Instytut Teleinformatyki

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

programowanie usług sieciowych

„Protokoły IPv4 i IPv6”

laboratorium: 02
system operacyjny: Linux

Kraków, 2009

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące protokołów internetowych IPv4 i IPv6. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji klient-serwer wykorzystując protokoły trzeciej warstwy modelu OSI: **IPv4 i IPv6**. IP (ang. *Internet Protocol*) udostępnia usługi pakowania i adresowania. IP identyfikuje hosty lokalne i zdalne. Gdy trasa do sieci docelowej wymaga pakietów mniejszych rozmiarów, implementacja IP dzieli pakiet na fragmenty, co pozwala na ich transmisję. Implementacja IP w stacji hosta docelowego jest odpowiedzialna za ich poprawną defragmentację. Ponadto protokół IP odrzuca pakiety przeterminowane (na podstawie limitu określonego w nagłówku – *Time To Live*) oraz przekazuje wyznaczone pakiety do protokołów warstw wyższych.

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi **IPv4**: [1, 2]

- budowa nagłówka IPv4 [4]
- adresowanie IPv4 [4]
- adresy prywatne i specjalne [5, 6]

A także z zagadnieniami dotyczącymi **IPv6**: [1, 3]

- format nagłówka IPv6
- nagłówki rozszerzeń IPv6 i ich kolejność
- adresowanie w protokole IPv6
- *Scope zone* i *zone ID* w protokole IPv6 [7, 9]

Zapoznać się należy także z narzędziami: `netstat`, `ifconfig` oraz interfejsem programowania gniazd (tworzenie gniazd, funkcje odpowiedzialne za wysyłanie, odbieranie danych i konwersję adresów IP) [1, 8].

Literatura:

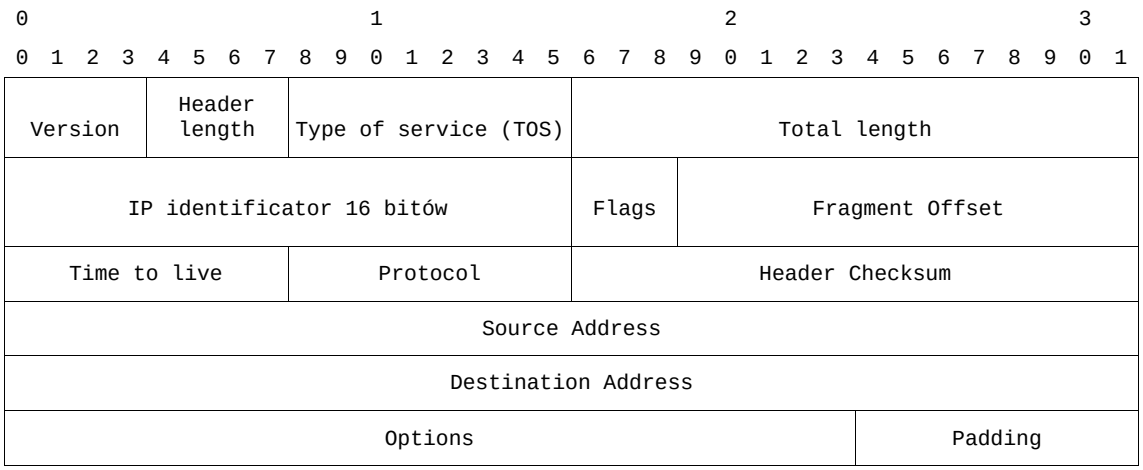
[1] W.R. Stevens, *„Programowanie Usług Sieciowych”*, *„API: gniazda i XTI”*.
[2] R. Scrimger, P. LaSalle, C. Leitzke, M. Parihar, M. Gupta, *„TCP/IP.Biblia”*
[3] IETF (http://www.ietf.org/), RFC 2460, *„Internet Protocol, Version 6 Specification”*
[4] IETF, RFC 791, *„Internet Protocol”*
[5] IETF, RFC 1918, *„Address Allocation for Private Internets”*
[6] IETF, RFC 3330, *„Special-Use IPv4 Addresses”*
[7] IETF, RFC 4007, *„IPv6 Scoped Address Architecture”*
[8] IETF, RFC 3493, *„Basic Socket Interface Extensions for IPv6”*
[9] Qing Li, T. Jinmei, K. Shima, *„IPv6 Core Protocols Implementation”*, Morgan Kaufman

1.3. Opis laboratorium

1.3.1. Ogólnie o IP

IP dostarcza procedur wystarczających do przesyłania danych między hostami znajdującymi się w połączonych sieciach. Definiuje strukturę i format pakietów oraz sposób ich adresowania. Nie realizuje jednak żadnych funkcji związanych z poprawnością transmisji, w szczególności nie identyfikuje pakietów, które mają być przesłane ponownie (retransmitowane). Nie potrafi także wykonywać wielu procesów związanych z odtwarzaniem prawidłowej sekwencji pakietów (pakiety podróżujące różnymi drogami mogą dotrzeć do celu w innej kolejności niż zostały nadane). Jest więc protokołem bezpołączeniowym - nie zapewnia stałego kanału komunikacyjnego.

