Sveučilište u Zagrebu
PMF – Matematički odjel

Mreže računala

Vježbe 04

Matko Botinčan
Zvonimir Bujanović
Igor Jelaska
Maja Karaga

Klijent / Server paradigma

- internet daje infrastrukturu koja omogućava komunikaciju između bilo koja 2 računala
- preciznije, komunikacija se odvija između aplikacija:
  - Firefox na jednom stroju komunicira sa web-serverom na drugom
  - traceroute aplikacija komunicira sa firmware-om u router-ima
- da bi aplikacije A i B mogla komunicirati, jedna od njih mora inicirati kontakt sa drugom – drugim riječima, jednoj treba usluga koju druga aplikacija pruža
- klijent (*client*) – aplikacija koja aktivno inicira kontakt
- server – aplikacija koja pasivno čeka kontakt

Klijent / Server paradigma

- klijentski software tipično...
  - aktivno inicira kontakt sa serverom
  - je aplikacija koja privremeno postaje klijent kada joj zatreba usluga neke udaljene aplikacije, također obavlja i neke druge lokalne operacije (izračunava nešto, crta sučelje za korisnika...)
  - je aplikacija koju korisnik pokrene, koristi neko vrijeme dok mu je potrebna, i onda ju ugasi (dakle, "živi" samo jednu sesiju)
  - se izvršava lokalno na osobnom računalu korisnika
  - ne zahtjeva specijalizirani hardware niti operativni sustav

Klijent / Server paradigma

- serverski software tipično...
  - pasivno čeka kontakt proizvoljnog udaljenog klijenta
  - je specijalizirana aplikacija koja služi isključivo tome da pruža jednu konkretnu uslugu
  - može posluživati nekoliko udaljenih klijenata odjednom
  - se pokreće automatski kada se računalo pali, radi cijelo vrijeme dok je računalo upaljeno, obrađuje brojne klijentske zahtjeve kako dolaze ("živi" kroz mnogo sesija)
  - se izvršava na specijaliziranom računalu (zato često samo to računalo nazivamo npr. "web-server", "mail-server")
  - zahtjeva snažan hardware i sofisticirani operativni sustav

Primjeri interakcije klijenata i servera

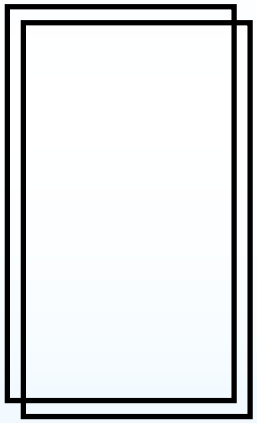
- obradit ćemo nekoliko primjera komunikacije između servera i klijenta
- redovito će način protokol komunikacije biti TCP

- daytime – “točno” vrijeme
- http – dohvaćanje web-stranice
- echo – “vrati nazad ono što ti pošaljem”
- telnet – izvršavanje naredbi na udaljenom računalu
- ftp – prijenos datoteka između računala

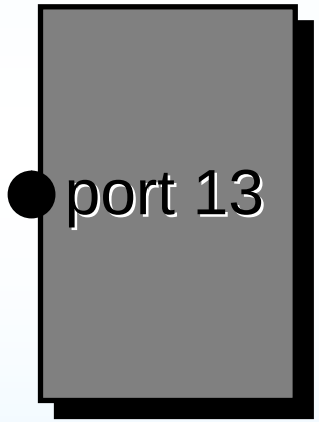
- http://beej.us/guide/bgnet/
- http://devmentor.org/articles/network/Socket%20Programming.pdf (Windows Sockets; pomalo zastarjelo)

daytime

- promatramo zasad samo klijenta, serveri cijelo vrijeme rade na većini UNIX strojeva i komuniciraju na portu 13



