Socket

DIS 2011/2012

Distribuirani informacijski sustavi

- Distribuirani (raspodijeljen, razdijeljen) informacijski sustavi u odnosu na neraspodijeljeni (undistributed, non-distributed, monolithic, nerazdijeljen) informacijski sustav se razlikuje u tome što u distribuiranom informacijskom sustavu postoje zasebni (raspodijeljeni) procesi (posebni tip distribuiranih sustava su sustavi raspodijeljeni na threadove, a ne na procese izvršavanja) koji rade zajedno na neki način i međusobno komuniciraju da bi distribuirani informacijski sustav pružio funkcionalnost koja je predviđena.
- Komunikacija između procesa (inter-proces komunikacija (*inter-process communication - IPC*) (vs. inter-proces komunikacija) se ostvaruje korištenjem različitih tehnologija (rješenja).

Distribuirani informacijski sustavi

- Već smo radili sa MySQL bazom podataka (danas su baze podataka tipični distribuirani informacijski sustav) koja se sastoji od MySQL servera, a aplikacije koje mi razvijamo i koje koriste bazu podataka se ustvari spajaju kao klijenti na MySQL server.

Distribuirani informacijski sustavi

- Taksonomija (to je klasifikacija (podjela na određene dijelove, tipove,.....) nečega) distribuiranih informacijskih sustava (ovo je samo jedna od mogućih podjela):

1. Mrežno distribuirani (na više računala) (postoje različiti tipovi mreža – i hardverski i softverski – danas de facto se uzima TCP/IP mrežna arhitektura)
2. Lokalno distribuirani (na jednom računalu)

Distribuirani informacijski sustavi

- Još jedna podjela distribuiranih informacijskih sustava može biti prema arhitekturi sustava (to je određena abstrakcija aplikacije) i de facto dvije osnovne arhitekture distribuiranih sustava su:

1. Klijent-server
2. Peer-to-peer

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. (Software Architecture in Practice, Second Edition**,** Len Bass, Paul Clements, Rick Kazman (2003))

Arhitektura

- Osnovno svojstvo klijent-server arhitekture ja da procesi u komunikaciji nisu ravnopravni. Ravnopravnost se najčešće očituje kroz mogućnost postavljanja zahtjeva za komunikacijom. Npr. soket server “čeka” na dolazne konekcije soket klijenta.
- Osnovno svojstvo peer-to-peer arhitekture ja da su procesi u komunikaciji ravnopravni. Ova arhitekura se rjeđe koristi.

Distribuirani informacijski sustavi

- Sljedećih nekoliko predavanja ćemo se baviti mrežno distribuiranim informacijskim sustavima.
- Pod pojmom mreže uzimamo bilo kakvu organizaciju zasebnih računala koja mogu komunicirati. Komunikacija je moguća kroz osiguravanje fizičke konekcije računala (žica, radio valovi,...), ali i komunikacijskih protokola. Postoje različite tehnologije, a mi ćemo se ovdje ograničiti na Internet mrežu sa tehnologijama i komunikacijskim protokolima koji se koriste na Internetu.

Komunikacijski protokoli

- Znači ograničavamo se na TCP/IP obitelj komunikacijskih protokola (IP, ICMP, TCP, UDP) koji svojim karakteristikama utječu na tehnologije mrežno distribuiranih informacijskih sustava.
- Određene tehnologije mrežno distribuiranih informacijskih sustava se čak oslanjaju na protokole više razine, pa npr. web servis i web klijent kao komunikacijski protokol koriste HTTP.

Distribuirani informacijski sustavi

- Jedna od taksonomija tehnologija mrežno distribuiranih sustava:
  1. Soket (socket)
  2. RPC (remote procedure call)
  3. ORPC (object remote procedure call)
  4. Message-oriented tehnologija

- Sve ove tehnologije su klijent-server arhitekture.

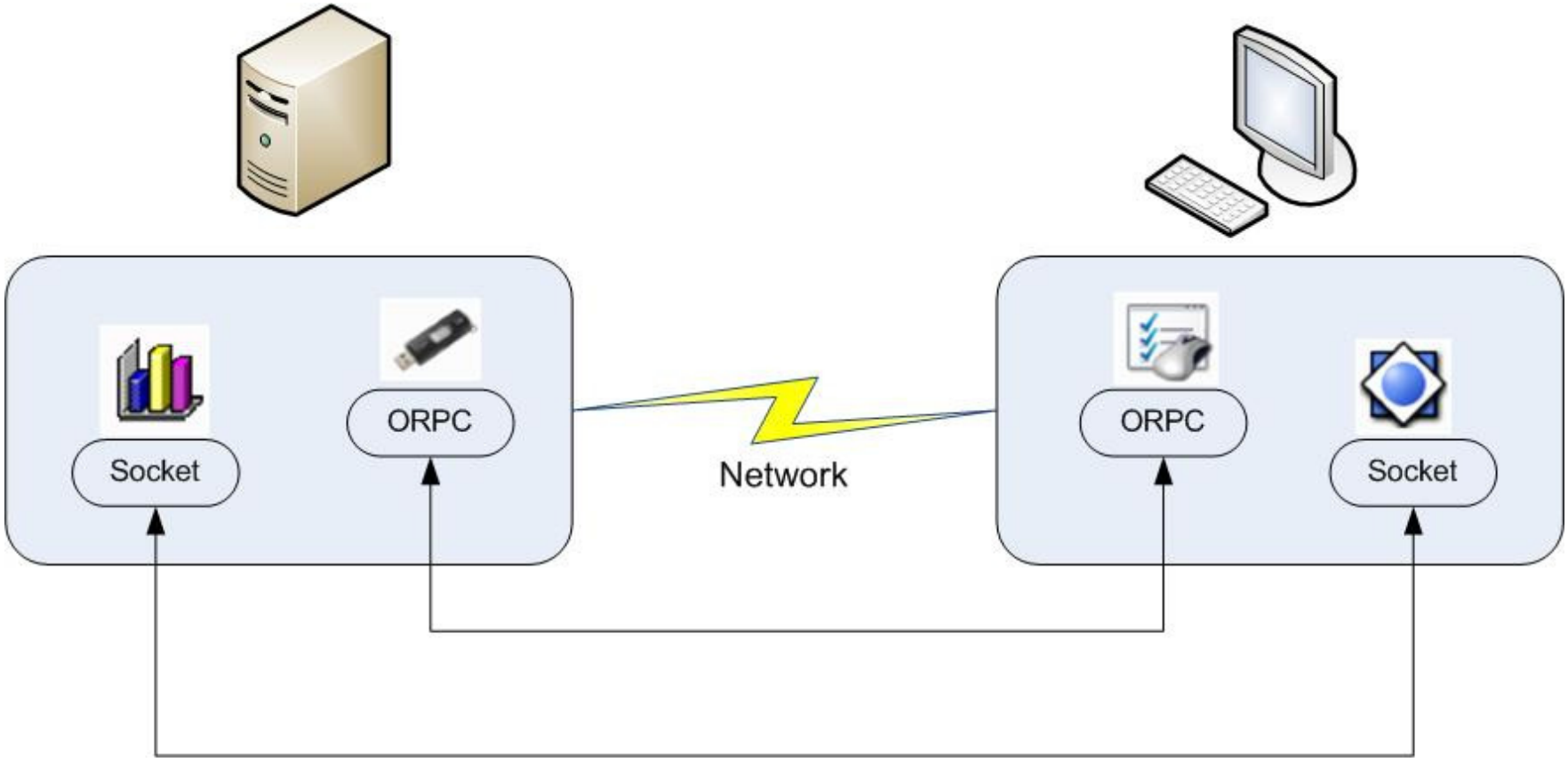
Soket tehnologija

- Osnovno svojstvo ove tehnologije ja da dva procesa koja komuniciraju međusobno razmjenjuju niz bajtova (byte stream) tj. informacije koje se razmjenjuju između dva procesa preko socket tehnologije nisu formatirane unutar same tehnologije već formatiranje podataka (iz i u niz bajtova) moraju napraviti sami procesi prema svojoj definiranoj logici.