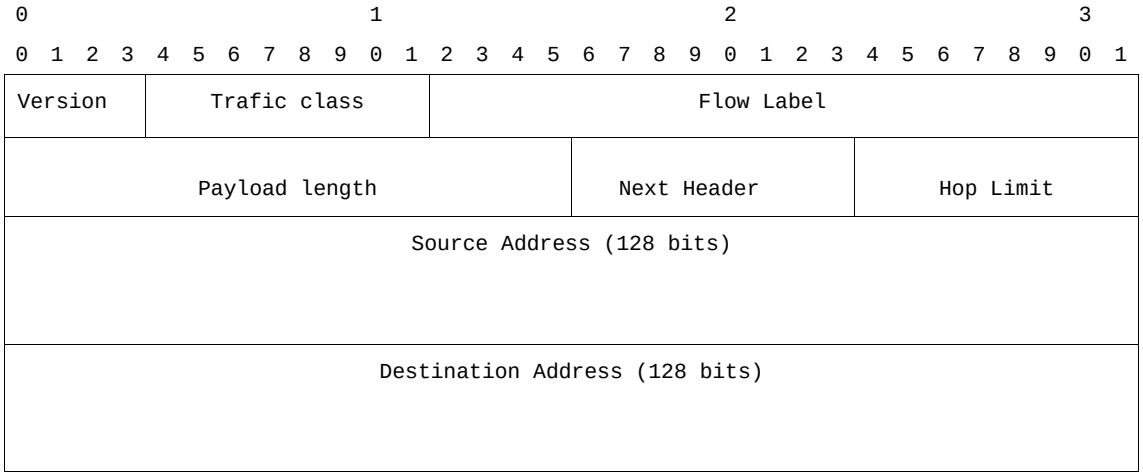
IPv4 (ang. *Internet Protocol Version 4*) – najpopularniejsza obecnie wersja protokołu IP. Identyfikacja hostów odbywa się za pomocą 32-bitowych adresów IP. Dane przesyłane są w postaci standardowych datagramów.



Rys. 1. Pola nagłówka IPv4[4].

IP wersja 6 (IPv6) jest nową wersją protokołu IP. Główne zmiany w porównaniu do IPv4 to:

- **Rozszerzone adresowanie**
  Powiększenie długości adresu z 32 do 128 bitów pozwala na stworzenie wielopoziomowej hierarchii adresowania, zaadresowanie znacznie większej ilości węzłów oraz uproszczenie autokonfiguracji adresów. Ulepszenie skalowalności routingu dla adresów *mulitacast* zostało osiągnięte poprzez zdefiniowanie zakresu adresu (ang. *address scope*). Dodanie nowego typu adresu - *anycast* - pozwala na wysyłanie pakietów do najbliższego węzła z grupy.
- **Uproszczenie formatu nagłówka**
  Niektóre (nieużywane lub nieprzydatne) pola z nagłówka IPv4 zostały usunięte, a inne stały się opcjonalne. Dzięki temu zabiegowi został zredukowany koszt przetwarzania nagłówków oraz ich wielkość. Owocuje to większą oszczędnością pasma.
- **Ulepszenie obsługi rozszerzeń nagłówków i opcji**
  Opcje IPv6 umieszczane są w oddzielnych nagłówkach znajdujących się po zasadniczym nagłówku IPv6. Nagłówki rozszerzeń nie muszą być przetwarzane przez wszystkie systemy pośredniczące w przesyłaniu pakietów na drodze od nadawcy do odbiorcy.
- **Zdolność etykietowania przepływu**
  Możliwość etykietowania pozwala na oznaczanie ruchu sieciowego, który ma być przesyłany z zapewnieniem danej jakości usługi (QoS).
- **Uwierzytelnianie i zdolność zapewnienia poufności**
  Zostały wyspecyfikowane rozszerzenia IPv6, które umożliwiają: uwierzytelnianie nadawcy, zapewnienie integralności oraz (opcjonalnie) poufności przesyłanych danych.



Rys. 2. Pola nagłówka IPv6[3].

1.3.2. Współdziałanie protokołów IPv4 oraz IPV6

Jednym z podstawowych założeń protokołu IPv6 była możliwość płynnej migracji z protokołu IPv4. Migracja taka, oznacza możliwość koegzystencji IPv4 i IPv6. Główne mechanizmy przejściowe to:

a. implementacja zarówno IPv4 jak i IPv6 w stosach TCP/IP stacji sieciowych (stacje dwuprotokołowe – ang. *dual stack*),
b. tunelowanie.

