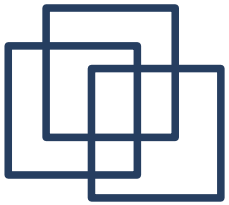


Plan wykładu

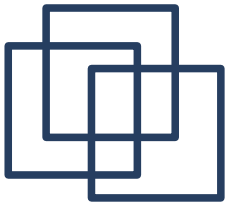
1. Programowanie z wykorzystaniem gniazd

usługa TIME,

usługa ECHO,

2. Popularne aplikacje TCP/IP – poczta elektroniczna cz. 1

protokół SMTP,



Programowanie z wykorzystaniem gniazd

Utworzenie gniazda:

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

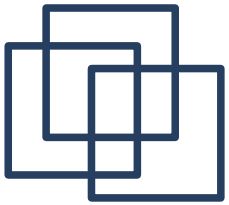
`domain` – rodzina protokołów używana do komunikacji w sieci, zdefiniowana w pliku **`<sys/socket.h>`** np.: **`PF_UNIX, PF_LOCAL, PF_INET, PF_INET6, PF_IPX`**.

`type` – rodzaj komunikacji np.: **`SOCK_STREAM, SOCK_DGRAM, SOCK_RDM`**.

`protocol` – protokół transmisji wykorzystywany przez gniazdo.

Wartość zwracana: deskryptor gniazda lub -1 w przypadku błędu.

Pliki nagłówkowe **`<sys/socket.h>, <sys/types.h>`**.



Programowanie z wykorzystaniem gniazd

Połączenie poprzez gniazdo:

```
int connect (int sock, const struct sockaddr *serv_addr,
int len)
```

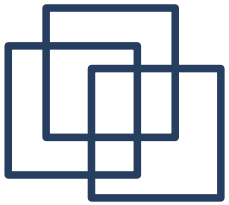
sock – deskryptor gniazda uzyskany funkcją **socket**.

serv_addr – wskaźnik do struktury opisującej punk docelowy połączenia.

len – długość struktury wskazywanej przez **serv_addr**.

Wartość zwracana: 0 w przypadku powodzenia, -1 w przypadku błędu.

Pliki nagłówkowe **<sys/socket.h>, <sys/types.h>**.



Programowanie z wykorzystaniem gniazd

Zamknięcie połączenia (gniazda):

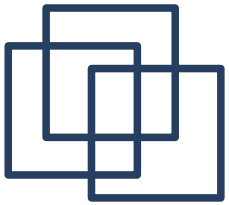
`int shutdown (int sock, int how)`

`sock` – deskryptor gniazda uzyskany funkcją **`socket`**.

`how` – sposób zamknięcia połączenia np.: **`SHUT_RD, SHUT_WR, SHUT_RDWR`**.

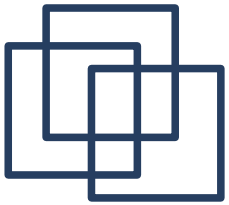
Wartość zwracana: 0 w przypadku powodzenia, -1 w przypadku błędu.

Pliki nagłówkowe **`<sys/socket.h>`**.



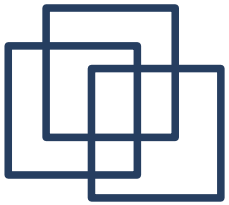
Programowanie z wykorzystaniem gniazd

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#include <stdio.h>
#include <errno.h>
#include <string.h>


int connectsock(char *host, char *service, char *protocol){
  struct hostent *phe;
  struct servent *pse;
  struct protoent *ppe;

  struct sockaddr_in sin;
  int type, s;

  bzero((char*) &sin, sizeof(sin));
  sin.sin_family = AF_INET;
```



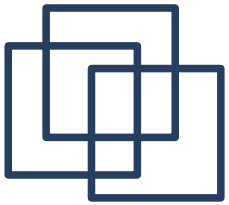
Programowanie z wykorzystaniem gniazd

