PRJ 5

一. 环境配置

先使用如下命令将可执行文件传输到你的虚拟机：

```
scp -r -P 22 ./pktRxTx  stu@192.168.32.134:~/lmy/
```

注意 `./pktRxTx` 修改为你的pktRxTx路径， `192.168.32.134` 换成你的服务器地址， `22` 换成你的端口， `~/lmy/` 是你想要放在的虚拟机文件夹。

加可执行权限：

```
chmod +x [虚拟机中你的pktRxTx路径]
```

1.下载libpcap和libnet

省流：先用如下命令下载安装包，后续直接按下面链接配置

```
wget https://www.tcpdump.org/release/libpcap-1.10.1.tar.gz
wget https://udomain.dl.sourceforge.net/project/libnet-dev/libnet-1.2-rc3.tar.gz
```

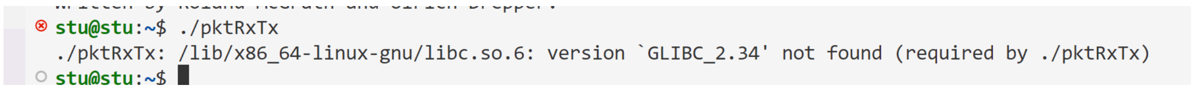
(9条消息) libnet安装配置_AAquiloo的博客-CSDN博客

(9条消息) Linux下libpcap的安装_HeroKern的博客-CSDN博客

2.安装GLIBC_2.34（废）

step 1

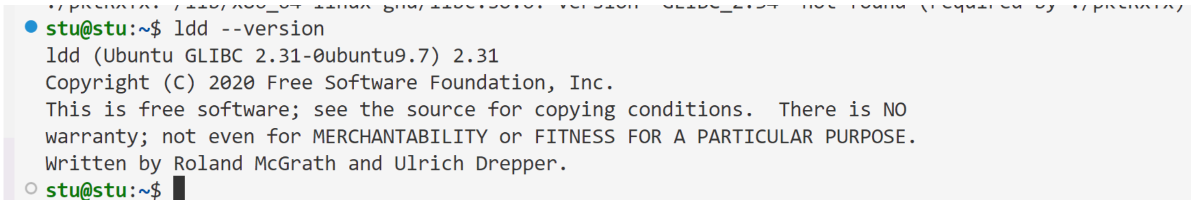
如果你也报了这个错：



```
stu@stu:~$ ./pktRxTx
./pktRxTx: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' not found (required by ./pktRxTx)
stu@stu:~$
```

查看本机glibc的版本：

```
ldd --version
```



```
stu@stu:~$ ldd --version
ldd (Ubuntu GLIBC 2.31-0ubuntu9.7) 2.31
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
stu@stu:~$
```

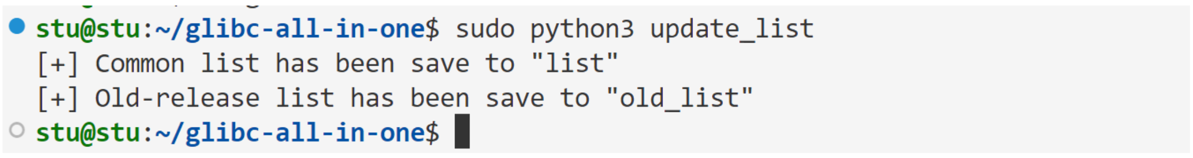
只能下载新版了……

```
sudo git clone https://github.com/matrix1001/glibc-all-in-one.git
cd glibc-all-in-one/
```

输入：

```
sudo python3 update_list
```

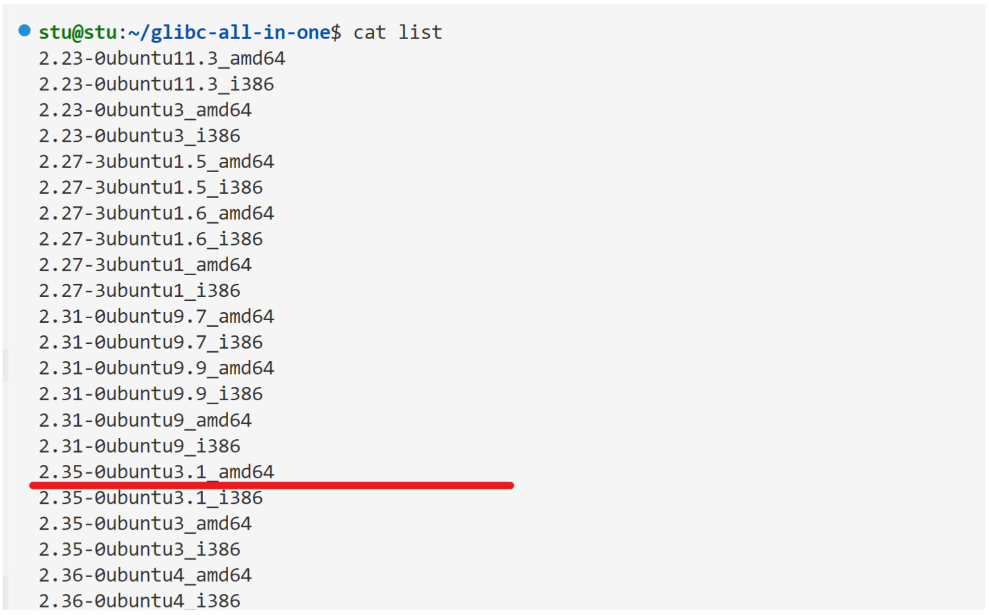
你应该会看到：



```
stu@stu:~/glibc-all-in-one$ sudo python3 update_list
[+] Common list has been save to "list"
[+] Old-release list has been save to "old_list"
stu@stu:~/glibc-all-in-one$ 
```

```
cat list
```

查看可下载的glibc版本：



```
stu@stu:~/glibc-all-in-one$ cat list
2.23-0ubuntu11.3_amd64
2.23-0ubuntu11.3_i386
2.23-0ubuntu3_amd64
2.23-0ubuntu3_i386
2.27-3ubuntu1.5_amd64
2.27-3ubuntu1.5_i386
2.27-3ubuntu1.6_amd64
2.27-3ubuntu1.6_i386
2.27-3ubuntu1_amd64
2.27-3ubuntu1_i386
2.31-0ubuntu9.7_amd64
2.31-0ubuntu9.7_i386
2.31-0ubuntu9.9_amd64
2.31-0ubuntu9.9_i386
2.31-0ubuntu9_amd64
2.31-0ubuntu9_i386
2.35-0ubuntu3.1_amd64
2.35-0ubuntu3.1_i386
2.35-0ubuntu3_amd64
2.35-0ubuntu3_i386
2.36-0ubuntu4_amd64
2.36-0ubuntu4_i386
```

好死不死没有2.34，，下一个稍微高一个版本的吧……

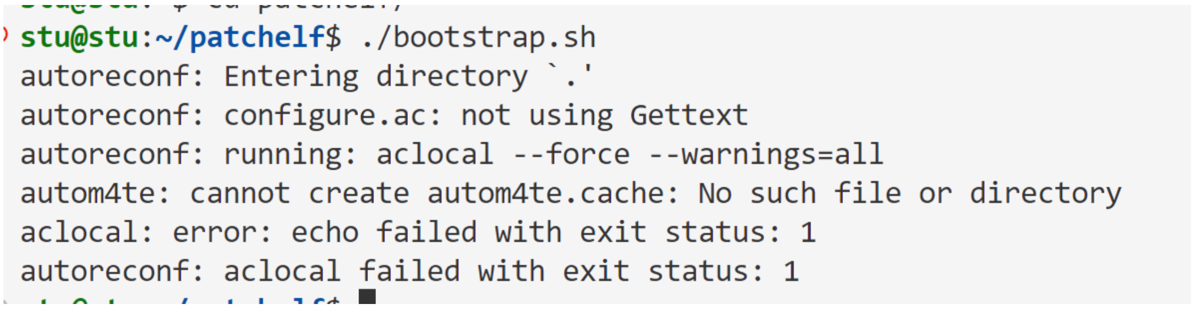
```
 sudo ./download 2.35-0ubuntu3_amd64
```

接下来看教程发现要使用 `patchelf` ，递归安装一波：

step 2：下载 patchelf

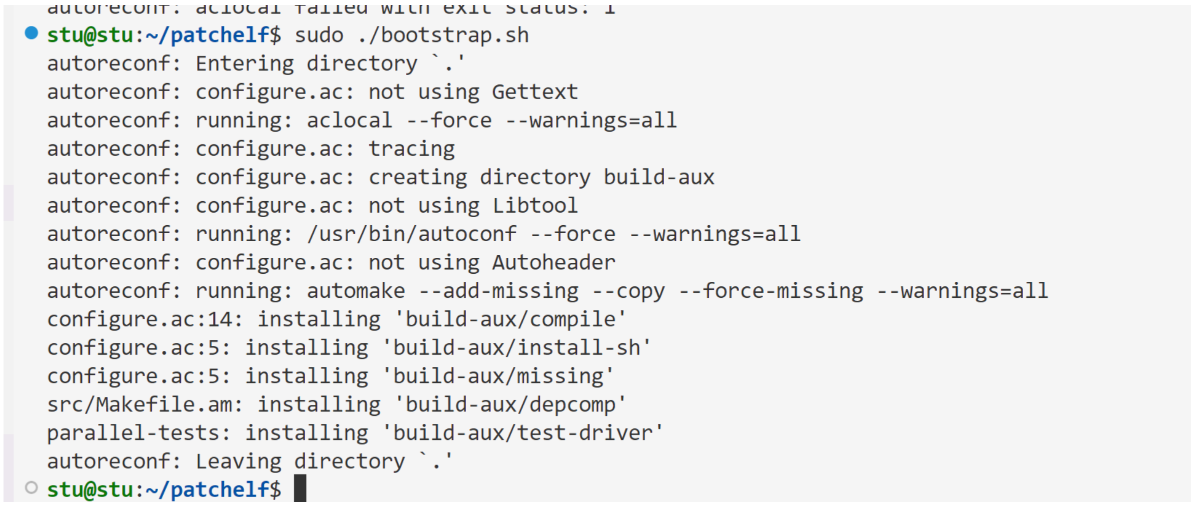
```
sudo git clone https://github.com/NixOS/patchelf.git
cd patchelf
./bootstrap.sh
```

