PROTOCOALE DE COMUNICAȚIE
Tema #2 Aplicație client-server TCP și UDP pentru gestionarea mesajelor

Termen de predare: 26.04.2020

Responsabili Temă: **Ion-Dorinel FILIP, Florin POP**

1 Obiectivele temei de casă

Scopul temei este realizarea unei aplicații care respectă modelul client-server pentru gestiunea mesajelor. Obiectivele temei sunt:

- înțelegerea mecanismelor folosite pentru dezvoltarea aplicațiilor de rețea folosind protocoalele TCP și UDP;
- multiplexarea conexiunilor TCP și UDP;
- definirea unui tip de date folosit de aplicația propusă într-un protocol UDP predefinit;
- dezvoltarea unei aplicații practice de tip client-server ce folosește *socketi*.

2 Descriere generală

Pentru realizarea obiectivelor temei de casă vom considera implementarea unei platforme în care avem trei componente:

- **Severul** (unic) va fi prima componentă implementată de către voi și va realiza legătura între clienții din platformă, cu scopul publicării și abonării la mesaje.
- **Clienții TCP** sunt a doua componentă. Aceștia vor fi implementați de studenți și vor avea următorul comportament: un client TCP se conectează la server, poate primi (în orice moment) de la tastatură (interacțiunea cu utilizatorul uman) comenzi de tipul `subscribe` și `unsubscribe` și afișează pe ecran mesajele primite de la server.
- **Clienții UDP** (care vor veni **gata implementați din partea echipei responsabile cu tema**) publică, prin trimiterea către server, mesaje în platforma propusă folosind un protocol predefinit.

Funcționalitatea dorită este ca fiecare client TCP să primească de la server acele mesaje, venite de la clienții UDP, care fac referire la topic-urile la care sunt abonați. Arhitectura include și o componentă de SF (store&foreward) cu privire la mesajele trimise atunci când clienții TCP sunt deconectați.

Serverul

Serverul va avea rolul de broker (componentă de intermediere) în platforma de gestionare a mesajelor. Acesta va deschide 2 *socketi* (unul TCP și unul UDP) pe un port primit ca parametru și va aștepta conexiuni/datagrame pe toate adresele IP disponibile local. Pornirea serverului se va face folosind comanda:

```
./server <PORT_DORIT>
```

Logarea activității la consolă. Pentru monitorizarea activității serverului, se cere afișarea (la ieșirea standard) a evenimentelor de conectare, respectiv deconectare a clienților TCP. Aceasta se va face folosind mesaje de forma:

```
New client (CLIENT_ID) connected from IP:PORT.
```

respectiv

```
Client (CLIENT_ID) disconnected.
```

Serverul nu va afișa alte mesaje în afara celor specificate. Acest aspect este explicat, mai pe larg, în secțiunea 5.

Comenzi acceptate. Serverul va accepta, de la tastatură, doar comanda `exit` ce va avea ca efect închiderea simultană a serverului și a tuturor clienților TCP conectați în acel moment.

Comunicarea cu clienții UDP. Pentru comunicare, se vor folosi mesaje ce respectă un format definit conform tabelului 1.

	topic	**tip_date**	**conținut**
Dimensiune	Fix 50 de bytes	1 octet	Maxim 1500 de octeți
Format	Șir de caractere de maxim 50 de caractere, terminat cu \0 pentru dimensiuni mai mici de 50.	`unsinged int` pe 1 octet folosit pentru a specifica tipul de date al conținutului	Variabil în funcție de tipul de date, descris în tabelul 2.

Tabela 1: Formatul mesajelor UDP (topic, tip_date, conținut).

3 Clienții TCP

Clienții de TCP pot fi în orice număr, la fel ca cei UDP, și vor fi rulați folosind comanda următoare:

```
./subscriber <ID_Client> <IP_Server> <Port_Server>
```

unde:

- `ID_Client` este un șir de caractere ce reprezintă ID-ul clientului;
- `IP_Server` reprezintă adresa IPv4 a serverului reprezentată folosind notația dotted-decimal (exemplu 1.2.3.4);
- `Port_Server` reprezintă portul pe care serverul așteaptă conexiuni.

