SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

**Matija Belec**

**Dario Grubešić**

**Mario Hudinčec**

**Kristijan Jedvaj**

**Vibor Kovačić**

**Marko Lesničar**

POSLUŽITELJ ZA PRONALAŽENJE

PROJEKTNA DOKUMENTACIJA

Varaždin, 2015.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Belec

Dario Grubešić

Mario Hudinčec

Kristijan Jedvaj

Vibor Kovačić

Marko Lesničar

POSLUŽITELJ ZA PRONALAŽENJE

PROJEKTNA DOKUMENTACIJA

Mentor:

Dr. sc. Nikola Ivković

Varaždin, siječanj 2015.

**Sadržaj**

[1. Uvod 1](#_Toc410088699)

[2. Serverski dio programskog koda 2](#_Toc410088700)

[2.1. Okvirni opis rada servera 2](#_Toc410088701)

[2.2. Priprema servera 4](#_Toc410088702)

[2.3. Pokretanje dretvi 4](#_Toc410088703)

[2.4. Prihvaćanje poruke 5](#_Toc410088704)

[2.4.1. Kritični odsječak 1 – dobavljanje ID-a praznog buffer-a 6](#_Toc410088705)

[2.4.2. Kritični odsječak 2 – spremanje ID-a punog buffer-a 6](#_Toc410088706)

[2.5. Obrada poruke – dretva za obradu poruke 7](#_Toc410088707)

[2.5.1. Slanje odgovora na poruku 8](#_Toc410088708)

[3. Konfiguracijske datoteke 9](#_Toc410088709)

[3.1. Struktura konfiguracijske datoteke servera 9](#_Toc410088710)

[3.2. Struktura datoteke sa popisom relay-a 9](#_Toc410088711)

[3.3. Učitavanje parametara iz konfiguracijske datoteke 10](#_Toc410088712)

[3.4. Struktura popisa relay-a unutar programskog koda 10](#_Toc410088713)

[3.5. Učitavanje popisa relay-a iz datoteke 11](#_Toc410088714)

[3.5.1. Priprema poruke MSG\_RELAY\_LIST 11](#_Toc410088715)

[4. Struktura poruke 12](#_Toc410088716)

[4.1. Struktura poruke 12](#_Toc410088717)

[4.2. Tipovi poruka 12](#_Toc410088718)

[4.3. Funkcije za pretvaranje poruke 14](#_Toc410088719)

[4.3.1. Pretvaranje poruke u strukturirani zapis 14](#_Toc410088720)

[4.3.2. Pretvaranje poruke iz strukturiranog zapisa 15](#_Toc410088721)

[5. Privremena 'baza' zapisa 17](#_Toc410088722)

[5.1. Struktura zapisa u bazi 17](#_Toc410088723)

[5.2. Pristup bazi 17](#_Toc410088724)

[5.2.1. Dodavanje zapisa u bazu 19](#_Toc410088725)

[5.2.2. Preuzimanje podataka o zapisu iz baze 19](#_Toc410088726)

[5.2.3. Brisanje zapisa iz baze 19](#_Toc410088727)

[6. Testiranje servera 21](#_Toc410088728)

[6.1. Karakteristike servera 21](#_Toc410088729)

[6.2. Test klijent 22](#_Toc410088730)

[6.3. Unaprijeđenje servera 22](#_Toc410088731)

[7. Interna interakcija na serveru 23](#_Toc410088732)

[7.1. Dostupne naredbe 23](#_Toc410088733)

[7.1.1. Naredba 'baza' 23](#_Toc410088734)

[7.1.2. Naredba 'prekid' 23](#_Toc410088735)

1. Uvod

Za izradu programskog koda korišten je programski jezik C++ sa novim standardom jezika poznatijom pod nazivom C++11 (ili C++0x).

Programski kod vezan uz serverski dio komunikacije je osmišljen za višedretveni rad s mogućnošću određivanja broja dretvi unutar konfiguracijske datoteke. Svaka dretva predstavlja jednu obradu zahtjeva koji se sastoji od tri dijela:

* prihvaćanje poruke
* obrada poruke
* slanje odgovora ako je potreban odgovor na primljenu poruku

Unutar programa je napravljena i manja „baza“ koja je implementirana pomoću strukture std::map koja služi za spremanje identifikatora toka i potrebnih podataka vezanih uz njih.

Implementirane su strukture za svaku od traženih tipova poruka, te funkcije za pretvaranje poruke iz strukturiranog zapisa u niz znakova (*std::string*/*unsigned char\**) i obrnuto.

Server također ima i svoju vrstu interakcije koja je izvedena unutar konzole. Dostupno je nekoliko naredbi koje služe za testiranje pravilnog rada servera. Detaljniji opis naredbi nalazi se u poglavlju posvećenom internoj interakciji sa serverom.

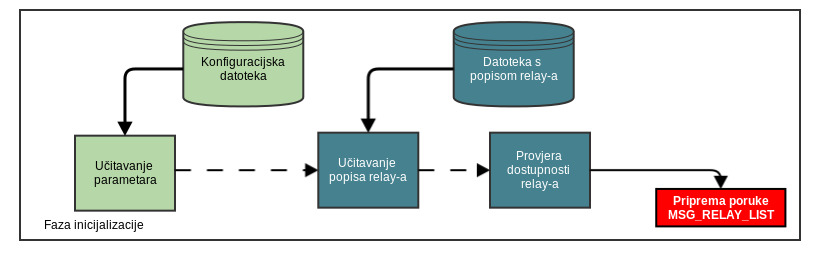
1. Serverski dio programskog koda
   1. Okvirni opis rada servera

Server se sastoji od dvije faze:

1. Faza inicijalizacije
2. Faza radnog režima

Kod faze inicijalizacije očitavaju se parametri iz konfiguracijske datoteke, slijedi učitavanje popisa relay-a. Svakom se poslužitelju (relay-u) sa popisa provjerava njegova dostupnost. Nakon što su svi poslužitelji provjereni – svi dostupni poslužitelji se upakiravaju u poruku tipa MSG\_RELAY\_LIST, te se ista sprema u globalnu varijablu. Kod svake primljene poruke tipa MSG\_REQ\_RELAY\_LIST, upravo se navedena spremljena poruka koristi kao odgovor.

