CS 378 (Spring 2003)

Linux Kernel Programming

Yongguang Zhang

(ygz@cs.utexas.edu)

Copyright 2003, Yongguang Zhang

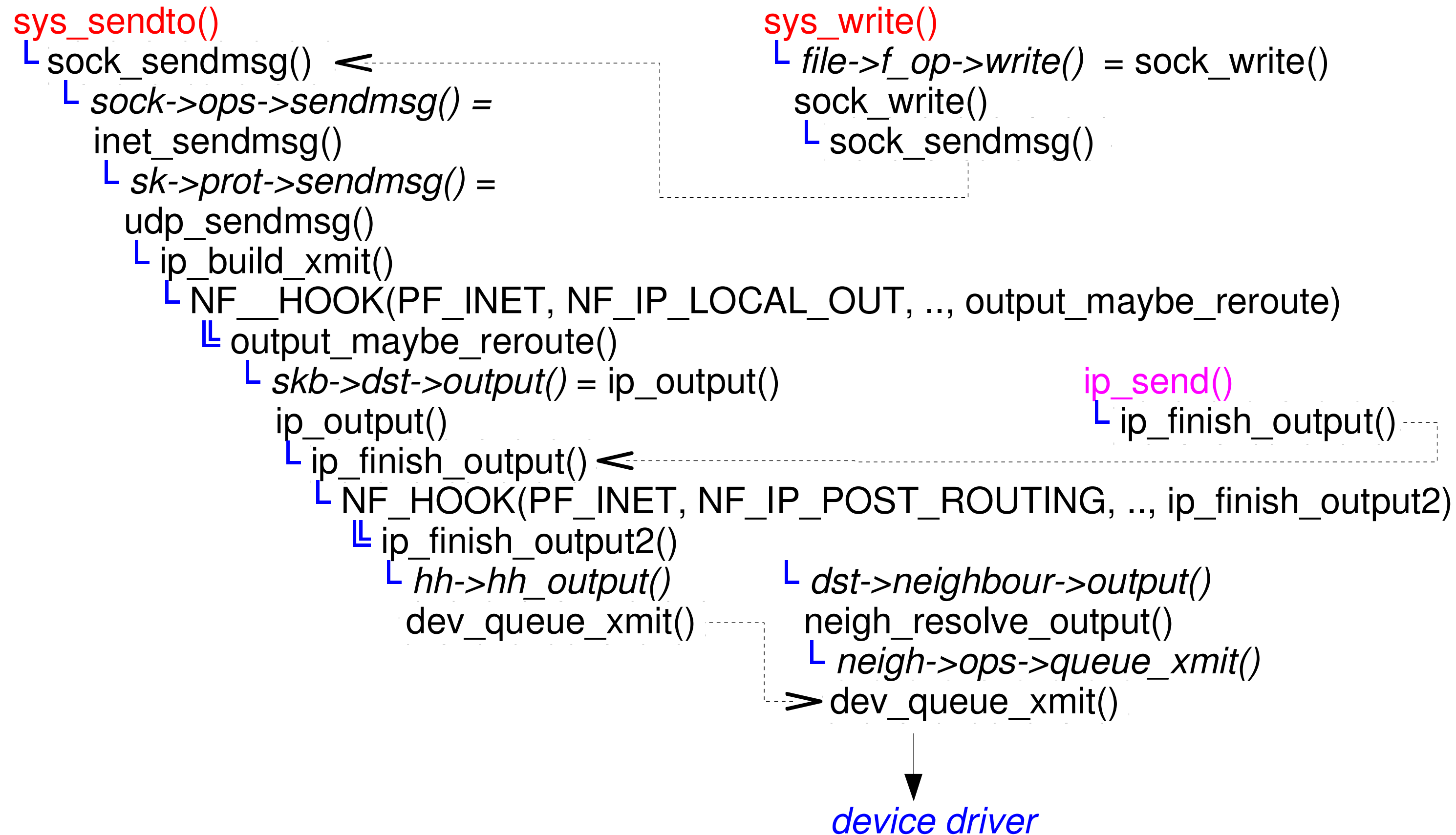
This Lecture

- Network Device Driver
- Brief notes about final projects

- Questions?

