Network Drivers

What to Expect?

- Understanding Network Subsystem
- Network Drivers: A different category
- Writing Network Drivers

Network Subsystem

- (Network) Protocol Stack
  - Typically, the TCP/IP Stack
  - Interfaces with the User Space through network interfaces, instead of device files
  - Unlike other drivers, does not provide any /sys or /dev entries
  - Interface Examples: eth0, wlan1, ...
  - Resides in the <kernel_source>/net folder
- Network Interface Card (NIC) Drivers
  - Driver for the Physical Network Cards
  - Provides uniform hardware independent interface for the network protocol layer to access the card
  - Typically, resides in the <kernel_source>/drivers/net folder

Related Data Structures

- Writing a NIC or Network Driver involves
  - Interacting with the underlying Card over its I/O Bus
  - Providing the standard APIs to the Protocol Stack
- This needs three kind of Data Structures
  - Core of Protocol Stack
    - struct sk_buff
  - NIC & Protocol Stack Interface
    - struct net_device
  - NIC I/O bus, e.g. PCI, USB, ...
    - I/O bus specific data structure

struct sk_buff

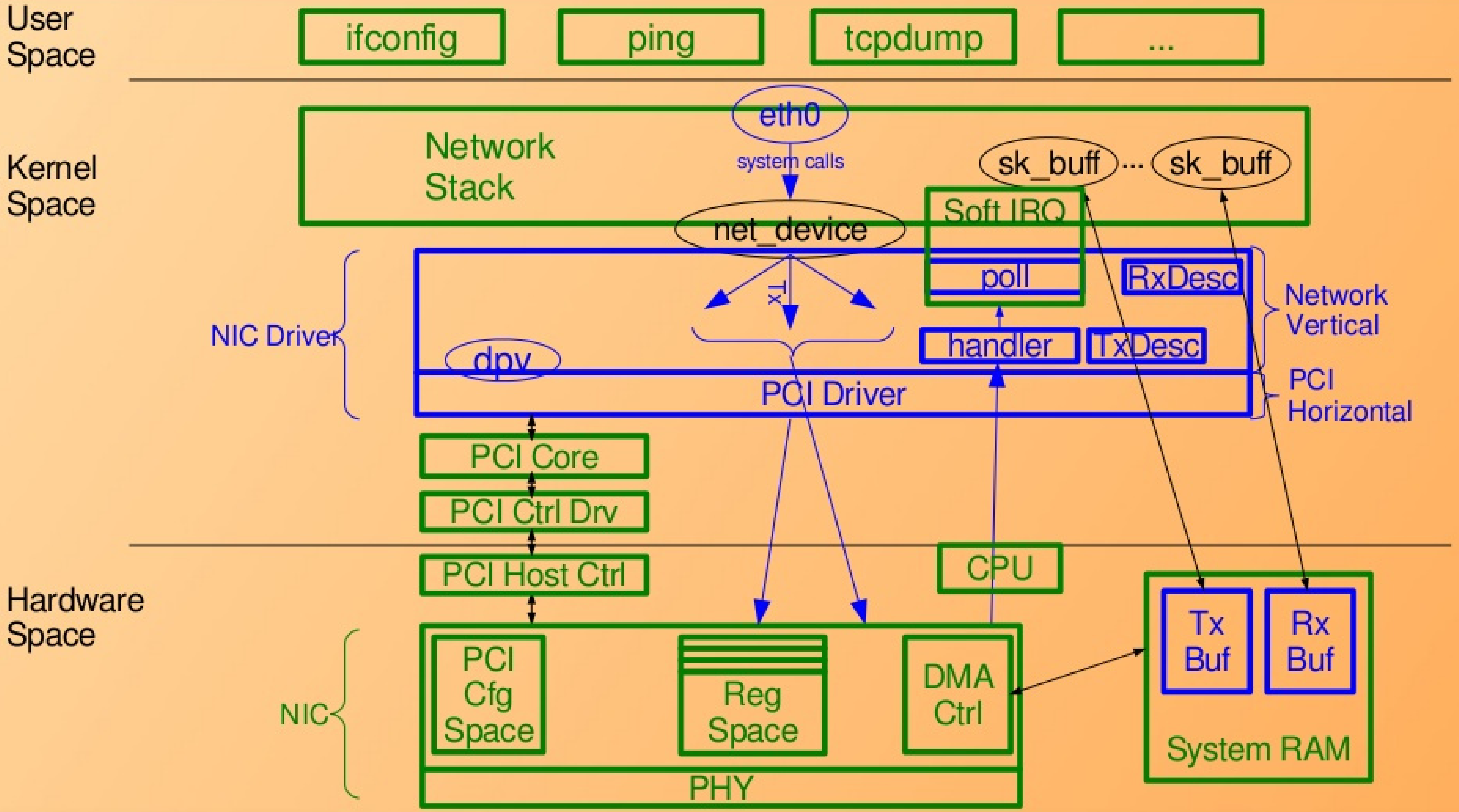
- Network Packet Descriptor
- Header: <linux/skbuff.h>
- Driver relevant Fields
  - head – points to the start of the packet
  - tail – points to the end of the packet
  - data – points to the start of the pkt payload
  - end – points to the end of the packet payload
  - len – amount of data that the packet contains

