Instytut Teleinformatyki

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

programowanie usług sieciowych

„Netfilter”

laboratorium: 07
system operacyjny: Linux

Kraków, 2009

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące **projektu Netfilter** oraz programowania aplikacji umożliwiających filtrowanie pakietów sieciowych i manipulację regułami *iptables*. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji bazujących na bibliotekach *Netfilter* (**libnetfilter_queue** oraz **libiptc**). *Netfilter* jest oficjalną nazwą projektu, który dostarcza oprogramowanie pozwalające na:

- filtrowanie pakietów (bezstanowe, stanowe oraz w warstwie aplikacji),
- manipulację pakietami (ang. *packet mangling*),
- logowanie pakietów,
- translację adresów (NAT),
- śledzenie połączeń (ang. *connection tracking*),
- synchronizację stanów połączeń między firewallami (tworzenie klastrów firewalli).

W skład projektu *Netfilter* wchodzi oprogramowanie działające zarówno w przestrzeni jądra systemu Linux (np. frameworki *IP Sets* i *Netfilter*), jak i w przestrzeni użytkownika:

- programy *iptables*, *ipset*, *conntrack-tools*, *ulogd,*
- biblioteki *libnetfilter_log*, *libnetfilter_queue* i *libnetfilter_conntrack*.

1.2. Zagadnienia do przygotowania

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

- obsługi programu *iptables* [1 - 4]
- zastosowania programów *ipset* oraz *conntrackd* [5, 6]
- etapów przetwarzania pakietu w systemie Linux [1]
- biblioteki *libnetfilter_queue* [7]
- biblioteki *libiptc* (źródła *iptables*) [1, 8]
- budowy nagłówka komunikatu ICMP Echo

Literatura:

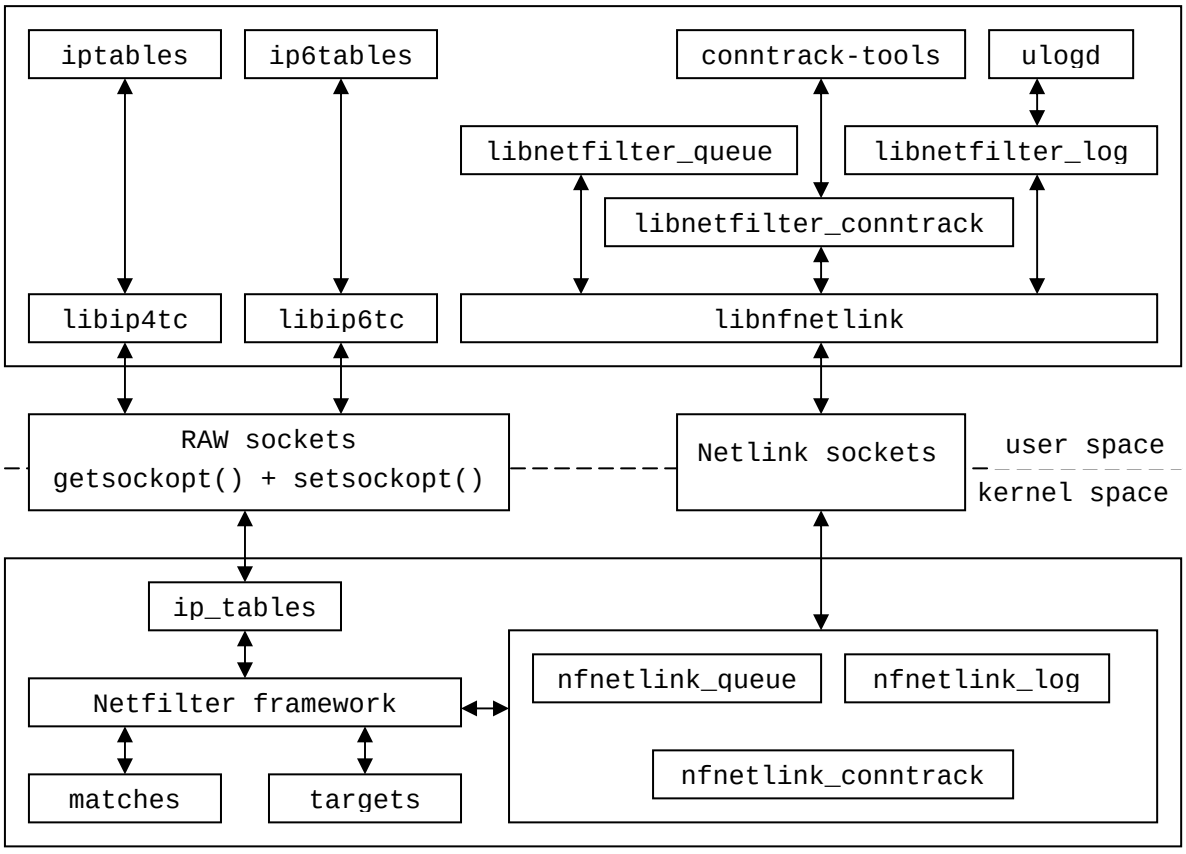
[1] Netfilter, http://www.netfilter.org/documentation/
[2] MAN (8), *„iptables”*
[3] Michael Rash, *„Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort*”, No Starch Press
[4] Lucian Gheorghe, *„Designing and Implementing Linux Firewalls and QoS using netfilter, iproute2, NAT and L7-filter”*, Packt Publishing
[5] IP Set, http://ipset.netfilter.org/
[6] conntrack-tools, http://conntrack-tools.netfilter.org/
[7] Dokumentacja w plikach źródłowych *libnetlink_queue* (*Doxygen*)
[8] *„Querying libiptc HOWTO”,* http://www.opalsoft.net/qos/libiptc/qlibiptc.html

1.3. Opis laboratorium

1.3.1. Netfilter

Netfilter jest projektem, w skład którego wchodzi oprogramowanie dające możliwość przetwarzania i filtrowania pakietów. Oprogramowanie składa się z:

- narzędzi i bibliotek działających w przestrzeni użytkownika,
- modułów jądra.

