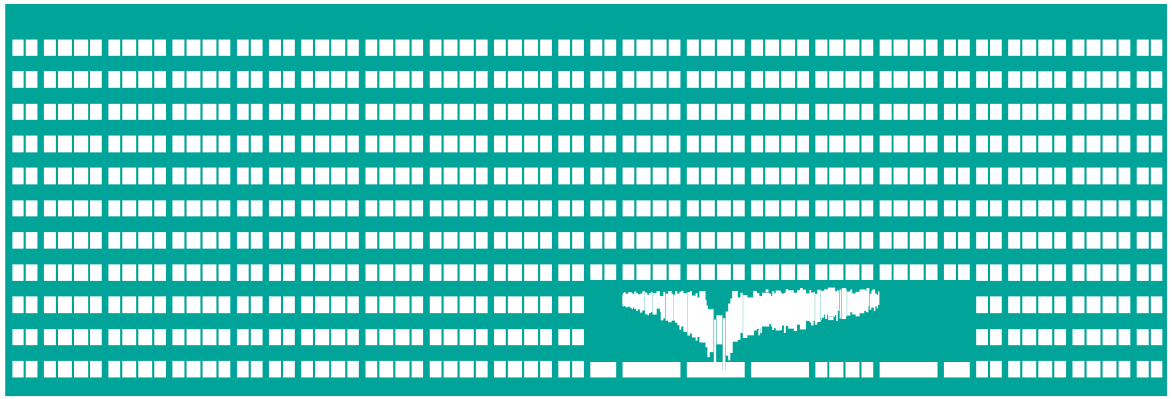
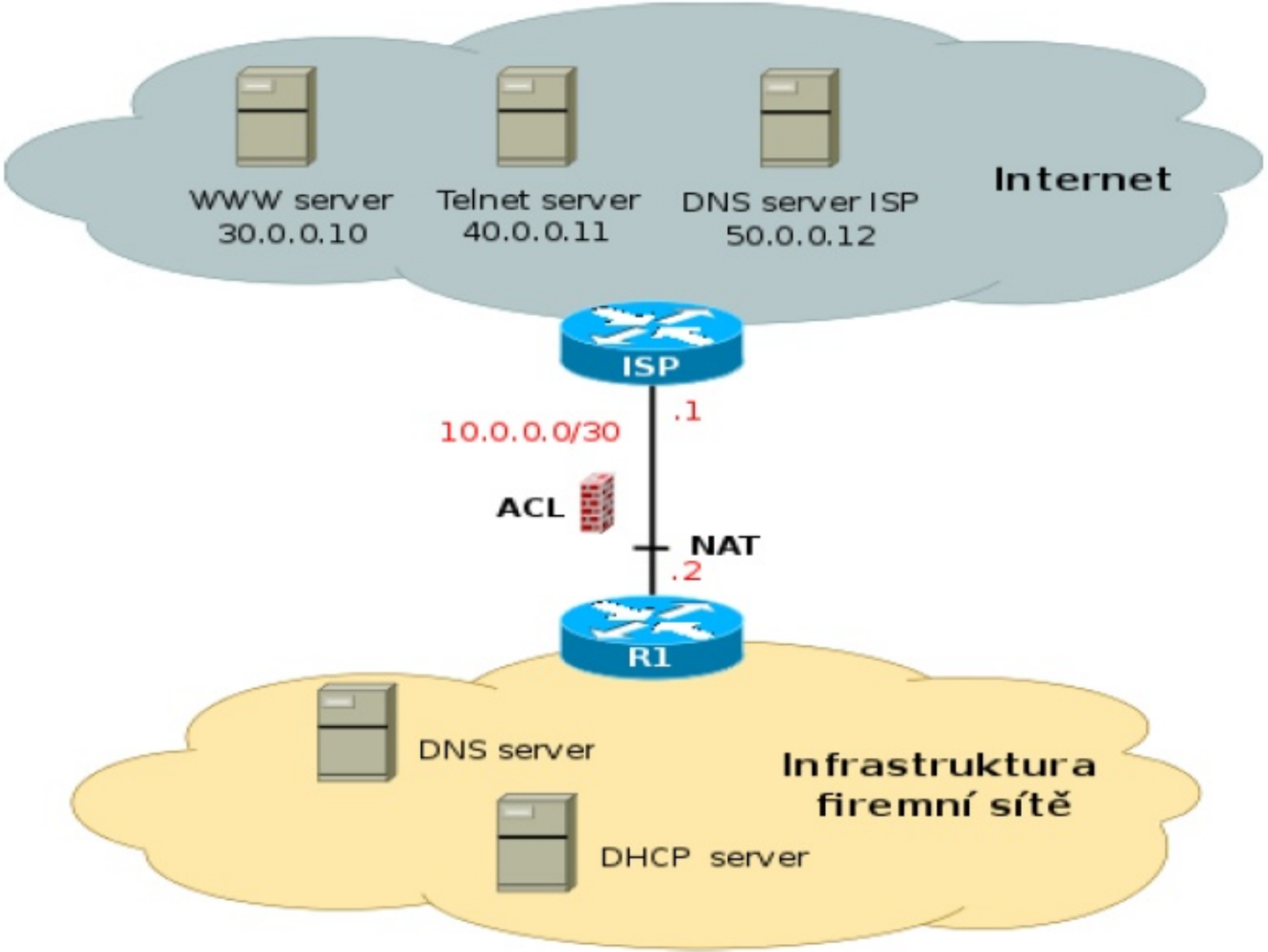
VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Rozhraní Sockets Java a C (BSD)