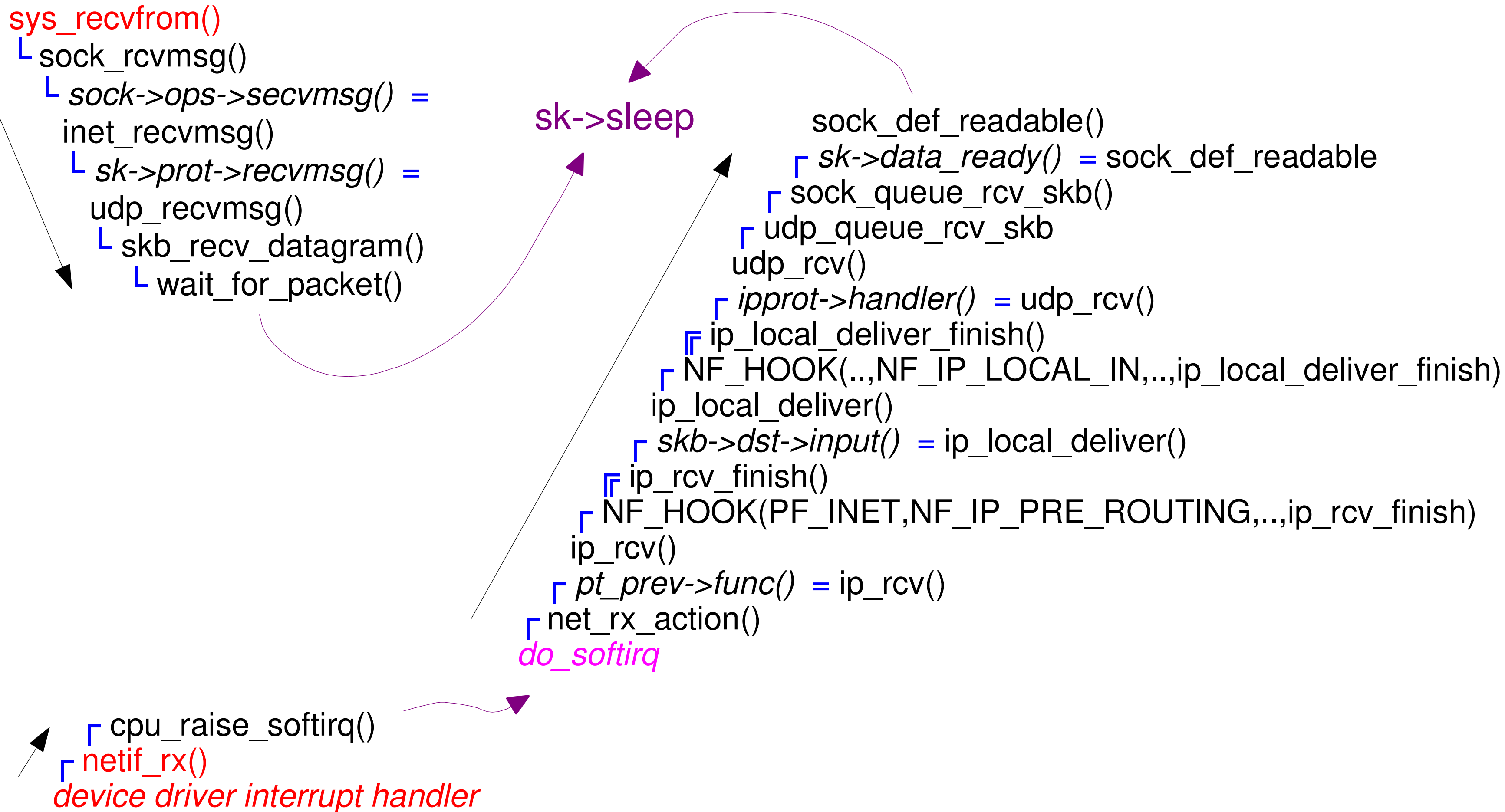
© 2003 Yongguang Zhang

Output Chain Function Calls



© 2003 Yongguang Zhang

Input Chain Function Calls



© 2003 Yongguang Zhang

Network Device Driver

- One of the three types of device driver in Linux
  - Interface: a set of service routines (no file interface)
- Network device driver
  - One network device object per loaded device, data type: struct net_device (see include/linux/netdevice.h)
  - An interrupt handler (except in some virtual devices)
  - A device/hardware independent queueing discipline
- Network device driver layer
  - Routines to access and manage the device driver

© 2003 Yongguang Zhang

Fields of struct net_device

- Visible fields
  - char name[IFNAMSIZ]: device name (e.g. "eth0")
  - Device configuration: base_addr, irq, if_port, dma, ..
  - State of the device: state
  - Init function: int (*init)(struct net_device *dev)
- Hidden fields:
  - Kitchen sink of everything related to this device
  - Device info: mtu, dev_addr[], mc_list, ...
  - Queueing disciplines: qdisc, qdisc_ingress, ...
  - Other device functions: open(), hard_start_xmit(), ...
  - Device-specific data (private pointer): priv

