Documentație W.O.N.

Adrian Petercă

Universitatea "Alexandru Ioan Cuza", Iași, secretariat@info.uaic.ro
https://www.info.uaic.ro

Abstract. Acest document contine detalii despre implementarea si functionalitatea proiectului W.O.N. In sectiunile urmatoare se gasesc toate comenzile posibile, modul de evaluare al mesajelor primite si tehnologiile utilizate.

Keywords: Retea sociala · Comunicarea server-client · TCP/IP

1 Introducere

Aplicatia W.O.N. (acronim pentru **W**orld **O**nline **N**etwork) este inspirata din tema proiectului *VirtualSoc*, aparitinand cursului Relete de Calculatoare[1]. Proiectul se axeaza pe dezvoltarea unui prototip al unei retele de socializare moderne, unde un utilizator se poate inregistra, poate trimite mesaje altor utilizatori sau poate citi postari apartinand altor utilizatori, totul prin intermediul unei interfete text.

2 Tehnologii

Ca tehnologii utilizate, aplicatia s-a folosit doar de protocolul de TCP/IP, fiind un aspect crucial ca mesajele transmise atat de partea de server, cat si de partea de client, sa ajunga intacte si in ordine. Astfel, s-a prioritizat corectitudinea pachetelor trimise in detrimentul memoriei utilizate si al timpului de executie.

In alte cuvinte, tehnologia pachetelor UDP (acronim pentru **U**ser **D**atagram **P**rotocol) este *neorietata conexiune*[1], insemnand ca nu realizeaza verificari de transmitere corecta si integrala a pachetelor, fapt care, in aplicatia noastra, ar putea duce la mesaje incorect transmise sau chiar pierdute pe traseu.

Pe de alta parte, tehnologia TCP (acronim pentru **T**ransfer **C**ontrol **P**rotocol) este *orientata conexiune*[1], astfel ca vine cu avantaje seminificative pentru aplicatie. Ca prima idee, pachetele sunt trimise integral si in ordinea dorita, facandu-se o verificare de primire a acestora intre cel care trimite si cel care primeste. In ciuda faptului ca aceste verificari pot avea un impact asupra timpului de executie, este de preferat asteptarea unui pachet corect si complet (cu care se poate lucra mai departe) decat a unui pachet transmis partial.

3 Arhitectura

3.1 Diagrama de utilizare

Aplicatia contine doua componente care comunica intre ele: clientul si serverul. Pentru a fi eficient, serverul a fost implementat intr-o maniera concurenta, utilizand apeluri ale functiei `fork()` pentru a crea procese-copil care vor servi cererile clientului.

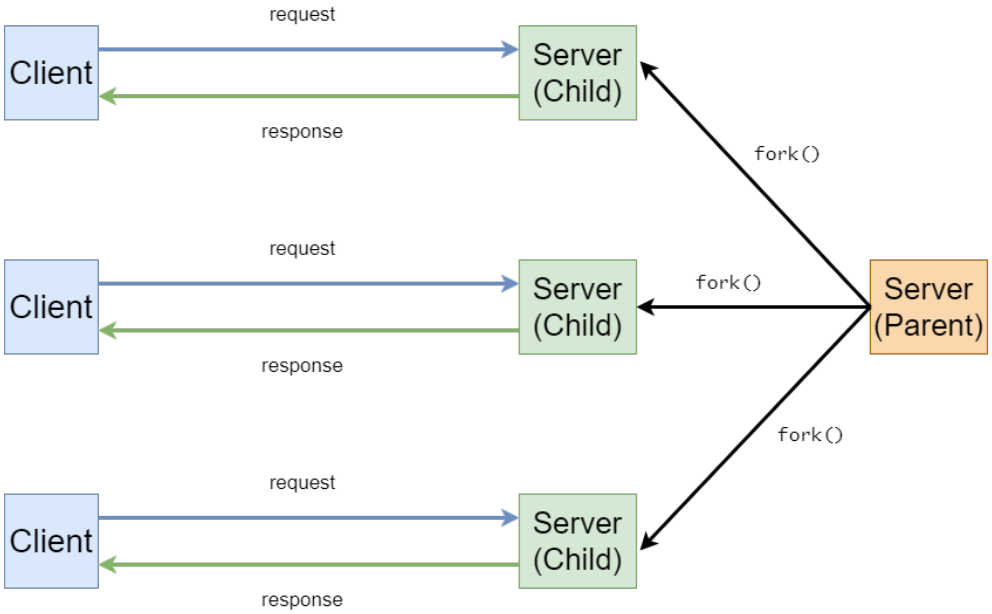

Fig. 1. Metoda de functionare a aplicatiei

3.2 Structuri de date

Ca structuri de date utilizate, aplicatia foloseste urmatoarele definitii:

– o structura de date numita `PostData` care retine informatii specifice unei postari (de catre cine a fost postata, catre ce grup se adreseaza, data postarii, textul acesteia)

```
struct PostData {
    char owner[11];
    int group;
    time_t date;
    char txt[500];
}
```

– o structura de date specifica mesajelor, numita `MessageData`, care retine numele utilizatorului ce a trimis mesajul, destinatarul, data la care a fost trimis si textul efectiv.

```
struct MessageData {
    char from[11];
    char to[11];
    time_t date;
    char txt[200];
}
```

– o structura de date pentru schimbarea Master Key-ului, ce contine atat noua cheie, cat si pe cea veche.

```
struct MasterKeyData {
    int oldKey;
    int newKey;
}
```

3.3 Arhitectura bazelor de date

Aplicatia foloseste, ca baze de date, fisiere binare. Motivul pentru alegerea lor consta in dorinta de a oferi un plus de securitate: prin stocarea lor in fisiere de format `.json` sau `.xml`, informatiile ar putea fi modificate relativ usor, lucru care trebuie evitat. Prin stocarea lor in format binar, modificarea informatiilor in mod direct se complica, insa ramane la aceeasi dificultate la nivel de cod.

Asadar, aplicatia stocheaza informatii dupa cum urmeaza: serverul are acces la doua baze de date fundamentale, numite **loginadmin** si **loginuser**. Fiecare dintre acestea retine, la inceputul fisierului, un numar de tip `int` care arata cati utilizatori sunt scrisi in fisierul respectiv, urmand ca, in continuare, fiecare utilizator sa fie inregistrat conform schemei din Fig. 2.

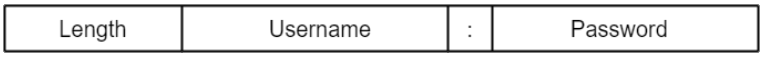

Fig. 2. Reprezentarea informatiei unui utilizator in baza de date

In momentul inregistrarii unui nou utilizator, aplicatia creeaza automat 4 fisiere noi in subdirectorul `.\users\`:

– `friends_USER_family` → acesta reprezinta lista de utilizatori apartinand grupului **Family** al utilizatorului **USER**. Ei sunt stocati conform Fig. 3
– `friends_USER_friends` → acesta reprezinta lista de utilizatori apartinand grupului **Friends** al utilizatorului **USER**. Ei sunt stocati conform Fig. 3
– `feed_USER` → in acest fisier se stocheaza posturile care nu sunt publice, dar sunt adresate si utilizatorului **USER** (acesta se afla intr-un grup al unui alt utilizator care a facut o postare). Stocarea se face conform Fig. 4.

– `inbox_USER` → in acest fisier se vor stoca mesajele primite de la alti utilizatori. Stocarea se face conform Fig. 5.

Fig. 3. Reprezentarea grupurilor in baza de date

Fig. 4. Reprezentarea postarilor in baza de date

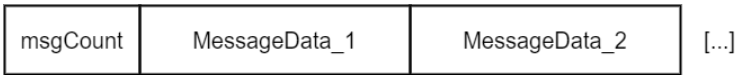

Fig. 5. Reprezentarea mesajelor in baza de date

De asemenea, serverul foloseste un fisier cu numele `posts`, unde stocheaza (dupa schema prezenta in Fig. 4) postarile publice, accesibile oricarui utilizator (indiferent daca este sau nu conectat).

4 Detalii de implementare

In cele ce urmeaza va fi descris protocolul folosit in cadrul aplicatiei, mai exact in ce format sunt trimise mesajele de catre client si ce raspuns va primi acesta de la server.

