Programowanie Aplikacji Sieciowych - Laboratorium 2

Tworzenie gniazd TCP

- **Język Python - klient/serwer**

```
#!/usr/bin/env python
import socket

if __name__ == '__main__':

        sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)

        sockIPv4.close()
        sockIPv6.close()
```

- **Język C/C++ - klient/serwer**

```
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int sockIPv4 = -1;

    if ((sockIPv4 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket");
        exit(1);
    }

    close(sockIPv4);

    return 0;
}
```

```
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int sockIPv6 = -1;

    if ((sockIPv6 = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
    {
        perror("socket");
        exit(1);
    }

    close(sockIPv6);

    return 0;
}
```

- **Język Java - klient**

```
import java.io.IOException;
import java.net.Socket;

public class Main {

   public static void main(String[] args) {

       Socket sockIPv4 = new Socket();

       try {
           sockIPv4.close();
       } catch (IOException ex) {
           System.out.println(ex);
       }
   }
}
```

```
import java.io.IOException;
import java.net.Socket;

public class Main {

       public static void main(String[] args) {

              System.setProperty("java.net.preferIPv6Addresses", "true");
              Socket sockIPv6 = new Socket();

              try {
                     sockIPv6.close();
              } catch (IOException ex) {
                     System.out.println(ex);
              }
       }
}
```

- **Język Java - serwer**

```
import java.io.IOException;
import java.net.ServerSocket;

public class Main {

   public static void main(String[] args) {
       try {
           ServerSocket sockIPv4 = new ServerSocket();
           sockIPv4.close();
       } catch (IOException e) {
           System.out.println(e);
       }
   }
}
```

```
import java.io.IOException;
import java.net.ServerSocket;

public class Main {

   public static void main(String[] args) {

       System.setProperty("java.net.preferIPv6Addresses", "true");

       try {
           ServerSocket sockIPv6 = new ServerSocket();
           sockIPv6.close();
       } catch (IOException ex) {
           System.out.println(ex);
       }
   }
}
```

Tworzenie gniazd UDP

- **Język Python - klient/serwer**

```
#!/usr/bin/env python
import socket

if __name__ == '__main__':

        sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sockIPv6 = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)

        sockIPv4.close()
        sockIPv6.close()
```

- **Język C/C++ - klient/serwer**

```
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int sockIPv4 = -1;

    if ((sockIPv4 = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("socket");
        exit(1);
    }

    close(sockIPv4);

    return 0;
}
```

```
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int sockIPv6 = -1;

    if ((sockIPv6 = socket(AF_INET6, SOCK_DGRAM, 0)) < 0)
    {
        perror("socket");
        exit(1);
    }

    close(sockIPv6);

    return 0;
}
```

- **Język Java - klient/serwer**

```
public class Main {

    public static void main(String[] args) {

        try {
            DatagramSocket sockIPv4 = new DatagramSocket();
            sockIPv4.close();

        } catch (SocketException ex) {
            System.out.println(ex);
        }
    }
}
```

```
public class Main {

    public static void main(String[] args) {

        System.setProperty("java.net.preferIPv6Addresses", "true");

        try {
            DatagramSocket sockIPv6 = new DatagramSocket();
            sockIPv6.close();

        } catch (SocketException ex) {
            System.out.println(ex);
        }
    }
}
```

Nawiązanie połączenia z serwerem

- **Język Python**

```
#!/usr/bin/env python
import socket

if __name__ == '__main__':

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.settimeout(5)

    try:
        sockIPv4.connect(('172.217.20.163', 80))
    except socket.error, exc:
        print "Wyjatek socket.error : %s" % exc

    sockIPv4.close()
```

```
#!/usr/bin/env python
import socket

if __name__ == '__main__':

    sockIPv4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockIPv4.settimeout(5)

    result = sockIPv4.connect_ex(('172.217.20.163', 80))
    if result == 0:
        print "Connected"

    sockIPv4.close()
```

