Računarske mreže
(2OER5O03)

Soketi

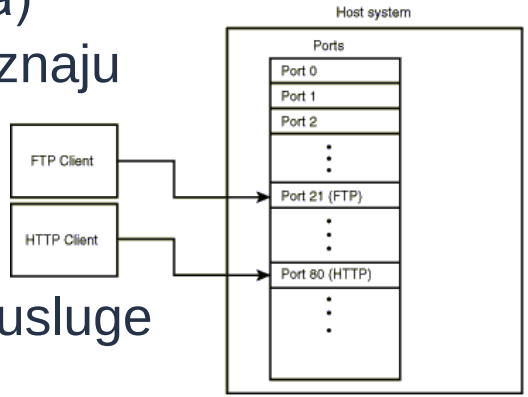
Auditivne vežbe

1

Osnovni pojmovi

Identifikacija usluge

- IP adresa jedinstveno određuje računar na Internetu, ali svaki računar može biti server većeg broja usluga (servisa)
- Na osnovu porta protokoli transportnog nivoa prepoznaju aplikaciju kojoj treba proslediti podatke
- Usluge se identifikuju brojem porta
- Port je 16-to bitni celi broj dodeljen nekom servisu
- Portovi do broja 1024 rezervisani su za standardne usluge



Podela portova

- Dobro poznati:

  1 – 1023
- Registrovani:

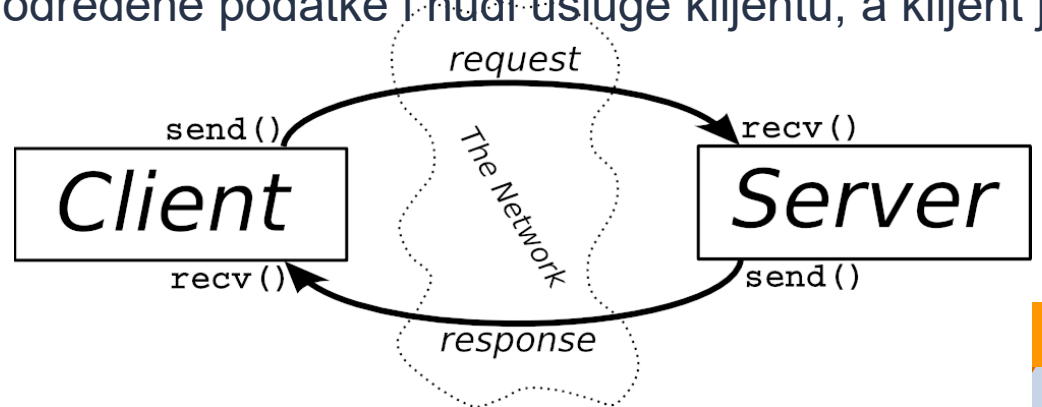
  1024 – 49151
- Dinamički (privatni):

  49152 – 65535

Port	Protokol	Servis
20	TCP	FTP data
21	TCP	FTP control (command)
25	TCP	SMTP
53	TCP/UDP	DNS
67	UDP	DHCP server
68	UDP	DHCP client
80	TCP	HTTP
110	TCP	POP3
520	UDP	RIP

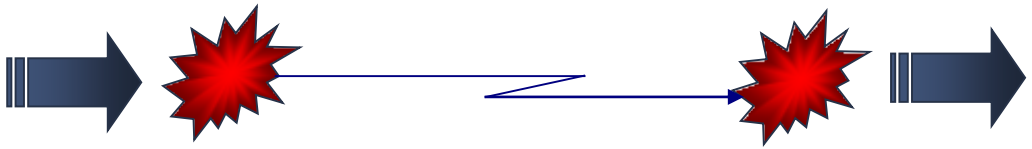
Klijent-server arhitektura

- Najveći deo saobraćaja na Internetu predstavlja uzajamnu komunikaciju parova mašina
- Da bi komunikacija bila moguća, mašine se moraju međusobno pronaći
- Mašina za koju se tačno zna “gde je” i koja osluškuje pozive naziva se **server**
- Mašina koja inicira komunikaciju naziva se **klijent**
- Server obično skladišti određene podatke i nudi usluge klijentu, a klijent je korisnik tih podataka i usluga

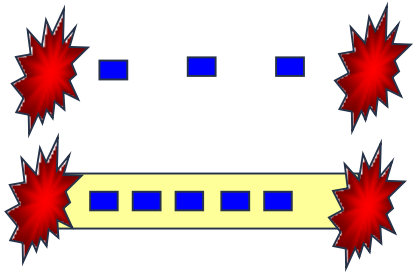


Soketi

- Soketi predstavljaju programsku apstrakciju za komunikacioni kanal, koji se uspostavlja između dva programa, tačnije između dva porta.



- Postoji tri tipa soketa:
  - datagram sockets
  - stream sockets
  - raw sockets



Datagram soketi

- **Conectionless**
- Ne održavaju otvorenu konekciju
- Svaki paket je nezavisan
- Mogu da koriste User Datagram Protocol (UDP)
  - IP telefonija

Stream soketi

- **Connection-oriented**
- Komunikacija u oba smera
- Mogu da koriste Transmission Control Protocol (TCP)
  - telnet, ssh, http

