Sistem client-server pentru criptare de conținut binar

Realizați o aplicație **client-server** (server-ul și client-ul sunt programe diferite), care oferă următoarele funcționalități:

- Serverul inițializează un obiect de tip NamedPipe, prin care așteaptă conexiuni de la clienți.
- Fiecare client care dorește să se conecteze la server, trimite mai întâi un pachet de inițializare a conexiunii la care serverul răspunde dacă este acceptată conexiunea sau nu.
- Dacă este acceptă conexiunea, clientul trimite un pachet de autentificare cu username și parolă.
    - Server-ul răspunde cu un pachet prin care comunică clientului dacă autentificarea a fost cu succes sau dacă autentificarea a eșuat, caz în care se închide conexiunea.
- După autentificare, clientul va trimite server-ului, prin mai multe pachete, maxim câte 4KB de date pentru a fi criptate de server.
- Fiecare pachet este criptat de către server și trimis înapoi la client.
- La final, clientul va anunța închiderea conexiunii printr-un pachet corespunzător
    - Server-ul va închide și capătul lui de conexiune.
- Opțional: Server-ul va trata cazul în care client-ul devine inactiv fără să anunțe închiderea conexiunii cu o valoare de timeout per conexiune.

Observații

Trebuie să proiectați propriului protocol de comunicare între client-server (ex. Structura unui pachet, tipuri de comenzi, specificarea dimensiunii pachetelor, etc.)

Tabela cu utilizatori se poate ține static, în memorie, sau citi dintr-un fișier.

Criptarea se va realiza cu un cifru XOR simetric - deci criptarea și decriptarea se efectuează prin aceeași operație.

Cheia de criptare poate fi de lungime mai mică decât mesajul în clar, și atunci cheia se va repeta de atâtea ori cât este nevoie pentru a acoperi lungimea mesajului.

Exemplu: mesajul de criptat: IamHere

cheie de criptare: Key

operatia de criptare: IamHere (xor) KeyKeyK

Client

Parametri (din linia de comandă):

- PipeName - numele Pipe-ului pe care clinetul ascultă server-ul (poate fi implicit/explicit, cu default=hardcoded)
- FilePath - calea către fișierul de criptat
- OutputPath - calea către fișierul unde se scrie conținutul criptat (varianta implicită =<FilePath>.enc)
- Username - username
- Password - parolă
- Key - cheia de criptare pentru sesiunea respectivă (varianta implicită =<Password>)
- Orice alți parametri adiționali de care aveți nevoie

Funcționalitate:

- Client-ul se conectează la Pipe-ul creat de server.
- Se autentifică (username, parolă).
- Dacă autentificarea are succes, stabilește o sesiune de criptare (stabilește cheia) și trimite conținutul binar din <FilePath> în pachete.
- Așteaptă să primească de la server înapoi pachete cu conținutul criptat.
- Scrie conținutul criptat în <OutptuPath>.
- Anunță și apoi efectuează închiderea conexiunii.

Server

Server-ul va iniția Pipe-ul pe care va aștepta conexiuni de la clienți.

Versiunea 1

Parametri (din linia de comandă):

- PipeName - numele pipe-ului (poate fi implicit/explicit, cu default=hardcoded)
- LogFile - opțional: fișier în care să scrie mesaje de log
- Orice alți parametri adiționali de care aveți nevoie

Funcționalitate:

- Server-ul creează Pipe-ul.
- În thread-ul principal, primește conexiuni de la clienți și îi servește, câte unul pe rând.
- Logging (Opțional) - Afișează pe ecran și în <LogFile> mesaje de informare la momentele importante ale rpotocolului de comunicare, pentru a putea depana cu ușurință eventualele probleme.
- Opțional, server-ul va avea și o metodă de a își termina execuția în mod controlat, pentru a nu întrerupe serverul în timp ce deservește un client - ex. poate verifica existența unui fișier <StopFlagFile> înainte de a primi noi conexiuni.

Versiunea 2

Parametri adiționali:

- NrClients - Număr maxim de clienți concurenți

Funcționalitate:

- Server-ul creează Pipe-ul și primește conexiuni noi pe thread-ul principal.
- După o autentificare cu succes a unui client, creează un "Client Thread" care deservește mai departe, în mod concurent, clientul tocmai autentificat.
- Numărul de clienți deserviți la un moment dat este cel mult <NoClients>.
    - Discuții: vom dicuta diferite abordări pentru implementarea acestui mecanism.
- Opțional, server-ul va avea și o metodă de a își termina execuția în mod controlat, pentru a nu întrerupe serverul în timp ce deservește un client - ex. poate verifica existența unui fișier <StopFlagFile> înainte de a noi conexiuni, așteptând să se termine cele deja stabilite.

Versiunea 3

Parametri adiționali:

- NrWorkers - Număr de worker threads.

Funcționalitate:

- Server-ul inițializează <NoWorkers> thread-uri, aflate la început în așteptare, și o structură sincronizată de tip coadă.
- Server-ul creează Pipe-ul și primește conexiuni noi pe thread-ul principal.
- După o autentificare cu succes, stabilește conexiunea și pornește un thread client care deservește clientul conectat.
- Numărul de clienți deserviți la un moment dat să nu depășească <NoClients>.
- Thread-ul clientului va citi pachetele și le va insera în coada de lucru, apoi va aștepta să fie semnalat.
- Thread-urile de tip "worker" vor lua pachetele din coadă și le vor cripta în mod concurent, fiecare semnalând thread-ul client care le-a pus în coadă, după ce termină.
- Când e semnalat, thread-ul client trimite pachetul înapoi la client. Acesta trebuie să asigure ordinea pachetelor.

Opțional, server-ul va ține un thread care comunică cu consola și va primi comenzi de la "administratorul" server-ului:

> list

Va afișa informații despre clienți: nume, status (on/off), număr de bytes criptați.

> exit

Va opri controlat server-ul, așteptând să termine toate thread-urile Worker și Client.

în această variantă, log-area, dacă există, se va face doar în fișierul <LogFile>, nu și pe ecran.