Sistemas Operativos

TP5 – Network Driver en JOS

Gabriel Robles – 95897

Repositorio: https://github.com/gabyrobles93/TP1-SisOp

Preguntas:

1. How did you structure your transmit implementation? In particular, what do you do if the transmit ring is full?

En primer lugar implementé la inicialización de la transmisión (e1000_tx_init), siguiendo los pasos de la hoja de datos de la placa de red. Aquí tome algunas decisiones de diseño, como por ejemplo:

- Definir en forma estática el arreglo de descriptores de transmisión (global y estático, en el archivo e1000.c)
- Definir en forma estática el arreglo de paquetes de transmisión, o buffers de datos a transmitir (global y estático, en el archivo e1000.c), donde la capacidad de cada buffer es el máximo permitido por Ethernet 1518 bytes.
- Ambos arreglos cuentan con una capacidad de 64 elementos (cota impuesta por el enunciado).

En la inicialización, se asocian los arreglos entre sí en un loop de 64 vueltas. El valor address de las estructuras de descriptor de transmisión se asocian uno a uno a las direcciones FISICAS de los paquetes de datos del segundo arreglo.

Luego, implementé la función de transmisión (e1000_send_packet), que valida que el tamaño del paquete a enviar sea menor al máximo impuesto por Ethernet (1518 bytes). Luego, obtiene el TAIL de la cola circular de descriptores mediante el valor del registro TDT. Luego chequea el valor del bit DD (Descriptor Done) del campo status del descriptor de transmisión: si es 1 significa que dicho descriptor está disponible para utilizarse. Si este valor DD es 0, significa que la cola está llena y se debe reintentar hasta que se libere algún descriptor.

En en caso de que se encuentre un descriptor libre, se copia la información del buffer en la dirección asociada al descriptor (y previamente alocada estáticamente en el arreglo de paquetes). Se setean mas parámetros del descriptor necesarios y por último se actualiza el valor de TAIL del arreglo circular como (i + 1) % N.

Para todas estas tareas implementé dos funciones auxiliares: setreg y getreg que reciben como parámetro un offset (desde la dirección base de registros de la placa) y el valor que se desea setear para getreg.

2. How did you structure your receive implementation? In particular, what do you do if the receive queue is empty and a user environment requests the next incoming packet?

En la inicialización de la recepción se setearon los registros de la placa siguiendo los pasos del manual y las recomendaciones del enunciado del trabajo. Aquí se tomaron algunas decisiones de diseño como:

- Hardcodear el valor de la MAC address de la placa (sugerido por el enunciado).
- Definir en forma estática el arreglo de descriptores de recepción (global y estático, en el archivo e1000.c)
- Definir en forma estática el arreglo de paquetes de recepción (global y estático, en el archivo e1000.c), donde el tamaño de cada buffer es de 2048 bytes (una de los posibles tamaños que ofrecía el manual de Intel).
- Ambos arreglos cuentan con una capacidad de 128 elementos (cota inferior impuesta por el enunciado).

En forma análoga a la transmisión, en la inicialización se asocian los arreglos entre si. El valor buffer address de las estructuras de descriptor de recepción se asocian uno a uno a las direcciones FISICAS de los paquetes de datos del segundo arreglo.

Luego implementé la función de recepción, que en primer lugar obtiene el indice del Tail en el anillo de descriptores. Se chequea si el anillo esta vacío viendo si el bit DD (Descriptor Done) esta encendido. En caso que el anillo no esté vacío, significa que hay algo para recibir, por lo tanto se mueve la información del buffer del descriptor al buffer que se recibe como parámetro. Luego se apagan los bits DD y EOP y por último se setea el registro RDT en la siguiente posición. Esta función retorna el largo del buffer recibido.

3. What does the web page served by JOS's web server say?

This file came from JOS. Cheesy web page!

4. How long approximately did it take you to do this lab?

Me tomo 10 días dedicando unas 2 horas cada día.