Soketi

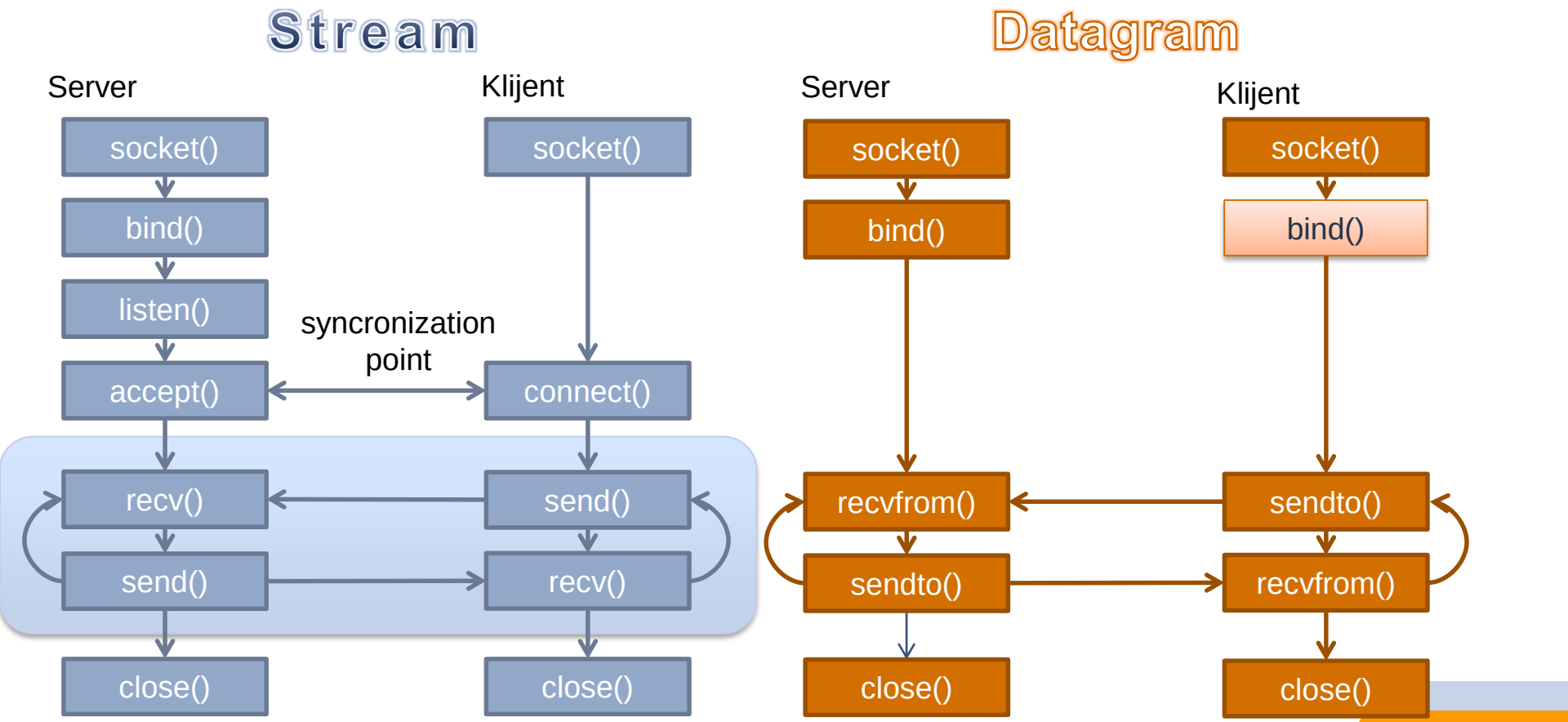
- Soket je jedinstveno određen:
  - Internet adresom
  - End-to-end protokolom (npr.TCP ili UDP)
  - Brojem porta

Rad sa soketima

Korišćenje soketa

- Konfiguracija soketa
  - Gde je remote mašina (IP adresa, hostname)
  - Koji servis prima podatke (port)
- Send & Receive
  - Dizajniran kao proširenje I/O u Unix OS
    - send – write
    - recv – read
- Zatvaranje soketa

Klijent – Server komunikacija



Inicijalizacija Winsock-a

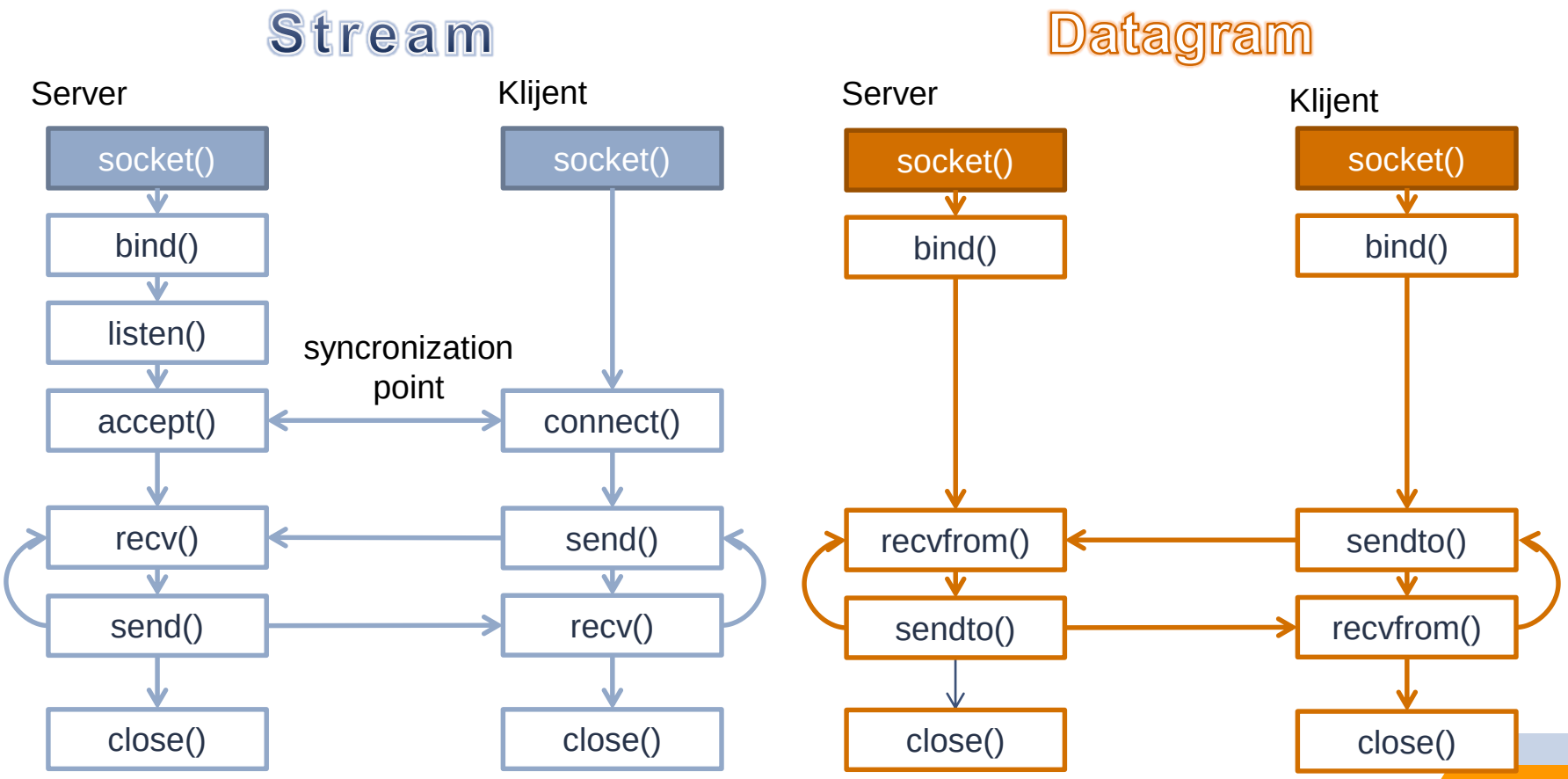
- Uključivanje zaglavlja i biblioteka:

```
#include <winsock.h>
#pragma comment(lib, "wsock32.lib")
```

