位置，触发 DMA 执行发送功能。之后 DMA 就会自行读取描述符中的信息，并将其发送出去。接收数据的过程也是类似。大家可以这样想象，DMA 就像函数，描述符就像传给函数的参数。我们通过读写相应的控制寄存器，将描述符的地址告知 DMA。DMA 就会自动读取描述符的内容，并完成相关的功能。

除了本任务书之外，同学们可以参考手册中的内容，以使得操作系统代码的内容可以符合硬件的约定。网卡相关的寄存器手册可以参考《ug585-Zynq-7000-TRM》，推荐同学们在学习和调试的过程多多参考手册。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。

2 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 网卡驱动和 TCP/IP 协议栈关系

TCP/IP 协议栈分为五层，其中自顶向下的前三层（应用层、传输层和网络层）都是软件实现的概念（集成在操作系统软件中）。而后两层（数据链路层和物理层），则有相应的硬件控制器实现，通常链路层的芯片集成在处理器内部，即 MAC Controller 芯片。而物理层，则是有独立于处理器之外的 PHY 芯片实现。

MAC 芯片实现的是数据链路层的功能，如寻址、数据帧构建、数据差错检查等。

PHY 芯片则是定义了数据传送与接收所需要的电与光信号、线路状态、数据编码等，并向数据链路层提供标准的接口，这次实验我们不需要考虑 phy 相关寄存器的设置。

我们使用的开发板集成了 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 控制器不能被其他模块使用。MAC 控制器寄存器包括 MAC 寄存器部分和 DMA 寄存器部分。MAC 寄存器的起始地址是需要从 FDT 中读取。为了简化实现，我们提供了相应的 sbi 函数用于实现该功能。

```
/* 需要分别读取 slcr、ethernet 和 plic 的基地址，以及 plic 支持多少个外部 irq */
slcr_bade_addr = sbi_read_fdt(SLCR_BADE_ADDR);
ethernet_addr = sbi_read_fdt(ETHERNET_ADDR);
plic_addr = sbi_read_fdt(PLIC_ADDR);
nr_irqs = sbi_read_fdt(NR_IRQS);
```

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。

需要注意的是，上面读取到的都是物理地址。但由于我们开启了虚存，所以无法直接访问物理地址。在存在虚存的情况下，需要将物理地址映射到内核段的虚地址上，通过虚地址进行访问。在 Linux 中，这一操作叫做 ioremap。也就是将一个物理地址映射到某个空闲的虚地址上。在我们的小系统中，同样需要自己实现 ioremap 操作，以顺利通过虚地址访问对应的物理地址。相信大家在做完 Project4 的虚存部分之后，对这样的映射操作已经并不陌生了，请参见 start-code 中 ioremap 的接口定义，实现函数功能。需要提醒的就是，做完 ioremap 的操作之后需要刷一下 TLB，因为页表发生了变化。

本实验中需要同学们关注的主要是如图P5-2所示的这些寄存器。其功能也已经写在表格中。需要特别介绍的是 TXSR，RXSR，ISR 这几个寄存器会影响网卡中断，因此在任务三中才会用到。图中标示的几个位都需要在网卡中断处理完毕时清零，否则不会触发下一次网卡中断。而其中 TXSR 和 RXSR 的功能介绍中表明了这两个寄存器需要通过写 1 的方式清零，而不是直接写 0。

图中这些寄存器都是 ethernet 寄存器，访问的时候请使用 xemacps_config.BaseAddress 这个虚地址，具体见 start-code 里面这个地址的 ioremap 方式。而 plic（中断控制器）和 slcr（系统级控制寄存器）是在 start-code 的其他地方使用的，同学们不需要对其直接进行修改。有兴趣的同学可以看一下 start-code 了解其具体用法。

2.2 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。

Register	Offset	需要的位（括号里代表这个域是寄存器的第几位）	作用
XEMACPS_NWCTRL_OFFSET	0x0	XEMACPS_NWCTRL_STARTTX_MASK(9)	Start transmission - writing one to this bit starts transmission.
		XEMACPS_NWCTRL_TXEN_MASK(3)	Transmit enable - when set, it enables the GEM transmitter to send data
		XEMACPS_NWCTRL_TXEN_MASK(2)	Receive enable - when set, it enables the GEM to receive data.
XEMACPS_RXQBASE_OFFSET	0x18	rx_q_baseaddr	Receive buffer queue base address - written with the address of the start of the receive queue.
XEMACPS_TXQBASE_OFFSET	0x1c	tx_q_base_addr	Transmit buffer queue base address - written with the address of the start of the transmit queue.
XEMACPS_TXSR_OFFSET	0x14	XEMACPS_TXSR_TXCOMPL_MASK(5)	Transmit complete - set when a frame has been transmitted. Cleared by writing a one to this bit.
XEMACPS_RXSR_OFFSET	0x20	XEMACPS_RXSR_FRAMERX_MASK(1)	Frame received - one or more frames have been received and placed in memory. Cleared by writing a one to this bit.
XEMACPS_ISR_OFFSET	0x24	XEMACPS_IXR_TXCOMPL_MASK(7)	Transmit complete - set when a frame has been transmitted.
		XEMACPS_IXR_FRAMERX_MASK(1)	Receive complete - a frame has been stored in memory.

