Programowanie sieciowe
Gniazda sieciowe PF_PACKET

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 · 12 marca 2019 r.

Plan wykładu

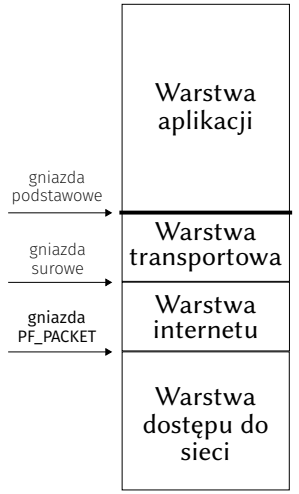
Gniazda sieciowe PF_PACKET oraz SOCK_PACKET

Gniazda sieciowe PF_PACKET oraz SOCK_PACKET (1/5)

W systemach operacyjnych GNU/Linux istnieją dwa mechanizmy systemowe umożliwiające dostęp do warstwy łącza danych: *gniazda sieciowe domeny komunikacyjnej* `PF_PACKET` (nowsze rozwiązanie) oraz *gniazda sieciowe typu* `SOCK_PACKET` (rozwiązanie przestarzałe).

Rozwiązanie nowsze oferuje więcej funkcji filtracji danych oraz działa efektywniej w odniesieniu do rozwiązania starszego.

Do utworzenia gniazd sieciowych obu rodzajów wymagane są uprawnienia administratora.



Rysunek 1: Gniazda sieciowe w odniesieniu do warstwowego modelu internetowego.

Gniazda sieciowe PF_PACKET oraz SOCK_PACKET (2/5)

PF_PACKET określa domenę komunikacyjną i wymaga typu SOCK_RAW lub SOCK_DGRAM, a SOCK_PACKET określa typ komunikacji w domenie PF_INET.

Wybranie typu SOCK_DGRAM dla gniazd sieciowych w domenie PF_PACKET spowoduje, że dane przekazywane z takiego gniazda **nie będą zawierały** nagłówka warstwy łącza danych (ang. *„cooked" packets*).
Dane wysyłane takim gniazdem sieciowym **nie powinny zawierać** nagłówka warstwy łącza danych — dane adresowe pobrane zostaną ze struktury adresowej.

Wybranie typu SOCK_RAW dla gniazda sieciowego w domenie PF_PACKET spowoduje, że dane przekazywane z takiego gniazda **będą zawierały** nagłówek warstwy łącza danych.
Dane wysyłane takim gniazdem **muszą zawierać** nagłówek warstwy łącza danych — niezmodyfikowane dane zostaną w całości przekazane do sterownika karty sieciowej i wysłane.

Gniazda sieciowe PF_PACKET oraz SOCK_PACKET (3/5)

Dla obu rodzajów gniazd sieciowy PF_PACKET (SOCK_DGRAM i SOCK_RAW) oraz SOCK_PACKET należy także określić wartość EtherType protokołu warstwy sieciowej — ramki tylko takiego protokołu będą przekazywane przez tak utworzone gniazdo sieciowe.

Wartość EtherType przekazuje się w sieciowym porządku bitowym.

Funkcjonowanie gniazd sieciowych SOCK_PACKET jest podobne do funkcjonowania gniazd sieciowych PF_PACKET typu SOCK_RAW.

zob. `packet(7)`

Gniazda sieciowe PF_PACKET oraz SOCK_PACKET (4/5)

Poniższe przykłady prezentują sposób utworzenia gniazd sieciowych obu rodzajów, które przekazują ramki *wszystkich* protokołów warstwy sieciowej:

- gniazda sieciowe domeny komunikacyjnej PF_PACKET (rozwiązanie nowsze):

```
int sfd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
```

- gniazda sieciowe typu SOCK_PACKET (rozwiązanie starsze):

```
int sfd = socket(PF_INET, SOCK_PACKET, htons(ETH_P_ALL));
```

W systemach operacyjnych UNIX oraz GNU/Linux, lista wartości EtherType zapisana jest m.in. w pliku `if_ether.h` (stałe ETH_P_*).

zob. `ethers(5)`

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

Gniazda sieciowe PF_PACKET oraz SOCK_PACKET (5/5)

