Programowanie sieciowe
Gniazda sieciowe PF_NETLINK

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 · 10 kwietnia 2019 r.

Plan wykładu

Tablice routingu w systemach operacyjnych GNU/Linux

Tablice routingu w systemach operacyjnych GNU/Linux

Współczesne systemy operacyjne GNU/Linux wspierają wiele tablic routingu.

W systemach tych, podstawowymi dwiema tablicami routingu są tablice: `local` (zarządzana przez jądro systemu) oraz `main` (domyślna tablica routingu dostępna dla administratora systemu).

Oprócz tych dwóch podstawowych tablic możliwe jest skonfigurowanie do 252 dodatkowych tablic routingu.

Mechanizm ten umożliwia konfigurowanie *polityk trasowania* (ang. *policy-based routing* lub *policy routing*) — wybór tras dla pakietów odbywa się wówczas dodatkowo na podstawie *polityki* (tj. zasad) ustalanych przez administratora systemu (np. w oparciu o adres nadawcy pakietu).

Tablice `local` oraz `main`

Tablica routingu `local` zarządzana jest w pełni przez jądro systemu operacyjnego i nie powinna być modyfikowana przez administratora systemu.

Tablica ta zawiera informacje o przypisaniu skonfigurowanych adresów IP do interfejsów sieciowych oraz o adresach rozgłoszeniowych w lokalnie dostępnych sieciach IP.

Komenda systemowa `route(8)` nie wyświetla zawartości tej tablicy, jest to jednak możliwe z zastosowaniem komendy systemowej `ip(8)`.

Tablica routingu `main` jest domyślną tablicą routingu w systemach operacyjnych GNU/Linux i to ona obsługiwana jest komendą systemową `route(8)`.

Skonfigurowanie nowego adresu IP w systemie skutkuje dodaniem nowych wpisów do tablic `local` oraz `main`.

Obsługa tablic routingu komendą systemową `ip(8)`

Obsługa tablic routingu komendą systemową `ip(8)` (1/3)

Komenda systemowa `ip(8)` umożliwia manipulowanie wszystkim tablicami routingu w systemie.

Odwołania do poszczególnych tablic mogą się odbywać z użyciem ich numerów lub nadanych im nazw zgodnie z odwzorowaniami zapisanym w pliku konfiguracyjnym `/etc/iproute2/rt_tables`.

PRZYKŁAD:

```
$ grep -v "^#" /etc/iproute2/rt_tables
255  local
254  main
253  default
0    unspec
```


Obsługa tablic routingu komendą systemową `ip(8)` (2/3)

PRZYKŁAD:

```
$ ip route show
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.2

$ ip route show table main
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.2

$ ip route show table default
$
```

Obsługa tablic routingu komendą systemową `ip(8)` (3/3)

Przykład:

```
$ ip route show table local
broadcast 127.0.0.0 dev lo proto kernel scope link src 127.0.0.1
local 127.0.0.0/8 dev lo proto kernel scope host src 127.0.0.1
local 127.0.0.1 dev lo proto kernel scope host src 127.0.0.1
broadcast 127.255.255.255 dev lo proto kernel scope link src
↪127.0.0.1
broadcast 192.168.1.0 dev eth0 proto kernel scope link src
↪192.168.1.2
local 192.168.1.2 dev eth0 proto kernel scope host src 192.168.1.2
broadcast 192.168.1.255 dev eth0 proto kernel scope link src
↪192.168.1.2
```


Konfiguracja polityki trasowania (1/2)

Poniższy przykład przedstawia konfigurację *dodatkowej* bramy domyślnej wybieranej na podstawie adresu nadawcy/odbiorcy.

Przykład:

```
$ ip rule list
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default

$ echo '1  my-tab' >> /etc/iproute2/rt_tables

$ ip route add 10.0.0.0/24 dev eth1 src 10.0.0.2 table my-tab
$ ip route add default via 10.0.0.1 dev eth1 table my-tab

$ ip rule add from 10.0.0.2/32 table my-tab
$ ip rule add to 10.0.0.2/32 table my-tab
```


Konfiguracja polityki trasowania (2/2)

Przykład (ciąg dalszy):

```
$ ip rule list
0:      from all lookup local
32764:  from all to 10.0.0.2 lookup my-tab
32765:  from 10.0.0.2 lookup my-tab
32766:  from all lookup main
32767:  from all lookup default

$ grep -v "^#" /etc/iproute2/rt_tables
255     local
254     main
253     default
0       unspec
1       my-tab
```

zob. `ip-rule(7)`

Gniazda sieciowe `PF_NETLINK`

Gniazda sieciowe PF_NETLINK (1/3)

Gniazda sieciowe PF_NETLINK umożliwiają wymianę informacji (komunikatów) pomiędzy jądrem systemu operacyjnego a procesami przestrzeni użytkownika.

