Raport tehnic – AppRepositoryS (B)

Mitrea Ana-Maria, An II, Gr. A5

1 Introducere

Acest raport tehnic prezintă proiectul AppRepositoryS (B) și are ca scop descrierea amănunțită a implementării arhitecturii unei aplicații client-server ce utilizează diverse tehnologii ale programării C/C++ în rețea, rulat pe un sistem de operare Linux/UNIX.

Aplicația AppRepositoryS este creată pentru a desemna un depozit de aplicații software, categorizate după anumite specificații hardware și software. Funcționalitatea acestei aplicații este constituită din baza de date și din comenzile date de client către server. Prin intermediul aplicației, orice client ce se conectează la server poate adăuga noi aplicații, specificând totodată și informațiile esențiale despre ele, cum ar fi: Nume, Sistem de operare necesar, minimul necesar pentru CPU, GPU, RAM, Hard Disk Storage, necesitatea conectării la Internet etc. De asemenea, clientul va avea la dispoziție opțiunea de a căuta aplicații în funcție de cuvântul cheie menționat. Serverul are capacitatea de a „răspunde" clientului prin legarea de o bază de date prin care se selectează informația ce trebuie trimisă înapoi.

2 Tehnologii utilizate

2.1 Interacțiunea Client-Server

Pentru a stabili interacțiunea dintre client și server, se va utiliza comunicarea bazată TCP-concurent, modelul orientat-conexiune.

Am ales TCP datorită siguranței transmiterii datelor în schimbul rapidității pe care o asigură UDP. Datorită folosirii TCP, comunicarea client-server este una sigură, ordonată, fără erori.

Avantaj folosire TCP

Prin folosirea acestui protocol, el ne permite să avem datele trimise de server exact în aceeași ordine în care au fost transmise inițial. În același timp, TCP-ul asigură dacă pachetele de date au fost primite cu succes și în caz de eroare (adică serverul nu primește confirmare – acknowledgment number), acesta va trimite din nou până când clientul primește informația (datorită numerelor de secvență asociate fiecărui pachet de date).

Totodată, înainte ca informația să fie trimisă, TCP-ul posedă proprietăți de verificare și detectare a erorilor. Datorită acestui fapt, TCP-ul trimite din nou pachetele de informații găsite ca fiind corupte și asigură destinatarului că mesajul pe care îl va primi nu conține erori. De asemenea, TCP-ul are prevederi pentru fluxul și congestia

de informații. Fiind orientat-conexiune, se asigură că nu există blocaj pe canalul de date care a fost configurat.

Dezavantaj folosire TCP
Dezavantajul major al acestui protocol de transmitere a datelor este faptul că mulțimea de caracteristici, management de conexiune, fiabilitate, control de flux de date, control al blocajelor, fac conexiunea TCP să fie mai înceată față de conexiunea UDP.

2.2 Interacțiunea Server-Bază de date

Pentru a lega serverul de baza de date, am ales folosirea bibliotecii SQLite care oferă un sistem de gestionare a bazelor de date relaționale în programarea C/C++. Această legătură se va face prin instalarea librăriei SQLite3.h cu ajutorul căreia putem lucra cu comenzile DDL, DML si DCL din limbajul SQL.

Baza de date va fi una simplă, mică, cu puține tabele. Acest lucru face ca librăria SQLite3 sa fie perfectă pentru această aplicație deoarece acceptă simultan mulți cititori, dar va permite doar un singur scriitor în tabele. Acest lucru nu este o problemă deoarece „scriitorii” vor sta la coadă, iar programul lucrează într-un timp foarte rapid cu baza de date și niciun blocaj nu durează mai mult de câteva zeci de milisecunde.

3 Arhitectura Aplicației

Arhitectura aplicației AppRepository este compusă din programul C++ (client și server) îmbinat cu o bază de date SQLite.