Tunelowanie (IPv6 over IPv4) zapewnia możliwość komunikowania się hostów IPv6 poprzez istniejącą infrastrukturę IPv4 (dzięki enkapsulacji pakietów IPv6 w datagramach IPv4).

Pomijając mechanizmy tunelowania, istnieje wiele wariantów komunikacji klient-serwer. Dla systemu Linux możliwe warianty przedstawia poniższa tabela:

	IPv4 server IPv4-only node	IPv4 server IPv6/IPv4 node	IPv6 server IPv6-only node	IPv6 server IPv6/IPv4 node	IPv4/IPv6 server IPv6/IPv4 node
IPv4 client IPv4-only node	IPv4	IPv4	-	IPv4	-
IPv4 client IPv4/IPv6 node	IPv4	IPv4	-	IPv4	-
IPv6 client IPv6-only node	-	-	IPv6	IPv6	-
IPv6 client IPv4/IPv6 node	IPv4	IPv4	IPv6	IPv6	-
IPv4/IPv6 client IPv4/IPv6 node	-	-	-	-	-

Komórki tabeli zawierają wersje protokołu IP dla pakietów przesyłanych podczas komunikacji klient-serwer.
W systemie Linux nie można utworzyć gniazd IPv4 oraz IPv6 wykorzystujących ten sam numer portu, ponieważ stos TCP/IP posiada jedną implementację warstwy transportowej dla obu wersji protokołu IP.

Stacje dwuprotokołowe powinny przestrzegać następujących zasad obsługi **gniazd nasłuchujących**:

- Dla nasłuchującego gniazda IPv4 można ustanawiać połączenia tylko z klientami IPv4.
- Jeżeli serwer ma nasłuchujące gniazdo IPv6 dowiązane do adresu uogólnionego, to dla takiego gniazda można ustanawiać połączenia zarówno z klientami IPv4, jak i klientami IPv6. Jako lokalny adres serwera dla połączenia z klientem IPv4 należy przyjąć odpowiedni adres IPv6 utworzony z adresu IPv4 (*IPv4-mapped IPv6 address*).

- Jeżeli serwer ma nasłuchujące gniazdo IPv6 dowiązane do adresu IPv6, który nie jest adresem IPv6 utworzonym z adresu IPv4, to dla takiego gniazda można ustanawiać połączenia tylko z klientami IPv6.

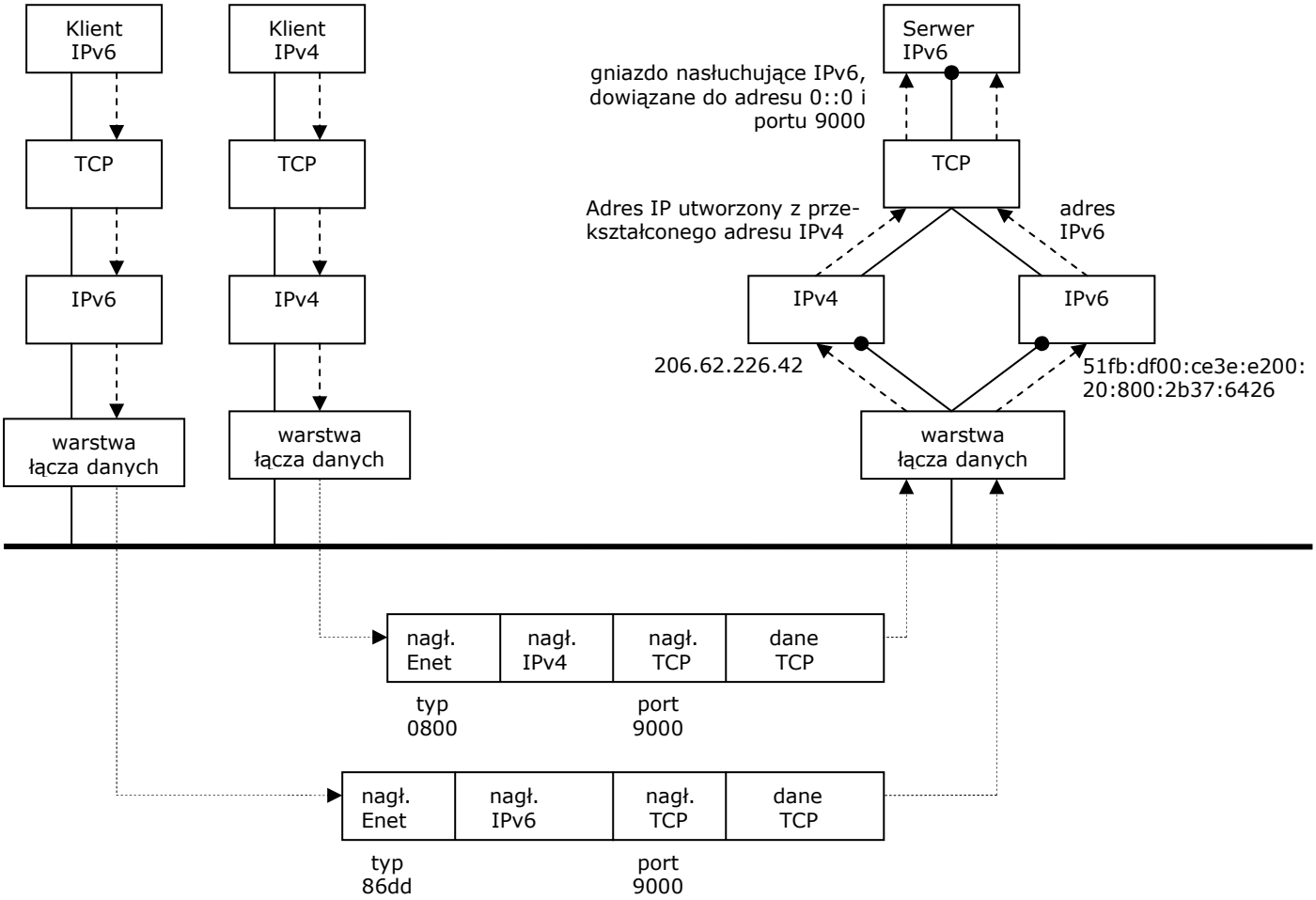
Schemat przetwarzania datagramów IPv4 oraz IPv6 odebranych przez **serwer uruchomiony na stacji dwuprotokołowej**:

- Jeżeli datagram IPv4 jest odbierany przez gniazdo IPv4, to nic się szczególnego nie dzieje. Między klientem a serwerem są wymieniane datagramy IPv4.
- Jeżeli datagram IPv6 jest odbierany przez gniazdo IPv6, to nic się szczególnego nie dzieje. Między klientem a serwerem są wymieniane datagramy IPv6.
- Jeżeli natomiast datagram IPv4 jest odbierany przez gniazdo IPv6, to jądro systemu przekazuje za pośrednictwem funkcji `accept()` lub `recvfrom()` odpowiedni adres IPv6 utworzony z adresu IPv4 (*IPv4-mapped IPv6*). Takie przekształcenie jest możliwe dzięki temu ze każdy adres IPv4 można reprezentować jako adres IPv6. Między klientem a serwerem są wymieniane datagramy IPv4.
- Stwierdzenie odwrotne do przedstawionego w poprzednim punkcie nie jest prawdziwe, ponieważ adresu IPv6 z zasady nie można przedstawić w postaci adresu IPv4.

Schemat przetwarzania datagramów IPv4 oraz IPv6 nadawanych przez **klienta uruchomionego na stacji dwuprotokołowej**:

- Jeżeli klient IPv4 TCP wywoła funkcję `connect()` lub klient IPv4 UDP wywoła `sendto()` przekazując jej adres IPv4 serwera, to nic się szczególnego nie dzieje. Między klientem a serwerem są wymieniane datagramy IPv4.
- Jeżeli klient IPv6 TCP wywoła funkcję `connect()` lub klient IPv6 UDP wywoła `sendto()` przekazując jej adres IPv6 serwera, to nic się szczególnego nie dzieje. Między klientem a serwerem są wymieniane datagramy IPv6.
- Jeżeli klient IPv6 TCP wywoła funkcję `connect()` lub klient IPv6 UDP wywoła `sendto()` przekazując jej adres IPv4 serwera przekształcony na adres IPv6, to wysyłany jest datagram IPv4.

1.3.3. Komunikacja klienta IPv4 z serwerem IPv6



Rys. 3. Serwer IPv6 uruchomiony na stacji dwuprotokołowej, który obsługuje klientów IPv4 oraz IPv6[1].

Przebieg komunikacji między klientem IPv4 i serwerem IPv6 uruchomionym na stacji dwuprotokołowej:

1. Serwer IPv6 tworzy nasłuchujące gniazdo IPv6. Zakładamy, że serwer wywołuje funkcję `bind()`, aby dowiązać do gniazda adres uogólniony.
2. Klient IPv4 wywołuje funkcję `gethostbyname()` i znajduje rekord A dotyczący serwera. Ponieważ stacja serwera obsługuje oba protokoły, więc odpowiedzią jest zarówno rekord A jak i rekord AAAA; jednak klient IPv4 żąda tylko rekordu A.
3. Klient wywołuję funkcje `connect()` i stacja klienta wysyła do serwera segment SYN, używając protokołu IPv4.
4. Stacja serwera odbiera segment SYN wysłany jako segment IPv4, ale przeznaczony dla nasłuchującego gniazda IPv6, po czym ustawia sygnalizator, aby zaznaczyć ze w tym połączeniu będą stosowane adresy IPv6 utworzone z adresów

IPv4 (IPv4-mapped IPv6). Następnie stacja serwera odpowiada segmentem SYN/ACK, korzystając z protokołu IPv4. Po ustanowieniu połączenia funkcja `accept()`, którą wywołał program serwera, przekazuje adres IPv6 utworzony z adresu IPv4.