Mechanizm te składa się z interfejsu gniazd sieciowych dla procesów przestrzeni użytkownika oraz z wewnętrznego interfejsu dla modułów jądra systemu operacyjnego.

Gniazda sieciowe PF_NETLINK mogą być tworzone jako gniazda surowe (SOCK_RAW) lub datagramowe (SOCK_DGRAM).

Podczas tworzenia gniazda sieciowego PF_NETLINK, trzeci argument wywołania funkcji systemowej socket(2) określa moduł jądra (lub grupę procesów jądra) z którymi odbywać się będzie komunikacja.

zob. netlink(7)

Gniazda sieciowe PF_NETLINK (2/3)

Przykład utworzenia gniazda sieciowego PF_NETLINK:

```
int sfd = socket(PF_NETLINK, SOCK_RAW, netlink_family);
```

lub:

```
int sfd = socket(PF_NETLINK, SOCK_DGRAM, netlink_family);
```

Stałe, które mogą być trzecim argumentem wywołania funkcji systemowej socket(2) dla gniazd sieciowych PF_NETLINK zdefiniowane są w pliku linux/netlink.h.

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

Gniazda sieciowe PF_NETLINK (3/3)

PRZYKŁAD:

```
$ grep " NETLINK_" ./netlink.h | head -n 10
#define NETLINK_ROUTE      0  /* Routing/device hook                */
#define NETLINK_UNUSED     1  /* Unused number                      */
#define NETLINK_USERSOCK   2  /* Reserved for user mode socket      */
#define NETLINK_FIREWALL   3  /* Unused number                      */
#define NETLINK_SOCK_DIAG  4  /* socket monitoring                  */
#define NETLINK_NFLOG      5  /* netfilter/iptables ULOG            */
#define NETLINK_XFRM       6  /* ipsec                              */
#define NETLINK_SELINUX    7  /* SELinux event notifications        */
#define NETLINK_ISCSI      8  /* Open-iSCSI                         */
#define NETLINK_AUDIT      9  /* auditing                           */
```


Struktura adresowa sockaddr_nl

Gniazda sieciowe PF_NETLINK muszą zostać powiązane z adresem procesu (funkcją systemową bind(2)), który określany jest w strukturze sockaddr_nl.

```
1 struct sockaddr_nl {
2   sa_family_t     nl_family;  /* AF_NETLINK             */
3   unsigned short  nl_pad;     /* Zero.                  */
4   __u32           nl_pid;     /* Process PID.           */
5   __u32           nl_groups;  /* Multicast groups mask. */
6 };
```

zob. netlink.h

Struktura nagłówka komunikatów nlmsghdr

Nagłówki komunikatów przesyłanych gniazdami sieciowymi PF_NETLINK opisane są strukturą nlmsghdr.

```
1 struct nlmsghdr {
2   __u32 nlmsg_len;   /* Length of message including header. */
3   __u16 nlmsg_type;  /* Type of message content.            */
4   __u16 nlmsg_flags; /* Additional flags.                   */
5   __u32 nlmsg_seq;   /* Sequence number.                    */
6   __u32 nlmsg_pid;   /* Sender port ID.                     */
7  };
```

zob. netlink.h

Dostępne flagi dla struktury nlmsghdr zdefiniowane są w pliku linux/netlink.h.

Gniazda obsługi tablic routingu
NETLINK_ROUTE

Gniazda obsługi tablic routingu NETLINK_ROUTE

```
int sfd = socket(PF_NETLINK, SOCK_RAW, NETLINK_ROUTE);
```

Gniazda obsługi tablic routingu NETLINK_ROUTE (nazywane rtnetlink) umożliwiają obsługę nie tylko tablic routingu, ale także m.in. adresacji IP, ustawień interfejsów sieciowych, pamięci podręcznej protokołu ARP oraz mechanizmów kolejkowania.