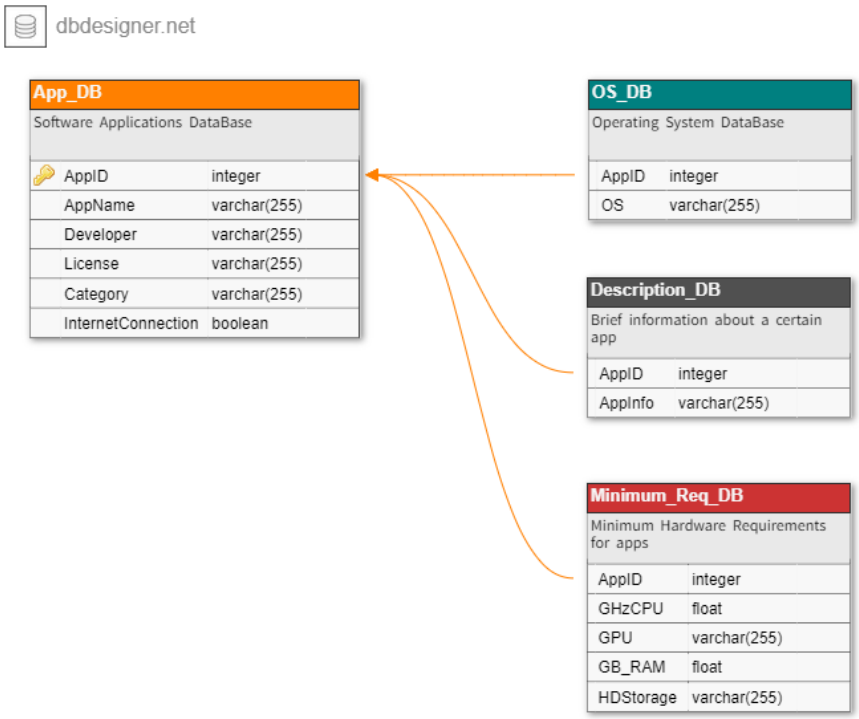
Pentru fiecare client conectat, serverul creează un proces copil astfel încât servirea lor să se facă în același timp. Serverul are rolul de a prelua date (comenzi) de la clienți, care mai apoi le prelucrează, iar informația prelucrată o trimite înapoi acestora.

Clientul introduce o comandă în terminal, serverul o primește, o prelucrează și decide care este răspunsul corespunzător interogării SQL a bazei de date.

3.1 Arhitectura bazei de date

Schema bazei de date cuprinde cel puțin următoarele tabele denumite astfel: App_DB, OS_DB, Minimum_Req_DB și Description_DB (fig.1) .

Fig. 1. Schema bazei de date



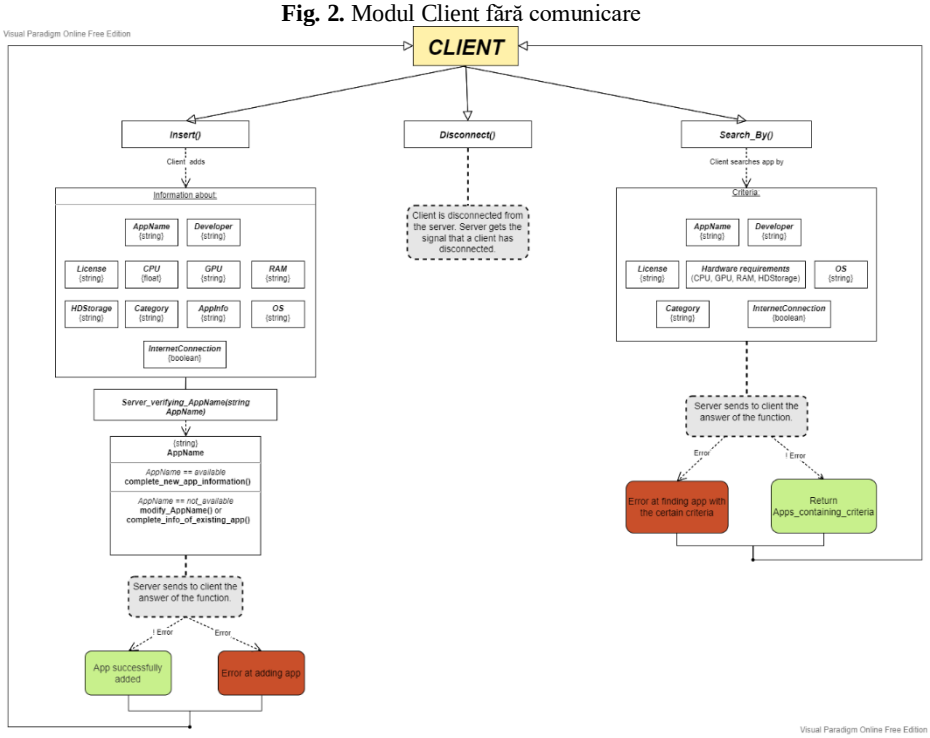
Pe parcursul implementării BD, se mai pot adăuga/șterge atribute/tabele în funcție de necesitate. Atributele tabelei App_DB conțin informații despre aplicațiile introduse, cele ale tabelului OS_DB reprezintă tipul sistemului de operare care poate rula aplicația respectivă, tabelul Minimum_Req_DB conține cerințele hardware minimale pentru rularea aplicațiilor, iar cele ale tabelului Description_DB reprezintă o scurtă informație despre fiecare aplicație. Toate atributele trebuie sa aibă valori diferite de

