

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Dynamic resampling of the image size based on bandwidth size variation

propusă de

TiVo

Sesiunea: iulie, 2017

Coordonator științific

Conf. Dr. Adrian Iftene

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ

Dynamic resampling of image size based on bandwidth size variation

Scripcariu Andrei

Sesiunea: iulie, 2017

Coordonator științific

Conf. Dr. Adrian Iftene

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul “Dynamic resampling of image size based on bandwidth size variation” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referință precisă a sursei;
- reformularea în cuvintele proprii a textelor scrise de către alți autori deține referință precisă;
- codul sursă, imagini etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, data

Absolvent Andrei Scripcariu

(semnătură în original)

Cuprins

Introducere	7
Contribuții	8
Capitolul 1. MPEG	9
1.1 Introducere	9
1.1.1 Compresia datelor	10
1.1.2 Principiile compresiei	11
1.1.3 Tehnici ale compresiei	11
1.2 Compresia video	12
1.2.1 Ochiul uman	12
1.2.2 Redundanța temporală	13
1.2.3 Redundanța spațială	14
1.2.3.1 DCT	14
1.2.3.2 Scanarea	15
1.2.4 Conceptele transmisiei într-un cadru video	16
1.2.4.1 Imagini de tip I	17
1.2.4.2 Imagini de tip P	17
1.2.4.3 Imagini de tip B	17
1.3 Compresia audio	17
1.3.1 Psihoacustica	18
1.3.2 Depozitul de filtre	18
1.3.3 Straturile MPEG	19
1.3.3.1 Stratul 1	19
1.3.3.2 Stratul 2	20
1.3.3.3 Stratul 3	21

1.4	Fluxuri de biți	21
1.4.1	Pachete și ștampile de timp	22
1.4.2	Fluxul de transport	23
1.4.3	Multiplexarea	25
Capitolul 2.	FFMPEG	26
2.1	Aplicații	28
2.1.1	Protocol de comunicare	29
2.2	FFprobe	30
2.3	FFPlay	31
2.4	Exemple de utilizare	31
Capitolul 3.	Arhitectura și implementarea aplicației	34
3.1	Modelul client-server	34
3.2	Clasa Queue	36
3.2.1	Membrii clasei	36
3.2.2	Metodele clasei	36
3.3	Multithreading	37
3.3.1	Firul de execuție writeT	38
3.3.2	Firul de execuție readT	38
3.4	Pipe	39
3.5	Descrierea aplicației	40
3.5.1	CreateFiles	40
3.5.2	Serverul	42
3.5.3	Clientul	43
3.5.4	Utilizarea aplicației	44
Capitolul 4.	Studiu de caz și consumul aplicației	45
4.1	Atomul moov	45

4.2 Consumul de memorie	47
Concluzii	49
Bibliografie	50

INTRODUCERE

În cuprinsul lucrării ce urmează voi prezenta o aplicație care permite utilizatorului să urmărească un video în timp real, prin intermediul unei rețele, optimizând limitele pe care le impune lățimea de bandă a utilizatorului.

Am fost inspirat de numărul de utilizatori noi al serviciilor video prin internet, care se află într-o creștere continuă în raport de numărul celor care au rămas fideli televizorului. Calitatea serviciilor depinde, în cazul nostru, de lățimea de bandă de care dispune utilizatorul. Motiv pentru care am căutat să găsesc o soluție pentru a îmbunătăți calitatea enunțată folosindu-mă de aceeași cantitate de resurse.

Există în momentul de față o aplicație asemănătoare, FFMpeg, care însă nu a fost concepută luând în calcul problemele pe care le ridică insuficiența lățimii de bandă a utilizatorului mediu. Voi încerca, bazându-mă pe structura oferită de FFMpeg, să optimizez aspectele insuficient tratate, de o manieră care să nu implice intervenția utilizatorului.

Lucrarea are o structura cvadripartită, structurat pentru a permite o înțelegere progresivă, plecând de la noțiunile de bază și avansând către efectul propriu-zis și impactul asupra domeniului.

În primul capitol sunt descrise în detaliu conceptele și algoritmi care stau la baza întregii aplicații, de asemenea sunt expuse comparativ conceptele de transmisie prin televiziune și transmisie prin rețea. Scopul întregului capitol este de a familiariza cititorul cu liniile directoare ale aplicației, constituindu-se în primul pas spre a înțelege modul de funcționare al aplicației.

Capitolul doi depășește registrul tehnic introducând situațiile practice, analizând și prezentând modul efectiv de funcționare al programului pe structura căruia voi dezvolta întreaga aplicație, FFMpeg.

Cel de-al treilea capitol are un scop informativ în care este detaliată implementarea aplicației, algoritmi și tehnicile utilizate, precum și înțelegerea mai bună a funcționalității programului prin exemple de cod, imagini ajutătoare și informații suplimentare. Structura capitolului este una progresivă care trece utilizatorul prin pașii făcuți în crearea aplicației.

În ultimul capitol vor fi definite teste ale consumului memoriei a aplicației și o arie de cercetare.

Voi concluziona referatul cu cunoștințele dobândite în urma lucrării la această aplicație, îmi voi expune părerea personală asupra domeniului precum și asupra posibilelor direcții de dezvoltare ale acesteia.

CONTRIBUȚII

Proiectul îmbină ideile personale cu cele ale domnului profesor coordonator, în scopul realizării unei aplicații concise, ușor de utilizat, îmbinând tehnicile de rețelistică învățate în facultate cu cele ale limbajului de programare C/C++.

Arhitectura aplicației este în mare parte gândită de mine, o parte a acesteia inspirându-mă din aplicații deja existente, îmbunătățind anumite aspecte ale acestora precum și optimizând anumiți factori pentru o utilizare cât mai lină.

Toate tehnologiile și librăriile utilizate sunt open-source, iar detaliile tehnice, imaginile comparative, testele și statisticiile arătate sunt gândite și concepute de mine.

Capitolul 1

MPEG

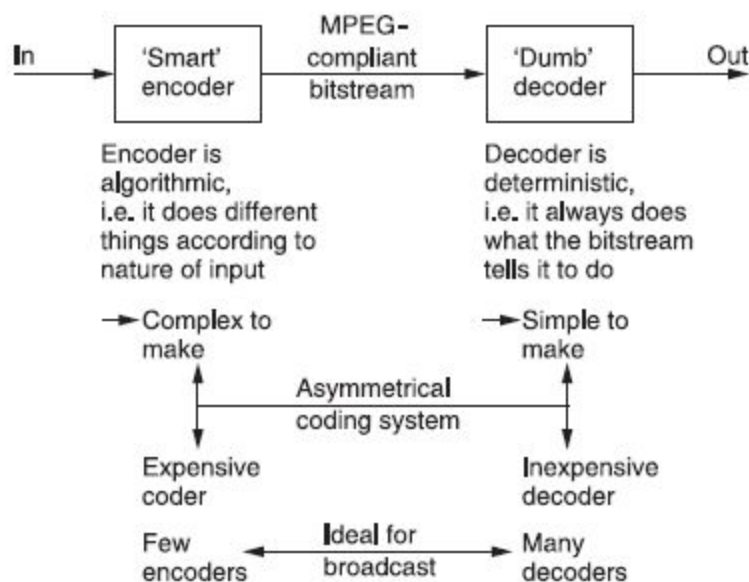
1.1 Introducere

MPEG este un acronim pentru Moving Pictures Experts Group care a fost format de ISO (International Standards Organization) cu scopul de a pune standardele pentru transmisia și compresia datelor de tip video și audio. Metoda de compresie a datelor este prezentată în Fig.1.a [1], aceasta constând în reducerea ratei de date la nivelul compresorului (coder), datele ce au fost compresate sunt transmise printr-un canal de comunicare, ca apoi, prin intermediul expandorului (decoder) să revină la rata inițială. Perechea coder-decoder formează un codec. Raportul dintre rata de date a sursei și rata de date a canalului se numește factor de compresie. Acest factor de compresie reprezintă un aspect important în a determina cât de bine vor fi micșorate datele, un factor cât mai mare ducând la o compresie cât mai bună a datelor.

După mai multe studii, s-a ajuns la concluzia că compresorul trebuie să aibă o structură mai complexă din punct de vedere algoritmic decât expandorul și să fie cât mai adaptiv. Expansorul, prin urmare, are ca principal țel să cunoască cum să interpreteze ce primește prin canalul de comunicare. Din această cauza, MPEG este definit ca fiind asimetric (Fig.1.b) [1]. Asimetria oferă un mare avantaj în aplicații unde se cere ca numărul de coderi să fie mici, iar numărul de decoderi mari. Astfel de aplicații implică difuzarea către telespectatori, unde partea compresorului este formată din cei care difuzează, iar ascultătorii sunt formați din expandori simpli, ce consumă un cost minim de resurse.



(a)



(b)

Fig.1.a Sistemul de compresie

Fig.1.b Asimetria metodei de compresie

1.1.1 Compresia datelor

Compresia datelor, reducerea ratei la care sunt transferați biții, reducerea datelor, toți acești termeni se referă în esență la transmiterea unei informații folosind o cantitate mai mică de date [3]. Sunt multe motive pentru care tehnicile de compresie sunt atât de populare:

- Compresia extinde timpul de difuzare al unui dispozitiv de stocare dat.
- Compresia admite miniaturizarea : cu mai puține date pentru stocare, același timp de difuzare este obținut cu un echipament mai slab.
- Cu mai puține date, densitatea de stocare poate fi redusă, astfel făcând echipamente care sunt mai rezistente la mediu înconjurător să fie mai ușor de întreținut.
- În sisteme de transmisie, compresia permite o reducere în lățimea de bandă care va rezulta în costuri mai reduse.

- Dacă o lăţime de bandă este validă, compresia permite un semnal cu o calitate mai bună folosind aceeaşi lăţime de bandă.

1.1.2 Principiile compresiei

Compresia datelor poate fi împărţită în două moduri, cea fără pierderi de date şi cea cu pierderi de date. Compresia fără pierderi de date constă în reducerea informaţiei sursă, atunci când informaţia compresată este decompresată nu există pierderi şi unul dintre cele mai importante aspecte este faptul că procesul este reversibil. Al doilea tip de compresie reduce informaţia sursă, însă când are loc decompresarea datelor se pierde o mică parte din informaţii [4].

Există două tipuri de componente ale semnalului: acele care sunt imprevizibile şi acelea care pot fi anticipate. Cele imprevizibile se numesc entropie şi reprezintă adevărata informaţie din semnal. Restul, este numit redundanţă, care poate fi de tip spaţial, ca în imaginile simple unde pixelii adiacenţi au aproape aceeaşi valoare, sau de tip temporal, ca de exemplu în imaginile succesive dintr-un film cu similarităţi mari. Toate sistemele de compresie separă entropia de redundanţă la nivelul coderului. Doar entropia este transmisă, iar la nivelul decoderului se reconstituie redundanţa din semnalul transmis. Acest concept este arătat în Fig.2.a [5].

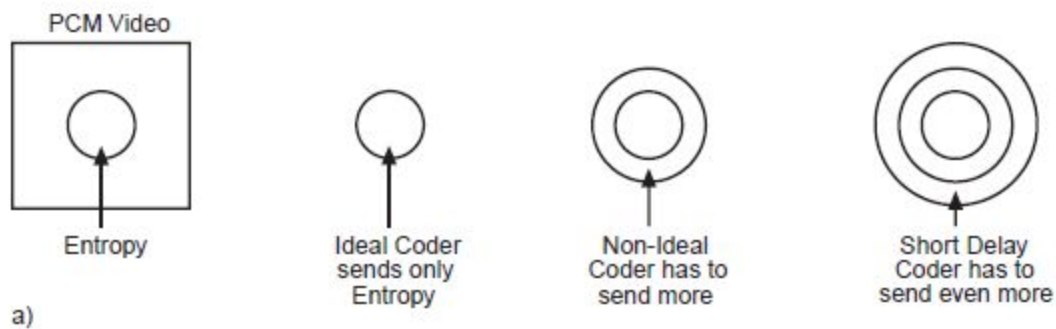


Fig.2. Conceptul de la baza compresorului

Într-un sistem ideal, doar entropia ar fi transmisă şi preluată de expander, însă acest ideal nu a fost atins încă. Deoarece adevărata informaţie nu poate fi anticipată, tehnicile de compresie exploatează redundanţa din semnal.

1.1.3 Tehnici ale compresiei

La baza compresiei datelor dintr-un cadru video stau două tehnici elementare ce elimină redundanța la nivelul compresorului, ca apoi să fie corect descifrat de către expander.

Prima tehnică se folosește de redundanța dintr-o imagine, și se bazează pe două caracteristici a imaginilor. Prima, nu toate frecvențele spațiale sunt prezente simultan, iar a doua, cu cât frecvențele spațiale sunt mai mari, cu atât amplitudinea este mai mică. Această analiză este scopul transformării cadrului spațial folosind DCT (conversia discretă de cosinus).

A doua tehnică exploatează redundanța temporală și se folosește de similaritățile găsite între imaginile succesive dintr-un film. De exemplu, dacă la nivelul expanderului o anumită imagine este validă, următoarea imagine poate fi reconstituită doar din diferențele dintre cele două.

În cadrul compresiei datelor de tip audio, lucrurile stau altfel. Un om nu discerne sunetele atât de bine cum o face o înregistrare, așa că tehnica de compresie exploatează acest deficit uman, tăind datele (valurile de sunet) care sunt considerate redundante (nu pot fi distinse de om), astfel rămânând o calitate bună a sunetului.