Komunikacija između procesa

- Komunikacija između dva procesa mora se odvijati na “istoj” razini (analogija sa ISO-OSI mrežnim modelom) tj. prema “istoj” tehnologiji.
- To znači da vam jedan proces ne može komunicirati preko socketa, a drugi preko ORPC-a i da komunikacija između ta dva procesa bude uspješno ostvarena.

Komunikacija između procesa



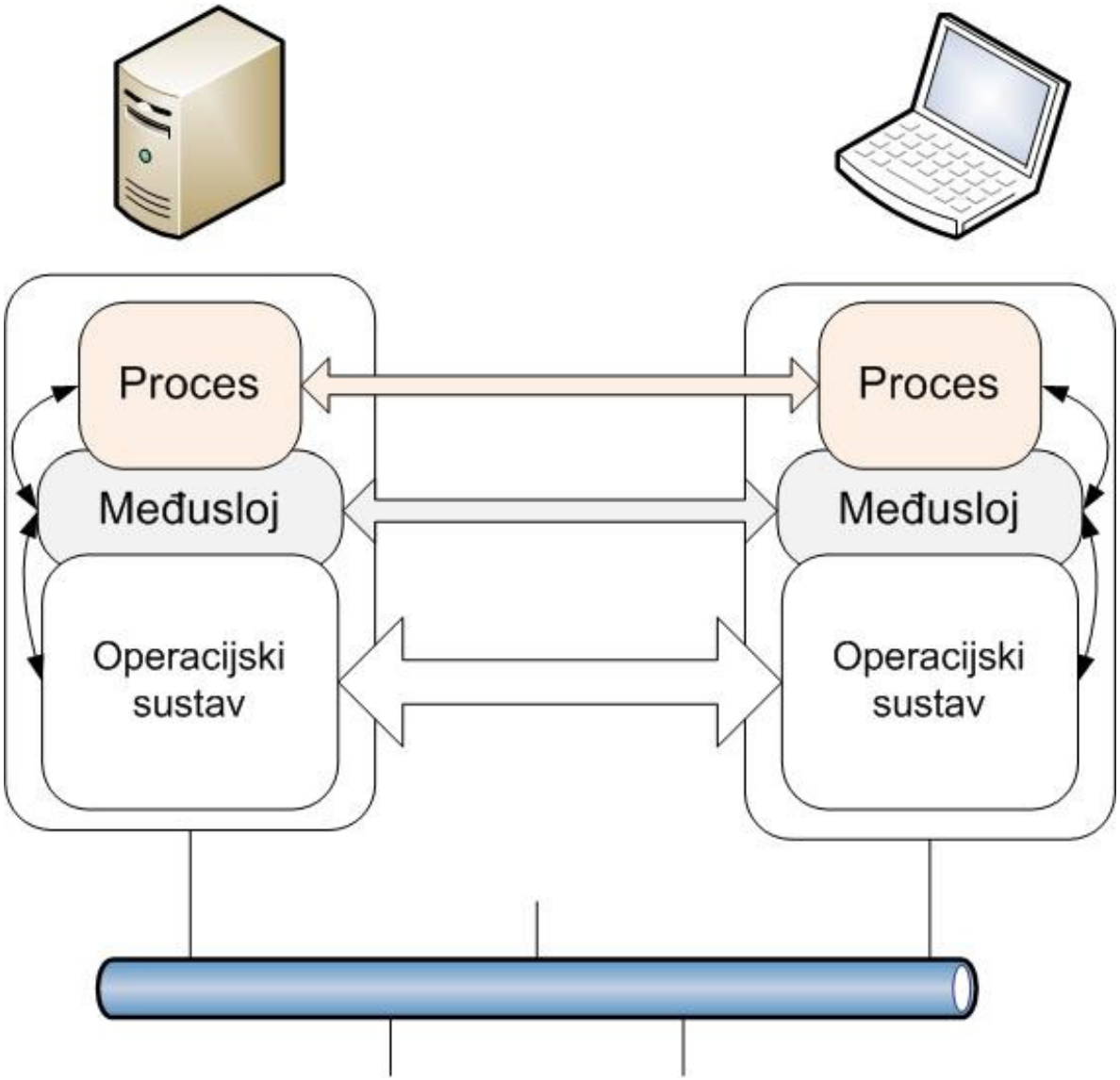
Međusloj (Middleware)

- U praksi se mrežna komunikacija odvija korištenjem tzv. međusloja (srednjeg sloja, međuprograma) (*middleware*).
- Međusloj se može definirati kao softver koji omogućava komunikaciju i upravljanje podacima u distribuiranim aplikacijama (izvor www.wikipedia.org).
- Ili kao softver koji posreduje između aplikacije i mreže (foldoc.org/middleware).
- Ili kao sofverski sloj koji leži između operacijskog sustava i aplikacije na svakoj strani sustava (middleware.objectweb.org)

Međusloj

- Međuslojem se nazivaju samo softverske komponente koje se koriste za RPC, ORPC i MO (tzv. MOM message-oriented middleware), mada se i soketi oslanjaju na softverski međusloj samo je obično soket međusloj sastavni (integrirani) dio operacijskog sustava i kao što smo već naveli ne uključuje nikakvo formatiranje podataka koji procesi razmjenjuju.

Međusloj

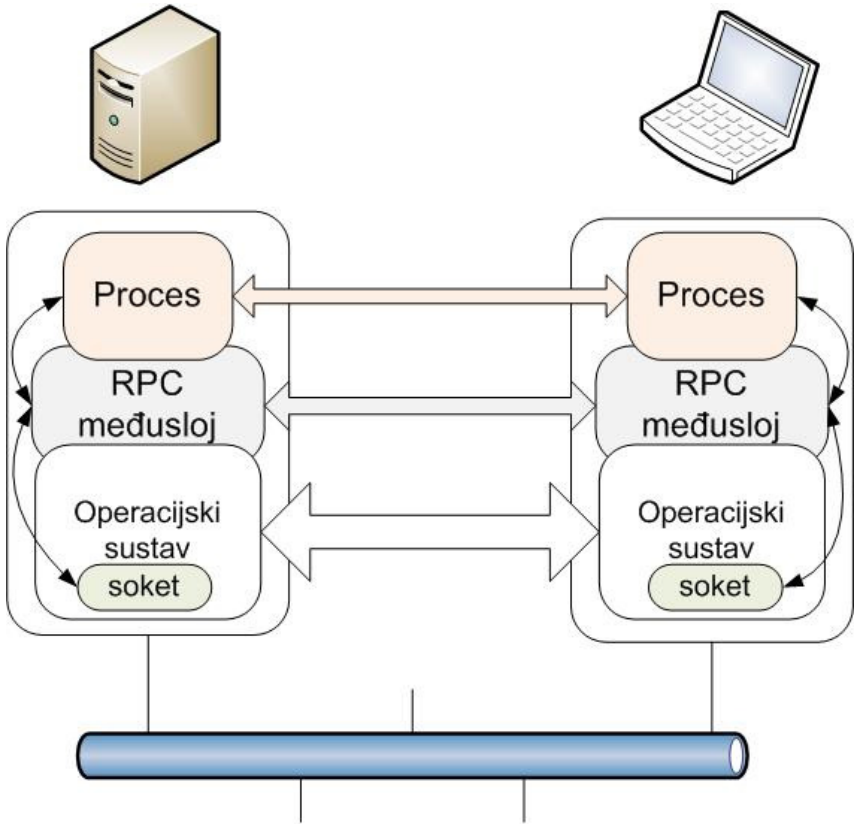

Proces
Međusloj
Operacijski sustav
Proces
Međusloj
Operacijski sustav

RPC tehnologija

- Osnovno svojstvo ove tehnologije je da dva procesa međusobno komuniciraju pozivajući procedure (funkcije, metode). Znači jedan proces može pozvati proceduru koja je implementirana u drugom procesu.
- Procesu pozivatelju "izgleda" kao da se "udaljena" procedura izvodi lokalno, ali se ona ustvari izvodi u drugom procesu. Proces pozivatelj preda procesu izvršitelju parametre procedure, ovaj izvrši proceduru i vrati nazad procesu pozivatelju povratnu vrijednost procedure.

