```
* Device drivers (receive and send data).

User vs. Kernel separation so far
Actually: User -> kernel -> hardware

NIC: hardware
- sits on a bus (PCI), and communicates with rest of computer
- Ethernet ports to talk to outside world
- has a processor
- memory (DRAM)
- a program running (firmware)
=> a NIC is a like a mini computer with a mind of its own.
=> every other peripheral device is also its own "mini computer"
=> so even on a single core CPU, you still have a "concurrent/distributed system"

NIC reading/receiving data:

1. NIC listens on wire (RJ45 Ethernet) for packets that belong to it
- look for data that is preceded by your own MAC (Ethernet) address (e.g.,
  48 bit)
2. If found bits that appear to belong to this NIC, then start to read them
   into my own memory buffers.
- if no free NIC buffer space, then "drop" packet (i.e., never receive it in
  the first place). Can be full b/c of previously received packets.
- if packet never received, then sender wouldn't get an ACK, then try to
  retransmit some time later.
- in TCP, you exponentially back off each time you don't get an ACK:
  senders essentially throttle themselves.
3. If have space in NIC mem, read entire packet into NIC ram and store it
   there.  Maybe do basic validity checks.
4. NIC has to give packet to OS quickly, so it doesn't take up NIC ram for
   too long (don't want NIC ram to be full)
5. NIC would interrupt the main CPU, actual h/w interrupt.
------
inside OS:
6. OS kernel will stop what's doing, save state, and invoke the interrupt
   handle routine for "networking".
- if OS is "busy", it won't take the interrupt:
        could be processing another interrupt (inside the ihandler)
        could be inside an important critical section that cannot be stopped.
7. networking interrupt handler: take packet from NIC and store in an SKbuf
7a. grab a free/unused skbuf (faster than trying to alloc a new skbuf)
7b. copy data from NIC to skbuf (fast)
- use actual I/O instructions (slow)
- or use DMA (direct mem access), which is async too.
7c. now you have data in some SKB
7d. signal (in/directly) to NIC that data was transferred to main RAM successfully
7e. add SKB into queue of "just arrived" new packets that need to be
    processed. (some other kthread will handle these packets, TBD)
------
Back inside NIC:
8. NIC can now free the space used by the packet, and make room for newer
   packets to be received.

NIC writing/sending data:

OS wants to send a packet to some destination
- packet is in some SKB
- SKB sitting on some queue of packets ready to be transmitted
Inside the OS networking device driver:
- need a way to xfer data (DMA, processor instructions)
- need a way for CPU to know if NIC is able to take data
        this is done by h/w device communicating w/ CPU over a special wire
        (b/t NIC on PCI bus and CPU), called the xon/xoff state.
```

```
- if NIC sets itself to LINK_STATE_XOFF state, the CPU can query a register
  with all XON/XOFF states, find out that the NIC's XON/XOFF is set to XOFF,
  and hence it means that the NIC is "busy".  If busy, CPU effectively
  self-throttles.
- when LINK_STATE is XON, CPU will send data from skb to NIC (via CPU
  instrux or DMA).
- when done, CPU will free this SKB, and resume other processing.
- at the NIC, now it's responsible for sending packet
- NIC will sense the wire until it is quiet (carrier sense techniques)
- once the wire is quiet, xmit bits on the wire
- once bits xmitted on wire, NIC can free its own memory

Inside OS, async net processing
- In old days, net ihandler took care of full pkt processing, all the way to
  handing it off to a process waiting on data, or getting data from process.
- nowadays NIC works "quickly", and rest is done by async kthreads (Soft
IRQs)

Linux Software interrupts (softirq):
- design multiple producer/consumer queues
- each queue has a designated softirq (a bit in a bitmap):
        SOFTIRQ_NET_TX: queue for transmitting network packets
        SOFTIRQ_NET_RX: queue for receiving network packets
        other queues for user/kernel route management, etc
        kernel devel. can add new SOFTIRQs
- kernel runs a [ksoftirqd/cpuN] kthreads, where N is 0, 1, 2, 3 (up to max
  of cpu cores)
  Note: anything in [square brackets] in "ps -ef" is a kthread.
- back inside a net ihandler receiving packet:
        a. ihandler has an SKB w/ newly received data
        b. act as producer, add SKB to the appropriate softirq queue (e.g.,
           SOFTIRQ_NET_RX).
        c. set the SOFTIRQ_NET_RX bit in a global bitmap of softirqs to be
           processed.
- later on, scheduler will wake up, look for any SOFTIRQs that are "on"
  (meaning there's work to be done), and therefore wakeup one or more
  ksoftirqd/cpuN kthreads.
- when one of these ksoftirqd/cpuN kthreads runs, it looks to see which
  queues have work, picks a queue, pulls an item from the queue, and then
  processes it.

In networking, each layer has a struct to describe the work to be done:
Ethernet, IP, UDP/TCP, etc.
- at lowest layer, ksoftirqd/cpuN verifies checksum, processes incoming
  Ethernet frames, assembles as needed, and then produce a new object
  containing an skb plus work for the next layer up.
- each layer processes its own objects (as a consumer), and produces new
  work to the next layer up, going from Ethernet layer queues, to IP, to
  TCP/UDP, etc. all the way up to a user process waiting for the data.
- once we have data ready for user process, we copy it from skb to __user
  buf, then move process from WAITING to READY state.
- on receiving bottom-most layers' queues can fill up, resulting in next layer
  filling, all the way to user process being throttled

Same process in reverse:
- user process writes to a socket
- packets moved into queues down the layers all the way to NIC
- if NICs busy, eventually user process writing data will block.
- so OS has time to process packets in/out

If ksoftirqd/cpuN wakes up, and finds that there's a LOT of work, in all net
queues in the system, at all layers, which ones should it process first?
1. If sending packets, then lowest layer (give to NIC asap)
2. if receiving packets, then uppermost layer (give to a process asap)
```