Vežba 4 –Utičnice (Sockets)

1. Teorijske osnove

A. ***Utičnice* (*Sockets*)**

Za svaki od transportnih protokola (TCP i UDP) možemo definisati utičnicu (engl. *socket*), odnosno par - IP adresa/broj porta koji jedinstveno identifikuje softverski proces (mrežnu aplikaciju) i host na kome je proces pokrenut. Ovako definisane utičnice su krajnje tačke komunikacije na transportnom sloju. Svaka serverska aplikacija po pokretanju kreira TCP ili UDP socket (u zavisnosti od tipa serverske aplikacije).

Kreiranje utičnice znači da na datoj IP adresi i na datom (dobro poznatom portu) serverski aplikacijski servis očekuje segmente koji će stići sa utičnice (par IP adresa klijenta/port > 1023) koji je otvorila klijentska aplikacija. Na primer: kada se na serveru pokrene softverski proces Web servera, on kreira - otvori utičnicu na IP adresi servera na dobro poznatom portu 80. Kada klijent na svojoj radnoj stanici pokrene Web pretraživač i u adresnom polju pretraživača unese adresu servera, Web pretraživač otvori socket na IP adresi klijenta na portu većem od 1023, na primer 39536. Krajnje tačke komunikacije na transportnom nivou su dve utičnice.

Jedna utičnica se može koristiti za više istovremenih komunikacionih veza. Drugim rečima, dve ili više veza mogu završavati na istoj utičnici. Veze se raspoznaju prema identifikatorima utičnica na oba kraja veze (socket1, socket2).

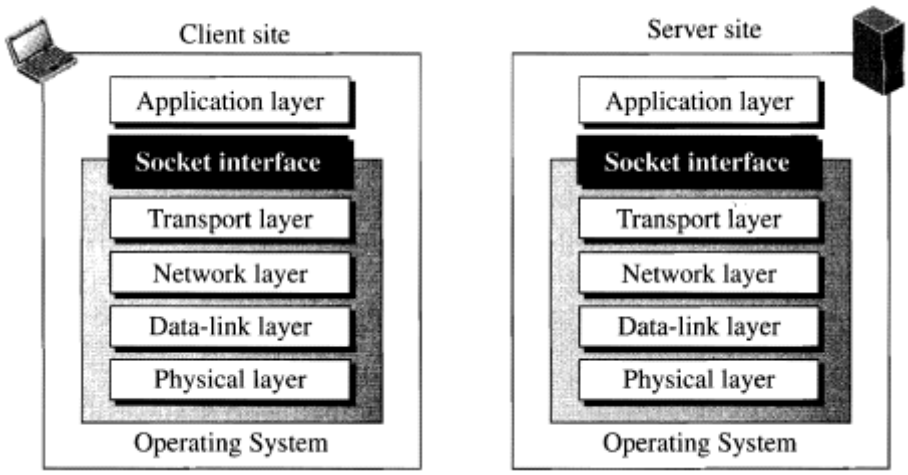
Sve TCP konekcije su full-duplex i point-to-point (tj. unicast) tipa. Protokol TCP ne podržava multicast i broadcast komunikaciju.

Usluge koje TCP i UDP nude aplikativnom sloju se razlikuju, pa su i utičnice ova dva protokola razlikuju. TCP nudi pouzdanu uslugu sa uspostavljanjem veze, pa TCP utičnice imaju svoje stanje, u smislu da li je TCP konekcija ostvarena, ili je u toku ostvarivanje odnosno prekid konekcije. Osim toga ukoliko je konekcija ostvarena poznata je i druga utičnica, na udaljenom hostu koji je drugi kraj veze. UDP servis ne poznaje uspostavu konekcije, pa UDP utičnice nemaju posebno stanje.