RPC tehnologija

- U implementaciji se RPC međusloj oslanja na sokete za komunikaciju, a sam međusloj implementira formatiranje podataka i sl. funkcionalnosti koje su potrebne za pozive procedure npr. upravljanje greškama.



ORPC tehnologija

- Osnovno svojstvo ove tehnologije je da dva procesa međusobno komuniciraju pozivajući metode na objektima procesa. Znači jedan proces može pozvati metodu koja je implementirana u objektu u drugom procesu.
- Procesu pozivatelju “izgleda” kao da se “udaljena” metoda na udaljenom objektu izvodi lokalno, ali se ona ustvari izvodi u drugom procesu. Proces pozivatelj predaje procesu izvršitelju “podatke” objekta i metode, ovaj “pronađe” odgovarajući objekt, izvrši metodu i vrati nazad procesu pozivatelju povratnu vrijednost metode.

ORPC tehnologija

- Svaka od navedenih tehnologija ima cijeli niz različitih implementacija koje mogu biti vezane uz različite verzije operacijskog sustava (procesi se u komunikaciji oslanjaju na međusloj koji je softverski sloj između procesa i os-a – određena implementacija tehnologije je vezana uz os!!!), mogu biti cross-platformske implementacije, mogu biti vezane uz određeni programski jezik,....

ORPC tehnologija

- Neke od implementacija ORPC-a (nisu jedine!!):

1. RMI (Remote methode invocation) – ORPC tehnologija u Java programskom jeziku
2. DCOM (Distributed component object model) – Microsoft Windows os međusloj za razvoj ORPC aplikacija (postoje API-ji za različite programske jezik)
3. .NET remoting – Microsoft Windows .NET framework (<=2.x) za ORPC aplikacije
4. .....

Message-oriented tehnologija

- Osnovno svojstvo message-oriented distribuiranih aplikacija je da dva procesa međusobno komuniciraju razmjenjujući poruke formatirane na točno definiran način.
- Poruke sadrže informaciju o funkcionalnosti koja se treba izvršiti na strani primatelja poruke ili o podacima koje jedan proces šalje drugom procesu npr. rezultat izvršavanja neke funkcionalnosti.

Message-oriented tehnologija

- Implementacija funkcionalnosti nije strogo definirana kao kod RPC ili ORPC. To može biti funkcija, može biti metoda u objektu, ali može biti i bilo što drugo.
- Možete zamisliti koliko ima implementacija ove tehnologije. Postojali su pokušaji standardizacije pa je 1993. bio osnovan i MOM konzorcij u svrhu postavljanja standarda, ali većina velikih kompanija je napravila svoje implementacije sa vlastitim razvijenim standardima.

Message-oriented tehnologija

- Trenutno postoji projekt koji se zove OpenMAMA projekt (www.openmama.org) koji nastavlja raditi na standardizaciji MOM.
- Neke od MOM implementacija su:

1. Microsoft Message Queue Server (MSMQ)
2. IBM WebSphere MQ
3. Java Message Service (JMS) (dio Java 2 Platform, Enterprise Edition)
4. .....

Message-oriented tehnologija

- Još jedan razlog zašto se MOM uglavnom više ne razvija je i uvođenje tehnologije web servisa koja također omogućava razvoj distribuiranih aplikacija koje razmjenjuju poruke, a web servisi su standardizirani i protokoli na kojima se temelje web servisi su opće prihvaćeni standardi i zato je danas u biti puno lakše svoju distribuiranu aplikaciju koja funkcionira razmjenom predefiniranih poruka razvijati temeljenu na web servisima nego koristiti baš neku proprietary tehnologiju koja ja zatvorena i ograničena.

SOA arhitektura

- Razvojem web servisa širi se i korištenje SOA (Service Oriented Architecture) arhitekture. “Servisne” aplikacije pružaju servis određenih funkcionalnosti koje “servisna” aplikacija pruža drugoj aplikaciji. Korisničke aplikacije zahtjevaju određenu funkcionalnost od servisne aplikacije, a rezultat te funkcionalnosti uključuju u vlastitu aplikaciju kao njen sastavni dio.
- Ovakav tip aplikacija je vrlo čest na webu, i većina “velikih” web aplikacija pruža različite servise drugim aplikacijama. Npr. Google, Amazon, ...

SOKETI

Soket

- TCP/IP protokol je sastavni dio operacijskog sustava. Što to znači da je protokol sastavni dio os-a? To znači da operacijski sustav u sebi ima isprogramirane funkcionalnosti definirane u standardima TCP i IP protokola (označi računalo sa IP adresom, složi poruku sa IP i TCP zaglavljem,.....).
- Operacijski sustav programeru pruža sučelje preko kojeg programer koristi TCP/IP protokol i to je u stvari soket.

Soket

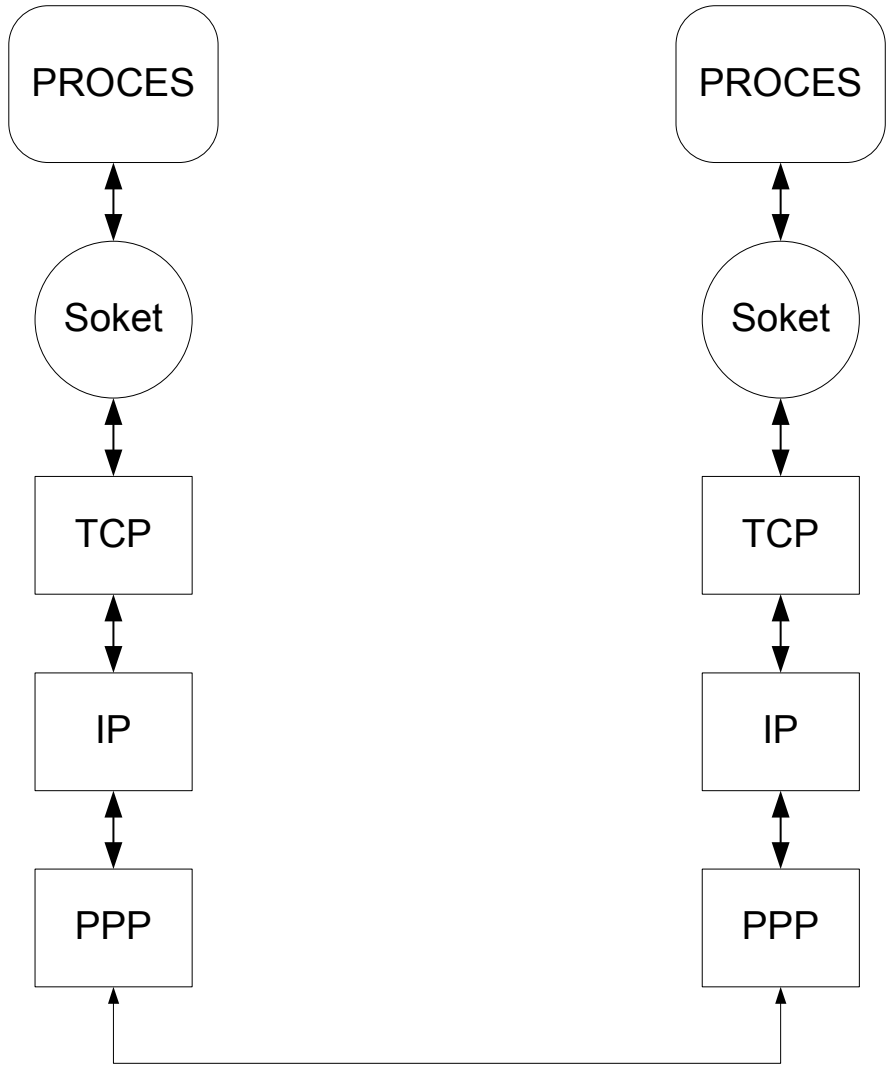
- TCP/IP standard ne specificira kako se aplikacijski softver povezuje sa softverom TCP/IP protokola već samo predlaže funkcionalnost koju treba imati sučelje između aplikacije i softvera TCP/IP protokola.
- Soket je u biti TCP/IP API (application programming interface). Podržan je od većine operacijskih sustava.

