Vežba 3 – Transmission Control Protocol

Teorijski uvod

Transport Control Protocol (TCP) predstavlja protokol transportnog nivoa koji nudi uslugu pouzdane isporuke toka (engl. *Stream*) korisničkih podataka.

U nastavku su date osnovne osobine TCP-a:

- Komunikacija je realizovana u vidu konekcije koja se uspostavlja pomoću metode rukovanja (engl. *Handshaking*). Iz tog razloga, ovaj protokol podržava samo unicast komunikaciju.
- Pruža pouzdan prenos toka bajtova (*byte stream*) i prijem po redosledu slanja
- Obavlja kontrolu toka podataka. Kontrola toka je realizovana tako što prijemnik oglašava količinu podataka koju je spreman da primi (polje *window* u zaglavlju).
- Obezbeđuje dvosmerni prenos podataka (full-duplex) po konekciji.
- Osnovna jedinica prenosa TCP-a je *segment podataka*. Segmenti se koriste za prenos upravljačke informacije (npr. poruke za uspostavu i raskid veze), ili za prenos podataka.

← 32 bita →

SOURCE PORT	DESTINATION PORT
SEQUENCE NUMBER	
ACKNOWLEDGEMENT NUMBER	
HEADER LENGTH, RESERVED, CONTROL BITS	WINDOW
CHECKSUM	URGENT POINTER
OPTIONS (IF ANY)	
APPLICATION DATA	

Slika 1 – Izgled TCP segmenta

Na slici 1. je dat prikaz TCP segmenta, koji se sastoji iz zaglavlja i dela u kome se nalaze podaci. Zaglavlje se sastoji iz sledećih polja:

- SOURCE PORT – port izvora identifikuje aplikaciju na izvornom računaru.
- DESTINATION PORT – port odredišta identifikuje aplikaciju na odredišnom računaru.
- SEQUENCE NUMBER – redni broj prvog bajta (okteta) u datom segmentu.
- ACKNOWLEDGEMENT NUMBER – redni broj narednog bajta (okteta) koji se očekuje od druge strane. Segmenti koji pristignu van redosleda, u zavisnosti od implementacije, mogu se odbaciti ili čuvati.
- HEADER LENGTH – dužina zaglavlja segmenta (mereno u umnošcima od 32 bita). Ono je potrebno pošto dužina polja OPTIONS varira u zavisnosti od izabranih opcija.
- RESERVED – 6-to bitno polje rezervisano za buduću upotrebu.
- CONTROL BITS – određuje namenu i sadržaj segmenta, što je prikazano na slici 2.

Bit (s leva na desno)	Značenje kada je bit postavljen
URG	Polje urgentnog pokazivača je važeće
ACK	Polje potvrde je važeće
PSH	Ovaj segment zahteva operaciju „push“
RST	Resetuj vezu
SYN	Sinhronizuj „sequence“ brojeve
FIN	Pošiljalac je došao do kraja toka podataka

- *Slika 2 – Kontrolni biti u TCP zaglavlju*

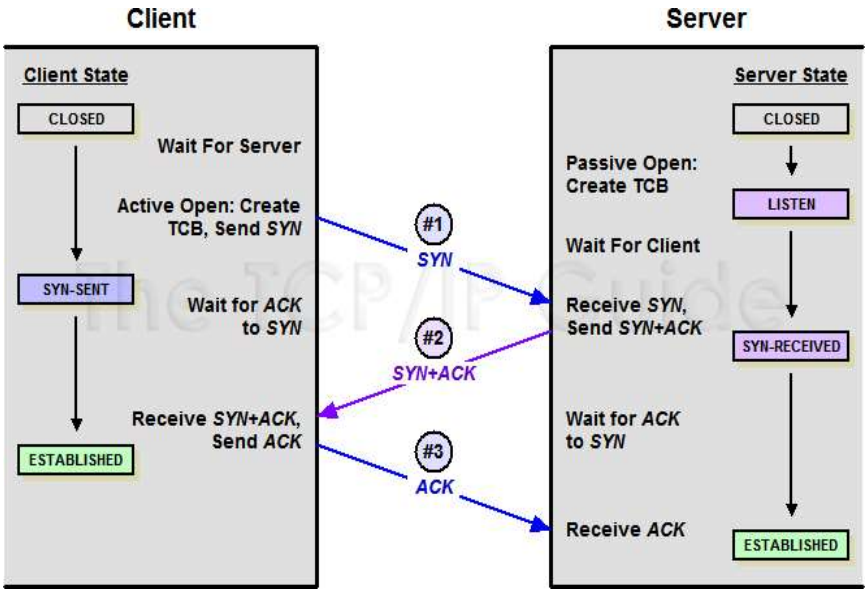
- WINDOW – veličina prijemnog bafera izražena u umnošcima od 32 bita. Označava broj bajtova koje je prijemnik u mogućnosti da primi
- CHECKSUM – kontrolna suma.
- URGENT POINTER – pozicija unutar segmenta gde se urgentni podaci završavaju.
- OPTIONS – odabrane opcije.

