# Coltan Cristian – C113D – TEMA\_2

Pentru ca aplicatia “**File Server**” sa fie realista, am creeat 2 directoare separate, **Server**, respectiv **Client**. De citit, **Model flux testat**, **Detalii implementare**, **Bug**.

**Continut arhiva:**

Folder-ul **Server**: Structura de fisiere pentru testare, incluzand si directoare; header.h; log.txt; server.c (main); utils.c; Makefile.

Folder-ul **Client**: client.c (main); header.h; utils.c; Makefile; upload\_test.txt.

**Conexiune client-server:**

Se realizeaza prin intermediul socket-urilor, am folosit **setsockopt** pentru a putea refolosi socket-ul si a evita asteptarea si erorile cu bind fail. Am configurat socket-ul, iar pe o bucla while infinita asteptam conectarea clientilor pe care ii tratam folosind un thread pentru fiecare in parte.

**Bug:**

Aplicatia functioneaza cum trebuie dar, atunci cand se incepe **Graceful termination**,apare accept failed deoarece inainte de a inchide programul complet, server ul mai asteapta inca o data in accept() un client sa se conecteze.

**Detalii de implementare:**

In primul rand am folosit o structura de fisiere, cum a fost recomandat in cerinte:

typedef struct

{

    char file\_name[30];

    int name\_dim;

    int content\_dim;

    int read;

    int write;

    wordFreq top[10];

    pthread\_mutex\_t f\_mutex;

    pthread\_cond\_t f\_cond;

} fileStats;

Am gandit aplicatia in felul urmator: Clientul se conecteaza, Acesta trimite o comanda prin intermediul functiei **send\_operation()**, server-ul primeste comanda, o parseaza, si trimite un raspuns **send\_response()**, client-ul primeste raspunsul, il parseaza la randul lui si afiseaza la stdout operatia, status-ul si un mic mesaj pentru a intelege usor ce s-a intamplat. Pentru a usura domnului profesor debugging-ul si urmarirea fluxului am printat si string-urile ce se trimit prin **send()** si **recv()** intre client si server.

Se creeaza thread-ul de indexare caruia ii trimitem ca argument o structura cu numarul de fisiere si fisierele, acesta indexeaza fisierele la inceput, apoi asteapta intr-o bucla while trimiterea semnalului pentru a se trezi.

while (!need\_update)

Functia **trigger\_reindexing()**, seteaza need\_update pe 1 si trimite **pthread\_cond\_signal()**, in cele din urma reindexand fisierele. Am printat din nou in stdout pentru a usura urmarirea fluxului. Check :

\*index\_thread\_function

Se creeaza thread-ul care asteapta la stdin “quit” si un semnal, utilizand o masca de semnale **setup\_signal\_fd()**. Totul este pus intr-un **epoll** pentru a face posibila asteptarea concurenta pe mai multi descriptori de fisier. Check:

\*listener\_thread\_function

Deoarece au fost create thread-uri speciale pentru indexare si ascultare, pe thread-ul principal (cel din main) a fost realizata parsarea clientilor folosind cate un thread pentru fiecare operatie.

Atunci cand se incepe **graceful termination** se asteapta ca toate thread-urile sa isi finalizeze executia.

String-urile trimise pentru realizarea operatiilor sunt delimitate prin **|** iar fisierele prin **\0**.

Functia **update\_log()** este **thread-safe** si printeaza in fisierul **log.txt** data si comenzile care au fost primite.

**LIST:** Clientul trimite: *LIST*, iar server-ul cauta recursiv in directoare fiserele de tip **DT\_REG** si trimite inapoi raspuns-ul.

**DOWNLOAD:** Clientul trimite (ex) DOWNLOAD 15 download\_test.txt, client-ul descarca in **just\_downloaded.txt** continutul. Serverul trimite Succes sau Fail daca fisierul care este cerut nu exista.

**UPLOAD:** Clientul trimite (ex) UPLOAD 15 upload\_test.txt. Server-ul trimite Succes sau Fail daca fisierul exista deja (Permission denied).

