Vežba 2 – Protokol korisničkih datagrama (UDP)

Teorijske osnove

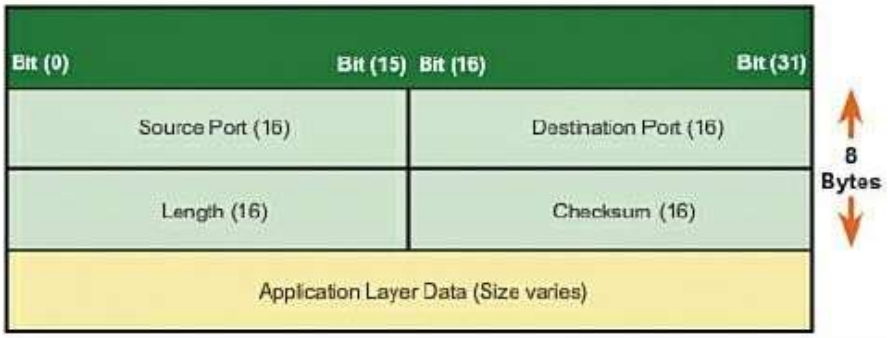
UDP (engl. *User Datagram Protocol*) protokol je protokol transportnog nivoa. Njegov zadatak je da uzme poruku sa aplikativnog nivoa, na nju doda portove primaoca i pošiljaoca i tako dobijeni datagram prosledi mrežnom sloju. Za jedinicu komunikacije koristi se datagram.

Najvažnije osobine UDP protokola:

- Najmanja moguća veličina zaglavlja. Poređenja radi, zaglavlje UDP sadrži 8 bajta, dok zaglavlje TCP segmenta sadrži 20 bajta.
- Za slanje segmenta se ne koristi uspostava konekcije i praćenje stanja komunikacije. Nepouzdana isporuka podataka
- Kontrola isporuke paketa i redosleda pristiglih poruka ostavljena je aplikativnom sloju.
- Tok podataka se ne kontroliše niti postoji zaštita od zagušenja prijemnika velikim brojem poruka.

Sve prethodne osobine omogućile su izuzetne performanse pri slanju UDP segmenata.

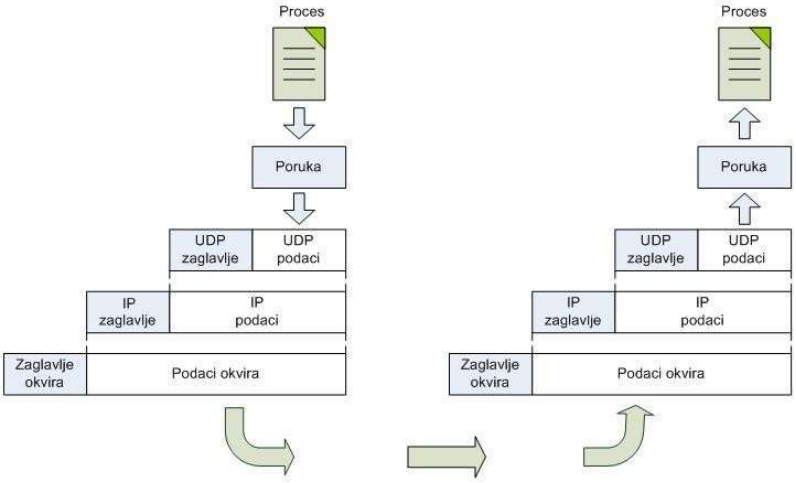
U nastavku je data struktura UDP datagrama i opis polja iz njenog zaglavlja (Slika 1).



Slika 1 - Struktura UDP datagrama

- **Source Port** - port pošiljaoca poruke. Ukoliko se ne korsiti stavlja se vrednost 0, u suprotnom predstavlja broj porta na koji treba slati odgovore.
- **Destination Port** - port primaoca poruke.
- **Length** - ukupna veličina UDP datagrama (uključujući zaglavlje i korisničke podatke) izražena u bajtima. Zbog postojanja različitih ograničenja veličine datagrama na različitim platformama, postoji preporuka da ona ne prelazi veličinu od 512 bajta.
- **Checksum** - kontrolna suma koja omogućava detekciju greške u poruci. Računa se na osnovu pseudo zaglavlja koje pored polja uz UDP zaglavlja obuhvata i deo polja iz IP zaglavlja.

UDP poruka se smešta u IP datagram, a on u fizički okvir (Slika 2).



Slika 2. Enkapsulacija podataka

Port predstavlja 16-bitni broj koji omogućava identifikaciju određenog servisa (aplikacije) na računaru. Uvođenjem koncepta porta, otvara se mogućnost da klijent koristi više servisa jednog te istog servera.

Za dodelu portova zadužen je operativni sistem. Dobro poznati servisi imaju rezervisanu vrednost porta (0 - 1023), dok se preostali opseg može slobodno koristiti.

Neki rezervisani portovi koje koristi UDP:

- Echo – 7
- Daytime – 13
- DNS – 53
- Bootpc – 68
- RPC – 111
- NTP – 123

UDP port se realizuje kao red čekanja (engl. *queue*). OS stvara ovaj red na zahtev aplikacije. Aplikacija može da zada ili promeni veličinu reda čekanja. Nakon prijema UDP datagrama, UDP provera broj prolaza sa brojevima koji su trenutno u upotrebi. Ako se zadati port ne koristi, šalje se ICMP poruka "port unreachable" i UDP datagram se odbacuje. U suprotnom, UDP ulančava UDP datagram u red čekanja, osim ako je on pun, kad dolazi do greške i odbacivanja primljenog UDP datagrama.

Razlika između TCP i UDP često se objašnjava analogijom između telefonskog i poštanskog sistema.

TCP - Telefonski sistem

Kada pozovete broj, telefon se javi i konekcija između dve strane je uspostavljena. Dok pričate, znate da druga strana čuje vaše reči onim redom kojim ih pričate. Ako je telefon zauzet ili se niko ne javlja, odmah to znate.

UDP- Poštanski sistem

Šaljete poštu na neku adresu. Većina pisama stigne, ali neka mogu biti izgubljena. Pisma verovatno stižu redom kojim su poslata, ali ne postoji garancija. Što smo dalje od primaoca, verovatnije je da će pošta biti izgubljena ili stići nekim drugim redosledom. Ako je to problem, možemo pisati redne brojeve a zatim tražiti od primalaca da ih urede i pošalju pismo kojim nas obaveštavaju koja pisma su stigla tako da možemo ponovo da pošaljemo ona koja nisu. Međutim, mi i primalac to moramo da dogovorimo unapred. Pošta to neće uraditi za nas

Kada koristiti UDP transportni protokol?

Postoji nekoliko slučajeva kada je bolje koristiti UDP umesto TCP protokola:

- Kada je blok podataka koji treba poslati mali, veličine jednog paketa – jednostavnije je, brže i efikasnije prenositi samo podatke (uz zaglavlje UDP-a), pa u slučaju pogrešno primljene poruke ponoviti slanje, nego uspostavljati vezu i proveravati pouzdanost prenosa.
- UDP koriste poruke tipa upita koje jedan računar šalje drugom, pri čemu se ako odgovor ne stigne u nekom određenom vremenu, zahtev ponovi ili se od njega odustane.
- Neke aplikacije imaju sopstvene tehnike za pouzdani prenos podataka i ne zahtevaju korišćenje TCP protokola, tako da je tada bolje koristiti UDP.

Funkcije za slanje i prijem podataka preko UDP utičnice

Slanje podataka

```
int sendto (SOCKET s, const char *buf, int len, int flags, const struct sockaddr
      *to, int tolen);
```

Funkcija:	Opis:
`sendto`	Šalje podatke na zadatu adresu
Parametri:	Opis:
`s [in]`	Utičnica preko koje se podaci šalju
`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 određuje način ponašanja funkcije pri pozivu
`to [in]`	Opcioni pokazivač na `sockaddr` strukturu koja sadrži adresu odredišnog soketa
`tolen [in]`	Veličina strukture `to`
Povratna vrednost	Opis
`int`	U slučaju uspešnog izvršenja, funkcija vraća ukupan broj poslatih bajtova (koji može biti manji od vrednosti `len`). U slučaju greške vraća vrednost SOCKET_ERROR. Pozivom funkcije WSAGetLastError saznaje se kôd nastale greške.

Primer:

```
// Create a socket for sending data
SOCKET sendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

// Set up the recvAddr structure with the IP address of
// the receiver ("192.168.1.1") and the specified port number (27015).
unsigned short port = 27015;
sockaddr_in recvAddr;
recvAddr.sin_family = AF_INET;
recvAddr.sin_port = htons(port);
recvAddr.sin_addr.s_addr = inet_addr("192.168.1.1");

// Send a datagram to the receiver
int bufLen = 1024;
 char *secret_message = "The Cheese is in The Toaster";

iResult = sendto(sendSocket, secret_message, strlen(secret_message), 0,
    (SOCKADDR *) &recvAddr, sizeof (recvAddr));

if (iResult == SOCKET_ERROR)
{
```

```
        printf("sendto failed with error: %d\n", WSAGetLastError());
        closesocket(sendSocket);
        WSACleanup();
        return 1;
    }
```

Prijem podataka

```
int recvfrom (SOCKET s, char *buf, int len, int flags, struct sockaddr *from,
      int *fromlen);
```

Funkcija:	Opis:
`recvfrom`	Funkcija za prijem datagrama
Parametri:	Opis:
`s [in]`	Utičnica koja prima podatke
`buf [out]`	Pokazivač na bafer u koji se smeštaju primljeni podaci
`len [in]`	Veličina prijemnog bafera (u bajtima)
`flags [in]`	Skup opcija koje menjaju ponašanje funkcije pri pozivu
`from [out]`	Opcioni pokazivač na `sockaddr` strukturu koja će po završetku funkcije sadržati adresu pošiljaoca podataka
`fromlen [in/out]`	Veličina strukture `from`
Povratna vrednost	Opis
`int`	U slučaju uspešnog izvršenja, funkcija vraća veličinu primljenih podataka u bajtima. U slučaju greške vraća vrednost SOCKET_ERROR. Pozivom funkcije *WSAGetLastError* saznaje se kôd nastale greške.

Primer

```
// Create a receiver socket to receive datagrams
SOCKET recvSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

// Bind the socket to any address and the specified port.
sockaddr_in recvAddr;
unsigned short port = 27015;
recvAddr.sin_family = AF_INET;
recvAddr.sin_port = htons(port);
recvAddr.sin_addr.s_addr = htonl(INADDR_ANY);

iResult = bind(recvSocket, (SOCKADDR *) &recvAddr, sizeof(recvAddr));
char recvBuf[1024];
int bufLen = 1024;
sockaddr_in senderAddr;
int senderAddrSize = sizeof (senderAddr);

// Call the recvfrom function to receive datagrams on the bound socket.
iResult = recvfrom(recvSocket, recvBuf, bufLen, 0,
            (SOCKADDR *) & senderAddr, &senderAddrSize));
```

```
if (iResult == SOCKET_ERROR)
{
     printf("recvfrom failed with error %d\n", WSAGetLastError());
     return 1;
}

// Add null to the end of string and print the message
recvBuf[iResult] = '\0';
printf("Message: %s", recvBuf);

// Close the socket when finished receiving datagrams
printf("Finished receiving. Closing socket.\n");
iResult = closesocket(recvSocket);
if (iResult == SOCKET_ERROR)
{
    printf("closesocket failed with error %d\n", WSAGetLastError());
    return 1;
}
```

Zadatak 1

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

1. Skicirati implementaciju klijenta
2. Skicirati implementaciju servera
3. Omogućiti klijentu da pošalje više poruka serveru.
4. Omogućiti komunikaciju između dva različita računara.
   Napomena: IP adresu drugog računara moguće je dobaviti korišćenjem ipconfig naredbe u command promt-u.

Zadatak 2

Date implementacije klijenta i servera koji koriste UDP protokol potrebno je unaprediti na sledeći način:

1. Kreirati novi solution u kome se nalaze dva projekta Server i Client .
2. Obezbediti serverskoj UDP utičnici `serverSocket` da može da prima poruke upućene ka adresiračunara na kome se program izvršava i na portu 15001.
3. Obezbediti klijentu da pošalje više od jedne poruke serveru.
4. Omogućiti završetak rada klijenta korišćenjem naredbe "stop client".
5. Primljenu poruku sa servera proslediti (poslati nazad) klijentu.
6. Na klijentu ispisati odgovor dobijen od servera.
7. Omogućiti gašenje servera ukoliko sa klijenta uzastopno stignu dve identične poruke.
8. Koristeći debugging mode (*prekidne tačke i watch window*) saznati ASCII vrednost slova „a“ , „z“ , „A“ i „Z“. Kao dokaz priložiti screenshot ekrana u kome je prikazan deo koda na kome se nalazi prekidna tačka i prikaz watch window-a.
9. Nakon prijema svake poruke, na serveru ispisati: dužinu primljene poruke, broj malih slova u tojporuci, broj velikih slova u poruci i broj drugih znakova (karaktera) u toj poruci.
10. Isprobati programsko rešenje korišćenjem više klijentskih aplikacija istovremeno.