Počítačové sítě 3. cvičení

Semestrální projekt (1)



Semestrální projekt (2)

Struktura projektu:

- Adresní plán a konfigurace VLAN
- Směrování a NAT
- DNS server
- DHCP server
- Zabezpečení sítě – ACL

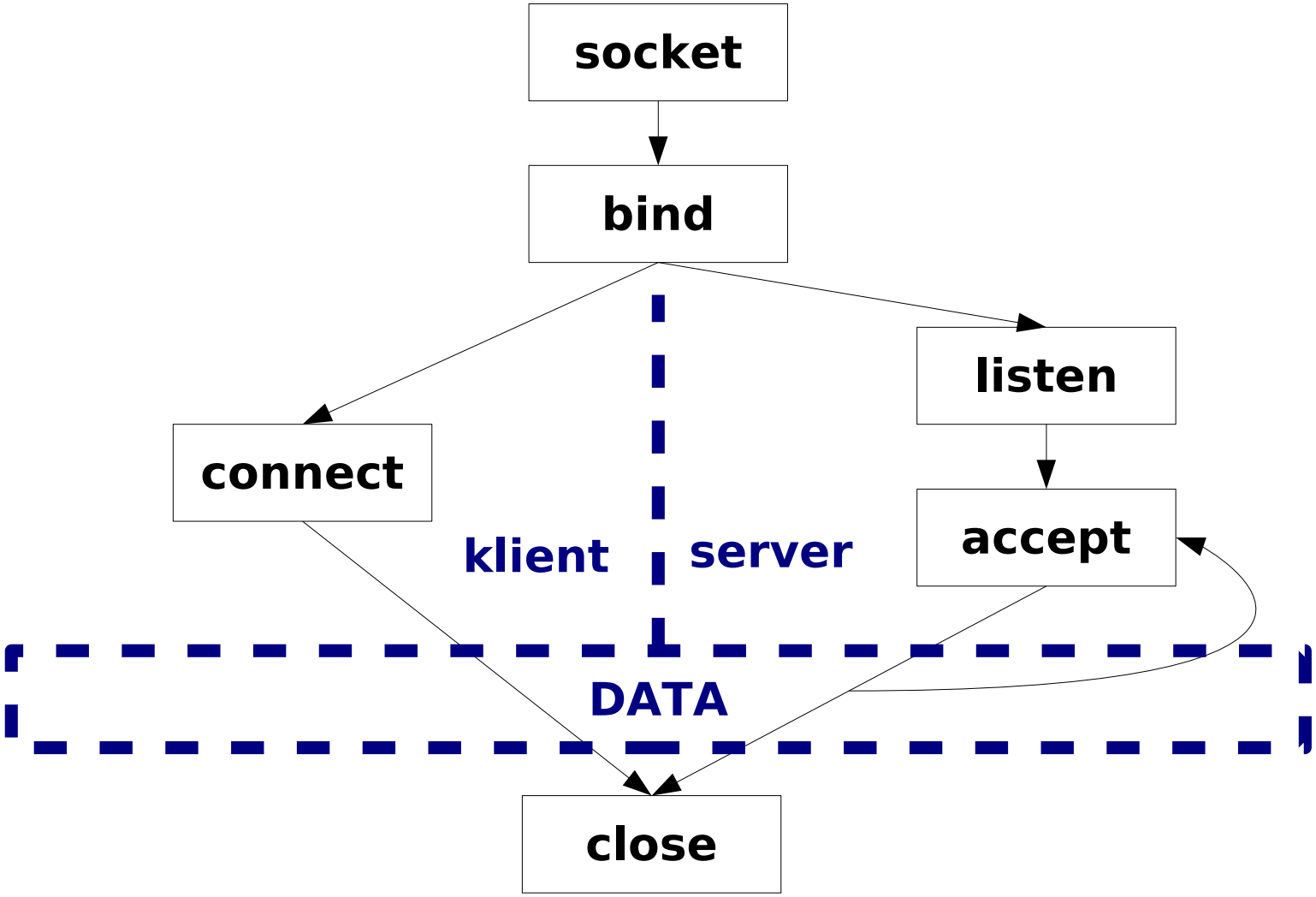
- Podmínky, co musí obsahovat jednotlivé části přesně zadány