1.2 Compresia video

Tehnicile MPEG de compresie a datelor video exploatează redundanțele statistice aflate în domeniul temporal și al cel spațial. Metoda de bază care este folosită de acest standard de compresie se bazează pe corelația dintre pixeli arătat în Fig.3.a [6] și a corelării dintre cadrele dintr-un film consecutive (Fig.3.b) [6].

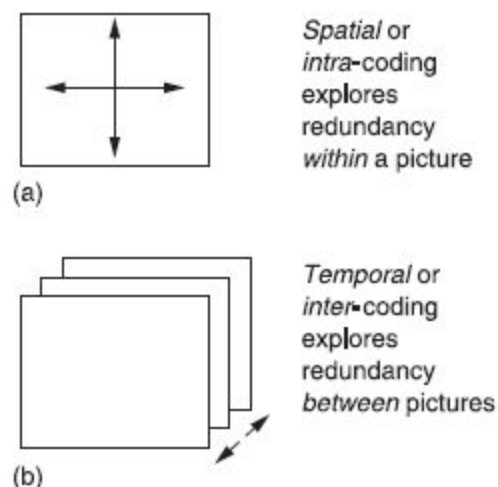


Fig.3.a. Tehnică de compresie bazată pe redundanța dintr-un cadru

Fig.3.b Tehnică de compresie bazată pe redundanța dintre cadre

1.2.1 Ochiul uman

Toate semnalele din imagini stârnesc o reacție al ochiului uman, deci rezultatul unei compresii este subiectiv din punctul de vedere al utilizatorului. Este esențial în cunoașterea compresiei unei imagini, cum funcționează ochiul uman. Retina este responsabilă pentru sensibilitatea la lumină și conține un număr de straturi prin care trebuie să treacă lumina pentru a fi percepută de om. Mai exact, ochiul uman este mult mai sensibil la schimbarea de luminozitate decât la schimbarea la nivel cromatic. Astfel, schema de compresie divide mai întâi imaginile în componente YUV, o componentă care conține o valoare ce semnifică luminozitatea din pixel, și două componente pentru reprezentarea culorii. Apoi, componentele ce reprezintă culoriile sunt reduse relativ la componenta de lumină cu o rație specificată de MPEG, și mai exact Y:U:V 4:1:1 sau 4:2:2.

1.2.2 Redundanța temporală

Mai întâi, cadrele sunt separate în blocuri disjuncte de pixeli adiacenți (16x16 pixeli). Când este folosită tehnica de compresie a redundanței temporale, imaginea curentă nu este trimisă către expander în întregime, ci diferența dintre cadrul curent și cadrul trimis anterior. Acesta având deja imaginea trimisă anterior, deci poate să umple noua imagine utilizând imaginea veche și diferența dintre cele două cadre. Acest concept de prezicere se mai numește și predicția compensată de mișcare și se bazează pe estimarea mișcării din imagine, mai exact este descris de un număr limitat de parametri ai mișcării care sunt ținuti în câte un vector pentru fiecare din blocurile descrise mai sus, după cum este prezentat în Fig.4 [7]. Vectorul și eroarea de prezicere sunt transmiși decoderului.

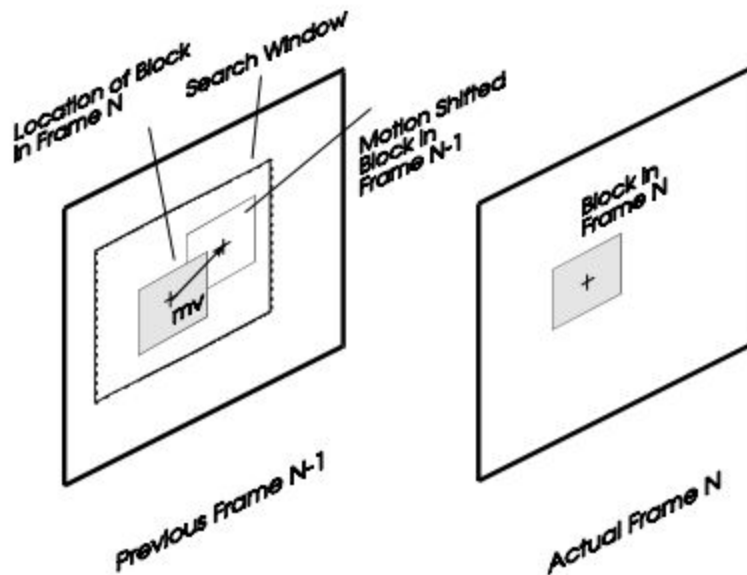


Fig.4. Aproximarea blocurilor folosit de tehnica compensației mișcării

Vectorul de mișcare (mv) este estimat pentru fiecare bloc de pixeli din cadrul curent N care trebuie să fie compresat. Vectorul de mișcare indică ca referință un bloc de pixeli de aceeași dimensiune care a fost compresat anterior în cadrul N-1 [7].

1.2.3 Redundanța spațială

Când spunem redundanță spațială ne gândim la pixelii adiacenți dintr-o imagine. Mai exact, exploatăm faptul că pixelii apropiați au aceeași valoare și folosim această informație pentru a anticipa anumite valori și a le trimite expanderului. Mai concret, prima etapă în compresia dintr-o imagine este de a transforma spațiul domeniului într-un domeniu de frecvențe utilizând o transformare discretă de cosinus.

1.2.3.1 DCT

Algoritmul este o versiune al algoritmului de transformare a cosinusului și implică transformarea unui bloc de pixeli (8 x 8) într-un bloc de coeficienți. La baza algoritmului se află transformarea acestor blocuri într-o formă care ajută la compresia datelor. Folosind schema prezentată în Fig.5 [14], fiecare coeficient rezultat din algoritm se împarte la valorile

corespondente din tabelul de cuantizare, care trece prin tabelul de valori și rezultă o nouă valoare a coeficienților.

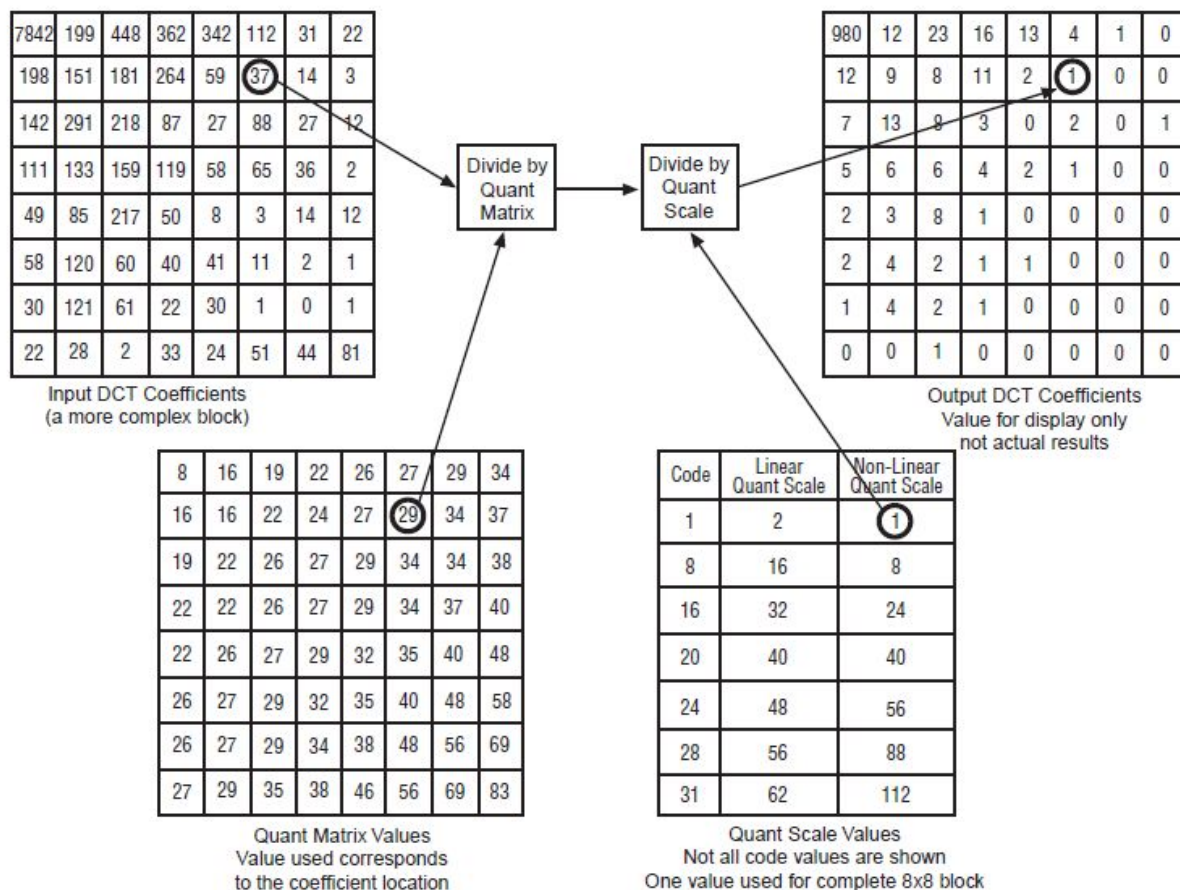


Fig.5. Tabelele DCT și cele de cuantizare

1.2.3.2 Scanarea

După cum putem observa din exemplul de mai sus, cele mai semnificative valori ale matricei se află la început (în partea din stânga sus), urmând ca valorile să scadă drastic spre finalul matricei (dreapta jos). Valorile foarte mici sunt trunchiate, ca apoi să fie trimise către expander valoarea zero și un număr care indică numărul de zerouri. Datorită faptului că valorile cele mai mari se află în partea de sus, are loc o scanare zig-zag care pornește din colțul cu cele mai semnificative valori și coboară către partea cu cele mai nesemnificative valori, după cum se poate observa în Fig.5.1 [8].

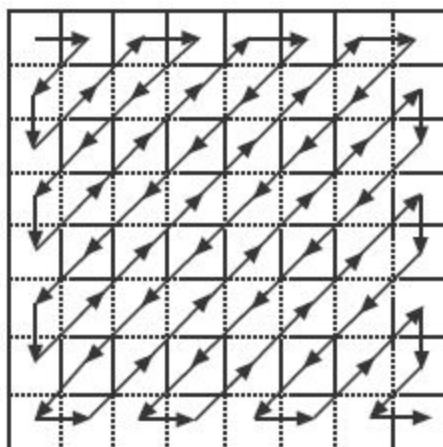


Fig.5.1 Scanarea zig-zag al unui bloc de coeficienți

Utilizând conceptele prezentate anterior, mai întâi transformând domeniul spațial într-un domeniu de frecvențe utilizând transformarea discretă a cosinusului, apoi cuantizând datele ca după să fie utilizat o scanare zig-zag, transmitem informațiile expanderului, acesta știind să opereze în sens invers și să ajungă, cu o mică pierdere de date, la rezultatul inițial.

1.2.4 Conceptele transmisiei într-un cadru video

În MPEG sunt definite trei tipuri de imagini numite imagini I, P, și B, pentru a putea avea o compresie de tip bidirecțională și mai ales pentru a minimiza rata de eroare. Pentru a înțelege mai bine acest concept, în Fig.6 [9] este prezentată ideea de compresie a imaginilor și modul în care sunt decompresate la nivelul expanderului.

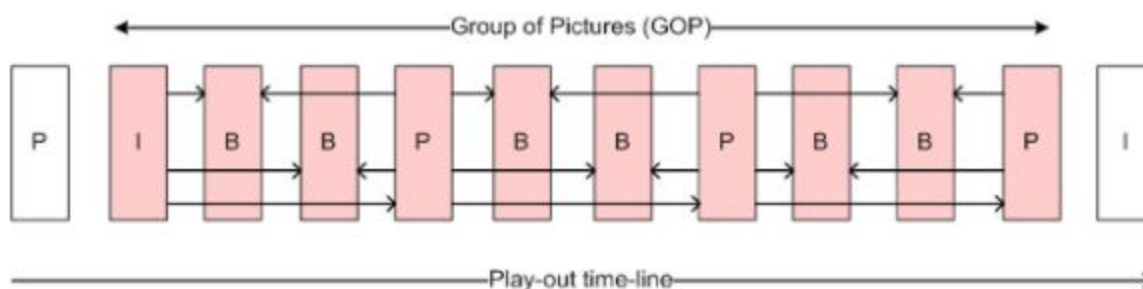


Fig.6. Reprezentarea unui grup de imagini

1.2.4.1 Imagini de tip I

Aceste imagini descriu imaginile pure, cele care nu mai au nevoie de informații adiționale pentru compresie, numite și imagini de compresie în cadru. Necesită multă memorie în comparație cu celelalte tipuri de imagini, sunt folosite ca punct de început într-o serie de cadre și ajută la oprirea propagării a erorii.

1.2.4.2 Imagini de tip P

Numite și cadre prezise dintr-o imagine anterioară, acest tip de dată este format de vectori care descriu de unde din imaginea anterioară să se uite decoderul pentru a prelua informații și a crea imaginea prezisă. Acestea folosesc jumătate din necesarul în creerea unui cadru de tip I.

1.2.4.3 Imagini de tip B

Imaginile de tip B, numite și bidirecționale, se folosesc de cadre de tip I și P pentru a se forma. Deoarece predicția de tip bidirecțional este foarte eficientă, memoria de care trebuie să dispună o astfel de imagine este cât un sfert din memoria alocată unei imagini de tip I.