Faza inicijalizacije servera prikazana je slikom:

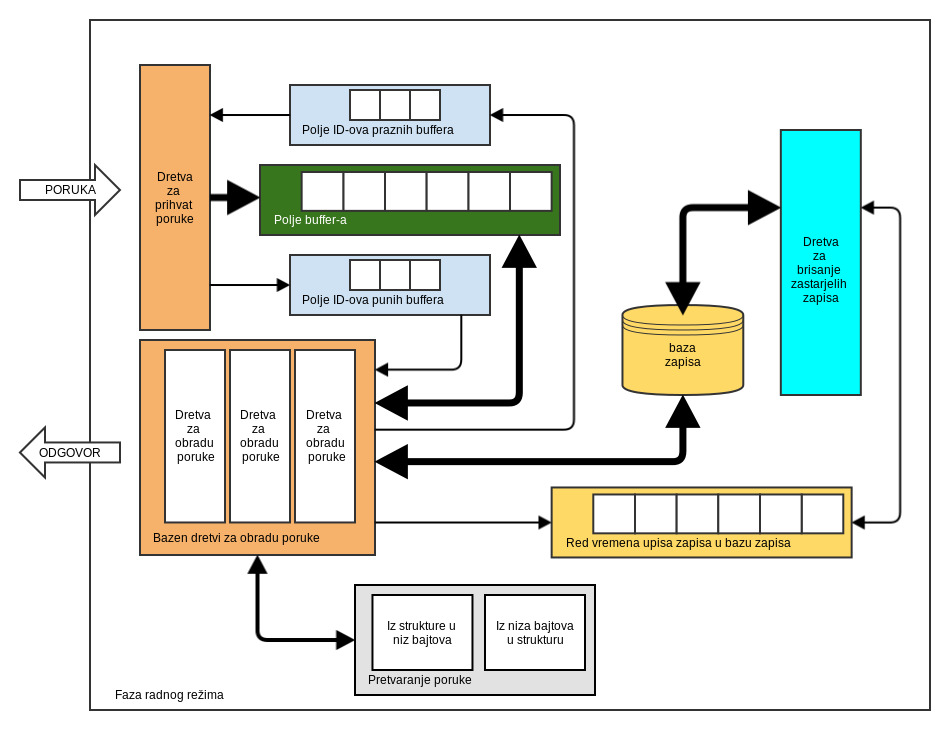


Slika : Faza inicijalizacije

Nakon završetka faze inicijalizacije slijedi faza radnog režima. U ovoj su fazi pokrenute dretve:

1. Dretva za prihvat poruke
2. Bazen dretvi za obradu poruke
3. Dretva za brisanje zastarjelih zapisa iz baze

Vizualni pogled na sam rad servera prikazan je slikom koja slijedi. Na njoj su vidljive sve navedene dretve, te također kako su one međusobno povezane i koji se dodatni spremnici koriste za ostvarivanje njihovih veza.



Slika : Faza radnog režima

Iz slike je moguće zaključiti kako server radi.

dretva za prihvat poruke čeka na poruku i istovremeno je vezana za jedan od slobodnih buffer-a unutar polja buffera. ID slobodnog buffera preuzima iz polja (reda) slobodnih buffer-a, a nakon što je poruka primljena (nalazi se unutar korištenog buffer-a), ID se buffera sprema u polje ID-ova punih buffera. Ovo se konstantno ponavlja unutar dretve za prihvat poruke.

Dretve za obradu poruke čekaju dok se u polju (redu) ID-ova punih buffer-a ne pojavi ID koji označava da je neki od buffer-a pun. Na kraju obrade ID se vraća u polje sa ID-ovima praznih buffer-a. Iz slike je vidljivo da ove dretve koriste pomoćne funkcije koje su prikazane uz donji rub slike (funkcije koje služe za pretvaranje poruke).

Dretve za obradu poruke i dretva za brisanje zapisa iz baze međusobno su povezane preko baze i reda vremena upisa zapisa u bazu. Sve dretve ovim resursima pristupaju monitorski što znači da samo jedna dretva u određenom trenutku vremena ima pristup bazi. Najveći problem kod ove veze je taj što dretva za brisanje najčešće briše više zapisa i ovdje postoji veći problem kod brzine, odnosno performansi samog servera. Dretva za brisanje zastarjelih zapisa spava određeno vrijeme (i u tom vremenu dretve iz bazena dretvi mogu normalno pristupati bazi), a zatim se „budi“ i pristupa bazi blokirajući sve dretve iz bazena da pristupe istoj.

Ovaj problem performansi je donekle riješen uvođenjem prikazanog reda vremena upisa zapisa u bazi zapisa kojim se postiže da dretva za provjeru zastarjelosti zapisa ne mora provjeravit sve zapise nego samo one koji su prvi „stigli“ u spomenuti red. Time se uvelike smanjuje broj provjera i procesorskih operacija.

* 1. Priprema servera

Na početku se priprema socket, u slučaju greške – program se prekida. Slijedi priprema varijabli za otvaranje prihvatnog mjesta za poruke.

/\* podaci servera \*/

    sock\_id = socket(AF\_INET, SOCK\_DGRAM, 0);

    if(sock\_id<0) {

#ifdef ERROR\_LOG

        cout << "Greska: nedostupan socket." << endl;

#endif

        exit(-1);

    }

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = htons(SERV\_PORT);

    server\_addr.sin\_addr.s\_addr = inet\_addr(SERV\_IP\_ADDRESS.c\_str() );

Zatim slijedi poziv funkcije **bind()**:

if(bind(sock\_id, (sockaddr\*)&server\_addr, sizeof(server\_addr) ) ) {

* 1. Pokretanje dretvi

Dretve se pokreću nakon što je pripremljen socket. Pokreće se dretva za prihvat poruka, dretva za brisanje zastarjelih zapisa iz baze, te N dretvi za obradu poruke koje čine bazen dretvi. Konstanta MAX\_N\_THREAD je ustvari parametar koji se učitava iz konfiguracijske datoteke i predstavlja ukupan broj dretvi za obradu poruke (to je ustvari bazen dretvi o kojem je riječ u ovom projektu).