Úvodní informace

- **BSD Sockety** – práce se síťovým spojením jako se soubory (POSIX)
- Protokoly rodiny **IP** – **TCP** a **UDP**, identifikace IP adresou a číslem portu

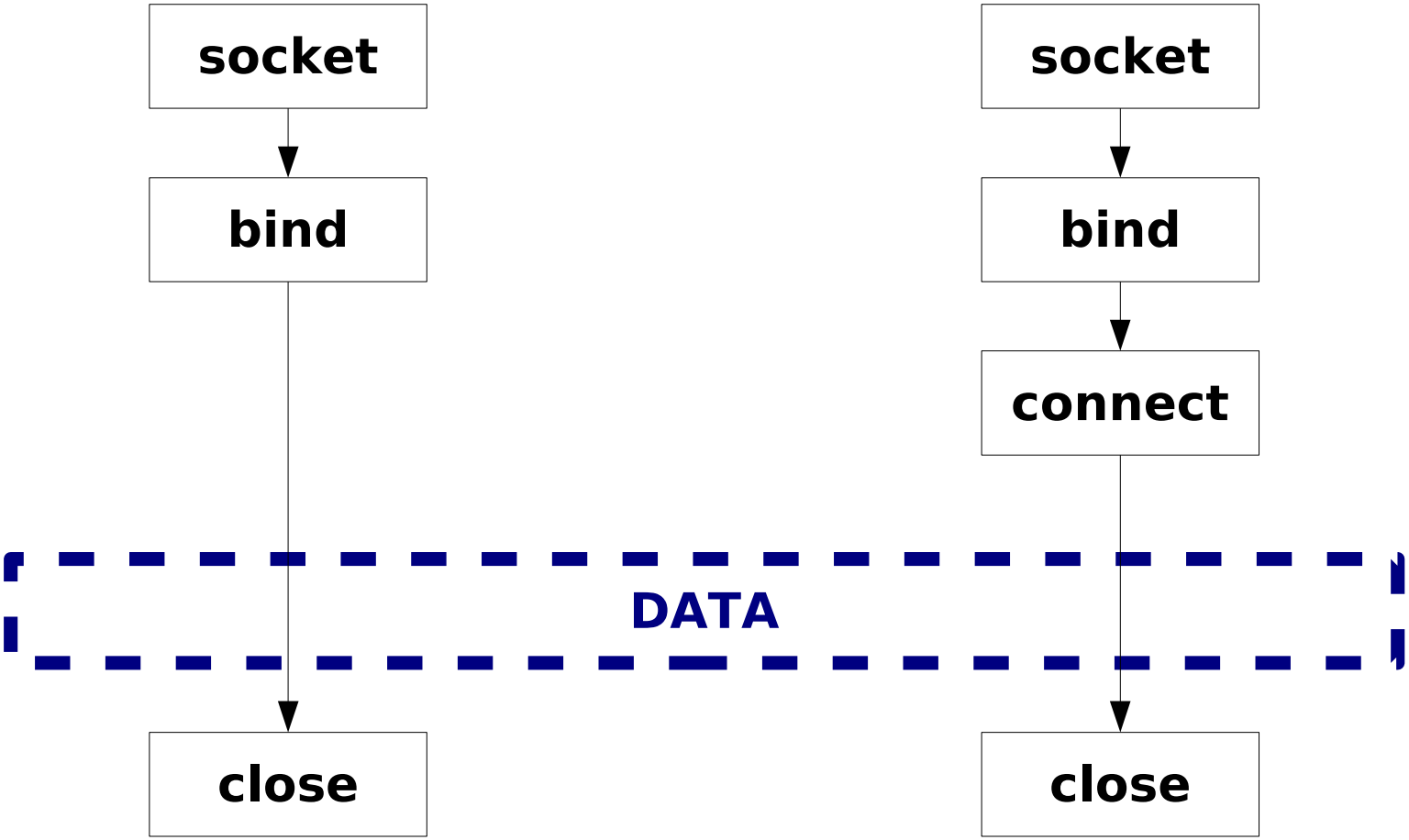
  - **TCP** (Transmission Control Protocol) - spolehlivý logický kanál. Před samotnou komunikací se naváže spojení, veškerá odeslaná data jsou druhou stranou potvrzována, na konci je nutno spojení ukončit (uzavřít).
  - **UDP** (User Datagram Protocol) - Nedochází k navázání spojení, data jsou odeslána na stanovenou IP adresu a daný UDP port a nevíme, zda data dorazila, zda nešlo k jejich zdvojenému doručení (není-li vyplněn kontrolní součet, nevíme zda v pořádku).

Schéma práce se sockety TCP



Data: send/recv, read/write

Schéma práce se sockety UDP



Data: sendto/recvfrom

Data: send/recv, read/write

Potřebné knihovny

- V Javě jsou potřebné třídy v
  - Java.net – Sockety
  - java.io – Vyjímky, streamy

- V C/C++ se připojují tyto hlavičky
  - (UNIX):
    - netdb.h
    - arpa/inet.h
    - sys/time.h
    - sys/socket.h
    - netinet/in.h

(Windows):

