**Offline Messenger**

Vasilie Florin-Paul

Facultatea de Informatică, Universitatea Alexandru Ioan Cuza, Iași

paul.vasilie@info.uaic.ro

**Abstract.** Proiectul constă in dezvoltarea unei aplicații care permite schimbul de mesaje între utilizatori. Totodata un utilizator online poate trimite mesaje catre alți utilizatori online sau offline, acestora din urma apărându-le mesajele atunci când se vor conecta la server(vor deveni online). De asemenea, utilizatorii vor avea la dispoziție o agendă, astfel încăt vor ști care utilizatori sunt online sau offline. Fiecare utilizator va avea posibilitatea de a răspunde (reply) mesajelor primite. Aplicați va oferi și istoricul conversațiilor pentru și cu fiecare utilizator în parte.

**Cuvinte cheie:** protocol de comunicare, client, server, TCP, interfață cu tilizatorii, online, offline, reply, istoric conversații.

**1 Introducere**

Aplicația de tipul client/server pentru Offline Messenger va oferi posibilitatea de a trimite mesaje între utilizatori. Pentru a putea trimite un mesaj, utilizatorul trebuie să fie logat(online) operație realizată de server. Fiecare utilizator are o agendă cu toți utilizatorii online si offline. După ce alege un utilizator din agenda, online sau offline, va avea următoarele opțiuni: "Scrie mesaj","Vezi istoricul", etc, opțiunile sunt alese de utilizator și trimise la server. Fiecare utilizator ce primește un mesaj poate răspunde mesajului(Reply) sau îl poate ignora. Daca un utilizator offline primește un mesaj, acesta va vizualiza mesajul la logarea sa.

**1.1 Opțiunea "Scrie mesaj"**

Fiecare utilizator logat(online) poate scrie un mesaj altui utilizator nu conteaza dacă este online/offline. Se va trimite la server o structura cu NumeUtilizator si Mesaj. Astfel, serverul cunoaște destinatarul mesajului. Totodata serverul va salva conversatia într-un fișier.txt.

**1.2 Opțiunea "Vezi istoricul"**

Fiecare utilizator logat(online) va putea vizualiza istoricul conversațiilor cu alt utilizator. Se va trimite la server numele utilizatorului pentru a vedea istoricul, serverul va citi din fisier pentru un utilizator dat si va trimite la client conversația.

**2 Tehnologii utilizate**

**2.1 TCP**

TCP este un protocol orientat-conexiune care oferă posibilitatea de a realiza comunicații full duplex sigure. Se bazează pe protocolul IP. TCP reprezintă un set de reguli standardizate la nivel international, de comunicare între două entități distincte, orientat conexiune, ofera mecanisme de control al fluxului și control al congestive, astfel asigurându-se că un emițător rapid nu trimite mai multe date decât un receptor mai lent ar putea să prelucreze.

## 2.1.1 Concepte fundamentale ale TCP

**Socket.** Un socket este un mecanism bidirecțional ce poate fi utilizat atât pentru a comunica între procese depe același calculator, dar în special pentru a asigura comunicarea in rețea. Fiecare conexiune este identificată printr-o pereche de forma adresă IP : port, unde adresa reprezintă un set de 4 numere între 0 si 225 separate prin punct, iar portul este un număr intre 1 și 65535 (0 este rezervat pentru system). Aceasta pereche formează un socket.

**Adresă IP.** Este un set de 4 numere între 0 și 225 separate prin punct, reprezentând o etichetă asignată unei entități existente într-o rețea de calculatoare.

**Port.** Este un număr pe 16 biți care identifică in mod unic procerele ce rulează pe un system de calcul. Orice aplicație care realizează o comunicare în rețea va atașa un port conexiunii respective. Porturile cuprinse între 0 și 1023 sunt rezervate unor procese de system.

**2.2 Alte concepte folosite în proiect**

**2.2.1 Server**

Serverul este procesul existent pe mașina care găzduiește fișierele și oferă seviciul de download pentru fiecare client conectat. Mediază comunicarea între clienți și ofera servicii acestora.