© 2003 Yongguang Zhang

Major Network Device Functions

- init(dev)
  - Called when network device is registered (at load time)
- open(dev)
  - Called when opened by network stack ("ifconfig ... up")
- hard_start_xmit(skb,dev)
  - Called to start transmitting a packet (in skb)
  - Caller must make sure device is not busy (check XOFF bit in dev->state), must hold lock (dev->xmit_lock)

© 2003 Yongguang Zhang

More Network Device Functions

- hard_header(skb,dev,type,daddr,saddr,len)
  - Called to fill in link-layer header in skb
- tx_timeout(dev)
  - Called when packet transmission times out
  - Called by interrupt handler (if NIC supports timeout), or by software timer (dev->watchdog_timer)
- do_ioctl(dev,ifr,cmd)
  - Called to do interface-specific ioctl command

© 2003 Yongguang Zhang

Queueing Discipline (qdisc)

- A packet queueing and scheduling subsystem
  - Enqueue packets generated from output chain
  - Determines which packet to send next and when
  - Attached to a network device, but device-independent
  - Most network devices have a qdisc (dev->qdisc)
- Data structure: struct Qdisc
  - Defined in include/net/pkt_sched.h
  - Two major routines: enqueue() and dequeue()
    - int (* enqueue)(struct sk_buff *, struct Qdisc *)
    - struct sk_buff (* dequeue)(struct Qdisc *)

© 2003 Yongguang Zhang

Kicking Device Queue

- Function qdisc_run(dev)
  - Must with dev->queue_lock locked and BH disabled
  - If the device is ready (i.e., not busy sending with XOFF bit set), call qdisc_restart(dev)
- qdisc_restart(dev):
  - Dequeue a packet from the queue discipline
  - If the device is not busy, call dev->hard_start_xmit()
  - Otherwise, requeue the packet and try next time (with netif_schedule())

© 2003 Yongguang Zhang

Device Transmission Control

- netif_stop_queue(dev): transmit off
  - Device is busy sending, so don't call hard_start_xmit()
  - Usually called by device driver before start sending
  - Implementation: set an XOFF bit in dev->state
  - Can be checked by netif_queue_stopped(dev)
- netif_wake_queue(dev): wake up device queue
  - Device ready for next packet -- hard_start_xmit() may be called now if it is safe (like in softirq time)
  - Usually called by device driver after done sending
  - Implementation: if device is XOFF, clean it and call __netif_schedule(dev) (to schedule next transmission)

© 2003 Yongguang Zhang

Schedule Next Transmission

- When to send next packet? At Softirq time
  - Why softirq? Device driver usually in interrupt when it knows it has done sending last packet
- __netif_schedule(dev)
  - Add device to per-CPU output queue

```
dev->next_sched = softnet_data[CPU].output_queue;
softnet_data[CPU].output_queue = dev;
```

  - Raise softirq

```
cpu_raise_softirq(cpu, NET_TX_SOFTIRQ);
```

- netif_schedule(dev):
  - Call __netif_schedule(dev) if device is ready

© 2003 Yongguang Zhang

Softirq NET_TX_SOFTIRQ

- Softirq handler: net_tx_action()
  - Free skb's that have been released in interrupt handler
  - For each device in the per-CPU output queue (i.e., have something to send), kick its packet scheduler

```
if (spin_trylock(&dev->queue_lock)) {
        qdisc_run(dev);
        spin_unlock(&dev->queue_lock);
```

  - And try next time if you cannot grab the lock

```
} else {
        netif_schedule(dev);
}
```


© 2003 Yongguang Zhang

Queue and Transmit from Net Stack

- Called in output chain: dev_queue_xmit(skb)
  - If device has a queue discipline, enqueue the packet and kick start the packet scheduler for sending next packet:

```
spin_lock_bh(&dev->queue_lock);
if (q->enqueue()) {
        int ret = q->enqueue(skb, q);
        qdisc_run(dev);
        spin_unlock_bh(&dev->queue_lock);
```

  - Otherwise (no queue discipline), if device is ready (no XOFF) send with dev->hard_start_xmit()
  - All other cases: drop packet with kfree_skb(skb)

