Proiect 2- Messaging application- Cerc C

Obiectiv principal:

Obiectivul principal al acestui proiect este realizarea unui sistem client-server care va functiona ca o aplicatie de mesagerie. In acest scop trebuie dezvoltate 2 aplicatii denumite, respectiv: **MessageClient.exe** si **MessageServer.exe**.
Aplicatiile client si server vor folosi libraria **MessageCommunicationLib.lib** si fisierele header pentru a instantia obiecte de tip server, respectiv client.

Fisiere header:
communication_api.h
Headerul include toate cele 4 fisiere header de mai jos si expune functiile de initializare/deinitializare a librariei si a modulului de logare, in proiect fiind singurul header care e necesar sa il includeti.

client_communication_api.h
Contine functii pentru instantierea si distrugerea unui client si functii pentru trimitere si primire de mesaje catre si de la server.
Se pot folosi mai multe obiecte de tip client (**CM_CLIENT**) in acelasi program pentru a implementa comunicare multithreaded.

server_communication_api.h
Contine functii pentru instantierea si distrugerea unui obiect de tip server (**CM_SERVER**), pentru asteptare sau abandon de conexiuni noi de la clienti si functii pentru trimitere si primire de mesaje catre si de la client.

communication_data.h
Contine functii pentru manipularea de buffere necesare trimiterii de mesaaje intr client si server.

communication_error.h
Contine macro-uri pentru validarea statusului de tip **CM_ERROR** si cateva erori predefinite care sunt returnate de functiile expuse din librarie.

Instantierea librariei:
Pentru a putea folosi libraria, inainte de orice API folosit, trebuie apelat InitCommunicationModule iar la oprirea clientului/serverului trebuie apelat UninitCommunicationModule.

Pentru fiecare client nou conectat la server (**CM_SERVER_CLIENT**) se va creea un thread nou care va executa comenzile primite de la clientul respectiv. Cele doua aplicatii pot sa foloseasca cod comun (de exemplu va fi necesar un thread-pool atat in client cat si in server) prin intermediul unui dll sau lib.

Pentru implementarea cerintelor de mai jos, puteti folosi ca si exemplu proiectele MessageClient si MessageServer din **MessageApp (Proiect2-2018).zip**
In directorul BasicTester din arhiva gasiti un script python **test.py** care testeaza comanda echo. Puteti sa il folositi ca si pentru validarea basic a proiectului.

Cerinte server:

Aplicatia va avea numele **MessageServer.exe** si va primi in linia de comanda un singur numar reprezentand numarul maxim de client care pot fi conectati concomitant la aplicatie. In cazul in care parametrul primit este invalid, server-ul va afisa urmatorul mesaj si se va inchide:
Error: invalid maximum number of connections
In caz de succes server-ul va afisa pe ecran:
Success
Exemplu de utilizare:
1.
Linie de comanda: **MessageServer.exe 12**
Output asteptat: **Success**
2.
Linie de comanda: **MessageServer.exe abcde**
Output asteptat: **Error: invalid maximum number of connections**
Daca o alta eroare intervine in pornirea serverului se va afisa pe ecran un mesaj in formatul urmator, iar serverul se va inchide:
Unexpected error: [some meaningful error message]

Cerinte client:

Aplicatia va avea numele **MessageClient.exe** si nu va primi niciun parametru in linia de comanda. Introducerea de comenzi in client se va face exclusive prin stdin, respectiv input in consola. La pornire clientul va incerca sa se conecteze la server. In functie de rezultat se vor afisa unul din urmatoarele mesaje:
Successful connection
Error: no running server found - in cazul in care server-ul nu e pornit
Error: maximum concurrent connection count reached - in cazul in care s-a atins deja numarul maxim de conexiuni concomitente la server
Unexpected error: [some meaningful error message] – in cazul in care o alta eroare a intervenit in initializarea clientului
In ultimele trei situatii, aplicatia client va iesi imediat.

Clientul va suporta urmatoarele comenzi:

1. **echo** [some string]
   Aceasta comanda va trimite server-ului string-ul primit ca parametru si va primi inapoi acelasi string. Atat server-ul cat si clientul vor afisa in consola string-ul primit.

   Exemplu de utilizare:
   Input: echo Ana are mere.
   Output client: Ana are mere.
   Output server: Ana are mere.

2. **register** [username] [password]

Aceasta comanda are ca scop inregistrarea unui nou utilizator in aplicatie. In cazul in care user-ul si parola sunt valide, server-ul va retine combinatia de user/parola ca o combinatie valida. Serverul va mentine un fisier in path-ul: C:\registration.txt cu toate combinatiile de user/parola, cate una pe linie separate prin virgula. Astfel intre rulari succesive ale serverului, userii inregistrati vor ramane persistenti.

Restrictii username: only alphanumeric characters (a-zA-Z0-9)
Restrictii parola: trebuie sa contina cel putin o litera mare si un symbol nealfanumeric, nu poate contine spatii sau virgule. Parola trebuie sa aiba cel putin 5 caracter.

Exemplu de utilizare:
Input: register ana Mere!
Output client: Success

Mesajele valide care pot fi returnate de aceasta comanda sunt:
Success
Error: Invalid username
Error: Invalid password – in cazul in care contine caractere invalide (spatiu sau virgula)
Error: Username already registered
Error: Password too weak - in cazul in care restrictiile pentru parola nu sunt indeplinite
Error: User already logged in – in cazul in care pe conexiunea curenta exista deja un user care e logged in

Exemplu de registration.txt:
ana,Mere!
paul,Parola1234

3. **login** [username] [password]
   Aceasta comanda are ca scop loggarea unui user pentru clientul la care este introdusa comanda.

   Mesajele valide care pot fi returnate de aceasta comanda sunt:
   Success
   Error: Invalid username/password combination
   Error: Another user already logged in - in cazul in care un user e deja logat pe conexiunea curenta
   Error: User already logged in - daca userul cu care se incearca logarea este deja logat pe o alta conexiune

4. **logout**
   Aceasta comanda va deloga user-ul curent logat, permitand ulterior un alt login pe aceeasi conexiune la server.

   Mesajele valide care pot fi returnate de aceasta comanda sunt:
   Success

Error: No user currently logged in

5. **msg** [user] [message content]
   Aceasta comanda va trimite un mesaj din partea userului logat catre un alt user al aplicatiei.
   Exemplu de utilizare: (presupunem ca user-ul curent este Ana)
   Input: msg Mihai Ana are mere.
   Output client curent: Success
   Output client Mihai: Message from Ana: Ana are mere.
   Mesajele valide care pot fi returnate de aceasta comanda sunt:
   Success
   Error: No user currently logged in – daca pe conexiunea curenta nu a fost inca niciun login
   Error: No such user – daca destinatarul mesajului nu este inregistrat in aplicatie

   Nota: Aplicatia ar trebui sa suporte trimiterea de mesaje catre useri inactivi. In acest caz server-ul ar trebui sa retina mesajele (pentru punctaj maxim ar trebui sa fie retinute persistent intr-un fisier) pana in momentul in care destinatarul conecteaza la server (face login).

6. **broadcast** [message content]
   Aceasta comanda va trimite un mesaj catre toate conexiunile active la server (in afara de emitator).

   Exemplu de utilizare: (presupunem ca user-ul curent este Ana, iar Mihai si Radu sunt alti useri conectati la aplicatie)
   Input: broadcast Ana are mere
   Output client curent: Success
   Output client Mihai: Broadcast from Ana: Ana are mere.
   Output client Radu: Broadcast from Ana: Ana are mere.

   Mesajele valide care pot fi returnate de aceasta comanda sunt:
   Success
   Error: No user currently logged in – daca pe conexiunea curenta nu a fost inca niciun login
   Nota: Mesajele trimise prin broadcast ar trebui sa ajunga si la clienti de pe care inca nu s-a facut niciun login

7. **sendfile** [user] [file path]
   Aceasta comanda va trimite asincron un fisier catre un alt utilizator (active) al aplicatiei. Asadar, in timp ce fisierul este transmis, emitatorul ar trebui sa poata executa alte comenzi in paralel. Fisierul va fi salvat pentru destinatar in acelasi folder ca si locatia client-ului sau (de exemplu Ana are client-ul in **C:\Users\Ana\MessageClient.exe**, iar Mihai in **C:\Users\Mihai\MessageClient.exe**, astfel fisierul transmis va fi salvat in folderul C:\Users\Mihai). Emitatorul va fi notificat in momentul in care fisierul a fost receptionat de destinatar.

   Exemplu de utilizare: (presupunem ca user-ul curent este Ana)

Input: sendfile Mihai c:\Users\Ana\Pictures\mere.jpg
Input: msg Mihai Ti-am trimis o poza
Output Ana: Succesfully sent file: C:\Users\Ana\Pictures\mere.jpg
Output Mihai: Message from Ana: Ti-am trimis o poza
Output Mihai: Received file: mere.jpg from Ana

Mesajele valide care pot fi returnate de aceasta comanda sunt:
Success
Error: No user currently logged in
Error: No such user
Error: User not active
Error: File not found

Nota: Fisierele pot fi de orice dimensiune, deci nu exista garantia ca se vor putea citi intregi in memorie. Se recomanda folosirea unui thread-pool pentru trimiterea eficienta a fisierului.

8. **list**
   Aceasta comanda va afisa, cate unul pe linie, toti userii logati in aplicatie, cate unul pe linie
   Exemplu de utilizare:
   Input: list
   Output:
   Ana
   Mihai
   Radu

9. **exit**
   Aceasta comanda va inchide clientul, delogand automat userul curent.

10. **history** [username] [count]
    Aceasta comanda va afisa cate unul pe linie ultimele maxim count mesaje transmise intre user-ul curent si userul primit ca parametru. Daca nu exista niciun mesaj, se va afisa o linie goala (\n).

    Exemplu de utilizare:
    Input: history Mihai 6
    Output:
    From Ana: Ana are mere.
    From Mihai: stop
    From Ana: Ana are mere.
    From Mihai: please, stop
    From Ana: Ana are mere.
    From Mihai: Why didn’t I build a block function in this thing?

Mesajele de eroare valide care pot fi returnate de aceasta comanda sunt:
Error: No user currently logged in
Error: No such user

Consideratii generale:

- E foarte important (pentru evaluare) ca toate comenzile, precum si mesajele afisate sa functioneze la fel ca si cele din exemplele de utilizare. Astfel, formatul de output pentru history, list, output-ul pentru destinatarii mesajelor trebuie sa fie in acelasi format ca si exemplele din acest document (ex. From [username]: [Message content] for history). La fel, numele aplicatiilor: **MessageClient.exe** si **MessageServer.exe** trebuie sa ramana neschimbate. Numele, locatia si formatul fisierului registration.txt trebuie sa fie si ele neschimbate!
- Pentru toate comenzile se considera valide si mesajele de forma:
  Unexpected error: [some meaningful error message]
  Acestea pot aparea in cazuri precum: serverul se opreste, clientul destinatar se deconecteaza, un API faileaza etc.
- Se va puncta si stilul de codare al aplicatiilor
- Pentru acest proiect va trebui sa va definiti propriul vostru protocol de comunicare intre client si server
- Fiecare output in consola (stdout) se va termina cu caracterul newline (\n)
- Trebuie avut grija la managementul de memorie precum si la inchiderea handle-urilor pentru a nu avea unexpected crashes datorita epuizarii resurselor
- **Proiectul trebuie predat in urmatoarea forma**: o arhiva cu numele vostru, care contine 2 subfoldere denumite src si bin. Folder-ul src va contine codul sursa, in timp ce folder-ul bin va contine cele 2 executabile precum si un dll (daca aveti aceasta dependinta)
- Pentru a minimiza dimensiunea arhivei, puteti folosi comanda din Visual Studio pentru **Clean Solution** inainte de a incarca sursele (Build->Clean Solution). De asemenea puteti sterge fisierul cu extensia **DB** din folderul MessageApp (de obicei numit **MessageApp.VC.db**)
- Fiecare linie de comanda pe care clientul trebuie sa o suporte va avea maxim 255 de caractere, nu exista o limita pentru dimensiunile fisierelor care sunt trimise prin aplicatie
- Este permisa folosirea oricaror API-uri expuse de windows, din biblioteca standard si din libraria de comunicare primita, dar este interzisa folosirea bibliotecilor externe

Stil de codare

1 Indentarea codului

Pentru editarea codului se va folosi indentarea cu 4 spatii, fara TAB-uri. In Visual Studio, setarile se fac astfel: la Tools — Options — Text Editor — C/C++ se seteaza: Tab size 4, Ident size 4, [x] Insert spaces.

2 Identificarea fisierelor header

Fiecare fisier .H va avea la inceput si sfarsit un #ifndef / #define / #endif cu un macro unic sau *#pragma once*, cu numele fisierului. De exemplu, pentru un fisier help_tools.h vom folosi:

```
#ifndef _HELP_TOOLS_H_
#define _HELP_TOOLS_H_

...

#endif // _HELP_TOOLS_H_
```

sau

```
#pragma once

...
```

3 Definirea si utilizarea macro-urilor

De regula definitiile de macro se scriu **UPPER CASE**. De exemplu:

```
#define FLAGS_REQUEST_NO_REPLY 0x00000001
#define PREPROCSTR(x) PREPROCSTR2(x)
```

Se va evita utilizarea excesiva a macro-urilor, si tendinta de definire a unui 'meta limbaj'. Macro-urile nu trebuie sa ascunda multiple functionalitati complexe, intrucat citirea codului si urmariea flow-ul de executie ar putea fi ingreunate.

Este interzisa in mod ferm crearea unor macro-uri care sa contina tratari automate de erori, sau instructiuni care modifica flow-ul de executie. Exemple de instructiuni interzise in macro-uri: return, goto, leave, break, continue etc.

4 Denumirea parametrilor functiilor

Denumirea parametrilor functiilor se face folosind **CapitalCase**. Exemplu:

```
int MyFunction( int FirstParameter );
```

5 Denumirea variabilelor locale

Denumirea variabilelor locale se face folosind **almostCapitalCase**. De exemplu:

```
int retStatus;
CC_LIST bufferLength;
```

Identificatorii dintr-un bloc de cod nu au voie sa aiba nume identic cu identificatori din blocurile ”parinte”, pentru a nu-i ascunde pe acestia.

5 Denumirea variabilelor globale

Denumirea variabilelor globale se face folosind **almostCapitalCase** iar numele va fi prefixat cu „**g**”. De exemplu:

```
int gCcVector;
CC_LIST gList;
```

Identificatorii dintr-o functie nu au voie sa aiba nume identic cu identificatori globali, pentru a nu-i ascunde pe acestia.

6 Denumiri pentru struct, union

Denumirea de struct si union-uri se face **UPPER CASE**, iar definirea se va face atat pentru structura, cat si prefixat P pentru pointer la structura. Totodata se va folosi si definire prefixata cu **_** la inceputul definitiei. Denumirea campurilor din definitii se va face CapitalCase. De exemplu:

```
typedef struct _MY_STRUCT {
    int Field;
    ...
} MY_STRUCT, *PMY_STRUCT;
```

7 Descrierea blocurilor de cod, utilizarea ¸si pozi¸tionarea acoladelor

Fiecare for, while, if, do, switch va fi urmat de { }, chiar daca exista o singura instructiune in bloc. Blocul {} se scrie la acelasi nivel de identare cu instructiunea, iar continutul blocului, indentat cu un nivel. In cazul if-ului, else-ul se scrie la acelasi nivel de identare cu if-ul. De exemplu:

```
for (i = 0; i < 100; i++)
{
...
}

if (a > b)
{
    while (x < 5)
    {
        x++;
    }
    // ...
}
else
{
...
}
```

8 Semnificatia numelor de variabile locale, functii, structuri etc

Parametrii, variabilele locale, functiile, structurile etc. trebuie sa aiba nume descriptive (si nu x, cucu, aB, myList, etc.). Singura exceptie este cazul variabilelor de ciclu / indecsi, dar si asta doar in cazul unor cicluri / indexari simple, evidente.

Practici de codare defensiva

1 Validarea parametrilor de intrare ai functiilor

Este obligatorie validarea parametrilor de intrare in functii. De exemplu, pentru un string se poate (presupunem ca nu acceptam parametrii de tip string de lungime 0):

```
if ((NULL == StrParam1) || (0 == StrParam1[0]))
{
   return -1;
}
```

Toate functiile care acceseaza un parametru trebuie sa il valideze inainte de utilizare.

2 Alocare/dezalocare memorie

Toate functiile care aloca memorie pentru folosire locala sau pentru apelarea altor functii, trebuie sa se asigure ca la iesirea din functie, se elibereaza memoria alocata.

Functiile care aloca memorie pentru folosire globala (de ex: vectorul in care se retin informatiile pentru ccvector), la apelul functiei de destroy, va trebui sa dezaloce memoria folosita de instanta respectiva. De asemenea, *trebuie sa se asigure ca nu se elibereaza memorie deja eliberata*.