1. Uspostava veze kod TCP utičnica

Kada aplikacije komuniciraju preko TCP-a, uspostavlja se logička veza između aplikacije na izvorišnom računaru i aplikacije na odredišnom računaru. Nakon što se uspostavi veza, podaci između dva hosta mogu da se razmenjuju u vidu dvosmernog toka bajtova.
TCP protokol koristi metodu „trostrukog rukovanja“ („*three way handshake*“) radi uspostavljanje veze.

1. Klijent inicira zahtev za uspostavu konekcije sa serverom
   - SYN=1 – ukazuje da je postavljena inicijalna vrednost broja sekvence. Ova vrednost bira se nasumično, kako bi se otklonila mogućnost interferencije između različitih konekcija.
2. Server odobrava uspostavu konekcije i šalje zahtev za uspostavom konekcije u suprotnom smeru
   - ACK=1 – Označava da je server primio i odobrio zahtev za uspostavu veze
   - SYN=1 – Za suprotan smer komunikacije server koristi sopstveni brojač sekvenci, koji se takođe inicijalno postavlja na nasumičan broj.
3. Klijent odobrava uspostavu konekcije u suprotnom smeru
   - ACK=1

Na slici 3 prikazano je trostruko rukovanje između klijenta i servera neophodno pri uspostavljanju TCP veze .



Slika 3 – Uspostavljanje TCP veze

TCP utičnica mora biti povezana sa drugom TCP utičnicom pre nego što se ikakvi podaci mogu poslati preko njih. U tom smislu upotreba TCP utičnica ima sličnosti sa upotrebom telefona. Pre nego što možete da razgovarate sa nekim telefonskim putem, morate prvo da birate odgovarajući broj, da sačekate uspostavu veze i ako se veza uspostavi možete pričati.

U procesu uspostave veze je najveća razlika između klijentske i serverske strane. Klijentska strana inicira uspostavu veze, dok serverska strana pasivno čeka klijentske pozive za

uspostavu veze, kako bi servisirao njihove zahteve. Server mora da osluškuje zahteve za konekcijom na unapred poznatoj IP adresi (lokalnog) interfejsa i određenom portu.

1.1 Serverska strana

Serverska strana treba da pozove sledeći niz *Winsock* funkcija kako bi došlo do uspostave veze sa klijentom:

1. Kreiranje utičnice pomoću funkcije `socket()`

```
SOCKET s = socket(AF_INET,      // IPv4 address family
                  SOCK_STREAM, // stream socket
                  IPPROTO_TCP); // TCP
```

2. Kreirana utičnica se povezuje sa IP adresom lokalnog interfejsa servera i brojem porta. Ovo je potrebno zato što server mora da osluškuje konekcije na dobro poznatoj adresi. Za ovu svrhu služi funkcija `bind()`.

