Tema 3 - File Server

Deadline: ~~20.01.2023~~ **29.01.2023**, ora **23:55**
NU AVEȚI VOIE SĂ FOLOSIȚI APELUL system() din libc

Tema va fi implementată pentru un sistem bazat pe Linux **sau** pentru Windows. Se va lua în considerare o singură implementare. Nu implementați pentru ambele sisteme.

Implementați o aplicație de tip “File Server”. Aplicația se va comporta ca un server ce expune printr-un socket accesul la un set de fișiere. Fișierele vor putea fi identificate printr-un path(ex: „/file1“, „/dir1/file2“). Serverul va expune următoarele operații către client:

1. **LIST** - Clientul va primi o listă cu fișierele disponibile pe server.
   - IN: operația
   - OUT: status; număr octeți răspuns; lista de fișiere, separate prin ‘\0’
2. **GET** - Clientul va putea descărca un fișier pe baza numelui.
   - IN: operația; nr. octeți nume fișier; numele fișierului
   - OUT: status; număr octeți răspuns; conținut fișier
3. **PUT** - Clientul va putea încărca un nou fișier
   - IN: operația; nr. octeți nume fișier; numele fișierului; nr. octeți conținut; conținut fișier
   - OUT: status
4. **DELETE** - Clientul va putea șterge un fișier existent
   - IN: operația; nr. octeți nume fișier; numele fișierului
   - OUT: status
5. **UPDATE** - Clientul va putea schimba conținutul unui fișier.
   - IN: operația; nr. octeți nume fișier; numele fișierului; octet start; dimensiune; caracterele noi
   - OUT: status
6. **SEARCH** - Clientul va putea căuta un cuvânt în toate fișierele expuse de server și va primi o listă cu numele fișierelor ce conțin acea secvență în conținutul lor
   - IN: operația; nr. octeți cuvânt; cuvânt
   - OUT: status; listă de fișiere separate prin ‘\0’

Detalii de implementare

Pentru căutarea mai rapidă a cuvintelor în fișiere va exista o funcționalitate de indexare a fișierelor. Se va calcula și se vor reține în memorie cu ajutorul unei structuri de date cele mai frecvente 10 cuvinte din fiecare fișier. Crearea și actualizarea acestei liste o va face un thread specializat care va exista pe tot parcursul execuției aplicației. Dacă nu se găsește în listă cuvântul, se va returna o lista goală.

Inițial acest thread va crea lista amintită mai sus la pornirea aplicației, și va intra într-o stare de așteptare. Thread-ul se va “trezi” și va actualiza lista în urma unei operații de PUT, DELETE sau UPDATE.

Pentru încheierea execuției aplicației, se va crea un handler de semnal pentru SIGTERM și Ctrl+C. La primirea acestui semnal, aplicația va începe un “graceful termination”, nu va mai accepta conexiuni

noi, va finaliza de servit conexiunile curente și va implementa un mecanism prin care va opri threadul de indexare într-un mod cât mai “curat”.

Serverul va fi de tip multi-thread, va avea un thread care ascultă pentru conexiuni noi și va crea câte un thread nou pentru fiecare nouă conexiune inițiată de un client. Tratarea cererii se va face astfel pe un thread separat pentru fiecare client. Socket-ul clientului și detaliile despre operație(ex. numele fișierului) vor fi salvate global, la nivel de thread (vezi Thread-Local Storage).

Pe timpul rulării aplicației, se va reține o listă cu fișierele disponibile “pe server”, listă care se va actualiza în urma operațiilor de DELETE, UPLOAD, PUT.

Pentru partea de comunicație client-server va fi nevoie să proiectați un protocol de comunicație. Acesta va trebui să suporte operațiile definite mai sus precum și să aibă următorul flux de lucru:

- Clientul se conectează la server
- Clientul trimite tipul de operație și parametrii
- Serverul execută operația
- Dacă operația s-a executat cu succes, se primește un cod de succes și la nevoie rezultatul operației(ex. lista de fișiere)
- Dacă operația a eșuat din diverse motive, se primește un cod de eroare și un mesaj de eroare.

Serverul va scrie toate operațiile efectuate într-un fișier de log disponibil global la nivelul aplicației. Veți implementa o funcție “thread-safe” care va scrie în acest fișier. Fiecare operație se va loga pe o linie separată și va avea următoarea formă:

- **Data, ora, tip operație[, nume fișier afectat][, cuvânt căutat]**

De avut în vedere:

- Aveți grijă cum codificați mesajele. Dacă aveți lungimi variabile, va trebui întâi să trimiteți dimensiunea(N) și apoi N octeți de date.
- Dacă simultan se primesc de la mai mulți clienți operații pe același fișier, s-ar putea să apară inconsistențe(ex. un client să citească în timp ce altul scrie). Trebuie să găsiți o metodă de a mitiga acest risc. O posibilă soluție ar fi reținerea unei liste cu fișierele cu care se lucrează și tipul de operație(read sau write). Vom avea următoarele cazuri:
  - Unul sau mai multe thread-uri vor să citească simultan - OK
  - Unul sau mai multe thread-uri citesc și vine o operație de scriere - se așteaptă finalizarea citirii, apoi se face scrierea
  - Un thread execută o operație de scriere, și apare o operație de citire sau scriere - se va aștepta încheierea operației de scriere, apoi se va executa noua operație
- Pentru trimiterea fișierelor pe rețea se va folosi apelul de tip zero-copy sendfile() pe Linux sau funcția TransmitFile() pe Windows.

Q&A

- **Q**: Se poate folosi C++ pentru implementare?
- **A**: Nu

- **Q**: Cum ar trebui implementat protocolul de comunicație?
- **A**: Este la alegerea voastră.

- **Q**: Putem să folosim structuri de date neimplementate de noi (luate de pe net sau din alte laboratoare, etc.) în rezolvarea temei?
- **A**: Da, cu mențiunea ca implementarea acestora **să fie în fișiere separate**, și să specificați în README fișierele în cauză și sursa.

- **Q**: Există schelet pentru temă?
- **A**: Nu. Implementarea se face de la zero.

- **Q**: Se poate folosi apelul system() ?
- **A**: Nu.