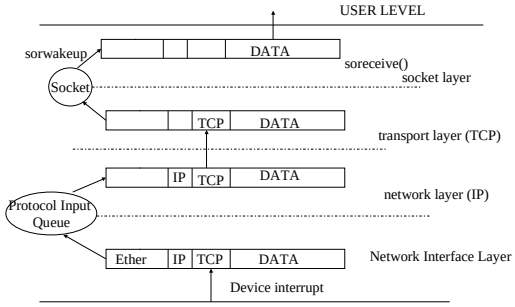
Fbufs: A High-Bandwidth Cross-Domain Transfer Facility

- Networking layers in the kernel
  - Reference: Chapter 12 in "The Design and Implementation of the 4.4 BSD Operating System"
- Fbufs: optimizations for high-bandwidth cross-domain transfer

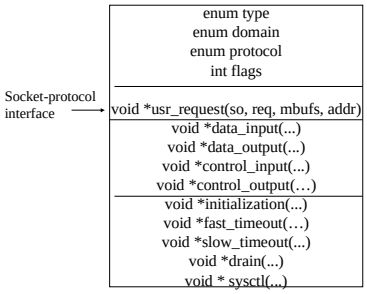
Data Flow in Network Subsystem



Passing Data through Layers

- chains of mbufs
  - 128-byte buffers used to hold data
  - header includes start and length of valid data in buffer to facilitate adding and trimming headers

Protocol Switch Structure



Socket-Protocol Interactions

- connect, bind, listen, accept, send, etc. are all called through usr_request(…)
- for receiving data, soreceive tries to read data off the buffer
- if data not available, soreceive put the process to sleep
- transport layer calls sorwakeup to wakeup soreceive when data is available

Protocol-Protocol Interactions

- pr_output(pcb, mbufs): upper level protocol calls lower level protocol
- pr_input(…, mbufs, len): lower level protocol calls upper level protocol
- pr_ctlinput(req,sock_addr): reporting ICMP type of conditions, lower level calls upon upper level

Network Subsystem in Linux

- Basic design similar to 4.4BSD
- protocol switch table slightly different
  - struct proto
  - individual functions for connect, listen, sendmsg, recvmsg, etc.

Fbufs

- Motivation: networks are getting 1Gb/s, while OS becomes the bottleneck
  - the copying overhead
- Solution: better buffer management
  - typical buffer management:
    - protocol data unit
    - application data unit

I/O buffer characteristics

- must support contiguous and non-contiguous aggregate buffer
- buffer allocated specially for I/O data
- I/O data path is known at time of allocation
- immutable buffers only
- two ways to enforce immutability: copy-on-write or lazily raise protection
- pageable buffers

Fbufs Basic Design

Steps

- 1. allocation (by originator)
  - a. find and allocate free virtual address range
  - b. allocation physical memory and clear contents
  - c. update physical page tables
- 2. send (by originator)
  - a. generate a list of fbufs from the aggregate objects
  - b. raise protection on originator
  - c. update physical page table, ensure TLB/cache consistency

Fbufs basic design

- 3. receive (by receiver)
  - a. find and reserve a free virtual address range
  - b. update physical page table
  - c. construct an aggregate object from the list of fbufs
- 4. free an aggregate object
  - a. deallocate virtual address range
  - b. update physical page table, ensure TLB/cache consistency
  - c. free physical memory pages if no reference

Optimizations

- Restricted virtual address allocation: all domains has the same fbuf region
  - eliminate step 3a
- eliminate copy-on-write by making write access illegal

Fbuf caching

- fbuf caching:
  - get rid of the free operation, instead, change back the permission for the originator to write and put the fbuf back on the freelist for the I/O path
  - if fbuf is cached, eliminate 1a-c, 2a-b, 4a-c, and no page clearing
  - only two page-table operations per transfer
  - caching improves cache locality

Transfering aggregate objects

- transfer aggregate objects in a fbuf
  - since all regions are at the same address in all domains, no need to convert pointers
  - eliminates step 2a and 3c
- mark pages as volatile:
  - no change to originator domain's permission
  - eliminate page table operations