JOS-Lab 6: Network Driver

Part A: Initialization and transmitting packets

Exercise 1
Add a call to time_tick() for every clock interrupt in kern/trap.c.
In sys_time_msec(), return time_msec().
Exercise 3
According to Intel's documentation, the vendor ID is 0x8086 and the device ID is 0x100e.
In e1000_attach(), just use pci_func_enable() to enable the device.
Finally, add an entry for e1000 in the pci_attach_vendor array.
Exercise 4
In e1000_attach(), use boot_map_region() to map E1000's BAR 0 at KSTACKTOP. The permission should be set as PTE_W|PTE_PCD|PTE_PWT.
Finally, print out the device status register.
Exercise 5
The memory for DMA use should not be cached to avoid consistency problem. So I modified pmap.c to reserve some memory with cache disable and write through in the memory space for descriptor arrays and the packet buffers.
In e1000_attach(), just initialize the registers as the manual described.
Exercise 6
The transmit function I wrote is int e1000_transmit(const char * buf, uint32_t len).
In e1000_transmit(), it first checks the packet's length. Then, check that the next descriptor is free (If the queue is full, it just returns -E_TX_QUEUE_FULL). Afterwards, it copies the packet data into the corresponding buffer and sets the descriptor. Finally, it updates TDT to inform the hardware.
Exercise 7
I added the syscall sys_net_try_transmit(). It uses user_mem_assert() to check the pointer passed from user space, then just calls e1000_transmit() to transmit the package.
Exercise 8
In output.c, just keep trying to receive a NSREQ_OUTPUT IPC. Then keep calling the sys_net_try_transmit() to send the packet (when the transmit queue is full, it will try again).

Part B: Receiving packets and the web server

Exercise 10
The allocation for receive queue is just like that of the transmit queue. In e1000_attach(), just initialize the registers as the manual described.
Exercise 11
The receive function I wrote is int e1000_receive(char * buf).
In e1000_receive(), it first checks if the next descriptor represents a received packet. If not, just return -E_RCV_QUEUE_EMPTY. Then it obtains the length of the packet and copies the packet's data into the buffer. Afterwards, it resets the descriptor's status and updates RDT. Finally, it returns the packet length.

I also added the syscall sys_net_try_receive() to expose it to user space.

Exercise 12

In input.c, just keep trying to receive a packet with sys_net_try_receive(). If a packet is received, it sends it to the network server with NSREQ_INPUT IPC. The page sent in the IPC will be reading from it for a while, so we need to allocate a new page for the next packet with sys_page_alloc(), just like low_level_output() in jif.c

Exercise 13

send_data() just reads data from the file and send them to the client.

In send_file(), open the requested file. Send a 404 error if the file does not exist.

Then check the file type using fstat(), send a 404 error if the file is a directory.