Izvedba, analiza i testiranje programske mrežne priključnice

Čehok, Marko

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:632987

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

University North Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI



Sveučilište
Sjever

Završni rad br. 381/EL/2016

Izvedba, analiza i testiranje programske mrežne priključnice

Marko Čehok, 2117/601

Varaždin, rujan 2016. godine



Sveučilište
Sjever

Odjel Elektrotehnike

Završni rad br. 381/EL/2016

Izvedba, analiza i testiranje programske mrežne priključnice

Student

Marko Čehok, 2117/601

Mentor

mr. sc. Matija Mikac, dipl. ing. el.

Varaždin, rujan 2016. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

UNIVERSITY NORTH

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za elektrotehniku

PRISTUPNIK: Marko Čehok | MATIČNI BROJ: 2117/601

DATUM: 05.09.2016 | KOLEGIJ: Programski jezici i algoritmi; Računalne mreže

NASLOV RADA: Izvedba, analiza i testiranje programske mrežne priključnice

NASLOV RADA NA ENGL. JEZIKU: Network socket implementation and performance analysis

MENTOR: Matija Mikac | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:

1. dr.sc. Ladislav Havaš, dipl.ing.el.
2. mr.sc. Ivan Šumiga, dipl.ing.el.
3. mr.sc. Matija Mikac, dipl.ing.el.
4. Miroslav Horvatić, dipl.ing.el.
5.

VŽ KC

MMI

Zadatak završnog rada

BROJ: 381/EL/2016

OPIS:

Mrežne priključnice (eng. network socket) predstavljaju uobičajen način za realizaciju komunikacije u računalnim mrežama. Obzirom na konvergenciju današnjih mreža, pri čemu su gotovo sve mreže bazirane na TCP/IP skupini protokola, primjena TCP/IP mrežnih priključnica (eng. Internet socket) uobičajena je na svim tipovima uređaja.

Standardno, mrežna priključnica se definira mrežnom adresom, transportnim protokolom i "transportnom adresom", tzv. vratima (eng. port). Za realizaciju komunikacije potrebno je na uređajima i programski implementirati rad sa priključnicama - programsku podršku za to nudi većina programskih jezika, pri čemu se oslanjaju na podršku operacijskih sustava koji pogone uređaje. Iako ih svakodnevno koristimo, priključnice su standardnim korisnicima mreža "sakrivene", implementirane u uobičajenom softveru.

U okviru ovog završnog rada potrebno je razmotriti mogućnosti programske realizacije TCP/IP mrežne priključnice, izvesti testne programe i provesti analizu dobivenih rezultata. Implementaciju provesti korištenjem programskog jezika C, za Windows operacijski sustav. Usput, prema primjerima, provjeriti i osnovne principe implementacije u programskom jeziku Java.

U radu je potrebno:
* objasniti pojam mrežne priključnice i dati kratki pregled mrežnih protokola (u okviru TCP/IP skupine) vezanih uz taj pojam
* dati kratki osvrt na princip komunikacije između poslužitelja i klijenta, uz naglasak na princip rada TCP/IP mrežnih priključnica
* proučiti mogućnosti programske implementacije mrežnih priključnica, primarno pouzdanih TCP priključnica
* programski realizirati poslužiteljski i klijentski program za Windows operacijski sustav koji će koristiti TCP priključnice
* osim jednostavne tzv. blokirajuće priključnice 1:1, realizirati poslužitelj koji izvodi višedretvenu obradu zahtjeva za klijente
* analizirati performanse realiziranih programa, ovisno o broju poslužitelja i klijenata - na istom računalu i unutar lokalne mreže
* provjeriti skalabilnost izvedenih programa i ukazati na moguća ograničenja koja se javljaju

ZADATAK URUČEN: 12.09.2016. | POTPIS MENTORA: M. Mikac

REPUBLIKA HRVATSKA KOPIRNICA SVEUČILIŠTE SJEVER

Predgovor

Završni rad pod nazivom „Izvedba, analiza i testiranje programske mrežne priključnice“ nastao je kao rezultat proučavanja funkcioniranja mrežne komunikacije u aplikacijama – konkretno, proučavanja funkcioniranja mrežnih priključnica i njihove programske realizacije. Završni rad, osim detalja o izvedbi praktičnog dijela (programske izvedbe mrežne priključnice), daje i kratak pregled mrežnih protokola vezanih za pojam mrežne priključnice. Praktični dio izveden je u C/C++ programskom jeziku, za Windows platformu korištenjem standardnih knjižnica za rad s priključnicama (*WinSock*).

Sažetak

Tema završnog rada je izvedba, analiza i testiranje programske mrežne priključnice. Ideja završnog rada je bila da se u C/C++ programskom jeziku realizira program za jednostavno mrežno povezivanje dvaju ili više računala, to je i realizirano kao praktični dio rada - razvijeni su programi u kojima su implementirani klijent i poslužitelj koji koriste mrežne priključnice. Glavnina završnog rada odnosi se na pregled principa korištenih u izvedbi praktičnog dijela, uz sažet pregled teoretskih osnova vezanih protokola i programskih knjižnica. Realizirani programi detaljno su testirani te je provedena i, kao sastavni dio rada, dokumentirana osnovna analiza performansi.

Ključne riječi: TCP/IP mrežni model, računalna mreža, internetska/mrežna priključnica (engl. *Internet/network socket*), C/C++ programski jezik, WinSock knjižnica.

Abstract

The topic of this final thesis is 'Design, analysis and testing of network sockets'. The idea for this final thesis was to create a program for simple network connection of two or more computers. Practical part of the thesis consists of network socket implementation into simple client/server programs. Majority of final thesis provides an overview on the principles used in the implementation of practical part, with a concise overview on theoretical basis of related protocols and program library. Integral part of this thesis is documented conducted performance analysis of implemented programs, which are thoroughly tested.

Keywords: TCP/IP computer networking model, computer network, Internet/network socket, C/C++ programming language, WinSock library.

Popis korištenih kratica

OS	Operativni sustav (engl. *Operating System)*
IP	Internet protokol (engl. *Internet Protocol*), uobičajeni protokol mrežnog sloja
DARPA	Agencija Ministarstva obrane SAD-a (engl. *Defense Advanced Research Projects Agency*)
UDP	Nepouzdan transportni protokol za prijenos podataka (engl. *User Datagram Protocol)*
TCP	Pouzdan transportni protokol, protokol s kontrolom toka (engl. *Transmission Control Protocol*)
WAN	Mreža širokog područja (engl. *Wide Area Network*)
LAN	Lokalna računalna mreža (engl. *Local Area Network*)
MAC	Kontrola pristupa mediju (engl. *Media Access Control)*
IPv4	Internetski protokol verzije 4 (engl. *Internet Protocol version 4*)
IPv6	Internetski protokol verzije 6 (engl. *Internet Protocol version 6*)
NAT	Protokol za prevođenje mrežnih adresa (engl. *Network Address Translation*)
IANA	Organizacija za numerizaciju na Internetu (engl. *Internet Assigned Numbers Authority*)
BSD	Berkeley distribucija softvera (engl. *Berkeley Software Distribution*)
DLL	Dinamička knjižnica (engl. *Dynamic-link library*)

Sadržaj

1. Uvod 1
2. Osnove mrežnih protokola vezanih za mrežne priključnice 3

2.1. TCP/IP model 3
2.2. Enkapsulacija 5
2.3. Mrežna priključnica 6
2.3.1. Mrežni sloj – IP protokol 6
2.3.2. Transportni sloj – TCP/UDP 7
2.3.3. Transportni sloj - vrata 9
2.3.4. Uobičajeno korištenje mrežnih priključnica 10

3. Princip rada i programiranje mrežnih priključnica 11

3.1. Klijentska priključnica 12
3.1.1. Stvaranje priključnice 12
3.1.2. Spajanje klijenta s poslužiteljem 13
3.1.3. Klijent – slanje/primanje 13
3.1.4. Zatvaranje priključnice 15

3.2. Poslužiteljska priključnica 15
3.2.1. Inicijalizacija priključnice 15
3.2.2. Vezivanje priključnice 16
3.2.3. Osluškivanje na priključnici 17
3.2.4. Prihvaćanje klijenta 17
3.2.5. Primanje podataka 18
3.2.6. Zatvaranje priključnice 18

3.3. Moguće izvedbe programske priključnice 18
3.3.1. Blokirajuća priključnica 19
3.3.2. Priključnica s nitima 20

4. Programska izvedba mrežnih priključnica 21

4.1. Blokirajuća internetska priključnica 21
4.1.1. Klijent – poslužitelj: početna verzija 22
4.1.2. Klijent – poslužitelj: ispis poruke 23
4.1.3. Klijent – poslužitelj: izmjena poruka 23

4.2. Priključnica s višestrukim klijentima 25
4.2.1. Klijent 25
4.2.2. Poslužitelj 27

5. Rezultati analize i testiranja programske izvedbe 29

5.1. Jedan poslužitelj, jedan klijent 30
5.2. Jedan poslužitelj, više klijenata 31
5.3. Više poslužitelja s više klijenata 32
5.4. Analiza posluživanja 200 klijenata 37
5.4.1. Klijenti i poslužitelj na računalu 1 37
5.4.2. Klijenti i poslužitelji na računalu 2 37
5.4.3. Poslužitelji i klijenti u mreži 38
5.4.4. Kombinacija poslužitelja/ klijenta između 2 računala 39

5.5. Analiza skalabilnosti razvijenih programa 39
5.6. Veličina TCP paketa 44

6. Zaključak 47

7. Literatura.....48

1. Uvod

U današnje vrijeme, svugdje smo okruženi umreženim uređajima, raznim mikrokontrolerima, automatizacijom sustava, regulacijom i mjerenjem raznih parametara sustava. To su primjeri u kojima će vjerojatno trebati mrežna komunikacija, a u nekim slučajevima i izvedba i programiranje takve komunikacije, ponekad i na nižoj razini od očekivane.

Istraživanjem načina kako se programski rješava povezivanje i razmjena podataka, stižemo do pojma mrežne priključnice (engl. *network socket*). Danas u većini slučajeva kada se spominje mrežna priključnica govorimo o internetskoj priključnici (engl. *Internet socket*), s obzirom da IP kao mrežni sloj dominira u svim aktualnim mrežama. U radu se pojam mrežna priključnica odnosi upravo na Internet priključnicu, a kod programske izvedbe koristi se pojam programska mrežna priključnica koja se odnosi na realizaciju.

Mrežne priključnice u programiranju su krajnje točke komunikacije. One u sebi sadrže tri bitna parametra koji ih definiraju: IP adresu, vrata (engl. *port*) i vrstu protokola.

Mrežne priključnice koje se koriste u radu su isključivo standardnog oblika definiranog u internetskim priključnicama TCP/IP mreža. Kao praktični dio rada izvedena je programska realizacija priključnica u jeziku C/C++, za mrežni podsustav Windows platforme, korištenjem standardne WinSock knjižnice.

Obzirom na parametre kojima definiramo mrežnu priključnicu, prije svega vezano uz vrstu transportnog protokola, razlikujemo i možemo realizirati dvije osnovne vrste priključnica. Jedna od njih, nama bitnija, je tzv. spojna priključnica (engl. s*tream socket*) koja koristi TCP protokol. Osigurava uspostavu logičke veze, kontrolu toka i zagušenja. Radi se o pouzdanom tipu veze u kojem se zadržava redoslijed podataka, po potrebi radi retransmisiju i druge postupke. Realizirana rješenja odnose se na TCP protokol i spojnu priključnicu.

Druga vrsta priključnice je tzv. paketna priključnica (engl. *datagram socket*) koja koristi UDP protokol. Ova vrsta komunikacije nema uspostavu veze između primatelja i pošiljatelja. Dijelovi komunikacije se mogu pogubiti na putu.

Praktični dio rada daje uvid u osnovne funkcije potrebne za programsku izvedbu mrežnih priključnica, pregled razvoja programa klijent – poslužitelj. Glavni dio završnog rada opisuje postupak testiranja i daje rezultate mjerenja nad izvedenim programskim rješenjem. Analiziraju se performanse različitih izvedba i kombinacije poslužitelja i klijenata.

Završni rad započinje pregledom osnovnih mrežnih protokola vezanih za mrežne priključnice. Nakon toga slijedi uvid u potrebne funkcije za izvedbu mrežnih priključnica kako na poslužiteljskoj tako i na klijentskoj strani. Kroz opis ključnih elemenata programske izvedbe

poslužitelja i klijenta daje se kratki pregled primjena tih funkcija. Na kraju završnog rada prikazani su i opisani načini testiranja i mjerenja, te prezentirani i komentirani dobiveni rezultati.

2. Osnove mrežnih protokola vezanih za mrežne priključnice

Komunikacija unutar računalne mreže ne zbiva se bez pravila. Mrežni protokoli opisuju pravila komunikacije umreženih računala. Potrebni protokoli za komunikaciju unutar mreže su grupirani i organizirani hijerarhijski, što je poznato kao slojevita arhitektura. Među ostalim protokoli su „zaslužni“ za oblikovanje podataka, proces odašiljanja podataka, javljanje pogrešaka te početak i prekid komunikacije između uređaja na mreži.

U današnjim računalnim mrežama komunikacija se dominantno obavlja korištenjem TCP/IP grupe protokola. Tijekom zadnjih dvadesetak godina došlo je do konvergencije različitih mreža. Te se mreže danas većinom razlikuju isključivo na fizikalnom sloju (ADSL, Ethernet, mobilna mreža, i slično).

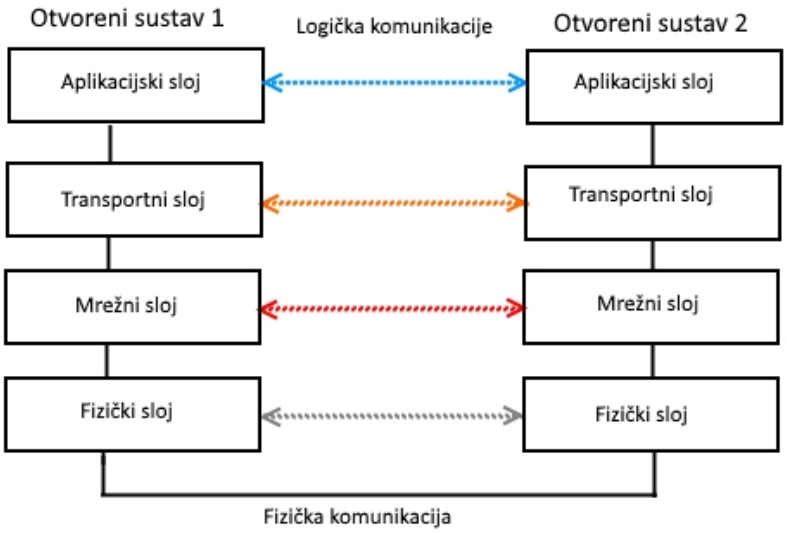
2.1. TCP/IP model

TCP/IP model je definiran unutar standarda RFC1122 [1], pod drugim imenom, Internet Protocol grupom. U sebi sadrži uvjete i pravila kako se obavlja komunikacija od jedne točke do druge, određuje kako se podaci adresiraju, pakiraju, odašilju, usmjeravaju i primaju. TCP/IP model možemo razdijeliti na četiri sloja, gdje pojedini sloj ovisi o sloju ispod sebe, a daje podršku za sloj iznad sebe [2]. Na dnu postoji fizički sloj koji je zaslužan za fizičku vezu između dva uređaja i način komunikacije preko medija. Iznad fizičkog sloja nalazi se mrežni sloj koji je zaslužan za usmjeravanje informacija na točnu mrežnu adresu. Transportni sloj rukuje podacima i provjerava moguće pogreške. Najviši sloj je aplikacijski, koji prima podatke i predstavlja ih korisniku u razumljivom obliku. Neki od poznatijih protokola u pojedinom sloju TCP/IP modela vidljivi su iz tablice 2.1.

TCP/IP model	Protokoli
4. Aplikacijski sloj	FTP, SSH, HTTP…
3. Transportni sloj	UDP, TCP
2. Mrežni sloj	IP
1. Fizički sloj	Ethernet, IEEE 802.11…

Tablica 2.1. Internet RFC1122 TCP/IP model

Kod uspostave komunikacije između dvaju računala, svaki pojedini sloj TCP/IP modela zadužen je za komunikaciju s istim tim slojem na drugom računalu (slika 2.1.).



Slika 2.1. Prikaz komunikacije između otvorenih sustava - horizontalna komunikacija

Komunikacija započinje s aplikacijskim slojem, gdje korisnik šalje neke podatke. Aplikacija ima svoje protokole – kako čitati i pisati te podatke. Transportni protokol tada te podatke oblikuje i dodaje informacije o servisu. Mrežni sloj dobiva od transportnog sloja preoblikovane paketiće podataka i na njih dodaje adresu primatelja i pošiljatelja. Fizički sloj je zadužen da preko fizičkog medija pošalje sve što mu mrežni sloj isporuči [3].

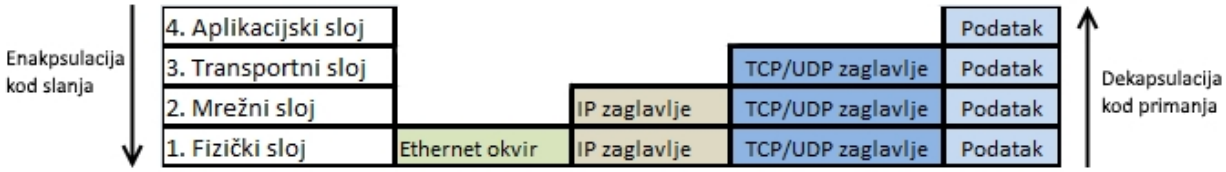
Na odredištu primljeni podaci tada od fizičkog sloja putuju prema aplikacijskom sloju i svaki pojedini sloj čita dio koji je namijenjen za njega. Cijeli taj postupak je znan i kao enkapsulacija.

Mrežne priključnice definiraju se na transportnom i mrežnom sloju. Za definiranje se koriste parametri: mrežna adresa, transportni protokol i vrata.

2.2. Enkapsulacija

Pod enkapsulacijom smatramo dodavanje ili omatanje čistog podatka s dodatnim protokol-informacijama; taj princip rada s podacima primjenjuje se u vertikalnoj komunikaciji među slojevima. Kako podatak putuje od aplikacijskog sloja prema fizičkom sloju, na njega pojedini protokol dodaje svoje kontrolne podatke i time se početni paket podataka omota slojevito i šalje na kraju prema odredištu. Primatelj tada mora početi s najnižim slojem prema najvišem da bi svaki protokol na određenom sloju pročitao i usmjerio podatak prema krajnjoj aplikaciji za koju je podatak namijenjen. Također, kako paket putuje mrežom, usmjerivači i preklopnici će također odmotati i pročitati određeni dio enkapsulacije i ponovo zapakirati paket, s ciljem da se paket usmjeri prema sljedećem čvorištu sve do odredišta. [4]

Pojednostavljeni prikaz je vidljiv iz slike 2.2.



Slika 2.2. Princip enkapsulacije

Aplikacijski sloj oblikuje podatak i dodaje mu zaglavlje koje sadrži informacije potrebne za rukovanje podatkom kod primljene aplikacije. To su korisničke informacije, sve ostale informacije koje drugi slojevi dodaju pripadaju upravljačkim informacijama.

Transportni sloj prima podatak od aplikacijskog sloja, omota taj podatak u TCP paket koji sadrži TCP zaglavlje.

Mrežni sloj primi TCP paket i dodaje IP informacije. Taj IP datagram se dalje predaje fizičkom sloju.

Fizički sloj paketu dodaje okvir (engl. *frame*), znači zaglavlje i podnožje u kojem je sadržana izvorišna i odredišna MAC adresa. Nadalje je paket poznat kao okvir.

Prilikom dolaska okvira na svako čvorište u mreži, uređaj čita okvir da bi znao kamo ga proslijediti, prema sljedećem čvorištu ili na viši sloj. Tako preklopnici (engl. *switch*) čitaju samo MAC adrese i njima se služe da usmjere okvir prema odredištu, dok usmjernici (engl. *router*) čitaju IP adrese u paketu i njima se služe za određivanje kamo paket treba usmjeriti.

Na odredištu se paket dekapsulira od fizičkog sloja prema aplikacijskom sloju i oduzima mu se pojedino zaglavlje kako stiže prema aplikaciji. [4]

2.3. Mrežna priključnica

Mrežna priključnica je točka komunikacije između dva procesa preko računalne mreže. Spada u transportni sloj TCP/IP modela jer usmjerava primljene podatke prema aplikaciji za koju su namijenjeni, a i definira transportni protokol koji će se koristiti za komunikaciju između dva procesa.

Da bi se dvije mrežne priključnice mogle logički spojiti, potrebno je definirati tri parametra na svakoj priključnici: IP adresu, broj vrata i vrstu protokola za transport. Bit će ukratko opisan svaki od potrebnih parametara.

Način implementacije mrežnih priključnica definira programsko sučelje u kojem se radi.

2.3.1. Mrežni sloj – IP protokol

Internet protokol je standardni protokol mrežnog sloja. Zadužen je za transport paketa od izvora, kroz mrežu, do odredišta definiranog mrežnom adresom. Oblik mrežne adrese definira upravo Internet protokol, poznatije pod imenom IP adrese. Definiraju se dvije verzije IP adresa IPv4 i IPv6, ovisno o verziji IP protokola koji se koristi.

IP adresa je brojčana oznaka dodijeljena uređajima koji sudjeluju u mrežnoj komunikaciji. U današnje vrijeme, na Internetu su istovremeno u upotrebi dvije verzije IP adresiranja: IPv4 i IPv6. IPv4 se polako zamjenjuje s IPv6, ali budući da IPv6 nije kompatibilan s IPv4, oba se koriste istovremeno pomoću tuneliranja i dual IP stack rješenja. [5]

Adrese IPv4 se sastoje od 32 bita, kojima je moguće ostvariti 2^{32} adresnih mjesta, što je jednako 4,2 milijarde adresnih mjesta. One su se počele iscrpljivati još 1990. godine, dok u današnje vrijeme skoro da nema slobodnih IPv4 adresa. IPv4 se, radi lakšeg pamćenja, zapisuje u dekadskom obliku, gdje je 32-bitni broj podijeljeni na četiri 8-bitna broja, koji se zatim prikazuju kao četiri decimalna broja odvojena točkom. Brojevi su u rasponu od 0 do 255. [6]

Svako računalo koje je povezano na Internet mora imati jedinstvenu IP adresu. Ta adresa naziva se i globalna. Svatko tko ima pristup Internetu, ima dodijeljenu jedinstvenu globalnu adresu.

Druga vrsta IPv4 adresa su privatne adrese. Te adrese se nikad ne smiju i neće pojaviti na globalnoj razini. Privatne adrese se koriste u privatnim zatvorenim mrežama bez izlaza na Internet, a ako postoji izlaz na Internet, onda se koristi NAT (engl. *Network address translation*) protokol. Ona omogućava privatnoj mreži da koristi samo jednu globalnu adresu kao izlaznu za sve svoje korisnike.

Na mrežnom sloju kod IP zaglavlja za mrežne priključnice bitne su izvorišna i odredišna IP adresa (slika 2.3.).



Slika 2.3. IP zaglavlje

Kada radimo sa programskim mrežnim priključnicama ne pristupamo direktno IP paketu i ne definiramo upravljačke informacije, već u WinSock prosljeđujemo osnovne parametre i to rezultira time da mrežni podsustav operacijskog sustava parametre postavi na za to predviđena mjesta po slojevima. Na klijentskoj strani zadajemo u programsku priključnicu IP adresu poslužitelja kao odredišnu adresu, izvorišna adresa se popunjava samostalno od strane protokola. Poslužitelj ima dodijeljenu izvorišnu adresu; ta adresa se postavlja u programsku priključnicu koja ga postavlja u IP zaglavlje. Kada se klijent poveže, njegova izvorišna adresa se uzima i postavlja od strane operativnog sistema kao odredišna adresa u IP zaglavlje poslužitelja.

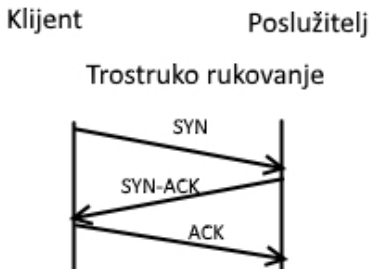
2.3.2. Transportni sloj – TCP/UDP

Kao što je već spomenuto, rad s mrežnim priključnicama odvija se na transportnom sloju TCP/IP modela. Dva glavna protokola u transportnom sloju su TCP (engl. *Transmission Control Protocol*) i UDP (engl. *User datagram protocol*).

Način slanja podataka preko mrežne priključnice ovisi o tome koji transportni protokol odlučimo koristiti u mrežnoj priključnici.

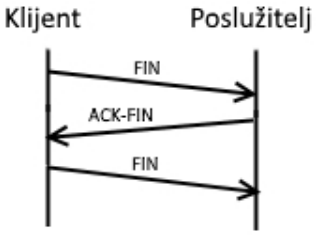
Tako, ako se koristi UDP, dobijemo paketnu priključnicu koja ima karakteristike UDP protokola, što znači da ne postoji uspostava direktne komunikacije između primatelja i pošiljatelja. Komunikacija je nepouzdana, podaci se u paketićima šalju u mrežu. Ti paketići se na putu prema odredištu mogu lako pogubiti. Ova vrsta priključnice koristi se kod servisa kojima nije bitno ako se pokoji dio podatka izgubi ili odbaci, nego je bitna brzina prijenosa (videokonferencije, zvučni razgovori, mrežne igre). [7]

Drugi protokol transportnog sloja, TCP, je onaj s kojim su ostvarene mrežne priključnice u ovom radu. Priključnice koje se dobiju korištenjem TCP protokola su takozvane spojne priključnice (engl. *stream socket*). Kod ovih priključnica imamo uspostavu komunikacije između klijenta i poslužitelja preko trostrukog rukovanja (engl. *three-way handshake*). [7]



Slika 2.4. Uspostava TCP veze trostrukim rukovanjem

Trostruko rukovanje se odvija na početku uspostava konekcije između dvaju računala. Računalo koje traži uspostavu konekcije – klijent – šalje sinkronizirajuću poruku SYN (postavlja se SYN zastavica u kontrolnim bitovima TCP zaglavlja), kojom obavještava drugo računalo – poslužitelja – da želi komunicirati, i slijedni broj. Ako je poslužitelj slobodan, šalje segment koji sadrži potvrdu prijema i svoj slijedni broj. Na kraju klijent šalje segment kojim potvrđuje prijem parametra i počinje slanje podataka. Kada je prijenos podataka završen, veza se raskida s razmjenom segmenata koji sadrže upravljačke informacije i zastavicu za FIN, koja označava prekid veze. [8]



Slika 2.5. Prekid TCP veze trostrukim rukovanjem

TCP protokol uzima podatak od aplikacijskog sloja i dodaje mu TCP zaglavlje. Zaglavlje se sastoji od minimalno 20 okteta (slika 2.6).



Slika 2.6. TCP zaglavlje

Zaglavlje sadrži informacije o izvorišnim i odredišnim brojevima vrata i te vrijednosti su vezane za rad s priključnicama. Pri tome se odredišna vrata definiraju, dok izvorišna vrata TCP protokol odredi kod klijenta prilikom uspostave konekcije s poslužiteljem. Kod poslužitelja su izvorišna vrata određena i kod tog broja se čeka na konekciju, a odredišna adresa se popunjava informacijama klijenta kada se uspostavi veza.

Ostale informacije TCP zaglavlja postavlja protokol.

2.3.3. Transportni sloj - vrata

U računalnim mrežama vrata su krajnja točka komunikacije u operativnom sustavu i definiraju određeni proces ili mrežni servis, vrata su vrsta adresiranja na transportnom sloju i adresa se predstavlja 16 bitnim brojem (engl. *Port number*). Vrata se uvijek povezuje s IP adresom poslužitelja i protokolom koji služi za komunikaciju i time nadopunjuje adresu komunikacijske sesije. [9]

Vrata su određena za pojedinu adresu i protokol 16-bitnim brojem. Brojka vrata se kreće u rasponu od 0 do 65.535. Prvih 1024 brojeva vrata su takozvani općepoznati brojevi (engl. *Well-known port numbers*) rezervirani najviše za servise preko WWW-a (engl. *World Wide Web*). Pokraj općepoznatih brojeva vrata postoje i registrirani brojevi koji se kreću u rasponu od 1024 do 49.151. O njima se brine agencija IANA (engl. *Internet Assigned Numbers Authority*) i mogu se provjeriti na njihovim listama dostupnim na Internetu. [6]

Brojevi vrata od 49.152 do 65.535 su dinamički brojevi dostupni za javnost (takozvani engl. *Ephemeral ports*). [10]

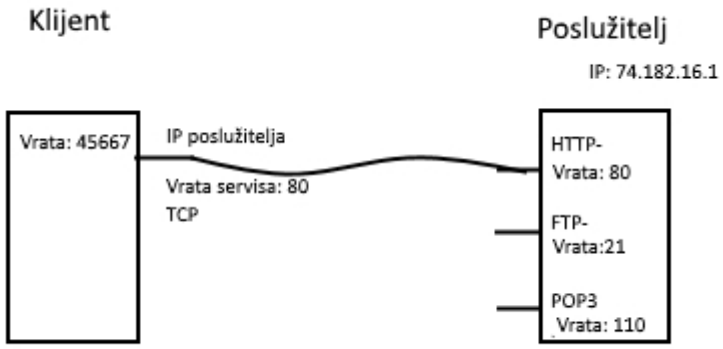
Ephemeral brojevi kod Linux sustava koriste opseg od 32.768 do 61.000 od verzije operativnog sustava 4.6 nadalje. Kod najnovijih Windows sustava koriste se IANA opsezi. No, postoji opcija da se posebno odredi opseg broj vrata unutar opsega 1025-65535. [10]

Kod svake uspostave komunikacije koriste se dva različita broja vrata. Izvorni broj vrata spaja se na odredišni broj vrata.

Za naše vlastite primjene mogu se koristiti svi brojevi iznad 1024 (može i brojeve niže od 1024), ali ako se stvara uspostava veze prema globalnoj mreži, onda treba voditi računa s kojom vrstom usluge želimo uspostaviti komunikaciju.

2.3.4. Uobičajeno korištenje mrežnih priključnica

Primjer uobičajenog korištenja mrežnih priključnica može biti pristup web poslužitelju prilikom dohvata nekih web sadržaja. Opće je poznat broj vrata za spajanje na http. IP adresa poslužitelja može se dobiti iz simboličke adrese (domene) korištenjem sustava DNS, ili ako se zna direktna IP adresa poslužitelja. HTTP protokol kao transportni protokol koristi TCP.



Slika 2.7. Primjer korištenja mrežne priključnice

Slika 2.7. prikazuje klijenta i poslužitelja. Poslužitelj na sebi ima stvoreno nekoliko priključnica, gdje svaka osluškuje na svojim vratima. Klijent stvara priključnicu koja ima u sebi zadane parametre IP adrese poslužitelja na koji se povezuje, broj vrata za servis koji želi koristiti i način uspostavljanja veze. Ako je veza uspješno uspostavljena, klijent je povezan na poslužitelja na traženim vratima servisa, a poslužitelj bilježi broj klijentskih vrata i IP adresu da može usmjeriti tražene podatke na točan cilj. Uobičajena izvedba poslužitelja omogućava da poslužitelj na isti broj vrata prima više klijenata, te ih obrađuje redom.

3. Princip rada i programiranje mrežnih priključnica

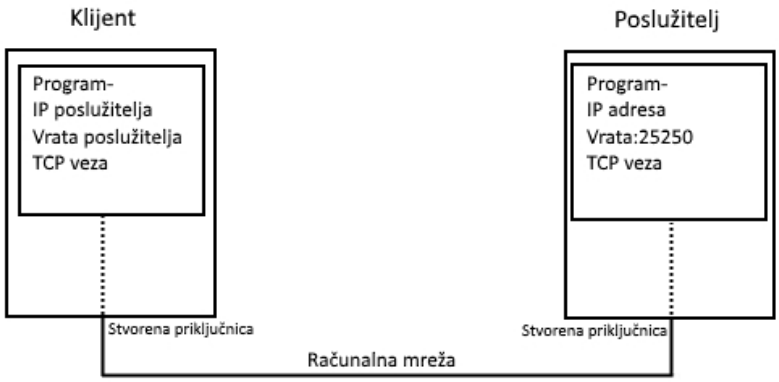
U ovom poglavlju razmatraju se osnovne funkcije i dijelovi programskoga koda potrebni za stvaranje osnovnoga programa klijent – poslužitelj.

Mrežna priključnica se povezuje s određenom IP adresom i brojem vrata. Kao što je već navedeno, postoje dvije vrste priključnica: paketna priključnica povezana s UDP načinom prijenosa podataka i *stream*-priključnica povezana s TCP načinom slanja podataka. Izvedba programske realizacije priključnica zasnovana je na TCP uspostavi veze.

Za programsku izvedbu mrežnih priključnica koristilo se Window aplikacijsko programsko sučelje (engl. *Application Programming interface API*) za priključnice iz programske knjižnice Winsock2 realizirane u C/C++ programskom jeziku. Mnoge funkcije su realizirane u istom stilu kao i funkcije Barkley priključnice koje se koriste u BSD Unix sustavu.

Dodatno se realizirala mrežna priključnica u Java programskom jeziku da se ukaže na mogućnosti i principe stvaranja priključnice iz njihovih knjižnica.

Za stvaranje priključnice između klijenta i poslužitelja potrebno je nekoliko osnovnih programskih funkcija.



Slika 3.1. Koncept korištenja priključnice

U programu se definiraju parametri priključnice koja zatim stvara mrežnu priključnicu koja uspostavlja vezu između klijenta i poslužitelja.

Programske priključnice, osim definiranja parametara, određuju podatke koji će se poslati, ostale funkcije realizira mrežni podsustav operacijskog sustava (segmentacija, rukovanje konekcijom, poštovanje protokola).

U sljedećim poglavljima detaljnije se razrađuju korištene programske funkcije za stvaranje priključnice, uspostava veze između klijenta i poslužitelja, slanje i primanje podataka.

Koriste se funkcije iz Winsock knjižnice.

3.1. Klijentska priključnica

Za programiranje klijentske strane priključnica koristi se Windowsov operacijski sustav i programsko razvojno okruženje Visual Studio 2012.

Korištene su knjižice za rad s priključnicama: `<winsock2.h>`, koje spadaju u čisti C jezik.

Za rad klijentske priključnice potrebno je:

- stvoriti priključnicu
- spojiti klijentsku priključnicu s poslužiteljem
- slanje i primanje na klijentskoj priključnici
- zatvoriti priključnicu.

3.1.1. Stvaranje priključnice

Programske priključnice stvara se određivanjem imena priključnice pozivanjem strukture *SOCKET*.

Da bismo tu programsku priključnicu pretvorili u mrežnu, potrebno joj je zadati parametre IP adrese, broj vrata, transportni protokol.

Sintaksa za stvaranje mrežne priključnice izgleda ovako:

```
SOCKET WSAAPI socket( _In_ int af, _In_ int type, _In_ int protocol);
```

Svaki parametar se određuje jednom od preddefiniranih konstanta iz knjižnice.

Af (in) – specifikacije verzije IP adrese, `AF_INET` za IPv4, `AF_INET6` za IPv6 itd.

Type (in) – specifikacije za vrstu priključnice, `SOCK_STREAM` za TCP vezu, `SOCK_DGRAM` za UDP vezu.

Protocol (in) – vrsta transportnog protokola u upotrebi, ograničena odabranom verzijom IP adrese i vrstom priključnice. `IPPROTO_TCP` za TCP protokol, `IPPROTO_UDP` za UDP protokol.[11]

Primjer realizacije:

```
int socAddrFamily;
int socType;
int socProtocol;
socAddrFamily = AF_INET;
socType = SOCK_STREAM;
socProtocol = IPPROTO_TCP;
SOCKET ConnS;
ConnS = socket(socAddrFamily, socType, socProtocol);
if (ConnS == INVALID_SOCKET){
      cout<< "Greska kod stvaranja prikljucnice: "<< WSAGetLastError()<< endl;
      WSACleanup();}
else  cout<< "Stvaranje prikljucnice uspjelo!"<< endl;
```

3.1.2. Spajanje klijenta s poslužiteljem

Za spajanje klijenta s poslužiteljem koristimo funkciju *connect()* koja u sebi sadrži stvorenu priključnicu i informacije o poslužitelju na koji se povezujemo. Znači, sadrži IP adresu i broj vrata poslužitelja. [12]

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

s (in) – deskriptor za korištenu priključnicu

name (in) – pokazivač (engl. *pointer)* na *sockaddr* strukturu s kojom se povezuje

namelen (in) – duljina u bajtovima *sockaddr* strukture na koju pokazuje *name* parametar.

Informacije o poslužitelju se preddefiniraju u strukturi koja sadrži informacije o IP adresi i broj vrata.

Primjer realizacije:

```
sockaddr_in iServ;
iServ.sin_addr.s_addr = inet_addr("127.0.0.1");
iServ.sin_family = AF_INET;
iServ.sin_port = htons(25252);

long spoj = 0;
spoj = connect (ConnS, (sockaddr*) & iServ, sizeof (iServ));
if  (spoj == SOCKET_ERROR)
{
      cout<< "Greska kod povezivanja sa posluziteljem: "<< WSAGetLastError()<< endl;
      spoj = closesocket(ConnS);
}
else cout<< "Povezano!"<<endl;
```

3.1.3. Klijent – slanje/primanje

Kada je klijent uspješno povezan na poslužitelj, može započeti slanje podataka.

Šalje se funkcijom *send()* u kojoj se nalaze parametri: stvorena priključnica, međuspremnik i veličina međuspremnika te dodatne zastavice po potrebi. [13]

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

s (in) – deskriptor za korištenu priključnicu

buf (in) – pokazivač na međuspremnik s podacima za slanje

len (in) – duljina podatka sadržanog u međuspremniku u bajtovima

flags (in) – zastavice koje određuju način poziva. Primjeri u radu ne sadrže zastavice.

Međuspremnike smo predodredili i popunili prije pozivanja funkcije *send()*.

Međuspremnik u završnom radu je definiran na veličinu 1024 bajta, puni se podacima koji se šalju. Priključnica šalje međuspremnik kao slijed bitova. Koliko je ukupno bitova podataka u

tom međuspremniku, određeno je s *_in_ int len* koji ukazuje na duljinu podatka u međuspremniku.

Primjer realizacije:

```
char *sendbuf ="This is a test";
int iPom;
iPom =send(ConnS, sendbuf,(int) strlen(sendbuf), 0);
if (iPom == SOCKET_ERROR)
{
      cout<< "Greska kod slanja:"<< WSAGetLastError()<< endl;
      closesocket(ConnS);
}
```

Kod primanja koristimo funkciju *recv()* koja sadrži: stvorenu priključnicu, predodređeni međuspremnik i duljinu međuspremnika te po potrebi dodatne zastavice. [14]

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

s (in) – deskriptor za korištenu priključnicu

buf (out) – pokazivač na međuspremnik za primljene podatke

len (in) – duljina međuspremnika za primljene podatke u bajtovima

flags (in) – zastavice koje određuju način rada funkcije. Primjeri u radu ne sadrže zastavice.

Kod primanja, povratna vrijednost *recv()* funkcije je duljina podatka koji se prima.

Primljeni podaci se pohranjuju u međuspremnik za primljene podatke koji je također u primjerima u završnom radu veličine 1024 byta.

Primjer realizacije:

```
int recvbuflen = DEFAULT_BUFLEN;
char recvbuf[DEFAULT_BUFLEN];
DWORD retVal = 0;
do {
      retVal= recv (ConnS, recvbuf, recvbuflen, 0);
      if (retVal > 0)
      {
                  cout<< string(recvbuf, retVal) <<endl;
                  cout<< "Bitovi primljeni: "<< retVal<< endl;
      }
      else if (iPom == 0){
            cout<< "Veza zatvorena."<< endl;
      }
      else cout <<"Greska Recv(): "<< WSAGetLastError()<< endl;

}while (iPom > 0);
```

3.1.4. Zatvaranje priključnice

Kada smo gotovi s priključnicom, obavlja se funkcija *closesocket()* koja zatvara priključnicu.

```
int closesocket( _In_ SOCKET s);
```

s (in) – deskriptor za korištenu priključnicu

Kada je program gotov s korištenjem Windowsova socket DLL-a, poziva se *WSACleanup()* funkcija da se oslobode resursi. [15]

```
spoj =closesocket(ConnS);
WSACleanup();
```

3.2. Poslužiteljska priključnica

Kod poslužiteljske priključnice postoje isti koraci kao i kod klijentske priključnice. Osim što se poslužitelj ne spaja, ona osluškuje na zadanoj IP adresi i broju vrata.

Sljedeći koraci za stvaranje priključnice su:

- inicijalizacija priključnice
- vezivanje priključnice
- priključnica se postavlja u osluškujuć rad
- na zahtjev za spajanje, veza se prihvaća i klijentska priključnica se prihvaća
- primaju se podaci od klijenta tako dugo dok klijent nije gotov
- zatvaranje priključnice.

3.2.1. Inicijalizacija priključnice

Inicijalizacija poslužiteljske priključnice obavlja se na isti način kao i kod klijentske priključnice. Kod poslužiteljske priključnice potrebno je stvoriti dvije programske priključnice: jednu koja će osluškivati na IP adresi i broju vrata, i drugu, koja će se povezati kada dođe zahtjev za vezom i razmjenjivati informacije s klijentom.

To znači da postoji jedna mrežna priključnica koja prihvaća klijente i šalje ih programskoj priključnici. Programska priključnica te klijente obrađuje i oslobađa mrežnu priključnicu za nove konekcije.

Primjer realizacije:

```
SOCKET serv, client;
serv = socket(AF_INET, SOCK_STREAM, 0);

if (serv != INVALID_SOCKET)
{
      cout<< "Socket() uspjelo! "<< endl;
}
else
{
      cout << " Greska socket (): "<< WSAGetLastError() << endl;
}
```

3.2.2. Vezivanje priključnice

Vezivanje priključnice kod poslužitelja odnosi se na punjenje priključnice informacijama o IP adresi i broj vrata te priključnice, kako bi kasnije mogla slušati i čekati zahtjeve klijenata.

Struktura funkcije *bind()* je ista kao i *connect()* kod klijenta. [16]

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

s (in) – deskriptor za korištenu priključnicu

name (in) – pokazivač na *sockaddr* strukturu s kojom se povezuje

namelen (in) – duljina u bajtovima *sockaddr* strukture na koju pokazuje *name* parametar

Primjer realizacije:

```
sockaddr_in info;
info.sin_addr.s_addr = inet_addr("127.0.0.1");
info.sin_family = AF_INET;
info.sin_port = htons(25252);
int inflen = sizeof(info);
long spoj= 0;
spoj = bind(serv, (sockaddr*)&info, inflen);
if (error != SOCKET_ERROR)
{
      cout<< "Socket() Uspjelo! "<< endl;
}
else
{
      cout << " Greska socket (): "<< WSAGetLastError() << endl;
}
```

3.2.3. Osluškivanje na priključnici

Osluškivanje na priključnici moguće je izvesti pomoću funkcije *listen()*, koja se jednostavno sastoji od imena inicijalizirane priključnice i brojke koja označava maksimalan moguć broj istovremenih konekcija. [17]

```
int listen(_In_ SOCKET s,_In_ int backlog);
```

s (in) – deskriptor za korištenu priključnicu

backlog (in) – maksimalna veličina reda čekanja za uspostavu veze. Ako se postavi na SOMAXCONN, poslužitelj će odrediti koliki je moguć zaostatak (engl. *backlog)*. Zaostatak tada može biti u granicama od 200 do 65.535.

Primjer realizacije:

```
long res = 0;
res = listen(serv, SOMAXCONN);
if (res != SOCKET_ERROR){
      cout<< "LIsten() Uspjelo! "<< endl;
}
else {
      cout << "Greska socket (): "<< WSAGetLastError() << endl;
}
```

3.2.4. Prihvaćanje klijenta

Zahtjevi za spajanjem od klijenata se prihvaćaju pomoću *accept()* funkcije. Ona se sastoji od priključnice koja je osluškivala i detalja o adresi klijenta koji se želi povezati. [18]

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

s (in) – deskriptor za korištenu priključnicu

addr (out) – neobvezan pokazivač na međuspremnik koji prima podatke adrese o nadolazećoj konekciji

addrlen (in, out) – neobvezujuć pokazivač na veličinu strukture na koju pokazuje *addr(in)* parametar

Informacije o poslužitelju se preddefiniraju u strukturi koja sadrži informacije o IP adresi i broj vrata.

Primjer realizacije:

```
client = accept(serv, (sockaddr*)&clientinfo, &clientinfolen);
if (client !=SOCKET_ERROR)
{
```

```
      cout<<"Klijent   spojen:   "   <<inet_ntoa   (clientinfo.sin_addr)   <<":"   <<
      ntohs(clientinfo.sin_port)<< endl;
}
```

3.2.5. Primanje podataka

Podatke se prima na isti način kao i kod klijenta. Dakle, koristi se funkcija *recv()*, koja sadrži drugu priključnicu koja nije određena za osluškivanje, međuspremnik i veličinu međuspremnika.

Primjer realizacije:

```
retVal = recv(client,aceptBuf, sizeof(aceptBuf), 0 );
if (retVal > 0)
{
      cout << retVal << endl;
      retVal = 0;
      continue;
}
else if (retVal == SOCKET_ERROR)
{
      cout << " Greska Recv() : "<< WSAGetLastError() <<endl;
}
else break;
```

3.2.6. Zatvaranje priključnice

Poslužiteljsku priključnicu se zatvara na isti način kao i kod klijenta. Kada smo gotovi s primanjem i slanjem podataka, poziva se *closesocket()*. Kada smo gotovi s radom u Windows socked DLL, čistimo i oslobađamo korištene resurse s *WSACleanup()*.

Nedostaci ovakve vrste realizacije klijenta i poslužitelja su u tome što je komunikacija moguća samo s jednim klijentom istodobno. Izvršavanje glavnog dijela programa zaustavlja se sve dok nije gotova komunikacija između klijenta i poslužitelja. Ovakva programska realizacija priključnica naziva se blokirajući model.

3.3. Moguće izvedbe programske priključnice

U počecima, Berkeley priključnice su omogućavale samo blokiranje (engl. *blocking*). To je značilo da su sve funkcije priključnice radile sinkronizirano i nisu vraćale odgovor dok se operacija nije obavila. Takav sustav nije optimalan jer se ponekad zahtijeva da program obavlja korisnički unos dok čeka na mrežni odgovor. Rješenje tog problema dobije se koristeći neblokirajuće (engl. *non-blocking*) priključnice. [18]

Kod pozivanja funkcije koja koristi neblokirajuću priključnicu, funkcija će se vratiti u program što brže, iako operacija koju je pozivala nije izvršena. Kada je operacija izvršena,

obavijest će biti poslana programu, tako da program može dalje obavljati normalan rad bez da čeka na izvršenje operacije.

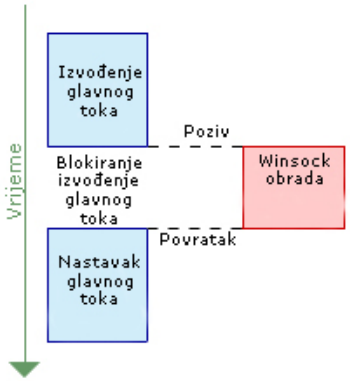
Načini implementacije priključnica u završnom radu izvedeni su kao blokirajuće priključnice u početku. Za prvih par realizacija sve se svodi na blokirajuće priključnice koje su radile komunikaciju „jedan na jedan“ i zaustavljale izvršavanje glavnog toka programa. To nije bilo zadovoljavajuće.

Da bi se dobila komunikacija s više klijenata istodobno na jednom poslužitelju, tražila su se nova rješenja. Rješenje je nađeno u stvaranju novih niti (engl. *thread*) za svakog pojedinog klijenta koji se poveže.

Time se dobilo to da glavni tok programa poslužitelja može obavljati druge stvari, dok nit obavlja komunikaciju s klijentom.

3.3.1. Blokirajuća priključnica

Blokirajuća priključnica (engl. *Blocking socket*) je najjednostavnija za korištenje. Ako se operacija obavljana na blokirajućoj priključnici ne može odmah izvršiti, priključnica počinje blokirati, zaustavlja rad programa dok se operacija ne izvrši (slika 3.2.). To znači da kada pozovemo Winsock funkciju za slanje i primanje, ono može potrajati prije nego se dobije odgovor. [19]

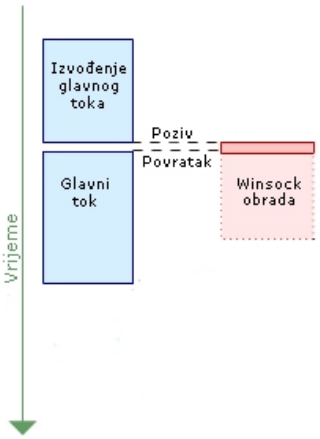


Slika 3.2. Prikaz rada blokirajuće priključnice [19]

Prema osnovnoj zadanoj postavci, priključnica je u blokirajućem radu i ponaša se prema prikazanom na slici 3.2.

3.3.2. Priključnica s nitima

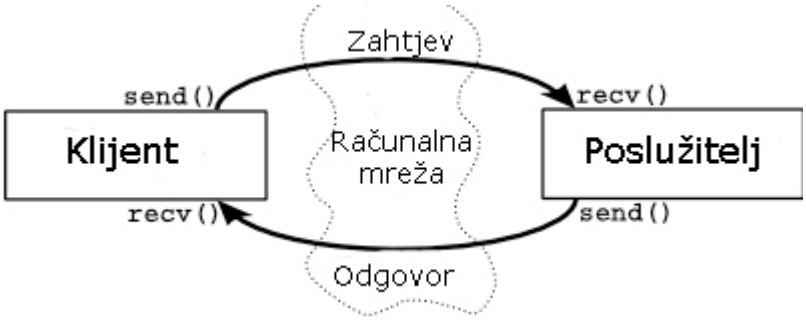
Priključnica koja za primanje i obradu klijenata koristi niti nastala je iz potrebe da se jednostavnim rješenjem omogući povezivanje više klijenata bez da je glavni dio programa blokiran (slika 3.3.).



Slika 3.3. Priključnica s nitima

4. Programska izvedba mrežnih priključnica

Izrađeni program za rad s priključnicama sastoji se od osnovnoga kostura koji je potreban za uspostavu jednostavne „klijent – poslužitelj“ komunikacije. Program radi na principu blokirajuće priključnice. To znači da će se program nastaviti tek po završetku izvršenosti zatražene funkcije. Pojednostavljeni prikaz rada programa klijent – poslužitelj vidljiv je iz slike 4.1.



Slika 4.1. Interakcija između klijenta i servera [20]

Klijent šalje poslužitelju zahtjev za pozivanje, server odgovara te, ako je zahtjev prihvaćen, klijent šalje podatke.

Razmatramo i objašnjavamo pojedine bitne dijelove programskoga koda, posebno za poslužiteljski i posebno za klijentski dio.

4.1. Blokirajuća internetska priključnica

Prva varijanta izvedbe svela se na blokirajuću priključnicu. U sebi sadrži osnovne funkcije potrebne za uspostavu veze između klijenta i poslužitelja, koje su opisane u 3. poglavlju.

Rad poslužitelja i klijenata svodi se na par bitnih točaka navedenih u nastavku.

Osnovne točke rada poslužitelja mogu se svesti na sljedeće:

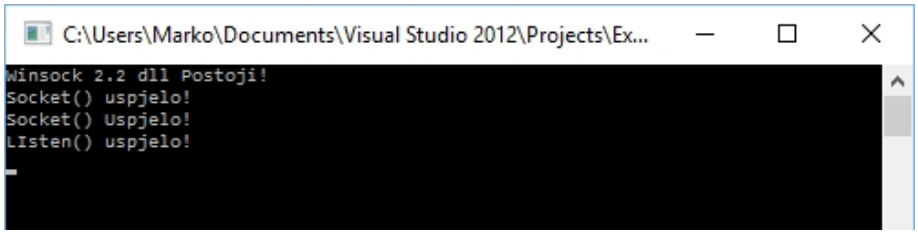
- poslužitelj stvara priključnicu
- priključnica se pridružuje IP adresi i broj vrata
- priključnica se postavlja u osluškujuć rad
- na zahtjev za spajanje, veza se prihvaća i klijentska priključnica se prihvaća
- primaju se podaci od klijenta tako dugo dok klijent nije gotov
- zatvaranje priključnice.

Programiranje klijentske strane priključnice je prilično slično poslužiteljskom dijelu programa. Moramo izvršiti sljedeće:

- stvaranje priključnice
- pridruživanje priključnice IP adresi i broju vrata
- funkcija za spajanje
- slanje podataka
- prekid spoja.

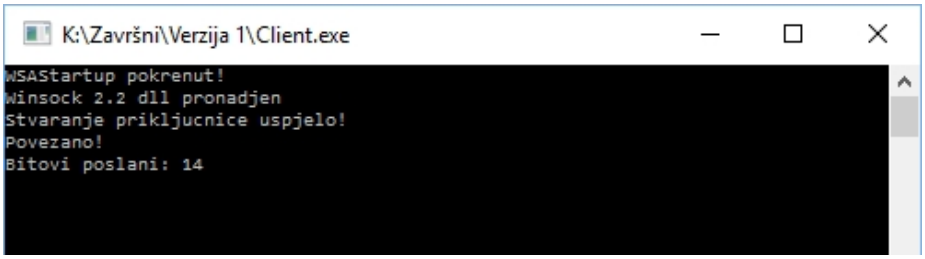
4.1.1. Klijent – poslužitelj: početna verzija

U prvoj verziji realizacije programa dobivamo poslužitelj koji osluškuje na lokalnoj povratnoj IP adresi 127.0.0.1.



Slika 4.2. Rad poslužitelja

Iz slike 4.2. vidimo primjer ispisa poslužiteljskog rada programa kada se klijent spoji na njega. Ispisuju se kontrolne točke definiranja internetske priključnice i adresa klijenta te njegov broj vrata. Posljednji red ispisuje broj bitova primljenih od klijenta.



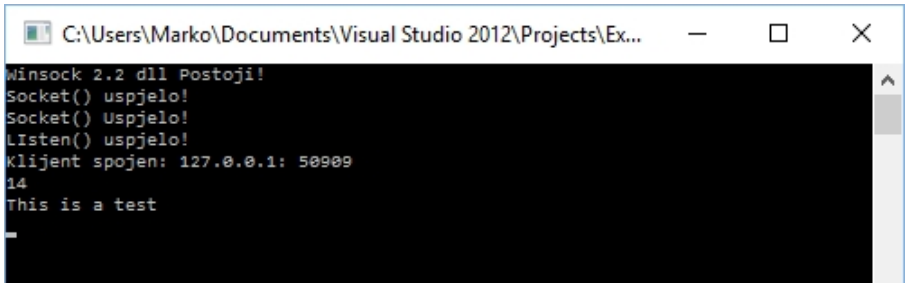
Slika 4.3. Ispis klijentske strane priključnice

Na klijentskoj strani programa također imamo ispis kontrole inicijalizacije priključnice, potvrdu da se klijent spojio na poslužitelja i potvrdu da su se poslali neki podaci.

Ovaj program izvršava konekciju na adresu koja je upisana u program i ne može se mijenjati. Također šalje neke podatke za koje ne znamo koji su (programer zna što se šalje, ali nije ispisano).

4.1.2. Klijent – poslužitelj: ispis poruke

Kao što je napomenuto, ne možemo očitati podatke koji se šalju. Malim izmjenama u programu, zaprimljene poruke možemo ispisati na poslužiteljskoj strani.



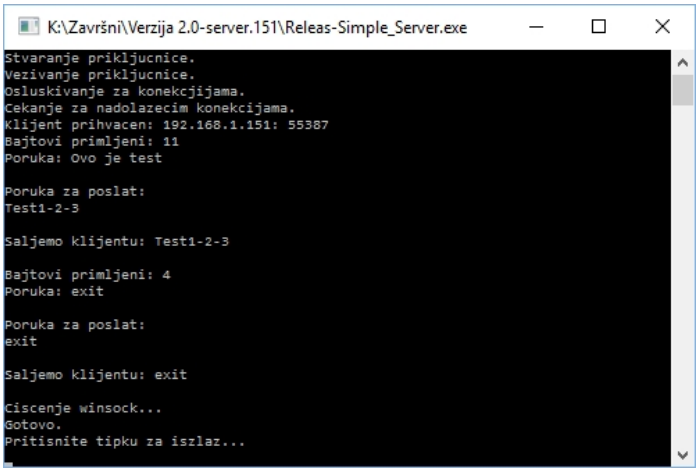
Slika 4.4. Ispis kod poslužitelja

Klijentska strana je ovdje ostala nepromijenjena, kod poslužitelja su izvršene izmjene. Osim zaprimljenih bitova, ispisuje se i poruka u nama čitljivom obliku (slika 4.4.).

Sljedeći korak u poboljšanju programa je omogućavanje unosa vlastite poruke za slanje te proširenje s jednog računala na dva različita i obavljanje testiranja.

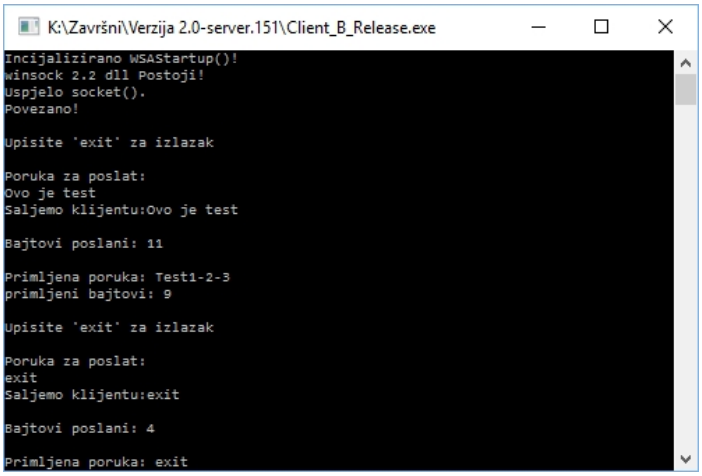
4.1.3. Klijent – poslužitelj: izmjena poruka

Izmjena poruka funkcionira tako da se ubacuje funkcija u program koja čeka unos poruke i šalje je prema poslužitelju, a s druge strane poslužitelj čeka na poruku i tada šalje povratnu poruku. Taj postupak se ponavlja dokle jedna strana ne označi kraj riječju *exit*. Uspostavili smo jednostavnu komunikaciju preko unosa tipkovnice (slika 4.5. i slika 4.6.).



Slika 4.5. Poslužitelj – razmjena poruka

Iz slike 4.5 je vidljivo da je poslužitelj primio klijenta na lokalnoj IP adresi 192.168.1.151. To je statička adresa računala na kojem se nalazio poslužitelj. Izmjenu poruka započinje klijent. Poruke se izmjenjuju nasumično (klijent – poslužitelj – klijent – poslužitelj).



Slika 4.6. Klijent – razmjena poruka

Na klijentskoj strani programa započinjemo sa slanjem poruke. Razmjena poruka se izvršava dok se ne unese *exit*; tada se izlazi iz petlje i program završava (slika 4.6.).

4.2. Priključnica s višestrukim klijentima

Prijašnji primjeri programa koji rade s priključnicama su jednostavni i dobri za svladanje osnovnih funkcija potrebnih za stvaranje i povezivanje priključnica.

Sljedeći korak je omogućavanje povezivanja višestrukih klijenata na poslužitelj. To se ostvarilo stvaranjem novih niti kod poslužitelja za svaku nadolazeću uspostavu veze klijenta.

Znači, klijenti se povezuju na poslužitelj, poslužitelj prihvaća klijenta i stvara novu programsku nit u kojoj se obavljaju komunikacija i obrada, čime se oslobađa glavni tok programa poslužitelja da nastavi s izvršavanjem. Efektivno time više ne spadamo u blokirajuće priključnice, budući da se program ne zaustavlja za obradu priključnice.

Detalji o programskom toku slijede u nastavku.

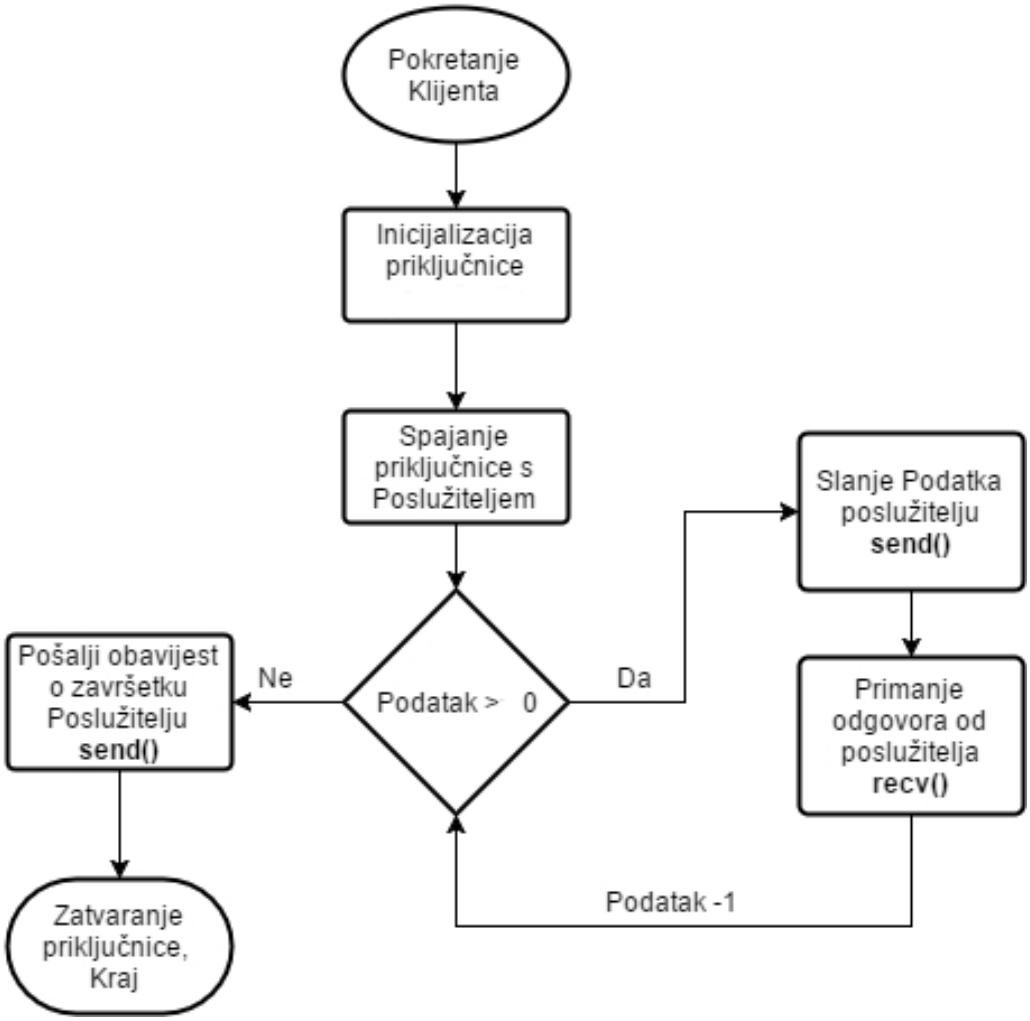
4.2.1. Klijent

Priključnice s višestrukim klijentima najviše se odnose na poslužiteljski dio koji će prihvaćati više klijenata istodobno.

Klijentski dio u sebi i dalje sadrži osnovne funkcije za stvaranje priključnica, izmjene su izvršene na podacima koji se šalju. Podatke ne možemo sami određivati, nego su stalni, radi potrebe testiranja poslužitelja.

Klijent započinje komunikaciju s poslužiteljem i u datoteku tipa *.csv* zapisuje vrijeme početka konekcije. Klijent šalje brojčani podatak iznosa dvadeset prema poslužitelju i zapisuje u *.csv* datoteku vrijeme slanja podatka. Poslužitelj radi obradu nad tim podatkom i šalje rezultat. Klijent prima taj podatak, zapisuje u *.csv* datoteku vrijeme primanja odgovora. Klijent umanjuje početno poslanu brojku za jedan i postupak slanja i primanja se ponavlja, sve dok se ne stigne do nule. Kada je klijent stigao do nule, šalje poslužitelju poruku *signal*, koja se sastoji od riječi *exit*, da je gotov sa slanjem. Poslužitelj i klijent prekidaju konekciju. Vrijeme završetka konekcije bilježi se u *.csv* datoteku.

Dijagram toka osnovnog koncepta programa vidljiv je na slici 4.7.



Slika 4.7. Dijagram toka klijenta

Podatak koji se šalje je brojčana vrijednost iznosa 20. Kada dođe odgovor od poslužitelja, podatak se umanji za 1 i šalje ponovo, sve dok ne stignemo do nule – tada se šalje potvrda o završetku.

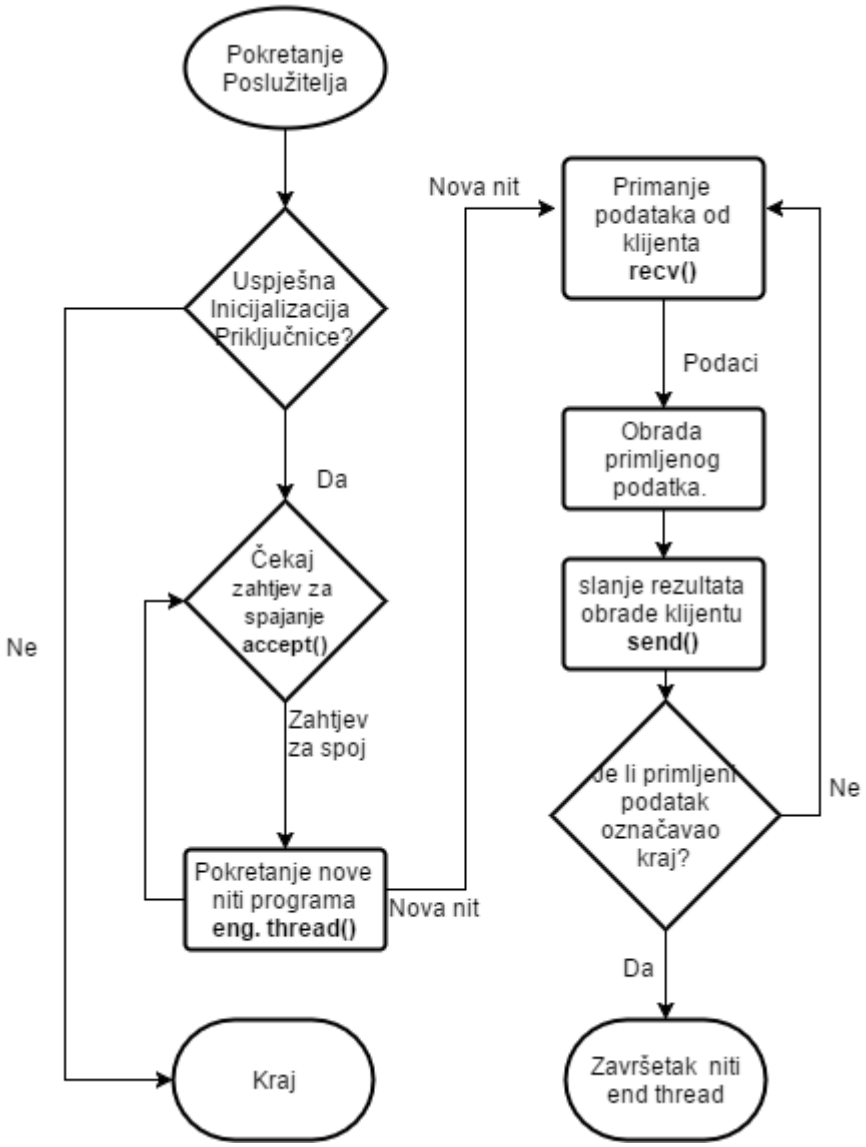
Vremenski zapisi u *.csv* datoteci rađeni su radi kasnije obrade rezultata mjerenja u poglavlju 5.

4.2.2. Poslužitelj

Poslužitelja koji može na sebe istodobno primiti više klijenata ostvarili smo stvaranjem niti (engl. *threads*) za pojedinoga klijenta koji se pokuša spojiti na poslužitelj.

Kada se stvori nova nit za svakoga klijenta, radimo bilježenje vremena početka komunikacije u *.csv* datoteku (tada započinje primanje podataka od klijenta). Kada se primi podatak, bilježi se vrijeme u *.csv* datoteku i tada se podatak obrađuje. Na njemu se radi izračun faktorijela. Rezultat faktorijela šalje se kao odgovor klijentu i u *.csv* datoteku bilježi se vrijeme slanja. Postupak se ponavlja tako dugo dok klijent ne pošalje riječ *exit* i tada se izlazi iz niti i bilježi vrijeme izlaska u *.csv* datoteku.

Dijagram toka osnovnog koncepta programa vidljiv je iz slike 4.8.



Slika 4.8. Dijagram toka poslužitelja

Prilikom pokušaja spajanja novoga klijenta na poslužitelj, poslužitelj stvara novu nit za toga klijenta. I svaki sljedeći klijent dobiva svoju nit. Obrada podataka se obavlja dok klijent ne pošalje oznaku za kraj i pritom se nit završava. Glavni program vrti beskonačnu petlju za prihvat klijenta.

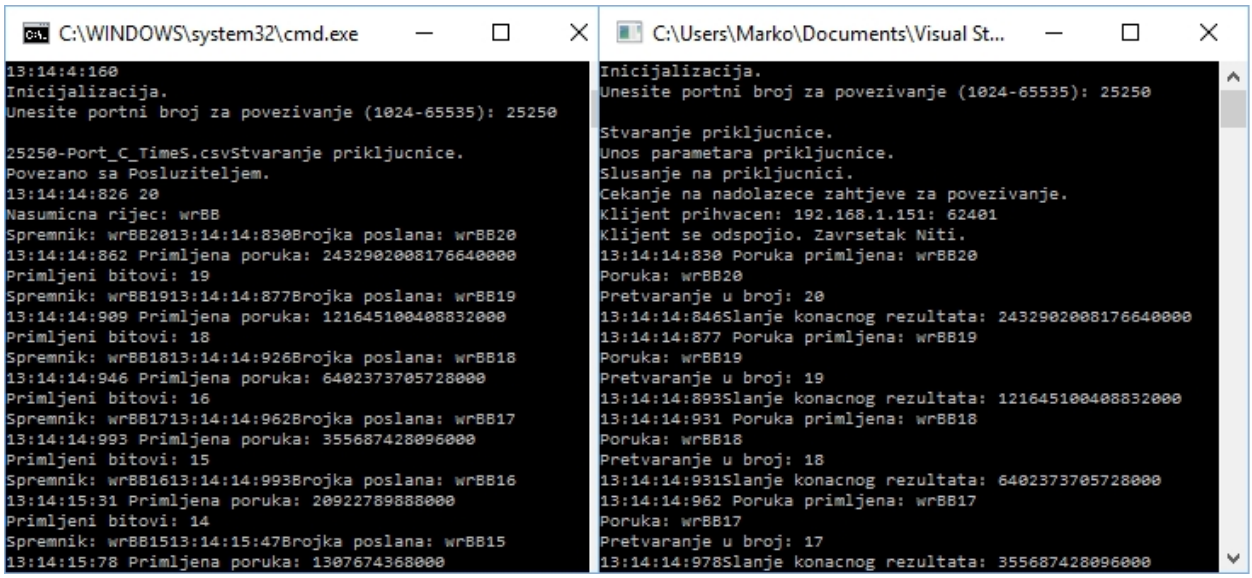
Radi omogućavanja mjerenja potrebnog vremena obrade podataka od klijenta, programirano je da se vrijeme primanja podataka i vrijeme slanje podataka zapisuje u datoteku na disku kako bi se kasnije podaci obradili i omogućili analizu rada i performansi.

Za dohvat informacije o vremenu koristila se funkcija *GetLocalTime()* koje je na svaki poziv vraćala preko parametara trenutno sistemsko vrijeme u milisekundama.

Definicija funkcije izgleda ovako:

```
void WINAPI GetLocalTime(_Out_ LPSYSTEMTIME lpSystemTime);
```

Gdje nam `LPSYSTEMTIME` pokazivač (engl. *pointer*) na strukturu koja će primiti trenutno sistemsko vrijeme. [21]



Slika 4.9. Klijent lijevo, poslužitelj desno – ispis na zaslon

Na slici 4.9. imamo prikaz ispisa na zaslon klijenta i poslužitelja na koji se spojio samo jedan klijent. Vrijeme se uzima lokalno vrijeme računala i mjeri u milisekundama.

Pokretanjem po nekoliko klijenata uzastopno i očitavanjem iz datoteke vremena, dobijemo neko srednje vrijeme obrade podataka po klijentu od jedne sekunde, ali bez nekog opterećenja na računalo i poslužitelj.

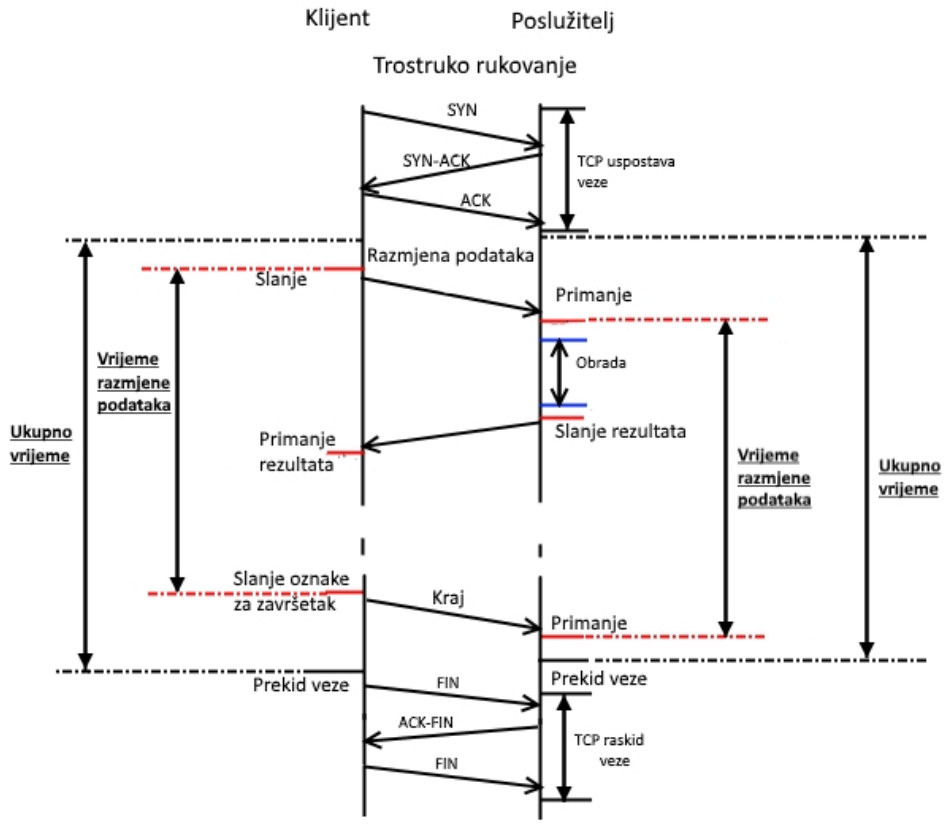
Detalji o načinu testiranja i mjerenja opterećenja poslužitelja slijede u 5. poglavlju.

5. Rezultati analize i testiranja programske izvedbe

Mjerenje se odnosi na zapis vremenskih trenutaka kada su poslužitelj i klijent primili ili poslali neki podatak. Podatak se sastoji od brojčane vrijednosti iznosa 20, koju klijent, kada uspostavi vezu, šalje poslužitelju. Poslužitelj tada radi faktorijelu toga broja i šalje rezultat natrag klijentu. Klijent umanjuje brojku 20 za 1 i šalje tu vrijednost poslužitelju. Poslužitelj ponovo izvrši faktorijelu i šalje rezultat. To se sve ponavlja u petlji dok se ne stigne do nule. Kada klijent stigne do nule, umjesto nule šalje signal, u našem slučaju riječ *exit*, da je slanje podataka gotovo. Poslužitelj prima signal i prekida konekciju. Klijent također prekida konekciju.

Također, istodobno se radi zapis početka i raskida TCP veze.

Vremenski zapisi se rade na klijentskoj i poslužiteljskoj strani zasebno.



Slika 5.1. Prikaz vremenskih mjerenja

Slika 5.1 prikazuje komunikaciju jednog poslužitelja i klijenta, a mjerimo i zapisujemo podatke o ukupnom trajanju konekcije i pojedino vrijeme obrade za svaki primljen podatak.

Mjerenjem se htjelo vidjeti postoji li vremenska razlika kod klijenta i servera za vrijeme obrade i izmjene podataka, i cjelokupno trajanje konekcije, zatim hoće li se vremena mijenjati i

kolika će biti promjena kada poslužitelj i klijent rade u neopterećenom stanju i kada se cijeli sustav optereti sa većim brojem zahtjeva.

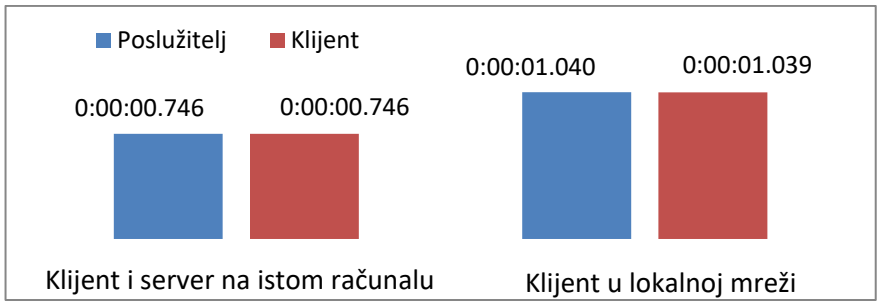
Računala koja su pokretala poslužitelje i klijente imaju navedene parametre konfiguracije:

Računalo 1 za poslužitelje: operativni sustav Windows 10 64-bit, procesor Intel core 2 quad Q8400 2.66 GHz, 4 GB DDR2 RAM memorije.

Računalo 2 koje pokreće klijente: operativni sustav Windows 10 64-bit, procesor Intel core i3-2120 3.3 GHz, 12 GB DDR3 RAM memorije.

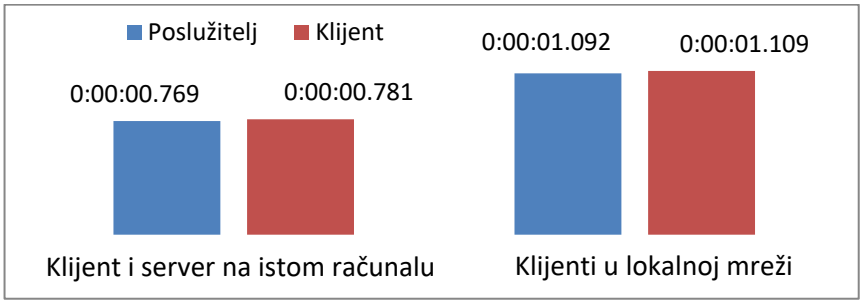
5.1. Jedan poslužitelj, jedan klijent

Prvo mjerenje je jednostavno, pokrenuli smo jednog poslužitelja i jednoga klijenta. Prvo se izvršilo nekoliko mjerenja dok su poslužitelj i klijent na istom računalu, a potom su se mjerenja izvršila s poslužiteljem na jednom računalu i klijentom na drugom računalu u lokalnoj mreži.



Slika 5.2. Srednje vrijeme potrebno za obradu jednoga klijenta

Iz slike 5.2. vidljivo je da je srednje vrijeme potrebno za izmjenu i obradu podataka koje klijent šalje poslužitelju na istom računalu nešto niže nego kod odvojenih poslužitelja i klijenta. To dodatno vrijeme dobije se kod putovanja kroz mrežu. Vremenske razlike između klijenta i poslužitelja nema. Pretpostavlja se da bi klijent trebao imao nešto duže vrijeme.



Slika 5.3. Srednje vrijeme trajanja konekcije

Također se može očekivati da će srednje vrijeme trajanja konekcije biti veće od vremena potrebnog za obradu podataka. Tako se iz slike 5.3. vidi da je vrijeme uistinu veće, ali ne mnogo.

5.2. Jedan poslužitelj, više klijenata

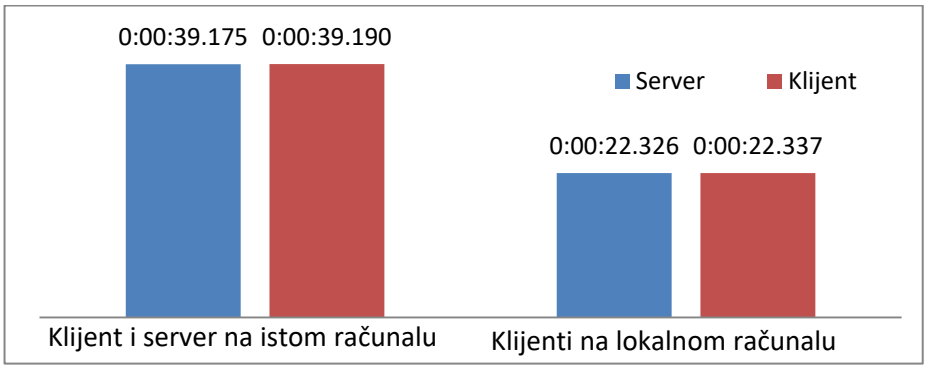
Kod sljedećeg mjerenja povećali smo broj klijenata koji se povezuju na jedan poslužitelj.

Pokretanje višestrukog broja klijenata odjednom izvršili smo pomoću programa koji će pokrenuti klijente. U program unosimo broj koliko puta želimo da nam pokrene funkciju *shellexecute()* koju otvara naš klijent.

Klijent je neovisan o tom programu i dalje radi za što je programiran.

Odabrali smo da će broj pokrenutih klijenta na jednog poslužitelja iznositi 100 klijenata.

Očekujemo znatno veće vrijeme obrade i trajanja konekcije.



Slika 5.4. Srednje vrijeme potrebno za obradu 100 klijenata

Iz slika 5.4. odmah se uočava da je vrijeme znatno manje kod računala u lokalnoj mreži. Opterećenje se podijelilo s jednog računala na dva. Vrijeme obrade na jednom računalu je i dalje poprilično nisko, manje od predviđene 1 minute.

Poslužiteljski program ima ugrađeni brojač koji broji ukupan broj niti koje se izvode istog trenutka. Ta vrijednost se ispisuje po završetku obrade klijenta u *.csv* datoteku.

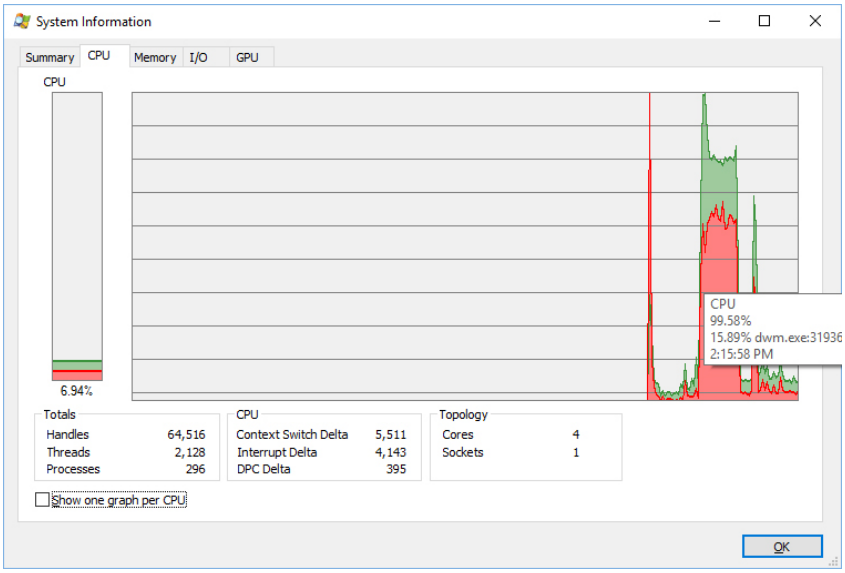
Srednji iznos broja aktivnih niti u istom trenutku za 100 klijenata po poslužitelju iznosi aktivnih 68 niti. To je srednja brojka dobivena iz oba načina povezivanja.

Dizanjem broja klijenata koji se na istom računalu povezuju na poslužitelj dolazimo do toga da poslužitelj odbacuje klijente ako se u istom trenutku na poslužitelj pokuša povezati više od 200 klijenata. To je posljedica *listen()* funkcije gdje je sa *SOMAXCONN* definirana maksimalna vrijednost reda čekanja. Budući da je vrijednost *SOMAXCONN* ovisna o platformi na kojoj se vrti poslužitelj, u našem slučaju maksimalni red čekanja je 200, što dovodi do odbacivanja ostalih

klijenta. Pokušaj zamjene parametra *SOMAXCONN* sa vlastito definiranom većom brojčanom vrijednosti nije dovelo ni do kakve promjene, pa je maksimalan red čekanja ostao 200.

Taj problem ne nastaje u slučaju kada su klijenti u lokalnoj mreži. U tom slučaju pokreće se i po 1000 klijenata na jedan poslužitelj i poslužitelj ih sve primi i obrađuje.

Snimljeno je opterećenje računala koje je pokretalo klijente i poslužitelj istodobno. U trenucima izvršavanja rada klijenta i poslužitelja vidljivo je znatno opterećenje na sustav (slika 5.5.).

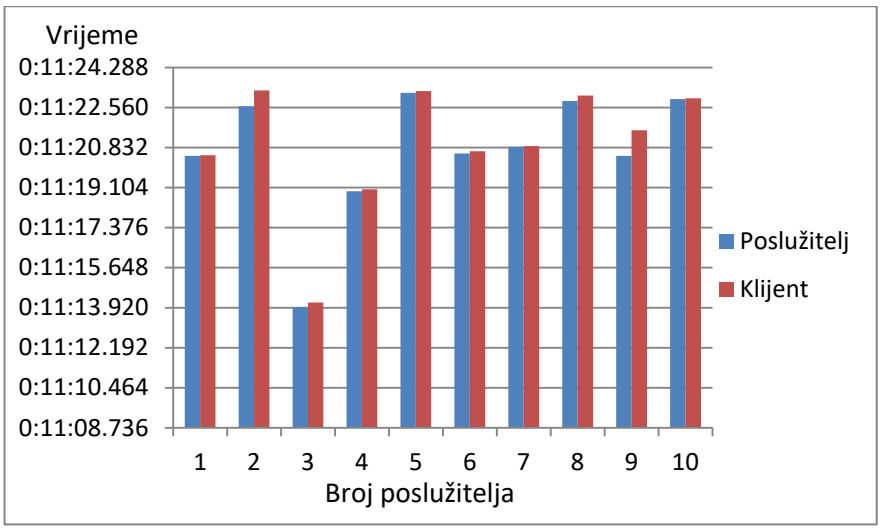


Slika 5.5. Opterećenje procesora za obradu 100 zahtjeva klijenata

5.3. Više poslužitelja s više klijenata

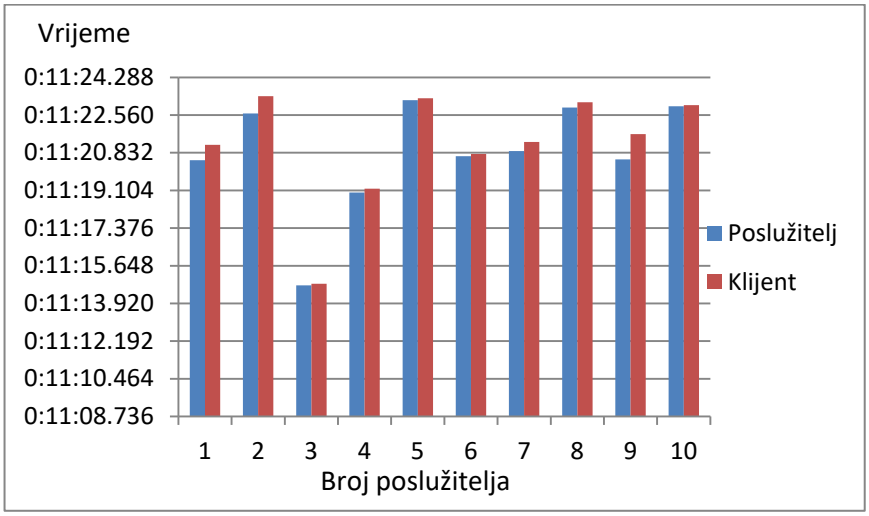
Sljedeće testiranje je zamišljeno kao pokušaj simulacije velikog opterećenja na poslužitelje, i to na računalo koje vrti više poslužitelja odjednom. U realnosti neki poslužitelj vrti više usluga na jednom sustavu. Tako je istodobno pokrenuto deset poslužitelja na istom računalu koji čekaju na klijente. Na istom tom računalu se pokreće sto klijenata po poslužitelju koji se istog trenutka povezuju na svoj poslužitelj i počinje komunikacija.

Vremenski zapisi prikazani u slici 5.6. prikazuju da je vrijeme obrade znatno poraslo. Vidljivo je da je vrijeme obrade za klijente znatno duže nego očekivano, budući da on počinje prvi mjeriti vrijeme obrade i čeka za obradu. Opterećenje na računalo u ovom testiranju je golemo. Naravno, to ovisi o sustavu na kojem se mjeri.



Slika 5.6. Vrijeme potrebno za obradu 100 klijenta po poslužitelju na istom računalu

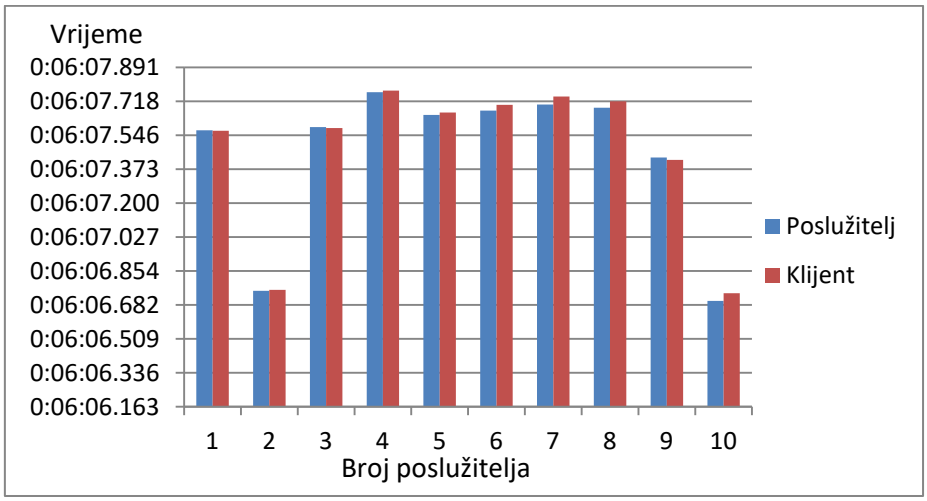
Naravno, interesantno je vidjeti i ima li velike razlike u trajanju ukupne TCP veze. Tako je iz slika 5.7. vidljivo da su vremena nešto veća od vremena mjerenja, kao što i treba biti.



Slika 5.7. Trajanje konekcije na istom računalu

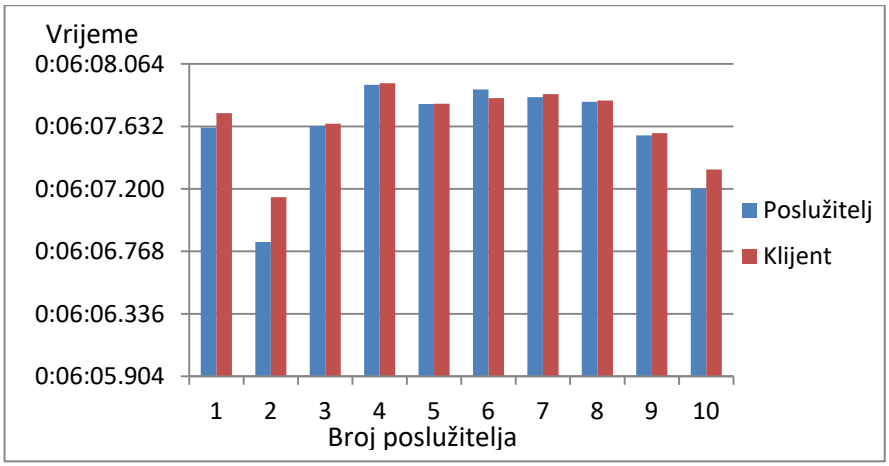
Za ovaj primjer testiranja, srednji iznos aktivnih niti u istom trenutku se malo smanjio. Tako imamo srednji iznos od 59 aktivnih niti u istom trenutku po poslužitelju. Broj aktivnih niti se smanjio budući da program koji pokreće klijente ne može istog trenutka pokrenuti sve klijente.

Vrijeme trajanja obrade na istom računalo je poprilično dugo. Kako bismo vidjeli koliko će biti vremensko trajanje ako se klijenti i poslužitelji odvoje na dva različita računala, izveli smo sljedeći test. Pokrenuli smo deset poslužitelja na jednom računalu. Oni čekaju zahtjeve za konekcijama od klijenta koji se pokreću na drugom računalu u lokalnoj mreži. Pokreće se sto klijenata po poslužitelju i obavlja se mjerenje. Dobiveni vremenski rezultati vidljivi u slici 5.8 prikazuju da se vrijeme obrade u nekim slučajevima upola smanjilo. U nekim slučajevima imamo abnormalnosti gdje poslužitelj ima dulje vrijeme obrade, što je posljedica opterećenosti sustava računala i kašnjenja s izvršavanjem zapisa vremena u datoteke.



Slika 5.8. Vrijeme potrebno za obradu 100 klijenata po poslužitelju u lokalnoj mreži

Iz ukupnog trajanja TCP konekcije između klijenta i poslužitelja vidljivo je da klijent dalje ima duže trajanje konekcije. U nekim iznimkama poslužitelj premašuje klijenta, što opet proizlazi iz opterećenosti i sporosti računala koje radi zapise u datoteku.



Slika 5.9. Vrijeme trajanja konekcije

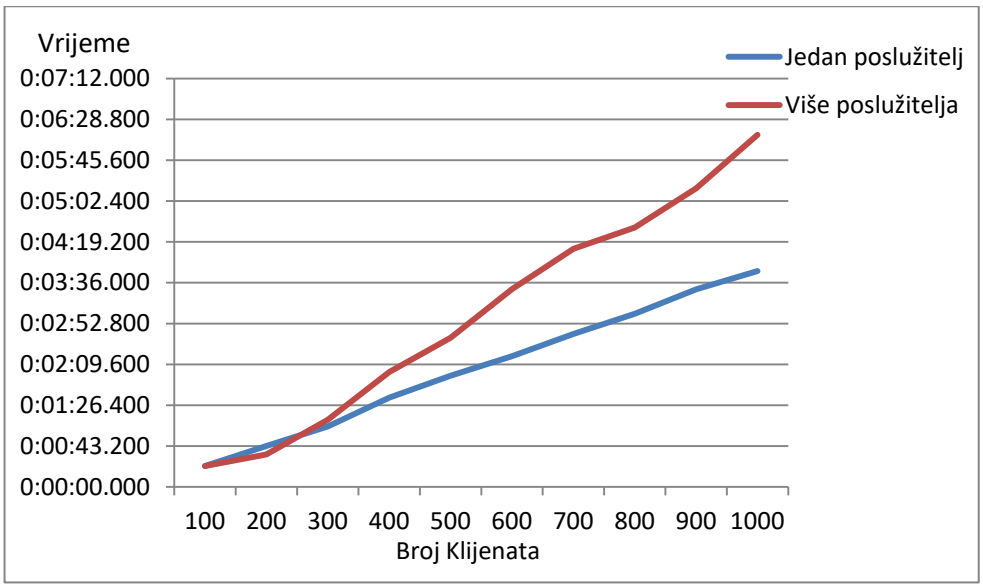
Broj aktivnih niti u istom trenutku za klijente u lokalnoj mreži približno je jednak rezultatu kada su klijenti na istom računalu. Dobivena je srednja vrijednost od 55 aktivnih niti u istom trenutku po poslužitelju.

Sljedeće testiranje zamišljeno je da se usporedi vrijeme trajanja obrade na poslužiteljima, klijenti su se povezivali sa drugog računala u lokalnoj mreži.
Prvo se mjerilo vrijeme na jednom poslužitelju (računalo 1) sa sto, dvjesto, tristo, četiristo, petsto, šesto, sedamsto, osamsto i tisuću klijenata pokrenutih sa računala 2. Poslužitelj je zasebno mjerio za svako povećanje klijenata. Drugo mjerenje se svelo na dva poslužitelja na svakog po sto klijenata, pa tri poslužitelja sa sto klijenata i tako sve do deset poslužitelja sa sto klijenata. Rezultati mjerenja su se uzeli i obradili u zajedničkoj tablici. Htjelo se usporedno vidjeti potrebno vrijeme za obradu pojedinog slučaja. Dobiveni rezultati vidljivi su u tablici 5.1 i slici 5.10.

Broj Poslužitelja	Broj klijenta po poslužitelju	Ukupno vrijeme trajanja veze	Broj niti
1	100	0:00:22.090	68
1	200	0:00:43.300	197
1	300	0:01:03.855	285
1	400	0:01:34.347	388
1	500	0:01:57.611	384
1	600	0:02:18.216	353
1	700	0:02:41.755	367
1	800	0:03:03.191	364
1	900	0:03:28.989	320
1	1000	0:03:48.351	397
2	100	0:00:34.225	95
3	100	0:01:10.992	94
4	100	0:02:01.430	96
5	100	0:02:37.684	97
6	100	0:03:29.137	90
7	100	0:04:11.859	77
8	100	0:04:34.467	66
9	100	0:05:15.813	58
10	100	0:06:12.452	57

Tablica 5.1 Vremenski rezultati i broj niti

Iz tablice 5.1 se može očitat i broj niti po poslužitelju za svako pojedino mjerenje.



Slika 5.10. Usporedni prikaz vremena obrade

Vidljivo je iz slike da je više poslužitelja imalo puno duže vrijeme obrade od jednog poslužitelja za ukupno isti broj klijenata. Računalo je moralo dijeliti resurse sa svakim pojedinim poslužiteljem, što kod pojedinačnog poslužitelja znači da ima na raspolaganju sve resurse.

5.4. Analiza posluživanja 200 klijenata

Interesira nas optimalna kombinacija klijenta/poslužitelja na jednom i dva računala za obradu 200 klijenta. Tako se izvelo nekoliko kombinacija mjerenja.

5.4.1. Klijenti i poslužitelj na računalu 1

Računalo 1 pokretalo je poslužitelje i klijente. Prvo 1 poslužitelj sa 200 klijenta. Tada su se klijenti rasporedili preko dva poslužitelja pa tri poslužitelja i završno četiri poslužitelja. Dobiveni vremenski rezultati vidljivi su iz tablice 5.2.

Računalo 1			
Broj poslužitelja	Broj klijenata po poslužitelju	Ukupno vrijeme obrade	Broj niti
1	200	0:01:01.143	45
2	100	0:01:17.808	62
3	67	0:01:35.432	64
4	50	0:01:23.948	42

Tablica 5.2 Potrebno vrijeme obrade 200 klijenta na računalu 1

5.4.2. Klijenti i poslužitelji na računalu 2

Ovdje se izvodi kombinacija poslužitelja i klijenta kao i u poglavlju 5.4.1 razlika je što se mjerenje vršilo na drugome računalu. Dobiveni vremenski rezultati vidljivi su iz tablice 5.3.

Računalo 2			
Broj poslužitelja	Broj klijenata po poslužitelju	Ukupno vrijeme obrade	Broj niti
1	200	0:00:43.788	106
2	100	0:01:05.441	91
3	67	0:00:58.998	66
4	50	0:00:56.816	50

Tablica 5.3 Potrebno vrijeme obrade 200 klijenta na računalu 2

Vrijeme obrade na računalu 2 je niže od vremena obrade na računalu 1.

5.4.3. Poslužitelji i klijenti u mreži

Ovo mjerenje odvijalo se između računala 1 i računala 2 u računalnoj mreži.

U prvome primjeru računalo 1 pokretalo je kombinacije poslužitelja, a računalo 2 je pokretalo sve kombinacije klijenta. Dobiveni vremenski rezultati obrade vidljivi su iz tablice 5.4.

računalo 1 - poslužitelji, računalo 2 - klijenti			
Broj poslužitelja	**Broj klijenata po poslužitelju**	**Ukupno vrijeme obrade**	**Broj niti**
1	200	0:01:17.277	116
2	100	0:01:01.036	99
3	67	0:00:51.087	65
4	50	0:00:46.603	50

Tablica 5.4 Računalo 1 pokreće poslužitelje, računalo 2 pokreće klijente

Pošto računalo 2 ima resursa da pokrene dovoljno brzo klijente i veći broj klijenta, raspodjelom klijenta na više poslužitelja dobilo se kraće vrijeme obrade.

Postupak se izveo i u drugom smjeru. Računalo 2 je pokretalo poslužitelje a računalo 1 je pokretalo klijente. Dobiveni vremenski rezultati za obradu 200 poslužitelja su prikazani u tablici 5.5, iz koje je vidljivo da raspodjelom klijenta na više poslužitelja u ovoj kombinaciji uzrokuje povećanje vremena obrade. Pošto je računalo 1 starije arhitekture i slabije od računala 2 pokretanjem višestrukih klijenta uzrokuje usporavanje.

računalo 2 - poslužitelji, računalo 1 - klijenti			
Broj poslužitelja	**Broj klijenata po poslužitelju**	**Ukupno vrijeme obrade**	**Broj niti**
1	200	0:01:04.157	198
2	100	0:01:11.679	99
3	67	0:01:24.238	66
4	50	0:01:17.468	50

Tablica 5.5 Računalo 2 pokreće poslužitelje, računalo 1 pokreće klijente.

5.4.4. Kombinacija poslužitelja/ klijenta između 2 računala

Izvela se prva kombinacija gdje je računalo 1 pokrenulo 2 poslužitelja. Klijenti su se povezivali sa računala 1 i računala 2, svaki po 100 klijenta na jedan poslužitelj. Vrijeme obrade na pojedinom poslužitelju vidljivo je iz tablice 5.6.

	Broj poslužitelja	Broj klijenata po poslužitelju	Ukupno vrijeme obrade	Broj niti
Klijenti računala 1	1	100	0:00:44.677	66
Klijenti računala 2	1	100	0:00:44.168	43

Tablica 5.6 Poslužitelji na računalu 1, 100 klijenta na računalu 1 i računalu 2

Zati su se poslužitelji pokrenuli na računalu 2. Klijenti su se povezivali sa računala 1 i računala 2, svaki po 100 klijenata na jedan poslužitelj. Vremenski rezultat obrade vidljiv je iz tablice 5.7. Ova kombinacija pokazuje najbolje vrijeme obrade za ukupno 200 klijenta.

	Broj poslužitelja	Broj klijenata po poslužitelju	Ukupno vrijeme obrade	Broj niti
Klijenti računala 1	1	100	0:00:26.382	99
Klijenti računala 2	1	100	0:00:26.941	99

Tablica 5.7 Poslužitelji na računalu 2, 100 klijenta na računalu 1 i računalu 2

5.5. Analiza skalabilnosti razvijenih programa

Interesira nas vremenska obrada poslužitelja za 1000 klijenta. Koje kombinacije sa dva poslužitelja i dva računala bi bile optimalno rješenje. Prilikom izvršavanja testa došlo je do problema skalabilnosti poslužitelja.

Izvedeno je mjerenje gdje računalo 1 vrti dva poslužitelja i na pojedinog poslužitelja se povezuje 500 klijenta, isto mjerenje se izvelo i na računalu 2 (tablica 5.8). Znači na istome računalu su klijent i poslužitelj, razlika je u arhitekturi računala.

	Broj Poslužitelja	Broj klijenta po poslužitelju	Ukupno vrijeme trajanja veze	Broj niti
Računalo 1	2	500	0:07:12.101	258
Računalo 2	2	500	0:05:17.214	487

Tablica 5.8 Vremenski rezultati za obradu 500 klijenta po poslužitelju

Iz tablice 5.2 je vidljivo da računalo 1 puno sporije obrađuje klijente, jer nema dovoljno resursa za pokretanje i obradu ukupno 1000 klijenta na istom sistemu gdje su i poslužitelji.

Izvršeno je mjerenje gdje računalo 1 vrti dva poslužitelja, a 500 klijenata se pokreće s računala 2. Zatim se na računalu 2 pokrenu poslužitelji, a računalo 1 je pokretalo 500 klijenata po poslužitelju. Vrijeme obrade vidljivo je iz tablice 5.9. Znači računalo 1 vrti po dva poslužitelja, računalo 2 pokreće 500 klijenta na pojedinog poslužitelja. Po završetku mjerenja na računalu 1, izvelo se mjerenje u kojem računalo 2 vrti dva poslužitelja, a računalo 1 pokreće klijente koji se povezuju na računalo 2.

	Broj Poslužitelja	Broj klijenta po poslužitelju	Ukupno vrijeme trajanja veze	Broj niti
Računalo 1	2	500	0:04:08.405	237
Računalo 2	2	500	0:08:28.620	478

Tablica 5.9 Vremenski rezultati za obradu 500 klijenta po poslužitelju sa lokalne mreže

Iz tablice 5.9 je vidljivo da je vrijeme obrade na računalu 1 puno brže ako se klijenti pokreću sa računala 2. Računalo 2 iako ima više resursa, ipak ima dulje vrijeme obrade 1000 klijenta, jer računalo 1 nije u mogućnosti da klijenti koje vrti imaju odgovore prema računalu 2 istog trenutka. Sve je to zbog načina na koje računala sažimaju i dodjeljuju slobodnu memoriju i procesorsko vrijeme pojedinim procesima kada dođe do opterećenja sistema.

Sljedeći postupak izveden je na način da računalo 1 pokrene jednog poslužitelja i računalo 2 jednog poslužitelja. Klijenti su se pokretali sa računala 1 i povezivali na računalo 2, sa računala 2 na računalo 1 u istome trenutku, s 500 klijenta po poslužitelju. Dobiveni vremenski rezultati vidljivi su iz tablice 5.10, računalo 2 je brže obradilo klijente od računala 1.

Prilikom mjerenja tablice 5.10 moglo se zamijetiti da je računalo 1 gotovo zaustavilo obradu poslužitelja. Klijenti su se pokretali i povezivali sa računalom 2. Dok računalo 1 gotovo da nije obrađivalo klijente od računala 2. Tek kada su se klijenti sa računala 1 povezali i neki obradili na računalu 2, tada je poslužitelj računala 1 krenuo punom brzinom obrađivat klijente sa računala 2.

	Broj Poslužitelja	Broj klijenta po poslužitelju	Ukupno vrijeme trajanja veze	Broj niti
Računalo 1	1	500	0:03:51.906	40
Računalo 2	1	500	0:02:58.409	449

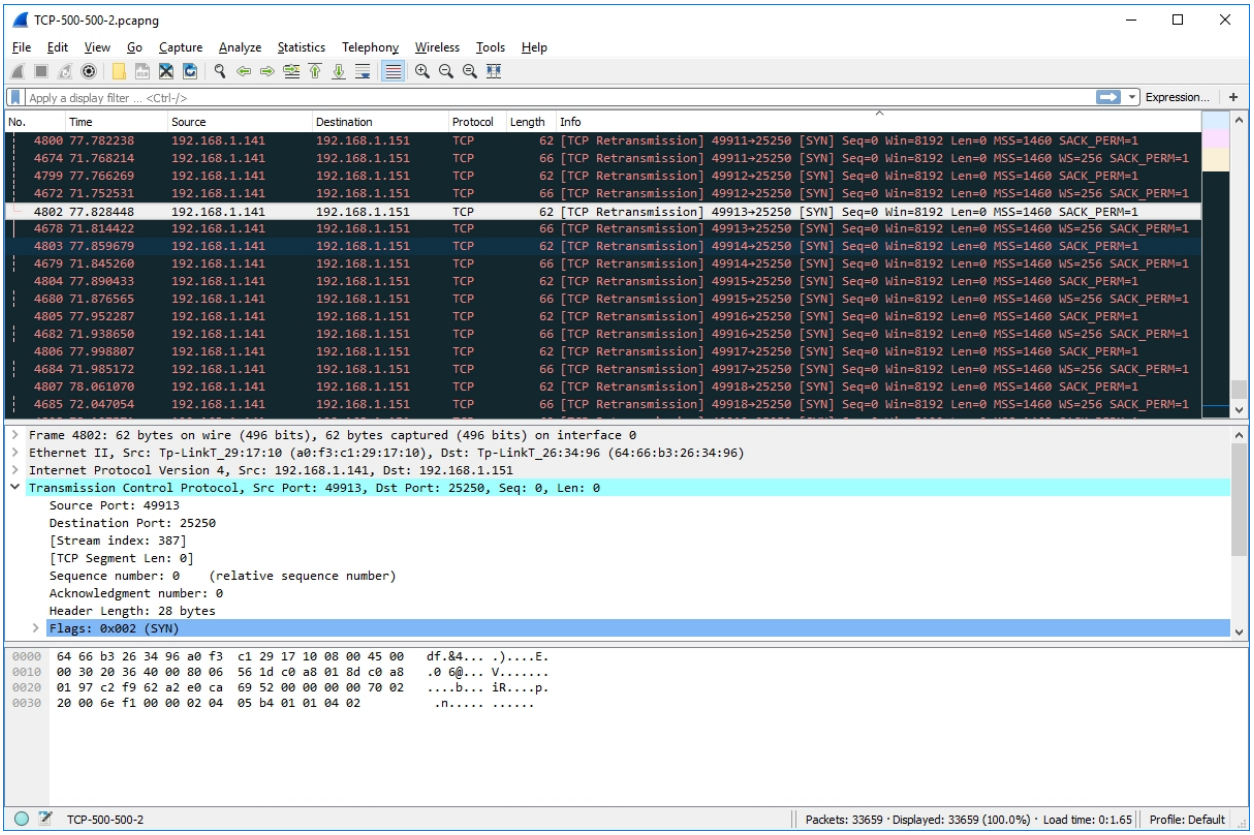
Tablica 5.10 Vrijeme obrade pokrenutih 500 klijenta u istom trenutku na pojedini poslužitelj

Računalo 2 ima kraće vrijeme obrade klijenta od računala 1, to je zbog toga jer je arhitektura novija i snažnija od računala 1.

Završni postupak za otkrivanje najboljeg načina za obradu 1000 klijenata izveden je na način da računalo 1 vrti dva poslužitelja. Klijenti se povezuju sa računala 1 na jedan poslužitelj i sa računala 2 na drugi poslužitelj, oba računala pokreću 500 klijenata. Čitanjem zapisa obrade dobivenih od poslužitelja vidljivo je da nije obrađeno 500 klijenta po poslužitelju. Obrađeno je u prosjeku 210 klijenta po pojedinom poslužitelju.

Test se ponovio nekoliko puta, sa dodatnim čekanjem *sleep()* kod pokretanja klijenta, rezultat je uvijek isti. Klijenti nisu u mogućnosti uspostaviti vezu sa poslužiteljem. Klijenti izbacuju grešku kod povezivanja priključnice sa poslužiteljem. Isti problem sa *SOMAXCONN* već spomenuto na stranici 31. Poslužitelji su zatrpani traženim upitima za povezivanjem i odbacuju klijente.

Praćenjem mrežne komunikacije preko programa Wireshark vidimo TCP pakete poslane od klijenata prema poslužitelju. Klijenti koji nisu uspostavili vezu šalju zahtjeve *SYN* za uspostavom veze. Poslužitelji ne odgovaraju, klijenti urade ponovno slanje zahtjeva, slika 5.11.



Slika 5.11. Wireshark analiza klijentskih TCP paketa

Detaljnim pregledom i praćenjem pojedinih klijentskih TCP paketa koji su radili retransmisiju skoro niti jedan nije tada primljena od poslužitelja. Ponegdje se klijent koji je uradio retransmisiju povezao sa poslužiteljem.

Izvršeno je mjerenje gdje se dizao broj klijenata sa računala 2 iznad tisuću. Tako se došlo do granice oko 2000 pokrenutih klijenta, iznad toga računalo nije bilo u mogućnosti pokrenuti klijente. Operativni sistem je izbacivao poruke da klijentski program nije moguće pokrenuti jer nije pronađen. Rezultati tog dizanja klijenta iznad 1000 pokrenutih vidljivi su iz tablice 5.11.

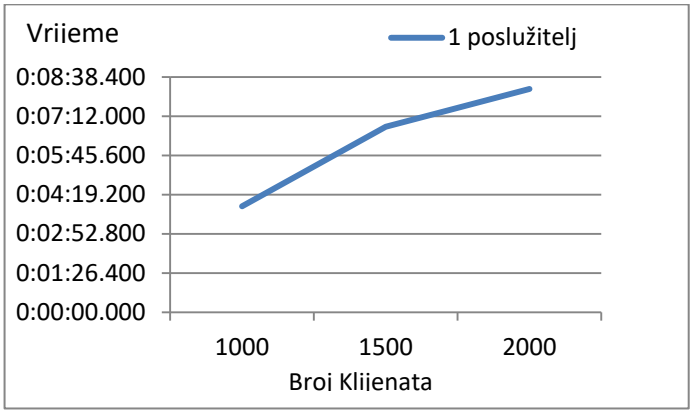
Broj Poslužitelja	Broj klijenta po poslužitelju	Ukupno vrijeme trajanja veze	Broj niti
1	1000	0:03:53.401	441
1	1500	0:06:48.759	412
1	2000	0:08:12.112	419
1	***3000**	*0:09:23.713*	*408*

Tablica 5.11. Rezultati mjerenja vremena klijenata iznad 1000

Pokušalo se mjeriti iznad 2000 klijenta na poslužitelju, ali ti dobiveni vremenski rezultati nisu precizni, jer kao što je navedeno iznad 2000 klijenta operativni sistem je javljao pogreške.

* Pokušalo se povezati i mjeriti 3000 klijenta, ali kasnijom provjerom datoteka vremenski zapisa vidljivo je da se obradilo 2000 klijenata. 3000 klijenata se nije obradilo. Iz vremenskih zapisa datoteka također se vidi da je poslužitelj obradio u nekim slučajevima malo manje od 2000 klijenata, a u nekima više od 2000 klijenta.

Iz slike 5.12 vidljivo je da nakon 1500 klijenta vrijeme počinje malo stagnirati, što je posljedica smanjenog broja klijenata zbog pogreške pokretanja.



Slika 5.12. Prikaz vremena trajanja konekcije za preko 1000 klijenta

Zaključak analize

Svi sustavi, posebice oni koji vrše obradu podataka i funkcioniraju kao poslužitelji ograničeni su dostupnim slobodnim resursima. Broj procesa se povećava, resursi računala se dijele na broj procesa koji ih zahtijevaju. Što imamo više procesa, imamo i više zahtjeva za resursima. Tako će računalo dodijeliti svakom procesu dio vremena korištenja resursa. Time dobivamo sporije vrijeme obrade za više procesa, ali se na kraju procesi obave. Zbog toga su dobivene linearnosti u grafičkim prikazima rezultata. Kada bi došao neki treći proces usred obrade naših postojećih klijenta/poslužitelja, naše vrijeme obrade otišlo bi van linearnosti, naglo bi poraslo, do neke točke. Nakon naglog rasta dalje bi se vrijeme dizalo linearno do završetka svih zatraženih procesa.

5.6. Veličina TCP paketa

Izradili smo još jedno završno mjerenje. Zanimalo nas je postoji li razlika u veličini TCP paketa između Windows priključnice i Java priključnice iz ovog primjera.

Iskoristio se gotov primjer [22] jednostavnog programa u Javi, klijent – poslužitelj, gdje se klijent povezuje na poslužitelja, unosi se tekst i klijent šalje uneseni tekst na poslužitelj. Poslužitelj jednostavno uzme i vrati taj isti tekst natrag klijentu.

- Programski kod klijenta izrađenog u Javi

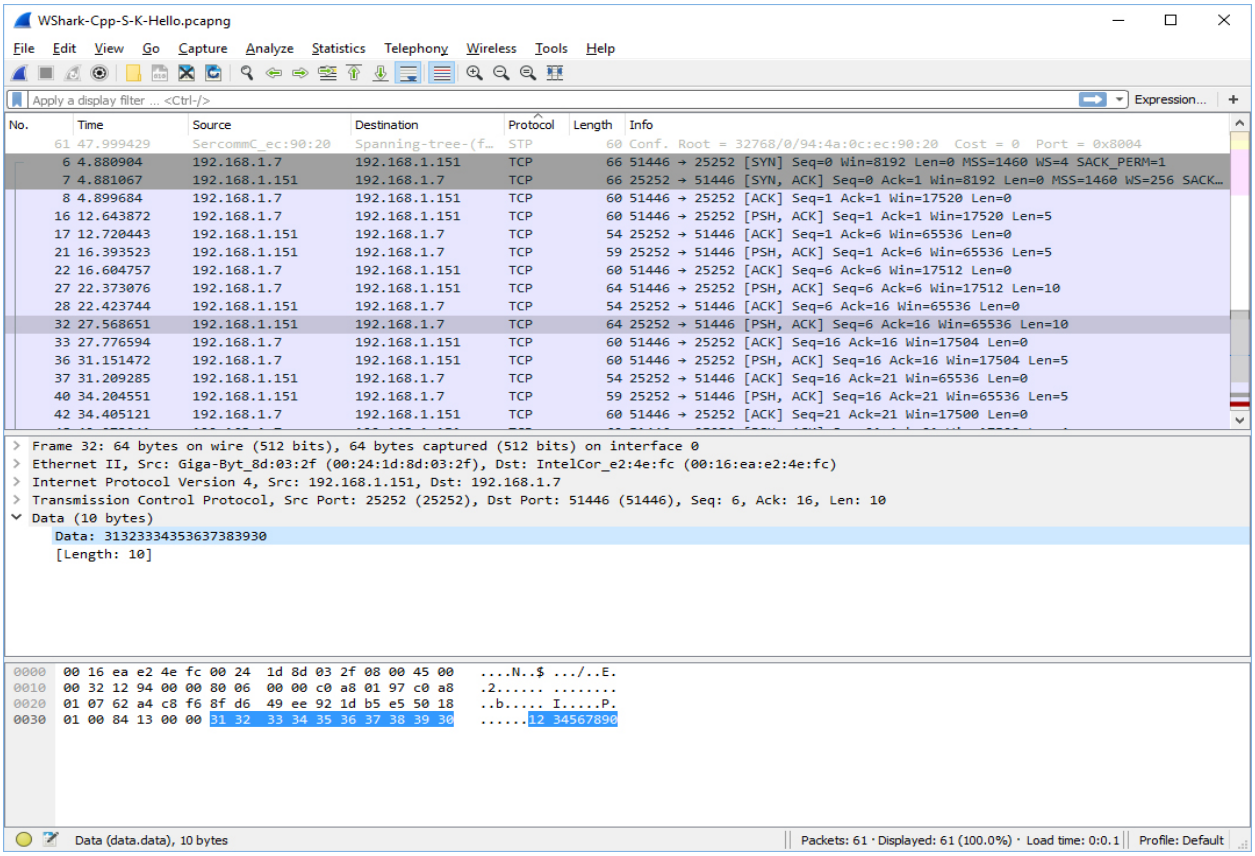
```
public class Klijent {
public static void main(String[] args)
       {
    Socket clientSocket = null;
    DataInputStream is = null;
    PrintStream os = null;
    DataInputStream inputLine = null;
    try {
       clientSocket = new Socket("192.168.1.151", 2222);
              os = new PrintStream(clientSocket.getOutputStream());
              is = new DataInputStream(clientSocket.getInputStream());
              inputLine=new DataInputStream(new bufferedInputStream(System.in));
    } catch (UnknownHostException e) {
      System.err.println("Don't know about host");
    } catch (IOException e) {
              System.err.println("Couldn't get I/O for the connection to host");
    }
if (clientSocket != null && os != null && is != null) {
try {
              System.out.println("The client started. Type any text. To quit it type
              'Ok'.");
              String responseLine;
              os.println(inputLine.readLine());
              while ((responseLine = is.readLine()) != null) {
          System.out.println(responseLine);
          if (responseLine.indexOf("Ok") != -1) {
            break;
          }
          os.println(inputLine.readLine());
        }
        os.close();
        is.close();
        clientSocket.close();
      } catch (UnknownHostException e) {
              System.err.println("Trying to connect to unknown host: " + e);
      } catch (IOException e) {
        System.err.println("IOException:  " + e);
      }
    }
    }
}
```

Programski kôd poslužitelja izrađenog u Javi

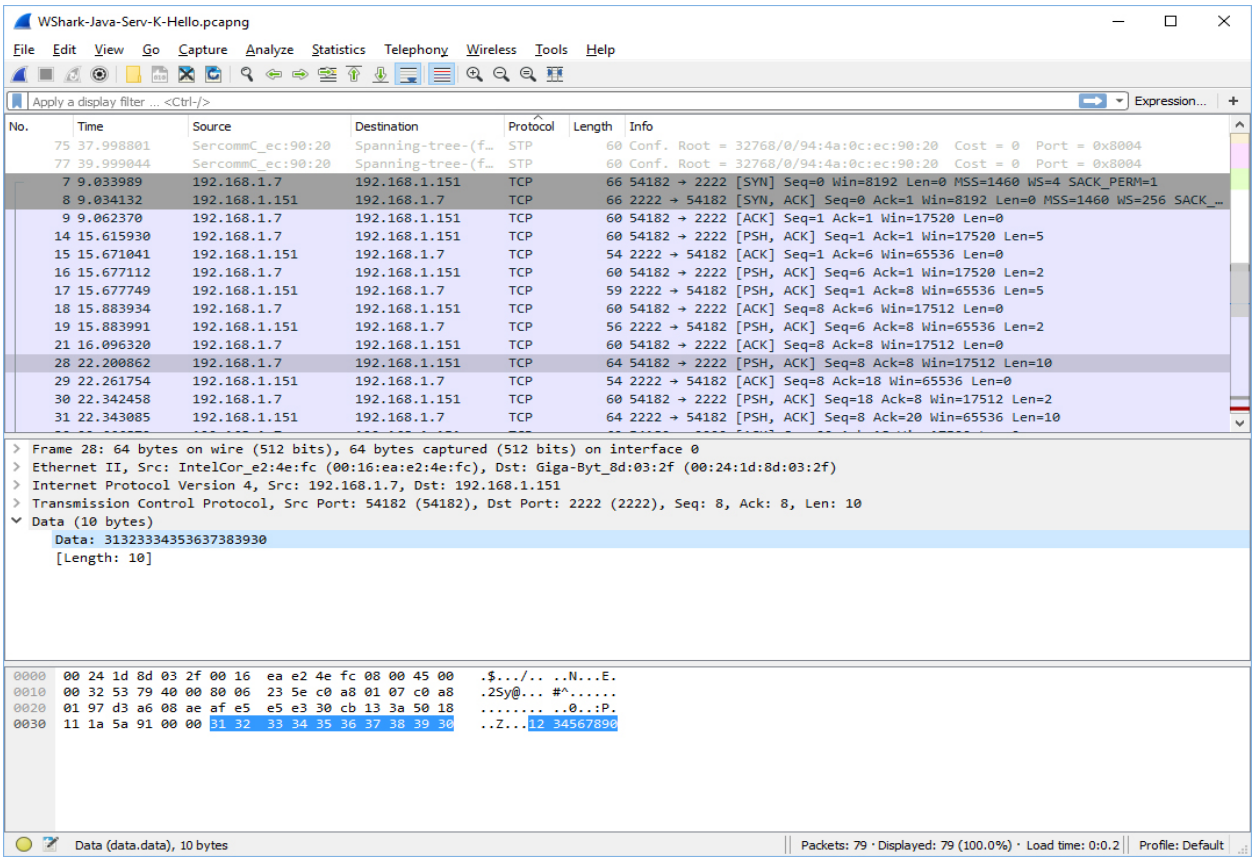
```
public class Server {
public static void main(String[] args)
    {
            ServerSocket echoServer = null;
            String line;
            DataInputStream is;
            PrintStream os;
            Socket clientSocket = null;
            try {
                  echoServer = new ServerSocket(2222);
            } catch (IOException e) {
                  System.out.println(e);
            }
     System.out.println("The server started. To stop it press <CTRL><C>.");
    try {
            clientSocket = echoServer.accept();
     is = new DataInputStream(clientSocket.getInputStream());
            os = new PrintStream(clientSocket.getOutputStream());
            while (true) {
                  line = is.readLine();
                  System.out.println("From client:");
                  os.println( line);
            }
     } catch (IOException e) {
    System.out.println(e);
    }
    }
}
```

Za bilježenje TCP paketa koristili smo Wireshark program. Poruke korištene u slanju su u oba slučaja jednake. Nakon snimljenih TCP paketa za obje verzije, očitavanjem vidimo da su podaci u TCP paketima jednake veličine (slika 5.13. i slika 5.14.).

Winsock i Java nude čistu implementaciju mrežnih priključnica bez posebnih dodavanja pomoćnih knjižnica. Java ima tu prednost što je jednostavnija za rukovanje i isti program može se pokrenuti na drugim operacijskim sustavima, a Winsock je ograničen na Windows sustave.



Slika 5.13. C++ TCP paketi



Slika 5.14. Java TCP paketi

6. Zaključak

Rad je nastao istraživanjem načina na koji pojedini programi i procesi uspostavljaju komuniciraju. To nas je dovelo do pojma mrežnih/Internet priključnica. Daljnjim proučavanjem i do teme završnog rada i praktične izvedbe jednostavnog modela poslužitelja i klijenata.

Implementirane programske mrežne priključnice su jednostavne za primjenu posebice blokirajuće priključnice. Njihovo svojstvo blokiranja izvršavanja ostatka programa dok se ne odradi mrežna razmjena informacija je korisno kod linearnog programiranja. Winsock operacije se izvode točno onda kada se i pozivaju. Korištenje osnovnih opcija priključnica je vrlo lako i jednostavno za svladavanje i pruža mogućnost učenja, širenja i nadogradnje programa što se više služimo priključnicama. Za veću primjenu priključnica potrebno je realizirati jedan od mnogo načina neblokirajućih programskih priključnica. U radu se takvo rješenje našlo u nitima koje rukuju s klijentima.

Obzirom da se nije previše optimiziralo za performanse klijenta i poslužitelja rezultatima se pokazalo da je takvo rješenje s nitima prihvatljivo do neke granice. Računala na kojima se mjerilo u nekim slučajevima nisu bila preopterećena, ali kod slučajeva s tisuću klijenata, računalo koje je vrtjelo klijente, imalo je ogromna opterećenja. Tako da treba voditi računa o brojci uspostavljanja zahtjeva i karakteristikama računala s kojih se pokreću programi.

Također, Winsock programiranje mrežnih priključnica je ograničeno na Windows platforme, dok primjere Java rješenja možemo pokretati na različitim platformama.

Ovaj rad može biti dobar temelj za daljne razvijanje i analiziranje programskih mrežnih priključnica.

U Varaždinu, _____

Student/ica:

_____

7. Literatura

[1] https://tools.ietf.org/html/rfc1122, rujan 2016.

[2] https://en.wikipedia.org/wiki/Internet_protocol_suite, rujan 2016.

[3] A. Bažant, V Sinković i dr.: Osnove arhitekture mreža, Zagreb, 2004.

[4] http://www.linux-tutorial.info/modules.php?name=MContent&obj=page&pageid=142, rujan 2016.

[5] https://en.wikipedia.org/wiki/IP_address, rujan 2016.

[6] http://mreze.layer-x.com/s030101-0.html, rujan 2016.

[7] https://en.wikipedia.org/wiki/Internet_protocol_suite, rujan 2016.

[8] http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html, rujan 2016.

[9] https://en.wikipedia.org/wiki/Port_(computer_networking), rujan 2016.

[10] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, rujan 2016.

[11] https://msdn.microsoft.com/en-us/library/windows/desktop/ms740506(v=vs.85).aspx, rujan 2016.

[12] https://msdn.microsoft.com/en-us/library/windows/desktop/ms737625(v=vs.85).aspx, rujan 2016.

[13] https://msdn.microsoft.com/en-us/library/windows/desktop/ms740149(v=vs.85).aspx, rujan 2016.

[14] https://msdn.microsoft.com/en-us/library/windows/desktop/ms740121(v=vs.85).aspx, rujan 2016.

[15] https://msdn.microsoft.com/en-us/library/windows/desktop/ms737582(v=vs.85).aspx, rujan 2016.

[16] https://msdn.microsoft.com/en-us/library/windows/desktop/ms737550(v=vs.85).aspx, rujan 2016.

[17] https://msdn.microsoft.com/en-us/library/windows/desktop/ms739168(v=vs.85).aspx, rujan 2016.

[18] https://msdn.microsoft.com/en-us/library/windows/desktop/ms737526(v=vs.85).aspx, rujan 2016.

[19] http://www.madwizard.org/programming/tutorials/netcpp/5, rujan 2016.

[20] http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html#indexId434909-59, rujan 2016.

[21] https://msdn.microsoft.com/en-us/library/windows/desktop/ms724338(v=vs.85).aspx, rujan 2016.

[22] http://makemobiapps.blogspot.hr/p/multiple-client-server-chat-programming.html, rujan 2016.

[23] J. Šribar, B. Motik: Demistificirani C++, Zagreb, 2014.

Prilozi

- **Poslužiteljski dio programskog koda**

```
// Posluzitelj programski kod
//Knjiznice u upotrebi
#include <string>
#include <iostream>
#include <ctime>
#include <sstream>
#include <fstream>
#include <winsock2.h>
#include <Windows.h>

#pragma comment (lib,"ws2_32.lib")
//Konstante
const int PORT =25252;
const int REQ_WERS = 2;
const int BUFFER = 1024;
//glabalne varijable
int Nulaz1= 0;
int Nulaz2=0;
//Deklariranje funkcije za rad s vremenom
std::string timeSL();
//Deklariranje funkcije za obradu niti
DWORD WINAPI ProcessC (LPVOID);

//Glavna funkcija programa
int main()
{
//Inicijalizacija Winsock.h knjiznice
WSADATA wsaData;
if (WSAStartup(MAKEWORD(REQ_WERS,2), &wsaData)==0)
{
	//Provjera dal postoji trazena verzija Winsock.h
	if (LOBYTE(wsaData.wVersion) >= 2 || HIBYTE(wsaData.wVersion) >= 2)
	{
		std::cout << "Inicijalizacija."<< std::endl;
		//inicijalizacija temeljnog broja vrata (eng.port)
		int portN = 25252;
		// pokretanje petlje za unos broja vratiju od strane korisnika
		do
		{

			std::cout<<"Unesite portni broj za povezivanje (1024-
			65535): ";
			std::cin>>portN;
			std::cout<<std::endl;
			if ((portN<= 1024) || (portN >65535))
			{
				std::cout<<"Krivi portni broj. Raspon (1024-
				65535)"<<std::endl;
			}
		}while((portN<= 1024) || (portN >65535));
		//Postavljanje informacija o prikljucnici u strukturu
		sockaddr_in info;
		//IP adresa na kojoj posluzitelj radi
		info.sin_addr.s_addr = ADDR_ANY;
		//specifikacije nacina prijenosa podataka, STREAM(TCP),DGRAM(UDP),
		RAW,
		info.sin_family = AF_INET;
		//Broj vratiju na koje posluzitelj radi
		info.sin_port = htons(portN);
		//Stvaranje objekta prikljucnice
		SOCKET lSocket = INVALID_SOCKET, client = INVALID_SOCKET;
		//Stvaranje prikljucnice
		std::cout<<"Stvaranje prikljucnice."<<std::endl;
		//Unos parametra (protokoli) prikljucnice i petlja za provjeru
		ispravnosti
```

```
		if ((lSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
		INVALID_SOCKET)
		{
			std::cout<< "Greska kod stvaranja prikljucnice:
			"<<WSAGetLastError<< std::endl;
		}
		std::cout<<"Unos parametara prikljucnice."<<std::endl;
		//Vezivanje prikljucnice sa IP adresom i brojem vratiju
		if (bind(lSocket, reinterpret_cast<sockaddr*>(&info),
		sizeof(info))!=0)
		{
			std::cout<< "Greska kod parametara."<<WSAGetLastError<<
			std::endl;
		}
		std::cout<<"Slusanje na prikljucnici."<<std::endl;
		//Postavljanje prikljucnice u osluskivanje za konekcijama
		if (listen(lSocket, SOMAXCONN) != 0)
		{
			std::cout<< "Greska kod slusanja: "<<WSAGetLastError<<
			std::endl;
		}
		std::cout<<"Cekanje na nadolazece zahtjeve za povezivanje." <<
		std::endl;
		// stvaranje strukture za pohranu informacija o povezanom klijentu
		sockaddr_in clientSockAddr;
		int	clientSockSize = sizeof(clientSockAddr);

		DWORD dwThreadID;
		//Pocetak beskonačne petlje za prihvacanje klijenta
		do
		{
			//Prihvacanje nadolazecih klijenata
			client = accept (lSocket, (SOCKADDR*) &clientSockAddr,
			&clientSockSize);
			if(client == INVALID_SOCKET)
			{
				std::cout<<"Greška kod spajanja:
				"<<WSAGetLastError<<std::endl;
				break;
			}
			//Brojac primljenih klijenta koji ulaze u nit
			Nulaz1 ++;
			//Ispis informacija o adresi i portnom broju klijenata
			std::cout<<"Klijent prihvacen: "
			<<inet_ntoa(clientSockAddr.sin_addr)<< ": "
				<< ntohs(clientSockAddr.sin_port)<< std::endl;
			//Stvaranje petlje za priljenog klijenta, poziv funkcije za
			obradu klijenta
			CreateThread (NULL, 0, ProcessC, (LPVOID)client, 0,
			&dwThreadID);
			std::cout<<"Glavni dio programa, beskonacna petlja!"
			<<std::endl;


		}while(TRUE);
	}
	else
	{
		std::cout << "Trazena verzija nije nadjena!"<< std::endl;
	}
	std::cout << "Ciscenje winsocka.... "<<std::endl;
	// Ciscenje koristenih resursa
	if (WSACleanup()!=0)
	{
		std::cout << "Greska kod ciscenja"<< std::endl;
	}
	std::cout << "Gotovo."<<std::endl;
}
else
{
```

```
	std::cout << "Pokretanje neuspjelo"<<std::endl;
}
// Ciscenje koristenih resursa
WSACleanup();
std::cout<<"Pritisnite neku tipku za Izlaz"<<std::endl;
getchar();
return 0;
}
//
DWORD WINAPI ProcessC (LPVOID lpParameter)
{
//Stvaranje objekta za rad sa zapisom u datoteke
std::ofstream oFile;
//stvaranje i otvaranje datoteke formata .csv
oFile.open("Total_Time.csv", std::fstream::app);
//zapis u datoteku
oFile<<timeSL()<<",Pocetak Veze!"<<std::endl;
oFile.close();
//stvaranje objekta prikljucnice za rad sa primljenim klijetom
SOCKET AcceptS = (SOCKET) lpParameter;
//incijalizacija medjuspremnika
int bytesSent;
int bytesRecv = SOCKET_ERROR;
char sendBuff [BUFFER];
char recvBufF [BUFFER];
DWORD retVal;
//incijalizacija varijable za usporedbu
std::string exit = "exit";
//---------------------------------
//Inicijalizacja varijabli za racunanje faktorijela
unsigned int recNum = 0;
unsigned long long Factor = 1;
//incijalizacija varijable za pohranjivanje primljene vrijednosti iz
medjuspremnika
std::stringstream strVal;
//Petlja za primanje/ slanje podataka
do
{
//Postavljanje varijabli u pocetne vrijednosti
Factor = 1;
recNum = NULL;
strVal.str(std::string());
strVal.clear();
//Funkcija za primanje podataka na prikljucnici
retVal = recv (AcceptS,recvBufF, sizeof(recvBufF), 0);
//Zapis vremenskih trenutaka i primljneih podataka u .csv datoteku
oFile.open("TimeStamp.csv", std::fstream::app);
oFile<<timeSL()<<","<<std::string(recvBufF, 4)<< ",-Primljeno!"<<std::endl;
oFile.close();
//petlje za provjeru i obradu primljenog podatka
if (retVal == 0 || std::string(recvBufF,4) =="exit" )
{
	break;
}
else if (retVal == SOCKET_ERROR)
{
	std::cout<<"Greska kod primanja"<< WSAGetLastError<<std::endl;
}
else
{
	// Obrada primljenog podatka
	std::cout <<timeSL()<<" Poruka primljena: "<< std::string(recvBufF,
retVal) << std::endl;
	//Izvlacenje brojčanje vrijednosti iz poruke
	std::string sB (recvBufF, retVal);
	std::cout<<"Poruka: "<< sB<< std::endl;
	size_t last_index = sB.find_last_not_of("0123456789");
	std::string result = sB.substr(last_index + 1);
	std::cout<<"Pretvaranje u broj: "<< result<< std::endl;
	strVal<<result;
```

```
		strVal>>recNum;
		strVal.str(std::string());
		strVal.clear();
		// Pokretanje petlje za racunanje faktorijela primljene vrijednosti
		if (std::string (recvBufF,4) != "exit" )
		{
			for (int i = 1; i <= recNum; i++)
			{
				Factor = Factor * i;
			}
			//punjenje medjuspremnika za slanje konacnog rezultata faktorijela
			sprintf_s(sendBuff, "%llu", Factor);
			std::cout<<timeSL()<<"Slanje konacnog rezultata: "<< sendBuff<<
			std::endl;
			//zapis u .csv datoteku trenutka slanja
			oFile.open("TimeStamp.csv", std::fstream::app);
			oFile<<timeSL()<<","<<std::string(recvBufF, 4)<<
			",Slanje:,"<<sendBuff<<std::endl;
			oFile.close();
			//Funkcija za slanje
			if(send(AcceptS,sendBuff, strlen(sendBuff), 0) == SOCKET_ERROR)
			{
			std::cout<<"Greska kod slanja: "<< WSAGetLastError<<std::endl;
			}
		}
}
//Uvjet petlje, petlja se ponavlja tako dugo dok klijent ne posalje rjec "exit"
}while (recvBufF != "exit" );
//provjera naveceg broja niti pokrenutih istog trenutka
if (Nulaz1 >Nulaz2)
{
		Nulaz2= Nulaz1;
}
//zapis vremnskog trenutka izlaska iz niti
std::cout << "Klijent obradjen, Izlazak iz Niti."<< retVal << std::endl;
oFile.open("Total_Time.csv", std::fstream::app);
oFile<<timeSL()<<",Kraj Veze!,"<<Nulaz2<<std::endl;
oFile.close();

//Ciscenje resusrsa prikljucnice
closesocket (AcceptS);
Nulaz1 --;
return 0;
}

//Funkcija za dohvacanje trenutnog lokalnog vremena
std::string timeSL()
{
std::string sTime;
SYSTEMTIME local;
GetLocalTime(&local);
std::ostringstream ssTime;
ssTime <<local.wHour<<":"<< local.wMinute<<":"<< local.wSecond<<":"<<
local.wMilliseconds;
sTime = ssTime.str();
return sTime;
}
```

- **Klijentski dio programskog koda**

```
//Programski kod Klijenta
//Knjiznice u upotrebi
#include <iostream>
#include <ctime>
#include <string>
#include <fstream>
#include <sstream>
#include <random>
#include <WinSock2.h>

#pragma comment (lib, "Ws2_32.lib")
using namespace std;
//Globalne konstante
const int REQ_WERS = 2;
#define DEFAULT_BUFLEN 1024
static const string alpha =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
//Deklaracija funkcije za rada s vremenom
string timeSL();

int main(){
//incijalizacija funkcije za nasumicnost
srand(time(0));
//ispis lokalnog trenutnog vremena
cout<<timeSL()<<endl;
//deklariranje vraijabli
ofstream oFile;
WORD VersionReq;
WSADATA wsaData;
int err;
int portN = 25252;
string portNum;
string portN2;
string fName = "-Port_C_TimeS.csv";
string Fname_T= "-Port_C_TotalT.csv";

//Inicijalizacija winsock.h knjiznice
if (WSAStartup(MAKEWORD(REQ_WERS,2), &wsaData)==0)
{
	if (LOBYTE(wsaData.wVersion) >= 2 || HIBYTE(wsaData.wVersion) >= 2)
	{
		std::cout << "Inicijalizacija."<< std::endl;
		//Socket Function

		//deklariranje variajble za vrstu IP adrese- IPv4, IPv6,IPX/SPX
		etc...
		int socAddrFamily;
		//deklariranje variajble za vrstu prikljucnice, STREAM(TCP),
		DGRAM(UDP), RAW,
		int socType;
		//deklariranje variajble za vrstu protokola, ICMP, IGMP, TCP, UDP,
		ICMPV6
		int socProtocol;
		//incijalizacija,deklariranih varijabli
		socAddrFamily = AF_UNSPEC;
		socType = SOCK_STREAM;
		socProtocol = IPPROTO_TCP;
		int iAdres= 0;
		//Petlja za unos i odabir borja vratiju (engl. port) broja
		do
		{
			std::cout<<"Unesite portni broj za povezivanje (1024-
			65535): ";
			std::cin>>portN;
			std::cout<<std::endl;
			if ((portN<= 1024) || (portN >65535))
			{
```

```
                std::cout<<"Krivi portni broj. Raspon (1024-
                65535)"<<std::endl;
        }
}while((portN<= 1024) || (portN >65535));


// konverzija broja vratiju u string podatak radi koristenja u
nazivu datoteke.
ostringstream numString;
numString.clear();
numString<<portN;
portNum = numString.str();
portN2 = numString.str();
portN2.append(Fname_T);
portNum.append(fName);
cout<<portNum;
//Punjenje strukture iServ sa podacima o posluzitelju
struct sockaddr_in iServ;
iServ.sin_addr.s_addr = inet_addr("192.168.1.151");
iServ.sin_family = AF_INET;
iServ.sin_port = htons(portN);
//incijalizacija objekta prikljucnice
SOCKET ConnS;
//deklariranje objekta sa parametrima pkriljucnice
ConnS = socket(socAddrFamily, socType, socProtocol);
if (ConnS == INVALID_SOCKET)
{
        cout<< "Greska kod stvaranja prikljucnice: "<<
        WSAGetLastError()<< endl;
        WSACleanup();
        return 0;

}
else  cout<< "Stvaranje prikljucnice."<< endl;
long spoj = 0;
//pozivanje funkcije connect() za spajanje sa posluziteljem
spoj = connect (ConnS, (SOCKADDR*) & iServ, sizeof (iServ));
//provjera ispravnosti funkcije connect()
if  (spoj == SOCKET_ERROR)
{
        cout<< "Greska kod povezivanja prikljucnice: "<<
        WSAGetLastError()<< endl;
        spoj = closesocket(ConnS);
        WSACleanup();
        return 0;
}
else
{
        //Zapis u .csv datoteku pocetnog vremena uspostave veze
        oFile.open(portN2.c_str(), fstream::app);
        oFile<<timeSL()<<", Pocetak TCP Veze!"<<endl;
        oFile.close();
        cout<< "Povezano sa Posluziteljem."<<endl;

        //deklaracija i incijalizacija medjuspremnika za primanje i
        slanje podataka
        int recvbuflen = DEFAULT_BUFLEN;
        char recvbuf[DEFAULT_BUFLEN];
        //dekalracija varijabli
        char *sendbuf;
        DWORD retVal = 0;
        int iPom;
        //dekalracija i incijalizacije varijable koja ozancuje kraj
        slanja
        string exit= "exit";
        // deklaracija i inicijalizacija podatka (broj 20)
        unsigned int rNum=20;
        //deklaracija pomocnih varijabli
        char in1[8];
        char in2[4];
```

```
char in3[4];
//stvaranje nasumicne rjeci
string sRand1;
for (int i= 0; i<4; i++)
{
	sRand1 += alpha[rand() % 52];
	in2[i] =sRand1[i];
}
cout<<"Nasumicna rijec: "<< sRand1<<endl;
//Petlja za smanjivanje brojek koja se salje prema
posluzitelju
for(int i = rNum; i >= 0; i --)
{
	//obrada podatka za slanje prema posluzitelju
	strncpy_s(in1,in2,sizeof (in2));
	sprintf_s(in3, "%d",i);
	strncat_s(in1,in3, sizeof(in1) );
	cout<<"Spremnik: "<< in1;
	if ( i > 0)
	{
		//Zapis u .csv datoteku vremenskog trenutka za
		slanje podatka
		oFile.open(portNum.c_str(), fstream::app);
		oFile<<timeSL()<<","<<string(in1,
		4)<<",Slanje!"<<endl;
		oFile.close();
		//sčanje podatka prema posluzitelju
		iPom =send(ConnS, in1,(int) strlen(in1), 0);
		//provjera eventualnih gresaka
		if (iPom == SOCKET_ERROR)
		{
			cout<< "Greska kod slanja: "<<
			WSAGetLastError()<< endl;
			closesocket(ConnS);
		}
	}
	// slanje kljucne rijeci za oznacavanje kraja slanja
	podataka
	else if (i == 0)
	{
		//punjenje medjuspremnika kljucnom rjeci
		sendbuf = "exit";
		//Zapis u .csv datoteku vremenskog trenutka za
		slanje podatka
		oFile.open(portNum.c_str(), fstream::app)oFile
		<<timeSL()<<","<<sendbuf<<",Slanje!"<<endl;
		oFile.close();
		//slanje posljednjeg parametra
		iPom =send(ConnS, sendbuf,(int)
		strlen(sendbuf), 0);
		if (iPom == SOCKET_ERROR)
		{
			cout<< "Greska kod slanja: "<<
			WSAGetLastError()<< endl;
			closesocket(ConnS);
		}
	}
	cout<<timeSL()<< "Brojka poslana: "<<string(in1,
	iPom)<< endl;
	//poziv funkcije za primanje odgovora od posluzitelja
	retVal= recv (ConnS, recvbuf, recvbuflen, 0);
	if (retVal > 0)
	{
		//Zapis u .csv datoteku vremenskog trenutka za
		primanje podatka
		oFile.open(portNum.c_str(), fstream::app);
		oFile<<timeSL()<<","<<string(in1, 4)<<",
		Primljeno!," <<string(recvbuf, retVal)<<endl;
		oFile.close();
```

```
                                        cout<<timeSL()<<" Primljena poruka:
                                        "<<string(recvbuf, retVal) <<endl;
                                        cout<< "Primljeni bitovi: "<< retVal<< endl;
                                }
                                else if (retVal == 0)
                                {
                                        //Zapis u .csv datoteku vremenskog trenutka za
                                        primanje podatka
                                        oFile.open(portN2.c_str(), fstream::app);
                                        oFile<<timeSL()<<","<<string(in1, 4)<<", Kraj
                                        TCP Veze!,"<<string(recvbuf, retVal)<<endl;
                                        oFile.close();
                                        //Zatvaranje konekcije
                                        cout<< "Veza zatvorena."<< endl;
                                        cout <<"Primanje nemoguce: "<<
                                        WSAGetLastError()<< endl<<endl;
                                }
                        }

                        //zatvaranje prikljucnice
                        spoj =closesocket(ConnS);
                        if (spoj == SOCKET_ERROR)
                        {
                                cout<< "Greska kod zatvaranja prikljucnice: "<<
                                WSAGetLastError<< endl;
                        }
                        else cout << "Uspjesno zatvorena prikljucnica."<< endl;
                }
        }
}
else
{
        cout<<"Greska kod pokretanja."<<endl;
}
//Ciscenje koristenih resursa prikljucnice
WSACleanup();
cout<<"Pritisnuti tipku za izlaz..."<<endl;
getchar();
return 0;
}

//Funkcija koja vraca trenutno lokalno vrijeme
string timeSL()
{
string sTime;
SYSTEMTIME local;
GetLocalTime(&local);
std::ostringstream ssTime;
ssTime <<local.wHour<<":"<< local.wMinute<<":"<< local.wSecond<<":"<<
local.wMilliseconds;
sTime = ssTime.str();
return sTime;
}
```

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, MARKO ČEHOK *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZVEDBA, ANALIZA I TESTIRANJE PROGRAMSKE MREŽNE PRIKLJUČNICE *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Ček M.
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, MARKO ČEHOK *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZVEDBA, ANALIZA I TESTIRANJE PROGRAMSKE MREŽNE PRIKLJUČNICE *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Ček M.
(vlastoručni potpis)