Katedra Informatyki

Wydział Informatyki i Telekomunikacji
Politechnika Krakowska

Programowanie Usług Sieciowych

mgr inż. Michał Niedźwiecki

Protokół Netlink

Laboratorium: 06,

system operacyjny: Linux

Kraków, 2021

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące protokołu Netlink oraz programowania aplikacji z jego wykorzystaniem. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie gniazd w oparciu o protokół Netlink. Netlink został zaprojektowany z myślą o wymianie informacji pomiędzy jądrem systemu Linux a procesami przestrzeni użytkownika. Początkowo (jądro 2.0), Netlink wykorzystywał do komunikacji urządzenia znakowe (ang. character devices), jednak mechanizm ten zastąpiono interfejsem gniazd w kolejnej, stabilnej wersji jądra (2.2). Gniazda (ang. sockets) są najbardziej uniwersalnym sposobem komunikacji spośród mechanizmów IPC (ang. Inter Process Communication - komunikacja międzyprocesowa). Gniazd Netlink używa się do wymiany informacji między procesami i modułami jądra działającymi w obrębie jednej maszyny.

Netlink stanowi interfejs pomiędzy komponentami CPC (ang. Control Plane Component), a FPC (ang. Forwarding Plane Component) architektury sieciowej. Wiele narzędzi programistycznych (np.: pakiety iproute2 i Quagga) wykorzystuje protokół Netlink do zarządzania stosem sieciowym, parametrami interfejsów, tablicą forwardingu, itp.

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi:

- obsługi programu ip z pakietu iproute2 [5, 6]
- adresowania w protokołach IPv4 i IPv6 [1, 2]
- stosowania gniazd w systemie Linux [1]
- stosowania funkcji sendto(), sendmsg() oraz [1]
  recvfrom() i recvmsg()
- konwersji adresów IP - inet_ntop(), inet_pton() [1]

Należy także zapoznać się z zagadnieniami dotyczącymi: [3, 7, 8]

- nagłówków komunikatów Netlink
- komunikacji za pomocą gniazd Netlink
- warstwy kontrolnej (ang. Control Plane) oraz [4]
  warstwy forwardingu (ang. Forwarding Plane)

Literatura:

[1] W.R. Stevens, „Programowanie Usług Sieciowych”, „API: gniazda i XTI”.

[2] IETF (http://www.ietf.org/), RFC 4291, „IP Version 6 Addressing Architecture”

[3] IETF, RFC 3549, „Linux Netlink as an IP Services Protocol”

[4] IETF, RFC 3746, „Forwarding and Control Element Separation Framework”

[5] „IPROUTE2 Utility Suite Howto”, http://www.policyrouting.org/iproute2.doc.html

[6] „Linux Advanced Routing & Traffic Control HOWTO”, http://lartc.org/howto/

[7] MAN (3, 7), „netlink”

[8] MAN (3, 7), „rtnetlink”

1.3. Opis laboratorium

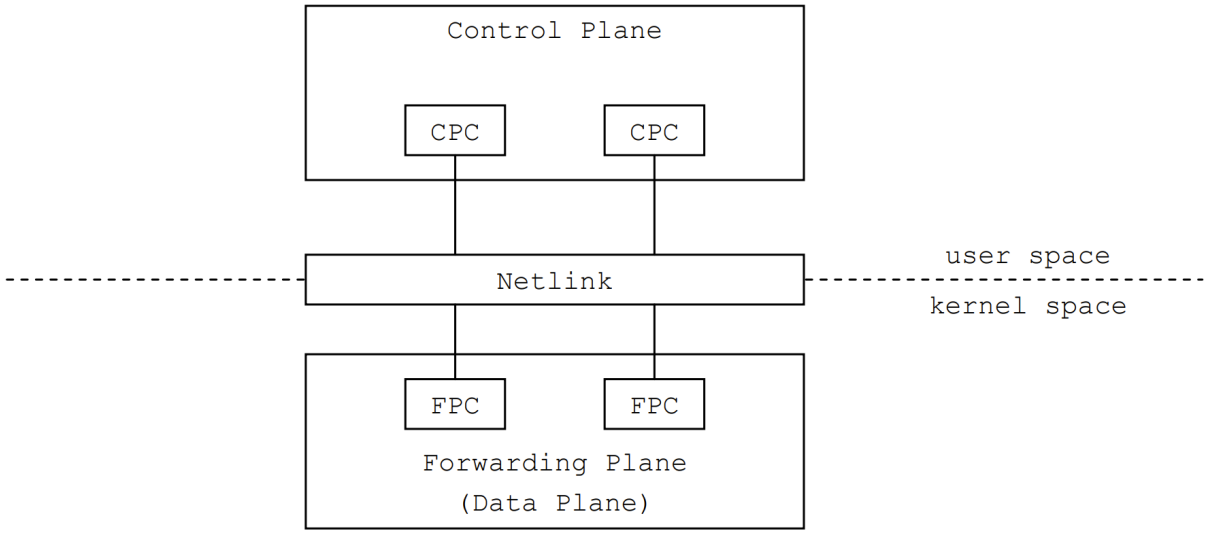
1.3.1. Protokół Netlink

Netlink jest protokołem, który umożliwia komunikację w obrębie jednej stacji sieciowej za pomocą interfejsu gniazd. Komunikacja może odbywać się:

- między procesami użytkownika
- między procesem użytkownika a jądrem (modułami jądra),
- między modułami jądra (oddzielne API).

Netlink nie zapewnia niezawodnej transmisji. Jest protokołem bezpołączeniowym, który dostarcza usług datagramowych.

Netlink zastępuje interfejs ioctl w kwestiach zarządzania i konfiguracji usług sieciowych. Za pomocą protokołu Netlink możliwa jest komunikacja między komponentami warstwy CP (ang. Control Plane) i FP (ang. Forwarding Plane) architektury sieciowej[4].



Rys.1. Architektura sieciowa z wyróżnioną warstwą danych oraz warstwą kontrolną.