- Inicijalizacija WSA strukture i aktiviranje WS2_32.DLL-a

```
WSAData wsa;
WSAStartup(0x0101, &wsa); // Verzija 1
WSAStartup(0x0202, &wsa); // Verzija 2
```

Kreiranje soketa

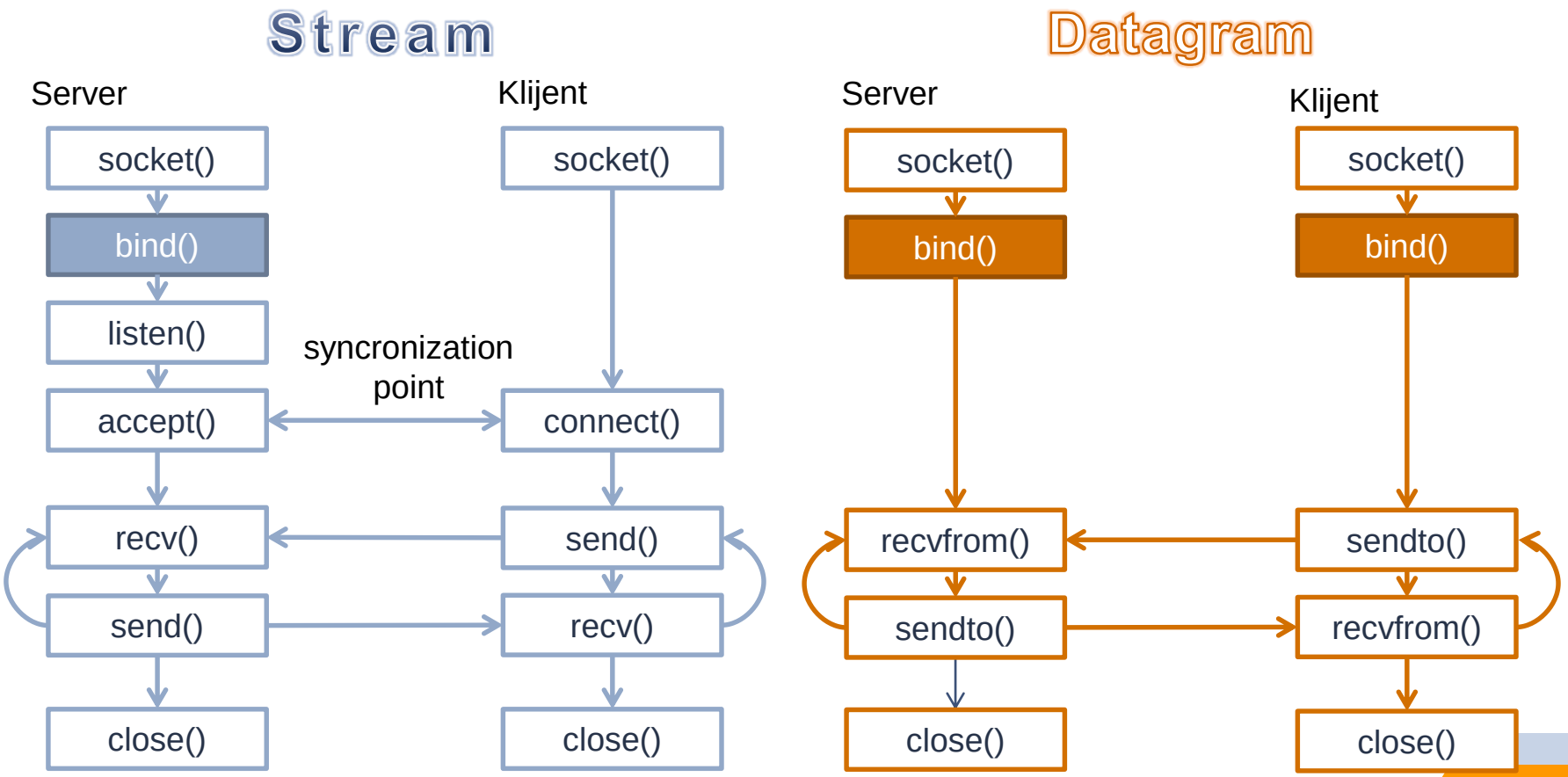


Kreiranje soketa

SOCKET sock = socket(family, type, protocol)

- **sock** - deskriptor soketa (popud handle na fajl), -1 u slučaju greške
- **family** - komunikacioni domen
  - **AF_INET**, IPv4 protokoli, Internet adrese
  - AF_UNIX, Lokalna komunikacija, fajl adrese
- **type** - tip komunikacije (soketa)
  - SOCK_STREAM i SOCK_DGRAM
  - U verziji 2 dodati su mnogi novi tipovi i ne moraju se definisati u ovom trenutku, jer aplikacija može dinamički odrediti atribute dostupnih protokola transportnog nivoa pomoću funkcije WSAEnumProtocols. Definicije tipova soketa mogu se naći u WINSOCK2.H datoteci.
- protocol: protokol koji će biti korišćen u komunikaciji, ako se prosledi 0, bira se podrazumevani za tip soketa

Postavljanje adrese



Specifikacija adrese

- Socket API definiše generički tip podataka za adrese, i posebnu formu za TCP/IP adrese

```
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address */
};

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

Konverzija adresa i portova

Različiti procesori različito predstavljaju podatke. Neki u *big endian*, a neki u *little endian* formatu. Da bi se razumeli, formati zapisa paketa koji putuju mrežom moraju biti uniformni. Izabran je *big endian* zapis. Obzirom da Intel pamti podatke u *little endian* formatu, neophodna je konverzija. Za to postoje sledeće funkcije:

- **htons** – konvertuje **short** (16-bit) iz host zapisa (*little endian*) u network zapis (*big endian*)
- **htonl** – konvertuje **long** (32-bit) iz host zapisa u network zapis
- **ntohs** – konvertuje **short** (16-bit) iz network zapisa u host zapis
- **ntohl** – konvertuje **long** (32-bit) iz network zapisa u host zapis

Dodela adrese soketu

Funkcija **bind** povezuje soket sa lokalnom komunikacionom tačkom (adresom i portom).

int status = bind(sock, &addrport, size)

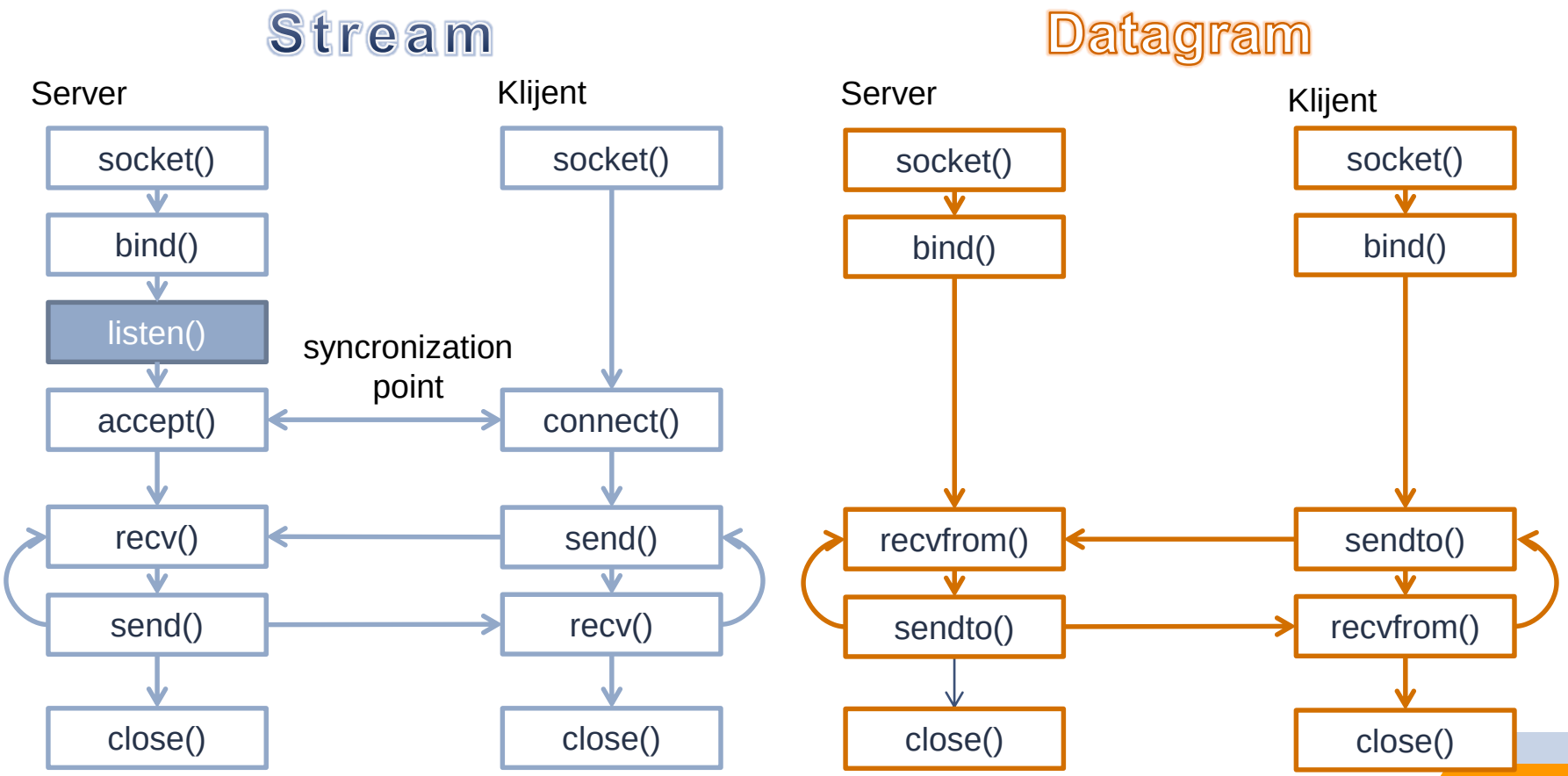
- **sock** - deskriptor soketa koji treba povezati
- **addrport** - struktura koja čuva IP adresu i port
  - za TCP/IP server najčešće INADDR_ANY
- **size** - veličina addrport strukture u bajtovima
- **status** - -1 ako je došlo do greške

bind() – primer za TCP server

```
auto sock = socket(AF_INET, SOCK_STREAM, 0);
sockaddr_in server;
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(5100);

if (bind(listensock, (struct sockaddr *)&server, sizeof(server)) != SOCKET_ERROR)
{
    //...
}
```

Osluškivanje



Osluškivanje

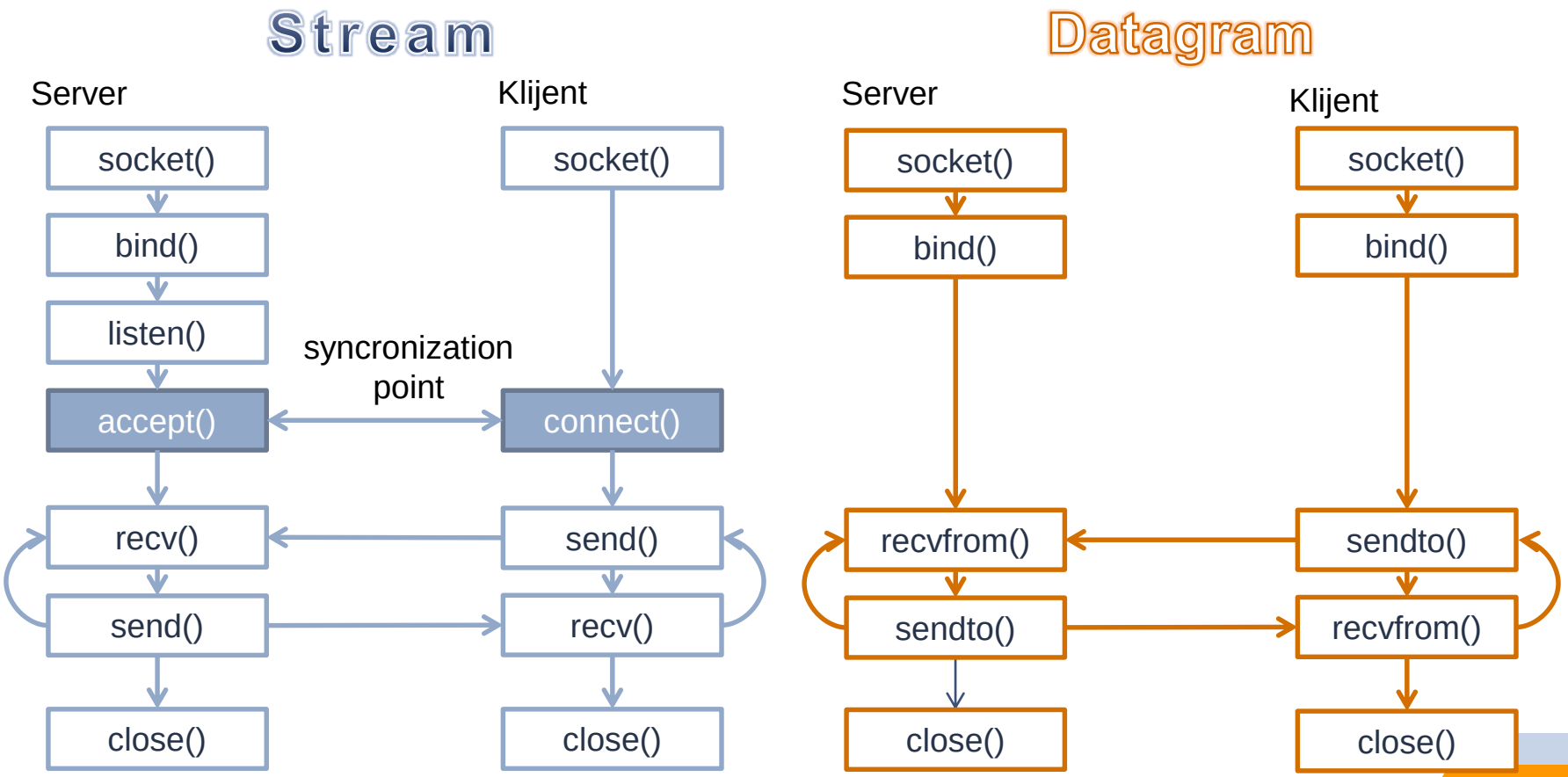
int status = listen(listeningSock, queueLimit)

- Služi da signalizira TCP protokolu da osluškuje dolazeće konekcije
  - **listeningSock** - deskriptor soketa
  - **queueLimit** - broj aktivnih učesnika koji mogu da čekaju za konekciju.
    - **SOMAXCONN** – maksimalna vrednost.
  - **status** - 0 za osluškivanje, -1 za grešku
- Listen nije blokirajuća funkcija!

Osluškivanje

- Soket za osluškivanje se NIKAD ne koristi za slanje i primanje
- Koristi ga server samo kao način za otvaranje novih soketa

Uspostavljanje konekcije – Stream socket only



Uspostavljanje konekcije – Stream socket klijent

- Klijent uspostavlja konekciju sa serverom pozivom funkcije connect()

int status = connect(sockid, &foreignAddr, addrLen)

  - **sockid** - deskriptor soketa koji se koristi za konekciju
  - **foreignAddr** - adresa pasivnog učesnika, udaljena komunikaciona tačka
  - **addrLen** - veličina foreignAddr
  - **status** - 0 ako je konekcija uspešna, -1 ako je došlo do greške

- connect() je blokirajuća funkcija!

Uspostavljanje konekcije – Stream socket server

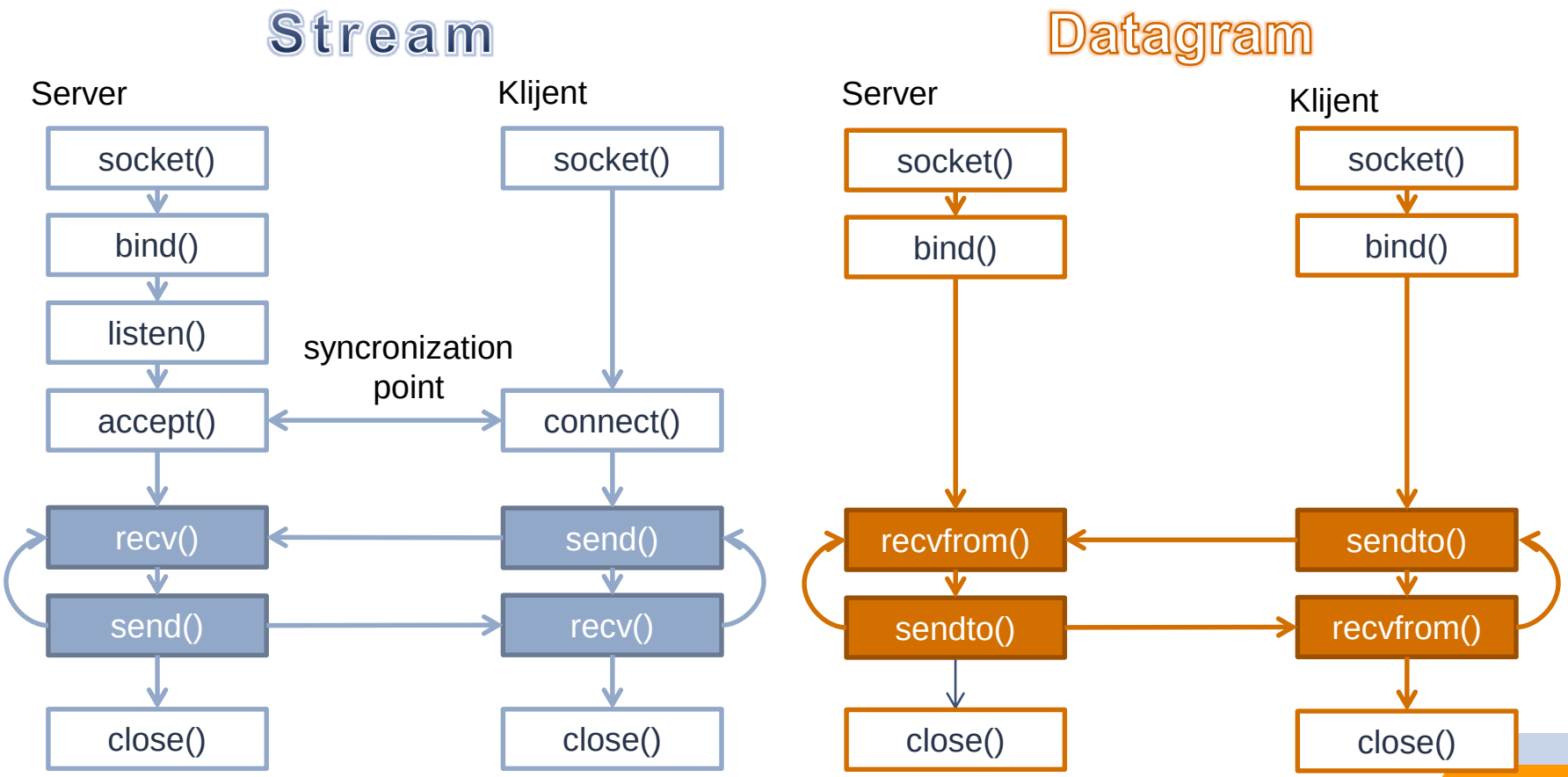
Server dobija soket za dolazeću konekciju sa klijentom pozivom funkcije accept(). Funkcija kreira novi soket preko koga se zapravo ostvaruje komunikacija sa klijentom. Serverski soket nastavlja da osluškuje nove konekcije.

SOCKET s = accept(sockid, &clientAddr, addrLen)

- **s** - deskriptor novog soketa
- **sockid** - deskriptor soketa na kom se osluškuje
- **clientAddr** - adresa aktivnog učesnika (popunjava funkcija)
- **addrLen** - sizeOf(clientAddr)

accept() je blokirajuća funkcija!

Razmena podataka



Razmena podataka – Stream soketi

int count = send(sockid, msg, msgLen, flags)

- **msg** - poruka koja se šalje
- **msgLen** - dužina poruke u bajtovima
- **flags** - specijalne opcije, obično 0
- **count** - broj prenesenih bajtova , -1 ako je došlo do greške
- send() je blokirajuća funkcija!

Razmena podataka – Stream soketi

int count = recv(sockid, recvBuf, bufLen, flags)

- **recvBuf** - bafer za dolaznu poruku
- **bufLen** - broj primljenih bajtova
- **flags** - specijalne opcije, obično 0
- **count** - broj prenesenih bajtova , -1 ako je došlo do greške
- recv() je blokirajuća funkcija!

Razmena podataka – Datagram soketi

int count = sendto(sockid, msg, msgLen, flags, &foreignAddr, addrLen)

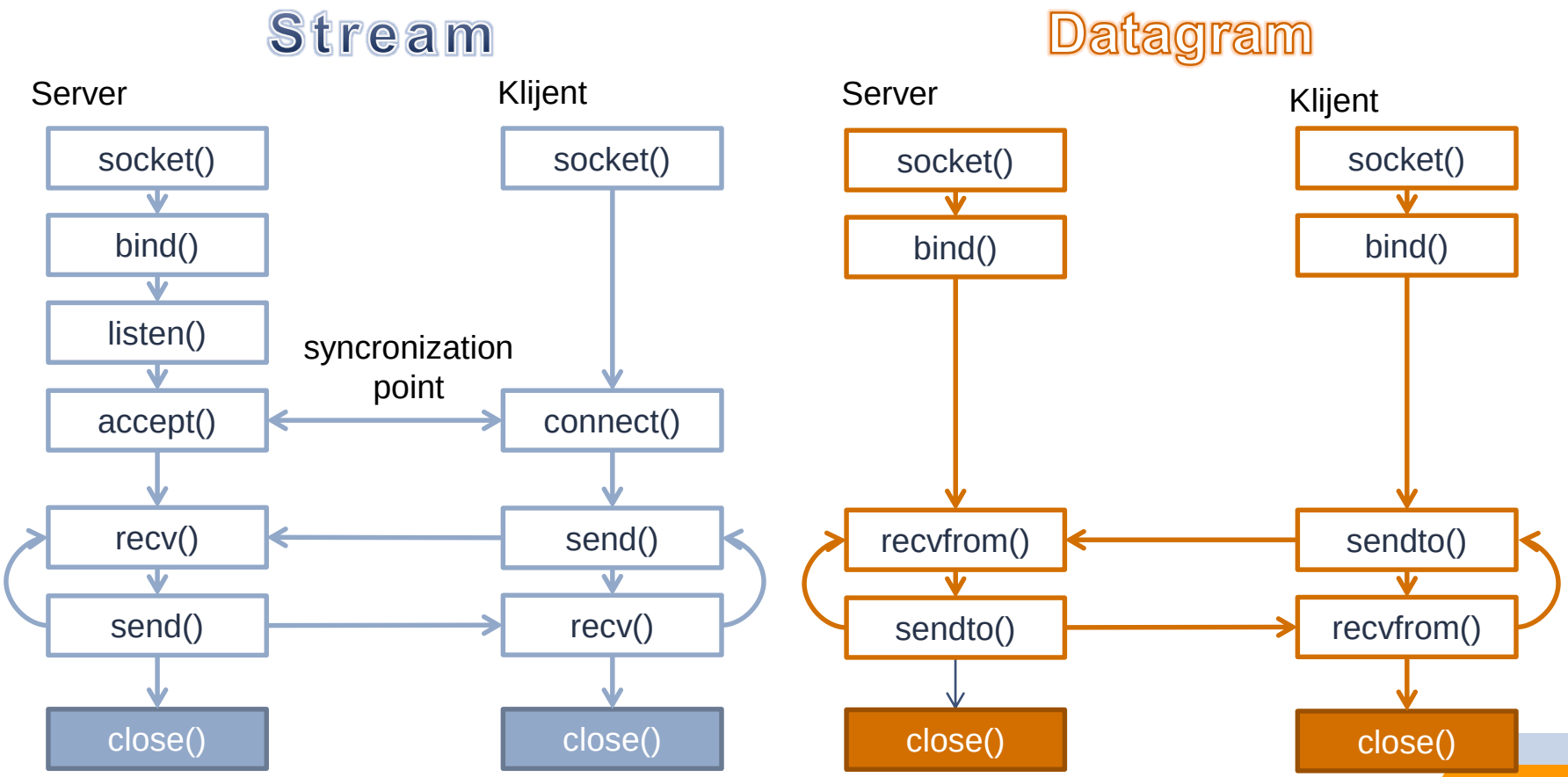
- **msg**, **msgLen**, **flags**, **count** - isto kao kod send()
- **foreignAddr** - adresa destinacije
- **addrLen** - sizeof(foreignAddr)
- sendto() je blokirajuća funkcija!

Razmena podataka – Stream soketi

int count = recvfrom(sockid, recvBuf, bufLen, flags, &clientAddr, addrLen)

- **recvBuf**, **bufLen**, **flags**, **count** - isto kao kod recv()
- **clientAddr** - adresa klijenta
- **addrLen** - sizeof(clientAddr)
- recvfrom() je blokirajuća funkcija!

Zatvaranje soketa



Zatvaranje soketa

- Nakon završetka rada sa soketom, isti je potrebno zatvoriti

auto status = close(sock)

  - **sock** – deskriptor soketa koji se zatvara
  - **status** – 0 ako je uspešno, -1 ako je došlo do greške
- Zatvaranje soketa
  - zatvara konekciju za stream sokete
  - oslobađa port koji je soket koristio

Funkcija inet_addr( )

Funkcija **inet_addr** konvertuje string koji sadrži IPv4 adresu u “tačkastoj” notaciji u odgovarajuću IN_ADDR strukturu

unsigned long inet_addr (const char FAR * *cp*);

- **cp** – IPv4 adresa u “.” notaciji

Primeri

Echo server i klijent preko stream i datagram soketa

Echo Server – Stream soketi