Corrida make grade

```
make[1]: Leaving directory '/home/grobles/Desktop/TP1-SisOp'
testtime: OK (7.6s)
pci attach: OK (0.8s)
testoutput [5 packets]: OK (1.4s)
testoutput [100 packets]: OK (2.2s)
Part A score: 35/35

testinput [5 packets]: OK (1.7s)
testinput [100 packets]: OK (1.1s)
tcp echo server [echosrv]: OK (0.9s)
web server [httpd]: OK
  http://localhost:26002/: OK (1.8s)
  http://localhost:26002/index.html: OK (0.9s)
  http://localhost:26002/random_file.txt: OK (1.1s)
Part B score: 70/70

Score: 105/105
```

Git diff

$ git diff c641bec7d4c1d18613ee0ef136f9dfc3f3daa8c4

```
@@ -60,6 +60,8 @@ int   sys_page_unmap(envid_t env, void *pg);
 int    sys_ipc_try_send(envid_t to_env, uint32_t value, void *pg,
int perm);
 int    sys_ipc_recv(void *rcv_pg);
 unsigned int sys_time_msec(void);
+int sys_transmit_packet(char * buffer, size_t size);
+int sys_receive_packet(void * buffer, size_t size);

 // This must be inlined.  Exercise for reader: why?
 static inline envid_t __attribute__((always_inline))
diff --git a/inc/syscall.h b/inc/syscall.h
index 36f26de..44433c2 100644
--- a/inc/syscall.h
+++ b/inc/syscall.h
@@ -18,6 +18,8 @@ enum {
        SYS_ipc_try_send,
        SYS_ipc_recv,
        SYS_time_msec,
+       SYS_transmit_packet,
+       SYS_receive_packet,
        NSYSCALLS
 };

diff --git a/jos.out b/jos.out
index 6b547dc..62699ec 100644
--- a/jos.out
+++ b/jos.out
@@ -1,5 +1,8 @@
 + ld obj/kern/kernel
 + mk obj/kern/kernel.img
+qemu-system-i386: -redir tcp:26001::7: The -redir option is
deprecated. Please use '-netdev user,hostfwd=...' instead.
+qemu-system-i386: -redir tcp:26002::80: The -redir option is
deprecated. Please use '-netdev user,hostfwd=...' instead.
+qemu-system-i386: -redir udp:26001::7: The -redir option is
deprecated. Please use '-netdev user,hostfwd=...' instead.
 6828 decimal is 15254 octal!
 Physical memory: 131072K available, base = 640K, extended =
130432K
 check_page_free_list() succeeded!
@@ -10,8 +13,17 @@ check_page_free_list() succeeded!
 check_page_installed_pgdir() succeeded!
 SMP: CPU 0 found 1 CPU(s)
 enabled interrupts: 1 2 4
+PCI: 00:00.0: 8086:1237: class: 6.0 (Bridge device) irq: 0
+PCI: 00:01.0: 8086:7000: class: 6.1 (Bridge device) irq: 0
```

```
+PCI: 00:01.1: 8086:7010: class: 1.1 (Storage controller) irq: 0
+PCI: 00:01.3: 8086:7113: class: 6.80 (Bridge device) irq: 9
+PCI: 00:02.0: 1234:1111: class: 3.0 (Display controller) irq: 0
+PCI: 00:03.0: 8086:100e: class: 2.0 (Network controller) irq: 11
+PCI function 00:03.0 (8086:100e) enabled
 [00000000] new env 00001000
 [00000000] new env 00001001
+[00000000] new env 00001002
+[00001001] new env 00001003
 FS is running
 FS can do I/O
 Device 1 presence: 1
@@ -20,26 +32,13 @@ superblock is good
 bitmap is good
 alloc_block is good
 file_open is good
-[00001001] new env 00001002
-[00001001] user panic in primespipe at user/primespipe.c:74:
generator write: 0, error 0
-Welcome to the JOS kernel monitor!
-Type 'help' for a list of commands.
-TRAP frame at 0xf029f07c from CPU 0
-  edi  0x00000000
-  esi  0x008023ed
-  ebp  0xeebfdf90
-  oesp 0xefffffdc
-  ebx  0xeebfdfa4
-  edx  0xeebfde48
-  ecx  0x00000001