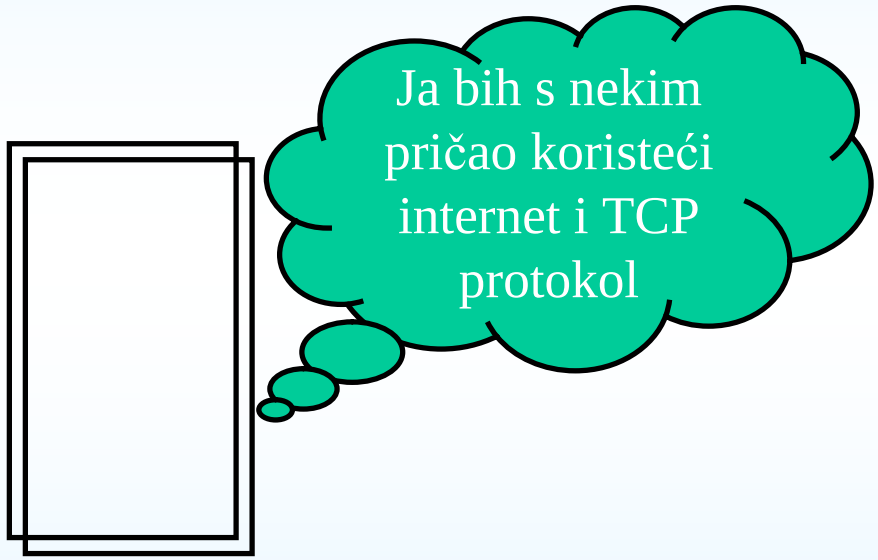
klijent



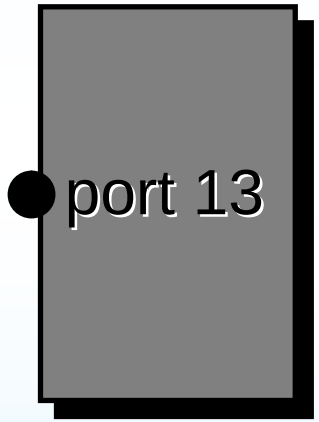
server
(crna kutija)
IP: 12.34.56.7

daytime

- klijent prvo treba stvoriti okruženje za komunikaciju
- treba definirati protokol koji želi koristiti



klijent



server
(crna kutija)
IP: 12.34.56.7

socket

Definira okruženje za komunikaciju – tzv. "utičnicu".

Za svako računalo/aplikaciju sa kojom se komunicira unutar klijenta/servera potrebno je stvoriti po jednu utičnicu.

Utičnicom će se jednoznačno u ostalim funkcijama određivati tko s kim i na koji način komunicira.

```
int socket( int domena, int tip, int protokol );
```

- domena = PF_INET (komunikacija internetom)
- tip = SOCK_STREAM za TCP protokol
- tip = SOCK_DGRAM za UDP protokol
- protokol = 0 za sve naše potrebe
- povratna vrijednost = -1 ako je došlo do greške (tada poziv funkcije perror daje detalje o greški)

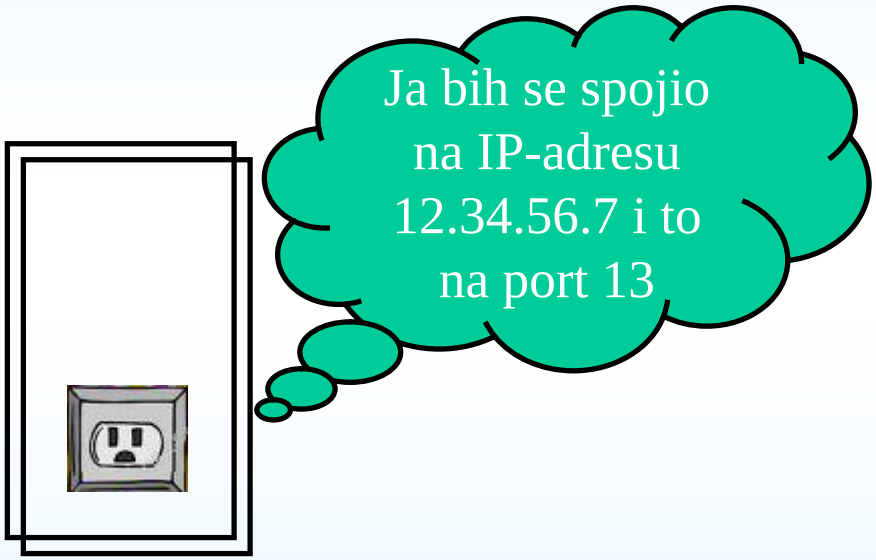
socket

Primjer:

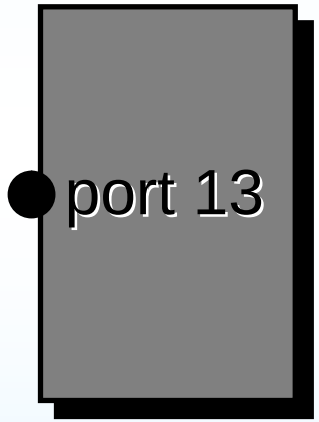
```
int mojSocket = socket( PF_INET, SOCK_STREAM, 0 );
if( mojSocket == -1 )
   perror( "socket" );
```

daytime

- jednom kad ima utičnicu, klijent treba specificirati na koji server se želi spojiti – treba znati i IP-adresu i port



klijent



server
(crna kutija)
IP: 12.34.56.7

connect

Specificira IP-adresu i port sa kojom se komunicira preko dane utičnice.

Adresa i port se specificiraju preko posebne strukture:

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

- sin_family = vrsta adrese, za nas AF_INET (komunikacija internetom)
- sin_port = port na serveru na koji se spajamo (broj između 0 i 65535)
- sin_addr = binarna IP-adresa servera
- sin_zero = polje čiji se svi elementi uvijek postave na 0

connect

```
int connect(
   int sock, struct sockaddr *servAddr, int lenAddr );
```

- sock – utičnica koju smo stvorili sa funkcijom socket
- servAddr – napunjena sockaddr_in structura (koristimo cast!), čuva podatke o serveru
- lenAddr = sizeof(servAddr)
- povratna vrijednost = -1 ako nije uspjelo (pozovi i perror za ispis detalja), 0 inače

Funkcija connect pokušava ostvariti konekciju između klijenta i udaljenog servera.