1.3 Compresia audio

Mecanismul prin care oamenii aud este descris în fizică ca și distorbanțele ce se propagă în aer. Prin definiție, calitatea sunetului unui compresor poate fi reevaluat de auzul uman. Mai exact, un compresor perceptiv bun poate fi modelat cu o bună cunoaștere al mecanismului auzului uman [10]. Majoritatea compresiei este rezultată din scoaterea părților perceptuale irelevante din semnalul audio.

Compresorul funcționează în următorul fel: fluxul audio trece printr-un depozit de filtre care divide semnalul în 32 de sub benzi de frecvență cu lățimea egală, și în același timp trece printr-un model psihoacustic care determină rația energiei semnalului către limita de mascare a fiecărei sub bandă. Apoi, blocul alocat zgomotelor folosește rația calculată pentru a repartiza numărul de biți valabili pentru cuantizarea semnalelor sub-benzilor pentru a minimiza zgomotele și formatează aceste date și câteva informații suplimentare într-un flux de biți compresați, așa cum este prezentată în Fig.7 [13].

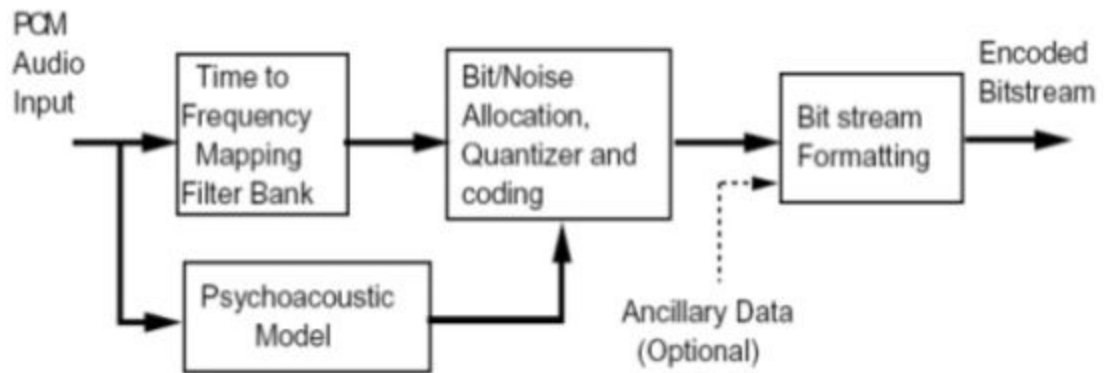


Fig.7. Arhitectura compresorului de date audio.

1.3.1 Psihoacustica

Algoritmul MPEG de compresie a datelor audio se datorează în mare la scoaterea părților irelevante dintr-un semnal audio. Mai exact, se folosește de inabilitatea mecanismului auditiv uman de a auzi anumite zgomote sub masca auditorie. Acest concept este o proprietate perceptuală care are loc atunci când un alt zgomot mai puternic se propagă în vecinătatea sunetului curent, făcând zgomotele mai mici de neauzit. O varietate de experimente psihoacustice sprijină acest fenomen de mascare [11].

Modelul psihoacustic prin care trece semnalul în procesul de compresie, analizează semnalul și calculează numărul de mascare a zgomotelor valabil ca o funcție de frecvență [12]. Această abilitate de mascare depinde de poziția frecvenței și a nivelului de zgomot. Compresorul folosește această informație pentru a decide cum să reprezinte semnalul audio primit.

1.3.2 Depozitul de filtre

Depozitul este componenta care se află în toate cele trei nivele din MPEG. Acesta împarte semnalul audio în 32 de subbenzi cu lățime egală, cu o complexitate relativ simplă ce produc o rezolvare destul de bună. Problema cu aceste subbenzi este aceea că nu reflectă sistemul auditor de comportament al omului în întregime, având mici scrupule. De exemplu, la frecvențe mici, o singură subbandă acoperă mai multe subbenzi critice, pentru o mai bună înțelegere, o vizualizare al comparației dintre subbenzile produse de depozit și cele critice este prezentat în Fig.7.1 [13]. O

altă problemă ar fi faptul că, chiar și fără cuantizare, procesul de ireversibilitate tot ar fi cu mici pierderi de date, în schimb, eroarea produsă de depozit este mică și neremarcată.

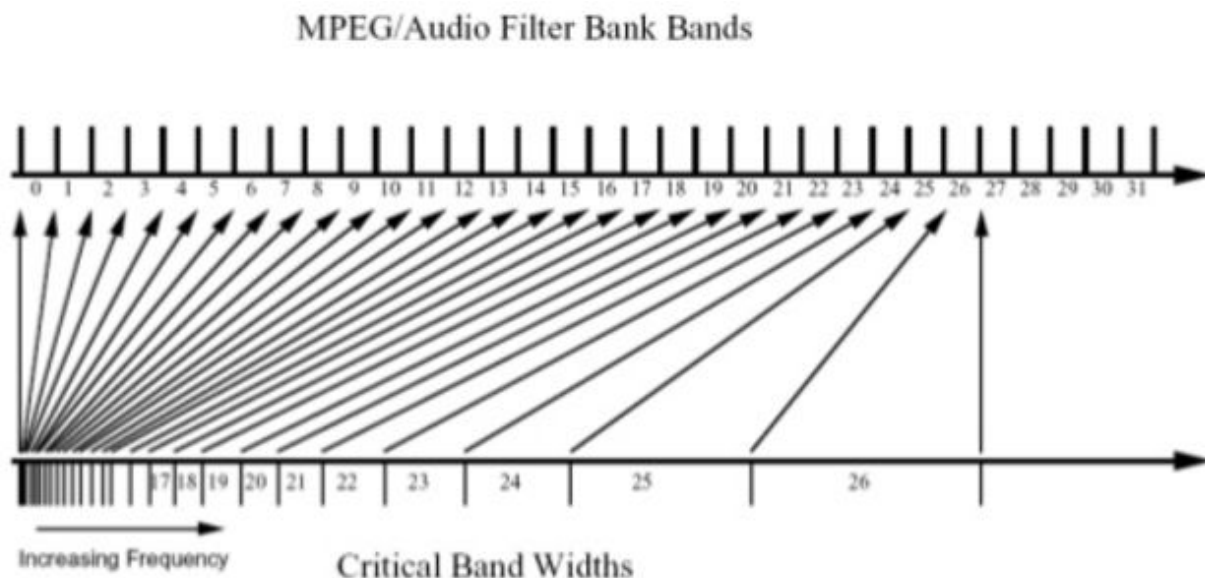


Fig.7.1. Diferența dintre subbenzile produse de mpeg și cele critice.

1.3.3 Straturile MPEG

Subiectul compresiei audio era destul de avansat atunci când s-a format grupul MPEG. Datorită acestui lucru, lucrurile nu au început chiar de la zero pentru autori, existând deja compresori de unde puteau adăuga funcționalitate. Ca parte a proiectului Eureka 147, un sistem cunoscut ca MUSICAM [15] a fost format de CCETT în Franța. MUSICAM a fost proiectat pentru a fi potrivit în domeniul difuzării audio digital.

Stratul unu al MPEG este o versiune simplificată al MUSICAM, care este folosit pentru compresie mică în aplicații de cost mic, mai exact pentru o rată de biți mai mare de 128 kbits/sec pe canal.

Stratul doi este identic cu MUSICAM și este folosit în compresia audio pentru Digital Audio Broadcasting (DAB) [16], pentru stocarea secvențelor sincronizate de video și audio pe CD-ROM.

Stratul al treilea este o combinație dintre cele mai bune caracteristici ale ASPEC și MUSICAM și este folosit în aplicații de telecomunicații unde sunt căutați factori de compresie mare.

1.3.3.1 Stratul 1

În Fig.7.1.2 [17] putem observa o pereche de compresor-expandor specific stratului 1 al compresiei MPEG, care reprezintă o versiune simplificată a MUSICAM. Un filtru divide spectrul audio în 32 de subbezi egale. Rezultatul filtrului de bancă este drastic simplificat, mai exact, rata de date nu este mai mare decât rata semnalului inițial pentru că fiecare subbandă a fost combinată cu un alt semnal de frecvență înaltă pentru a produce un semnal de frecvență joasă.

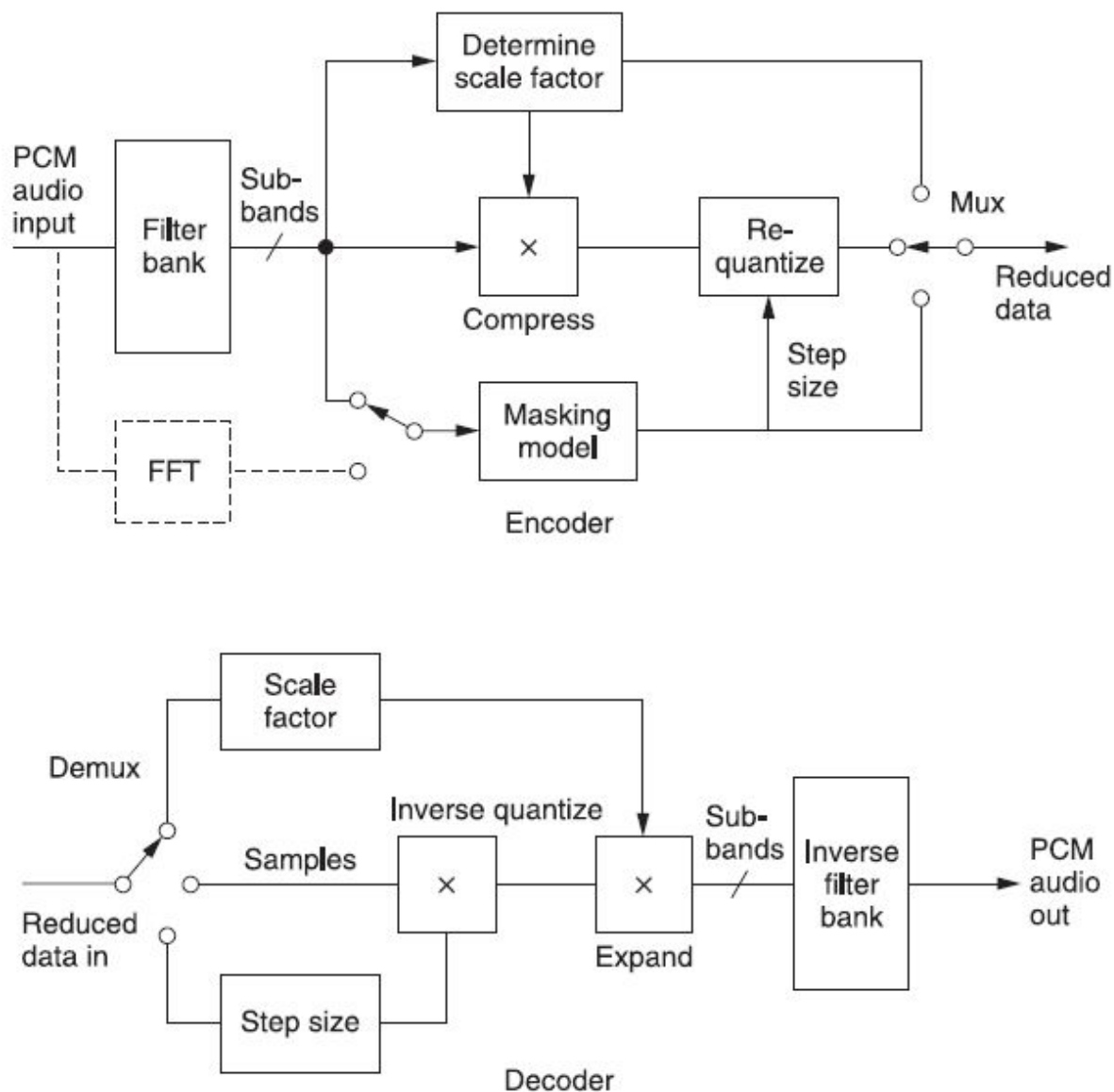


Fig.7.1.2. O pereche compresor-expandor folosit în stratul 1 al compresiei audio MPEG.

1.3.3.2 Stratul 2

Stratul al doilea este identic cu MUSICAM. Același depozit de filtre de 32-benzi și aceeași schemă care este folosită și în primul nivel. Diferența este că lungimea blocului este de trei ori mai mare decât în primul strat, corespunzând la 24 ms la 48 kHz. Fig.7.1.3 [17] ne arată că un expander de strat 2 trebuie să fie puțin mai complex decât cel din stratul precedent deoarece apar factorii de scalare și nevoia de a expanda granulele.

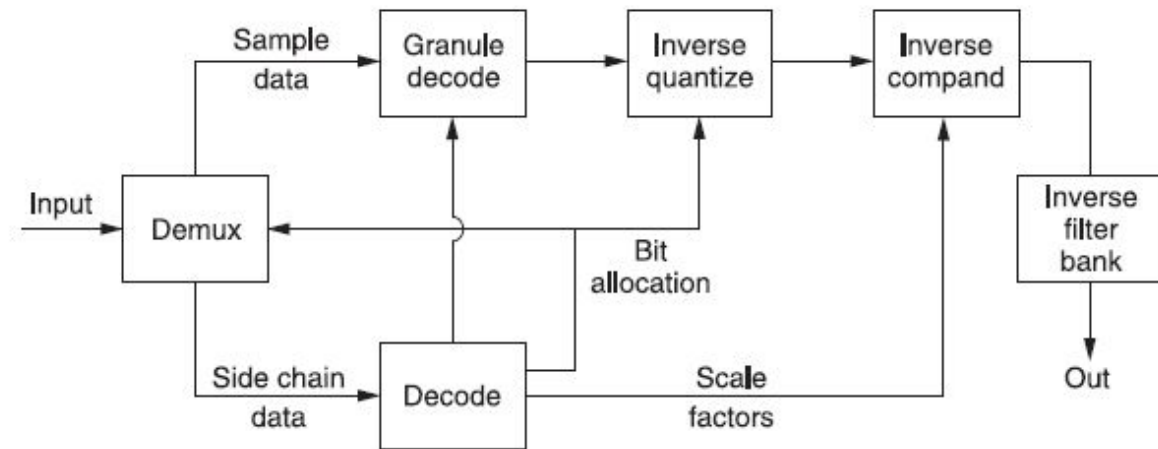


Fig.7.1.3. Un expander de strat 2.