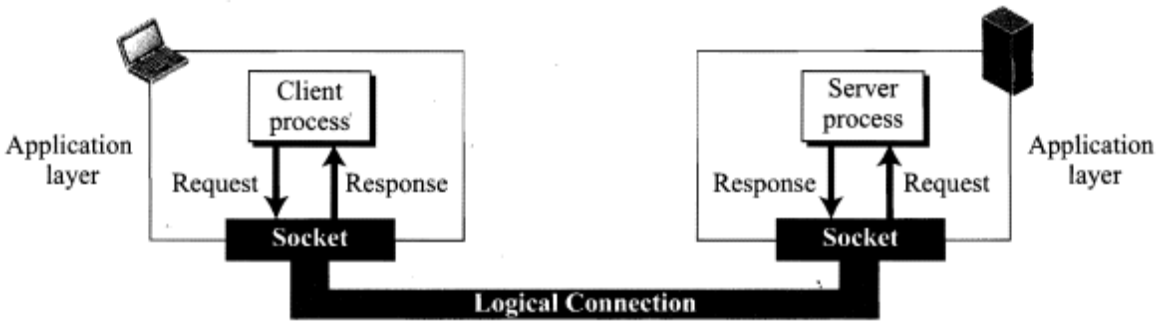
Kada jedan proces komunicira sa drugim procesom potrebno je da postoji skup instrukcija koji će nižim četiri nivoa TCP/IP skupa protokola ukazati da otvore konekciju, predaju i primaju podatke od drugog kraja, i zatvore konekciju. Skup instrukcija ove vrste naziva se aplikaciono-programski interfejs (engl. *Application Programming Interface* - API). API sa tačke gledišta programiranja se vidi kao skup instrukcija između dve celine. U konkretnom slučaju, jedna od celina je proces koji se izvršava na aplikacionom nivou, a drugi je operativni sistem koji enkapsulira prva četiri nivoa skupa protokola TCP/IP. Drugim rečima, proizvođač

računara ugrađuje prva četiri nivoa TCP/IP protokola u operativni sistem, a posebno se instalira API, čime je obezbeđeno da procesi koji se izvršavaju na aplikacionom nivou mogu da komuniciraju sa operativnim sistemom putem slanja i prijema poruka preko Interneta. Postoji nekoliko API-a, a poznatiji su: *socket* interfejs, *Transport layer interface*, TLI, i STREAM. Mi ćemo analizirati samo *socket* interfejs.

Socket interfejs (slika 1) predstavlja skup instrukcija pomoću kojih se ostvaruje komunikacija između aplikacionog nivoa i operativnog sistema.



Slika 1. Pozicija *socket* interfejsa

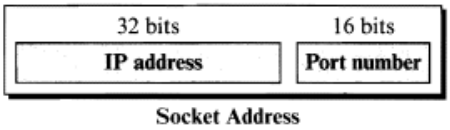


Slika 2. Korišćenje *socket*-a kod komunikacije tipa proces-ka-procesu

B. Socket adrese

Interakcija između klijenta i servera je dvosmerna. Da bi se ostvarila dvosmerna komunikacija potrebne su dve IP adrese: lokalna od predajnika i udaljena od prijemnika. Kod jednog istog računara lokalna adresa u jednom smeru (odlaznom) istovremeno predstavlja i udaljenu adresu u drugom smeru (dolaznom). S obzirom da se klijent-server komunikacija ostvaruje između dva *socket*-a, takođe za potrebe komunikacije potreban je par *socket* adresa: lokalna *socket* adresa i udaljena *socket* adresa.

Svaki računar povezan na Internet na jedinstven način je definisan svojom IP adresom, tj. pomoću 32-bitne celobrojne vrednosti u slučaju *ipv4* Internet verzije. Treba naglasiti da se po nekoliko klijent ili server procesa mogu istovremeno izvršavati na istom računaru, što znači da nam je potreban dodatni identifikator kojim se specificira koji klijent ili server učestvuje u komunikaciji. Kao što smo već prethodno ukazali, aplikacioni program se može definisati pomoću broja porta (*port number*), a to je 16-bitna celobrojna vrednost. To znači da *socket* adresa predstavlja kombinaciju IP adrese i port adrese (slika 3).