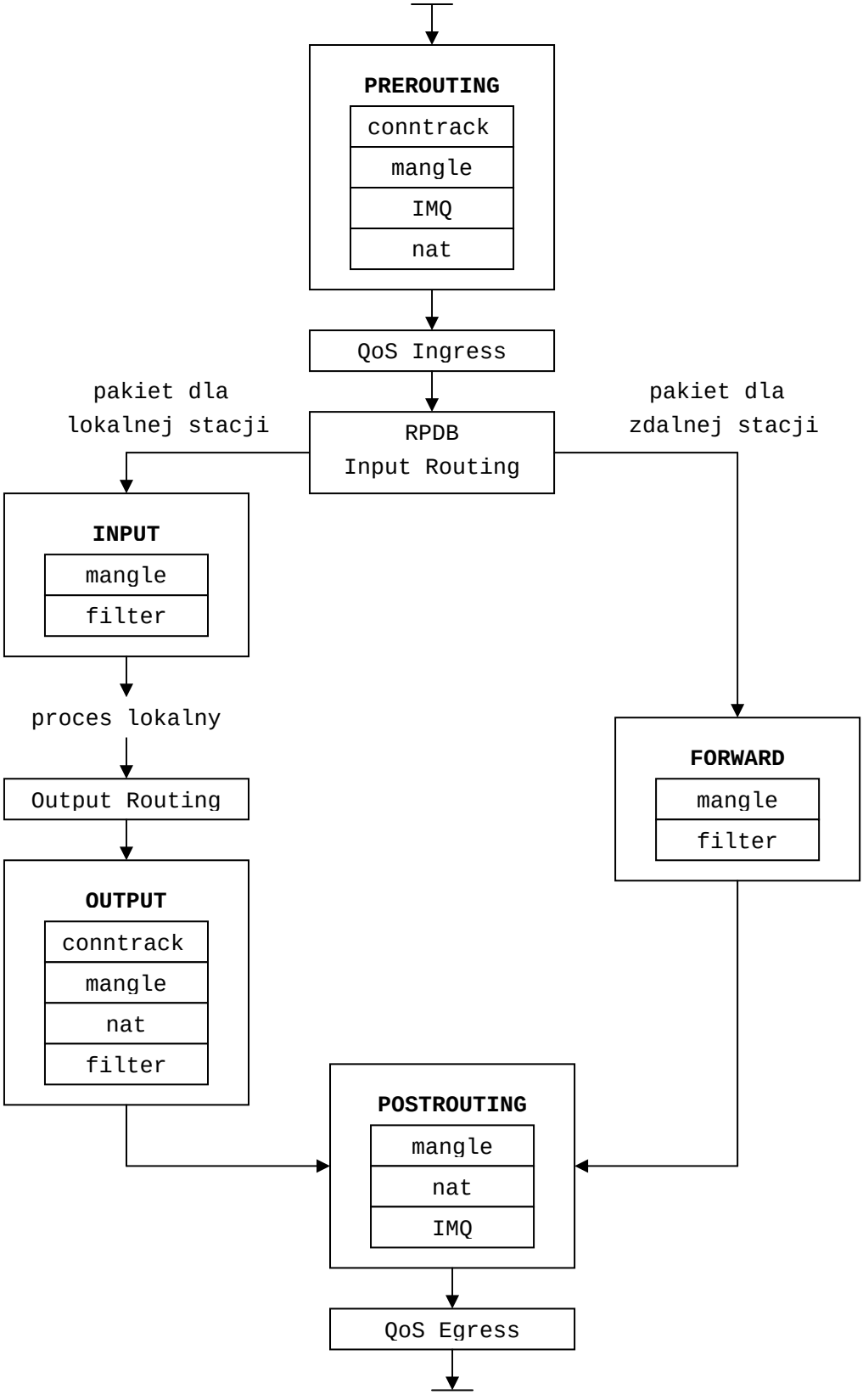


Rys.1. Główne komponenty projektu *Netfilter*.

Podstawowe komponenty *Netfilter*, to:

- **libnfnetlink**
  Niskopoziomowa biblioteka wykorzystywana do komunikacji między przestrzenią użytkownika, a modułami jądra systemu Linux (za pomocą gniazd *Netlink*). Biblioteka nie jest używana przez programy użytkownika, a przez pozostałe komponenty *Netfilter* i wymaga jądra 2.6.14 lub nowszego.
- **libnetfilter_conntrack**
  Biblioteka w przestrzeni użytkownika, która umożliwia zarządzanie wpisami w tablicy połączeń (ang. *connection tracking table*). Biblioteka komunikuje się z podsystemem *nfnetlink_conntrack* w jądrze systemu za pomocą *libnfnetlink*.
- **conntrack-tools**
  Pakiet narzędzi dla administratorów, które bazują na bibliotece *libnetfilter_conntrack*. Program *conntrack* umożliwia zarządzanie tablicą połączeń i zastępuje interfejs */proc/net/ip_conntrack*. Program *conntrackd* umożliwia z kolei synchronizację informacji na temat połączeń pomiędzy firewallami śledzącymi stan połączeń, tj. pracującymi w trybie SPI (ang. *stateful packet inspection*).
- **libnetfilter_log**
  Biblioteka w przestrzeni użytkownika używana do logowania pakietów. Kopie pakietów spełniających określone kryteria są przekazywane za pomocą gniazd *Netlink* do jednego lub więcej procesów użytkownika (transmisja typu multicast). Komunikacja z jądrem systemu Linux – a dokładnie z podsystemem *nfnetlink_log* - odbywa się za pomocą biblioteki *libnfnetlink*. Biblioteka zastępuje mechanizm logowania oparty na *syslog*/*dmesg*.
- **ulogd**
  Demon umożliwiający logowanie pakietów spełniających określone kryteria do pliku lub bazy danych. Program bazuje na bibliotece *libnetfilter_log*.
- **libnetfilter_queue**
  Biblioteka w przestrzeni użytkownika wykorzystywana do odbierania pakietów zakolejkowanych przez jądro systemu (podsystem *nfnetlink_queue*) oraz do określenia akcji związanej z pakietem (porzucenie pakietu lub zezwolenie na dalsze przetwarzanie przez jądro). Zastępuje starszą bibliotekę *libipq*.
- **iptables**
  Program umożliwiający zarządzanie polityką (strategią) filtrowania i przetwarzania pakietów przez jądro systemu. Program *iptables* bazuje na frameworku *Netfilter*.
  Pakiety przetwarzane przez system Linux przechodzą w jądrze systemu kolejne etapy nazywane ścieżką forwardingu (ang. *forwarding path*, rys. 2). Framework *Netfilter*, umożliwia *iptables* rejestrowanie na różnych etapach funkcji, które są odpowiedzialne za wykonywanie operacji na pakietach. Operacje wykonywane są na podstawie określonych kryteriów, które spełnia (lub nie) pakiet. W terminologii reguł *iptables* operacje na pakietach nazywane są celami (ang. *targets*), a kryteria dopasowań pakietów określane są terminem *matches*. Cele i dopasowania są w istocie funkcjami implementowanymi przez moduły jądra, reje-

strowanymi na różnych etapach przetwarzania pakietów (punktach zaczepienia – ang. *hooks*). Podczas rejestracji funkcji określa się ich priorytet, który determinuje kolejność wywołania poszczególnych funkcji na danym etapie.



Rys.2. Ścieżka forwardingu w systemie Linux.

Rys.2. operuje na terminologii zaczerpniętej z programu *iptables*. Przedstawione są na nim tablice (np.: *filter*, *nat*, *mangle*) oraz łańcuchy (ang. *chains*, np.: INPUT, FORWARD). Łańcuchy *iptables* odpowiadają punktom zaczepienia w jądrze systemu. Przykładowo, dla protokołu IPv4 łańcuch FORWARD odpowiada punktowi zaczepienia o nazwie `NF_IP_FORWARD`. Łańcuchy wykorzystywane są w celu grupowania reguł *iptables* (reguła = dopasowania + cel).

Tablice wydzielają pewną funkcjonalność – grupują łańcuchy o podobnym zastosowaniu. Przykładowo, proces translacji adresów (NAT) może odbywać się w punktach zaczepienia `NF_IP_PRE_ROUTING`, `NF_IP_POST_ROUTING`, `NF_IP_LOCAL_OUT` (łańcuchy PREROUTING, POSTROUTING oraz OUTPUT). Z uwagi na fakt, że reguły w tych łańcuchach mają podobne zastosowanie (translacja adresów), wymienione łańcuchy znajdują się w tablicy o nazwie *nat*.

Mechanizmy QoS (ang. *Quality of Service*), PR (ang. *Policy Routing*) oraz IMQ (ang. *Intermediate Queueing Device*) nie stanowią tematu laboratorium. Informacje na ich temat można znaleźć w dokumentacji pakietu *iproute2* oraz w dokumentacji LARTC (ang. *Linux Advanced Routing and Traffic Control*).

- **libiptc**

  Biblioteka *libiptc* jest wykorzystywana przez programy *iptables* oraz *ip6tables* do komunikowania się z jądrem systemu Linux w celu określenia strategii przetwarzania pakietów (na podstawie reguł *iptables*). Mechanizmem komunikacji stosowanym przez *libiptc* są gniazda surowe oraz funkcje `getsockopt()` i `setsockopt()`. Programista korzystający z biblioteki nie musi być świadomy tego faktu. W obecnej formie zarówno biblioteka *libiptc*, programy *iptables*, jak i część modułów jądra są zależne od protokołu warstwy sieciowej modelu ISO/OSI (niepotrzebna nadmiarowość kodu). W przyszłych wersjach planuje się zastąpienie mechanizmu gniazd surowych przez gniazda *Netlink* i uniezależnienie biblioteki od konkretnego protokołu. Z tego względu biblioteka *libiptc* nie stanowi publicznego API, a dostęp do niej można uzyskać na podstawie plików źródłowych programu *iptables*.

1.3.2. Interfejs programistyczny biblioteki libnetfilter_queue

API biblioteki *libnetfilter_queue* składa się z trzech grup funkcji odpowiedzialnych za:

- inicjalizację biblioteki,
- zarządzanie kolejką,
- przetwarzanie pakietów i meta-danych.