```
#include <iostream>
#include <string>
#include <winsock.h>
#pragma comment(lib, "wsock32.lib")
#define SERVER_PORT 8888
#define BUF_SIZE 1024

void ExitWithError(const std::string& message) {
        std::cout << message << "Error code:" << WSAGetLastError() << std::endl;
        WSACleanup();
        exit(1);
}
void printMessage(const std::string& message, int len) {
        std::cout << "Primljena poruka:";
        for (auto i = 0; i < len; std::cout << message[i++]);
        std::cout << std::endl;
}
```

dodavanje biblioteke
pomoćne funkcije

Echo Server – Stream soketi

```
int main() {
	// Inicijalizacija winsoketa
	WSAData wsa;
	SOCKET listensock;
	if (WSAStartup(0x0202, &wsa) != 0)
		ExitWithError("Startup failed.");

	// Kreiranje listen soketa
	if ((listensock = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
		ExitWithError("Listening socket not created");
```

inicijalizacija winsocket-a
kreiranje listen soketa

Echo Server – Stream soketi

```
int main() {
*** NASTAVAK
// Dodela porta listen soketu, bind
    sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(SERVER_PORT);

    if (bind(listensock, (struct sockaddr *)&server, sizeof(server)) == SOCKET_ERROR)
        ExitWithError("bind() failed!");