Potrebno ju je pozivati samo za spojne protokole poput TCP (nije potrebna za UDP).

connect

Primjer:

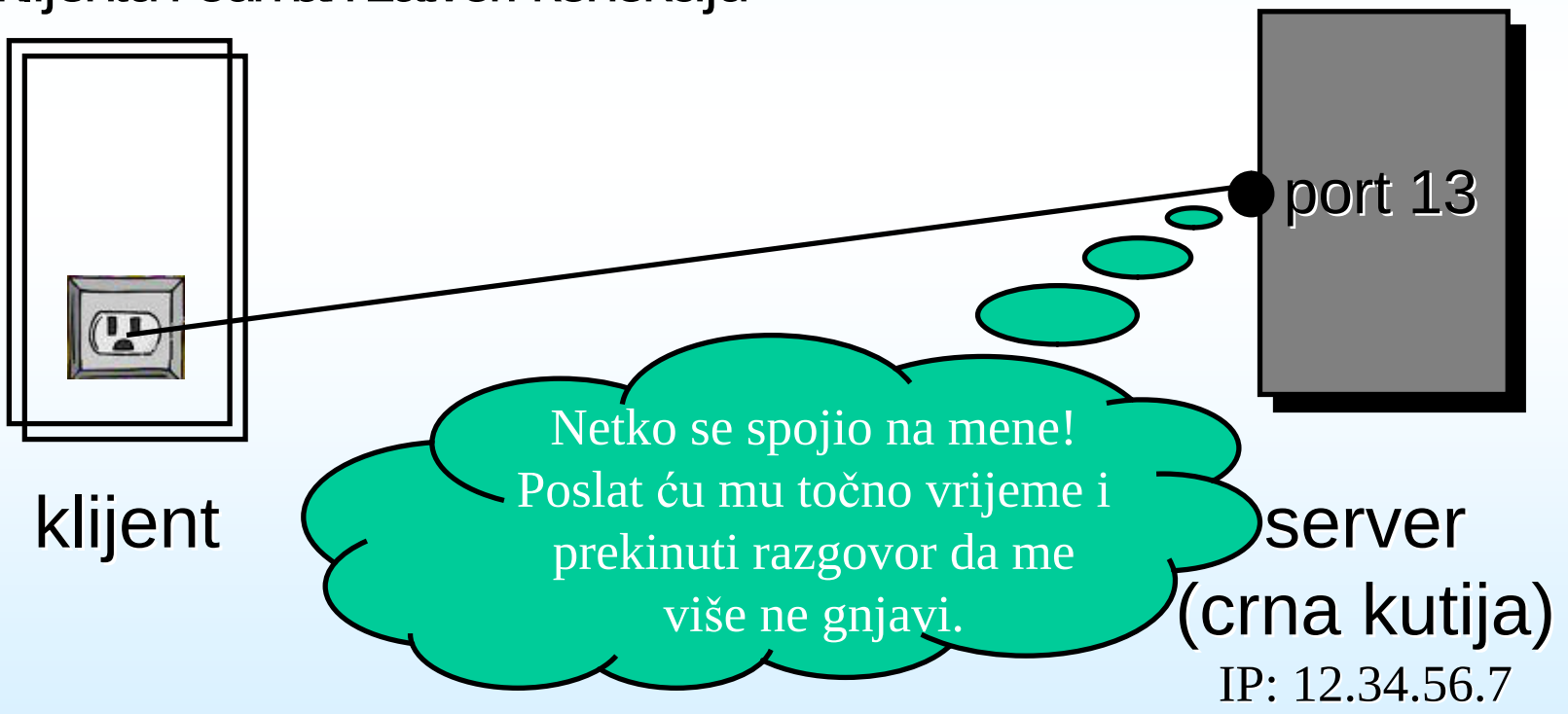
```
char dekadskiIP[] = "12.34.56.7";
struct sockaddr_in adresaServera;

adresaServera.sin_family = AF_INET;
adresaServera.sin_port = htons( 13 );

if( inet_aton( dekadskiIP, &adresaServera.sin_addr ) == 0 )
   printf( "%s nije dobra adresa!\n", dekadskiIP );

memset( adresaServera.sin_zero, '\0', 8 );

if( connect( mojSocket,
             (struct sockaddr *) &adresaServera,
             sizeof( adresaServera ) ) == -1 )
   perror( "connect" );
```

htons / htonl / ntohs / ntohl

- svaki broj (2 ili 4 byte-ni) koji se šalje preko mreže treba imati poredak byte-ova u tzv. *Network orderu*, koji se može razlikovati od poretka byte-ove korištenog na lokalnom računalu (tzv. *Host order*)
- gornje funkcije rade odgovarajuće konverzije:
  - short htons(short x) – prima x u Host orderu, vraća u Network
  - long htonl(long x) – prima x u Host orderu, vraća u Network
  - short ntohs(short x) – prima x u Network orderu, vraća u Host
  - long ntohl(long x) – prima x u Network orderu, vraća u Host
- funkcija inet_aton vraća binarnu IP-adresu u Network orderu, pa tu ne treba konverzija

daytime

- sada je ostvarena konekcija između klijenta i servera i možemo razmijenjivati podatke po daytime protokolu
- protokol je vrlo jednostavan: server pošalje trenutni datum i vrijeme klijentu i odmah zatvori konekciju



recv

Služi za primanje poruke sa udaljenog računala.

