Instytut Teleinformatyki

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

programowanie usług sieciowych

„Opcje IP i gniazda surowe”

laboratorium: 03
system operacyjny: Linux

Kraków, 2009

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące **opcji IP** oraz **gniazd surowych**. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematem laboratorium są gniazda surowe i opcje protokołu IP. Idea programowania surowych gniazd (ang. *raw sockets*) opiera się na ręcznej modyfikacji nagłówków poszczególnych protokołów. W przypadku gniazd surowych utworzenie nagłówków pakietu, wypełnienie pól nagłówków oraz wysłanie spreparowanego pakietu staje się obowiązkiem programisty. Dzięki stosowaniu gniazd tego rodzaju nie jesteśmy ograniczeni tylko do protokołów TCP czy UDP, ale mamy dostęp do warstwy sieciowej (IP, ICMP) i warstwy transportowej (TCP, UDP). Możemy implementować własne protokoły, jak również obsługiwać te, do których normalnie nie mielibyśmy dostępu stosując gniazda typu `SOCK_STREAM` lub `SOCK_DGRAM`.

Opcje IP (ang. *IP Options*) to dodatkowe pole nagłówka protokołu IP. Wszystkie opcje są definiowane przez jeden podzielony na trzy części bajt. Pierwsza część jest jednobitowa i określa zachowanie opcji w przypadku fragmentacji, druga część (dwubitowa) definiuje klasę opcji, a trzecia określa numer opcji. Dzięki opcjom nagłówka IP możliwe jest ustalenie m.in. dokładnego routingu źródłowego, zapisywanie trasy datagramu, czy też sterowanie opcjami bezpieczeństwa.

1.2. Zagadnienia do przygotowania

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

- pola nagłówka IPv4 [1]
- nazwy i przeznaczenie opcji IPv4 [1]
- stosowanie opcji IPv4 [6, 8]

Należy także zapoznać się z zagadnieniami dotyczącymi **gniazd surowych**:

- budowa nagłówków ICMP, TCP, UDP [3,4,5]
- budowa nagłówka IPv6 [2]

Literatura:

[1] IETF (http://www.ietf.org/), RFC 791, *„Internet Protocol”*

[2] IETF, RFC 2460, *„Internet Protocol, Version 6 Specification”*

[3] IETF, RFC 792, *„Internet Control Message Protocol”*

[4] IETF, RFC 793, *„Transmission Control Protocol”*

[5] IETF, RFC 768, *„User Datagram Protocol”*

[6] IETF, RFC 1122, *„Requirements for Internet Hosts - Communication Layers”*

[7] IETF, RFC 1071, *„Computing the Internet Checksum”*

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

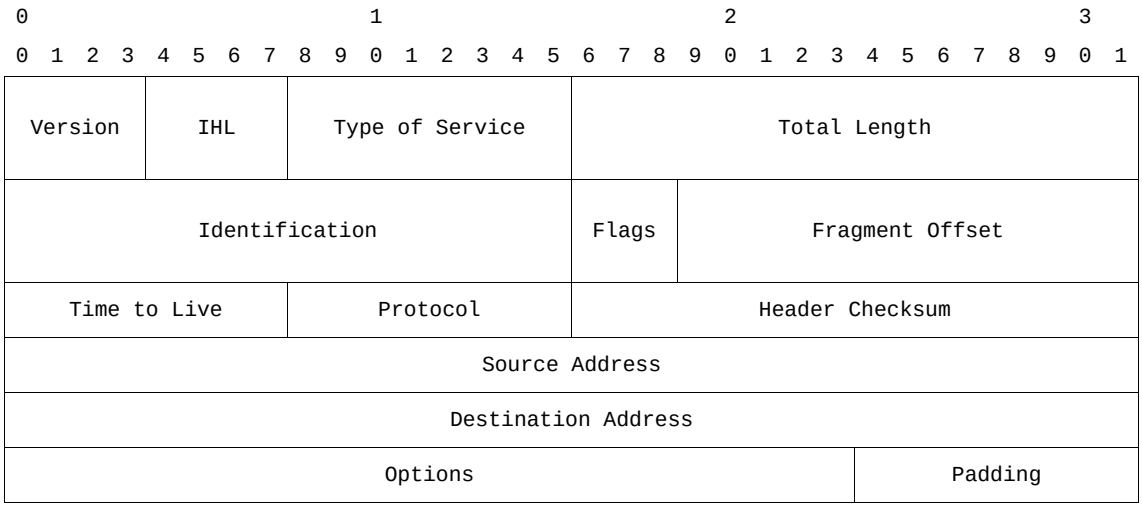
[9] CERT (http://www.cert.org/), Advisory CA-1996-21, *„TCP SYN Flooding and IP Spoofing Attacks”*