Comenzi primite de la tastatură

.

Clienții de TCP pot primi de la tastatură una dintre următoarele comenzi:

- `subscribe` - anunță serverul că un client este interesat de un anumit topic; comanda are următorul format: `subscribe topic SF` unde:
  - `topic` reprezintă topicul la care clientul urmează să se aboneze;
  - `SF` poate avea valoarea 0 sau 1 (se va explica în Secțiunea 4).
- `unsubscribe` - anunță serverul că un client nu mai este interesat de un anumit topic; comanda are următorul format: `unsubcribe topic` unde `topic` reprezintă topicul de la care clientul urmează să se aboneze;
- `exit` - comanda va fi folosită pentru a realiza închiderea clientului.

Mesajele afișate

Pentru comenzile de la tastatură: pentru fiecare comandă primită de la tastatură, clientul va afișa o linie de feedback de tipul `(un)subscribed topic`. Mesajul va fi afișat doar după ce comanda a fost trimisă către server.

Pentru mesajele primite de la server: pentru fiecare mesaj primit de la server, se va afișa imediat un mesaj de forma:

```
IP:PORT client_UDP - topic - tip_date - valoare mesaj
```

De exemplu, dacă un client UDP ce are adresa IP 1.2.3.4 publică la topicul `UPB/precis/1/temperature` valoarea `23.5`, ca număr real cu 2 zecimale, folosind port-ul sursă `4573`, atunci în client se va afișa:

```
1.2.3.4:4573 - UPB/precis/1/temperature - SHORT_REAL - 23.5
```

Pentru tipurile de date, se vor folosi notațiile din tabelul 2, iar valorile vor fi afișate humman-readable.

Tipul payload-ului	**Identificator tip (folosit de clienți)**	**Valoarea tip_date**	**Format payload**
Număr întreg fără semn	INT	0	octet de semn urmat de un uint32_t formatat conform network byte-order
Număr real pozitiv cu 2 zecimale	SHORT_REAL	1	`uint16_t` reprezentând modulul numărului înmulțit cu 100
Număr real	FLOAT	2	Un byte de semn + un `uint32_t` (în network order) reprezentând modulul numărului obținut din alipirea părții întregi de partea zecimală a numărului; un `uint8_t` ce reprezintă modulul puterii negative a lui 10 cu care trebuie înmulțit modulul pentru a obține numărul original (în modul).
Sir de caractere	STRING	3	Șir de maxim 1500 de caractere, terminat cu \0 pentru lungimi mai mici de 1500 de caractere.
*octetul de semn va fi 0/1 pentru numerele pozitive/negative.			

Tabela 2: Tipuri de date.

4 Funcționarea aplicației

Inițializarea aplicației este dată de pornirea serverului, la care ulterior se vor putea conecta un număr variabil de clienți TCP / UDP. Se cere ca serverul să permită conectarea/deconectarea de (noi) clienți la orice moment.

Fiecare client va fi identificat prin `client_id`-ul cu care acesta a fost pornit. La un moment dat, nu vor exista 2 clienți conectați cu același ID.

ID-ul unui client va fi un șir de maxim 10 caractere ASCII.

Serverul trebuie să țină evidența topic-urilor la care este abonat fiecare dintre clienți. La primirea unui mesaj UDP valid, serverul trebuie să asigure trimiterea acestuia către toți clienții TCP care sunt abonați la topicul respectiv.

Comanda de abonare are un parametru de `SF` (store&forward). Dacă acesta este setat la 1, înseamnă că un client dorește să se asigure că nu pierde niciun mesaj trimis pe acel topic. Astfel, dacă un client TCP se deconectează, apoi revine, el va trebui să primească de la server toate mesajele ce nu i-au fost trimise deja, chiar dacă acestea au fost trimise/publicate în timpul în care acesta a fost deconectat, conform unei politici de tipul store&forward. Acest lucru se aplică doar pentru abonările făcute cu `SF=1`, active înainte de publicarea mesajului. Pentru `SF=0`, mesajele trimise când clientul este offline se vor pierde, însă clientul va rămâne abonat la acel topic.