Blokira daljnje izvršavanje programa sve dok zaista nešto ne primi.

```
ssize_t recv( int sock,
   void *buffer, size_t duljinaBuffera, int opcije );
```

- sock – utičnica stvorena sa sock i povezana sa connect
- buffer – adresa (najčešće polje znakova; može biti i npr. adresa samo jednog int-a) na koju spremamo podatke koji dolaze sa servera
- duljinaBuffera = sizeof(buffer)
- opcije = 0 za sve naše potrebe

recv

recv prima maksimalno onoliko podataka kolika je duljinaBuffera.

Povratna vrijednost:

- 0, ako je druga strana prekinula konekciju
- -1, ako je došlo do pogreške (pozovi perror za detalje)
- inače, broj byte-ova koji je stigao od udaljenog računala

Problem: kako znati kada je druga strana završila sa slanjem?

Nekoliko mogućih scenarija:

1. znamo da poruka mora biti velika N byte-ova – ponavljamo recv sve dok ukupno ne primimo N byte-ova
2. druga strana prvo kaže: poslat ću M byte-ova, mi primimo tu poruku, i zatim učitavamo sve dok ne primimo M byte-ova
3. druga strana je prekinula konekciju

recvfrom

- uoči: da bismo mogli primati podatke sa recv, nužno treba postojati konekcija
- ako koristimo npr. UDP protokol (SOCK_DGRAM kod poziva funkcije socket), ne trebamo pozivati funkciju connect, ali onda umjesto recv koristimo:

```
ssize_t recvfrom( int sock,
   void *buffer, size_t duljinaBuffera, int opcije
   struct sockaddr *servAddr, socklen_t *lenAddr
);
```

  - prva 4 parametra su isti kao kod recv, zadnja 2 su isti kao kod connect

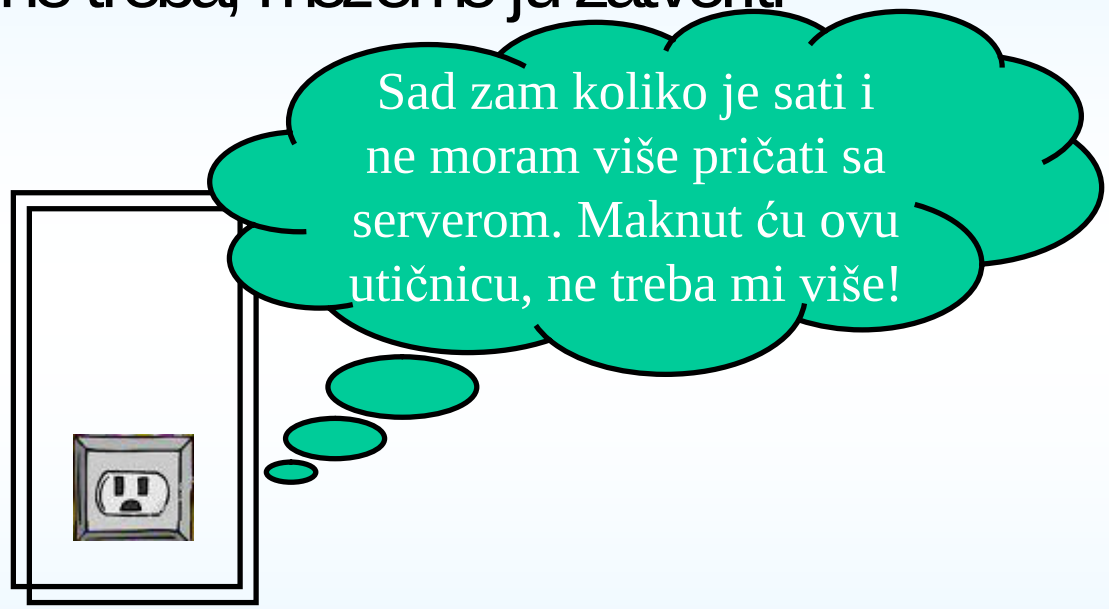
recv

Primjer za daytime (znamo da će server prekinuti konekciju kada nam pošalje točno vrijeme):