如果报错：



```
stu@stu:~/patchelf$ ./bootstrap.sh
autoreconf: Entering directory `.'
autoreconf: configure.ac: not using Gettext
autoreconf: running: aclocal --force --warnings=all
autom4te: cannot create autom4te.cache: No such file or directory
aclocal: error: echo failed with exit status: 1
autoreconf: aclocal failed with exit status: 1
```

加上 `sudo` 试试，此时你应该看到：

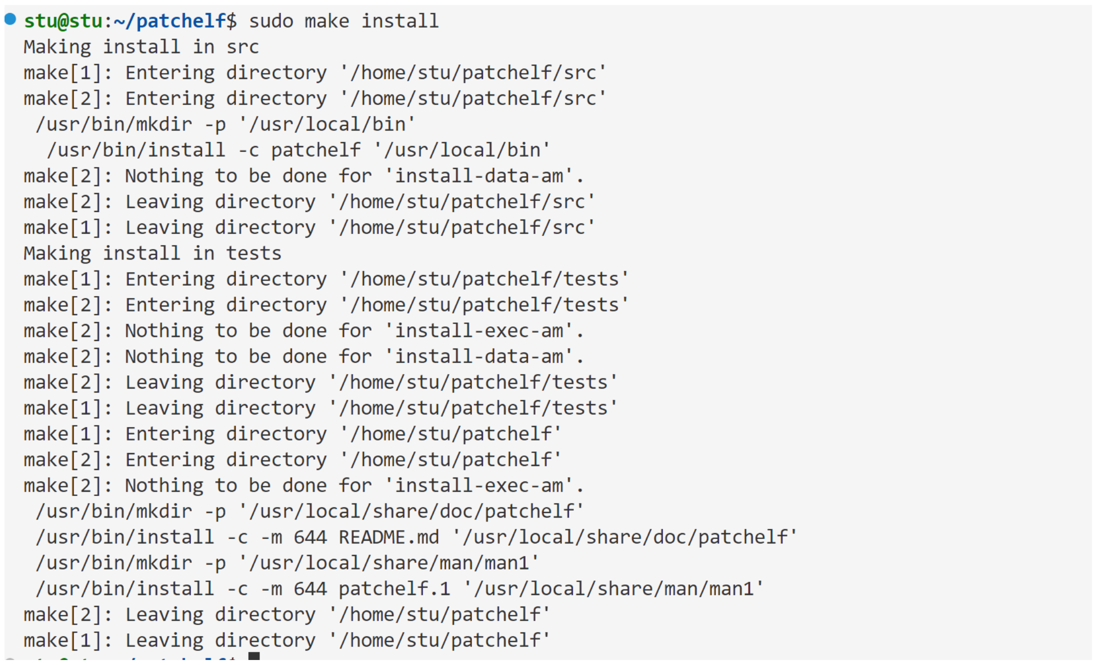


```
stu@stu:~/patchelf$ sudo ./bootstrap.sh
autoreconf: Entering directory `.'
autoreconf: configure.ac: not using Gettext
autoreconf: running: aclocal --force --warnings=all
autoreconf: configure.ac: tracing
autoreconf: configure.ac: creating directory build-aux
autoreconf: configure.ac: not using Libtool
autoreconf: running: /usr/bin/autoconf --force --warnings=all
autoreconf: configure.ac: not using Autoheader
autoreconf: running: automake --add-missing --copy --force-missing --warnings=all
configure.ac:14: installing 'build-aux/compile'
configure.ac:5: installing 'build-aux/install-sh'
configure.ac:5: installing 'build-aux/missing'
src/Makefile.am: installing 'build-aux/depcomp'
parallel-tests: installing 'build-aux/test-driver'
autoreconf: Leaving directory `.'
stu@stu:~/patchelf$
```

以下命令都要加sudo，否则权限不够：

```
sudo ./configure
sudo make
sudo make check
sudo make install
```

此时你应该看到：



```
stu@stu:~/patchelf$ sudo make install
Making install in src
make[1]: Entering directory '/home/stu/patchelf/src'
make[2]: Entering directory '/home/stu/patchelf/src'
 /usr/bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c patchelf '/usr/local/bin'
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/stu/patchelf/src'
make[1]: Leaving directory '/home/stu/patchelf/src'
Making install in tests
make[1]: Entering directory '/home/stu/patchelf/tests'
make[2]: Entering directory '/home/stu/patchelf/tests'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/stu/patchelf/tests'
make[1]: Leaving directory '/home/stu/patchelf/tests'
make[1]: Entering directory '/home/stu/patchelf'
make[2]: Entering directory '/home/stu/patchelf'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/share/doc/patchelf'
 /usr/bin/install -c -m 644 README.md '/usr/local/share/doc/patchelf'
 /usr/bin/mkdir -p '/usr/local/share/man/man1'
 /usr/bin/install -c -m 644 patchelf.1 '/usr/local/share/man/man1'
make[2]: Leaving directory '/home/stu/patchelf'
make[1]: Leaving directory '/home/stu/patchelf'
```

```
patchelf --version
```

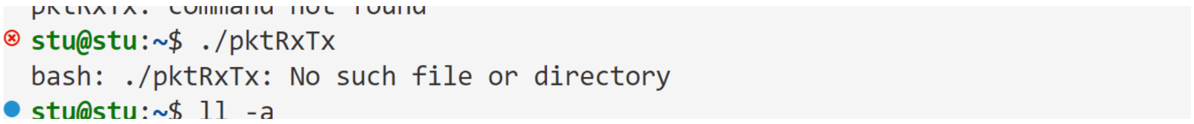
查看是否安装成功。

step 3：让elf文件使用特定的glibc版本去运行

```
patchelf --set-interpreter ~/glibc-all-in-one/libs/2.35-0ubuntu3_amd64/ld-linux-x86-64.so.2  pktRxTx

patchelf --set-rpath  ~/glibc-all-in-one/libs/2.35-0ubuntu3_amd64/ pktRxTx
```

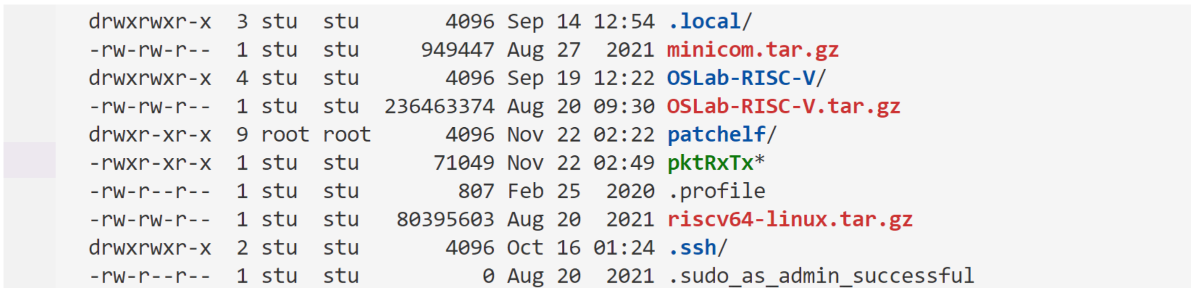
第一个path请改为你自己下载 `glibc-all-in-one` 目录，第二个path改为你的 `pktRxTx` 可执行文件路径。

执行：



```
pktRxTx: command not found
stu@stu:~$ ./pktRxTx
bash: ./pktRxTx: No such file or directory
stu@stu:~$ ll -a
```

如果你也出现了如上case， `ll -a` 查看一下：



```
drwxrwxr-x  3 stu  stu       4096 Sep 14 12:54 .local/
-rw-rw-r--  1 stu  stu     949447 Aug 27  2021 minicom.tar.gz
drwxrwxr-x  4 stu  stu       4096 Sep 19 12:22 OSLab-RISC-V/
-rw-rw-r--  1 stu  stu  236463374 Aug 20 09:30 OSLab-RISC-V.tar.gz
drwxr-xr-x  9 root root      4096 Nov 22 02:22 patchelf/
-rwxr-xr-x  1 stu  stu      71049 Nov 22 02:49 pktRxTx*
-rw-r--r--  1 stu  stu        807 Feb 25  2020 .profile
-rw-rw-r--  1 stu  stu   80395603 Aug 20  2021 riscv64-linux.tar.gz
drwxrwxr-x  2 stu  stu       4096 Oct 16 01:24 .ssh/
-rw-r--r--  1 stu  stu          0 Aug 20  2021 .sudo_as_admin_successful
```

带星号表示其可执行。咋回事捏，，，明明文件也在，，猜测是glibc2.25下载时失败？到下载目录 `~/glibc-all-in-one/libs/2.35-0ubuntu3_amd64/` 下一看，发现除了一个隐藏文件夹 `.debug` 其他啥也没有......

弄不明白就去问助教，，喜提不用安装新版本glibc的好消息！！

助教新编译了一个，直接 `sudo ./pktRxTx -m` 指定模式就可了√

3. 用MinGW编译测试程序

4. VMware ssh在运行时失效

先看看相关代码：

```
IFNAME=ens32

set -x
if [ -n "$1" ];then
#create bridge, add physical interface to bridge
        ip link set $IFNAME down
        ip link add br0 type bridge
        ip link set br0 up
        ip link set $IFNAME master br0
        ip link set $IFNAME up

#add tap device to bridge
                ip link set $1 up
                sleep 0.5s
                ip link set $1 master br0

#config ip fro bridge
                pkill -9 dhclient
        sleep 5
                dhclient -v br0

                exit 0
```