[10] IANA (http://www.iana.org/assignments/ip-parameters), *„IP OPTION NUMBERS”*

1.3. Opis laboratorium

1.3.1. Opcje IP

Tematyką pierwszej części laboratorium są opcje IP. Nagłówek protokołu IPv4 nie ma stałej długości. Oprócz wymaganych pól o długości 20 bajtów, może wystąpić pole opcji. Pole opcji znajduje się w pakiecie IP po adresie przeznaczenia pakietu (*ang. Destination Address*). Format pakietu IPv4 i umiejscowienie pola opcji zostało przedstawione na poniższym rysunku:



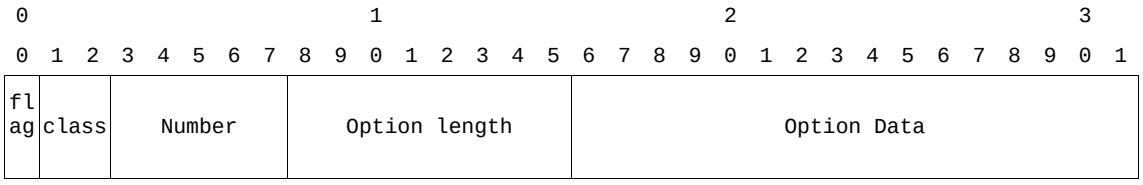
Rys. 1. Nagłówek datagramu IP[1].

Długość pola opcji nie może przekraczać 40 bajtów ponieważ znajdujące się w nagłówku 4 bitowe pole długości nagłówka IHL ogranicza jego długość do 60 bajtów. Podstawowe informacje w nagłówku IP zajmują 20 bajtów zatem na opcje IP pozostaje 40 bajtów (długość pola opcji może być zmienna). Do jednego pakietu może zostać dołączona jedna lub wiele opcji. Istnieją dwa formaty pola opcji:

- pojedynczy oktet z typem opcji,
- pole składające się z oktetu typu opcji, oktetu długości opcji oraz oktetu danych opcji.

Oktet długości opcji zlicza liczbę wszystkich oktetów danej opcji (1 oktet typu, 1 oktet długości i liczba oktetów danych).

Budowę opcji dla drugiego z wymienionych formatów przedstawia poniższy rysunek:



Rys. 2. Format opcji IP[1].

Pole `flag` określa zachowanie opcji podczas fragmentacji (1 – opcja kopiowana do wszystkich fragmentów), a pole `class` definiuje klasę opcji[1] (np.: klasa kontrolna). Zostało zdefiniowanych około 30 różnych opcji IP[10].

W przypadku protokołu IPv6, w nagłówku nie ma pola opcji. Jednakże istnieją dodatkowe nagłówki rozszerzeń (ang. *extension headers*), za pomocą których można transmitować opcjonalne informacje. Pakiet IPv6 może mieć zero, jeden lub większą liczbę nagłówków rozszerzeń umieszczonych pomiędzy nagłówkiem IPv6 oraz nagłówkiem protokołu warstwy wyższej.

1.3.2. Opcje gniazd

Funkcje `setsockopt()` oraz `getsockopt()` używane są w celu ustawienia lub pobrania parametru związanego z gniazdem sieciowymi.

