Programska potpora komunikacijskim sustavima

9. predavanje, 17. svibnja 2023.

doc. dr. sc. Jelena Božek

Protokoli HTTP i TCP

SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

- Predavanja izradio prof. dr. sc. Krešimir Pripužić, 2022.
- Predavanja doradila doc. dr. sc. Jelena Božek, 2023.

Sadržaj predavanja

- Protokol HTTP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom HTTP
- Protokol TCP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom TCP

Protokol HTTP

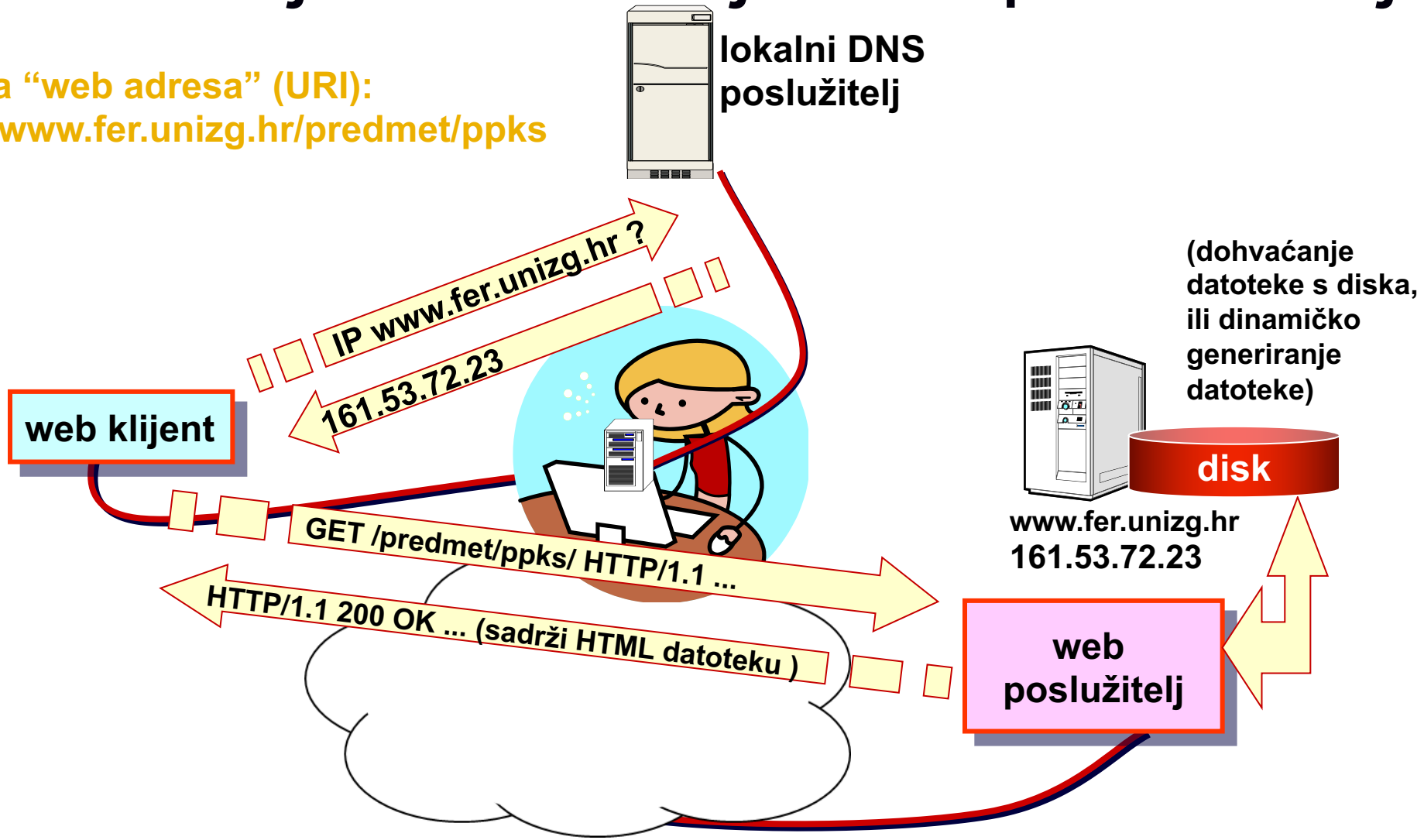
SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Protokol *Hypertext Transfer Protocol (HTTP)*

- internetski protokol aplikacijskog sloja
- definira format i način razmjene poruka
  - tekstualan zapis
- vrste poruka:
  - zahtjev (“metoda” - naziv potječe od terminologije iz područja objektno-orijentiranog programiranja)
    - definira operaciju (metodu), resurs, protokol
  - odgovor (ishod zahtjeva i rezultat)
    - ishod zahtjeva (uspjeh, neuspjeh, greška,...) opisan statusnim kôdom
    - za neke vrste zahtjeva, kao rezultat uspješnog ishoda, u tijelu odgovora dostavlja se sadržaj zatraženog resursa

Komunikacija HTTP klijenta i poslužitelja

Odabrana “web adresa” (URI):
http://www.fer.unizg.hr/predmet/ppks



Specifikacije protokola HTTP [1]

- prva verzija HTTP/0.9 - ograničene mogućnosti
  - podržava prijenos samo hipertekstualnih dokumenata
- **HTTP/1.0 -** prvi (*Informational*) RFC (RFC 1945), 1996.
  - podržava prijenos različitih tipova podataka
    - posuđuje koncepte iz *media type* standarda
  - zadržava kompatibilnost unazad
  - sredinom 90-tih - HTTP promet dominira - HTTP/1.0 neučinkovit
    - svako sjedište mora biti na drugom poslužitelju
    - preko jedne konekcije ostvaruje se jedan HTTP zahtjev
    - nema podrške za upravljanje performansama - *cache*, *proxy*, djelomični dohvat
  - WWW prozvan “World Wide Wait”

Specifikacije protokola HTTP [2]

- **HTTP/1.1** - RFC 2616, lipanj 1999.
  - zadržava kompatibilnost unazad prema HTTP 1.1
  - RFC 2617 - autentifikacija i sigurnost
  - poboljšanja:
    - jedno fizičko računalo - više Web poslužitelja - "virtualni host"
    - trajne konekcije - više zahtjeva preko jedne TCP konekcije
    - djelomični dohvat sadržaja
    - bolja podrška za priručna spremišta (*cache*) i zastupnike(*proxy*)
    - pregovaranje o sadržaju datoteke (*content negotiation*)
    - bolji sigurnosni mehanizmi – autentifikacija
- Postoje i novije verzije: HTTP/2 (2015. godine) i HTTP/3 (predloženi standard, lipanj 2022.)