Typy *operacji* (wiadomości) gniazd rtnetlink opisane są następującymi stałymi:

- RTM_NEWLINK, RTM_DELLINK, RTM_GETLINK,
- RTM_NEWADDR, RTM_DELADDR, RTM_GETADDR,
- RTM_NEWROUTE, RTM_DELROUTE, RTM_GETROUTE,
- RTM_NEWNEIGH, RTM_DELNEIGH, RTM_GETNEIGH,
- RTM_NEWRULE, RTM_DELRULE, RTM_GETRULE,
- RTM_NEWQDISC, RTM_DELQDISC, RTM_GETQDISC,
- RTM_NEWTCLASS, RTM_DELTCLASS, RTM_GETTCLASS,
- RTM_NEWTFILTER, RTM_DELTFILTER, RTM_GETTFILTER.

zob. rtnetlink(7)

Struktury nagłówków komunikatów gniazd rtnetlink (1/2)

Wiadomości transmitowane gniazdami rtnetlink, po nagłówku nlmsghdr, zawierają kolejny nagłówek, którego struktura zależna jest od typu operacji, np.: rtmsg, ifaddrmsg, ifinfomsg, ndmsg, tcmsg.

```
1  struct rtmsg {
2    unsigned char rtm_family;
3    unsigned char rtm_dst_len;
4    unsigned char rtm_src_len;
5    unsigned char rtm_tos;
6    unsigned char rtm_table;    /* Routing table id */
7    unsigned char rtm_protocol; /* Routing protocol */
8    unsigned char rtm_scope;    /* See below        */
9    unsigned char rtm_type;     /* See below        */
10   unsigned      rtm_flags;
11 };
```

zob. rtnetlink(7), rtnetlink.h

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

Struktury nagłówków komunikatów gniazd rtnetlink (2/2)

Wiadomości transmitowane gniazdami rtnetlink mogą wymagać dodatkowego nagłówka zawierającego atrybut operacji, który opisany jest stukturą rtattr:

```
1  struct rtattr {
2    unsigned short rta_len;   /* Length of option. */
3    unsigned short rta_type;  /* Type of option.   */
4    /* Data follows */
5  };
```

zob. rtnetlink.h

Pojedyncza wiadomość może zawierać do RTATTR_MAX nagłówków atrybutów.

Typy atrybutów zdefiniowane są w pliku linux/rtnetlink.h.

W konsekwencji, *sekwencja struktur* jest następująca:

nlmsghdr | rtmsg | rtattr$_1$ + *dane* | rtattr$_2$ + *dane* | ... { rtattr$_{RTATTR_MAX}$ + *dane* }

Makra obsługi gniazd rtnetlink (1/3)

Dane odbierane gniazdami rtnetlink mogą być „*strumieniem struktur*” (flaga NLM_F_MULTI i typ NLMSG_DONE):

nlmsghdr(NLM_F_MULTI)$_1$ ... | nlmsghdr$_2$ ... | ... | nlmsghdr(NLMSG_DONE)$_n$ ...

W związku z tym, dla łatwiejszego operowania na odbieranych danych, dla programistów dostępne są odpowiednie makra, m.in.:

```
int NLMSG_OK(struct nlmsghdr *nlh, int len);
```

weryfikuje, czy wiadomość gniazd sieciowych PF_NETLINK odebrana została w całości.

```
struct nlmsghdr *NLMSG_NEXT(struct nlmsghdr *nlh, int len);
```

zwraca wskaźnik, na kolejną wiadomość w strumieniu odbiorczym.

```
void *NLMSG_DATA(struct nlmsghdr *nlh);
```

zwraca wskaźnik na dane wiadomości gniazd sieciowych PF_NETLINK.

zob. netlink(3)

Makra obsługi gniazd rtnetlink (2/3)

Nieudokumentowane makra obsługi struktur rtmsg:

```
#define RTM_RTA(r)  ((struct rtattr*)(((char*)(r)) \
  + NLMSG_ALIGN(sizeof(struct rtmsg))))
```

```
#define RTM_PAYLOAD(n) NLMSG_PAYLOAD(n,sizeof(struct rtmsg))
```

zob. rtnetlink.h

Makra obsługi gniazd rtnetlink (3/3)

