Offline Messenger

Tema 2 RC

Ionescu Paul-Andrei

Introducere:

Aplicatia Offline Messenger este o aplicatie TCP server/client care ofera clientului o metoda de a trimite mesaje altor utilizatori care sunt online, chiar si offline. Utilizatorii offline pot vizualiza mesajele primate cat timp au fost delogati cu o comanda de afisare a istoricului intre el si persoana care a trimis mesajul. Aplicatia in acelasi timp ofera clientului posibilitatea de a da reply la anumite mesaje trimise.

Tehnologii utilizate:

TCP:

Transmission Control Protocol (sau **TCP**, în traducere liberă din engleză *Protocolul de Control al Transmisiei*) este un protocol folosit de obicei de aplicații care au nevoie de confirmare de primire a datelor. Efectuează o conectare virtuală full duplex între două puncte terminale, fiecare punct fiind definit de către o adresă IP și de către un port TCP.

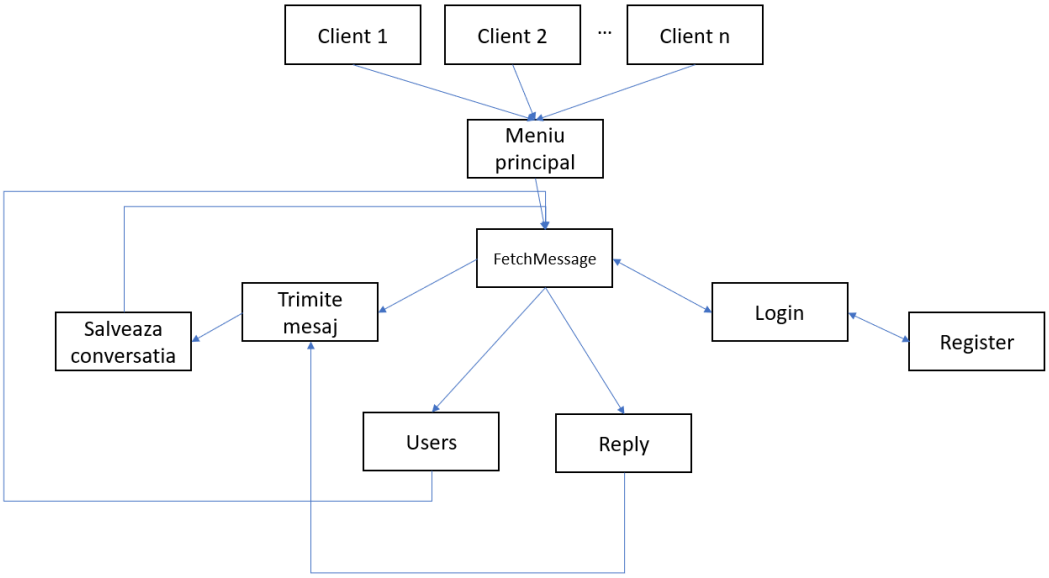
Transmission Control Protocol (TCP) este unul dintre protocoalele de bază ale suitei de protocoale Internet. TCP este unul dintre cele două componente originale ale suitei (celalalt fiind Protocolul Internet, sau IP), astfel încât întreaga suita este frecvent menționată ca stiva TCP/IP. În special, TCP oferă încredere, asigura livrarea ordonata a unui flux de octeți de la un program de pe un computer la alt program de pe un alt computer aflat în rețea. Pe lângă sarcinile sale de gestionare a traficului, TCP controlează mărimea segmentului de date, debitul de informație, rata la care se face schimbul de date, precum și evitarea congestionării traficului de rețea. Printre aplicațiile cele mai uzuale ce utilizează TCP putem enumera World Wide Web (WWW), posta electronica (e-mail) și transferul de fișiere (FTP).

De ce TCP?

Caracteristicile protocolului TCP sunt perfecte pentru o aplicatie de mesagerie instanta, deoarece protocolul respectiv garanteaza trimiterea in mod intact si nemodificat de mesaje prin socket.uri

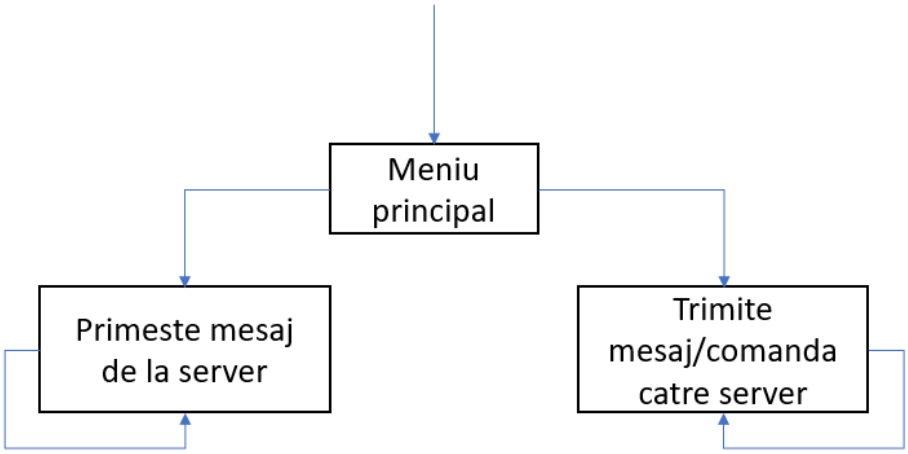
Arhitectura aplicatiei:

