enfabrica

Merging the Networking Worlds

David Ahern, Shrijeet Mukherjee

October 2022

BSD Socket APIs

enfabrica

Simple, easy to use, well understood

Control path:

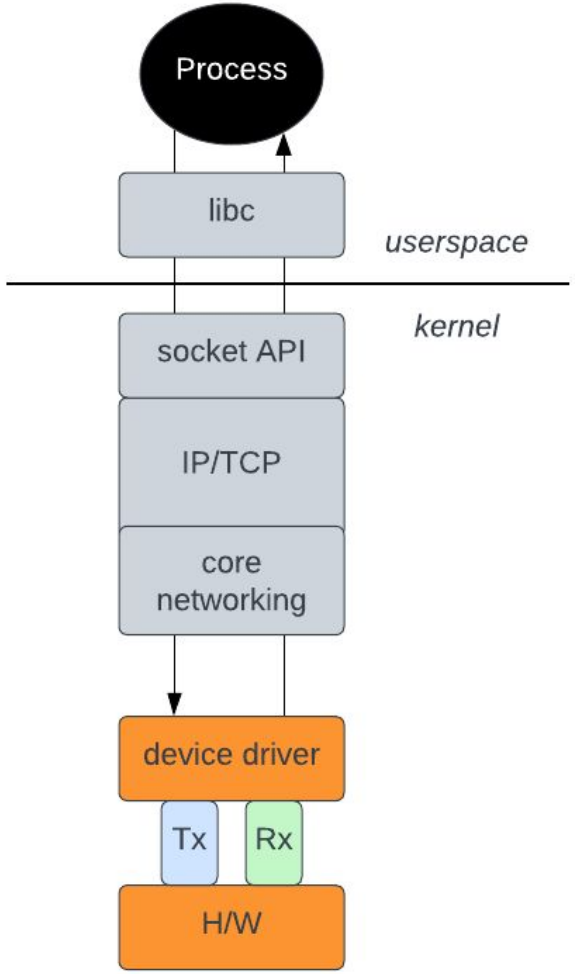
- socket(), [bind(),] { connect(), listen() + accept() }

Data path:

- recvmsg(), sendmsg()

libc provides an application interface to the kernel APIs

Relies on OS for network addressing, routing, and interaction with H/W



BSD Socket APIs

enfabrica

Simple, easy to use, well understood

Control path:

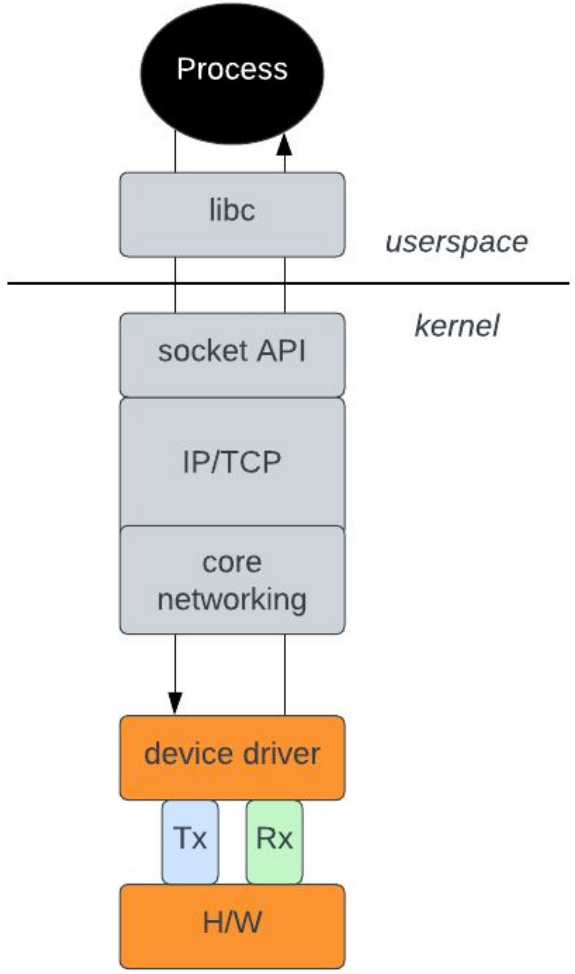
- socket(), [bind(),] { connect(), listen() + accept() }