Situatii posibile:

– Clientul s-a conectat anonim la server.
  Aceasta are urmatoarele comenzi posibile:
  - `help` → va fi afisata o lista cu aceste comenzi (nu se comunica cu serverul, deoarece lista de comenzi posibile poate fi dedusa local)
  - `register` → dupa un proces de inregistrare, se va construi un mesaj de forma `register:TYPE:USERNAME:PASSWORD`, unde `TYPE` poate fi `admin` sau `user`. Ca raspunsuri posibile, clientul poate primi:
    * `register:success` → inregistrarea s-a efectuat cu succes
    * `register:failed` → inregistrarea nu s-a efectuat (exista deja un utilizator cu acest nume)
    * `register:error` → inregistrarea nu s-a efectuat datorita unei erori pe partea de server

- `login` → clientul va avea o interfata de logare. Dupa citirea numelui si a parolei oferite, se va construi un mesaj de forma `login:USERNAME:PASSWORD`, care va fi trimis catre server. Ca raspuns, clientul va primit:
  - `login:success` → logarea s-a efectuat cu succes. Din acest moment, vor exista alte comenzi posibile.
  - `login:failed` → numele utilizatorului sau parola sunt gresite.
  - `login:error` → a aparut o eroare din partea serverului
- `surf` → este metoda prin care un utilizator fara cont poate vedea postarile publice ale altor utilizatori. Catre server se va trimite o cerere de forma `surf:nologin`, care va returna unul din raspunsurile:
  - `surf:success` → catre client se va intoarce o postare publica printr-o structura de date `PostData`, iar aceasta va fi afisata.
  - `surf:error` → a intervenit o eroare pe partea de server.
    Deoarece in baza de date specifica postarilor publice exista mereu o postare initiala (de test), nu exista necesitatea tratarii cazului in care nu ar exista postari publice.
- `quit` → aceasta va termina conexiunea cu serverul si va inchide sesiunea.

– Clientul s-a conectat ca **USER** la server.
Atunci, acesta are acces la urmatoarele comenzi:
- `help` → la fel ca in cazul initial, va aparea pe ecran lista de comenzi posibile.
- `surf` → se va construi un mesaj de forma `surf:USERNAME` care va fi trimis catre server. Ca raspuns, clientul va putea primi:
  - `surf:success` → acesta va primi de la server o postare (fie publica, fie specifica unui grup al unui utilizator in care se afla cel care a facut cererea) prin structura de date specifica.
  - `surf:error` → a aparut o eroare pe partea de server, iar postarea nu a fost trimisa.
- `add USER GROUP` prin aceasta comanda, utilizatorul va putea adauga un alt utilizator la un anumit grup. Astfel, se construieste un mesaj de forma `add:ACTIVE_USER:USERNAME:GROUP` care va fi trimis catre server. Acesta din urma va raspunde prin mesajele:
  - `add:success` → adaugarea utilizatorului la grupul specificat s-a realizat cu success.
  - `add:failed` → utilizatorul specificat nu exista
  - `add:error` → o eroare pe partea de server a impiedicat adaugarea utilizatorului.
- `post` → prin aceasta comanda, utilizatorul va putea crea o postare. Acesta va trebui sa specifice un tip al postarii (publica sau pentru un anumit grup), urmand sa apara un mesaj care va cere textul postarii. Pentru server, se vor construi doua mesaje: primul va fi cererea, care va avea forma `post`, urmata de o structura de date de tip `PostData`, ambele fiind trimise catre server. Ca raspuns, clientul va primi unul din urmatoarele mesaje:

  - `post:success` → postarea a fost adaugata in baze de date cu success.
  - `post:error` → a intervenit o eroare pe partea de server.
- `message USERNAME [...]` → folosind aceasta comanda, utilizatorul poate trimite un mesaj privat utilizatorului `USERNAME`. Astfel, se vor construi doua mesaje ce vor fi trimise catre server: o cerere de tip `message:USERNAME`, pentru a putea serverul identifica cui i se trimite mesajul, urmat de o structura de date `MessageData` care contine mesajul efectiv. Clientul va putea primi urmatoarele mesaje:
  - `message:USERNAME:success` → mesajul a fost receptionat si trimis cu success.
  - `message:USERNAME:failed` → mesajul a fost receptionat, insa utilizatorul respectiv nu exista.
  - `message:USERNAME:error` → pe partea de server a intervenit o eroare.