Arhitectura Server:



Serverul este procesul/aplicatia existent pe masina care face host si gazduieste clientii la un “IP”:”PORT” definit intr-un fisier de configurare, acesta face legatura intre client prin protocolul TCP definit mai sus, si salveaza istoricul mesajelor dintre client. Serverul prin metoda “Fetch message” serveste comenzile clientilor logati, in mod concurent.

Arhitectura Client:



Clientul este procesul/aplicatia existenta pe masina unui utilizator, acest proces foloseste un fork() pentru a putea trimite si primi mesaje intr-un mod dezorganizat, cum ii de obicei intr-o aplicatie de mesagerie.

Detalii de implementare:

Meniu principal: Aceasta presupusa metoda din diagrama aplicatiei server este o metoda abstracta care nu trebuie implementata, mai exact aceasta metoda este main() din aplicatie, aceasta ii conecteaza pe clienti la aplicatie.

Metoda FetchMessage:

```
int fetchMessage(int fd)
{
 char buffer[1024];                 /* mesajul */
 int bytes = 0;                     /* numarul de octeti cititi/scrisi */
 char msg[1024];                    //mesajul primit de la client
 char msgrasp[1030]=" ";       //mesaj de raspuns pentru client
 int clientulServit;
 bzero(buffer,1024); bzero(msg,1024), bzero(msgrasp,1030);

 for(int i=0;i<100;i++){
  if(clientDet[i].descriptor==fd && clientDet[i].folosit==1)
    clientulServit=i;
 }

 bytes = read (clientDet[clientulServit].descriptor, msg, sizeof (buffer));
 if (bytes < 0)
 {
  perror ("Eroare la read() de la client.\n");
  return 0;
 }
 msg[strlen(msg)-1]='\0';
 printf ("[server]Mesajul a fost receptionat de la %d...%s\n", fd, msg);
 /*Tratarea mesajului primit de la client comanda/mesaj*/
 if(msg[0]=='/'){
```

```
  printf("[Server]S-a dat o comanda catre server!\n");
  if(strcmp(strstr(msg,"login"),"login")!=0){
    char *param = getParameterFromCommand(strstr(msg,"login"));
    printf("Clientul %s incearca sa se logheze!\n",param);
    if(Login(param, clientulServit))
    {
      bzero(msgrasp,1030);
      sprintf(msgrasp,"Utilizatorul %s s-a logat cu succes!\n", clientDet[clientulServit].nume);
      if(write(clientDet[clientulServit].descriptor,msgrasp,sizeof(msgrasp)) <= 0){
        perror("write catre client error\n"); return 0;
      }
    }
    else{
      bzero(msgrasp,1030);
      sprintf(msgrasp,"Utilizatorul %s nu s-a logat corect!\n", clientDet[clientulServit].nume);
      if(write(clientDet[clientulServit].descriptor,msgrasp,sizeof(msgrasp)) <= 0){
        perror("write catre client error\n"); return 0;
      }
    }
  }
  else if(strcmp(strstr(msg,"register"),"register")==0){
    Register(getParameterFromCommand(strstr(msg,"register")));
  }
}
else{
  bzero(msgrasp,1030);
  sprintf(msgrasp,"%s:%s\n", clientDet[clientulServit].nume,msg);
  if(write(clientDet[clientulServit].descriptor,msgrasp,sizeof(msgrasp)) <= 0){
    perror("write catre client error\n"); return 0;
  }
}
```

```
  return bytes;
}
```

Aceasta metoda este baza aplicatiei server, cu aceasta functie serverul primeste un mesaj sau o comanda de la client, si aceasta functie creaza si organizeaza copiii pentru a satisface comenzile trimise de la client, iar parintele trimite, dupa caz, mesaj de confirmare inapoi la client.