© 2003 Yongguang Zhang

Transmission Timeline

dev_queue_xmit(skb)

qdisc_run(dev)
└ qdisc_restart(dev)

netif_schedule(dev)
└ __netif_schedule(dev)
└ cpu_raise_softirq(...)

dev->hard_start_xmit(skb,dev)

net_stop_queue(dev)

Network device dirver
(start transmission)

┌ cpu_raise_softirq(...)
┌ __netif_schedule(dev)
net_wake_queue(dev)

Network device interrupt
(transmission complete)

do_softirq()
└ net_tx_action()

netif_schedule(dev)
└ __netif_schedule(dev)
└ cpu_raise_softirq(...)

qdisc_run(dev)
└ qdisc_restart(dev)

dev->hard_start_xmit(skb,dev)

Network device dirver
(start transmission)

© 2003 Yongguang Zhang

Skeleton hard_start_xmit()

- Typical code
  - Call netif_stop_queue(dev)
  - Call I/O instructions to copy the packet to hardware and start transmission.
  - If skb won't be accessed any more free it now (otherwise you can free it after transmission done)
    - dev_kfree_skb(skb);
  - Record transmission time (for device watchdog timer)
    - dev->trans_start = jiffies;

© 2003 Yongguang Zhang

Skeleton Interrupt Handler

- Check interrupt type
- For incoming packet
  - Allocate skb, copy packet from hardware
  - Call netif_rx(skb)
- For transmission complete
  - Update statistics
  - If need to free skb here, call dev_kfree_skb_irq(skb)
  - Call netif_wake_queue(dev)
- For time-out: similar to above

© 2003 Yongguang Zhang

Other Skeleton Functions

- dev->open(dev)
  - Call netif_start_queue(dev)
- dev->tx_timeout(dev)
  - Update statistics, free skb if needed
  - Call netif_wake_queue(dev)
- dev->init(dev)
  - Probe the hardware (may have more than one devices)
  - Prepare the hardware I/O and interrupts
  - Initialize fields in dev, and assign device functions
  - For Ethernet device, call ether_setup(dev) – common setup procedure for all Ethernet NIC

Managing All Network Devices

- Kernel maintains list of device: dev_base
  - Builtin network devices declared and linked to dev_base in drivers/net/Space.c
  - Dynamic loaded device driver: register the network device object by calling register_netdev(dev)
- Major steps in register_netdev()
  - Call net_dev_init() if the device layer is uninitialized
  - Call dev->init()
  - Add dev to the list of device (dev_base)
  - Initialize queueing disciplines (dev_init_scheduler())
  - Notify network callback chain about a new device

© 2003 Yongguang Zhang

Initialize the Device Layer

- Function net_dev_init(void)
  - Called by the first register_netdev() call
  - Initialize all builtin devices in dev_base
    - Call dev->init(dev)
  - Remove unresponded devices from dev_base
  - Set up two network-related softirq:
    - open_softirq(NET_TX_SOFTIRQ, net_tx_action, NULL);
    - open_softirq(NET_RX_SOFTIRQ, net_rx_action, NULL);
  - Initialize destination cache, multicast, ...

© 2003 Yongguang Zhang

More about Queueing Disciplines

- Every network device (non-virtual) should have a queueing discipline (dev->qdisc)
  - Default: pfifo_fast (FIFO) with max queue len = 100
  - Non-asynchronous device (e.g., virtual interface) need not have a queueing discipline
  - You may add queueing discipline for receiving packets too (dev->qdisc_ingress)
- Many queueing disciplines implemented in Linux
  - Like TBF, SFQ, CBQ, HTB, ..., see net/sched/
  - You can choose to use any qdisc at any device
  - Allow complicated traffic control with "tc" command

© 2003 Yongguang Zhang

Summary

- Network Device Driver
  - LDD2: §14
  - LKP: §8.3
  - ULK: §18
- More about queueing disciplines, advanced routing, traffic control:
  - URL: http://ds9a.nl/2.4Networking/

© 2003 Yongguang Zhang

Brief Notes about Final Projects

- Candidate projects will be posted in next few days
  - Include the scope and the number of students per team
  - By this Friday, you should have chosen a project and formed a team
- You can propose a project yourselves
  - Must be a kernel project (not majority in usermode)
  - Must be similar in size (not too hard, not too easy)
  - Write a description and send it to me and the TA

© 2003 Yongguang Zhang

My Pet Projects

- Mobile ad-hoc network implementations
  - In kernel
- IPsec improvements
  - Modify the IPsec implemetnation in Linux 2.5
- Talk to me soon if you are interested

© 2003 Yongguang Zhang