```
if (pse = getservbyname(service, protocol)){
  sin.sin_port = pse->s_port;
}else if ((sin.sin_port = htons((u_short)atoi(service)))==0){
  fprintf(stderr, "nieznana usluga: %s\n", service);
  return -1;
}

if (phe = gethostbyname(host)){
  bcopy(phe->h_addr, (char *) &sin.sin_addr, phe->h_length);
}else if ((sin.sin_addr.s_addr = inet_addr(host))==INADDR_NONE){
  fprintf(stderr, "nieznany host: %s\n", host);
  return -1;
}

if ((ppe = getprotobyname(protocol))==NULL){
  fprintf(stderr, "nieznany protokol: %s\n", protocol);
  return -1;
}
```



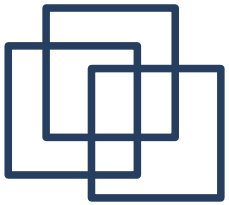
Programowanie z wykorzystaniem gniazd

```
  if (strcmp(protocol, "udp")==0){
    type = SOCK_DGRAM;
  }else{
    type = SOCK_STREAM;
  }

  if ((s = socket(PF_INET, type, ppe->p_proto))<0){
    fprintf(stderr, "nieznany protokol: %s\n", protocol);
    return -1;
  }

  if (connect(s, (struct sockaddr *)&sin, sizeof(sin))<0){
    fprintf(stderr, "nie moge ustanowic polaczenia z %s:%s %s\n",
            host, service, strerror(errno));
    return -1;
  }
  return s;
}
```



Programowanie z wykorzystaniem gniazd

```
#define LINELEN 100

int main(int argc, char **argv){
  int s, n;
  char buf[LINELEN+1];

  if (argc<4){
    printf("sposob uzycia: client serwer serwis protokol\n");
    return 0;
  }
  s = connectsock(argv[1], argv[2], argv[3]);
  if(s<0) return 0;
  while ((n=read(s, buf, LINELEN))>0){
    buf[n] = '\0';
    fputs(buf, stdout);
  }
  if (shutdown(s, SHUT_RDWR)<0){
    fprintf(stderr, "problem z zamknieciem polaczenia: %s\n",
strerror(errno));
    return 0;
  }
  return 1;
}
```



Programowanie z wykorzystaniem gniazd

Przykładowe uruchomienie programu:

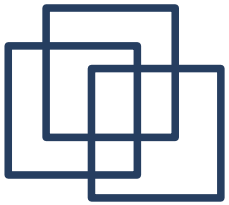
`client theta.uoks.uj.edu.pl daytime tcp`

Wynik działania:

`Tue Oct 14 15:27:25 2008`

lista serwisów (usług, portów): **`/etc/services`**

lista protokołów: **`/etc/protocols`**



Gniazda w Javie

Pakiet (biblioteka): **`java.net`**

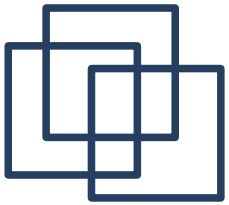
Wybrane klasy:

`Socket` – reprezentuje gnaizdo wykorzystywane do transmisji strumieniowej (TCP), istnieje wersja do transmisji szyfrowanej **`SSLSocket`**;

`DatagramSocket` – gniazdo do transmisji datagramowej (UDP);

`InetAddress` – adres IP, istnieją dwie klasy potomne: **`Inet4Address`** i **`Inet6Address`**;

`URL` – Uniform Resource Locator, identyfikator zasobu w sieci opisujący sposób dostępu do niego np. **`http://www.if.uj.edu.pl/images/misc/ujlogo.gif`**, w ogólnym wypadku stosuje się klasę **`URI`** – Uniform Resource Identyficator.



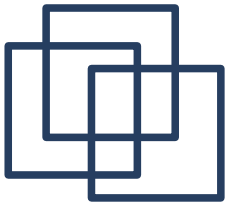
Gniazda w Javie

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

public class Client {
  private static final int LINELEN = 100;
  public static void main(String[] args) {
    byte[] buffer = new byte[Client.LINELEN];
    if (args.length<2)
      System.out.println("wywolanie java Client serwer port");
    try {
      Socket sock = new Socket(args[0], Integer.parseInt(args[1]));
      InputStream in = sock.getInputStream();
      while (in.read(buffer) > 0)
        System.out.println(new String(buffer));
      sock.close();
    }catch (Exception e) { e.printStackTrace(); }
  }
}
```