- **Język C/C++**

```
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <arpa/inet.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int sockIPv4 = -1;
    struct sockaddr_in server;

    if ((sockIPv4 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket");
        exit(1);
    }

    server.sin_addr.s_addr = inet_addr("172.217.20.163");
    server.sin_family = AF_INET;
    server.sin_port = htons(80);

    if (connect(sockIPv4 , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("connect");
        exit(1);
    }

    close(sockIPv4);

    return 0;
}
```

1. Napisz program, który z serwera ntp.task.gda.pl pobierze aktualną datę i czas, a następnie wyświetli je na konsoli. Serwer działa na porcie 13.

2. Napisz program klienta, który połączy się z serwerem TCP działającym pod adresem 212.182.24.27 na porcie 2900, a następnie wyśle do niego wiadomość i odbierze odpowiedź.

3. Napisz program klienta, który połączy się z serwerem TCP działającym pod adresem 212.182.24.27 na porcie 2900, a następnie będzie w pętli wysyłał do niego tekst wczytany od użytkownika, i odbierał odpowiedzi.

4. Napisz program klienta, który połączy się z serwerem UDP działającym pod adresem 212.182.24.27 na porcie 2901, a następnie wyśle do niego wiadomość i odbierze odpowiedź.

5. Napisz program klienta, który połączy się z serwerem UDP działającym pod adresem 212.182.24.27 na porcie 2901, a następnie będzie w pętli wysyłał do niego tekst wczytany od użytkownika, i odbierał odpowiedzi.

6. Napisz program klienta, który połączy się z serwerem UDP działającym pod adresem 212.182.24.27 na porcie 2902, a następnie prześle do serwera liczbę, operator, liczbę (pobrane od użytkownika) i odbierze odpowiedź.

7. Zmodyfikuj program numer 6 z laboratorium nr 1 w ten sposób, aby oprócz wyświetlania informacji o tym, czy port jest zamknięty, czy otwarty, klient wyświetlał również informację o tym, jaka usługa jest uruchomiona na danym porcie.

8. Zmodyfikuj program numer 7 z laboratorium nr 1 w ten sposób, aby oprócz wyświetlania informacji o tym, czy porty są jest zamknięte, czy otwarte, klient wyświetlał również informację o tym, jaka usługa jest uruchomiona na danym porcie.

9. Napisz program klienta, który połączy się z serwerem UDP działającym pod adresem 212.182.24.27 na porcie 2906, a następnie prześle do serwera adres IP, i odbierze odpowiadającą mu nazwę hostname.

10. Napisz program klienta, który połączy się z serwerem UDP działającym pod adresem 212.182.24.27 na porcie 2907, a następnie prześle do serwera nazwę hostname, i odbierze odpowiadający mu adres IP.

11. Zmodyfikuj program nr 2 z laboratorium nr 2 w ten sposób, aby klient wysłał i odebrał od serwera wiadomość o maksymalnej długości 20 znaków. Serwer TCP odbierający i wysyłający wiadomości o długości 20 działa pod adresem 212.182.24.27 na porcie 2908. Uwzględnij sytuacje, gdy:

    - wiadomość do wysłania jest za krótka - ma być wówczas uzupełniania do 20 znaków znakami spacji
    - wiadomość do wysłania jest za długa - ma być przycięta do 20 znaków (lub wysłana w całości - sprawdź, co się wówczas stanie)

12. Funkcje `recv` i `send` nie gwarantują wysłania / odbioru wszystkich danych. Rozważmy funkcję `recv`. Przykładowo, 100 bajtów może zostać wysłane jako grupa po 10 bajtów, albo od razu w całości. Oznacza to, iż jeśli używamy gniazd TCP, musimy odbierać dane, dopóki nie mamy pewności, że odebraliśmy odpowiednią ich ilość. Zmodyfikuj program nr 11 z laboratorium nr 2 w ten sposób, aby mieć pewność, że klient w rzeczywistości odebrał / wysłał wiadomość o wymaganej długości.