SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6339

Oblikovanje i programska izvedba jednostavne aplikacije za strujanje zvuka putem Interneta

David Glavaš

Zagreb, lipanj 2019.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Napredak tehnologije, prvenstveno procesorske snage, te pojava širokopojasnog i brzog bežičnog pristupa mreži, promijenio je način na koji konzumiramo višemedijski sadržaj. To se odrazilo i na način konzumacije zvučnih zapisa. Fizički mediji pohrane zvučnih zapisa, prvenstveno glazbe, svake godine zauzimaju sve manji udio tržišne vrijednosti i bivaju zamijenjeni digitalnim. Međutim, udio tržišne vrijednosti "klasičnih" digitalnih medija, točnije preuzimanja i pohrane zapisa (*download*), također je u padu unazad nekoliko godina. Kao alternativa *klasičnim* medijima pohrane, pojavilo se strujanje glazbe putem Interneta. Kod mladih to je daleko najpopularniji oblik legalne konzumacije glazbe[4]. U zadnjih pet godina, udio tržišta koje zauzima strujanje gotovo se ušesterostručio[6].

Koristeći servise strujanja glazbe, krajnji korisnik više ne pohranjuje zapis trajno na svoj uređaj, već je on spremljen na udaljenom poslužitelju (ili drugom klijentu ako se radi o *peer-to-peer* (P2P) mreži) i dostupan je na zahtjev.

Budući da je nekomprimirani zvučni zapis često prevelik da bi se omogućilo neprekinuto strujanje, stvorila se potreba za uporabom što veće kompresije uz što manje gubitke koje korisnik može percipirati. Kao rješenje istaknula su se kodiranja poput Advanced Audio Coding (AAC), Ogg Vorbis, MPEG-1 Audio Layer-II i III te neke kompresije bez gubitaka kao što su FLAC (Free Lossless Audio Codec) i ALAC (Apple Lossless Audio Codec).

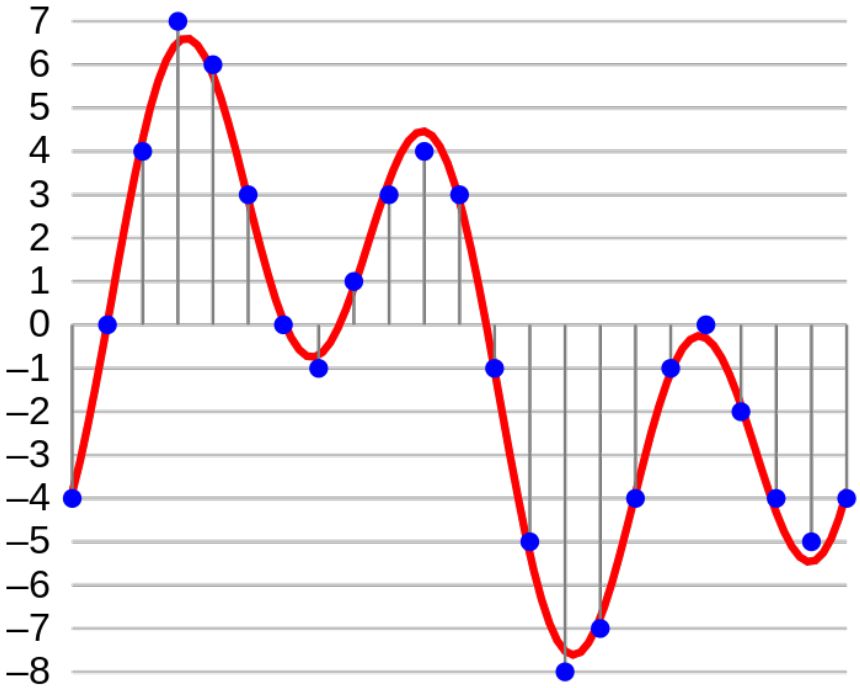
U ovom radu osmišljena je i programski izvedena umrežena višemedijska aplikacija po modelu klijent - poslužitelj koja omogućuje strujanje odabrane zvučne datoteke. Pretpostavljeni format strujanja, opisan u drugom poglavlju, je MPEG-1 Audio Layer-III. Upite obrađuje poslužitelj koji ima zapise pohranjene u istom, a dohvaćanje i reprodukcija vrše se na klijentskoj aplikaciji. Arhitektura servisa i protokol kon ceptualno su opisani u trećem poglavlju. Četvrto poglavlje bavi se konkretnom implementacijom sustava. U petom poglavlju opisan je način pokretanja poslužitelja, odnosno klijentske aplikacije, a u posljednjem poglavlju se prati degradacija iskustvene kvalitete koja se pojavljuje gubitkom paketa.

2. Format MPEG-1 Audio Layer III

2.1. Nekomprimirani digitalni zapis zvuka

Najstariji oblik digitalnog zapisa zvuka, koji je i dan danas u širokoj primjeni, jest impulsno-kodna modulacija ili *Pulse-code modulation* (PCM). Nastao je kasnih 1930-ih u Parizu i primarna namjena bila mu je uklanjanje šuma u telefoniji[8]. Zbog tadašnje kompleksne izvedbe pomoću elektronskih cijevi, u komercijalnoj uporabi PCM pojavljuje se tek koncem 50-ih godina 20. stoljeća zahvaljujući masovnoj proizvodnji tranzistora. Danas je PCM standardni oblik zvučnog zapisa u računalima, kompaktnim diskovima (CD) i digitalnoj telefoniji. U telekomunikacijama, PCM definira preporuka *ITU-T G.711 Pulse Code Modulation for voice frequencies (PCM)*[7].

PCM signal pretvara u digitalni tako da uzorkuje ulazni, analogni signal određenom frekvencijom uzorkovanja, primjerice u preporuci *ITU-T G.711* ona iznosi 8 kHz, te svaki taj uzorak kvantizira pomoću 8 bitova. Linearni PCM znači da su udaljenosti između svake dvije kvantizirane susjedne razine jednake.



Slika 2.1: PCM uzorkovanje rezolucije 4 bita [1]

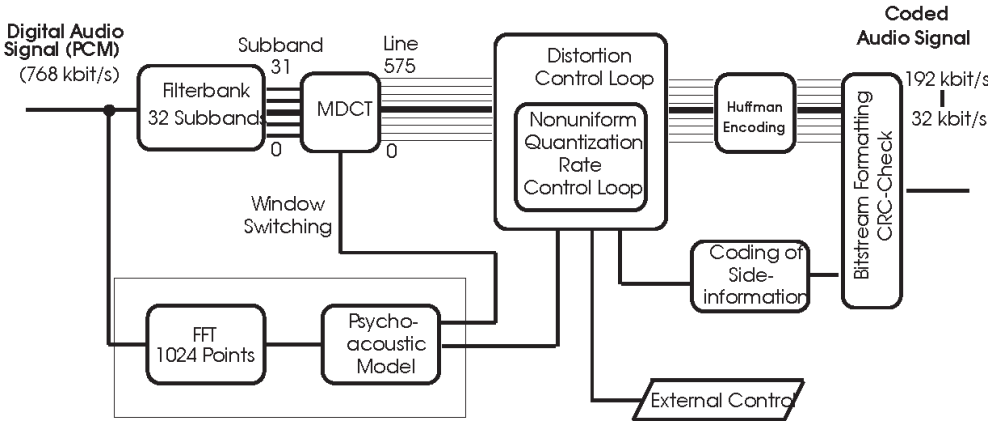
Raspon ljudskog sluha iznosi približno 20 kHz. Želimo li izbjeći gubitke pri uzorkovanju, po Shannon-Nyquistovom teoremu uzorkovanja moramo imati frekvenciju uzorkovanja barem dvostruko veću nego frekvencijski raspon signala kojega uzorkujemo. Stoga je zvučni zapis na kompaktnim diskovima, definiran standardom *IEC 60908*, uzorkovan s frekvencijom uzorkovanja 44100 Hz[1], rezolucijom od 16 bita. Iako standardna frekvencija uzorkovanja u industriji iznosi 48 kHz[9], zbog popularnosti CD-a kao nosača zvuka, 44.1 kHz je i dalje vodeća frekvencija uzorkovanja u svijetu digitalnog zapisa glazbe. Uzevši u obzir da je glazba snimana u dva kanala, zvučni zapis na CD-u brzine je 1411.2 kbps što bi moglo biti preveliko opterećenje mreži za konzistentan i kvalitetan prijenos putem mreže.

Kada su se 90-ih godina 20. stoljeća pojavile zvučne kartice i snažni procesori, glavno ograničenje bio je prostor za pohranu podataka te brzina pristupa mreži. S toga se pojavila potreba za kompresijom zvučnih zapisa sa malo gubitaka koje konzument medija može percipirati. MPEG-1 Audio Layer III (MP3) koji, ovisno o opcijama kodera, može smanjiti izvornu datoteku i do 10 puta, zadržavajući zadovoljavajuću kvalitetu zapisa.

[1] Sony je definirao taj standard. Razlog tome je prilagođenost i NTSC i PAL video kodiranjima.