Komponenty FPC są odpowiedzialne za przetwarzanie i forwarding pakietów, enkapsulację (tunelowanie) i dekapsulację, traffic shaping, NAT, szyfrowanie, filtrowanie ruchu, itp. Kolejne etapy przez jakie przechodzi pakiet w warstwie FP noszą nazwę ścieżki forwardingu (ang. forwarding path).

Komponenty CPC kontrolują i dyktują zachowanie komponentów FPC. W warstwie kontrolnej zaimplementowane są protokoły routingu dynamicznego (np.: RIP, OSPF, BGP), protokół SNMP oraz programy (CLI lub GUI), które umożliwiają zarządzanie i konfigurację

warstwy forwardingu. Komponenty CPC najczęściej są zaimplementowane w przestrzeni użytkownika.

Przykładowo, implementacje protokołów routingu dynamicznego w warstwie kontrolnej utrzymują własne tablice routingu - RIB (ang. Routing Information Base). Na podstawie otrzymanych pakietów z aktualizacjami tras modyfikują RIB i eksportują trasy do tablicy forwardingu – FIB (ang. Forwarding Information Base) w warstwie FP.

Eksport tras między warstwami jest dokonywany za pomocą interfejsu gniazd i protokołu Netlink. Na podstawie FIB podejmowane są decyzje związane z forwardingiem datagramów IP. Tablice FIB są więc zoptymalizowane pod kątem wyszukiwania najlepszej trasy. Natomiast tablice RIB są zoptymalizowane pod kątem łatwego i szybkiego uaktualniania przez protokoły routingu. W celu pominięcia szczegółów implementacyjnych często stosuje się nazwę „tablica routingu” mówiąc o FIB.

1.3.2. Funkcje operujące na gniazdach Netlink

W celu utworzenia gniazda dla protokołu Netlink należy wywołać funkcję socket():

```
#include <sys/types.h>
#include <sys/socket.h>

socket(int domain, int type, int protocol);
```

Argument domain powinien zawsze przyjmować wartość PF_NETLINK, a argument type wartość SOCK_DGRAM lub SOCK_RAW. Parametr protocol określa komponent stosu sieciowego, z którym będzie odbywać się komunikacja. Użycie wartości NETLINK_ROUTE pozwala na zarządzanie:

- parametrami warstwy łącza danych interfejsów sieciowych,
- adresacją IP,
- tablicą forwardingu,
- tablicą ARP i tablicą sąsiedztwa (dla IPv6),
- dyscyplinami (strategiami) kolejkowania,
- sterowaniem przepływu danych (ang. traffic shaping).

Parametr protocol nazywa się rodziną Netlink (ang. Netlink family).

Komunikaty Netlink mogą być wysyłane za pomocą funkcji systemowych sendto() lub sendmsg().

Funkcja sendto() wysyła dane z bufora wskazywanego przez argument buff pod adres określony przez argument to:

```
#include <sys/socket.h>

ssie_t sendto(int sockfd, const void *buff, size_t nbytes, int flags,
    const struct sockaddr *to, socklen_t addrlen);
```

Parametr	Opis
sockfd	deskryptor gniazda
*buff	wskaźnik do bufora, z którego wysyła się dane
nbytes	rozmiar wysyłanych danych w bajtach
flags	opcjonalne flagi
*to	wskaźnik na strukturę zawierającą adres pod który mają być wysłane dane; parametr jest wykorzystywany w przypadku protokołów bezpołączeniowych (jak *Netlink*)
addrlen	rozmiar struktury gniazdowej

Funkcja sendmsg() jest najbardziej ogólną spośród funkcji odpowiedzialnych za wysyłanie danych:

```
#include <sys/socket.h>

ssie_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

Parametr	Opis
sockfd	deskryptor gniazda
*msg	wskaźnik na strukturę zawierającą informacje adresowe, dane do wysłania oraz informacje kontrolne
flags	opcjonalne flagi

Poniżej przedstawiona jest postać struktury msghdr.

```
#include <bits/socket.h> /* Załączany przez <sys/socket.h> */

struct msghdr {
    void *msg_name;
    socklen_t msg_namelen;
    struct iovec *msg_iov;
    size_t msg_iovlen;
    void *msg_control;
    socklen_t msg_controllen;
    int msg_flags;
};
```

Znaczenie poszczególnych pól struktury wyjaśnia tabela:

Parametr	Opis
`*msg_name`	wskaźnik na strukturę zawierającą adres; w przypadku `sendmsg()` jest to adres pod który mają zostać wysłane dane, a w przypadku `recvmsg()` – adres nadawcy wiadomości
`msg_namelen`	rozmiar struktury adresowej w bajtach
`*msg_iov`	wskaźnik na tablicę struktur `iovec`; każda struktura `iovec` zawiera bufor przechowujący dane do wysłania (lub odebrane dane) oraz rozmiar bufora
`msg_iovlen`	liczba struktur `iovec` w tablicy wskazywanej przez `msg_iov`
`*msg_control`	wskaźnik na bufor przechowujący informacje kontrolne/pomocnicze; w przypadku protokołu *Netlink*, pole nie ma sprecyzowanego zastosowania (wskaźnik jest ustawiany na NULL)
`msg_controllen`	rozmiar bufora dla informacji pomocniczych (zero w przypadku protokołu *Netlink*)
`flags`	opcjonalne flagi; wykorzystywane tylko podczas odbierania wiadomości za pomocą `recvmsg()`

Dane wysyłane za pomocą gniazda są przechowywane są w tablicy struktur iovec:

```
#include <bits/uio.h> /* Załączany przez <sys/uio.h> i <sys/socket.h> */

struct iovec
{
    void *iov_base; /* Wskaźnik na dane. */
    size_t iov_len; /* Rozmiar danych w bajtach. */
};
```

Ze względu na sposób w jaki konstruuje się datagram Netlink, tablica danych do wysłania wskazywana przez parametr msg_iov zawiera najczęściej jeden element (pole msg_iovlen struktury msghdr przyjmuje wartość 1).

Odbieranie komunikatów Netlink jest możliwe za pomocą funkcji recvfrom() lub recvmsg().

Funkcja recvfrom() pozwala określić adres z jakiego otrzymany został datagram Netlink.