/\* pokretanje svih dretvi \*/

for(int i=0; i<MAX\_N\_THREAD; i++)

bazen\_dretvi.push\_back(std::thread(obradi\_zahtjev, i) );

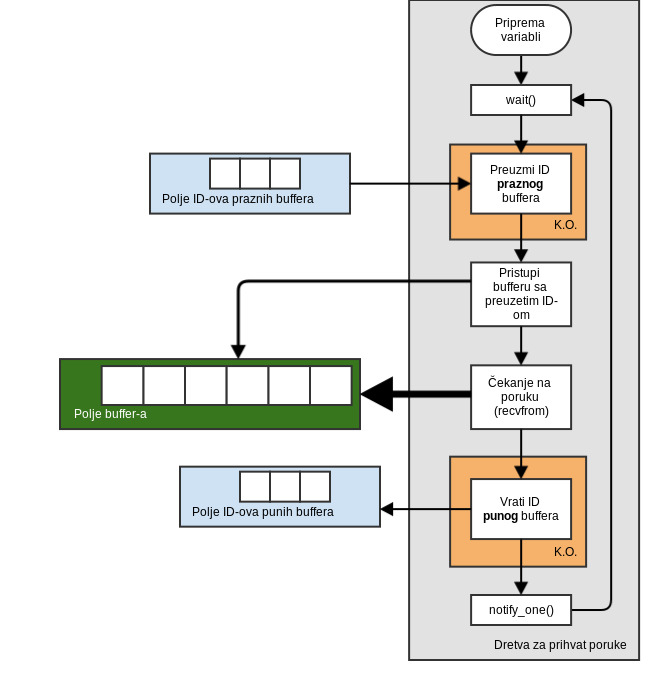
/\* dretva za prihvat poruke \*/

prihvat = std::thread(prihvati\_poruku);

* 1. Prihvaćanje poruke

Dretva za prihvati poruku se sastoji od nekoliko dijelova. Prvo se pokušava dohvatiti ID slobodnog buffera. Nakon toga, dretva čeka na poruku (recvfrom). Kad je poruka prihvaćena – ona je spremljena u buffer sa ID-om koji je prvotno dobavljen iz reda ID-ova praznih buffera. Nakon toga slijedi postavljanje ID-a korištenog buffera u red ID-ova punih buffera. Kad je ID postavljen u red ID-ova punih buffera, dretva obavještava dretve u bazenu dretvi da mogu pokušati preuzeti ID buffera i krenuti u obradu prihvaćene poruke.

Gore navedeni princip rada dretve za prihvaćanje poruke, vizualno je moguće prikazati kako je navedeno na sljedećoj u nastavku.



Slika : Dretva za prihvat poruke

* + 1. Kritični odsječak 1 – dobavljanje ID-a praznog buffer-a

Programski kod korišten za dobavljanje ID-a slobodnog buffera je ustvari prvi kritični odsječak koji dretva radi – pristup redu praznih buffera mora biti u kritičnom odsječku zbog toga što redu pristupaju i sve dretve iz bazena dretvi.

{

/\* zakljucaj pristup redu praznih \*/

std::unique\_lock<std::mutex> zakljucano(monitor);

while(red\_praznih.empty() )

dostupni\_prazni\_buffer.wait(zakljucano);

/\* preuzmi id buffera iz reda praznih \*/

id\_buffera = red\_praznih.front();

red\_praznih.pop();

/\* otkljucaj pristup redu punih \*/

}

* + 1. Kritični odsječak 2 – spremanje ID-a punog buffer-a

Nakon što je poruka prihvaćena u buffer sa prvotno nabavljenim ID brojem buffer-a, slijedi drugi kritični odsječak, odnosno pristup redu ID-ova punih buffera. Programski kod korišten za ovaj dio je sljedeći:

/\* zakljucaj pristup redu praznih \*/

monitor.lock();

/\* dodaj koristeni id buffera u red punih \*/

red\_punih.push(id\_buffera);

/\* obavijesti jednu dretvu za obradu poruke \*/

dostupna\_nova\_poruka.notify\_one();

