RegisterS

Ignat Gabriel-Andrei, anul II, grupa A4, *gabriel.ignat.v09@gmail.com*

Facultatea de Informatica Iasi

Abstract. Acest document contine informatii despre realizarea proiectului RegisterS, anume tehnologiile utilizate, arhitectura aplicatiei in detaliu, detalii de implementare, cod relevant.
Pe langa informatii despre serverul RegisterS care se cerea a fi implementat, aceasta documentatie contine informatii generale si despre celelalte aplicatii implicate in procesul de comunicare, anume servere care ofera servicii si clienti care cer informatii.

Keywords: TCP · Concurent · Threads · RegisterS · Computer Networks

1 Introducere

Oarecum similar unui protocol DNS, proiectul RegisterS are rolul de a oferi o solutie de inregistrare a unor adrese IP si porturi pentru un anumit serviciu, informatii preluate apoi de catre un client care va incerca sa foloseasca acel serviciu.
De ce este importanta aceasta mapare a unor adrese IP si porturi pentru un anumit serviciu?
Raspunsul este imediat: pentru ca pentru un utilizator uman este mult mai usor sa-si aminteasca un nume decat o adresa IP si un PORT.

Un server care ofera un anumit serviciu va fi inregistrat in serverul RegisterS in momentul pornirii lui.
Clientul care doreste sa foloseasca un anumit serviciu va trimite un mesaj catre serverul RegisterS in care va specifica numele serviciului, iar serverul RegisterS va raspunde cu adresa IP si PORT-ul asociat serviciului cerut, pe care clientul le va folosi pentru a beneficia de serviciul dorit.

2 Tehnologii utilizate

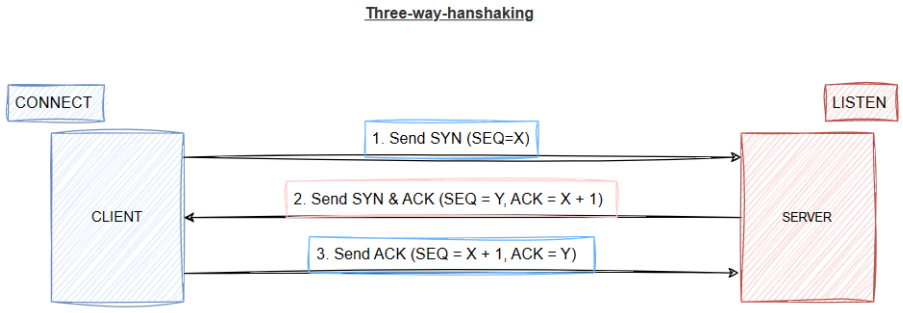
2.1 Protocolul TCP/IP

TCP(Transmission Control Protocol) este un protocol de retea aflat in Nivelul Transport al modelului TCP/IP, fiind folosit pentru a asigura o comunicare de incredere: trimite si primeste informatie ca un stream de octeti, si de maxima

calitate.
TCP este un protocol orientat-conexiune, dotat cu mecanisme de control al fluxului si de control al congestiei (creste/scade viteza de transmitere a octetilor in functie de puterea de citire a receptorului), cu mecanisme de confirmare, asigura ca informatia este trimisa/primita in ordine corecta, fara pierderi sau duplicate; deasemenea protocolul TCP ofera o comunicare full-duplex, ce permite transmiterea simultana de octeti in ambele sensuri.

Deoarece proiectul RegisterS presupune realizarea unei conexiuni stabile intre clienti si servere, dar si o transmisie/receptie corecta a octetilor, am decis sa folosesc protocolul TCP pentru a implementa arhitectura client/server.
Caracteristici:

- folosind protocolul TCP, va trebuie sa realizam conexiunea client/server, anume prin metoda **three-way-handshaking** si facem aceasta prin apelarea primitivelor BSD **listen()**,**connect()** si **accept()**.