Funkcje inicjalizujące

Funkcja **`nfq_open()`** tworzy połączenie *Netlink* z podsystemem *nfnetlink_queue* w jądrze systemu Linux i zwraca uchwyt połączenia lub `NULL` w przypadku błędu.


```
#include <libnetfilter_queue/libnetfilter_queue.h>

struct nfq_handle* nfq_open(void);
```

W celu zamknięcia połączenia i zwolnienia związanych z nim zasobów należy wywołać funkcję **`nfq_close()`**. Funkcja `nfq_close()` przyjmuje jako parametr wskaźnik na strukturę `nfq_handle` (uchwyt połączenia) i w przypadku powodzenia zwraca wartość zero.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int  nfq_close(struct nfq_handle* h);
```

Wywołując funkcję **`nfq_bind_pf()`**można powiązać uchwytu połączenia z określoną rodziną protokołów (np.: `PF_INET` lub `PF_INET6`). W ten sposób, dla danego połączenia przetwarzane będą pakiety związane tylko z wyspecyfikowaną rodziną protokołów.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int  nfq_bind_pf(struct nfq_handle* h, u_int16_t pf);
```

Efekt przeciwny do opisanego powyżej można uzyskać za pomocą funkcji **`nfq_unbind_pf()`**.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int  nfq_unbind_pf(struct nfq_handle* h, u_int16_t pf);
```

Funkcje `nfq_bind_pf()` oraz `nfq_bind_pf()` zwracają wartość mniejszą od zera w przypadku wystąpienia błędu.

Funkcje zarządzające kolejką

Po poprawnym utworzeniu uchwytu połączenia dla wybranej rodziny protokołów można przystąpić do powiązania uchwytu połączenia (pośrednio gniazda *Netlink*) z kolejką o określonym numerze.
Każda kolejka identyfikowana jest przez 16-bitową liczbę. Numer kolejki, do której mają trafiać pakiety określa się w programie *iptables* za pomocą opcji `--queue-num` celu `NFQUEUE`. Przykładowo reguła:


```
$ iptables –A INPUT --src 192.168.2.1 –j NFQUEUE --queue-num 5
```

spowoduje, że pakiety IPv4 o adresie źródłowym 192.168.2.1 będą przesyłane do kolejki o numerze 5.

Powiązanie gniazda z kolejką odbywa się za pomocą funkcji **`nfq_create_queue()`**. Funkcja `nfq_create_queue()` zwraca uchwyt kolejki (ang. *queue handle*), tj. wskaźnik do `nfq_q_handle` lub `NULL` w przypadku błędu.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

struct nfq_q_handle *nfq_create_queue(struct nfq_handle* h,
                                      u_int16_t          num,
                                      nfq_callback*      cb,
                                      void*              data);
```

Parametr	Opis
`h`	uchwyt połączenia uzyskany za pomocą `nfq_open()`
`num`	numer kolejki, z którą powiązane zostanie gniazdo
`cb`	funkcja *callback* wywoływana dla zakolejkowanego pakietu
`data`	dane użytkownika przekazywane do funkcji *callback*

Typ `nfq_callback` jest zdefiniowany jako:

```
#include <libnetfilter_queue/libnetfilter_queue.h>

typedef int nfq_callback(struct nfq_q_handle* gh,
                         struct nfgenmsg*     nfmsg,
                         struct nfq_data*     nfad,
                         void*                data);
```

Parametr	Opis
`gh`	uchwyt kolejki uzyskany za pomocą `nfq_create_queue()`
`nfmsg`	struktura wiadomości zawierająca pakiet
`nfad`	uchwyt danych (używany przez funkcje `nfq_get_*()`)
`data`	dane użytkownika przekazane za pomocą parametru `data` funkcji `nfq_create_queue()`

Programista jest odpowiedzialny za zdefiniowanie funkcji *callback* zgodnej z definicją typu przedstawioną powyżej, ale nie za jej wywołanie. Zastosowanie funkcji *callback* zostanie wyjaśnione przy okazji omawiania `nfq_handle_packet()`.

Zwolnienie uchwytu kolejki odbywa się za pomocą funkcji **`nfq_destroy_queue()`**. Funkcja `nfq_destroy_queue()` przyjmuje uchwyt kolejki (wskaźnik na strukturę `nfq_q_handle`) i w przypadku powodzenia zwraca wartość zero.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

typedef int nfq_destroy_queue(struct nfq_q_handle* gh);
```

Kolejnym krokiem po utworzeniu uchwytu kolejki jest określenie w jakim trybie dane będą kopiowane z kolejki do przestrzeni użytkownika. Funkcją odpowiedzialną za określenie trybu kopiowania jest **`nfq_set_mode()`**.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

nt nfq_set_mode(struct nfq_q_handle* qh,
                u_int8_t             mode,
                unsigned int         len);
```

Parametr	Opis
`qh`	uchwyt kolejki uzyskany za pomocą `nfq_create_queue()`
`mode`	tryb kopiowania z kolejki
`range`	dopuszczalny rozmiar pakietu

Parametr `mode` może przyjmować następujące wartości:

Wartość parametru	Opis
`NFQNL_COPY_NONE`	nie będą kopiowane żadne dane
`NFQNL_COPY_META`	kopiowane będą tylko meta-dane na temat pakietu
`NFQNL_COPY_PACKET`	kopiowany będzie cały pakiet (z meta-danymi)

Meta-dane to informacje, które nie są bezpośrednio przenoszone w pakiecie, ale są z nim związane, np.: numer interfejsu, za pomocą którego pakiet został otrzymany lub znacznik czasowy (ang. *timestamp*) otrzymania pakietu.

