# Operating Systems 2

## Networking

Arkadiusz Chrobot

Department of Computer Science

June 8, 2020

# Outline

# Introduction

Unix is one of the first operating systems that offered an implementation of network communication. Nowadays, most of the Internet servers run on Linux, the Unix-like operating system. In this lecture a short overview of the Linux kernel network subsystem is given. This subject is complex, so only the most important concepts are presented. The content is split into three parts:

- kernel-level packet processing,
- network device drivers,
- the netfilter implementation.

# The TCP/IP Stack

The part of kernel that is responsible for handling the incoming and outgoing network packets is called the TCP/IP stack. Figure 1 presents the flow of packet processing inside the Linux kernel[1]. The Linux kernel network subsystem consists of three parts that corresponds to three layers of ISO/OSI model — the data link layer, the network layer and the transport layer. To send data through a network a user process invokes an appropriate system call that activates the `write()` method of a file object associated with the process network socket. Depending on the transport protocol used by this socket the `write()` method calls either the `tcp_sendmsg()` or `udp_sendmsg()` kernel function. These functions are responsible for building a header of the required protocol.

[1] This part of the lecture is based on: https://www.linuxfoundation.org/collaborate/workgroups/networking/network_overview and William Stallings "Operating Systems: Internals and Design Principles", Pearson Education, Inc, London, 2005

# The TCP/IP Stack

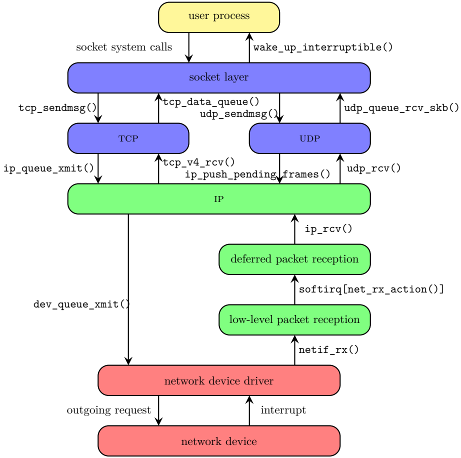


Figure 1 : Incoming and outgoing packets processing inside the kernel

# The TCP/IP Stack

The transport protocol header is attached to the data from the user process. Then a function responsible for creating and adding to the packet an IP header is invoked. In case of a UDP packet it is the `ip_push_pending_frames()` function. For the TCP packet the `ip_queue_xmit()` is called. The packet with all required headers is passed to a network device with the help of the `dev_queue_xmit()` function. However, before the packet will be send, its route to the destination host (a computer or other device connected to network) must be established. This is the responsibility of `ip_route_output()` function that checks caches or (if necessary) routing tables to determine the packet destination. Should the package be sent to other hosts in the network, then it is further processed by the `ip_output()` function.

When a network device receives an incoming packet it usually triggers an interrupt. Optionally two other interrupts may be raised when a transmission of a packet is finished or when a transmission exception occurs.

# The TCP/IP Stack

There are cases when the network device doesn't trigger the interrupt after reception of a packet. This is explained in the next section where the NAPI is described. The network device driver allocates memory for a packet buffer and sets its pointer to point to the packet IP header. The packet inside the buffer is passed to the `netif_rx()` function which adds it to a queue. All packets from the queue are processed by the `ip_rcv()` function that calls the `ip_local_deliver()` function. The latter invokes the `tcp_v4_rcv()` functions for TCP packets or the `udp_rcv()` function for UDP packets. Next, the functions informing the user process that a packet has arrived are called. In case of TCP packet it is the `tcp_data_queue()` function and for the UDP protocol it is the `udp_queue_rcv()` function.

# The TCP/IP Stack

The main data structure used by the kernel network subsystem is the packet buffer called `sk_buff`. The data type of this buffer is `struct sk_buff`. The structure stores not only the received or sent data but also metadata required for processing the packet. The metadata are located in the packet header. The packet buffer is designed to be efficiently transfered from one queue to another. If it is copied then only its header is duplicated. The header has three fields (members) that point to the private headers storing metadata associated with the three layers of ISO/OSI model. The `transport_header` points to the transport layer header. The network layer header is pointed by the `network_header`. Finally the `mac_layer` points to the data link layer header. All packet buffers are stored in a queue implemented as a doubly linked list.