图 P5-2: 本次实验同学们需要用到的寄存器

2.3 DMA 描述符

DMA 描述符是 MAC 驱动和硬件的交互接口，记录了数据包的内存地址和传输状态。在此分别定义了发送描述符 (Tx Desciptor) 和接收描述符 (Rx Descriptor) 两种数据结构。两种描述符是以环式 (ring mode) 相连，以供 MAC 使用。

每一个 DMA 描述符包含两个部分，第一个部分主要记录数据的地址，第二部分主要记录描述符的状态，将在下文进行详细说明。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，同时保证使用小尾端。

DMA 接收描述符

如图P5-3所示，DMA 接收描述符包含两个 32 位的字。

word0 中 31:2 位表示该接收描述符对应的接收内容 buffer 的地址，第 1 位表示该描述符是否是最后一个描述符，第 0 位表示描述符的控制权归谁所有。

word1 表示接收描述符的状态，同学们在实验的过程中如果需要用到请仔细查看它们所表示的意思。

接收描述符以环式 (ring mode) 相连，MAC 寄存器中的 RXQbase 寄存器会指向当前使用的描述符，如图P5-4所示。

DMA 发送描述符

如图P5-5所示，DMA 接收描述符包含两个 32 位的字。

word0 表示该发送描述符对应的发送内容 buffer 的地址。

word1 包含对发送描述符的控制和描述符的状态，第 31 位表示发送是否完成，第 30 位表示该描述符是否是最后一个描述符，其他的位请同学们自己查看。

同样的，发送描述符也是以环式 (ring mode) 相连，与接收描述符类似。

Bit	Function
	Word 0
31:2	Address of beginning of buffer.
1	Wrap - marks last descriptor in receive buffer descriptor list.
0	Ownership - needs to be zero for the controller to write data to the receive buffer. The controller sets this to `1` once it has successfully written a frame to memory. Software must clear this bit before the buffer can be used again.

Bit	Function
	Word 1
31	Global all ones broadcast address detected.
30	Multicast hash match.
29	Unicast hash match.
28	Reserved.
27	Specific address register match found,. bit 25 and bit 26 indicate which specific address register causes the match.
26:25	Specific address register match. Encoded as follows: `00b`: Specific address register 1 match `01b`: Specific address register 2 match `10b`: Specific address register 3 match `11b`: Specific address register 4 match If more than one specific address is matched only one is indicated with priority 4 down to 1.
24	This bit has a different meaning depending on whether RX checksum offloading is enabled. • With RX checksum offloading disabled: (bit [24] clear in Network Configuration) Type ID register match found, bit [22] and bit [23] indicate which type ID register causes the match. • With RX checksum offloading enabled: (bit [24] set in Network Configuration) `0b`: The frame was not SNAP encoded and/or had a VLAN tag with the CFI bit set. `1b`: The frame was SNAP encoded and had either no VLAN tag or a VLAN tag with the CFI bit not set.
23:22	This bit has a different meaning depending on whether RX checksum offloading is enabled. With RX checksum offloading disabled: (bit [24] clear in Network Configuration) Type ID register match. Encoded as follows: `00b`: Type ID register 1 match `01b`: Type ID register 2 match `10b`: Type ID register 3 match `11b`: Type ID register 4 match If more than one Type ID is matched only one is indicated with priority 4 down to 1. With RX checksum offloading enabled: (bit [24] set in Network Configuration) `00b`: Neither the IP header checksum nor the TCP/UDP checksum was checked. `01b`: The IP header checksum was checked and was correct. Neither the TCP or UDP checksum was checked. `10b`: Both the IP header and TCP checksum were checked and were correct. `11b`: Both the IP header and UDP checksum were checked and were correct.
21	VLAN tag detected – type ID of `0x8100`. For packets incorporating the stacked VLAN processing feature, this bit is set if the second VLAN tag has a type ID of `0x8100`.
20	Priority tag detected – type ID of `0x8100` and null VLAN identifier. For packets incorporating the stacked VLAN processing feature, this bit is set if the second VLAN tag has a type ID of `0x8100` and a null VLAN identifier.
19:17	VLAN priority – only valid if bit [21] is set.
16	Canonical format indicator (CFI) bit – only valid if bit 21 is set.
15	End of frame – when set the buffer contains the end of a frame. If end of frame is not set, then the only valid status bit is start of frame (bit 14).
14	Start of frame – when set the buffer contains the start of a frame. If both bits 15 and 14 are set, the buffer contains a whole frame.
13	This bit has a different meaning depending on whether ignore FCS mode are enabled. This bit is zero if ignore FCS mode is disabled. With ignore FCS mode enabled: (bit [26] set in Network Configuration Register). This indicates per frame FCS status as follows: `0b`: Frame had good FCS. `1b`: Frame had bad FCS, but was copied to memory as ignore FCS enabled.
12:0	These bits represent the length of the received frame which might or might not include FCS depending on whether FCS discard mode is enabled. • With FCS discard mode disabled: (bit [17] clear in Network Configuration Register) Least significant 12-bits for length of frame including FCS. • With FCS discard mode enabled: (bit [17] set in Network Configuration Register) Least significant 12-bits for length of frame excluding FCS.