2.2. Građa kodera MPEG-1 Audio Layer III

MPEG-1 Audio Layer III, ili kraće MP3, percepcijski je koder definiran standardom *ISO/IEC 11172-3:1993*. On ulazni signal prvo dijeli na 32 potpojasa, pa zatim svaki potpojas na po još 18 - ukupno 576. Potom se svaki potpojas kvantizira što prati vanjska kontrola i proces se ponavlja dok se ne dobije zadovoljavajući rezultat. To ovisi o psihoakustičnom modelu koji pomoću filtriranih potpojaseva ili kombinacijom izračuna energetskih vrijednosti maski računa izlaz. Izlaz predstavlja vrijednosti praga maske ili dozvoljenog šuma za svaki potpojas. Ako je šum ispod dozvoljenog praga, razlika između kompresiranog i originalnog signala trebala bi biti nečujna[2].



Slika 2.2: Blok dijagram MP3 kodera [2]

Implementacija psihoakustičnog modela nije strogo definirana u standardu, već ovisi o pojedinačnoj implementaciji kodera. Posljedica toga je da će različiti koderi dati različite rezultate za iste opcije. LAME (*LAME Ain't an MP3 Encoder*) je trenutno jedan od najpopularnijih i najzastupljenijih MP3 kodera te je dio FFmpeg paketa. FFmpeg, a samim time i LAME, se koriste u ovom radu za kodiranje i dekodiranje MP3 datoteka.

2.3. Interna struktura MP3 datoteke

MP3 datoteka građena je od MP3 okvira koji se sastoje od 32-bitnog zaglavlja i podataka. U MPEG verziji 1, koja je pretpostavljena verzija ovog rada, svaki okvir nezavisan je od drugih i moguće ga je reproducirati zasebno. To svojstvo će biti iskorišteno za prijenos podataka i *buffer* u samoj implementaciji, koja je detaljno objašnjena u idućem poglavlju [3].

Kodiranje MP3 datoteke može se vršiti na dva načina: *constant bit-rate* (CBR) i *variable bit-rate* (VBR). Koristimo li CBR, izlazna MP3 datoteka ima fiksne veličine okvira i njena brzina je samim time konstantna. VBR pak koristi napredniji model te veličina okvira ovisi o samom sadržaju - ako je sadržaj okvira tišina ili ton koji ima samo jednu frekvenciju, veličina okvira bit će znatno manja nego da su prisutni kompleksniji tonovi. To svojstvo može bitno smanjiti datoteku ne utječući na kvalitetu, no procesorski je zahtjevnije za kodiranje i dekodiranje.

Problem koji se pojavljuje postaje očigledan kada koristimo VBR. Kako možemo znati kada okvir završava, odnosno sljedeći okvir počinje ako unaprijed ne znamo veličine okvira? Svojstvo koje nam pomaže u rješavanju tog problema nalazi se u zaglavlju MP3 datoteke. Prvih 11 bitova zaglavlja, a i 12. bit ako koristimo MPEG verzije 1 i 2, uvijek postavljeni na 1. Ta sekvenca bitova, heksadekadski prikazano *0xFFF* pojavljuje se isključivo u zaglavlju - nije moguće da koder generira izlaz *0xFFF* u podatkovnom djelu okvira.

2.3.1. ID3 oznake

Sve verzije MP3 datoteka imaju podršku za ID3 oznake. ID3 oznaka je blok koji enkapsulira MP3 datoteku, sadržavajući podatke o samom zvučnom zapisu kao što su autor, naslov djela, album s kojeg djelo dolazi i sl. Koriste se dvije verzije ID3 oznaka - ID3v1 i ID3v2. ID3v1 je jednostavnija i već manje korištena oznaka - velika je 128 okteta i sadrži naslov, autora, album, godinu, komentare i žanr djela spremljenog u MP3 datoteci koju enkapsulira. Tih 128 okteta nalaze se uvijek na kraju datoteke, a 3 okteta dugo zaglavlje ima vrijednost *TAG* u ASCII kodu.

ID3v2 oznaka značajno je kompleksnija od verzije 1 koju mijenja. Duljina same strukture u datoteci promjenjiva je ovisno o podacima koje sadrži. Za razliku od ID3v1 oznake, čija prva 3 okteta zaglavlja u ASCII kodu imaju vrijednost *TAG*, prva 3 okteta ID3v2 oznake u ASCII kodu imaju vrijednost *ID3*[5].

Za potrebe ovog rada ID3 oznake se ignoriraju.

2.4. Izbor formata

Iako nije prvi izbor u industriji, MP3 je format dosta pogodan za strujanje zvučnih zapisa putem Interneta što zbog svoje strukture, što zbog svojeg omjera kompresije i perceptivne kvalitete. Također, zbog svoje rasprostranjenosti, programska potpora za reprodukciju MP3 datoteka ugrađena je u skoro sve veće operacijske sustave što ga

čini portabilnim rješenjem. Iako, trema imati na umu da MP3 format ima i svoje mane. Zbog načina rada zvučnih kartica, uvijek će biti potrebno dekodiranje što može uzrokovati sinkronizacijski problem te problem glatkog reproduciranja uzastopnih zapisa. Isto tako, MP3 ima svog nasljednika koji je lakši i brži - Advanced Audio Coding, koji je dio MPEG-2 i MPEG-4 specifikacija.

3. Strujanje zvuka putem Interneta

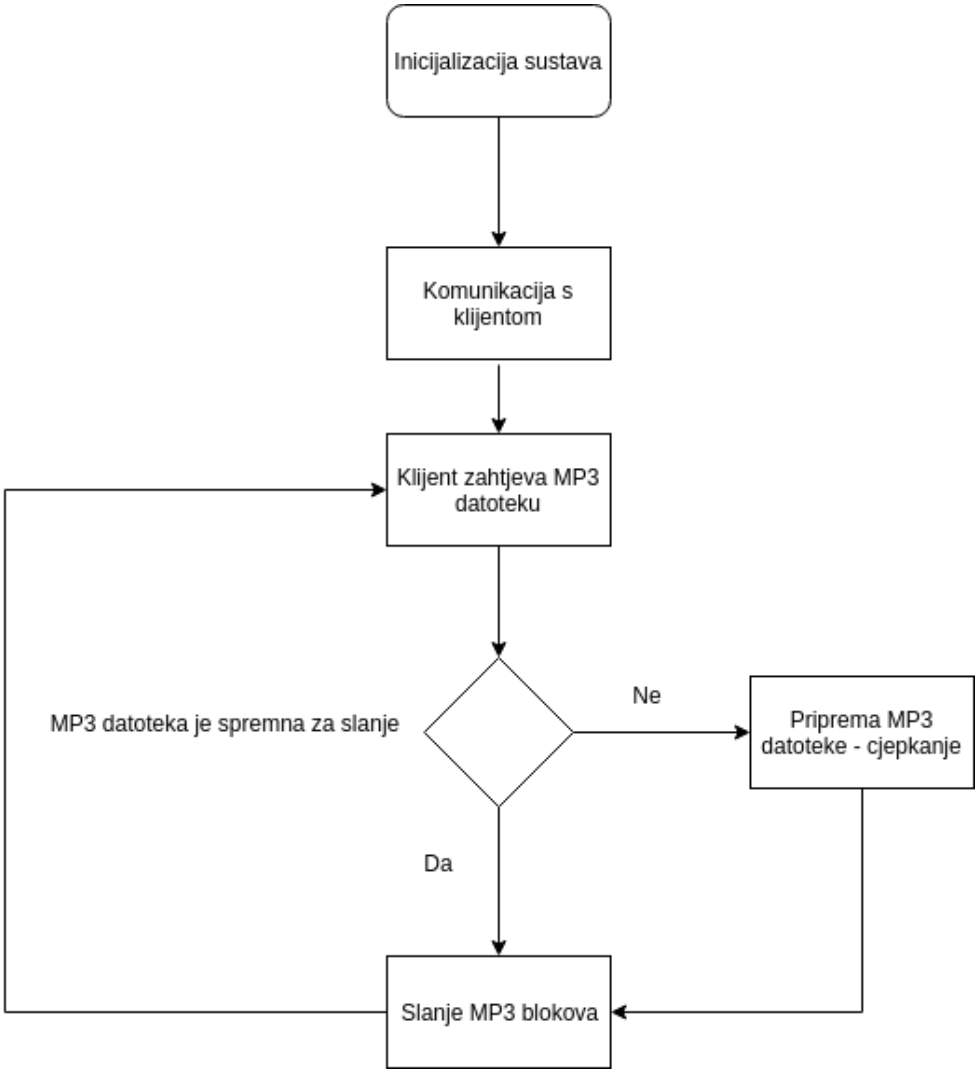
3.1. Arhitektura sustava

Sustav se može podijeliti na klijentski i poslužiteljski dio. Oni međusobno komuniciraju preko TCP veze protokolom osmišljenim za ovaj konkretni problem. Vlastiti protokol osmišljen je radi stjecanja programerskog iskustva te jednostavnosti i fleksibilnosti, dok bi u stvarnoj primjeni bilo prikladnije koristiti odgovarajući standardni protokol, primjerice, Real-time Transport Protocol (RTP).

