Networking in STOS

Ivan Delalande

colona@lse.epita.fr
http://lse.epita.fr

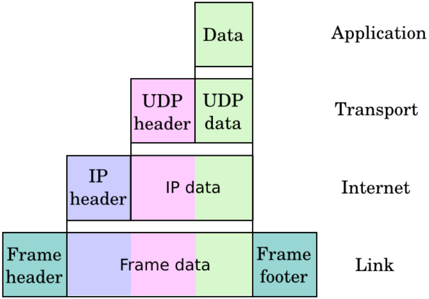
LSE Summer Week 2013

Outline

1. e1000 driver

2. Network Stack

Introduction
Goals

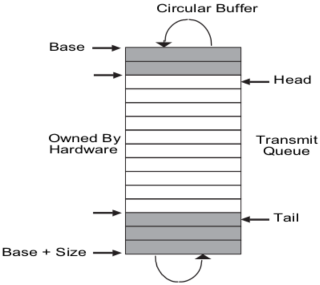


- Build a full network stack for the STOS kernel,
- provide userland access to the network using syscalls.

e1000 driver

e1000 Driver

Previously on e1000TV. . .



- Send data,
- read data from the device's input ring,
- NO interrupt handling.

Interrupt handling

Revenge of the PCI

- The card uses PCI interrupts for event notifications (packet reception, changes of connection state. . .),
- PCI interrupts are shared,
- 4 interrupt lines for all the PCI devices.

Interrupt handling
APIC

- The interrupts go through the APIC controller,
- this controller allow the customization of interrupt numbers,
- configuration of the PCI interrupt lines are stored in the MP tables and ACPI tables.

Interrupt handling

PCI interrupt handling

- The device generates an interrupt on the right interrupt line,
- the PCI system in the kernel call all the handlers of the drivers associated with this interrupt line,
- each handler checks if the interrupt concerns its device by checking an internal state and advising the PCI system.

Interrupt handling

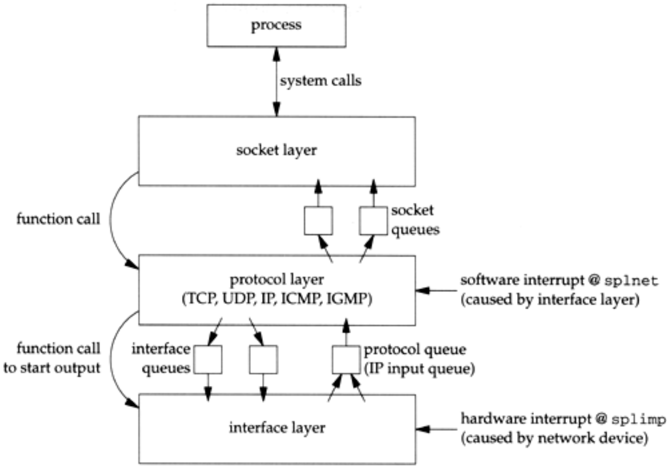
Configuration

- **Lots of** configuration options to reduce the amount of interruptions sent by the device,
- possibility to configure various threshold values for the input data ring.

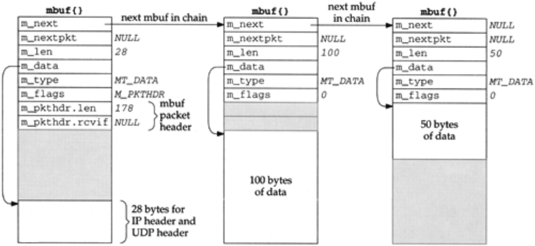
Network Stack

- Architecture defined by the *Net/1* distribution of *4.3BSD*, finalized in the *4.4BSD* distribution *Net/3*,
- used by many unix derivatives,
- explained in depth in *TCP/IP Illustrated, Volume 2: The Implementation* by Gary R. Wright and W. Richard Stevens.

Network Stack
Interfaces

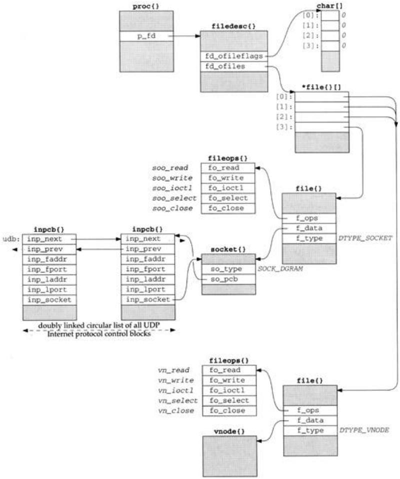


Network Stack
Mbuf



- Memory buffer,
- store one packet,
- designed for easy encapsulation and decapsulation,
- used up to the transport layer (TCP, UDP, SCTP...).

Network Stack
Socket file descriptors



- Allow users to manipulate connections as sockets, which are regular file descriptors,
- packet chain stored in the `struct file`.

Network Stack

Sending data

- User write data on a socket,
- data is split in packets,
- all the protocols' headers are added to the packets,
- the resulting frames are enqueued in the output data ring of the device.

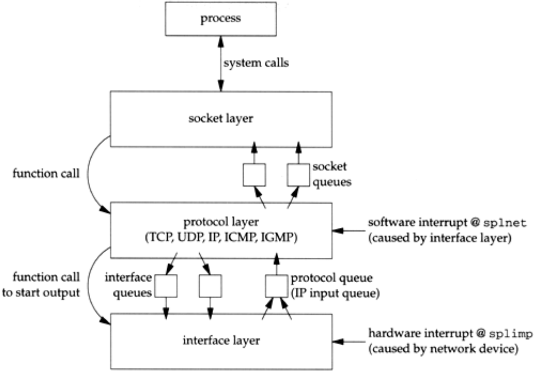
- Device receive data from the medium, store them in the input data ring and trigger an interrupt,
- data is copied to a `mbuf` chain passed to the ethernet layer,
- ethernet headers are checked and removed,
- data is stored in the IP input queue and the interrupt handler returns,
- in another kernel thread, the IP input handler processes data in the input queue, so does the transport layer,
- the user process is woken up and its `recv` returns with the data.

Conclusion

- Continue with the network layer (routing, fragmentation. . .),
- implements a raw socket interface, with `socket(AF_PACKET, {SOCK_RAW,SOCK_DGRAM}, )`
  - `SOCK_RAW`: direct i/o at the level of the device driver,
  - `SOCK_DGRAM`: ethernet header are added and removed by the kernel.

FIN, ACK&FIN, ACK

Thanks for your attention



process
system calls
socket layer
function call
socket
queues
protocol layer
(TCP, UDP, IP, ICMP, IGMP)
software interrupt @ splnet
(caused by interface layer)
function call
to start output
interface
queues
protocol queue
(IP input queue)
interface layer
hardware interrupt @ splimp
(caused by network device)