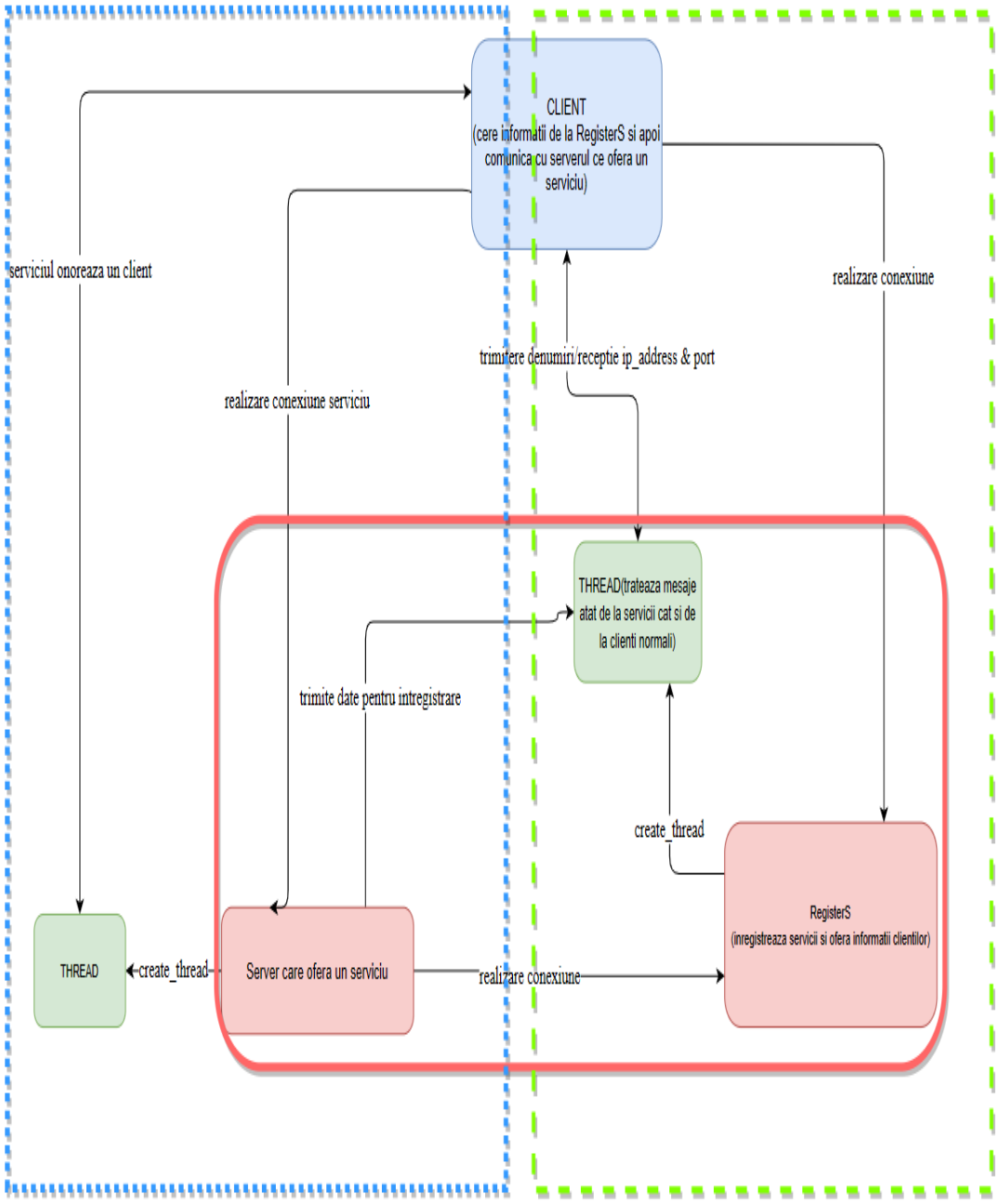
- la nivelul retea vom folosi protocolul IP, de aici si protocolul TCP/IP mentionat mai sus, ce consista in utilizarea unei adrese IPv4, pe 32 de octeti, pentru a identifica nodurile din retea. De asemenea, voi folosi si porturi, **communication end-points**, ce au rolul de a identifica o anume aplicatie din sistem;
- comunicarea se va realiza prin intermediul **socket**-urilor BSD, la care se vor atasa o adresa IP si un PORT, putand fi folosite similar unor descriptori de fisiere.
- transmiterea/receptia de bytes se face prin intermediul primitivelor **read** si **write** din standardul POSIX; transmiterea/receptia datelor va fi modelata in asa fel incat putem verifica daca s-a transmis/primit informatia in intregime(mesajele vor fi prefixate de lungimile lor).