Klijent spojen na poslužitelja šalje zahtjev koji sadrži ime zvučnog zapisa kojeg hoće strujati. Poslužitelj taj zahtjev obradi, nađe zapis, podijeli ga na manje blokove te potom šalje blok po blok klijentu. Sa strane klijenta se čeka da se napuni *buffer* te potom kreće reprodukcija dok se ostali blokovi dohvaćaju.

3.2. Poslužiteljski dio sustava

Zadatak poslužitelja je prihvatiti zahtjev za vezu klijenta, poslati mu popis dostupnih zapisa, obraditi zahtjev za potražnjom zapisa te isti taj zapis pripremiti i strujati klijentu. Za potrebe *debugginga*, poslužitelj zapisuje kritične radnje u *.log* datoteku.



Slika 3.1: Dijagram toka rada poslužitelja

3.2.1. Inicijalizacija sustava

Prvim pokretanjem poslužitelja obavlja se inicijalizacija. Prvu stvar koju poslužitelj čini je postavljanje četiri interne varijable: *PORT* - vrata koja će koristiti, *FOLDER* - direktorij u kojem je spremljen sadržaj koji se struja, *MAXCONNECTIONS* - maksimalni dozvoljeni broj veza, *TEMPSIZE* - maksimalna dozvoljena veličina priručnog spremnika. Ideja je da te četiri varijable budu spremljene u datoteku imena *server.ini*.

```
port=40004
musicFolder=music
maxconnections=100
tempsize=1073741824
```

Izvorni kod 1: Izgled server.ini datoteke

Ukoliko datoteka ne postoji ili je format datoteke pogrešan, poslužitelj generira datoteku s pretpostavljenim vrijednostima.

Potom poslužitelj traži sve dostupne MP3 datoteke iz poddirektorija definiranog varijablom *FOLDER* te slaže listu imena dostupnih MP3 datoteka u *.txt* datoteku. Pretpostavka je da se sadržaj tog poddirektorija neće mijenjati nakon uspješne inicijalizacije te se lista generira samo jednom. Ime datoteke je oblika *YYYY-MM-DD.txt* te se prilikom ponovnog pokretanja i inicijalizacije poslužitelja stara datoteka prepiše.

Nakon postavljanja internih sustavskih varijabli i generiranja liste dostupnih zapisa, poslužitelj definira i otvara *socket* na vratima definirana varijablom *PORT*. Nakon otvaranja *socketa* inicijalizacija završava te poslužitelj čeka zahtjev za povezivanjem.

3.2.2. Priprema MP3 datoteka

Poslužitelj barata MP3 datotekama preko modula napisanog za potrebu ovog rada imena *mp3loader*. *mp3loader* ima dvije glavne funkcije - podjelu MP3 datoteke na manje blokove te spremanje tih blokova kao nove MP3 datoteke u priručni spremnik.

Kada poslužitelj pozove funkciju za podjelu datoteke, prvo se provjerava nije li već datoteka podijeljena (te jesu li ti dijelovi valjani) i traže se datoteke u priručnom spremniku. Priručni spremnik nalazi se u direktoriju gdje se nalazi i glavni program poslužitelja, *server.py*, a direktorij se zove *_temp*. Unutra se mogu nalaziti direktoriji koji nose ime tražene MP3 datoteke, unutar kojih se nalaze blokovi zapisani kao male MP3 datoteke. Interna varijabla sustava *TEMPSIZE* definira veličinu priručnog spremnika u oktetima te ukoliko se popuni, njegov sadržaj se kompletno briše da se omogući njegovo ponovno korištenje.

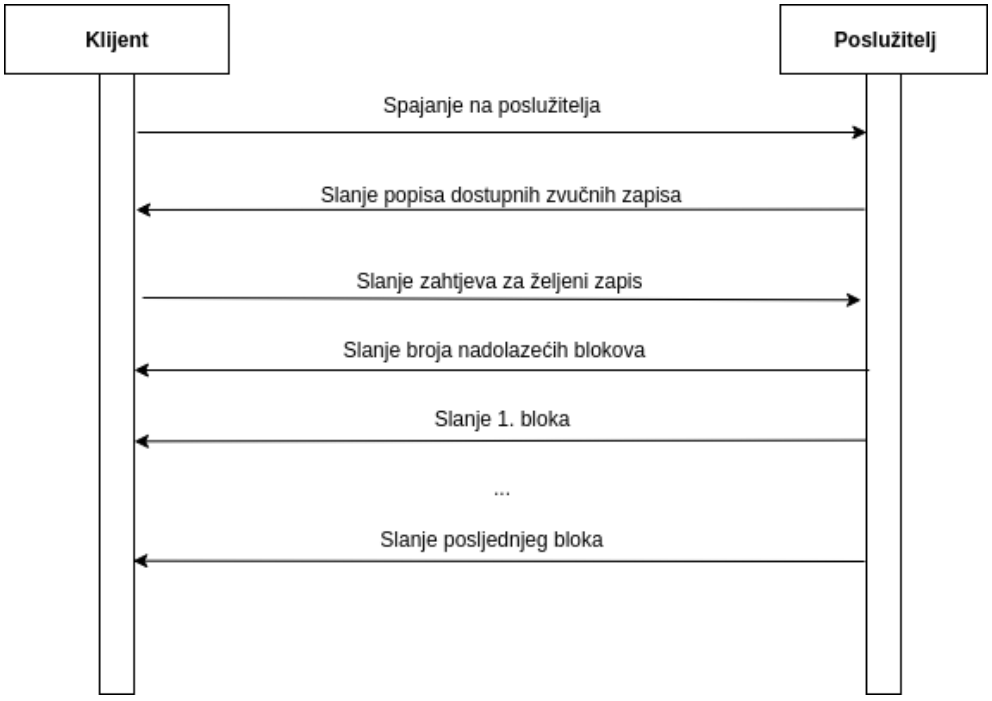
Ukoliko je pri pozivu funkcije za cjepkanje MP3 datoteke međuspremnik prazan ili se tražena datoteka ne nalazi u njemu, odvija se sljedeće. MP3 datoteka se otvara kao binarna te se čita sekvencijski. Traži se svaka pojava maske *0xFFFB0000*, koja predstavlja zaglavlje, te se do pojave idućeg zaglavlja svi okteti zapisuju u binarna

varijablu. Ta varijabla predstavlja jedan MP3 okvir te se ona dodaje na listu svih okvira. Tih je okvira puno (govorimo o desecima tisuća za MP3 datoteku trajanja 3 minute) te se oni zbog toga grupiraju u male blokove. Blok je pretpostavljene veličine od oko[1] 8 kB. Kada je podjela na blokove gotova svaki blok se zapisuje u priručni spremnik kao zasebna MP3 datoteka. Za zapis trajanja od oko 3 minute očekuje se par stotina blokova. Ti blokovi su onda spremni za strujanje klijentu.

3.2.3. Rukovanje s klijentima

Kako bi poslužitelj mogao rukovati s više klijenata istovremeno, on mora biti višedretveni ili višeprocesni program.

Svaki put kada poslužitelj prihvati vezu klijenta pokrene se nova dretva. Prvo se šalje lista dostupnih zapisa, pa se čeka naziv zapisa kojeg klijent zahtjeva te se konačno šalje zapis ukoliko je dostupan. Ta procedura se ponavlja sve dok se klijent ne odspoji. Tehnički dio rukovanja opisan je u potpoglavlju *Komunikacijski protokol*.



Slika 3.2: Sekvencijski dijagram komunikacije s klijentom

[1]Cilj je da se MP3 okvir ne razdijeli na dva bloka, tako da nije moguće odrediti fiksnu veličinu bloka. Izraz minimalno bio bi prihvatljiviji, no zadnji blok je uvijek manji od 8 KB.

3.2.4. Logging

Za potrebe *debugginga* za vrijeme razvoja te kontrolu za vrijeme rada poslužitelja napisan je modul *logger*. Iako već postoje gotova, otvorena i javno dostupna rješenja, ona su često prekompleksna za potrebe ovog relativno jednostavnog sustava. Modul *logger* zapisuje akcije koje se odvijaju na poslužitelju u *.log* datoteku formata *YYYY-MM-DD.log*. Poruke su spremljene u 4 razine važnosti - *STATUS*, *WARNING*, *ERROR* i *SUCCESS*. Uz razinu važnosti piše i trenutak odvijanja akcije te opisna poruka.

```
[2019-06-06-15:45:24] - SUCCESS - Successfully created the
↪  playlist: 2019-06-06.txt
[2019-06-06-15:45:24] - SUCCESS - Server successfully
↪  started and listening.
[2019-06-06-15:45:24] - STATUS - Server address:
↪  127.0.1.1:40004
[2019-06-06-15:45:25] - STATUS - Got connection from
↪  ('127.0.0.1', 43112)
[2019-06-06-15:45:25] - STATUS - Playlist sent to
↪  ('127.0.0.1', 43112)
[2019-06-06-15:45:38] - STATUS - Success! Anri - Cat's
↪  Eye.mp3 split into 7245 packets sized 1059 bytes
[2019-06-06-15:45:38] - STATUS - 19 MB of cache remaining.
[2019-06-06-15:45:38] - SUCCESS - Created directory at:
↪  /home/david/Desktop/ZavrsniRad_temp/Anri - Cat's Eye/
[2019-06-06-15:45:38] - SUCCESS - Playlist _temp/Anri -
↪  Cat's Eye/Anri - Cat's Eye.m3u sucessfully created.
[2019-06-06-15:45:38] - SUCCESS - Playlist and mp3's cached
↪  at: _temp/Anri - Cat's Eye/
[2019-06-06-15:45:38] - STATUS - Sending Anri - Cat's Eye
↪  to ('127.0.0.1', 43112)
[2019-06-06-15:45:38] - SUCCESS - Anri - Cat's Eye
↪  successfully sent to ('127.0.0.1', 43112)
```

