Programowanie usług sieciowych												
02	Temat:	***Protokoły IPv4 i IPv6***	Zadania:									Data:
	Autor:	**Witold Karaś**	1	2	3	4	5					**2017-03-01**
	Autor:	**Łukasz Maj**	e	e	m	m	m					**14:30-16:00**

Zadanie 1. Przekształcanie adresów IP

Celem zadania była analiza kodu podanego programu oraz zaobserwowanie w jaki sposób adresy przechowywane są w pamięci komputera.

Program demonstruje sposób przekształceń dokonywanych pomiędzy adresami IP w notacji kropkowo-dziesiętnej, a postacią zrozumiałą dla komputera (binarną).

Wykorzystane funkcje:

- *inet_aton()* - przeprowadza konwersję adresu IPv4 z postaci kropkowo-dziesiętnej (pierwszy parametr) do postaci binarnej. Drugim parametrem jest wskaźnik na strukturę in_addr, gdzie in_addr jest strukturą przechowującą adres IP w porządku sieciowym
- *inet_addr()* - przeprowadza konwersję adresu IPv4 z postaci kropkowo-dziesiętnej (parametr funkcji) do postaci binarnej
- *inet_pton()* - przeprowadza konwersję adresu IPv4 lub IPv6 (pierwszy parametr) z postaci tekstowej (drugi parametr) do postaci binarnej (trzeci parametr)
- *inet_ntoa()* - przeprowadza konwersję adresu IPv4 z postaci binarnej do postaci tekstowej (wartość zwracana). Kolejne wywołanie funkcji spowoduje nadpisanie bufora
- *inet_ntop()* - przeprowadza konwersję adresu IPv4 lub IPv6 (pierwszy parametr) z postaci binarnej do postaci tekstowej. Ostatnim parametrem jest rozmiar bufora dla adresu w formie tekstowej

Przykładowe działanie programu:

```
qhoros@Qhoros:~/Documents/Pk/PUS/PUS-02-Protokoly_IPv4_i_IPv6-Linux/src$ ./ipaddr 192.168.20.1
Internet host address to network address structure conversion:

inet_aton(): 11000000.10101000.00010100.00000001 (binary)
inet_addr(): 11000000.10101000.00010100.00000001 (binary)
inet_pton(): 11000000.10101000.00010100.00000001 (binary)
inet_pton(): invalid IPv6 address!

Network address structure to internet host address conversion:

inet_ntoa(): 192.168.20.1
inet_ntop(): 192.168.20.1
```

Adres IP przechowany jest w strukturze zależnej od wersji adresu:

- IPv4 - w strukturze *sockaddr_in*
- IPv6 - w strukturze *sockaddr_in6*

Adresy przechowywane są binarnie w porządku big endian. Big endian jest to forma zapisu danych, w której najbardziej znaczący bajt (zwany też górnym bajtem) umieszczony jest jako pierwszy.

Zadanie 2. Odwzorowanie nazwy domenowej na adres IP

Celem zadania była analiza kodu programu tłumaczącego nazwy domenowe na adresy IP.

Program wykorzystuje w tym celu funkcję *getaddrinfo()*. Funkcja ta korzystając z protokołu DNS pozwala na uzyskanie adresów IP podanego hosta niezależnie od wersji protokołu IP. Program uwzględnia możliwość uzyskania wielu adresów IP dla jednej nazwy domenowej. Związana jest ona z techniką równoważenia obciążenia tzw. *Round-robin DNS.*

Warte wspomnienia funkcje wykorzystywane w programie:

- *getaddrinfo()* - omawiana wyżej
- *getnameinfo()* - wykorzystywana do przetłumaczenia adresu IP z postaci binarnej na tekstową, zrozumiałą dla człowieka.

