Programowanie sieciowe
Obsługa interfejsów sieciowych

Michał Kalewski

Instytut Informatyki
Politechnika Poznańska
ul. Piotrowo 2, 60–965 Poznań

mkalewski@cs.put.poznan.pl
http://www.cs.put.poznan.pl/mkalewski

Poznań · 5 marca 2019 r.

Plan wykładu

Warstwa łącza danych

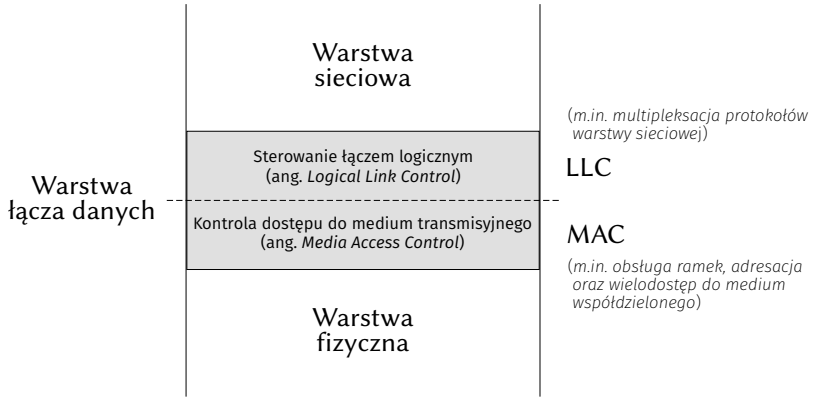
Zastosowania dostępu do warstwy łącza danych

Korzyści wynikające z dostępu programistycznego do warstwy łącza danych:

- możliwość implementacji analizatorów komunikacji sieciowej — w szczególności w tzw. *trybie nasłuchiwania* (ang. *promiscuous mode*);
- możliwość implementacji protokołów warstwy sieciowej poza jądrem systemu operacyjnego;
- możliwość realizacji komunikacji sieciowej bez potrzeby używania lub konfigurowania stosu protokołów internetowych.

Rysunek 1: Model warstwowy ISO/OSI.

Podwarstwy warstwy łącza danych



Rysunek 2: Podwarstwy LLC i MAC warstwy łącza danych.

Obsługa interfejsów sieciowych komendami systemowymi

Komendy systemowe obsługi interfejsów sieciowych (1/4)

Podstawowymi komendami obsługi interfejsów sieciowych w systemach operacyjnych UNIX oraz GNU/Linux są komendy `ifconfig(8)` oraz `ip(8)`.

Komenda `ifconfig(8)` pochodzi z pakietu o nazwie *net-tools*[1], a komenda `ip(8)` z pakietu nowszego o nazwie *iproute2*[2].

Do obsługi parametrów specyficznych dla sieci IEEE 802.3/Ethernet można posłużyć się komendą `ethtool(8)`.

Do obsługi parametrów specyficznych dla sieci bezprzewodowych IEEE 802.11/Wi-Fi można posłużyć się komendą `iwconfig(8)` lub `iw(8)`.

> Komendy powyższe w systemach GNU/Linux mogą się różnić od komend o tych samych nazwach w systemach operacyjnych UNIX.

Porównanie komend z pakietów net-tools oraz iproute2 dostępne jest m.in. pod następującym adresem: https://dougvitale.wordpress.com/2011/12/21/deprecated-linux-networking-commands-and-their-replacements/.

[1] https://wiki.linuxfoundation.org/networking/net-tools/
[2] https://wiki.linuxfoundation.org/networking/iproute2/

Komendy systemowe obsługi interfejsów sieciowych (2/4)

Przykład 1 — Ubuntu 14.04.5 LTS:

```
$ ifconfig eth0
eth0 Link encap:Ethernet  HWaddr 00:07:e9:5d:b5:b3
     inet addr:192.168.0.106  Bcast:192.168.0.255
Mask:255.255.255.0
     inet6 addr: fe80::207:e9ff:fe5c:b7a3/64 Scope:Link
     UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
     RX packets:10753289 errors:0 dropped:34 overruns:0 frame:0
     TX packets:1375800 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:1152728857 (1.1 GB)  TX bytes:289068293 (289.0 MB)
```

Komendy systemowe obsługi interfejsów sieciowych (3/4)

Przykład 2 — openSUSE Leap 15.0:

```
$ ifconfig em1
em1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.0.106  netmask 255.255.255.0  broadcast
192.168.0.255
    ether 5c:f9:dc:6a:8a:b8  txqueuelen 1000  (Ethernet)
    RX packets 9637  bytes 16305754 (15.5 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 6942  bytes 826529 (807.1 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```


Komendy systemowe obsługi interfejsów sieciowych (4/4)

Przykład 3 — os x:

```
$ ifconfig en0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST>
mtu 1500
    ether 78:31:c1:d3:08:58
    inet6 fe80::7a31:c1ff:fed3:858%en0 prefixlen 64 duplicated
scopeid 0x4
    inet 192.168.0.106 netmask 0xffffff00 broadcast 192.168.0.255
    nd6 options=9<PERFORMNUD,IFDISABLED>
    media: autoselect
    status: active
```


Detekcja systemu operacyjnego w powłoce bash

```
1 case "$OSTYPE" in
2   solaris*) echo "SOLARIS" ;;
3   darwin*)  echo "OSX" ;;
4   linux*)   echo "LINUX" ;;
5   bsd*)     echo "BSD" ;;
6   *)        echo "unknown: $OSTYPE" ;;
7 esac
```


Lista dostępnych interfejsów sieciowych (1/3)

Informację o wszystkich dostępnych w systemie interfejsach sieciowych można uzyskać komendą systemową `ifconfig` z przełącznikiem `-a`.

Przykład:

```
$ ifconfig | grep -v "^$" | grep -v "^ "
em1       Link encap:Ethernet  HWaddr 5C:F9:DD:78:58:EA
lo        Link encap:Local Loopback

$ ifconfig -a | grep -v "^$" | grep -v "^ "
em1       Link encap:Ethernet  HWaddr 5C:F9:DD:78:58:EA
lo        Link encap:Local Loopback
p4p1      Link encap:Ethernet  HWaddr 90:E2:BA:1A:52:EC
wlan0     Link encap:Ethernet  HWaddr 08:ED:B9:4C:31:97

$ ethtool -p em1 15
```

Lista dostępnych interfejsów sieciowych (2/3)

Alternatywnie, możliwe jest zastosowanie nowszej komendy systemowej `ip` dla obiektu `link` z poleceniem `show` (lub `list`).

Przykład:

```
$ ip link show | cut -d " " -f 1-5
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
    link/loopback
2: p4p1: <BROADCAST,MULTICAST> mtu 1500
    link/ether
3: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether
4: wlan0: <BROADCAST,MULTICAST> mtu 1500
    link/ether
```


Lista dostępnych interfejsów sieciowych (3/3)

Ostatecznie, lista wszystkich dostępnych interfejsów sieciowych przechowywana jest w pliku `/proc/net/dev`. *zob.* `proc(5)`

Przykład:

```
$ cat /proc/net/dev
Inter-|  Receive
 face |bytes    packets errs drop fifo frame compressed multicast
 wlan0:       0       0    0    0    0     0          0         0
    lo:       0       0    0    0    0     0          0         0
   em1: 9772345   10416    0    0    0     0          0      1238
  p4p1:       0       0    0    0    0     0          0         0
Inter-|  Transmit
 face |bytes    packets errs drop fifo colls carrier compressed
 wlan0:       0       0    0    0    0     0       0          0
    lo:       0       0    0    0    0     0       0          0
   em1:  494496    3341    0    0    0     0       0          0
  p4p1:       0       0    0    0    0     0       0          0
```


Włączanie i wyłączanie interfejsów sieciowych

Interfejsy sieciowe mogą być włączane i wyłączane, nawet jeżeli nie posiadają skonfigurowanych adresów warstwy sieciowej (tj. najczęściej adresów IPv4 lub IPv6).

Przykład:

```
$ ifconfig | grep -v "^$" | grep -v "^ "
lo        Link encap:Local Loopback
$ ifconfig em1 up && ifconfig | grep -v "^$" | grep -v "^ "
em1       Link encap:Ethernet  HWaddr 5C:F9:DD:78:58:EA
lo        Link encap:Local Loopback
$ ifconfig em1 down && ifconfig | grep -v "^$" | grep -v "^ "
lo        Link encap:Local Loopback
$ ip link set up em1 && ifconfig | grep -v "^$" | grep -v "^ "
em1       Link encap:Ethernet  HWaddr 5C:F9:DD:78:58:EA
lo        Link encap:Local Loopback
$ ip link set down em1 && ifconfig | grep -v "^$" | grep -v "^ "
lo        Link encap:Local Loopback
```


Tryb nasłuchiwania

W trybie nasłuchiwania (ang. *promiscuous mode*), interfejs sieciowy *dostarcza* do systemu wszystkie odbierane ramki niezależnie od zawartego w nich adresu fizycznego (MAC) odbiorcy.

Przykład:

```
$ ifconfig em1 promisc && ifconfig em1 | tail -n 6 | head -n 1
      UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
$ ifconfig em1 -promisc && ifconfig em1 | tail -n 6 | head -n 1
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
$ ip link set em1 promisc on && ip link show em1 | cut -f 2-3 -d "
"
em1: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP>
$ ip link set em1 promisc off && ip link show em1 | cut -f 2-3
-d " "
em1: <BROADCAST,MULTICAST,UP,LOWER_UP>
```

Obsługa interfejsów sieciowych funkcją systemową `ioctl(2)`

Funkcja systemowa `ioctl(2)` (1/2)

Funkcja systemowa `ioctl(2)` nie jest częścią standardów POSIX i w związku z tym jej wywołania mogą się różnić pomiędzy systemami UNIX oraz GNU/Linux.

Głównym zastosowaniem funkcji systemowej `ioctl(2)` jest obsługa parametrów:

- gniazd sieciowych;
- plików dyskowych;
- *interfejsów sieciowych*;
- tablic protokołu ARP;
- tablic routingu;
- systemu STREAMS.

> Przedstawione dalej przykłady kodów programów przeznaczone są dla systemów operacyjnych GNU/Linux.

Funkcja systemowa `ioctl(2)` (2/2)

GNU/Linux:

```
int ioctl(int d, int request, ... /* void* arg */ );
```

UNIX:

```
int ioctl(int fildes, unsigned long request, ...);
```

Parametry:

`d` / `fildes` otwarty deskryptor pliku.

`request` żądanie specyficzne dla obiektu, którego parametry mają podlegać manipulacji; istnieją żądania *wejścia* (ang. *in*) oraz żądania *wyjścia* (ang. *out*); makra i definicje żądań znajdują się w pliku `ioctls.h`.

`...` wskaźnik na obszar w pamięci (jego znaczenie i rozmiar zależne są od rodzaju żądania).

Wartość zwracana: `0` (lub wartość dodatnia) w przypadku poprawnego wykonania funkcji lub `-1` w przypadku błędu.

Dokumentacja żądań dla interfejsów sieciowych

Dokumentacja żądań funkcji systemowej `ioctl(2)` dla interfejsów sieciowych zawarta jest w podręczniku systemowym pod nazwą `netdevice(7)`.

PRZYKŁAD:

```
$ grep -A8 -i "socket" sockios.h
/* Socket configuration controls. */
#define SIOCGIFNAME     0x8910  /* get iface name           */
#define SIOCSIFLINK     0x8911  /* set iface channel        */
#define SIOCGIFCONF     0x8912  /* get iface list           */
#define SIOCGIFFLAGS    0x8913  /* get flags                */
#define SIOCSIFFLAGS    0x8914  /* set flags                */
#define SIOCGIFADDR     0x8915  /* get PA address           */
#define SIOCSIFADDR     0x8916  /* set PA address           */
#define SIOCGIFBRDADDR  0x8919  /* get broadcast PA address */
```

https://github.com/torvalds/linux/blob/master/include/uapi/linux/sockios.h

Struktura `ifreq` (1/2)

Struktura `ifreq` wykorzystywana jest do manipulacji parametrami interfejsu sieciowego.

```
struct ifreq {
  union {
    char ifrn_name[IFNAMSIZ];      /* Interface name. */
  } ifr_ifrn;
  union {
    struct sockaddr ifru_addr;
    struct sockaddr ifru_dstaddr;
    struct sockaddr ifru_broadaddr;
    ...
  } ifr_ifru;
};
```

zob. `if.h`

https://github.com/torvalds/linux/blob/master/include/uapi/linux/if.h

Struktura `ifreq` (2/2)

```
#define ifr_name      ifr_ifrn.ifrn_name
#define ifr_hwaddr    ifr_ifru.ifru_hwaddr
#define ifr_addr      ifr_ifru.ifru_addr
#define ifr_dstaddr   ifr_ifru.ifru_dstaddr
#define ifr_broadaddr ifr_ifru.ifru_broadaddr
#define ifr_netmask   ifr_ifru.ifru_netmask
#define ifr_flags     ifr_ifru.ifru_flags
#define ifr_metric    ifr_ifru.ifru_ivalue
#define ifr_mtu       ifr_ifru.ifru_mtu
#define ifr_map       ifr_ifru.ifru_map
#define ifr_slave     ifr_ifru.ifru_slave
#define ifr_data      ifr_ifru.ifru_data
#define ifr_ifindex   ifr_ifru.ifru_ivalue
#define ifr_bandwidth ifr_ifru.ifru_ivalue
#define ifr_qlen      ifr_ifru.ifru_ivalue
#define ifr_newname   ifr_ifru.ifru_newname
```


</> Włączanie i wyłączanie interfejsów sieciowych