5 Alte observații

- Pentru a evita ambiguitatea la parsarea comenzilor din client, topicurile folosite în aplicație vor fi șiruri de caractere ASCII fără spații;
- În afara unei situații de eroare și pentru input corect, componentele software nu trebuie să afișeze (la stdin/stdout sau prin intermediul fișierelor) alte mesaje în afara celor specificate în cerință. În cazul în care considerați utilă afișarea altor mesaje, este necesar ca printarea lor să fie dezactivată în mod implicit;
- Modul de tratare al situațiilor de eroare nespecificate în enunț este la alegerea voastră. Totuși, se cere ca implementarea făcută să fie una **robustă**, fiind obligatorie verificarea situațiilor de eroare semnalate de API-ul `socket.h` și folosirea principiilor de programare defensivă cel puțin față de inputul utilizatorilor și al clienților, având în vedere politica de a fi cât mai strict cu formatarea mesajelor trimise și cât mai permisivi/precauți cu parsarea mesajelor/comenzilor primite. De exemplu:
  - Nu este acceptabil ca o aplicație de rețea să ajungă într-o stare nedefinită/de eroare datorită unei comenzi introduse eronat de utilizator sau datorită unui mesaj corupt;
  - În cazul în care o astfel de situație apare, este recomandat să se afișeze un mesaj sugestiv pentru eroarea apărută.
- La testarea temei, se vor folosi clienți UDP implementați de către echipa de asistenți, conform specificației din enunț. Pentru testarea temei, va trebui (probabil, dacă doriți) să vă dezvoltați și un astfel de client, dar nu va trebui să-l includeți în arhiva temei;
- Tema 2 se va testa fără un sistem automat de evaluare;
- Pentru comunicarea peste TCP va trebui să vă definiți propriul protocol, având în vedere că aplicația trebuie să meargă atât la rate de câteva mesaje pe oră cât și la viteze mari (zeci/sute de mesaje trimise în fiecare secundă). Astfel:
  - Operațiile trebuiesc realizate eficient, din punctul de vedere a utilizării rețelei. De exemplu, clienții trebuie să primească doar mesajele la care sunt abonați;
  - Dacă un client trimite mai multe mesaje în timp scurt, TCP le poate uni. Voi trebuie să vă asigurați că separarea lor se va face corect;
  - Toate comenzile și mesajele trimise în platformă trebuie să își producă efectul imediat, motiv pentru care trebuie să aveți în vedere dezactivarea algoritmului Neagle (hint: `TCP_NODELAY`);
  - Pentru ca rezolvarea să fie notată este **obligatorie** utilizarea unui protocol eficient (structuri de mesaje, peste TCP) pentru pentru reprezentarea mesajelor, iar acesta trebuie descris în cadrul fișierului `readme.txt`.
- Pentru SF, serverul va trebui să stocheze cantități variabile de informație, abordarea uzuală fiind scrierea ei în fișiere. Pentru realizarea temei, puteți presupune că serverul dispune de suficientă memorie RAM pentru a stoca toate mesajele în memorie, atât timp cât stocarea lor se face eficient, iar mesajele care nu mai sunt necesare sunt eliminate din memorie.

6 Trimiterea și notarea temei

Implementarea temei se poate face în C sau C++.

Trimiterea temei de casă și notarea acesteia va respecta regulamentul general al cursului. Tema va fi trimisă ca o arhivă `.zip` care va conține: `server.c`, `subscriber.c`, alte surse adăugate de voi, `Makefile`, `readme.txt` (format text).

Pentru trimiterea temelor se va folosi site-ul de curs: `acs.curs.pub.ro`.

Punctajul acordat temei de casă are următoarea structură: 50p server, 30p client TCP (subscriber), 20p stil implementare și informațiile cuprinse în fișierul readme.txt. O temă care nu compilează nu va fi notată. Punctajul pentru o componentă se oferă doar dacă acesta poate fi utilizată practic, ca parte a unei aplicații, conform descrierii temei. Depunctările pentru depășirea deadline-ului se vor aplica conform regulamentului.