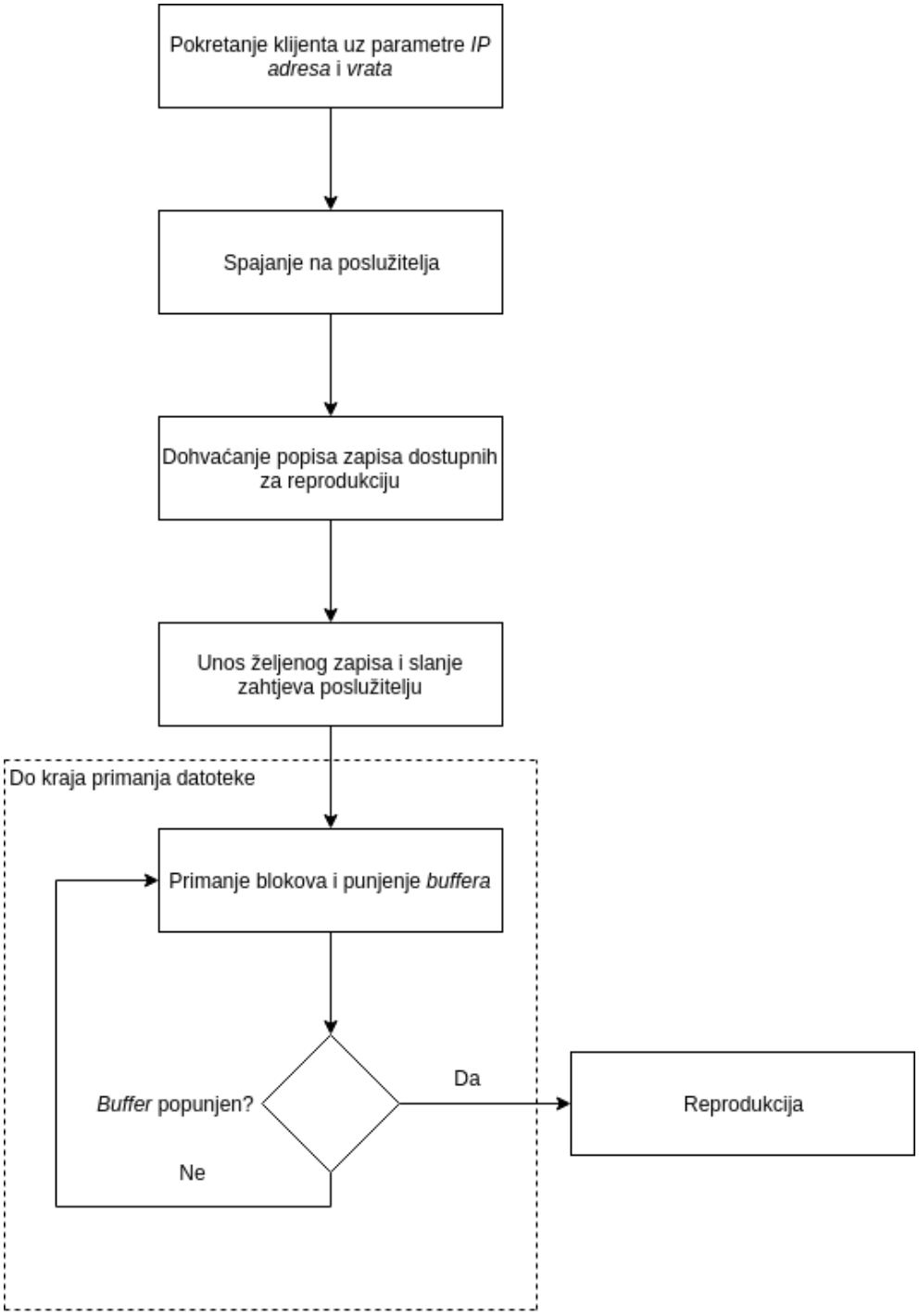
Izvorni kod 2: Primjer .log datoteke

3.3. Klijentska aplikacija

Svrha klijentske aplikacije je povezivanje s poslužiteljem, dohvaćanje te reprodukcija MP3 struje podataka. Pokreće se iz komandne linije gdje se kao argumenti navode IP adresa poslužitelja te vrata.

Pokretanjem programa ispituje se valjanost IP adrese i porta. Ako adresa nije valjana program ispisuje poruku i završava. U suprotnom nastavlja se normalni rad aplikacije.

Nakon što se aplikacija pokrene ona se odmah pokuša povezati na poslužitelja, dohvaća popis zapisa te čeka daljnju naredbu klijenta. Moguće naredbe koje klijent može dati su dvije - ili navodi ime zapisa kojeg želi dohvatiti ili naredbom *exit* odspaja se od poslužitelja i završava rad aplikacije. Odabere li korisnik dohvatiti zvučni zapis, poslužitelj struji MP3 datoteku te ju klijent reproducira.



Slika 3.3: Dijagram toka rada klijenta

3.3.1. Korisničko sučelje

Klijentska aplikacija nema grafičko korisničko sučelje, već se komunikacija sa korisnikom odvija preko konzole. Budući da je, osim unosa parametara pri pokretanju programa, jedini unos koji se od korisnika očekuje ime zapisa kojeg želi strujati, sučelje

preko konzole je adekvatno.

Kako bi se ipak uljepšalo korisničko iskustvo implementiran je *autocomplete* prilagođen za potrebe ove aplikacije.

3.3.2. Dohvaćanje i obrada struje zvučnog zapisa

Kada klijent odabere zapis koji želi, poslužitelj počinje slati blok po blok. Nakon dohvaćanja prvog bloka, koji je prvotno oblika MP3 datoteke te koji se pretvara u PCM kodiranu *wave* datoteku. Razlog tome je da se *WAVE*[2] datoteka reproducira bez dodatnog dekodiranja te za razliku od MP3 datoteke nema nekoliko milisekundi tišine na početku datoteke. To će olakšati kontinuiranu reprodukciju slijeda blokova. Taj se potom blok sprema kao *wave* datoteka u priručni spremnik klijentske aplikacije koji se nalazi u poddirektoriju *_cache*. Treba imati na umu da su *WAVE* datoteke nekomprimirane i desetak puta veće od svojih "izvornih" MP3 datoteka. Zbog toga se priručni spremnik prazni krajem svake reprodukcije zapisa.

Reprodukcija ne počinje odmah dohvaćanjem prvog bloka. Prvo se puni *buffer* veličine 50 blokova. Eksperimentom, točnije "metodom pokušaja i promašaja", se pokazalo da je približno 400 kB idealna veličina *buffera* za ovu aplikaciju. Tek kad se *buffer* popuni sa prvih 100 blokova kreće reprodukcija gdje se reproducira blok po blok slijedno. Dok se blokovi reproduciraju, novi se *buffer* puni sa sljedećih 50 blokova i tako do kraja struje. Nakon što reprodukcija završi korisnika se ponovno pita za naredbu i proces se ponavlja.

3.3.3. Reprodukcija struje zvučnog zapisa

Kod reprodukcije javljao se jedan od većih problema u programskoj implementaciji ovog sustava. Kako slijedno reproducirati tisuće okvira bez prekida između njih? Taj se problem riješio u tri koraka:

1. *Izbor formata zapisa pri reprodukciji* - Iako je MP3 bitno manji od nekomprimirane *WAVE* datoteke, taj format nije optimalan za reprodukciju. Dekodiranje datoteke se odvija prije reprodukcije u programu te se tu gubi, ovisno o računalu, nekoliko milisekundi koje se manifestiraju kao kratki prekid. Taj prekid se može skratiti uporabom nekomprimiranih i nekodiranih formata koji ne zahtijevaju dodatno dekodiranje.

2. *Provećanje veličine blokova* - Zapis je za vrijeme slanja podijeljen na stotine blokova veličine oko 8 kB koji sadrže, ovisno o kvaliteti kodiranja izvorne datoteke,

[2] Waveform Audio File Format

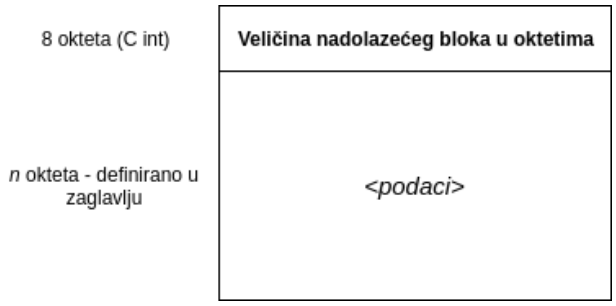
25 do 85 milisekundi sadržaja. Grupirajući blokove po 50, produljuje se vrijeme trajanja jednog bloka na petnaestak sekundi što smanjuje broj prijelaza između blokova i operacija čitanja i dohvaćanja datoteka s tvrdog diska.