```
#include <sys/socket.h>

ssie_t recvfrom(int sockfd, void *buff, size_t nbytes, int flags,
     const struct sockaddr *from, socklen_t *addrlen);
```

Parametr	Opis
sockfd	deskryptor gniazda
*buff	wskaźnik do bufora, do którego pobiera się dane
nbytes	rozmiar bufora w bajtach
flags	opcjonalne flagi
*from	wskaźnik do struktury adresowej wypełnianej przez funkcję recfrom() adresem gniazda partnera
*addrlen	wskaźnik do rozmiaru struktury gniazdowej

Podobne możliwości posiada funkcja recvmsg():

```
#include <sys/socket.h>

ssize_t recvmsg(int s, struct msghdr *msg, int flags);
```

Atutem funkcji recvmsg() w stosunku do recvfrom() jest możliwość wykorzystania pola flags struktury msghdr. Na jego podstawie można stwierdzić, czy datagram Netlink został przycięty. Jeżeli pole ma wartość MSG_TRUNC oznacza to, że odebrana została część datagramu (rozmiar datagramu w buforze gniazda jest większy od rozmiaru bufora zaalokowanego przez proces).

Funkcje odpowiedzialne za wysyłanie i odbieranie danych wykorzystują strukturę adresową sockaddr_nl zdefiniowaną w pliku nagłówkowym <linux/netlink.h>.

```
#include <linux/netlink.h>

struct sockaddr_nl {
    sa_family_t nl_family; /* AF_NETLINK */
    unsigned short nl_pad; /* Pole wyzerowane */
    __u32 nl_pid; /* Process ID */
    __32 nl_groups; /* Multicast groups mask */
};
```

Istotną cechą protokołu Netlink jest możliwość transmisji w trybie unicast lub multicast. Pole nl_pid struktury sockaddr_nl przechowuje adres typu unicast, który jednoznacznie identyfikuje gniazdo. Zazwyczaj jest to PID procesu, do którego należy gniazdo. Jeżeli proces jest właścicielem kilku gniazd Netlink, to tylko jedno z nich może być identyfikowane przez PID procesu, pozostałym gniazdom należy przypisać unikalne identyfikatory.

Identyfikator można związać z gniazdem za pomocą funkcji bind(). Jeżeli do funkcji bind() przekaże się strukturę sockaddr_nl z wyzerowanym polem nl_pid, to jądro systemu będzie

odpowiedzialne za przypisanie unikalnego identyfikatora. Dla gniazda, które zostało utworzone jako pierwsze, jądro przypisze PID procesu.

Transmisja typu unicast wymaga wyzerowania pola nl_groups.

Wysyłając komunikat Netlink należy pamiętać o następujących zasadach:

- Jeżeli odbiorcą komunikatu ma być jądro systemu, pole nl_pid musi być wyzerowane.
- W przypadku, gdy odbiorcą komunikatu ma być proces użytkownika, pole nl_pid powinno zawierać identyfikator zdalnego gniazda (zazwyczaj jest to PID procesu, do którego należy zdalne gniazdo).

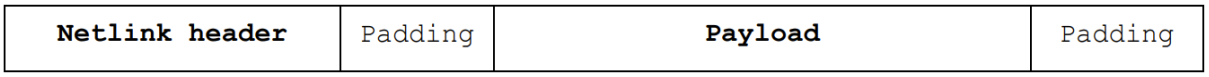
Pole nl_groups jest maską bitową, która określa numery grup multicast. Pole ma 32 bity z czego wynika, że dla danej rodziny Netlnik (np.: NETLINK_ROUTE) mogą istnieć 32 grupy multicast. Kiedy dla gniazda wywoływana jest funkcja bind(), maska nl_groups w strukturze sockaddr_nl określa, że gniazdo jest zainteresowanym odbiorcą komunikatów ze zdefiniowanych grup multicast. Jeżeli pole nl_groups ma wartość zero, oznacza to, że gniazdo nie będzie odbierać komunikatów multicast.

Ustawienie pola nl_groups na wartość różną od zera podczas wysyłania komunikatu spowoduje, że komunikat zostanie wysłany do zdefiniowanych grup multicast. Tylko procesy uprzywilejowane lub ze zdolnością (ang. capability) CAP_NET_ADMIN mogą wykorzystywać transmisję typu multicast.

Nazwy grup multicast dla rodziny NETLINK_ROUTE są zdefiniowane w pliku nagłówkowym <linux/rtnetlink.h>. Przykładowo, RTMGRP_IPV4_IFADDR pozwala na odbiór komunikatów multicast informujących o zmianie adresów IPv4 (dodanie lub usunięcie adresu), a RTMGRP_LINK pozwala na odbiór powiadomień związanych ze zmianą stanu lub parametrów interfejsów sieciowych (adresu MAC, MTU, stanu administracyjnego, itp.).

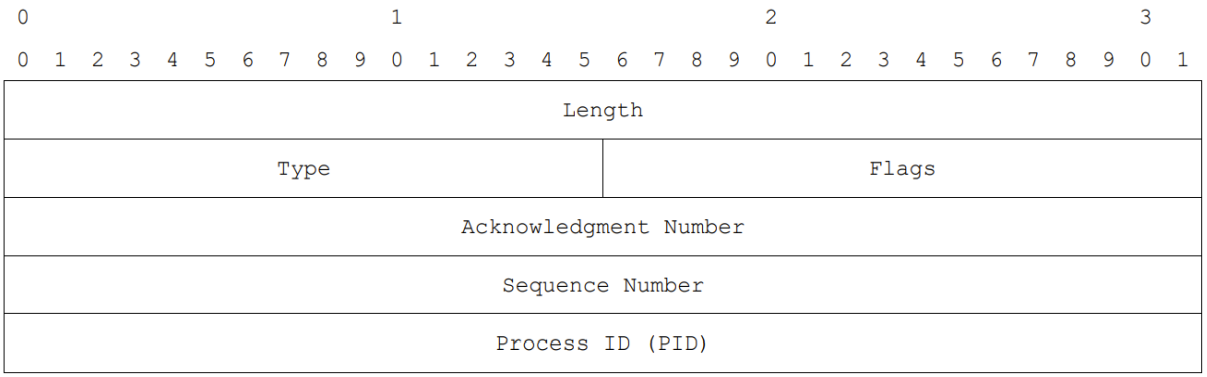
1.3.3. Budowa komunikatu Netlink

Datagram Netlink składa się z nagłówka Netlink (ang. Netlink header) oraz pola danych (ang. payload). Pomiędzy nagłówkiem a danymi oraz bezpośrednio za danymi może znajdować się opcjonalne dopełnienie (ang. padding).



Rys.2. Datagram protokołu Netlink widziany jako strumień bajtów.

Format nagłówka Netlink przedstawiony jest na rys. 3.



Rys.3. Budowa nagłówka Netlink[3].

- Length – długość komunikatu (nagłówka i danych) w bajtach,
- Type – typ komunikatu,
- Flags – flagi określające dodatkowe opcje,
- Sequence Number – numer sekwencyjny komunikatu. Aplikacja jest odpowiedzialna za wypełnienie lub weryfikację tego pola.
- Process ID – PID procesu wysyłającego komunikat lub zero, gdy komunikat jest wysyłany przez jądro systemu.

Struktura nagłówka Netlink, flagi oraz podstawowe typy komunikatów zdefiniowane są w pliku <linux/netlink.h>.