Soket

- Različiti programski jezici omogućuju korištenje soketa kroz vlastite biblioteke koda koje od operacijskog sustava traže soket funkcionalnosti.
- Soket se obično formalno definira kao apstrakcija krajnje komunikacijske točke (*socket is an abstraction for an end point of communication*).

Soket

- Soketi omogućavaju dvosmjernu komunikaciju od točke do točke (point-to-point) između dva procesa.



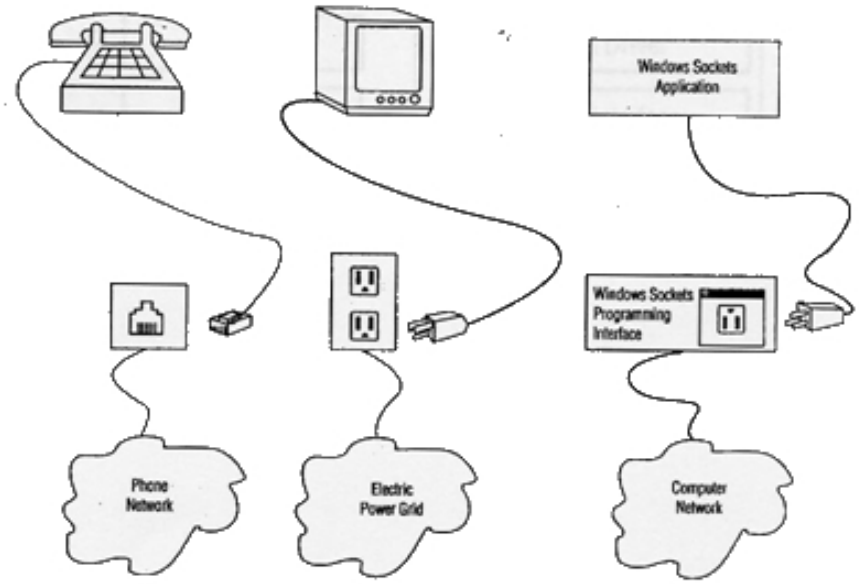
Soket

- Prednosti
  - Fleksibilnost (za soket su preneseni podaci samo stream bajtova) i podrška od različitih os-ova.
- Nedostatci
  - Detalji sučelja prema TCP/IP softveru razlikuju se od jednog do drugog os-a.
  - Složenost aplikacije
  - Podložnost koda greškama
  - Svaki dio komunikacije treba biti eksplicitno programiran

- Često korišteni API za TCP/IP:
  - Berkeley UNIX socket Interface
  - System V UNIX Transport Layer Interface (TLI)
  - Windows Sockets Interface
  - MacTCP

Soket

- Soket sučelje za TCP/IP protokol je razvijeno na Berkeleyu za UNIX.
- Od 1993. se koristi i na Windows operacijskim sustavima pod imenom WinSock. To je samo implementacija Berkeley soketa na Windows OS.



TCP/IP API

- Funkcionalnosti koje treba uključivati TCP/IP API su određene samim svojstvima protokola, a uključuju:

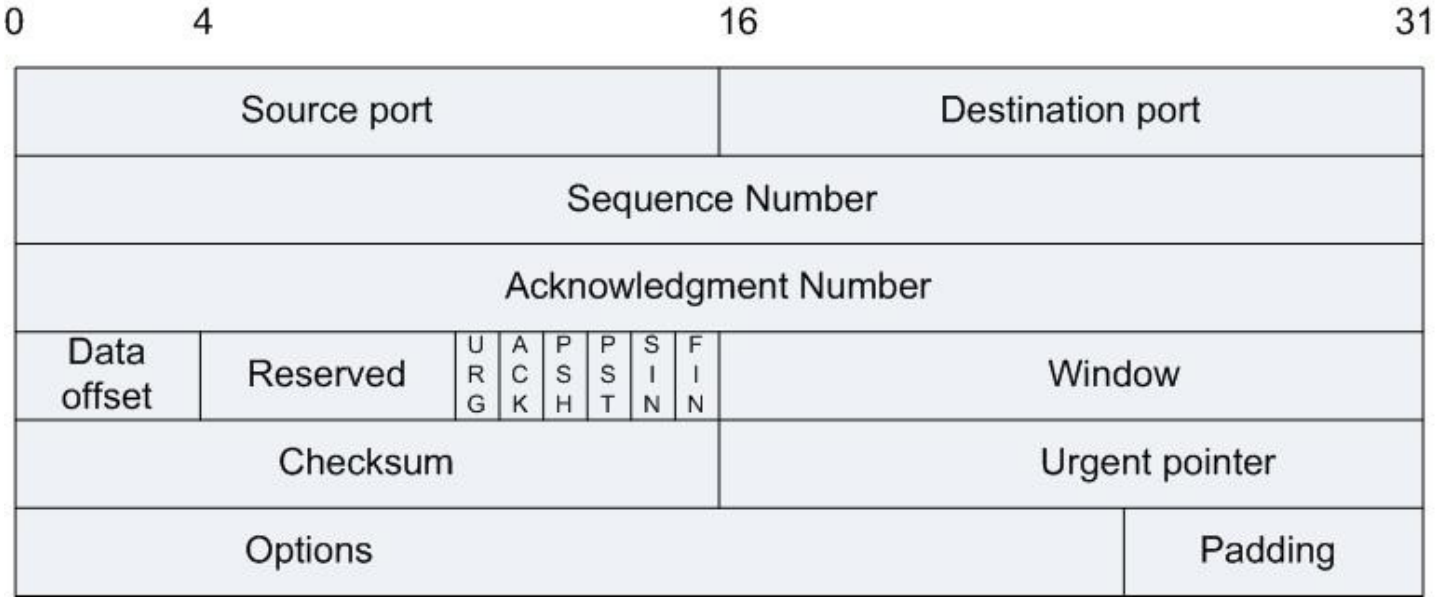
1. Alokacija resursa za komunikaciju.
2. Specificiranje lokalne i udaljenje krajnje točke komunikacije.
3. Iniciranje konekcije (na strani klijenta).
4. Čekanje na dolaznu konekciju (na strani server).
5. Slanje i primanje podataka.

TCP/IP API

6. Utvrđivanje dolaska podataka.
7. Generiranje hitnih podataka.
8. Obrada dolaznih hitnih podataka.
9. Skladan raskid konekcije (FIN *handshake*).
10. Obrada raskida konekcije s udaljenog mjesta.
11. Prekid komunikacije.
12. Obrada grešaka i prekida komunikacije.
13. Oslobađanje lokalnih resursa nakon završetka komunikacije.

TCP protokol

- Svojstva TCP protokola:
  - uspostavlja vezu (*connection-oriented*),
  - pouzdan (kontrola i ispravljanje grešaka),
  - obosmjeran (*full-duplex*),
  - protokol za prijenos niza bajtova (*byte-stream*).