怀疑对象1： `dhclient` 被kill了，这个进程似乎在动态获取or释放IP地址？不懂......救命......注释掉没有用。

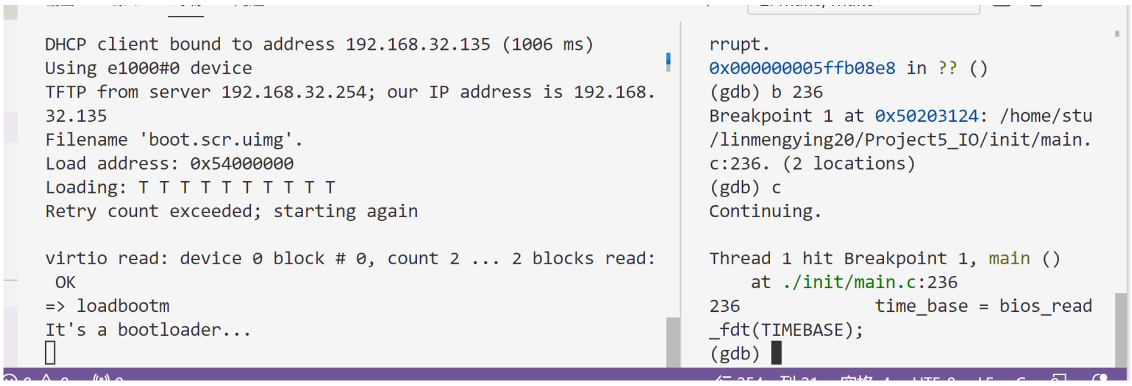
怀疑对象2：得知Vbox就没这种事，猜测是因为ssh连接时使用了虚拟机的本机ip，然后在/qemu/etc中的qemu-ifup和ifdown中将ens32对应的ip暂时置为down后再拉高，可能会有影响？

- 应对方式1：查看vscode连接远程机的log：

```
...
[18:46:40.383] Tunneled 34991 to local port 9501
[18:46:40.383] Resolved "ssh-remote+oslab" to "127.0.0.1:9501"
[18:46:40.394] ------




[18:46:40.438] [Forwarding server 9501] Got connection 1
[18:46:40.467] [Forwarding server 9501] Got connection 2

```

发现原本的ip似乎被解析为 `127.0.0.1:9501` ，尝试用这个地址和port连接，不成功……（听说有的同学是可以的）

- 应对方式2：反正咱啥也不懂，注释总会了，把qemu-ifdown和qemu-ifup里对ens32的down和up操作都注释掉。
  成了：



```
DHCP client bound to address 192.168.32.135 (1006 ms)
Using e1000#0 device
TFTP from server 192.168.32.254; our IP address is 192.168.
32.135
Filename 'boot.scr.uimg'.
Load address: 0x54000000
Loading: T T T T T T T T T T
Retry count exceeded; starting again

virtio read: device 0 block # 0, count 2 ... 2 blocks read:
 OK
=> loadbootm
It's a bootloader...
```

```
rrupt.
0x000000005ffb08e8 in ?? ()
(gdb) b 236
Breakpoint 1 at 0x50203124: /home/stu
/linmengying20/Project5_IO/init/main.
c:236. (2 locations)
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, main ()
    at ./init/main.c:236
236             time_base = bios_read
_fdt(TIMEBASE);
(gdb)
```

5. VMware虚拟机实验过程中ip总是跳变

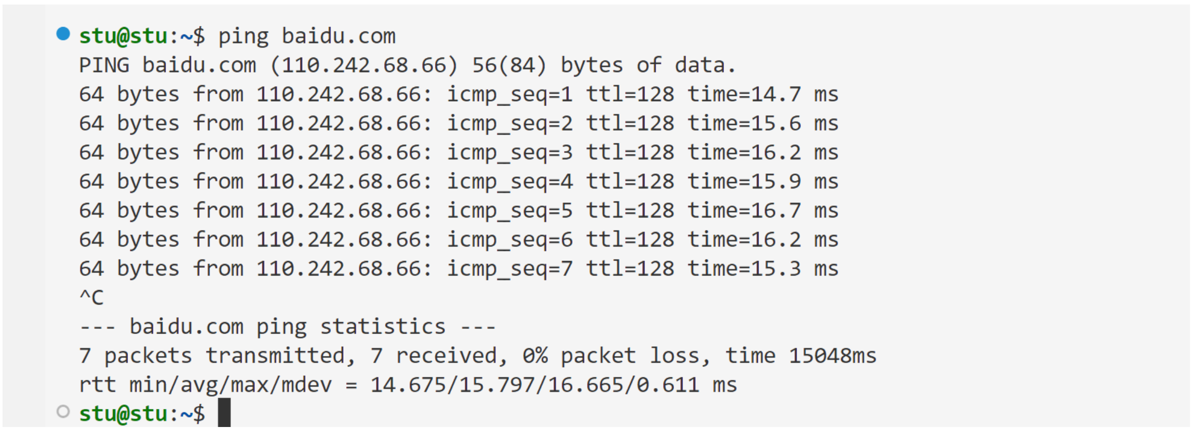
在实验中对ens32网卡的操作导致其ip地址一直改变，每次ssh登录都需要更新ip，查看原本网卡配置：

```
# This is the network config written by 'subiquity'
network:
  ethernets:
    ens32:
      dhcp4: true
  version: 2

```

修改为（相关信息可以用ip addr看，目前用的是VMnet8，可以在windows中用ipconfig查看nameserver的地址）：

```yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    ens32:
      dhcp4: false
      addresses: [192.168.32.132/24]
      gateway4: 192.168.32.2
      nameservers:
          addresses: [192.168.32.1]
  version: 2
```

此时就固定好了静态ip，ping一下网络连接也没问题：



```
stu@stu:~$ ping baidu.com
PING baidu.com (110.242.68.66) 56(84) bytes of data.
64 bytes from 110.242.68.66: icmp_seq=1 ttl=128 time=14.7 ms
64 bytes from 110.242.68.66: icmp_seq=2 ttl=128 time=15.6 ms
64 bytes from 110.242.68.66: icmp_seq=3 ttl=128 time=16.2 ms
64 bytes from 110.242.68.66: icmp_seq=4 ttl=128 time=15.9 ms
64 bytes from 110.242.68.66: icmp_seq=5 ttl=128 time=16.7 ms
64 bytes from 110.242.68.66: icmp_seq=6 ttl=128 time=16.2 ms
64 bytes from 110.242.68.66: icmp_seq=7 ttl=128 time=15.3 ms
^C
--- baidu.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 15048ms
rtt min/avg/max/mdev = 14.675/15.797/16.665/0.611 ms
stu@stu:~$ 
```

> Ubuntu20.04 网络配置的过程 - 简书 (jianshu.com)
>
> ubuntu 20.04 server设置静态 IP 地址 - 知乎 (zhihu.com)

最后发现静态IP+修改 `vmnetnat.conf` （在 `C:\ProgramData\VMware` 文件夹下）就可以实现稳定端口转发√，可使用本地环回ip的2223端口登录。

> 后来还是弃车跑到vbox了，vmware在run-net初始化的时候真的慢......

二. Debug记录

TASK 1 发包

(1) 建立映射后无法正常访问数据

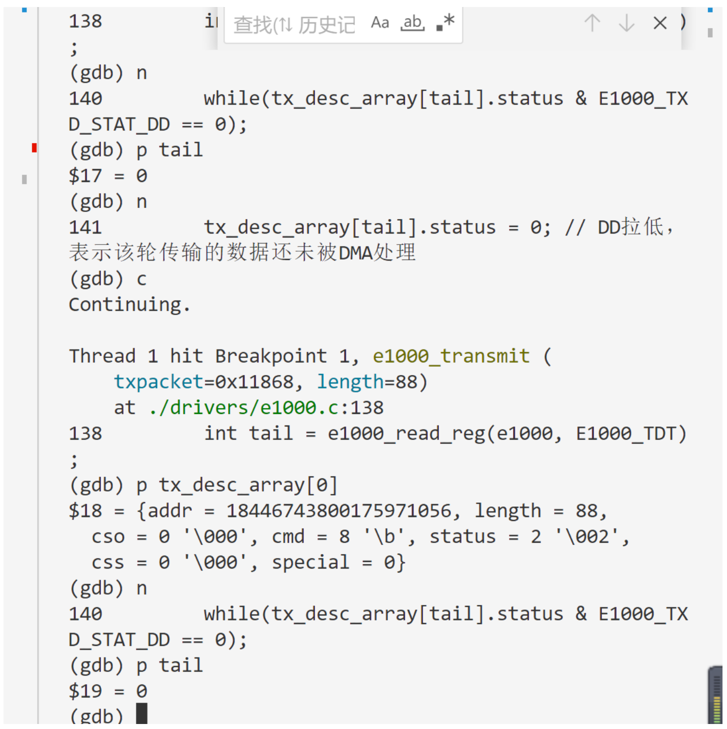
qemu抓包看，似乎发出了，但是长度不对？内容也不对......

```
        0x0030:  0000 0000 0000 0000 0000 0000          ............
11:56:57.762359 IP (tos 0x0, ttl 4, id 32291, offset 0, flags [none], proto UDP (17), length 165)
    192.168.32.1.49824 > 239.255.255.250.1900: [udp sum ok] UDP, length 137
        0x0000:  0100 5e7f fffa 0050 56c0 0008 0800 4500  ..^....PV.....E.
        0x0010:  00a5 7e23 0000 0411 6781 c0a8 2001 efff  ..~#....g.......
        0x0020:  fffa c2a0 076c 0091 71fd 4d2d 5345 4152  .....l..q.M-SEAR
        0x0030:  4348 202a 2048 5454 502f 312e 310d 0a48  CH.*.HTTP/1.1..H
        0x0040:  6f73 743a 2032 3339 2e32 3535 2e32 3535  ost:.239.255.255
        0x0050:  2e32 3530 3a31 3930 300d 0a53 543a 2075  .250:1900..ST:.u
        0x0060:  726e 3a73 6368 656d 6173 2d75 706e 702d  rn:schemas-upnp-
        0x0070:  6f72 673a 6465 7669 6365 3a49 6e74 6572  org:device:Inter
        0x0080:  6e65 7447 6174 6577 6179 4465 7669 6365  netGatewayDevice
        0x0090:  3a31 0d0a 4d61 6e3a 2022 7373 6470 3a64  :1..Man:."ssdp:d
        0x00a0:  6973 636f 7665 7222 0d0a 4d58 3a20 330d  iscover"..MX:.3.
        0x00b0:  0a0d 0a                                  ...
```