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>

#pragma comment(lib, "Ws2_32.lib")
```

Vytvoření socketu

- K vytvoření socketu v Javě slouží
  - `new Socket([adresa, c_port])`
  - `new ServerSocket(z_port [, bl])`
  - `new DatagramSocket([z_port])` – UDP
  - `new MulticastSocket([z_port])` – UDP multicasty
- K vytvoření socketu v C/C++ slouží funkce
  - `int socket(int domain, int type, int protocol)`
    - **domain** určuje způsob komunikace (**PF_INET**: IP)
    - **type** je typ socketu
      - **SOCK_STREAM**: TCP
      - **SOCK_DGRAM**: UDP
    - **protocol** není v našem případě třeba využit a je tudíž 0.

Pojmenování socketu

- V Javě je třeba specifikovat konkrétní port
  - `public void bind(SocketAddress bindpoint)`
- V C/C++ pojmenuje socket funkce
  - `int bind(int sck, struct sockaddr* name, int namelen)`
    - **sck** – deskriptor socketu (z funkce socket)
    - **name** – struktura sockaddr_in s IP adresou socketu a číslem portu
      - **sin_family** – **AF_INET** – protokol IPv4
      - **sin_addr.s_addr** – IP adresa (**INADDR_ANY**)
      - **sin_port** – port na počítači
        - Porty do 1024 jsou rezervováno pro roota !
    - **namelen –** velikost struktury – `sizeof(sockaddr_in))`

Připojení klienta

- V Javě lze zadat adresu v konstruktoru (TCP), nebo
  - `void **connect**(SocketAddress endpoint)`
    - Pro UDP lze použít třídy **DatagramPacket** a metod **send** a **receive** třídy **DatagramSocket**
- V C/C++ se připojíme k serveru funkcí
  - `int **connect**(int sck, struct sockaddr* name, int namelen)`
    - **name** – jako u bind(), ale jde o cílovou adresu.
    - V případě UDP není **connect** nutný, jsou-li používány funkce **sendto** a **recvfrom**

Provoz serveru

- V Javě existují metody **bind** (odpovídá **bind+listen**) a **accept** ve třídě **ServerSocket**
  - Uvedeme č. portu a max počet požadavků, čekajících ve frontě (backlog) v konstruktoru nebo `void **bind**(SocketAddress endpoint, int backlog)`
  - Metoda `void **accept**()` - otevření příchozího socketu
- V C/C++ slouží k naslouchání na socketu funkce
  - `int **listen**(int sck, int backlog)`
    - **backlog** – max počet požadavků, čekajících ve frontě
  - `int **accept** (int sck, struct sockaddr* addr, int* addrlen)`
    - Zablokuje se do přijetí požadavku, do addr je uložena IP adresa klienta (často po accept následuje int **fork()**)

Zasílání dat

- V Javě využíváme reader/writer vytvořený nad streamy z **getInputStream()**, **getOutputStream()** nebo třídy **DatagramPacket**