HTTP poruke

- zahtjev i odgovor moraju biti ispravno formatirani

- HTTP definira opći format poruke
  - tekstualan zapis
  - naslanja se na format e-mail poruke (RFC 822) i *media type* standarda
    - dijele neka načela, ali ne sasvim i ne potpuno
    - npr. ne koriste se sva *media type* zaglavlja
    - npr. tijelo ne mora biti 7-bitni ASCII

Poruke protokola HTTP

zahtjev

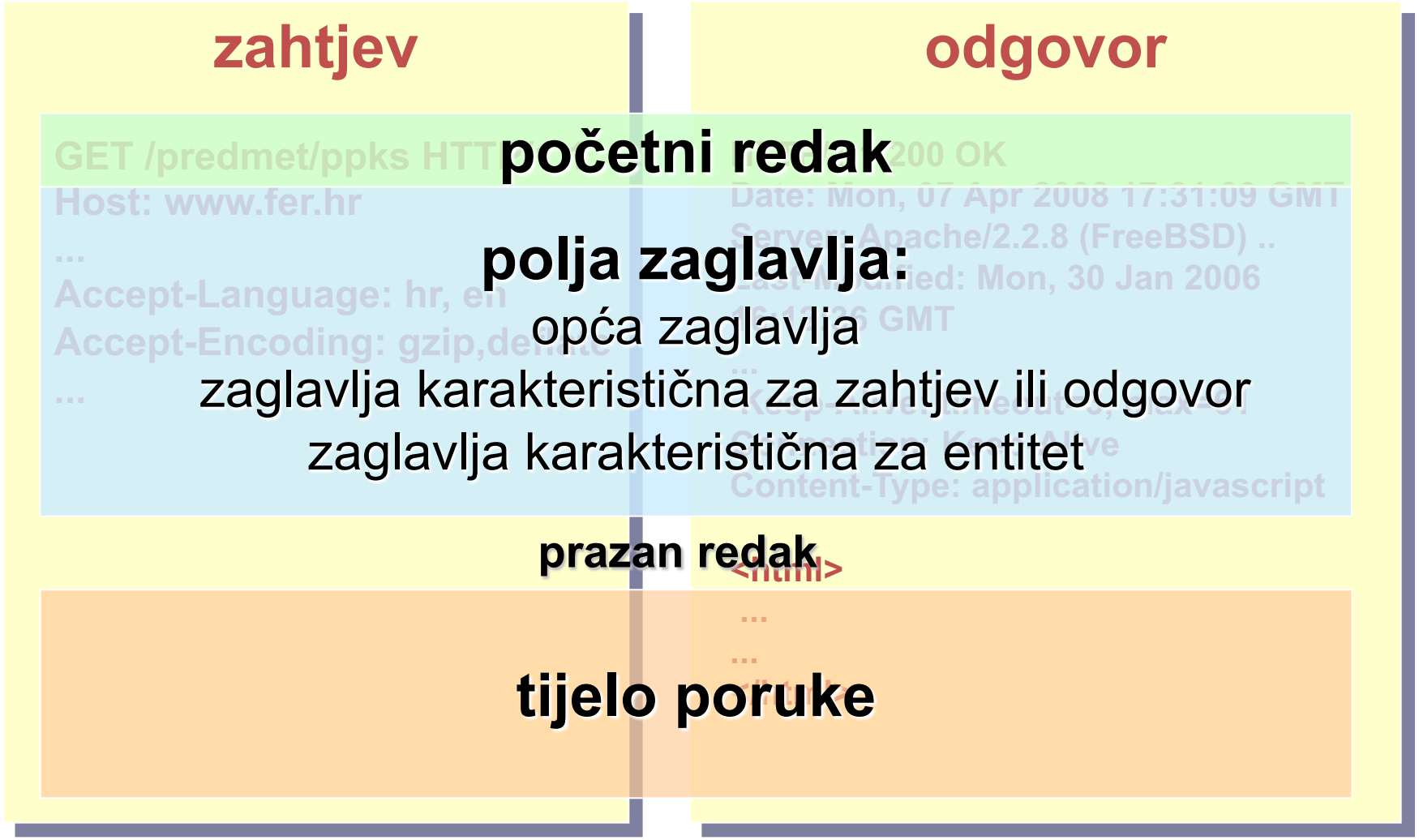
```
GET /predmet/ppks HTTP/1.1
Host: www.fer.hr
...
Accept-Language: hr, en
Accept-Encoding: gzip,deflate
...
```

odgovor

```
HTTP/1.1 200 OK
Date: Mon, 07 Apr 2022 17:31:09 GMT
Server: Apache/2.2.8 (FreeBSD) ..
Last-Modified: Mon, 30 Jan 2022
16:12:36 GMT
...
 Keep-Alive: timeout=3, max=61
Connection: Keep-Alive
Content-Type: application/javascript

<html>
 ...
...
</html>
```

Format poruka



Oblikovanje zahtjeva

- početni redak sadrži (*request line*):
  - resurs nad kojim je podnesen zahtjev
  - metodu (operacija) koja se traži nad tim resursom
  - verziju protokola koju koristi

<*metoda*> <*URI*> <*verzija*>

- primjeri:
  - GET / HTTP/1.0
  - POST /shop/order HTTP/1.1
  - HEAD /search?q=raspored HTTP/1.0

Metode zahtjeva

- metoda zahtjeva određuje *što* se traži od resursa
- HTTP/1.1 definira 8 metoda i omogućuje dodavanje novih metoda (*extensions*):
  - OPTIONS
  - GET
  - HEAD
  - POST
  - PUT
  - DELETE
  - TRACE
  - CONNECT
- važna svojstva metoda: sigurna (*safe*), indempodentna (*idempotent*), pamtljiv odgovor (*cachable*)

Zaglavlja zahtjeva

METHOD:
- GET
- POST
- HEAD
- PUT
- OPTIONS
- TRACE
- CONNECT
- DELETE

Method Request-URI HTTP-Version
parametar1: *vrijednost*
parametar2: *vrijednost*
parametar3: *vrijednost*
....
<prazni redak>

HTTP/1.0
HTTP/1.1

general_header
Cache-Control:
Connection:
Date:
Pragma:
Trailer:
Transfer-Encoding:
Upgrade:
Warning:

request_header
Accept:
Accept-Charset:
Accept-Encoding:
Accept-Language:
Authorization:
Expect:
From:
Host:
If-Match:
If-Modified-Since:
If-None-Match:
If-Range:
If-Unmodified-Since:
Max-Forwards:
Proxy-Authorization:
Range:
Referer:
TE:
User-Agent:

entity_header
Allow:
Content-Encoding:
Content-Language:
Content-Length:
Content-Location:
Content-MD5:
Content-Range:
Content-Type:
Expires:
Last-Modified:
extension-header

Odgovor poslužitelja

- **početni redak** sadrži:
  - verziju protokola
  - statusni kôd
  - opisnu frazu