发送一个包之后查看tx描述符状态：

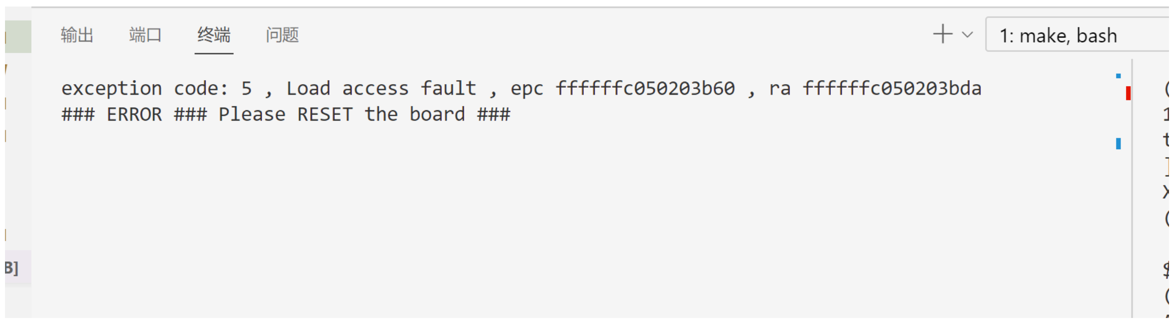
```
(gdb) p tx_desc_array[0]
$6 = {addr = 18446743800175971056, length = 88,
  cso = 0 '\000', cmd = 8 '\b', status = 0 '\000',
  css = 0 '\000', special = 0}
```

发现每次读出的tail值皆未发生改变：



```
138         i                                          )
;
(gdb) n
140         while(tx_desc_array[tail].status & E1000_TX
D_STAT_DD == 0);
(gdb) p tail
$17 = 0
(gdb) n
141         tx_desc_array[tail].status = 0; // DD拉低，
表示该轮传输的数据还未被DMA处理
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, e1000_transmit (
    txpacket=0x11868, length=88)
    at ./drivers/e1000.c:138
138         int tail = e1000_read_reg(e1000, E1000_TDT)
;
(gdb) p tx_desc_array[0]
$18 = {addr = 18446743800175971056, length = 88,
  cso = 0 '\000', cmd = 8 '\b', status = 2 '\002',
  css = 0 '\000', special = 0}
(gdb) n
140         while(tx_desc_array[tail].status & E1000_TX
D_STAT_DD == 0);
(gdb) p tail
$19 = 0
(gdb) 
```

意思到tail忘记更新了......修改后：

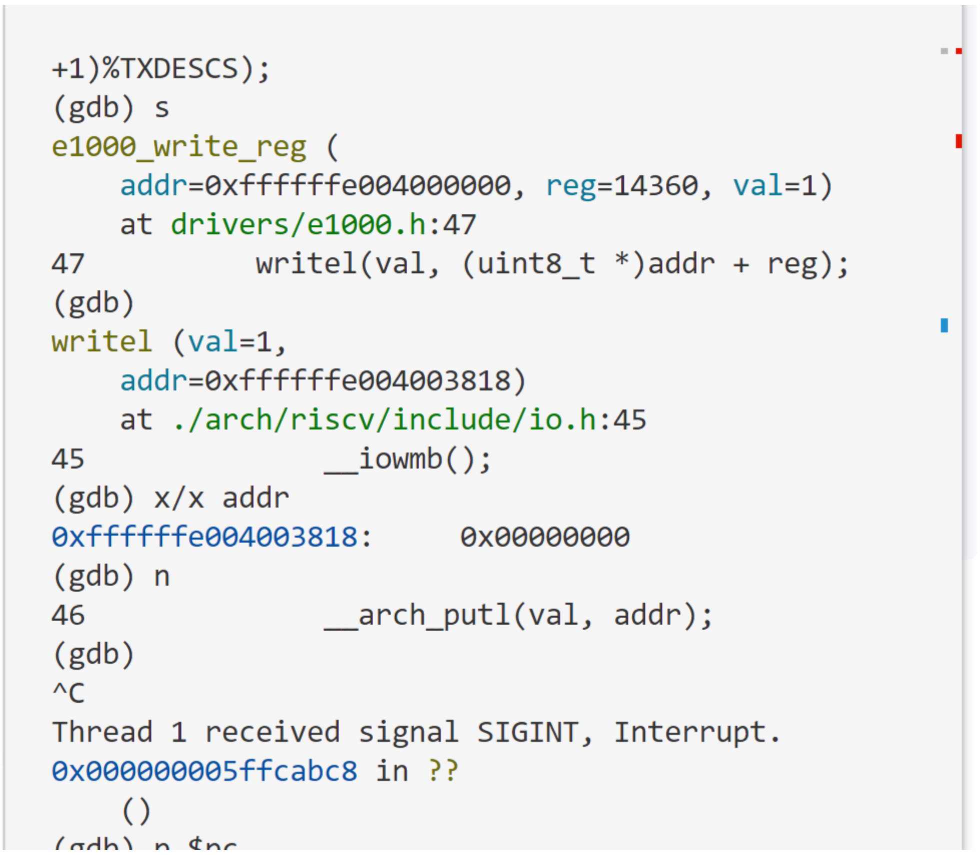


```
exception code: 5 , Load access fault , epc ffffffc050203b60 , ra ffffffc050203bda
### ERROR ### Please RESET the board ###
```

查看对应pc：

```
static inline void writel(unsigned int val, volatile void *addr)
{
ffffffc050203b60:	1101                	addi	sp,sp,-32
ffffffc050203b62:	ec22                	sd	s0,24(sp)
ffffffc050203b64:	1000                	addi	s0,sp,32
ffffffc050203b66:	87aa                	mv	a5,a0
ffffffc050203b68:	feb43023          	sd	a1,-32(s0)
```

要命的是gdb跟踪那个地址是可以看的：



```
+1)%TXDESCS);
(gdb) s
e1000_write_reg (
    addr=0xffffffe004000000, reg=14360, val=1)
    at drivers/e1000.h:47
47            writel(val, (uint8_t *)addr + reg);
(gdb)
writel (val=1,
    addr=0xffffffe004003818)
    at ./arch/riscv/include/io.h:45
45                __iowmb();
(gdb) x/x addr
0xffffffe004003818:     0x00000000
(gdb) n
46                __arch_putl(val, addr);
(gdb)
^C
Thread 1 received signal SIGINT, Interrupt.
0x000000005ffcabc8 in ??
    ()
```

后续修改了一个映射地址居然就成了（成谜。不过后面实验改回来了又可以，暂时没搞懂什么原因）

（2）未将给硬件的地址转化为实地址

修改映射地址后可以跑，但debug时状态寄存器都设置正确却死活不发包？后意识到状态描述符里的地址应该使用实地址：

```
tx_desc_array[i].addr = kva2pa(tx_pkt_buffer[i]);
```

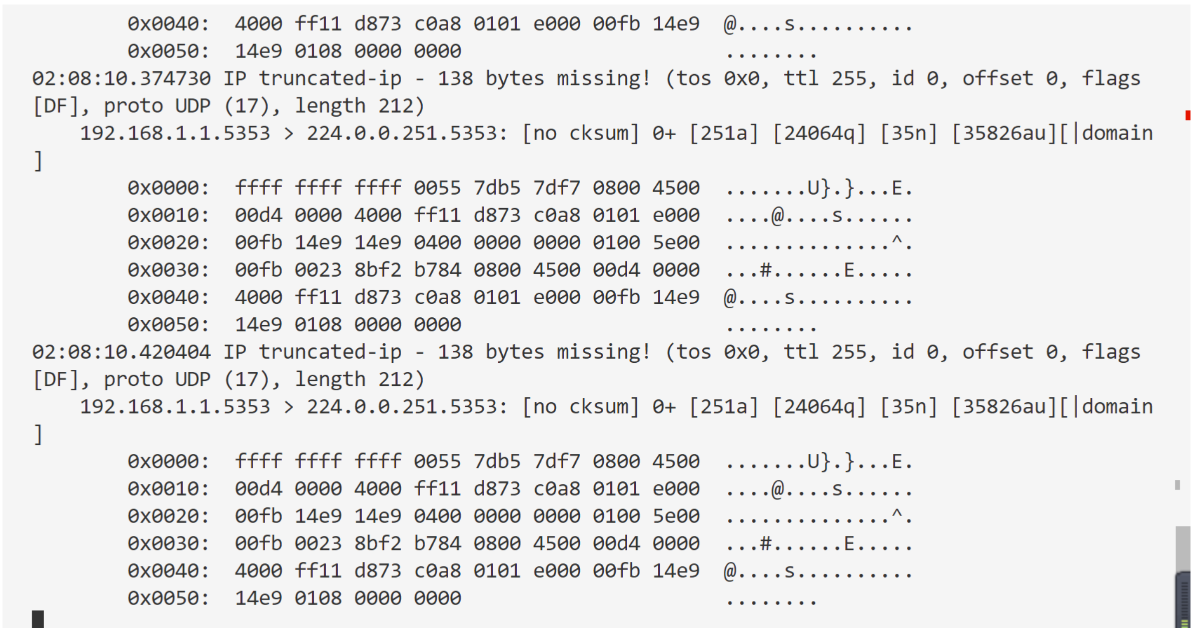
同时gdb跟踪时意识到映射可能出现了问题，前后数据对不上：

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
0x0000000000010000 in ?? ()
(gdb) c
```

```
Continuing.

Thread 1 hit Breakpoint 1, _boot ()
    at ./arch/riscv/kernel/start.S:20
20          csrw CSR_SIE, zero
(gdb) x/10 0x40000000
0x40000000:     0x08140240      0x00000000 0x80080783
0x00000000
0x40000010:     0x00000188      0x00000000 0x00000000
0x00000000
0x40000020:     0x182541e0      0x00000000
(gdb) x  0x40000000+0x3810
0x40003810:     0x00000000
(gdb) x  0x40000000+0x3818
0x40003818:     0x00000000
(gdb) x  0x40000000+0x3828
0x40003828:     0x01010000
(gdb) x  0x40000000+0x3808
0x40003808:     0x00000080
(gdb) x  0x40000000+0x3800
0x40003800:     0x5fffc780
(gdb) x  0x40000000+0x3804
0x40003804:     0x00000000
(gdb) x  0x40000000+0x3900
0x40003900:     0x00000000
(gdb) x  0x40000000+0x3904
0x40003904:     0x00000000
(gdb) x 0x5fffc780
0x5fffc780:     0x5fffd6e0
(gdb) x/5x 0x5fffc780
0x5fffc780:     0x5fffd6e0      0x00000000 0x0b000156
0x00000001
0x5fffc790:     0x5fffd6e0
(gdb) c
Continuing.

Thread 1 hit Breakpoint 2, boot_kernel (
    mhartid=0)
    at ./arch/riscv/kernel/boot.c:72
72              if (mhartid == 0) {
(gdb) b e1000_reset
Breakpoint 4 at 0x50203c4a: e1000_reset. (2 locations)
(gdb) c
Continuing.