-  eax  0x00000001
-  es   0x----0023
-  ds   0x----0023
-  trap 0x00000003 Breakpoint
-  err  0x00000000
-  eip  0x008002dd
-  cs   0x----001b
-  flag 0x00000286
-  esp  0xeebfdf88
-  ss   0x----0023
-qemu-system-i386: terminating on signal 15 from pid 28013 (make)
+file_get_block is good
+file_flush is good
+file_truncate is good
+file rewrite is good
+[00001001] new env 00001004
+[00001001] new env 00001005
+ns: 52:54:00:12:34:56 bound to static IP 10.0.2.15
```

```
+NS: TCP/IP initialized.
+Waiting for http connections...
+qemu-system-i386: terminating on signal 15 from pid 16137 (make)
diff --git a/kern/e1000.c b/kern/e1000.c
index 7570e75..c5ebfaf 100644
--- a/kern/e1000.c
+++ b/kern/e1000.c
@@ -1,3 +1,151 @@
 #include <kern/e1000.h>
+#include <kern/pmap.h>
+#include <kern/env.h>
+#include <inc/string.h>

 // LAB 6: Your driver code here
+
+static volatile uint32_t * bar_0;
+static void e1000_tx_init(void);
+static void e1000_rx_init(void);
+
+static struct tx_desc tx_descriptors[E1000_MAX_TX_DESCRIPTORS]
__attribute__ ((aligned (16)));
+static struct tx_packet tx_packets[E1000_MAX_TX_DESCRIPTORS];
+
+static struct rx_desc rx_descriptors[E1000_MAX_RX_DESCRIPTORS]
__attribute__ ((aligned (16)));
+static struct rx_packet rx_packets[E1000_MAX_RX_DESCRIPTORS];
+
+// Inicializa la placa de red, es invocada por el código de PCI
(kern/pci.c) que detecta el la placa de red
+// en el puerto PCI del motherboard y luego recorre el arreglo de
dispositivos PCI
+// y matchea Vendor y Device ID para invocar a esta función de
inicialización.
+int e1000_init(struct pci_func *pcif) {
+    //pci_func_enable negotiates an MMIO region with the E1000
and stores its base and size in BAR 0
+    // (that is, reg_base[0] and reg_size[0]). This is a range of
physical memory addresses assigned to the device
+    pci_func_enable(pcif);
+
+    // Mapeamos la memoria física destinada a la placa E1000 para
I/O en direcciones virtuales de MMIO
+    bar_0 = (uint32_t *) mmio_map_region(pcif->reg_base[0], pcif-
>reg_size[0]);
+
+    // Imprimimos el estado de la placa E1000, en hexadecimal,
que es un registro de 4 bytes que está en a partir del byte 8 en
+    // el espacio de registros.
+    // cprintf("e1000 Status: %x\n", getreg(E1000_STATUS));
+
+    // Inicializacion de transmision (Exercise 5)
```

```
+    e1000_tx_init();
+    // Inicializacion de recepción (Exercise 10)
+    e1000_rx_init();
+
+    return 0;
+}
+
+uint32_t getreg(uint32_t offset) {
+    return (volatile uint32_t) bar_0[offset/4];
+}
+
+void setreg(uint32_t offset, uint32_t value) {
+    bar_0[offset/4] = value;
+}
+
+static void e1000_tx_init(void) {
+    // Escribo ceros en el arreglo de descriptores de tx para inicializar esta memoria
+    memset(tx_descriptors, 0, E1000_MAX_TX_DESCRIPTORS * sizeof(struct tx_desc));
+    // Seteo el registro TDBAL (Transmit Descriptor Base Address Low)
+    // Va la dirección física, pues la placa accede a memoria sin pasar por la MMU
+    setreg(E1000_TDBAL, PADDR(tx_descriptors));
+    // Seteo el registro TDLEN (Transmit Descriptor Lenght)
+    setreg(E1000_TDLEN, sizeof(struct tx_desc) * E1000_MAX_TX_DESCRIPTORS);