HTTP/1.1 303 See Other
HTTP/1.1 200 OK
HTTP/1.1 404 Not Found

- u **tijelu** odgovora se obično prenosi reprezentacija resursa (“entitet”) koju preglednik treba prikazati korisniku
- neka polja zaglavlja:
  - **Content-Type:** format entiteta
  - **Content-Length:** duljina entiteta u tijelu u oktetima

Statusni kôd odgovora

- sastoji se od tri dekadske znamenke

- pet kategorija:
  - 1xx – **Informativne** - ne naznačuju ni uspjeh, ni neuspjeh
  - 2xx – **Uspjeh** - poslužitelj je primio, razumio i ispunio zahtjev
  - 3xx – **Preusmjeravanje** - potrebno poduzeti dodatne akcije
  - 4xx – Greška na **klijentu** - zahtjev je neispravan
  - 5xx – Greška na **poslužitelju** - zahtjev je ispravan, ali poslužitelj ga ne može ispuniti

Primjer HTTP klijenta

- Napravit ćemo klijenta za REST poslužitelja s prethodnog predavanja korištenjem klase `java.net.http.HttpClient`
- Implementirat ćemo samo HTTP metodu GET
  - GET /persons treba dohvatiti listu osoba
  - GET /persons/4 treba dohvatiti osobu čiji je ID jednak 4

Proširenje klase `Person`

```
...
import com.google.gson.Gson;

public class Person {

    private static final Gson gson = new Gson();

...

    public String toJson() {
        return gson.toJson(this);
    }

    public static Person fromJson(String json) {
        return (Person) gson.fromJson(json, Person.class);
    }


    public static String listToJson(List<Person> list) {
        return gson.toJson(list);
    }

    public static List<Person> listFromJson(String json) {
        Type listOfPerson = new TypeToken<List<Person>>() {}.getType();
        List<Person> list = gson.fromJson(json, listOfPerson);
        return list;
    }
}
```

Klasa PersonsHTTPGetClient [1]

```
public class PersonHTTPGetClient implements PersonInterface {

    private final String schemeHostPort;
    private final HttpClient httpClient;

    public PersonHTTPGetClient(String host, int port) {
        schemeHostPort = "http://" + host + ":" + port;
        httpClient = HttpClient.newBuilder().build();
    }


    public static void main(String[] args) throws Exception {

        PersonHTTPClient personHTTPClient = new PersonHTTPClient("localhost", 8000);

        //read resources
        List<Person> list = personHTTPClient.get();
        System.out.println(list);
        System.out.println("");

        ...
    }
}
```

Klasa PersonsHTTPGetClient [2]

```
@Override
public List<Person> get() {
    try {
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(schemeHostPort + "/persons"))
                .GET()
                .build();
        HttpResponse response = httpClient.send(request, BodyHandlers.ofString());
        System.out.println("Read all persons:");
        System.out.println(response);
        System.out.println(response.body());
        List<Person> list = Person.listFromJson(response.body().toString());
        return list;
    } catch (IOException | InterruptedException ex) {
        return null;
    }
}
```

Klasa PersonsHTTPGetClient [3]

```
@Override
public Person get(long id) {
    try {
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(schemeHostPort + "/persons/" + id))
                .GET()
                .build();
        HttpResponse response = httpClient.send(request, BodyHandlers.ofString());
        System.out.println("Read person with id=" + id + ":");
        System.out.println(response);
        System.out.println(response.body());
        Person person = (Person) Person.fromJson(response.body().toString());
        return (response.statusCode() == 200) ? person : null;
    } catch (IOException | InterruptedException ex) {
        return null;
    }
}
```

Klasa `PersonsHTTPClient`

- Ova klasa je proširenje klase `PersonsHTTPGetClient`
- Predstavlja potpuno implementiranog klijenta za rad s REST-uslugom s prethodnog predavanja
  - Sadrži sve metode za upravljanje osobama kao resursima, a ne samo metode za dohvaćanje resursa (kao kod klase `PersonsHTTPGetClient`)
- Testira ispravan rad aplikacije s prethodnog predavanja

Primjer HTTP poslužitelja

- Napravit ćemo REST poslužitelja s prethodnog predavanja korištenjem klase `com.sun.net.httpserver.HttpServer`
- Implementirat ćemo samo HTTP metodu GET
  - GET /persons treba vratiti listu osoba
  - GET /persons/4 treba vratiti osobu čiji je ID jednak 4

Klasa PersonsHTTPGetServer

```
public class PersonsHTTPGetServer {

    private final PersonService personService;
    private final HttpServer httpServer;

    public PersonsHTTPGetServer(int port) throws IOException {
        personService = new PersonService();
        httpServer = HttpServer.create(new InetSocketAddress(port), 0);
        httpServer.createContext("/persons", new PersonsHTTPGetServer.PersonsHandler());
    }

    public void start() {
        httpServer.start();
    }

    public static void main(String[] args) throws IOException {
        PersonsHTTPGetServer server = new PersonsHTTPGetServer(8000);
        server.start();
    }
```

Unutarnja klasa PersonsHandler [1]

```
class PersonsHandler implements HttpHandler {

    @Override
    public void handle(HttpExchange httpExchange) throws IOException {
        switch (httpExchange.getRequestMethod()) {
            case "GET":
                handleGet(httpExchange);
                break;
            case "POST":
                handlePost(httpExchange);
                break;
            case "PUT":
                handlePut(httpExchange);
                break;
            case "DELETE":
                handleDelete(httpExchange);
                break;
        }
    }
```

Unutarnja klasa PersonsHandler [2]

```
private void handleGet(HttpExchange httpExchange) throws IOException {
    URI uri = httpExchange.getRequestURI();
    byte[] responseBody = new byte[0]; //default body for malformed requests
    int responseCode = 404; //default status code for malformed requests

    if (uri.getPath().equals("/persons")) {//return all persons
        responseBody = Person.listToJson(personService.get()).getBytes("UTF-8");
        responseCode = 200;
    } else if (uri.getPath().matches("/persons/[0-9]*")) {//return person with given id
        String[] split = uri.getPath().split("/");
        Person person = personService.get(Integer.parseInt(split[2]));
        if (person != null) {
            responseBody = person.toJson().getBytes("UTF-8");
            responseCode = 200;
        }
    }
    httpExchange.getResponseHeaders().set("content-type", "application/json");
    httpExchange.sendResponseHeaders(responseCode, responseBody.length);
    try (OutputStream os = httpExchange.getResponseBody()) { //write and close the response body
        os.write(responseBody);
    }
}
```

Unutarnja klasa PersonsHandler [3]

```
    private void handlePost(HttpExchange httpExchange) throws IOException {
        //return status code 501 - no implemented
        httpExchange.sendResponseHeaders(501, -1);
    }

    private void handlePut(HttpExchange httpExchange) throws IOException {
        //return status code 501 - no implemented
        httpExchange.sendResponseHeaders(501, -1);
    }

    private void handleDelete(HttpExchange httpExchange) throws IOException {
        //return status code 501 - no implemented
        httpExchange.sendResponseHeaders(501, -1);
    }
}
}
```