Wywołanie programu:

javac Client.java

java Client theta.uoks.uj.edu.pl 13



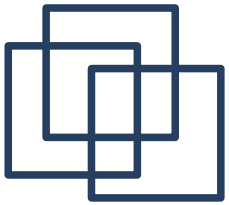
Gniazda w C#

```
using System;
using System.IO;
using System.Net.Sockets;

public class socketexample {
  public static void Main() {
    try {
      TcpClient tcpclnt = new TcpClient();
      tcpclnt.Connect("theta.uoks.uj.edu.pl",13);
      Stream stm = tcpclnt.GetStream();
      byte[] bb = new byte[100];
      int k = stm.Read(bb,0,100);
      for (int i = 0; i<k; i++)
        Console.Write(Convert.ToChar(bb[i]));
      tcpclnt.Close();
    } catch (Exception e) { Console.WriteLine(e.StackTrace); }
  }
}
```

Wywołanie programu:

mcs socketexample.cs

mint socketexample.exe lub **mono socketexample.exe**



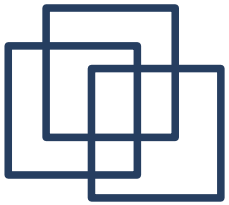
Gniazda w Perlu i PHP

Perl:

```
use IO::Socket;
my $sock =
    new IO::Socket::INET (PeerAddr => 'theta.uoks.uj.edu.pl',
                          PeerPort => '13', Proto => 'tcp', );
die "Nie można utworzyc gniazda: $!\n" unless $sock;
while(<$sock>)
    print $_;
close($sock);
```

PHP:

```
<?php
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
$result = socket_connect($socket, 'theta.uoks.uj.edu.pl', '13');
$out = socket_read($socket, 1024);
echo $out;
socket_close($socket);
?>
```



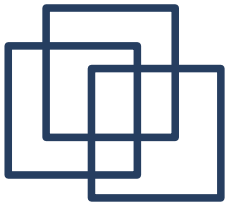
Gniazda w Pythonie i Ruby

Python:

```
import socket

s = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
s.connect(("theta.uoks.uj.edu.pl", 13))
data = s.recv(1024)
print data
s.close()
```

Ruby:

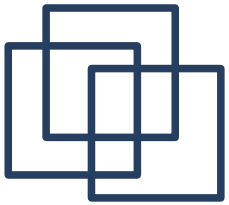
```
require 'socket'
print TCPSocket.open("theta.uoks.uj.edu.pl", "daytime").read
```



Usługa TIME

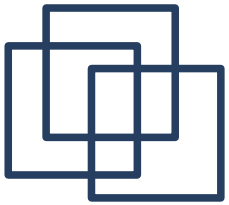
Usługa TIME [RFC 868] służy do synchronizacji czasu pomiędzy komputerami w sieci. Czas jest przesyłany w postaci 32 bitowej liczby całkowitej określającej liczbę sekund od 1 stycznia 1900 roku np. C5 1B 77 8A. Usługa jest dostępna zarówno za pośrednictwem TCP i UDP poprzez port 37.

Inne usługi wykorzystywane do synchronizacji czasu to NTP (*Network Time Protocol*) [RFC 1305] i jego odmiana SNTP (*Simple Network Time Protocol*) [RFC 1769].



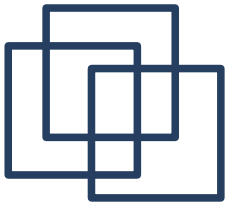
Klient usługi TIME (UDP)

```
#include "connectsock.c"
#define UNIXEPOCH 2208988800ul

int main(int argc, char **argv){
  int s;
  char msg[] = "ktora godzina?";
  time_t now;

  if ((s = connectsock(argv[1], "time", "udp"))<0)   return 0;
  write(s, msg, strlen(msg));
  if (read(s, (char *)&now, sizeof(now))<0) return 0;
  else{
    now = ntohl((u_long)now);
    now -= UNIXEPOCH;
    printf("%s\n", ctime(&now));
  }
  if (shutdown(s, SHUT_RDWR)<0) return 0;

  return 1;
}
```



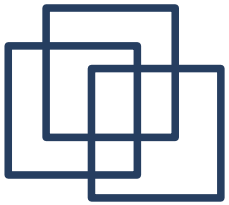
Klient usługi TIME (UDP)