图 P5-3: DMA 接收描述符

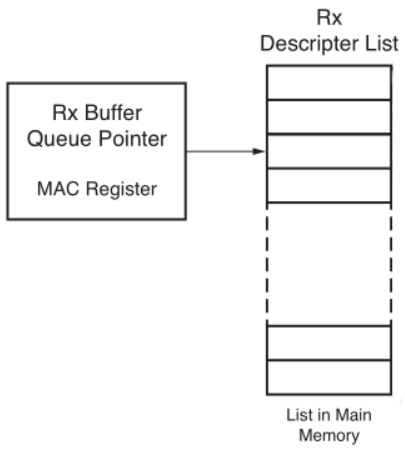

图 P5-4: DMA 接收描述符示意图

Bit	Function
	Word 0
31:0	Byte address of buffer.
	Word 1
31	Used – must be zero for the controller to read data to the transmit buffer. The controller sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected.
28	Always set to `0`.
27	Transmit frame corruption due to AHB error – set if an error occurs whilst midway through reading transmit frame from the AHB, including HRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and tx_er asserted).
26	Late collision, transmit error detected. Late collisions only force this status bit to be set in gigabit mode.
25:23	Reserved.
22:20	Transmit IP/TCP/UDP checksum generation offload errors: • `000b`: No Error. • `001b`: The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it. • `010b`: The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it. • `011b`: The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6. • `100b`: The Packet was not identified as VLAN, SNAP or IP. • `101b`: Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted. • `110b`: Packet type detected was not TCP or UDP. TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted. • `111b`: A premature end of packet was detected and the TCP/UDP checksum could not be generated.
19:17	Reserved.
16	No CRC to be appended by MAC. When set this implies that the data in the buffers already contains a valid CRC and hence no CRC or padding is to be appended to the current frame by the MAC. This control bit must be set for the first buffer in a frame and is ignored for the subsequent buffers of a frame. Note that this bit must be clear when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution does not occur.
15	Last buffer, when set this bit indicates that the last buffer in the current frame has been reached.
14	Reserved.
13:0	Length of buffer.

图 P5-5: DMA 发送描述符

系统初始化的时候需要先初始化好描述符的地址空间和数据结构。在 main 函数调用的网卡初始化函数 EmacPsInit 中会调用 EmacPsSetupBD 函数，请同学们将描述符的初始化流程实现在这个函数中。start-code 中对应的代码也已经写好注释标注了这一功能需求。

两个描述符的补充说明

这次实验中，大家主要需要关注发送描述符和接收描述符是否写的正确。主要需要注意: Tx 描述符需要设置 buffer 地址和 length，清空 used 位，设置 last 位。last 位代表当前的描述符中的 buffer 是当前帧的最后一个 buffer，因为一个帧也许需要多个描述符描述。但在我们的实验中，一个帧只需要一个描述符来描述，没有更复杂的情况。Rx 描述符需要设置 buffer 地址和 wrap 位 (wrap 代表描述符列表中的最后一个描述符，所以 wrap 位只需要在最后一个 Rx 描述符上设就可以)。当收到包以后，硬件会自动将收到的包的长度写到 Rx 描述符的对应位置。wrap 位的作用是: 网卡的发送或接收使能了以后，网卡 dma 就会顺着 RXQBase 或 TXQBase 每收到一个包之后当前描述符的指针就会往后偏移一个，直到遇到某个描述符设置了 wrap 位，就会重新指回最开始 RXQBase 或 TXQBase 设置的位置重新来。