Protokol TCP

SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

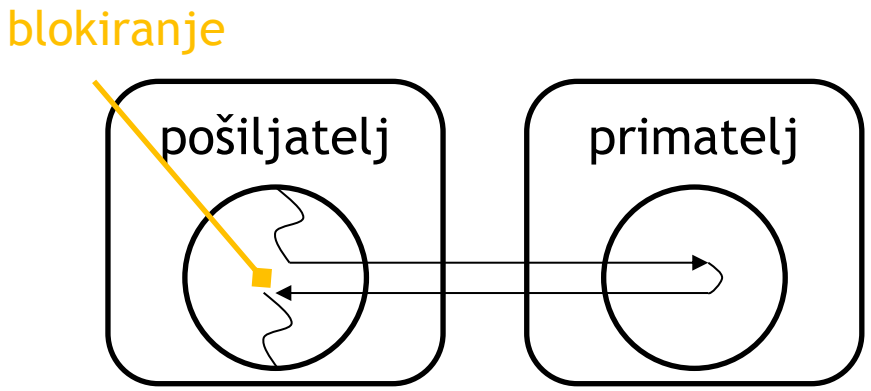
Obilježja komunikacije [1]

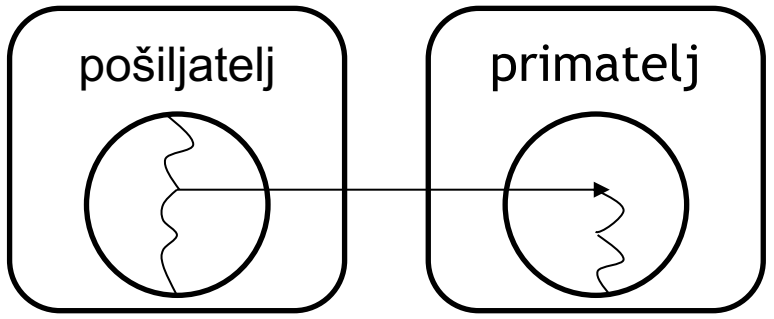
- konekcijska
  - procesi eksplicitno kreiraju konekciju prije razmjene podataka, postoje kontrolne poruke za uspostavu konekcije
- bezkonekcijska
  - sve poruke prenose podatke, nema kontrolnih poruka za uspostavu konekcije među procesima

- perzistentna komunikacija
  - garantira isporuku poruke, poruka se pohranjuje u sustavu i isporučuje primatelju kada je to moguće
- tranzijentna komunikacija
  - nepouzdana, garantira isporuku poruke samo ako su pošiljatelj i primatelj poruke istovremeno dostupni

Obilježja komunikacije [2]

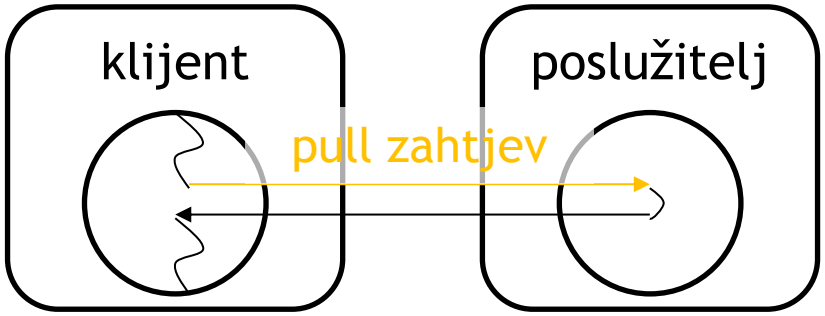
- sinkrona komunikacija
  - blokira pošiljatelja do primitka potvrde od strane primatelja
- asinkrona komunikacija
  - omogućuje pošiljatelju nastavak obrade odmah nakon slanja poruke

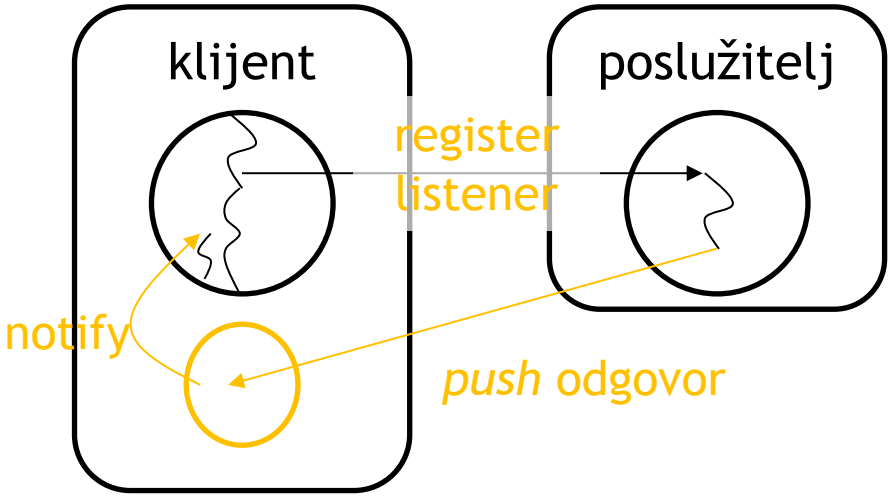




Obilježja komunikacije [3]

- *pull*
  - "klasični" model zahtjev-odgovor
- *push*
  - klijent registrira zahtjev i "sluša" odgovor, poslužitelj šalje odgovor nakon što završi obradu zahtjeva





Obilježja komunikacije protokolom TCP

- vremenska ovisnost
  - klijent i poslužitelj moraju biti istovremeno dostupni
- tranzijentna komunikacija
- konekcijska komunikacija
- sinkrona komunikacija
  - klijent šalje zahtjev za kreiranje konekcije i proces je blokiran do uspostave konekcije
- pokretanje komunikacije na načelu *pull*
  - klijent mora znati identifikator poslužitelja