+    // Seteo el registro TDH (Transmit Descriptor Head). Debe ir en 0 según documentación
+    setreg(E1000_TDH, 0);
+    // Seteo el registro TDT (Transmit Descriptor Tail). Debe ir en 0 según documentación
+    setreg(E1000_TDT, 0);
+    // Seteo el registro TCTL (Transmit Control REGISTER) según documentación
+    setreg(E1000_TCTL, E1000_TCTL_EN | E1000_TCTL_PSP | E1000_TCTL_CT | E1000_TCTL_COLD);
+    // Seteo el registro TIPG (TX Inter-packet gap) según documentación
+    setreg(E1000_TIPG, (10 << 0) | (6 << 20) | (4 << 10));
+    // Asocio los buffers de paquete con el address de los descriptors y enciendo bit DD (descriptor se puede utilizar)
+    for (int i = 0; i < E1000_MAX_TX_DESCRIPTORS; i++) {
+        tx_descriptors[i].addr = PADDR(tx_packets[i].buffer);
+        tx_descriptors[i].status |= E1000_TXD_STAT_DD;
+    }
+}
+
+static void e1000_rx_init(void) {
+    // Escribo ceros en el arreglo de descriptores de tx para
```

```
inicializar esta memoria
+    memset(rx_descriptors, 0, E1000_MAX_RX_DESCRIPTORS *
sizeof(struct rx_desc));
+
+    // Escribo el dirección MAC de la placa en el registro
Receive Address (HARDCODED)
+    setreg(E1000_RA, 0x12005452);
+    // Escribo el dirección MAC de la placa en el registro
Receive Address (HARDCODED)
+    setreg(E1000_RA + 4, 0x5634 | E1000_RAH_AV);
+
+    // Seteo el registro MTA en 0 como indica el manual
+    //setreg(E1000_MTA, 0);
+
+    // Va la dirección física, pues la placa accede a memoria sin
pasar por la MMU
+    setreg(E1000_RDBAL, PADDR(rx_descriptors));
+    setreg(E1000_RDBAH, 0);
+
+    // Seteo el registro RDLEN (Receive Descriptor Lenght)
+    setreg(E1000_RDLEN, sizeof(struct rx_desc) *
E1000_MAX_RX_DESCRIPTORS);
+
+    // Seteo el registro RDH
+    setreg(E1000_RDH, 0);
+    // Seteo el registro RDT
+    setreg(E1000_RDT, E1000_MAX_RX_DESCRIPTORS-1);
+
+     //Seteo el registro Receive Control RCTL
+    setreg(E1000_RCTL, E1000_RCTL_EN | E1000_RCTL_BAM |
E1000_RCTL_SECRC);
+
+    // Asocio los buffers de paquete con el address de los
descriptors
+    for (int i = 0; i < E1000_MAX_RX_DESCRIPTORS; i++) {
+        rx_descriptors[i].buffer_addr =
PADDR(rx_packets[i].buffer);
+    }
+}
+
+int e1000_send_packet(char * buffer, size_t size) {
+    // Chequeamos si hay algún descriptor libre
+    // Obtenemos el índice del Tail de la cola de descriptores de
transmisión.
+    uint32_t td_tail = getreg(E1000_TDT);
+
+    // Si el tamaño a enviar es mas grande que el maximo
permitido por Ethernet entonces error
+    if (size > ETHERNET_MAX_PACKET_LEN) {
+        return -E_INVAL;
+    }
```

```
+
+    struct tx_desc * current_tx_desc = tx_descriptors + td_tail;
+
+    if (current_tx_desc->status & E1000_TXD_STAT_DD) { // El
hardware procesó este descriptor y se puede reciclar
+        memmove(tx_packets[td_tail].buffer, buffer, size);
+        current_tx_desc->length = (uint16_t) size;
+        current_tx_desc->status &=  ~E1000_TXD_STAT_DD;
+        current_tx_desc->cmd |= E1000_TXD_CMD_RS |
E1000_TXD_CMD_EOP;
+        setreg(E1000_TDT, (td_tail+1) %
E1000_MAX_TX_DESCRIPTORS);
+    } else {