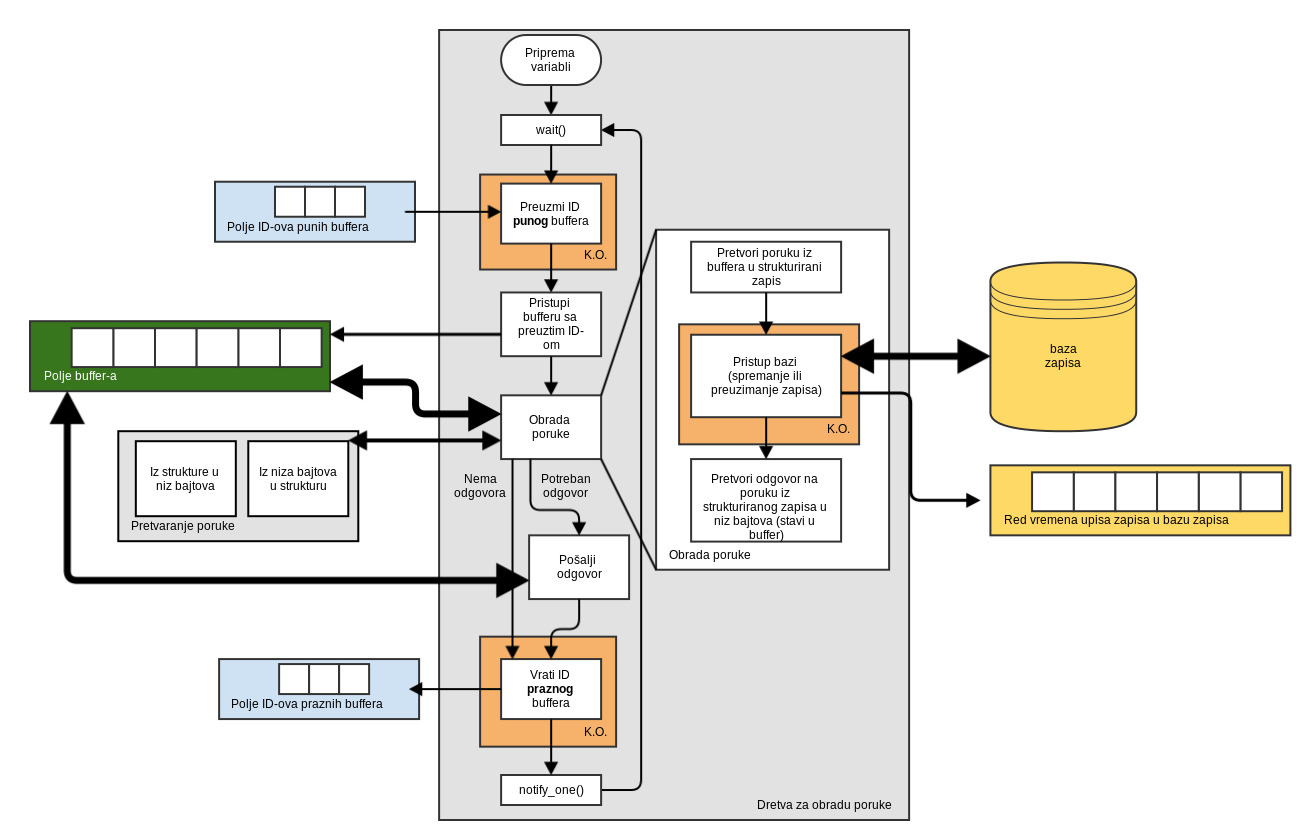
/\* otkljucaj pristup redu praznih \*/

monitor.unlock();

Nakon što je u red ID-ova punih buffer-a postavljen ID korištenog buffera, sa funkcijom **notify\_one()** obavještavamo jednu od dretvi unutar bazena dretvi, te ona kreće u preuzimanje ID-a buffera, a kasnije i sa obradom poruke.

* 1. Obrada poruke – dretva za obradu poruke

Bazen dretvi sadrži N dretvi za obradu poruke. Vizualni izgled kako bi se shvatila važnost ove dretve prikazan je na slici ispod.



Slika : Dretva za obradu poruke

Princip rada ovih dretvi sličan je radu dretve za prihvat poruke. Također, ova dretva radi i slanje poruke odgovora, ako je potreban odgovor na primljenu poruku. Dretva se također sastoji od dva kritična odsječka za pristup redovima ID-ova buffera.

Pristup je u ovom slučaju suprotan dretvi za prihvat poruka. Prvo se čeka da bude dostupan ID punog buffera (kako bi se isti mogao čitati). To je vidljivo i u sljedećem isječku programskog koda:

{

            /\* zakljucaj pristup redu punih \*/

            std::unique\_lock<std::mutex> zakljucano(monitor);

            while(red\_punih.empty() )

                dostupna\_nova\_poruka.wait(zakljucano);

            /\* preuzmi id buffera iz reda punih \*/

            id\_buffera = red\_punih.front();

            red\_punih.pop();

            /\* otkljucaj pristup redu punih \*/

}

Na kraju, kada je poruka obrađena, te ako je potreban odgovor na poruku, onda i isti poslan, dretva vraća ID buffera u red ID-ova praznih buffera (kako bi dretva za prihvat poruke mogla opet koristiti isti buffer za prihvat nove poruke). Drugi je kritični odsječak vidljiv u programskom kodu ispod:

/\* zakljucaj pristup redu praznih \*/

monitor.lock();

/\* dodaj koristeni id buffera u red praznih \*/

red\_praznih.push(id\_buffera);

/\* otkljucaj pristup redu praznih \*/

monitor.unlock();

/\* obavijesti da je jedan buffer prazan \*/

dostupni\_prazni\_buffer.notify\_one();

* + 1. Slanje odgovora na poruku

Dretva koja radi slanje odgovora na poruku je ista ona koja radi obradu poruke. Prije nego se ulazi u drugi kritični odsječak (vraćanje ID-a buffera u red ID-ova slobodnih buffera) dretva zadnji put koristi buffer iz kojeg je preuzela poruku kako bi spremila odgovor na poruku i isti poslala. Koristi funkciju sendto().

1. Konfiguracijske datoteke

Dvije se datoteke koriste u programskom rješenje. Prva datoteka je konfiguracijska datoteka za parametre servera. Druga datoteka je datoteka sa popisom *relay* poslužitelja. Obje datoteke su zapisane u standardnom načinu zapisa kod tekstualnih datoteka (ASCII kodovi znakova).

* 1. Struktura konfiguracijske datoteke servera

Konfiguracijske datoteka se sastoji od:

|  |
| --- |
| 127.0.20.12 1234 \*ipadresai port  1000 \*max velicinabuffera  10 \*brojdretvi |

127.0.20.12 predstavlja ip-adresu, 1234 port predstavlja broj porta, sve što se nalazi iza (obično navedeno iza oznake \*) predstavlja opis kojim želimo pojednostaviti što se nalazi ispred.

1000 predstavlja maksimalnu veličinu buffer-a (broj bajtova), na kraju slijedi kratki opis druge linije.

10 predstavlja broj dretvi koje će se pokrenuti unutar bazena dretvi. Opet, kao i kod drugih parametara, na kraju dolazi opis (komentar) uz podatak u liniji.

* 1. Struktura datoteke sa popisom relay-a

Ovdje se nalazi primjer datoteke s popisom relay-a. Vidljivo je da svaka linija predstavlja podatke o jednom relay-u.

|  |
| --- |
| h abcdef 23  h abc.com 35  h rel1 32  ip4 124.232.52.1 12  h defgh 32  ip4 35.32.1.53 53 |

Kao što je rečeno, svaka linija datoteke označava podatke jednog relay-a: prvi dio je vezan za tip adrese („h, ip4, ip6“), drugi dio je ip adresa ili hostname (ovisno o prvom podatku) relay-a, a treći dio predstavlja broj porta.