Metoda Login (Modificat din Tema1):

```
int Login(char * username, int client_servit){
  printf("Logging in user: \"%s\"...\nSearching for username...\n", username);
  FILE *usr = fopen("users.cfg","r");
  char line[10];
  if(!usr) {
    perror("[Error]User file is NULL\n");
    exit(0);
  }
  printf("Looking into the file...\n");
  while(fgets(line,sizeof(line),usr))
  {
    line[strlen(line)-1] = '\0';
    if(strcmp(username,line) == 0)
    {
      printf("Found the user...\nLoging in...\n");
      clientDet[client_servit].logat=1;
      clientDet[client_servit].nume = username;
      fclose(usr);
      return 1;
    }
  }
  if(clientDet[client_servit].logat == 0)
    printf("User not found, not logging in.\n");
  fclose(usr);
  return 0;
}
```

Aceasta metoda este folosita si este foarte importanta in aplicatia noastra deoarece logheaza clientii la server ca mai tarziu sa se poata face diferenta dintre cei logati si cei anonimi. In acelasi timp prin metoda login se poate completa la structura client un nume, prenume de utilizator pentru a il putea identifica mai tarziu.

Metoda „Trimite Mesaj”:

Aceasta metoda foloseste descriptorii folositi de clienti la citire iar cand **clientul x** trimite un mesaj catre **clientul y**, *FetchMesage()* isi creaza un proces copil care scrie in descriptorul **clientului y** mesajul trimis de **clientul x**.

Metoda „Salveaza conversatia”:

Aceasta metoda este apelata imediat ce s-a trimis un mesaj de la un client x la y, se da append la fisierul care salveaza conversatia (eg:IonescuPopescuChat, citit ca si conversatia dintre Ionescu si Popescu) mesajul este salvat si indexat dupa un id special al mesajului trimis pentru a ajuta la folosirea functiei *„Reply”* si se intoarce inapoi in *meniul principal*.

Metoda „User”:

Aceasta metoda este una simpluta care trece prin toti clientii existenti si verifica daca in structura client variablia logat este 1 (adevarat) rezultand in mesajul „clientul x ONLINE” si daca logat este 0 (fals) atunci mesajul rezultat este „clientul y OFFLINE”

Metoda „Reply”:

Aceasta metoda foloseste fisierul de conversatie dintre clienti, cand clientul x vrea sa dea reply la un anumit mesaj trimis de clientul y acesta se va referi la id.ul asignat acelui mesaj, desigur ca intr-o consola va arata urat aceasta metoda, dar cu o interfata grafica care va face structura mesajului mult mai usoara de citit si de dat comanda catre server aceasta metoda ii va trimite mesaj clientului y de la clientul x „replied to message z: ha ha ha”, tocmai de aceia aceasta metoda va fi finalizata si va apela functia de trimitere al unui mesaj, care la randul ei va apela salvarea conversatiei si apoi se intoarce in main.

Clientul in mare:
Clientul este aplicatia care trimite si primeste mesaje de la server, acest proces face un fork() in care copilul se ocupa cu trimisul de mesaje catre server iar parintele se ocupa cu primirea de mesaje de la server, cand clientul foloseste comanda „/exit” acesta trimite la server stringul „/exit”, iese din bucla infinita din copil, termina executia iar parintele cand primeste inapoi de la server raspunsul pentru „/exit” acesta iese si el din bucla infinita de citire mesaje de la server si asteapta copilul sa isi termine executia pentru a nu lasa procese zombie pe dispozitivul clientului.

Concluzii:

In concluzie, acest proiect, Offline Messenger, este un proiect destul de mare si complex incat se poate trece prin toata materia, este o problema interesanta de rezolvat cand vine vorba de cum dai „reply” unui anumit mesaj, nu numai unei anumite persoane deoarece mesajele de preferat ar fi sa fie stocate si folosite din nou numai la cererea clientului, unde am putea sa trimitem fisierul raw (adica fara nicio modificare facuta la el), dar luand ca exemplu ca Mark de la Facebook a reusit sa implementeze un reply feature in aplicatia lui de mesagerie instanta, presupun ca este foarte posibil de realizat un astfel de task.
In termeni de imbunatarile a solutiei prezentate, realizarea unei interfete grafice ar fi ideala

in special pentru functia „Reply” de care am discutat mai devreme, dar ca sa nu raman blocat pe functia aceasta o alta imbunatatire pe care as putea sa o vad la aplicatie ar fi o metoda prin care clientul x poate sa ii trimita clientului y o imagine .png, .jpg, .gif, etc. prin intermediul TCP, folosindu-ne de buffer sa partitionam datele fisierului in bucatele mari cat bufferul si trimise prin TCP catre client, care va salva bucatile si va forma imaginea din nou pe masina utilizatorului, eventual intr-un folder separat cu numele „poze de la x”.

Bibliografie:

Folosit pentru detalii generale despre protocolul TCP folosit in „Tehnologii Utilizate”. Site.ul oficial „Computer Networks” UAIC: https://profs.info.uaic.ro/~computernetworks impreuna cu wikipedia: https://ro.wikipedia.org/wiki/Transmission_Control_Protocol.