Rozmiar kolejki, z której pobierane są dane do procesu użytkownika można zdefiniować za pomocą funkcji **`nfq_set_queue_maxlen()`**.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int nfq_set_queue_maxlen(struct nfq_q_handle* qh, u_int32_t qlen);
```

Często rozmiar bufora gniazda *Netlink* dla danych odbieranych stanowi większe ograniczenie od rozmiaru kolejki w jądrze. Rozmiarem bufora gniazda można zarządzać za pomocą funkcji `getsockopt()` oraz `setsockopt()` – opcja `SO_RCVBUF`. System Linux wprowadza ograniczenie na maksymalny rozmiar bufora gniazda. Informacje o domyślnym i maksymalnym rozmiarze bufora dla danych odbieranych przez gniazda można uzyskać za pomocą systemu plików *proc*:

```
$ cat /proc/sys/net/core/rmem_default
$ cat /proc/sys/net/core/rmem_max
```

Podczas wywołania funkcji `setsockopt()` z opcją `SO_RCVBUF` proszę pamiętać, że jądro systemu Linux podwaja wartość podaną w argumencie wywołania funkcji.

Po ustaleniu trybu w jakim odbierane będą dane z kolejki (funkcja `nfq_set_mode()`) można przystąpić do odbierania i przetwarzania pakietu. W tym celu należy wywołać funkcję **`nfq_fd()`**, która dla podanego w argumencie wywołania uchwytu połączenia zwraca deskryptor gniazda *Netlink*.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int nfq_fd(struct nfq_handle* h);
```

Pakiety mogą zostać odebrane za pomocą funkcji `recv()`. Ponieważ *Netlink* jest protokołem datagramowym, cały pakiet jest odbierany za pomocą jednego wywołania funkcji `recv()`. Otrzymany pakiet powinien zostać przekazany do funkcji **`nfq_handle_packet()`**.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int nfq_handle_packet(struct nfq_handle* h, char* buf, int len);
```


Parametr	Opis
h	uchwyt połączenia uzyskany za pomocą nfq_open()
buf	bufor zawierający pakiet
range	rozmiar pakietu w buforze

Funkcja nfq_handle_packet() wywołuje funkcję *callback* przekazując jej pakiet odebrany z kolejki. W przypadku powodzenia funkcja zwraca wartość zero.

Zadaniem funkcji *callback* jest **analiza/modyfikacja** pakietu i wydanie werdyktu:

- akceptacja pakietu – NF_ACCEPT
- porzucenie pakietu – NF_DROP

Werdykt jest wydawany za pomocą funkcji **nfq_set_verdict()**.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int nfq_set_verdict(struct nfq_q_handle* qh,
                    u_int32_t            id,
                    u_int32_t            verdict,
                    u_int32_t            data_len
                    unsigned_char*       buf);
```

Parametr	Opis
qh	uchwyt kolejki uzyskany za pomocą nfq_create_queue()
id	identyfikator przypisany do pakietu przez *Netfilter*
verdict	NF_ACCEPT lub NF_DROP
data_len	liczba bajtów w buforze (jeżeli pakiet został zmodyfikowany)
buf	bufor zawierający pakiet (jeżeli pakiet został zmodyfikowany)

Identyfikator przypisany przez framework *Netfilter* jest częścią meta-danych pakietu. Dostęp do meta-danych i właściwych danych (ang. *payload*) pakietu można uzyskać za pomocą funkcji nfq_get_*().

Funkcje odpowiedzialne za przetwarzanie pakietów

Funkcje z rodziny nfq_get_*() umożliwiają dostęp do pakietu oraz do informacji związanych z pakietem.

Najważniejsze funkcje odpowiedzialne za przetwarzanie pakietów i meta-danych to:

- `nfq_get_msg_packet_hdr()`,
- `nfq_get_indev()`,
- `nfq_get_packet_hw()`,
- `nfq_get_payload()`.

Wszystkie wymienione powyżej funkcje przyjmują jako argument wskaźnik na strukturę `nfq_data` (parametr funkcji *callback*). Funkcja `nfq_get_payload()` przyjmuje dodatkowo wskaźnik do właściwych danych pakietu. Pakiet jest zgodnie z definicją RFC 1122 jednostką danych przekazywaną między warstwą sieciową, a warstwą łącza danych modelu ISO/OSI. Składa się więc z nagłówka IP i danych.

Funkcja **`nfq_get_msg_packet_hdr()`** zwraca meta-nagłówek odpowiadający przetwarzanemu pakietowi. Meta-nagłówek jest zdefiniowany w pliku `<libnetfilter_queue/linux_nfnetlink_queue.h>`.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

struct nfqnl_msg_packet_hdr {
       /* Unikalny ID pakietu: */
        u_int32_t       packet_id;
       /* Wartość pola Ethertype/Type ramki Ethernet:*/
        u_int16_t       hw_protocol;
       /* Punkt zaczepienia Netfilter: */
        u_int8_t        hook;
};
```

Pola meta-nagłówka są przechowywane w porządku sieciowym (ang. *network byte order*).

Funkcja **`nfq_get_msg_packet_hw()`** zwraca adres warstwy łącza danych dla przetwarzanego pakietu (chodzi o adres źródłowy). Struktura zwracana przez funkcję jest zdefiniowana w pliku `<libnetfilter_queue/linux_nfnetlink_queue.h>`.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

struct nfqnl_msg_packet_hw {
        /* Długość adresu (porządek sieciowy, tj. big-endian): */
        u_int16_t       hw_addrlen;
        u_int16_t       _pad;
        u_int8_t        hw_addr[8]; /* w porządku big-endian */
};
```


Funkcja **nfq_get_indev()** zwraca indeks interfejsu, przez który otrzymany został pakiet. Indeks o wartości 0 oznacza, że pakiet został wygenerowany lokalnie lub indeks interfejsu nie jest znany.

Ostatnią z omawianych funkcji biblioteki *libnetfilter_queue* jest **nfq_get_payload()**.

```
#include <libnetfilter_queue/libnetfilter_queue.h>

int nfq_get_payload(struct nfq_data *nad, char ** data);
```

Funkcja `nfq_get_payload()` przyjmuje wskaźnik na strukturę `nfq_data` oraz parametr `data`, który po wywołaniu będzie wskazywał na bufor zawierający pakiet. W przypadku powodzenia funkcja zwraca rozmiar pakietu (wartość większą lub równą 0). To, czy dane zostaną zwrócone zależy od trybu kopiowania zdefiniowanego przez funkcję `nfq_set_mode()`.