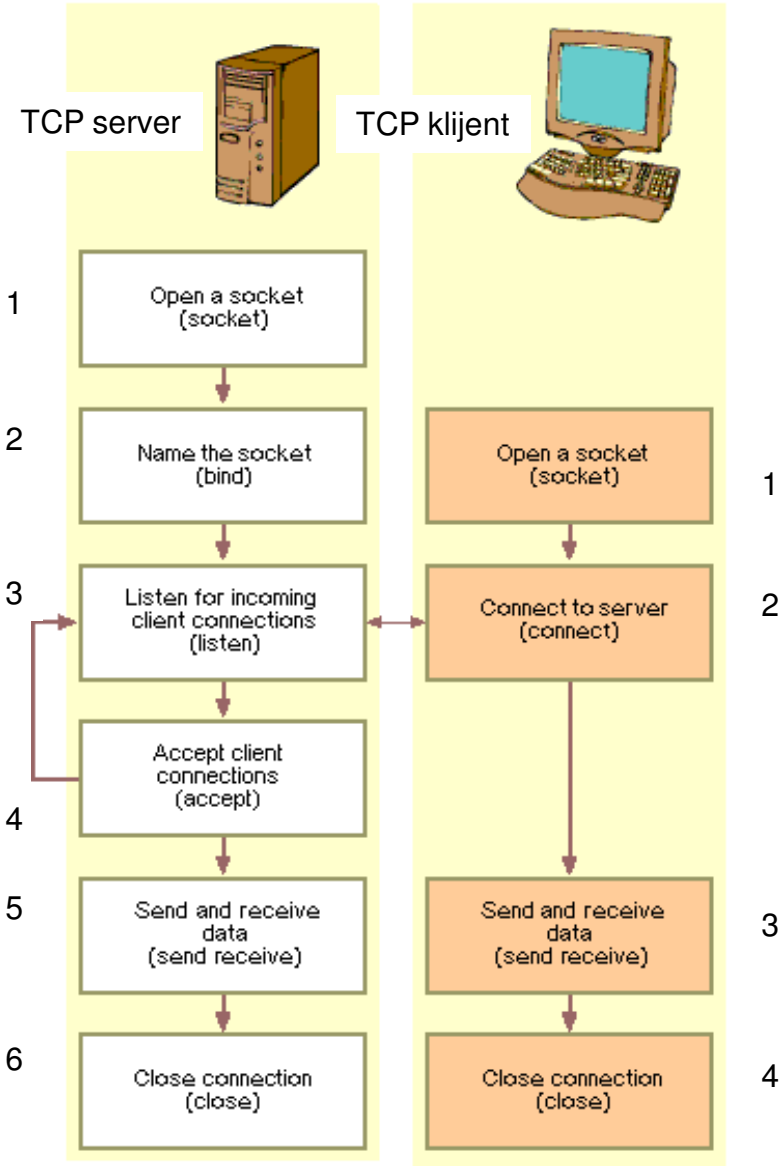


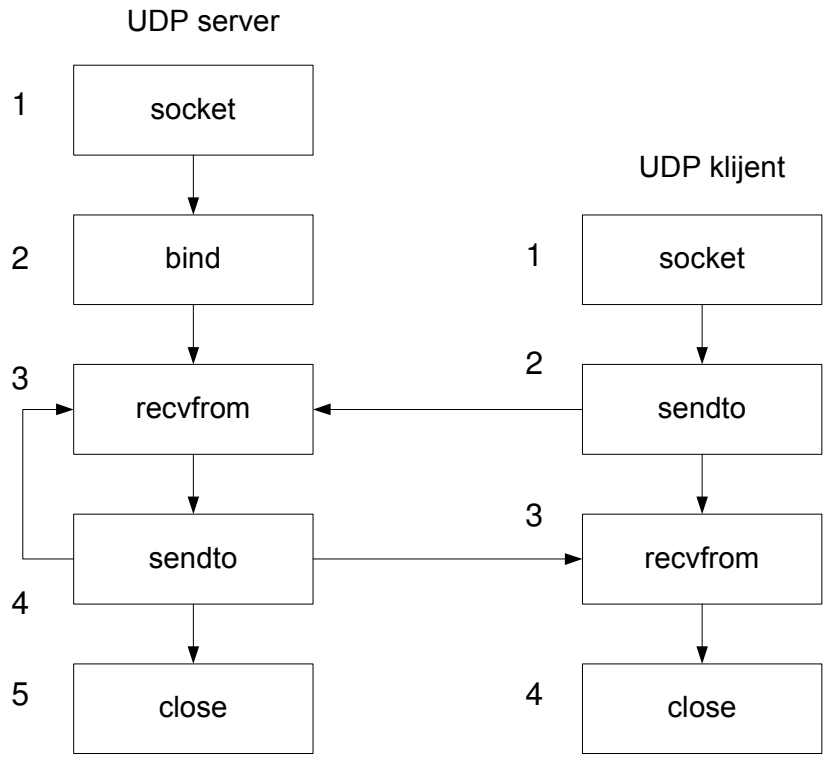
Soket

- U praksi postoje tri tipa soketa:
  - TCP soket (*stream socket*)
  - UDP soket (*datagram socket*)
  - IP soket (*row socket*) (ne na Winsock-u)

- Ovisno o potrebama procesa koji komuniciraju izaberete tip soketa. Ako komunikacija treba biti pouzdana, onda ćete koristiti TCP soket, jer TCP protokol u sebi ima integriranu kontrolu isporuke podataka.

Soket


TCP server
TCP klijent
1
Open a socket (socket)
2
Name the socket (bind)
3
Listen for incoming client connections (listen)
4
Accept client connections (accept)
5
Send and receive data (send receive)
6
Close connection (close)
1
Open a socket (socket)
2
Connect to server (connect)
3
Send and receive data (send receive)
4
Close connection (close)


UDP server
1
socket
2
bind
3
recvfrom
4
sendto
5
close
UDP klijent
1
socket
2
sendto
3
recvfrom
4
close

REALIZACIJA SOKETA U C PROGRAMSKOM JEZIKU NA WINDOWS I NA LINUX OS

Kreiranje soketa

- Da bi aplikacija mogla koristiti soket treba ga kreirati funkcijom:

int socket(int *family*, int *type*, int *proto*); UNIX

SOCKET socket(int *af*, int *type*, int *protocol*); Windows

- Poziv funkcije *socket* vraća handle na soket. Handle je tip podatka poput pokazivača, ali bez direktnog pristupa adresnom prostoru koji handle referencira.

Kreiranje soketa

- Funkcija prima tri argumenta:
- Prvi argument određuje “obitelj” protokola koji će se koristiti (PF_INET (*protocol family Internet*) ili AF_INET (*adress family Internet*) za TCP/IP, INET-Internet) jer se soketi mogu koristiti i sa drugim protokolima, a ne samo sa TCP, UDP protokolima.
- Sljedeći argument određuje tip soketa (SOCK_STREAM, SOCK_DGRAM, SOCK_RAW (samo Unix)).
- Zadnji argument definira protokol i obično se postavlja u 0 koji znači korištenje *defaultnog* protokola.

Vezivanje (*binding*)

- Sljedeći korak na strani servera je vezivanje (*binding*) soketa na određenu IP adresu i port na kojem će soket biti otvoren. Ovaj korak se može preskočiti (ne mora) na strani klijenta jer će sa taj dio napraviti prilikom spajanja na server.

int bind(int *sockfd*, const struct sockaddr* *name*, int *localaddrlen*);
int bind(SOCKET *s*, const struct sockaddr* *name*, int *namelen*);

Vezivanje (*binding*)

- Funkcija prima tri argumenta:
- Prvi argument funkcije je handle na soket, zatim ide pokazivač na strukturu koja sadrži adresu na koju se soket vezuje i zadnji argument je veličina adresne strukture (obično se samo stavi sizeof(sockaddr)).
- Funkcija vraća 0 u slučaju uspješnog *binding*, a -1 u slučaju greške.