```
int sfd;
struct ifreq ifr;

sfd = socket(PF_INET, SOCK_DGRAM, 0);
strncpy(ifr.ifr_name, argv[1], IFNAMSIZ);

ioctl(sfd, SIOCGIFFLAGS, &ifr);

ifr.ifr_flags |= IFF_UP;
ioctl(sfd, SIOCSIFFLAGS, &ifr);

ifr.ifr_flags &= ~IFF_UP;
ioctl(sfd, SIOCSIFFLAGS, &ifr);
```


</> Tryb nasłuchiwania

```
int sfd;
struct ifreq ifr;

sfd = socket(PF_INET, SOCK_DGRAM, 0);
strncpy(ifr.ifr_name, argv[1], IFNAMSIZ);

ioctl(sfd, SIOCGIFFLAGS, &ifr);

ifr.ifr_flags |= IFF_PROMISC;
ioctl(sfd, SIOCSIFFLAGS, &ifr);

ifr.ifr_flags &= ~IFF_PROMISC;
ioctl(sfd, SIOCSIFFLAGS, &ifr);
```

Struktura `ifconf`

Struktura `ifconf` wykorzystywana jest do pozyskiwania informacji o wszystkich interfejsach sieciowych w systemie, które posiadają skonfigurowany adres warstwy sieciowej (żądanie `SIOCGIFCONF`).

```
struct ifconf {
  int ifc_len;                      /* Size of buffer.    */
  union {
    __caddr_t ifcu_buf;
    struct ifreq *ifcu_req;
  } ifc_ifcu;
};
#define ifc_buf ifc_ifcu.ifcu_buf  /* Buffer address.     */
#define ifc_req ifc_ifcu.ifcu_req  /* Array of structures.*/
```

zob. `if.h`

https://github.com/torvalds/linux/blob/master/include/uapi/linux/if.h

</> Lista dostępnych interfejsów sieciowych

```
int sfd, len, lastlen;  char* buf;  struct ifconf ifc;
lastlen = 0;
len = 100 * sizeof(struct ifreq);
sfd = socket(PF_INET, SOCK_DGRAM, 0);
while(1) {
  buf = malloc(len);
  ifc.ifc_len = len;
  ifc.ifc_buf = buf;
  ioctl(sfd, SIOCGIFCONF, &ifc);
  if (ifc.ifc_len == lastlen)
    break;
  lastlen = ifc.ifc_len;
  len += 10 * sizeof(struct ifreq);
  free(buf);
}
```


Uwagi dodatkowe

W niektórych systemach operacyjnych (np. *Solaris*), istnieje żądanie `SIOCGIFNUM`, które pozwala na pozyskanie informacji o liczbie interfejsów sieciowych w systemie (np. przed wywołaniem funkcji systemowej `ioctl(2)` z żądaniem `SIOCGIFCONF`).

```
#ifdef SIOCGIFNUM
ioctl (sfd, SIOCGIFNUM, &ifnum);
#endif
```

Pozyskanie informacji o dostępnych w systemie interfejsach sieciowych (i ich parametrach) możliwe jest także z pominięciem bezpośrednich wywołań `ioctl(2)` przy użyciu funkcji `getifaddrs(3)`:

```
int getifaddrs(struct ifaddrs **ifap);
```

(Funkcja ta wymaga zwolnienia zaalokowanej pamięci przy użyciu funkcji komplementarnej `freeifaddrs(3)`).

Podsumowanie

Podsumowanie

- Komendy systemowe obsługi interfejsów sieciowych: `ifconfig(8)` i `ip(8)`.
- Funkcja systemowa `ioctl(2)`.
- Struktura `ifreq` oraz włączanie/wyłączanie interfejsów sieciowych i włączanie/wyłączanie trybu nasłuchiwania.
- Struktura `ifconf` oraz lista dostępnych interfejsów sieciowych w systemie.
- Żądanie `SIOCGIFNUM` i funkcja `getifaddrs(3)`.

Pytania i zadania

Pytania i zadania — praca własna

1. W jakich celach włącza się interfejsy sieciowe bez skonfigurowanych adresów IP?
2. Zapoznaj się z dokumentacją komendy `ethtool(8)`, `iwconfig(8)` i `iw(8)`.
3. Sprawdź jaka jest różnica pomiędzy trybem nasłuchiwania (ang. *promiscuous mode*) a trybem monitorowania (ang. *monitor mode*) w interfejsach sieci bezprzewodowych IEEE 802.11/Wi-Fi.
4. Zapoznaj się z pełną listą żądań funkcji systemowej `ioctl(2)` dla interfejsów sieciowych — *zob.* `netdevice(7)`.
5. Zapoznaj się dokumentacją funkcji `getifaddrs(3)`.
6. Zapoznaj się z implementacją funkcji `get_ifi_info` w podręczniku „Unix Network Programming, Volume 1: The Sockets Networking API” (rozdział nr 17, sekcja nr 6).

Dziękuję za uwagę!