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Date;

public class TimeUDPClient {
    private static final int PORT_TIME = 37;
    private static final String QUERY = "Ktora godzina?";
    private static final int LINELEN = 5;
    private static final long UNIXEPOCH = 2208988800L;

    public static void main(String[] args) {
        try {
            byte[] buffer = new byte[LINELEN];
            DatagramSocket sock = new DatagramSocket();
            DatagramPacket dp = new DatagramPacket(
                    QUERY.getBytes(), 0, QUERY.length(),
                    InetAddress.getByName(args[0]), PORT_TIME);
```



Klient usługi TIME (UDP)

```
        sock.send(dp);
            dp = new DatagramPacket(buffer, LINELEN);
            sock.receive(dp);
            long time = 0;
            int i;
            for (i=0; i<4; i++){
                time *= 256;
                time += (buffer[i] & 255);
            }
            time -= UNIXEPOCH;
            time *= 1000;

            Date d = new Date(time);
            System.out.println(d);
            sock.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```



Usługa NTP/SNTP

Usługa NTP (Network Time Protocol) jest współcześnie wykorzystywana do synchronizacji czasu. NTP wykorzystuje protokół UDP oraz port 127. W praktyce, jeśli duża dokładność synchronizacji nie jest wymagana, używa się uproszczonej wersji protokołu: SNTP [RFC 2030].

Przykladowy pakiet wysyłany przez klienta:

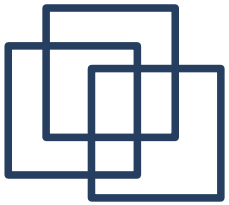
d9: 11	zegar niezsychronizowany
011	wersja protokołu NTP (3)
001	tryb aktywny symetryczny (1). Tryb klienta: 3
00:	wartwa (stratum) komputera klienckiego: nieustalona

Usługa NTP/SNTP

0a:	odstęp między kolejnymi wiadomościami: 2^10 sekund.
fa:	dokładność zegara: 2^(-6) sekundy.
00 00 00 00:	opóźnienie względem źródła czasu: 0.0 (liczba zmiennoprzecinkowa ze znakiem - kropka pomiędzy bitem 15 i 16).
00 01 02 90:	błąd względem źródła czasu 1.01 s.

Usługa NTP/SNTP

00 00 00 00:	identyfikator zegara referencyjnego – brak
00 00 00 00 00 00 00 00:	czas ostatniej aktualizacji lokalnego zegara
00 00 00 00 00 00 00 00:	czas w którym żądanie zostało wysłane
00 00 00 00 00 00 00 00:	czas dotarcia żądania do serwera
cc a8 03 c9 aa 25 88 4f:	przesyłany czas – liczba sekund od 1.01.1900, kropka pomiędzy bitem 31 i 32:



Usługa NTP/SNTP

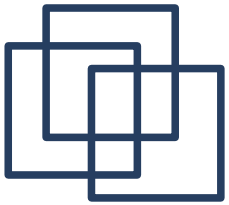
przykładowa odpowiedź serwera:

1a 01 0a e3 00 00 00 00

00 00 00 00 41 43 54 53

cc a8 04 2b 12 b4 49 7b

cc a8 03 c9 aa 25 88 4f:	czas otrzymany od klienta (T1)
cc a8 04 42 4d 12 73 fa:	czas dotarcia żądania do serwera (T2)
cc a8 04 42 4d 13 38 d7:	aktualny czas (T3)



Usługa NTP/SNTP

Po odebraniu odpowiedzi od serwera, klient ustala aktualny czas w oparciu o:

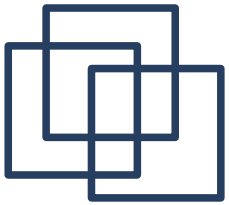
- czas wysłania żądania T1 (wg lokalnego zegara),
- czas odebrania odpowiedzi przez serwer T2 (wg zegara serwera),
- czas otrzymany z serwera T3 (wg zegara serwera)
- czas otrzymania odpowiedzi T4 (wg lokalnego zegara)

Różnicę czasów klient oblicza ze wzoru:

$$T = ((T2 – T1) + (T3 – T4)) / 2.$$

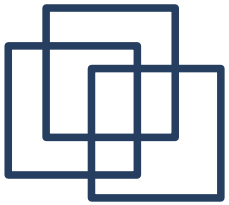
Opóźnienie wyraża się zależnością:

$$D = (T4 – T1) – (T3 – T2).$$



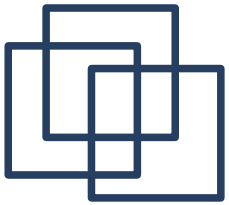
Usługa ECHO

Usługa ECHO [RFC 347] odsyła klientowi wszystkie otrzymywane od niego dane. Interakcja trwa dopóki klient nie zakończy transmisji. Usługa ECHO jest wykorzystywana głównie do celów diagnostycznych, podobnie jak komenda ping. W przeciwieństwie do niej korzysta z protokołu TCP lub UDP. Do komunikacji wykorzystuje port 7.