* 1. Učitavanje parametara iz konfiguracijske datoteke

Funkcija za učitavanje parametara iz konfiguracijske datoteke je funkcija **ucitaj\_konfig\_podatke()** koja prvo provijerava da li je datoteka dostupna, te ako nije program se odmah prekida jer nije namijenjen rad bez navedenih parametara.

Ukoliko je datoteka dostupna vrši se učitavanje podataka iz konfiguracijske datoteke. Iz konfiguracijske se datoteke dobavlja ip-adresa i port servera u varijable SERV\_IP\_ADRESS i SERV\_PORT, a zatim se dobavlja maksimalna veličina buffer-a u varijablu MAX\_BUFFER\_SIZE. Na kraju se dobavlja broj koji predstavlja broj dretvi u bazenu (dretve za obradu pristiglih poruka). Taj broj sprema se u varijablu MAX\_N\_THREAD.

* 1. Struktura popisa relay-a unutar programskog koda

Struktura zapisa informacije za pojedini relay je ustvari zapis tipa **sadresa** dan sljedećim kodom (potrebno je pamtiti samo adresu i port poslužitelja):

struct sadresa{

uint8\_t tip\_adrese;

union {

uint32\_t adr\_ipv4;

struct {

uint64\_t adr\_ipv6\_1;

uint64\_t adr\_ipv6\_2;

};

unsigned char adr\_hostname[100];

};

uint16\_t broj\_porta;

};

Svaki se poslužitelj dodaje na listu svih poslužitelja tokom čitanja iz datoteke. Svi relay-ovi nakon čitanja će se nalaziti u spremniku **popis\_relaya**.

std::vector<sadresa> popis\_relaya;

Kao što je na dijelu koda iznad vidljivo - cijeli popis relay-a se dodaje unutar spremnika relay-a koji je definiran pomoću std::vector strukture podataka. Kasnije će se isti popis provjeriti unutar funkcije za provjeru dostupnosti relay-a. Međutim, pošto nije implementirana PING-PONG akcija, smatra se da je svaki od poslužitelja sa liste ustvari dostupan i ostaje u spremniku (popisu).

* 1. Učitavanje popisa relay-a iz datoteke

Sav popis relay-a iz datoteke se čita u programu pomoću funkcije **ucitaj\_relaye()** gdje se pomoću **std::ifstream idat** otvara datoteka u kojoj su zapisani relay-i. Prije čitanja datoteke, radi se provjera da li je datoteka dostupna. Ukoliko je datoteka dostupna pokreće se petlja pomoću koje se učitavaju podaci jednog relay-a, koja prvo provjerava o kojem tipu adrese se radi („*h*“, „*ip4*“ ili „*ip6*“), te ukoliko adresa nije niti jednog valjanog tipa program javlja grešku „neispravan zapis adrese relay-a“. Zatim se redom dodaju podaci o svakom sljedećem relay-u unutar datoteke u popis relay-a.

Ako je pročitani zapis adresa poslužitelja u IPv4 formatu, onda se koristi dio koda gdje je korištena funkcija **inet\_aton** za pretvaranje adrese iz čitljivog zapisa u zapis bajtova (**in\_addr**):

} else if(tip=="ip4") {

pom.tip\_adrese = ADR\_IPv4;

if(!inet\_aton(adresa.c\_str(), (in\_addr\*)&adr) ) {

#ifdef INFO\_LOG

cout << "adresa nije ispravnog zapisa" << endl;

#endif

}

pom.adr\_ipv4 = adr.s\_addr;

}

Ako je adresa poslužitelja (relay-a) IPv6 ona se ignorira (server nije testiran sa IPv6 adresama) – ovaj dio koda treba se implementirati u eventualnim kasnijim izvedbama.

* + 1. Priprema poruke MSG\_RELAY\_LIST

Na kraju funkcije za provjeru dostupnosti relay-a radi se priprema poruke tipa MSG\_RELAY\_LIST koja sadrži popis svih dostupnih relay-a. Poruka tipa MSG\_RELAY\_LIST dostavlja se kao odgovor na poruku MSG\_REQ\_RELAY\_LIST.

1. Struktura poruke
   1. Struktura poruke

U porukama se često koristi kombinacija od tri uzastopna polja koja služe za indetifikaciju socketa. Prvo polje označava tip poruke, drugo polje je promjeljive duljine i označava adresu dok je treće polje broj UDP porta.

Adresa socketa specificira se s tri polja. Prvo polje obrađuje tip adrese te o njemu ovisi značenje i veličina drugog polja. Imamo tri moguća tipa, a to su ADR\_IPv4, ADR\_IPv6 i ADR\_HOSTNAME. Treće polje je adresa ovisno o specificiranom tipu adrese.

Sama struktura za poruke napisana je pod nazivom *sadrese*, te u njoj spremamo jedan od tri moguća tipa adrese, te broj porta. Tip adrese je *ipv4*, *ipv6* ili *hostname*. Svaka adresa sadrži i broj porta (varijabla **broj\_porta**). Veličina tipa poruke je 8 bita (1 bajt). Ako je tip adrese ipv4 tada je veličine zapisa adrese 32 bita (4 bajta), veličina zapisa adrese ipv6 zauzima 128 bita (16 bajta), a ako je adresa tipa hostname tada zauzima 800 bita (100 bajta). Broj porta zauzima uvijek 16 bita (2 bajta).

struct sadresa{

uint8\_t tip\_adrese;

union {

uint32\_t adr\_ipv4;

struct {

uint64\_t adr\_ipv6\_1;

uint64\_t adr\_ipv6\_2;

};

unsigned char adr\_hostname[100];