Slika 3. S*ocket* adresa (ipv4)

Pošto se *socket*-om definiše komunikacija tipa *end-to-end* može se reći da se *socket*-om identifikuje par *socket* adresa: lokalna i udaljena.

Pitanje koje se sada postavlja je sledeće: Na koji način klijent i server definišu adrese za komunikaciiju? Na svakoj strani situacija je različita.

C. Server strana

Da bi komunicirao, serveru je potrebna lokalna (server) i udaljena (klijent) *socket* adresa:

Lokalna *socket* adresa – ova adresa obezbeđuje se od strane operativnog sistema. Operativni sistem zna IP adresu računara na kome se izvršava server proces. Da bi se formirala *socket* adresa potrebno je IP adresi pridružiti port adresu. Ako je server proces standardan i definisan od strane Internet komisije za donošenje standarda, tada je i port broj takođe definisan. Tako na primer, port broj 80 je dodeljen HTTP-u i ne može se koristiti od strane drugih procesa. Ako server proces nije standardan, tada projektant server procesa može da izabere port broj (iz opsega brojeva dodeljenih od strane Internet komisije za standarde) i da ga dodeli procesu.

Udaljena *socket* adresa – ova adresa za potrebe servera predstavlja *socket* adresu klijenta kojim server ostvaruje konekciju. S obzirom da server može da opslužuje veći broj klijenta, on unapred ne zna udaljenu *socket* adresu sa kojom treba da komunicira. Server određuje *socket* adresu kada klijent pokuša da se poveže sa serverom. Klijent *socket* adresa koja je sastavni deo paketa (*request packet*) poslat od klijenta prema serveru, prihvata se od strane servera kao udaljena *socket* adresa, na koju treba poslati odgovor. Drugim rečima, i pored toga što je lokalna *socket* adresa za potrebe servera fiksna i koristi se u toku njegovog celokupnog životnog veka, udaljena *socket* adresa se menja u toku svake interakcije sa različitim klijentima.

D. Klijent strana

Klijent strani je takođe potrebna lokalna (klijent) i udaljena (server) *socket* adresa.

Lokalna *socket* adresa – ova adresa obezbeđuje se od strane operativnog sistema. Postupak dodele lokalne *socket* adrese skoro je identičan postupku dodele lokalne adrese na strani server procesa.

Udaljena *socket* adresa – za klijenta određivanje udaljene (server) *socket* adrese je nešto složenije. Kada klijent startuje on treba da zna *socket* adresu servera ako želi da se poveže sa njim. U ovom slučaju postoje sledeća dva karakteristična slučaja:

1) Ponekad, korisnik koji startuje klijent proces zna kako server port broj, tako i IP adresu računara na kome se server izvršava. U takvoj situaciji programer obe ove informacije saopštava klijent programu.

2) I pored toga što je svakoj standardnoj aplikaciji dodeljen unapred poznati broj, u najvećem broju slučajeva, ne zna se IP adresa. Ova situacija je tipična kada želimo da kontaktiramo Web stranicu, da pošaljemo elektronsku poštu, da kopiramo fajl sa udaljenog računara, i dr. U ovim slučajevima, server ima svoje ime, i identifikator kojim se na jedinstven način definiše server proces. Primeri ovih identifikatora su URL-ovi (*Uniform Resource Locator*) kakav je *www.xxx.yyy* , ili e-mail adresa kakva je recimo *xxx@yyy.com*. Klijent proces treba sada da promeni ovaj identifikator (ime) u odgovarajuću *socket* adresu. Klijent proces normalno zna broj porta jer to mora biti dobro (unapred) poznati port broj, ali IP adresu dobija (saznaje) koristeći drugu klijent-server aplikaciju koja se naziva *Domain Name System* (DNS).