NULL, altfel valorile necompletate de clienți sunt prelucrate în 0 pentru numere și “-” pentru șiruri de caractere.

Cele patru tabele sunt legate prin cheia primară AppID si cu ajutorul căreia se identifică aplicațiile software. În același timp, acest atribut (cheie primară), AppID, are valori unice și se auto incrementează atunci când se adaugă aplicații noi în BD. Această constrângere ne oferă o garanție pentru unicitatea unei coloane sau a unui set de coloane.

3.2 Arhitectură client

Fig. 2. Modul Client fără comunicare



Concepte implicate

Clientul conectat are la dispoziție 3 comenzi valabile: *Insert()*, *Search_By()* și *Disconnect()*.

Dacă clientul alege funcția *Insert()*, aceasta are ca scop adăugarea în baza de date a unei aplicații software. Inițial, i se va cere clientului să completeze informația despre numele aplicației ce se va adăuga. Se va trimite către server numele și acesta îl va verifica dacă există o duplicare. În acest fel se verifica unicitatea aplicațiilor.

Dacă numele este valabil, clientul își continuă procesul de completarea a informațiilor. În caz de duplicare, clientul are următoarele opțiuni: fie numele nu este valabil și alege alt nume reprezentativ, fie completează informația lipsă despre aplicația deja existentă.

Dacă clientul alege funcția *Search_By()*, aceasta are ca scop căutarea detaliată a aplicațiilor. Clientul conectat are la dispoziție o gamă variată de criterii după care se poate face căutarea, cum ar fi: după numele aplicației, numele dezvoltatorului, licențe, cerințe hardware, sisteme de operare și altele. Rezultatul căutării va fi numărul de aplicații găsite și lista cu aplicațiile ce îndeplinesc criteriul dorit.

Dacă clientul alege funcția *Disconnect()*, aceasta are ca scop deconectarea acestuia de pe server și anunțarea serverului că un client a fost deconectat.