**2.2.2 Client**

Este partea de aplicaţie care permite conectarea unui utilizator uman la un server pe care sunt găzduite.

**2.2.3 Interfață**

Este partea de afișare a aplicației ce realizează comunicarea efectivă cu utilizatorul conectat prin intermediul clientului.

**2.3 Limbaj utilizat**

Limbajul ales pentru realizarea proiectului este C.

**3 Arhitectura aplicației**

**3.1 Descrierea modulelor funcționale**

3.1.1 Modulul logare

Se ocupă cu logarea cu un username și o parolă la server pe portul specificat.

3.1.2 Modulul utilizator online

Se afisează pentru fiecare client, utilizatorii online.

3.1.3 Modulul de trimitere a unui mesaj(send)

Se ocupă cu trimiterea de mesaje. Se trimite mesajul la server, serverul stocheaza mesajul intr-un array de mesaje pentru fiecare utilizator.

3.1.4 Modulul de afisare a mesajelor(read)

Pentru fiecare utilizator logat, verific in stuctura de utilizatori daca are mesaj nou, daca are mesaj nou citesc din array mesajul si il trimit catre client prin write.

3.1.5 Modulul de afisare a istoricului

Pentru fiecare utilizator se va afisa istoricul conversațiilor cu toti utilizatorii. Salvez fiecare mesaj intr-un array numit arhiva si apoi in functia readArchive trimit catre utilizator continutul array-ului

3.1.6 Modulul meniu

În fuctia menu, citesc mesajul de la client care este de tipul : comanda user mesaj. În interiorul acestei funcție parsez mesajul primit astefel încât sa știu ce comanda trimite clientul, către cine și ce mesaj. În funcție de comanda primite, se apeleaza functiile.

**3.2 Schema modulelor funcționale**

Schema aplicației pentru server.

Client 1

Modulul de logare

Modulul afiseaza arhiva

Client 2

Modulul meniu

Client n

Mod citește mesaj read

Modulul utilizator online

Mod. de trimitere msg

(send

**3.3 Implementarea modulelor**

**3.1.1 Modulul logare**

int verifyLogin(thData \* tdL)

{

int size, rsize, contor = 0, i = 0;

char msg[2000], rmsg[2000], user[100], pass[100];

char \* tmp;

bzero (&msg, sizeof(msg));

bzero (&rmsg, sizeof(rmsg));

bzero (&user, sizeof(user));

bzero (&pass, sizeof(pass));

strcpy(msg, "\nBine ai venit la Offline Messenger!\n\tIntrodu Username si Password pentru logare.");

while(contor != -1)

{

if(contor == 0)

strcat(msg, "\nUsername:\n ");

else

strcat(msg, "\nPassword:\n ");

size = strlen(msg);

if(write(tdL->cl, &size, sizeof(int)) <= 0)

{

printf ("[Server] error sending msg to [Client].\n");

}

if(write(tdL->cl, msg, size) <= 0)

{

printf ("[Server] error sending msg to [Client].\n");

}

if(rsize = read(tdL->cl, &size, sizeof(int)) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

if(rsize = read(tdL->cl, rmsg, size) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

printf("[Server] Client response: %s\n", rmsg);

rmsg[strlen(rmsg) -1] = '\0';

if(contor == 0)

{

for(i = 0; i < MAX; i++)

{

if(strlen(u[i].name) < 0)

break;

if(strcmp(rmsg, u[i].name) == 0)

{

contor = 1;

tmp = u[i].name;

}

}

}

else

{

for(i = 0; i < MAX; i++)

{

if(strlen(u[i].pass) < 0)

break;

if((strcmp(rmsg, u[i].pass) == 0) && (strcmp(tmp,u[i].name)== 0))

{

contor = -1;

printf ("[Server] User %s s-a logat cu succes!\n", u[i].name);

tdL->id = u[i].id;

strcpy(tdL->name,tmp);

strcat(usersconnected,tmp);

strcat(usersconnected,"\n");