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

int getsockopt(int s, int level, int optname, void *optval,
               socklen_t *optlen);

int setsockopt(int s, int level, int optname, const void *optval,
               socklen_t optlen);
```


Parametr	Opis
s	deskryptor gniazda, dla którego ustawiamy/pobieramy opcję
level	parametr określający poziom, którego dotyczy opcja: poziom gniazda - SOL_SOCKET lub poziom określonego protokołu (IPPROTO_IP, IPPROTO_IPV6, IPPROTO_TCP)
optname	nazwa opcji do ustawienia/pobrania (np. IP_TTL dla poziomu IPPROTO_IP)
optval	wskaźnik do bufora, w którym znajduje się wartość opcji do ustawienia / zapisana będzie wartość opcji
optlen	rozmiar bufora zawierającego wartość opcji

Przykład zastosowania funkcji setsockopt() dla ustalenia wartości TTL wysyłanych datagramów IP:

```
int       sockfd;
const int ttl = 64;

sockfd = socket(PF_INET, SOCK_STREAM, 0);

setsockopt(sockfd, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl));
```

Za pomocą funkcji setsockopt() można również zdefiniować zawartość pola opcji IP - określając poziom IPPROTO_IP oraz nazwę IP_OPTIONS. Kiedy ustanawia się opcje IP używając setsockopt(), wówczas opcje IP będą się znajdować we wszystkich datagramach IP wysyłanych z danego gniazda.
Funkcję getsockopt() można wywołać z argumentem IP_OPTIONS jedynie w celu pobrania odwrotność opcji wyznaczania trasy przez nadawcę. Jest to możliwe tylko dla połączonego gniazda TCP, które zostało utworzone przez funkcję accept(). W opisywanym przypadku, funkcja getsockopt() przekazuje odwrotność opcji odebranej wraz z segmentem SYN od klienta.

1.3.3. Gniazda surowe

Za wysyłanie oraz wypełnianie poszczególnych pól nagłówków IP, TCP, UDP odpowiedzialny jest system operacyjny. Jeśli nasza aplikacja chciałaby np. zmienić adres nadawcy, musimy samodzielnie i od podstaw skonstruować cały pakiet, łącznie z nagłówkiem IP. Korzystamy wtedy z gniazd surowych. W przypadku gniazd surowych, obowiązek stworzenia oraz analizy przesłanych pakietów spoczywa na użytkowniku.

Różnice pomiędzy gniazdami surowymi a gniazdami TCP i UDP:

- Surowe gniazda pozwalają czytać oraz pisać pakiety typu ICMPv4, IGMPv4, ICMPv6. Właściwość ta pozwala aplikacjom używać protokołów ICMP i IGMP przez procesy użytkownika (nie trzeba umieszczać kodu w jądrze).
- Korzystając z gniazd surowych można czytać i pisać pakiety, które nie będą przetwarzane przez jądro. Większość systemów przetwarza jedynie pakiety zawierające w polu protokół nagłówka IP wartości: 1 (ICMP), 2 (IGMP), 6 (TCP), 17 (UDP).
- Korzystając z gniazd surowych można zbudować własny nagłówek IPv4.

Tworzenie gniazda surowego

Programy tworzące gniazda surowe **wymagają zdolności CAP_NET_RAW**. Uprawnienia roota gwarantują jej posiadanie.

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

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

Parametr/opcja	Opis
`domain`	domena, w której nastąpi komunikacja poprzez gniazdo (konieczne jest określenie rodziny protokołów przypisanych do gniazda)
`type`	typ gniazda
`protocol`	rodzaj protokołu

Funkcja `socket()` zwraca wartość -1 w przypadku błędu lub deskryptor gniazda, jeśli udało się utworzyć gniazdo. Deskryptor gniazda spełnia analogiczną rolę, jak deskryptor pliku. Dzięki temu, do obsługi gniazd można wykorzystywać funkcje typowe dla obsługi plików (np. `write()`, `read()`, `close()`).
Funkcja `socket()` tworzy gniazdo surowe, kiedy argumentem `type` jest `SOCK_RAW`. W tym przypadku argument `protocol` najczęściej nie jest zerem, ale przyjmuje konkretną wartość (np.: `IPPROTO_ICMP`, `IPPROTO_TCP`, `IPPROTO_UDP`).

Dla gniazda surowego można ustawić opcję `IP_HDRINCL`:

```
int       sockfd;
const int option = 1;
```



```
sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP);
setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &option, sizeof(option));
```

Jeżeli opcja `IP_HDRINCL` nie jest ustawiona, to początkiem danych do wysłania jest pierwszy bajt za nagłówkiem IP. Wtedy jądro samo zbuduje nagłówek IP i przyłączy go do danych. Pole *Protocol* w nagłówku IP zostanie ustawione na takie, jakie było wyspecyfikowane w wywołaniu funkcji `socket()`.

Jeśli opcja `IP_HDRINCL` jest ustawiona, to początkiem danych do wysłania jest pierwszy bajt nagłówka IP. Długość danych do wysłania musi zawierać rozmiar nagłówka IP. Użytkownik buduje cały nagłówek IP za wyjątkiem:

- pola identyfikator (może być ustawione na zero, wtedy jądro samo ustawi wartość tego pola),
- pola sumy kontrolnej protokołu IP (pole to jest obliczane przez jądro),
- pola opcje IP (nie musi być zawarte).

W systemie Linux, opcji `IP_HDRINCL` nie można ustawić dla protokołu IPv6. Oznacza to, że programista nie może utworzyć nagłówka IPv6, dołączyć do niego dane (lub inne nagłówki) i wysłać tak spreparowany pakiet za pomocą gniazda surowego.

Dla funkcji `bind()` oraz `connect()` wywoływanych na gniazdach surowych nie mają znaczenia numery portów:

- `bind()` kojarzy adres z lokalnym gniazdem (datagramy IP wysyłane za pomocą gniazda będą posiadać źródłowy adres IP określony przez funkcję bind(), przy założeniu, że opcja `IP_HDRINCL` nie jest ustawiona).
- `connect()` kojarzy gniazdo z adresem zdalnym, co pozwala na użycie funkcji `write()` i `send()` zamiast `sendto()`.

Najczęściej z gniazdami surowymi wykorzystuje się funkcje `sendto()` oraz `recvfrom()`.

Wysyłanie datagramów

Wysyłanie odbywa się poprzez wywołanie funkcji `sendto()` albo `sendmsg()` wraz z określeniem adresu docelowego.

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

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


Parametr	Opis
`sockfd`	deskryptor gniazda
`*buff`	wskaźnik do bufora, z którego wysyła się dane
`nbytes`	liczba wysyłanych bajtów
`flags`	opcjonalne flagi
`*to`	wskaźnik na strukturę zawierającą adres pod który mają być wysłane dane
`addrlen`	rozmiar struktury gniazdowej

Odbieranie danych z gniazda surowego

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

ssize_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 `recvfrom()` adresem gniazda procesu partnera
`*addrlen`	wskaźnik do zmiennej przechowującej rozmiar struktury gniazdowej

Reguły dotyczące datagramów przychodzących z sieci:

- odbierając datagramy UDP i TCP, jądro nigdy nie przesyła ich do gniazd surowych; datagramy te można przeczytać jedynie na poziomie *datalink*,
- większość komunikatów ICMP dochodzi do gniazd surowych po przetworzeniu ich przez jądro; komunikaty ICMP: *echo request*, *timestamp request* i *address mask request* są w całości przetwarzane przez jądro i nie dochodzą do gniazd,
- wszystkie datagramy IGMP przechodzą do gniazd surowych po ich przetworzeniu przez jądro,
- wszystkie datagramy IP z nieznanym polem protokołu dla jądra systemu przechodzą bezpośrednio do gniazd surowych.

Dla datagramów, jądro zawsze sprawdza adres docelowy, sumę kontrolna, wersję protokołu i długość nagłówka.
Jeśli datagram przychodzi we fragmentach, to nie jest przesyłany do gniazda surowego dopóki nie zostanie złożony w całość.

Następnie wszystkie gniazda surowe (we wszystkich procesach) są sprawdzane pod kątem dopasowania do datagramu. Kopia datagramu jest dostarczana **do każdego** pasującego gniazda.