Data path:

- recvmsg(), sendmsg()

libc provides an application interface to the kernel APIs

Well known overhead in the data path affecting performance (both throughput and latency) and CPU cycles for a packet load



Overhead with Socket APIs and Linux

enfabrica

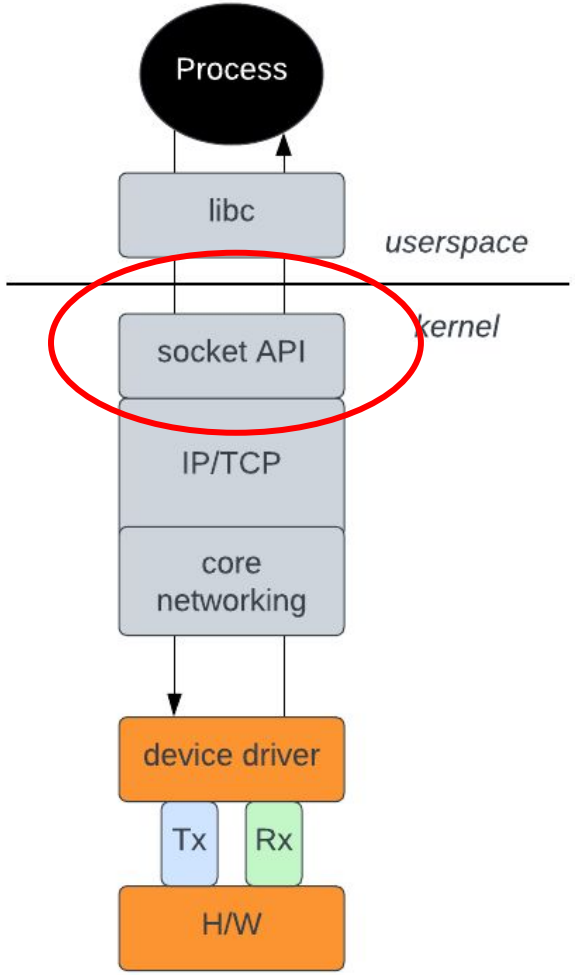
system calls

- send or receive a message
- poll/select waiting for a message

memcpy on each send and receive

- the biggest limiter to performance

kernel buffer allocation for Tx



Overhead with Socket APIs and Linux

enfabrica

Generic infrastructure hooks in the data path

- GRO packet processing on Rx
- packet sockets
- netfilter, tc, ebpf hooks

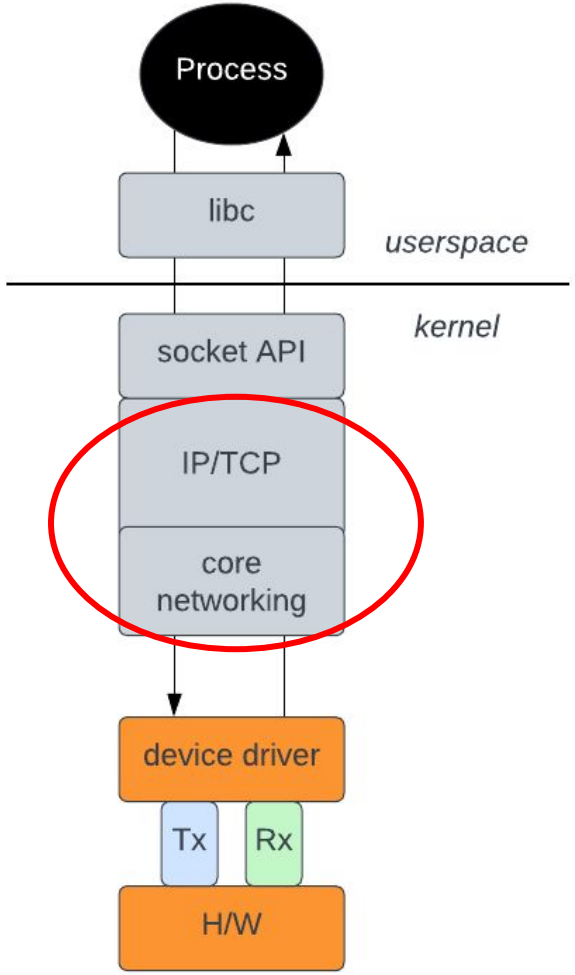
L3 - FIB lookup

- where is packet going (local or forward)