```
#include <linux/netlink.h>

struct nlmsghdr {
    __u32 nlmsg_len;
    __u16 nlmsg_type;
    __u16 nlmsg_flags;
    __u32 nlmsg_seq;
    __u32 nlmsg_pid;
};
```

Przykładowe flagi[7] to:

Flaga	**Znaczenie**
NLM_F_REQUEST	musi być ustawiona dla wszystkich komunikatów, które są zapytaniem (ang. *request*)
NLM_F_MULTI	ustawiona jeżeli datagram należy do wiadomości złożonej z wielu komunikatów (ang. *multipart message*); ostatni nagłówek *Netlink* w sekwencji takich komunikatów posiada pole nlmsg_type ustawione na wartość NLMSG_DONE
NLM_F_ACK	żądanie potwierdzenia w przypadku powodzenia operacji
NLM_F_ROOT	żądanie zwrócenia całej tablicy, a nie jednego wpisu

Podstawowe typy komunikatów Netlink przedstawia poniższa tabela:

Typ	Znaczenie
NLMSG_NOOP	prośba o zignorowanie komunikatu
NLMSG_ERROR	sygnalizuje wystąpienie błędu; dane (*payload*) zawierają kod błędu oraz nagłówek (bez danych) komunikatu *Netlink*, który spowodował wystąpienie błędu
NLMSG_DONE	komunikat jest ostatni w sekwencji wiadomości (ang. *multipart message*); nagłówek pełni wyłącznie funkcję kontrolną i nie jest wykorzystywany do transportu danych; uwaga: nagłówek typu NLMSG_DONE może być przesyłany zawsze, nawet po jednym datagramie *Netlink*

W przypadku komunikatu o typie NLMSG_ERROR, dostęp do kodu błędu i nagłówka komunikatu, który spowodował jego wystąpienie można uzyskać rzutując wskaźnik

ustawiony na początek pola danych (ang. payload) na wskaźnik do struktury nlmsgerr.

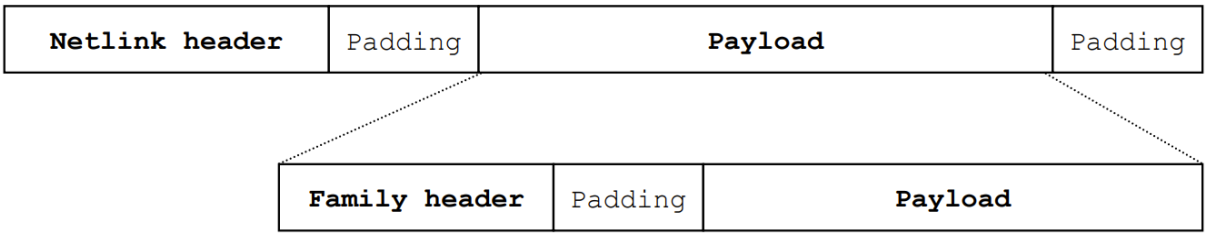
```
#include <linux/netlink.h>