3.3 Arhitectură server

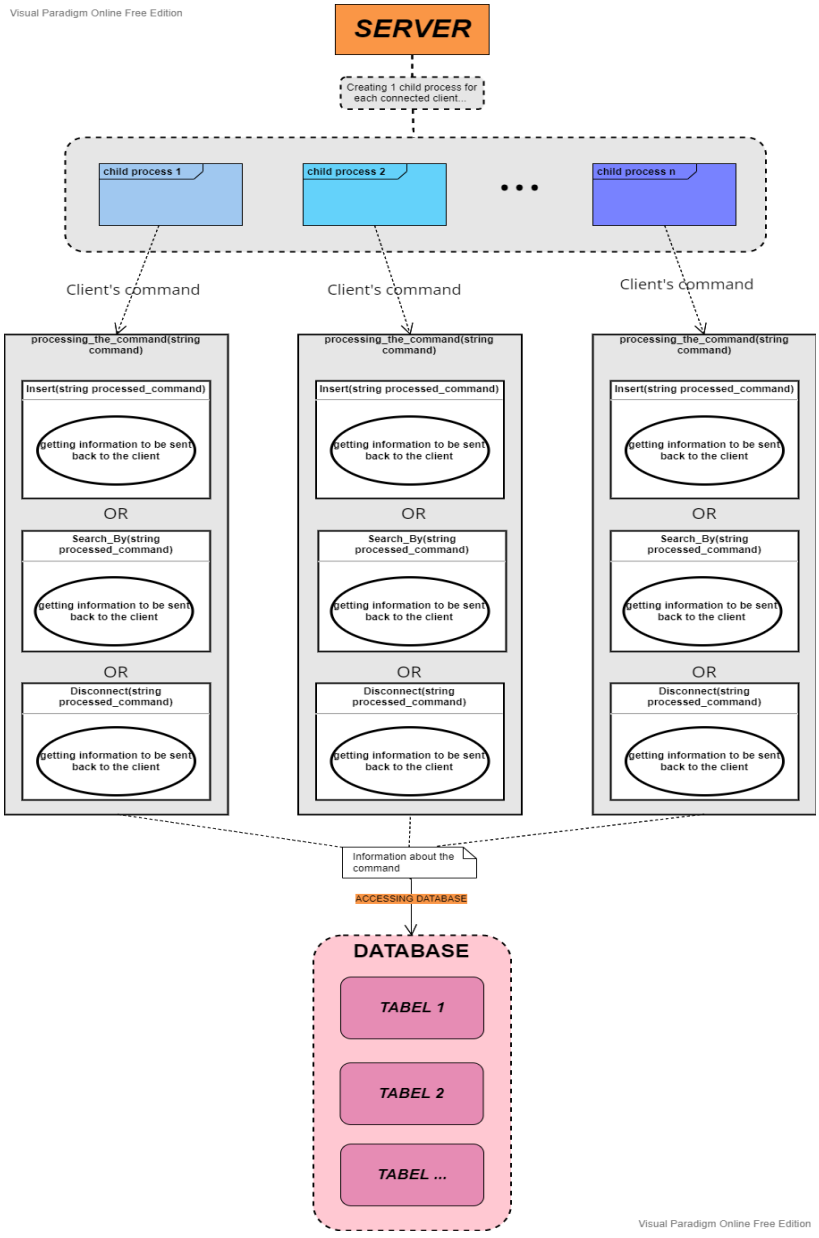


Fig. 3. Modul Server fără comunicare

Concepte implicate

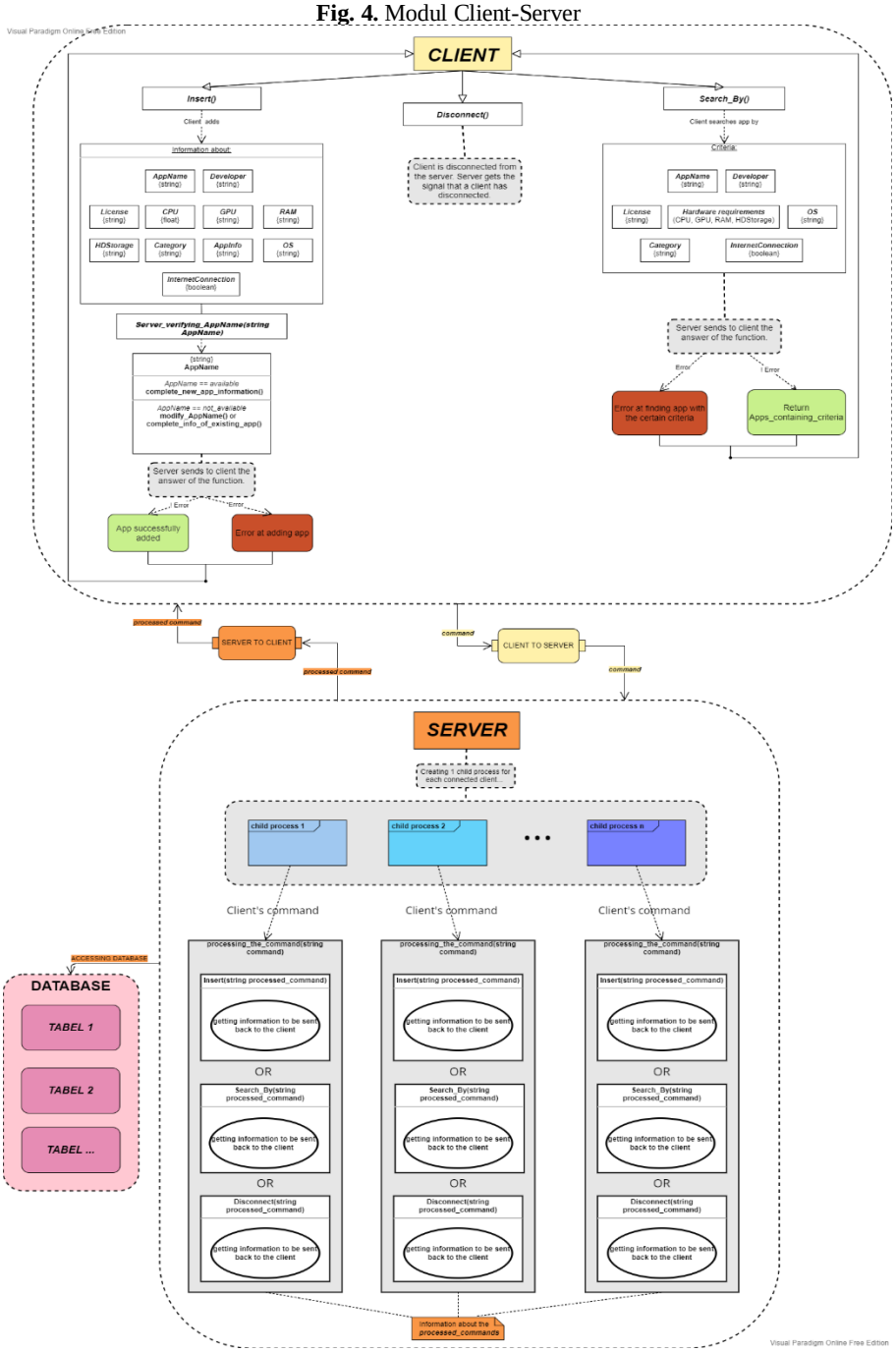
Serverul are ca funcționalitate răspunderea la comenzile trimise de client prin canalul de comunicare. Acesta primește mesajele și folosește interogările SQL pentru a prelucra informațiile care urmează a fi trimise.