Protocol Headers (Network and Transport)

- Tx: create, Rx: validate

Socket lookup (Rx)



Overhead with Socket APIs and Linux

enfabrica

Memory management for receiving packets

- Rx ring needs buffers to land packets

skb management (Rx)

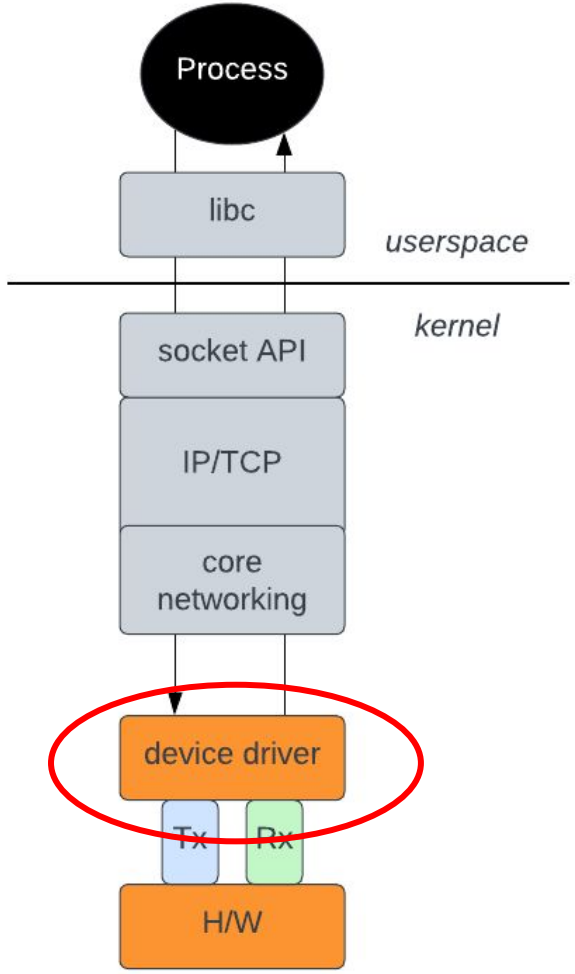
- allocation, payload representation (skb frag)

DMA mapping skb fragments (Tx)

irq latency

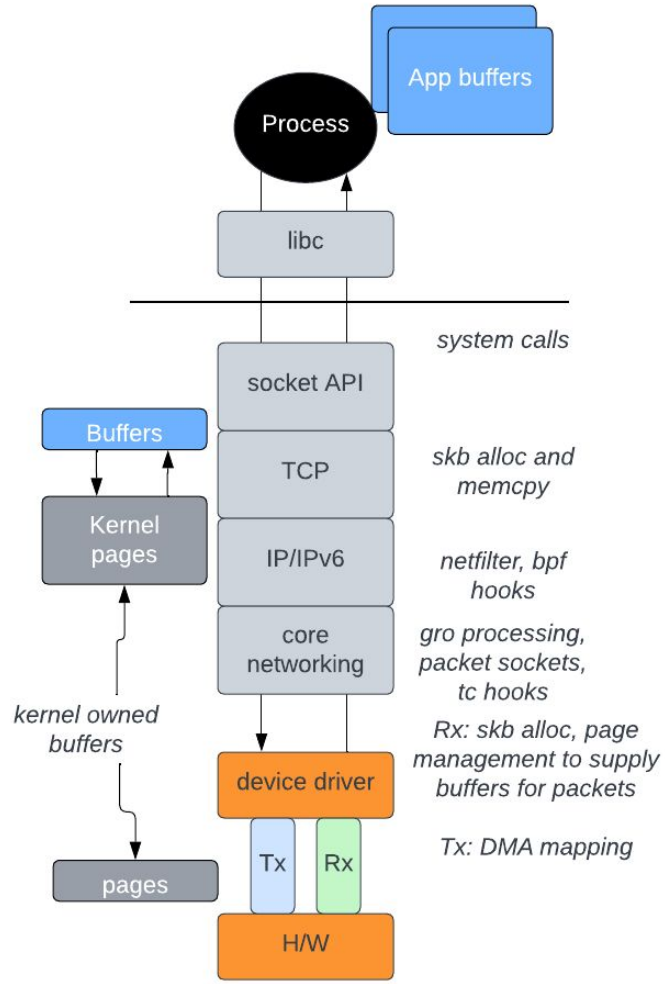
CPU cache locality

- CPU processing packets and CPU running process



Socket API Overhead

How to keep the good parts of Linux - e.g, established protocols, TCP congestion algorithms, well known admin tools - but remove overhead from the datapath?