3. *Izbor biblioteke za reprodukciju* - Cilj je bio izabrati *player* koji će što brže učitati datoteku sa tvrdog diska kako bi se omogućila kontinuirana reprodukcija zapisa.

Odabirom VLC-a nije riješen problem reprodukcije uzastopnih zapisa. Potrebni su dodatni mehanizmi - višedretvenost i red (biblioteka *queue*). Reprodukcija VLC-om odvija se u zasebnoj dretvi koja čita blokove iz reda. No, zbog načina rada VLC-a, kad se pozove sama metoda za reprodukciju stvara se nova dretva i samim time dodatak višedretvenosti i reda čekanja gubi smisla jer se dretva izvrši više puta u jednoj sekundi. Taj problem riješio se čekanjem. Dretva čeka duljinu trajanja zapisa kojeg VLC reproducira minus nekoliko desetaka milisekundi. Taj kraći period koji se oduzima duljina približna je duljina trajanja vremena dohvaćanja datoteke i početka njene reprodukcije u VLC-u od poziva funkcije. Time se osigurala kontinuirana reprodukcija više blokova.

3.4. Komunikacijski protokol

Komunikacijski protokol dizajniran za rješenje problema strujanja baziran je na TCP-u. Šalju se strukture podataka oblika (*veličina nadolazećeg podatka*, *podatak*). Paket *veličina nadolazećeg podatka* fiksno je velik 8 okteta, što je česta veličina *integera* u C-u te više nego dovoljno za interpretaciju broja u decimalnom obliku.



Slika 3.4: Struktura podataka u protokolu

Prvo se klijent spoji na poslužitelj, odnosno poslužitelj prihvati klijentov zahtjev za povezivanje. Potom poslužitelj šalje strukturu koja sadrži listu zapisa te čeka odgovor klijenta. Primitkom liste zapisa klijent šalje strukturu koja sadrži ime zapisa kojeg želi dohvatiti. Primivši paket od klijenta poslužitelj šalje ugnježđenu strukturu oblika (*broj*

nadolazećih okvira, (*veličina nadolazećeg podatka 1*, *podatak 1*), (*veličina nadolazećeg podatka 2*, *podatak 2*), ...) u kojem šalje pojedine blokove zvučnog zapisa.

Veza se prekida kada klijent izađe iz programa ili pošalje poruku *exit*.

4. Implementacija

4.1. Izbor programskog jezika

Budući da se radi o jednostavnoj programskoj potpori, izbor jezika implementacije nije bio ograničen - bilo koji viši programski jezik bi bio dovoljan alat za implementiranje rješenja. Iako se Java možda nameće kao očigledan izbor, zbog prijašnjih iskustava i osobne preference, programski jezik u kojem je ovaj sustav realiziran jest Python 3[1].

Python 3 je jezik visoke razine i opće namjene. Za razliku od jezika poput C-a, Python se ne prevodi već interpretira. Jednostavnost baratanja *socketima*, redovima te višedretvenosti bile su presudne za izbor Pythona kao jezika implementacije.

4.2. Poslužiteljski dio

Programski kod poslužitelja strukturiran je u četiri skripte: *listgenerator.py*, *logger.py*, *mp3loader.py* i *server.py*.

4.2.1. server.py

server.py glavna je skripta poslužitelja. Sagrađena je od dvije veće funkcije, *initialise()* i *serveClient(clientSocket, address)*, te glavnog dijela programa. U njoj se nalaze i varijable stanja sustava, *MAXCONNECTIONS*, *PORT*, *FOLDER* i *TEMPSIZE* , te konstanta[2] *INTSIZE* koja određuje veličinu zaglavlja paketa koji se šalje. Osim preostale tri skripte, *server.py* koristi module *socket*, *os*, *random*, *sys* i *threading*. Svaka važnija radnja u skripti zapisuje se u *.log* datoteku.

initialise() Ideja funkcije *initialise()* je da postavi varijable stanja sustava koje čita iz datoteke *server.ini*. Prvo provjerava egzistenciju datoteke te ako datoteka postoji iz nje

[1] https://www.python.org, inačica u kojem je programska potpora napisana - 3.6.7

[2] Iako se tehnički radi o varijabli, ideja je da se vrijednost nje tijekom izvođenja programa ne mijenja.

čita red po red. Sadržaj svakog reda počisti od praznih znakova (razmaka, tabova te znaka novog reda), rastavi na znaku jednakosti funkcijom *split()* te dobivena lista od dva člana sadrži ime varijable i njenu vrijednost. *if-elif-else* grananjem se uspoređuje textualna vrijednost imena varijable s predefiniranim vrijednostima te se adekvatno tome postavi vrijednost interne sustavske varijable. Za svaku granu, odnosno internu sustavsku varijablu, vrše se dodatne provjere da se utvrdi da će postavljena vrijednost biti valjana.

```
        ...

        chompLine = line.replace(" ", "")
        splitLine = chompLine.split("=")

        setting = splitLine[0]
        value = splitLine[1]

        ''' CASE: PORT '''
        if (setting.lower() == "port"):

            portValue = int(value)

        ...
```

Slika 4.1: Čitanje reda, njegova prilagodba te provjera vrijednosti

Ako datoteka *server.ini* ne postoji, ona se generira sa pretpostavljenim vrijednostima.

```
MAXCONNECTIONS = 100
PORT = 40000
FOLDER = "music"
TEMPSIZE = 1024 * 1024 * 1024
```

Slika 4.2: Pretpostavljene vrijednosti datoteke server.ini

Kada postavi interne varijable sustava, zadnja stvar koju funkcija čini jest provjera

postoji li direktorij priručnokg spremnika *_temp*. Ne postoji li direktorij *_temp* funkcija ga stvara.

serveClient(clientSocket, address) Funkcija *serveClient* je funkcija koju progrma poziva spajanjem klijenta u novoj dretvi. Gotovo je cijela u beskonačnoj petlji, jedino je definicija logičke varijable *isHandshake* izvan petlje. Varijabla *isHandshake* provjerava radi li se o prvoj iteraciji petlje. Radi li se o prvoj iteraciji, u konzoli i *logu* se ispisuje da je veza postavljena, čita se datoteka s listom zapisa te se sadržaj iste šalje klijentu.

```
while True:

    ''' The first "handshake". Accepts the connection
    ↪  and sends the
        client the playlist. Does it only once. '''
    if(isHandshake):

        print('Got connection from', address)
        logger.log("Got connection from " +
        ↪  str(address), "STATUS")
        ListFile = open(todaysList, "rb")

        ''' After the connection has been established,
        ↪  the server
            sends the playlist to the client'''

        # Sends the size of the playlist
        size = os.stat(todaysList)

        sizeOfPlaylist = int(size.st_size)
        connection.send(sizeOfPlaylist.to_bytes(INTSIZE,
        ↪  'big'))

        # Sends the playlist itself
        sendList = ListFile.read(sizeOfPlaylist)
        connection.send(sendList)

        ListFile.close()

        logger.log("Playlist sent to " + str(address),
        ↪  "STATUS")
        print('Playlist sent.')

        isHandshake = False
```

Slika 4.3: Prva iteracija petlje - slanje popisa dostupnih zvučnih zapisa

Nakon početnog rukovanja, funkcija čeka da klijent pošalje naredbu - ili riječ *exit* za kraj rada, ili zapis kojeg klijent zahtijeva od poslužitelja. Primi li funkcija valjano ime zapisa poziva se funkcija iz skripte *mp3loader.py* koja će zahtijevani zapis pretvoriti u blokove u memoriji. Prvo će se klijentu poslati broj blokova, a nakon toga pomoću for petlje i sami blokovi. Nakon slanja svakog bloka funkcija čeka dodatnu sinkronizacijsku potvrdu, odnosno riječ dugu 1 oktet. Ako ime zahtijevanog zapisa nije valjano, funkcija klijentu šalje riječ dugu 1 oktet da obavijesti klijenta da traženi zapis ne postoji.

Glavni dio programa - "main()" Prva stvar koju glavni dio programa radi je poziva funkciju *initialise()* koja će inicijalizirati varijable stanja sustava, funkciju modula *mp3loader mp3loader.initialise(TEMPSIZE)* te generira listu dostupnih zapisa pomoću funkcije *listgenerator.generateList(FOLDER)*. Obje funkcije koje nisu iz skripte, te sami moduli *listgenerator* i *mp3loader*, bit će detaljnije opisani u nastavku.

Nakon početnih inicijalizacija slijedi stvaranje *socketa*. Pozivaju se funkcije za stvaranje, povezivanje[3] i slušanje[4]. Potom se poziva beskonačna petlja koja čeka zahtjev za povezivanjem. Kada se klijent poveže, pokreće se nova dretva koja će komunicirati s njim, a petlja čeka daljnja povezivanja.