+        return -E_AGAIN;    // Cola llena, se debe reintentar,
quizá el hardware libere algun descriptor
+    }
+
+    return 0;
+}
+
+int e1000_receive_packet(char * buffer, size_t size) {
+
+    uint32_t next_desc = (getreg(E1000_RDT) + 1) %
E1000_MAX_RX_DESCRIPTORS;
+
+    uint32_t pkt_len;
+
+    struct rx_desc * current_rx_desc = rx_descriptors +
next_desc;
+
+    if (current_rx_desc->length > size) return -E_INVAL;
+
+    if (current_rx_desc->status & E1000_RXD_STAT_DD) { //  If the
DD bit is set, you can copy the packet data out of that
descriptor's packet buffer
+        memmove(buffer, rx_packets[next_desc].buffer, (size_t)
current_rx_desc->length);
+        pkt_len = current_rx_desc->length;
+        current_rx_desc->status &= ~E1000_RXD_STAT_DD;
+        current_rx_desc->status &= ~E1000_RXD_STAT_EOP;
+        setreg(E1000_RDT, next_desc);
+    } else {
+        return -E_AGAIN;    // No se recibió paquete.
+    }
+
+    return pkt_len;
+}
\ No newline at end of file
diff --git a/kern/e1000.h b/kern/e1000.h
index abdf80d..72d708d 100644
```

```
--- a/kern/e1000.h
+++ b/kern/e1000.h
@@ -1,4 +1,95 @@
 #ifndef JOS_KERN_E1000_H
 #define JOS_KERN_E1000_H

+#include <inc/types.h>
+#include <kern/pci.h>
+
+#define ETHERNET_MAX_PACKET_LEN 1518
+
+#define E1000_STATUS   0x00008  /* Device Status - RO */
+#define E1000_TDBAL    0x03800  /* TX Descriptor Base Address Low
- RW */
+#define E1000_TDLEN    0x03808  /* TX Descriptor Length - RW */
+#define E1000_TDH      0x03810  /* TX Descriptor Head - RW */
+#define E1000_TDT      0x03818  /* TX Descripotr Tail - RW */
+#define E1000_TCTL     0x00400  /* TX Control - RW */
+#define E1000_TCTL_EN     0x00000002    /* enable tx */
+#define E1000_TCTL_PSP    0x00000008    /* pad short packets */
+#define E1000_TCTL_CT     0x00000100    /* collision threshold */
+#define E1000_TCTL_COLD   0x00040000    /* collision distance */
+#define E1000_TIPG     0x00410  /* TX Inter-packet gap -RW */
+
+#define E1000_TXD_STAT_DD    0x00000001 /* Descriptor Done */
+
+#define E1000_TXD_CMD_RS     0x00000008 /* Report Status */
+#define E1000_TXD_CMD_RPS    0x00000010 /* Report Packet Sent */
+#define E1000_TXD_CMD_EOP    0x00000001 /* End of Packet */
+
+#define E1000_RCTL_LBM_NO         0x00000000    /* no loopback
mode */
+#define E1000_RCTL_RDMTS_HALF     0x00000000    /* rx desc min
threshold size */
+#define E1000_RCTL_MO_0           0x00000000    /* multicast
offset 11:0 */
+#define E1000_RCTL_SZ_2048        0x00000000    /* rx buffer size
2048 */
+#define E1000_RCTL_LPE            0x00000020    /* long packet
enable */
+#define E1000_RCTL_BSEX           0x02000000    /* Buffer size
extension */
+
+#define E1000_RAH_AV  0x80000000        /* Receive descriptor
valid */
+#define E1000_RA       0x05400  /* Receive Address - RW Array */
+#define E1000_MTA      0x05200  /* Multicast Table Array - RW
Array */
+#define E1000_IMS      0x000D0  /* Interrupt Mask Set - RW */
+#define E1000_RDBAL    0x02800  /* RX Descriptor Base Address Low
- RW */
```

```
+#define E1000_RDBAH    0x02804  /* RX Descriptor Base Address
High - RW */
+#define E1000_RDLEN    0x02808  /* RX Descriptor Length - RW */