3 Arhitectura aplicatiei

3.1 Paradigma client/server concurent

Pentru a trata in mod eficient cererile clientilor, am decis sa folosesc paradigma client/server concurent, in acest fel fiind posibila tratarea simultana a mai multor clienti.
Aspectul concurent va fi asigurat de crearea, in server, a cate unui thread pentru fiecare client. Acest lucru il fac folosind primitiva **pthread_create()**. Deoarece poate aparea fenomenul de **data race-condition**, voi utiliza conceptul de blocare a resurselor, implementat prin mecanismul de **mutex** cu ajutorul primitivelor **phtread_mutex_lock()** si **phtread_mutex_unlock()** ce vor bloca, respectiv debloca, o sectiune critica.
De ce am ales sa folosesc **thread**-uri in loc de **procese copil**(fork())?
Deoarece, procesele create prin fork() functioneaza prin mecanismul copy-on-write, care presupune copierea in intregime a procesului parinte, lucru care poate fi costisitor, ceea ce nu se intampla si in cazul crearii de thread-uri, numite si **light weight processes**.
De asemenea, deoarece thread-urile partajeaza un spatiu de date cu procesul care le-a creat(de fapt threadurile si variabilele globale/alocate dinamic apartin aceluiasi proces), inter-thread communication va fi mult mai facila decat inter-process communication; Asadar voi putea sa folosesc o aceeasi structura pentru a salva informatiile despre servicii(evitand astfel folosirea de pipe-uri, fifo-uri, etc. daca as fi ales metoda prin fork).
Pentru procesul client, nu este necesara o abordare concurenta, deoarece el doar va trimite un mesaj la un moment dat si va astepta raspunsul, fie de la serverul RegisterS, fie de la un anumit serviciu.

Diagrama detaliata a aplicatiei



1. Zona rosie reprezinta conectarea unui serviciu la serverul RegisterS si trimiterea informatiilor pentru inregistrare, dupa care va termina conexiunea.

2. Zona verde reprezinta conectarea unui client la serverul RegisterS si schimbul de mesaje cu acesta: clientul trimite o denumire si primeste adresa IP si port, cu care se va conecta la acel serviciu. Conexiunea cu serverul RegisterS va fi pastrata pana cand se decide terminarea clientului.
3. Zona albastra reprezinta conectarea unui client la un anumit serviciu(server) si schimbul de mesaje cu acesta: clientul trimite un mesaj in functie de serviciul ales si primeste raspunsul de la acel serviciu, dupa care inchide conexiunea cu el.

4 Detalii de implementare

4.1 Descriere generala

Flow-ul comunicarii dintre clienti si serverul RegisterS:

- pornire server RegisterS : ./<executabil_server_RegisterS>;
- serverul Registru va astepta(**listen**) conectarea de clienti si va **accept**a conexiunile;
- se pornesc anumite servere ce ofera servicii: ./<executabil_server_service> <ip_address_RegisterS> <port_number_RegisterS>;
  - initial vor trimite serverului RegisterS mesajul "<ip_address> <port_number> <name_service>" pentru a se inregistra; dupa trimiterea mesajului, vor inchide conexiunea;
  - apoi vor astepta conexiuni de la clienti si le va onora;
- pornire client : ./<executabil_client> <ip_address_RegisterS> <port_number_RegisterS>;
- clientul va incerca sa se conecteze la serverul RegisterS(**connect**):
  - pas1: clientul citeste de la STDIN o denumire de serviciu pe care o trimite serverului RegisterS;
  - pas2: daca primeste adresa IPv4 si port-ul se va conecta la acel serviciu si va introduce date pentru prelucrare, apoi va afisa rezultatul; la final se va incheia conexiunea cu serviciul;
  - dupa care va trece iar la pasul **pas1**