Utworzenie gniazd sieciowych obu rodzajów, które przekazują tylko ramki transmitujące pakiety protokołu IPv4 wymaga następujących wywołań:

- gniazda sieciowe domeny komunikacyjnej PF_PACKET (rozwiązanie nowsze):

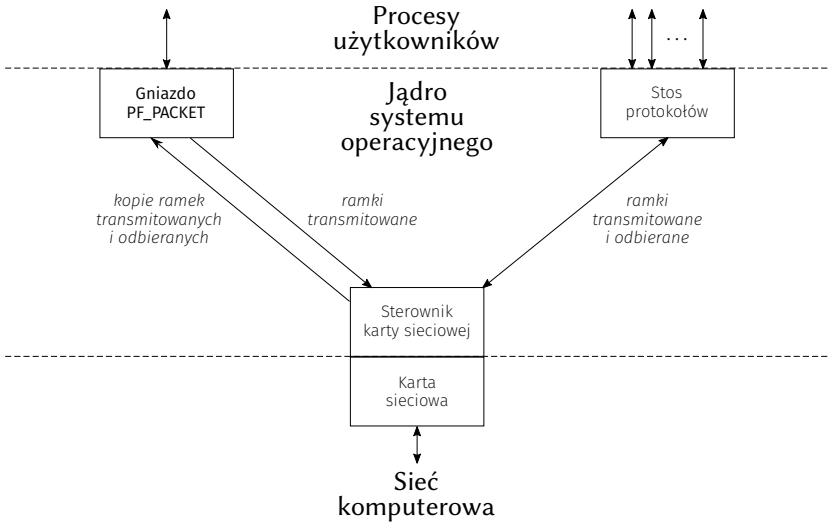
```
int sfd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP));
```

- gniazda sieciowe typu SOCK_PACKET (rozwiązanie starsze):

```
int sfd = socket(PF_INET, SOCK_PACKET, htons(ETH_P_IP));
```

Utworzenie gniazd sieciowych domeny komunikacyjnej PF_PACKET oraz typu SOCK_PACKET nie powoduje włączenia trybu nasłuchiwania w żadnym interfejsie sieciowym. W razie potrzeby, musi to zostać wykonane niezależnie, np. poprzez wywołanie funkcji systemowej `ioctl(2)`.

Gniazda sieciowe PF_PACKET w jądrze systemu operacyjnego



Rysunek 2: Gniazda sieciowe domeny komunikacyjnej PF_PACKET w jądrze systemu operacyjnego GNU/Linux.

Struktura adresowa warstwy łącza danych

Struktura sockaddr_ll (1/2)

Struktura sockaddr_ll opisuje adresy warstwy łącza danych i jest wykorzystywana podczas transmitowania i odbierania ramek oraz podczas wiązania gniazda sieciowego z adresem przy użyciu funkcji systemowej bind(2).

```
1  struct sockaddr_ll {
2    unsigned short sll_family;   /* Always PF_PACKET.        */
3    __be16         sll_protocol; /* Network layer protocol.  */
4    int            sll_ifindex;  /* Interface number.        */
5    unsigned short sll_hatype;   /* ARP hardware type.       */
6    unsigned char  sll_pkttype;  /* Packet type.             */
7    unsigned char  sll_halen;    /* Length of address.       */
8    unsigned char  sll_addr[8];  /* Data link layer address. */
9  };
```

zob. packet(7) oraz if_packet.h

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

Struktura sockaddr_ll (2/2)

Numer interfejsu, tj. wartość dla pola sll_ifindex, można uzyskać funkcją systemową ioctl(2) z żądaniem SIOCGIFINDEX.

Typy sprzętowe protokołu ARP, tj. wartość dla pola sll_hatype, zapisane są w pliku if_arp.h.

PRZYKŁAD:

```
$ cat ./if_arp.h | grep "_ETHER"
#define ARPHRD_ETHER  1
```

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

Typy ramek, tj. wartość pola sll_pkttype, zapisane są w pliku if_packet.h i zawierają następujące możliwości:

```
1  #define PACKET_HOST       0   /* To us.              */
2  #define PACKET_BROADCAST  1   /* To all.             */
3  #define PACKET_MULTICAST  2   /* To group.           */
4  #define PACKET_OTHERHOST  3   /* To someone else.    */
5  #define PACKET_OUTGOING   4   /* Originated by us.   */
```

Ramki sieciowe kopiowane do gniazda PF_PACKET

Domyślnie do gniazda sieciowego domeny komunikacyjnej PF_PACKET kopiowane są wszystkie transmitowane i odbierane ramki w systemie.