Struktura *sockaddr*

- Struktura *sockaddr* je generička struktura soket adrese definirana ANSI C standardom. ANSI C je poslije preuzela i ISO organizacija, većina C kompajlera radi prema tom standardu.

```
struct sockaddr {
    uint8_t sa_len;  /*veličina strukture*/
    sa_family_t sa_family; /*obitelj adresa: AF_xxx*/
    char sa_data[14];  /*adresa određena protokolom*/
} /*deklarirana <sys/socket.h> */


/*Windows*/
struct sockaddr {
    u_short sa_family; /*obitelj adresa: AF_xxx*/
    char sa_data[14];  /*adresa određena protokolom*/
}; /*deklarirana Winsock.h */
```

Struktura *sockaddr*

- Tip *sockaddr* automatski određuje strukturu adrese ovisno o postavljenoj “obitelji” adresa (AF_INET).
- Detaljnija specifikacija adrese daje se u strukturi *sockaddr_in* koja će se koristiti i na Windowsima i na Unixu ukoliko se radi s IPV4, a struktura *sockaddr_in6* ukoliko se radi s IPV6.

```
struct sockaddr_in
{short sin_family;
unsigned short sin_port;
struct   in_addr sin_addr;
char sin_zero[8]; };

struct sockaddr_in6
{ short sin6_family;
u_short sin6_port;
u_long sin6_flowinfo; struct
     in6_addr sin6_addr;
u_long sin6_scope_id; };
```

Struktura *sockaddr*

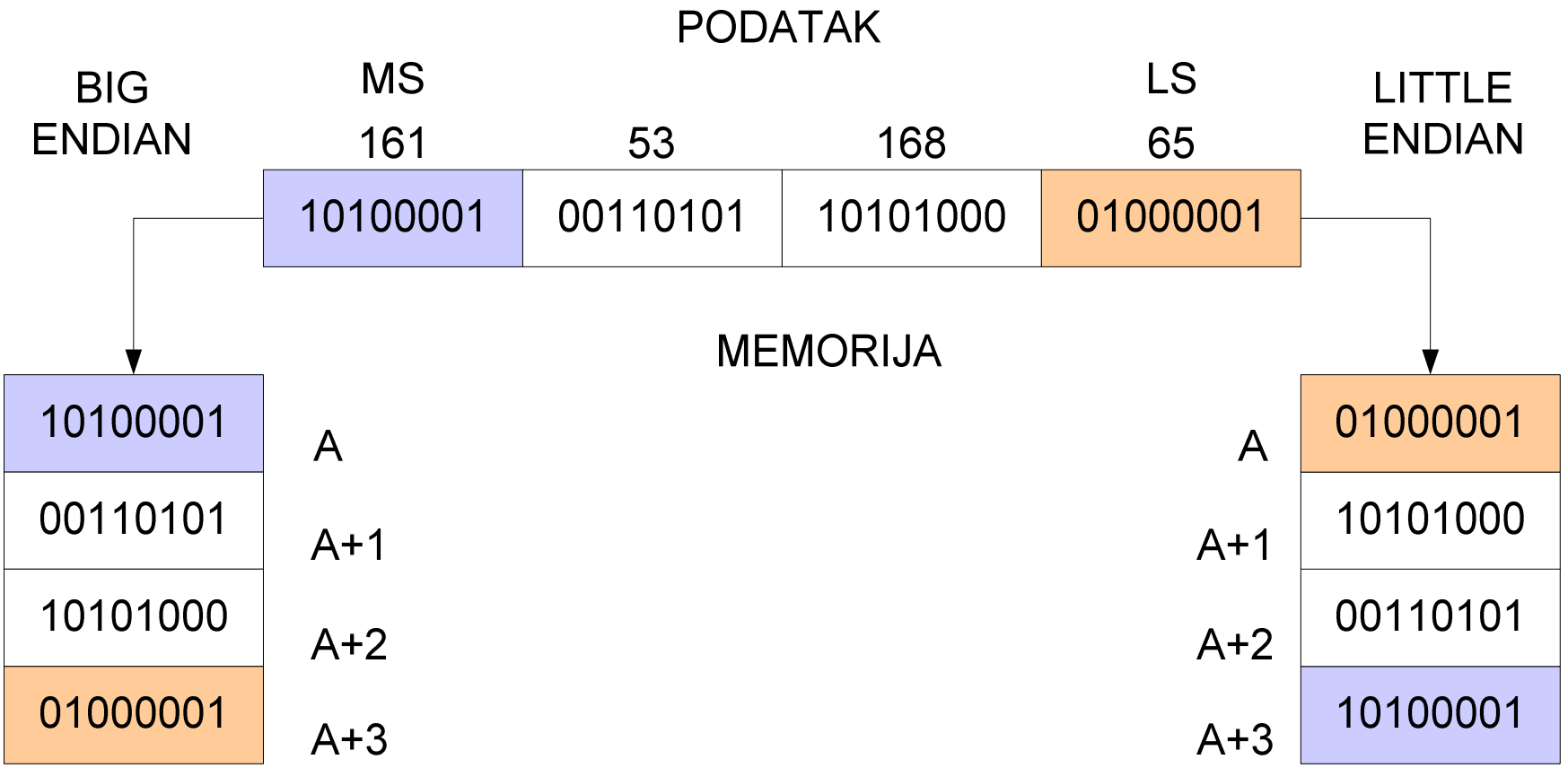
- Zapis 32-bitne IP adrese također je formatiran strukturom:

```
struct in_addr {    union {
struct {  unsigned  char   s_b1, s_b2,   s_b3, s_b4;  }  S_un_b;
        /*Adresa formatirana kao 4 u_char-a*/
struct { unsigned  short  s_w1, s_w2;   }  S_un_w;
        /*Adresa formatirana kao dva u_short-a*/
unsigned long  S_addr;       /*Adresa formatirana kao jedan
   long*/ } S_un;  };
```

Mrežni redoslijed bajtova

- IP adresa i broj porta u adresnim strukturama soketa trebaju biti zapisani u mrežnom redoslijedu bajtova (*network byte order - NBO*).
- Npr. IP adresa 161.53.168.65 ima 32 bita odnosno 4 bajta i može se u memoriji zapisati na dva načina.
- Prvi način je da se najznačajniji bajt (*MS-most significant*) spremi prvi. Taj način se naziva *big-endian* zapis. *Little-endian* zapis sprema najmanje značajan bajt (*LS-least significant*) na prvo mjesto.

Mrežni redoslijed bajtova



Mrežni redoslijed bajtova

- O čemu ovisi način pohrane 32 bitne vrijednosti u memoriju?
- PC arhitektura s Intelovim procesorom koristi *little-endian* zapis.
- RISC arhitekture i *mainframe* računala koriste *big-endian* zapis.
- Kao standard na mreži je prihvaćen *big-endian* zapis i svi paketi koji idu preko mreže trebaju imati zapisanu IP adresu (i port) u ovom obliku.

Funkcije za rad sa NBO

- Programeru su na raspolaganju funkcije koje omogućavaju pretvaranje jednog načina zapisa u drugi.
- Imenovanje funkcija je postavljeno na način da funkcije koje pretvaraju iz formata zapisa na *hostu* u mrežni format zapisa počinju s ‘h’ (*host byte order*), a funkcije koje pretvaraju iz mrežnog formata u format zapisa na *hostu* počinju s ‘n’ (*network byte order*).

Funkcije za rad sa NBO

- Funkcije za pretvaranje 16 bitnih vrijednosti (port) završavaju sa slovom ‘s’ (*short*), a za pretvaranje 32-bitnih vrijednosti završavaju s ‘l’ (*long*).