}

}

}

bzero (&msg, sizeof(msg));

bzero (&rmsg, sizeof(rmsg));

}

return 1;

}

**3.1.2 Modulul utilizator online**

void onlineUsers(void \* arg)

{

struct thData tdL;

tdL= \*((struct thData\*)arg);

char msg[4000],rmsg[4000];

char back[100],commback[100];

int size,rsize,j;

bzero(&msg,4000);

strcat(msg,usersconnected);

strcat(msg,"\nTastati back pentru a reveni\n");

size = strlen(msg);

if(write(tdL.cl, &size, sizeof(int)) <= 0)

{

printf ("[Server] error menu1 sending msg to [Client].\n");

}

if(write(tdL.cl, msg, size) <= 0)

{

printf ("[Server] error menu3 sending msg to [Client].\n");

}

if(rsize = read(tdL.cl, &size, sizeof(int)) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

if(rsize = read(tdL.cl, rmsg, size) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

printf("[Server] Client response menu reply: %s\n", rmsg);

rmsg[strlen(rmsg) -1] = '\0';

strcat(back,rmsg);

strcpy(commback,"back");

if (strcmp(back,commback)==0)

{

menu(&tdL);

}

else

{

menu(&tdL);

}

}

**3.1.3 Modulul de trimitere a unui mesaj(send)**

void sendMessage(char\* touser, char\* message)

{

int k = getIdByName(touser);

int i=0;

strcpy(u[k].mesaj[i], message);

strcat(u[k].arhiva[index1],message);

strcat(u[k].arhiva[index1],"\n");

i++;

index1++;

u[k].mesajenoi=1;

}

**3.1.4 Modulul de citire a mesajelor(read)**

void readMessage(int inID, void \* arg)

{

struct thData tdL;

tdL= \*((struct thData\*)arg);

char msg[4000],rmsg[4000];

char reply[100],commreply[100];

int size,rsize,j;

bzero(&msg,4000);

for(j=0;j<100;j++)

{

if (strlen(u[inID].mesaj[j]) == 0)

break;

strcat(msg,u[inID].mesaj[j]);

strcat(msg,"\n");

}

strcat(msg,"\nTastati reply pentru a raspunde/cancel pentru a revenit la optiuni\n");

size = strlen(msg);

if(write(tdL.cl, &size, sizeof(int)) <= 0)

{

printf ("[Server] error menu1 sending msg to [Client].\n");

}

if(write(tdL.cl, msg, size) <= 0)

{

printf ("[Server] error menu3 sending msg to [Client].\n");

}

if(rsize = read(tdL.cl, &size, sizeof(int)) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

if(rsize = read(tdL.cl, rmsg, size) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

printf("[Server] Client response menu reply: %s\n", rmsg);

u[inID].mesajenoi=0;

rmsg[strlen(rmsg) -1] = '\0';

strcat(reply,rmsg);

strcpy(commreply,"reply");

if (strcmp(reply,commreply)==0)

{

replystatus=1;

menu(&tdL);

}

else

{

replystatus=0;

menu(&tdL);

}

}

**3.1.5 Modulul de afisare a istoricului**

void readArchive(int inID, void \* arg)

{

struct thData tdL;

tdL= \*((struct thData\*)arg);

char msg[4000],rmsg[4000];

char back[100],commback[100];

int size,rsize,j;

bzero(&msg,4000);

for(j=0;j<400;j++)

{

if (strlen(u[inID].arhiva[j]) == 0)

break;

strcat(msg,u[inID].arhiva[j]);

strcat(msg,"\n");

}

strcat(msg,"\nTastati back pentru a reveni\n");

size = strlen(msg);

if(write(tdL.cl, &size, sizeof(int)) <= 0)

{

printf ("[Server] error menu1 sending msg to [Client].\n");

}

if(write(tdL.cl, msg, size) <= 0)

{

printf ("[Server] error menu3 sending msg to [Client].\n");