1.3.3.3 Stratul 3

Este cel mai complex strat, și este necesar atunci când cele mai severe constrângeri a ratei de date trebuie respectate. Mai este cunoscut ca și MP3 în aplicația sa de transfer al muzicii pe Internet. Este bazat pe un sistem numit ASPEC [30] cu câteva modificări pentru a da un grad de rudenie cu cel de-al doilea strat.

În Fig.7.1.4 [17] este prezentat compresorul stratului, care ne conferă o mai bună vizualizare a conexiunii dintre cuantizator și “tampon”.

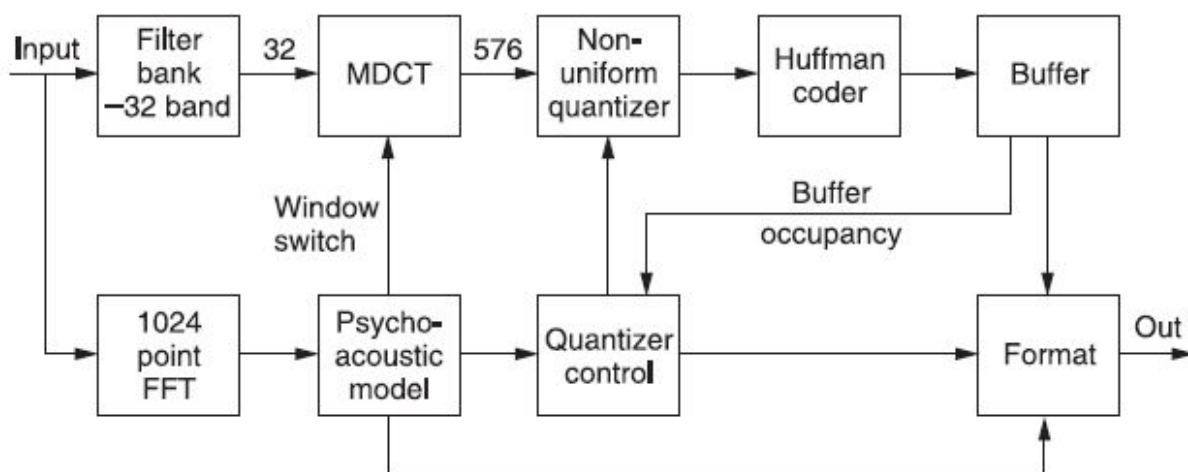


Fig.7.1.4. Compresorul stratului al treilea.

1.4 Fluxuri de biți

Fluxul de biți este rezultatul compresiei de tip audio sau video, numit și fluxul elementar și poate varia în funcție de nevoia în aplicație a acestuia, așa cum este arătat în Fig.8.1 [18]. În transmisia televizată, aceste fluxuri sunt îmbinate și crează un așa numit flux de transport. Îmbinarea semnalelor video cu audio necesită pachete de dimensiune fixă, fiind avantajos ca acestea să aibă o dimensiune cât mai mică. Fluxul de transport are o structură mai complexă deoarece are nevoie de o secțiune specială de metadata ce indică ce pachete audio coincid cu pachetele video. Este posibil de a avea un singur program de flux de transport (SPTS) care conține fluxul elementar dintr-un singur program TV.

Un program de flux este un flux de biți simplificat care îmbină audio și video într-un singur program. Față de fluxul de transport, structura acestuia este compusă din blocuri mai mari, cu o dimensiune nefixă.

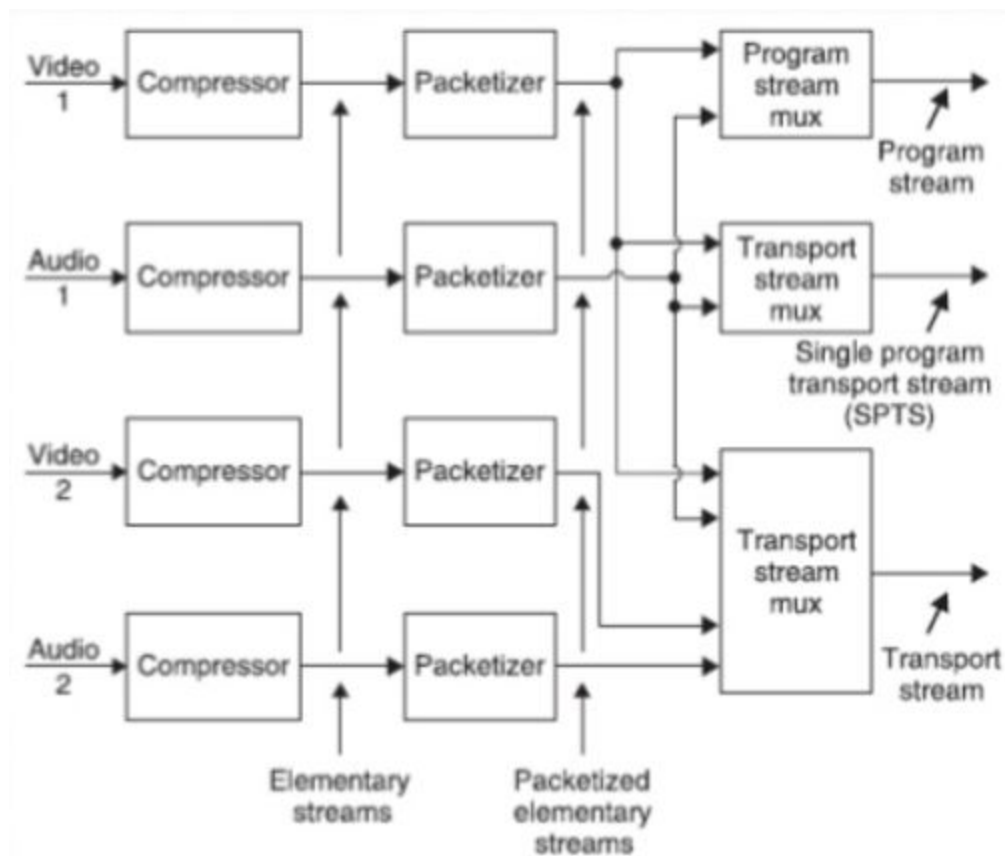


Fig.8.1 Tipurile de fluxuri de biți ale MPEG-2

1.4.1 Pachete și ștampile de timp

Fluxul elementar video este un flux infinit de biți care reprezintă imagini care nu sunt în aceeași ordine neapărat și ce au nevoie de un timp variat de transmisie. În MPEG, fluxurile elementare sunt împachetate, formând un flux elementar împachetat (PES) așa cum este prezentat în Fig.8.2 [19]. Pachetul este format dintr-un antet ce conține un indentificator unic de început de pachet și un cod ce reprezintă tipul de date din flux. Antetul mai poate conține și una sau mai multe ștampile de timp care sunt folosite în sincronizarea expanderului video cu timpul real pentru a obține sincronizarea dintre video și audio.

Packet Start Code Prefix	Stream ID	PES Packet Length	"10"	PES Header Flags	PES Header Length	PES Header Fields	PES Packet Data Block
3 bytes	1 byte	2 bytes	2 bits	14 bits	1 byte		

Fig.8.2 Reprezentarea structurii unui flux elementar împachetat

În Fig. 8.3 [20] este prezentată procesul de formare a unei ștampile de timp, care este o stare a unui contor care se folosește de un ceas de 90 kHz. Acesta este obținut prin împărțirea ceasului primar de 27 MHz al MPEG.

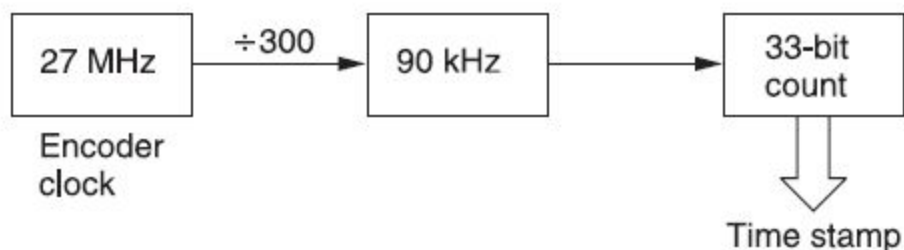


Fig.8.3 Reprezentarea formării unei ștampile de timp

Există două tipuri de ștampile, ștampila de prezentare care determină când imaginea asociată ar trebui să fie afișată și ștampila de expandare care determină când imaginea ar trebui să fie expandată.

1.4.2 Fluxul de transport

Fluxul de transport este o îmbinare a mai multor programe TV cu canalele de date și sunet asociate. Fluxul se bazează pe pachete de dimensiune fixă de 188 biți ce ușurează adăugarea de corecție a erorii la un strat mai înalt. Pachetele au în compoziție un antet care este relativ mic pentru o eficiență mai bună, dar poate fi extins pentru anumite scopuri, și datele propriu-zise, reprezentat în Fig.8.4.a [21].

În începutul compoziției antetului este un bit unic care ajută expansorul în despachetarea datelor. Un flux de transport poate conține mai multe fluxuri elementare și de aici vine nevoia de un cod de identificare de 13 biți pentru fiecare dintre aceste fluxuri, care este inclus în antet. Atunci când are loc multiplexarea, un flux de transport poate conține pachete din multe alte programe, așa că există un contor ce crește în funcție de pachetele care vin ce au același cod de identificare. Această abordare va găsi întotdeauna toate pachetele cu același cod unic, indiferent de cum sunt primite la nivelul expanderului. Pentru o mai bună înțelegere, în Fig.8.4.b [29], folosind **DvbExplorer** [28], este ilustrat informațiile și compoziția unui fișier cu extensia .ts, mai exact un fișier ce poate fi incapsulat într-un flux de transport.

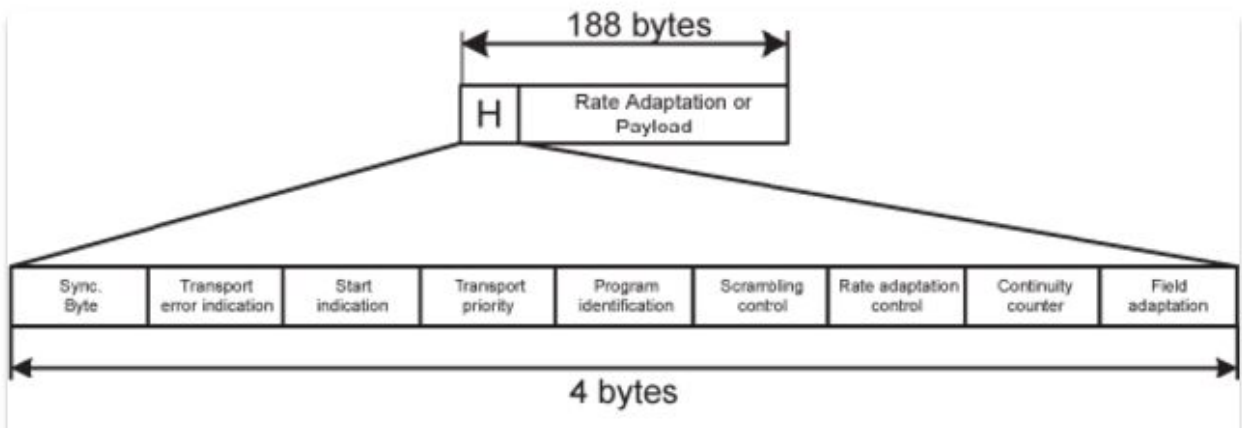


Fig.8.4.a. Reprezentarea structurii unui pachet al fluxului de transport.