Linux Zerocopy APIs

enfabrica

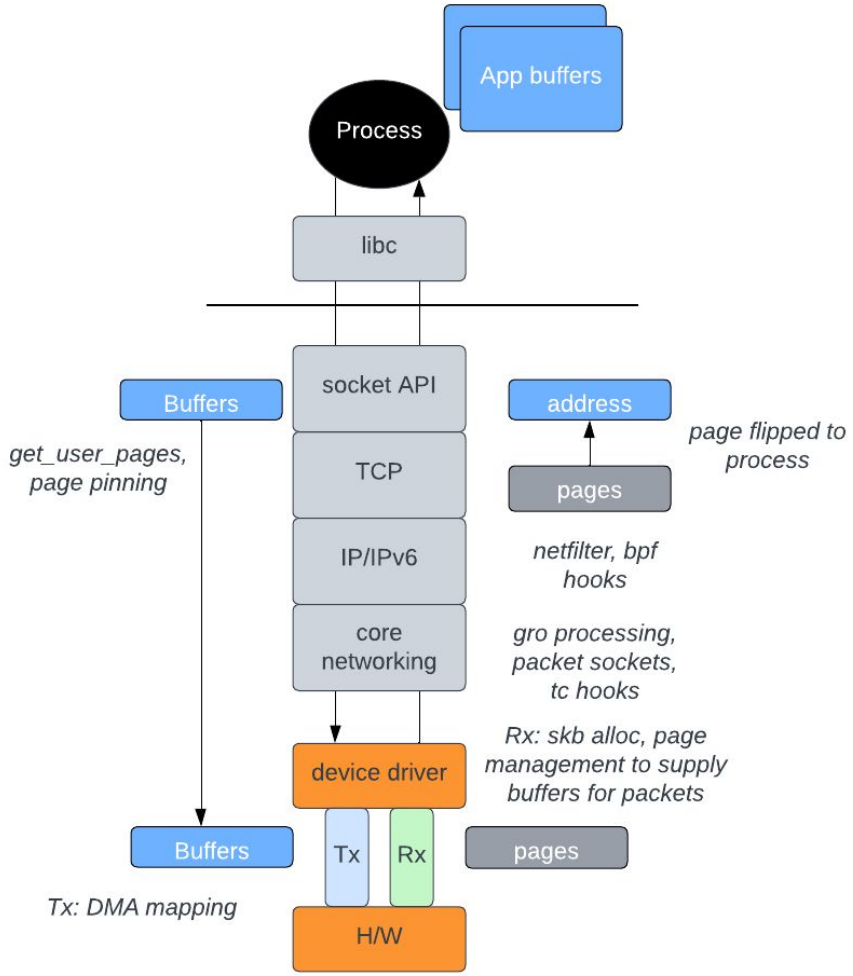
ZC API's target memcpy overhead

Tx: fairly easy to use, but has its overhead

- get_user_pages (and variants) plus reaping completions (recvmsg syscalls)

Rx: very limited and tricky to use

- Requires a specific MTU size and header split such that payloads are exactly PAGE_SIZE
- Side band with memcpy for data less than PAGE_SIZE



io_uring

Addresses performance issues at user-kernel interface

- Reduces system calls for datapath (batching submissions)