2. GNU C biblioteka vs. WinSock biblioteka

Inicijalizacija/deinicijalizacija WinSock biblioteke (Windows)

Funkcija:	Opis:
`WSAStartup`	Vrši inicijalizaciju *Winsock* biblioteke
Prametri:	Opis:
`WORD wVersionRequested`	Sadrži najvišu i najnižu verziju biblioteke koju korisnik želi da koristi.
`LPWSADATA lpWSAData`	Pokazivač na strukturu u kojoj će biti smeštene informacije o inicijalizivanoj verziji biblioteke.
Povratna vrednost:	Opis:
`int`	Funkcija vreća vrednost 0 u slučaju uspešnog izvršenja ili kod greške.

Funkcija:	Opis:
`WSACleanup`	Vrši deinicijalizaciju *Winsock* biblioteke
Prametri:	Opis:
	Nema parametara.
Povratna vrednost:	Opis:
`int`	Funkcija vreća vrednost 0 u slučaju uspešnog izvršenja ili kod greške.

Inicijalizacija/deinicijalizacija GNU C biblioteke (Raspbian)

GNU C biblioteka nema ekvivalentne funkcije funkcijama *WSAStartup* i *WSACleanup* WinSock programske biblioteke. GNU C biblioteka je automatski inicijalizovana pri uključivanju u programski kod (*sys/socket.h*).

Zadavanje adrese - Raspbian (Linux)

Za zadavanje adrese koristi se *sockaddr* struktura.

```
include <netinet/in.h>

// All pointers to socket address structures are often cast to pointers
// to this type before use in various functions and system calls:

struct sockaddr {
```

```
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

Sadržaj strukture može biti različit u zavisnosti od protokola korišćenog u komunikaciji. U slučaju TCP/IP protokola, za zadavanje adrese se koristi sledeća struktura:

```
// IPv4 AF_INET sockets:

struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET, AF_INET6
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};
```

Struktura *in_addr* sadrži IP adresu.

```
struct in_addr {
    unsigned long s_addr;          // load with inet_pton()
};
```

Zadavanje adrese - Windows

Za zadavanje adrese koristi se *sockaddr* struktura.

```
typedef struct sockaddr {

#if (_WIN32_WINNT < 0x0600)
    u_short sa_family;
#else
    ADDRESS_FAMILY sa_family;           // Address family.
#endif //(_WIN32_WINNT < 0x0600)

    CHAR sa_data[14];                   // Up to 14 bytes of direct
address.
} SOCKADDR, *PSOCKADDR, FAR *LPSOCKADDR;
```

Sadržaj strukture može biti različit u zavisnosti od protokola korišćenog u komunikaciji. U slučaju TCP/IP protokola, za zadavanje adrese se koristi sledeća struktura:

```
typedef struct sockaddr_in {

#if(_WIN32_WINNT < 0x0600)
    short   sin_family;
#else //(_WIN32_WINNT < 0x0600)
    ADDRESS_FAMILY sin_family;
#endif //(_WIN32_WINNT < 0x0600)

    USHORT sin_port;
    IN_ADDR sin_addr;
    CHAR sin_zero[8];
} SOCKADDR_IN, *PSOCKADDR_IN;
```

Struktura *in_addr* sadrži IP adresu koja može biti zadata u različitim formatima.