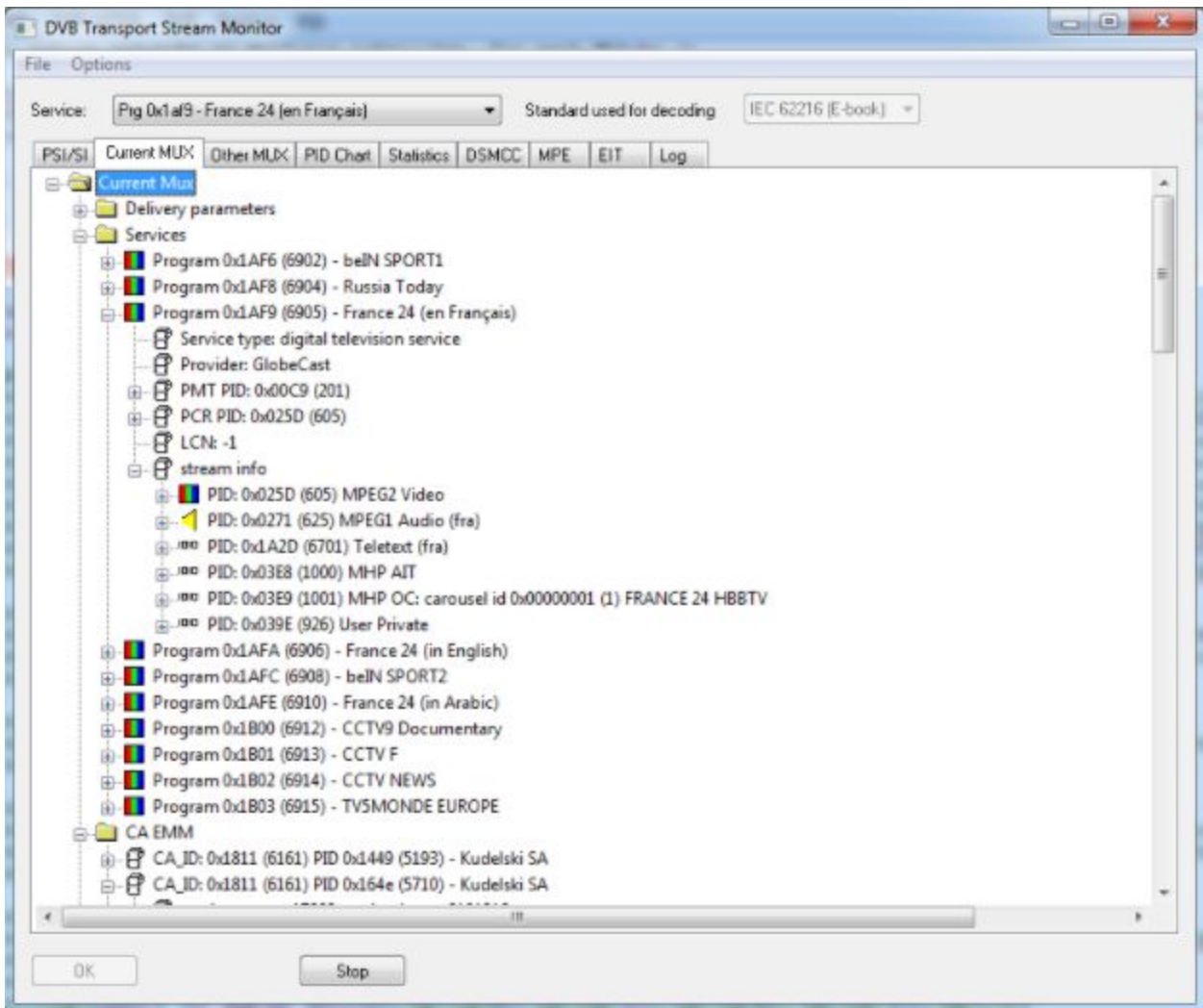


Fig.8.4.b Reprezentarea sub formă de arbore a informațiilor dintr-un fișier .ts

1.4.3 Multiplexarea

Un multiplexator al fluxului de transport este un dispozitiv complex pentru că are de îndeplinit mai multe funcții. Într-un multiplexor fix, rata de biți pentru fiecare program trebuie specificată deoarece suma tuturor datelor din pachete nu trebuie să fie mai mare decât rata de biți a fluxului de transport.

Fiecare flux elementar trece mai întâi printr-un depozit care este apoi împărțit în zone cu datele pachetelor. Multiplexorul trebuie să ia pachetele din fiecare program astfel încât să nu aibă loc umplerea mai mult decât trebuie a zonei respective. Pentru a atinge acest rezultat, acesta trebuie să primească pachetele cât mai ordonat din programul respectiv. Cu cât zona respectivă este mai plină, cu atât este o șansă mai mare ca un pachet să fie citit și scos. Acest proces este ilustrat în Fig.8.5 [17].

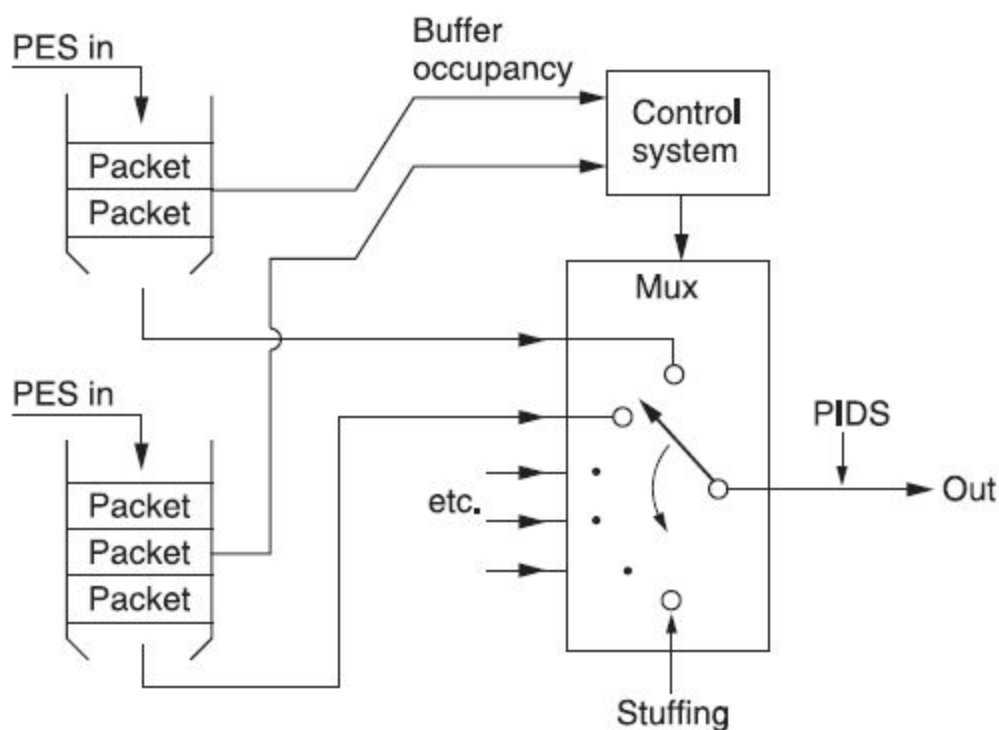


Fig.8.5 Reprezentarea procesului de multiplexare, având mai multe programe asincrone între ele

Capitolul 2

FFMPEG

2.1 Aplicații

FFmpeg este o aplicație open-source care se ocupă cu crearea și îmbunătățirea librăriilor și programelor ce produc și manevrează datele multimedia. Pe lângă acestea, implementează protocoale și metode de comunicare prin rețea.

Este o aplicație de tip command-line, care convertește datele audio precum și cele video, fiind unealta de bază al întregului concept. Unul dintre atuurile sale este convertirea din aproape orice format video, precum și manipularea acestora cu scop informativ, ca de exemplu comanda din Fig.9.1 va copia datele din fișierul dat ca input și va crea un nou fișier de tip video cu același conținut, dar cu rata de biți de 64kb/s.

```
[andreeiii@localhost ~]$ ffmpeg -i video.mkv -b:v 64k -bufsize 64k output.mkv
```

Fig.9.1 Exemplificarea unei comenzi de tip ffmpeg

Din comparația Fig.9.2.a cu Fig.9.2.b se poate vedea clar diferențele în calitate ce pot apărea dacă rata de biți nu atinge pragul nevoit.



Fig.9.2.a Imagine preluată din fișierul dat ca input



Fig.9.2.b Imagine preluată din fișierul dat ca output

Procesul de codificare prin care a trecut crearea fișierului respectiv este exemplificat în diagrama din Fig.9.3 [21]. Mai întâi, ffmpeg se folosește de librăria libavformat pentru a citi și extrage pachete de date din fișierul de input, ca apoi aceste pachete să fie transportate expandrolui, iar acesta să le despacheteze și, după filtrare să fie împachetate din nou și trimise compresorului, care le scrie în fișierul rezultat.

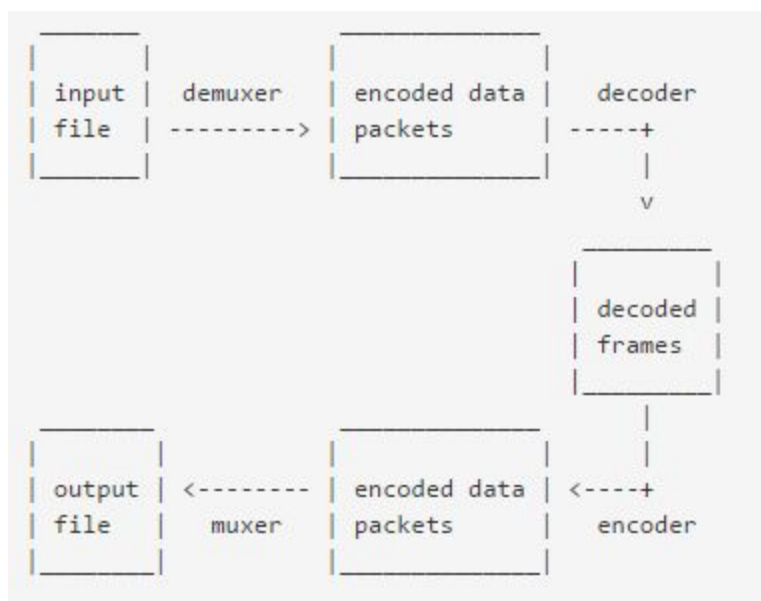


Fig.9.3 Procesul de codificare

2.1.1 Protocol de comunicare

Unele dintre facilitățile pe care le mai produce și menține FFmpeg sunt protocoalele de comunicare implementate de aceștia. Au conexiuni de tip TCP/IP pentru a avea o comunicare sigură, precum și UDP pentru difuzare de imagini în timp real.

În Fig.9.4 este prezentată o comandă ce crează o comunicare pe rețea de tip RTP, care este un “wrapper” peste UDP, fiind puțin mai sigură și de un control de erori mai riguros, mai exact este creată partea transmițătorului, în care sunt transmise imagini în timp real preluate de la o cameră video.

```

andrei@ubuntu:~$ ffmpeg -f dshow -i video="Virtual-Camera" -vcodec libx264 -tune zerolatency -b 900k -f mpegts udp://10.1.0.102:1234
  
```

Fig.9.4 Transmiterea datelor preluate dintr-o cameră video către un ip

2.2 FFprobe

Cea de-a doua unealtă de care dispune ffmpeg, este ffprobe, cu utilitatea de a strânge informații din mediul multimedia și de a-l expune într-o manieră lizibilă. Modul de utilizare este unul progresiv, fluent, ce implică totuși o anumită cunoaștere a utilizatorului asupra manipulării acestuia.

Se poate utiliza ca o aplicație de tip stand-alone, sau în combinație cu filtru de text, se pot enunța anumiți parametrii pentru a specifica ce informație să fie afișată.

Poate fi folosit, ca de exemplu, pentru verificarea și eventual corectarea unui fișier, așa cum este arătat în Fig.9.5, comandă ce are ca rezultat extragerea ratei de biți a cadrelor video din primul flux de biți al fișierului indicat.

```
andrei@ubuntu:~/Licenta/Program2$ ffprobe -v error -select_streams v:0 -show_entries stream=avg_frame_rate -of default=noprint_wrappers=1:nokey=1 video2.mp4
30000/1001
```

Fig.9.5. Exemplu de utilizare ffprobe

FFmpeg se folosește implicit de această aplicație pentru a cunoaște anumite informații despre fișierele manipulate. După cum se poate observa în Fig.9.6, sunt afișate în formă lizibilă informații privind fișierul specificat.

```
[andreeiii@localhost Videos]$ ffmpeg -i output.mkv -hide_banner
Input #0, matroska,webm, from 'output.mkv':
  Metadata:
    MINOR_VERSION      : 0
    COMPATIBLE_BRANDS  : iso6avc1mp41
    MAJOR_BRAND        : dash
    ENCODER             : Lavf56.25.101
  Duration: 00:00:44.44, start: 0.000000, bitrate: 171 kb/s
  Stream #0:0(und): Video: h264 (High), yuv420p, 1280x720 [SAR 1:1 DAR 16:9], 29.97 fps, 29.97 tbr, 1k tbn, 59.94 tbc (default)
  Metadata:
    CREATION_TIME      : 2015-04-22 07:55:06
    LANGUAGE           : und
    HANDLER_NAME       : VideoHandler
    ENCODER            : Lavc56.26.100 libx264
  Stream #0:1(eng): Audio: vorbis, 44100 Hz, stereo, fltp (default)
  Metadata:
    LANGUAGE          : eng
    ENCODER           : Lavc56.26.100 libvorbis
```

Fig.9.6. Exemplu de afișare în urma utilizării unelei ffprobe

2.3 FFplay

FFplay este unealta portabilă ce face posibilă redarea filmelor și a fișierelor audio. Se folosește de librăriile dispuse de ffmpeg, precum și de cele ale SDL și este în special folosită pentru testare.