Przykładowe działanie programu:

```
qwerty@qwerty ~/PUS-02-Protokoly_IPv4_i_IPv6-Linux/src $ ./hostname2ip facebook.
com
IPv4: 31.13.93.36
IPv6: 2a03:2880:f11c:83:face:b00c:0:25de
qwerty@qwerty ~/PUS-02-Protokoly_IPv4_i_IPv6-Linux/src $ ./hostname2ip microsoft
.com
IPv4: 104.43.195.251
IPv4: 23.100.122.175
IPv4: 23.96.52.53
IPv4: 191.239.213.197
IPv4: 104.40.211.35
```

Zadanie 3. Komunikacja klienta IPv4 z serwerem IPv6

Celem zadania była implementacja programów serwera IPv6 i klienta IPv4 komunikujących się z użyciem protokołu TCP.

Struktura programu jest zbliżona do przykładów zamieszczonych w dokumentacji wykorzystywanych funkcji API Linuxa.

Program serwera tworzy nowy socket o parametrach wskazanych w poleceniu do zadania:

```
//AF_INET6 - communications domain = IPv6
//SOCK_STREAM - socket type = TCP
//IPPROTO_TCP - TCP (see netinet/in.h)
int socketDescriptor=socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
```

a także strukturę typu sockaddr_in6 i inicjalizuje ją:

```
memset(&serverSocketAddress, 0, sizeof(serverSocketAddress));
serverSocketAddress.sin6_family=AF_INET6;
serverSocketAddress.sin6_port=htons(portNumber);
serverSocketAddress.sin6_addr=in6addr_any;
```

po czym następuje bindowanie socketa do struktury *bind()* i ustawienie trybu nasłuchiwania *listen()).*
Następnie serwer w pętli oczekuje i akceptuje przychodzące połączenia, drukuje informacje o kliencie, wysyła do niego komunikat, po czym zamyka gniazdo.

Program klienta w analogiczny sposób przygotowuje strukturę sockaddr_in (wersja dla IPv4) oraz socket. Następnie łączy się z serwerem funkcją *connect(),* odbiera wysłany komunikat, drukuje go na ekran, po czym kończy działanie.
Zrzut okna programu *tcpdump* (serwer uruchomiony na porcie: 2222, port klienta: 34249):

```
qwerty@qwerty ~/PUS02 $ sudo tcpdump -i lo
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 65535 bytes
13:51:11.281876 IP localhost.34249 > localhost.2222: Flags [S], seq 682034822, win 43690, options [mss 65495,sackO
K,TS val 4294949056 ecr 0,nop,wscale 7], length 0
13:51:11.281885 IP localhost.2222 > localhost.34249: Flags [S.], seq 3169100402, ack 682034823, win 43690, options
 [mss 65495,sackOK,TS val 4294949056 ecr 4294949056,nop,wscale 7], length 0
13:51:11.281892 IP localhost.34249 > localhost.2222: Flags [.], ack 1, win 342, options [nop,nop,TS val 4294949056
 ecr 4294949056], length 0
13:51:11.281947 IP localhost.2222 > localhost.34249: Flags [P.], seq 1:17, ack 1, win 342, options [nop,nop,TS val
 4294949056 ecr 4294949056], length 16
13:51:11.281950 IP localhost.34249 > localhost.2222: Flags [.], ack 17, win 342, options [nop,nop,TS val 429494905
6 ecr 4294949056], length 0
13:51:11.281956 IP localhost.2222 > localhost.34249: Flags [F.], seq 17, ack 1, win 342, options [nop,nop,TS val 4
294949056 ecr 4294949056], length 0
13:51:11.282041 IP localhost.34249 > localhost.2222: Flags [F.], seq 1, ack 18, win 342, options [nop,nop,TS val 4
294949056 ecr 4294949056], length 0
13:51:11.282044 IP localhost.2222 > localhost.34249: Flags [.], ack 2, win 342, options [nop,nop,TS val 4294949056
 ecr 4294949056], length 0
```

Odpowiedzi na pytania:

a) Czy istnieje komunikacja między programem klienta IPv4 i serwera IPv6?
   Tak, komunikacja między tymi programami zachodzi.
b) Jakie warunki muszą zostać spełnione, aby taka komunikacja była możliwa?
   Adres IPv4 musi zostać zmapowany na adres IPv6. Jest to wykonywane automatycznie przez funkcję *getpeername()* podczas łączenia się klienta IPv4 z serwerem IPv6 z wykorzystaniem funkcji *accept()*.

c) Jaką postać mają podczas komunikacji przepływające datagramy: IPv6 czy IPv4?
Datagramy mają postać IPv4 (*tcpdump -i lo -vv ip6* nie wyświetla żadnych wyników - brak datagramów IPv6).