```
typedef struct in_addr {
        union {
                struct { UCHAR s_b1,s_b2,s_b3,s_b4; } S_un_b;
                struct { USHORT s_w1,s_w2; } S_un_w;
                ULONG S_addr;
        } S_un;
#define s_addr  S_un.S_addr /* can be used for most tcp & ip code */
#define s_host  S_un.S_un_b.s_b2    // host on imp
#define s_net   S_un.S_un_b.s_b1    // network
#define s_imp   S_un.S_un_w.s_w2    // imp
#define s_impno S_un.S_un_b.s_b4    // imp #
#define s_lh    S_un.S_un_b.s_b3    // logical host
} IN_ADDR, *PIN_ADDR, FAR *LPIN_ADDR;
```

Opis pojedinih polja strukture dat je u narednoj tabeli.

Struktura:	Opis:
`struct sockaddr_in`	Struktura korišćena za zadavanje adrese pri korišćenju TCP/IP protokola.
Polje:	**Opis:**
`short sin_family`	Polje *sin_family* odgovara polju *sa_family sockaddr* strukture. Određuje familiju protokola.
`u_short sin_port`	Broj komunikacionog kanala.
`struct in_addr sin_addr`	IP adresa.
`char sin_zero[8]`	Polje dodato da bi veličina strukture odgovarala *sockaddr* strukturi. Popunjava se nulama.
Napomena:	Sve vrednosti u strukturi moraju biti u mrežnom redosledu okteta.

Funkcije za rad sa mrežnim redosledom okteta

Redosled okteta može biti različit na različitim platformama (*big-endian* ili *little-endian*). Zbog toga je uveden pojam mrežnog redosleda okteta koji ne zavisi od platforma. Da bi programski kod bio nezavisan od platforme, koriste se funkcije koje odgovarajuće vrednosti prevode u vrednosti sa mrežnim redosledom okteta.

U nastavku su navedene funkcije za rad sa mrežnim redosledom okteta:

u_long ntohl(u_long netlong);

u_long htonl(u_long hostlong);

u_short ntohs(u_short netshort);

u_short htons(u_short hostshort);

Legenda:

- h - Redosled okteta na lokalnoj platformi (*host byte order*).
- n - Mrežni redosled okteta (*network byte order*).
- s - short (16 bit).
- l - long (32 bit).

IPv4 pretvaranje adresa:

Funkcija:	Opis:
`inet_addr`	Pretvara niz znakova koji sadrži IP adresu u standardnoj decimalnoj notaciji u adresu korišćenu u strukturi `IN_ADDR`.
Prametri:	Opis:
`const char FAR * cp`	Niz znakova koji sadrži IP adresu u standardnoj decimalnoj notaciji.
Povratna vrednost:	Opis:
`unsigned long`	U slučaju uspešnog izvršenja funkcija vraća odgovarajuću adresu u binarnoj predstavi. U slučaju greške funkcija vraća INADDR_NONE vrednost.

Funkcija:	Opis:
`inet ntoa`	Pretvara IP mrežnu adresu u standardnu decimalnu notaciju.
Prametri:	Opis:
`struct in_addr in`	IN_ADDR.struktura koja sadrži IP adresu.
Povratna vrednost:	Opis:
`char FAR *`	U slučaju uspešnog izvršenja funkcija vraća pokazivač na niz znakova koji sadrži IP adresu u standardnoj decimalnoj notaciji. U slučaju greške funkcija vraća NULL pokazivač.

Inicijalizacija utičnice - *Socket* funkcija:

- **Raspbian**

Funkcija:	Opis:
`socket`	Formiranje utičnice.
Parametri:	Opis:
`int domain`	Definiše familiju protokola.
`int type`	Tip protokola koji će biti korišćen u komunikaciji. -SOCK_STREAM obezbeđuje pouzdanu, dvosmernu komunikaciju, sa uspostavom veze. -SOCK_DGRAM obezbeđuje rad sa datagramima (UDP), nepouzdanu komunikaciju bez uspostave veze. …
`int protocol`	Određuje protokol koji će biti korišćen.
Povratna vrednost:	Opis:
`int socket`	Ukoliko se funkcija uspešno izvrši povratna vrednost je identifikator napravljene utičnice (nenegativna celobrojna vrednost). U suprotnom funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`socket`	Formiranje utičnice.
Parametri:	Opis:
`int af`	Definiše familiju protokola.
`int type`	Tip protokola koji će biti korišćen u komunikaciji. -SOCK_STREAM obezbeđuje pouzdanu, dvosmernu komunikaciju, sa uspostavom veze. -SOCK_DGRAM obezbeđuje rad sa datagramima (UDP), nepouzdanu komunikaciju bez uspostave veze. …