Modul de utilizare este unul simplist, prezentat în Fig.9.7, conceput astfel pentru a avea o înțelegere deplină cu utilizatorul.

```
ffplay [options] [input_url]
```

Fig.9.7 Mod de utilizare ffplay

Una din metodele în care poate fi folosită unealta de redare multimedia prin înșiruirea de comenzi prin specificarea parametrului “pipe:[0|1]”. Prima comandă din Fig.9.8.a transmite informațiile fișierului video exemplificat către a doua comandă, care o redă în timp real, prezentat în Fig.9.8.b.

```
andrei@ubuntu:~/Licenta/Program2$ ffmpeg -i video0.flv -f avi pipe:1 | ffplay -i -
```

Fig.9.8.a Comenzi înșiruite



Fig.9.8.b Fișierul de tip video afișat

2.4 Exemple de utilizare

După cum am menționat și mai sus, unul dintre atuurile de bază ale aplicației este manipularea și conversia datelor de tip video și audio. În continuare voi prezenta exemple de comenzi ale ffmpeg.

```
andrei@ubuntu:~/Licenta/Program2$ ffmpeg -i output.mkv -vn -ar 44100 -ac 2 -ab 192 -f mp3 audio.mp3
```

Fig.9.9.a Comandă exemplu unu

Comanda exemplificată mai sus preia fluxul de biți de tip audio din fișierul dat ca input și îl copie într-un nou fișier de tip mp3, argumentele date comenzii având ca scop setarea anumitor parametri.

În primul capitol am prezentat din ce este compus un flux de biți de tip transport, mai exact că acesta conține un cod de identificare pe care putem să îl manipulăm prin folosirea comenzii din Fig.9.9.b.


```
andrei@ubuntu:~/Licenta/Program2$ ffmpeg -i input.ts -codec:v:0 copy -codec:a:1 copy -streamid 0:52 -stramid 1:54 -f mpegts output.ts
```

Fig.9.9.b Comandă exemplu doi

O altă funcționalitate a aplicației este de a manipula nivelul de la care se poate reda o înșiruire de cadre, mai exact conceptul de “zoom” care este format din doi pași. Mai întâi scalează filmul la dimensiunea dorită, apoi taie fișierul video la dimensiunea originală.

```
andrei@ubuntu:~/Licenta/Program2$ ffmpeg -i video0.flv -vf "scale=2*iw:-1, crop=iw/2:ih/2" output.mp4
```

Fig.9.9.c Comandă exemplu trei

În urma comenzii prezentate mai sus, atingem funcționalitatea descrisă, comparativ cu imaginea prezentată mai sus, aceasta este privită la un nivel mult mai apropiat (Fig.9.9.d).



Fig.9.9.d Imagine manipulată prin tehnica “zoom”

Capitolul 3

Arhitectura și implementarea aplicației

3.1 Modelul client-server

Ideea de bază a aplicației se rezumă la un transfer de date pe rețea de tip client-server, în care am implementat propriul protocol peste UDP folosind socket.

Am ales UDP peste TCP deoarece funcționalitatea pe care am vrut să o ating este aceea de a transfera într-o manieră cât mai rapidă date între două entități, fără a avea nevoie de anumite confirmări și alte verificări de erori ce sunt încapsulate în protocolul de tip TCP. Datagramele UDP sunt caracterizate prin lungimea acestora, iar prin manipularea acestora putem avea o rată de transfer variată. Așa cum se observă în Fig.10 [22], o datagramă UDP conține un antet în care sunt specificate adresa sursei și a destinației și datele propriu-zise.

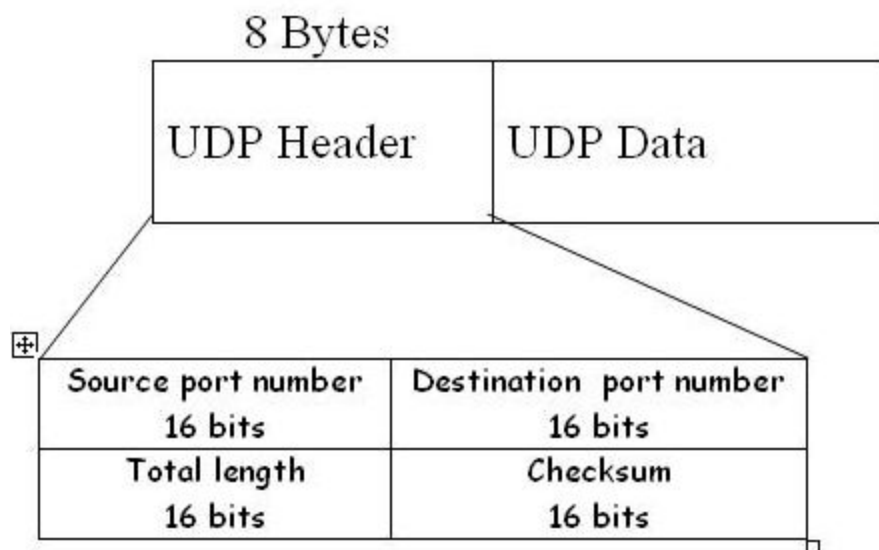


Fig.10. O datagramă UDP

În momentul în care pornim serverul, trebuie să specificăm lungimea pachetelor UDP și fișierul pe care îl transferăm, în cazul nostru un fișier cu extensia .ts care conține mai multe “fișiere” de tip video și audio așa cum este sugerat în Fig.10.1.

```
andrei@ubuntu:~/Licenta/Server$ ./a.out 2560 football.ts
```

Fig.10.1. Comanda de pornire a serverului

În Fig.10.2 este prezentată succint modelul client-server al aplicației și modul în care interacționează cele două.

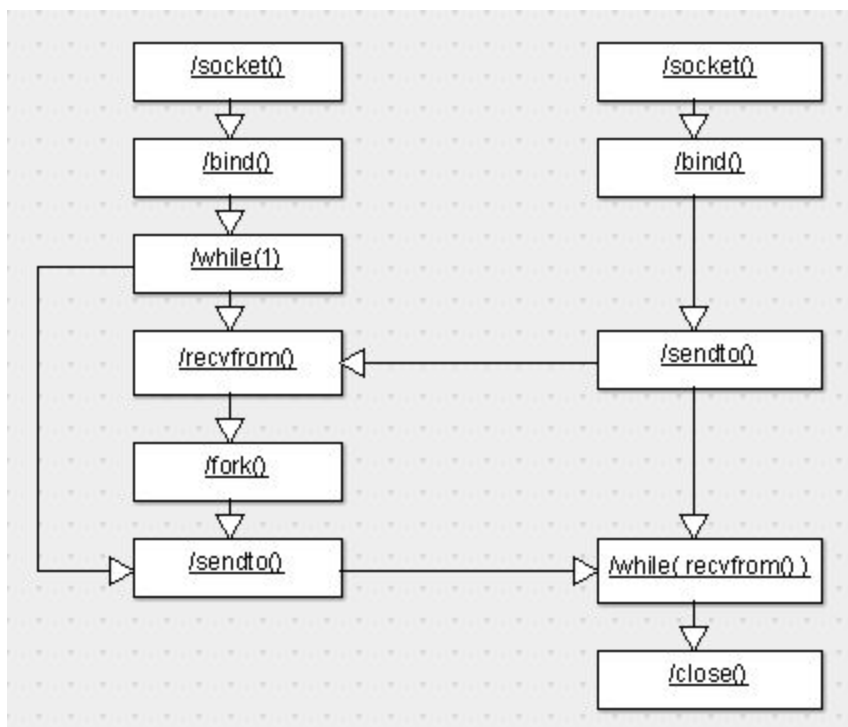


Fig.10.3. Modelul client-server al aplicației

Procesul nu este unul greu de înțeles, amândouă fac bind la adresă, serverul servește clienții în mod iterativ, câte unul pe instanță. Clientul face o cerere serverului prin comanda din Fig.10.4, parametrii fiind adresa de conectare a serverului și portul la care este atașat. După ce s-a conectat, i se afișează un panou cu filmele valabile pentru vizualizare și alege indexul fișierului.

```
andrei@ubuntu:~/Licenta/Client$ ./a.out 127.0.0.1 8554
```

Fig.10.4. Comanda de conectare și cerere la server

3.2 Clasa Queue

Queue este clasa care implementează o coadă de tip primu intrat, primu ieșit ce stă la baza aplicației. Este implementată în partea clientului și are ca scop primirea datelor de la server, punerea lor în coadă și apoi transferarea lor către un pipe.

3.2.2 Membrii clasei

Membrul clasei care stă la baza acesteia este o matrice buf de tip char ce conține un număr predefinit de elemente a câte o dimensiune variabilă de caractere definite de server. În interiorul acestei variabile sunt stocate pachetele primite din partea serverului, alocarea fiind dinamică.

Modul în care sunt extrase pachetele este implementat prin ajutorul a două variabile integer read_ptr și write_ptr. Acestea dau circularitatea bufferului, read_ptr arătând spre zona din matrice de unde putem extrage datele, iar write_ptr poitând spre zona de unde putem scrie pachetele.

Ultima variabilă definită în clasă este un counter ce reprezintă numărul de elemente curente al bufferului, aceasta fiind folosită în două metode a clasei.

3.2.3 Metodele clasei

Clasa Queue împarte funcționalitatea cu a unei clase de tip “circular buffer”, cu patru funcții predefinite ce stau la baza implementării acesteia.

```
void push(char element[]) // scrie de la server in buffer
{
    MutexGuard g_lock;
    buf[write_ptr] = (char*)malloc(chunkSize);
    memcpy (buf[write_ptr], element, chunkSize);
    counter++;
    write_ptr++;
    write_ptr = write_ptr % NELEMENTS;
}
```

Fig.11.1. Funcția push a clasei Queue

Așa cum este prezentată în Fig.11.1, funcția push alocă dinamic memorie la poziția indicată, adaugă un element primit ca parametru în matrice, crește numărul de elemente curente și mută pointerul de scriere să indice spre următoarea zonă din buffer.

Funcția pop, fiind în antiteză cu funcția prezentată anterior, copie într-o variabilă zona de memorie aflată la poziția curentă a pointerului de citire, numărul de elemente este decrementat, read_ptr indică următoarea poziție din matrice și se face o verificare, anume cât timp nu este primit un anumit string (stringul ce anunță încheierea primirii datelor), este scris elementul respectiv către un pipe.

Ultimele două metode implementate de clasă sunt funcțiile isFull respectiv isEmpty ce returnează un bool, cu funcționalitatea de a specifica atunci când numărul de elemente din buffer este maxim, respectiv zero.

3.3 Multithreading

Conceptul pe care îl voi prezenta în acest sub capitol este o tehnică prin care un singur set de instrucțiuni poate fi folosit de mai multe procese în diferite stagii ale execuției acestora, mai exact împarte o sarcină grea în mai multe fire de execuție ce se execută în același timp, astfel câștigând timp.

În implementarea acestei tehnici stau la bază threadurile POSIX, mai exact firele de execuție ce țin de platforma LINUX. Un astfel de fir de execuție conține mai multe date despre el, mai exact ID-ul, și atunci când este creat primește ca parametru o funcție ce reprezintă instrucțiunea pe care trebuie să o execute, crearea lor fiind specificată în Fig.11.2.

```
// Creez primul thread, care ia datele primite de la server si scrie in buffer  
rc1 = pthread_create(&writeT, NULL, &writeToBuf, (void*)&length);
```

Fig.11.2. Crearea unui thread POSIX

În partea clientului, sunt implementate două astfel de fire de execuție ce ajută la un paralelism pe o zonă de memorie comună, mai exact bufferul descris în clasa Queue.

3.3.2 Firul de execuție writeT

Threadul writeT are ca funcționalitate scrierea în buffer a datelor primite de la server. Funcția pe care o execută, writeToBuf, are la bază transferarea pachetelor primite de la server către buffer și este formată dintr-o buclă while care se execută cât timp vin date de la acesta. În interiorul buclei este făcută și o verificare dacă bufferul este plin, în cazul acesta se apelează metoda usleep() ca procesul respectiv să aștepte să se mai elibereze din matrice. După terminarea primirii datelor, este transmis un string specific către matrice ca să denote terminarea acestuia.

3.3.3 Firul de execuție readT

Threadul readT are ca funcționalitate scoaterea din buffer a datelor și trimiterea către pipe, aceste funcționalități fiind implementate de clasa Queue așa cum a fost prezentat anterior. Funcția pe care trebuie să o execute threadul conține o buclă de tip while ce se execută atâta timp cât nu este citit acel string specific de terminare a datelor.

Amândouă threaduri folosesc metoda de sistem usleep(), ceea ce stă la baza implementării propriului protocol peste UDP, transferul de date fiind unul fluent, fără posibilitatea de overrun sau overflow a bufferului.

În funcția main, trebuie să așteptăm ca ambele threaduri să își termine execuția înainte de a închide aplicația, funcționalitate implementată de metoda join, așa cum este ilustrată în Fig.11.3.

```
pthread_join(writeT, NULL);  
pthread_join(readT, NULL);
```

Fig.11.3. Apelul funcțiilor join

3.4 Pipe

În sistemele de operare UNIX, un pipeline este o secvență de procese înlănțuite, astfel încât rezultatul primei instrucțiuni este folosit ca input în următoarea instrucțiune ș.a.m.d. Procesul este arătat în Fig.12.1.a [23], unde primul program primește datele de la tastatură și trimite rezultatul către al doilea, al doilea procesează datele primite și trimite rezultatul către al treilea program, iar acesta le afișează.