+#define E1000_RDH      0x02810  /* RX Descriptor Head - RW */
+#define E1000_RDT      0x02818  /* RX Descriptor Tail - RW */
+#define E1000_RCTL     0x00100  /* RX Control - RW */
+#define E1000_RCTL_EN             0x00000002    /* enable */
+#define E1000_RCTL_BAM            0x00008000    /* broadcast
enable */
+#define E1000_RCTL_SECRC          0x04000000    /* Strip Ethernet
CRC */
+#define E1000_RXD_STAT_DD       0x01    /* Descriptor Done */
+#define E1000_RXD_STAT_EOP      0x02    /* End of Packet */
+
+#define E_AGAIN 1  /* Error, la cola está llena */
+#define E_INVAL 2  /* Error, paquete inválido */
+
+#define E1000_MAX_TX_DESCRIPTORS 64
+#define E1000_MAX_RX_DESCRIPTORS 128
+#define E1000_RCV_BUFFER_SIZE 2048
+
+
+
+struct tx_desc
+{
+       uint64_t addr;
+       uint16_t length;
+       uint8_t cso;
+       uint8_t cmd;
+       uint8_t status;
+       uint8_t css;
+       uint16_t special;
+}__attribute__((packed));;
+
+struct rx_desc {
+    uint64_t buffer_addr; /* Address of the descriptor's data
buffer */
+    uint16_t length;     /* Length of data DMAed into data buffer
*/
+    uint16_t csum;       /* Packet checksum */
+    uint8_t status;      /* Descriptor status */
+    uint8_t errors;      /* Descriptor Errors */
+    uint16_t special;
+}__attribute__((packed));;
+
+struct tx_packet
+{
+    uint8_t buffer[ETHERNET_MAX_PACKET_LEN];
+}__attribute__((packed));;
```

```
+
+struct rx_packet
+{
+    uint8_t buffer[E1000_RCV_BUFFER_SIZE];
+}__attribute__((packed));;
+
+int e1000_init(struct pci_func *pcif);
+uint32_t getreg(uint32_t offset);
+void setreg(uint32_t offset, uint32_t value);
+int e1000_send_packet(char * buffer, size_t size);
+int e1000_receive_packet(char * buffer, size_t size);
+
 #endif  // JOS_KERN_E1000_H
diff --git a/kern/pci.c b/kern/pci.c
index 784e072..bcc01f1 100644
--- a/kern/pci.c
+++ b/kern/pci.c
@@ -31,6 +31,7 @@ struct pci_driver pci_attach_class[] = {
 // pci_attach_vendor matches the vendor ID and device ID of a PCI device. key1
 // and key2 should be the vendor ID and device ID respectively
 struct pci_driver pci_attach_vendor[] = {
+       { 0x8086, 0x100E, e1000_init},
        { 0, 0, 0 },
 };

diff --git a/kern/syscall.c b/kern/syscall.c
index 12de9cb..fd9e397 100644
--- a/kern/syscall.c
+++ b/kern/syscall.c
@@ -12,6 +12,7 @@
 #include <kern/console.h>
 #include <kern/sched.h>
 #include <kern/time.h>
+#include <kern/e1000.h>

 // Funcion propia para validar direcciones
 // de memoria virtual pasadas a la syscall
@@ -540,7 +541,22 @@ static int
 sys_time_msec(void)
 {
        // LAB 6: Your code here.
-       panic("sys_time_msec not implemented");
+       return time_msec();
+       //panic("sys_time_msec not implemented");
+}
+
```

```
+static int
+sys_transmit_packet(char * buffer, size_t size)
+{
+       if (!buffer || ((uintptr_t) buffer >= UTOP)) return -
E_INVAL;
+       return e1000_send_packet(buffer, size);
+}
+
+static int
+sys_receive_packet(char * buffer, size_t size)
+{
+       if (!buffer || ((uintptr_t) buffer >= UTOP)) return -
E_INVAL;
+       return  e1000_receive_packet(buffer, size);
 }

