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

Termen de predare: 08.05.2022 (soft)

Responsabili temă:
Florin POP, Dorinel FILIP
Radu CIOBANU, Bogdan MOCANU
Silvia NISTOR, Alina VÎRTAN, Andrei PANTELIMON

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 TCP și UDP;
- multiplexarea conexiunilor TCP și UDP;
- definirea și utilizarea unui tip de date petru un protocol predefinit peste UDP;
- dezvoltarea unei aplicații 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:

- **serverul** (unic) va realiza legătura între clienții din platformă, cu scopul publicării și abonării la mesaje;
- **clienții TCP** 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 (așa cum se poate observa și în figura 1). Aplicația va include și o funcționalitate de SF (store-and-forward) cu privire la mesajele trimise atunci când clienții TCP sunt deconectați.

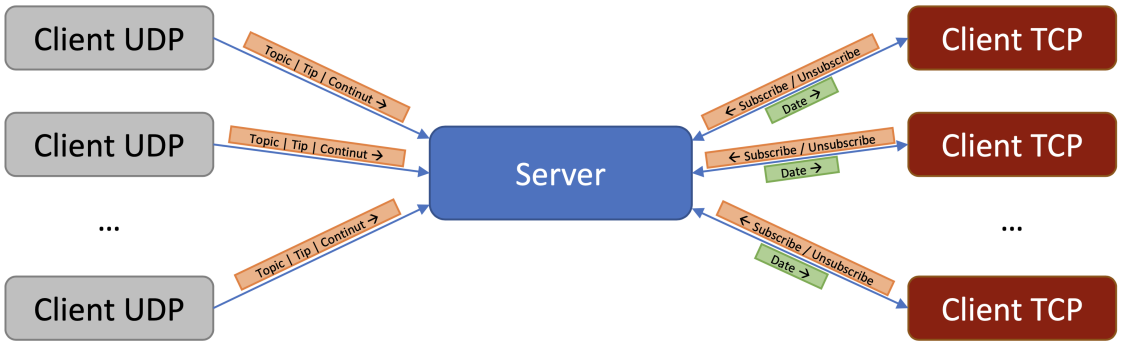

Figura 1: Arhitectura aplicației

3 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.

Pornire

Pornirea serverului se va face folosind comanda:

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

Mesaje afișate

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 <ID_CLIENT> connected from IP:PORT.
Client <ID_CLIENT> disconnected.
```

Serverul nu va afișa alte mesaje în afara celor specificate. Acest aspect este explicat, mai pe larg, în secțiunea 6.

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

Serverul va primi mesaje de la clienții UDP, ce respectă formatul definit în tabelul de mai jos.

	topic	**tip_date**	**conținut**
Dimensiune	50 de octeți	1 octet	Maxim 1500 de octeți
Format	Șir de maxim 50 de caractere, terminat cu \0 pentru dimensiuni mai mici de 50	`unsigned int` pe 1 octet folosit pentru a specifica tipul de date al conținutului	Variabil în funcție de tipul de date, așa cum este descris în secțiunea 4

4 Clienții TCP

Clienții TCP se pot abona și dezabona la topic-uri prin mesaje trimise către server.

Pornire

Pornirea unui client UDP se face cu următoarea comandă:

```
./subscriber <ID_CLIENT> <IP_SERVER> <PORT_SERVER>
```

În comanda de mai sus, `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), iar `PORT_SERVER` reprezintă portul pe care serverul așteaptă conexiuni. La pornire, clientul se conectează la server cu ID-ul dat ca parametru.

Comenzi acceptate

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

- `subscribe <TOPIC> <SF>` - anunță serverul că un client dorește să se aboneze la mesaje pe topic-ul `TOPIC`, iar `SF` poate avea valoarea 0 sau 1 (se va explica în secțiunea 5)
- `unsubscribe <TOPIC>` - anunță serverul că un client dorește să se dezaboneze de la topic-ul `TOPIC`
- `exit` - comanda va fi folosită pentru deconectarea clientului de la server și închiderea sa.

Mesaje afișate

Pentru fiecare comandă primită de la tastatură, clientul va afișa o linie de feedback de tipul următor:

```
Subscribed to topic.
Unsubscribed from topic.
```

Aceste mesaje vor fi afișate doar după ce comenzile au fost trimise către server.

Pentru fiecare mesaj primit de la server (adică date de pe un topic la care clientul este abonat), se va afișa imediat un mesaj de forma:

```
<IP_CLIENT_UDP>:<PORT_CLIENT_UDP> - <TOPIC> - <TIP_DATE> - <VALOARE_MESAJ>
```

De exemplu, dacă un client UDP cu adresa IP 1.2.3.4 publică la topic-ul `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 de mai jos, iar valorile vor fi afișate în format human-readable.

Tip payload	**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 octet de semn, urmat de un `uint32_t` (în network byte order) reprezentând modulul numărului obținut din alipirea părții întregi de partea zecimală a numărului, urmat de un `uint8_t` ce reprezintă modulul puterii negative a lui 10 cu care trebuie înmulțit modulul pentru a obține numărul original (în modul)
Șir de caractere	STRING	3	Șir de maxim 1500 de caractere, terminat cu \0 sau delimitat de finalul datagramei pentru lungimi mai mici
*Octetul de semn va fi 0 pentru numerele pozitive, 1 pentru cele negative			

5 Funcționarea aplicației

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

Fiecare client TCP va fi identificat prin ID-ul cu care acesta a fost pornit (`ID_CLIENT`), care poat fi un șir de maxim 10 caractere ASCII. La un moment dat, nu vor exista 2 clienți TCP conectați cu același ID. Dacă un client TCP încearcă să se conecteze la server cu același ID ca al altui client deja conectat, serverul va afișa mesajul următor (în același timp, clientul se va închide):