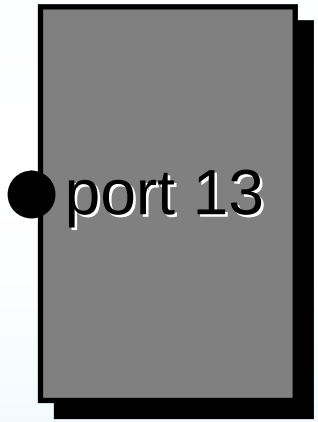
```
char buffer[100];
int primljeno = 0; // koliko smo byte-ova ukupno primili
int novoprimljeno; // koliko je primljeno u zadnjem recv

while( 1 )
{
   novoprimljeno = recv(
      mojSocket,
      buffer + primljeno,
      sizeof( buffer ) – primljeno - 1, // zbog '\0'
      0 );

   if( novoprimljeno == -1 ) { perror( "recv" ); exit(0); }
   else if( novoprimljeno == 0 ) break;
   else primljeno += novoprimljeno;
}
buffer[primljeno] = '\0'; printf( "%s", buffer );
```

daytime

- kada smo završili komunikaciju sa serverom i utičnica nam više ne treba, možemo ju zatvoriti



klijent



server
(crna kutija)
IP: 12.34.56.7

close

Prestanak upotrebe utičnice, oslobađa se memorija koju je ona trošila.

```
int close( int sock );
```

- sock – utičnica koju više ne trebamo koristiti
- povratna vrijednost = -1 ako je došlo do greške (perror za detalje), 0 inače.

Primjer:

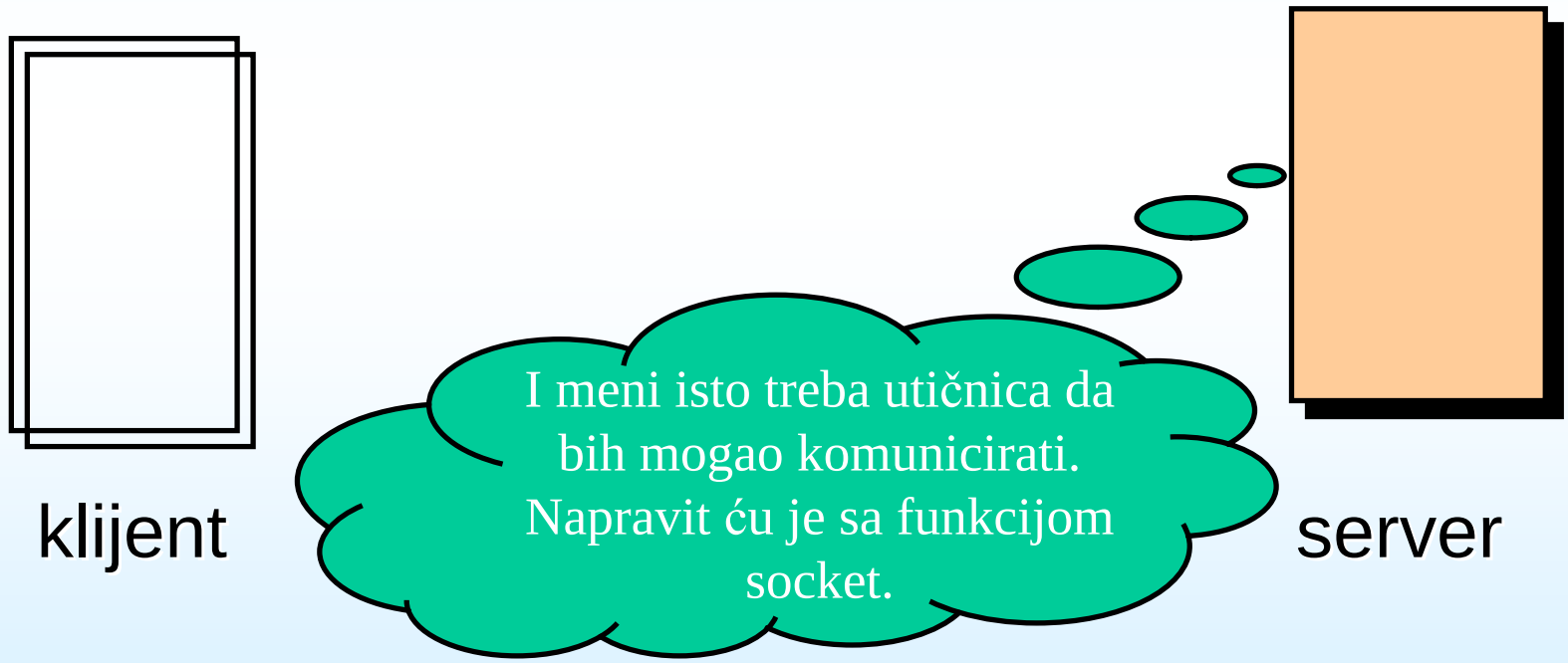
```
if( close( mojSocket ) == -1 )
   perror( "close" );
```

Zadatak 1

- Spojite sve navedene primjere u funkcionalni program.
- Pokušajte se spojiti na više različitih daytime-servera tako da modificirate IP-adresu servera.