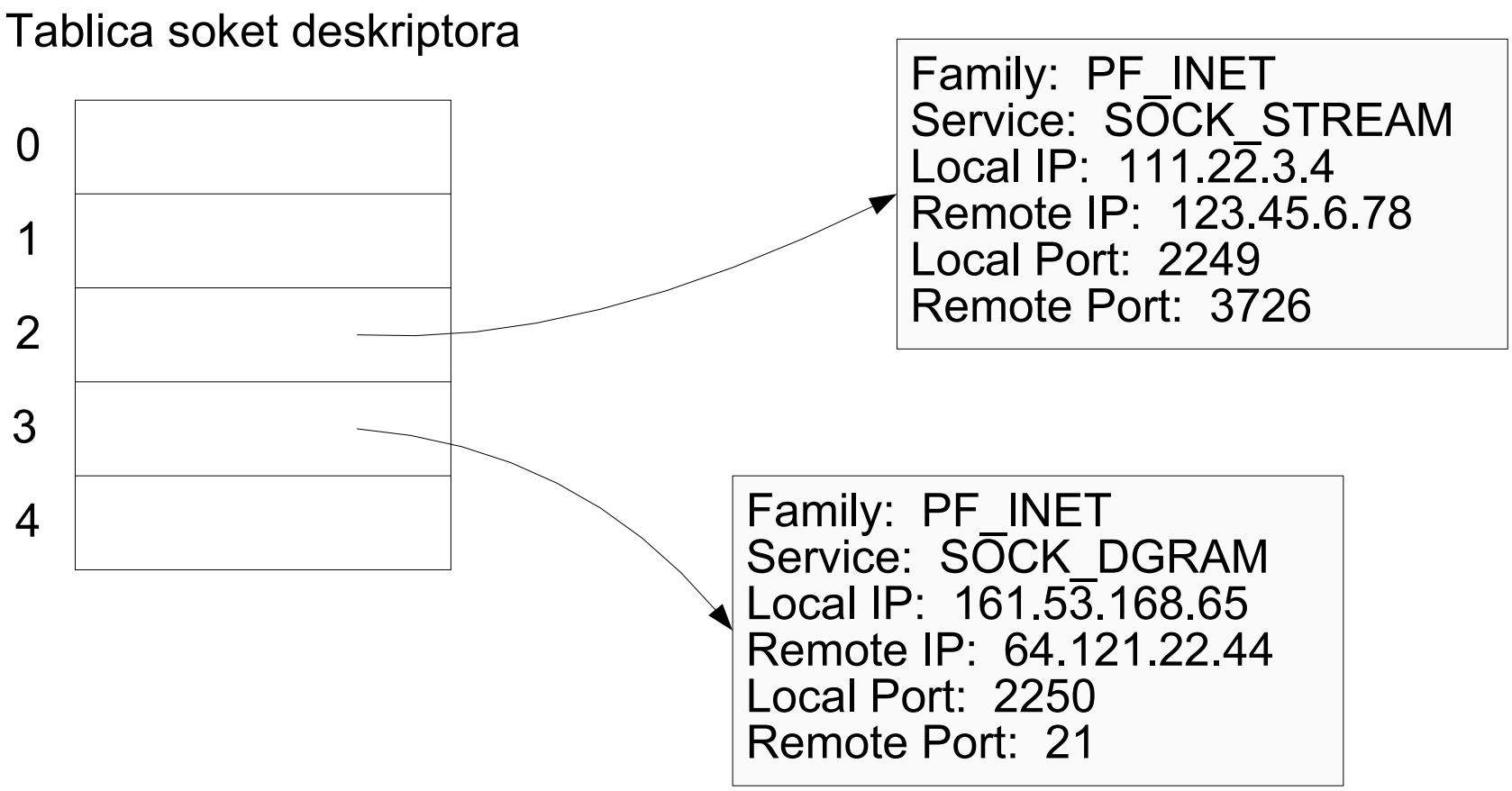
```
uint16_t htons(uint16_t);
uint16_t ntohs(uint_16_t);
uint32_t htonl(uint32_t);
uint32_t ntohl(uint32_t);
```

- Na Windowsima je tip uint_16_t -> u_short.

Adresa i port

- Ukoliko niste sigurni koju adresu koristiti (npr. imate više mrežnih sučelja) možete zatražiti od OS da procesu dodijeli adresu.
- Također možete zatražiti i da OS procesu dodijeli slobodni port što klijentske aplikacije u pravilu i rade.
- Za serverske aplikacije obično je definirano na kojem portu “slušaju” (npr. HTTP server – 80) pa trebaju koristiti upravo taj port jer će klijentske aplikacije pokušati otvarati konekciju prema serveru na tom portu.

Soket


Tablica soket deskriptora
0
1
2
3
4
Family: PF_INET
Service: SOCK_STREAM
Local IP: 111.22.3.4
Remote IP: 123.45.6.78
Local Port: 2249
Remote Port: 3726
Family: PF_INET
Service: SOCK_DGRAM
Local IP: 161.53.168.65
Remote IP: 64.121.22.44
Local Port: 2250
Remote Port: 21

Soket

- Nakon kreiranja soketa i vezivanja IP adrese i porta uz soket handle ili soket deskriptor, server za TCP konekciju (SOCK_STREAM tip soketa) “osluškuje” dolazne zahtjeve za konekcijama od strane klijenata funkcijom:

  int listen(int (SOCKET) *sockfd*, int *backlog*);

- Prvi argument funkcije je soket deskriptor, a drugi argument je maksimalni broj konekcija koji se može otvoriti na tom soketu.

Soket

- Nakon kreiranja soketa i vezivanja IP adrese i porta uz soket deskriptor, klijent za TCP konekciju (SOCK_STREAM tip soketa) pokušava uspostaviti konekciju prema nekom serveru funkcijom:

  int connect(int (SOCKET) *sockfd*, const struct sockaddr* *name*, int *namelen*);

- Prvi argument funkcije je soket deskriptor, drugi argument je pokazivač na strukturu s podacima o serveru (IP adresa i port) prema kojem se uspostavlja konekcija, a zadnji je argument veličina strukture.

Soket

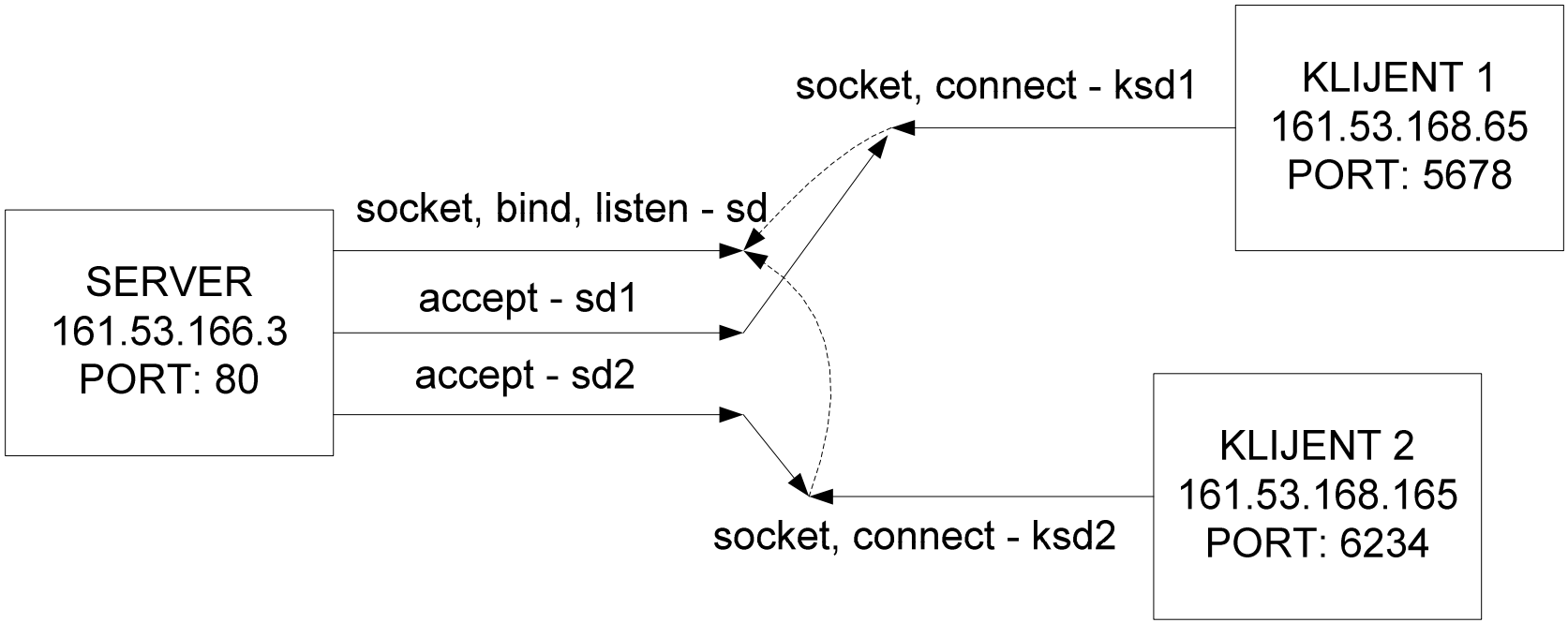
- Nakon što je server preko *listen* funkcije dobio zahtjev od klijenta za otvaranje konekcije ukoliko prihvaća klijentov zahtjev konačno se konekcija uspostavlja funkcijom:

  int (SOCKET) accept(int (SOCKET) *sockfd*, struct sockaddr* *addr*, int* *addrlen*);

- Prvi argument funkcije je soket deskriptor, drugi argument je pokazivač na strukturu s podacima o klijentu koji se spojio i treći argument je pokazivač na duljinu strukture koji je opcionalan. U slučaju uspješnog izvršavanja i ova funkcija vraća 0.