3 实验内容

3.1 任务一：实现轮询式的网卡收发包功能

实验要求

本次实验需要完成: 不注册网卡中断处理函数，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。

本次实验需要使用 wireshark 或 tcpdump 软件进行抓包，查看网卡有没有把数据包发送出来，如图P5-6所示，为发包成功的参考图。

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P5-7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 1 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，如图所示。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》。

网卡收包成功的打印效果图如图P5-8所示:

实验步骤

1. 完善 XEmacPs_SetMacAddress () 函数，该函数用于设置 mac 地址;
2. 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 ();
3. 分别实现 do_net_send () 和 do_net_recv ()，实现网卡的发送和接收数据包;

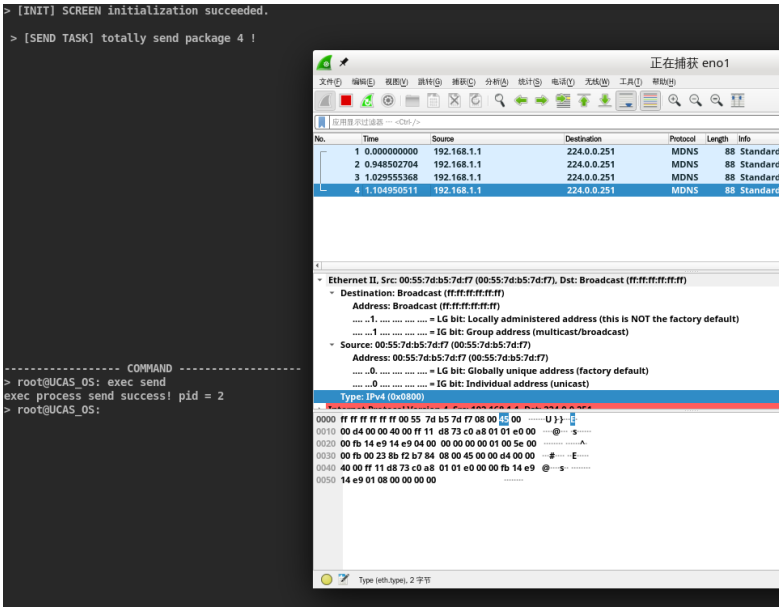

图 P5-6: 发包效果示意图

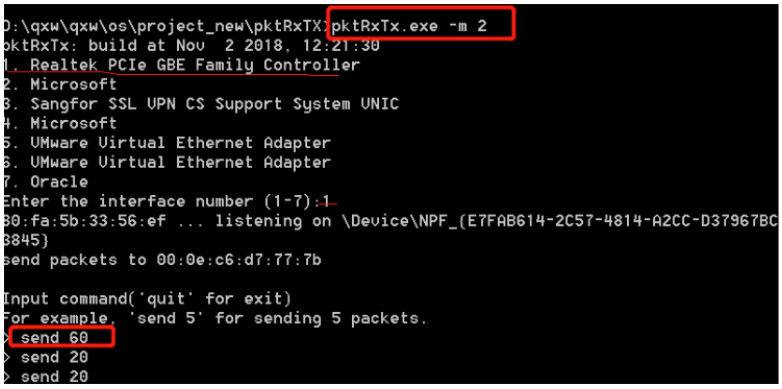

图 P5-7: pktRxTx 小工具

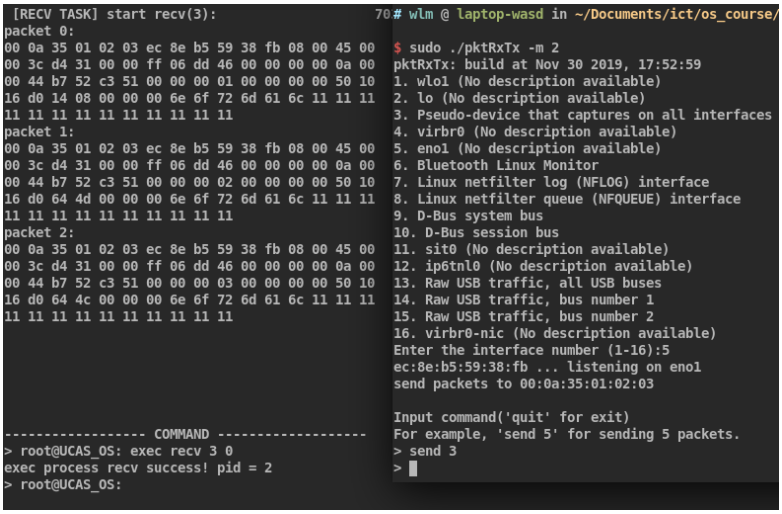

图 P5-8: 收包效果示意图

4. 完善 EmacPsSend () 函数，该函数用于设置发送描述符，包括但不限于建立发送描述符环、设置发送描述符的 word0、word1 中 used、wrap、length 位等；