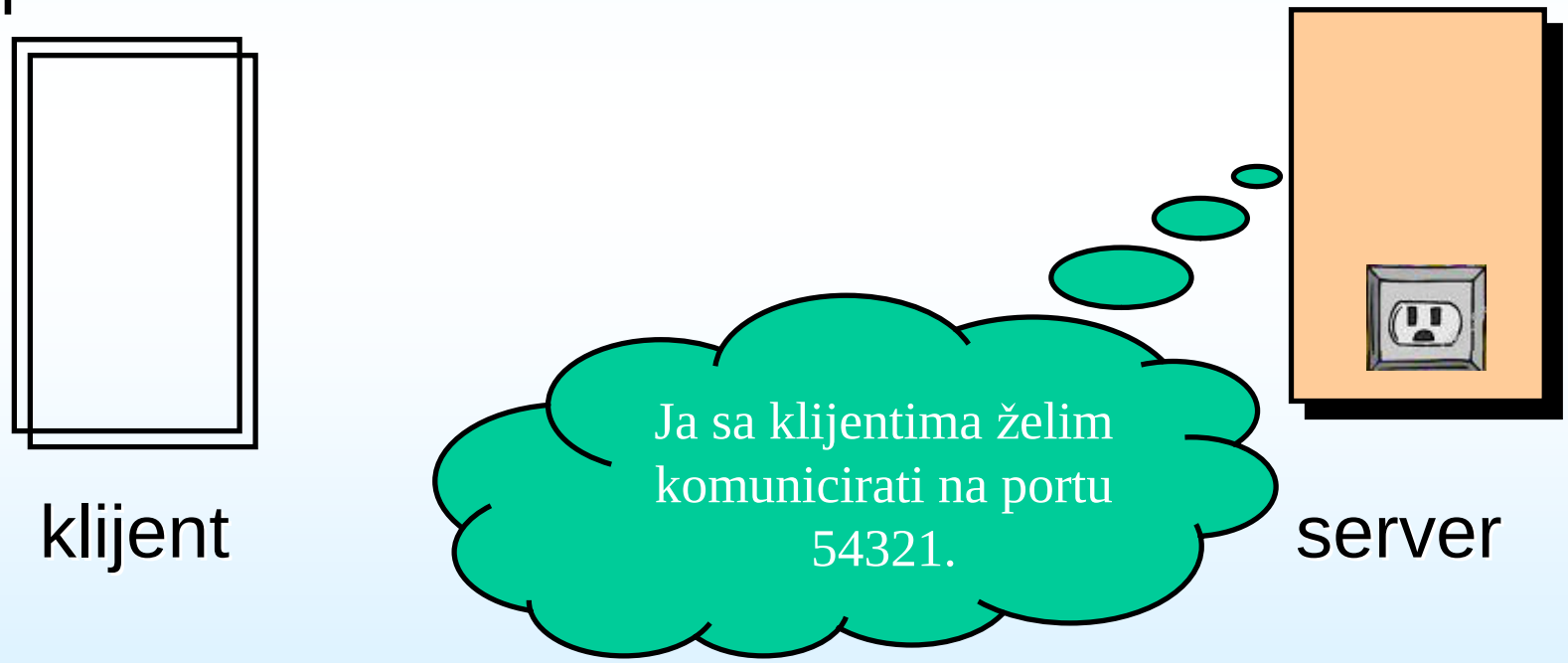
daytime – server

- Sada ćemo napisati i server. Koristit ćemo neki drugi port (npr. 54321) za našu aplikaciju (13 je rezervirani port!)



daytime – server

- Kada napravi utičnicu, server treba reći na kojem će portu vršiti komunikaciju sa klijentima. Više utičnica može komunicirati putem jednog porta. Moguća je komunikacija i kroz više portova.



bind

Specificira na kojem će portu i kojoj IP-adresi komunicirati utičnica napravljena sa socket().

```
int bind( int sock,
   struct sockaddr *servAddr, socklen_t addrlen );
```

- sock – utičnica koju smo napravili sa socket()
- servAddr – napunjena sockaddr_in structura (koristimo cast!), čuva podatke o serveru. Ako ne znamo serverovu (tj. vlastitu) IP-adresu, ili znamo da imamo samo jednu IP-adresu, možemo postaviti servAddr.s_addr = INADDR_ANY;

  i to polje će automatski biti ispunjeno na ispravan način.
- addrlen = sizeof(servAddr);
- povratna vrijednost = -1 ako je došlo do greške (perror za detalje), 0 inače

bind

Primjer:

```
struct sockaddr_in mojaAdresa;

mojaAdresa.sin_family      = AF_INET;
mojaAdresa.sin_port        = htons( 54321 );
mojaAdresa.sin_addr.s_addr = INADDR_ANY;
memset( mojaAdresa.sin_zero, '\0', 8 );

if( bind( listenerSocket,
          (struct sockaddr *) &mojaAdresa,
          sizeof( mojaAdresa ) ) == -1 )
   perror( "bind" );
```

daytime – server

- Sada server treba rezervirati utičnicu da jednostavno čeka da neki klijent pokuša uspostaviti konekciju na portu 54321...

listen

Rezerviracija utičnice koja će služiti serveru za osluškivanje nadolazećeg kontakta na danom portu.

```
int listen( int sock, int maxKonekcija );
```

- sock – utičnica koju smo napravili sa socket() i dali joj port pomoću bind()
- maxKonekcija – koliko maksimalno klijenata odjednom može čekati na uslugu servera. Ostali će pri pokušaju connect-a dobiti pogrešku "Connection refused"
- povratna vrijednost = -1 ako je došlo do greške (perror za detalje), 0 inače

listen

Primjer:

```
if( listen( listenerSocket, 10 ) == -1 )
   perror( "listen" );
```

Serveri često istovremeno komuniciraju sa više klijenata.