Makra obsługi struktur rtattr:

```
int RTA_OK(struct rtattr *rta, int rtabuflen);
```

weryfikuje, czy rta wskazuje na poprawną strukturę atrybutu.

```
void *RTA_DATA(struct rtattr *rta);
```

zwraca wskaźnik na dane atrybutu.

```
unsigned int RTA_PAYLOAD(struct rtattr *rta);
```

zwraca rozmiar danych atrybutu.

```
struct rtattr *RTA_NEXT(struct rtattr *rta, unsigned int rtablen);
```

zwraca wskaźnik na kolejną strukturę atrybutu w odebranej wiadomości.

zob. rtnetlink(3)

</> Odebranie tablicy routingu — fragmenty (1/3)

```
struct reqhdr {
  struct nlmsghdr nl;
  struct rtmsg    rt;
};

sfd = socket(PF_NETLINK, SOCK_RAW, NETLINK_ROUTE);
snl.nl_family = AF_NETLINK;
snl.nl_pid = 0;
req.nl.nlmsg_len = NLMSG_LENGTH(sizeof(struct rtmsg));
req.nl.nlmsg_type = RTM_GETROUTE;
req.nl.nlmsg_flags = NLM_F_REQUEST | NLM_F_ROOT;
req.nl.nlmsg_seq = 0;
req.nl.nlmsg_pid = getpid();
req.rt.rtm_family = AF_INET;
req.rt.rtm_table = RT_TABLE_MAIN;
```

cdn.

</> Odebranie tablicy routingu — fragmenty (2/3)

ciąg dalszy

```
sendto(sfd, (void*) &req, sizeof(struct reqhdr), 0,
       (struct sockaddr*) &snl, sizeof(struct sockaddr_nl));
ptr = buf;
nllen = 0;
do {
  rclen = recv(sfd, ptr, sizeof(buf) - nllen, 0);
  nlp = (struct nlmsghdr*) ptr;
  ptr += rclen;
  nllen += rclen;
} while(nlp->nlmsg_type == NLMSG_DONE);
```

cdn.

</> Odebranie tablicy routingu — fragmenty (3/3)

ciąg dalszy

```
nlp = (struct nlmsghdr*) buf;
for(;NLMSG_OK(nlp, nllen); nlp = NLMSG_NEXT(nlp, nllen)) {
  rtp = (struct rtmsg*) NLMSG_DATA(nlp);
  atp = (struct rtattr*) RTM_RTA(rtp);
  atlen = RTM_PAYLOAD(nlp);
  for(;RTA_OK(atp, atlen); atp = RTA_NEXT(atp, atlen)) {
    // check and process every attribute
  }
}
```


Podsumowanie

Podsumowanie

- Tablice routingu w systemach operacyjnych GNU/Linux (m.in. `local`, `main`, `default` oraz `unspec`).
- Obsługa tablic routingu komendą systemową `ip(8)`.
- Gniazda sieciowe `PF_NETLINK`;
  - stałe `NETLINK_*`;
  - struktura adresowa `sockaddr_nl`;
  - struktura nagłówka wiadomości `nlmsghdr`.
- Gniazda obsługi tablic routingu `NETLINK_ROUTE` (`rtnetlink`);
  - typy operacji/wiadomości — stale `RTM_*`;
  - struktura nagłówka wiadomości `rtmsg`;
  - struktura atrybutu `rtattr`;
  - makra obsługi gniazd `rtnetlink`.

Pytania i zadania

Pytania i zadania — praca własna

1. Korzystając z komendy systemowej `ip(8)` sprawdź zawartość tablicy routingu o nazwie `unspec` w systemie operacyjnym GNU/Linux.
2. Jakie mogą być zastosowania polityk trasowania (z użyciem wielu tablic routingu)?
3. Zapoznaj się z dokumentacją dotyczącą obsługi tablic i polityk routingu dostępną pod następującym adresem internetowym: http://linux-ip.net/html/.
4. Zapoznaj się z artykułem pt. „Communicating between the kernel and user-space in Linux using Netlink sockets” dostępnym pod następującym adresem internetowym: http://onlinelibrary.wiley.com/doi/10.1002/spe.981/abstract.

Dziękuję za uwagę!