Zadanie 4. Komunikacja klienta IPv6 z serwerem IPv4

Celem zadania było zaimplementowanie programów serwera IPv4 oraz klienta IPv6 komunikujących się za pomocą protokołu TCP.
Przykładowe działanie:

```
qhoros@Qhoros:~/Documents/Pk/PUS$ ./server 9000
SERVER: Waiting...
SERVER: Connection established
ADRES IPv4: 127.0.0.1
CLIENT PORT: 58872

qhoros@Qhoros:~/Documents/Pk/PUS$ ./client ::ffff:127.0.0.1 9000 lo
CLIENT: Server msg: Laboratorium PUS
```

Serwer IPv4 wykorzystuje strukturę *sockaddr_in*, jako, że może odbierać dane z dowolnego interfejsu użyto flagi *INADDR_ANY*. Serwer obsługuje tylko jednego klienta jednocześnie.

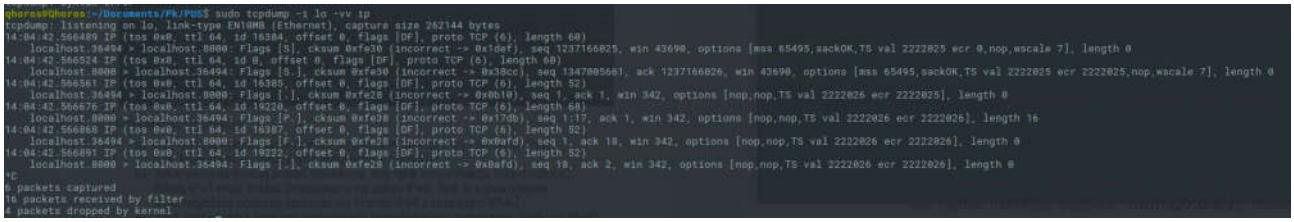
KLIENT: Wykorzystane funkcje:

- **int socket(int** *domain***, int** *type***, int** *protocol***) -** tworzy uchwyt dla komunikacji oraz zwraca deskryptor pliku, który odnosi się do tegoż uchwytu, gdzie *domain* to AF_INET6 (adres IP w wersji 6), *type* to SOCK_STREAM (sekwencyjne dwukierunkowe połączenie bazujące na strumieniu bajtów), *protocol* to IPPROTO_TCP (protokół zewnętrzny bramy)
- **ssize_t recv(int** *sockfd***, void *****buf***, size_t** *len***, int** *flags***) -** służy do otrzymywania wiadomości z socketu, gdzie *sockfd* jest rezultatem wykonania funkcji *socket*, **buf* zawiera przesłaną wiadomość, *len* przechowuje maksymalną liczbę znaków, pole *flags* nie zostało wypełnione
- **int connect(int** *sockfd***, const struct sockaddr *****addr***, socklen_t** *addrlen***)** - łączy deskryptor pliku z adresem, pole *sockfd* zostało już omówione, **addr* jest wskaźnikiem na strukturę *sockaddr* zaś *addrlen* przechowuje rozmiar struktury (sizeof).
- **unsigned int if_nametoindex(const char *****ifname***)** - funkcja zwraca indeks interfejsu sieciowego skojarzonego z podaną przez użytkownika nazwą *ifname.*