}

if(rsize = read(tdL.cl, &size, sizeof(int)) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

if(rsize = read(tdL.cl, rmsg, size) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

printf("[Server] Client response menu reply: %s\n", rmsg);

rmsg[strlen(rmsg) -1] = '\0';

strcat(back,rmsg);

strcpy(commback,"back");

if (strcmp(back,commback)==0)

{

menu(&tdL);

}

else

{

menu(&tdL);

}

}

**3.1.6 Modulul meniu**

void menu(void \* arg)

{

struct thData tdL;

tdL= \*((struct thData\*)arg);

int size, rsize, contor = 0, i = 0,j=0,k=0;

char msg[2000], rmsg[2000];

char commsend[4],commread[4],commarchive[6],commonusers[8];

char command[100];

char touser[100];

char message[1500];

char msgDeTrimis[2000];

bzero (&msg, sizeof(msg));

bzero (&rmsg, sizeof(rmsg));

bzero (&msgDeTrimis, sizeof(msgDeTrimis));

strcpy(msgDeTrimis, tdL.name);

strcat(msgDeTrimis,": ");

if (u[tdL.id].mesajenoi==1)

strcpy(msg,"\nAveti un mesaj nou !: read\n");

else

strcpy(msg,"\nIntroduceti optiunea dorita: on.users, send, read, arhiva\n");

if (replystatus==1)

{

strcpy(msg,"\nFolositi comanda send pentru a raspunde\n");

replystatus=0;

}

size = strlen(msg);

if(write(tdL.cl, &size, sizeof(int)) <= 0)

{

printf ("[Server] error menu1 sending msg to [Client].\n");

}

if(write(tdL.cl, msg, size) <= 0)

{

printf ("[Server] error menu2 sending msg to [Client].\n");

}

if(rsize = read(tdL.cl, &size, sizeof(int)) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

if(rsize = read(tdL.cl, rmsg, size) < 0 )

{

printf ("[Server] error reading msg from [Client].\n");

}

printf("[Server] Client response menu: %s\n", rmsg);

rmsg[strlen(rmsg) -1] = '\0';

for (i=0;i<strlen(rmsg);i++)

{

if (contor <=1)

{

if (rmsg[i]==' ')

{

contor++;

continue;

}

}

if (contor == 0)

{

command[i]=rmsg[i]; //aici sunt comenzile

}

if (contor == 1)

{

touser[j++]=rmsg[i]; // aici userul catre cine vreau sa trimit

}

if (contor > 1)

{

message[k++]=rmsg[i]; //aici mesajul

}

}

strcpy(commsend,"send");

strcpy(commread,"read");

strcpy(commarchive,"arhiva");

strcpy(commonusers,"on.users");

strcat(msgDeTrimis,message);

if (strcmp(command,commsend)==0)

{

printf("Comanda1 trimisa catre server este: %s\n",command); //aici daca comanda e send

strcat(u[tdL.id].arhiva[index2],msgDeTrimis);

strcat(u[tdL.id].arhiva[index2],"\n");

index2++;

sendMessage(touser,msgDeTrimis);

menu(&tdL);

}

if (strcmp(command,commread)==0)

{

printf("Comanda2 trimisa catre server este: %s\n",command); //comanda read

readMessage(tdL.id,&tdL);

}

if (strcmp(command,commarchive)==0)

{

readArchive(tdL.id,&tdL);

}

if (strcmp(command,commonusers)==0)

{

onlineUsers(&tdL);

}

**}**

**4 Concluzii**

Proiectul, Offline Messenger, poate sta la baza realizării unei aplicații mai mari de tipul chaturilor. Totodata prin realizarea acestui proiect consinder că voi intelege mai bine noțiunile legate de această materie.

**5.1 Cum poate fi imbunătățit aces proiect**

În primul rând, ar putea fi îmbunătățit prin adăugarea de noi optiuni: adăugare contact nou, ștergere contact, trimiterea acelui mesaj mai multor utilizatori, crearea de conturi noi, etc.

**5 Bibliografie**

Alboaie.L. *Reţele de calculatoare - curs.* Iaşi, România, 2014-2015;

www.wikipedia.org