5. W dalszym przebiegu komunikacji między klientem a serwerem używane są wyłącznie datagramy IPv4.
6. Jeżeli program serwera nie sprawdzi, czy adres klienta jest adresem IPv6 utworzonym z adresu IPv4, to nie dowie się o tym, że komunikuje się z klientem IPv4. Również klient IPv4 nie posiada informacji o tym, ze komunikuje się z serwerem IPv6.

1.3.4. Komunikacji klienta IPv6 z serwerem IPv4

Przebieg komunikacji między klientem IPv6 uruchomionym na stacji dwuprotokołowej i serwerem IPv4:

1. Serwer IPv4 uruchomiony w stacji obsługującej tylko protokół IPv4 tworzy nasłuchujące gniazdo IPv4.
2. Uruchomiony klient IPv6 wywołuje funkcje `gethostbyname`, żądając jedynie adresów IPv6 (ustawia opcje `RES_USE_INET6`). Ponieważ stacja serwera obsługującego wyłącznie protokół IPv4 utrzymuje tylko rekordy A, wiec do klienta będzie przekazany adres IPv6 utworzony z przekształconego adresu IPv4.
3. Klient IPv6 wywołuje funkcje `connect`. Wewnątrz struktury adresowej gniazda IPv6 klient przekazuje adres IPv6 utworzony z przekształconego adresu IPv4. Jadro systemu klienta, rozpoznaje adres przekształcony i automatycznie określa protokół IPv4 dla segmentu SYN wysyłanego do serwera.
4. W odpowiedzi stacja serwera wysyła segment SYN/ACK, korzystając z protokołu IPv4. Ustanowione w ten sposób połączenie jest obsługiwane zgodnie z protokołem IPv4.

1.3.5. Struktury adresowe

sockaddr

Funkcje operujące na gniazdach powstały przed opracowaniem standardu ANSI C (typ `void` nie istniał). Struktura `sockaddr` jest ogólną strukturą gniazdową (wskaźnik do struktury `sockaddr` można traktować jak wskaźnik na typ `void`). Wskaźniki na struktury adresowe określonych protokołów (np. protokołów IPv4 i IPv6) rzutuje się na wskaźnik do struktury `sockaddr`. Dzięki zastosowaniu wskaźnika do struktury `sockaddr`, funkcje operujące na gniazdach posiadają jednolity interfejs umożliwiający przyjmowanie struktur adresowych różnych protokołów.

sockaddr_in

Poniżej przedstawiona jest struktura `sockaddr_in` przechowująca informacje adresowe dla protokołu IPv4:

```
#include <netinet/in.h>

struct sockaddr_in {
      short int       sin_family; /* AF_INET */
      unsigned short  sin_port;   /* Numer portu */
      struct in_addr  sin_addr;   /* Adres IP */
      unsigned char   sin_zero[8] /* Wyzerowane */
};
```

Liczba pól struktury może różnić się w zależności od systemu operacyjnego, ale zawsze występują 3 pola:

- `sin_family` – domena adresowa (`AF_INET` dla IPv4),
- `sin_port` – numer portu przechowywany w porządku sieciowym,
- `sin_addr` – struktura przechowująca adres IP w porządku sieciowym (*big endian*); struktura przedstawiona jest poniżej:

```
#include <netinet/in.h>

struct in_addr {
      unsigned int s_addr;
};
```

Konwencją jest wyzerowanie całej struktury `sockaddr_in` przed jej użyciem, np. za pomocą funkcji `memset()`.

sockaddr_in6

Poniżej przedstawiona jest struktura `sockaddr_in6` przechowująca informacje adresowe dla protokołu IPv6:

```
#include <netinet/in.h>

struct sockaddr_in6 {
      short int       sin6_family;    /* AF_INET6 */
      unsigned short  sin6_port;      /* Numer portu */
      unsigned int    sin6_flowinfo;  /* Kontrola przepływu */
      struct in6_addr sin6_addr;      /* Adres IP */
      unsigned int    sin6_scope_id;  /* Scope ID */
};
```

Pola struktury `sockaddr_in6`:

- `sin6_family`: domena adresowa; zawsze `AF_INET6`,
- `sin6_port`: numer portu w porządku sieciowym (*big endian*),

- `sin6_flowinfo`: wyzerowane (obecna specyfikacja nie określa zastosowania),
- `sin6_scope_id`: indeks interfejsu dla adresów link-local.
- `sin6_addr`: struktura adresowa przechowująca adres IPv6 w porządku sieciowym, przedstawiona poniżej:

```
#include <netinet/in.h>

struct in6_addr {
      unsigned char s6_addr[16];
};
```

Przykład wykorzystania struktur adresowych przez proces serwera:

IPv4

```
struct sockaddr_in server;

memset(&server, 0, sizeof(server));
server.sin_family      = AF_INET;
server.sin_port        = htons(80);
server.sin_addr.s_addr = htonl(INADDR_ANY);

bind(sockfd, (struct sockaddr*)&server, sizeof(struct sockaddr_in));
```

IPv6

```
struct sockaddr_in6 server;

memset(&server, 0, sizeof(server));
server.sin6_family      = AF_INET6;
server.sin6_port        = htons(80);
server.sin6_addr        = in6addr_any;

bind(sockfd, (struct sockaddr*)&server, sizeof(struct sockaddr_in6));
```

sockaddr_storage

Struktura adresowa niezależna od rodziny protokołów oraz wersji protokołu (wystarczająco duża, aby pomieścić adres IPv4 lub IPv6). Z programistycznego punktu widzenia interesujące jest tylko pole `ss_family` typu `short`. Pole to określa rodzinę protokołów, dla której struktura przechowuje informacje adresowe (np. `AF_INET`). Na podstawie pola `ss_family` można rzutować wskaźnik do struktury `sockaddr_storage` na wskaźnik do struktury specyficznej dla danego protokołu (np. `sockaddr_in*`) i uzyskać dostęp dla odpowiednich pól. Najczęściej strukturę wykorzystuje się w programach niezależnych od wersji protokołu, w wywołaniach funkcji `acccept()`, `recvfrom()` oraz `getsockname()` i `getnameinfo()` – kiedy nie można z góry określić jakie informacje adresowe będzie przechowywać struktura danych. W wywołaniach wymie-

nionych wyżej funkcji należy wskaźnik do struktury `sockaddr_storage` rzutować na wskaźnik do struktury `sockaddr`.
Przykład wykorzystania struktury `sockaddr_storage` przez proces serwera:

```
int                             sockfd, connfd;
struct sockaddr_storage         client;
socklen_t                       client_len;
char                            host[NI_MAXHOST], serv[NI_MAXSERV];

(…)
client_len = sizeof(struct sockaddr_storage);

connfd = accept(sockfd, (struct sockaddr*)&client, &client_len);

getnameinfo((struct sockaddr*)&client, client_len, host, NI_MAXHOST,
            serv, NI_MAXSERV, NI_NUMERICHOST|NI_NUMERICSERV);

printf("TCP connection accepted from address %s port %s\n",
       host, serv);
```

1.4. Cel laboratorium

Celem ćwiczenia jest zapoznanie się z:

- budową nagłówka IPv4,
- budową nagłówka IPv6,
- reprezentacją adresów IP w pamięci komputera,
- komunikacją między klientem i serwerem wykorzystującymi protokół IPv4,
- komunikacją między klientem i serwerem wykorzystującymi protokół IPv6,
- komunikacja między klientem wykorzystującym protokół IPv4 oraz serwerem IPv6,
- komunikacja między klientem wykorzystującym protokół IPv6 oraz serwerem IPv4.

2. Przebieg laboratorium

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

2.1. Przygotowanie laboratorium

Przed wykonaniem ćwiczenia należy zweryfikować czy system operacyjny wspiera protokół IPv6 (najczęściej poprzez moduł jądra).

Można to sprawdzić na następujące sposoby:

1. Sprawdzając, czy w systemie plików istnieje plik `/proc/net/if_inet6`:
   `$ test –f /proc/net/if_inet6 && echo “IPv6-ready”`
   System operacyjny wspiera protokół IPv6, jeżeli na standardowym wyjściu pojawi się napis: „IPv6-ready”.

2. Wykorzystując program *ifconfig* z pakietu *net-tools*:
   `$ ifconfig`
   Jeżeli w opisie interfejsów znajdzie się linia podobna do tej (z adresem IPv6):
   `inet6 addr: fe80::230:4fff:fe18:d8eb/64 Scope:Link`
   oznacza to, że jest aktywna obsługa protokołu IPv6.

3. Wykorzystując program *ip* z pakietu *iproute2*:
   `$ ip addr`
   Jeżeli w opisie interfejsów znajdzie się linia podobna do tej (z adresem IPv6):
   `inet6 fe80::230:4fff:fe18:d8eb/64 scope link`
   oznacza to, że jest aktywna obsługa protokołu IPv6.

Każdy interfejs obsługujący protokół IPv6 musi posiadać co najmniej jeden adres *link-local* (RFC 3513).

2.2. Zadanie 1. Przekształcanie adresów IP

Program demonstruje w jaki sposób dokonuje się przekształceń pomiędzy adresami IP w notacji zrozumiałej dla człowieka (np. w notacji kropkowo-dziesiętnej dla IPv4) a postacią, która może być przetwarzana przez maszynę (odpowiednia dla danego protokołu struktura danych). Zadanie polega na analizie kodu przykładowego programu oraz zaobserwowaniu w jaki sposób adresy przechowywane są w pamięci komputera (szczególną uwagę należy zwrócić na porządek bajtów). W celu łatwiejszego zrozumienia organizacji danych adresowych w pamięci, program wypisuje adresy IPv4 w postaci binarnej i adresy IPv6 w postaci heksadecymalnej bezpośrednio z adresowych struktur danych.