Buffer registration

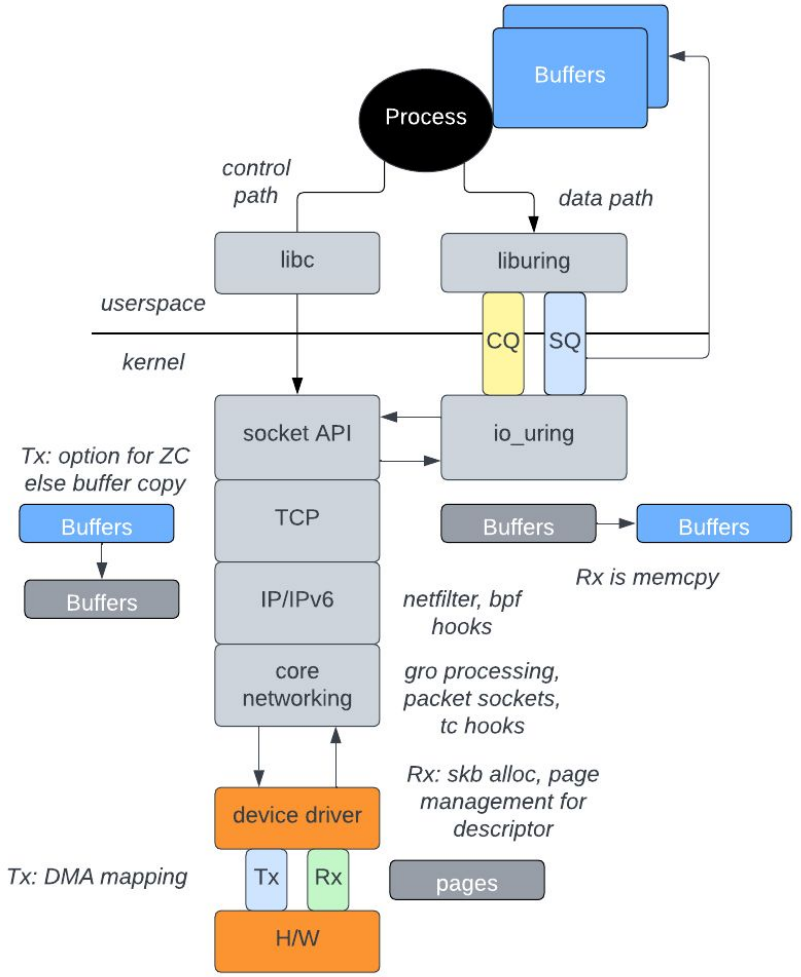
- Amortize page pinning and reference counts

liburing provides an interface to the kernel APIs

Supports networking APIs - recvmsg/sendmsg invoked kernel side with buffers from SQ

- Recent support for Tx ZC API

Still have a lot of overhead in the networking stack and hardware interface



XDP Sockets

enfabrica

Full kernel bypass

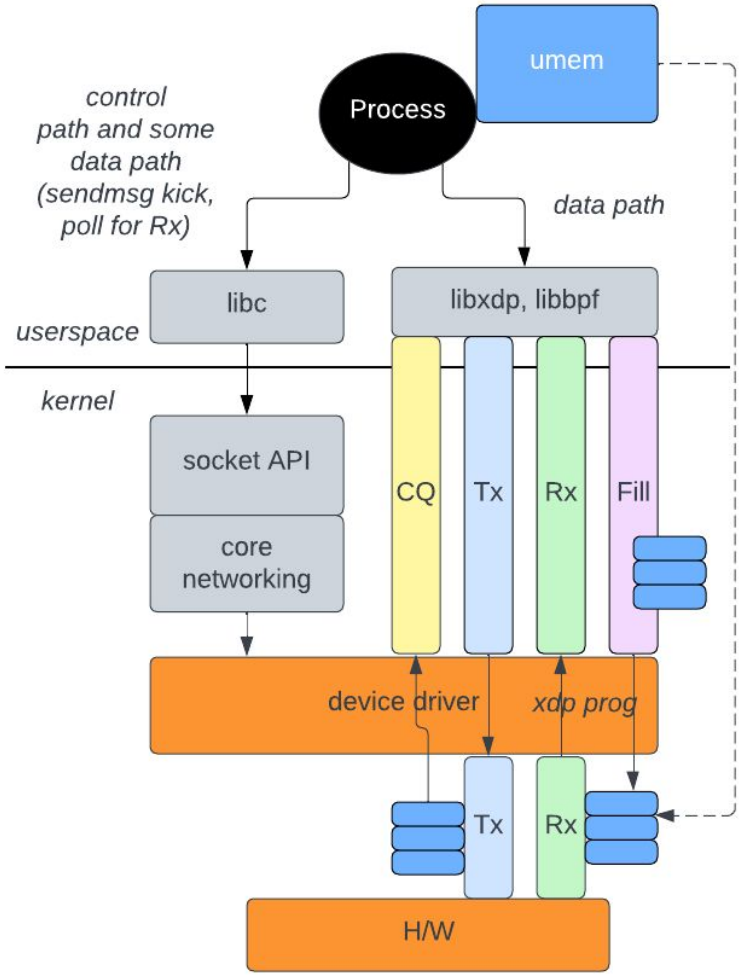
- All of the data path hooks and networking protocols
- Linux stack used only for setup and data path kicks