4.2 Structuri de date

Serverul RegisterS va folosi un vector in care un element este o structura cu urmatoarele campuri:

- char* **ip_address** - adresa IPv4 a serverului ce ofera serviciul respectiv;
- char* **port_number** - portul pe care asculta serverul respectiv;
- char* **name_service** - denumirea serviciului;

In acest vector serverul RegisterS va introduce informatii despre anumite servicii, le va sterge cand acestea devin indisponibile, le va partaja cu fiecare thread pentru interogare si/sau afisare.

4.3 (Pseudo)Cod sugestiv

Pregatirea serverului pentru acceptarea de conexiuni:

- in functia socket(AF_INET, SOCK_STREAM, 0): AF_INET specifica protocolul de retea IPv4; SOCK_STREAM indica protocolul de comunicare utilizat, anume TCP; 0 indica nivelul Transport;
- adresa IP a serverului este **IPv4**, 127.0.0.1(localhost) in cazul meu, dar se poate seta si drept INADDR_ANY care semnifica faptul ca se accepta conexiuni cu orice adresa IPv4.

```
/* ------------- start get_ready_server */
ERROR_EXIT((socket_server = socket (AF_INET, SOCK_STREAM, 0)) == -1,
           "[register]Eroare la socket().\n");

int on=1;
ERROR_EXIT(setsockopt(socket_server,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on)) != 0,
           "[register]Eroare la setsockopt().\n");

bzero (&server, sizeof (server));

  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl (IPv4);
  server.sin_port = htons (PORT);

ERROR_EXIT(bind (socket_server, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1 ,
           "[register]Eroare la bind().\n");
ERROR_EXIT(listen (socket_server, MAX_LEN_QUEUE) == -1 ,
           "[register]Eroare la listen().\n");
/* ------------- end get_ready_server */
```

Tratarea concurenta a clientilor

```
/* tratare clienti */
while (1)
{
    int client;
    thData * td;
    socklen_t length = sizeof (from);
    bzero (&from, sizeof (from));

    PRINT_MSG ("[register]Asteptam la portul %d...\n",PORT);

    ERROR_CONTINUE( (client = accept (socket_server, (struct sockaddr *) &from, &length)) < 0,
                    "[register]Eroare la accept().\n");

    ERROR_CONTINUE( (td = (struct thData*)malloc(sizeof(struct thData))) == NULL,
                    "[register]Eroare la malloc.\n");
    td->idThread = i++;
    td->cl = client;

    /* thread CREATION */
    ERROR_CONTINUE(pthread_create(&th[i], NULL, &treat, td) != 0,
                   "[register]Eroare la phtread_create().\n");
}/* while */
ERROR_EXIT(close(socket_server) == -1,
           "[register]Eroare la close().\n");
```

Algorithm 1 Functia de tratare a clientilor

```
1: procedure TREAT_CLIENT
2:     while true do ▷ Am folosit aceasta bucla infinita, deoarece am gandit modelul in care un client poate face mai multe request-uri intr-o aceeasi conexiune.
3:         read_msg(..);
4:         if is_msg_add_service(..) then  ▷ verifica daca mesajul vine de la un alt server care vrea sa se inregistreze.
5:             add_service(..)
6:         else  ▷ altfel, mesajul vine de la un client care cere informatii
7:             if is_service_in_list(..) then ▷ Daca serviciul se afla macar in lista de inregistrari a serverului RegisterS.  ▷ Odata cu aceasta cautare se vor elimina si serviciile indisponibile, eventual si cel cautat
8:                 if verify_if_service_is_available(..) then  ▷ Daca serviciul este disponibil, adica serverul este pornit.
9:                     msg_answer = "<ip_address> <port_number>"
10:                else
11:                    msg_answer = "service unaivailable" ▷ Serviciul s-a inchis intre timp. La raspuns se adauga si o lista cu serviciile disponibile
12:                end if
13:            else
14:                msg_answer = "service name incorrect"  ▷ Se adauga si o lista cu servicile disponibile
15:            end if
16:            send_msg(msg_answer,..);
17:        end if
18:    end while
19: end procedure
```