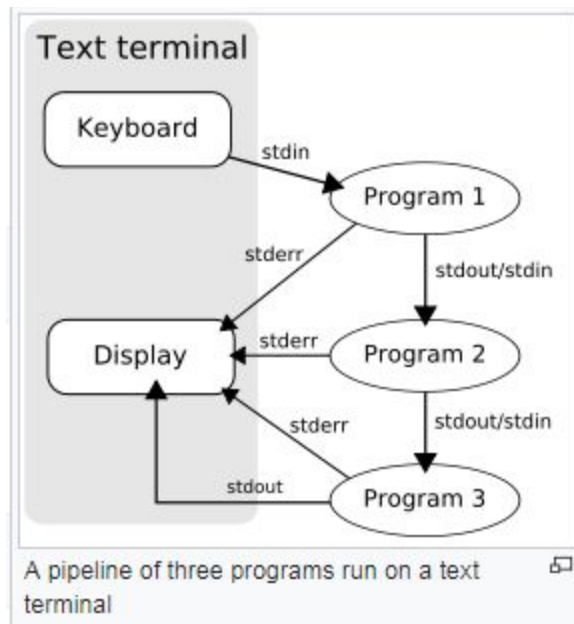


Fig.12.1.a Conceptul de pipeline a trei programe

Un exemplu de astfel de comandă este prezentat în Fig.12.1.b.

```
andrei@ubuntu:~/Licenta/Client$ ls -l | grep key | less
```

Fig.12.1.b. Înșiruire de trei comenzi

Aplicația mea folosește acest concept, mai exact folosește un pipe cunoscut de server și de client, care este un fișier creat de client prin comanda din Fig.12.2. După instrucțiunea respectivă este creat la adresa specificată un fișier de tip “named pipe”.

```
mknod ( "/tmp/mypipe", S_IFIFO, 0 );
```

Fig.12.2. Instrucțiunea de creare pipe

După ce am creat pipeul, îl deschidem din aplicație cu dreptul de write, care este blocant până când un alt proces citește din el. Aici intervine instrucțiunea ilustrată în Fig.12.3 , care se folosește de playerul implementat de aplicația ffmpeg pentru a citi date din pipe și a afișa rezultatul utilizatorului, mai exact videoul cerut.

```
andrei@ubuntu:~/Licenta/Client$ ffmpeg /tmp/mypipe
```

Fig.12.3 Instrucțiune de citire din pipe

3.5 Descrierea aplicației

Implementarea finală a aplicației constă în modelarea unei rețele de tip client-server, fiecare fiind descris în subcapitolele ce urmează. Serverul se folosește de un program intermediar numit CreateFiles, ce stă la baza funcționării programului.

3.5.1 CreateFiles

Programul intermediar are ca scop extragerea fluxurilor de date dintr-un fișier cu extensia .ts și manevrarea acestora astfel încât să le facă “pipe-ready”. Pentru a înțelege funcționalitatea acestuia, este nevoie de cunoașterea librăriei implementată de ffmpeg și mai exact conceptele din spatele acesteia. Oricărui fișier de tip media i se atribuie un AVFormatContext care reprezintă structura de dată principală. Următorul pas este de a deschide un fișier media cu instrucțiunea din Fig.12.4 și copierea metadatelor într-o structură de tip AVFormatContext.

```
ret = avformat_open_input(&ifmt_ctx, in_filename, 0, 0)
```

Fig.12.4. Atribuirea contextului variabilei ifmt_ctx al fișierului in_filename

A doua structură de date esențială este AVOutputContext, care descrie contextul fișierului output ce va fi creat, folosind metoda avformat_alloc_output_context2. După ce am construit variabilele necesare, ne plimbăm prin streamurile fișierului dat ca input și populăm variabilele ce țin de fișierul output cu datele găsite în acesta. Aici intervine puterea utilizatorului, dacă dorește să extragă videoul de la indexul 1 trebuie să specifice în comanda de rulare. Instrucțiunile din Fig.12.5 modifică fișierul final astfel încât să poată fi pipe-ready, mai exact îi mută anumite date ce normal stau la finalul fișierului, la început, pentru ca playerul să se poată executa, toate scrise în antetul acestuia.

```
AVDictionary *d = NULL;  
av_dict_set(&d, "frag_size", "1048576", 0);  
ret = avformat_write_header(ofmt_ctx, &d);
```

Fig.12.5. Manipularea și scrierea antetului fișierului final

În final, într-o buclă se copie pachetele ce constituie streamul din fișierul dat ca input, citindu-se cadru cu cadru prin funcția `av_read_frame`, punând datele în structura `AVPacket`, însă pachetele mai trebuie manevrate astfel încât decompresorul să cunoască momentul în care pachetele să fie decompresate și durata fiecărui pachet, exemplificat în Fig.12.6, fiind apoi transferate datele din pachete către fișier prin funcția `av_interleaved_write_frame`.

```
pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base, out_stream->time_base, AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX);
pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base, out_stream->time_base, AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX);
pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
pkt.pos = -1;
log_packet(ofmt_ctx, &pkt, "out");
```

Fig.12.6. Copierea ștampilei de timp

Modul de operare al acestui program este ilustrat în Fig 12.7. După efectuarea acesteia va fi creat un nou fișier cu denumirea `video0.mp4`, mai exact videoul de la indexul 0 pregătit pentru a fi transferat printr-un pipe.

```
andrei@ubuntu:~/Licenta/FFmpegProgram3$ ./main football.ts video 0
```

Fig.12.7. Instrucțiunea de utilizare a programului `CreateFiles`

3.5.2 Serverul

Implementarea serverului începe cu pregătirea structurilor de date pentru crearea unei comunicări de tip UDP. Se crează un socket de tip UDP, apoi stabilim familia de socket, umplem structura folosită de server ca să accepte orice adresă și să folosească un port utilizator și atașăm socketul prin funcția `bind`. Apoi, pentru ca utilizatorul să vadă ce filme valabile poate vizualiza, citim dintr-un fișier informații ale `.ts` dat ca argument și i le trimitem clientului. Servim clienții în mod iterativ, astfel avem o instrucțiune `while(1)`, următorul pas fiind așteptarea indexului fișierului de transmis și trimiterea dimensiunii pachetelor. Dacă am trimis datele cu succes, facem un apel la funcția `fork()` care crează un nou proces, procesul copil al acestuia, unde se execută programul **CreateFiles** cu argumentele extrase din pachetul primit de la client. Procesul inițial, părinte, așteaptă execuția procesului fiu, ca apoi să trimită un pachet de verificare că acesta s-a terminat către client, clientul rămânând blocat până când execuția programului **CreateFiles** nu

este gata. Ilustrat în Fig.12.8, cât timp sunt date de citit din fișierul dat ca input, trimite pachete de lungime specificată clientului.

```
{
    rFile = fopen(fileName, "rb");
    // Cat timp avem ce citi din fisier
    while (fread(dChunk, sizeof(char), chunkSize, rFile) != 0)
    {
        // trimitem date catre client
        if (sendto(sd, dChunk, chunkSize, 0, (struct sockaddr*) &client, length) <= 0)
        {
            perror ("[server]Eroare la sendto() catre client.\n");
            continue; // continuam sa ascultam
        }
        else
        {
            printf ("[server]S-a trimis cu succes un data chunk\n");
            usleep(3000);
        }
    }
}
```

Fig.12.8. Citirea și trimiterea datelor către client

3.5.3 Clientul

Pentru a putea implementa tehnica de **multithreading**, avem nevoie de o structură de date numită mutex care blochează resursa comună folosită de cele două fire de execuție. Clasa MutexGuard, prezentată în Fig.12.9, are ca scop blocarea resursei comune până la terminarea execuției instrucțiunii firului de execuție, pentru a putea avea acces amandouă în paralel la aceasta.

```
class MutexGuard
{
public:
    MutexGuard() { pthread_mutex_lock(&my_mutex); }
    ~MutexGuard() { pthread_mutex_unlock(&my_mutex); }
};
```

Fig.12.9. Clasa MutexGuard

Implementarea clientului începe cu deschiderea pipeului în care vor fi puse datele fișierului media, crearea socketului de tip UDP, pregătirea structurii folosită în realizarea dialogului cu serverul, apoi afișarea pe ecran utilizatorului informații despre fișierul .ts ce îl transmite serverul și alegerea unui index de film. Prin apelul funcției sendto(), trimitem la server un pachet ce conține un string cu indexul fișierului pe care dorim a fi difuzat, primim de la server

ce dimensiune vor avea pachetele primite și este urmată de instrucțiunea `recvfrom()` care este blocantă până când nu primim un răspuns de la server, mai exact așteptăm ca acesta să își termine execuția programului **CreateFiles**. Ilustrat în Fig.13.1, bucata de cod arată modul în care facem apelul `recvfrom()` să fie non-blocant, altfel clientul ar fi așteptat primirea datelor după ce serverul și-a încheiat transferul. Apoi lansăm firele de execuție și așteptăm să se termine.

```
read_timeout.tv_sec = 0;  
read_timeout.tv_usec = 10;  
errCode = setsockopt(sd, SOL_SOCKET, SO_RCVTIMEO, &read_timeout, sizeof(read_timeout));
```

Fig.13.1. Facem apelul funcției `recvfrom` să fie non-blocant

3.5.4 Utilizarea aplicației

În primă fază, lansăm serverul cu comanda specificată în Fig.13.2.a, ca apoi să ne conectăm la server prin intermediul comenzii din Fig.13.2.b, clientul intrând într-o stare blocantă până când citim din pipe, specificat în Fig.13.2.c.

```
andrei@ubuntu:~/Licenta/Server$ ./a.out 2560 football.ts  
[server]Astept la portul 8554...
```

Fig.13.2.a. Lansarea serverului

```
andrei@ubuntu:~/Licenta/Client$ ./a.out 127.0.0.1 8554
```

Fig.13.2.b. Lansarea clientului

```
andrei@ubuntu:~/Licenta/FFmpegProgram3$ ffplay /tmp/mypipe
```

Fig.13.3.b. Lansarea playerului

După ultima comandă, pe ecranul utilizatorului va apărea video-ul cu indexul 0 din fișierul `football.ts`, usecase ilustrat în Fig.13.4.

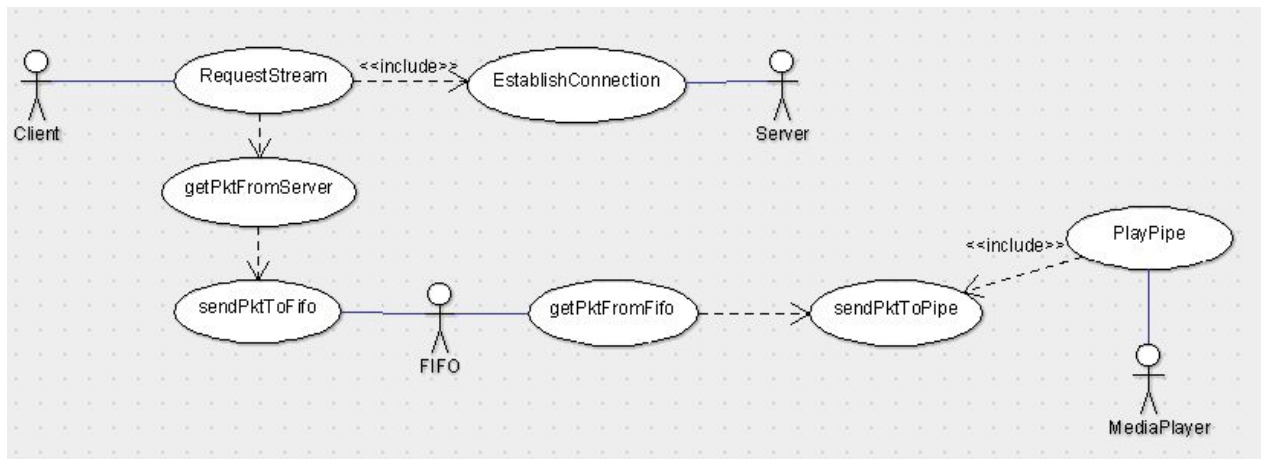


Fig.13.4. Diagrama use-case a aplicației

Capitolul 4

Studiu de caz și consumul aplicației

4.1 Atomul moov

În capitolul precedent am discutat despre acest atom și faptul că, prin manipularea antetului unui fișier putem schimba poziția din video a acestuia pentru a putea fi accesat prin intermediul unui pipe.