 // Dispatches to the correct kernel function, passing the
arguments.
@@ -595,7 +611,15 @@ syscall(uint32_t syscallno, uint32_t a1,
uint32_t a2, uint32_t a3, uint32_t a4,
        case SYS_env_set_trapframe: {
                return (int32_t)
sys_env_set_trapframe((envid_t)a1, (struct Trapframe*)a2);
        }
-
+       case SYS_time_msec: {
+               return (int32_t) sys_time_msec();
+       }
+       case SYS_transmit_packet: {
+               return (int32_t) sys_transmit_packet((void *)a1,
(size_t)a2);
+       }
+       case SYS_receive_packet: {
+               return (int32_t) sys_receive_packet((void *)a1,
(size_t) a2);
+       }
        default:
                return -E_INVAL;
        }
diff --git a/kern/trap.c b/kern/trap.c
index 0826e97..6652a5e 100644
--- a/kern/trap.c
+++ b/kern/trap.c
@@ -306,6 +306,7 @@ trap_dispatch(struct Trapframe *tf)
        }
        case IRQ_OFFSET + IRQ_TIMER: {
                lapic_eoi();            // Avisamos al hardware
que atrapamos la interrupcion
+               time_tick();            // Incrementamos el
contador de interrupciones de clock
```

```
                sched_yield();  // Actuamos en consecuencia de la
interrupcion (round-robin)
                return;
        }
@@ -351,7 +352,7 @@ trap_dispatch(struct Trapframe *tf)


        // Handle keyboard and serial interrupts.
-       // LAB 5: Your code here.
+       // LAB 5: Your code here

        // Unexpected trap: The user process or the kernel has a
bug.
        print_trapframe(tf);
diff --git a/lib/syscall.c b/lib/syscall.c
index 9e1a1d9..b351805 100644
--- a/lib/syscall.c
+++ b/lib/syscall.c
@@ -122,3 +122,15 @@ sys_time_msec(void)
 {
        return (unsigned int) syscall(SYS_time_msec, 0, 0, 0, 0,
0, 0);
 }
+
+int
+sys_transmit_packet(char * buffer, size_t size)
+{
+       return syscall(SYS_transmit_packet, 0, (uint32_t) buffer,
(uint32_t) size, 0, 0, 0);
+}
+
+int
+sys_receive_packet(void * buffer, size_t size)
+{
+       return syscall(SYS_receive_packet, 0, (uint32_t) buffer,
(uint32_t) size, 0, 0, 0);
+}
\ No newline at end of file
diff --git a/net/input.c b/net/input.c
index 4e08f0f..bc5cc58 100644
--- a/net/input.c
+++ b/net/input.c
@@ -2,15 +2,39 @@

 extern union Nsipc nsipcbuf;

+
 void
 input(envid_t ns_envid)
```

```
 {
-       binaryname = "ns_input";
-
-       // LAB 6: Your code here:
-       //      - read a packet from the device driver
-       //      - send it to the network server
-       // Hint: When you IPC a page to the network server, it
will be
-       // reading from it for a while, so don't immediately
receive
-       // another packet in to the same physical page.
-}
+    binaryname = "ns_input";
+
+    // LAB 6: Your code here:
+    //  - read a packet from the device driver
+    //  - send it to the network server
+    // Hint: When you IPC a page to the network server, it will
be
+    // reading from it for a while, so don't immediately receive
+    // another packet in to the same physical page.
+
+    int32_t r;
+    int32_t len;
+
+    //struct jif_pkt * pkt = (struct jif_pkt *) REQVA;
+    union Nsipc * pkt = (union Nsipc *) REQVA;
+    sys_page_alloc(0, pkt, PTE_P | PTE_W | PTE_U);