Thread 1 hit Breakpoint 3, main ()
    at ./init/main.c:238
238                 uint64_t plic_addr = bios_read_fdt(PLIC_ADDR);
(gdb) c
```

```
Continuing.

Thread 1 hit Breakpoint 4, e1000_reset ()
    at ./drivers/e1000.c:28
28          e1000_write_reg(e1000, E1000_RCTL, 0);
(gdb) x/10x e1000
0xffffffc05c000000:     0x100000df      0x00000000      0x100004df      0x00000000
0xffffffc05c000010:     0x100008df      0x00000000      0x10000cdf      0x00000000
0xffffffc05c000020:     0x100010df      0x00000000
(gdb) x e1000+0x3800
0xffffffc05c003800:     0x00000000
(gdb) x 0xffffffc05fffc780
0xffffffc05fffc780:     0x5fffd6e0
(gdb)
```

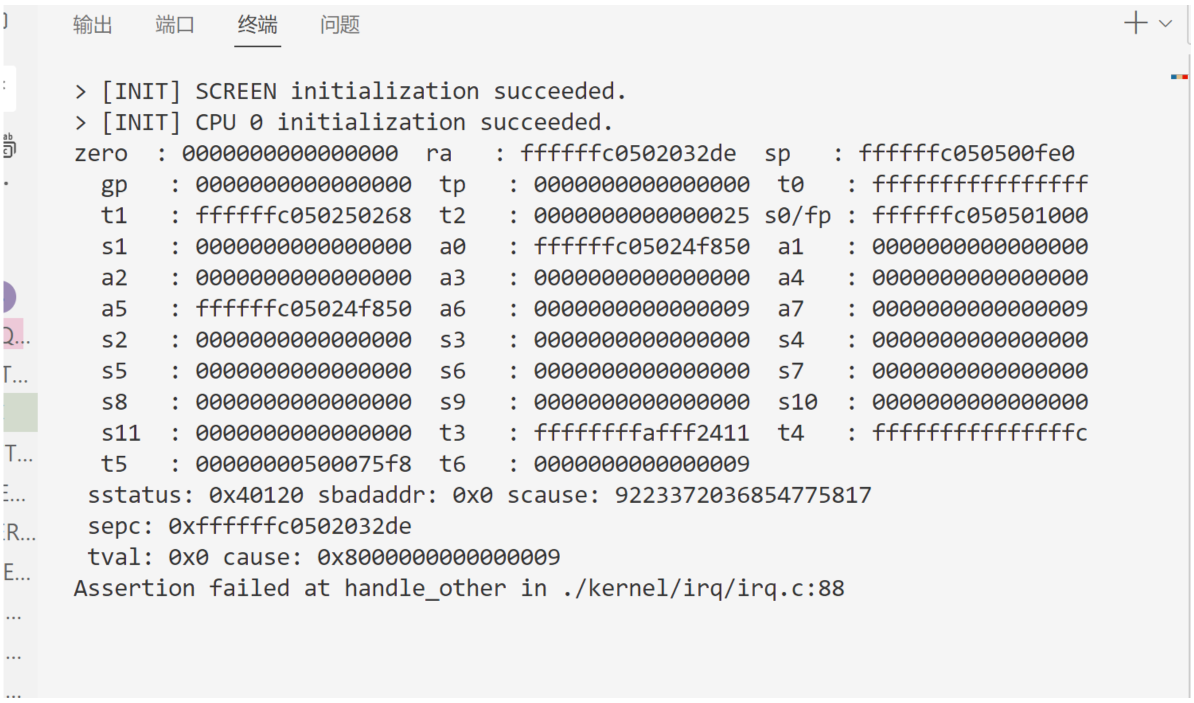
修改映射后收到包（长度不对，但据说是qemu历史遗留问题）：



```
        0x0040:  4000 ff11 d873 c0a8 0101 e000 00fb 14e9  @....s..........
        0x0050:  14e9 0108 0000 0000                      ........
02:08:10.374730 IP truncated-ip - 138 bytes missing! (tos 0x0, ttl 255, id 0, offset 0, flags
[DF], proto UDP (17), length 212)
    192.168.1.1.5353 > 224.0.0.251.5353: [no cksum] 0+ [251a] [24064q] [35n] [35826au][|domain
]
        0x0000:  ffff ffff ffff 0055 7db5 7df7 0800 4500  .......U}.}...E.
        0x0010:  00d4 0000 4000 ff11 d873 c0a8 0101 e000  ....@....s......
        0x0020:  00fb 14e9 14e9 0400 0000 0000 0100 5e00  ..............^.
        0x0030:  00fb 0023 8bf2 b784 0800 4500 00d4 0000  ...#......E.....
        0x0040:  4000 ff11 d873 c0a8 0101 e000 00fb 14e9  @....s..........
        0x0050:  14e9 0108 0000 0000                      ........
02:08:10.420404 IP truncated-ip - 138 bytes missing! (tos 0x0, ttl 255, id 0, offset 0, flags
[DF], proto UDP (17), length 212)
    192.168.1.1.5353 > 224.0.0.251.5353: [no cksum] 0+ [251a] [24064q] [35n] [35826au][|domain
]
        0x0000:  ffff ffff ffff 0055 7db5 7df7 0800 4500  .......U}.}...E.
        0x0010:  00d4 0000 4000 ff11 d873 c0a8 0101 e000  ....@....s......
        0x0020:  00fb 14e9 14e9 0400 0000 0000 0100 5e00  ..............^.
        0x0030:  00fb 0023 8bf2 b784 0800 4500 00d4 0000  ...#......E.....
        0x0040:  4000 ff11 d873 c0a8 0101 e000 00fb 14e9  @....s..........
        0x0050:  14e9 0108 0000 0000                      ........
```

（3）谜之硬件中断

上板出错：



```
> [INIT] SCREEN initialization succeeded.
> [INIT] CPU 0 initialization succeeded.
zero  : 0000000000000000  ra   : ffffffc0502032de  sp   : ffffffc050500fe0
  gp  : 0000000000000000  tp   : 0000000000000000  t0   : ffffffffffffffff
  t1  : ffffffc050250268  t2   : 0000000000000025  s0/fp : ffffffc050501000
  s1  : 0000000000000000  a0   : ffffffc05024f850  a1   : 0000000000000000
  a2  : 0000000000000000  a3   : 0000000000000000  a4   : 0000000000000000
  a5  : ffffffc05024f850  a6   : 0000000000000009  a7   : 0000000000000009
  s2  : 0000000000000000  s3   : 0000000000000000  s4   : 0000000000000000
  s5  : 0000000000000000  s6   : 0000000000000000  s7   : 0000000000000000
  s8  : 0000000000000000  s9   : 0000000000000000  s10  : 0000000000000000
  s11 : 0000000000000000  t3   : ffffffffafff2411  t4   : fffffffffffffffc
  t5  : 00000000500075f8  t6   : 0000000000000009
 sstatus: 0x40120 sbadaddr: 0x0 scause: 9223372036854775817
 sepc: 0xffffffc0502032de
 tval: 0x0 cause: 0x8000000000000009
Assertion failed at handle_other in ./kernel/irq/irq.c:88
```

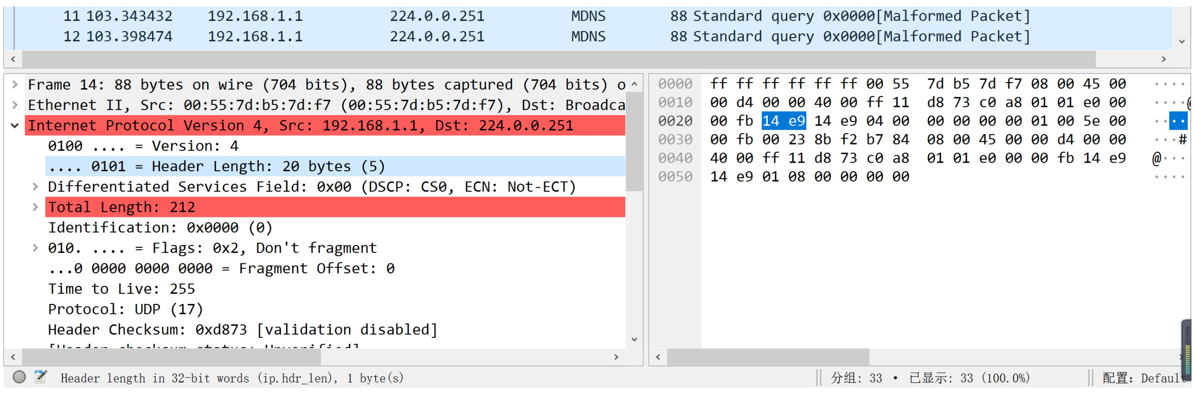
查看出错原因：

```
#define IRQ_M_TIMER     7
#define IRQ_U_EXT       8
#define IRQ_S_EXT       9
#define IRQ_M_EXT       11
```

收到了硬件中断？把sie的硬件中断位先关掉。

> 后面实验得知该中断并非来自网卡，而是从核和uart的接口

上板后：



```
   11 103.343432    192.168.1.1      224.0.0.251     MDNS     88 Standard query 0x0000[Malformed Packet]
   12 103.398474    192.168.1.1      224.0.0.251     MDNS     88 Standard query 0x0000[Malformed Packet]

> Frame 14: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) o
> Ethernet II, Src: 00:55:7d:b5:7d:f7 (00:55:7d:b5:7d:f7), Dst: Broadca
v Internet Protocol Version 4, Src: 192.168.1.1, Dst: 224.0.0.251
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  > Total Length: 212
    Identification: 0x0000 (0)
  > 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 255
    Protocol: UDP (17)
    Header Checksum: 0xd873 [validation disabled]

0000  ff ff ff ff ff ff 00 55  7d b5 7d f7 08 00 45 00
0010  00 d4 00 00 40 00 ff 11  d8 73 c0 a8 01 01 e0 00
0020  00 fb 14 e9 14 e9 04 00  00 00 00 00 01 00 5e 00
0030  00 fb 00 23 8b f2 b7 84  08 00 45 00 00 d4 00 00
0040  40 00 ff 11 d8 73 c0 a8  01 01 e0 00 00 fb 14 e9
0050  14 e9 01 08 00 00 00 00