1.3.3. Biblioteka libiptc

Biblioteka *libiptc* wchodzi w skład pakietu *iptables* i nie jest dostępna niezależnie od niego. Programy *iptables* oraz *ip6tables* wykorzystują bibliotekę *libiptc* do zarządzania regułami filtrowania i przetwarzania pakietów. Bibliotekę można utworzyć kompilując pakiet *iptables* za pomocą dołączonego pliku *Makefile*. Ogólną procedurę kompilacji i instalacji *iptables* przedstawia poniższy listing.

```
$ cd /usr/local/src/iptables-x.y.z
$ ./configure
$ make
$ make install
$ ls libiptc
libip4tc.c  libip6tc.c  libiptc.a  linux_list.h
libip4tc.o  libip6tc.o  libiptc.c  linux_stddef.h
```

W wyniku wywołania polecenia *make* w katalogu *libiptc* tworzona jest m.in. biblioteka statyczna *libiptc.a*. Aby wykorzystać możliwości biblioteki należy wyodrębnić wymagane przez nią pliki nagłówkowe. W tym celu należy skopiować do oddzielnego folderu następujące pliki:

```
/usr/local/src/iptables-x.y.z/include/iptables.h
/usr/local/src/iptables-x.y.z/include/xtables.h
/usr/local/src/iptables-x.y.z/include/libiptc/ipt_kernel_headers.h
/usr/local/src/iptables-x.y.z/include/libiptc/libip6tc.h
```



```
/usr/local/src/iptables-x.y.z/include/libiptc/libiptc.h
```

Część praktyczna nie wymaga przeprowadzenia opisanych powyżej operacji. Program *iptables* powinien być zainstalowany (można to zweryfikować wydając polecenie `iptables -V`). Pliki nagłówkowe biblioteki znajdują się w katalogu */usr/local/include*, a biblioteka statyczna w katalogu */usr/local/lib*. Podane ścieżki są automatycznie przeszukiwane przez kompilator *gcc* podczas kompilacji i linkowania programów.

Funkcje libiptc wykorzystywane w części praktycznej

Funkcja **`iptc_init()`** jest odpowiedzialna za wykonanie zrzutu (ang. *snapshot*) wszystkich reguł dla podanej tabeli. Funkcja zwraca uchwyt `iptc_handle_t` lub `NULL` w przypadku błędu. Uchwyt jest wykorzystywany przez pozostałe funkcje do zarządzania łańcuchami, regułami i domyślną polityką w obrębie podanej tablicy.

```
#include <libiptc/libiptc.h>

iptc_handle_t iptc_init(const char *tablename);
```

W przypadku wystąpienia błędu funkcje *libiptc* ustawiają wartość zmiennej `errno`. Za pomocą funkcji **`iptc_strerror()`** można określić błąd odpowiadający podanej wartości `errno`.

```
#include <libiptc/libiptc.h>

const char *iptc_strerror(int err);
```

Po zakończeniu pracy z *libiptc* należy zwolnić uchwyt do tablicy za pomocą funkcji **`iptc_free()`.**

```
#include <libiptc/libiptc.h>

const char *iptc_free(iptc_handle_t *h);
```

Iteracja przez łańcuchy tabeli odbywa się za pomocą funkcji `iptc_first_chain()` oraz `iptc_next_chain()`. Funkcja **`iptc_first_chain()`** przyjmuje wskaźnik do uchwytu tablicy (uzyskanego za pomocą `iptc_init()`) i zwraca nazwę pierwszego łańcucha w tablicy. Jeżeli tablica nie posiada łańcuchów, to funkcja zwraca wartość `NULL`. Z kolei funkcja **`iptc_next_chain()`** zwraca nazwę kolejnego łańcucha lub wartość `NULL`, jeżeli tablica nie posiada więcej łańcuchów.


```
#include <libiptc/libiptc.h>

const char *iptc_first_chain(iptc_handle_t *handle);
const char *iptc_next_chain(iptc_handle_t *handle);
```

Poniżej przedstawiony jest fragment kodu wykorzystujący omówione funkcje.

```
int main(int argc, char **argv) {

  iptc_handle_t  h; /* Uchwyt */
  const char     *ch = NULL; /* Nazwa łańcucha */
  const char     *table_name = "filter"; /* Nazwa tablicy */

  h = iptc_init(table_name); /* Snapshot dla podanej tablicy */
  if (!h) {
    fprintf(stderr, "iptc_init(): %s\n", iptc_strerror(errno));
    exit(EXIT_FAILURE);
  }

  /* Iteracja przez łańcuchy tablicy: */
  for (ch = iptc_first_chain(&h); ch; ch = iptc_next_chain(&h)) {
    fprintf(stout, "Chain %s\n", ch);
  }

  iptc_free(&h); /* Zwolnienie uchwytu */
  exit(EXIT_SUCCESS);
}
```

Podczas wykonywania części praktycznej przydatne mogą okazać się funkcje **iptc_is_chain()** oraz **iptc_builtin()**. Obie funkcje przyjmują nazwę łańcucha i uchwyt do tablicy. Pierwsza sprawdza, czy podany łańcuch istnieje w tablicy, natomiast druga, czy łańcuch jest łańcuchem wbudowanym (ang. *butli-in*). Funkcje zwracają wartość 1, jeżeli odpowiedź to prawda i wartość 0 w przeciwnym przypadku.

```
#include <libiptc/libiptc.h>

int iptc_is_chain(const char *chain, const iptc_handle_t handle);
int iptc_builtin(const char *chain, const iptc_handle_t handle);
```