Functia verify_if_service_is_available:

- folosim primitiva popen() pentru a verifica daca un anumit port de la o anumita adresa IP este activ.
- comanda **nc -z -v** afiseaza statusul conexiunii serviciului aflat la adresa si portul dat, fara a realiza o conexiune propriu-zisa.

```
bool verify_if_service_is_available(const char* ip_addr, const char* port_nr, bool* res)
{
  *res = true;
  char command[LARGE_LEN_BUFFER];
  ERROR_RET(sprintf(command,"nc -z -v %s %s 2>&1", ip_addr, port_nr) < 0 , false,
            "Eroare la sprintf() in thread.\n");

  FILE *fp = popen(command,"r");
  if(fp == NULL)  { *res = false; return true;  }

  char buffer[LARGE_LEN_BUFFER];
  fgets(buffer,LARGE_LEN_BUFFER,fp);
  pclose(fp);

  if(strstr(buffer,"succeeded") == NULL)
    { *res = false; return true;}
  *res = true;
  return true;
}
```

Utilizare mutex:

De fiecare data cand voi accesa/modifica acel vector de servicii disponibile, ma voi asigura ca niciun alt thread nu are acces la vreo zona critica.
Acest lucru este realizat prin urmatoarele apeluri:

- phtread_mutex_init():

```
ERROR_EXIT(pthread_mutex_init(&mutex, NULL) != 0,
           "[register]Eroare la initializare mutex.\n"); /* pentru a evita race-condition */
```

- phtread_mutex_lock() & phtread_mutex_unlock() (exemplu la inregistrarea unui serviciu):

```
/* sectiune critica -> folosim mutex  */
ERROR_RET(pthread_mutex_lock(&mutex) == -1, false,
          "[Thread %d] Eroare la phtread_mutex_lock().\n",tdL.idThread);
ERROR_RET(add_service(ip_addr, port_nr, service, tdL.idThread) == false, false, /*adaugare serviciu*/
          "[Thread %d] Eroare la add_services().\n",tdL.idThread);
ERROR_RET(pthread_mutex_unlock(&mutex) == -1, false,
          "[Thread %d] Eroare la phtread_mutex_unlock().\n",tdL.idThread);
```

5 Concluzii

O astfel de aplicatie in care se mapeaza denumiri cu adrese IP si port se dovedesc a fi foarte folositoare pentru un utilizator uman.

Implementarea propusa indeplineste cerintele minimale, putand fi imbunatatita astfel:

- realizarea unei comunicari encriptate
- folosirea unei baze de date pentru maparea numelor de servicii cu adresa IP si portul
- posibilitatea de a cauta nume de servicii dupa adresa IP si port(similar DNS)

References

1. Curs "Retele de Calculatoare", Facultatea de Informatica Iasi.
   Autori : Alboaie Lenuta si Panu Andrei. **https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php**
2. Laborator "Retele de Calculatoare", Facultatea de Informatica Iasi.
   Autor: Bogdan Ioana, **https://profs.info.uaic.ro/ ioana.bogdan/**
   Autor: Matei Madalin, **https://profs.info.uaic.ro/ matei.madalin/rc/#/**
3. Curs "Sisteme de Operare", Facultatea de Informatica Iasi.
   Autor : Vidrascu Cristian. **https://profs.info.uaic.ro/ vidrascu/SO/index.html**
4. Use of popen :
   https://www.man7.org/linux/man-pages/man3/popen.3.html
5. Use of nc(Netcat) :
   https://phoenixnap.com/kb/nc-command
6. Mutexes for threads:
   https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
7. Draw diagrams online : **https://app.diagrams.net/**
8. Latex tutorial : **https://latex-tutorial.com/tutorials/**
9. How to write pseudocode in Latex :
   https://shantoroy.com/latex/how-to-write-algorithm-in-latex/
   https://www.overleaf.com/learn/latex/Algorithms
10. LNCS template :
    https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines