



UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
Odsek za računarstvo i automatiku
Katedra za računarsku tehniku i računarske komunikacije



DIPLOMSKI - MASTER RAD

Kandidat: Nenad Stanković

Broj indeksa: E10495

Tema rada: **Jedno rešenje programske podrške
serversko-klijentskog sistema za govorne
komunikacije**

Mentor rada: prof. Dr. Nikola Teslić

Novi Sad, Maj 2008.



UNIVERZITET U NOVOM SADU • **FAKULTET TEHNIČKIH NAUKA**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, RBR:			
Identifikacioni broj, IBR:			
Tip dokumentacije, TD:	Monografska publikacija		
Tip zapisa, TZ:	Tekstualni štampani materijal		
Vrsta rada, VR:	Diplomski-master rad		
Autor, AU:	Nenad Stanković		
Mentor, MN:	prof. Dr. Nikola Teslić		
Naslov rada, NR:	Jedno rešenje programske podrške serversko-klijentskog sistema za govorne komunikacije		
Jezik publikacije, JP:	srpski		
Jezik izvoda, Ji:	srpski		
Zemlja publikovanja, ZP:	Srbija		
Uže geografsko područje, UGP:	Vojvodina		
Godina, GO:	2008		
Izdavač, IZ:			
Mesto i adresa, MA:			
Fizički opis rada, FO: (poglavlja/strana/citata/tabela/slika/grafika/priloga)			
Naučna oblast, NO:	Elektrotehnika i računarska tehnika		
Naučna disciplina, ND:			
Predmetna odrednica/Ključne reči, PO:			
UDK			
Čuva se, ČU:	U biblioteci FTN, Novi Sad, Trg Dositeja Obradovića 6		
Važna napomena, VN:			
Izvod, IZ:			
Datum prihvatanja teme, DP:			
Datum odbrane, DO:			
Članovi komisije, KO:	Predsednik:	prof. Dr. Miodrag Temerinac	
	Član:	doc. Dr. Nebojša Pjevalica	Potpis mentora
	Član, mentor:	prof. Dr. Nikola Teslić	



UNIVERSITY OF NOVI SAD • **FACULTY TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, ANO :			
Identification number, INO :			
Document type, DT :		Monographic's publication	
Type of record, TR :		Word printed record	
Contents code, CC :		Diploma-Master work	
Author, AU :		Nenad Stanković	
Mentor, MN :		Ph. D.E.E. Nikola Teslić	
Title, TI :		A Software Solution for Server/Client Voice Communication System	
Language of text, LT :		Serbian	
Language of abstract, LA :		Serbian	
Country of publication, CP :		Serbia	
Locality of publication, LP :		Vojvodina	
Publication year, PY :		2008	
Publisher, PB :			
Publication place, PP :			
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)			
Scientific field, SF :		Electrical Engineering and Computer Science	
Scientific discipline, SD :			
Subject/Key words, S/KW :			
UC			
Holding data, HD :		Library of FTN, Novi Sad, Trg Dositeja Obradovića 6	
Note, N :			
Abstract, AB :			
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	Ph. D.E.E. Miodrag Temerinac	
	Member:	Ph. D.E.E. Nebojša Pjevalica	Mentor's sign
	Member, Mentor:	Ph. D.E.E. Nikola Teslić	

SKRAČENICE:

3G	- 3 rd Generation
3GPP	- 3 rd Generation Partnership Protocol
ACELP	- Algebraic Code Excited Linear Prediction
AMR	- Adaptive Multi-Rate
CELP	- Code Excited Linear Prediction
CNG	- Comfort Noise Generation
DTX	- Discontinuous Transmission
FTP	- File Transfer Protocol
GPRS	- General Packet Radio Service
GUI	- Graphical User Interface
HTTP	- Hypertext Transfer Protocol
IETF	- Internet Engineering Task Force
ITU	- International Telecommunication Union
IP	- Internet Protocol
LAN	- Local Area Network
MOS	- Mean Opinion Scores
NAT	- Network Address Translation
PC	- Personal Computer
PDA	- Personal Digital Assistans
PSQM	- Perceptual Speech Quality Measure
QoS	- Quality of Service
RTP	- Real-Time Transfer Protocol
RTSP	- Real-Time Streaming Protocol
SDK	- Software Development Kit
SDP	- Session Description Protocol
SIP	- Session Initiation Protocol
SRTP	- Secure Real-Time Transfer Protocol
TCP	- Transmission Control Protocol
UDP	- User Datagram Protocol
UID	- User Identifier
VAP	- Voice Activity Detection
VoIP	- Voice over Internet Protocol
WCDMA	- Wideband Code Devision Multiple Access System
Wi-Fi	- Wireless-Fidelity

SADRŽAJ:

1. UVOD.....	1
1.1. Zadatak rada	1
2. SERVERSKO-KLIJENTSKI SISTEM - TEORIJSKE OSNOVE.....	3
2.1. Načini komunikacije	3
2.2. Komunikacioni protokoli	4
2.2.1. Kanali prenosa podataka u komunikaciji	4
2.2.2. Protokol registracije	5
2.2.3. Protokol slanja spiska registrovanih klijenata	7
2.2.4. Protokol uspostave komunikacione sesije između klijenata	8
2.2.5. Tehnologija prenosa govora putem Internet protokola	10
2.2.6. Signalizacioni protokol uspostave sesije.....	11
2.3. Kompresija govora	17
3. SERVERSKO-KLIJENTSKI SISTEM - OPIS REALIZACIJE I UPOTREBA	20
3.1. Server	20
3.1.1. Opis arhitekture	21
3.1.2. Opis platforme i razvojnog okruženja.....	22
3.1.3. Modularnost.....	22
3.1.3.1. Modul A - Startovanje i obustava rada serverske aplikacije.....	24
3.1.3.2. Modul B - Mrežni protokoli i upravljanje modulima C, D, E	26
3.1.3.3. Modul C - Lista klijenata	33
3.1.3.4. Modul D - Obrada poruka signalizacionog protokola	36
3.1.3.5. Modul E - Kanal komunikacije za razgovor	37
3.1.3.6. Modul F - Parser poruka signalizacionog protokola.....	37
3.1.4. Način upotrebe.....	38
3.2. Klijent	38
3.2.1. Opis arhitekture	38
3.2.2. Opis platforme i razvojnog okruženja.....	39
3.2.3. Modularnost.....	40
3.2.3.1. Modul A - Startovanje i obustava rada klijentske aplikacije	42
3.2.3.2. Modul B - Kanal kontrole.....	44
3.2.3.3. Modul C - Kanal razmene poruka.....	45
3.2.3.4. Modul D - Razgovor i upravljanje modulima E, G, H.....	46
3.2.3.5. Modul E - Kanal komunikacije za razgovor	47
3.2.3.6. Modul F - Parser poruka signalizacionog protokola.....	48
3.2.3.7. Modul G - Rad sa ulaznim i izlazim audio uređajem	48
3.2.3.8. Modul H - Kodek	49
3.2.4. Način upotrebe.....	49
4. SERVERSKO-KLIJENTSKI SISTEM - ZAHTEVI I OGRANIČENJA	59
4.1. Performanse uređaja	59
4.2. Mreža - Brzina i propusni opseg	59
4.3. Mreža - “Skrivenost” klijenta.....	60
4.4. Prenos koda na drugu platformu	60

5. SERVERSKO-KLIJENTSKI SITSTEM - ISPITIVANJE	61
5.1. Ispitivanje - Klijent.....	61
5.2. Ispitivanje - Server	64
6. ZAKLJUČAK	67
7. LITERATURA	68

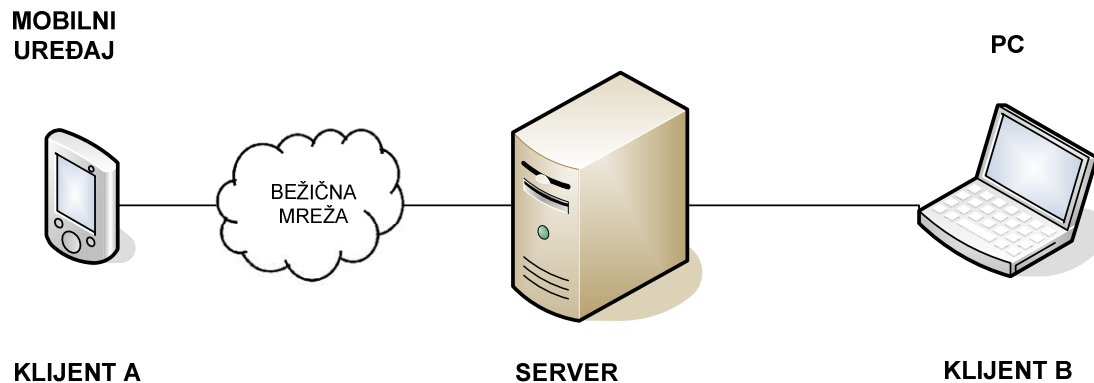
1. UVOD

Razvoj nauke i telekomunikacionih tehnologija omogućio je nastajanje mobilnih uređaja, koji zbog svoje veličine i lakoće prenosa pružaju čoveku spektar raznih novih mogućnosti u pogledu, kako brzine izvršavanja, tako i samog načina obavljanja posla. Pristup Internetu putem mobilnih uređaja, kao i sve veća rasprostranjenost mobilnih uređaja, dovela je do stvaranja ideje da se putem Internet protokola (IP) prenese govor.

1.1. Zadatak rada

Zadatak rada je realizacija programske podrške serversko-klijentskog sistema za komunikaciju između PC (engl. *Personal Computer*) i mobilnih uređaja sa primenom na VoIP (engl. *Voice over IP*) tehnologiju, odnosno zadatak podrazumeva:

- realizovanje serverske aplikacije za PC platformu sa operativnim sistemom *Windows XP*,
- realizovanje klijentske aplikacije za PC platformu sa operativnim sistemom *Windows XP* i Pocket PC platformu sa operativnim sistemom *Windows Mobile 5.0*,
- mogućnost registrovanja klijenata na server, sa obaveštavanjem klijenta o ostalim klijentima registrovanim na serveru,
- uspostavu komunikacione sesije između klijenata i servera, pri čemu se komunikacija obavlja razmenom tekstualnih poruka (engl. *chat*) i/ili razgovorom u realnom vremenu,
- sprovođenje standardnih komunikacionih protokola: TCP (engl. *Transmission Control Protocol*), UDP (engl. *User Datagram Protocol*), SIP (engl. *Session Initiation Protocol*); TCP se koristi za prenos kontrolnih poruka, SIP poruka i tekstualnih poruka, dok se UDP koristi za prenos razgovora,
- upotrebu AMR kodeka za kompresiju razgovora.
- prenos podataka i obavljanje komunikacije korišćenjem različitih tipova mreža.



Slika 1.1-1: Serversko-klijentski sistem

Na slici 1.1-1 prikazan je serversko-klijentski sistem, koji se sastoji iz klijentske i serverske aplikacije, čiji je cilj da obezbedi posredstvom servera komunikaciju između različitih klijentskih uređaja (na primer, PC - engl. *Personal Computer*, PDA - engl. *Personal Digital Assistans*, mobilni telefon i slično).

Pod komunikacijom se podrazumeva:

- razmena audio podataka u realnom vremenu putem IP protokola, kompresovanih korišćenjem AMR (engl. *Adaptive Multi-Rate*) kodeka, pri čemu je za komunikaciju primenjivana VoIP tehnologija, i
- razmena tekstualnih poruka između strana u komunikaciji.

2. SERVERSKO-KLIJENTSKI SISTEM - TEORIJSKE OSNOVE

Predmet ovog poglavlja biće sagledavanje osobina i načina obavljanja komunikacije između dva klijenta, **A** i **B** (gde je klijent **A** klijent koji inicira poziv sa klijentom **B**, a klijent **B** je klijent koga poziva klijent **A**), i jednog servera, što će prikazati samo minimalnu funkcionalnost serversko-klijentskog sistema. Osim navedenog, u ovom poglavlju će biti dat i opis postojećih tehnologija korišćenih prilikom realizacije serversko-klijentskog sistema.

2.1. Načini komunikacije

Komunikacija u okviru serversko-klijentskog sistema može se vršiti na dva načina kada:

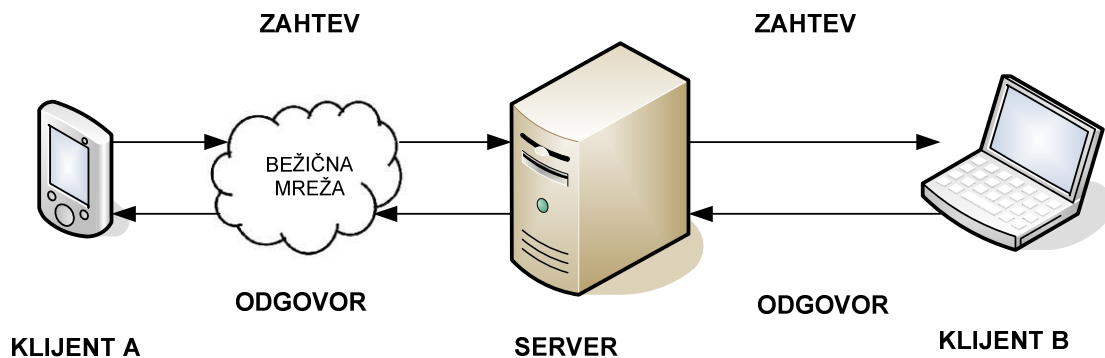
- klijent upućuje zahteve serveru na obradu - **klijent-server komunikacija**,
- klijent upućuje zahteve ili korisne podatke, u vidu teksta ili audio zapisa, ka drugom klijentu - **klijent-server-klijent komunikacija**.

Klijent-server komunikacija, ili jedan njen ciklus, započinje na taj način što klijentska strana upućuje serverskoj strani zahteve i čeka odgovor od serverske strane. Serverska strana prima zahtev razmatra ga i obrađuje, i nakon obrade klijentskoj strani šalje adekvatan odgovor. Kada klijentska strana primi odgovor, ovim se smatra da je jedan ciklus **klijent-server komunikacije** završen. Jedan ciklus **klijent-server komunikacije** prikazan je na slici 2.1-1.



Slika 2.1-1: Klijent-server komunikacija

Klijent-server-klijent komunikacija, ili jedan njen ciklus, započinje na taj način što klijent **A** upućuje zahtev, ili korisne podatke, ka klijentu **B**. Kako klijenti moraju komunicirati posredstvom servera, zahtev od klijenta **A** prvo je primljen od strane servera, pri čemu ga server razmatra i prosleđuje ka klijentskoj strani kojoj je namenjen, odnosno u ovom slučaju klijentu **B**. Kada klijent **B** primi zahtev, razmatra ga i obrađuje. Kada klijent **B** obradi zahtev, šalje odgovor ka serveru, da bi bio prosleđen klijentu **A**. Server prima odgovor, razmatra ga, nakon čega ga prosleđuje klijentskoj strani kojoj je namenjen, odnosno u ovom slučaju klijentu **A**. Kada klijent **A** primi odgovor, ovim se smatra da je jedan ciklus **klijent-server-klijent komunikacije** završen. Jedan ciklus **klijent-server-klijent komunikacije** prikazan je na slici 2.1-2.



Slika 2.1-2: Klijent-server-klijent komunikacija

Komunikacija serversko-klijentskog sistem može se vršiti između:

- PC uređaja,
- mobilnih uređaja i
- PC i mobilnih uređaja.

2.2. Komunikacioni protokoli

Zahtevi i odgovori, kao i sama komunikacija, regulisani su pravilima koje nameću protokoli, od kojih je korišćeno više protokola; nestandardni protokoli: **protokol registracije** klijenta na server, **protokol slanja spiska registrovanih klijenata** i **protokol uspostave komunikacione sesije između klijenata**; standardni protokol: **SIP**. Svi navedeni protokoli oslanjaju se na protokole **TCP/IP** i **UDP** (TCP i UDP protokol su realizovani korišćenjem *Winsock 2* biblioteke, odnosno *Windows Sockets API*-ja).

2.2.1. Kanali prenosa podataka u komunikaciji

Prilikom prenosa podataka u komunikacionoj sesiji između klijenata i servera koriste se tri kanala prenosa podataka:

- **TCP kanal kontrole,**
- **TCP kanal komunikacije**
- **UDP kanal komunikacije**

Svi zahtevi od klijenata upućeni serveru prenose se kroz mrežu putem TCP kanala, koji se naziva **TCP kanal kontrole**.

Definisani su sledeći zahtevi koje server može primiti od klijenta:

- **zahtev za registraciju klijenta na server,**
- **zahtev za uspostavu komunikacione sesije između klijenata**

Zahtev za registraciju klijenta na server je zahtev koji server prima od klijenta prilikom registracije klijenta na server. Klijent može tražiti ovaj zahtev samo ukoliko nije registrovan na server, u suprotnom server obaveštava klijenta da je već registrovan.

Zahtev za uspostavu komunikacione sesije između klijenata je zahtev koji server prima od registrovanog klijenta ukoliko taj klijent traži uspostavu komunikacione veze sa drugim klijentom, registrovanim na serveru.

TCP kanalom kontrole prenose se, osim **zahteva za registraciju klijenta na server** i **zahteva za uspostavu komunikacione sesije između klijenata**, i odgovori servera na ove zahteve, kao i greške do kojih dolazi u radu serversko-klijentskog sistema, kao na primer, ukoliko je jedna klijentska strana nasilno prekinula vezu sa serverom u toku komunikacione sesije, ili ukoliko je došlo do greške na serveru, i slično. Isto tako, server **TCP kanalom kontrole** šalje klijentima **spisak registrovanih klijenata**. **Spisak registrovanih klijenata** odražava ono stanje **liste klijenata** u kojem se nalazi **lista klijenata** u trenutku slanja ovog spiska, i sadrži samo one informacije o klijentima koje su relevantne za klijente, a to su **korisničko ime klijenta** i **jedinstveni identifikacioni broj, UID** - broj koji klijent dobija nakon uspešne registracije na server. Slanje **spiska registrovanih klijenata**, odnosno njegovo ažuriranje kod klijenata, server obavlja prilikom svake prijave novog ili odjave starog klijenta, čime se klijentima obezbeđuje informacija o stanju klijenata trenutno prijavljenih na serveru, odnosno koliko i koji su klijenti prijavljeni na server.

Prenos SIP poruka, koje služe, kako za iniciranje komunikacione sesije, tako i za pregovaranje između klijenata u komunikacionoj sesiji, obavlja se putem TCP kanala, koji se naziva **TCP kanal komunikacije**. Osim razmene SIP poruka, **TCP kanalom komunikacije** obavlja se i razmena tekstualnih poruka, odnosno komunikacija putem teksta u realnom vremenu.

Komunikacija u vidu razgovora u realnom vremenu prenosi putem UDP kanala, koji se naziva **UDP kanal komunikacije**. UDP je korišćen iz tog razloga što nema vremena za ponovno slanje izgubljenih paketa dok se vodi razgovor u realnom vremenu.

TCP kanal kontrole se stvara u trenutku prijave klijenta na server, i ovaj kanal prisutan je od trenutka prijave klijenta na server, pa sve do njegove odjave sa servera. **TCP kanalom kontrole** obavlja se **klijent-server komunikacija**. **TCP kanal komunikacije** i **UDP kanal komunikacije** stvaraju samo kad ima potrebe za njima, odnosno samo ukoliko jedan klijent želi da komunicira sa drugim klijentom u mreži. Ovim kanalima (**TCP kanalom komunikacije** i **UDP kanalom komunikacije**) obavlja se **klijent-server-klijent komunikacija**.

2.2.2. Protokol registracije

Protokol registracije je nestandardni protokol koji se koristi prilikom registracije, odnosno prijave, klijenta na server.

Registrovanje klijenta na server obavlja se tako što klijent šalje serveru **zahtev za registraciju klijenta na server**, sledećeg formata:

REGISTER:<user_name>

Na osnovu formata, uočava se da se **zahtev za registraciju klijenta na server** sastoji iz dva dela:

- **REGISTER**, na osnovu koga server zna da se radi o **zahtevu za registraciji klijenta na server**,
- **<user_name>**, obavezni parametar zahteva, i informacija serveru o korisničkom imenu klijenta, pod kojim klijent želi biti prijavljen na server.

Ova dva dela razdvojena su delimiterom, odnosno graničnikom, dvotačka, “:”.

Mogući odgovori od servera klijentu na **zahtev za registraciju klijenta na server** su:

- **502 BAD_REGISTRATION**,
- **501 USER_PRESENT**,
- **500 SERVER_ERROR**,
- **spisak registrovanih klijenata**.

Nakon primljenog **zahteva za registraciju klijenta na server**, server proverava ispravnost formata zahteva. Ukoliko je format zahteva neispravan, registracija klijenta na server je nemoguća i klijent se obaveštava porukom “**502 BAD_REGISTRATION**”. Ukoliko je format zahteva ispravan, zahtev se parsira, odnosno svodi se na njegove osnovne delove. Nakon što je zahtev isparsiran, server proverava da li je korisničko ime klijenta, dobijeno iz zahteva, jedinstveno, odnosno da li u **listi klijenata** ne postoji ni jedan klijent sa istim korisničkim imenom. U slučaju da klijent sa istim korisničkim imenom već postoji u listi klijenata, tada je registracija klijenta na server nemoguća i klijent se obaveštava porukom “**501 USER_PRESENT**”. U slučaju da je korisničko ime klijenta, koji je zahteva registraciju, jedinstveno, tada se taj klijent upusuje u listu klijenata, odnosno njegove karakteristike (samo klijentske karakteristike relevantne za server). Upisom u listu klijenata, server dodeljuje klijentu jedinstven identifikacioni broj, **UID**. Kada je registracija klijenta na serveru završena, server šalje **spisak registrovanih klijenata**, kako trenutno registrovanom klijentu, tako i svim ostalim registrovanim klijentima.

Ukoliko na serveru, prilikom registracije klijenta, dođe do greške, tada server šalje poruku “**500 SERVER_ERROR**”.

Na slici 2.2.2-1 prikazan je **protokol registracije** klijenta na server. Tekst na slici **SERVEROV ODGOVOR** predstavlja supstituciju jednog od četiri moguća serverova odgovora na upućeni klijentski **zahtev za registraciju klijenta na server**.

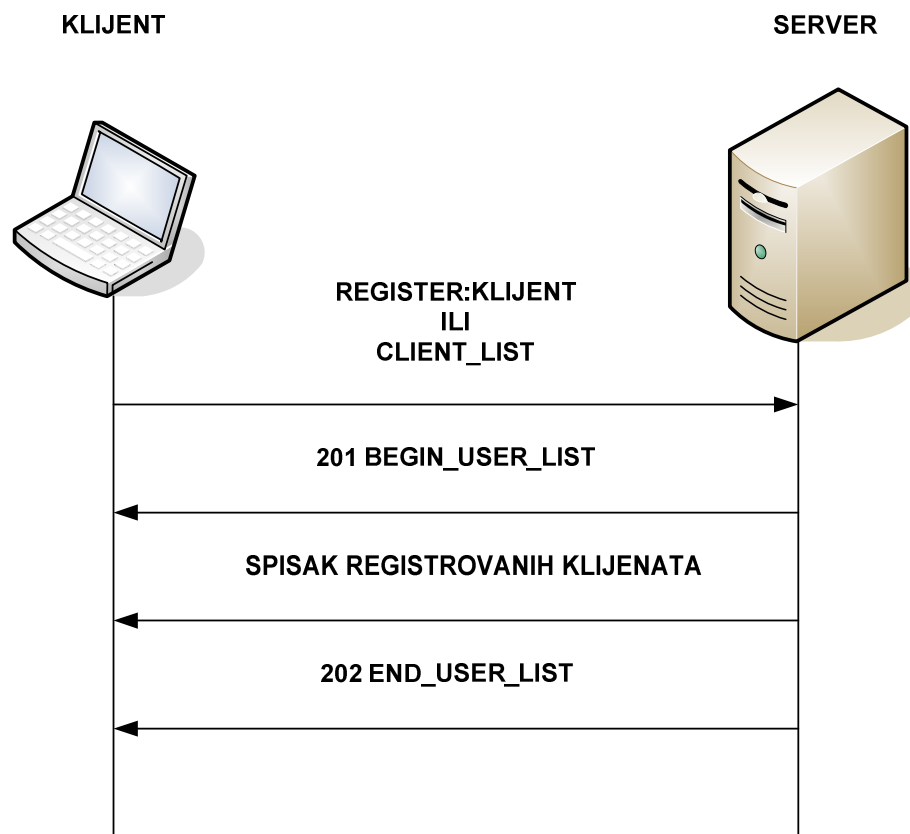


Slika 2.2.2-1: Protokol registracije

2.2.3. Protokol slanja spiska registrovanih klijenata

Protokol slanja spiska registrovanih klijenata je nestandardni protokol koji reguliše slanje **spiska registrovanih klijenata** od strane servera.

Ovaj protokol se primenjuje nakon uspešne registracije klijenta na server, ali postoji mogućnost primene ovog protokola i na klijentov zahtev (implementacija klijentskog zahteva za dobijanje spiska registrovanih klijenata je podržana od strane klijentske aplikacije, ali se razmatra o njenom uklanjanju u budućim verzijama razvoja klijentske aplikacije, s obzirom na to da server šalje **spisak registrovanih klijenata** prilikom svake prijave novog klijenta, odnosno odjave starog klijenta, te je postojanje ovog zahteva suvišno, pa stoga ovaj zahtev nije pominjan ranije, ali svakako će u ovom poglavlju biti objašnjen, jer razlika između dobijanja **spiska registrovanih klijenata** nakon uspešne registracije klijenta na server i klijentovim zahtevanjem dobijanja ovog spiska je samo u zahtevu za dobijanje spiska registrovanih klijenata, ostalo je sve isto).



Slika 2.2.3-1: Protokol slanja spiska registrovanih klijenata

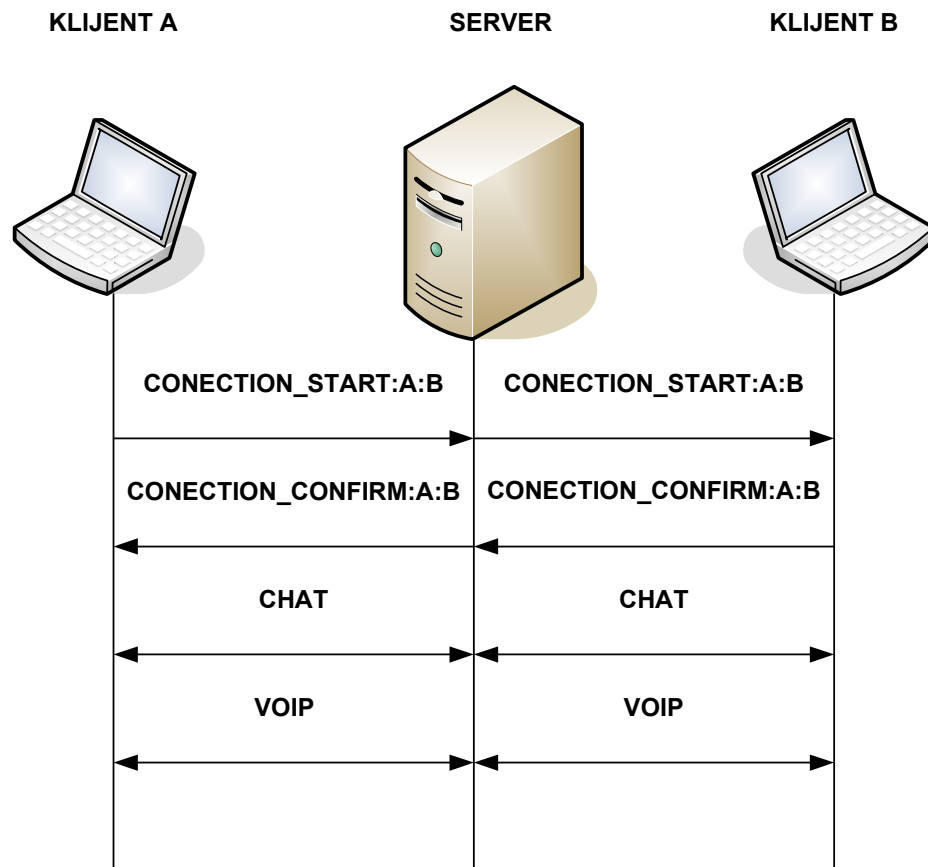
Bilo da je klijent poslao zahtev za dobijanje spiska registrovanih klijenata, **“CLIENT_LIST”**, ili da je klijent uspešno registrovan na server, **protokol slanja spiska registrovanih klijenata** je uvek isti.

Server pošalje klijentu poruku **“201 BEGIN_USER_LIST”**, tako da ga obavesti da će sledeći podaci, koje će primati, putem **TCP kanala kontrole**, biti u stvari **spisak registrovanih klijenata**, nakon čega ubrzo započinje i samo slanje spiska. Podaci o klijentu koje server šalje u **spisku registrovanih klijenata** su: **korisničko ime klijenta** i **jedinstveni identifikacioni broj klijenta (UID)**, za svakog klijenta posebno (ovi podaci su čuvani na serveru u **listi klijenata**). Kada server pošalje podatke o poslednjem klijentu iz **liste klijenata**, tada on kao sledeću poruku šalje **“202 END_USER_LIST”**, čime klijent zna da je primio čitav **spisak registrovanih klijenata**.

Server, takođe, može poslati poruku **“505 USER_LIST_EMPTY”** klijentu, koja se šalje samo u slučaju kada je **lista klijenata** prazna, što je praktično nemoguće, osim ukoliko nešto nije u redu sa **listom klijenata** na serveru, a samim tim to implicira da sa serverom nešto nije u redu.

2.2.4. Protokol uspostave komunikacione sesije između klijenata

Protokol uspostave komunikacione sesije između klijenata je nestandardni protokol kojim se reguliše uspostava komunikacione sesije između klijenata, posredstvom servera.



Slika 2.2.4-1: Protokol uspostave komunikacione sesije između klijenata

Protokol uspostave komunikacione sesije između klijenata razrešava ogroman problem koji se javio prilikom rada serversko-klijentskog sistema. Naime, reč je o problemu, odnosno činjenici, da klijentima u nekim mrežama nije omogućeno da slušaju na određenom prolazu, odnosno nije im omogućeno da primaju zahteve za uspostavom veze (engl. *accepting connections*), što je posledica “skrivenosti” klijenata u mreži, odnosno klijenti u mreži ne poseduju javne IP adrese (engl. *public IP address*). Uvođenjem **protokola uspostave komunikacione sesije između klijenata** problem se razrešava.

Prvobitan serversko-klijentski sistem funkcionisao je na taj način tako da klijenti preko servera dobiju informacije o svim klijentima registrovanim na server, nakon čega su klijenti bili u mogućnosti da ostvaruju komunikaciju među sobom nezavisno od servera. Uloga servera u prvobitnom sistemu je praktično bila samo slanje **spiska registrovanih klijenata**, ali kako se javio gore pomenuti problem, nemogućnosti klijenata da poseduju javne IP adrese, server dobija posredničku ulogu u komunikaciji unutar serversko-klijentskog sistema, i on predstavlja glavnu i ne izostavnu komponentu ovog sistema.

Primena protokola obavlja se kada jedan klijent, **A**, hoće da uspostavi komunikacionu sesiju sa drugim klijentom, **B**, i samim tim ostvari komunikaciju sa tim klijentom.

Protokol uspostave komunikacione sesije između klijenata započinje tako što klijent **A** uputi serveru **zahtev za uspostavu komunikacione sesije između klijenata**, koji ima sledeći format:

CONNECTION_START:<source_UID>:<destination_UID>.

Na osnovu formata, uočava se da se **zahtev za uspostavu komunikacione sesije između klijenata** sastoji iz tri dela:

- **CONNECTION_START**, na osnovu koga server zna da se radi o **zahtevu za uspostavu komunikacione sesije između klijenata**,
- **<source_UID>**, obavezni parametar zahteva, i informacija serveru o jedinstvenom identifikacionom broju klijenta koji je inicirao sesiju, odnosno informacija o **UID-u klijenta A**,
- **<destination_UID>**, obavezni parametar zahteva, i informacija serveru o jedinstvenom identifikacionom broju klijenta koji se poziva, odnosno informacija o **UID-u klijenta B**.

Tri dela u formatu zahteva razdvojena su delimeterom, odnosno graničnikom, dvotačka, “.”.

Kada server primi **zahtev za uspostavu komunikacione sesije između klijenata**, tada server iz ovog zahteva parsira informacije koje jednoznačno određuju klijenta koji je uputio zahtev, odnosno klijenta **A**, i informacije koje jednoznačno određuju klijenta sa kojim klijent **A** želi uspostaviti komunikacionu sesiju, odnosno klijenta **B**. Nakon provere ispravnosti informacija o klijentima, **A** i **B**, server upućuje zahtev za uspostavom veze klijentu **B**, pri čemu klijent **B** razmatra i obrađuje zahtev. Nakon obrade klijent **B** odgovara serveru. **Odgovor na zahtev za uspostavu komunikacione sesije između klijenata** poslat serveru od klijenta **B** ima sledeći format:

CONNECTION_CONFIRM:<source_UID>:<destination_UID>.

Na osnovu formata, uočava se da se **odgovor na zahtev za uspostavu komunikacione sesije između klijenata** sastoji iz tri dela:

- **CONNECTION_CONFIRM**, na osnovu koga server zna da se radi o **odgovoru na zahtev za uspostavu komunikacione sesije između klijenata**,
- **<source_UID>**, obavezni parametar zahteva, i informacija serveru o jedinstvenom identifikacionom broju klijenta koji je inicirao sesiju, odnosno informacija o **UID-u klijenta A**,
- **<destination_UID>**, obavezni parametar zahteva, i informacija serveru o jedinstvenom identifikacionom broju klijenta koji je pozivan, odnosno informacija o **UID-u klijenta B**.

Delovi u formatu odgovora na zahtev razdvojeni su delimeterom, odnosno graničnikom, dvotačka, “.”.

Kad server primi odgovor od klijenta **B** (a to je upravo **odgovor na zahtev za uspostavu komunikacione sesije između klijenata**), server isti odgovor šalje klijentu **A**, čime se označava da je zahtevana uspostava komunikacione sesije uspešno obavljena, i da komunikacija između klijenata **A** i **B** može započeti (VoIP razgovor i razmena tekstualnih poruka). U slučaju neprimljenog odgovora od klijenta **B**, čak iako server ne šalje ništa klijentu **A**, klijent **A** ima vremenski brojač (engl. *timer*), podešen na određeno vreme. Kad to vreme istekne, klijent **A** će znati da uspostava komunikacione sesije nije bila uspešna.

2.2.5. Tehnologija prenosa govora putem Internet protokola

VoIP predstavlja tehnologiju za prenošenje govora putem Interneta ili neke druge paketski komutirane mreže (engl. *packet-switched network*). VoIP se uglavnom koristi kao abstraktna predstava prenosa govora u mreži, nego kao sama njegova implementacija u protokol. Takođe, VoIP je poznat i pod imenima IP telefonija (engl. *IP Telephony*), Internet telefonija (engl. *Internet Telephony*) i slično.

Govor se putem Interneta prenosi u obliku digitalnog audio zapisa, pa se stoga na govor mogu primeniti razne tehnike kompresije govora, kao na primer, kompresija korišćenjem AMR kodeka. Kompresovani govor se pakuje u pakete određene veličine, i tako zapakovani, paketi se šalju jedan za drugim, u kontinuiranom toku, kroz IP mrežu.

VoIP pruža razne mogućnosti i servise, čija je implementacija veoma teška i skupa u okviru tradicionalne telefonske mreže, kao što su na primer:

- sposobnost uspostave i prenosa jednog ili više telefonskih razgovora kroz isti mrežni priključak, čime se pruža mogućnost lakog dodavanja i korišćenja VoIP telefonskih linija,
- servisne dodatke: tri načina zvanja (engl. *3-way calling*), preusmeravanje poziva (engl. *call forwarding*), automatsko ponovno biranje broja (engl. *automatic redial*), i identifikaciju pozivaoca (engl. *caller ID*); nabrojane dodatke VoIP usluga pruža besplatno za razliku od tradicionalne telefonije koja ih dodatno naplaćuje,
- zaštitu poziva korišćenjem standardnih protokola, kao što je SRTP protokol (engl. *Secure Real-Time Transport Protocol*). Kod tradicionalne telefonije nailazi se na ogromne teškoće u sveri zaštite telefonske linije, odnosno telefonskog razgovora, dok je kod VoIP razgovora to relativno lako izvodljivo zbog toga što je razgovor već digitalizovan, i s obzirom na to moguća je primena raznih algoritama šifrovanja,
- lokacijski je nezavisan, odnosno za korišćenje VoIP usluge jedino je neophodan pristup internetu,
- mogućnost integracije sa drugim uslugama dostupnih na Internetu, uključujući video konverzaciju, konverzaciju putem poruka, razmenu dokumenta putem FTP protokola (engl. *File Transfer Protocol*), audio konferencije, i slično; sve nabrojane usluge mogu se koristiti paralelno sa korišćenjem VoIP usluge.

Sama implementacija VoIP usluge (engl. *service*) pruža programerima mnogobrojne izazove.

Kako za prenos govora VoIP koristi UDP, a UDP sam po sebi nije protokol koji pruža pouzdanost prilikom prenosa podataka, i nikad se sa sigurnošću ne može znati da li su podaci dostavljeni, kao i to da UDP ne pruža QoS uslugu (engl. *Quality of Service*), tako se prilikom implementacije VoIP usluge programeri suočavaju sa problemima u kašnjenju paketa podataka, kao i sa smetnjama prilikom prenosa podataka, na primer u obliku šuma, treperenja faze signala, i slično. Ovi problemi su uglavno rešeni buferovanjem govora.

Drugi izazov koji se stavlja pred programere je prenos VoIP saobraćaja kad su klijenti “sakriveni” iza sistema zaštite korisnika od napada sa mreže, popularno nazivanim **firewall**, i/ili kada su klijenti “sakriveni” iza, recimo, NAT servera, zastupničkog servera i slično.

Osim ovih izazova tu su još mnogi drugi, od kojih se mogu izdvojiti sledeći:

- adaptacija protoka u mreži, odnosno korišćenje jače ili slabije kompresije govora u mreži u zavisnosti od same brzine protoka podataka u mreži,
- gubljenje paketa, odnosno njegovo smanjivanje ili potpuno sprečavanje, ali tako da ne utiče na komunikaciju između klijenata,
- kašnjenje paketa - latencija, odnosno korišćenje naprednih algoritama za smanjenje kašnjenja podataka u mreži,
- eho, odnosno njegovo sprečavanje,
- sigurnost, odnosno uvođenje algoritama zaštite i šifrovanja prilikom trajanja VoIP razgovora,
- pouzdanost u prenosu govora, i slično.

VoIP može koristiti razne protokole za uspostavljanje komunikacione sesije od koji se izdvajaju dva konkurentna protokola:

- **ITU** (engl. *International Telecommunication Union*) **standard H.323**, i
- **IETF** (engl. *Internet Engineering Task Force*) **standard SIP**.

U mrežama gde je sve u kontroli mrežnog operatora ili tradicionalne telefonije, H.323 protokol je bolji izbor protokola u odnosu na SIP. Međutim, SIP protokol za razliku od H.323 protokola ima mogućnost da zaobiđe NAT i **firewall**-ove, dok H.323 protokol to ne može. (Dodatna objašnjenja o načinu rada i sprovođenja VoIP protokola pogledati u literaturi [7. \[1\]](#)).

2.2.6. Signalizacioni protokol uspostave sesije

SIP protokol predstavlja aplikacioni, odnosno signalizacioni, protokol namenjen za kreiranje, modifikovanje, prekidanje sesija sa jednim ili više učesnika. Ovaj protokol može biti iskorišćen za kreiranje dvostranih, višestranih sesija i sesija u kojima se korisni podaci istovremeno upućuju ka grupi korisnika, odnosno multikast sesija (engl. *multicast session*), s ciljem ostvarivanja Internet telefonski razgovora, distribucije multimedijalnog sadržaja, kao i multimedijalne konferencije. SIP protokol je protokol nezavisan od transportnih protokola koji se nalaze ispod njega: TCP, UDP i slično. Ovo znači da se SIP protokol može sprovesti bilo da se kao protokol nižeg nivoa koristi TCP, ili UDP, ili neki drugi protokol. SIP je protokol baziran na razmeni tekstualnih poruka. S obzirom na to da je baziran na razmeni tekstualnih poruka, ljudima je omogućeno čitanje SIP poruka.

Kako je SIP protokol signalizacioni protokol, on služi za uspostavu komunikacije između SIP klijenata i SIP servera. SIP klijenti koriste TCP ili UDP protokol kako bi uspostavili vezu sa SIP serverom. Kada se uspostavi veza između SIP klijenata i SIP servera, tada oni (SIP klijenti i SIP serveri) putem istog protokola, TCP ili UDP, obavljaju razmenu SIP poruka između sebe. SIP protokol se koristi za pregovaranje o načinu na koji će biti obavljana komunikaciona sesija između strana u komunikaciji (SIP klijenata i SIP servera).

Kako SIP služi za uspostavu komunikacione sesije, isto tako on služi i za raskidanje komunikacione sesije. S obzirom na to da se SIP protokol koristi za uspostavu i raskid komunikacionih sesija, ovaj protokol se može koristiti u svim aplikacijama gde postoji potreba za uspostavom i raskidom komunikacione sesije. SIP protokol je uglavnom korišćen za uspostavu i raskid audio i video razgovora, koji se uglavnom prenosi putem Interneta ili neke druge mreža.

Sam prenos audio i video podataka ne obavlja se korišćenjem SIP protokola, već korišćenjem, na primer, RTP (engl. *Real-Time Transport Protocol*), RTSP (engl. *Real-Time Streaming Protocol*) i drugih protokola specijalizovanih za regulisanje prenosa audio i video podataka u mreži.

SIP protokol namenjen je samo za podešavanje, signaliranje, uspostavu i raskid komunikacione sesije između SIP klijenata i SIP servera.

U SIP poruke može biti integrisan SDP (engl. Session Description Protocol), kojim se opisuje multimedijalni sadržaj koji će biti prenošen u sesiji, koji će se kodek koristiti za njegovo enkodovanje i dekodovanje, koji će se prolazi koristiti za prenos multimedijalnog sadržaja, i slično. Praktično, SIP sesije podrazumevaju jednostavan prenos audio i video podataka organizovanih u pakete, gde se sam prenos obavlja u realnom vremenu, regulisan najčešće RTP protokolom.

SIP je sličan HTTP protokolu (engl. Hypertext Transfer Protocol) u pogledu principa korišćenih prilikom sprovođenja SIP protokola: čitljiv je za ljude, i zasniva se na razmeni poruka zahteva i odgovora, gde neke poruke imaju isto značenje kao kod HTTP protokola, na primer: “404 not found”, i slično.

Serversko-klijentski sistem za uspostavu komunikacione sesije između klijenata koristi SIP protokol.

Podržani SIP zahtevi prilikom uspostave komunikacione sesije u okviru serversko-klijentskog sistema su:

- **INVITE**,
- **BYE**,
- **CANCEL**.

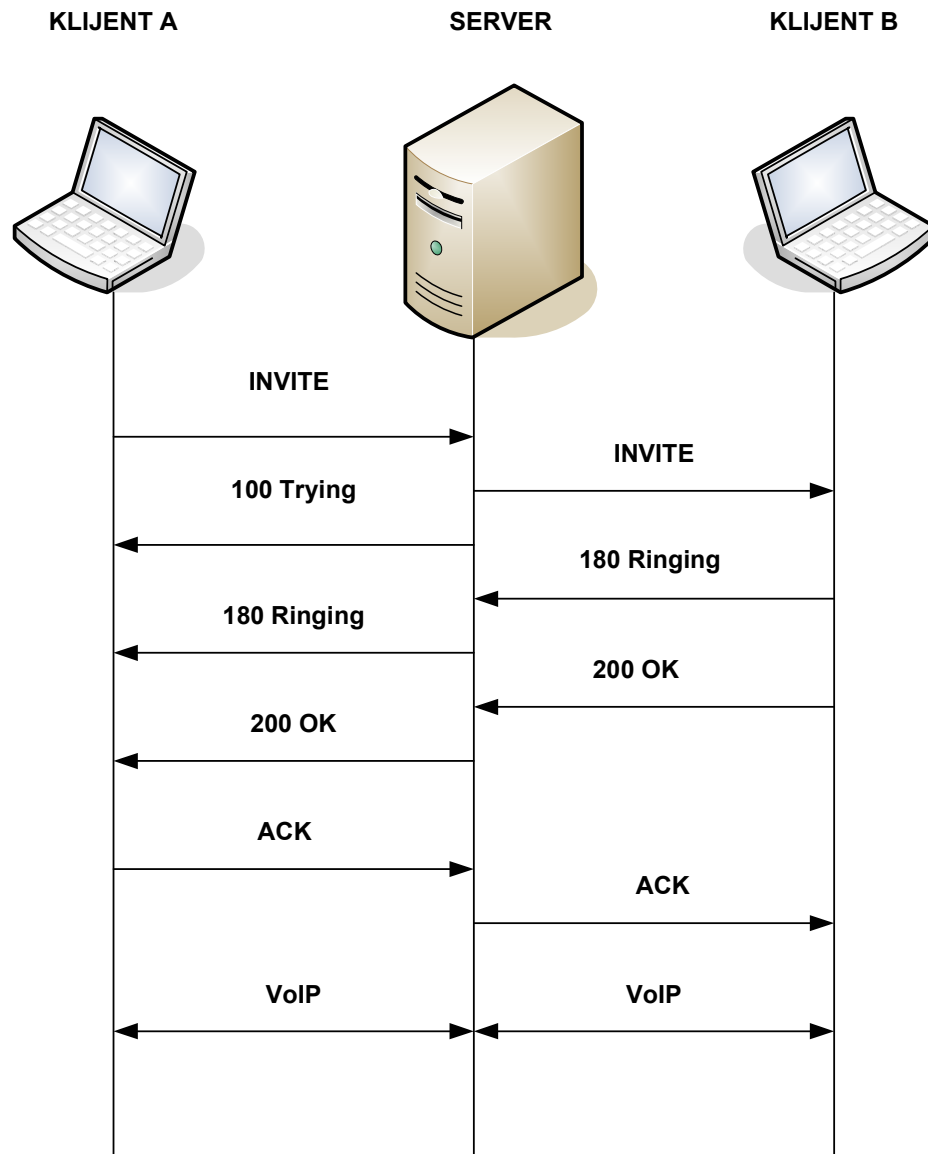
SIP odgovori na podržane zahteve prilikom uspostave komunikacione sesije u okviru serversko-klijentskog sistema su:

- **100 Trying**,
- **180 Ringing**,
- **200 OK**,
- **ACK**.

Radi lakšeg izlaganja čemu koji zahtev služi, a samim tim i odgovora na te zahteve, u nastavku sledi primer uspostave komunikacione SIP sesije između klijenata i servera u okviru serversko-klijentskog sistema.

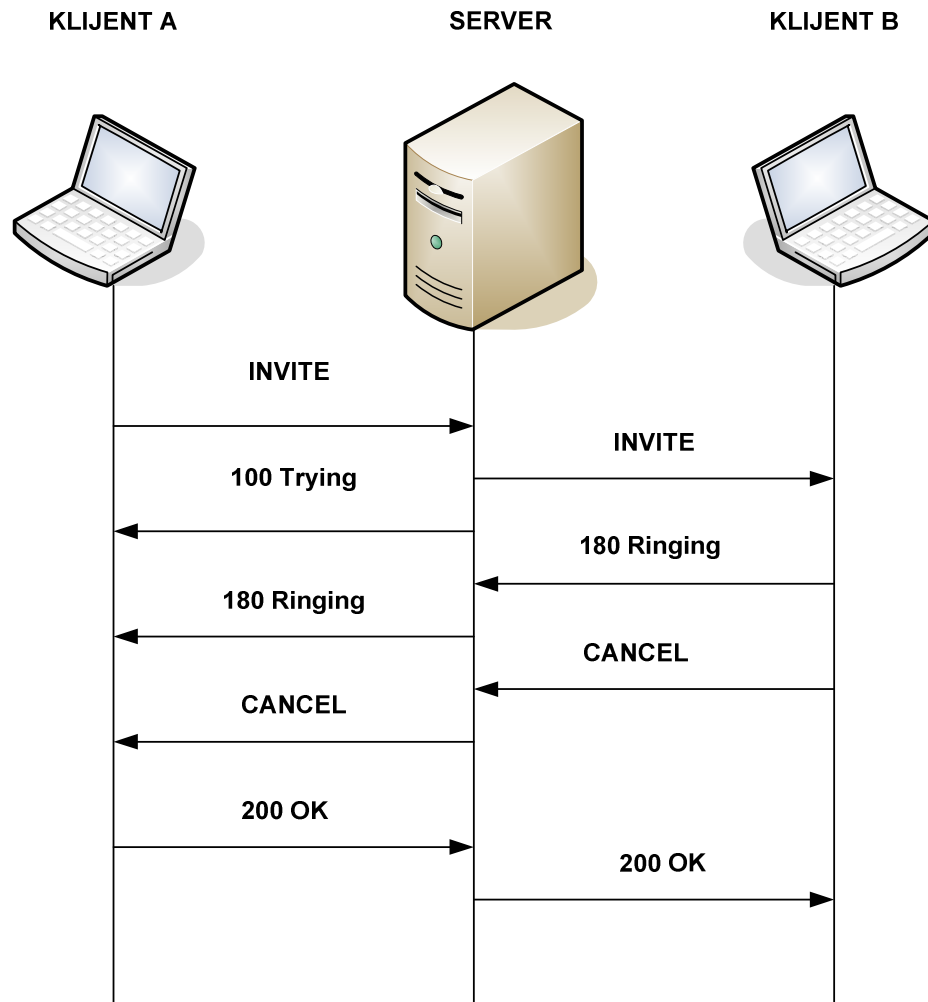
U okviru serversko-klijentskog sistema nalaze se dva klijenta, **A** i **B** (definisani su ranije), i jedan server. Nakon uspešno sprovedenog **protokola registracije**, kao i **protokola uspostave komunikacione sesije između klijenata** (pogledaj [2.2.2](#) i [2.2.4](#)), klijenti započinju pregovaranje o načinu obavljanja komunikacione sesije putem SIP protokola.

SIP protokol započinje tako što klijent **A** upućuje **INVITE** zahtev ka serveru, i čeka adekvatan odgovor od servera. Server prima **INVITE** zahtev i razmatra ga s ciljem da iz tog zahteva sazna informaciju o klijentu kojem je zahtev upućen. Nakon utvrđivanja kojem klijentu je upućen zahtev, server taj zahtev prosleđuje tom klijentu, u ovom slučaju je to klijent **B**, a klijentu **A**, šalje SIP poruku **100 Trying**, kao obaveštenje da pozivani klijent postoji u mreži i da mu je prosleđen **INVITE** zahtev. Kada klijent **B** primi **INVITE** zahtev, on automatski ka serveru šalje poruku **180 Ringing**, kao znak da je zahtev primljen na obradu kod klijenta **B**. Server prima tu poruku, razmatra je i prosleđuje ka klijentu **A**. (U trenutku slanja poruke **180 Ringing**, u slučaju da se u komunikaciji koriste SIP telefoni, kod klijenta **B** zvoni telefon. A kada poruka **180 Ringing** stigne do klijenta **A**, on na slušalici svog telefona ima indikaciju da klijentu **B** zvoni telefon).

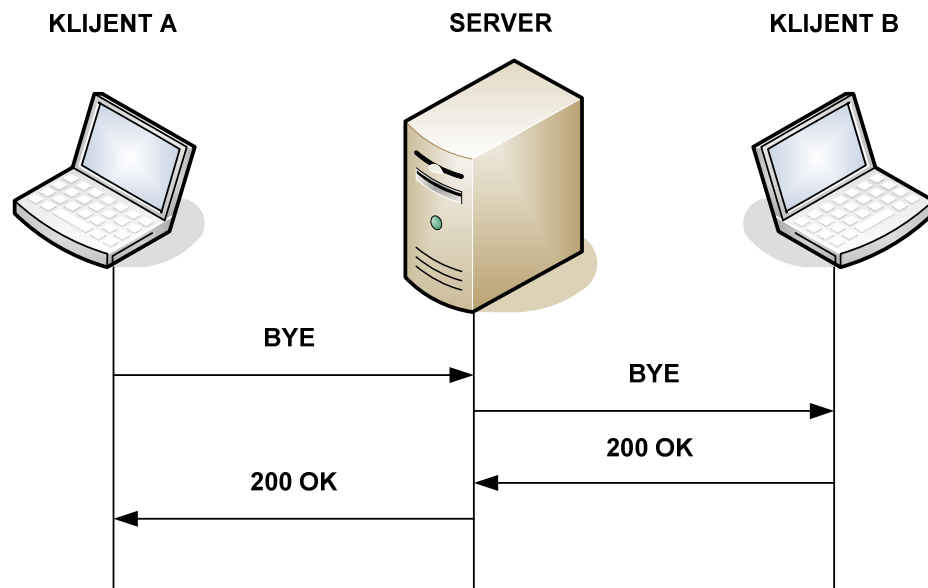


Slika 2.2.6-1: Uspešna uspostava komunikacione sesije korišćenjem SIP protokola;
Klijent B je prihvatio poziv od klijenta A

Takođe, nakon primljenog **INVITE** zahteva, klijent **B** treba da odluči da li će prihvatiti poziv od klijenta **A** ili će odbiti taj poziv. Ukoliko klijent **B** odbija poziv on šalje poruku serveru **CANCEL**, kojom obaveštava servera da je poziv odbijen. Nakon što server primi **CANCEL** SIP poruku, on je prosleđuje klijentu **A**, kako bi klijent **A** znao da je klijent **B** odbio poziv, pri čemu klijent **A** odgovara sa **200 OK** serveru koji dalje prosleđuje **200 OK** klijentu **B**. Ukoliko klijent **B** prihvata poziv on upućuje prema serveru SIP poruku **200 OK**, kao indicaciju da je poziv od klijenta **A** prihvaćen. Server prima poruku **200 OK** i prosleđuje ga ka klijentu **A**, pri čemu klijent **A** šalje **ACK** SIP poruku, kao indicaciju da je primio poruku **200 OK** i da komunikaciona sesija može da počne. Server prima **ACK** poruku i prosleđuje je ka klijentu **B**. Kada klijent **B** primi **ACK** poruku, to predstavlja početak komunikacione sesija, bilo u vidu govora u realnom vremenu, razmenom tekstualnih poruka ili pak ostvarivanjem video razgovora, ili na neki drugi način.



Slika 2.2.6-2: Neuspešna uspostava komunikacione sesije korišćenjem SIP protokola;
Klijent B je odbio poziv od klijenta A



Slika 2.2.6-3: Raskidanje komunikacione sesije korišćenjem SIP protokola

Kada bilo koji od klijenata, klijent A ili klijent B, želi da završi komunikacionu sesiju on šalje serveru poruku **BYE**, koju ovaj prosleđuje ka drugoj strani u komunikacionoj sesiji. Kada ta druga strana primi **BYE**, kao odgovor šalje **200 OK**, što je indikacija da je primljena poruka **BYE**. Kada klijent, koji poslao poruku **BYE**, primi **200 OK**, komunikaciona sesija se smatra završenom.

Razmena SIP poruka između komunikacionih strana u komunikacionoj sesiji ilustrovana je na slikama 2.2.6-1, 2.2.6-2 i 2.2.6-3. (Dodatna objašnjenja o načinu rada i sprovođenja SIP protokola pogledati u literaturi [7. \[2\]](#)).

U nastavku slede primeri pomenutih SIP poruka.

INVITE:

```
INVITE sip:101@192.168.9.100 SIP/2.0
Via: SIP/2.0/UDP 192.168.9.86:80
Max-Forwards: 70
Contact: <sip:100@192.168.9.86:80>
To: "KLIJENT B" <sip:101@192.168.9.100>
From: "KLIJENT A" <sip:100@192.168.9.86:80>
Call-ID: DkyMDY3Y2FkMWQxMGQ0MGM2YWU2MmZhNmUzYWZjZWU.
CSeq: 1 INVITE
Content-Type: application/sdp
Content-Length: 357
```

```
v=0
o=Bla 12819370434213 12819370434215 IN IP4 192.168.9.86
s=Bla Audio call
c=IN IP4 192.168.9.86
t=0 0
m=audio 2487 RTP/AVP 8 0 2 3 97 110 101
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:2 G726-32/8000
a=rtpmap:3 GSM/8000
a=rtpmap:97 iLBC/8000
a=rtpmap:110 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
```

100 Trying:

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 192.168.9.86:80
From: "KLIJENT A" <sip:100@192.168.9.86:80>
To: "KLIJENT B" <sip:101@192.168.9.100>
Call-ID: DkyMDY3Y2FkMWQxMGQ0MGM2YWU2MmZhNmUzYWZjZWU.
CSeq: 1 INVITE
Contact: <sip:101@192.168.9.100>
Max-Forwards: 70
Content-Length: 0
```

180 Ringing:

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 192.168.9.86:80
From: "KLIJENT A" <sip:100@192.168.9.86:80>
To: "KLIJENT" <sip:101@192.168.9.100>
Call-ID: DkyMDY3Y2FkMWQxMGQ0MGM2YWU2MmZhNmUzYWZjZWU.
CSeq: 1 INVITE
Contact: <sip:101@192.168.9.100>
Max-Forwards: 70
Content-Length: 0

200 OK:

SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.9.86:80
From: "KLIJENT A" <sip:100@192.168.9.86:80>
To: "KLIJENT B" <sip:101@192.168.9.100>
Call-ID: DkyMDY3Y2FkMWQxMGQ0MGM2YWU2MmZhNmUzYWZjZWU.
CSeq: 1 INVITE
Contact: <sip:101@192.168.9.100>
Max-Forwards: 70
Content-Type: application/sdp
Content-Length: 322

v=0
o=- 3383885447 3383885447 IN IP4 192.168.9.100
c=IN IP4 192.168.9.100
t=0 0
m=audio 5062 RTP/AVP 8 0 2 3 97 110 101
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:110 speex/8000
a=rtpmap:101 telephone-event/8000

ACK:

sip:101@192.168.9.100 SIP/2.0
Via: SIP/2.0/UDP 192.168.9.86:80
Max-Forwards: 70
Contact: <sip:100@192.168.9.86:80>
To: "KLIJENT B" <sip:101@192.168.9.100>
From: "KLIJENT A" <sip:100@192.168.9.86:80>
Call-ID: DkyMDY3Y2FkMWQxMGQ0MGM2YWU2MmZhNmUzYWZjZWU.
CSeq: 1 ACK
Content-Length: 0

CANCEL:

CANCEL sip:100@192.168.9.86:80 SIP/2.0
Via: SIP/2.0/TCP 192.168.9.100
From: "KLIJENT B" <sip:101@192.168.9.100>
To: "KLIJENT A" <sip:100@192.168.9.86:80>
Call-ID: DkyMDY3Y2FkMWQxMGQ0MGM2YWU2MmZhNmUzYWZjZWU.
CSeq: 2 CANCEL
Max-Forwards: 70
Content-Length: 0

BYE:

BYE sip:100@192.168.9.86:80 SIP/2.0
Via: SIP/2.0/TCP 192.168.9.100
From: "KLIJENT B" <sip:101@192.168.9.100>
To: "KLIJENT A" <sip:100@192.168.9.86:80>
Call-ID: DkyMDY3Y2FkMWQxMGQ0MGM2YWU2MmZhNmUzYWZjZWU.
CSeq: 2 BYE
Max-Forwards: 70
Content-Length: 0

2.3. Kompresija govora

Prilikom VoIP komunikacije prenosi se digitalizovan audio signal, odnosno govor. S obzirom na to da se govor prenosi digitalizovan između strana u komunikacionoj sesiji serversko-klijentskog sistema, taj govor može biti kompresovan. Kompresija govora je poželjna i opravdana iz nekoliko razloga, a to su:

- prvi razlog je taj da se prilikom prenosa podataka (u obliku paketa) kroz mrežu, mreža opterećuje. Kompresija podataka u ovom slučaju doprinosi smanjivanju opterećenja u mreži, jer se kompresijom podataka smanjuje veličina paketa koji se treba preneti kroz mrežu,
- drugi razlog je taj da se klijenti ne nalaze uvek u mrežama koje omogućavaju velike brzine protoka podataka. Podaci koji stižu do klijenta, u mrežama sa sporom brzinom prenosa podataka, su uglavnom zakasneli, što se kod govora reprodukuje neprestanim prekidima tišine i nerazumnošću onoga koji priča. Problem može rešiti kompresovanje podataka, tako da se podaci kompresuju na onu količinu podataka, čijim se prenosom u mreži male brzine prenosa podataka ne izaziva kašnjenje.

Kompresija podataka, odnosno govora, može se obaviti korišćenjem bilo kod kodeka, stoga je za kompresiju podataka u komunikaciji između strana serversko-klijentskog sistema izabran AMR kodek (objašnjenja vezana za AMR pogledati u literaturi [7. \[3\] i \[4\]](#)). Mnogobrojni su razlozi izbora ovog kodeka, a glavni razlog je taj što je to kodek koji podržava osam različitih brzina kodovanja sa bitskim protocima (engl. *bit rate*) u rasponu od 4.75 kbit/s do 12.2 kbit/s, što znači da brzina protoka podataka u mreži prilikom prenosa govora mora biti veća od 4.75 kbit/s (kako se govor prenosi putem nekog protokola, uglavnom UDP, stoga bi se prilikom određivanja minimalne neophodne brzine protoka podataka u mreži trebalo uzeti u obzir i veličina zaglavlja koje protokol pridružuje prilikom pakovanja podataka u pakete koji se onda tako zapakovani prenose kroz mrežu).

AMR kodek je razvijan od strane 3GPP (engl. *3rd Generation Partnership Project*) organizacije i 1999 godine standardizovan je za GSM. Osim u GSM sistema, njegova primena je veoma značajna i u 3G WCDMA sistemima (engl. *3rd Generation Wideband Code Division Multiple Access System-s*). Standardizovani kodek zasnovan je na CELP (engl. *Code Excited Linear Prediction*) tehnologiji. U odnosu na prethodne GSM govorne kodeke, AMR pruža značajan napredak u rukovanju greškama, kao i otpornosti na greške, prilikom adaptacije govora i kodovanja kanala u zavisnosti od stanja u kom se kanal nalazi. Međutim, danas se u nedovoljnoj meri iskorišćavaju mogućnosti AMR kodeka, bar što se tiče njegove sposobnosti da podržava različite brzine bitskog protoka (u opsegu 4.75 - 12.2 kbit/s).

Prilikom adaptacije prenosa audio signala, odnosno govora, u mreži, AMR kodek omogućava izbor jednog od osam modova, gde je izbor zasnovan na kompromisu između srednje vrednosti brzine bitskog protoka i kvaliteta signala govora.

Ukoliko se posmatra GPRS mreža, gde dolazi do oscilacija u brzini prenosa podataka, to jest brzina protoka podataka u mreži se smanjuje, odnosno povećava, tokom određenog posmatranog vremenskog perioda, tako za slučaj kada brzina protoka podataka u mreži opada, žrtvuje se kvalitet signala govora, a kao rezultat toga smanjuje se kašnjenje podataka, jer se smanjivanjem kvaliteta govora smanjuje i veličinu paketa koji se prenosi u mreži. Isto tako za slučaj kada brzina protoka podataka u mreži raste, kvalitet govora se povećava srazmerno veličini paketa koji se treba preneti kroz mrežu, tako da se ne izazove kašnjenje podataka u mreži, odnosno veličinom paketa se ne sme zagušiti mreža.

Karakteristike koje AMR kodek poseduje su:

- frekvencija odabiranja (engl. *sampling frequency*) 8 kHz/13bit, 160 odbiraka (engl. *sample*) za okvir (engl. *frame*) od 20 ms, filtrirana u opsegu 0.2-3.4 kHz,
- koristi 8 stepena kompresije sa bitskim protocima 12.2, 10.2, 7.95, 7.40, 6.70, 5.90, 5.15 i 4.75 kbit/s,
- generiše dužinu okvira od 95, 103, 118, 134, 148, 159, 204, ili 244 bita za bitske protoke 4.75, 5.15, 5.90, 6.70, 7.40, 7.95, 10.2, ili 12.2 kbit/s, respektivno,
- podržava DTX (engl. *Discontinuous Transmission*), VAD (engl. *Voice Activity Detection*) i CNG (engl. *Comfort Noise Generation*); smanjuje prenos podataka u mreži tokom perioda tišine,
- algoritamsko kašnjenje je 20 ms po okviru. Za bitski protok od 12.2 kbit/s nema algoritamskog “look-ahead” kašnjenja, dok za osale bitske protoke to kašnjenje iznosi 5 ms. Treba naglasiti da je 5ms “dummy” “look-ahead” kašnjenja uvedeno kako bi se omogućilo prebacivanje modova rada AMR kodeka, odnosno promena brzina bitskih protoka,
- hibridni je govorni koder koji koristi ACELP (engl. *Algebraic Code Excited Linear Prediction*),
- kompleksnost algoritma je ocenjena ocenom 5, pri čemu je korišćenja relativna skala gde je G.711 kodek ocenjen sa 1, a G.729 kodek sa 15,
- Rezultati PSQM testiranja (engl. *Perceptual Speech Quality Measure testing*) u idealnim uslovima, izraženi u MOS (engl. *Mean Opinion Scores*) vrednostima, za AMR (12.2 kbit/s) iznose 4.14, dok za G.711 (u-law) iznose 4.45,
- Rezultati PSQM testiranja u uslovima stresa u okviru mreže, izraženi MOS vrednostima, za AMR (12.2 kbit/s) iznose 3.79, dok za G.711 (u-law) iznose 4.13.

AMR se, takođe, može koristiti u formi datoteke namenje za čuvanje audio signala, odnosno govora, kompresovanog AMR kodekom. Mnogi moderni mobilni telefoni dopuštaju čuvanje kratkih govornih zapisa u AMR formatu, a neki komercijalni programi pružaju mogućnost konverzije između AMR formata i nekog drugog formata, na primer MP3, pri čemu ne treba zaboraviti da je AMR govorni format i on neće davati idealne rezultate za druge formate. Uobičajna ekstenzija datoteke za čuvanje AMR kompresovanog govora je amr.

U stvarnosti AMR kodek omogućava četrnaest različitih modova (tabela 2.3-1), osam podržanih FR kanalu (engl. *full rate channel*) i 6 podržanih na HR kanalu (engl. *half rate channel*).

Tabela 2.3-1: Modovi podržani AMR kodekom

Mod	kbit/s	Kanal	Kompatibilno sa mrežom
AMR_12.20	12.2	FR	ETSI GSM unapređen FR
AMR_10.20	10.2	FR	
AMR_7.95	7.95	FR/HR	
AMR_7.40	7.4	FR/HR	TIA/EIA IS-641 TDMA unapređen FR
AMR_6.70	6.7	FR/HR	ARIB 6.7 kbit/s unapređen FR
AMR_5.90	5.9	FR/HR	
AMR_5.15	5.15	FR/HR	
AMR_4.75	4.75	FR/HR	
AMR_SID	1.8	FR/HR	

3. SERVERSKO-KLIJENTSKI SISTEM - OPIS REALIZACIJE I UPOTREBA

Osnovna svrha serversko-klijentskog sistema je uspostava veze i sama komunikacija između servera i klijenata. Server, odnosno serverska aplikacija, smešten je na uređaju čije performanse su daleko iznad prosečnih, što je opravdano s obzirom da server mora obrađivati ogromnu količinu podataka, a da se njegovo prisustvo u mreži ne primeti, dok klijent, odnosno klijentska aplikacija, može biti smešten na uređajima prosečnih performansi (prosečna performansa podrazumeva onu performansu uređaja koju danas poseduje jedan mobilni uređaj, koji se može kupiti na tržištu). Takođe, osim performanse uređaja, neophodno je obezbediti i adekvatnu brzinu prenosa podataka kroz mrežni priključak (engl. *network bandwidth*), jer iako su performanse uređaja, na kome je pokrenuta, bilo serverska ili klijentska, aplikacija, zadovoljavajuće, komunikacija između klijenata i servera u sistemu, zbog male brzine prenosa podataka u mreži, i dalje može biti loša, što se dalje odražava i na lošu komunikaciju i između samih klijenata, koji posredstvom servera komuniciraju.

Da bi serversko-klijentski sistem prikazao svoju minimalnu funkcionalnost, neophodno je postojanje jednog servera i dva klijenta, tako da će ovim radom biti prikazana minimalna funkcionalnost serversko-klijentskog sistema. Međutim, serversko-klijentski sistem može sadržati više od jednog servera, pri čemu bi serveri morali međusobno da komuniciraju za slučaj ako neki klijent, registrovan na serveru "A", želi da ostvari komunikaciju sa nekim klijentom, registrovanim na serveru "B", ali, takođe, serversko-klijentski sistem može da ima više od dva klijenta, bilo da je više klijenata prijavljeno na jednom serveru ili da je više klijenta prijavljeno na dva ili više različitih servera.

U ovom poglavlju će biti prikazan način realizovanja i upotrebe serverske i klijentske aplikacije serversko-klijentskog sistema.

3.1. Server

Server predstavlja uređaj na kojem se izvršava serverska aplikacija. Uloga servera u serversko-klijentskom sistemu je posredovanje prilikom obavljanja **klijent-server** i **klijent-server-klijent** komunikacije (pogledati poglavlje [2.1](#)).

3.1.1. Opis arhitekture

Aplikacija servera se, zbog zahteva za performansama uređaja daleko iznad prosečnih, izvršava na PC uređajima čije performanse su određene količini zahteva koje upućuju klijenti, kao i samom broju klijenata. Osim performanse, mrežni priključak na kome se nalazi server mora obezbeđivati veliku brzinu prenosa podataka. Takođe, IP adresa servera u mreži mora biti javna IP adresa (engl. *public IP address*), odnosno da je omogućen pristup serverskom PC uređaju od strane svih klijenata u mreži.

Nakon što se aplikacija servera pokrene i izvršavanje na PC uređaju, taj uređaj se proglašava serverom. Server sluša, odnosno prima zahteve od klijenata, na prolazu 80, pri čemu ovaj prolaz može biti promenjen ukoliko se to zahteva. Pokrenuti server čeka zahteve od klijenata.

Kada server primi od neregistrovanog klijenta **zahtev za registraciju klijenta na server**, taj klijent se upisuje u listu klijenata. Lista klijenata služi za evidenciju karakteristika svih registrovanih klijenata na serveru. Kao ogroman problem nameće se mogućnost klijenta da dodeli sam sebi korisničko ime, koje se kao jedna od karakteristika čuva u **listi klijenata** (o listi klijenata i karakteristikama klijenata koje ona čuva biće više reči u poglavlju [3.1.3.3](#)), tako da može doći do pojave dva ili više klijenta sa istim korisničkim imenom, a to bi dovelo do konfuzije, ne toliko na serveru, jer on u listi klijenata, sem korisničkog imena, čuva i druge karakteristike klijenata po kojima bi se klijent mogao odrediti, koliko bi isto korisničko ime klijenta izazvalo konfuziju među samim klijentima, jer nikad zaista klijent ne bi znao da li je sa druge strane veze pozvao pravog klijenta ili ne, pa se stoga, kao jedini način razrešavanja ovog problema, na serveru uvodi zabrana registrovanja dva ili više klijenata sa istim korisničkim imenom, odnosno korisničko ime klijenta mora biti jedinstveno.

Kada server primi **zahtev za uspostavu komunikacione sesije između klijenata**, tada server iz ovog zahteva parsira informacije koje jednoznačno određuju klijenta koji je uputio zahtev, odnosno **klijenta A**, i informacije koje jednoznačno određuju klijenta sa kojim klijent **A** želi uspostaviti komunikacionu sesiju, odnosno **klijenta B**. Nakon provere ispravnosti informacija o klijentima, **A** i **B**, server upućuje zahtev za uspostavom veze klijentu **B** pri čemu on odgovara serveru. Kad server primi pozitivan odgovor od klijenta **B**, on šalje klijentu **A** poruku da je zahtevana uspostava komunikacione sesije uspešno obavljena, i da komunikacija između klijenata **A** i **B** može započeti. U slučaju neprimljenog odgovora od klijenta **B**, čak iako server ne šalje ništa klijentu **A**, klijent **A** ima vremenski brojač (engl. *timer*), podešen na određeno vreme. Kad to vreme istekne, klijent **A** će znati da uspostava komunikacione sesije nije bila uspešna. **Zahtev za uspostavu komunikacione sesije između klijenata** nije nužan ukoliko se klijenti imaju javne IP adrese u mreži u kojoj se nalaze, ali kako to većinom nije slučaj, odnosno često se klijenti nalaze “sakriveni” iza nekog zastupničkog servera (engl. *proxy server*) ili iza NAT servera (engl. *Network Address Translation Server*) ili su “skriveni” na neki drugi način, to je bio glavni razlog uvođenja pomenutog zahteva. Pored ovog problema, što klijent u većini slučajeva nema javnu IP adresu, tu je i problem što se klijent u većini mreža ne može ponašati kao server, odnosno zabranjeno mu je slušanje (engl. *listen-ing*) na prolazu od strane mrežnog operatera, a samim tim i prihvatanje zahteva za vezom (engl. *accept-ing connections*) sa mreže.

Nakon što se komunikaciona sesija uspešno uspostavi, klijenti imaju mogućnost komuniciranja, putem teksta, odnosno mogu da razmenjuju tekstualne poruke, kao i mogućnost razgovora, u realnom vremenu. Treba napomenuti da sva pregovaranja u komunikaciji, posredstvom servera, između klijenata, vrše se korišćenjem SIP protokola. Server nije zadužen za bilo kakvu obradu poruka SIP protokola, već njegov zadatak čini samo prosleđivanje poruka SIP protokola ka odgovarajućim klijentskim stranama u komunikacionoj sesiji (SIP protokol je detaljnije objašnjen u poglavlju [2.2.6](#)).

3.1.2. Opis platforme i razvojnog okruženja

Za pisanje programskog koda serverske aplikacije izabrani su programski jezici C i C++. Mnogobrojni su razlozi izbora programskih jezika C i C++, od kojih su najznačajniji sledeći razlozi:

- programi napisani na programskim jezicima C i C++ se znatno brže izvršavaju u odnosu na ostale programske jezike (Java, C#, Pascal, VB, ...),
- programski kod može biti preveden (engl. *compile*) za izvršavanje na mnogobrojnim kompjuterskim platformama koje rade pod različitim operativnim sistemima (među kojima su najznačajniji Linux i Windows), pri čemu su izmene u programskom kodu male, ili ih uopšte nema.
- velika je rasprostranjenost C i C++ programskih jezika, odnosno njihovo korišćenje prilikom realizacije funkcionalnosti raznih tehničkih proizvoda.

Platforma na kojoj se izvršava serverska aplikacija, **Server_PC-W_XP**, je PC koji radi pod operativnim sistemom *Windows XP*. Za pisanje i prevođenje programskog koda serverske aplikacije korišćeno je programsko okruženje *Microsoft Visual C++ 6.0*. S obzirom na to da se radi o serverskoj aplikaciji, korisnički interfejs realizovan je u obliku Win32 konzolne aplikacije.

3.1.3. Modularnost

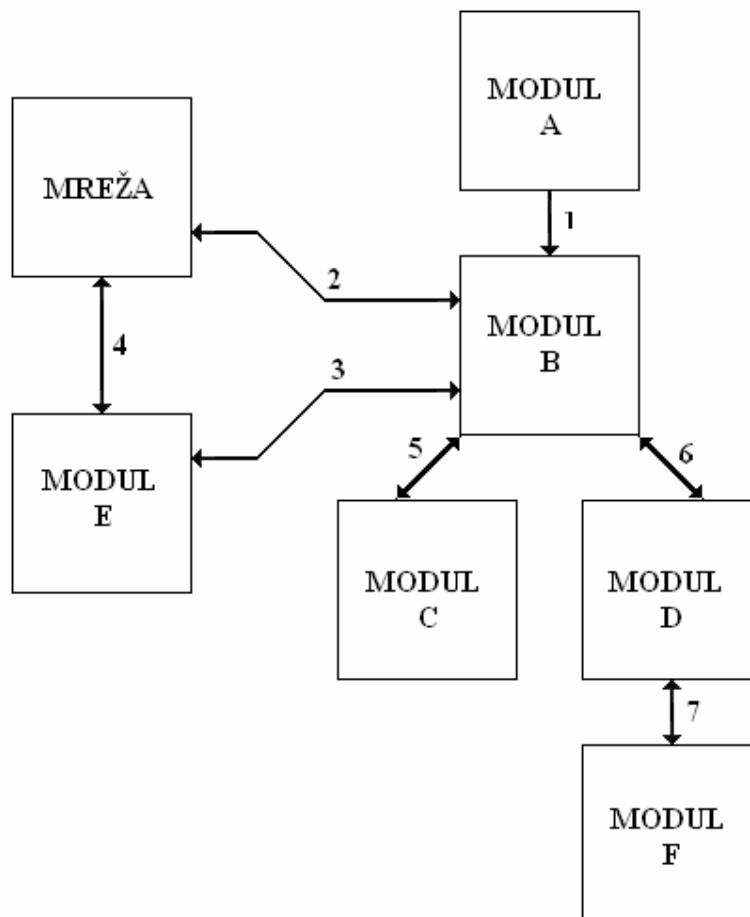
Programski kod serverske aplikacije organizovan je u module. Postoji šest modula serverske aplikacije, a to su:

- **A modul** - modul zadužen za startovanje i obustavu rada serverske aplikacije,
- **B modul** - modul zadužen za upravljanje modulima **C**, **D**, **E**, za upravljanje mrežom korišćenjem mrežnih protokola i razmenu poruka sa mrežom,
- **C modul** - modul zadužen za upravljanje serverovom **listom klijenata**,
- **D modul** - modul zadužen za obradu SIP poruka dobijenih od **B modula**. Tokom obrade, za parsiranje SIP poruka, **D modul** koristi usluge **F modula**,
- **E modul** - modul zadužen za komunikaciju u realnom vremenu,
- **F modul** - modul zadužen za parsiranje SIP poruka. On sam predstavlja SIP parser. Za njega se može reći da je pomoćni modul **modula D** i da predstavlja produžetak **modula D**.

Na slici 3.1.3-1 prikazano je svih šest modula serverske aplikacije, gde **B modul** predstavlja jezgro servera. Takođe, na slici je prikazano obavljanje razmene poruka između samih modula, ali i obavljanje razmene poruka između modula i mreže.

Svaki od modula (osim **modula A**) sastoji se iz dva dela:

- 1) zaglavlja, čiji sadržaj se nalazi u *.h datotekama, i
- 2) izvršnog dela, čiji sadržaj se nalazi u *.cpp datotekama.



SIGNALI		
1	→ B	SERVER - START
	→ B	SERVER - STOP
2	→ B	TCP (REGISTRACIJA, SIP, CHAT)
	← B	
3	← B	ZAHTEV ZA UDP RAZGOVOROM
	→ B	UDP RAZGOVOR ZAVRŠEN
4	← E	UDP RAZGOVOR
	→ E	
5	← B	ZAHTEV ZA LISTU KLIJENATA
	→ B	LISTA KLIJENATA
6	← B	ZAHTEV ZA PARSIRANJE PORUKE
	→ B	ISPARSIRANA PORUKA
7	← D	ZAHTEV ZA PARSIRANJE SIP PORUKE
	→ D	ISPARSIRANA SIP PORUKA

Slika 3.1.3-1: Moduli serverske aplikacije

3.1.3.1. Modul A - Startovanje i obustava rada serverske aplikacije

Prvi modul koji se aktivira na početku rada servera je **modul A**, jer se prilikom startovanja serverske aplikacije prvo poziva *main()* funkcija koja je sastavni deo ovog modula. Realizacija implementacije **modula A** nalazi se u datoteci *main.cpp*.

Zadaci modula A su:

- startovanje rada jezgra servera, odnosno **modula B**,
- obustava rada jezgra servera, ali i same serverske aplikacije,
- ispis glavnog menija servera i poziv odgovarajuće funkcije na izbor neke od opcija iz glavnog menija.

U nastavku sledi opis rada **modula A**.

Kao što je već rečeno, prilikom startovanja serverske aplikacije poziva se *main()* funkcija, kojoj je neophodno proslediti jedan parametar, a to je serverski prolaz, odnosno broj prolaza na kome server sluša. Kada *main()* funkcija dobije parametar, vrši se startovanje **modula B**, ispis glavnog menija, i zatim, **modul A** prelazi u stanje čekanja, i čeka na izbor jedne opcije iz glavnog menija.

Startovanje **modula B**, odnosno jezgra servera, vrši se instanciranjem objekta *net* klase *Network* (objašnjenje klase *Network* nalazi se u poglavlju [3.1.3.2.](#)), i to na sledeći način:

Network net(server_port);

Prilikom instanciranja objekta *net* konstruktoru klase *Network* prosleđuje se serverski prolaz, odnosno isti onaj prolaz koji je kao parametar prosleđen funkciji *main()*. Nakon što je instanciran objekat *net*, **modul B** je startovan i on se izvršava paralelno sa **modulom A**. U stvari modul B je programska nit. U ovom trenutku na serverskom uređaju postoje dve niti, nit glavnog programa - **modula A**, i nit serverskog jezgra - **modula B**.

Ispis glavnog menija vrši se pozivom funkcije *MainMenuDisplay()*, nakon čijeg ispisa **modul A** prelazi u stanje čekanja, gde čeka na izbor opcije iz glavnog menija.

Opcije dostupne serverskom administratoru u svakom trenutku rada servera su:

- **L** - ispis **liste klijenata** na konzoli serverske aplikacije,
- **A** - dodavanje novog korisnika u **listu klijenata**,
- **U** - uklanjanje klijenta iz **liste klijenata** na osnovu klijentskog **UID**-a,
- **F** - uklanjanje svih klijenata iz **liste klijenata**,
- **C** - brisanje konzole serverske aplikacije i ponovni ispis glavnog menija,
- **M** - ispis glavnog menija bez brisanja konzole serverske aplikacije,
- **Q** - kraj izvršavanja programa serverske aplikacije.

Izborom opcija **L**, **A**, **U**, **F** pristupa se **listi klijenata**. Da bi **modul A** pristupio listi klijenata, on obaveštava **modul B** da mu je neophodan rad sa listom klijenata. Zatim, **modul B** dobavlja **listu klijenata** od **modula C** i prosleđuje je **modulu A**. Nakon prosleđene **liste klijenata**, **modul B** izostavlja sebe iz dalje komunikacije, tako da ostavlja **modul A** i **C** da komuniciraju samostalno. Dakle, nakon dobijene **liste klijenata** od **modula B**, **modul A** koristi usluge **modula C**: za ispisivanje liste klijenata, dodavanje novih klijenata u listu klijenata i uklanjanje klijenata iz liste klijenata. Usluge **modula C**, odnosno njegove funkcije, **Modul A** koristi za implementaciju svojih funkcija, s ciljem da koristi samo ono što je njemu (**modulu A**) zaista potrebno.

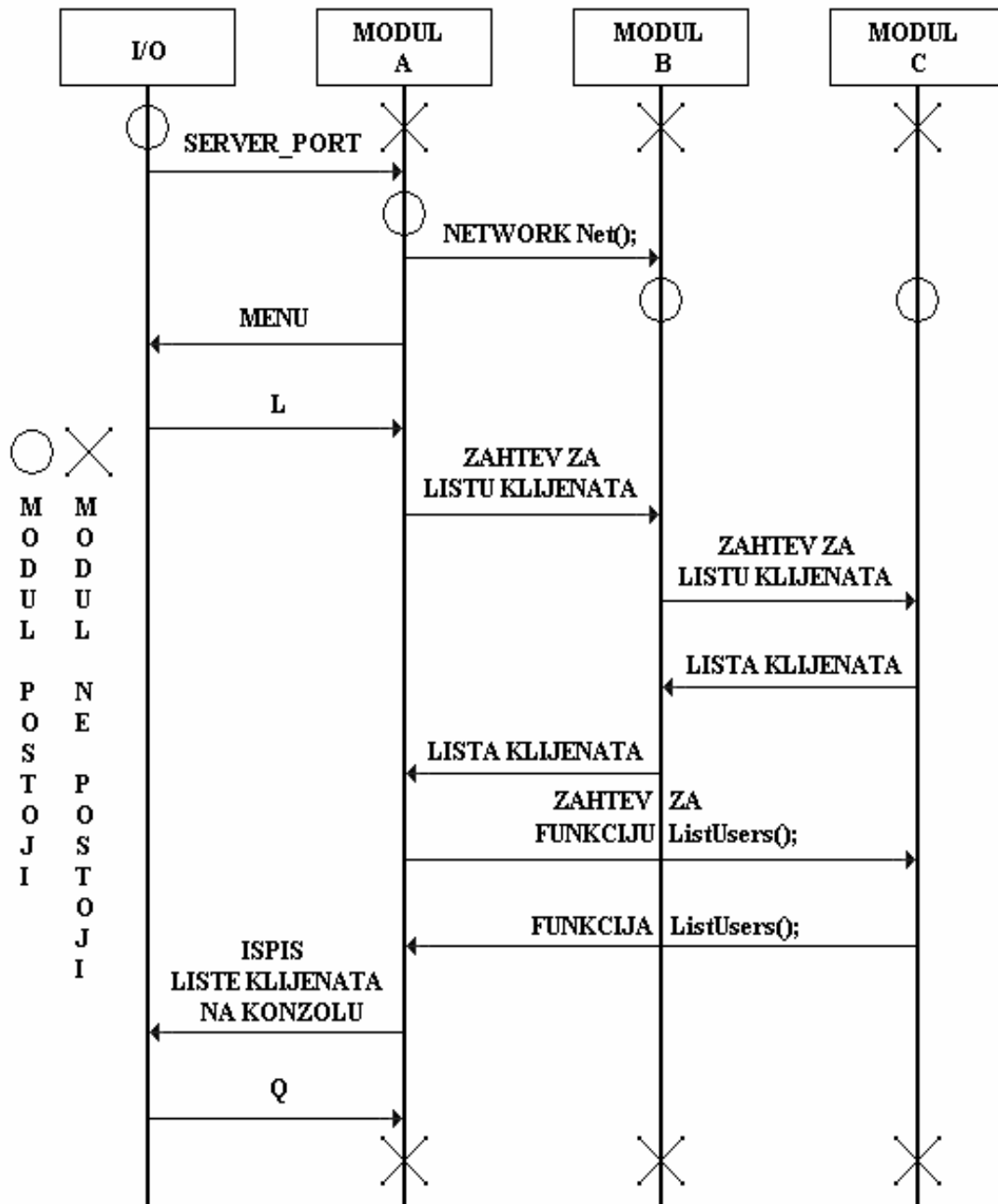
U nastavku sledi spisak funkcija **modula A** koje odgovaraju opcijama **L, A, U, F, C, M, Q** iz glavnog menija:

- **L** - *ListUsers()*, funkcija kojom se ispisuje **lista klijenata** na konzolu serverske aplikacije. Kao parametar funkcije prosleđuje se **lista klijenata** koja je prethodno dobijena od **modula B**,
- **A** - *AddUser()*, funkcija kojom se dodaje novi korisnik u **listu klijenata**. Kao parametar funkcije prosleđuje se **lista klijenata** koja je prethodno dobijena od **modula B**,
- **U** - *RemoveUser()*, funkcija kojom se uklanja jedan klijent iz **liste klijenta**, gde se klijent iz **liste klijenata** uklanja na osnovu **UID**-a. Kao parametar funkcije prosleđuje se **lista klijenata** koja je prethodno dobijena od **modula B**,
- **F** - *RemoveAll()*, funkcija kojom se uklanjaju svi klijenti iz **liste klijenata**. Kao parametar funkcije prosleđuje se **lista klijenata** koja je prethodno dobijena od **modula B**,
- **C** - *MainMenuDisplay(1)*, funkcija kojom se brše konzola serverske aplikacije i ispisuje glavni meni. Kao parametar funkcije prosleđuje se **1**, kojim se funkciji specificira da izvrši brisanje konzole servera pre ispisa glavnog menija,
- **M** - *MainMenuDisplay(0)*, funkcija kojom se se briše konzola serverske aplikacije i ispisuje glavni meni. Kao parametar funkcije prosleđuje se **0**, kojim se funkciji specificira da izvrši samo ispis glavnog menija bez prethodnog brisanja.
- **Q** - ne poziva se nijedna funkcija. Izborom opcije **Q** završava se *main()* funkcija (pri čemu je to regularan završetak).

Završetkom *main()* funkcije (regularnim) dolazi do obustave rada serverske aplikacije, što podrazumeva i obustavu rada svih njenih modula.

Slikom 3.1.3.1-1 prikazan je rad modula A, i to:

- početak rada serverske aplikacije - prosleđivanje serverskog prolaza **modulu A, SERVER_PORT**, može se smatrati početkom rada serverske aplikacije, jer je ovaj parametar očekivan od strane *main()* funkcije,
- rad **modula A** sa **listom klijenata** - kada se iz glavnog menija bira opcija **L**, čime se zahteva ispis **liste klijenata** na konzoli serverske aplikacije,
- završetak rada serverske aplikacije - kada se iz glavnog menija bira opcija **Q** kojom se završava *main()* funkcija, a samim tim i rad **modula A, B i C**, što je na slici označeno sa **X** (O – modul je prisutan i radi X – modul nije prisutan i ne radi).



Slika 3.1.3.1-1: Rad modula A

3.1.3.2. Modul B - Mrežni protokoli i upravljanje modulima C, D, E

Modul B predstavlja jezgro serverske aplikacije. Realizacija implementacije **modula B** nalazi se u datotekama **network.cpp** i **network.h**.

Realizacija **modula B** predstavljena je klasom *Network*, koja je definisana u **network.h** (na slici 3.1.3.2-1 prikazana je klasa *Network*).

Zadaci **modula B** su:

- upravljanje **modulom C**, kada **modul B** zahteva rad sa **listom klijenata**,
- upravljanje **modulom D**, kada **modul B** zahteva obradu SIP poruka primljenih sa mreže,
- upravljanje **modulom E**, kada **modul B** zahteva ostvarivanje komunikacije sa mrežom sprovođenjem UDP protokola,
- ostvarivanje komunikacije sa mrežom sprovođenjem TCP protokola.

U nastavku sledi opis rada **modula B**.

Pozivom konstruktora klase *Network* vrši se inicijalizacija serverskog **modula B**, pri čemu:

- se poziva funkcija *WSAStartup()*, kojom se startuje *Winsock* biblioteka *WS2_32.dll*, odnosno specificira se da će za rad sa mrežom biti korišćena *Winsock* biblioteka *WS2_32.dll*,
- se kreira TCP utičnica (engl. *socket*) (kreiranje se vrši pozivom funkcije *socket()*) i povezuje sa IP adresom i prolazom servera (povezivanje se vrši pozivom funkcije *bind()*),
- se startuje rad **modula E** što se čini pozivom funkcije *InitializeUDPServer()* (ova funkcija je objašnjena u poglavlju [3.1.3.5](#)). Startovanjem **modula E** vrši se inicijalizacija serverskog dela, koji komunicira sa mrežom putem UDP protokola,
- se poziva funkcija *listen()*, kojom se omogućava serveru da sluša na kreiranoj TCP utičnici.
- se startuje nit *NetAcceptThread()*, kojom se čekaju zahtevi za uspostavom TCP veze, koje šalju klijenti prilikom prijave na server. Ovi zahtevi se primaju na utičnici na kojoj server sluša.
- se kreiraju semafori *RefreshUserList*, čiji je broj jednak maksimalnom broju klijenata koji server može uslužiti. Jednim semaforom *RefreshUserList* se kontroliše rad jedne niti *RefreshUserList()*,
- se startuje rad **modula C** radi preuzimanja **liste klijenata**. Preuzimanje **liste klijenata** vrši se pozivom funkcije *DB.GetDataBasePointer()*, (*DB* je objekat klase *DataBase*, a klasa je definisana u **modulu C**, pogledati u poglavlju [3.1.3.3](#)). Ova funkcija vraća memorijsku adresu početka **liste klijenata**, za čije čuvanje je zadužen pokazivač *base* (memorijska adresa početka liste uzima se samo jednom i to sledećom linijom programskog koda pozvanom u konstruktoru: *base = DB.GetDataBasePointer()*).

Praktično, inicijalizacija **modula B** predstavlja i inicijalizaciju samog servera, jer se tada u stvari vrši proglašavanje utičnice na kojoj se sluša, odnosno tek od tada server može prihvatati zahteve za konekcijom od klijenata.

Ukoliko inicijalizacija **modula B** nije bila uspešna tada dolazi do zaustavljanja rada serverske aplikacije, pri čemu se administrator servera obaveštava o tome.

Nakon uspešne inicijalizacije **modula B** startovana je nit *NetAcceptThread()*, koja postoji za sve vreme rada serverske aplikacije, odnosno nit se završava tek kad se završi program serverske aplikacije. U okviru niti *NetAcceptThread()* poziva se funkcija *accept()*, kojom se blokira rad niti dok se ne primi **zahtev za konektovanje** (to je zahtev koji se šalje pozivom *Winsock* funkcije *connect()*). U serversko-klijentskom sistemu klijentska strana vrši poziv funkciju *connect()*. Dobijanjem **zahteva za konektovanje**, blokirana nit *NetAcceptThread()* nastavlja svoj rad i stvara nit *NetReceiveThread()*. Nakon stvaranja niti *NetReceiveThread()*, nit *NetAcceptThread()* se opet blokira na funkciji *accept()* i čeka novi **zahtev za konektovanje**.

```

class Network {
private:
    //-- Winsock:
    WSADATA WsaDat;
    //--TCP:
    //TCP socket
    SOCKET Socket;
    //IP adresa, koja se povezuje sa TCP socketom
    SOCKADDR_IN SockAddr;
    //-- Parser:
    //podrzava funkcije za obradu SIP poruka
    TParser Parser;
    //-- Sesijska:
    //Broj trenutno ostvarenih sesija na serveru
    int nSessionsCount;
    //Semafor za signalizaciju osvezavanja liste korisnika
    HANDLE RefreshUserList;
public:
    //Konstruktor klase Network: inicijalizacija servera
    Network(int pPort);
    //Destruktor klase Network: obustava rada servera
    ~Network();
    //Slanje liste korisnika klijentu
    int ClientListResponse(SOCKET Client);
    //Signaliziranje refresh user liste svim klijentima
    void RefreshListToAll(int nUserCount);
    //Povećavanje brojacha trenutnih sesija za 1
    void IncrementSessionCounter();
    //Smanjenje brojacha trenutnih sesija za 1
    void DecrementSessionCounter();
    //TCP
    //prijateljska nit za prihvatanje zahteva za konekcijama
    //na server
    friend UINT NetAcceptThread(LPVOID pParam);
    //prijateljska nit za prihvatanje poruka sa mreze
    //od "source" klijenta (TCP)
    friend UINT NetReceiveThread(LPVOID pParam);
    //prijateljska nit za prihvatanje poruka sa mreze
    //od "destination" klijenta (TCP)
    friend UINT NetReceiveDestThread(LPVOID pParam);
    //prijateljska nit za slanje informacija o klijentima
    //na serveru, pri čemu se te informacije prosledjuju
    //svim klijentima na serveru
    friend UINT RefreshUserList(LPVOID pParam);
    //UDP
    //prijateljska nit za prihvatanje audio podataka sa mreze,
    //koji se prenose prilikom obavljanja razgovora u
    //realnom vremenu (UDP)
    friend UINT NetUDPReceiveThread(LPVOID pParam);
};

```

Slika 3.1.3.2-1: Klasa Network

Programski kod pseudo konstruktora klase *Network* prikazan je slikom 3.1.3.2-2.

```

Network::Network(int pPort)
{
    //Preuzimanje liste klijenata
    base = DB.GetDataBasePointer();
    //Anuliranje brojava sesija
    nSessionsCount = 0;
    //Inicijalizacija Winsock biblioteke WS2_32.dll
    if (WSAStartup(MAKEWORD(1, 1), &WsaDat) != 0)
    {
        //greska - zavrsetak serverskog programa
        exit(1);
    }
    //Kreiranje TCP socketa,
    //jer se prosledjuje parametar SOCK_STREAM
    Socket = socket(AF_INET, SOCK_STREAM, 0);
    if (Socket == INVALID_SOCKET)
    {
        //greska - zavrsetak serverskog programa
        exit(1);
    }
    SockAddr = InitilizeSockAddr();
    //Povezivanje TCP socketa sa IP adresom i portom*/
    if (bind(Socket,
              (SOCKADDR *)(&SockAddr),
              sizeof(SockAddr)
              ) == SOCKET_ERROR)
    {
        //greska - zavrsetak serverskog programa
        exit(1);
    }
    //Inicijalizacija serverskog dela,
    //koji komunicira sa mrezom putem UDP protokola
    if(!(error = InitializeUDPServer(server_ip,
                                     8983,
                                     &UDP_server_socket,
                                     &UDP_server
                                     )))
    {
        //greska - zavrsetak serverskog programa
        DeinitializeUDPServer(&UDP_server_socket);
        exit(1);
    }
    //Slusanje MAX dolaznih konekcija na kreiranom TCP socketu
    if (listen(Socket, MAX) != 0)
    {
        //greska - zavrsetak serverskog programa
        exit(1);
    }
    //Startovanje niti NetAcceptThread
    acceptThread = AfxBeginThread(NetAcceptThread, 0);
    //Startovanje semafora RefreshUserList
    RefreshUserList = CreateSemaphore(0,0,MAX+(MAX/2),0);
}

```

Slika 3.1.3.2-2: Konstruktor klase *Network*

Na svaki dobijeni **zahtev za konektovanje** stvara se po jedna nit *NetReceiveThread()*. Za svakog klijenta server dobija dva zahteva za konektovanje:

- prvi **zahtev za konektovanje** se dobija kada klijenta zahteva registrovanje na server,
- drugi **zahvet za konektovanje** se dobija kada klijent zahteva uspostavu komunikacione sesije sa drugim klijentom.

Dobijanjem **zahteva za konektovanje** stvaraju se TCP kanali komunikacije između klijenata i servera (pogledati poglavlje [2.2.1.](#)):

- prvim **zahtevom za konektovanje** stvara se **TCP kanal kontrole**,
- drugim **zahvetom za konektovanje** stvara se **TCP kanal komunikacije**.

Uloga niti *NetReceiveThread()* je da primi i obradi podatke koji dolaze TCP kanalima sa mreže. U zavisnosti od vrste podataka koji su pristigli sa mreže zavisi njihova sama obrada. Na osnovu obrade podataka određuje se koja će biti sledeća akcija servera.

Za primanje podataka sa mreže, kontrolisane TCP protokolom, koristi se blokirajuća funkcija *Winsock* biblioteke *recv()*. Funkcija *recv()* je iskorišćena u realizaciji niti *NetReceiveThread()* da bi se blokirao njen rad do trenutka kad se na TCP utičnici ne pojave podaci, koji će daljim radom niti biti obrađivani.

U nastavku sledi opis **vrste podataka**, koji se primaju sa mreže, **načina obrade**, koji se primenjuju nad tim podacima, i **akcija**, koje se preduzimaju na osnovu obrada. Generalno, sledi opis rada niti *NetReceiveThread()*. Kako se na serveru za svakog klijenta stvaraju po dve niti *NetReceiveThread()*, tako ćemo:

- nit stvorenu za prijem onih podataka na obradu koji dolaze **TCP kanalom kontrole** nazvati *NetReceiveThread()*¹, a
- nit stvorenu za prijem onih podataka na obradu koji dolaze **TCP kanalom komunikacije** nazvati *NetReceiveThread()*².

Nit *NetReceiveThread()*¹ se koristi za sprovođenje **protokola registracije** klijenta na server, **protokola slanja spiska registrovanih klijenata** i **protokola uspostave komunikacione sesije između klijenata** (objašnjenje protokola nalazi se u poglavlju [2.2.](#)).

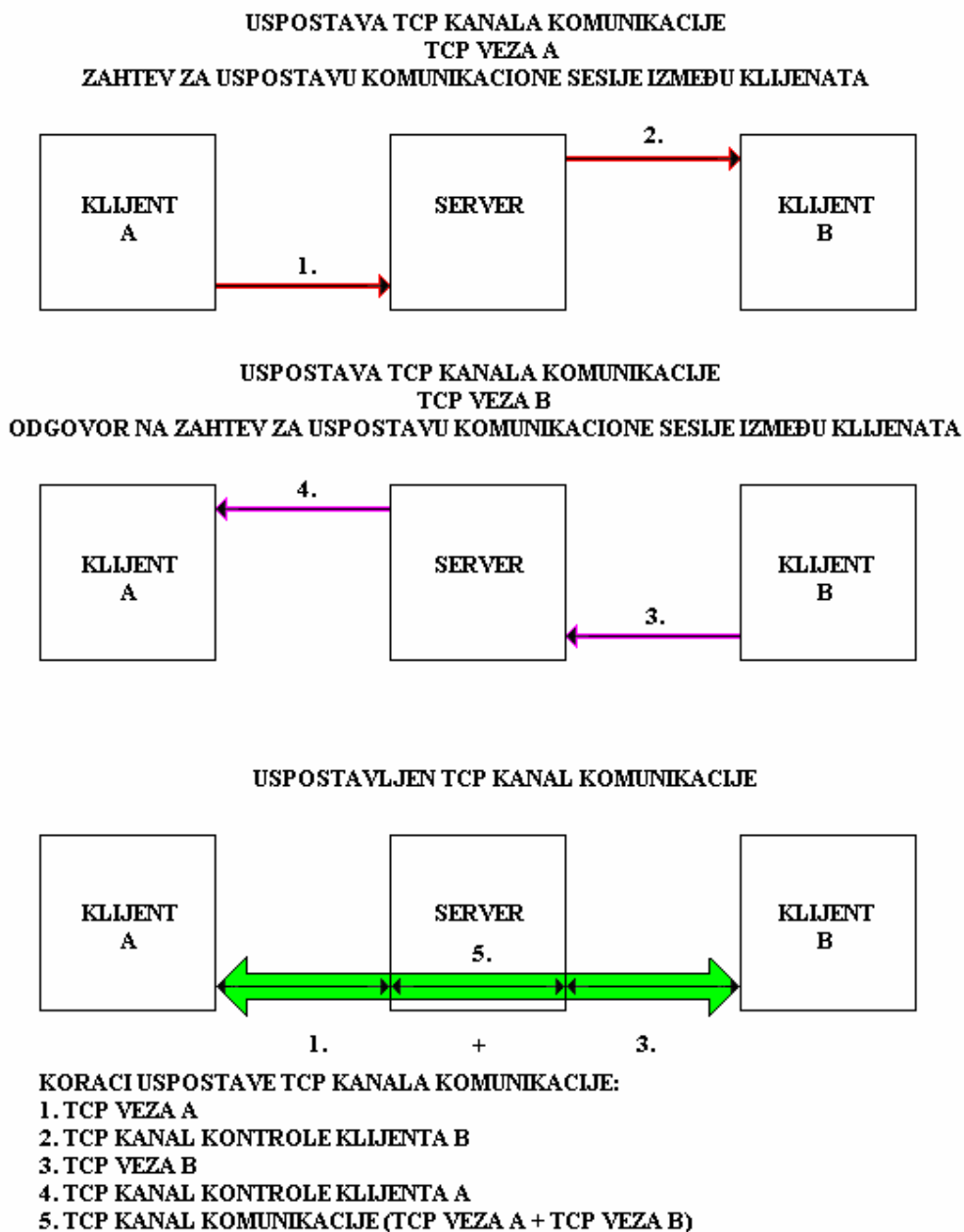
Sprovođenje protokola registracije zajedno sa sprovođenjem protokola slanja spiska registrovanih klijenata - nit *NetReceiveThread()*¹ prima sa **TCP kanala kontrole** zahtev za **registraciju klijenta na server**, i poziva funkciju *NetRegisterParser()* kojom se parsira primljeni zahtev. Nakon parsiranja zahteva, nit zahteva usluge za rad sa **listom klijenata** od **modula C** radi:

- proveriti da li je klijent sa istim korisničkim imenom već registrovan na serveru, što se obavlja pozivom funkcije *base->IsUserPresent()* (*base* je pokazivač na memorijsku adresu početka **liste klijenata**), i
- dodavanje klijenta u **listu klijenata**, što se obavlja pozivom funkcije *base->RegisterUser()* (dodavanje se obavlja samo ukoliko klijent ima jedinstveno korisničko ime).

Ukoliko se ne izvrši dodavanje klijenta u **listu klijenata**, registracija se smatra neuspešnom te se raskida **TCP kanal kontrole** sa klijentom uz prethodno obaveštavanje klijenta o nemogućnosti njegove registracije na server. Takođe, se uništava nit *NetReceiveThread()*¹.

Ukoliko se izvrši dodavanje klijenta u **listu klijenata**, registracija se smatra uspešnom te se klijentima sprovodi **protokol slanja spiska registrovanih klijenata**, a to se čini stvaranjem niti *RefreshUserList()*, koja na početku rada odlazi u čekanje na semafor *RefreshUserList*, zbog toga je neophodno nakon stvaranja niti *RefreshUserList()* pozvati funkciju *RefreshListToAll()*, kojom se aktivira onoliko semafora *RefreshUserList* koliko ima i klijenata na serveru (jer se za svakog klijenta stvara se po jedna nit *RefreshUserList()*), kako bi se signalizirale sve niti *RefreshUserList()*. Kada je nit *RefreshUserList()* signalizirana semaforom *RefreshUserList* nit izlazi iz stanja čekanja i prelazi u stanje slanja **spiska registrovanih klijenata**. Kada se završi slanje **spiska registrovanih klijenata**, nit *RefreshUserList()* prelazi u stanje ponovnog čekanja na semafor *RefreshUserList*.

Završetkom **protokola registracije** se smatra kada server pošalje svim klijentima **spisak registrovanih klijenata**.



Slika 3.1.3.2-3: Uspostava TCP kanal komunikacije

Kada je na serveru registrovano bar dva klijenta, tada bilo koji od klijenta može da zatraži od servera spajanje sa onim drugim klijentom. Prilikom spajanja dva klijenta sprovodi se **protokol uspostave komunikacione sesije između klijenata** i stvara se **TCP kanal komunikacije**.

Na slici 3.1.3.2-3 prikazano je uspešno uspostavljanje **TCP kanala komunikacije**, za šta je neophodno da se ostvare dve veze (slika 3.1.3.2-3):

- **TCP veza A** - veza između klijenta koji poziva i servera, odnosno klijenta **A** i servera,
- **TCP veza B** - veza između klijenta koji je pozivan i servera, odnosno klijenta **B** i servera.

Sprovođenje protokola uspostave komunikacione sesije između klijenata - da bi se iniciralo sprovođenje **protokola uspostave komunikacione sesije između klijenata**, klijent **A** upućuje **zahtev za konektovanje** serveru, pri čemu se stvara **TCP veza A**. Na primanje **zahteva za konektovanje** odblokira se serverska nit `NetAcceptThread()` koja stvara nit `NetReceiveThread()`^{2A} (značenje indeksa **2A** je: “2” označava da se radi o `NetReceiveThread()`² niti, a “A” označava da je ta nit stvorena na serveru za potrebe usluživanja klijenta **A**). Ubrzo nakon **zahteva za konektovanje**, putem **TCP veze A** klijent **A** šalje **zahtev za uspostavu komunikacione sesije između klijenata**. Nit `NetReceiveThread()`^{2A} primanjem ovog zahteva se odblokira i vrši parsiranje zahteva. Parsiranjem **zahteva za uspostavu komunikacione sesije između klijenata** dobijaju se **UID-i** klijenata između kojih je zatražena uspostava komunikacione sesije. Nakon dobijanja **UID-a**, nit `NetReceiveThread()`^{2A} poziva funkciju modula **C** `base->FindUserByUID(UID, &client)`, kojom se na osnovu **UID-a** obavlja pretraživanje **liste klijenata** kako bi se pronašao klijent čiji je **UID** isti kao zadati **UID**. S obzirom na to da su parsiranjem **zahteva za uspostavu komunikacione sesije između klijenata** dobijena dva **UID-a**, **lista klijenata** se pretražuje dva puta. Rezultati pretrage se smeštaju u objekte klase `User` (klasa `User` definisana je u **modulu C**, pogledati poglavlje [3.1.3.3.](#)), `local_user` i `destination_user`, gde objekat `local_user` sadrži informacije koje odgovaraju klijentu **A**, a objekat `destination_user` sadrži informacije koje odgovaraju klijentu **B**. Ovi objekti su definisani kao lokalni objekti niti `NetReceiveThread()`^{2A}. Kada su klijenti pronađeni, server vrši dopunu **liste klijenata** dodatnim podacima o klijentu **A** i putem **TCP kanala kontrole** prosleđuje klijentu **B** **zahtev za uspostavu komunikacione sesije između klijenata**. Nakon poslatog zahteva nit `NetReceiveThread()`^{2A} se blokira na funkciji `recv()`.

Server, pre **odgovora na zahtev za uspostavu komunikacione sesije između klijenata**, od klijenta **B** prima **zahtev za konektovanje**, pri čemu je stvorena **TCP veza B**. Ovim zahtevom se opet odblokira rad niti `NetAcceptThread()`, koja, dalje, stvara nit `NetReceiveThread()`^{2B}, gde se novostvorena nit blokira na funkciji `recv()`. Kada server putem **TCP veze B** primi očekivani odgovor od klijenta **B**, nit `NetReceiveThread()`^{2B} se odblokira i obavlja parsiranje **odgovora na zahtev za uspostavu komunikacione sesije između klijenata**. Iz parsiranog odgovora se dobijaju **UID-i** klijenata između kojih se uspostavlja komunikacija. Na osnovu **UID-a** klijenata se vrši pretraživanje liste klijenata pozivom funkcije `base->FindUserByUID(UID, &client)` (pretraživanje se vrši dva puta za različite klijentske **UID-e**). Rezultati pretrage upisuju se u lokalno definisane objekte niti `NetReceiveThread()`^{2B}, `local_user` i `destination_user`, gde objekat `local_user` sadrži informacije koje odgovaraju klijentu **B**, a objekat `destination_user` sadrži informacije koje odgovaraju klijentu **A**. Kada su klijenti pronađeni, server vrši dopunu **liste klijenata** dodatnim podacima o klijentu **B** i putem **TCP kanala kontrole** prosleđuje klijentu **A** **odgovor na zahtev za uspostavu komunikacione sesije između klijenata**. Nakon poslatog zahteva nit `NetReceiveThread()`^{2B} se blokira na funkciji `recv()`.

Završetkom **protokola uspostave komunikacione sesije između klijenata** se smatra trenutak kada server pošalje klijentu **A** **odgovor na zahtev za uspostavu komunikacione sesije između klijenata**.

Nakon što se **TCP kanal komunikacije** uspostavi između klijenata (klijenta **A** i **B**), klijenti mogu da započnu pregovaranje putem SIP protokola kako bi mogli da obavljaju razgovor u realnom vremenu ili da razmenjuju tekstualne poruke.

Kada server primi SIP poruku sa mreže **TCP kanalom komunikacije**, on je prosleđuje na obradu **modulu D**, i to čini pozivom funkcije *Parser.MsgParser()* (*Parser* je objekat klase *TParser* definisane u **modulu D**. Pogledati poglavlje [3.1.3.4.](#)). Nakon obrade SIP poruke, funkcija *Parser.MsgParser()* vraća odgovor koji server prosleđuje odgovarajućem klijentu, odnosno klijentu naznačenom u odgovoru.

Kada se završi sprovođenje SIP protokola, server stvara dve niti:

- nit *NetReceiveDestThread()* - nit za prihvatanje tekstualnih poruka od strane klijenta koji je pozivan,
- nit *NetUDPReceiveThread()* - za obavljanje razgovora u realnom vremenu, pri čemu je za to neophodno sprovođenje UDP protokola. Za ostvarivanje UDP protokola koriste se funkcije **modula E**.

Nakon što su stvorene niti *NetReceiveDestThread()* i *NetUDPReceiveThread()*, server postaje posrednik između klijenta, odnosno njegova jedina funkcija postaje prosleđivanje tekstualnih i/ili govornih poruka između klijenta **A** i klijenta **B**.

Završetkom komunikacije između klijenata, razgovora i/ili razmene tekstualnih poruka, server raskida **TCP kanal komunikacije** i uništava sledeće niti:

- nit *NetReceiveDestThread()*,
- nit *NetUDPReceiveThread()*,
- *NetReceiveThread()*^{2A}, i
- *NetReceiveThread()*^{2B}.

Prilikom odjave klijenta sa servera vrši se raskidanje TCP kanala kontrole sa tim, kao i uništavanje niti *NetReceiveThread()*¹ koja odgovara tom klijentu.

3.1.3.3. Modul C - Lista klijenata

Zadatak **modula C** je da obezbedi rad sa **listom klijenata**. Realizacija implementacije modula C definisana je u datotekama **registerDB.h** i **registerDB.cpp**.

Modul C definiše tri klase:

- klasu *User*,
- klasu *UserDB*, i
- klasu *DataBase*.

Klasom *User* (Slika 3.1.3.3-1) definiše se osnovni element **liste klijenata**, ali, takođe, i operacije koje se mogu obavljati nad tim elementom. Osnovini element **liste klijenata** je klijent, pa stoga objekti članovi klase *User* predstavljaju osnovne karakteristike koje opisuju jednog klijenta, a to su:

- *name* - ime klijenta,
- *IP* - klijentova IP adresa,
- *UID* - jedinstveni identifikacioni broj klijenta koji se dodeljuje svakom klijentu prilikom prijave na server,
- *ControlUserSocket* i *ControlSocketFrom* - utičnica **TCP kanala kontrole** i informacije o adresi koja je vezana za tu utičnicu,
- *UserSocket* i *SocketFrom* - utičnica **TCP kanala komunikacije** i informacije o adresi koja je vezana za tu utičnicu.

```

class User {
private:
    //ime korisnika
    char name[40];
    //IP korisnika
    char IP[15];
    //UID - identifikacioni broj korisnika
    int UID;
    //TCP soketi i adrese vezane za te sockete
    SOCKET ControlUserSocket;
    SOCKADDR_IN ControlSocketFrom;
    SOCKET UserSocket;
    SOCKADDR_IN SocketFrom;
public:
    User* next;
    bool end;
    //Postavljanje imena korisnika
    void SetName(char *pName);
    //Postavljanje IP adrese korisnika
    void SetIP(char *pIP);
    //Postavljanje identifikacionog broja korisnika - UID
    void SetUID(int pID);
    //Postavljanje socketa i adresa vezanih za sockete
    void SetControlUserSocket(SOCKET control_user_socket);
    void SetControlSocketFrom(SOCKADDR_IN control_socket_from);
    void SetUserSocket(SOCKET user_socket);
    void SetSocketFrom(SOCKADDR_IN socket_from);
    //Preuzimanje imena korisnika
    char* GetName();
    //Preuzimanje IP adrese korisnika
    char* GetIP();
    //Preuzimanje identifikacionog broja korisnika - UID
    int GetUID();
    //Preuzimanje socketa i adresa vezanih za sockete
    SOCKET GetControlUserSocket();
    void GetControlSocketFrom(SOCKADDR_IN *socket_from);
    SOCKET GetUserSocket();
    void GetSocketFrom(SOCKADDR_IN *socket_from);
};

```

Slika 3.1.3.3-1: Klasa *User*

Klasom *UserDB* definiše se **lista klijenata**, kao i operacije za rad sa tom listom:

- *ListUsers()* - ispis svih klijenata koji se nalaze u listi klijenata,
- *RegisterUser()* - dodavanje klijenta u listu klijenata,
- *FreeDB()* - brisanje svih klijenata iz liste,
- *UnregisterUserByID()* - izbacivanje klijenta iz liste na osnovu UID klijenta,
- *GetUserIPByID()* - dobijanje IP adrese klijenta na osnovu UID klijenta,
- *IsUserPresent()* - proveru da li klijent postoji u listi klijenata,
- *IsValidUser()* - proveru da li je klijentsko ime validno,
- *FindUserByUID()* - pretraživanje liste klijenata sa preuzimanjem informacija o klijentu,
- *ChangeUserParamsByUID()* - promena informacija o klijentu sa zadatim UID-om

U klasi *UserDB* definisani su pokazivač na prvi element liste, *first*, pokazivač na trenutni element liste, *current*, pokazivač na poslednji element liste, *end*, kao i ukupan broj elemenata u listi, *last*. Klasa *UserDB* prikazana je na slici 3.1.3.3-2.

```
class UserDB {
private:
    //Niz za mapiranje korisnickih UID brojeva
    char* UserIDArray;
    //Inicijalizacija UID niza
    void InitUserIDArray(char** pBuffer, int pSize);
    //Dodavanje novog polja u UID nizu
    void AddUserIDArray(char** pBuffer, int pSize);
    //Oznacavanje iskoriscenog UID broja
    void RemUserIDArray(char** pBuffer, int pID);
    //Pronalazenje slobodnog ID broja za dodelu novom korisniku
    int GetFreeUID(char** pBuffer);
    //Oslobadjanje memorije zauzete od strane UID niza
    void FreeUserIDArray(char** pBuffer);
public:
    //DEBUG ispis UserIDArray
    void IspisUserIDArray();
    //Konstruktor baze korisnika
    UserDB(){last = 0;};
    //Ukupan broj korisnika u bazi
    //(indeks poslednje registrovanog korisnika u bazi)
    int last;
    //Prvi element u bazi
    User* first;
    //Trenutni element u bazi
    User* current;
    //Poslednji element u bazi
    User* end;
    //Listanje korisnika koji su trenutno u bazi
    int ListUsers();
    //Registracija novog korisnika u bazu, vraca dodeljeni UID
    int RegisterUser(User *new_user);
    //Brisanje baze i oslobadjanje memorije
    int FreeDB();
    //Uklanjanje korisnika iz baze po njegovom ID broju
    int UnregisterUserByID(int pID);
    //Preuizmanje IP adrese korisnika po njegovom ID broju
    char* GetUserIPByID(int pID);
    //Provera da li postoji korisnik sa zadatim korisnickim imenom
    bool IsUserPresent(char* sUserName);
    //Provera da li je korisnicko ime ispravno
    //(ne sadrzi nedozvoljene znakove)
    bool IsValid(char* sUserName);
    //Pretrazivanje baze po UID korisnika
    //sa preuzimanjem informacija o korisniku
    bool FindUserByUID(int pID, User *found_user);
    //Promena informacija o korisniku sa zadatim UID-om
    int ChangeUserParamsByUID(int pID, User *changed_user);
};
```

Slika 3.1.3.3-2: Klasa *UserDB*

Klasa *DataBase* (Slika 3.1.3.3-3) je klasa kojom se reguliše pristup listi klijenata. Pristup listi klijenata regulisan je korišćenjem funkcija:

- *LockDataBase()* - blokiranje pristupa listi klijenata, i
- *UnlockDataBase()* - odblokiranje pristupa listi klijenata.

```

class DataBase
{
private:
    //baza podataka
    UserDB* base;
    //kriticna sekcija
    CRITICAL_SECTION DATA_BASE;
public:
    //konstruktor klase DataBase
    DataBase() { InitializeCriticalSection(&DATA_BASE); base = new UserDB(); }
    //blokiranje pristupa bazi podataka
    void LockDataBase() { EnterCriticalSection(&DATA_BASE); }
    //oblokiranje pristupa bazi podataka
    void UnlockDataBase() { LeaveCriticalSection(&DATA_BASE); }
    //preuzimanje pokazivaca na bazu podataka
    UserDB* GetDataBasePointer() { return base; }
    //destruktor klase DataBase
    ~DataBase() { DeleteCriticalSection(&DATA_BASE); delete base; }
};

```

Slika: 3.1.3.3-3: Klasa *DataBase*

3.1.3.4. Modul D - Obrada poruka signalizacionog protokola

```

class TParser {
private:
    //Poruka primljena TCP kanalom koja treba da se parsira
    TMESSAGE *Message;
    RESPONSE *Response;
    //promenljiva kojom se odlucuje
    //da li se poruka salje source klijentu ili dest klijentu
    //true - source klijent
    //false - dest klijent
    bool source_dest;

    sip_t *sip;
    //MESSAGE *msg_req;
    MESSAGE *msg_resp;
    int session;
    //DEBUG kreiranje Message poruke
    //(strukture iz ulaznog niza karaktera)
    void SetMsg(char* pMsg);
    //DEBUG preuzimanje Message poruke
    char* GetMsg();
public:
    //Default konstruktor
    TParser();
    //Destruktor
    ~TParser();
    //Postavljanje promenljive source_dest
    void SetSourceDest(bool s_d);
    //Parsiranje poruke i generisanje odgovora
    //i preuzimanje parametara source klijenta
    RESPONSE* MsgParser(char* pMsg, MESSAGE* pHost);
};

```

Slika 3.1.3.4-1: Klasa *TParser*

Zadatak **modula D** je obrađivanje SIP poruka i poziv **modula F** radi parsiranja SIP poruka.

Funkcija obrade SIP poruka je *MsgParser()*. Kao rezultat obrade SIP poruka **modul D** vraća poruku odgovora koja je upućena jednom od klijenata koji se koriste SIP protokolom (informacija o kojem klijentu je reč je deo pomenutog odgovora).

Realizacija implementacije **modula D** definisana je u datotekama *parser.h* i *parser.cpp*, i to klasom *TParser* prikazanom na slici 3.1.3.4-1.

3.1.3.5. Modul E - Kanal komunikacije za razgovor

Implementacija **modula E** realizovana je u datotekama *UDPServer.h* i *UDPServer.cpp*.

Zadaci **modula E** su:

- uspostava **UDP kanala komunikacije**,
- razmena podataka putem **UDP kanala komunikacije**, i
- UDP indentifikacija klijenta,

Funkcije **modula E** koje koristi **modul D** su:

- *InitializeUDPServer()*,
- *DeinitializeUDPServer()*,
- *ServerMainFunction()*,
- *UDP_ClientIdentificationThread()*,
- *UDP_ClientsCommunicationThread()*,

InitializeUDPServer() - funkcija kojom se kreira UDP utičnica preko koje će se obavljati razmena poruka sa **UDP kanalom komunikacije** (kreiranje se vrši pozivom funkcije *socket()*) i povezuje tu utičnicu sa klijentskom IP adresom (povezivanje se vrši pozivom funkcije *bind()*).

DeinitializeUDPServer() - funkcija kojom se raskida **UDP kanal komunikacije**.

ServerMainFunction() - funkcija kojom se poziva sekvenca funkcija koje služe za razmenu podataka sa **UDP kanalom komunikacije**. U ovoj funkciji se stvaraju dve niti *UDP_ClientsIdentificationThread()* i *UDP_ClientsCommunicationThread()*.

UDP_ClientsIdentificationThread() - nit kojom se sprovodi klijentska identifikacija, odnosno server putem UDP kanala komunikacija dobija **UID** od klijenta (**UID** je dobijen prilikom registracije klijenta na server). Server kada primi **UID** i identifikuje klijenta u sesiji on zapamti njegov UDP prolaz. Na tom prolazu je moguća UDP razmena podataka sa pomenutim klijentom. Ovo je urađeno iz razloga kako bi se izbegla "skrivenost" klijenta iza NAT server ili **firewall**-a i slično. Ukoliko je klijent identifikovan, server šalje signal potvrde klijentu putem **UDP kanala komunikacije**.

UDP_ClientsCommunicationThread() - nit kojom se obavlja komunikacija između klijenata putem **UDP kanala komunikacije**.

3.1.3.6. Modul F - Parser poruka signalizacionog protokola

Modul F služi kao podrška **modulu D**. On je realizovan u nekoliko datoteka koje koriste **oSIP** module (The oSIP library implements the Session Initiation Protocol (SIP -rfc3261-) Copyright (C) 2001,2002,2003 Aymeric MOIZARD jack@atosc.org), a te datoteke su:

- *Methods.h* i *Methods.cpp*, i
- *SIP_methods.h* i *SIP_methods.cpp*

U datotekama *Methods* i *SIP_methods* realizovane su funkcije koje olakšavaju rad sa **oSIP** modulima. S obzirom na to da ima mnogo funkcija, a relativno su jednostavne za razumevanje, iz ovih razloga one se neće posebno objašnjavati.

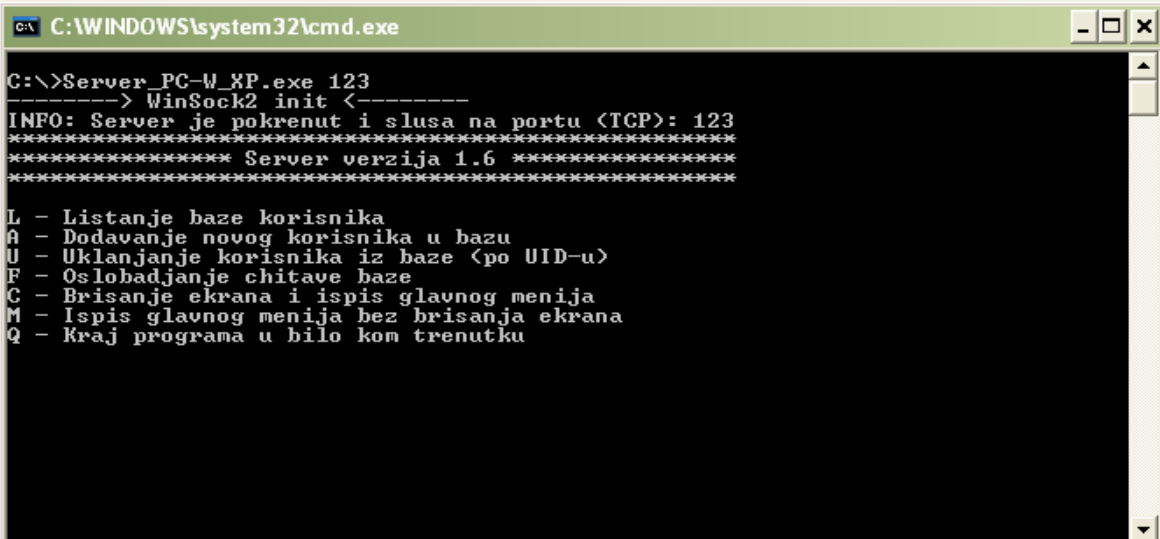
3.1.4. Način upotrebe

Startovanje servera obavlja se pozivom programa iz komandne linije i to na sledeći način:

Server_PC-W_XP.exe <port_number>,

gde je **Server_PC-W_XP.exe** ime izvršne datoteke dobijene kao rezultat prevođenja programskog koda prevodioca, a **port_number** predstavlja broj prolaza na kojem server sluša i obavezan je parametar prilikom poziva serverske aplikacije u izvršavanje.

Nakon poziva programa iz komandne linije i njegovog uspešnog startovanja, u komandnom prozoru ispisuje se sadržaj prikazan slikom 3.1.4-1. Uspešnim startovanjem server je spreman da prihvata **zahteve za konektovanje** od klijenata i koristi opcije iz menija (meni je, takođe, prikazan na slici 3.1.4-1).



```

C:\>Server_PC-W_XP.exe 123
-----> WinSock2 init <-----
INFO: Server je pokrenut i slusa na portu (TCP): 123
***** Server verzija 1.6 *****
*****
L - Listanje baze korisnika
A - Dodavanje novog korisnika u bazu
U - Uklanjanje korisnika iz baze <po UID-u>
F - Oslobadjanje chitave baze
C - Brisanje ekrana i ispis glavnog menija
M - Ispis glavnog menija bez brisanja ekrana
Q - Kraj programa u bilo kom trenutku

```

Slika 3.1.4-1: Izgled komandnog prozora nakon uspešnog startovanja serverske aplikacije

U toku rada servera u komandnom prozoru se ispisuju informacije o trenutnom stanju servera, informacije o porukama, koje klijenti šalju serveru i koje server šalje klijentima, informacije o greškama u radu servera i mnoge druge informacije vezane za rad servera.

3.2. Klijent

Klijent predstavlja uređaj na kojem se izvršava klijentska aplikacija. Uloga klijenta u serversko-klijentskom sistemu je da se prijavi na server i posredstvom servera stupi u komunikaciju (razmenu tekstualnih poruka ili razgovor) sa drugim klijentima, koji su prijavljeni na serveru.

3.2.1. Opis arhitekture

Aplikacija klijenta može biti smeštena na PC ili/i mobilnom uređaju performansi koje odgovaraju proseku performansi uređaja koji se u današnje vreme mogu pronaći na tržištu.

Pokretanjem klijentske aplikacije na PC ili mobilnom uređaju, taj uređaj se proglašava klijentom. Korisnik klijentske aplikacije, odnosno klijent, na početku bira opciju registracije na server, pri čemu se šalje **zahtev za registraciju klijenta na server**, i prelazi u stanje čekanja odgovora od servera.

Pre same registracije klijent dodeljuje sebi korisničko ime o kojem se obaveštava server prilikom registracije, jer se korisničko ime prosleđuje u okviru **zahteva za registraciju klijenta na server**. Ukoliko klijent sa istim korisničkim imenom već postoji na serveru, server obaveštava klijenta o tome i od njega zahteva da izabere novo korisničko ime kako bi se mogao registrovati na server. Ukoliko korisničko ime, koje je korisnik izabrao, odnosno dodelio sam sebi, ne postoji na serveru, tada server upusuje klijenta u svoju listu klijenata i obaveštava klijenta o uspešnoj registraciji. Nakon uspešnog registrovanja klijenta na server, klijent dobija od servera **spisak registrovanih klijenata** na serveru, koji se ažurira prilikom svake prijave novog klijenta na server, odnosno odjave već postojećeg klijenta sa servera. Kako klijent zna koliko i koji su klijenti registrovani na server, tada on može ostvariti komunikaciju sa bilo kojim klijentom iz **spiska registrovanih klijenata**.

Prilikom stupanja u komunikaciju sa nekim od registrovanih klijenata na serveru, klijent šalje **zahtev za uspostavu komunikacione sesije između klijenata** i čeka odgovor od servera. **Zahtev za uspostavu komunikacione sesije između klijenata** nosi informacije o klijentu koji je uputio zahtev i informacije o klijentu sa kojim se želi uspostaviti komunikaciona sesija. Ukoliko su podaci u zahtevu ispravni i klijent sa kojim se želi uspostaviti komunikaciona sesija i dalje postoji na serveru u toku slanja zahteva, kao i da pozivani klijent želi stupiti u vezu sa klijentom, koji je inicirao komunikacionu sesiju, tada klijent, koji je uputio zahtev, biva obavešten od strane servera da je ostvarena uspešna uspostava komunikacione sesije. Nakon informacije o uspešnoj uspostavi komunikacione sesije, klijenti međusobno razmenju SIP poruke, odnosno pregovaraju o načinu na koji će se obavljati komunikacija između njih (detaljnija objašnjenja o komunikaciji u serversko-klijentskom sistemu nalaze se u poglavlju 2.).

Kada se klijenti međusobno dogovore o načinu na koji će se obavljati komunikaciona sesija, odnosno kada se SIP protokol završi, tada klijenti, u zavisnosti da li pregovori imaju pozitivan ili negativan ishod, mogu da započnu razmenu tekstualnih poruka i razgovor u realnom vremenu posredstvom servera, ali isto tako pozivani klijent može odbiti poziv, te se komunikaciona sesija obustavlja.

Kako klijent ima mogućnost uspostave komunikacione sesije, tako mu se pruža mogućnost raskida komunikacione sesije. Obe klijentske strane u komunikacionoj sesiji mogu raskinuti komunikacionu sesiju, bez obzira na to da li je klijent inicirao uspostavu komunikacione sesije ili nije.

3.2.2. Opis platforme i razvojnog okruženja

Za pisanje programskog koda klijentske aplikacije izabrani su programski jezici C i C++. (Razlozi izbora C i C++ programskih jezika objašnjeni su u poglavlju 3.1.2.)

Prilikom realizacije klijentske aplikacije stvorene su dve verzije aplikacije:

- verzija **Client_PC-W_XP**, verzija klijentske aplikacije koja se izvršava na PC platformi sa operativnim sistemom *Windows XP*,
- verzija **Client_PPC-W_M5**, verzija klijentske aplikacije koja se izvršava na Pocket PC platformi sa operativnim sistemom *Windows Mobile 5.0*.

Programski kod obe verzije klijentske aplikacije razvijan je na PC platformi, s tim što se za razvoj verzije **Client_PPC-W_M5** koristio *Windows Mobile 5.0 Pocket PC* emulator, kojim je PC platformi data sposobnost da imitira, odnosno oponaša, sve osobine i način rada *Pocket PC* platforme sa operativnim sistemom *Windows Mobile 5.0*.

Za pisanje i prevodenje programskog koda klijentske aplikacije, kod obe verzije korišćeno je programsko okruženje *Microsoft Visual Studio 2005*, s tim što se za verziju **Client_PPC-W_M5** morao instalirati *Windows Mobile 5.0 Pocket PC SDK* (engl. *SDK - Software Development Kit*), čime se omogućilo korišćenje emulatora za *Windows Mobile 5.0 Pocket PC* platformu.

Korisnički interfejs klijentske aplikacije realizovan je korišćenjem naprednih funkcije za rad za grafičkim korisničkim interfejsom, odnosno sa GUI interfejsom (engl. Graphical User Interface). GUI funkcije su omogućene od strane MFC biblioteke (engl. *Microsoft Foundation Class library*) koja je sastavni deo *Microsoft Visual Studio 2005*. Izrada GUI interfejsa klijentske aplikacije korišćenjem MFC biblioteke je izabrana iz razloga što su MFC funkcije podržane i od strane Windows Mobile 5.0 operativnog sistema.

Treba napomenuti da je prvobitno obavljena realizacija verzija **Client_PC-W_XP**, koja je, zatim, prilagođena za izvršavanje na Windows Mobile 5.0 Pocket PC platformi u obliku nove verzije **Client_PPC-W_M5**.

3.2.3. Modularnost

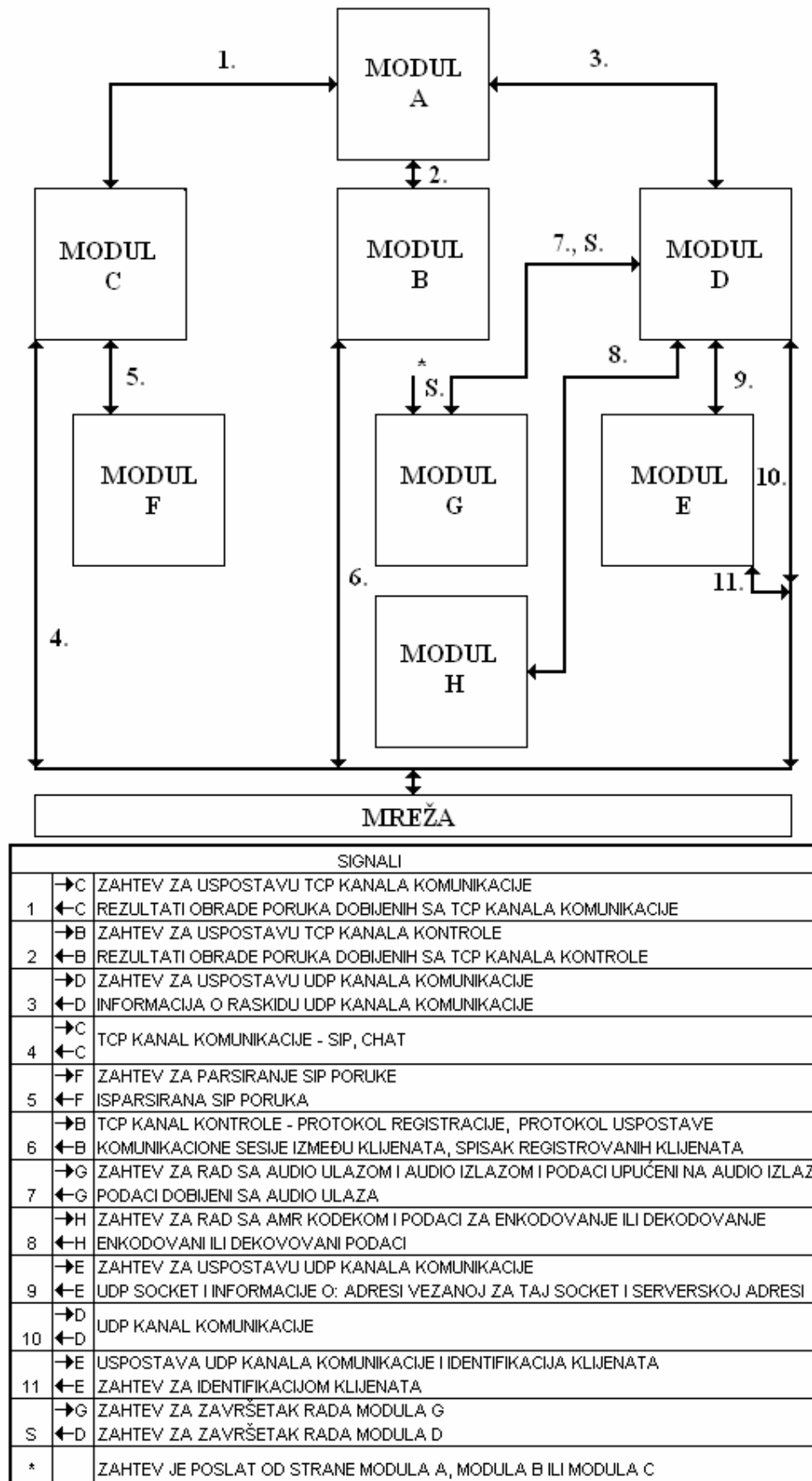
Programski kod klijentske aplikacije je organizovan u module. Postoji 8 modula klijentske aplikacije.

S obzirom na to da postoje dve verzije klijentske aplikacije, **Client_PC-W_XP** i **Client_PPC-W_M5**, modularnost programskog koda obe verzije je ista.

Na slici 3.2.3-1 prikazano je svih 8 modula klijentske aplikacije. Takođe, na slici je prikazano obavljanje razmene poruka između samih modula, ali i obavljanje razmene poruka između modula i mreže.

Moduli klijentske aplikacije su:

- **A modul** - modul zadužen za startovanje i obustavu rada klijentske aplikacije, iscrtavanje GUI interfejsa aplikacije, kontrolu nad objektima GUI interfejsa i poziv odgovarajuće funkcije kao reakciju na upotrebu objekata GUI interfejsa,
- **B modul** - modul zadužen za uspostavu **TCP kanala kontrole** i sprovođenje **protokola registracije** i **protokola uspostave komunikacione sesije između klijenata**, kao i primanje **spiska registrovanih klijenata** na serveru.
- **C modul** - modul zadužen za uspostavu **TCP kanala komunikacije** i sprovođenje SIP protokola, kao i obavljanje razmene tekstualnih poruka. Za parsiranje SIP poruka, **modul C** koristi usluge **modula F**,
- **D modul** - modul zadužen za upravljanje **modulima E, G i H**, i slanje podataka putem **UDP kanala komunikacije**,
- **E modul** - modul zadužen za uspostavu **UDP kanala komunikacije** i UDP identifikaciju klijenta,
- **F modul** - modul zadužen za parsiranje SIP poruka. On sam predstavlja SIP parser. Za njega se može reći da je pomoćni modul **modula C** i da predstavlja produžetak **modula C**.
- **G modul** - modul zadužen za rad sa audio ulaznim i audio izlaznim uređajem (mikrofonom i zvučnicima),
- **H modul** - modul zadužen za rad sa AMR kodekom.



Slika 3.2.3-1: Moduli klijentske aplikacije

Svaki od modula sastoji se iz dva dela:

- 1) zaglavlja, čiji sadržaj se nalazi u *.h datotekama, i
- 2) izvršnog dela, čiji sadržaj se nalazi u *.cpp datotekama.

3.2.3.1. Modul A - Startovanje i obustava rada klijentske aplikacije

Modul A je prvi modul koji se stvara nakon pokretanja programa klijentske aplikacije. Realizacija implementacije **modula A** je smeštena u datotekama:

- **AK_Klijent_Visual.h** i **AK_Klijent_Visual.cpp**,
- **AK_Klijent_VisualDlg.h** i **AK_Klijent_VisualDlg.cpp**,
- **PropertiesDlg.h** i **PropertiesDlg.cpp**.

Zadaci modula A su:

- startovanje i obustava rada klijentske aplikacije,
- iscrtavanje korisničkog GUI interfejsa,
- kontrola rada nad objektima GUI interfejsa i reagovanje pozivom odgovarajuće funkcije prilikom upotrebe nekog objekta GUI interfejsa,
- upravljanje **modulom B**, **modulom C** i **modulom D** (i **modulom G** u slučaju **S**. signala prikazanog na slici 3.2.3-1)

Za realizaciju GUI interfejsa klijentske aplikacije koriste se funkcije MFC biblioteke za kreiranje dialog prozora i objekata na tim prozorima (na primer, dugme - *engl. button*, polje za upis i ispis, koje će se koristiti pod nazivom **edit kontrola** - *engl. edit control*), kao i veliki broj operacija koje se mogu obavljati nad prozorom i njegovim objektima.

Za upravljanje, ali i komunikaciju, između **modula A** i ostalih modula (**B**, **C**, **D** i **G**) koristi se događaji, pri čemu se najviše koristi događaj **WM_KEYDOWN** (to su u stvari događaji, jer se za svaki pritisnut taster izaziva se **WM_KEYDOWN** događaj koji nosi sa sobom numerički broj tastera koji je pritisnut. Za detaljnija objašnjenja pogledati MFC dokumentaciju [7\[8\]](#)), te se zbog ovog događaja na početku rada klijentske aplikacije onemogućuju prekidi sa tastature.

Kako je **modulom A** definisano mnogo funkcija, od kojih su neke i redefinisane funkcije MFC biblioteke, u nastavku se navode samo glavne funkcije **modula A**:

- *OnKeyDown()*,
- *OnSignInOut()*,
- *OnSignout()*,
- *OnCall()*,
- *OnEndCall()*,
- *OnSettings()*,
- *OnClose()*,
- *OnSendChat()*,
- *UDP_Stream()*.

OnKeyDown() predstavlja funkciju kojom je realizovan rukovalac **WM_KEYDOWN** događajem. Kada se desi **WM_KEYDOWN** događaj, funkcija *OnKeyDown()* na osnovu numeričkog broja, koji ovaj događaj nosi sa sobom, određuje koja će sledeća akcija biti izvršena.

Neke od akcija koje se mogu izvršiti nakon što se desi *WM_KEYDOWN* događaj :

- startovanje niti *UDP_Stream()*,
- ispis informacija o radu klijenta i servera u **edit kontroli** dialog prozora klijentske aplikacije (informacije o greškama, trenutnom stanju klijenta, i informacije o greškama, trenutnom stanju servera i slično),
- poziv funkcija modula **B**, **C** i **D**, i slično.

OnSignInOut() predstavlja funkciju koja se poziva na pritisak dugmeta za prijavu/odjavu klijenta na server GUI interfejsa klijentske aplikacije. Funkcija *OnSignInOut()* zadužena je za prijavu klijenta na server što se obavlja:

- pozivom funkcije *ClientList()* kojom se startuje **modul B** (pogledati poglavlje [3.2.3.2](#)), i
- pozivom funkcije *ClientRegister()* kojom se započinje sprovođenje **protokola registracije**.

Osim prijave klijenta na server, funkcija *OnSighInOut()* obavlja i odjavu klijenta sa servera i to pozivom funkcije *OnSignout()*.

OnSignout() je funkcija kojom se obavlja odjava sa servera i završetak rada modula **B**, **C**, **D**, i **E**, ako su moduli bili prethodno startovani. Ako je klijent prijavljen na server i pri tome još komunicira sa drugim klijentom, tada se pozivom *OnSignout()* završava ta komunikacija (pozivom funkcije *OnEndCall()*), nakon čega se obavlja i odjava klijenta sa servera (pozivom sekvence funkcija za završetak rada modula **B**, **C**, **D**, i **E**).

OnClose() funkcijom obavlja se obustava rada i gašenje klijentsko serverske aplikacije. Ako je klijent prijavljen na server i pri tome još komunicira sa drugim klijentima, tada se pozivom funkcije *OnClose()* završava poziv i obavlja odjava klijenta sa servera (to se čini pozivom funkcije *OnSignOut()* unutar *OnClose()* funkcije).

OnCall() je funkcija koja se poziva na pritisak dugmeta za poziv klijenta GUI interfejsa klijentske aplikacije. Funkcija *OnCall()* je zadužena za startovanje **modula C** radi uspostave **TCP kanala komunikacije**, što se obavlja pozivom funkcije *Client()* (pogledaj poglavlje [3.2.3.3](#)).

OnEndCall() je funkcija koja se poziva na pritisak dugmeta za završetak poziva sa klijentom GUI interfejsa klijentske aplikacije. Funkcija *OnEndCall()* je zadužena za završetak razgovora ostvarenog između klijenta, i to se čini pozivom sekvence funkcija kojima se obavlja obustava rada modula **B**, **C**, **D** i **G** (**modulu G** se šalje signal **S**, prikazan na slici 3.2.3-1).

OnSettings() predstavlja funkciju koja se poziva na pritisak dugmeta za podešavanje parametara klijentske aplikacije. Pozivom ove funkcije iscrtava se novi dijalog prozor, u kome se mogu podešavati parametri vezani za klijentsku aplikaciju, a to su: korisničko ime klijenta, IP adresa i prolaz servera na koga se klijent prijavljuje i preko koga klijent ostvaruje komunikaciju sa drugim klijentima (realizacija implementacije pomenutog dijaloga prozora nalazi se u datotekama **Properties.h** i **Properties.cpp**).

OnSendChat() je funkcija koja preuzima tekstualne poruke od GUI interfejsa klijentske aplikacije i prosleđuje ih ka **modulu C**, kako bi bili poslani drugom klijentu putem **TCP kanala komunikacije** (tekstualne poruke se preuzimaju iz edit kontrole).

Funkcija *UDP_Stream()* predstavlja u stvari nit u kojoj se poziva funkcija *StreamUDP()*, čijim se pozivom startuje rad **modula D**, i to iz potrebe da se stvori **UDP kanal komunikacije** kako bi se mogao odvijati razgovor između klijenata.

3.2.3.2. Modul B - Kanal kontrole

Implementacija **modula B** realizovana je u datotekama *ClientUserList.h* i *ClientUserList.cpp*.

Zadaci **modula B** su:

- uspostava **TCP kanala kontrole** sa serverom,
- sprovođenje **protokola registracije i protokola uspostave komunikacione sesije između klijenata**, i
- primanje **spiska registrovanih klijenata** od serveru.

S obzirom na to da **modul B** sadrži veliki broj funkcija, neke od bitnijih funkcija su:

- *ClientList()*,
- *ClientListSendThread()*,
- *ClientListReceiveThread()*,
- *ClientRegister()*,
- *ClientListSendMessage()*,
- *GetUserFromList()*,
- *GetUserList()*,
- *DeinitClientList()*.

ClientList() - funkcija kojom se startuje rad **modula B** i obavlja uspostava **TCP kanala kontrole**. Funkcijom *ClientList()* kreira se TCP utičnica preko koje će se obavljati razmena poruka sa **TCP kanalom kontrole** (kreiranje se vrši pozivom funkcije *socket()*). Kada je utičnica kreirana, ona se povezuje sa IP adresom servera na koju se upućuje **zahtev za konektovanje** (povezivanje sa IP adresom servera i slanje **zahteva za konektovanje** obavlja se pozivom funkcije *connect()*). Osim kreiranja utičnice i slanja **zahteva za konektovanje**, funkcija *ClientList()* stvara dve niti: *ClientListSendThread()* i *ClientListReceiveThread()*.

ClientListSendThread() - to je nit zadužena za slanje poruka upućenih ka serveru putem **TCP kanala kontrole**.

ClientListReceiveThread() - to je nit zadužena za primanje poruka koje stižu sa **TCP kanala kontrole**. S obzirom na to da je nit *ClientListReceiveThread()* zadužena za primanje poruka sa **TCP kanala kontrole**, u ovoj niti implementiran je i parser primljenih poruka. Primljene poruke su u stvari poruke koje se razmenjuju prilikom sprovođenja **protokola registracije, protokola uspostave komunikacione sesije između klijenata**, kao i poruke primljenje prilikom sprovođenja **protokola slanja spiska registrovanih klijenata**.

ClientRegister() - funkcija koja služi za registraciju klijenta na server (pre poziva ove funkcije neophodno je prethodno pozvati funkciju *ClientList()*). Kao ulazni parametar ove funkcije prosleđuje se korisničko ime pod kojim bi se klijent registrovao na server.

ClientListSendMessage() - kao ulazni parametar ove funkcije prosleđuje se poruka koja se želi poslati serveru. S obzirom na to da je nit *ClientListSendThread()* zadužena za slanje poruka serveru, funkcija *ClientListSendMessage()* u stvari prosleđuje svoj ulazni parametar ovoj niti (to se vrši oslobađanjem semafora *client_sem*, koji se signalizira niti *ClientListSendThread()* da postoji spremna poruka za slanje serveru).

GetUserFromList() - funkcija kojom se vrši parsiranje jednog korisnika iz **spiska registrovanih klijenata** koji je poslat od servera. Kao rezultat ova funkcija vraća strukturu *USERLIST* u kojoj se nalaze isparsirani podaci o korisniku.

GetUserList() - funkcija kojom se vraća adresa na prvu memorijsku lokaciju na kojoj se čuva **spisak registrovanih klijenata** dobijen od servera, odnosno početna memorijska adresa niza *USERLIST* *UserList* (treba spomenuti da se niz *UserList* popunjava sa podacima koji su rezultat parsiranja korisnika iz **spiska registrovanih klijenata** funkcijom *GetUserFromList()*).

DeinitClientList() - funkcija kojom se oslobađa sva memorija zauzeta od strane **modula B** i raskida **TCP kanal kontrole**.

3.2.3.3. Modul C - Kanal razmene poruka

Implementacija **modula C** realizovana je u datotekama *Client.h* i *Client.cpp*.

Zadaci **modula C** su:

- uspostava **TCP kanala komunikacije** sa serverom,
- obrada SIP poruka i sprovođenje SIP protokola, i
- razmena tekstualnih poruka između klijenata.

Kako je jedan od zadataka **modula C** obrada SIP poruka i sprovođenje SIP protokola, **modulu C** su, prilikom obrade SIP poruka, neophodne usluge parsiranja SIP poruka, koje pruža **modul F**.

Modul C je realizovan tako da obavlja dve vrste transakcija, a to su:

- **serverska transakcija** - **modul C** prima zahteve sa mreže, obrađuje ih i šalje odgovore na zahteve natrag na mrežu, odnosno klijentska aplikacija se ponaša kao server. **Modul C** obavlja **serversku transakciju** kada je klijent primalac poziva, odnosno kad je on pozivan od strane drugog klijenta,
- **klijentska transakcija** - **modul C** šalje zahteve na mrežu i čeka odgovore sa mreže, koji su neophodni za njegov dalji rad. **Modul C** obavlja **klijentsku transakciju** kada je klijent pozivalac, odnosno kada klijent poziva drugog klijenta.

S obzirom na to da **modul C** sadrži veliki broj funkcija, neke od bitnijih funkcija su:

- *SetClientServer()*,
- *Client()*,
- *ClientSendThread()*,
- *ClientReceiveThread()*,
- *ClientSendMessage()*,
- *ServerSendMessage()*,
- *DeinitClient()*,
- *DeinitServer()*.

SetClientServer() - funkcija kojom se u zavisnosti od njenog ulaznog parametra podešava transakcija koju će obavljati **modul C**, pri tome ulazni parametri mogu biti: znak 's' - modul C obavlja serversku transakciju, ili znak 'c' - modul C obavlja klijentsku transakciju.

Pre same upotrebe ostalih funkcija **modula C** neophodno je prethodno odrediti koju će transakciju **modul C** obavljati, odnosno jedan poziv funkcije *SetClientServer()* mora prethodi pozivu funkcije *Client()* kojom se startuje rad **modula C**.

Client() - funkcija kojom se startuje rad **modula C** i obavlja uspostava **TCP kanala komunikacije**. Funkcijom *Client()* kreira se TCP utičnica preko koje će se obavljati razmena poruka sa **TCP kanalom komunikacije** (kreiranje se vrši pozivom funkcije *socket()*). Kada je utičnica kreirana, ona se povezuje sa IP adresom servera na koju se upućuje **zahtev za konektovanje** (povezivanje sa IP adresom servera i slanje **zahteva za konektovanje** obavlja se pozivom funkcije *connect()*). Osim kreiranja utičnice i slanja **zahteva za konektovanje**, funkcija *Client()* stvara dve niti: *ClientSendThread()* i *ClientReceiveThread()*.

ClientSendThread() - to je nit zadužena za slanje poruka upućenih ka serveru putem **TCP kanala komunikacije**.

ClientReceiveThread() - to je nit zadužena za primanje poruka koje stižu sa **TCP kanala komunikacije**. S obzirom na to da je nit *ClientReceiveThread()* zadužena za primanje poruka sa **TCP kanala komunikacije**, u ovoj niti implementiran je i parser primljenih poruka. Primljene poruke su u stvari poruke koje se razmenjuju prilikom sprovođenja SIP protokola i razmene tekstualnih poruka.

ClientSendMessage() - kao ulazni parametar ove funkcije prosleđuje se poruka koja se želi poslati serveru. S obzirom na to da je nit *ClientSendThread()* zadužena za slanje poruka serveru, funkcija *ClientSendMessage()* u stvari prosleđuje svoj ulazni parametar ovoj niti (to se vrši oslobađanjem semafora *client_semaphore*, koji signalizira niti *ClientSendThread()* da postoji spremna poruka za slanje serveru. Nit *ClientSendThread()* će reagovati na semafor *client_semaphore* samo ukoliko je **modul C** bio podešen da obavlja **klijentsku transakciju**).

ServerSendMessage() - isto kao i kod funkcije *ClientSendMessage()*, kao ulazni parametar ove funkcije prosleđuje se poruka koja se želi poslati serveru. S obzirom na to da je nit *ClientSendThread()* zadužena za slanje poruka serveru, funkcija *ServerSendMessage()* u stvari prosleđuje svoj ulazni parametar ovoj niti (to se vrši oslobađanjem semafora *server_semaphore*, koji signalizira niti *ClientSendThread()* da postoji spremna poruka za slanje serveru. Nit *ClientSendThread()* će reagovati na semafor *server_semaphore* samo ukoliko je **modul C** bio podešen da obavlja **serversku transakciju**).

DeinitClient() - funkcija kojom se:

- 1) oslobađa sva memorija zauzeta od strane **modula C** za obavljanje **klijentske transakcije**, i
- 2) raskida **TCP kanal komunikacije**.

DeinitServer() - funkcija kojom se:

- 1) oslobađa sva memorija zauzeta od strane **modula C** za obavljanje **serverske transakcije**, i
- 2) raskida **TCP kanal komunikacije**.

3.2.3.4. Modul D - Razgovor i upravljanje modulima E, G, H

Implementacija **modula D** realizovana je u datotekama *StreamUDP.h* i *StreamUDP.cpp*.

Zadaci **modula D** su:

- startovanje i upravljanje **modulima E, G i H**,
- slanje podataka putem **UDP kanala komunikacije**,

Glavne funkcije **modula D** su:

- *StreamUDP()*,
- *UDP_ReceiveThread()*,
- *UDP_SendThread()*,

StreamUDP() - funkcija je zadužena za startovanje **modula E**, sa ciljem da se uspostavi **UDP kanal komunikacije, modula G**, sa ciljem da se ostvari komunikacija sa ulaznim i izlaznim audio uređajem (mikrofon i zvučnici PC ili mobilnog uređaja), i **modula H**, sa ciljem da se pripremi AMR kodek za korišćenje. Osim startovanja **modula E, G i H**, obavlja se i startovanje dve niti: *UDP_ReceiveThread()* i *UDP_SendThread()*.

Nakon startovanja modula i niti, funkcija *StreamUDP()* odlazi u cekanje na događaj, kojim se prekida rad **modula D** (signal **S** na slici 3.2.3-1). Kada se ovaj događaj desi tada **modul D** obaveštava **module E, G i H** da prestanu sa radom, nakon čega i sam prestaje sa radi.

UDP_ReceiveThread() - to je nit zadužena za primanje podataka koji stižu sa **UDP kanala komunikacije**. S obzirom na to da se putem **UDP kanala komunikacije** primaju AMR kompresovani audio podaci, ova nit prosleđuje te podatke AMR dekomoderu na dekodovanje (pozivom funkcije **modula H** *decoder()*), i zatim, nakon što su podaci dekodovani, nit ih prosleđuje na izlazni audio uređaj (pozivom funkcije **modula G** *PlayWave()*)

UDP_SendThread() - to je nit zadužena za slanje audio podataka putem **UDP kanal komunikacije**. Podaci koji se šalju su primljeni sa ulaznog audio uređaja (funkcijom **modula G** *getWaveInData()* se preuzimaju audio podaci), ali pre samog slanja ovih podataka oni se prosleđuju na kompresiju AMR enkoderu (funkcijom **modula H** *encoder()* poziva se AMR enkoder). Nit *UDP_SendThread()*, pre nego što dobije audio podatke, čeka na događaj koji signalizira da postoje audio podaci koji su spremni za slanje putem **UDP kanal komunikacije** (funkcijom **modula G** *WaitEvent()* proverava se da li se dogodio pomenuti događaj).

3.2.3.5. Modul E - Kanal komunikacije za razgovor

Implementacija **modula E** realizovana je u datotekama *UDPClient.h* i *UDPClient.cpp*.

Zadaci **modula E** su:

- uspostava **UDP kanala komunikacije**, i
- UDP indentifikacija klijenta,

Funkcije **modula E** koje koristi **modul D** su:

- *InitializeUDPClient()*,
- *DeinitializeUDPClient()*,
- *ClientIntroduction()*,

InitializeUDPClient() - funkcija kojom se kreira UDP utičnica preko koje će se obavljati razmena poruka sa **UDP kanalom komunikacije** (kreiranje se vrši pozivom funkcije *socket()*) i povezuje tu utičnicu sa klijentskom IP adresom (povezivanje se vrši pozivom funkcije *bind()*).

DeinitializeUDPClient() - funkcija kojom se raskida **UDP kanal komunikacije**.

ClientIntroduction() - funkcija kojom se sprovodi klijentska identifikacija, odnosno klijent putem UDP kanala komunikacija šalje **UID** serveru (**UID** je dobijen prilikom registracije klijenta na server). Server kada primi **UID** i identifikuje klijenta u sesiji on zapamti njegov UDP prolaz. Na tom prolazu je moguća UDP razmena podataka sa pomenutim klijentom. Ovo je urađeno iz razloga kako bi se izbegla "skrivenost" klijenta iza NAT server ili firewalla i slično. Nakon što pošalje serveru svoj **UID**, klijent prelazi u stanje čekanja do trenutka dok ne dobije signal potvrde od servera da je klijent identifikovan (i to putem **UDP kanala komunikacije**).

3.2.3.6. Modul F - Parser poruka signalizacionog protokola

Modul F služi kao podrška **modulu C**. On je realizovan u nekoliko datoteka koje koriste **oSIP** module (The oSIP library implements the Session Initiation Protocol (SIP -rfc3261-) Copyright (C) 2001,2002,2003 Aymeric MOIZARD jack@atosc.org), a te datoteke su:

- `Methods.h` i `Methods.cpp`, i
- `SIP_methods.h` i `SIP_methods.cpp`

U modulima *Methods* i *SIP_methods* realizovane su funkcije koje olakšavaju rad sa **oSIP** modulima. S obzirom na to da **modul F** ima mnogo funkcija, a relativno su jednostavne za razumevanje, iz ovih razloga one se neće posebno objašnjavati.

3.2.3.7. Modul G - Rad sa ulaznim i izlazim audio uređajem

Modul G se sastoji iz dva dela:

- **waveIn modul** - deo za rad sa ulaznim audio uređajem, čija je implementacija realizovana u datotekama ***waveIn.h*** i ***waveIn.cpp***, i
- **waveOut modul** - deo za rad sa izlazim audio uređajem, čija je implementacija realizovana u datotekama ***waveOut.h*** i ***waveOut.cpp***.

Osnovni zadatak **modula G** je da obezbedi rad sa audio ulaznim i audio izlaznim uređajima (mikrofonom i zvučnicima, bilo kod PC ili mobilnog uređaja).

Bitne funkcije **modula G** za rad sa ulazim audio uređajem (*waveIn*) su:

- *InitWaveIn()*,
- *DeinitWaveIn()*,
- *DoRecord()*,
- *waveInProc()*,
- *DoRecordEnd()*,
- *getWaveInData()*,
- *WaitEvent()*,
- *WaveInEndControl()*.

InitWaveIn() - funkcija kojom se inicijalizuju događaji koje koristi **waveIn modul**.

DeinitWaveIn() - funkcija kojom se oslobađaju događaji koje koristi **waveIn modul**.

DoRecord() - funkcija kojom se obavlja priprema ulaznih bafera za prihvatanje audio podataka sa ulaznog audio uređaja, startovanje niti *waveInProc()*, startovanje rada i zauzimanje ulaznog audio uređaja. Nakon izvršenih prethodno opisanih radnji, funkcija *DoRecord()* odlazi u čekanju na događaj *wave_in_end_event*, koji je izazvan samo onda kada se želi obustaviti rad sa ulaznim audio uređajem i to pozivom funkcije *DoRecordEnd()*. Kada je funkcija *DoRecord()* signalizirana događajem *wave_in_end_event*, tada se obavlja oslobađanje ulaznih bufera za prihvatanje audio podataka, završava se rad sa ulaznim uređajem i obavlja se njegovo oslobađanje.

waveInProc() - to je nit zadužena za prikupljanje audio podataka sa audio ulaznog uređaja u ulazne bafere. Kada je jedan ulazni bufer pun i spreman za dalju obradu tada se izaziva događaj *wave_in_data_event* i puni se drugi prazan bufer. Ukoliko se ulazni uređaj oslobodi, čime se označava da je završeno njegovo korišćenje, nit *waveInProc()* završava svoj rad i uništava se.

DoRecordEnd() - funkcija kojom se signalizira kraj rada sa ulaznim audio uređajem, a samim tim i kraj rada sa čitavim **waveIn modulom** (u stvari šalje se signal **S** prikazan na slici 3.2.3-1).

getWaveInData() - funkcija kojom se preuzimaju audio podaci iz jednog od ulaznih bafera **waveIn modula**.

WaitEvent() - funkcija kojom se čeka na događaj *wave_in_data_event*.

WaveInEndControl() - funkcija koja vraća vrednost, koja označava da li se ulazni audio uređaj još koristi ili ne.

Bitne funkcije **modula G** za rad sa izlaznim audio uređajem (*waveOut*) su:

- *InitWaveOut()*,
- *DeinitWaveOut()*,
- *PlayWave()*,
- *EndPlayWave()*.

InitWaveOut() - funkcija kojom se startuje rad i zauzima izlazni audio uređaj, inicijalizuju izlazni baferi za smeštanje audio podataka koji treba da se proslede izlaznom audio uređaju.

DeinitWaveOut() - funkcija kojom se završava rad i oslobađa izlazni audio uređaj. Ovim se označava i kraj rada sa **waveOut modulom**.

PlayWave() - funkcija kojom se obavlja reprodukcija zvuka na izlaznom audio uređaju.

EndPlayWave() - funkcija kojom se čeka da se isprazne svi izlazni baferi i samim tim reprodukuje zakasneli zvuk na izlaznom uređaju.

Završetak rada **modula waveIn** i **waveOut** označava i završetak rada **modula G**.

3.2.3.8. Modul H - Kodek

Modul H služi za rad sa AMR kodekom, koji je “*open source*”. U njemu su definisane funkcije za rad sa AMR enkoderom i dekoderom (s obzirom da je kod AMR kodka “*open source*” način njegovog rada se neće objašnjavati, jer je on samo korišćen u ovom radu, ništa novo nije implementirano, osim sto je dodat novi modul **codec** (**codec.h** i **codec.c**), koji olakšava korišćenje funkcija AMR kodeka i objedinjuje module AMR kodeka u taj jedan modul).

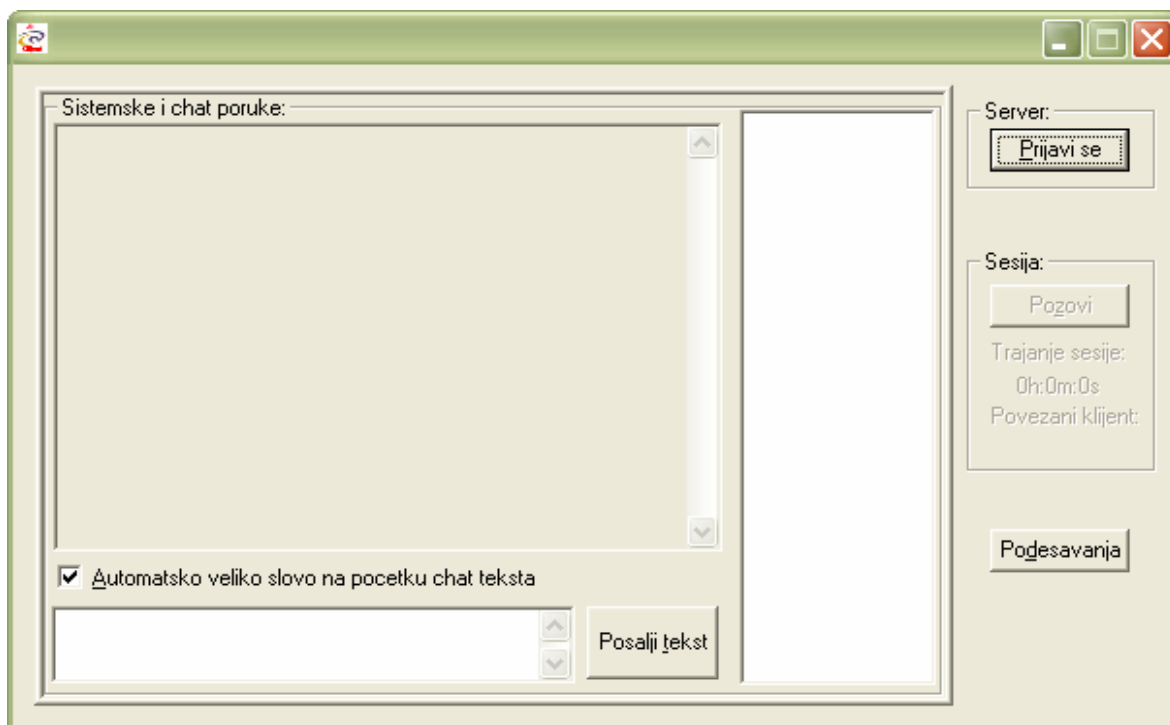
3.2.4. Način upotrebe

U ovom poglavlju biće opisan način upotrebe klijentske aplikacije, odnosno način na koji se upotrebljava GUI klijentske aplikacije.

Startovanje klijentske aplikacije obavlja se dvostrukim klikom na izvršnu datoteku .exe. Kako postoje dve verzije klijenta, tako postoje i dve verzije .exe datoteke, i to:

- verzija **Client_PC-W_XP.exe**, verzija klijentske aplikacije koja se izvršava na PC platformi koja radi pod operativnim sistemom *Windows XP*,
- verzija **Client_PPC-W_M5.exe**, verzija klijentske aplikacije koja se izvršava na Pocket PC platformi koja radi pod operativnim sistemom *Windows Mobile 5.0*.

Nakon startovanja klijentske aplikacije iscrtava se prozor klijentske aplikacije.



Slika 3.2.4-1: Prozor PC klijentske aplikacije



Slika 3.2.4-2: Prozor Pocket PC klijentske aplikacije

Na slici 3.2.4-1 prikazan je prozor klijentske aplikacije pokrenute na PC uređaju, dok je na slici 3.2.4-2 prikazan prozor klijentske aplikacije pokrenute na Pocket PC uređaju.

GUI interfejs obe verzije klijentske aplikacije se ne razlikuje po načinu upotrebe. Svi dugmići i kontrole koje postoje na GUI interfejsu PC klijentske aplikacije, postoje i na GUI interfejsu Pocket PC klijentske aplikacije, ali su zbog veličine ekrana Pocket PC uređaja dugmići i kontrole GUI interfejsa ispremeštani kako bi bili vidljivi na ekranu.

Oba prozora klijentske aplikacije uvek sadrže četiri dugmeta i tri kontrole. Prilikom rada klijentske aplikacije, neki od dugmića mogu promeniti svoje ime i imati drugačiju namenu ili pak može biti dopuštena njihova upotreba, što zavisi od stanja u kom se nalazi klijentska aplikacija (na primer, upotreba dugmeta **Pozovi**, u trenutku kad je aplikacija startovana, je onemogućena). Ono što se nikad nemanja su kontrole:

- **edit kontrola 1** - za ispis informacija o trenutnom stanju klijenta i servera, kao i za ispis tekstualnih poruka poslatih ka, i dobijenih od drugog klijenta (na slici 3.2.4-2 to je plava **edit kontrola**),
- **edit kontrola 2** - za pisanje tekstualne poruke koja se šalje drugom klijentu (na slici 3.2.4-2 **edit kontrola** levo od dugmeta **Posalji tekst**),
- **list box kontrola** - za ispis **spiska registrovanih klijenata** na serveru (na slici 3.2.4-2 **list box kontrola** (engl. *List Box Control*) iznad dugmeta **Podesavanja**).

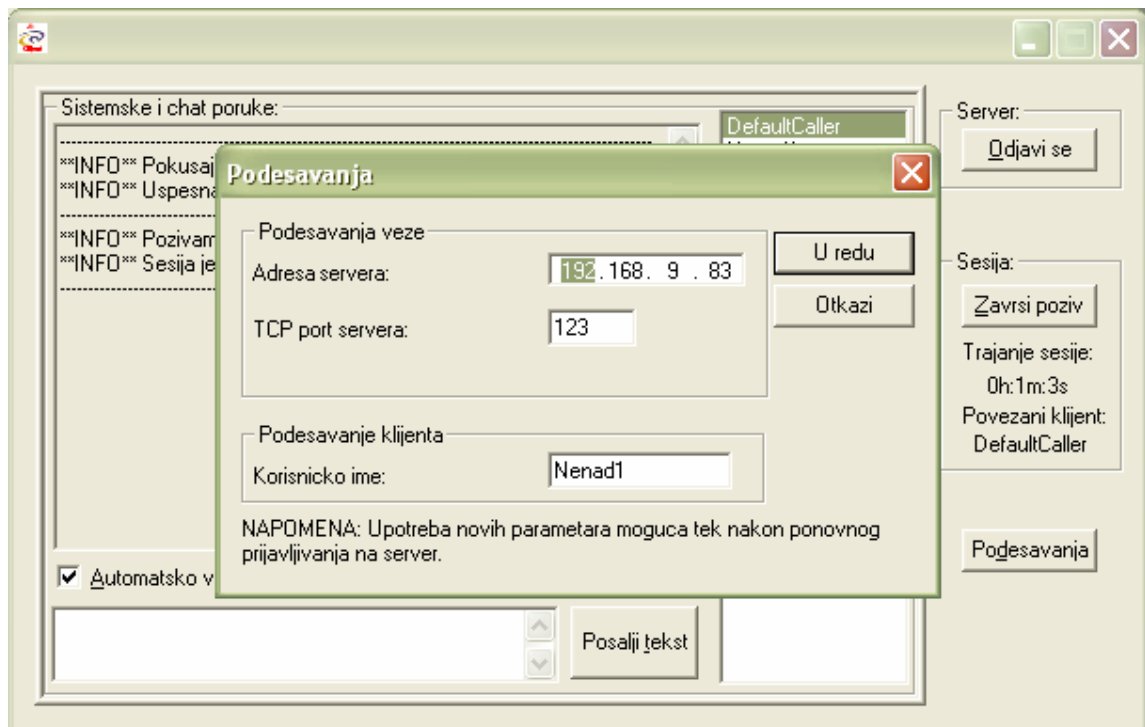
Na slici 3.2.4-2 se takođe vidi i **check box kontrola** (engl. *Check Box Control*), čija je namena da prvo slovo svake nove tekstualne poruke pretvori u veliko slovo, i to samo onda kada je **check box kontrola** otključena (pod jednom tekstualnom porukom se podrazumeva tekst nakon koga je pritisnuto dugme **Posalji tekst** ili taster sa tastature **Enter**).

Na početku rada klijentske aplikacije, prozor aplikacije sadrži sledeće dugmiće:

- dugme **Prijavi se** - pritiskom na ovo dugme obavlja se prijava i registracija klijenta na server, i to samo ukoliko postoji server (pritiskom na ovo dugme poziva se funkcija *OnSignInOut()* objašnjena u poglavlju [3.2.3.1.](#)),
- dugme **Pozovi** - na početku rada klijentske aplikacije upotreba dugmeta **Pozovi** je onemogućena. Kada je njegova upotreba omogućena, ovim dugmetom se vrši poziv klijenta sa kojim se želi ostvariti komunikacija (pritiskom na ovo dugme poziva se funkcija *OnCall()* objašnjena u poglavlju [3.2.3.1.](#)),
- dugme **Podesavanja** - pritiskom na ovo dugme na klijentskom ekranu prikazuje se novi prozor u kome je moguće obaviti podešavanje raznih parametara vezanih kako za klijenta tako i za servera (pritiskom na ovo dugme poziva se funkcija *OnSettings()* objašnjena u poglavlju [3.2.3.1.](#)),
- dugme **Posalji tekst** - pritiskom na ovo dugme šalje se tekstualna poruka drugom klijentu (pritiskom na ovo dugme poziva se funkcija *OnSendChat()* objašnjena u poglavlju [3.2.3.1.](#)).

U nastavku sledi opis prijave na server i poziva klijenta.

Pre same prijave na server, neophodno je da klijent podesi parametre klijentske aplikacije, a to se čini pritiskom na dugme **Podesavanja**. Nakon pritiska na dugme **Podesavanja** na klijentskom ekranu prikazuje se prozor za podešavanje raznih parametara vezanih za klijenta i servera.



Slika 3.2.4-3: Prozor za podešavanje PC klijentske aplikacije



Slika 3.2.4-4: Prozor za podešavanje Pocket PC klijentske aplikacije

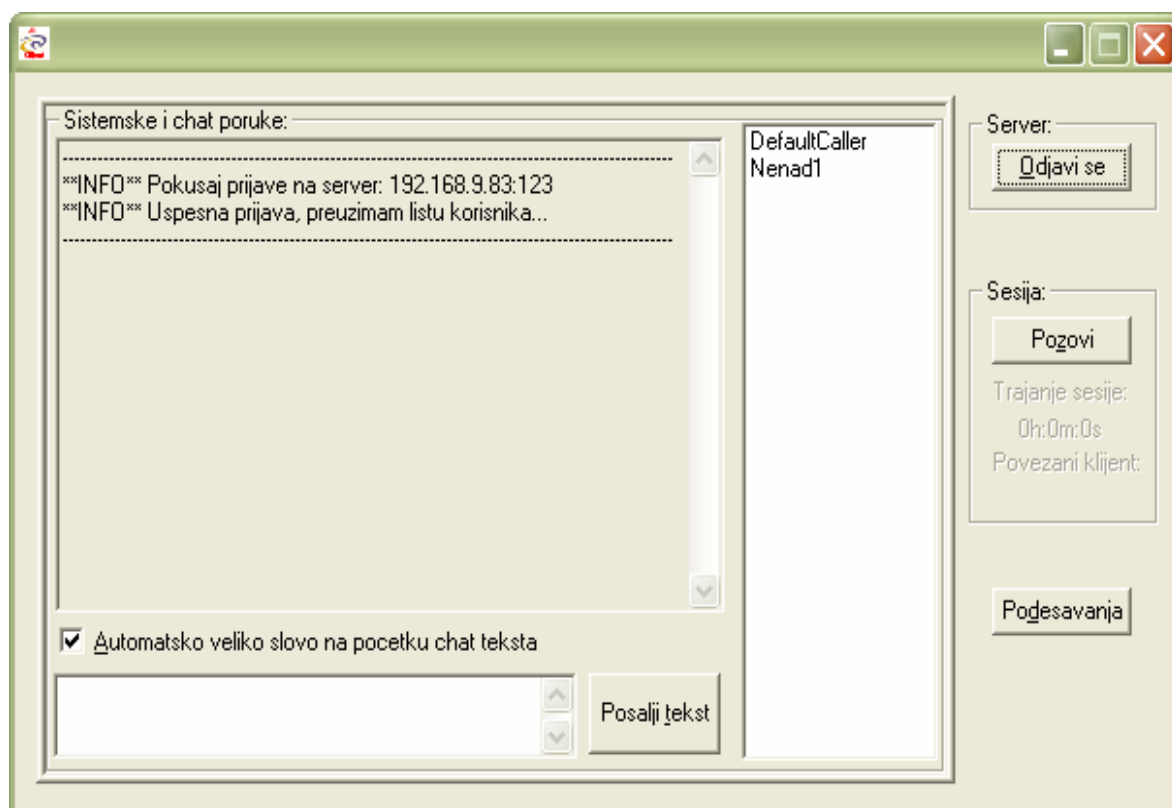
Na slici 3.2.4-3 i 3.2.4-4 prikazan je prozor za podešavanje klijentske aplikacije za PC i Pocket PC, respektivno. Na ovim slikama se vidi da je moguće podešavati IP adresu i prolaz servera na koji se klijent prijavljuje i korisničko ime klijenta pod kojim se on registruje na server. Podešavanja klijentske aplikacije moguće je obavljati u svakom trenutku rada aplikacije, ali sa napomenom da će upotreba novih parametara biti moguća tek nakon ponovnog prijavljivanja klijenta na server.

Kada je podešeno korisničko ime klijenta, i IP adresa i prolaz servera, tada se može obaviti prijava na server, i to se obavlja pritiskom na dugme **Prijavi se**. Ukoliko server postoji i prijava zajedno sa registracijom na server bude uspešno obavljena, tada se omogućuje dugme **Pozovi**, ali, takođe, dugme **Prijavi se** menja svoj naziv u **Odjavi se**, što je prikazano na slici 3.2.4-5 za PC klijentsku aplikaciju i slici 5.2.3-6 za Pocket PC klijentsku aplikaciju.

Dugme **Odjavi se** je dugme, čijim pritiskom se obavlja odjava klijenta sa servera (pritiskom na ovo dugme poziva se funkcija *OnSignInOut()* objašnjena u poglavlju [3.2.3.1.](#)).

Nakon uspešne prijave na server, server šalje klijentu spisak registrovanih klijenata, i taj spisak se upisuje u **list box kontrolu** klijentske aplikacije. Kada neki klijent želi da pozove drugog klijenta, tada taj klijent bira željenog klijenta iz **list box kontrole**. Biranje klijenta iz **list box kontrole** obavlja se pozicioniranjem pokazivaca miša PC uređaja (olovke Pocket PC uređaja) na korisničko ime onog klijenta koji se želi kontaktirati, i izvrši se njegovo selektovanje, klikom na levi taster miša PC uređaja (olovkom Pocket PC uređaja).

Nakon selektovanja korisničkog imena klijenta, pritiskom na dugme **Pozovi** taj klijent se poziva.

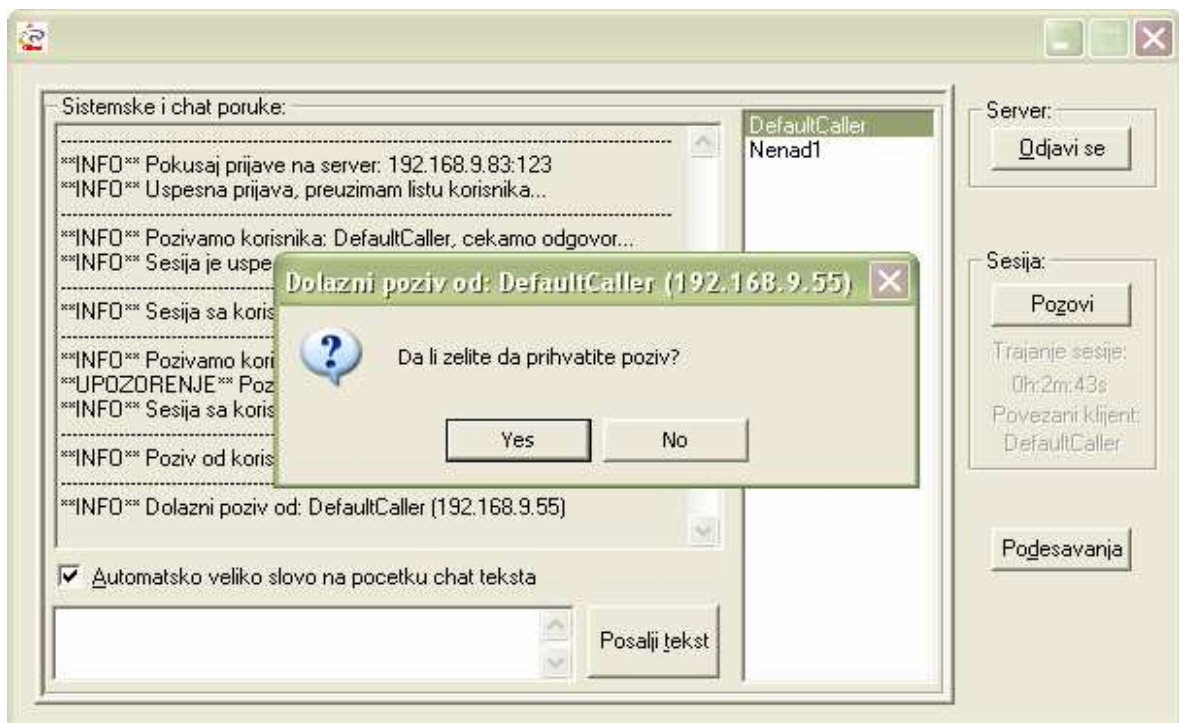


Slika 3.2.4-5: Prozor PC klijentske aplikacije nakon prijave i registrovanja na server



Slika 3.2.4-6: Prozor Pocket PC klijentske aplikacije nakon prijave i registrovanja na server

Kada je klijent pozvan na njegovom ekranu se prikazuje prozor poruke (engl. *Message Box*) kao što je prikazano na slikama 3.2.4-7 za PC i 3.2.4-8 za Pocket PC klijentsku aplikaciju.

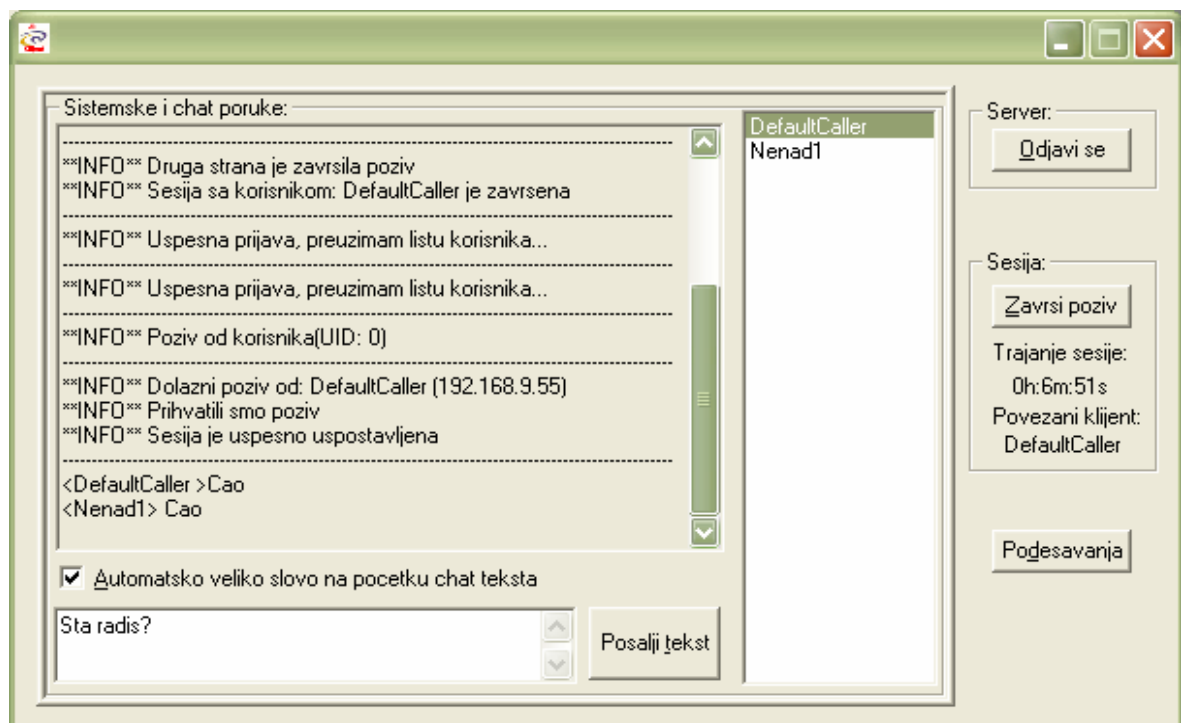


Slika 3.2.4-7: Prozor poruke PC klijentske aplikacije



Slika 3.2.4-8: Prozor poruke Pocket PC klijentske aplikacije

Kao što se vidi sa slika 3.2.4-7 i slika 3.2.4-8, pozivani klijent je upitan da li da prihvati poziv. Pritiskom na dugme **Yes** poziv se prihvata, dok se pritiskom na taster **No** poziv odbija.



Slika 3.2.4-9: Uspostavljena komunikaciona sesija - PC klijentska aplikacija



Slika 3.2.4-10: Uspostavljena komunikaciona sesija - Pocket PC klijentska aplikacija

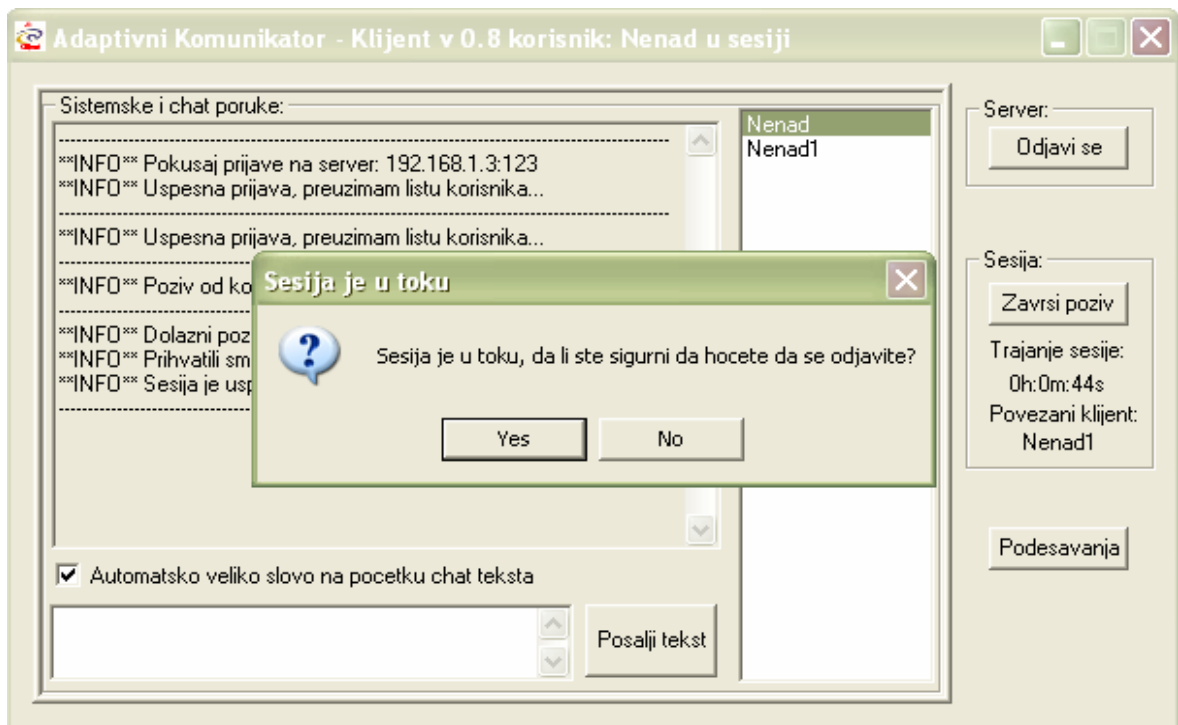
Kada je poziv prihvaćen, klijenti mogu da obavljaju komunikaciju, bilo da je u pitanju razmena tekstualnih poruka ili razgovor. Takođe, prihvatanjem poziva dugme **Pozovi** menja svoje ime u **Završi poziv** i dobija novu funkcionalnost. Tako da, pritiskom na dugme **Završi poziv** ostvarena komunikacija između klijenata se raskida (pritiskom na ovo dugme poziva se funkcija *OnEndCall()* objašnjena u poglavlju [3.2.3.1.](#)), i nakon što je komunikacija raskinuta ovo dugme vraća svoje prvobitno ime i funkcionalnost (a to je **Pozovi**). Isto se dešava i sa ranije objašnjenim dugmetom **Prijavi se - Odjavi se**).

Na slikama 3.2.4-9 i 3.2.4-10 se može videti da u oblasti označenoj kao **Sesija** postoje informacije o:

- vremenu trajanja sesija, i
- korisničkom imenu klijenta sa kojim je uspostavljena sesija.

Takođe, na pomenutim slikama se vidi i da se u **edit kontroli 1** ispisuju tekstualne poruke koje se razmenjuju između klijenata u komunikacionoj sesiji, a u **edit kontroli 2** se upisuje tekst tekstualne poruke koja će biti poslata odredišnom klijentu nakon pritiska dugmeta **Posalji tekst** ili tastera tastatuje **Enter**.

U toku rada klijentske aplikacije, klijent može uspostavljati i raskidati komunikacione sesije neograničen broj puta, i pri tome te komunikacione sesije ostvarivati prema različitim odredišnim klijentima.



Slika 3.2.4-11: Prozor poruke PC klijentske aplikacije na pritisak X;
Komunikaciona sesija je u toku



Slika 3.2.4-12: Prozor poruke Pocket PC klijentske aplikacije na pritisak **ok**;
Komunikaciona sesija je u toku

Ukoliko klijent želi da obustavi rad klijentske aplikacije, to se čini pritiskom na **X** dugme u prozoru PC klijentske aplikacije, ili na **ok** dugme u prozoru Pocket PC klijentske aplikacije. Klijent može obustaviti rad klijentske aplikacije u bilo kom trenutku rada aplikacije, bez obzira da li je u komunikacionoj sesiji ili ne.

Ukoliko je klijent bio u komunikacionoj sesiji pojaviće se prozor poruke kao na slici 3.2.4-11 za PC i slici 3.2.4-12 za Pocket PC klijentsku aplikaciju, gde se klijent pita da li je siguran da raskine komunikacionu sesiju i odjavi se sa servera i samim tim obustavi rad klijentske aplikacije. Pritiskom na dugme **Yes** klijent se odjavljuje sa servera, prekida komunikacionu sesiju i obustavlja rad klijentske aplikacije, dok pritiskom na **No** klijentska aplikacija se ponaša kao da nije ni bilo pritiska na **X** (PC) ili **ok** (Pocket PC) dugme.

4. SERVERSKO-KLIJENTSKI SISTEM - ZAHTEVI I OGRANIČENJA

U ovom poglavlju će biti opisani problemi koji se nameću u radu serversko-klijentskog sistema.

4.1. Performanse uređaja

Nametnuti problem je potreba za performansama uređaja na kojima se izvršava aplikacija, bilo ona klijentska ili serverska. Server zahteva performanse uređaja koje mogu da podrže obradu ogromne količine podataka, istovremeno, gde se ti podaci razmenjuju između velikog broja klijenata. Dakle, performansa serverskog uređaja u komunikacionom sistemu mora da bude iznad prosečne performanse, odnosno uređaj treba da bude konstruisan od najbržih komponenti koje se mogu naći na današnjem tržištu, pri čemu se akcenat stavlja na brze procesore i brzu memoriju kako bi se ostvarila velika brzina obrade podataka. Sporom obradom podataka na serverskoj strani, narušava se transparentnost servera u komunikaciji serversko-klijentskog sistema, odnosno sporost obrade podataka uzrokuje kašnjenja poruka u toku njihove razmene između klijenata i servera, bilo da su u pitanju poruke registracije, SIP pregovaranja, ili da je u pitanju prenos tekstualnih poruka ili govora. Za razliku od servera, klijent zahteva uređaj performansi današnjih SMART uređaja, kao što su: SMART telefon, PDA, Pocket PC (džepni PC) i slično.

4.2. Mreža - Brzina i propusni opseg

Osim potrebe za performansama uređaja, na serversko-klijentski sistem utiče mrežni priključak na koji su komunikacione strane priključene. Mrežni priključak na kojem se nalaze klijenti ili serveri ima ogromni uticaj na kvalitet prenosa podataka koji se ostvaruje u realnom vremenu u toku komunikacione sesije. Tako da, ukoliko mrežni priključak obezbeđuje manju brzinu protoka podataka u mreži od one brzine potrebne za prenos podataka, tada će doći do pojave kašnjenja u prenosu podataka između strana u komunikaciji, dok u suprotnom, kad mrežni priključak obezbeđuje dovoljnu brzinu protoka podataka (ili brzinu veću od dovoljne), problem kašnjenja se uglavnom neće javiti. Ovaj problem najviše utiče na prenos podataka u realnom vremenu. Njegov najveći uticaj u serversko-klijentskom sistemu ispoljava se prilikom obavljanja VoIP razgovora između klijenata, jer se problem manifestuje kao nerazumljiv razgovor sa prekidima u obliku tišina, gde su česti i kratki prekidi rezultat oscilacija u promeni brzina protoka podataka, dok su duži prekidi rezultat velikih kašnjenja u prenosu podataka.

4.3. Mreža - “Skrivenost” klijenta

Kao još jedan od problema tokom rada serversko-klijentskog sistema javlja se problem “skrivenosti” komunikacionih strana u mreži, gde se “skrivenost” u mreži odnosi na to da se uređaji nalaze iza NAT servera, zastupničkih servera, iza firewall-ova i slično. Neophodno je da server bude dostupan od strane svih klijenata u mreži tako da ima javnu IP adresu. Klijenti se uglavnom nalaze u okviru privatne mreže, odnosno mreže zatvorenog tipa, gde imaju privatne adrese, a na Internet izlaze preko nekog servera (recimo, NAT servera) koji im pridružuje jedinstvenu IP adresu, stoga server u okviru serversko-klijentskog sistema mora podržavati mehanizam prepoznavanja klijenata. Takođe, klijentima u okviru nekih mreža zabranjeno je da prihvataju zahteve koji dolaze sa mreže, odnosno da se ponašaju kao serveri, to jest da imaju mogućnost slušanja na prolazima. (više o “skrivenosti” klijenta pogledati [7. \[5\], \[6\] i \[7\]](#))