struct nlmsgerr {
    int error;
    struct nlmsghdr msg;
};
```

Moduły jądra i procesy użytkownika powinny wysyłać wiadomość typu NLMSG_ERROR w odpowiedzi na każdy komunikat odpowiedzialny za wystąpienie błędu.

Jeżeli wysyłając komunikat zażądaliśmy potwierdzenia (przez ustawienie flagi NLM_F_ACK), to zostanie ono przesyłane w polu danych komunikatu Netlink. Potwierdzenie ma formę określoną przez strukturę nlmsgerr i charakteryzuje się wyzerowanym kodem błędu. Protokół Netlink nie zapewnia mechanizmu potwierdzeń – to moduł jądra lub aplikacja użytkownika jest odpowiedzialna za wysyłanie potwierdzeń. Z reguły, jądro systemu Linux wysyła potwierdzenia, jeżeli proces użytkownika zażąda ich explicite – przez ustawienie flagi NLM_F_ACK. W przypadku procesów użytkownika, programista jest odpowiedzialny za zaimplementowanie kodu odpowiedzialnego za wysyłanie potwierdzeń.

Pole danych komunikatu Netlink jest zbudowane z nagłówka rodziny (ang. family header), opcjonalnego dopełnienia oraz pola danych. Budowę pola danych komunikatu

Netlink przedstawia rys. 4.



Rys.4. Budowa pola danych komunikatu Netlink.

Nagłówek rodziny różni się w zależności od typu komunikatu (pole nlmsg_type struktury nlmsghdr). Z kolei typ komunikatu jest zależny od rodziny Netlink (parametr protocol funkcji socket()).

Dla rodziny NETLINK_ROUTE możliwe typy komunikatów są zdefiniowane w pliku <linux/rtnetlink.h>, a struktury reprezentujące nagłówek rodziny w plikach:

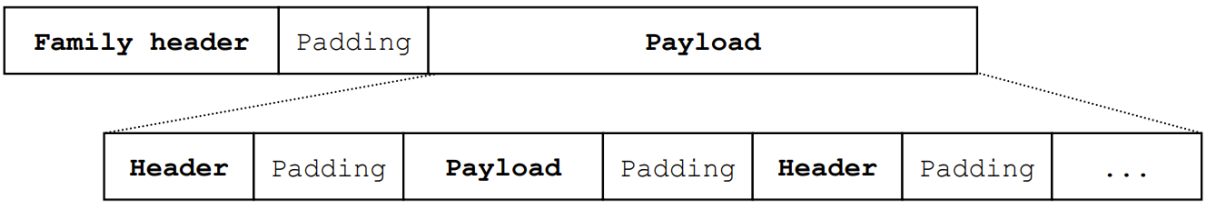
- `<linux/rtnetlink.h>`
- `<linux/if_link.h>`
- `<linux/if_addr.h>`
- `<linux/neighbour.h>`

Przykładowo, za konfigurowanie adresów IP (dodawanie, usuwanie i pobieranie informacji) odpowiadają typy RTM_NEWADDR, RTM_DELADDR, RTM_GETADDR. Wszystkie trzy wykorzystują strukturę ifaddrmsg.

```
#include <linux/if_addr.h>

struct ifaddrmsg {
    unsigned char ifa_family; /* AF_INET lub AF_INET6 */
    unsigned char ifa_prefixlen; /* Długość prefiksu adresu IP */
    unsigned char ifa_flags; /* Flagi określające typ adresu */
    unsigned char ifa_scope; /* Zakres adresu */
    int ifa_index; /* Indeks interfejsu */
};
```

Struktura ifaddrmsg nie posiada pól, które mogą przechowywać adres IP, czy etykietę interfejsu. Szczegółowe informacje i parametry nie są przenoszone w nagłówku rodziny, a w polu danych – jako atrybuty.

Pole danych za nagłówkiem rodziny może przenosić opcjonalne atrybuty w formie TLV (ang. type-length-value). Nagłówek atrybutu określa typ i długość atrybutu, a wartość atrybutu reprezentowana jest przez pole payload. Przedstawia to rys.5.



Rys.5. Format opcjonalnych atrybutów za nagłówkiem rodziny.

W przypadku rodziny NETLINK_ROUTE, nagłówek atrybutu jest zdefiniowany przez strukturę rtattr.

```
#include <linux/rtnetlink.h>

struct rtattr {
    /* Długość atrybutu w bajtach (łącznie z nagłówkiem): */
    unsigned short rta_len;
    unsigned short rta_type; /* Typ atrybutu */
};
```

Podobnie jak nagłówek rodziny, typ atrybutu zależy od typu komunikatu Netlink (pole nlmsg_type struktury nlmsghdr). Typy atrybutów są zdefiniowane w tych samych plikach nagłówkowych, co struktury reprezentujące nagłówek rodziny.

Przykładowe typy atrybutów dla omówionych typów komunikatów - RTM_NEWADDR, RTM_DELADDR, RTM_GETADDR - to:

- IFA_LOCAL – adres IP interfejsu,
- IFA_LABEL – etykieta interfejsu (alias dla programu ifconfig),
- IFA_ANYCAST - adres anycast dla protokołu IPv6.

Do wykonania części praktycznej konieczna będzie znajomość jeszcze jednej struktury – ifinfomsg:

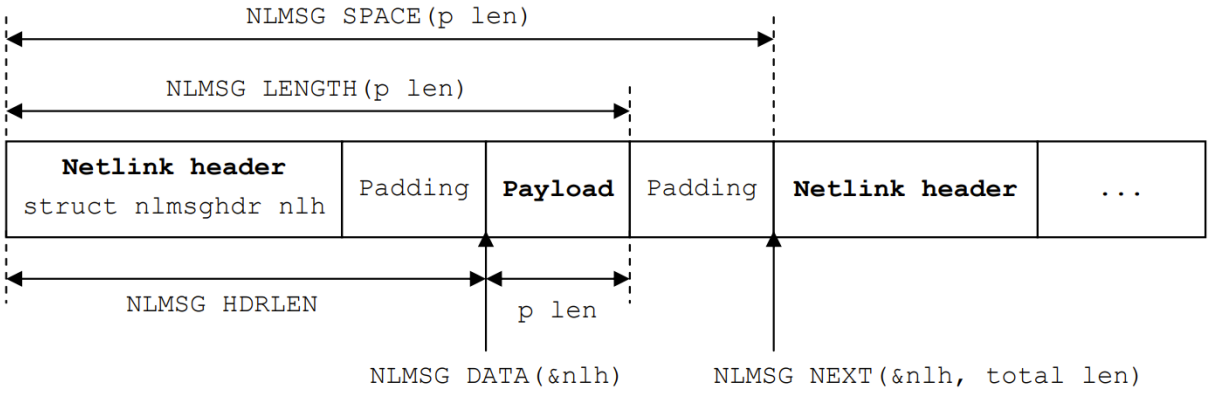
```
#include <linux/rtnetlink.h>

struct ifinfomsg{
    unsigned char ifi_family; /* Zawsze AF_UNSPEC */

    /* Dopełnienie - nie jest wykorzystywane przez programistę */
    unsigned char __ifi_pad;