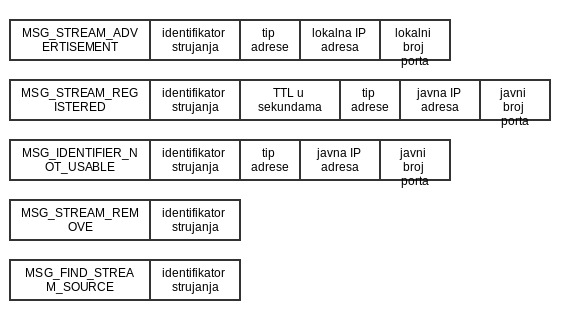
};

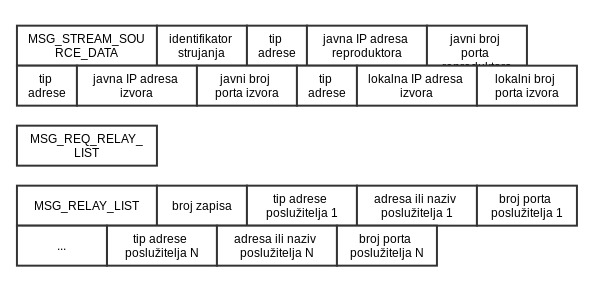
uint16\_t broj\_porta;

};

* 1. Tipovi poruka

Unutar programskog koda vidljivo je dosta različitih struktura od kojih svaka predstavlja po jedan tip poruke. Popis svih poruka sa kojima server radi vidljive su na slici ispod. Vidljivo je da većina poruka ima polje „identifikator strujanja“. U programskom kodu servera dostupne su strukture i ostalih poruka koje se koriste kroz sustav strujanja multimedije. Na sljedećoj je slici popis i struktura poruka koje server poslužitelj prepoznaje i sa kojima radi. Ostale poruke koje su vidljive u programskom kodu servera, a nisu ovdje navedene su poruke koje bi se u kasnijim dopunama programskog koda mogle upotrebljavati.





Slika : Strukture poruka

Glavna struktura koja sadržava uniju svih ostalih tipova poruka je struktura **sporuka**. Programski kod koji prikazuje oblik strukture **sporuka** je sljedeći:

struct sporuka {

uint8\_t tip\_poruke;

union {

smsg\_multimedia msg\_multimedia;

smsg\_stream\_advertisement msg\_stream\_advertisement;

smsg\_stream\_registered msg\_stream\_registered;

smsg\_identifier\_not\_usable msg\_identifier\_not\_usable;

smsg\_stream\_remove msg\_stream\_remove;

smsg\_request\_streaming msg\_request\_streaming;

smsg\_find\_stream\_source msg\_find\_stream\_source;

smsg\_stream\_source\_data msg\_stream\_source\_data;

smsg\_forward\_player\_ready msg\_forward\_player\_ready;

smsg\_player\_ready msg\_player\_ready;

smsg\_source\_ready msg\_source\_ready;

smsg\_req\_relay\_list msg\_req\_relay\_list;

smsg\_relay\_list msg\_relay\_list;

smsg\_shutting\_down msg\_shutting\_down;

smsg\_please\_forward msg\_please\_forward;

smsg\_ping msg\_ping;

smsg\_pong msg\_pong;

smsg\_pong\_reg\_req msg\_pong\_reg\_req;

smsg\_register\_forwarding msg\_register\_forwarding;

};

};

Ovdje je vidljiv popis svih mogućih poruka koje se koriste unutar projekta. Većinu navedenih poruka server ne koristi trenutno, ali su ostavljene za slučaj da iste budu potrebne u budućnosti.

* 1. Funkcije za pretvaranje poruke

Poruke se unutar programa lakše obrađuju ako su u strukturiranom obliku. Pošto poruka dolazi u obliku niza znakova (ili točnije: niza bajtova), iste je potrebno pretvoriti u strukturirani zapisa. Kad se poruka treba poslati kao odgovor onda je potrebno poruku iz strukturiranog zapisa prevoriti u zapis niza bajtova kako bi se poruka mogla poslati.

* + 1. Pretvaranje poruke u strukturirani zapis

Tip poruke određen je prvim bajtom sadržajem poruke i onda se prema njemu određuje vrsta zapisa poruke iz gore navedene strukture. Kod pretvorbe poruka u strukturirani zapis uzimamo tip poruke od 1 bajta (koji se nalazi na prvom bajtu poruke zapisane u nizu bajtova u mrežnom redoslijedu zapisa bajtova). Funkcija vraća strukturirani zapis poruke.

case MSG\_STREAM\_REMOVE: {

p.msg\_stream\_remove.identif\_strujanja =

be64toh(

poruka[m+1]\*0x100000000000000 +

poruka[m+2]\*0x1000000000000 +

poruka[m+3]\*0x10000000000 +

poruka[m+4]\*0x100000000 +

poruka[m+5]\*0x1000000 +

poruka[m+6]\*0x10000 +

poruka[m+7]\*0x100 +

poruka[m+8]);

break;

}

Gore je navede primjer jedne pretvorbe iz niza bajtova u strukturirani zapis poruke. Svaka ostala pretvorba se radi na sličan način. U funkciji postoji switch case izbor, a svaki case predstavlja jedan tip poruke – tip poruke je u **poruka[0]** bajtu koji se nalazi ispred samog switch-a.

* + 1. Pretvaranje poruke iz strukturiranog zapisa

Prvi bajt u zapisu bajtova poruke govori o tipu same poruke, te se prema njemu određuje kojim načinom će se sadržaj strukturiranog zapisa prebaciti u niz bajtova. Svaki atribut strukture u strukturiranom zapisu je određene veličine. Kod pretvorbe u niz bajtova potrebno ga je rasčlaniti na posebne bajtove (npr. ako se radi o tipu zapisa **uint32\_t** onda se radi o 32 bitnom zapisu, a on zauzima 4 bajta, takav broj se mora pretvoriti u zapis od 4 bajta mrežnog zapisa bajtova). Pretvaranje poruke iz strukturiranog zapisa odvija se na način da iz strukture **sporuka**uzimam jednu po jednu vrijednost (atribut), te je pretvaramo u mrežni zapis bajtova pomoću funkcija **htobe64()**, te funkcija **hton\*()**.