//Osluskivanje
    if ((listen(listensock, 3)) < 0)
        ExitWithError("listen() failed!");
***
```

Dodela porta listen soketu
Osluškivanje

Echo Server – Stream soketi

```
int main() {
*** NASTAVAK
    while (true) {
            sockaddr_in client;
            int cLen = sizeof(struct sockaddr_in);
            SOCKET clientSock;
            if ((clientSock = accept(listensock, (struct sockaddr *)&client, &cLen)) < 0)
                        HandleError("accept() failed!");
            // Primanje poruke od klijenta
            std::vector<char> echoBuf(BUF_SIZE);
            int recvMsgSize = recv(clientSock, echoBuf.data(), echoBuf.size(), 0);

            if (recvMsgSize < 0)
                         HandleError("recv() failed");
            printMessage(echoBuf.data(), recvMsgSize);
```

petlja za prihvatanje novih konekcija komunikacija

Echo Server – Stream soketi

Komunikacija
Zatvaranje soketa

```
int main() {
*** NASTAVAK
            ***// Prosledjivanje primljenog stringa sve do prekida transmisije
            while (recvMsgSize > 0)
            {
                if (send(clientSock, echoBuf.data(), recvMsgSize, 0) != recvMsgSize)
                        HandleError("send() failed");
                if ((recvMsgSize = recv(clientSock, echoBuf.data(), echoBuf.size(), 0)) < 0)
                        HandleError("recv() failed");
                printMessage(echoBuf.data(), recvMsgSize);
            } // while (recvMsgSize > 0)
            closesocket(clientSock);
} //while (true)
closesocket(listensock);
WSACleanup();
```

Echo Klijent– Stream soketi

```
int main() {
	WSAData wsa;
	SOCKET clientSocket;
	if (WSAStartup(0x0202, &wsa) != 0)
		ExitWithError("Startup failed.");
	if ((clientSocket = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
		ExitWithError("Listening socket not created");
	sockaddr_in server;
	server.sin_family = AF_INET;
	server.sin_addr.s_addr = inet_addr(serverAddr);
	server.sin_port = htons(serverPort);
	if (connect(clientSocket, (struct sockaddr *)&server, sizeof(server)) == SOCKET_ERROR)
		ExitWithError("connect() failed!");
```

inicijalizacija winsocket-a
kreiranje soketa
povezivanje

Echo Klijent – Stream soketi

```
int main() {
*** NASTAVAK
      while (true) {
            std::string echoBuf;
            std::getline(std::cin, echoBuf);
            if (send(clientSocket, echoBuf.c_str(), echoBuf.length(), 0) != echoBuf.length())
                        ExitWithError("send() failed");
            int recvMsgSize = 0;
            std::vector<char> buff(BUF_SIZE);
            do {
                        if ((recvMsgSize = recv(clientSocket, buff.data(), buff.size(), 0)) < 0)
                                    ExitWithError("recv() failed");
                        printMessage(buff.data(), recvMsgSize);
            } while (recvMsgSize == BUF_SIZE);
      }
      closesocket(listensock);
      WSACleanup();
}
```

Komunikacija
Zatvaranje soketa

Echo Server – Datagram soketi

```
int main()
{
	WSAData wsa;
	SOCKET serverSock;
	if (WSAStartup(0x0202, &wsa) != 0)
		ExitWithError("Startup failed.");
	if ((serverSock = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET)
		ExitWithError("Listening socket not created");
	sockaddr_in server;
	server.sin_family = AF_INET;
	server.sin_addr.s_addr = (INADDR_ANY);
	server.sin_port = htons(serverPort);
	if (bind(serverSock, (struct sockaddr *)&server, sizeof(server)) == SOCKET_ERROR)
		ExitWithError("bind() failed!");
```

inicijalizacija winsocket-a
kreiranje soketa
povezivanje

Echo Server – Datagram soketi

```
int main() {
	*** NASTAVAK
	sockaddr_in client;
	int addrlen = sizeof(sockaddr);
	while (true) {
		char echoBuf[bufSize];
		int recvMsgSize = recvfrom(serverSock, echoBuf, bufSize, 0, (sockaddr*)&client, &addrlen);
		if (recvMsgSize < 0)
			ExitWithError("recv() failed");
		printMessage(echoBuf, recvMsgSize);
		if (sendto(serverSock, echoBuf, recvMsgSize, 0, (sockaddr*)&client, sizeof(sockaddr)) != recvMsgSize)
			ExitWithError("send() failed");
	} //while (true)
	closesocket(serverSock);
	WSACleanup();
}
```

komunikacija
zatvaranje soketa

Echo Klijent– Datagram soketi

```
int main() {
      WSAData wsa;
      SOCKET clientSocket;
      if (WSAStartup(0x0202, &wsa) != 0)
            ExitWithError("Startup failed.");
      if ((clientSocket = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET)
            ExitWithError("Listening socket not created");

      sockaddr_in server;
      server.sin_family = AF_INET;
      server.sin_addr.s_addr = inet_addr(serverAddr);
      server.sin_port = htons(serverPort);
      std::string echoBuf;
      int sz = sizeof(sockaddr);
****
```

inicijalizacija winsocket-a
kreiranje soketa
specifikacija adrese

Echo Klijent– Datagram soketi

```
int main() { *** NASTAVAK
while (true)
{
    std::getline(std::cin, echoBuf);
    if (sendto(clientSocket, echoBuf.c_str(), echoBuf.length(), 0, (sockaddr *)&server, sz) != echoBuf.length())
           ExitWithError("send() failed");
    int recvMsgSize = 0;
    char buff[bufSize];
    if ((recvMsgSize = recvfrom(clientSocket, buff, bufSize, 0, (sockaddr *)&server, &sz)) < 0)
           ExitWithError("recv() failed");
    printMessage(buff, recvMsgSize);
}
closesocket(clientSocket);
WSACleanup();
}
```

inicijalizacija winsocket-a
kreiranje soketa