În server se va folosi comunicarea concurentă, creându-se cu fork() câte un proces copil pentru fiecare client conectat. În acest mod, fiecare client interacționează cu serverul neavând un timp de așteptare pentru comunicare.

Procesul copil execută una dintre comenzile Insert(), Search_By(), Disconnect() și accesează la necesitate baza de date pentru a da clientului răspunsul.

3.4 Arhitectură client-server

Fig. 4. Modul Client-Server



Concepte implicate

Arhitectura client-server este cuprinsă din cele două arhitecturi client și server, legate printr-un canal de comunicare care se stabilește cu ajutorul TCP-ului concurent. Comenzile introduse de client sunt transmise către server, acesta le prelucrează, construiește răspunsurile pe baza interogărilor, apoi se trimit înapoi către clienți.

3.5 Detalii de implementare

Implementare protocol de comunicare

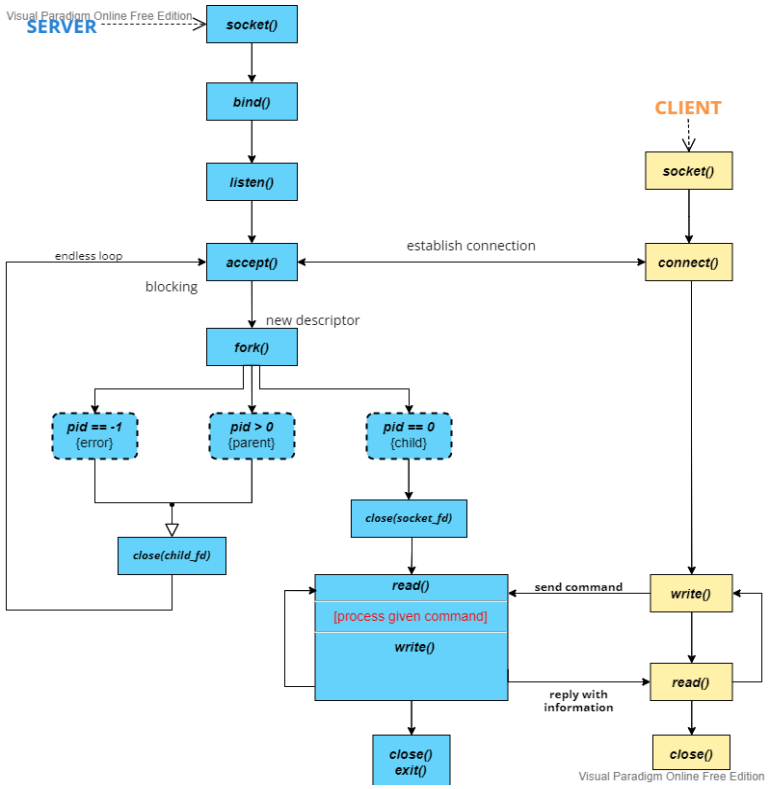


Fig. 5. Diagrama TCP Concurent

Protocolul de comunicare se construiește pe baza diagramei de mai sus, în care pentru fiecare client conectat, serverul creează cu ajutorul apelului de sistem fork() câte un proces copil. Procesul copil închide descriptorul socketului („socket_fd") și își folosește noul descriptor numit „child_fd". În caz de eroare la apelurile read/write din procesul copil, se închide forțat (close(child_fd)) conexiunea cu clientul, apoi se con-