  In situatia in care sunt specificati mai multi utilizatori, mesajul va fi trimis fiecaruia, pe rand, iar raspunsurile primite de server vor fi evaluate inainte de a trimite catre urmatorul utilizator mesajul respectiv.
- `read` → comanda folosita pentru a citi mesajele primite. Aceasta va construi o cerere de forma `read:ACTIVE_USERNAME` si o va trimite catre server, iar acesta se va folosi de numele utilizatorului primit pentru a intoarce un mesaj. Posibilele raspunsuri sunt:
  - `read:success` → acest raspuns este urmat de o structura de date de tip `MessageData` trimisa de catre server, urmand a fi afisata pe ecran.
  - `read:empty` → nu mai exista mesaje noi (necitite)
  - `read:error` → a aparut o eroare pe partea de server, iar mesajul nu a fost trimis inapoi catre client.
- `quit` → folosind aceasta comanda, clientul poate inchide sesiunea, neasteptand un mesaj de la server. Pe partea de server se va termina executia procesului copil asignat acestui utilizator.

– Clientul s-a conectat la server ca **ADMIN**.
Pe langa comenzile enumerate anterior, un utilizator cu cont de administrator are acces si la urmatoarele comenzi speciale:

- `masterkey` → aceasta comanda permite utilizatorului sa schimbe Master Key-ul necesar in cazul inregistrarii unui admin. Dupa citirea noii chei, se vor trimite doua mesaje catre server: unul de forma `masterkey`, urmat de o structura de date `MasterKeyData`, care contine atat valoarea precedenta, cat si valoarea noua. Ca raspuns, clientul va putea primi:
  - `masterkey:success` → schimbarea s-a efectuat cu succes.
  - `masterkey:failed` → schimbarea nu s-a efectuat vechea cheie transmisa nu corespunde cu cheia curenta de pe server.
  - `masterkey:error` → datorita unei erori pe partea de server, schimbarea a esuat.

5 Concluzii

In acest moment, aplicatia functioneaza corespunzator, comenzile si mesajele aferente sunt parsate corect, insa exista anumite puncte care pot fi imbunatatite. Pentru inceput, securitatea mesajelor transmise nu a fost prioritara, astfel incat acestea sunt trimise in format obisnuit (necriptate), acest aspect fiind un risc in plus pentru utilizatorul obisnuit.

De asemenea, functionalitatea unui administrator este destul de limitata. Pe viitor, ar trebui adaugate mai multe comenzi importante, precum `shutdown` (care permite inchiderea serverului de la distanta), `delete POST_ID` (fiecare postare sa fie identificata printr-un ID unic, astfel incat ele sa poata fi moderate), `remove USER` sau `suspend USER`.

Pe de alta parte, reteaua permite doar vizionarea unor postari si citirea de mesaje transmise. O idee pentru a mentine utilizatorul pe platforma ar consta in adaugarea unor jocuri accesibile prin comenzi, precum `tictactoe` care va incepe o sesiune de TicTacToe cu serverul sau cu un anumit utilizator, ori `connect4` care va instantia o sesiune de Connect4.

Totodata, interfata text vine cu limitarile ei specifice. Asadar, ideea unui GUI atragator (cel putin pe partea de client) reprezinta un punct de plecare concret pentru actualizari viitoare.

References

1. Retele de Calculatoare - ultima accesare: 15 Decembrie 2020
   `https://profs.info.uaic.ro/~computernetworks/`
2. StackOverflow - ultima accesare: 10 Decembrie 2020
   `https://stackoverflow.com`
3. Beej's Guide to Network Programming Using Internet Sockets - ultima accesare: 10 Decembrie 2020
   `https://www.gta.ufrj.br/ensino/eel878/sockets/sockaddr_inman.html`
4. The Open Group - ultima accesare: 10 Decembrie 2020
   `https://pubs.opengroup.org/onlinepubs/009695399/functions/inet_addr.html`
5. Ioana Bogdan, pagina web din cadrul cursului - ultima accesare: 9 Decembrie 2020
   `https://profs.info.uaic.ro/~ioana/`
6. Techie Delight - ultima accesare: 9 Decembrie 2020
   `https://www.techiedelight.com/print-current-date-and-time-in-c/`
7. Tex StackExchange - ultima accesare: 14 Decembrie 2020
   `https://tex.stackexchange.com/`
8. Diagram Software and Flowchart Maker - ultima accesare: 15 Decembrie 2020
   `https://www.diagrams.net/`