Za komunikaciju sa svakim od klijenata obično imaju po jednu utičnicu (vidi dalje), dok je jedna utičnica rezervirana samo za osluškivanje novih nadolazećih konekcija. (tzv. *listener-socket*)

daytime – server

- Kada klijent pokuša uspostaviti komunikaciju sa serverom, server ju može i ne mora prihvatiti.

accept

Služi sa prihvaćanje prijedloga konekcije od strane klijenta.

Doznat ćemo klijentovu adresu i dobiti novu utičnicu za komunikaciju sa njime.

```
int accept(
   int listenerSock,
   struct sockaddr *klijentAddr, unsigned int *lenAddr );
```

- listenerSock – listener utičnica kojom osluškujemo dolazeće konekcije. Možemo ju ponovno upotrijebiti.
- klijentAddr – adresa klijenta, tip joj je sockaddr_in, trebamo napraviti cast
- lenAddr – treba postaviti na adresu varijable u kojoj je sizeof(klijentAddr)
- klijentAddr popunjava funkcija accept (a ne mi)
- povratna vrijednost = -1 ako je došlo do greške (perror za detalje), inače nova utičnica za komunikaciju sa klijentom

accept

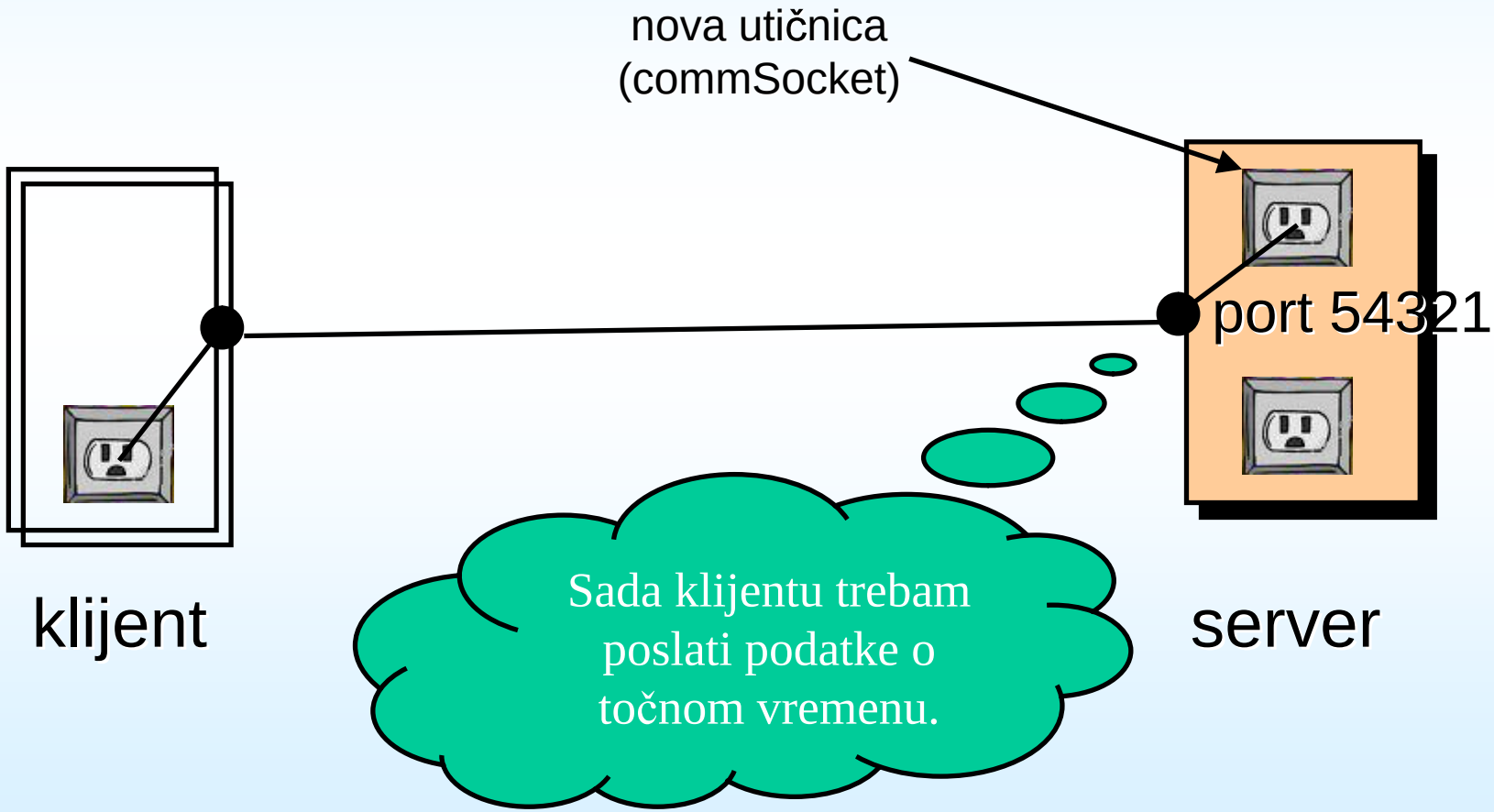
Primjer:

```
struct sockaddr_in klijentAdresa;
unsigned int lenAddr = sizeof( klijentAdresa );
int commSocket = accept( listenerSocket,
                         (struct sockaddr *) &klijentAdresa,
                         &lenAddr );
if( commSocket == -1 )
   perror( "accept" );

char *dekadskiIP = inet_ntoa( klijentAdresa.sin_addr );
printf( "Prihvatio konekciju od %s\n", dekadskiIP );
```