```
Client <ID_CLIENT> already connected.
```

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 topic-ul respectiv.

Comanda de abonare la un topic are un parametru de `SF` (store-and-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 în timpul în care clientul a fost deconectat, conform unei politici de tipul store-and-forward. Acest lucru se aplică doar pentru abonările făcute cu `SF` setat la 1 care erau active înainte de publicarea mesajului. Pentru `SF` egal cu 0, mesajele trimise când clientul este offline se vor pierde, însă clientul va rămâne abonat la acel topic.

6 Alte observații

Găsiți mai jos o listă de observații de care trebuie să țineți cont la implementarea temei:

- pentru a evita ambiguitatea la parsarea comenzilor din client, topic-urile folosite în aplicație vor fi șiruri de caractere ASCII fără spații
- componentele software nu trebuie să afișeze la STDOUT 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
- toate afișările trebuie să aibă o linie nouă la final; buffering-ul la afișare trebuie să fie dezactivat atât în server, cât și în clientul TCP, lucru care se poate face prin apelul funcției `setvbuf()` la începutul celor două programe, astfel: `setvbuf(stdout, NULL, _IONBF, BUFSIZ)`
- modul de tratare a 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`, precum și folosirea principiilor de programare defensivă cel puțin față de input-ul utilizatorilor și al clienților; nu este acceptabil ca o aplicație de rețea să ajungă într-o stare nedefinită sau de eroare din cauza unei comenzi introduse eronat de utilizator sau a unui mesaj corupt; în cazul în care o astfel de situație apare, este recomandat să se afișeze la STDERR 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, implementarea unui client UDP vă este pusă la dispoziție de către echipă (în arhiva de resusre a temei), dar puteți de asemenea să vă dezvoltați voi înșivă un astfel de client
- pentru testarea temei, aveți la dispoziție în arhiva de resurse pentru temă un script de testare, care va fi folosit și la evaluarea temei
- **important!** 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 trebuie realizate eficient, din punctul de vedere al 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 aplicație trebuie să își producă efectul imediat, motiv pentru care trebuie să aveți în vedere dezactivarea algoritmului Nagle (hint: `TCP_NODELAY`)
  - pentru ca rezolvarea să fie notată, este **obligatorie** utilizarea unui protocol eficient (structuri de mesaje peste TCP) pentru reprezentarea mesajelor, iar acesta trebuie descris în cadrul fișierului `readme.txt`.

- pentru store-and-forward, 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.

7 Trimiterea și notarea temei

Implementarea temei se poate face în C sau C++, iar trimiterea și notarea acesteia vor respecta regulamentul general al cursului. Tema va fi trimisă ca o arhivă `.zip` care va conține **în rădăcină** fișierele sursă, un fișier `readme.txt` (în format text), precum și un `Makefile` cu target-urile `server` (pentru compilarea serverului), `subscriber` (pentru compilarea clienților TCP) și `clean`. Pentru trimiterea temelor se va folosi site-ul de curs: `curs.upb.ro`.

Cele **100 de puncte** aferente temei se vor aloca după cum urmează:

- **40 de puncte** se primesc dacă serverul acceptă conexiuni și mesajele sunt afișate conform cerinței, **în clienții potriviți**, pentru toate mesajele primite de la clienții UDP
- **20 de puncte** se primesc dacă și numai dacă toate comenzile de abonare și dezabonare funcționează corect (un client TCP primește doar mesajele la care este abonat), conform enunțului pentru toate cazurile
- **20 de puncte** se primesc dacă opțiunea store-and-forward este implementată și abonările unui client se păstrează la deconectare și reconectare cu același ID
- **10 puncte** se primesc dacă programul funcționează corect cu mai mulți clienți și cu mesaje trimise repede (**atenție!** mult mai multe puncte - până la 100% - se pot pierde dacă cerința nu este respectată din cauza lipsei implementării unui protocol de nivel aplicație, care să facă încadrarea mesajelor sau a utilizării necorepunzătoare a socketilor)
- **10 puncte** se primesc pentru cod îngrijit și pentru calitatea informațiilor din fișierul `readme.txt`.

Punctajele de mai sus sunt orientative și se vor aloca de către corectori, proporțional cu nivelul de atingere a obiectivelor temei. Pe lângă posibilitatea neacordării sau acordării parțiale a punctajului pentru o anumită componentă, se vor aplica depunctări pentru soluții ineficiente, neajunsuri în funcționare sau de construcție a codului sursă (precum lipsa elementelor de programare defensivă) sau pentru nerespectarea bunelor practici privind programarea cu socketi. O parte dintre aceste depunctări sunt descrise în tabelul ce urmează.

Depunctarea	Când se aplică
-100 de puncte	Pentru soluțiile care nu folosesc biblioteca de socketi și/sau multiplexarea I/O
-100 de puncte	Pentru lipsa implementării unui protocol de nivel aplicație care să facă încadrarea mesajelor
-5 puncte	Pentru limitarea numărului de clienți sau de mesaje [1]
-5 puncte	Pentru afișarea de mesaje în plus față de cele cerute în enunț [2]
-10 puncte	Pentru neverificarea valorilor întoarse de apelurile de sistem
-10 puncte	Pentru implementările care au probleme la input invalid
-5 puncte	Pentru nedezactivarea algoritmului lui Nagle
-10 puncte	Pentru neînchiderea clienților TCP odată cu serverul