    /* Typ interfejsu (np. ARPHRD_ETHER)- z pliku <net/if_arp.h> */
    unsigned short ifi_type;

    /* Indeks interfejsu */
    int ifi_index;

    /* Flagi interfejsu (np. IFF_UP) – z pliku <net/if.h> */
    unsigned ifi_flags;

    /* Pole zarezerwowane. Zawsze 0xFFFFFFFF */
    unsigned ifi_change;
};
```

Struktura ifinfomsg reprezentuje nagłówek rodziny Netlink dla komunikatów typu RTM_NEWLINK, RTM_DELLINK i RTM_GETLINK. Komunikaty te są używane do zarządzania konfiguracją i stanem interfejsów sieciowych. Strukturę ifinfomsg wykorzystuje się z atrybutami, których typy zdefiniowane są w pliku <linux/if_link.h>. Przykładowe typy atrybutów to:

- IFLA_ADDRESS – adres sprzętowy (warstwy łącza danych),
- IFLA_IFNAME – nazwa interfejsu,
- IFLA_MTU – Maximum Transmission Unit interfejsu.

1.3.4. Makra

Nagłówki oraz pola danych komunikatów Netlink muszą być w odpowiedni sposób wyrównane w buforze wykorzystywanym przez gniazda (opcjonalny padding). Z tego względu, konstrukcja komunikatów oraz dostęp do ich pól powinny odbywać się tylko za pomocą makr. Standardowe makra - dla wszystkich typów komunikatów - są zdefiniowane w pliku nagłówkowym <linux/netlink.h>.



Rys.6. Zastosowanie standardowych makr Netlink[7]

- NLMSG_ALIGN() zaokrągla wartość określoną przez parametr len do wielokrotności NLMSG_ALIGNTO (wartość zdefiniowana w <linux/netlink.h>).

```
int NLMSG_ALIGN(size_t len);
```

- NLMSG_HDRLEN jest stałą, która określa rozmiar nagłówka Netlink (struktura nlmsghdr) uwzględniając padding (jeżeli występuje). Stała jest wykorzystywana przez inne makra.
- NLMSG_LENGTH() przyjmuje jako argument rozmiar danych (payload) i zwraca rozmiar komunikatu bez uwzględnienia paddingu za polem danych. Makro jest używane do ustawienia wartości pola nlmsg_len struktury nlmsghdr.

```
int NLMSG_LENGTH(size_t len);
```

- NLMSG_SPACE() przyjmuje jako argument rozmiar pola danych i zwraca wartość określającą rozmiar komunikatu. Rozmiar komunikatu uwzględnia nagłówek, dane i wszystkie opcjonalne dopełnienia.

```
int NLMSG_SPACE(size_t len);
```

- NLMSG_DATA() zwraca wskaźnik do pola danych. Parametrem makra jest wskaźnik na strukturę nlmsghdr reprezentującą nagłówek przed polem danych.

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

- NLMSG_NEXT() jest używane w przypadku odbierania komunikatów Netlink. Jeżeli w buforze znajduje się kilka komunikatów tworzących wspólnie wieloczęściową wiadomość (tzw. multipart message), to makro NLMSG_NEXT() zwraca wskaźnik do początku następnego komunikatu (względem komunikatu wskazywanego przez parametr nlh). Parametr len określa rozmiar komunikatów w buforze. Początkowo jest to liczba bajtów zwrócona przez funkcję recvfrom() lub recvmsg(). Makro zmniejsza wartość len o długość komunikatu wskazywanego przez nlh, a dokładnie o NLMSG_ALIGN(nlh->nlmsg_len).

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

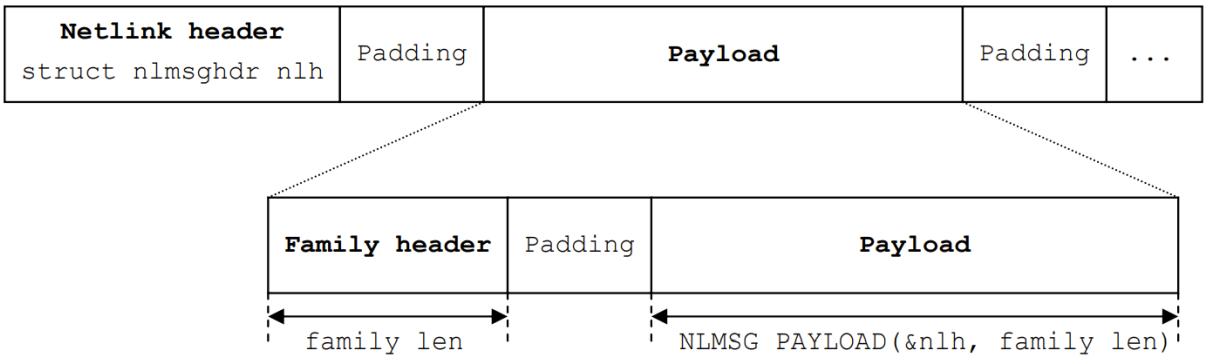
- NLMSG_OK() zwraca 1 w przypadku, gdy komunikat wskazywany przez parametr nlh nie został przycięty i może być przetwarzany. W przeciwnym wypadku zwracana jest wartość 0. Parametr len ma znaczenie takie, jak w przypadku makra NLMSG_NEXT(), ale nie jest w żaden sposób modyfikowany przez NLMSG_OK().

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

- NLMSG_PAYLOAD() przyjmuje wskaźnik na strukturę reprezentującą nagłówek Netlink – nlh oraz rozmiar nagłówka rodziny - len. Makro zwraca rozmiar pola danych za nagłówkiem rodziny.

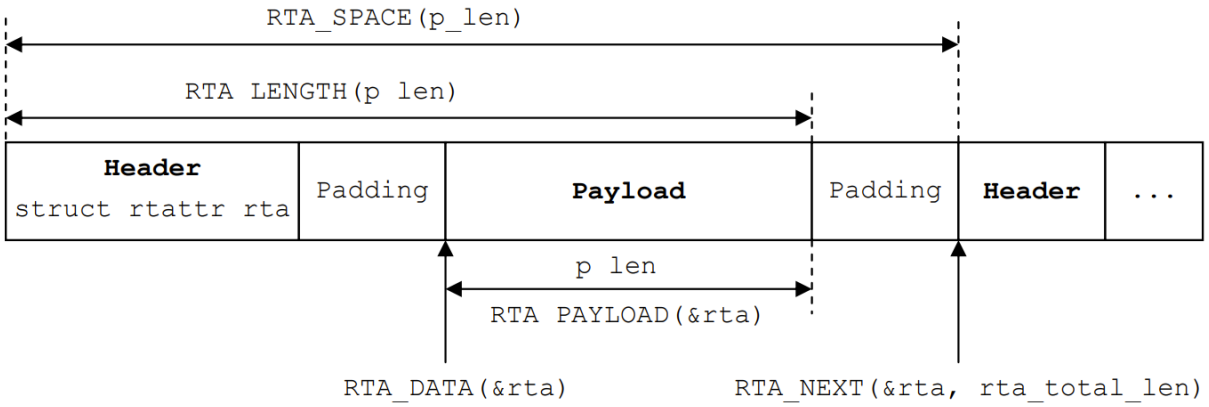
```
int NLMSG_PAYLOAD(struct nlmsghdr *nlh, int len);
```

Zastosowanie makra NLMSG_PAYLOAD() przedstawia rys. 7.



Rys.7. Zastosowanie makra NLMSG_PAYLOAD()[7].

Podobne makra zostały zdefiniowane w pliku nagłówkowym <linux/rtnetlink.h> do celów manipulacji atrybutami rodziny NETLINK_ROUTE.



Rys.8. Makra atrybutów NETLINK_ROUTE[8].

- RTA_ALIGN() zaokrągla wartość określoną przez parametr len do wielokrotności RTA_ALIGNTO (stała zdefiniowana w <linux/rtnetlink.h>).