4.4. Prenos koda na drugu platformu

Treba napomenuti i problem sa emulatorom za **Windows Mobile 5.0**. Osim za PC uređaj sa operativnim sistemom Windows XP, klijentska aplikacija je pisana i za *Windows Mobile 5.0*. Stoga je programski kod klijentske aplikacije spušan, kako na Pocket PC emulator za *Windows Mobile 5.0*, tako i na pravi Pocket PC uređaj sa *Windows Mobile 5.0*. U toku rada klijentske aplikacije utvrđeno je da se programski kod klijentske aplikacije spuštene na pravom uređaju bolje ponaša u odnosu na programski kod iste aplikacije na emulatoru. Razlike u radu klijentske aplikacije na emulatoru i pravom Pocket PC uređaju su sledeće:

- programski kod klijentske aplikacije se brže izvršava na pravom uređaju nego na emulatoru,
- na emulatoru je, zbog sporog izvršavanja programskog koda, dolazilo do kratkotrajnih prekida (seckanja) u razgovoru, a i tekstualne poruke su kasnile, dok se na pravom uređaju to nije dešavalo.
- na emulatoru su postojali problemi u komunikaciji sa mrežom, te je bilo neophodno naknadno podešavanjem Pocket PC emulatora da radi sa mrežnim protokolima, TCP/IP i UDP. Rad TCP/IP protokola je podešen relativno brzo, ali rad UDP protokolom je bilo veoma teško podesiti, i to podešavanje je oduzelo veoma mnogo vremena. Ovi problemi nisu postojali na pravom Pocket PC uređaju. Takođe, na pravom Pocket PC uređaju je bilo nekih podešavanja, ali ta podešavanja su bila relativno jednostavna i nisu oduzela toliko vremena koliko je oduzelo podešavanje mreže na Pocket PC emulatoru sa *Windows Mobile 5.0*.

Problemi sa UDP protokolom su se javili iz razloga što program *Microsoft ActiveSync* (4.5 i ranije verzije), koji omogućava emulatoru da se poveže sa mrežim adapterom PC uređaja (na pomenutom PC uređaju je pokrenut emulator), ne podržava UDP protokol, nego samo TCP/IP (ovo je dokumentovano na *Microsoft*-ovoj web stranici [7. \[8\]](#)). Tako da se za rad sa UDP protokolom morao instalirati novi softver *Virtual PC 2007* (takođe, je dokumentovano na *Microsoft*-ovoj web stranici [7. \[8\]](#)), koji omogućava stvaranje virtuelnog mrežnog adaptera preko koga Pocket PC emulator može da komunicira sa mrežom, ostvarujući pri tome i TCP/IP i UDP protokol. Međutim, egzistencija samog procesa *Microsoft ActiveSync* programa dolazila je u koliziju sa radom virtuelnog mrežnog adaptera, što je za posledicu imalo nepravilan rad virtuelnog mrežnog adaptera, tako da se morao obustaviti rad tog procesa, pri čemu je ime procesa *wcescomm.exe*.

Drugi problem sa emulatorom je taj što klijent, pokrenut na emulatoru, nije mogao uspostaviti vezu (engl. connection) sa serverom koji je pokrenut na istom PC uređaju kao i klijent, jer emulator nije hteo uspostavljati vezu sa *localhost* adresom servera, te je programski kod servera izvršavan na drugom PC uređaju. Tek premeštanjem programskog koda servera na drugi PC uređaj, klijent pokrenut na Pocket PC emulatoru je uspostavio vezu sa serverom.

5. SERVERSKO-KLIJENTSKI SITSTEM - ISPITIVANJE

Ispitivanje rada serversko-klijentskog sistema obavljano testiranjem funkcionalnosti aplikacija servera i klijenta, kao i podataka koje oni međusobno razmenjuju (poruke protokola, tekstualne poruke ili audio podaci razgovora).

Takođe, kako se prenos i razmena podataka obavlja putem Interneta, ispitivanje rada serversko-klijentskog sistema je obavljano korišćenjem različitih tipova mreža za prenos podataka i to: LAN (engl. *Local Area Network*), GPRS (engl. *General Packet Radio Service*) i WiFi (engl. *Wireless-Fidelity*).

U ovom poglavlju biće prikazano samo ispitivanje serversko-klijentskog sistema po metodu sive kutije (engl. *grey-box testing*). Metod sive kutije podrazumeva da tester ima pristup samom programskom kodu, ali da je testiranje obavljeno na nivou crne kutije (engl. *black-box testing*), odnosno na korisničkom nivou.

5.1. Ispitivanje - Klijent

Testovi koji su se obavljali prilikom ispitavanja klijenta su prikazani u Tabeli 5.1-1

Tabela 5.1-1 Testovi klijentske aplikacije

Testovi					
Test	Opis testa	Prethodni uslovi	Akcija	Očekivani rezultat	Rezultat
1	Startovanje klijentske aplikacije	Klijentska aplikacija nije pokrenuta	Pokretanje izvršne datoteke	Na ekranu uređaja se iscrtaava prozor klijentske aplikacije.	Uspešan
2	Izgled glavnog prozora klijentske aplikacije	Klijentska aplikacija je pokrenuta	/	Prikazan glavni prozor klijentske aplikacije. Izgled prozora kao sa slike 3.2.4-1 (slike 3.2.4-2)	Uspešan
3	Dialog za podešavanje parametara klijentske aplikacije	Klijentska aplikacija je pokrenuta	Pritisak na dugme "Podesavanja"	Prikazan dialog prozor za podešavanje klijentske aplikacije. Izgled prozora kao sa slike 3.2.4-3 (slike 3.2.4-4)	Uspešan
4	Dialog za podešavanje parametara klijentske aplikacije	Klijentska aplikacija je pokrenuta	1. Pritisak na dugme "Podesavanja" 2. Upis parametara klijentske aplikacije (Umesto korisničkog imena DefaultCaller upisati TestCaller) 3. Pritisak na dugme "U redu"	Na glavnom prozoru klijentske aplikacije ispisuje se poruka: ***INFO** Odredisni server je: 192.168.1.3:123, korisnik: TestCaller	Uspešan

5	Dialog za podešavanje parametara klijentske aplikacije	Klijentska aplikacija je pokrenuta	1. Pritisak na dugme "Podesavanja" 2. Upis parametara klijentske aplikacije (Umesto korisničkog imena DefaultCaller upisati TestCaller) 3. Pritisak na dugme "Otkazi"	Na glavnom prozoru klijentske aplikacije nema promena, ime korisnika nije promenjeno.	Uspešan
6	Registracija klijenta kada server nije u mreži	Klijentska aplikacija je pokrenuta	Pritisak na dugme "Prijavi se"	Na glavnom prozoru klijentske aplikacije ispisuje se poruka: ***GRESKA** Server nije online, ne mozemo se prijaviti	Uspešan
7	Registracija klijenta kada je server u mreži	Klijentska aplikacija je pokrenuta	Pritisak na dugme "Prijavi se"	1. Na glavnom prozoru klijentske aplikacije ispisuje se poruka ***INFO** Uspešna prijava, preuzimam listu korisnika... . 2. U kontroli list box nalazi se spisak registrovanih klijenata na serveru. 3. Izgled prozora klijentske aplikacije kao sa slike 3.2.4-5 (slike 3.2.4-6). Dugme "Prijavi se" menja naziv u "Odjavi se"	Uspešan
8	Registracija klijenta kada je server u mreži, pri čemu je klijent izabrao za korisničko ime, ono korisničko ime koje već postoji na serveru.	1. Klijentska aplikacija je pokrenuta 2. Na serveru postoji korisni sa korisničkim imenom "TestCaller"	Pritisak na dugme "Prijavi se"	Na glavnom prozoru klijentske aplikacije ispisuje se poruka ***GRESKA** Korisnik sa korisničkim imenom TestCaller već postoji na serveru, pokušajte ponovo sa drugim korisničkim imenom .	Uspešan
9	Pozivanje samog sebe	1. Klijentska aplikacija je pokrenuta 2. Uspešna prijava klijenta na server pod korisničkim imenom "TestCaller" 3. Klijent poziva sam sebe	1. Izbor liste iz klijenta "TestCaller" 2. Pritisak na dugme "Pozovi"	Na glavnom prozoru klijentske aplikacije ispisuje se poruka ***UPOZORENJE** Nema svrhe pozivati samog sebe .	Uspešan
10	Klijenta se poziva. Odbijen poziv.	1. Klijentska aplikacija je pokrenuta 2. Uspešna prijava klijenta na server pod korisničkim imenom "TestCaller". Na serveru se nalazi klijent sa korisničkim imenom "DefaultCaller". 3. Klijent poziva klijenta sa korisničkim imenom "DefaultCaller"	1. Izbor liste iz klijenta "DefaultCaller" 2. Pritisak na dugme "Pozovi"	Na glavnom prozoru klijentske aplikacije ispisuje se poruka: ***INFO** Pozivamo korisnika: DefaultCaller, cekamo odgovor... **UPOZORENJE** Pozvani klijent je odbio poziv ***INFO** Sesija sa korisnikom: DefaultCaller je završena	Uspešan
11	Klijenta se poziva. Prihvaćen poziv.	1. Klijentska aplikacija je pokrenuta 2. Uspešna prijava klijenta na server pod korisničkim imenom "TestCaller". Na serveru se nalazi klijent sa korisničkim imenom "DefaultCaller". 3. Klijent poziva klijenta sa korisničkim imenom "DefaultCaller"	1. Izbor liste iz klijenta "DefaultCaller" 2. Pritisak na dugme "Pozovi"	Na glavnom prozoru klijentske aplikacije ispisuje se poruka: ***INFO** Pozivamo korisnika: DefaultCaller, cekamo odgovor... ***INFO** Sesija je uspesno uspostavljena	Uspešan

12	Dialog za prihvrat poziva. Odbijen poziv.	1. Klijentska aplikacija je pokrenuta 2. Uspješna prijava klijenta na server pod korisničkim imenom "TestCaller". Na serveru se nalazi klijent sa korisničkim imenom "DefaultCaller". 3. Klijent sa korisničkim imenom "DefaultCaller" poziva "TestCaller" klijenta	Pritisak na dugme "No"	1. Na ekranu uređaja prikazan je dialog prozor kao na slici 3.2.4-7 (slike 3.2.4-8) 2. Na glavnom prozoru klijentske aplikacije ispisuje se poruka: ***INFO** Dolazni poziv od: DefaultCaller (192.168.1.3) ***INFO** Odbili smo poziv ***INFO** Sesija nije uspostavljena	Uspešan
13	Dijalog za prihvrat poziva. Prihvaćen poziv.	1. Klijentska aplikacija je pokrenuta 2. Uspješna prijava klijenta na server pod korisničkim imenom "TestCaller". Na serveru se nalazi klijent sa korisničkim imenom "DefaultCaller". 3. Klijent sa korisničkim imenom "DefaultCaller" poziva "TestCaller" klijenta	Pritisak na dugme "Yes"	1. Na ekranu uređaja prikazan je dialog prozor kao na slici 3.2.4-7 (slike 3.2.4-8) 2. Na glavnom prozoru klijentske aplikacije ispisuje se poruka: ***INFO** Dolazni poziv od: TestCaller (192.168.1.3) ***INFO** Prihvatili smo poziv ***INFO** Sesija je uspesno uspostavljena	Uspešan
14	Izgled glavnog prozora klijentske aplikacije nakon uspostavljene sesije	Uspostavljena komunikaciona sesija	/	Na ekranu uređaja prikazan je prozor kao na slici 3.2.4-9 (slike 3.2.4-10)	Uspešan
15	Slanje tekstualnih poruka.	Uspostavljena komunikaciona sesija	1. Unos teksta u edit kontrolu 2. Pritisak na dugme "Posalji tekst"	Na glavnom prozoru klijentske aplikacije ispisuje se tekst poruke	Uspešan
16	Razgovor	Uspostavljena komunikaciona sesija	/	Klijenti međusobno razgovaraju, bez ikakvih smetnji	Uspešan
17	Klijent je završio razgovor.	1. Uspostavljena je komunikaciona sesija 2. "TestClient" je završio razgovor	Pritisak na dugme "Završi poziv"	Na glavnom prozoru klijentske aplikacije ispisuje se: ***INFO** Sesija sa korisnikom: DefaultCaller je završena 2. Dugme "Završi poziv" menja naziv u "Pozovi"	Uspešan
18	Završen razgovor od strane drugog klijenta	1. Uspostavljena je komunikaciona sesija 2. "DefaultClient" je završio razgovor	Pritisak na dugme "Završi poziv"	Na glavnom prozoru klijentske aplikacije ispisuje se: ***INFO** Sesija sa korisnikom: TestCaller je završena 2. Dugme "Završi poziv" menja naziv u "Pozovi"	Uspešan
19	Klijent se odjavio sa servera.	Klijent je uspešno prijavljen na server	Pritisak na dugme "Odjavi se"	Na glavnom prozoru klijentske aplikacije ispisuje se: ***INFO** Odjavili smo se sa servera: 192.168.1.3:123"	Uspešan
20	Klijent se odjavio sa servera u toku sesije	1. Uspostavljena komunikaciona sesija 2. Odjava sa servera.	1. Pritisak na dugme "Odjavi se" 2. Pritisak na dugme "Yes" ili "No"	1. Prikazan dialog prozor za raskid komunikacione sesije. Izgled prozora kao sa slike 3.2.4-11 (slike 3.2.4-12) 2. a) "Yes" klijent se odjavljuje sa servera i komunikaciona sesija se završava. Ispisuje se: ***INFO** Sesija sa korisnikom: TestCaller je završena ***INFO** Odjavili smo se sa servera: 192.168.1.3:123" b) "No" klijent se ne odjavljuje sa servera i ne raskida se komunikaciona sesija	1. Uspešan 2. a) Uspešan b) Uspešan

21	Drugi klijent se odjavio sa servera u toku sesije	1. Uspostavljena komunikaciona sesija 2. Odjava klijenta sa kojim smo uspostavili komunikacionu sesiju sa servera.	Pritisak na dugme "Odjavi se"	Na glavnom prozoru klijentske aplikacije ispisuje se: ***GRESKA** Druga strana se nasilno isključila sa servera **INFO** Sesija sa korisnikom: DefaultCaller je završena"	Uspešan
22	Gašenje Aplikacije. Klijent nije prijavljen na server.	Klijentska aplikacija je pokrenuta	Pritisak na dugme "X"	Aplikacija se gasi	Uspešan
22	Gašenje Aplikacije. Klijent prijavljen na server.	Uspešna prijava klijenta na server	Pritisak na dugme "X"	Aplikacija se gasi, pri čemu se klijent odjavljuje sa servera.	Uspešan
23	Gašenje Aplikacije. Uspostavljena komunikaciona sesija	Uspostavljena komunikaciona sesija	Pritisak na dugme "X"	Očekivani rezultat je kao u testu 20, sa tom izmenom da se u 2. a) aplikacija gasi.	Uspešan

Testovi su sprovedeni na različitim platformama: PC uređaju, emulatoru Pocket PC uređaja sa operativnim sistemom Windows Mobile 5.0 i Pocket PC uređaju sa operativnim sistemom Windows Mobile 5.0.

5.2. Ispitivanje - Server

Testovi koji su se obavljali prilikom ispitavanja klijenta su prikazani u Tabeli 5.2-1

Tabela 5.2-1 Testovi serverske aplikacije

Testovi					
Test	Opis testa	Prethodni uslovi	Akcija	Očekivani rezultat	Rezultat
1	Startovanje serverske aplikacije	Serverska aplikacija nije pokrenuta	Pokretanje izvršne datoteke	Na ekranu uređaja se iscrta prozor serverske aplikacije.	Uspešan
2	Izgled prozora serverske aplikacije	Serverska aplikacija je pokrenuta	/	Prikazan glavni prozor serverske aplikacije. Izgled prozora kao sa slike 3.1.4-1	Uspešan
2	Registracija klijenta	Serverska aplikacija je pokrenuta	/	Na glavnom prozoru serverske aplikacije ispisuje se poruka "INFO: Klijentska IP adresa: 192.168.1.3 DEBUG: Accepting a TCP socket DEBUG: NETReceiveNit pokrenuta! DEBUG: Priljen registracioni header: REGISTER:DefaultCaller INFO: Korisnik DefaultCaller je registrovan u bazi korisnika sa UID brojem: 1 DEBUG: Kreirana nit za refresh user liste DEBUG: Slanje liste svim korisnicima zbog prijave novog korisnika"	Uspešan
3	Registracija klijenta kada je server u mreži, pri čemu je klijent izabrao za korisničko ime, ono korisničko ime koje već postoji na serveru.	1. Serverska aplikacija je pokrenuta 2. Na serveru postoji korisni sa korisničkim imenom "DefaultCaller"	/	Na glavnom prozoru serverske aplikacije ispisuje se poruka "INFO: Klijentska IP adresa: 192.168.1.3 DEBUG: Accepting a TCP socket DEBUG: NETReceiveNit pokrenuta! DEBUG: Priljen registracioni header: REGISTER:DefaultCaller ERROR: Korisnik nije registrovan jer takav korisnik (DefaultCaller) vec postoji u bazi DEBUG: NETReceiveNit završena!"	Uspešan

4	Klijent se poziva. Odbijen poziv.	1. Serverska aplikacija je pokrenuta 2. Na serveru se nalaze klijenti sa korisničkim imenom "DefaultCaller" i "TestCaller". 3. Klijent "TestCaller" poziva klijenta sa korisničkim imenom "DefaultCaller"	/	1. Na glavnom prozoru klijentske aplikacije ispisuje se poruka: "INFO: Klijentska IP adresa: 192.168.1.3" DEBUG: Accepting a TCP socket DEBUG: NETReceiveNit pokrenuta! INFO: Klijentska IP adresa: 192.168.1.3 DEBUG: Accepting a TCP socket DEBUG: NETReceiveNit pokrenuta! DEBUG: Primljen header za resolving UID to IP *Received -> INVITE sip:DefaultCaller@192.168.1.3:0 SIP/2.0 *Received -> CANCEL sip:TestCaller@192.168.1.3:123 SIP/2.0 *Received -> SIP/2.0 200 OK"	Uspešan
5	Klijent se poziva. Prihvaćen poziv. Sesija je uspostavljena.	1. Serverska aplikacija je pokrenuta 2. Na serveru se nalaze klijenti sa korisničkim imenom "DefaultCaller" i "TestCaller". 3. Klijent "TestCaller" poziva klijenta sa korisničkim imenom "DefaultCaller"	/	1. Na glavnom prozoru klijentske aplikacije ispisuje se poruka: "INFO: Klijentska IP adresa: 192.168.1.3" DEBUG: Accepting a TCP socket DEBUG: NETReceiveNit pokrenuta! INFO: Klijentska IP adresa: 192.168.1.3 DEBUG: Accepting a TCP socket DEBUG: NETReceiveNit pokrenuta! DEBUG: Primljen header za resolving UID to IP *Received -> INVITE sip:DefaultCaller@192.168.1.3:0 SIP/2.0 *Received -> SIP/2.0 100 Trying *Received -> SIP/2.0 180 Ringing *Received -> SIP/2.0 200 OK *Received -> ACK sip:DefaultCaller@192.168.1.3:0 SIP/2.0 INFO: Identifikovan DESTINATION CLIENT - UID:1 INFO: Identifikovan SORURCE CLIENT - UID:0"	Uspešan
6	Klijent je završio razgovor.	1. Uspostavljena je komunikaciona sesija 2. Jedan od klijenata je završio razgovor	/	1. Na glavnom prozoru klijentske aplikacije ispisuje se: "*Received -> BYE sip:TestCaller@192.168.1.3:123 SIP/2.0" *Received -> SIP/2.0 200 OK INFO: Klijent 192.168.1.3 se diskonetkovao. DEBUG: Slanje liste svim korisnicima zbog odjave korisnika/kraja sesije DEBUG: NETReceiveNit zavrshena! DEBUG: Slanje liste svim korisnicima zbog odjave korisnika/kraja sesije DEBUG: NETReceiveNit zavrshena! INFO: Klijent 192.168.1.3 se diskonetkovao."	Uspešan
7	Klijent se odjavio sa servera.	Klijent je uspešno prijavljen na server (klijent može biti i u komunikacionoj sesiji)	/	Na glavnom prozoru serverske aplikacije ispisuje se: "INFO: Klijent 192.168.1.3 se diskonetkovao. INFO: Korisnik sa UID brojem 0 je uklonjen iz baze korisnika DEBUG: Slanje liste svim korisnicima zbog odjave korisnika/kraja sesije DEBUG: NETReceiveNit zavrshena! DEBUG: Kraj niti za refresh user liste"	Uspešan
8	Opcija glavnog menija serverske aplikacije "L"	Na serveru se nalaze klijenti sa korisničkim imenom "DefaultCaller" i "TestCaller".	Pritisak na taster tastature "L"	Na glavnom prozoru serverske aplikacije ispisuje se: "Lista registrovanih korisnika: (UID: Username, IP adresa, UDP port, TCP port) =====	Uspešan
9	Opcija glavnog menija serverske aplikacije "A". Dodat je klijent "User".	Na serveru se nalaze klijenti sa korisničkim imenom "DefaultCaller" i "TestCaller".	1. Pritisak na taster tastature "A" i unos parametara koji se traže 2. Pritisak na taster tastature "L"	Na glavnom prozoru serverske aplikacije ispisuje se: "Lista registrovanih korisnika: (UID: Username, IP adresa, UDP port, TCP port) =====	Uspešan

10	Opcija glavnog menija serverske aplikacije "U". Uklonjen je klijent "User".	Na serveru se nalaze klijenti sa korisničkim imenom "DefaultCaller", "TestCaller" i "User".	1. Pritisak na taster tastature "U" i unos parametara koji se traže 2. Pritisak na taster tastature "L"	Na glavnom prozoru serverske aplikacije ispisuje se: "Lista registrovanih korisnika: (UID: Username, IP adresa, UDP port, TCP port)" ===== 1: DefaultCaller, 192.168.1.3, X, X 0: TestCaller, 192.168.1.3, X, X ===== kraj liste ===== INFO: Baza sadrzi 2 korisnika."	Uspešan
11	Opcija glavnog menija serverske aplikacije "F". Uklonjeni su svi klijenti sa servera.	Na serveru se nalaze klijenti sa korisničkim imenom "DefaultCaller", "TestCaller" i "User".	1. Pritisak na taster tastature "F" 2. Pritisak na taster tastature "L"	Na glavnom prozoru serverske aplikacije ispisuje se: "Lista registrovanih korisnika: (UID: Username, IP adresa, UDP port, TCP port)" ===== kraj liste ===== INFO: Baza sadrzi 0 korisnika."	Uspešan
12	Opcija glavnog menija serverske aplikacije "C"	Server je pokrenut	Pritisak na taster tastature "C"	Sav tekst na glavnom prozoru serverske aplikacije je obrisani i ispisani samo glavni meni	Uspešan
13	Opcija glavnog menija serverske aplikacije "M"	Server je pokrenut	Pritisak na taster tastature "M"	Ispisani su glavni meni.	Uspešan
14	Gašenje Aplikacije.	Serverska aplikacija je pokrenuta	Pritisak na dugme "X" ili "Q"	Aplikacija se gasi	Uspešan

6. ZAKLJUČAK

Raširenost Interneta, kako u privatnoj, tako i u poslovnoj sferi ljudskog života, kao i njegova relativno niska cena, bili su glavni razlozi stvaranja ideje i pokretačka snaga za nastanak i razvoj serversko-klijentskog sistema.

Razvijanjem programske podrške serversko-klijentskog sistema zasnovanog na VoIP tehnologiji dovelo je do sticanja neprocenjivog iskustva. Neka od najznačajnijih iskustava stečenih prilikom rada na serversko-klijentskom sistemu su:

- način konstruisanja i pisanja aplikacija servera i klijenta,
- upoznavanje i rad sa različitim mrežnim protokolima (TCP/IP, UDP, SIP) i njihovo objedinjavanje i integracija u jednom sistemu,
- korišćenje različitih alata i biblioteka za razvoj programske podrške (MFC biblioteka i Winsock biblioteka, Microsoft Visual Studio 6.0 i Microsoft Visual Studio 2005),
- upoznavanje i rad sa različitim računarskim platformama (PC i Pocket PC) sa različitim operativnim sistemima (Windows XP i Windows Mobile 5.0), kao i sa emulatorom Pocket PC uređaja sa Windows Mobile 5.0 operativnim sistemom,
- suočavanje sa raznim problemima koji su nastajali prilikom razvoja programske podrške i pronalaženje načina za njihovo razrešavanje,
- upoznavanje i rad sa AMR kodekom.

Primena serversko-klijentskog sistema je moguća na svim lokacijama gde komunikacija igra primarnu ulogu, i naravno, gde postoji Internet.

Serversko-klijentski sistem prikazan ovim radom predstavlja polaznu osnovu za razvoj nekog mnogo složenijeg sistema. Budući razvoj serversko-klijentskog sistema podrazumevao bi implementaciju:

- podrške za ostvarivanje audio-vizelne konferencije,
- podrške za algoritam prilagođavanja promenama u brzini prenosa podataka u mrežama različitog tipa,
- podršku slanja elektronske pošte,
- gledanja TV programa putem Interneta, i slično.

7. LITERATURA

- [1] VOIP - <http://www.fcc.gov/voip/>
- [2] SIP - RFC 3261
- [3] AMR - <http://www.digitalpreservation.gov/formats/fdd/fdd000254.shtml>
- [4] Source signal based rate adaptation for GSM AMR speech codec [.pdf]
- [5] Peer-to-Peer Communication Across Network Address Translators [.pdf]
- [6] NAT and VoIP - <http://www.voip-info.org/wiki-NAT+and+VOIP>
- [7] NAT and Peer-to-peer networking - <http://alumnus.caltech.edu/~dank/peer-nat.html>
- [8] Microsoft Documentation - <http://msdn2.microsoft.com/en-us/default.aspx>