Soket

Višestruke konekcije na istom portu????



Soket

- Za prijenos podatak tipom soketa SOCK_STREAM koriste se sljedeće funkcije:

  /*Unix*/

  int read(int *sockfd*, const void **buf*, int *buflen*);

  int write(int *sockfd*, const void **buf*, int *buflen*);

  /*Windows*/

  int recv(SOCKET *s*, char* *buf*, int *len*, int *flags*);

  int send(SOCKET *s*, const char* *buf*, int *len*, int *flags*);

- Ukoliko su se uspješno izvršile povratne vrijednosti su broj pročitanih ili poslanih bajtova.

Soket

- Za prijenos podatak tipom soketa SOCK_DGRAM koriste se sljedeće funkcije:

int sendto(SOCKET *s*, const char* *buf*, int *len*, int *flags*, const struct sockaddr* *to*, int *tolen*);

int recvfrom(SOCKET *s*, const char* *buf*, int *len*, int *flags*, const struct sockaddr* *to*, int *tolen*);

Soket

- Za zatvaranje soketa koristi se funkcija:

  int close(int *sockfd*);

- Ukoliko se IP adresa predaje kao argument programu obično je zapisana u string formatu. Funkcije za konverziju u in_addr strukturu:

  int inet_aton(char *, struct in_addr *); /*Unix*/

  unsigned long inet_addr(const char* *cp*); /*Windows*/

- Funkcija za konverziju iz in_addr formata u string format:

  char *inet_ntoa(struct in_addr);

REALIZACIJA SOKET APLIKACIJE U DRUGIM PROGRAMSKIM JEZICIMA

Soket server u C#-u

- .NET framework sokete implementira kroz nekoliko klasa koje se nalaze u imenskom prostoru System.Net.Sockets.
- Klasa TcpListener omogućava stvaranje TCP soket servera. Metode klase omogućavaju osluškivanje i prihvaćanje dolaznih konekcija na blokirajući sinkroni način.
- Adresa i port servera se definiraju u konstruktoru objekta preko objekta IPEndPoint, ili preko objekta IPaddress i 32-bitnog integera broja porta. Ukoliko želimo da OS serveru da IP adresu i broj porta onda se za IP adresu koristi svojstvo Any IPaddress klase i 0 za broj porta.

Soket server u C#-u

- Metoda Start započinje osluškivanje dolaznih zahtjeva klijenata za konekcijom. Svi dolazni zahtjevi će se smještati u red sve dok se ili ne pozove metoda Stop ili dok broj dolaznih zahtjeva ne nadmaši vrijednosti svojstva MaxConnections.
- Za prihvaćanje dolaznog zahtjeva koristi se metoda AcceptSocket ili metoda AcceptTcpClient. Ove metode su blokirajuće. Za neblokirajući način rada može se koristiti metoda Pending koja provjerava da li postoje dolazne konekcije prije prihvaćanja konekcije.
- Metoda Stop zatvara TcpListener.

```
public static void Main()
{    try  {Int32 port = 13000;
    IPAddress localAddr = IPAddress.Parse("127.0.0.1");
    TcpListener server = new TcpListener(localAddr, port);
    // Start listening for client requests.
    server.Start();
    // Buffer for reading data
    Byte[] bytes = new Byte[256];  String data = null;
    // Enter the listening loop.
    while(true)
    {      Console.Write("Waiting for a connection... ");
     // Perform a blocking call to accept requests.
     TcpClient client = server.AcceptTcpClient();
      Console.WriteLine("Connected!"); data = null;
      // Get a stream object for reading and writing
      NetworkStream stream = client.GetStream(); Int32 i;
     // Loop to receive all the data sent by the client.
      while((i = stream.Read(bytes, 0, bytes.Length))!=0)
      {     // Translate data bytes to a ASCII string.
        data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);
        Console.WriteLine(String.Format("Received: {0}", data));
        // Process the data sent by the client.
        data = data.ToUpper(); Byte[] msg = System.Text.Encoding.ASCII.GetBytes(data);
        // Send back a response.
        stream.Write(msg, 0, msg.Length);
        Console.WriteLine(String.Format("Sent: {0}", data));   }
     // Shutdown and end connection
     client.Close();    }  }
  catch(SocketException e)
  {    Console.WriteLine("SocketException: {0}", e);  }
     Console.WriteLine("\nHit enter to continue...");
   Console.Read(); }
```

Soket klijent u C#-u

- Klasa TcpClient omogućava kreiranje TCP klijenta.
- Spajanje na server radi se preko metode Connect kojoj se kao argumenti predaju IP adresa i port servera na koji se klijent spaja.
- Ako se odmah u konstruktoru navede IP adresa i port prilikom kreiranja klijenta ovaj će se odmah pokušati i spojiti na server.
- Klijent se zatvara metodom Close.

UDP soket klijent u C#-u

- Za UDP klijenta koristi se klasa UdpClient.
- Kako bi klijent mogao slati i primati podatke koristi se klasa NetworkStream (System.Net.Sockets.NetworkStream).
- Za dohvaćanje instance klase na kreiranom soketu koristi se metoda GetStream.
- Metode Write i Read klase NetworkStream koriste se za slanje tj. za primanje podataka.

```
using System;
using System.Net;
using System.Net.Sockets;
namespace NETSoketKlijent
{
   class Class1 {
       [STAThread]
       static void Main(string[] args) {
       Byte [] returnarray = new Byte[20];
      String s = "Klijent u C#";
       TcpClient cl = new TcpClient();
       cl.Connect("localhost", 5001);
       NetworkStream str = cl.GetStream();
       Byte [] array =
  System.Text.Encoding.ASCII.GetBytes(s);
       str.Write(array, 0, array.Length);
       str.Read(returnarray, 0, 20);
Console.WriteLine(System.Text.Encoding.ASCII.Get
  String(returnarray, 0, 20));
       str.Close();
       cl.Close();
      }}}
```

Soketi u Javi

- Java .net paket ima dvije klase:
  - Socket – za implementiranje klijenta
  - ServerSocket – za implementiranje servera
- Implementiranje soket server u Javi ->

1. Otvaranje soketa:
```
ServerSocket server;
DataOutputStream os;
DataInputStream is;
server = new ServerSocket( PORT );
```
2. Bindinig servera:
```
server.bind(java.net.SocketAddress
   adresa);
```
3. Čekaj i prihvati klijentov zahtjev:
```
Socket client =
server.accept();
```
4. Kreiraj I/O strimove za komunikaciju s klijentom:
```
is = new DataInputStream(
    client.getInputStream() );
os = new DataOutputStream(
    client.getOutputStream()
    );
```
5. Komunikacija s klijentom:
   Receive from client: `String line = is.readLine();`
   Send to client: `os.writeBytes("Hello\n");`
6. Zatvori soket:
```
client.close();
```

Soket klijent u Javi

1. Kreiranje soket objekta:
   client = new Socket(server, port_id);
2. Kreiraj I/O tokova za komunikaciju sa serverom:
   is=new DataInputStream(client.getInputStream());
   os=new DataOutputStream(client.getOutputStream());
3. Komunikacija sa serverom:
   Receive data from the server:
      String line = is.readLine();
   Send data to the server:
      os.writeBytes("Hello\n");
4. Zatvori soket:
   client.close();