Userspace has to implement packet processing (and network protocols) of interest

libxdp + libbpf provide an interface to the kernel APIs

User buffers can be used in hardware queues

- Example of ownership of application buffers being cycled between H/W and process



RDMA, IB Verbs

enfabrica

Targeted at high performance computing

- High throughput, low latency

RDMA and Infiniband are a separate ecosystem

- Hardware, protocols, software

Allows application to application transfers without OS involvement

- Stated another way: OS is used for control plane setup and then gets out of the way

Existing subsystem in Linux

libibverbs provides an interface to the kernel APIs

RoCE

- RDMA over ethernet; v2 uses UDP and IP/IPv6

RDMA, IB Verbs Concepts

enfabrica

Memory Regions (MR)

- Allows a program to describe a set of contiguous memory (virtual or physical)
- Pages must be pinned to avoid being swapped out and keep virtual-to-physical mapping
- On registration virtual-to-physical conversion is written to hardware
- Set permissions on the MR (local write, remote read, remote write, atomic, bind)

Queue Pairs (QP)

- Set of receive and transmit queues; used to submit work requests

Completion Queue (CQ)

- Notification of completed work entries

Protection Domain (PD)

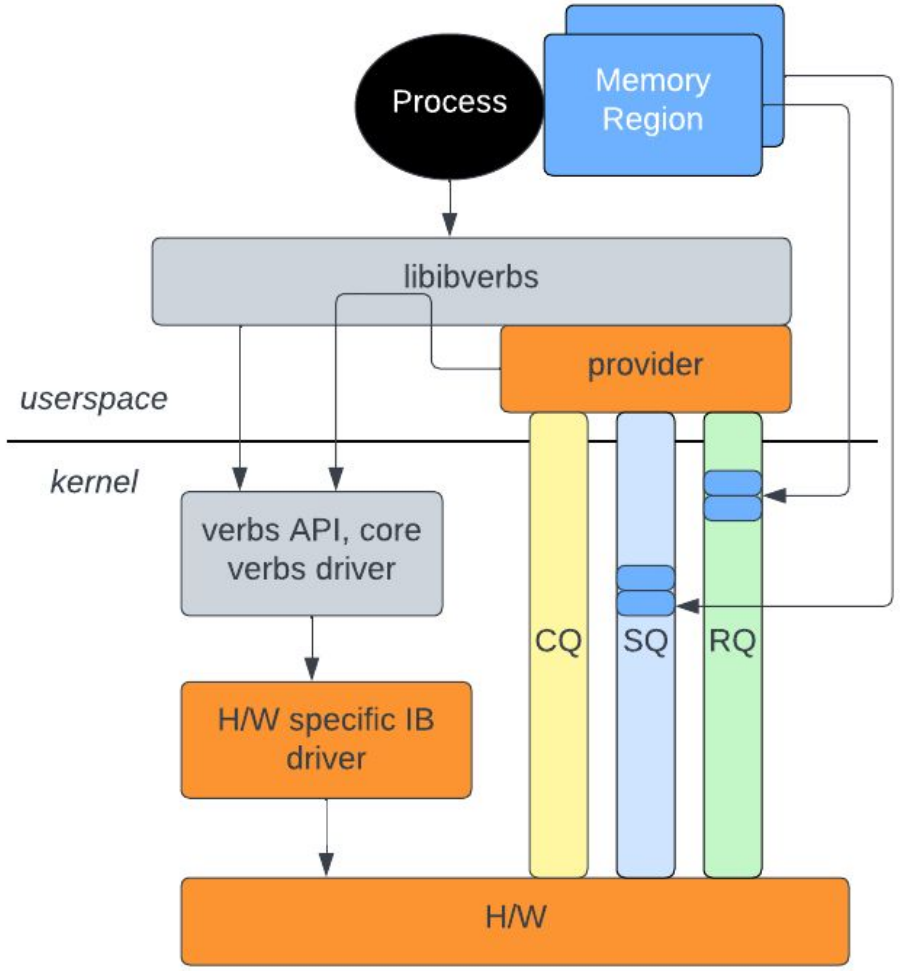
- used to associate QPs with MRs

RDMA, IB Verbs - S/W Arch

enfabrica

Userspace - H/W queues

- RQ = Receive queue. Application submits work queue entries for Rx (e.g., amount of data expected to receive in a message)
- SQ = Send queue. Application submits work queue entries for Tx (e.g., buffer/message to send)
- CQ = Completion queue. Entries noting completion of work request entries in SQ and RQ. Can have separate CQ for SQ and RQ



Merging Networking Features

enfabrica

Pull in Memory Region concept

- Page pinning, DMA mapping with hardware in control path
- Reduces overhead managing buffers for connection

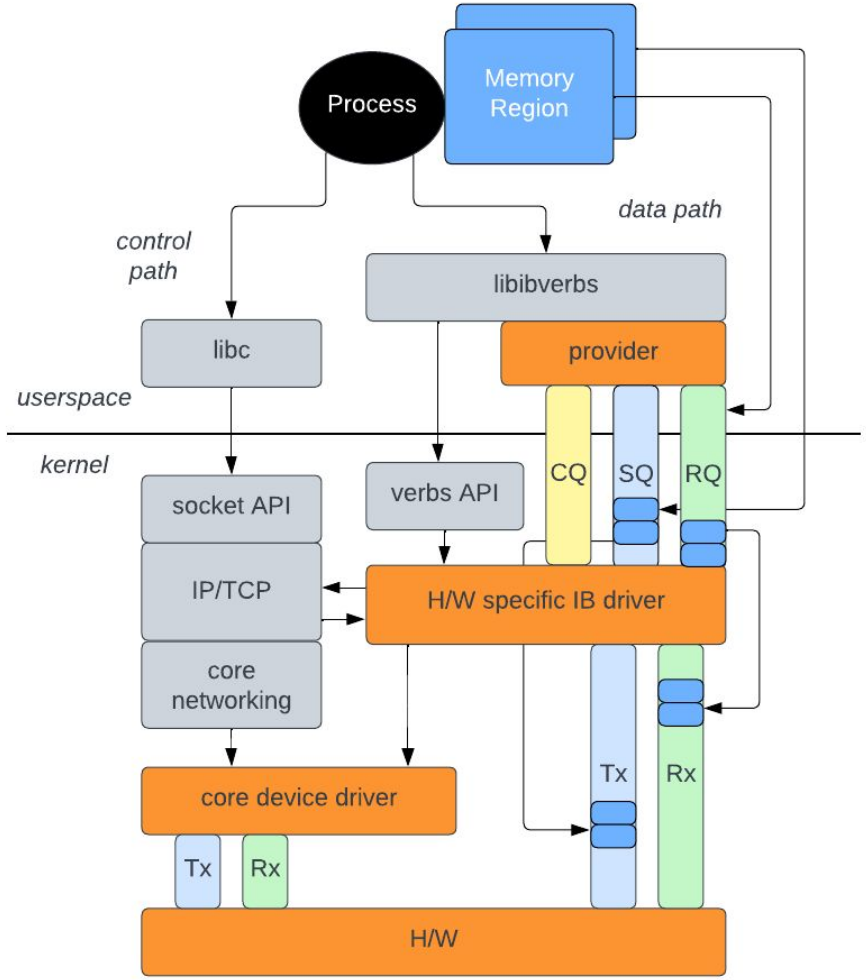
User-kernel S/W queues

- Reduce system calls submitting work requests and getting completions (CQ can be polled)

Dedicated H/W queues for flow

- Rx with application based buffers - avoids memcpy
- RSS to direct packets for flow to Rx queue