3. Utičnica se stavlja u stanje osluškivanja (listening mode) što se postiže pozivom funkcije `listen()`.

4. Kada klijent zatraži konekciju, server je prihvata sa funkcijom `accept()`.

Povezivanje utičnice sa adresom

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

Funkcija:	Opis:
`bind`	Povezivanje utičnice sa IP adresom lokalnog interfejsa i brojem porta
Parametri:	Opis:
`s [in]`	Utičnica koja se povezuje. U slučaju servera to je utičnica na kojoj se čekaju klijentske konekcije.
`name [in]`	Pokazivač na adresnu strukturu lokalne adrese koja će biti pridružena utičnici.
`namelen [in]`	Veličina adresne strukture koja se prosleđuje (u bajtima)
Tip povratne vrednosti	Opis
`int`	Ako se funkcija uspešno izvrši vraća vrednost 0. U suprotnom, vraća se SOCKET_ERROR, a kod konkretne greške se dobija nakon poziva funkcije WSAGetLastError.

```
// Initialize serverAddress structure used by bind
memset((char*)&serverAddress, 0, sizeof(serverAddress));

serverAddress.sin_family = AF_INET; //set server address protocol family
serverAddress.sin_addr.s_addr = INADDR_ANY;
serverAddress.sin_port = htons(SERVER_PORT);
```

```
// Setup the TCP listening socket
// Bind port number and local address to socket
iResult = bind(listenSocket, (struct sockaddr*)&serverAddress,
   sizeof(serverAddress));
```

Stavljanje utičnice u stanje osluškivanja

```
int listen (SOCKET s, int maxReq);
```

Funkcija:	Opis:
`listen`	Stavljanje utičnice u status osluškivanja zahteva za uspostavu veze
Parametri:	Opis:
`s [in]`	Utičnica koja je „bind-ovana“ tj. pridružena joj je adresa i port na kojima se osluškuju klijentski zahtevi za uspostavu veze.
`maxReq [in]`	Određuje maksimalan broj klijentskih zahteva koji mogu stići istovremeno i koji će biti stavljeni u red za opsluživanje.
Tip povratne vrednosti	Opis
`int`	Ako se funkcija uspešno izvrši vraća vrednost 0. U suprotnom, vraća se SOCKET_ERROR, a kod konkretne greške se dobija nakon poziva funkcije WSAGetLastError

Primer:

```
// Set listenSocket in listening mode
iResult = listen(listenSocket, SOMAXCONN);
if (iResult == SOCKET_ERROR)
{
	printf("listen failed with error: %d\n", WSAGetLastError());
	closesocket(listenSocket);
	WSACleanup();
	return 1;
}
```

Prihvatanje konekcije

Serversko prihvatanje klijentskog zahteva za konekcijom postiže se pozivom funkcije `accept()`.

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

Funkcija:	Opis:
`accept`	Prihvatanje dolaznog zahteva za uspostavu veze
Parametri:	**Opis:**
`s [in]`	Ovo je povezana („bind-ovana“) serverska utičnica koja je u statusu osluškivanja *(„listening state“)*.
`addr [out]`	Adresa validne `sockaddr_in` strukture. Pozivom funkcije `accept` servisira se prvi zahtev za konekciju u redu za opsluživanje konekcija. Kada se funkcija izvrši, parametar `addr` sadrži IPv4 adresnu informaciju o klijentu koji je zahtevao konekciju.
`addrlen[in/out]`	Označava veličinu adresne strukture `addr` u bajtima
Tip povratne vrednosti	**Opis**
`SOCKET`	Ako se funkcija uspešno izvrši, vraća se deskriptor `SOCKET` za novu utičnicu koja je kreirana prilikom prihvatanja klijentske konekcije. Za sve naredne operacije sa datim klijentom koristi se ova nova utičnica. Prvobitna utičnica u stanju osluškivanja i dalje ostaje u tom stanju i njen zadatak je prihvatanje novih zahteva za konekcijom. U suprotnom, vraća se INVALID_SOCKET, a kod konkretne greške se dobija nakon poziva funkcije WSAGetLastError

Ukoliko su polja addr i addrlen postavljena na NULL, neće biti vraćene informacije o adresi povezanog soketa.

```
acceptedSocket = accept(listenSocket, NULL, NULL);
if (acceptedSocket == INVALID_SOCKET)
{
	printf("accept failed with error: %d\n", WSAGetLastError());
	closesocket(listenSocket);
	WSACleanup();
	return 1;
}
```

1.2 Klijentska strana

Klijentska strana je zahteva poziv 3 *Winsock* funkcije kako bi se uspostavila TCP konekcija sa serverom.

1. Kreiranje soketa na klijentskoj strani korišćenjem funkcije `socket()`.

```
SOCKET s = socket(AF_INET,
                  SOCK_STREAM,
                  IPPROTO_TCP);
```

2. Popuniti `SOCKADDR_IN` strukturu sa IP adresom servera i brojem porta na kojem serverska aplikacija osluškuje.

```
// create and initialize address structure
sockaddr_in serverAddress;
serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = inet_addr(SERVER_IP_ADDRESS);
serverAddress.sin_port = htons(SERVER_PORT);
```

3. Inicirati uspostavu veze pozivom funkcije `connect()`. Kada se funkcija `connect()` vrati, veza je uspostavljena i preko nje mogu da se razmenjuju podaci (šalju i/ili primaju).

Iniciranje uspostave veze

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

Funkcija:	Opis:
`connect`	Funkcija uspostavlja konekciju na specificiranoj utičnici
Parametri:	**Opis:**
`s [in]`	Deskriptor koji identifikuje klijentsku utičnicu koja traži uspostavu veze
`name [in]`	Pokazivač na `sockaddr` strukturu prema kojoj bi trebala da se ostvari veza (podaci o adresnoj familiji, IP adresi i portu na serverskoj strani)
`namelen[in]`	Dužina `sockaddr` strukture u bajtima
Tip povratne vrednosti	**Opis**
`int`	Ako se funkcija uspešno izvrši vraća vrednost 0. U suprotnom, vraća se SOCKET_ERROR, a kod konkretne greške se dobija nakon poziva funkcije WSAGetLastError.

```
// connect to server specified in serverAddress and socket connectSocket
iResult = connect( connectSocket, (SOCKADDR*)&serverAddress,
       sizeof(serverAddress));
if(iResult == SOCKET_ERROR)
{
       printf("Unable to connect to server.\n");
       closesocket(connectSocket);
       WSACleanup();
       return 1;
}
```

2. Funkcije za slanje i prijem podataka preko utičnice

Kada dve TCP utičnice ostvare konekciju, mogu da počnu sa slanjem i prijemom podataka. Kao što je ranije rečeno, klijentska strana pozivom `connect()` inicira uspostavu konekcije i nakon njenog izvršenja dobija konektovanu utičnicu, a serverska strana posle izvršenja funkcije `accept()` dobija svoju konektovanu utičnicu. Kroz konektovane TCP utičnice, komunikacija se odvija pozivom funkcija `send()` i `recv()`.

2.1 Slanje podataka

Funkcija `send` služi za slanje podataka preko utičnice koja je uspostavila vezu sa drugom utičnicom. Ova funkcija kao parametar ima deskriptor konektovane utičnice preko koje će se podaci slati. `Buf` je pokazivač na niz bajtova koji će se poslati, a njihov broj je određen parametrom `len`. Podrazumevano ponašanje funkcije `send()` je da se ona blokira dok se svi podaci ne pošalju.
Parametar `flag` omogućava izmene podrazumevanog načina izvršenja funkcije. Postavljanjem vrednosti `flag` na 0 postiže se podrazumevano (default) izvršenje.

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

Funkcija:	Opis:
`send`	Funkcija za slanje podataka preko utičnice koja je uspostavila vezu sa drugom utičnicom
Parametri:	Opis:
`s [in]`	Utičnica preko koje se šalju podaci
`buf [in]`	Pokazivač na bafer koji sadrži podatke koji se šalju.
`len [in]`	Veličina podataka (u bajtima) koji se šalju.
`flags [in]`	Skup flegova koji utiču na način izvršenja funkcije. Ako se postavi 0 onda imamo podrazumevano (*default*) izvršenje.
Povratna vrednost	Opis
`int`	U slučaju uspešnog izvršenja, funkcija vraća ukupan broj poslatih bajtova (koji može biti manji od zadate vrednosti `len`). U slučaju greške vraća vrednost SOCKET_ERROR. Pozivom funkcije *WSAGetLastError* saznaje se kôd nastale greške.