5. 完善 EmacPsRecv () 函数，该函数用于设置接收描述符；

6. 完善 EmacPsWaitSend () 函数，该函数用于在开启发送后，寻找硬件写好的描述符，并取回需要的内容；

7. 完善 EmacPsWaitRecv () 函数，该函数用于在开启接收后，寻找硬件写好的描述符，并取回需要的内容。

注意事项

如果前面的 project 只完成了 S-core 的同学，这里可能会遇到之前没有实现系统调用的问题，那么这里可以直接只用 do_net_recv () 和 do_net_send () 函数。

想详细了解网卡驱动相关寄存器或者在实验中遇到了问题可以详细阅读《ug585-Zynq-7000-TRM》[1] 手册中的内容。

需要注意的是，网卡是不认识虚地址的。所以，请注意所有给网卡看的地址都是**物理地址**。另外，由于 DMA 和 RISC-V 处理器都需要读写描述符以及接收和发送缓冲区，所以必须保证读写的 cache 一致性。确保 DMA 设备和 RISC-V 处理器都能看到双方的修改。这需要借助于 RISC-V 的 fence 指令。在 start code 中给了一段写好的嵌入式汇编的 fence 指令的示例：

```
#define RISCV_FENCE(p, s) \
        __asm__ __volatile__ ("fence " #p "," #s : : : "memory")

/* These barriers need to enforce ordering on both devices or memory. */
#define mb()            RISCV_FENCE(iorw,iorw)
#define rmb()           RISCV_FENCE(ir,ir)
#define wmb()           RISCV_FENCE(ow,ow)
```

这里特别说明一下这几个函数的关系，以发送为例：用户程序会调用 sys_net_send()，之后通过系统调用的相关机制转到内核中的 do_net_send() 进行实际的处理。do_net_send() 函数需要调用驱动中的 EmacPsSend() 设置描述符并触发 DMA 将包发送出去。之后，调用 EmacPsWaitSend() 确保发送完成（通过轮询或等待中断的方式）。在 start code 中将完整的 send 过程分为了 send 和 waitsend 两个部分是为了允许内核实现在发送包以后调度其他进程执行的功能，避免整个内核因为等待发送完成而阻塞。当然也可以在后面的任务中再实现这些功能。

另外，接收与发送的网络包数目需要支持大于描述符的数量，以体现描述符的组织是环式相连的。请同学们设置的描述符数量至少等于 32。

3.2 任务二：利用时钟中断实现有阻塞的网卡收发包功能

本任务为 A-core 同学需要完成的任务。

实验要求

本实验与任务一的不同点就是: 在网卡的接收进程中, 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测试程序给定的数目时，接收包进程被阻塞，当有数据包到达网卡时，接收包进程被唤醒，开始打印接收的数据。唤醒的位置判断在时钟中断中。发送进程也要做类似的处理，设置好发送的描述符之后发送进程进入阻塞，在时钟中断里判断发送是否完成，如果完成则唤醒发送进程。

实验步骤

1. 不同于任务一中在轮询下检查状态寄存器的值，在任务二里需要在时钟中断里判断 TXSR 和 RXSR 的值。

3.3 任务三：有网卡中断的收发包

本实验为 C-core 同学需要完成的任务。

实验要求

本实验与任务二的不同点就是：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或发送进程。同样地, 在网卡的接收包进程中, 当没有数据包到达网卡或到达的数据包不满测试程序给定的数目时，接收包进程被阻塞，当有数据包到达网卡时，接收包进程被唤醒，开始打印接收的数据。唤醒的位置在网卡中断内。发送数据包也要实现类似的网卡中断流程。

实验步骤

在 RISC-V 架构下 CPU 和设备之间存在着一个中断控制器（Platform-Level Interrupt Controller），当外部设备发生中断时，它负责和 CPU 交互。

任务三这个处理的接口已经帮同学们实现好了(在发生外部中断时调用 plic_handle_irq, 中断原因依然需要判断 scause 寄存器, 网卡中断对应 supervisor external interrupt)，需要同学们完成的时候在 plic_irq_handle 中调用的 handle_irq 函数，函数参数为上下文寄存器和硬件中断号，当然这里由于我们只实现网卡中断，所以硬件中断号不需要做判断；handle_irq 的最后需要调用 plic_irq_eoi 完成外部中断处理，plic_irq_eoi 这个函数已经在 start-code 中实现好了。

