

PART 1: Function Implemented

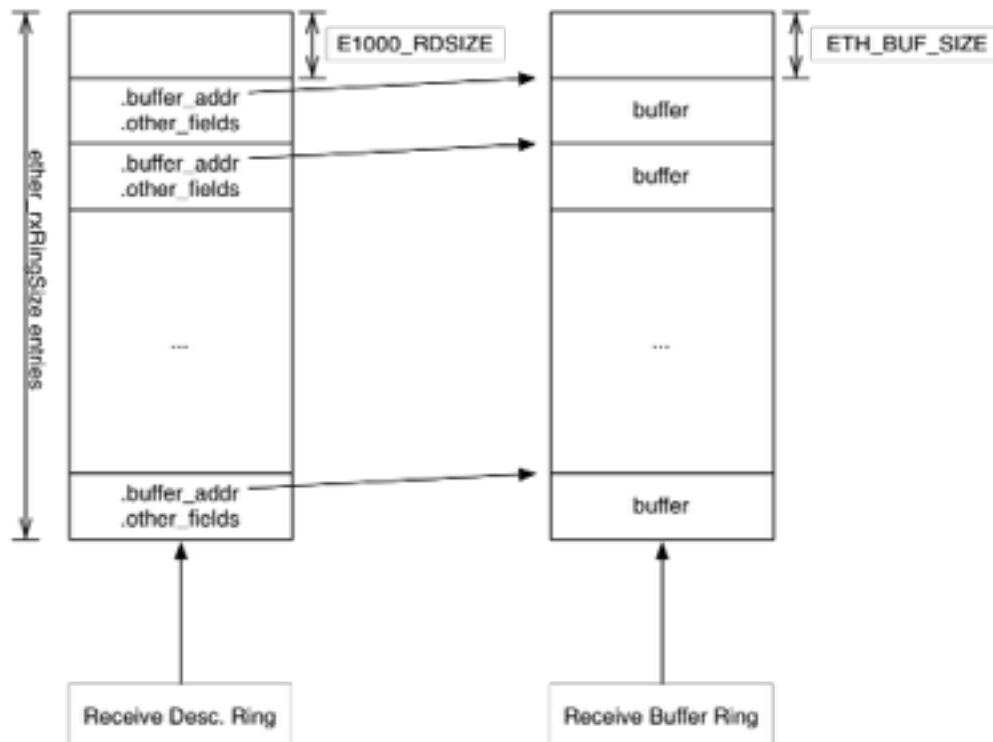
```

in /include/e1000.h
    e1000_io_{writel, readl}();
in /device/eth/ethInit.c
    ethInit();
in /device/eth/82545EMinit.c
    _82545EMInit();
    _82545EM_configure_{tx, rx}();
    _82545EM_{init, reset}_hw();
in /device/eth/e1000Read.c
    e1000Read();
in /device/eth/e1000Write.c
    e1000Write();

```

PART 2: Key Data Structure

Receive ring and buffer:



Left is a descriptor ring, each entry is of “`struct e1000_rx_desc`”. Right is a buffer ring, each is of size `ETH_BUF_SIZE`. Each ring takes a consecutive memory block. Allocation and initialization are in `_82545EMInit.c`. And The ring operation is performed with modulo in `e1000Read.c`

Transmit ring and buffer is similar.

PART 3: Testing

MAC address:

08:00:27:14:ba:b3

IP address:

10.0.2.15

Subnet mask:

255.255.255.0

Router:

10.0.2.2

```
Booting Xinu on i386-pc (vxinu)...

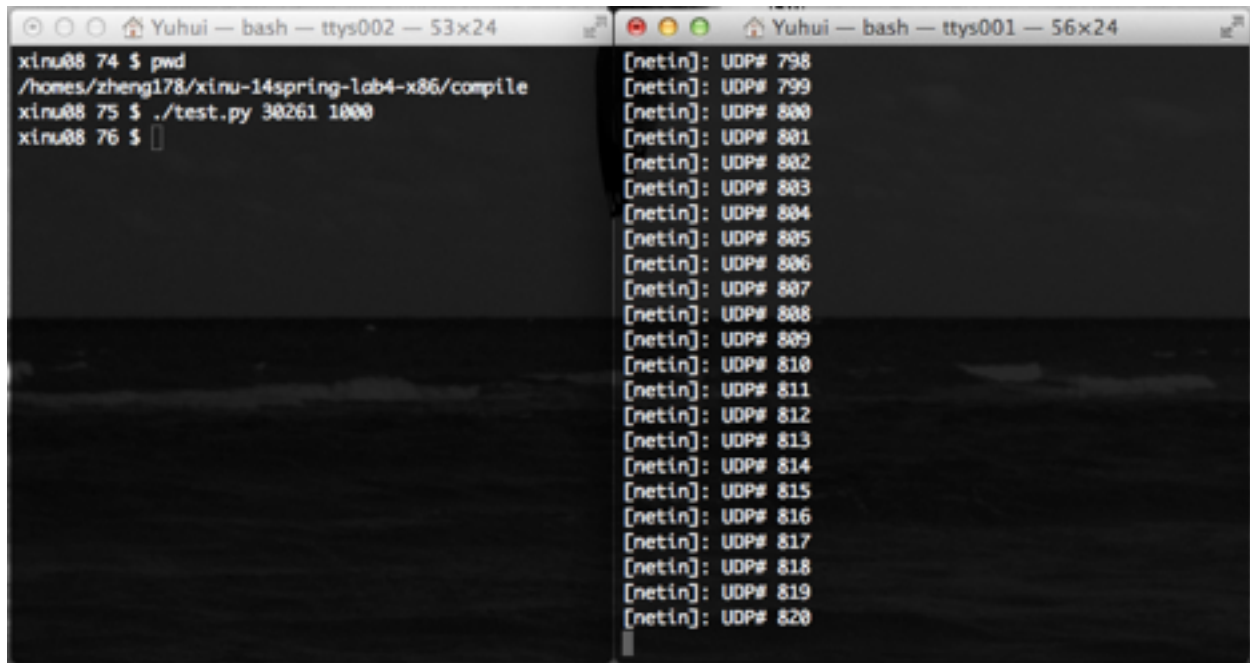
(x86 Xinu) #105 (zheng178@xinu08.cs.purdue.edu) Wed May 7 16:58:13 EDT 2014

16777216 bytes physical memory.
[0x00000000 to 0x00FFFFFF]
44719 bytes of Xinu code.
[0x00000000 to 0x0000AEAE]
32917 bytes of data.
[0x0000AEAF to 0x00012F43]
577720 bytes of heap space below 640K.
15728640 bytes of heap space above 1M.
[0x00100000 to 0x00FFFFFF]
main: calling getlocalip
MAC address is 08:00:27:14:ba:b3
Requesting IP address...
[e1000Read]:
[udp]:Here a msgOffered IP
I'm here to continue!

#####

IP address is 10.0.2.15 0a00020f
Subnet mask is 255.255.255.0 and router is 10.0.2.2
█
```

In side netin(), the number of packets reached udp_in() is as below:



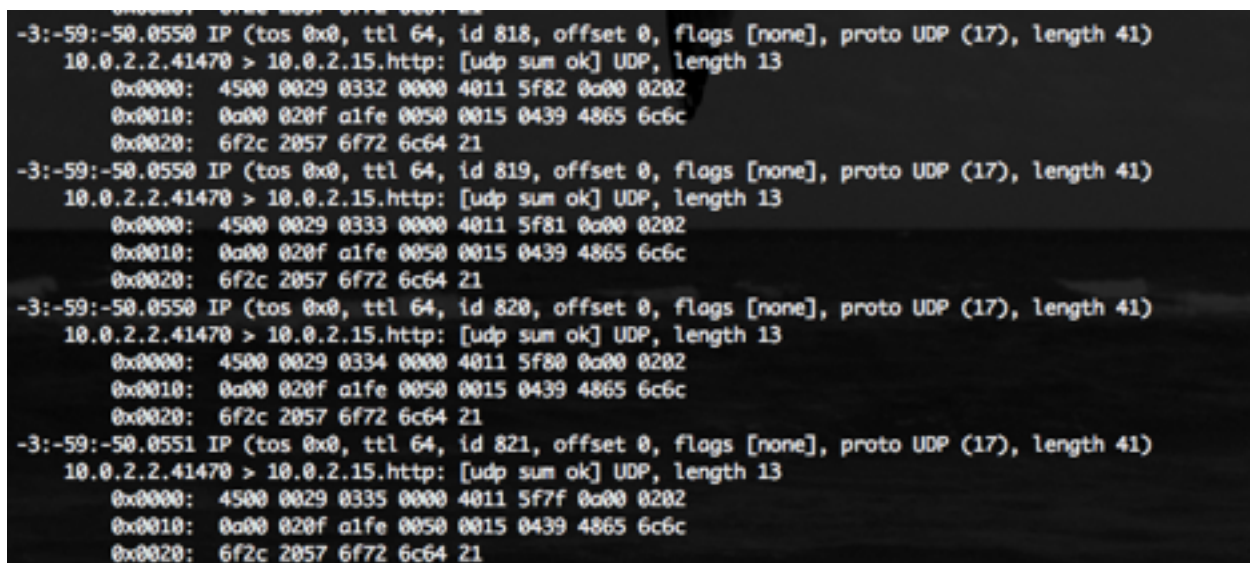
```

Yuhui — bash — ttys002 — 53x24
xinu08 74 $ pwd
/homes/zheng178/xinu-14spring-lab4-x86/compile
xinu08 75 $ ./test.py 30261 1000
xinu08 76 $

Yuhui — bash — ttys001 — 56x24
[netin]: UDP# 798
[netin]: UDP# 799
[netin]: UDP# 800
[netin]: UDP# 801
[netin]: UDP# 802
[netin]: UDP# 803
[netin]: UDP# 804
[netin]: UDP# 805
[netin]: UDP# 806
[netin]: UDP# 807
[netin]: UDP# 808
[netin]: UDP# 809
[netin]: UDP# 810
[netin]: UDP# 811
[netin]: UDP# 812
[netin]: UDP# 813
[netin]: UDP# 814
[netin]: UDP# 815
[netin]: UDP# 816
[netin]: UDP# 817
[netin]: UDP# 818
[netin]: UDP# 819
[netin]: UDP# 820

```

With above transmission, tcpdump gets (partial):



```

-3:-59:-50.0550 IP (tos 0x0, ttl 64, id 818, offset 0, flags [none], proto UDP (17), length 41)
  10.0.2.2.41470 > 10.0.2.15.http: [udp sum ok] UDP, length 13
    0x0000: 4500 0029 0332 0000 4011 5f82 0a00 0202
    0x0010: 0a00 020f a1fe 0050 0015 0439 4865 6c6c
    0x0020: 6f2c 2057 6f72 6c64 21

-3:-59:-50.0550 IP (tos 0x0, ttl 64, id 819, offset 0, flags [none], proto UDP (17), length 41)
  10.0.2.2.41470 > 10.0.2.15.http: [udp sum ok] UDP, length 13
    0x0000: 4500 0029 0333 0000 4011 5f81 0a00 0202
    0x0010: 0a00 020f a1fe 0050 0015 0439 4865 6c6c
    0x0020: 6f2c 2057 6f72 6c64 21

-3:-59:-50.0550 IP (tos 0x0, ttl 64, id 820, offset 0, flags [none], proto UDP (17), length 41)
  10.0.2.2.41470 > 10.0.2.15.http: [udp sum ok] UDP, length 13
    0x0000: 4500 0029 0334 0000 4011 5f80 0a00 0202
    0x0010: 0a00 020f a1fe 0050 0015 0439 4865 6c6c
    0x0020: 6f2c 2057 6f72 6c64 21

-3:-59:-50.0551 IP (tos 0x0, ttl 64, id 821, offset 0, flags [none], proto UDP (17), length 41)
  10.0.2.2.41470 > 10.0.2.15.http: [udp sum ok] UDP, length 13
    0x0000: 4500 0029 0335 0000 4011 5f7f 0a00 0202
    0x0010: 0a00 020f a1fe 0050 0015 0439 4865 6c6c
    0x0020: 6f2c 2057 6f72 6c64 21

```

test.py is sending "Hello, World!". Since packets received contain:

"48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 21",

which is exactly "Hello, World!" in ASCII code. Thus, functionality is correct.