Sloj raspodijeljenog sustava za komunikaciju među procesima

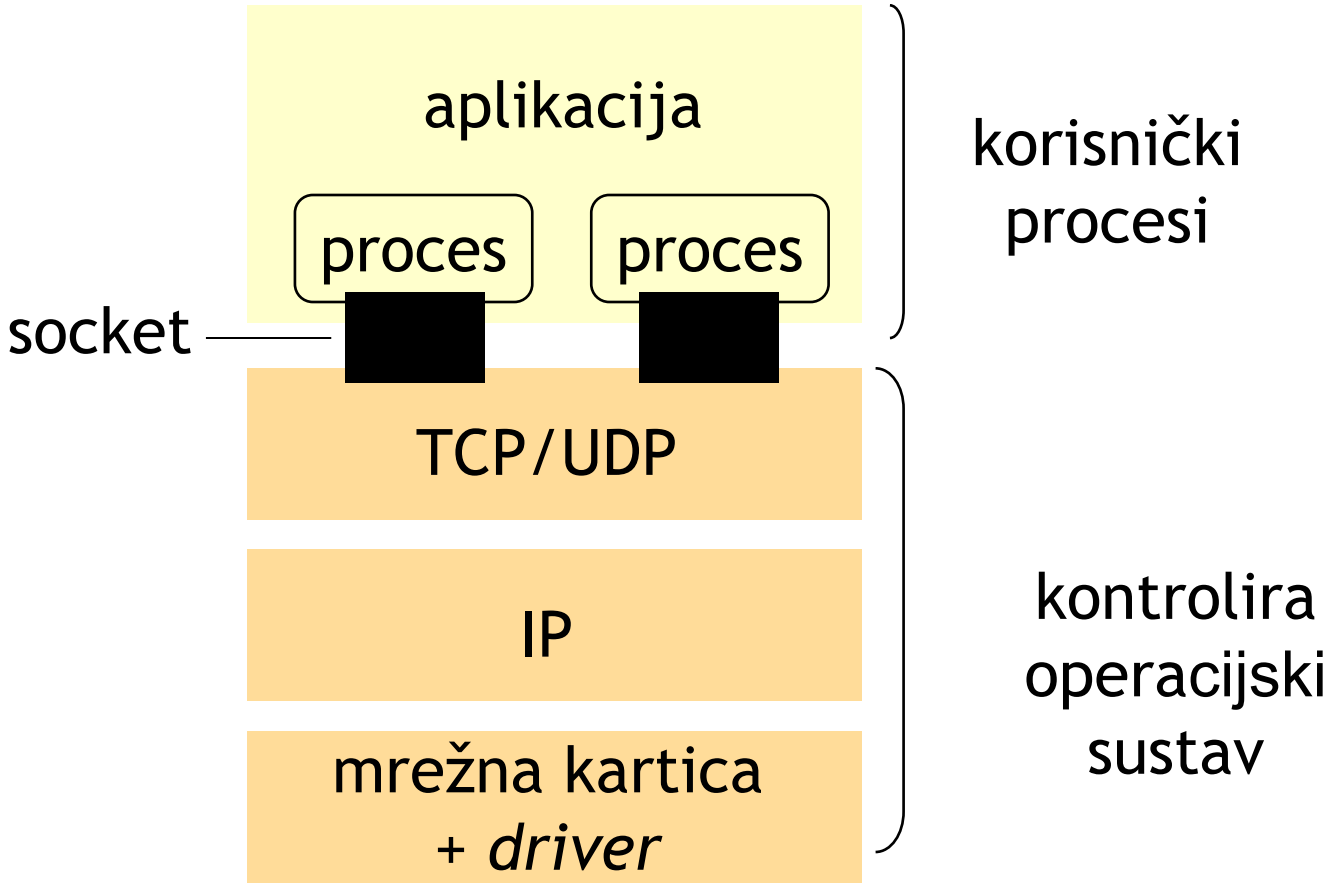
- Postojeća rješenja za komunikaciju raspodijeljenih procesa
  1. komunikacija korištenjem priključnica (*socket API*)
  2. poziv udaljene procedure (*Remote Procedure Call,* RPC)
  3. raspodijeljeni objekti - poziv udaljene metode (*Remote Method Invocation,* RMI)
  4. komunikacija razmjenom poruka (*message-oriented interaction*)
  5. model objavi-pretplati (*publish/subscribe*)
  6. REST (*REpresentational State Transfer*)
  7. SOAP (*Simple Object Access Protocol*)

Komunikacija korištenjem priključnica

Socket API

- koristi funkcionalnost transportnog sloja
  - TCP – konekcijski protokol, pouzdan prijenos podataka
  - UDP – prijenos nezavisnih paketa (*datagrami*), nepouzdan prijenos
- priključnica (*socket*)
  - pristupna točka preko koje aplikacija šalje podatke u mrežu i iz koje čita primljene podatke
  - viši nivo apstrakcije nad komunikacijskom točkom koju operacijski sustav koristi za pristup transportnom sloju
  - veže se uz vrata (*port*) koja jednoznačno određuju aplikaciju kojoj su poruke namijenjene