`int protocol`	Određuje protokol koji će biti korišćen.
Povratna vrednost:	Opis:
`SOCKET`	Ukoliko se funkcija uspešno izvrši povratna vrednost je identifikator napravljene utičnice. U suprotnom funkcija vraća vrednost *INVALID_SOCKET*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Bind funkcija:

- **Raspbian**

Funkcija:	Opis:
`bind`	Datoj utičnici pridružuje određenu IP adresu i komunikacionu liniju.
Parametri:	Opis:
`int socket`	Utičnica.
`const struct sockaddr *address`	Struktura koja sadrži IP adresu i broj komunikacione linije.
`socklen_t address_len`	Veličina strukture *address* u bajtima.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`bind`	Datoj utičnici pridružuje određenu IP adresu i komunikacionu liniju.
Parametri:	Opis:
`SOCKET s`	Utičnica.
`const struct sockaddr FAR* name`	Struktura koja sadrži IP adresu i broj komunikacione linije.
`int namelen`	Veličina strukture *name* u bajtima.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Accept funkcija:

- **Raspbian**

Funkcija:	Opis:
`accept`	Prihvatanje dolazne veze. Funkcija je blokirajuća (iz funkcije se izlazi tek po prihvatanju veze).
Parametri:	Opis:
`int socket`	Utičnica.
`struct sockaddr *restrict address`	Struktura u koju se smešta informacija o adresi druge strane u komunikaciji ili NULL pokazivač.
`socklen_t *restrict address_len`	Veličina strukture *address* u bajtima ili NULL pokazivač ukoliko je *address* NULL pokazivač.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća nenegativnu celobrojnu vrednost identifikatora utičnice preko koje je veza uspostavljena. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`accept`	Prihvatanje dolazne veze. Funkcija je blokirajuća (iz funkcije se izlazi tek po prihvatanju veze).
Parametri:	Opis:
`SOCKET s`	Utičnica.
`const struct sockaddr FAR* name`	Struktura u koju se smešta informacija o adresi druge strane u komunikaciji.
`int namelen`	Veličina strukture *name* u bajtima.
Povratna vrednost:	Opis:
`SOCKET`	U slučaju uspešnog izvršenja funkcija vraća identifikator utičnice preko koje je veza uspostavljena. U slučaju greške funkcija vraća vrednost *INVALID_SOCKET*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Connect funkcija:

- **Raspbian**

Funkcija:	Opis:
`connect`	Uspostavlja vezu preko navedene utičnice.
Parametri:	Opis:
`int socket`	Utičnica.
`const struct sockaddr *address`	Struktura u koju se smešta informacija o adresi druge strane u komunikaciji.
`socklen_t address_len`	Veličina strukture *address* u bajtima.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`connect`	Uspostavlja vezu preko navedene utičnice.
Parametri:	Opis:
`SOCKET s`	Utičnica.
`const struct sockaddr FAR* name`	Struktura u koju se smešta informacija o adresi druge strane u komunikaciji.
`int namelen`	Veličina strukture *name* u bajtima.
Povratna vrednost:	Opis:
`SOCKET`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Listen funkcija:

- **Raspbian**

Funkcija:	Opis:
`listen`	Prevođenje utičnice u stanje slušanja dolazne veze.
Parametri:	Opis:
`int socket`	Utičnica.
`int backlog`	Definiše maksimalan broj veza na datoj utičnici.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`listen`	Prevođenje utičnice u stanje slušanja dolazne veze.
Parametri:	Opis:
`SOCKET s`	Utičnica.
`int backlog`	Definiše maksimalan broj veza na datoj utičnici.
Povratna vrednost:	Opis:
`SOCKET`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Dobijanje informacija o lokalnom računaru:

- **Raspbian**

Funkcija:	Opis:
`gethostname`	Funkcija vraća naziv lokalnog računara.