Może to jednak ulec zmianie w zależności od następujących czynników:

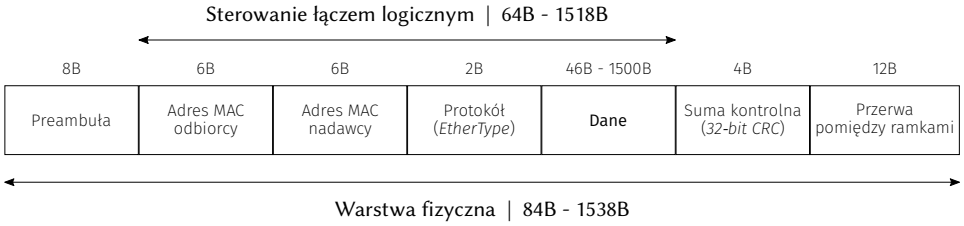
- włączenia lub wyłączenia trybu nasłuchiwania;
- wartości EtherType podanej jako trzeci parametr funkcji systemowej `socket(2)`;
- powiązania gniazda sieciowego z adresem warstwy łącza danych (i konkretnym interfejsem sieciowym) funkcją systemową `bind(2)`.

Podczas wiązania gniazda sieciowego domeny komunikacyjnej PF_PACKET z adresem warstwy łącza danych, wartości następujących pól struktury `sockaddr_ll` brane są pod uwagę: `sll_family` (w tym przypadku z wartością AF_PACKET), `sll_protocol` oraz `sll_ifindex`.

Pozostałe pola mogą zawierać wartość `0`.

Przykład komunikacji w sieci Ethernet

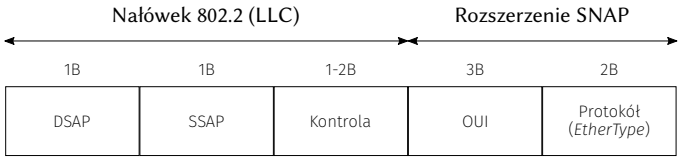
Ramka sieci Ethernet II



Preambuła = 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101011
CRC = Cyclic Redundancy Check

Rysunek 3: Format ramki sieci Ethernet II.

Rozszerzenie SNAP



DSAP = Destination Service Access Point
SSAP = Source Service Access Point
OUI = Organizationally Unique Identifier

DSAP=0xAA | SSAP=0xAA | Kontrola=0x03
OUI=0x00
Protokół=*EtherType*

Rysunek 4: Format ramki rozszerzenia SNAP (ang. *Subnetwork Access Protocol*).

Struktura ethhdr

Struktura ethhdr opisuje nagłówek ramki sieci IEEE 802.3/Ethernet II.

```
#define ETH_ALEN      6    /* Octets in one ethernet addr.  */
#define ETH_HLEN      14   /* Total octets in header.       */
#define ETH_ZLEN      60   /* Min. octets in frame sans FCS.*/
#define ETH_DATA_LEN  1500 /* Max. octets in payload.       */
#define ETH_FRAME_LEN 1514 /* Max. octets in frame sans FCS.*/
#define ETH_FCS_LEN   4    /* Octets in the FCS.            */

struct ethhdr {
 unsigned char h_dest[ETH_ALEN];   /* Destination eth addr. */
 unsigned char h_source[ETH_ALEN]; /* Source eth addr.      */
 __be16        h_proto;            /* Packet type ID field. */
} __attribute__((packed));
```

zob. `if_ether.h`

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

</> Transmisja ramki własnej sieci Ethernet (1/2)

```
#define ETH_P_CUSTOM 0x8888
int sfd, ifindex;
char *frame, *fdata;
struct ethhdr* fhead;
struct ifreq ifr;
struct sockaddr_ll sall;
sfd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_CUSTOM));
frame = malloc(ETH_FRAME_LEN);
memset(frame, 0, ETH_FRAME_LEN);
fhead = (struct ethhdr*) frame;
fdata = frame + ETH_HLEN;
strncpy(ifr.ifr_name, argv[1], IFNAMSIZ);
ioctl(sfd, SIOCGIFINDEX, &ifr);
ifindex = ifr.ifr_ifindex;
ioctl(sfd, SIOCGIFHWADDR, &ifr);
memset(&sall, 0, sizeof(struct sockaddr_ll));
```

cdn.