Biblioteka *libiptc* udostępnia szereg funkcji umożliwiających modyfikację reguł. Zmiany nie są wprowadzane natychmiastowo, a dopiero po wywołaniu `iptc_commit()`. Funkcja **`iptc_commit()`** przyjmuje wskaźnik do uchwytu modyfikowanej tablicy i w przy-

padku powodzenia zwraca wartość 1. Uwaga – funkcja `iptc_commit()` zwalnia pamięć zaalokowaną na rzecz uchwytu i ustawia uchwyt na wartość `NULL`. Po wywołaniu funkcji uchwyt nie nadaje się do użycia.

```
#include <libiptc/libiptc.h>

int iptc_commit(iptc_handle_t *handle);
```

Następujące funkcje zostaną wykorzystane w części praktycznej i wymagają zatwierdzenia zmian przez wywołanie `iptc_commit()`:

- `iptc_create_chain()`,
- `iptc_delete_num_entry()`,
- `iptc_set_policy()`.

Funkcja **`iptc_create_chain()`** dodaje do tablicy nowy łańcuch. Nazwę łańcucha określa parametr `chain` (można go traktować jak wskaźnik na `char`).

```
#include <libiptc/libiptc.h>

typedef char ipt_chainlabel[32];

int iptc_create_chain(const ipt_chainlabel chain,
                      iptc_handle_t *handle);
```

Funkcja **`iptc_delete_num_entry()`** usuwa z łańcucha `chain` regułę o numerze określonym przez parametr `rulenum`.

```
#include <libiptc/libiptc.h>

int iptc_delete_num_entry(const ipt_chainlabel chain,
                          unsigned int rulenum,
                          iptc_handle_t *handle);
```

Ostatnia z wymienionych funkcji, **`iptc_set_policy()`** definiuje domyślną politykę łańcucha. Proszę pamiętać, że domyślną politykę (ACCEPT lub DROP) może posiadać tylko łańcuch wbudowany.

```
#include <libiptc/libiptc.h>

int iptc_set_policy(const ipt_chainlabel chain,
```



```
                    const ipt_chainlabel policy,
                    struct ipt_counters *counters,
                    iptc_handle_t *handle);
```

Parametr	Opis
`chain`	nazwa łańcucha
`policy`	domyślna polityka (string *”ACCEPT”* lub *”DROP”*)
`counters`	licznik pakietów oraz bajtów łańcucha (można przekazać `NULL`)
`handle`	wskaźnik na uchwyt tablicy

1.4. Cel laboratorium

Celem laboratorium jest zapoznanie się z możliwościami filtrowania i przetwarzania pakietów w systemie Linux. Na podstawie bibliotek projektu *Netfilter* przedstawione zostaną techniki zarządzania warstwą forwardingu pakietów z poziomu procesów użytkownika. Podczas realizacji tego laboratorium zapoznasz się z:

- podstawowymi opcjami programu *iptables*,
- możliwościami i zastosowaniem biblioteki *libnetlink_queue*,
- podstawowymi funkcjami biblioteki *libiptc*.

2. Przebieg laboratorium

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

2.1. Zadanie 1. Analiza pakietów w procesie użytkownika

Zadanie polega na analizie kodu przykładowego programu. Program tworzy gniazdo *Netlink* (funkcja `nfq_open()` ukrywa ten proces przed programistą) i wiąże gniazdo z kolejką o numerze 5. Zakolejkowane w jądrze pakiety są wysyłane do gniazda w procesie użytkownika. Dla każdego odebranego pakietu wypisywane są następujące informacje:

- identyfikator przypisany przez *Netfilter*,
- identyfikator protokołu przenoszony w polu *EtherType/Type* ramki Ethernet,
- nazwa punktu zaczepienia frameworku *Netfilter* w konwencji zgodnej z *iptables*,
- adres MAC nadawcy pakietu
- interfejs wejściowy i wyjściowy (jeżeli są znane); w danym punkcie zaczepienia informacja na temat interfejsów może nie być znana, np.: w punkcie INPUT nie jest znany interfejs wyjściowy, ponieważ pakiet jest przeznaczony do lokalnego procesu,
- rozmiar pakietu (nagłówka IP oraz danych).

Uwagi:

- Uruchomienie programu i manipulacja regułami *iptables* wymaga uprawnień *roota*.
- Do wykonania zadania potrzebne są 2 stacje sieciowe.

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 queue –lnetfilter_queue queue.c`
3. Uruchomić program (nie przyjmuje żadnych argumentów):
   `$ ./queue`
4. Na stacji, na której uruchomiono program *queue* proszę za pomocą programu *iptables* dodać następującą regułę:


```
$ iptables –I INPUT –p ICMP –src <adres IPv4 zdalnej stacji> \
-j NFQUEUE –queue-num 5
```

Reguła jest odpowiedzialna za przesyłanie komunikatów ICMP pochodzących od drugiej stacji sieciowej do kolejki o numerze 5.

Opcja `–I INPUT` spowoduje dodanie reguły na początek łańcucha INPUT.

Po wykonaniu zadania proszę nie usuwać reguły (będzie wykorzystywana w kolejnym zadaniu).