+
+    while(1) {
+        while ( (len = sys_receive_packet(pkt->pkt.jp_data,
2048)) < 0) {
+            sys_yield();
+        }
+
+        pkt->pkt.jp_len = len;
+
+        while ((r = sys_ipc_try_send(ns_envid, NSREQ_INPUT,
&(pkt->pkt), PTE_P | PTE_W | PTE_U)) < 0) {
+            if (r == -E_IPC_NOT_RECV) sys_yield();
+        }
+
+        sys_page_unmap(0, pkt);
+        pkt = (union Nsipc *) REQVA;
+        sys_page_alloc(0, pkt, PTE_P | PTE_W | PTE_U);
+    }
+}
\ No newline at end of file
diff --git a/net/output.c b/net/output.c
```

```
index f577c4e..dc05514 100644
--- a/net/output.c
+++ b/net/output.c
@@ -1,4 +1,5 @@
 #include "ns.h"
+#include <inc/lib.h>

 extern union Nsipc nsipcbuf;

@@ -10,4 +11,18 @@ output(envid_t ns_envid)
        // LAB 6: Your code here:
        //      - read a packet from the network server
        //      - send the packet to the device driver
+       envid_t from_env;
+       int32_t value;
+
+       while(1) {
+               value = ipc_recv(&from_env, &nsipcbuf, NULL);
+
+               if ((from_env != ns_envid) || (value != NSREQ_OUTPUT)) continue;
+
+               do {
+                       value = sys_transmit_packet(nsipcbuf.pkt.jp_data, nsipcbuf.pkt.jp_len);
+               } while(value < 0);
+       }
 }
+
+
diff --git a/user/httpd.c b/user/httpd.c
index af1979a..6a87ea9 100644
--- a/user/httpd.c
+++ b/user/httpd.c
@@ -78,7 +78,24 @@ static int
 send_data(struct http_request *req, int fd)
 {
        // LAB 6: Your code here.
-       panic("send_data not implemented");
+       int r, w = 0;
+       char * buffer[BUFFSIZE];
+       r = read(fd, buffer, BUFFSIZE);
+
+       while (r > 0) {
+               w = write(req->sock, buffer, BUFFSIZE);
+               if (w < 0) break;
+               r = read(fd, buffer, BUFFSIZE);
```

```
+       }
+
+       if (r < 0) {
+               panic("send_data failed reading data on fd %i.\n",
fd);
+       }
+       if (w < 0) {
+               panic("send_data failed writing data on req fd
%i.\n", req->sock);
+       }
+
+       return 0;
 }

 static int
@@ -222,14 +239,31 @@ send_file(struct http_request *req)
        int r;
        off_t file_size = -1;
        int fd;
+       struct Stat statbuf;

        // open the requested url for reading
+       fd = open(req->url, O_RDWR);
        // if the file does not exist, send a 404 error using
send_error
+       if (fd < 0) {
+               send_error(req, 404);
+               r = fd;
+               goto end;
+       }
+
+       r = fstat(fd, &statbuf);
+       if (r < 0) {
+               panic("send_file failed on fsstat %i\n", r);
+       }
+
        // if the file is a directory, send a 404 error using
send_error
-       // set file_size to the size of the file
+       if (statbuf.st_isdir) {
+               send_error(req, 404);
+               r = -1;
+               goto end;
+       }

-       // LAB 6: Your code here.
-       panic("send_file not implemented");
+       // set file_size to the size of the file
```

```
+       file_size = statbuf.st_size;

        if ((r = send_header(req, 200)) < 0)
                goto end;
```