Header length in 32-bit words (ip.hdr_len), 1 byte(s)    分组: 33 · 已显示: 33 (100.0%)    配置: Defaul
```

发现收到了包，但是包似乎是有误的，根据报错信息查找资料：

(9条消息) Bogus,bad length value xxx > IP Payload length_媒体盒子的博客-CSDN博客

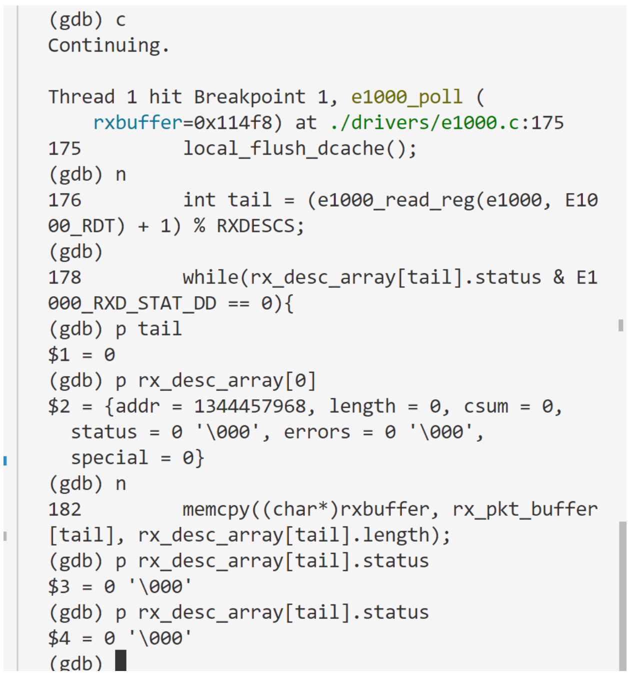
(9条消息) 调试FPGA与上位机进行udp通讯，wireshark报错：Bad udp Length xx ｜ IP PAYLOAD Length没有水杯和雨伞的工科男的博客-*CSDN*博客udp, bad length

似乎是转接口太拉了......下次用别人的试试......（后续询问助教，得知发送的包不是标准的UDP格式，报错正常）

TASK 2 收包

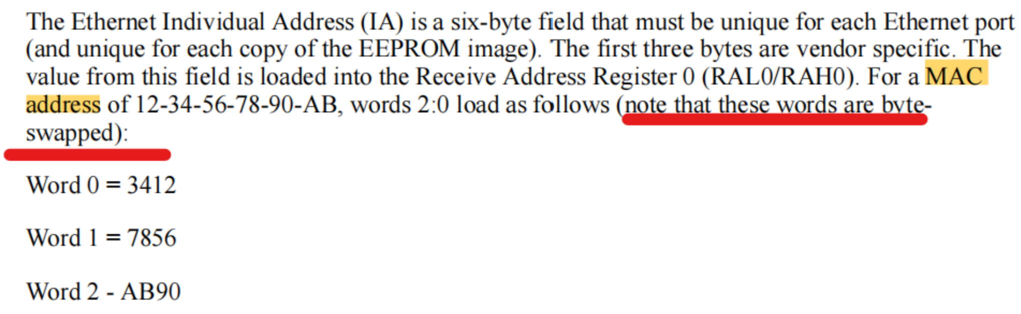
（1）MAC地址的存储方式

我最早初始化的时候未仔细看讲义，将tail和head都置为了0，导致一直收不到包，修改tail初始化为最后一个index后仍然收不到，gdb查看：



```
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, e1000_poll (
    rxbuffer=0x114f8) at ./drivers/e1000.c:175
175         local_flush_dcache();
(gdb) n
176         int tail = (e1000_read_reg(e1000, E10
00_RDT) + 1) % RXDESCS;
(gdb)
178         while(rx_desc_array[tail].status & E1
000_RXD_STAT_DD == 0){
(gdb) p tail
$1 = 0
(gdb) p rx_desc_array[0]
$2 = {addr = 1344457968, length = 0, csum = 0,
  status = 0 '\000', errors = 0 '\000',
  special = 0}
(gdb) n
182         memcpy((char*)rxbuffer, rx_pkt_buffer
[tail], rx_desc_array[tail].length);
(gdb) p rx_desc_array[tail].status
$3 = 0 '\000'
(gdb) p rx_desc_array[tail].status
$4 = 0 '\000'
(gdb) 
```

发现在while处没有按照预期的等待，意识到可能与符号优先级有关，果然按位与/或的优先级小于==，但是这并不能改变rx descriptor丝毫没被DMA修改这件事……查看Ethernet说明书：



The Ethernet Individual Address (IA) is a six-byte field that must be unique for each Ethernet port (and unique for each copy of the EEPROM image). The first three bytes are vendor specific. The value from this field is loaded into the Receive Address Register 0 (RAL0/RAH0). For a MAC address of 12-34-56-78-90-AB, words 2:0 load as follows (note that these words are byte-swapped):

Word 0 = 3412

Word 1 = 7856

Word 2 - AB90

意识到要将bytes swap一下

13.5.2 Receive Address Low

RAL (05400h + 8*n; R/W)

16 registers contain the lower bits of the 48-bit Ethernet address. All 32 bits are valid. Software can access the High and Low registers as a register pair if it can perform a 64-bit access to the PCI bus. The addresses stored in these registers are used for unicast/multicast address filtering.

The first receive address register (RAL0, RAH0) is also used for exact match PAUSE frame checking (Valid PAUSE packet that is addressed to the station's address). Therefore, RAL0 and RAH0 always should be used to store the individual Ethernet MAC address of the Ethernet controller.

Table 13-90. RAL Register Bit Description

31	0
RAL	

Field	Bit(s)	Initial Value	Description
RAL	31:0	X	Receive address low Contains the lower 32-bit of the 48-bit Ethernet address. RAL0 should be used to store the lower 32-bit of the Ethernet controller's Ethernet MAC address.

13.5.3 Receive Address High

RAH (05404h + 8∗n; R/W)

16 registers contain the upper bits of the 48-bit Ethernet address. The complete address is {RAH, RAL}. Software can access the High and Low registers as a register pair if it can perform a 64-bit access to the PCI bus. The addresses stored in these registers are used for unicast/multicast address filtering.

The first receive address register (RAL0, RAH0) is also used for exact match Pause frame checking (Valid Pause packet that is addressed to the station's address). Therefore, RAL0 and RAH0 always should be used to store the individual Ethernet MAC address of the Ethernet controller.

Note: When writing to this register, always write low-to-high. When clearing this register, always clear high-to-low.

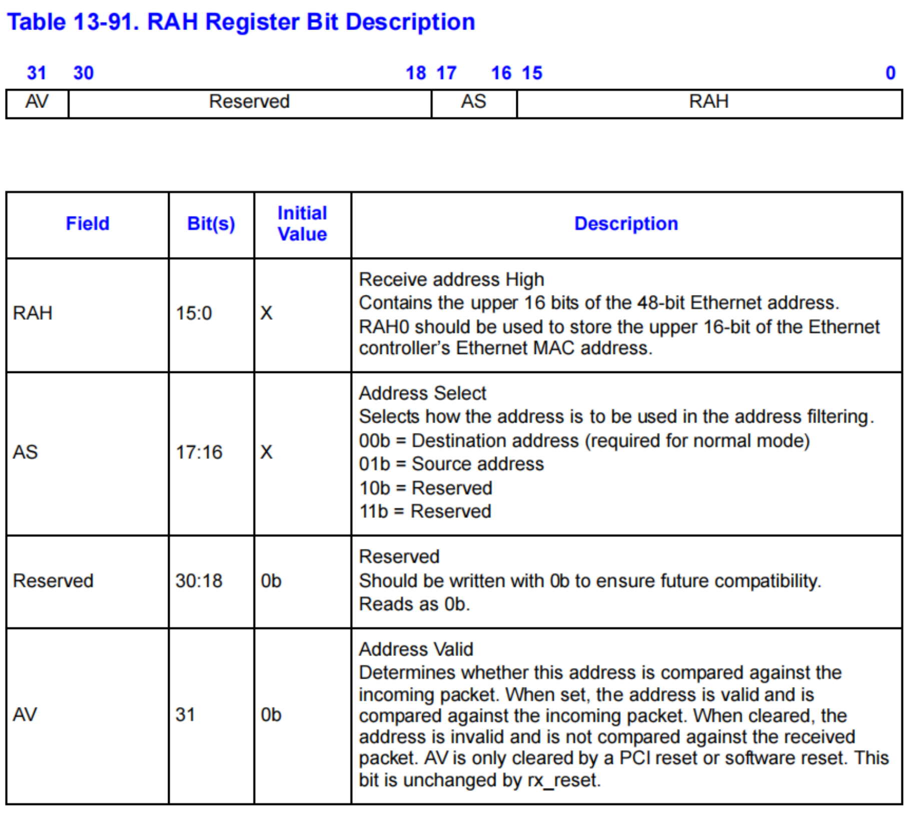


Table 13-91. RAH Register Bit Description

31	30	18	17	16	15	0
AV	Reserved		AS		RAH	

Field	Bit(s)	Initial Value	Description
RAH	15:0	X	Receive address High Contains the upper 16 bits of the 48-bit Ethernet address. RAH0 should be used to store the upper 16-bit of the Ethernet controller's Ethernet MAC address.
AS	17:16	X	Address Select Selects how the address is to be used in the address filtering. 00b = Destination address (required for normal mode) 01b = Source address 10b = Reserved 11b = Reserved
Reserved	30:18	0b	Reserved Should be written with 0b to ensure future compatibility. Reads as 0b.
AV	31	0b	Address Valid Determines whether this address is compared against the incoming packet. When set, the address is valid and is compared against the incoming packet. When cleared, the address is invalid and is not compared against the received packet. AV is only cleared by a PCI reset or software reset. This bit is unchanged by rx_reset.

即存储方式应该为：

```
static const uint8_t enetaddr[6] = {0x00, 0x0a, 0x35, 0x00, 0x1e, 0x53};
/**be stored as (l)0a00,0035,(h)531e**/
```

即使这么存了，还是收不到……，查看小程序的输出：