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 ipaddr ipaddr.c`
3. Uruchomić program podając jako argument adres IPv4 lub IPv6 w notacji zrozumiałej dla człowieka i przeanalizować działanie programu:
   `$ ./ipaddr <adres IPv4 lub IPv6>`

2.3. Zadanie 2. Odwzorowanie nazwy domenowej na adres IP

Program demonstruje w jaki sposób dokonuje się odwzorowania nazw domenowych na adresy IP. Zadanie polega na analizie kodu przykładowego programu, który został zaimplementowany z wykorzystaniem funkcji `getaddrinfo()`, niezależnej od wersji protokołu IP.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami.
2. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o hostname2ip hostname2ip.c`
3. Uruchomić program podając jako argument nazwę domenową i przeanalizować działanie programu:
   `$ ./hostname2ip <nazwa domenowa>`
4. Dlaczego niektóre nazwy domenowe są odwzorowywane na kilka adresów IP? Przykład:

```
$ ./hostname2ip google.pl
IPv4: 66.249.93.104
IPv4: 72.14.221.104
IPv4: 216.239.59.104
```


2.4. Zadanie 3. Komunikacja klienta IPv4 z serwerem IPv6

Zadanie polega na zaimplementowaniu programów serwera IPv6 oraz klienta IPv4 komunikujących się za pomocą protokołu TCP.

Program serwera należy zaimplementować w oparciu o poniższe założenia:

1. Serwer ma zostać zaimplementowany z wykorzystaniem struktury adresowej `sockaddr_in6` (adres struktury przekazać funkcji `bind()`).
2. Serwer może odebrać dane z dowolnego interfejsu sieciowego (`in6addr_any`).
3. Serwer obsługuje jednego klienta na raz (model iteracyjny: oczekiwanie w nieskończonej pętli na akceptację połączenia za pomocą funkcji `accept()`).
4. Po zaakceptowaniu nowego połączenia TCP, serwer wypisuje adres i numer portu klienta (można posłużyć się funkcją `inet_ntop()`).
5. W dalszej kolejności serwer sprawdza, czy adres zwrócony przez funkcję `accept()` jest adresem IPv6, czy IPv4-mapped IPv6 (wskazówka: można posłużyć się makrodefinicją `IN6_IS_ADDR_V4MAPPED`) i wypisuje na standardowe wyjście odpowiedni komunikat.
6. Kolejnym etapem jest przesłanie do klienta krótkiej wiadomości: „Laboratorium PUS”. Po przesłaniu komunikatu, serwer powinien zakończyć obsługę bieżącego połączenia TCP i oczekiwać na nowe połączenie.

Program klienta IPv4 ma mieć możliwość nawiązania połączenia TCP z serwerem o adresie IPv4 określonym przez argument wywołania programu. Po nawiązaniu połączenia klient powinien odebrać wiadomość od serwera, wypisać ją na standardowe wyjście i zakończyć działanie. Klient ma zostać zaimplementowany z wykorzystaniem struktury `sockaddr_in` (adres struktury przekazać funkcji `connect()`).

W celu uruchomienia programów należy wykonać następujące czynności:

1. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client4 client4.c
   $ gcc –o server6 server6.c
   ```
2. Na jednej ze stacji uruchomić sniffer z odpowiednimi opcjami (np. tcpdump).
3. Uruchomić serwer IPv6 podając wybrany przez siebie numer portu:
   ```
   $ ./server6 <numer portu>
   ```
4. Uruchomić klienta podając adres IPv4 oraz numer portu, na którym nasłuchuje serwer (adres interfejsu *loopback*: 127.0.0.1) i przeanalizować działanie programów:
   ```
   $ ./client4 <adres IPv4> <numer portu>
   ```
5. Należy odpowiedzieć na następujące pytania:
   a. Czy istnieje komunikacja pomiędzy programem klienta IPv4 i serwera IPv6?

b. Jakie warunki muszą zostać spełnione, aby taka komunikacja była możliwa?
c. Jaką postać mają podczas komunikacji przepływające datagramy: IPv6 czy IPv4?

2.5. Zadanie 4. Komunikacja klienta IPv6 z serwerem IPv4

Zadanie polega na zaimplementowaniu programów serwera IPv4 oraz klienta IPv6 komunikujących się za pomocą protokołu TCP.

Program serwera należy zaimplementować w oparciu o poniższe założenia:

1. Serwer ma zostać zaimplementowany z wykorzystaniem struktury adresowej `sockaddr_in` (adres struktury przekazać funkcji `bind()`).
2. Serwer może odebrać dane z dowolnego interfejsu sieciowego (`INADDR_ANY`).
3. Serwer obsługuje jednego klienta na raz (model iteracyjny: oczekiwanie w nieskończonej pętli na akceptację połączenia za pomocą funkcji `accept()`).
4. Po zaakceptowaniu nowego połączenia TCP, serwer wypisuje adres i numer portu klienta (można posłużyć się funkcją `inet_ntop()`.
5. Kolejnym etapem jest przesłanie do klienta krótkiej wiadomości: „Laboratorium PUS”. Po przesłaniu komunikatu, serwer powinien zakończyć obsługę bieżącego połączenia TCP i oczekiwać na nowe połączenie.

Program klienta IPv6 ma mieć możliwość nawiązania połączenia TCP z serwerem o adresie IPv6 określonym przez argument wywołania programu. Po nawiązani połączenia klient powinien odebrać wiadomość od serwera, wypisać ją na standardowe wyjście i zakończyć działanie. Klient, ma zostać zaimplementowany z wykorzystaniem struktury `sockaddr_in6` (adres struktury przekazać funkcji `connect()`). Pole `sin6_scope_id` struktury `sockaddr_in6` powinno zostać wypełnione indeksem interfejsu przez który ma nastąpić komunikacja. W tym celu oprócz adresu IPv6 i numeru portu serwera, klient musi jako argument wywołania przyjmować nazwę interfejsu, przez który nastąpi komunikacja. Nazwę interfejsu można odwzorować na indeks za pomocą funkcji `if_nametoindex()`. Pole `sin6_scope_id` ma znaczenie dla komunikacji z adresami *link-local* (polecenie `ifconfig` oznacza takie adresy za pomocą `Scope:Link`).

W celu uruchomienia programów należy wykonać następujące czynności:

1. Skompilować programy źródłowe do postaci binarnej:
   ```
   $ gcc –o client6 client6.c
   $ gcc –o server4 server4.c
   ```
2. Na jednej ze stacji uruchomić sniffer z odpowiednimi opcjami (np. tcpdump).

3. Uruchomić serwer IPv4 podając wybrany przez siebie numer portu:
   `$ ./server4 <numer portu>`
4. Uruchomić klienta podając adres *IPv4-mapped IPv6*, na którym nasłuchuje serwer (adres interfejsu *loopback:* ***::ffff:127.0.0.1***), numer portu serwera oraz nazwę interfejsu (*lo*):
   `$ ./client6 <adres IPv6> <numer portu> <nazwa interfejsu>`
5. Należy odpowiedzieć na następujące pytania:
   a. Czy istnieje komunikacja pomiędzy programem klienta IPv6 i serwera IPv4?
   b. Jakie warunki muszą zostać spełnione, aby taka komunikacja była możliwa?
   c. Jaką postać mają podczas komunikacji przepływające datagramy: IPv6 czy IPv4?

2.6. Zadanie 5. API niezależne od protokołu

Celem zadania jest modyfikacja klienta IPv4 w taki sposób, aby stworzyć program niezależny od wersji protokołu IP. Można to osiągnąć za pomocą funkcji `getaddrinfo()`. Funkcja ta przyjmuje jako pierwszy argument nazwę domenową lub adres IP (IPv4 lub IPv6) w notacji zrozumiałej dla człowieka. `getaddrinfo()` zwraca strukturę (jedną lub kilka) `addrinfo`, która z kolei zawiera wskaźnik na gniazdową strukturę adresową `sockaddr`. Za pomocą zwróconej przez funkcję struktury `addrinfo` można utworzyć gniazdo odpowiedniego typu:

a. Jeżeli argumentem funkcji był adres IPv6 to struktura `addrinfo` zawiera dane potrzebne do utworzenia gniazda, które będzie wykorzystywane do komunikacji w domenie `AF_INET6`,
b. Jeżeli argumentem funkcji był adres IPv4 to struktura `addrinfo` zawiera dane potrzebne do utworzenia gniazda, które będzie wykorzystywane do komunikacji w domenie `AF_INET`.

Po ustanowieniu połączenia TCP za pomocą funkcji `connect()`, klient powinien wypisać adres IP oraz numer portu powiązane z lokalnym gniazdem (funkcje `getsockname()` oraz `getnameinfo()`), odebrać komunikat serwera (wypisać go na standardowe wyjście) i zakończyć działanie. W wywołaniach funkcji `getsockname()` oraz `getnameinfo()` proszę wykorzystać strukturę `sockaddr_storage` – niezależną od wersji protokołu IP.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:
   `$ gcc –o client client.c`
2. Na jednej ze stacji uruchomić sniffer z odpowiednimi opcjami (np. tcpdump).

3. Przeanalizować działanie programu i komunikację sieciową z serwerami z poprzednich zadań. Uruchamianie programu ma odbywać się wg schematu:
   ```
   $ ./client <adres IPv4 lub IPv6> <numer portu>
   ```


3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„Protokoły IPv4 i IPv6”* 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.