SERWER: Wykorzystane funkcje

- **int bind(int** *sockfd***, const struct sockaddr *****addr***, socklen_t** *addrlen***)** - kojarzy deskryptor pliku z adresem (argumenty zostały omówione)
- **int listen(int** *sockfd***, int** *backlog***) -** oznacza socket jako pasywny, oznacza to socket, którego nadchodzące połączenie zostanie zaakceptowane, gdzie *backlog* definiuje maksymalną wielkość, do jakiej kolejka oczekujących połączeń może urosnąć (lub od wersji linuxa 2.2 połączeń dokonanych). Jeżeli wartość argumentu przekracza */proc/sys/net/core/somaxconn,* wtedy liczba ta jest automatycznie przycinana do wartości domyślnej (na ogół jest to 128).
- **int accept(int** *sockfd***, struct sockaddr *****addr***, socklen_t *****addrlen***) -** wydobywa pierwszą prośbę o połączenie z połączeń oczekujących (argumenty zostały już omówione)
- **ssize_t send(int** *sockfd***, const void *****buf***, size_t** *len***, int** *flags***)** - używana do przesyłania wiadomości do innego socketa (argumenty zostały już omówione)

Zrzut ekranu programu *tcpdump* dla datagramów w postaci IPv4:



Zrzut ekranu programu *tcpdump* dla datagramów w postaci IPv6:

```
qhoros@Qhoros:~/Documents/Pk/PUS$ sudo tcpdump -i lo -vv ip6
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

Odpowiedzi na pytania:

a) Czy istnieje komunikacja między programem klient IPv6 i serwera IPv4?
Tak, komunikacja między tymi programami zachodzi.

b) Jakie warunki muszą zostać spełnione, aby taka komunikacja była możliwa?
Adres IPv6 musi zostać zmapowany na adres IPv4. Jako, że nie można odwzorować adresu IPv6 na IPv4 użytkownik sam taki zmapowany adres musi wprowadzić np. wpisując ::ffff:127.0.0.1.

c) Jaką postać mają podczas komunikacji przepływające datagramy: IPv6 czy IPv4?
Datagramy mają postać IPv4 (*tcpdump -i lo -vv ip6* nie wyświetla żadnych wyników - brak datagramów IPv6).

Zadanie 5. API niezależne od protokołu

Celem zadania było takie zmodyfikowanie klienta IPv4 (zadanie 3), aby mógł się on komunikować przy użyciu zarówno protokołu IPv4 jak i IPv6.

Zastosowany protokół zależy od typu wprowadzonego przez użytkownika adresu IP.
Rozpoznanie typu adresu zrealizowane jest następująco:

Tworzona i inicjalizowana jest pomocnicza struktura:

```
addrinfo hints;
addrinfo *serverSocketAddressArray;

hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = 0;
hints.ai_protocol = IPPROTO_TCP;
```

następnie wywoływana jest funkcja:

```
getaddrinfo(argv[1], argv[2], &hints, &serverSocketAddressArray
```

która na podstawie pomocniczej struktury *hints,* wejściowego adresu IP i numeru portu w postaci tekstowej generuje strukturę (a właściwie ich listę) zawierającą adres IP w postaci zrozumiałej dla komputera. Następnie program tworzy socket wykorzystując typ adresu IP uzyskany przed chwilą:

```
int socketDescriptor=socket(serverSocketAddressList->ai_family, serverSocketAddressList->ai_socktype, serverSocketAddressList->ai_protocol);
```

oraz łączy się z gniazdem. Dalej, tworzona jest struktura typu sockaddr_storage wykorzystana do wywołania funkcji: *getsockname()* i *getnameinfo()*, których celem jest uzyskanie ze struktury adresu IP i numeru portu w postaci tesktowej - które następnie program drukuje.

Pozostałe aspekty działania programu są identyczne jak w przypadku zad.3.

Program w połączeniu z serwerem IPv4 komunikuje się przez protokół IPv4, a z serwerem IPv6 w zależności od typu użytego adresu. Przykład działania programu dla adresu IPv6 i serwera IPv6:

```
qwerty@qwerty ~/PUS02 $ ./client ::1 2223
IPv4/v6 client - PUS Lab02-5
Witold Karaś, Łukasz Maj
Port number:2223
IPv6 socket family
Server IP Address: ::1 server port number: 34976
Message from server: Laboratorium PUS
```