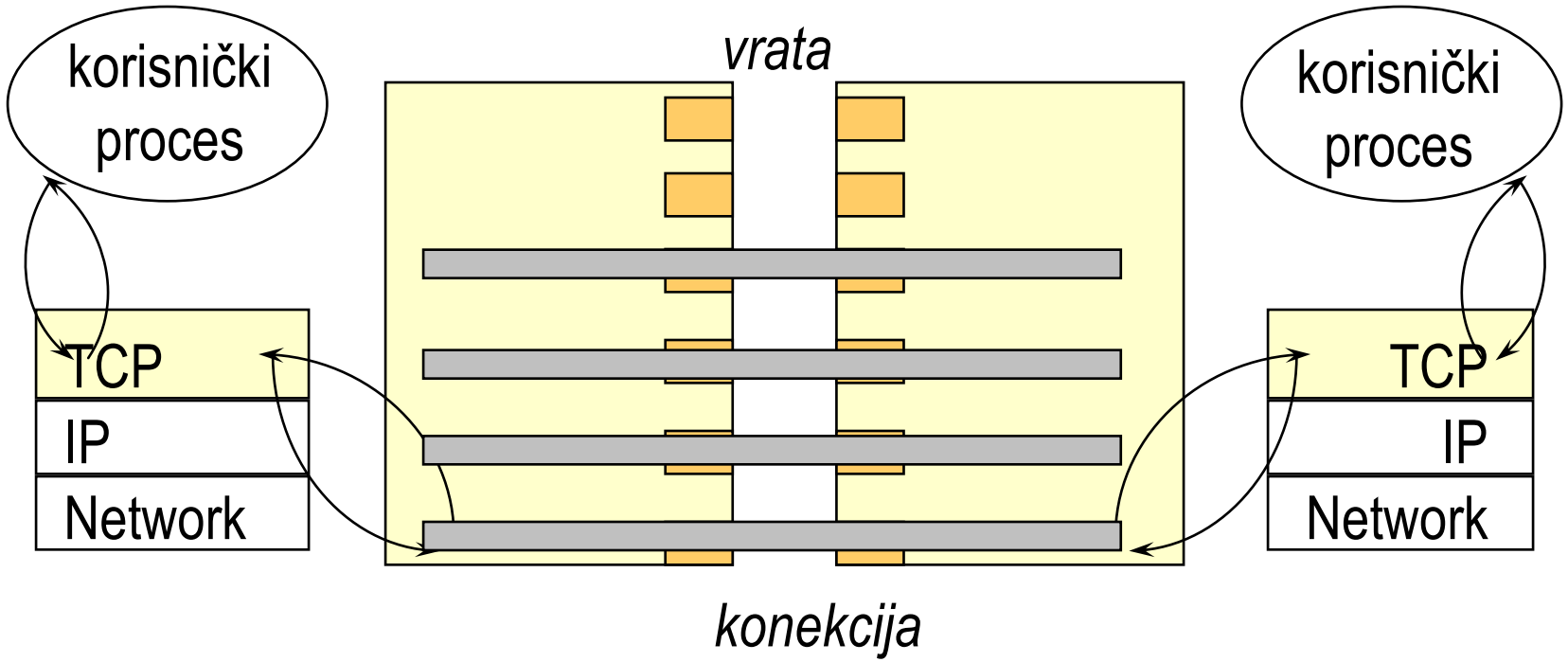
Komunikacija pomoću priključnica



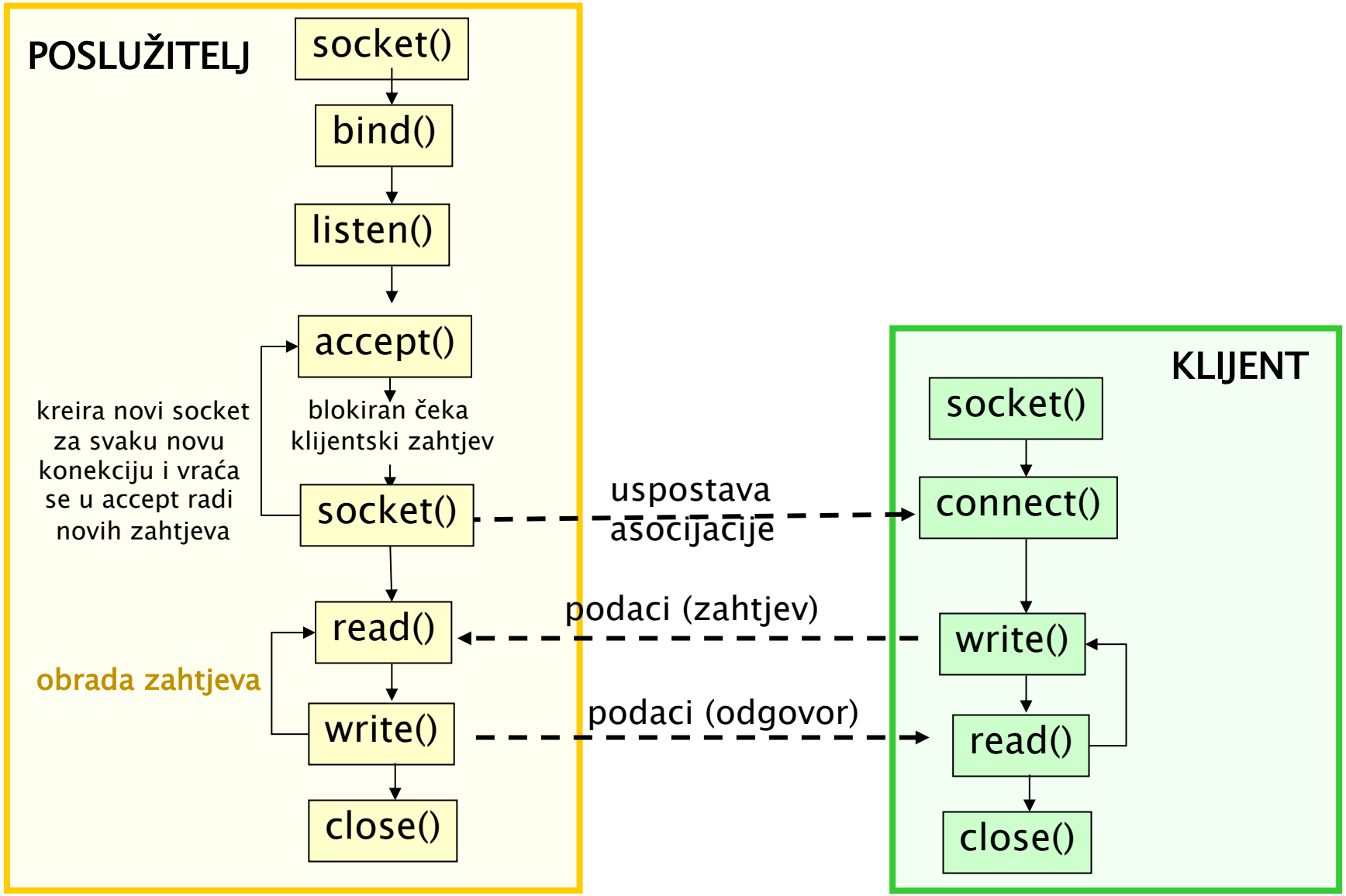
Transportni protocol TCP

Transmission Control Protocol (TCP)

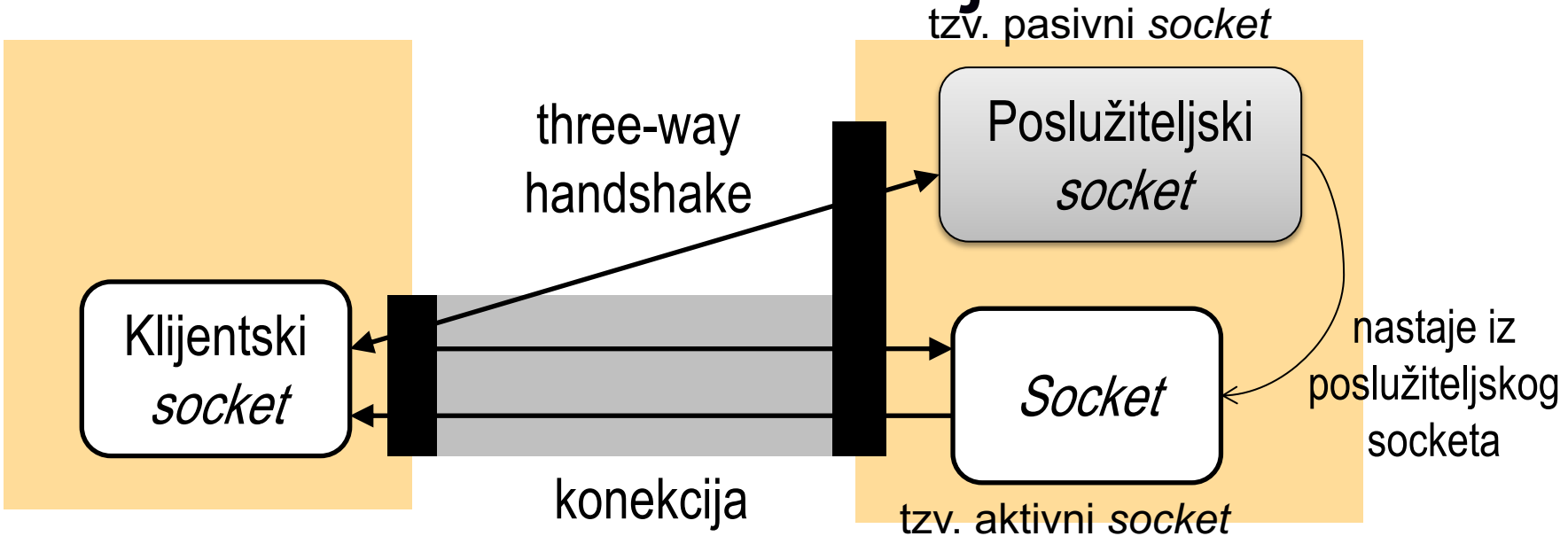
- konekcija između dvije krajnje točke koje se moraju dogovoriti o uspostavi konekcije