5. Operację dodania reguły można zweryfikować za pomocą polecenia:
   `$ iptables –L INPUT –n`
6. Na drugiej stacji sieciowej należy wywołać program *ping* podając adres stacji, na której uruchomiono program *queue*:
   `$ ping –c 3 <adres IPv4>`
7. Proszę przeanalizować informacje wypisywane przez programy *queue* oraz *ping* (czy między stacjami zachodzi wymiana komunikatów ICMP?).
8. Dodatkowe informacje na temat kolejki można uzyskać wydając polecenie:
   `$ cat /proc/net/netfilter/nfnetlink_queue`

   Dane prezentowane są w formie tabeli. Kolejne kolumny oznaczają:
   - identyfikator kolejki,
   - identyfikator procesu powiązanego z kolejką,
   - liczbę pakietów czekających w kolejce na decyzję wysłania do powiązanego gniazda *Netlink*,
   - tryb kopiowania (2 dla `NFQNL_COPY_PACKET`),
   - maksymalny rozmiar danych kopiowanych z kolejki do gniazda *Netlink*,
   - liczbę pakietów porzuconych przez jądro systemu,
   - liczbę pakietów porzuconych przez proces użytkownika (np. z powodu braku miejsca w buforze gniazda),
   - liczbę pakietów przetworzonych przez kolejkę,
   - liczbę podmiotów korzystających z kolejki.

2.2. Zadanie 2. Modyfikacja pakietów

Celem zadania jest modyfikacja poprzedniego programu. Program, oprócz wypisywania informacji powinien przekształcać pole danych otrzymanych komunikatów *ICMP Echo*. W tym celu proszę wykorzystać funkcje `swap_bytes()` oraz `internet_checksum()` zaimplementowane w pliku *libqueue.c*. Pierwsza z nich zamienia kolejność bajtów we wskazanej tablicy. Parametrami funkcji `swap_bytes()` są wskaźnik na tablicę bajtów oraz rozmiar tablicy. Drugą funkcję można wykorzystać do obliczenia sumy kontrolnej komunikatu *ICMP* (jest to konieczne po modyfikacji pola danych komunikatu). W tym celu należy wywołać funkcję `internet_checksum()` przekazując jej wskaźnik na początek komunikatu *ICMP* oraz rozmiar komunikatu (nagłówka i da-

nych). Przed wywołaniem funkcji, proszę pamiętać o wyzerowaniu pola *Checksum* komunikatu *ICMP*.
Modyfikacja komunikatów odbywa się w łańcuchu INPUT (reguła *iptables* z poprzedniego zadania). Po opuszczeniu łańcucha INPUT przez komunikat, jądro systemu zweryfikuje jego sumę kontrolną. Jeżeli komunikat *ICMP Echo* zostanie uznany za poprawny, jądro wyśle odpowiedź *ICMP Reply*. W przeciwnym wypadku pakiet zostanie porzucony.

Uwagi:

- Proszę pamiętać o przekazaniu zmodyfikowanego pakietu i jego rozmiaru do funkcji `nfq_set_verdict()`.
- Do wykonania zadania potrzebne są 2 stacje sieciowe.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Upewnić się, że reguła *iptables* z poprzedniego zadania jest poprawnie skonfigurowana.
2. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o queuemod –lnetfilter_queue queuemod.c`
3. Uruchomić program (nie powinien przyjmować żadnych argumentów):
   `$ ./queuemod`
4. Na drugiej stacji sieciowej proszę uruchomić sniffer (konfiguracja powinna umożliwiać analizę komunikatów *ICMP Echo* oraz *ICMP Reply*).
5. Wywołać polecenie *ping* z adresem stacji, na której uruchomiono program *queuemod*:
   `$ ping –c 3 <adres IPv4>`
6. Czy między stacjami zachodzi wymiana komunikatów *ICMP*?
7. Jakie informacje wypisuje program *ping*?

2.3. Zadanie 3. Zarządzanie regułami iptables

Zadanie polega na analizie kodu przykładowego programu. Program umożliwia tworzenie nowych łańcuchów oraz usuwanie reguł *iptables*. Argumenty wywołania programu są zgodne z argumentami *iptables*. Proszę zwrócić szczególną uwagę na wywołania funkcji biblioteki *libiptc*.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o iptc –liptc iptc.c`
2. Usunąć utworzoną w zadaniu 1 regułę:
   `$ ./iptc –t filter –D INPUT 1`
3. Zweryfikować wynik operacji za pomocą programu *iptables*.
4. Utworzyć łańcuch *user_chain* w tablicy *nat*:
   `$ ./iptc –N user_chain –t nat`

5. Zweryfikować działanie programu i usunąć łańcuch:
   `$iptables –t nat –X user_chain`

2.4. Zadanie 4. Domyślna polityka łańcucha iptables

Celem zadania jest uzupełnienie funkcjonalności programu *iptc* o możliwość definiowania domyślnej polityki łańcucha – ACCEPT lub DROP. Określanie polityki ma odbywać się za pomocą opcji: `-P <nazwa łańcucha> <domyślna polityka>`. Proszę pamiętać, że tylko łańcuchy wbudowane mogą posiadać politykę domyślną. W przypadku próby zdefiniowania polityki dla łańcucha użytkownika, program powinien wypisać odpowiedni komunikat.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o iptcmod –liptc iptcmod.c`
2. Zdefiniować domyślną politykę łańcucha INPUT tablicy *filter* na DROP:
   `$ ./iptcmod –t filter –P INPUT DROP`
3. Zweryfikować wynik operacji za pomocą programu *iptables*.
4. Za pomocą polecenia *ping* proszę spróbować nawiązać połączenie ze stacją w sieci lokalnej. Jeżeli komputer jest podłączony do sieci Internet można wykorzystać przeglądarkę internetową w celu połączenia się z dowolną stroną WWW.
5. Zdefiniować domyślną politykę na ACCEPT.

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„Netfilter”* 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.