Parametri:	Opis:
`char *name`	Bafer u koji se smešta naziv.
`size_t namelen`	Veličina bafera *name*.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost -1.

Funkcija:	Opis:
`gethostbyname`	Funkcija vraća podatke o lokalnom računaru na osnovu njegovog naziva.
Parametri:	Opis:
`const char *name`	Naziv lokalnog računara.
Povratna vrednost:	Opis:
`struct hostent*`	U slučaju uspešnog izvršenja funkcija vraća pokazivač na HOSTENT strukturu u kojoj su smešteni podaci. U slučaju greške funkcija vraća NULL pokazivač.

- **Windows**

Funkcija:	Opis:
`gethostname`	Funkcija vraća naziv lokalnog računara.
Parametri:	Opis:
`char FAR* name`	Bafer u koji se smešta naziv.
`int namelen`	Veličina bafera *name*.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Funkcija:	Opis:
`gethostbyname`	Funkcija vraća podatke o lokalnom računaru na osnovu njegovog naziva.
Parametri:	Opis:
`const char FAR* name`	Naziv lokalnog računara.
Povratna vrednost:	Opis:
`struct hostent FAR*`	U slučaju uspešnog izvršenja funkcija vraća pokazivač na HOSTENT strukturu u kojoj su smešteni podaci. U slučaju greške funkcija vraća NULL pokazivač.

Funkcije za slanje i prijem podataka preko utičnice

Sendto funkcija:

- **Raspbian**

Funkcija:	Opis:
`sendto`	Šalje podatke na zadatu adresu.
Parametri:	Opis:
`int socket`	Utičnica preko se podaci šalju.
`const void *message`	Pokazivač na bafer koji sadrži podatke koji se šalju.
`size_t length`	Veličina podataka koji se šalju u bajtima.
`int flags`	Tip transmisije podataka.
`const struct sockaddr *dest_addr`	Opciona struktura koja sadrži odredišnu adresu.
`socklen_t dest_len`	Veličina structure *dest_addr*.
Povratna vrednost:	Opis:
`ssize_t`	U slučaju uspešnog izvršenja funkcija vraća poslan broj bajta. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`sendto`	Šalje podatke na zadatu adresu.
Parametri:	Opis:
`SOCKET s`	Utičnica preko se podaci šalju.
`const char FAR * buf`	Pokazivač na bafer koji sadrži podatke koji se šalju.
`int len`	Veličina podataka koji se šalju u bajtima.
`const struct sockaddr FAR* to`	Opciona struktura koja sadrži odredišnu adresu.
`int tolen`	Veličina structure *to*.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća poslan broj bajta. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Recvfrom funkcija:

- **Raspbian**

Funkcija:	Opis:
`recvfrom`	Prijem podataka sa utičnice.
Parametri:	Opis:
`int socket`	Utičnica korišćena za prijem podataka.
`void *restrict buffer`	Pokazivač na bafer u koji se smeštaju primljeni podaci.
`size_t length`	Veličina prijemnog bafera u bajtima.
`int flags`	Tip prijema podataka.
`struct sockaddr *restrict address`	Opciona struktura koja sadrži adresu pošiljaoca podataka.
`socklen_t *restrict address_len`	Veličina structure *address*.
Povratna vrednost:	Opis:
`ssize_t`	U slučaju uspešnog izvršenja funkcija vraća veličinu primljenih podataka u bajtima. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`recvfrom`	Prijem podataka sa utičnice.
Parametri:	Opis:
`SOCKET s`	Utičnica korišćena za prijem podataka.
`char FAR * buf`	Pokazivač na bafer u koji se smeštaju primljeni podaci.
`int len`	Veličina prijemnog bafera u bajtima.
`const struct sockaddr FAR* from`	Opciona struktura koja sadrži adresu pošiljaoca podataka.
`int fromlen`	Veličina structure *from*.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća veličinu primljenih podataka u bajtima. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Send funkcija:

- **Raspbian**

Funkcija:	Opis:
`send`	Šalje podatke na zadatu adresu.
Parametri:	Opis:
`int socket`	Utičnica preko se podaci šalju.