</> Transmisja ramki własnej sieci Ethernet (2/2)

ciąg dalszy

```
 sall.sll_family = AF_PACKET;
 sall.sll_protocol = htons(ETH_P_CUSTOM);
 sall.sll_ifindex = ifindex;
 sall.sll_hatype = ARPHRD_ETHER;
 sall.sll_pkttype = PACKET_OUTGOING;
 sall.sll_halen = ETH_ALEN;
 sscanf(argv[2], "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
   &sall.sll_addr[0], &sall.sll_addr[1], &sall.sll_addr[2],
   &sall.sll_addr[3], &sall.sll_addr[4], &sall.sll_addr[5]);
 memcpy(fhead->h_dest, &sall.sll_addr, ETH_ALEN);
 memcpy(fhead->h_source, &ifr.ifr_hwaddr.sa_data, ETH_ALEN);
 fhead->h_proto = htons(ETH_P_CUSTOM);
 memcpy(fdata, argv[3], strlen(argv[3]) + 1);
 sendto(sfd, frame, ETH_HLEN + strlen(argv[3]) + 1, 0,
        (struct sockaddr*) &sall, sizeof(struct sockaddr_ll));
 free(frame);  close(sfd);
```


</> Odbiór ramki własnej sieci Ethernet (1/2)

```
#define ETH_P_CUSTOM 0x8888
int sfd, i;  ssize_t len;
char *frame, *fdata;
struct ethhdr* fhead;
struct ifreq ifr;
struct sockaddr_ll sall;
sfd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_CUSTOM));
strncpy(ifr.ifr_name, argv[1], IFNAMSIZ);
ioctl(sfd, SIOCGIFINDEX, &ifr);
memset(&sall, 0, sizeof(struct sockaddr_ll));
sall.sll_family = AF_PACKET;
sall.sll_protocol = htons(ETH_P_CUSTOM);
sall.sll_ifindex = ifr.ifr_ifindex;
sall.sll_hatype = ARPHRD_ETHER;
sall.sll_pkttype = PACKET_HOST;
sall.sll_halen = ETH_ALEN;
```

cdn.

</> Odbiór ramki własnej sieci Ethernet (2/2)

ciąg dalszy

```
17  bind(sfd, (struct sockaddr*) &sall,
18       sizeof(struct sockaddr_ll));
19  while(1) {
20    frame = malloc(ETH_FRAME_LEN);
21    memset(frame, 0, ETH_FRAME_LEN);
22    fhead = (struct ethhdr*) frame;
23    fdata = frame + ETH_HLEN;
24    len = recvfrom(sfd, frame, ETH_FRAME_LEN, 0, NULL, NULL);
25    /* ... */
26    free(frame);
27  }
28  close(sfd);
```


Podsumowanie

Podsumowanie

- Gniazda sieciowe domeny komunikacyjnej `PF_PACKET` oraz gniazda sieciowe typu `SOCK_PACKET` (domeny komunikacyjnej `PF_INET`).
- Struktura adresowa warstwy łącza danych `sockaddr_ll`.
- Struktura adresowa ramki sieci IEEE 802.3/Ethernet II `ethhdr`.
- Transmisja i odbiór ramek własnych w sieciach IEEE 802.3/Ethernet II.

Pytania i zadania

Pytania i zadania — praca własna

1. Czy istnieją wartości EtherType mniejsze niż `1536` (`0x0600`)?
2. Ramki protokołów sieci bezprzewodowych IEEE 802.11 nie posiadają pola typ, w którym zawarta byłaby wartość EtherType. Na jakiej podstawie rozpoznawany jest zatem w tym przypadku protokół warstwy sieciowej?
3. Czy gniazdo sieciowe domeny komunikacyjnej `PF_PACKET` służące do wysyłania danych można powiązać z adresem przy użyciu funkcji `bind(2)`? Jeżeli tak, to w jakim celu?

Dziękuję za uwagę!