```
int RTA_ALIGN(size_t len);
```

- RTA_LENGTH() przyjmuje jako argument rozmiar danych atrybutu (payload) i zwraca rozmiar atrybutu bez uwzględnienia paddingu za polem danych. Makro jest używane do ustawienia wartości pola rta_len struktury rtattr.

```
unsigned int RTA_LENGTH(unsigned int length);
```

- RTA_SPACE() przyjmuje jako argument rozmiar pola danych i zwraca wartość określającą rozmiar atrybutu. Rozmiar atrybutu uwzględnia nagłówek, dane atrybutu i wszystkie opcjonalne dopełnienia.

```
unsigned int RTA_SPACE(unsigned int length);
```

- RTA_DATA() zwraca wskaźnik do pola danych atrybutu. Parametrem makra jest wskaźnik na strukturę rtattr reprezentującą nagłówek atrybutu.

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

- RTA_NEXT() jest używane w przypadku odbierania komunikatów Netlink. Jeżeli w buforze znajduje się komunikat z kilkoma atrybutami, to makro RTA_NEXT() zwraca wskaźnik do początku następnego atrybutu (względem atrybutu wskazywanego przez parametr rta). Parametr rtabuflen określa rozmiar atrybutów danego komunikatu (łącznie z dopełnieniami). Początkowo jest to liczba bajtów zwrócona przez makro NLMSG_PAYLOAD() – rys. 7. Makro RTA_NEXT() zmniejsza wartość rtabuflen o długość atrybutu wskazywanego przez rta, a dokładnie o RTA_ALIGN(rta->rta_len).

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

- RTA_OK() zwraca 1 w przypadku, gdy atrybut wskazywany przez parametr rta nie został przycięty i może być przetwarzany. W przeciwnym wypadku zwracana jest wartość 0. Parametr rtabuflen ma znaczenie takie, jak w przypadku makra RTA_NEXT(), ale nie jest w żaden sposób modyfikowany przez RTA_OK().

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

- RTA_PAYLOAD() przyjmuje wskaźnik na strukturę reprezentującą nagłówek atrybutu – rtattr. Makro zwraca rozmiar pola danych atrybutu.

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

1.4. Cel laboratorium

Celem laboratorium jest poznanie elementarnych technik przesyłania komunikatów Netlink między procesami użytkownika, a jądrem systemu Linux. Techniki te są bazą w tworzeniu aplikacji zarządzających warstwą forwardingu architektury sieciowej.

Podczas realizacji tego laboratorium zapoznasz się z:

- budową komunikatu oraz makrami Netlink,
- sposobem transmisji danych za pomocą protokołu Netlink,
- możliwościami rodziny NETLINK_ROUTE,
- odbieraniem powiadomień od jądra systemu Linux.

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Zadanie 1. Komunikacja z wykorzystaniem funkcji recvmsg() oraz sendmsg()

Zadanie polega na analizie kodu przykładowego programu. Program wypisuje na standardowe wyjście parametry konfiguracyjne wszystkich interfejsów sieciowych (administracyjnie włączonych). Proszę zwrócić szczególną uwagę na kolejne etapy, w których konstruowany i wysyłany jest komunikat Netlink:

- utworzenie gniazda typu SOCK_RAW dla domeny komunikacyjnej PF_NETLINK,
- powiązanie gniazda z identyfikatorem procesu za pomocą funkcji bind(),
- zaalokowanie pamięci dla komunikatu; dokładny rozmiar bufora jest określony za pomocą makra NLMSG_SPACE(),
- wypełnienie pól nagłówka Netlink, m.in.:
    - typ komunikatu – RTM_GETADDR,
    - flagi – NLM_F_REQUEST (komunikat jest zapytaniem) i NLM_F_ROOT (żądanie zwrócenia informacji na temat wszystkich interfejsów),
- wypełnienie pól struktury msghdr wykorzystywanej przez funkcję sendmsg() i wysłanie komunikatu; adres odbiorcy (jądra systemu) jest określony przez pole msg_name.

Odpowiedź może zawierać kilka komunikatów Netlink (multipart message). Ostatni komunikat będący odpowiedzią na wysłane zapytanie posiada nagłówek, w którym pole type (rys. 3) ma wartość NLMSG_DONE. Komunikat o typie NLMSG_DONE nie przenosi informacji - pełni wyłącznie funkcję sygnalizacyjną. Dostęp do nagłówków i atrybutów odebranych wiadomości odbywa się za pomocą makr. Program porównuje numer

sekwencyjny odebranych komunikatów z numerem sekwencyjnym wysłanego zapytania. Jeżeli występuje niezgodność, wypisywany jest odpowiedni komunikat.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami (rozpakować je w razie konieczności).
2. Skompilować program źródłowy do postaci binarnej:

   ```
   $ gcc -o list list.c
   ```

3. Uruchomić program podając jako argument wywołania wersję protokołu IP:

   ```
   $ ./list <4 lub 6>
   ```

4. Proszę porównać informacje zwrócone przez program z informacjami jakie można uzyskać za pomocą polecenia ip addr.

2.2. Zadanie 2. Przesyłanie atrybutów

Zadanie polega na analizie kodu przykładowego programu, który umożliwia konfigurację adresów IPv4 interfejsów sieciowych. Program przyjmuje jako argumenty wywołania:

- nazwę interfejsu (etykietę), dla którego przeprowadzana jest konfiguracja,
- opcję określającą typ operacji – „add” dla dodania nowego adresu lub „del” dla usunięcia adresu.
- adres IPv4 oraz długość prefiksu (maski sieciowej).

Komunikat Netlink jest konstruowany w sposób podobny do opisanego w poprzednim zadaniu, z tą różnicą, że zawiera dodatkowe atrybuty.

Indeks interfejsu oraz długość prefiksu adresu IP są przenoszone w nagłówku rodziny (struktura ifaddrmsg). Pozostałe parametry (etykieta interfejsu, adres IP i adres rozgłoszeniowy) są wysyłane w formie atrybutów. Adres rozgłoszeniowy jest określany na podstawie argumentów wywołania programu - adresu IP oraz długości prefiksu.

Komunikat Netlink jest wysyłany na adres zdefiniowany w wywołaniu funkcji sendto().

Uwaga: Modyfikacja parametrów konfiguracyjnych wymaga uprawnień roota.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami.
2. Skompilować program źródłowy do postaci binarnej:

   ```
   $ gcc ipaddr.c libnetlink.c –o ipaddr
   ```

3. Uruchomić program podając nazwę lub alias interfejsu, adres IPv4 oraz długość prefiksu:

   ```
   $ ./ipaddr <nazwa interfejsu> add <adres IPv4> <długość prefiksu>
   ```

Interfejs „bazowy” dla konfigurowanego aliasu musi istnieć w systemie. Przykładowo, interfejsem „bazowym” dla aliasu eth0:switch jest eth0.

4. Zweryfikować poprawność działania programu za pomocą polecenia ip addr.
5. Usunąć adres skonfigurowany w punkcie 3:

```
$ ./ipaddr <nazwa aliasu> del <adres IPv4> <długość prefiksu>
```

6. Proszę ponownie zweryfikować konfigurację za pomocą polecenia ip addr.
7. Czy utworzenie gniazda surowego dla domeny PF_NETLINK wymaga uprawnień roota?

2.3. Zadanie 3. Transmisja z potwierdzeniami

Celem zadania jest modyfikacja programu umożliwiającego konfigurację adresów IPv4 interfejsów sieciowych (ipaddr.c zapisać pod ipaddrmod.c i wprowadzić w nim odpowiednie zmiany). Program powinien wysłać komunikat Netlink z ustawioną flagą NLM_F_ACK i za pomocą funkcji recvfrom() oczekiwać na odpowiedź. Flaga NLM_F_ACK wymusza wysyłanie potwierdzeń przez jądro systemu Linux.

Program ma przeanalizować odpowiedź i wypisać na standardowe wyjście, czy jest to potwierdzenie dotyczące wysłanego komunikatu, czy też informacja o błędzie.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:

```
$ gcc ipaddrmod.c libnetlink.c -o ipaddrmod
```

2. Uruchomić program podając nazwę lub alias interfejsu, adres IPv4 oraz długość prefiksu:

```
$ ./ipaddrmod <nazwa> add <adres IPv4> <długość prefiksu>
```

3. Zweryfikować poprawność działania programu za pomocą polecenia ip addr.
4. Proszę ponownie uruchomić program z tymi samymi argumentami. Jaka będzie odpowiedź na wysłany komunikat?
5. Usunąć adres skonfigurowany w punkcie 2:

```
$ ./ipaddrmod <nazwa> del <adres IPv4> <długość prefiksu>
```

6. Proszę zweryfikować konfigurację za pomocą polecenia ip addr.
7. Jaką odpowiedź wyświetli program podczas próby usunięcia nieistniejącego adresu, albo próby dodania poprawnego adresu bez uprawnień roota?

2.4. Zadanie 4. Powiadomienie o wystąpieniu zdarzenia

Zadanie polega na zaimplementowaniu programu umożliwiającego otrzymywanie powiadomień (multicast) od jądra systemu Linux. Program ma wypisywać na standardowe wyjście informacje związane z następującymi zdarzeniami:

- dodanie lub usunięcie adresu IPv4,
- zmiana stanu administracyjnego interfejsu sieciowego,
- zmiana adresu MAC oraz MTU interfejsu.

Program powinien wypisywać nazwę interfejsu, którego dotyczy dane powiadomienie oraz parametr związany bezpośrednio z wystąpieniem zdarzenia (np.: adres IP w przypadku dodania nowego adresu).

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:

   ```
   $ gcc event.c -o event
   ```

2. Uruchomić program:

   ```
   $ ./event
   ```

3. Za pomocą programu ifconfig lub ip wywołać odpowiednie zdarzenie.

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. „Protokół Netlink” polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.

Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.