# Network Device Drivers

The main data structure used by a network device is a structure of the `struct net_device` type [2]. The most important fields of this structure are: the `mtu` — it specifies the maximum size of a packet that can be transmitted by the device, the `flags` — it specifies the state of the device, the `dev_addr` — stores the MAC address, the `promiscuity` — it is a counter that stores the number of request to set the device in promiscuous mode and the `ip_ptr` — it points to a variable that stores IPv4 specific data.
Earlier implementations of network device drivers required the device to acknowledge every packet reception by triggering an interrupt. In effect a heavy network traffic could cause a kernel overload. In the 2.6 kernel series a new API for network device drivers was created and named NAPI.

[2]This part of the lecture is based on http://www.linuxfoundation.org/collaborate/workgroups/networking/network_overview and http://www.linuxfoundation.org/collaborate/workgroups/networking/napi

# Network Device Drivers

The New API enables the driver to switch the device into a polling mode and allow it to accumulate a number of incoming packets that are later processed by the kernel. This reduces the number of interrupts triggered by the device and as a consequence lower the kernel load. The driver can also drop some of the packets before they are passed for processing to other parts of the kernel. It is called *packet throttling*. The NAPI requires a buffer in the RAM for DMA transmissions or a hardware support in a form of a DMA *ring*.

# The Netfilter

The netfilter (the name is an abbreviation of the expression "Network Filter") is a set of function pointers, called *hooks*, that are located in strategic places inside the TCP/IP stack. These pointers can be used for creating firewalls or NAT (Network Address Translation) subsystems. Functions pointed by hooks are usually implemented inside a kernel module[3]. There are five hooks in the kernel network system:

`NF_IP_PRE_ROUTING` functions associated with this hook are called when a packet is received,

`NF_IP_LOCAL_IN` functions associated with this hook perform processing of packets delivered to the host,

`NF_IP_FORWARD` functions associated with this hook perform processing of packets that should be forwarded to other hosts,

[3] http://www.paulkiddie.com/2009/11/creating-a-netfilter-kernel-module-which-filters-udp-packets

# The Netfilter

**NF_IP_POST_ROUTING** functions associated with this hook perform processing of packages with established route that are intended to be sent,

**NF_IP_LOCAL_OUT** functions associated with this hook perform processing of packets that were sent locally.

Each function associated with any of the hooks can perform any operation on a packet that is necessary, but it has to eventually return one of the following values:

**NF_ACCEPT** the packet is accepted for further processing,

**NF_DROP** the packet is rejected,

**NF_REPEAT** the function call should be repeated for this package,

**NF_STOLEN** the function "steals" the packet, which means that this packet will be processed in a different way than the other packages,

# The Netfilter

- `NF_QUEUE` the packet is added to a queue from where it will be transfered to the user-space,
- `NF_STOP` processing of the packet is stopped.

A single function associated with a hook is represented by a structure of the `nf_hook_ops` type. The definition of the type is given in the Listing 1. The `list` field allows these structures to be stored in a linked list. The `hook` field is a pointer to the packet processing function. The `dev` field is a pointer to a structure that represents the network device. The `priv` member is a pointer to an area of the memory that stores private data of the packet processing function. The `pf` field stores the identifier of protocol family. Packets of that protocols will be processed by the function. The `hooknum` field stores the hook number and the `priority` field stores the hook priority that specifies the order in which the hooks are performed (the `NF_IP_PRI_FIRST` constant defines th highest priority).

# The Netfilter

```
struct nf_hook_ops
{
        struct list_head        list;
        nf_hookfn               *hook;
        struct net_device       *dev;
        void                    *priv;
        u_int8_t                pf;
        unsigned int            hooknum;
        int                     priority;
};
```

Listing 1: The definition of the `struct nf_hook_ops`

# The Netfilter

Structures of the `struct nf_hook_ops` type are registered with the use of the `nf_register_hook()` function and unregistered with the help of the `nf_unregister_hook()` function. Each packet processing function has to return a number of the `unsigned int` type and take three arguments: an address of its private data (it is passed by the `void *` type parameter), the address of a packet buffer (the buffer is of the `struct sk_buff` type) and finally an address of a structure that stores the state of the hook. This structure is of the `struct nf_hook_state` type.

# Questions

?

# The End

Thank You for Your attention!