case MSG\_STREAM\_REMOVE: {

uint64\_t identif\_strujanja =

htobe64(poruka.msg\_stream\_remove.identif\_strujanja);

unsigned char id\_str[9];

id\_str[8] = '\0';

for(int i=7; i>=0; i--) {

id\_str[i] = identif\_strujanja%0x100;

identif\_strujanja /= 0x100;

}

p += MSG\_STREAM\_REMOVE;

p += (char\*)id\_str;

break;

}

Gornji primjer koda prikazuje kako je riješen problem prebacivanja poruke iz strukturiranog zapisa poruke u niz bajtova. Varijabla **p** predstavlja niz bajtova koji se dobiju nakon pretvorbe. Vidljivo je da se na sam početak tog niza postavlja tip poruke (u ovom slučaju MSG\_STREAM\_REMOVE), a zatim slijedi ostatak sadržaja poruke. Mora se paziti da se brojevi (zapisi od više bajtova) pretvore u niz bajtova koji su zapisanu u mrežnom redoslijedu bajtova – za to su korištene gore navedene pomoćne funkcije.

1. Privremena 'baza' zapisa

Baza zapisa koju program koristi napravljena je pomoću spremnika tipa std::map. Ključ svakog zapisa je jedinstveni identifikator toka strujanja koji je vidljiv i kod većine strukutura poruka.

Zbog problema performansi kod dretve brisanja zastarjelih zapisa, bazi je dodan i pomoćni red koji sadrži popis vremena unosa ili ažuriranja zapisa u bazi.

* 1. Struktura zapisa u bazi

Kao što je navedeno, baza se sastoji od dva dijela: dio realiziran pomoću spremnika std::map i služi za pohranu identifikatora toka, te pristigle poruke kod koje je isti dodan u spremnik, te drugog dijela – red vremena upisa zapisa u privremenu bazu. U daljnjem tekstu – pojam „baza“ predstavlja upravo navedeni spremnik tip std::unordered\_map.

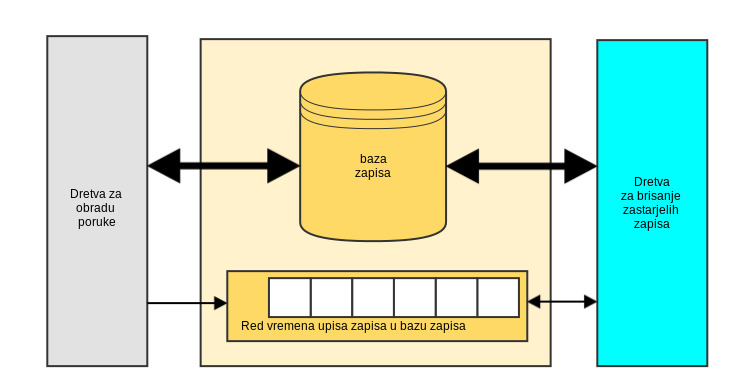
U redu se zapisuje struktura koja sadrži vrijeme zapisa, te identifikator toka zapisa. U ovaj red se zapisuje slijedno kada se radi unos u bazu ili se radi modifikacija vremena pojedinog zapisa u bazi.

Navedeni se red vremena upisa koristi kod dretve za brisanje zastarjelih zapisa iz baze. Više o brisanju zapisa navedeno je u poglavlju pristupa bazi.

Važno je napomenuti i to da je u projektu kao spremnik navedenog reda ustvari korišten std::vector zbog lakše implementacije programskog koda koji je napravljen za dretvu brisanja zapisa iz baze.

* 1. Pristup bazi

Pristup bazi je reguliran pomoću posebnog monitora kako bi se ostvarilo međusobno isključivanje dretvi. Vizualni prikaz pristupa bazi prikazan je na sljedećoj slici.



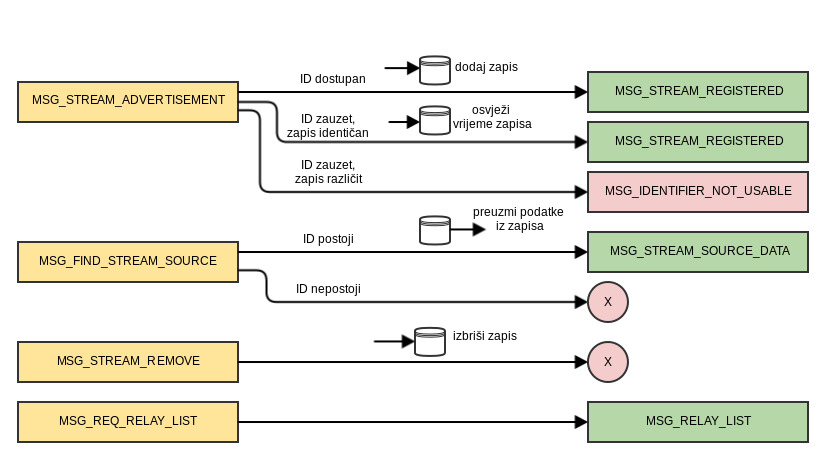
Slika : Pristup bazi zapisa

Na taj način samo jedna dretva u određenom trenutku vremena pristupa bazi, dok sve ostale dretve čekaju na svoj red pristupa.

Moguće su četiri operacije kod baze:

1. Dodavanje novog zapisa u bazu
2. Ažuriranje vremena već postojećeg zapisa
3. Preuzimanje podataka iz postojećeg zapisa
4. Brisanje postojećeg zapisa iz baze

Ovisno o kojem se tipu dolazne poruke radi, dretva za obradu poruke koja prihvati poruku pristupa bazi te radi neku od nabrojanih operacija. Kod određenih poruka nema pristupa bazi. Na sljedećoj slici je označeno kod koje poruke se pristupa bazi i o kakvoj se operaciji nad bazom radi:



Slika : Pristup bazi prema tipu poruke

* + 1. Dodavanje zapisa u bazu

Dodavanje zapisa u bazu događa se jedino u slučaju primitka poruke tipa MSG\_STREAM\_ADVERTISEMENT i to u slučaju kada identifikator toka već nije zauzet unutar baze.