```
8. Linux netfilter queue (NFQUEUE) interface
Enter the interface number (1-8): 3
Info: Here, MAC Address is 128:250:91:51:86:239, listening on device enp0s3 ...
Info: MAC Address of the opposite is 00:10:53:00:30:83 ...
Info: Input command. For example:
        --- 'send 60': send 60 packets to the opposite
        --- 'test 60': keep sending packets in 60 seconds
        --- 'quit': quit this program
> sned 10
```

发现 MAC好像不是很匹配的样子……而且查找发现高低位的划分很玄学：

[MAC地址格式详解 - lsgxeva - 博客园 (cnblogs.com)]

后来询问助教得知应该按小端序存储，如下：

```
    // RA寄存器组，低位写在前
    uint32_t ral0 = (enetaddr[3]<<24) | (enetaddr[2]<<16) | (enetaddr[1]<<8) | enetaddr[0];
    uint32_t rah0 = E1000_RAH_AV | (enetaddr[5]<<8) | enetaddr[4];
```

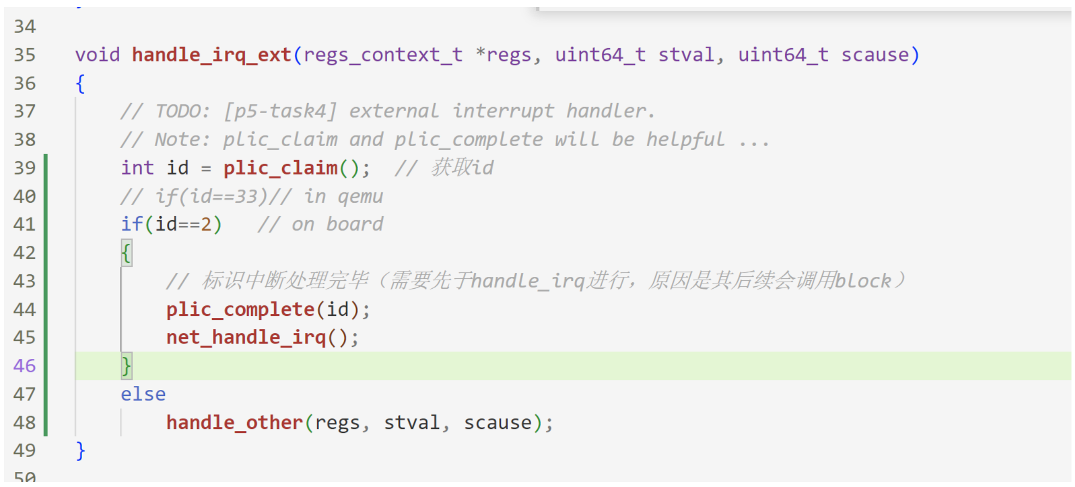
TASK 3 阻塞+时钟中断

顺顺利利，一些无关紧要的小bug。

TASK 4 阻塞+外设中断

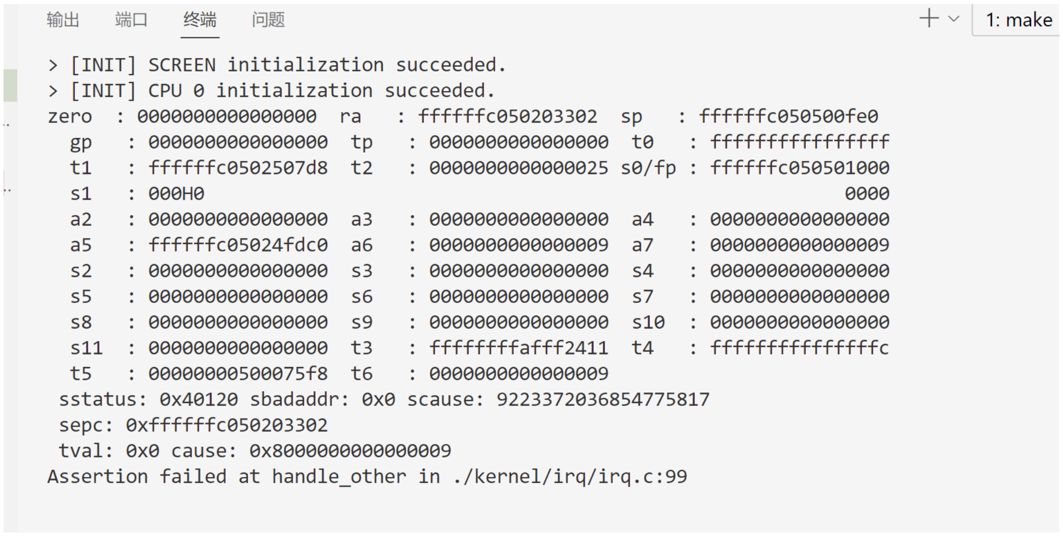
(1) 关于e1000的真实id

原本在判断不是网卡的中断时就会跳到handle other：



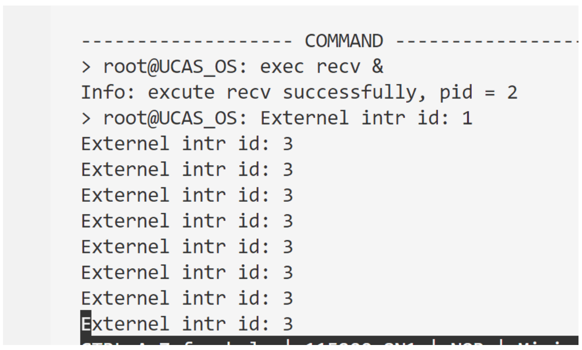
```c
void handle_irq_ext(regs_context_t *regs, uint64_t stval, uint64_t scause)
{
    // TODO: [p5-task4] external interrupt handler.
    // Note: plic_claim and plic_complete will be helpful ...
    int id = plic_claim();  // 获取id
    // if(id==33)// in qemu
    if(id==2)   // on board
    {
        // 标识中断处理完毕（需要先于handle_irq进行，原因是其后续会调用block）
        plic_complete(id);
        net_handle_irq();
    }
    else
        handle_other(regs, stval, scause);
}
```

还没进shell就会开门红：



```
> [INIT] SCREEN initialization succeeded.
> [INIT] CPU 0 initialization succeeded.
zero  : 0000000000000000  ra   : ffffffc050203302  sp   : ffffffc050500fe0
  gp   : 0000000000000000  tp   : 0000000000000000  t0   : ffffffffffffffff
  t1   : ffffffc0502507d8  t2   : 0000000000000025 s0/fp : ffffffc050501000
  s1   : 000H0                                                        0000
  a2   : 0000000000000000  a3   : 0000000000000000  a4   : 0000000000000000
  a5   : ffffffc05024fdc0  a6   : 0000000000000009  a7   : 0000000000000009
  s2   : 0000000000000000  s3   : 0000000000000000  s4   : 0000000000000000
  s5   : 0000000000000000  s6   : 0000000000000000  s7   : 0000000000000000
  s8   : 0000000000000000  s9   : 0000000000000000  s10  : 0000000000000000
  s11  : 0000000000000000  t3   : fffffffafff2411  t4   : fffffffffffffffc
  t5   : 00000000500075f8  t6   : 0000000000000009
 sstatus: 0x40120 sbadaddr: 0x0 scause: 9223372036854775817
 sepc: 0xffffffc050203302
 tval: 0x0 cause: 0x8000000000000009
Assertion failed at handle_other in ./kernel/irq/irq.c:99
```

后续处理时在遇到非网卡的外部中断一律直接complete。但这样还是不得行，处理recv时会卡住。打印出id看看：



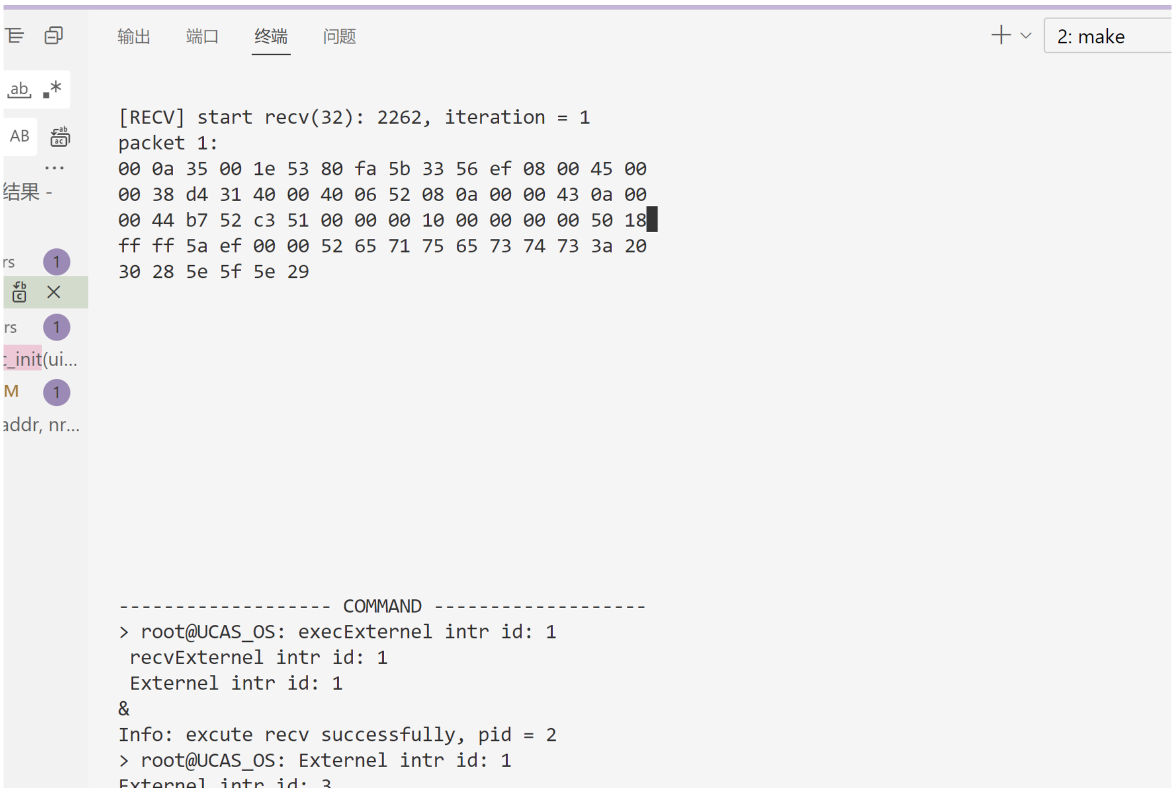
```
------------------- COMMAND ----------------
> root@UCAS_OS: exec recv &
Info: excute recv successfully, pid = 2
> root@UCAS_OS: Externel intr id: 1
Externel intr id: 3
Externel intr id: 3
Externel intr id: 3
Externel intr id: 3
Externel intr id: 3
Externel intr id: 3
Externel intr id: 3
Externel intr id: 3
```

发现收到了1和3的中断，但就是没有2的（不详的预感）......

把e1000的id改成3试试：

```c
void handle_irq_ext(regs_context_t *regs, uint64_t stval, uint64_t scause)
{
    // TODO: [p5-task4] external interrupt handler.
    // Note: plic_claim and plic_complete will be helpful ...
    int id = plic_claim();  // 获取id
    printk("Externel intr id: %d\n", id);
    if(id==3)   // on board
    {
        // 标识中断处理完毕（需要先于handle_irq进行，原因是其后续会调用block）
        ....
```

过了🤡?