tinuă „ascultarea” viitorilor clienți. După terminarea prelucrării comenzilor în acest proces copil, se trimite un semnal părintelui cum că acest proces copil și-a terminat execuția. Se va „altera” cu apelul de sistem signal() semnalul SIGCHLD astfel încât părintele așteaptă în mod blocant terminarea procesului copil.

În caz de eroare la apelul de sistem fork() sau dacă procesul curent este procesul părinte, se va închide conexiunea cu clientul respectiv.

Modul de transmitere a mesajelor de la client la server

Comenzile introduse de fiecare client vor fi de o anumită sintaxă menționată în aplicație.

Înainte ca fiecare comandă să fie trimisă către server, se va trimite numărul de bytes ca integer, apoi serverul citește comanda scrisă.

```
// -----IN CLIENT (trimitere comanda către server)-----
char* command;
/* ...preluare si trimitere comanda introdusa de cli-
ent...*/
int bytes = strlen(command) + 1;

if (write (sd, &bytes, sizeof(int)) <= 0)
{
    perror ("[client] Error at writting num bytes for
server.\n");
    return errno;
}

if (write (sd, command, bytes) <= 0)
{
    perror ("[client] Error at writting command for ser-
ver.\n");
    return errno;
}

// ------IN SERVER (citire comanda de la client)------
char command[100];
int bytes_sent;
// citire bytes si comanda trimisa de client

/* s-a realizat conexiunea, se astepta mesajul */
bzero (command, 100);
printf ("[server] Waiting client to write command...\n");
fflush (stdout);

/* Citire bytes si comanda de la client*/
int bytes_sent;  //bytes trimisi de client
```

```
if (read (client, &bytes_sent, sizeof(int)) < 0)
{
    perror ("[server] Error at reading num bytes from
client.\n");
    close(client); /* inchidem conexiunea cu clientul */
    break;  /* continuam sa ascultam */
}
if (read (client, command, bytes_sent) < 0)
{
    perror ("[server] Error at reading command from cli-
ent.\n");
    close(client);  /* inchidem conexiunea cu clientul */
    break;  /* continuam sa ascultam */
}
//prelucrare comanda...
```

Scenarii de utilizare

Scenarii uzuale de utilizare

- Clientul nu introduce corect toate argumentele in linia de comandă (i.e. adresă ip și port)
  - Se verifică la începutul programului corectitudinea sintaxei comenzii scrise de client în terminal și în caz de eroare, se returnează -1, oprindu-se execuția.
- Nu se poate crea un proces copil pentru clientul conectat
  - Serverul închide conexiunea cu clientul respectiv și continuă „ascultarea" următorilor clienți.
- Clientul trimite către server o comandă inexistentă
  - Clientului i se va specifica comenzile valabile. În acest caz nu se va citi niciun răspuns de la server.
- Serverul nu primește număr de bytes/comandă de la client
  - În aceste două cazuri se va închide conexiunea cu clientul respectiv, se iese forțat din bucla infinită din procesul copil și se continuă „ascultarea" următorilor clienți.

Scenarii particulare de utilizare

- Clientul se deconectează brusc de la server (i.e. pierde conexiunea)
  - Verificarea clientului dacă încă este conectat sa va face în continuu prin verificarea statusului pentru opțiunea „SO_KEEPALIVE". Se va modifica semnalul SIGPIPE trimis de socket astfel încât să se oprească forțat comunicarea dintre client și server.
- Clientul trimite o cantitate foarte mare de date
  - Se verifică dacă informația trimisă nu are mai mult de 1500 de bytes. În caz că lungimea informației trece de 1500 de bytes, se vor fragmenta pachetele de date ce urmează a fi trimise.
- Clientul nu completează numele aplicației pe care urmează să o adauge
  - În acest caz nu se poate adăuga aplicația. Se va specifica că acest câmp se completează obligatoriu, apoi se continuă procedura.
- Clientul nu alege criteriul de căutare a unei aplicații
  - În acest caz nu se pot căuta aplicații. Se va specifica că acest câmp se completează obligatoriu, apoi se continuă procedura.