Primer:

```
// Message to send
char *messageToSend = "this is a test";

// Send an prepared message
iResult = send(connectSocket, messageToSend, (int)strlen(messageToSend), 0 );
```

2.2 Prijem podataka

Funkcija `recv` služi za prijem podataka preko utičnice koja je uspostavila vezu sa drugom utičnicom. Ova funkcija kao parametar ima deskriptor konektovane utičnice preko koje će se podaci primiti. `Buf` je pokazivač na bafer gde će se primljeni podaci smestiti, a parametar `len` određuje maksimalni broj bajta koji se odjednom može primiti. Podrazumevano ponašanje funkcije `recv()` je da se ona blokira dok ne primi bar nekoliko bajtova podataka (na većini sistema ta minimalna količina podataka je 1 bajt).

Važna napomena: Broj bajta primljen jednim pozivom funkcije `recv()` nije nužno jednak broju bajtova koji su poslati jednim pozivom funkcije `send()`.

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

Funkcija:	Opis:
`recv`	Funkcija za prijem podataka preko utičnice koja je uspostavila vezu sa drugom utičnicom
Parametri:	Opis:
`s [in]`	Utičnica preko koje se primaju podaci
`buf [out]`	Pokazivač na prijemni bafer u koji se smeštaju dolazni podaci
`len [in]`	Veličina prijemnog bafera (u bajtima).
`flags [in]`	Skup flegova koji određuje način na koji se funkcija izvršava Ako se postavi na 0 onda imamo podrazumevano (*default*) ponašanje
Povratna vrednost	Opis
`int`	U slučaju uspešnog izvršenja, funkcija vraća ukupan broj primljenih bajtova i pokazivač `buf` na bafer u koji su smešteni primljeni podaci. U slučaju greške vraća vrednost SOCKET_ERROR. Pozivom funkcije *WSAGetLastError* saznaje se kôd nastale greške.

Primer:

```
iResult = recv(acceptedSocket, recvbuf, INCOMING_BUFFER_SIZE, 0);
printf("Client sent: %s\n", dataBuffer);
```

3. Sprečavanje slanja i/ili prijema podataka preko utičnice

Funkcija koja sprečava slanje i/ili prijem preko utičnice (bilo kog tipa, TCP ili UDP) je funkcija `shutdown`. Funkcija `shutdown` ne vrši zatvaranje utičnice, što znači da bilo koji resursi zauzeti od strane utičnice neće biti oslobođeni nakon poziva `shutdown`. Da bi se obezbedilo da su svi podaci na konektovanoj utičnici poslati i primljeni, aplikacija bi trebala prvo da pozove `shutdown`, a zatim da zatvori utičnicu sa `closesocket`.

```
int shutdown(SOCKET s, int option);
```

Funkcija:	Opis:
`shutdown`	Funkcija koja onemogućava slanje ili prijem podatka preko utičnice.
Parametri:	Opis:
`s [in]`	Deskriptor utičnice
`option [in]`	Ovaj parametar opisuje koja operacija preko utičnice više neće biti dostupna. Moguće vrednosti su: - SD_RECEIVE : onemogućen prijem podatka - SD_SEND : onemogućeno slanje podataka - SD_BOTH : onemogućen i prijem i slanje podataka
Povratna vrednost	Opis
`int`	U slučaju uspešnog izvršenja, funkcija vraća 0. U slučaju greške vraća vrednost SOCKET_ERROR. Pozivom funkcije *WSAGetLastError* saznaje se kôd nastale greške.

Primer:

```
// Shutdown the connection since no more data will be sent
iResult = shutdown(ClientSocket, SD_SEND);
if (iResult == SOCKET_ERROR)
{
    printf("shutdown failed with error: %d\n", WSAGetLastError());
    closesocket(ClientSocket);
    WSACleanup();
    return 1;
}
```

4. Dobijanje informacija o utičnici

Za svaku utičnicu koja je povezana sa drugom utičnicom (drugim računarom), sistem prepoznaje dve adrese: lokalnu i udaljenu. Ove adrese se mogu saznati pozivom funkcija `getsockname` (za lokalnu adresu) i `getpeername` (za udaljenu adresu). Obe funkcije vraćaju `sockaddr` strukturu, a ona je u slučaju `getpeername` funkcije popunjena sa informacijom o IP adresi i portu udaljene utičnice sa kojom je uspostavljena veza.

```
int getpeername(SOCKET s, struct sockaddr *remoteAddress, int *addrLen);
```

Funkcija:	Opis:
`getpeername`	Vraća adresu utičnice povezane sa utičnicom `s`
Parametri:	Opis:
`s [in]`	Deskriptor koji identifikuje povezanu utičnicu
`remoteAddress [out]`	Pokazivač na SOCKADDR strukturu koja prima adresu utičnice sa kojom je uspostavljena veza (peer socket)
`addrLen [in, out]`	Pokazivač na veličinu bafera `remoteAddress` u bajtima
Povratna vrednost	Opis
`int`	Ako nema greške vraća 0. U slučaju greške vraća SOCKET_ERROR (pozivom funkcije *WSAGetLastError* saznaje se kôd konkretne greške.)

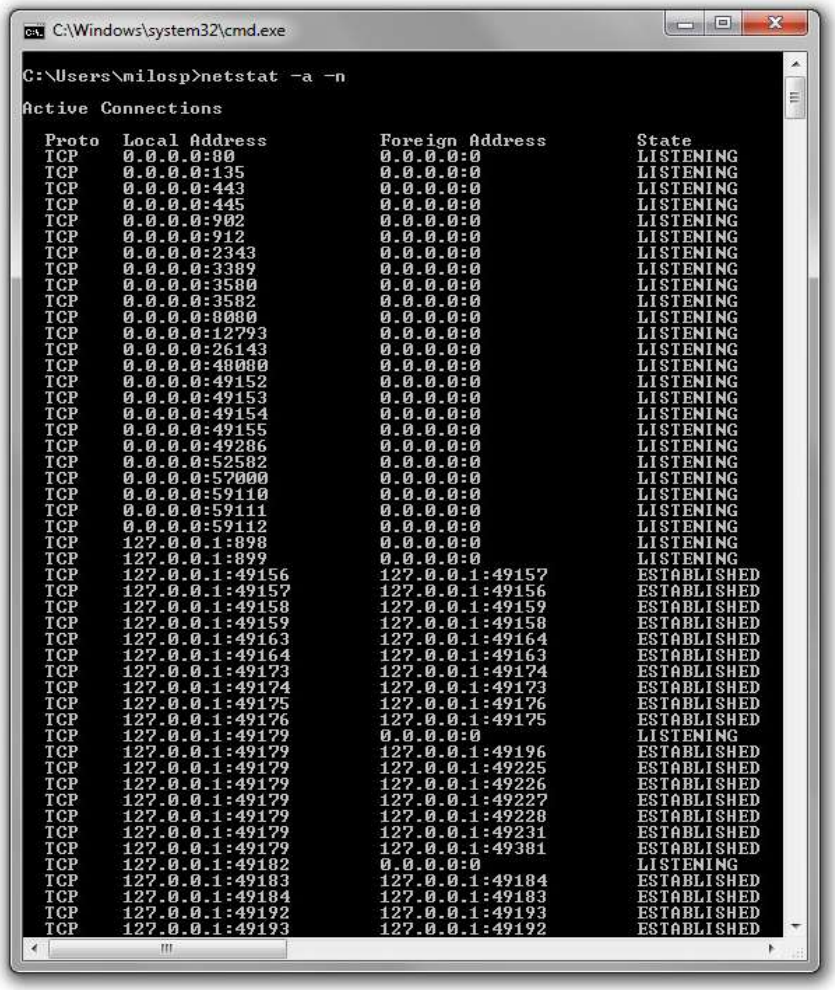
Primer:

```
SOCKET connectedSocket;
struct sockaddr_in peerAddress;
int peerlenAddr = sizeof(peerAddress); // We must put the length in a variable.

// Ask getpeername to fill in peer's socket address.
if (getpeername(connectedSocket, &peerAddress, &peerlenAddr) == -1)
 {
      printf("getpeername() failed with error: %d\n", WSAGetLastError());
      return 1;
 }

// Print the IP address and port
printf("Peer's IP address is: %s\n", inet_ntoa(peerAddress.sin_addr));
printf("Peer's port is: %d\n",  ntohs(peerAddress.sin_port));
```

NAPOMENA: Specifikacija funkcije `getsockname` može se pronaći u 3 vežbi.

5. Pregled utičnica na lokalnom računaru

Netstat dijagnostički program prikazuje statistike protokola i trenutne TCP/IP konekcije sa i prema drugim mrežnim uređajima. Na komandnoj liniji potrebno je upisati **netstat –a** da bi se prikazale sve konekcije i portovi na kojima se te konekcije uspostavljaju. Opcija **–n** upućuje Netstat da ne prevodi adrese i brojeve portova u imena, čime se ubrzava izvršavanje. Moguće je kombinovati različite opcije unutar jedne komande. Na slici 4 prikazan je primer komande **netstat – a –n**.



Slika 4 - Primer netstat naredbe

Broj iza dvotačke je broj porta koji konekcija koristi, dok kolona *State* predstavlja trenutno stanje konekcije po pojedinoj adresi i portu (tj. utičnici).

Postoji veći broj (tačnije 10) mogućih stanja konekcije od kojih ćemo izdvojiti samo neke:

- LISTENING – Server je spreman za prihvatanje konekcije
- ESTABLISHED – Uspostavljena konekcija sa udaljenim hostom
- CLOSED – Zatvorena konekcija prema udaljenom hostu
- CLOSE_WAIT – Server je u procesu raskida konekcije prema klijentu
- TIME_WAIT – Klijent je u procesu raskida konekcije prema serveru.

ZADATAK 1

U prilogu materijala date su implementacije klijenta i servera koji koriste TCP transportni protocol za komunikaciju unutar jednog računara.

1. Skicirati implementaciju klijenta.
2. Skicirati implementaciju servera.
3. Pratiti stanja soketa na serveru i klijentu koristeći kontrolisano izvršavanje programa i netstat dijagnostički program .
4. Omogućiti komunikaciju između dva različita računara.
5. Omogućiti serveru da nakon prijema poruke server pošalje poruku klijentu.

Napomena: IP adresu drugog računara moguće je dobaviti korišćenjem ipconfig naredbe u commandpromt-u.

ZADATAK 2

Koristeći primer jednostavne implementacije TCP klijenta i TCP servera koji je dat u prilogu prethodne vežbe napisati program koji realizuje igru „Na slovo na slovo“:

1. U igri učestvuju dva klijenta. Da bi igra započela potrebno je da prethodno obaklijenta uspostave vezu sa serverom. Ispisati na ekranu adresne informacije o oba klijenta nakon što se uspostavi veza.
2. Nakon uspešnog konektovanja na server, na serveru se zadaje slovo na kojeće se igrati. Server klijentima šalje obaveštenje o početku igre i slovo na koje se igra, npr. “Počinje igra na slovo G. Šaljite reči.”
    - Za formatiran upis stringa u memorijski bafer preporučujemo korišćenje `sprintf` funkcije, čiji primer poziva je dat u nastavku:

```
sprintf(dataBuffer, "Some text: %d.", number);
```

3. Svaki od klijenata ima mogućnost da bira (unosi) svoju reč na zadato slovoi svoj izbor šalje serveru.
4. Server proverava da li prisitigle reči počinju na zadato slovo. Rezultatprovere ispisati na ekranu servera.
5. Igra se nastavlja sve dok jedan klijent ne pogreši ili ne odustane. Klijentodustaje od igre slanjem reči “Kraj”.
6. Ako oba klijenta istovremeno pogreše igra se nastavlja.
7. Na kraju igre server šalje klijentima rezultat igre.
    - Rezultat prikazati u formatu poruke koja sadrži: broj ispravnih reči prvog klijenta, broj ispravnih reči drugog klijenta i pobednika igre.
8. Server prihvata reč ako je poslata i na malo i na veliko zadato slovo.
9. Ako oba klijenta istovremeno odustanu, pobednik je onaj čiji je ukupan zbirdužina poslatih ispravnih reči veći.
10. Ukoliko se u toku igre desi greška pri prenosu ili prijemu poruke, igra seprekida.