-5 puncte	Pentru nerealizarea conversiilor corecte de byte order
-20 de puncte	Pentru o funcționare lentă a aplicației

8 Întrebări frecvente

În această secțiune găsiți răspunsurile pentru cele mai frecvente întrebări legate de temă, precum și alte precizări pe care le considerăm lămuritoare. Această secțiune nu adaugă noi cerințe (răspunsurile fiind deja incluse în celelalte secțiuni sau în bunele practici învățate la curs și laborator), însă recomandăm citirea ei, dacă aveți nelămuriri.

- **Î: În enunț scrie că nu pot exista 2 clienți cu același ID la un moment dat. Ce facem dacă un client se deconectează, apoi se reconectează cu același ID? Este permis acest lucuru?**

  R: Da, ID-ul clientului se poate refolosi, această reconectare cu un ID care a mai fost folosit având rolul de întoarcere a unui client și însemnătate pentru implementarea părții de store-and-forward.

- **Î: Dacă un client se deconectează și apoi revine, la ce topic-uri va fi abonat? Ce mesaje (din urmă) va primi imediat ce se conectează?**

  R: Toate abonarile (atât cele cu `SF` setat pe 1, cât și cele cu `SF` 0) se păstrează asociate ID-ului de client până când accesta se dezabonează (cu `unsubscribe`)m indiferent de câte ori (sau din ce motiv) acesta se deconectează.

  Dacă subscribe s-a făcut cu `SF` setat pe 1, atunci mesajele care se primesc pe perioada în care clientul este deconectat trebuie păstrate, pentru a-i fi trimise imediat după reconectare.

  Mesajele nu se păstrează dacă un client nu este abonat la acel topic sau este abonat cu `SF` setat pe 0.

- **Î: Când se creează un topic? Se poate ca un client UDP să trimită mesaje pe un topic la care nu s-a abonat nimeni? Mă pot abona la un topic pe care încă nu s-a trimis nimic?**

  R: Topic-urile nu trebuie create.

  Clienții UDP pot trimite mesaje pe orice topic, la orice moment de tip, iar apelul de `subscribe` trebuie să funcționeze și dacă încă nu s-a publicat niciun mesaj pentru acel topic.

  Pentru calibrare, puteți să vă gândiți la situația în care vă abonați pentru a primi notificări atunci când apar noi rezultate în Google Search pentru numele unei firme. Rezultate noi pot apărea și dacă nu este nimeni abonat, iar dacă firma este nouă (și încă nu a scris nimeni nimic despre ea), vă puteți abona și înainte să apară primul rezultat de căutare pentru aceasta.

- **Î: Trebuie să implementez și clientul UDP?**

  R: În arhiva de resurse a temei, găsiți un client UDP care poate fi utilizat pentru generarea unor mesaje ce respectă specificațiile din enunț. Implementarea unui client propriu de UDP este recomandată, pentru a putea face mai multe teste, însă acest lucru nu este punctat și **nu trebuie să includeți un client UDP în rezolvare**.

[1] Se aplică pentru limitarea numărului de clienți sau a altor caracteristici, de exemplu, prin folosirea alocării statice de memorie.

[2] Loggingul este recomandat pentru debugging și audit. Totuși, aceste mesaje pot genera sincronizare falsă și negarantată față de bufferele de sistem. Astfel, ele trebuiesc dezactivate înainte de evaluarea aplicației.

- **Q: Am dezactivat algoritmul lui Nagle. Mai trebuie să fac ceva referitor la mesajele care se unesc? Trebuie să implementez un alt algoritm echivalent?**

  R: Algoritmul lui Nagle este folosit de implementarea TCP pentru a crește eficiența utilizării legăturii (el acționând ca o euristică prin care se favorizează trimiterea de segmente cât mai pline, având dimensiunea încărcăturii cât mai aproape dimensiunea maximă a unui segment).

  În temă se cere să dezactivați acest algoritm din cauză că aduce latență în plus și, pentru un broker de mesaje, aceast lucru este nedorit. **Nu trebuie să implementați voi un alt protocol echivalent cu Nagle**. Trebuie doar să dezactivați Nagle pentru a obține o comunicație cu latență scăzută.

  Cu toate că Nagle afectează recepția părților din fluxul TCP, nu este efectiv cauza „unirii de mesaje" despre care se vorbește în enunț și dezactivarea lui nu vă scutește de a fi atenți la acest caz (chiar dacă el ar putea fi întâlnit ceva mai rar apoi).

- **Q: Ce înseamnă că TCP transmite informația ca un flux de date?**

  TCP tratează informația transmisă ca pe un flux de date (octeții in ordine) și are un proces complex (cu ACK-uri, timere, etc.) care trebuie ascuns de utilizator (pentru a fi ușor de folosit). Astfel, apelul `send()` nu determină trimiterea imediată a unui mesaj/segment, ci face append datelor oferite la un buffer local de transmitere, trimiterea și recepționarea segmentelor de date fiind administrată de către stiva TCP/IP (implementată în kernel) în mod asincron.

  La rândul său, `recv()` se deblochează atunci când orice cantitate de informație devine disponibilă în buffer-ul de recepție, neexistând o încadrare implicită (în mesaje) a octeților din flux.

  În consecință, dacă protocolul de nivel aplicație vede transmisia ca fiind alcătuită din mesaje, atunci trebuie să ne așteptăm că este posibil să apară oricând:

  - trunchieri → din niște octeți trimiși de către o aplicație, în procesul de la distanță devine disponibilă doar o parte din ei, și totuși apelul `recv()` se întoarce, punând la dispoziția apelantului doar o parte din mesaj
  - concatenări → mai mult de un mesaj se citește cu un singur apel de `recv()`.

  Bineînțeles, pentru interpretarea corectă a părților din fluxul de octeți, puteți găsi soluții pe baza unei convenții folosite în protocolul de nivel aplicație creat de voi (cum, de altfel, este cerut și în enunțul temei).