**DELETE:** Clientul trimite (ex) DELETE 15 upload\_test.txt. Server-ul trimite Succes sau Fail daca fisierul nu exista

**MOVE:** Clientul trimite (ex) MOVE 17 fisiere2/move.txt 16 fisiere/move.txt. Server-ul trimite Succes sau Fail daca un fisier cu acelasi nume exista deja in noul path.

**UPDATE:** Clientul trimite (ex) UPDATE 15 fisiere2/f4.txt 4 3 Ana. Server-ul trimite Succes sau Fail daca fisierul nu exista.

**SEARCH:** Clientul trimite (ex) SEARCH 3 Ana. Server-ul trimite Succes (lista de fisiere sau lista goala).

Server-ul trimite doar codul, iar clientul stie sa interpreteze raspunsul si afiseaza daca a fost operatie efectuata cu succes sau eroare + o mica explicatie pentru a intelege.

Daca operatia nu exista server-ul va trimite 0x10 - unknown operation, iar daca numarul de clienti maxim a fost atins va trimite 0x8 - server busy (numarul maxim de clienti a fost atins).

**Flux testat:**

*Fri Jan 12 12:57:48 2024 0x2 - UPLOAD [upload\_test.txt]*

*Fri Jan 12 12:58:06 2024 0x4 - DELETE [upload\_test.txt]*

*Fri Jan 12 12:58:30 2024 0x20 - SEARCH [Ada]*

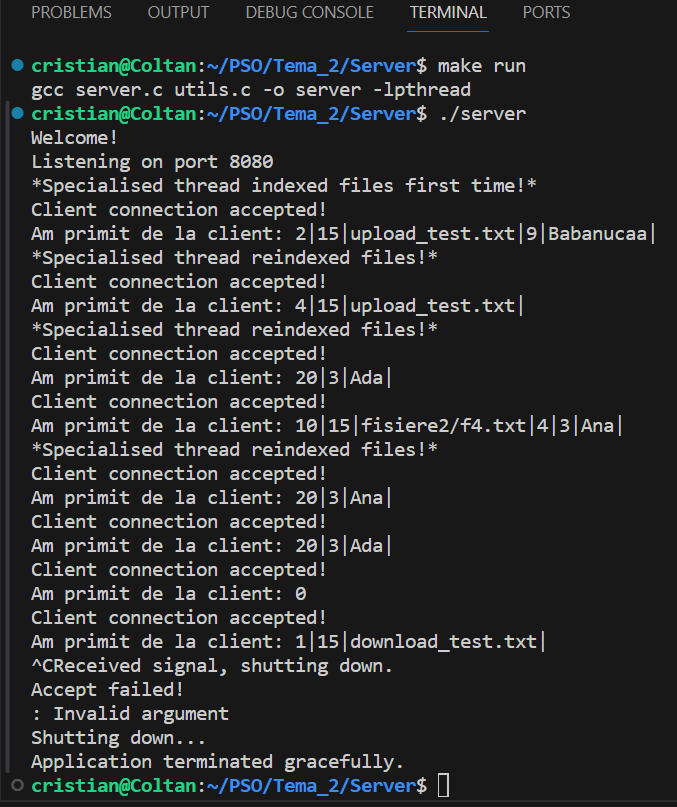
*Fri Jan 12 13:00:24 2024 0x10 - UPDATE [fisiere2/f4.txt]*

*Fri Jan 12 13:00:44 2024 0x20 - SEARCH [Ana]*

*Fri Jan 12 13:00:52 2024 0x20 - SEARCH [Ada]*

*Fri Jan 12 13:01:44 2024 0x0 - LIST*

Am dat **upload** la **upload\_test.txt** si apoi am dat **delete** la acelasi fisier, am dat **search** “Ada” (stiind ca exista o singura aparitie a cuvantului) am primit calea fisierului, apoi, am dat **update** si am schimbat “Ada” cu “Ana”, am dat **search** “Ana” si **search** “Ada”, unde am primit o lista goala.

****

