Network Drivers

CS3500 Operating Systems Jul-Nov 2025

Sanjeev Subrahmaniyan S B

EE23B102

1 Objective

The objective of this lab is to complete the network driver for the E1000 network interface card emulated in the environment. The tasks primarily are to complete the receive and tranmsit functionality of the network card.

2 Code

The sections of code that needed to be modified were specified in the handout for the assignment. Specifically, additions are made in the e1000.c file which performs the initialization of the card, provides the interrupt handlers and implements the transmit and receive handlers.

2.1 kernel/net.c

```
diff --git a/kernel/e1000.c b/kernel/e1000.c
index 009e977..c043443 100644
--- a/kernel/e1000.c
+++ b/kernel/e1000.c
@@ -101,8 +101,26 @@ e1000_transmit(char *buf, int len)
   // the TX descriptor ring so that the e1000 sends it. Stash
   // a pointer so that it can be freed after send completes.
   //
-
+  acquire(&e1000_lock);
+  uint32 tx_ring_idx = regs[E1000_TDT];
+  uint32 tx_ring_idx_h = regs[E1000_TDH];
+  if(!(tx_ring[tx_ring_idx].status & E1000_TXD_STAT_DD) || !(tx_ring_idx_h != (
   tx_ring_idx + 1) % TX_RING_SIZE)){
+    release(&e1000_lock);
+    kfree(buf);
+    return -1; // previous transaction not complete
+  }
+
+  if(tx_bufs[tx_ring_idx] != 0) // free when previously used
+    kfree((void *)(tx_bufs[tx_ring_idx]));
+
+  tx_ring[tx_ring_idx].addr = (uint64) buf;
+  tx_ring[tx_ring_idx].length = len;
+  tx_ring[tx_ring_idx].cmd = E1000_TXD_CMD_EOP | E1000_TXD_CMD_RS;
+  tx_ring[tx_ring_idx].status = 0;
+  tx_bufs[tx_ring_idx] = buf;
+  regs[E1000_TDT] = (tx_ring_idx + 1) % TX_RING_SIZE;
+  release(&e1000_lock);
   return 0;
 }
```

```
@@ -116,6 +134,25 @@ e1000_recv(void)
   // Create and deliver a buf for each packet (using net_rx()).
   //

+  uint32 rx_ring_idx_base = regs[E1000_RDT];
+  uint32 rx_ring_idx = (rx_ring_idx_base + 1) % RX_RING_SIZE;
+
+  while(rx_ring_idx != rx_ring_idx_base){ // preventing overflow
+    if(!(rx_ring[rx_ring_idx].status & E1000_RXD_STAT_DD)){
+      // check if there is a packet ready to be read
+      break;
+    } else {
+      uint64 length = rx_ring[rx_ring_idx].length;
+      net_rx(rx_bufs[rx_ring_idx], length); // send
+      if((rx_bufs[rx_ring_idx] = kalloc()) == 0) return;
+      // bind buffers and descriptors
+      rx_ring[rx_ring_idx].status = 0;
+      rx_ring[rx_ring_idx].addr = (uint64) rx_bufs[rx_ring_idx];
+      rx_ring_idx = (rx_ring_idx + 1) % RX_RING_SIZE;
+    }
+  }
+  regs[E1000_RDT] = (rx_ring_idx - 1) % RX_RING_SIZE; // point of last descriptor
+  return;
 }
```

In the net.c file, the e1000_transmit() and e1000_recv() functions are implemented. Briefly, the functions performed by each of them are as follows.

2.2 Transmit

1. Acquires the tail pointer of the transmit ring and inspects if the previous operation has completed.
2. If the previous process is not yet complete, free is called and error is returned.
3. If the previous process is complete, the buffer used previously is freed for later use.
4. The entry in the transmit ring is updated by setting appropriate parts of the descriptors. Specifically, the address of the buffer, size of the message buffer and command bits are set.
5. The pointer is updated to reflect the new position.

2.3 Receive

1. Acquires the tail pointer of the receive ring and inspects if there is a packet ready to be run.
2. If a packet is not ready, the process is terminated.
3. If a process is ready, it is read and sent using the net_rx() function from the networking stack.
4. A new page is allocated and the buffers and descriptors are bound together.
5. The ring pointer position is updated to the last descriptor.

3 Execution and Output

The program is tested using **nettest grade** from inside the QEMU environment and calling the python script as **./nettest.py grade** in another terminal. The output is shown below:

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ nettest grade
txone: sending one packet
arp_rx: received an ARP packet
ip_rx: received an IP packet
bp is 0!
ping0: starting
ping0: OK
ping1: starting
ping1: OK
ping2: starting
ping2: OK
ping3: starting
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
bp is 0!
ping3: OK
dns: starting
DNS arecord for pdos.csail.mit.edu. is 128.52.129.126
dns: OK
free: OK
$ QEMU: Terminated
```