4.2.2. listgenerator.py

Skripta *listgenerator.py* ima dvije funkcije - stvaranje liste dostupnih datoteka za strujanje i provjera postoji li zahtjevana datoteka, tj. nalazi li se ona na samoj listi. Funkcije, u programerskom smislu riječi, koje obavljaju te zadatke su redom *generateList(directory)* i *isInPlaylist(song, playlist)*.

generateList(directory) Funkcija *generateList*, kao što joj samo ime kaže, generira listu dostupnih zapisa koji se nalaze u danom direktoriju. Prvo stvara listu svih dostupnih MP3 datoteka u danom direktoriju i njegovim poddirektorijima. Ako postoji barem jedna takva datoteka, stvara se nova tekstualna datoteka gdje će se pronađene MP3 datoteke zapisati. Ta datoteka imat će ime oblika *YYYY-MM-DD.txt*.

isInPlaylist(song, playlist) Funkcija *isInPlaylist* je u suštini jednostavna. Prima dva argumenta, prvi je naziv traženog zapisa, a drugi datoteka u kojoj ga tražimo. Funkcija

[3] eng. bind
[4] eng. listen

otvori datoteku i provjerava red po red pojavljuje li se ime traženog zapisa u njoj. Vraća logičku istinu ako ju pronađe na popisu. Treba imati na umu da ovaj pristup nije namijenjen velikim zapisima (10000+ redaka) jer je spor. Za demonstracijske potrebe je funkcija dostatna, ali za neku ozbiljniju primjenu bilo bi dobro poslužiti se naprednijim algoritmima i strukturama podataka.

4.2.3. mp3loader.py

Modul *mp3loader* dio je poslužitelja koji barata s MP3 datotekama i priručnim spremnikom. Ima tri varijable stanja sustava - *TEMPSIZE* koja predstavlja maksimalnu dozvoljenu veličinu priručnog spremnika u oktetima, *TEMPFOLDER* koja pokazuje putanju do direktorija priručnog spremnika te *BLOCKSIZE* čija vrijednost predstavlja veličinu bloka također u oktetima. Osim varijabli stanja sustava, u modulu se nalazi i pet funkcija, *initialise(size)* koja postavlja maksimalnu dozvoljenu veličinu priručnog spremnika, *emptyCache()* koja priručni spremnik prazni, *validateCache(directory, songName* koja provjerava postoji li traženi zapis u priručnom spremniku, *split(path, mp3File)* koja dijeli zadanu MP3 datoteku u manje blokove te funkcija *exportMP3(filename, blocks)* koja svaki blok sprema u zasebnu MP3 datoteku u priručni spremnik.

initialise(size) Funkcija *initialise* samo postavlja vrijednost varijable *TEMPSIZE*.

emptyCache() *emptyCache* koristi modul *shutil* kako bi izbrisala poddirektorij definiran u varijabli *TEMPFOLDER*.

validateCache(directory, songName) Funkcija *validateCache* provjerava nalazi li se zapis imena *songName* podijeljen u blokove u direktoriju *directory*. Provjeru bazira na činjenici da funkcija *exportMP3* na kraju rada generira i *.m3u* datoteku koja je zapravo popis svih blokova u MP3 formatu. Zapravo se samo traži postojanje te *.m3u* datoteke.

split(path, mp3File) Funkcija *split* cjepka MP3 datoteku zadanu argumentima u blokove okvirne veličine *BLOCKSIZE*. Prvo otvara traženu MP3 datoteku i provjerava postoji li ona već podijeljena u priručnom spremniku. Ako postoji, samo čita sve datoteke redom i sprema binarne vrijednosti u listu koju funkcija vraća. Ako ne onda se onda dijeli. Prvo se datoteka otvori u binarnom načinu rada, definira se varijabla brojača *i* (koja se koristi samo u dodatnim provjerama), lista *blocks*, u kojoj će pojedini okviri biti spremljeni te prazni, privremeni binarni string *temp*.

Unutar *while* petlje, koja se vrti do kraja datoteke, provjeravaju se sljedeća 4 okteta u datoteci. Ta se četiri okteta maskiraju maskom *0xFFFB8000* te se gleda jesu li maskirani okteti jednaki maski. Ako jesu, znamo da se radi o zaglavlju MP3 okvira te započinjemo zapisivanje okteta u privremenu varijablu *temp*. Prvo spremimo zaglavlje te prvi idući oktet. Potom kako *while* petlja nastavlja radom, zapisujemo sve oktete dok ne dođemo u situaciju gdje su iduća četiri okteta novo zaglavlje. Tada doda posljednji oktet u varijablu *temp*, tu varijablu dodamo na listu *blocks* te ju ponovno postavimo kao praznu i postupak se ponavlja. Tako dobijemo listu pojedinih okvira.

Kako su ti okviri premali da bi ih imalo smisla slati jedan po jedan, oni se grupiraju u blokove. Ti blokovi će biti spremljeni u listu *output* nakon što se pojedini okviri pročitaju i spreme u listu *blocks*. U *for* petlji se prolaze svi pojedini okviri unutar liste *blocks* i oni se grupiraju tako da veličina grupe bude veći od *BLOCKSIZE* okteta. Potom se te grupe dodaju listi *output* koju se sprema kao MP3 datoteke funkcijom *exportMp3* te ju funkcija vraća.

exportMP3(filename, blocks) *exportMP3* sprema blokove *blocks* u direktorij imena *filename* u priručnom spremniku. Prvo provjerava ima li još dovoljno mjesta u priručnom spremniku. Ako nema prvo pozove funkciju *emptyCache*. Nakon toga stvara novu datoteku oblika <*filename*>*.m3u* u kojoj će biti spremljena imena svih pojedinih blokova spremljenih kao MP3 datoteke. Nakon toga ulazi u *for* petlju kroz listu blokova *blocks* te svaki pojedini blok sprema u datoteku imena <*filename*>*_XXX.mp3*, gdje *X* predstavlja i-tu iteraciju na tri znamenke. Nakon što se svaki pojedini blok spremi u priručni spremnik, zatvara se i pisanje *.m3u* datoteke i funkcija završava rad.

4.2.4. logger.py

Modul *logger* jednostavan je modul koji zapisuje *log* poslužitelja. Sastoji se od jedne funkcije - *log(message, level)* koja zapisuje danu poruku sa *timestampom* i razinom nužnosti *level*. Postoje četiri predefinirane razine nužnosti *ERROR*, *WARNING*, *STATUS* i *SUCCESS*. Pomoću modula *datetime* se dobije *timestamp*. *logger* sprema sve poruke u istom danu u jednu datoteku imena *YYYY-MM-DD.log*

4.3. Klijentski dio

Klijentski dio sastoji se od dvije skripte - *client.py* i *autocompleter.py*. Vanjske biblioteke koje se koriste su *pydub* i *python-vlc*.

4.3.1. client.py

client.py glavna je skripta klijentske aplikacije. Sadrži četiri varijable stanja sustava - *INTSIZE* koji ima isti značaj kao i u poslužiteljskom dijelu i njihove vrijednosti moraju biti jednake, *BUFFERSIZE*, čija vrijednost predstavlja koliko blokova ide u jedan *buffer*, *DEBUG*, logička varijabla koja govori jesmo li u razvojnom načinu rada ili ne te varijabla *DELAY*. Osim varijabli stanja sustava, postoje i četiri funkcije te glavni dio programa. Te funkcije su *printDebug(string)*, *printList(givenList)*, *validate(ip, port)* i *playVLC(q)*.

printDebug(string) *printDebug* je funkcija koja ispisuje dani string ako je interna varijabla stanja sustava *DEBUG* jednak logičkoj istini (*True*).

printList(givenList) *printList* ispisuje danu listu red po red prolazeći kroz nju *for* petljom.

validate(ip, port) Funkcija *validate* provjerava jesu li zadani parametri valjane vrijednosti. Za IP adresu provjerava postoji li, a za vrata je li u dozvoljenom rasponu.

playVLC(q) *playVLC* je funkcija koja se vrši na zasebnoj dretvi i njene operacije smještene su u beskonačnoj *while* petlji. U njoj se čita iz ime zapisa iz reda *q*, kalkulira duljinu trajanja reprodukcije istog te se pokreće reprodukcija u *VLC Media Playeru* u zasebnom procesu. Dretva onda čeka duljinu trajanja zapisa umanjen za *DELAY* te obavještava redu *q* da je gotov s obradom podatka. Taj proces se ponavlja dokle god ima blokova u redu.

Glavni dio programa - "main" Glavni dio programa prvo provjerava dane argumente pri startanju aplikacije. Od korisnika se očekuje da proslijedi dva argumenta, prvi je IP adresa poslužitelja, a drugi su vrata na koja se spaja. Ti se argumenti potom provjeravaju funkcijom *validate*. Ukoliko broj argumenata nije valjan ili njihove vrijednosti nisu valjane program završava s radom obavještavajući korisnika zašto.

Ako su argumenti valjani slijedi provjera postoji li priručni spremnik klijentske aplikacije, a to je direktorij *_cache*. Ne postoji li, stvara se.