.
- C/C++ nabízí několik způsobů zasílání dat
  - POSIXové souborové funkce **read**/**write**
    - `int read(int sck, char* buf, unsigned buflen)`
    - `int write(int sck, char* buf, unsigned buflen)`
  - Fukce send/recv
    - `int send(int sck, char* buf,int buflen, int flags)`
    - `int recv(int sck, char* buf,int buflen, int flags)`
      - **flags** mohou dále specifikovat data (urgentní, ...)
  - Funkce **sendto**/**recvfrom** pro datagramy bez connect
    - `int sendto(<jako send>, struct sockaddr* to, int tolen)`
    - `int recvfrom(<jako recv>, struct sockaddr* from, int *fromlen)`

Ukončení spojení

- V Javě lze slouží k ukončení spojení metoda
  - `void close()`
    - Na Socketu resp. na Input/Output streamu

- V C/C++ slouží k ukončení spojení funkce
  - `int close(int sck)`
    - klasické uzavření spojení
  - `int shutdown(int sck, int how)`
    - how = 0 – uzavřít pro příjem
    - how = 1 – uzavřít pro vysílání
    - how = 2 – reset spojení

Nastavení parametrů socketu

- Nastavování v Javě pomocí metod **get**/**set** tříd *Socket (např. setSoTimeout)
- Nastavování v C/C++
  - Funkce pro sockety
    - `int **getsockopt**(int sck, int lvl, int optname, char* optval, int* optlen)`
    - `int **setsockopt**(int sck, int lvl, int optname, char* optval, int optlen)`
      - **lvl**: SOL_SOCKET, IPPROTO_TCP, ...
  - Funkce pro práci se soubory (fcntl.h, unistd.h)
    - `int **fcntl**(int sck, int cmd, long arg)`
  - Funkce pro práci s V/V zařízeními (sys/ioctl.h)
    - `int **ioctl**(int d, int request, ...)`

Poznámky k Javě

- „Non-blocking“ sockety nelze přímo použít, ale na Socketu lze volat metodu
  - **`setSoTimeout`**`(milisekund)`
    - Pokud blokující operace (čtení, zápis) neskončí včas→výjimka **java.net.SocketTimeoutException**
      - Socket není zneplatněn, může následovat nový pokus o provedení operace po ošetření výjimky
- Převod doménového jména na InetAddress
  - `InetAddress ia=InetAddress.`**`getByName`**`(jméno)`
    - `ia.getHostAddress()` – vrací IP adresu jako řetězec

Non-blocking sockety v C/C++

- Nemůžeme-li v programu čekat na příchod dat:
  - ```
    int flag=0;
    flag=fcntl(sck, F_GETFL, 0);
    if(flag == -1) { flag = 0; }
    fcntl(sck, F_SETFL, flag | O_NONBLOCK);
    ```
- Staré systémy nemající O_NONBLOCK:
  - ```
    int flag=1;
    ioctl(sck, FIOBIO, &flag);
    ```
- Blokující funkce pak vracejí místo zablokování -1
  - **errno** nastaveno na EWOULDBLOCK

Řešení timeoutu v C