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。初始化的函数 EmacPsInit 中需要调用 XEmacPs_IntEnable 函数打开网卡中断，具体请见 start-code 中的注释。

3.4 实验测试程序

我们给同学们提供了两个测试程序，test/send.c 和 test/recv.c，分别用于测试发包和收包，使用它们的方法是在 shell 中使用 exec 指令分别启动。

对于 send 测试程序，在启动时有一个命令行参数，它表明是否开启中断模式，在实现了任务三后需要开启中断模式测试。

对于 recv 测试程序，在启动时有两个命令行参数，第一个参数表示收包的个数，第二个参数表示是否开启中断模式，在进行不同测试时要相应设置成正确的参数值。

例如，执行

```
# 接收 3 个包，轮询模式
exec recv 3 0
# 接收 2 个包，中断模式
exec recv 2 1
# 发送数据包，轮询模式
exec send 0
# 发送数据包，中断模式
exec send 1
```

3.5 任务四：双端口监听

本任务依然是 C-core 的同学需要完成的任务。同学们需要自行实现测试程序实现如下功能:

该测试程序会启动两个线程，分别接收两个端口，即在现有的接收接口上增加一个端口参数。内核在网卡中断的时候根据包的内容判断该包发给哪个接口，然后唤醒对应的监听线程。包的格式为 **以太网报头（14 字节）+IP 报头（20 字节）+TCP 报头（20 字节）+ 数据 payload**，端口部分是 TCP 报头的 dport 字段（TCP 报头开始的第 3、4 字节，uint16_t 类型），设置了两个定死的值 50001 和 58688。这两个端口值没有特殊的含义，只是随机的生成了这两个数字。我们这里实现设备驱动并不需要关心网络包的具体含义，仅满足网络包的格式即可，TCP/IP 协议的相关知识请大家自行查阅计算机网络的知识。在使用 pktRxTx -m 1 命令之后，pktRxTx 程序就会随机发出这两种端口号的数据包。要求两个线程都能接收到数据，不能出现某个线程一段时间饿死的情况。

4 实验须知

4.1 本机网络的注意事项

1. 当使用网线连接开发板和本机后，IPv4 和 IPv6 协议会向连接处发包，在监测网卡时这些包会被监测到。如果不想受到这些包的干扰，请自行关闭 IPv4 和 IPv6 协议。

2. 本机的网卡连接速度和双工模式必须设置为自动检测，默认情况下就是这个设置，如果需要修改，在 WINDOWS 下打开以太网配置中高级-连接速度和双工模式，如图P5-9所示。