Dopasowanie datagramu do gniazda odbywa się wg poniższych reguł:

- jeśli trzeci argument funkcji `socket()` był niezerowy, to następuje porównanie pola protokół w datagramie przychodzącym z wartością określoną przez ten argument; jeśli protokół różni się od zadanego, to taki pakiet nie jest dostarczany do danego gniazda surowego,
- jeśli adres lokalny został powiązany z gniazdem surowym za pomocą funkcji `bind()`, wówczas adres docelowy datagramu musi być taki sam jak adres przypisany do gniazda; w przeciwnym wypadku datagram nie trafi do tego gniazda,
- jeśli adres zdalny został powiązany z gniazdem za pomocą funkcji `connect()`, `to` adres źródłowy datagramu musi być taki sam jak adres powiązany z gniazdem surowym,
- jeśli gniazdo surowe zostało utworzone z parametrem `protocol` równym zero oraz nie była wywołana funkcja `bind()` albo `connect()`, to wszystkie datagramy będą kierowane do tego gniazda.

W przypadku IPv4 do gniazda surowego przekazywane są całe datagramy - wraz z nagłówkiem IP. Dla IPv6 jedynie dane za nagłówkiem lub nagłówkami rozszerzeń (jeżeli występują).

1.4. Cel laboratorium

Celem laboratorium jest poznanie podstawowych opcji IP, które mogą zostać dołączone do nagłówka protokołu IP. Ponadto zostanie przećwiczone tworzenie gniazd surowych oraz korzystanie z nich do wysyłania i odbierania pakietów. Podczas realizacji tego laboratorium zapoznasz się z:

- budową pola opcji protokołu IPv4,
- z techniką tworzenia gniazd surowych,
- z budową nagłówków IPv4, IPv6, UDP oraz TCP,
- z możliwościami i ograniczeniami gniazd surowych,
- sposobem obliczania sumy kontrolnej dla IP, UDP oraz TCP.

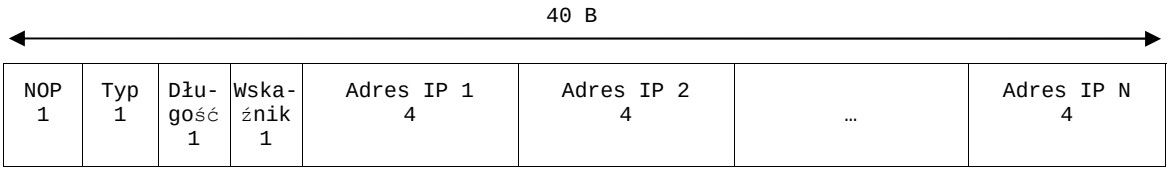
2. Przebieg laboratorium

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

2.1. Zadanie 1. Wyznaczanie trasy przez nadawcę

Program demonstruje możliwość wysyłania datagramu z opcją IP o nazwie SSRR za pomocą gniazda surowego. SSRR **-** rygorystyczne trasowanie według nadawcy (*ang. strict source and record route*) służy do przesyłania pakietów wzdłuż zdefiniowanej trasy. Opcja ta wskazuje drogę pakietu do stacji docelowej. W przypadku tej opcji datagram IP musi przejść przez wszystkie zdefiniowane węzły i tylko przez nie. Zatem kolejne węzły wyspecyfikowane przez opcję muszą być swoimi sąsiadami.

Poniżej został przedstawiony format opcji NOP oraz SSRR w pakiecie IP. Proszę zwrócić uwagę na rozmiar opcji i zastanowić się nad maksymalną liczbą węzłów jakie może przejść datagram IP z opcją SSRR.



Rys. 3. Rygorystyczne trasowanie według nadawcy (SSRR) wraz z opcją NOP.

Opis opcji IP z rys. 3:

Parametr/opcja	**Opis**
NOP	Opcja, która powoduje wyrównanie położenia adresów IP do wielokrotności czterech bajtów.
Typ	Typ opcji IP: 0x89=10001001 (class:0, option number: 9)
Długość	Rozmiar opcji SSRR w bajtach.
Wskaźnik	Określa numer pierwszego bajtu adresu IP, który będzie

	przetwarzany w następnej kolejności (licząc od pierwszego bajtu opcji SSRR).
Adres IP 1 – Adres N	Kolejne adresy IP przez które ma przejść pakiet.

Wywołanie funkcji `setsockopt()` z opcjami IP jak na rys.3 (wypełnionymi poprawnymi wartościami) powoduje, że datagram z opcją SSRR jest wysyłany na Adres 1, a za Adresem N dołączany jest adres określony w wywołaniu funkcji `sendto()`. W ten sposób nagłówek IP wysyłanego datagramu posiada łącznie N+2 adresów IP: *Source Address*, *Destination Address* (poprzedni Adres 1 z opcji SSRR) oraz N adresów określonych za pomocą opcji SSRR.