Funkcija accept blokira daljnje izvršavanje serverske aplikacije sve dok neki klijent ne pokuša pristupiti serveru.

daytime – server

- Nakon prihvaćanja konekcije, server može poslati podatke o točnom vremenu.



send

Slanje podataka udaljenom računalu. Funkcionira po posve istoj logici kao recv (jedino što sad unaprijed znamo koliko byte-ova treba poslati).

```
ssize_t send( int sock,
   const void *buffer, size_t duljinaBuffera, int opcije );
```

- sock – utičnica preko koje se odvija komunikacija
- buffer – adresa (najčešće polje znakova; može biti i npr. adresa samo jednog int-a) na koju spremamo podatke koji dolaze sa servera
- duljinaBuffera = sizeof(buffer)
- opcije = 0 za sve naše potrebe

sendto

- podatke pomoću send može slati i klijent
- ako klijent koristi npr. UDP protokol (SOCK_DGRAM kod poziva funkcije socket), ne treba pozivati funkciju connect, ali onda umjesto send koristi:

```
ssize_t sendto( int sock,
   void *buffer, size_t duljinaBuffera, int opcije,
   struct sockaddr *servAddr, socklen_t *lenAddr
);
```

  - prva 4 parametra su isti kao kod send, zadnja 2 su isti kao kod connect

send

Primjer:

```
#include <time.h>

time_t trenutnoVrijeme;
time( &trenutnoVrijeme );

char buffer[100];
sprintf( buffer, "%s", ctime( &trenutnoVrijeme ) );

int trebaPoslati = strlen( buffer );
int poslano = 0; // broj do sada poslanih byte-ova
int poslanoZadnje; // koliko je poslano u zadnjem send

while( poslano != trebaPoslati )
{
   poslanoZadnje = send( commSocket,
      buffer + poslano, trebaPoslati – poslano, 0 );
   if( poslanoZadnje == -1 ) perror( "send" );
   else poslano += poslanoZadnje;
}
```

daytime – server

- Nakon što je poslao točno vrijeme, server može prekinuti konekciju tako da zatvori komunikacijsku utičnicu. Nakon toga može nastaviti osluškivati eventualne nove klijente.

Zadatak 2

- Spojite sve navedene primjere u funkcionalni daytime-server koji može posluživati više klijenata (uoči: ne odjednom, već jednog za drugim!)
- Promijenite daytime-klijenta iz Zadatka 1 tako da se može spajati na port 54321. Spojite se promijenjenim klijentom na server iz prve točke.

Zadatak 3

- Napišite *echo*-klijent. Echo-klijent treba:
  - spojiti se na računalo student na port 7 (tamo se nalazi echo-server)
  - u petlji učitavati riječi sve dok se ne učita riječ kraj
  - svaku učitanu riječ poslati serveru
  - pročitati serverov odgovor i ispisati ga na ekran
- Što radi echo-server? Napišite ga (sami odaberite port komunikacije).
- Prilagodite svoj echo-klijent tako da sa komandne linije dobiva IP-adresu i port echo-servera na kojeg se treba spojiti. Testirajte sa echo-serverom na studentu i onim kojeg ste sami napisali.

Zadatak 4

- Napišite primitivni *web-browser* koji će moći prikazati jednu web-stranicu u tekstualnom modu.
- za web se koristi tzv. *http*-protokol.
- web-browser (tj. http-klijent) treba:
  - spojiti se na proizvoljni host-name kojeg dobijete iz komandne linije na port 80 (tamo se uvijek nalaze http-serveri)
  - poslati serveru poruku "GET / HTTP/1.0\n\n"
  - učitavati odgovor servera i ispisati ga na ekran
- Modificirajte program tako da ispisujete samo ono što se nalazi između <body i </body> (ili <BODY i </BODY>). Nemojte ispisivati ništa što se nalazi između znakova "<" i ">" (niti njih).