Nakon toga slijedi stvaranje *socketa* te se spaja na poslužitelja. Nakon spajanja, klijentska aplikacija iščekuje *INTSIZE* okteta u kojoj će biti poslana duljina nadolazeće liste dostupnih zapisa. Potom se prima lista koja se sprema u datoteku *playlist.txt*. U njoj su spremljena imena zapisa bez nastavka *.mp3*. Nakon toga se iz iste datoteke

čita u listu koja se sortira. Ta se lista ispiše funkcijom *printList* te se čeka unos naredbe. Naredba može biti ključna riječ *exit* s kojom se odspaja od poslužitelja i izlazi iz aplikacije ili naziv zapisa kojeg želimo strujati. Koristeći modul *readline* i skriptu *autocompleter.py* stvoren je *autocomplete* sustav za jednostavniji unos naziva.

Slanjem imena željenog zapisa poslužitelju, iščekuje se okvir duljine *INTSIZE* okteta u kojemu se nalazi broj nadolazećih blokova. Slijedi *for* petlja u kojoj se dohvaća svaki pojedini blok. Prvo se dohvati veličina nadolazećeg bloka, a potom u *while* petlji dio po dio bloka dok se nema više što za dohvatiti. *while* petlja služi kao dodatni sinkronizacijski mehanizam jer bez nje dolazilo je do prelijevanja između *socketa*. Pristigli blokovi spremaju se u *buffer*.

Kada se dohvati *BUFFERSIZE* blokova, svi blokovi unutar *buffera* spajaju se u jedno, privremeno spremaju kao MP3 datoteka koja se potom konvertira u WAV i na kraju dodaju u red koji će se proslijeđivati dretvi za reprodukciju. MP3 datoteke brišu se čim se generiraju odgovarajuće WAV datoteke, a imena WAV datoteka spremaju se u listu *fileListToRemove* u kojoj se nalazi popis datoteka koje će se izbrisati na kraju reprodukcije. Nakon što se *buffer* isprazni i stavi u red, počinje se ponovo puniti i proces se ponavlja.

```
...
if((((i+1) % BUFFERSIZE) == 0) | ((i + 1) ==
 ↪  numberOfFrames)):              # every
 ↪  BUFFERSIZE KB (give or take) of DATA

    newName = str("temp_" + str(i) + ".mp3")

    tempMP3File = open("_cache/" + newName,
     ↪  "wb")
    tempMP3File.write(tempMP3)
    tempMP3File.close()
    tempMP3 = b''

    newSound =
     ↪  pydub.AudioSegment.from_mp3("_cache/" +
     ↪  newName)
    os.remove("_cache/" + newName)

    waveName = "_cache/temp_" + str(i) +".wav"
    newSound.export(waveName, format="wav")
    fileListToRemove.append(waveName)

    #waveSound =
     ↪  pydub.AudioSegment.from_wav(waveName)
    thQueue.put([waveName,
     ↪  newSound.duration_seconds])
    ...
```

Slika 4.4: Spremanje buffera kao WAV datoteku i njegovo stavljanje u red

Nakon što postoje barem dvije skupine blokova u redu pokreće se dretva za reprodukciju. Ona poziva funkciju *playVLC* te kao parametar prosljeđuje red u kojem se blokovi nalaze.

Glavna dretva prima blokove dok se zadnji blok ne prihvati te potom čeka povratak iz dretve za reprodukciju. Kad reprodukcija završi ponovno se traži unos naredbe.

autocompleter.py *autocompleter* prilagođena je skripta originala imena *tabCompleter*[5]. Temelji se na objektu tabCompleter koji sadrži jednu metodu - *createListCompleter(self, ll)*. *createListCompleter* čita unos s tipkovnice iz *buffera* te generira listu mogućih rezultata.

[5]Feeney, Joe; https://gist.github.com/iamatypeofwalrus/5637895

5. Pokretanje sustava

Ovaj sustav testiran je isključivo na Linux operacijskim sustavima.

5.1. Preduvjeti

- **Python 3**
- **pip** za Python 3
- VLC Media Player i Python biblioteka za spajanje VLC-a i Pythona **python-vlc** za reprodukciju zvuka
- Instalirana biblioteka **pydub**
- Instaliran **FFmpeg** za dekodiranje MP3 datoteke u WAV

5.2. Instalacija na Ubuntu Linux

S obzirom na popularnost i činjenicu da je open-source te da je sustav napisan u njemu, ova dokumentacija će pokrivati potrebne korake za konfiguraciju okruženja na Ubuntu Linuxu inačice 18.04.

Prvi korak je instalacija Python verzije 3, ukoliko već nije instalirana.[1]

```
sudo apt-get install python3
```

Potom je potrebno instalirati proširenje *pip* za Python3. *pip* je alat za jednostavnu instalaciju Python biblioteka.

```
sudo apt-get install python-pip3
```

Kako bi *pydub* u potpunosti funkcionirao terba mu set kodera. Najbolji izbor za to je otvoreni *FFmpeg*. Njega također instaliramo pomoću alata *apt-get*

[1] Obavezno python3, samo "apt-get install python" dohvaća i instalira inačicu 2

```
sudo apt-get install ffmpeg
```

Instalirajući *pip* i *ffmpeg*, možemo instalirati eksterne Python biblioteke: *vlc* i *pydub*.

```
sudo pip3 install pydub
sudo pip3 install python-vlc
```

Instaliravši sve potrebne preduvjete poslužiteljska i klijentska aplikacija su upotrebljive.[2]

5.3. Pokretanje poslužitelja

Poslužiteljske datoteke, *listgenerator.py*, *logger.py*, *mp3loader.py* i *server.py*, potrebno je smjestiti u direktorij po volji. U istom direktoriju napraviti potrebno je napraviti poddirektorij imena navedenog u datoteci *server.ini*. Poslužitelj će se pokrenuti i bez datoteke *server.ini* jer će ju generirati s pretpostavljenim vrijednostima. Ukoliko *server.ini* ne postoji, potrebno je napraviti poddirektorij *music*.

```
ls
listgenerator.py  logger.py  mp3loader.py  music
 ↪  server.ini  server.py
```

U poddirektorij za zvučne zapise, u uputama dalje *music*, smjestimo već kodirane MP3 zvučne zapise. Budući da se ID3 oznake ne prenose, poželjno je adekvatno nazvati zapise da ih se može lakše pretraživati.

Nakon što su sve datoteke na mjestu i poddirektorij popunjen zapisima, poslužitelj jednostavno pokrećemo naredbom:

```
python3 server.py
```

Izvršavanjem gore navedene naredbe poslužitelj je pokrenut sve dok ne pošaljemo signal za prekid pomoću tipkovničke kratice *CTRL+C*.

Poslužitelj generira listu zapisa pod imenom *YYYY-MM-DD.txt*, log datoteku imena *YYYY-MM-DD.log* te direktorij priručnog spremnika *_temp*.

[2]Imati na umu da će poslužitelj raditi sa instaliranim samo Python3

Slika 5.1: Screenshot poslužitelja

5.4. Pokretanje klijenta

Za pokretanje klijenta potrebno je datoteke klijentske aplikacije, *client.py* i *autocompleter.py*, smjestiti unutar proizvoljnog direktorija. Nakon toga se preko komandne linije i naredbe *cd* smjestimo u taj direktorij i pozovemo naredbu:

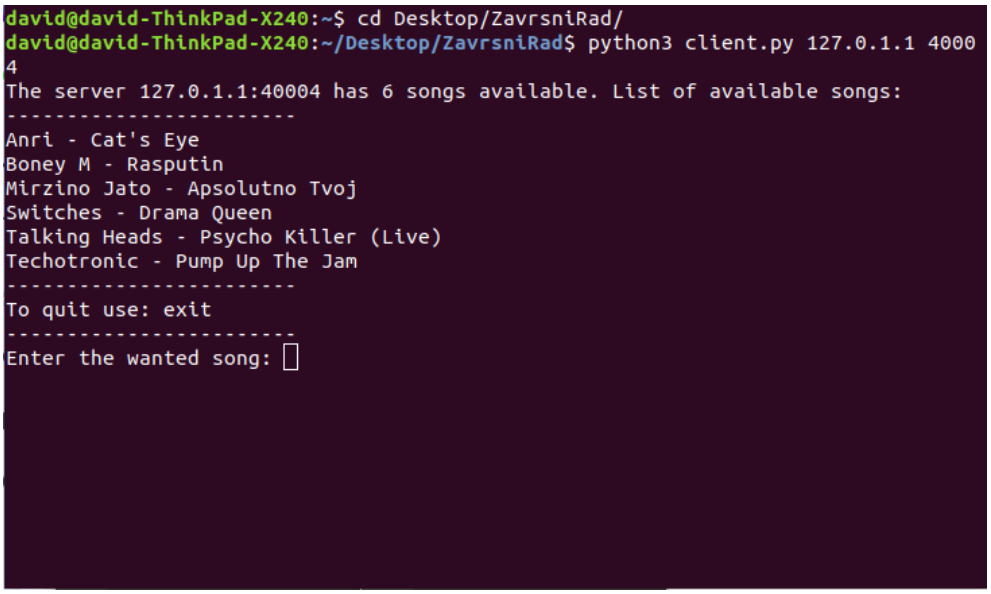
```
python3 client.py 127.0.1.1 40000
```