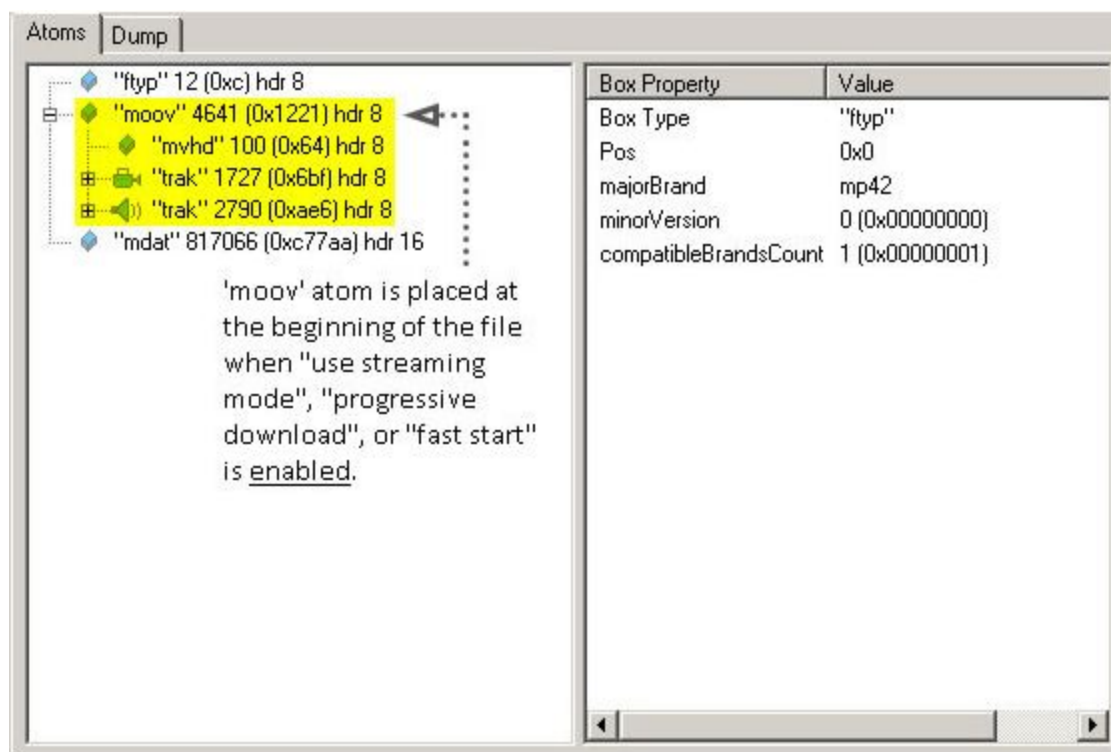
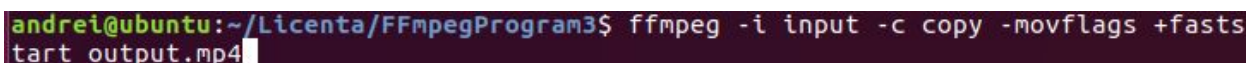


Fig.14.1. Analiza unui fișier video cu AtomicParsley

Atomul **moov** numit și atomul de film, este o unitate ce conține informații despre fișierul video: numărul cadrelor video, a eșantioanelor audio, index de capitol, titlu etc. și metadata tehnice care să îi ofere expanderului informații pentru a putea despacheta corect datele. Dacă privim containerul MPEG-4, putem observa că are o structură de arbore prin intermediul unelei **AtomicParsley** [25] sau **Mp4Creator** [26], ilustrat în Fig.14.1 [24], atomul acționând ca un

index de video pentru acesta, începerea videoului nefiind posibilă fără accesarea acestei zone de informații.

Importanța poziției atomului este dată de tipul de transfer de date pe care îl dorește utilizatorul. Într-un mediu de simplu transfer de date, în care un fișier media este transmis dintr-o parte în alta și apoi redat, informațiile pot fi la finalul fișierului, în momentul în care se face play, acestea sunt citite și puse într-un buffer la care are acces playerul media. În schimb, dacă dorim să trimitem pachete ce compun un fișier media și să le vizualizăm în momentul în care le primim, atomul de film trebuie să fie manipulat și mutat la începutul fișierului. Această tehnică se poate face prin setarea unui anumit flag, prezentat prin comanda din Fig.14.2, sau manipulat din interiorul programului **CreateFiles**, prezentat în capitolul anterior.



```
andrei@ubuntu:~/Licenta/FFmpegProgram3$ ffmpeg -i input -c copy -movflags +faststart  
tart output.mp4
```

Fig.14.2. Mutarea atomului la începutul fișierului multimedia

4.2 Consumul de memorie

Consumul de memorie este un factor crucial în stabilirea nivelului unei aplicații. Neînchiderea unei conexiuni, obiecte ce rămân încă deschise după terminarea programului, linii de cod redundante, toate duc la o scăpare de memorie ce avariază mediul de operare și consumul de memorie.

htop [27] este un program scris în limbajul C ce oferă utilizatorului o interfață prietenoasă cu funcționalitatea de a observa procesele deschise din mediul respectiv, ce memorie consumă, ilustrate descrescător în funcție de CPU utilizat. Din Fig.14.3.a se observă ce memorie consumă playerul ce citește din pipe.

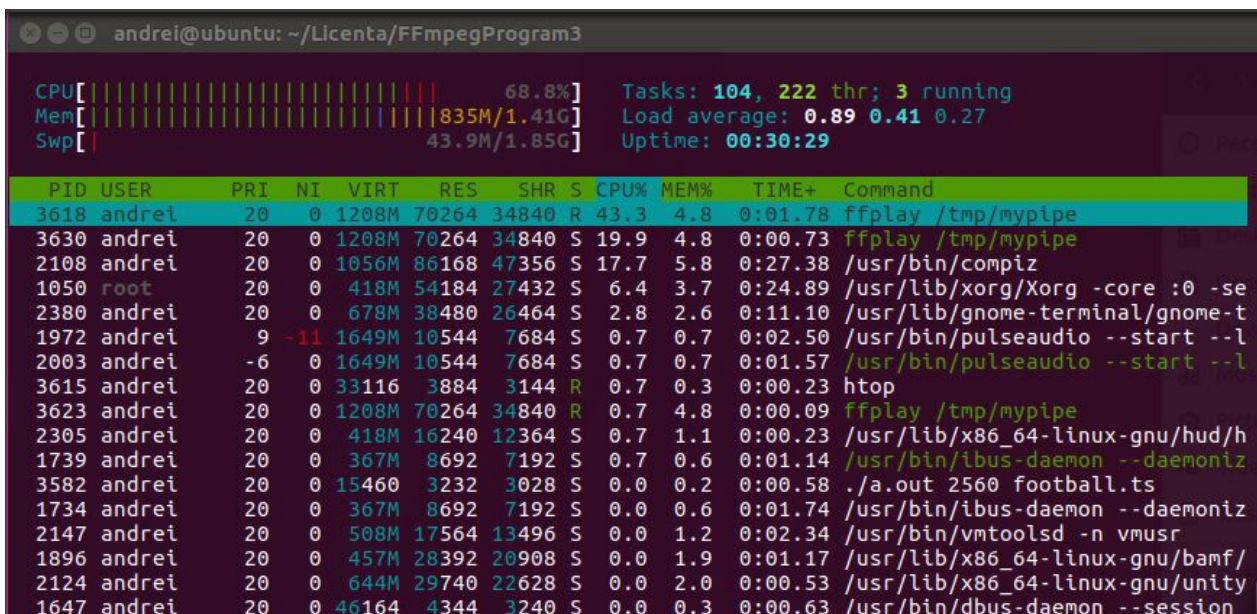


Fig.14.3.a. Consumul procesului ffplay în momentul afișării fișierului media pe ecran

Pentru a reduce consumul memorie/procesor, putem mări numărul de usleep, astfel încât după o astfel de modificare de la usleep(3000) la usleep(6000) vedem o mare îmbunătățire din punct de vedere al performanței, observabil în Fig.14.3.b.

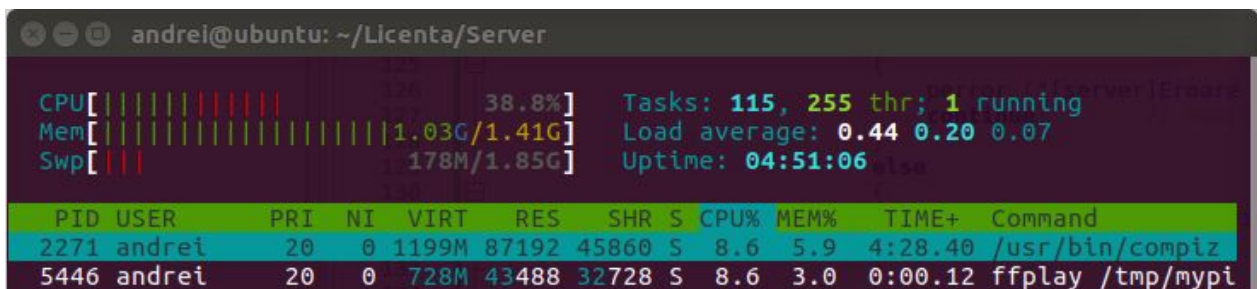


Fig.14.3.b. Consumul procesului ffplay după optimizare

Ilustrat în Fig.14.4, putem observa diferențele de consum memorie/procesor dintre o instanță server și una client.

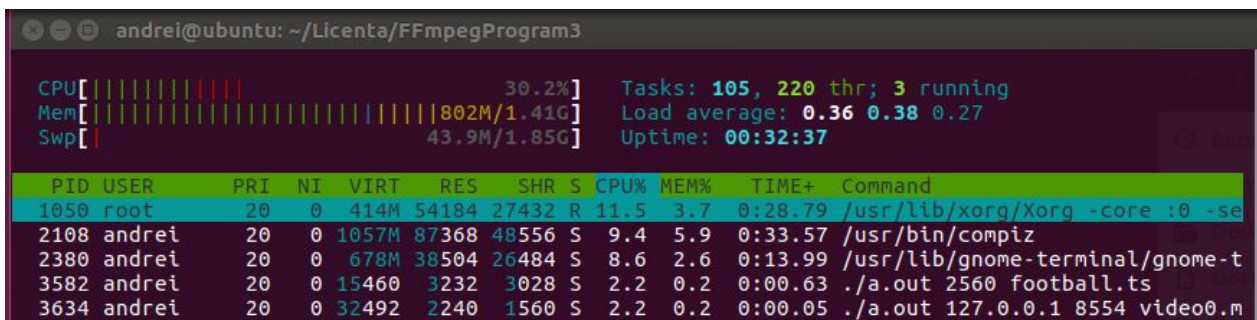


Fig.14.4. Consumul memorie/procesor a unei instanțe server și client

CONCLUZII

În urma lucrării am dobândit cunoștințe noi în manipularea mai multor fire de execuție, mi-am întărit cunoștințele legate de limbajul de programare C/C++ și am reușit să stăpânesc comunicarea prin rețea de tip UDP.

Aplicația creată atinge punctele propuse specificate în introducere, înțelegerea structurii fișierelor media, stăpânirea comunicării pe rețea, multithreading și multiprocessing, însă poate fi optimizată prin implementarea unui player media personal, iar protocolul implementat poate suferi modificări spre directiva unui protocol de tip TCP/IP pentru a avea o mai mare siguranță în transferul de date. O altă directivă de viitor ar fi implementarea unei interfațe grafice prietenoasă, în care utilizatorul interacționează cu serverul, selectând fișierele media pe care le dorește să fie transmise.

BIBLIOGRAFIE

- [1] John Watkinson The MPEG Handbook, 2004, p. 3.
- [2] Moving Pictures Experts Group website.
(<http://mpeg.chiariglione.org/>)
- [3] (<http://www1.curriculum.edu.au/digitalvideo/compression.htm>)
- [4] Compression principles Text and Image p. 2.
(https://fenix.tecnico.ulisboa.pt/downloadFile/3779571976903/licao_4.pdf)
- [5] A Guide to MPEG Fundamentals and Protocol Analysis pp. 5-6.
(http://www.img.lx.it.pt/~fp/cav/Additional_material/MPEG2_overview.pdf)
- [6] John Watkinson The MPEG Handbook, 2004, p. 250.
- [7] Digital Consumer Electronics Handbook – McGRAW-HILL BOOK COMPANY by T.Sikora p.5.
(<http://www.di.univr.it/documenti/OccorrenzaIns/matdid/matdid279434.pdf>)
- [8] Standard Codecs: Image compression to Advanced Video Coding by Mohammed Ghanbari p.202.
(<http://flylib.com/books/en/2.537.1.65/1/>)
- [9]
(<https://sites.google.com/site/sachinkagarwal/home/code-snippets/video-quality-analysis---part-1>)
- [10] Johnston, J.D., Transform coding of audio signals using perceptual noise criteria. IEEE J. Selected Areas in Comms., JSAC-6, pp. 314-323 (1988).
(<http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=E6CC0057D612B2C84B1655CECC58AF74?doi=10.1.1.422.1835&rep=rep1&type=pdf>)
- [11] B. Scharf, “Critical Bands,” Foundations of Modern Auditory Theory, J.Tobias, pp 159-202, Academic Press, New York and London, 1970.
- [12] J.D. Johnston, “Estimation of Perceptual Entropy Using Noise Masking Criteria,” Proc. Of the Int. Conf. IEEE ASSP, pp 2524-2527, 1988.
- [13] (<https://www.slideshare.net/BrewsterXO/mpegaudio-compression>)
- [14] A Guide to MPEG Fundamentals and Protocol Analysis p. 10.
(http://www.img.lx.it.pt/~fp/cav/Additional_material/MPEG2_overview.pdf)

- [15] Wiese, D., MUSICAM: flexible bitrate reduction, standard for high quality audio. Presented at Digital Audio Broadcasting Conference (London, March 1992).
- [16] G.Plenge, "A Versatile High Quality Radio System for the Future Embracing Audio and Other Applications," 94th AES-Convention, Berlin 1994.
- [17] John Watkinson The MPEG Handbook, 2004, pp. 202-208.
- [18] (<https://www.slideshare.net/devashishraval/avs-chapter-3>)
- [19] (<http://happy.emu.id.au/lab/tut/dttb/dtbtut4b.htm>)
- [20] <https://metabroadcast.com/blog/stream-processing-is-hard>
- [21] <https://ffmpeg.org/ffmpeg.html>
- [22] https://en.wikibooks.org/wiki/Communication_Networks/TCP_and_UDP_Protocols
- [23] [https://en.wikipedia.org/wiki/Pipeline_\(Unix\)](https://en.wikipedia.org/wiki/Pipeline_(Unix))
- [24] http://www.adobe.com/devnet/video/articles/mp4_movie_atom.html
- [25] <http://atomicparsley.sourceforge.net/>
- [26] <http://mp4creator.sourceforge.net/>
- [27] <http://hisham.hm/htop/>
- [28] <http://www.dvbstreamexplorer.com/>
- [29] <http://www.dvbstreamexplorer.com/dvbse/dvbse.php>
- [30] https://www.mp3-tech.org/programmer/docs/trev_283-popp.pdf