Konekcijska komunikacija pomoću *socketa* TCP



Konkurentni korisnički zahtjevi



- za svaki novi korisnički zahtjev kreira se **novi *socket*** (s dva *buffer*-a, *in* i *out*) **koji se veže uz konekciju** <poslužiteljska IP adresa i broj vrata (broj vrata ostaje isti kao za poslužiteljski *socket*), klijentska IP adresa i broj vrata>
- originalni poslužiteljski *socket* mora konstantno biti u stanju “osluškivanja”

TCP: implementacija klijenta

1. Kreirati klijentski socket:

```
clientSocket = new Socket( address, port );
```

2. Kreirati I/O stream za komunikaciju s poslužiteljem:

```
InputStream is = clientSocket.getInputStream();
OutputStream os = clientSocket.getOutputStream();
```

3. Komunikacija s poslužiteljem:

```
//Receive byte[] b from server:
int length = is.read(b);

//Send byte[] b to server:
os.write(b);
```

4. Zatvoriti socket:

```
clientSocket.close();
```

TCP: implementacija poslužitelja

1. Kreirati socket poslužitelja:

```
ServerSocket serverSocket;
serverSocket = new ServerSocket( PORT );
```

2. Čekati korisnički zahtjev (blokira proces do klijentskog zahtjeva!!!) i kreirati kopiju originalnog socketa:

```
Socket clientSocket = serverSocket.accept();
```

3. Kreirati I/O stream za komunikaciju s klijentom

```
InputStream is = clientSocket.getInputStream();
OutputStream os = clientSocket.getOutputStream();
```

4. Komunikacija s klijentom
5. Zatvoriti kopiju socketa:

```
clientSocket.close();
```

6. Zatvoriti poslužiteljski socket:

```
serverSocket.close();
```

Primjer TCP klijenta i poslužitelja

- Klijent šalje senzorska očitanja u JSON formatu
- Server parsira primljena senzorska očitanja i ispisuje ih na konzolu

Klasa TCPClient [1]

```
public class TCPClient {
    public static void main(String[] args) {
        try ( Socket clientSocket = new Socket("localhost", 12345); //SOCKET -> CONNECT
                    BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(
clientSocket.getOutputStream(), "UTF-8")); //
                    BufferedReader reader = new BufferedReader(new InputStreamReader(
clientSocket.getInputStream(), "UTF-8"))) {

            //start sending readings – NA SLJEDEĆEM SLAJDU!!!
            while (true) {
                ...
            }
        } catch (IOException ex) {
            Logger.getLogger(TCPClient.class.getName()).log(Level.SEVERE, null, ex);
        } //CLOSE
    }
}
```

Klasa `TCPClient` [2]

```
//start sending readings
while (true) {
    //send a reading
    Reading reading = new Reading("id_5", "temperature", (Math.random() * 60) - 20, "C");
    writer.write(reading.toJson()); writer.newLine(); //WRITE
    writer.flush();
    System.out.println("Sent: " + reading);

    //receive confirmation
    String confirmation = reader.readLine(); //READ
    System.out.println("Received: " + confirmation);

    try {
        Thread.sleep(5000);
    } catch (InterruptedException ex) {
        //do nothing
    }
}
```

Klasa SingleClientTCPServer [1]

```
public class SingleClientTCPServer {

    private int port;

    public SingleClientTCPServer(int port) {
        this.port = port;
    }

    public static void main(String[] args) {
        SingleClientTCPServer server = new SingleClientTCPServer(12345);
        server.start();
    }
```

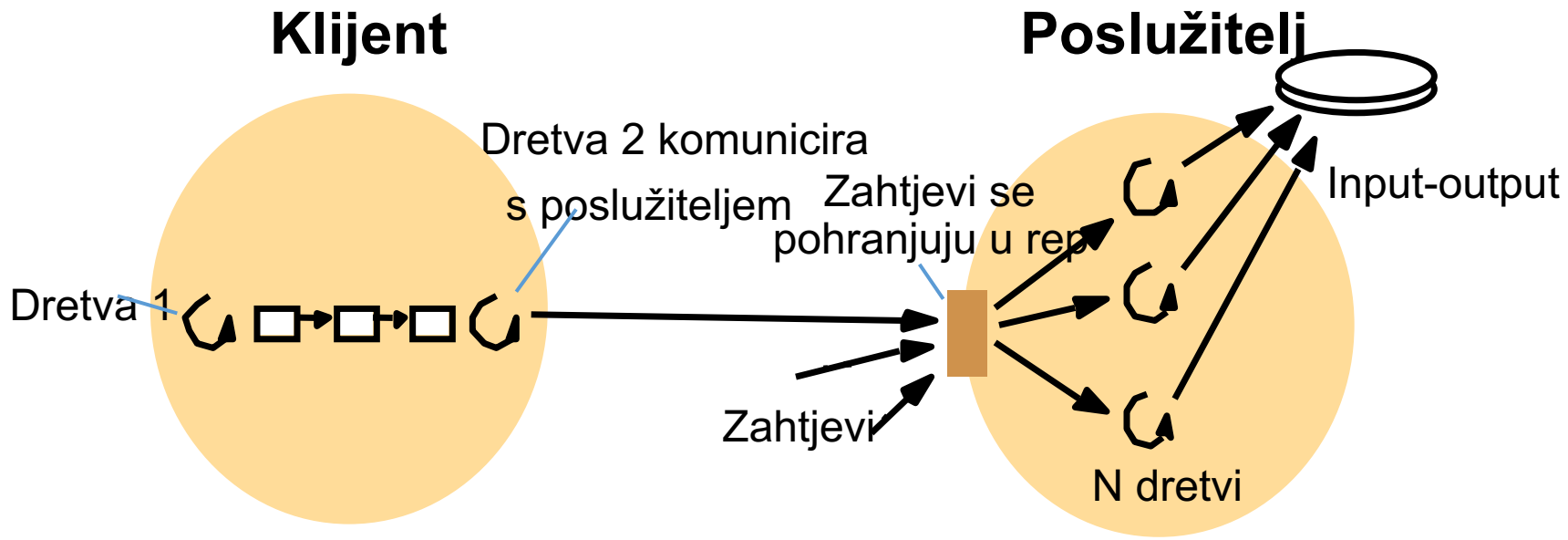
Klasa SingleClientTCPServer [2]

```
public void start() {
        try ( ServerSocket serverSocket = new ServerSocket(port); //SOCKET -> BIND -> LISTEN
                Socket clientSocket = serverSocket.accept(); //ACCEPT -> SOCKET
                BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(), "UTF-8")); //
                BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream(), "UTF-8"))) {

            //start consuming readings
            String jsonReading;
            while ((jsonReading = reader.readLine()) != null) { //READ
                //receive a reading
                Reading reading = Reading.fromJson(jsonReading);
                System.out.println("Received: " + reading);

                //send a confirmation
                String confirmation = Messages.CONFIRM;
                writer.write(confirmation); writer.newLine(); writer.flush(); //WRITE
                System.out.println("Sent: " + confirmation);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } //CLOSE
    }
}
```