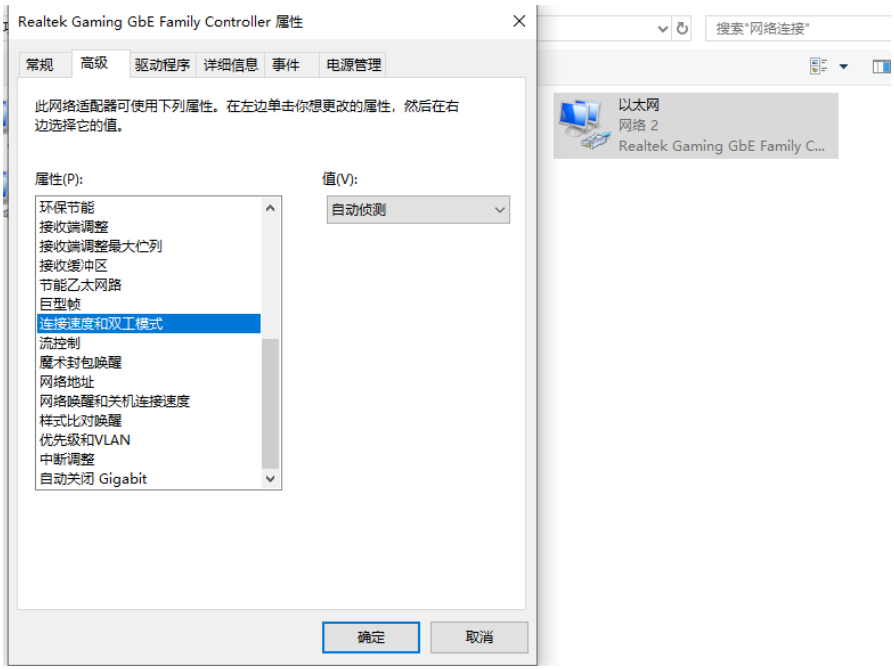

图 P5-9: windows 下 aotudetect 模式设置

4.2 使用 QEMU 调试网络功能需要注意的事项

由于 qemu 能模拟的网卡中不包含实验板上的网卡，Project5 中采取的方法是使用 QEMU 模拟 intel e1000 网卡，将同学们对实验板网卡的操作转换为 e1000 的操作。接下来具体说明使用 QEMU 模拟 e1000 网卡的步骤：1. 修改 QEMU 目录/etc 下的 qemu-ifup，将 IFNAME=enp0s3 中等号后的部分替换为使用机器 ifconfig 之后显示的内容，即 ifconfig 命令打印出来的网卡名称。

2. 执行脚本 e1000.sh(如果提示 script 不能执行注意要给 qemu-ifup 和 qemu-ifdown 执行权限)

3. 由于在 QEMU 中模拟的板卡和实验实际用的不同，所以需要对自己实现的操作系统中 init/main.c 的内容作出一些修改：在对网卡 ioremap 时，需要分配 9 页内容，并把 xemacps_config.BaseAddress 设为最后一页的起始地址，具体修改方式如图。

```
    xemacps_config.BaseAddress =
-       (uintptr_t)ioremap((uint64_t)ethernet_addr, NORMAL_PAGE_SIZE);
+       (uintptr_t)ioremap((uint64_t)ethernet_addr, 9 * NORMAL_PAGE_SIZE);
+     xemacps_config.BaseAddress += 0x8000;
```

图 P5-10: ioremap 需要修改的语句

4. 由于 QEMU 中模拟的 descriptor 有限，测试的时候最好收发包各不超过 6 个。

5. 要配合 QEMU 使用 tcpdump 的话，指令为：sudo tcpdump -i tap0。

4.3 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P5-11所示:

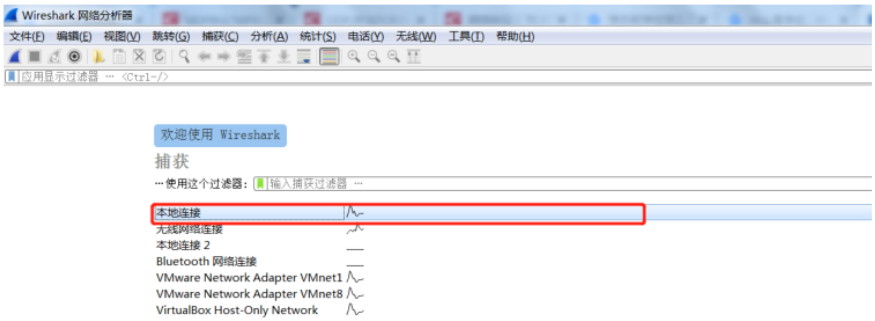

图 P5-11: wireshark 界面

在搜索框输入 udp，按 enter 键，则会显示捕捉到 udp 的报文。如图P5-12所示，在测试发包时查看显示的 udp 报文即可。

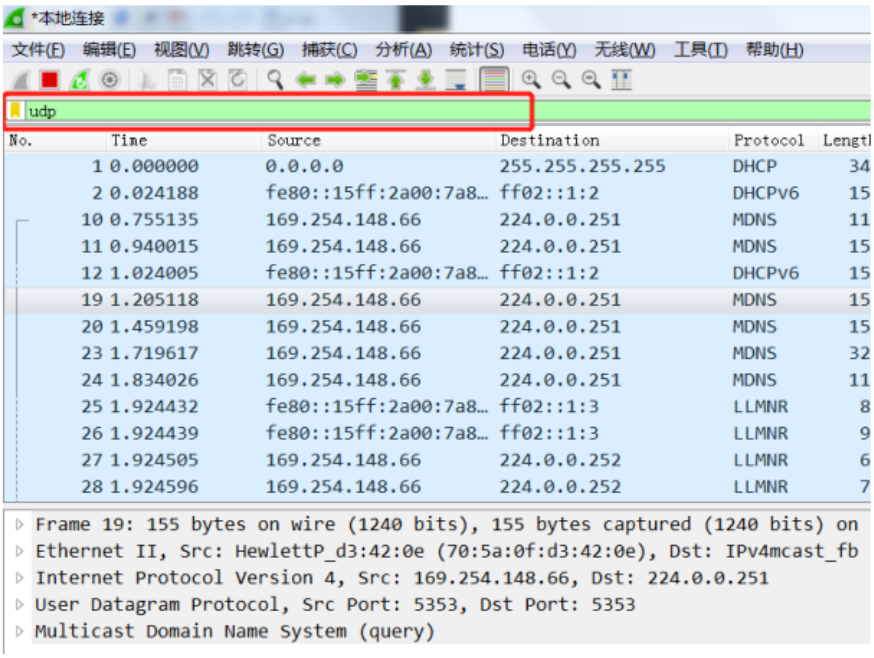

图 P5-12: wireshark 界面（续）

收包详看《pktRxTx 使用说明》。

4.4 mac/linux 中收发包

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P5-13所示，在 terminal 中输入命令:ifconfig