```
#include <unistd.h>
#include <signal.h>
#define MAX_REP 10                    //Max. počet retransmisí
int repeat=0;                         //Aktuální počet retransmisí
void alarmed(int signo) {
  if (++repeat > MAX_REP) {           //Překročeno MAX_REP
                            //Informovat, že vysílání selhalo
    signal(SIGALRM, NULL); exit(-1); //Ukončit
  } else {                            //Provést retransmise
    alarm(1);                         //Znovu nastavit timeout
    }
  }
…                                        //Odeslat data (1. pokus)
  alarm(1); signal(SIGALRM,alarmed); //Nastavit timeout
          // Příjem potvrzení (pokud nepřijde během 1s, bude
            vyvolán SIGALRM)
  repeat=0; alarm(0);                    //Zrušit timeout
```

Pomocné funkce v C

- Převody mezi formáty reprezentací
  - **htonl, htons** – převádí endian z PC→síť (long, short)
  - **ntohl, ntohs** – převádí endian z síť→PC (long, short)
  - `unsigned long` **`inet_addr`**`(char* cp)`
    - převod z tečkové notace
  - `char*` **`inet_ntoa`**`(struct in_addr in)`
    - převod do tečkové notace
- Práce s doménovými jmény
  - `int` **`gethostname`**`(char* name, int namelen)`
    - název lokální stanice
  - `struct hostent*` **`gethostbyname`**`(char* name)`
    - adresa stanice je v **`char * he->h_addr`**
  - `struct hostent*` **`gethostbyaddr`**`(char* addr, int len, int type)`
    - doménové jméno z IP adresy **`char* he->h_name`**

Alternativní prog. jazyky

- Tutoriál pro (BSD) Winsock pro C/C++ ve Windows

https://docs.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock

- Python
  - Klientské sokety: https://docs.python.org/3/library/socket.html
  - Server https://docs.python.org/3/library/socketserver.html

- C# (.NET)
  - Obecně: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.socket
  - Ukázka (synchronního) TCP klienta – 1. část zadání: https://docs.microsoft.com/en-us/dotnet/framework/network-programming/synchronous-client-socket-example
  - Ukázka UDP příjemce – 2. část zadání https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-udp-services

Ukázka příkladů

- Detailní popis a příklady (na konci textu)
  - C/C++
    - http://www.cs.vsb.cz/grygarek/PS/sockets.html
  - Java
    - http://www.cs.vsb.cz/grygarek/PS/dosys/inprise/net/examples
      - Ex1 – TCP
      - Ex2 – UDP
      - Ex3 – multicasty
      - Ex4 – získání dat pomocí URL(třída URLConnection)

Java – připojení přes TCP Socket