`const void *buffer`	Pokazivač na bafer koji sadrži podatke koji se šalju.
`size_t length`	Veličina podataka koji se šalju u bajtima.
`int flags`	Tip transmisije podataka.
Povratna vrednost:	Opis:
`ssize_t`	U slučaju uspešnog izvršenja funkcija vraća poslan broj bajta. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`send`	Šalje podatke na zadatu adresu.
Parametri:	Opis:
`SOCKET s`	Utičnica preko se podaci šalju.
`const char FAR * buf`	Pokazivač na bafer koji sadrži podatke koji se šalju.
`int len`	Veličina podataka koji se šalju u bajtima.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća poslan broj bajta. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Recv funkcija:

- **Raspbian**

Funkcija:	Opis:
`recv`	Prijem podataka sa utičnice.
Parametri:	Opis:
`int socket`	Utičnica korišćena za prijem podataka.
`void *buffer`	Pokazivač na bafer u koji se smeštaju primljeni podaci.
`size_t length`	Veličina prijemnog bafera u bajtima.
`int flags`	Tip prijema podataka.
Povratna vrednost:	Opis:
`ssize_t`	U slučaju uspešnog izvršenja funkcija vraća veličinu primljenih podataka u bajtima. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`recv`	Prijem podataka sa utičnice.
Parametri:	Opis:
`SOCKET s`	Utičnica korišćena za prijem podataka.
`char FAR * buf`	Pokazivač na bafer u koji se smeštaju primljeni podaci.
`int len`	Veličina prijemnog bafera u bajtima.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća veličinu primljenih podataka u bajtima. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Shutdown funkcija:

- **Raspbian**

Funkcija:	Opis:
`shutdown`	Sprečava prijem i slanje podataka.
Parametri:	Opis:
`int socket`	Utičnica.
`int how`	Označava koja operacija neće biti dozvoljena. SHUT_WR– Zabranjuje slanje preko navedene utičnice. SHUT_RD– Zabranjuje prijem preko navedene utičnice. SHUT_RDWR–Zabranjuje slanje i prijem preko navedene utičnice.
Povratna vrednost:	Opis:
`ssize_t`	U slučaju uspešnog izvršenja funkcija vraća veličinu primljenih podataka u bajtima. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`shutdown`	Sprečava prijem i slanje podataka
Parametri:	Opis:
`SOCKET s`	Utičnica.
`int how`	Označava koja operacija neće biti dozvoljena. SD_RECEIVE – Zabranjuje slanje preko navedene utičnice. SD_SEND – Zabranjuje prijem preko navedene utičnice. SD_BOTH–Zabranjuje slanje i prijem preko navedene utičnice.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

Zatvaranje utičnice:

- **Raspbian**

Funkcija:	Opis:
`close`	Zatvara ranije otvorenu utičnicu.
Parametri:	Opis:
`int` *`socket`*	Identifikator utičnice.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost -1 i *errno* fleg postavlja na odgovarajući kod greške.

- **Windows**

Funkcija:	Opis:
`closesocket`	Zatvara ranije otvorenu utičnicu.
Parametri:	Opis:
`SOCKET s`	Identifikator utičnice.
Povratna vrednost:	Opis:
`int`	U slučaju uspešnog izvršenja funkcija vraća vrednost 0. U slučaju greške funkcija vraća vrednost *SOCKET_ERROR*. Pozivom funkcije *WSAGetLastError* moguće je saznati kod nastale greške.

ZADACI VEŽBE

- Implementirati programe klijent-server arhitekture na primeru slanja i prijema TCP paketa.
- Prevesti i pokrenuti program klijenta i servera prvo na jednom a zatim na više raspberry pi 2 računara.
- Izmeniti klijenta tako da serveru šalje karaktere koje prihvata sa tastature.
- Izmeniti server program tako da na konzoli prikaže primljene podatke u tekstualnom obliku.
- Izmeniti server program da primljene podatke vrati klijentu. Na klijentskoj strani sačekati povratnu poruku i nakon toga prekinuti sa radom.
- Modifikovati oba programa da rade sa tekstualnim podacima proizvoljne dužine.