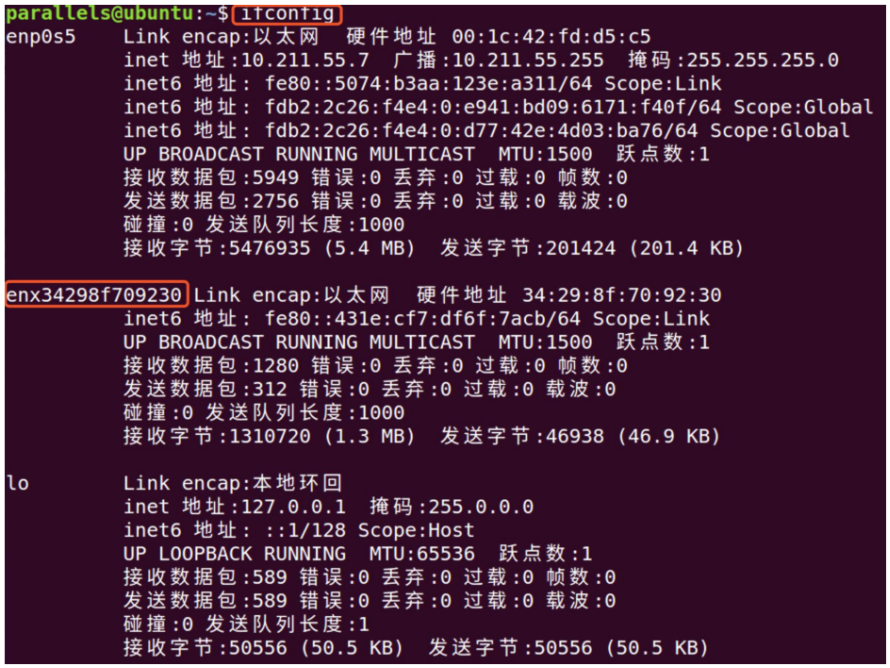

图 P5-13: 获得网卡名

如图P5-14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。

收包详看《pktRxTx 使用说明》。

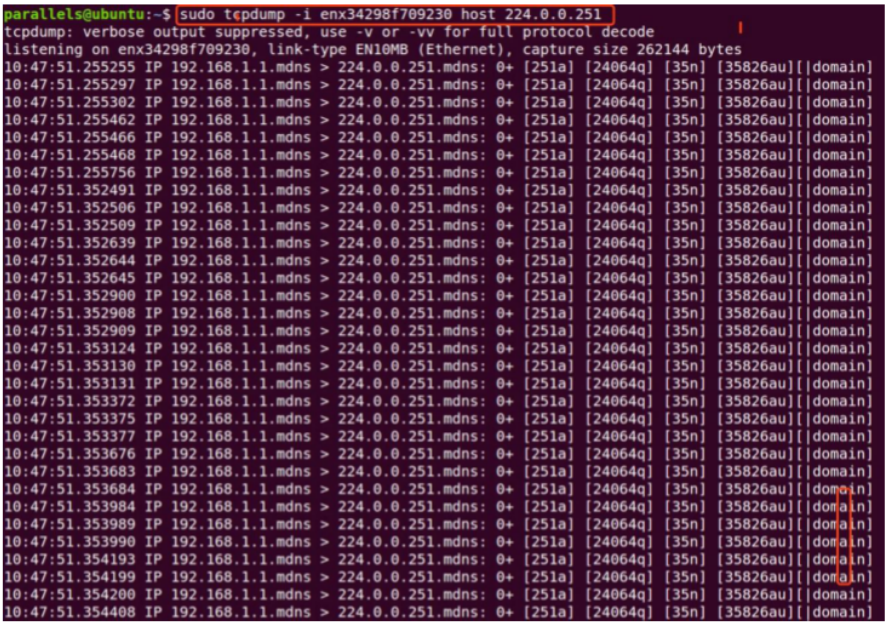

图 P5-14: tcpdump

参考文献

[1] UG585, “Zynq-7000 soc.” https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf, 2018. [Online; accessed 20-September-2021].