Zadanie polega na analizie kodu przykładowego programu oraz zaobserwowaniu budowy nagłówka datagramu IP. Proszę zwrócić szczególną uwagę na:

a. opcje IP przekazywane jako argument do funkcji `setsockopt()`,
b. postać opcji SSRR w zależności od adresu IP podawanego jako argument wywołania programu.

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 ssrr ssrr.c`
3. Uruchomić sniffer z odpowiednimi opcjami (np. tcpdump).
4. Uruchomić program podając jako argument adres IPv4 lub nazwę domenową i przeanalizować działanie programu:
   `$ ./ssrr <adres IP lub nazwa domenowa>`

2.2. Zadanie 2. Gniazda surowe – protokół UDP

Zadanie polega na analizie kodu przykładowego programu. Program wykorzystuje gniazdo surowe dla protokołu UDP, co 1 sekundę wysyłając puste datagramy (nagłówek UDP bez danych) na adres i numer portu określone przez argumenty wywołania. Zamiast adresu IP, argumentem wywołania może być nazwa domenowa. Numer portu i adres źródłowy zostały zdefiniowane jako stałe wartości.
W celu utworzenia gniazda typu `SOCK_RAW` program wykorzystuje funkcję `getaddrinfo()`. Funkcja ta przyjmuje jako pierwszy argument nazwę domenową lub adres IP. Zwrócona przez `getaddrinfo()` struktura `addrinfo` jest wykorzystywana w wywołaniu funkcji `socket()`. Po utworzeniu gniazda surowego, za pomocą funkcji `setsockopt()` ustawiana jest opcja `IP_HDRINCL`, a następnie wypełniane są nagłówki IP (`<netinet/ip.h>`) oraz UDP (`<netinet/udp.h>`). Wartość pola *Checksum* nagłówka UDP jest wypełniana sumą kontrolną obliczaną na podstawie pseudo-nagłówka (określo-

nych pól nagłówka IP oraz całego datagramu UDP). Wysyłanie datagramu zostało zrealizowane za pomocą funkcji `sendto()`.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami.
2. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o udp udp.c`
3. Uruchomić sniffer z odpowiednimi opcjami.
4. Uruchomić program podając jako argument adres IPv4 lub nazwę domenową oraz numer portu:
   `$ ./udp <adres IP lub nazwa domenowa> <numer portu>`
5. Przeanalizować działanie programu oraz postać wysyłanych pakietów.

2.3. Zadanie 3. Gniazda surowe – protokół TCP

Celem zadania jest zaimplementowanie programu wykorzystującego gniazdo surowe dla protokołu TCP. Program powinien zostać wykonany w oparciu o założenia z poprzedniego zadania (do implementacji można posłużyć się kodem źródłowym programu z zadania 2). Po utworzeniu gniazda proszę wypełnić nagłówki IP (`<netinet/ip.h>`) oraz TCP (`<netinet/tcp.h>`) poprawnymi wartościami. Nagłówek TCP należy wypełnić w taki sposób, aby program SYN flood'ował zadany adres IP na wybranym porcie. Wartość pola *Checksum* nagłówka TCP powinna zostać wypełniona sumą kontrolną obliczoną na podstawie pseudo-nagłówka (określonych pól nagłówka IP oraz całego segmentu TCP).

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o tcp tcp.c`
2. Uruchomić sniffer z odpowiednimi opcjami.
3. Uruchomić program podając jako argument adres IPv4 lub nazwę domenową oraz numer portu:
   `$ ./tcp <adres IP lub nazwa domenowa> <numer portu>`
4. Przeanalizować działanie programu oraz postać wysyłanych pakietów.

2.4. Zadanie 4. Gniazda surowe – protokół IPv6

Zadanie polega na zaimplementowaniu programu wykorzystującego gniazdo surowe dla protokołu IPv6. Przed wykonaniem ćwiczenia należy zweryfikować czy system operacyjny wspiera protokół IPv6. Można to sprawdzić na następujące sposoby:

a. Sprawdzając, czy w systemie plików istnieje `/proc/net/if_inet6`:


```
$ test –f /proc/net/if_inet6 && echo “IPv6-ready”
```

b. Wykorzystując program *ifconfig* (pakiet *net-tools) lub* program *ip (*pakiet *iproute2)*:

```
$ ifconfig
$ ip addr
```

Jeżeli w opisie interfejsów znajdzie się linia z adresem IPv6, oznacza to, że jest aktywna obsługa protokołu IPv6.

Program powinien co sekundę wysyłać puste datagramy UDP na adres i numer portu określone jako argumenty wywołania. W przeciwieństwie do programu z zadania 2, system operacyjny, a nie programista ma być odpowiedzialny za wypełnienie pól nagłówka IP. Za pomocą funkcji `setsockopt()` proszę - w sposób odpowiedni dla protokołu UDP - ustawić opcję `IPV6_CHECKSUM` (jądro systemu ma być odpowiedzialne za obliczanie i weryfikację sumy kontrolnej w nagłówku UDP). Opcja `IPV6_CHECKSUM`[8] wymaga określenia położenia (*offsetu*) pola sumy kontrolnej nagłówka UDP. Dla protokołu UDP *offset* wynosi 6 bajtów.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
   ```
   $ gcc –o udpv6 udpv6.c
   ```
2. Uruchomić sniffer z odpowiednimi opcjami.
3. Uruchomić program podając jako argument adres IPv6 oraz numer portu:
   ```
   $ ./udpv6 <adres IPv6> <numer portu>
   ```
4. Przeanalizować działanie programu oraz postać wysyłanych pakietów.

2.5. Zadanie 5. Prosty program ping

Ostatnim zadaniem jest implementacja prostego programu ping. Program jako argument wywołania ma przyjmować adres IP lub nazwę domenową. Po uruchomieniu, proces główny ma utworzyć proces potomny odpowiedzialny za odbieranie komunikatów ICMP Echo Reply, a sam ma zająć się wysyłaniem komunikatów ICMP Echo.
Proces główny powinien kolejno: utworzyć gniazdo surowe dla protokołu ICMP, ustawić za pomocą funkcji `setsockopt()` opcję `IP_TTL` na poprawną wartość, wysłać 4 komunikaty ICMP Echo w odstępach czasowych wynoszących 1s. Wartość pola *Identifier* komunikatu ICMP (`<netinet/ip_icmp.h>`) ma zostać ustawiona na PID procesu głównego, a pole *Sequence Number* ma zawierać numer wysyłanego komunikatu (1-4). Pole danych komunikatu należy wypełnić losowymi znakami alfabetu i cyframi tak, aby całkowita długość komunikatu nie przekraczała 64 bajtów.
Proces potomny ma utworzyć gniazdo surowe, za pomocą funkcji `recvfrom()` odebrać komunikaty ICMP i wypisać na standardowe wyjście dane z nagłówków IP i ICMP. Wśród informacji ma znaleźć się:

- adres hosta, który przesłał odpowiedź,

- TTL odebranego pakietu,
- rozmiar nagłówka IP,
- adres docelowy (z nagłówka IP),
- informacje na temat komunikatu ICMP (typ, kod, identyfikator i numer sekwencyjny).

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o ping ping.c`
2. Uruchomić sniffer z odpowiednimi opcjami.
3. Uruchomić program podając jako argument adres IP lub nazwę domenową:
   `$ ./ping <adres IP lub nazwa domenowa>`
4. Przeanalizować działanie programu oraz postać wysyłanych i odbieranych komunikatów ICMP.
5. Proszę ponownie uruchomić program i w trakcie działania (natychmiast po uruchomieniu) wywołać systemowy *ping* podając ten sam lub inny adres IP:
   `$ ping –c 1 <adres IP>`
6. Proszę zwrócić uwagę na identyfikatory i numery sekwencyjne wypisywane przez program, który był tematem zadania. Czy odebrany został komunikat ICMP związany z systemowym programem *ping*?

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„Opcje IP i gniazda surowe”* 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.