```
[RECV] start recv(32): 2262, iteration = 1
packet 1:
00 0a 35 00 1e 53 80 fa 5b 33 56 ef 08 00 45 00
00 38 d4 31 40 00 40 06 52 08 0a 00 00 43 0a 00
00 44 b7 52 c3 51 00 00 00 10 00 00 00 00 50 18
ff ff 5a ef 00 00 52 65 71 75 65 73 74 73 3a 20
30 28 5e 5f 5e 29



------------------- COMMAND -------------------
> root@UCAS_OS: execExternel intr id: 1
 recvExternel intr id: 1
 Externel intr id: 1
&
Info: excute recv successfully, pid = 2
> root@UCAS_OS: Externel intr id: 1
Externel intr id: 3
```

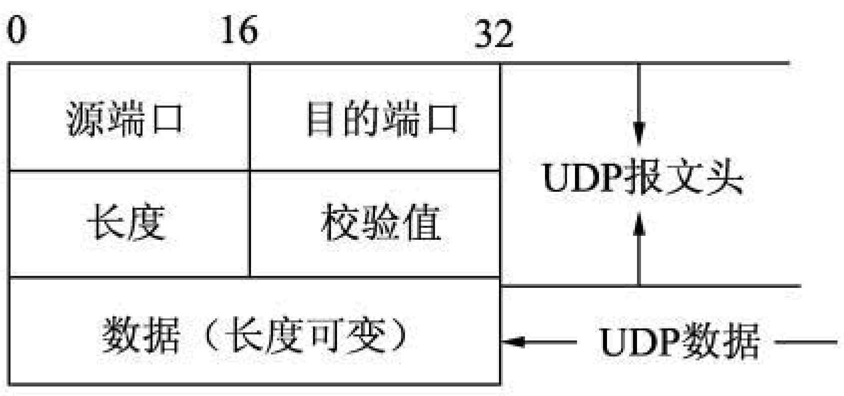
同时可以判断出之前task1困扰我的硬件中断应该不是来自网卡，而是id=1的神秘人。

TASK 5 收发包校对

(1) 包的格式

起初以为是UDP包：

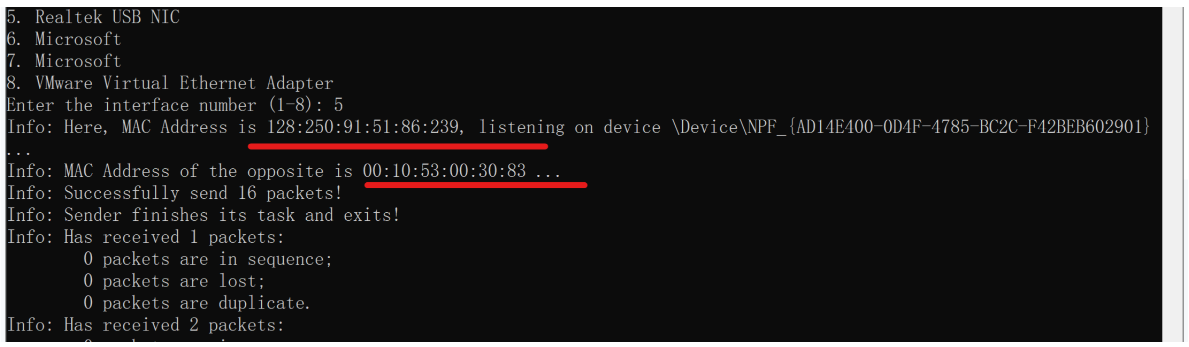
UDP报文格式详解 (biancheng.net)



起初按着这么写，发现不对劲......人工解读一下收到的包：

```
[ECHO] start send & recv
[RECV] recv 1126 bytes, 16 packages
[SEND] start send(16):
[SEND] replying 5/16
00 0a 35 00 1e 53 80 fa 5b 33 56 ef 08 00 45 00
00 38 d4 31 40 00 40 06 52 08 0a 00 00 43 0a 00
00 44 b7 52 c3 51 00 00 00 40 00 00 00 00 50 18
ff ff 57 bf 00 00 52 65 71 75 65 73 74 73 3a 20
34 28 5e 5f 5e 29
```

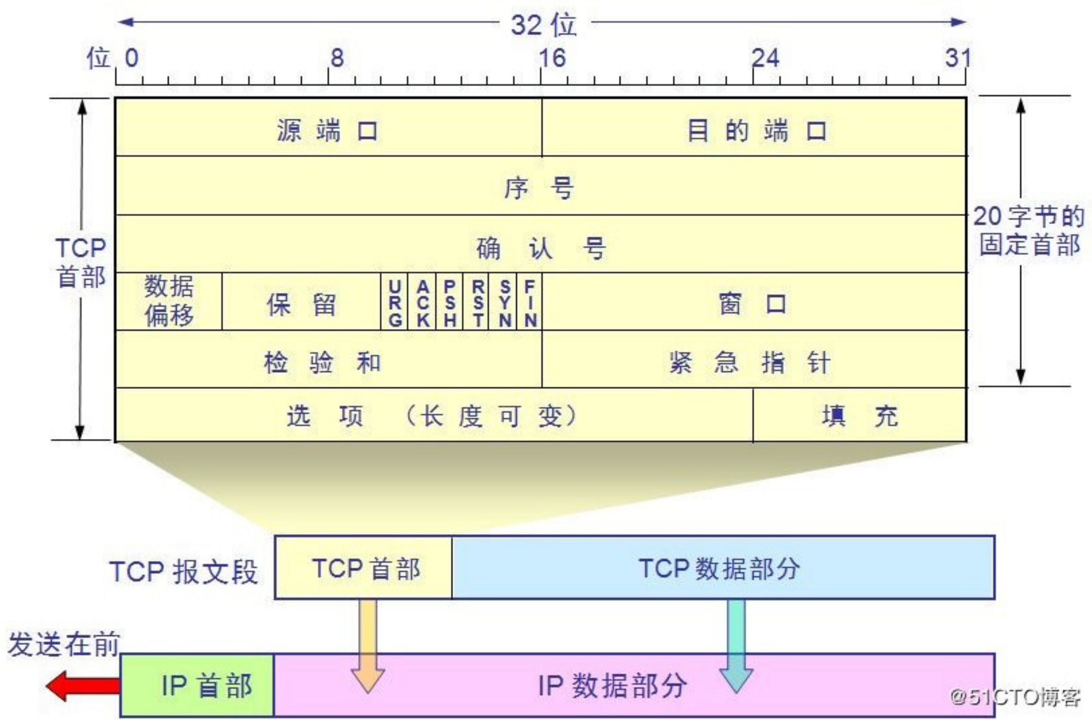
`00 0a 35 00 1e 53` 这个似乎是目的地址， `80 fa 5b 33 56 ef` 似乎是源地址：



```
5. Realtek USB NIC
6. Microsoft
7. Microsoft
8. VMware Virtual Ethernet Adapter
Enter the interface number (1-8): 5
Info: Here, MAC Address is 128:250:91:51:86:239, listening on device \Device\NPF_{AD14E400-0D4F-4785-BC2C-F42BEB602901}
...
Info: MAC Address of the opposite is 00:10:53:00:30:83 ...
Info: Successfully send 16 packets!
Info: Sender finishes its task and exits!
Info: Has received 1 packets:
        0 packets are in sequence;
        0 packets are lost;
        0 packets are duplicate.
Info: Has received 2 packets:
```

而"Requests: "的ASCII序列： `0x52 0x65 0x71 0x75 0x65 0x73 0x74 0x73 0x3a 0x20` ，救命，，，似乎在后面？？？？offset=38（这究竟有什么规律）

问了助教才知道这次是TCP包，，，



[TCP报文格式解析 (biancheng.net)]

三. Design Review

1. Tx/Rx的内容

- buffer的**实地址**
- 当前包（如果有的话）的长度
- CMD的info
  - 对于TX：拉高RS位确保硬件会在处理完毕后填写STAT.DD位
  - EOP位
    - 对于TX：传输完整个包后需要拉高
    - 对于RX：查看之判断是否为完整的包
- STAT的info
  - 查看DD位判断是否完成传输

发送完一个包后的TX示例：

```
(gdb) p tx_desc_array[0]
$6 = {addr = 18446743800175971056, length = 88,
  cso = 0 '\000', cmd = 8 '\b', status = 0 '\000',
  css = 0 '\000', special = 0}
```

2. MAC中断

(1) 外设中断流程

- 预备环节：注册中断处理函数 `handle_irq_ext`
- 外设中断处理函数中根据ID区分中断来源
  - ID==网卡ID? → 跳转到 `net_irq_handler`
  - else? →目前的处理方式：直接complete
- 网卡中断：进一步判断中断原因
  - 接受描述符不够？(RXDMT0，接受描述符仅剩余一半)
  - 发送队列满？ (TXEQ)

(2) 使能&屏蔽&判断

- 使能：设置IMS
- 屏蔽：设置IMC
- 判断：读取ICR

(3) 具体处理

- 发送队列满：
  - 阻塞当前发包进程
  - 在task 4：顺便check有没有可唤醒的recv（其实这一步也可以不做）
- 接受描述符不够：
  - 唤醒被阻塞的recv（如果有的话）

3. 收到包的判别方法

读取RDT指向的**下一个**描述符的STAT.DD

> RDH初始化为0，RDT为描述符数-1