Višedretveni poslužitelj i klijenti



Uobičajene zadaće na strani klijenta:

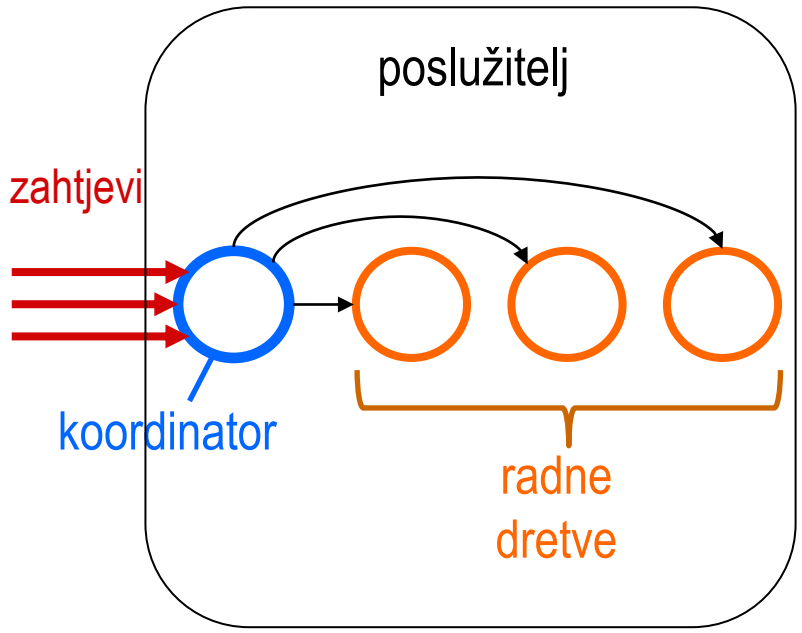
- korisničko sučelje,
- otvaranje mrežne konekcije i primanje podataka

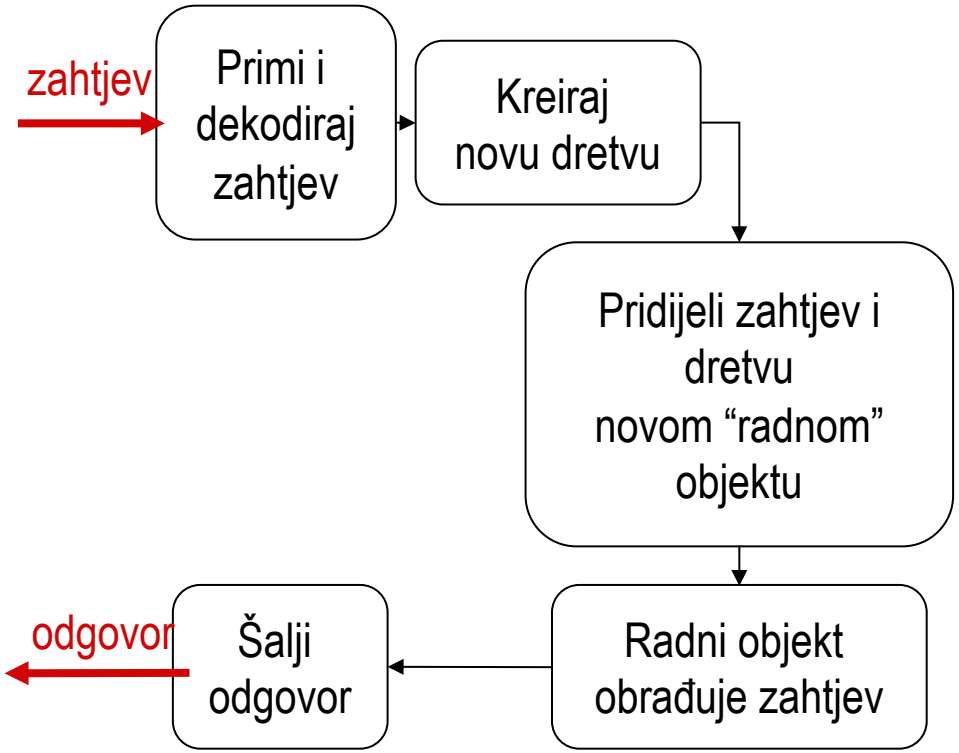
Uobičajene zadaće na strani poslužitelja:

- primanje konkurentnih klijentskih zahtjeva
- složena obrada podataka
- rad s diskom/bazom podataka

Višedretveni poslužitelj

Model koordinator/radna dretva (*dispatcher/worker model*)





Klasa TCPServer [1]

```
public class TCPServer {
    private int port;

    public TCPServer(int port) {
        this.port = port;
    }

    public void start() {
        try ( ServerSocket serverSocket = new ServerSocket(port)) { //SOCKET -> BIND -> LISTEN
            while (true) {
                try {
                    Socket clientSocket = serverSocket.accept(); //ACCEPT -> SOCKET
                    new Thread(new ServerTask(clientSocket)).start();
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } //CLOSE
    }

    public static void main(String[] args) {
        TCPServer server = new TCPServer(12345);
        server.start();
    }
```

Klasa TCPServer [2]

```
private class ServerTask implements Runnable {
    private final Socket clientSocket;

    public ServerTask(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }
    @Override
    public void run() {
        try ( BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(), "UTF-8")); //
                BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream(), "UTF-8"))) {
            //start consuming readings
            String jsonReading;
            while ((jsonReading = reader.readLine()) != null) { //READ
                //receive a reading
                Reading reading = Reading.fromJson(jsonReading);
                System.out.println("Received: " + reading);
                //send confirmation
                String confirmation = Messages.CONFIRM;
                writer.write(confirmation); writer.newLine(); writer.flush(); //WRITE
                System.out.println("Sent: " + confirmation);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
} } }
```