Osvježavanje vremena zapisa radi se u slučaju primitka istog tipa poruke (MSG\_STREAM\_ADVERTISEMENT), ali na već „zauzeti“ identifikator toka i to upravo isti zapis koji je prije primljen (u slučaju različitog zapisa poruke, vrijeme se ne osvježuje).

* + 1. Preuzimanje podataka o zapisu iz baze

Podaci se iz baze, osim kod slučaja brisanja zapisa zbog provjere vremena upisa, preuzimaju kod poruke tipa MSG\_FIND\_STREAM\_SOURCE, te se koriste za popunjavanje poruke koja služi kao odgovor na navedenu poruku. Poruka odgovora je poruka čiji je tip poruke MSG\_STREAM\_SOURCE\_DATA.

* + 1. Brisanje zapisa iz baze

Brisanje zapisa iz baze radi dretva za brisanje zapisa. Radi se o dretvi koja se ponavlja u pozadini servera. Dretva spava određeno vrijeme (npr. jednu minutu), a zatim ulazi u monitor, te pristupa bazi.

U kritičnom odsječku, dretva redom čita zapise iz reda vremena upisa zapisa u bazu sve do zapisa kojem vrijeme nije zastarjelo (zadnja promjena vremena zapisa nije starija od navedene jedne minute za ovaj primjer). Prije nego se iz baze izbriše zapis čiji se identifikator toka nalazi u zapisu iz reda vremena upisa zapisa, potrebno je provjeriti da li je vrijeme u zapisu iz reda i vrijeme u zapisu iz baze jednako. U slučaju da je vrijeme različito – zapis koji je dodan u red vremena upisa zapisa predstavlja 'staro' vrijeme zapisa i u tom se slučaju zapis iz baze ne briše.

Dok dretva provjeri sve zapise koji su stariji od određenog vremena, dretva otključava monitorski pristup, odnosno, dopušta dretvama za obradu poruka da nastave pristup bazi ukoliko su ga zatražile.

1. Testiranje servera
   1. Karakteristike servera

Sastoji se od tri vrste dretvi:

1. Dretva za prihvat poruke
2. Dretva za obradu poruke (N dretvi unutar bazena dretvi)
3. Dretva za brisanje zapisa iz baze zapisa

Server se također sastoji i od pomoćnih spremnika i struktura kojima su regulirani pristupi spremnicima od strane dretvi:

1. Polje buffer-a
2. Polje (red) ID-ova praznih buffer-a
3. Polje (red) ID-ova punih buffer-a
4. Baza zapisa (std::unordered\_map)
5. Red vremena upisa zapisa u bazu zapisa
6. Funkcije za pretvaranje poruka u/iz strukturiranog zapisa

Postoji nekoliko kritičnih točaka servera.

Jedna od kritičnih točaka servera je pristup buffer-u (točnije: redovima ID-ova praznih, odnosno punih, buffer-a pošto više dretvi može istovremeno pristupiti svom buffer-u u polju buffer-a).

Druga kritična točka je pristup bazi podataka. Radi se o pristupu od strane dretve za brisanje zastarjelih zapisa, te pristupu dretve za obradu poruke iz bazena dretvi. Dretve za obradu poruke imaju relativno kratak pristup bazi (dodaj/ažuriraj/pročitaj zapis, te dodaj vrijeme dodavanja/ažuriranja u red vremena upisa zapisa), ali zato dretva za brisanje zastarjelih zapisa iz baze ima nešto duži pristup i drastično usporava rad servera ako se vrijeme za provjeru zapisa spusti na manji broj sekundi (standardno je server postavljen na 60 sekundi, odnosno svake minute jednom se provjerava zastarjelost zapisa).

Jednim dijelom je optimiziran pristup bazi kod brisanja zapisa koristeći pomoćni red vremena upisa (čime se dobiva na tome što bazu najčešće više nije potrebno cijelu provjeravati, već samo zapise koji se u redu nalaze najduže vremena, ako je to vrijeme starije od navedenih 60 sekundi).

* 1. Test klijent

Server je testiran od strane programa „klijenta“ koji je ustvari program koji ima određene poruke u bazi i šalje ih prema serveru random načinom. Praćen je rad servera tokom primanja tih poruka i nisu ustanovljene nikakve pogreške kod rada servera.

Server je isprobavan na lokalnom računalu, te nije testiran za rad izvan lokalne mreže.

* 1. Unaprijeđenje servera

Jedna od stavki koju bi trebalo dodati je log datoteka (koja bi bilježila pogreške u radu servera). Time bi se omogućilo dodatno nadziranje rada programa, te lakše otkrivanje eventualnih pogrešaka u programskom kodu (*debug*).

Optimizacija pristupa navedenim kritičnim točkama servera, te optimizacija rada sa porukama (optimiziranje poruka za pretvaranje poruke u/iz strukturiranog zapisa).

1. Interna interakcija na serveru

Pošto se radi o poslužiteljskom (serverskom) programu, nije rađeno posebno korisničko grafičko sučelje, već su osnovne naredbe za testiranje i rad sa poslužiteljem dostupne preko konzolne interakcije.

* 1. Dostupne naredbe

Dostupno je nekoliko naredbi koje služe prvenstveno ispitivanju pravilnog rada servera. Naredbe koje se mogu koristiti su:

* baza
* prekid
  + 1. Naredba 'baza'

Služi za ispis svih zapisa iz privremene baze. Time se omogućava provjera koji su identifikacijski brojevi zauzeti.

Pristup bazi je monitorski što uzrokuje blokiranje pristupa bazi, tijekom dohvaćanja zapisa iz baze, od strane dretvi za obradu poruke, te od strane dretve za brisanje zapisa iz baze.

* + 1. Naredba 'prekid'

Ova naredba služi za prekid rada programa. Prekidaju se sve dretve i zaustavlja se rad servera. Gube se svi zapisi iz pomoćne baze (std::unordered\_map) kao i ostali privremeno pohranjeni podaci.