Socket Buffer APIs

- Header: <linux/skbuff.h>
- SK Buffer Storage
  - struct sk_buff *dev_alloc_skb(len);
  - dev_kfree_skb(skb);
- SK Buffer Operations
  - void skb_reserve(struct sk_buff *, int len);
  - struct sk_buff *skb_clone(struct sk_buff *, gfp_t);
  - unsigned char *skb_put(struct sk_buff *, int len);

The Big Picture

(with a PCI NIC)



struct net_device

- Header: <linux/netdevice.h>
- Driver relevant Operation Fields
  - Activation: open, stop, ioctl
  - Data Transfer: start_xmit, poll (NAPI: new API)
  - WatchDog: tx_timeout, int watchdog_timeo
  - Statistics: get_stats, get_wireless_stats
    - Typically uses struct net_device_stats, populated earlier
  - Configuration: struct *ethtool_ops, change_mtu
    - Structure defined in Header: <linux/ethtool.h>
  - Bus Specific: mem_start, mem_end
- Latest kernels have all the operations moved under
  - struct net_dev_ops
  - And typically, a prefix ndo_ added and some name changes

Net Device Registration

- Header: <linux/netdevice.h>
- Net Device Storage
  - struct net_device *alloc_etherdev(sizeof_priv);
  - struct net_device *alloc_ieee80211dev(sizeof_priv);
  - struct net_device *alloc_irdadev(sizeof_priv);
  - struct net_device *alloc_netdev(sizeof_priv, name, setup_fn);
  - void free_netdev(struct net_device *);
- Registering the Net Device
  - int register_netdev(struct net_device *);
  - void unregister_netdev(struct net_device *);

Network Device Open & Close

- A typical Network Device Open
  - Allocates ring buffers and associated sk_buffs
  - Initializes the Device
  - Gets the MAC, ... from device EEPROM
  - Requests firmware download, if needed
    - int request_firmware(fw, name, device);
  - Register the interrupt handler(s)
- A typical Network Device Close
  - Would do the reverse in chronology reverse order

Packet Receive Interrupt Handler

- A typical receive interrupt handler
  - Minimally handles the packet received
    - Sanity checks
  - Puts back equal number of sk_buffs for re-use
  - Passes the associated sk_buffs (& ring buffers) to the protocol layer by the NET_RX_SOFTIRQ
  - On higher load, switch to poll mode, if supported, which then passes the associated sk_buffs (& ring buffers) to the protocol layer by the NET_RX_SOFTIRQ

Flow Control related APIs

- For interaction with the protocol layer
- Header: <linux/netdevice.h>
- APIs
  - void netif_start_queue(struct net_device *);
  - void netif_stop_queue(struct net_device *);
  - void netif_wake_queue(struct net_device *);
  - int netif_queue_stopped(struct net_device *);

Network Driver Examples

- Driver: snull
- Browse & Discuss

- Driver for Realtek NIC 8136
- Browse & Hack

What all have we learnt?

- Linux Network Subsystem
- How are Network Drivers different?
- Writing Network Drivers
  - Key Data Structures & their APIs
    - sk_buff & net_device
  - Network Device Registration
  - Network Device Operations
  - Interrupt Handling for Packets received
  - Flow Control related APIs

Any Queries?