Optionally User-H/W queues



Merging Networking Worlds

enfabrica

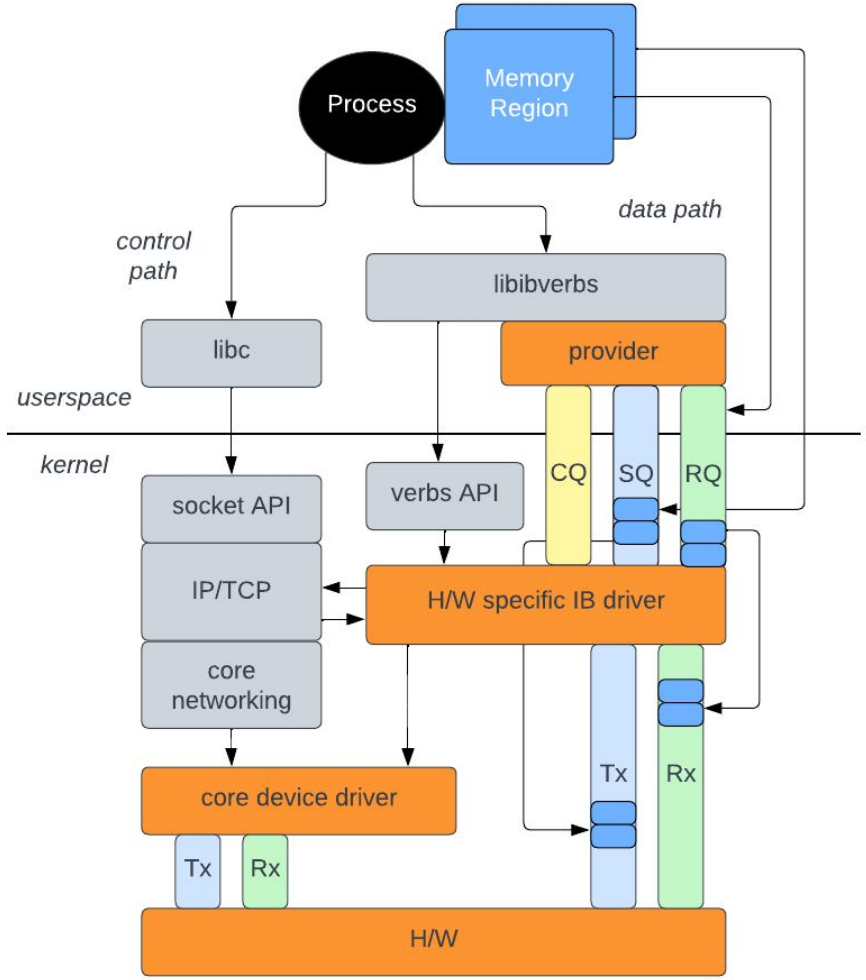
Kernel module takes ownership of socket for data path

- Manages process specific H/W queues for QP
- Manages Rx queue and sending data through TCP
- Manages SQ work requests and sending data through TCP
- Sends CQEs as Rx and Tx work requests are completed

Data path bypasses all of the infra hooks (overhead)

Userspace provider and kernel module from hardware vendor allow for better hardware integration

Existing subsystems and code, just a different wiring with vendor glue



Application Example

enfabrica

Find and open IB device, get supported attributes

- ibv_get_device_list
- ibv_get_device_name
- ibv_open_device
- ibv_query_device
- ibv_query_port

Create a PD for application

- ibv_alloc_pd

Allocate buffers for send and receive

- mmap or malloc

Register buffers with hardware as MRs

- ibv_reg_mr()

Create CQs

- ibv_create_cq()

Create QP

- ibv_create_qp
- creates hardware queues to handle both Rx and Tx
- creates and maps software queues - SQ to submit send requests, RQ to submit receive buffers, WQ to submit buffers)

Establish socket connection via typical socket APIs

Application Example

enfabrica

Transition QP through states: INIT -> RTR - > RTS

- ibv_modify_qp
- post receive buffers using ibv_post_recv if needed
- socket is handed off to kernel module

Sender posts send requests

- ibv_post_send

Both ends Poll for completion

- ibv_poll_cq
- sender: ack from peer buffer received
- receiver: posted buffer consumed

Demo

enfabrica

Summary

enfabrica

Keep the socket APIs and Linux IP/TCP stack

Get the OS out of the way in the data path

- Avoids system calls and memory copy
- Avoids infrastructure hooks up and down stack
- Simplify memory management
  - buffers handled by application (it knows best what is available for new packets)

Still have skb management

- Linux stack is skb based
- Local cache of skbs helps here

Hardware features (e.g., LRO) can be managed by H/W specific IB module

Thank You