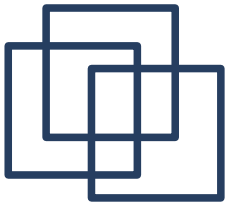
Klient usługi ECHO (TCP)

```
int main(int argc, char **argv){
  int s, i, n1, n2;
  char buf[LINELEN+1];

  if ((s = connectsock(argv[1], "echo", "tcp"))<0)
   return 0;

  for(i=2; i<argc; i++){
    n1 = write(s, argv[i], strlen(argv[i]));
    while (n1>0 && (n2=read(s, buf, LINELEN))>0){
      buf[n2] = '\0';
      n1 -= n2;
      fputs(buf, stdout);
    }
    fputs("\n", stdout);
  }

  if (shutdown(s, SHUT_RDWR)<0)
    return 0;
  return 1;
}
```



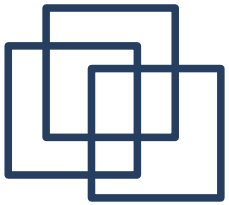
Klient usługi ECHO (UDP)

```
int main(int argc, char **argv){
  int s, i, n;
  char buf[LINELEN+1];

  if ((s = connectsock(argv[1], "echo", "udp"))<0)
   return 0;

  for(i=2; i<argc; i++){
    write(s, argv[i], strlen(argv[i]));
    if ((n=read(s, buf, LINELEN))>0){
      buf[n] = '\0';
      fputs(buf, stdout);
    }
    fputs("\n", stdout);
  }

  if (shutdown(s, SHUT_RDWR)<0)
    return 0;
  return 1;
}
```



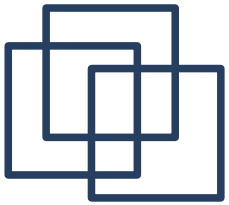
Klient usługi ECHO (TCP)

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class EchoTCPClient {
    private static final int PORT_ECHO = 7;
    private static final int LINELEN = 100;

    public static void main(String[] args) {
        byte[] buffer = new byte[LINELEN];
        if (args.length<2){
            System.out.println("wywolanie java EchoTCPClient
                                serwer argument1 argument2 ...");
            return;
        }
```



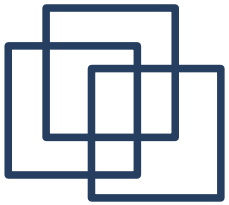
Klient usługi ECHO (TCP)

```
    try {
            Socket sock = new Socket(args[0], PORT_ECHO);
            OutputStream os = sock.getOutputStream();
            InputStream in = sock.getInputStream();
            int i, k, j;
            for (i=1; i<args.length; i++){
                os.write(args[i].getBytes());
                for(j=0; j<args[i].length(); j+=k){
                    k = in.read(buffer);
                    System.out.print(new String(buffer, 0, k));
                }
                System.out.println();
            }
            sock.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



Klient usługi ECHO (UDP)

```
public class EchoUDPClient {
    private static final int PORT_ECHO = 7;
    private static final int LINELEN = 100;
        public static void main(String[] args) {
        try {
            byte[] buffer = new byte[LINELEN];
            DatagramSocket sock = new DatagramSocket();
            DatagramPacket dp;
            for (int i=1; i<args.length; i++){
                dp = new DatagramPacket(args[i].getBytes(), 0,
                       args[i].length(),
                       InetAddress.getByName(args[0]), PORT_ECHO);
                sock.send(dp);
                dp = new DatagramPacket(buffer, LINELEN);
                sock.receive(dp);
                System.out.println(new String(dp.getData(), 0,
                                              dp.getLength()));
            }
            sock.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```



Podsumowanie

1. Programowanie z wykorzystaniem gniazd - kontynuacja

- gniazda w Javie,
- transmisja TCP i UDP,
- protokół NTP / SNTP.