```
try {
  s=new Socket(host,port);
  BufferedReader sis = new BufferedReader(new InputStreamReader(System.in));
  BufferedWriter os = new BufferedWriter(
                                        new OutputStreamWriter(s.getOutputStream()));
  BufferedReader is = new BufferedReader(
                                        new InputStreamReader(s.getInputStream()));
  String l;
  do {
    System.out.println("Type a line to send to server.");
    l=sis.readLine();
    os.write(l);
    os.newLine();
    os.flush();
    System.out.println("Server: " + is.readLine());
  } while (!l.equals("exit") && !l.equals("down"));
  s.close();
} catch (IOException e) { System.out.println(e); }
```

Java – TCP Server

```
…
try {
 ServerSocket s=new ServerSocket(port);
 Socket cs;
 do {
   cs=s.accept();
   BufferedReader is = new BufferedReader
                       (new InputStreamReader(cs.getInputStream()));
   BufferedWriter os = new BufferedWriter
                       (new OutputStreamWriter(cs.getOutputStream()));
   do {
    msg=is.readLine();
    os.write(String.valueOf(msg.length()));
    os.newLine(); os.flush();
   }  while (!msg.equals("exit") && !msg.equals("down"));
   cs.close();
 } while (!msg.equals("down"));
 s.close();
} catch (IOException e) { System.out.println(e); }
```

Java – MultiThreading TCP Server

Vylepšuje předchozí příklad pomocí vláken, můžeme zpracovat více požadavků zároveň

```
public class MyApplication implements Runnable {
      protected Socket cs;   static ServerSocket s;
      static int port=8000;
      public static void main(String[] args) {
        Socket cs;
        try {
            s=new ServerSocket(port);
            do {cs=s.accept(); new Thread(new MyApplication(cs)).start();
            } while (true);
        } catch (IOException e) {if(!e instanceof SocketException){System.out.println(e);}}
      }
      public MyApplication(Socket cs) {this.cs=cs;}
   public void run() {
     /* Zde patří fialový text z předchozího slide */
      if (msg.equals("down")) s.close(); // Uzavře server socket a ukončí aplikaci
   }
}
```

Java – UDP Client

```
String data;  //Data získáme později, např. čtením ze  System.in
int port=8000;
String server="www.cs.vsb.cz";
…
try {
  DatagramSocket s=new DatagramSocket();
  DatagramPacket p = new DatagramPacket(data.getBytes(), data.length(),
                                        InetAddress.getByName(server), port);
   s.send(p);
   s.receive(p);
   reply=new String(p.getData(),0,p.getLength());
   System.out.println("Reply arrived from "+ p.getAddress() +" : "+ p.getPort()+
                                                    " > " + reply);
   s.close();
} catch (IOException e) { System.out.println(e); }
```

Java – UDP Server

Pro datagramové servery není samostatná třída, není ji potřeba

```
…
try {
DatagramSocket s=new DatagramSocket(port);
DatagramPacket p;
String msg;
do {
  p=new DatagramPacket(new byte[512], 512);
  // ^ vzhledem k fixní velikosti bufferu pokaždé alokujeme znovu
  s.receive(p);
  msg = new String(p.getData(),0,p.getLength());
  System.out.println("DG from " + p.getAddress() + " : " + p.getPort() + " > " + msg);
  p.setData(msg.toUpperCase().getBytes());
  p.setLength(msg.length());
  s.send(p);
} while (!msg.equals("down"));
s.close();
} catch (IOException e) { System.out.println(e); }
```

Úloha – hromadný chat

- Poslání zprávy na „talk“ server přes **TCP/8000**
- Příjem zpráv na **UDP/8010** ~~z broadcastu~~ (nyní kvůli on-line výuce unicastem na IP adresu z TCP spojení)
- Zpráva
  - max. 255 ASCII znaků
  - ukončeno <LF> (tj. \n)
  - Při posílání UDP zpět je zprávě předřazeno 9 bytů :
    - 4 byte - IP adresa odesílatele (int, formát lo-hi)
    - 4 byte - htonl(time_t)
    - 1 byte - délka následujícího řetězce (0-255).
    - | 127 | 0 | 0 | 1 | 0x4A | 0xAE | 0x1A | 0x30 | 3 | SSS |