Gdje su *127.0.0.1* IP adresa poslužitelja, a *40000* vrata. Vidimo da je spajanje bilo uspješno kada nam se preuzme lista dostupnih zapisa te se ona ispiše u prozoru aplikacije. Potom se od korisnika traži naredba - ili da pošalje zahtjev za konkretnu pjesmu s liste dostupnih ili da pošalje naredbu *exit* da se odspoji s poslužitelja.

```
python3 client.py 127.0.1.1 40004
The server 127.0.1.1:40004 has 6 songs available. List of
 ↪  available songs:
------------------------
Anri - Cat's Eye
Boney M - Rasputin
Mirzino Jato - Apsolutno Tvoj
Switches - Drama Queen
Talking Heads - Psycho Killer (Live)
Techotronic - Pump Up The Jam
------------------------
To quit use: exit
------------------------
Enter the wanted song:
```

Izvorni kod 3: Primjer izbora zapisa u klijentskoj aplikaciji

Slanjem naziva zapisa poslužitelju provjerava se njegova egzistencija na samom poslužitelju. Potom poslužitelj ili započinje strujanje, ukoliko zapis postoji, ili pošalje poruku greške da zapis ne postoji i traži drugi. Prije reprodukcije puni se prvi *buffer* te kad dosegne prag pokreće se reprodukcija. Klijent je neaktivan dokle god traje reprodukcija zapisa, a kad reprodukcija završi ponovno očekuje od korisnika naredbu.



Slika 5.2: Screenshot klijenta

6. Degradacija kvalitete usluge povodom mrežnih smetnji

Nije neobično da mrežne smetnje utječu na komunikaciju između dva računala, pa tako i u ovom sustavu. Kako bi se minimizirala degradacija iskustvene kvalitete korišteni su sljedeći mehanizmi:

1 Uporaba protokola baziranom na **TCP-u** - TCP će uvijek tražiti retransmisiju ukoliko dođe do gubitaka paketa. Kako se u implementaciji šalje mnogo malih paketa, vrijeme retransmisije manje je nego kod većih.

2 Korištenje *buffera* - mehanizam *buffera* koji čeka određeni prag prije početka reprodukcije kupuje vrijeme za dohvaćanje sljedećeg bloka podataka.

3 Sinkronizacija pomoću odgovora od jednog okteta - Poslužitelj neće poslati novi blok dokle god mu klijent ne javi da je primio prethodni u potpunosti. To poništava šansu da blokovi neće doći ispravnim redoslijedom.

Također treba uzeti u obzir da se podrazumijeva da poslužitelj ima sve datoteke pripremljene u odgovarajućem formatu. Na "davatelju usluge" da osigura da su MP3 datoteke optimalno kodirane za svoju namjenu.

6.1. Gubitak paketa

Česti gubici paketa mogu biti znak većeg problema u mreži. Gubitak paketa se u strujanju može manifestirati kao tišina, nekakav zvučni artefakt ili u sustavima koji koriste *Dynamic Adaptive Streaming over HTTP* (DASH), zvuk niže kvalitete. U ovom sustavu može isključivo doći do tišine.

Izgubi li se paket u slanju, TCP automatski traži retransmisiju. Ovisno o izgubljenom vremenu, *buffer* će se napuniti ili neće do idućeg u redu. Ne napuni li se, doći će do "štucanja" ili potpune tišine, ovisno do trenutka kad izgubljeni paket stigne.

Nema mehanizma preskakanja paketa izgubljenih u mreži.

7. Zaključak

Iz svega navedenog može se vidjeti da, unatoč činjenici da se radi o jednostavnoj aplikaciji za strujanje zvuka na zahtjev putem Interneta, zadani problem nije trivijalno rješiv. Kako bi programska potpora ostala relativno jednostavna, za reprodukciju i kodiranje/dekodiranje datoteka korištena su gotova rješenja - *VLC Media Player* i njegove ekstenzije[1] za Python te *FFmpeg* paket kodeka za kodiranje/dekodiranje datoteka. Format zvučnog zapisa na kojem se sustav bazira jest *MPEG-1 Audio Layer III*.

Poslužiteljski dio bavi se pripremom korisnika skupljenih MP3 datoteka, rukovanja s klijentima te slanjem ranije navedenih datoteka rascjepkane u tisuće blokova. Cjepkanje datoteke riješeno je uporabom svojstava MP3 datoteka i zaglavlja pojedinih okvira unutar njih, a rukovanje s klijentima koristi višedretvenost i vlastiti protokol.

Klijentski dio dohvaća pojedine blokove datoteke od poslužitelja, dekodira svaki pojedini MP3 blok u WAV datoteke, grupira ih u *buffere* te svaki *buffer* slijedno reproducira pomoću VLC-a. Kako bi osigurali to na programskoj razini koriste se mehanizmi višedretvenosti i redova[2]. Za ugodnije korisničko iskustvo modul *autocomplete* omogućuje korisniku dovršavanje tipkanja imena zapisa kojeg zahtjeva pomoću tipke TAB.

Protokol kojem poslužitelj i klijent komuniciraju baziran je na TCP-u gdje se prenose strukture podataka oblika (*veličina nadolazećeg bloka podataka*, *blok podataka*). To osigurava da će se blok zaista poslati i doći do odredišta. Blokovi su reda veličine 8 kB.

Kako se koristi TCP bazirani protokol, računa se na to da će uvijek doći do retransmisije paketa. To potencijalno može izazvati kratkotrajne tišine i prekide u reprodukciji u slučaju većih mrežnih smetnji.

U konačnici, programsko rješenje je jednostavno, modularno te zahtjeva minimalne resurse za ispravan rad. Protokol jamči određenu razinu pouzdanosti, a uporaba *buffera* i višedretvenosti glatku reprodukciju zvuka.

[1] eng. bindings

[2] eng. queues

LITERATURA

[1] Nepoznati autor. *An example of 4-bit pulse code modulation*. URL https://commons.wikimedia.org/wiki/File:Pcm.svg.

[2] Karlheinz Brandenburg. *MP3 and AAC explained*. Audio Engineering Society, 2017. URL https://www.iis.fraunhofer.de/content/dam/iis/de/doc/ame/conference/AES-17-Conference_mp3-and-AAC-explained_AES17.pdf.

[3] International Organization for Standardization. *ISO/IEC 11172-3:1993*. 2013.

[4] Neil Howe. *How Music Streaming Won Over Millennials*. 2019. URL https://www.forbes.com/sites/neilhowe/2019/01/16/how-music-streaming-won-over-millennials/.

[5] Martin Nilsson. *ID3 tag version 2.4.0 - Main Structure*. 2000. URL http://id3.org/id3v2.4.0-structure.

[6] The Recording Industry Association of America (RIAA). *US Recorded Music Revenues by Format*. 2018. URL https://www.riaa.com/u-s-sales-database/.

[7] Igor S. Pandžić, Alen Bažant, Željko Ilić, Zdenko Vrdoljak, Mladen Kos, i Vjekoslav Sinković. *Uvod u teoriju informacije i kodiranje*. element, 2012.

[8] Alec Harley Reeves. *Electric Signaling System*. United States Patent Office, 1939. URL https://patentimages.storage.googleapis.com/eb/8e/9f/32ad53d114d2d6/US2272070.pdf.

[9] Audio Engineering Society. *AES5-2018: AES recommended practice for professional digital audio - Preferred sampling frequencies for applications employing pulse-code modulation (revision of AES5-2003)*. 2003.

Oblikovanje i programska izvedba jednostavne aplikacije za strujanje zvuka putem Interneta

Sažetak

Strujanje zvuka na zahtjev putem interneta unazad zadnjih nekoliko godina postaje dominantni oblik konzumacije komercijalnih zvučnih zapisa. Potaknut rastom te tehnologije, cilj ovog rada je bio osmisliti i programski izvesti jednostavnu platformu za strujanje zvučnih zapisa te opažati utjecaj smetnji u mreži na iskustvenu kvalitetu krajnjeg korisnika. Rad je implementiran u programskom jeziku Python, kao pretpostavljeni format zvučnog zapisa koristi MP3, klijent i poslužitelj komuniciraju protokolom baziranom na TCP-u, a reprodukcija se vrši u *VLC Media Playeru*.

Ključne riječi: strujanje zvuka na zahtjev, Python, MP3, TCP, višekorisnički servis, iskustvena kvaliteta

Design and implementation of a basic audio streaming application over the Internet

Abstract

Streaming audio on demand has become the most popular legal way of consumption of commercially available records. Inspired by the success of the given technology, the goal of this thesis was to design and implement a simple audio streaming platform and observe how network interference affect user experience. The program was written in the Python programming language, it uses MP3 as the preferred audio format, the client and server communicate via a protocol based on TCP and the playback is done through VLC Media Player.

Keywords: streaming audio on demand, Python, MP3, TCP, multi-user service, user experience