- Serverul nu poate accesa baza de date
  - Se va verifica baza de date dacă se poate deschide înainte de a trimite interogările SQL.

Cod relevant particular proiectului

COD CLIENT (EXECUTABIL ȘI COMPILABIL)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <string.h>

extern int errno; //codul de eroare returnat de anumite apeluri

int port;

int main (int argc, char *argv[])
{
  int sd;          // descriptor socket
  struct sockaddr_in server;  // structura folosita pentru conectare
  char command[100];      // comanda introdusa

  /* Verificare existenta argumente in linia de comanda */
  if (argc != 3)
    {
      printf ("Sintaxa: %s <adresa_server> <port>\n", argv[0]);
      return -1;
    }

  /* Stabilire port */
  port = atoi (argv[2]);

  /* Creare socket */
  if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
    {
      perror ("[client] Error socket().\n");
      return errno;
    }
```

```
/* Umplere structura folosita pt realizarea conexiunii cu serverul */
/* Familia socket-ului */
server.sin_family = AF_INET;
/* Adresa IP a serverului */
server.sin_addr.s_addr = inet_addr(argv[1]);
/* Portul de conectare */
server.sin_port = htons (port);

/* Conectare client la server */
if (connect (sd, (struct sockaddr *) &server,sizeof (struct sockaddr)) == -1)
  {
    perror ("[client] Error connect().\n");
    return errno;
  }

printf("\n");
printf("Available commands on the server: \n");
printf("[1] To insert your Application, please write \"Insert\". \n");
printf("[2] To search an Application, please write \"Search\". \n");
printf("[3] To Disconnect from the server, please write \"Disconnect\" \n");
printf("\n");

while(1)
{
  int ok = 1;
  /* Citire comanda introdusa */
  bzero (command, 100);
  printf ("[client] Write your command: ");
  fflush (stdout);
  read (0, command, 100);

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

  // Trimitere comanda introdusa catre server
  if(strcmp(command,"Disconnect") == 0)
  {
    /// trimitere bytes la server
    int bytes = strlen("Disconnect") + 1;

    if (write (sd, &bytes, sizeof(int)) <= 0)
    {
       perror ("[client] Error at writting num bytes for server.\n");
       return errno;
    }
```

```
  // trimitere comanda la server
  if (write (sd, command, bytes) <= 0)
  {
     perror ("[client] Error at writting command for server.\n");
     return errno;
  }

  printf("You have disconnected from the server...\n");

  close(sd);
  return 0;
}
else
if(strcmp(command,"Insert") == 0)
{
  // trimitere bytes la server
  int bytes = strlen("Insert") + 1;

  if (write (sd, &bytes, sizeof(int)) <= 0)
  {
     perror ("[client] Error at writting num bytes for server.\n");
     return errno;
  }

  // trimitere comanda catre server
  if (write (sd, command, bytes) <= 0)
  {
     perror ("[client] Error at writting command for server.\n");
     return errno;
  }
}
else
if(strcmp(command,"Search") == 0)
{
  // trimitere bytes la server
  int bytes = strlen("Search") + 1;

  if (write (sd, &bytes, sizeof(int)) <= 0)
  {
     perror ("[client] Error at writting num bytes for server.\n");
     return errno;
  }
```

```
        if (write (sd, command, bytes) <= 0)
        {
          perror ("[client] Error at writting message for server.\n");
          return errno;
        }
      }
      else
      {
         printf("\n%s is an unavailable command. \n", command);
         printf("Available commands on the server: \n");
         printf("[1] To insert your Application, please write \"Insert\". \n");
         printf("[2] To search an Application, please write \"Search\". \n");
         printf("[3] To Disconnect from the server, please write \"Disconnect\" \n");
         printf("\n");
         ok = 0;
      }

      if(ok == 1)
      {
        int bytes_sent;
        /* citirea raspunsului dat de server (apel blocant pana cand serverul raspunde)
*/
        if (read (sd, &bytes_sent, sizeof(int)) < 0)
        {
          perror ("[client] Error at reading num bytes from server.\n");
          return errno;
        }

        char information[bytes_sent];
        bzero(information, bytes_sent);

        if (read (sd, information, bytes_sent) < 0)
        {
          perror ("[client] Error at reading message from server.\n");
          return errno;
        }

        information[strlen(information)] = '\0';

        /* afisam mesajul primit */
        printf ("[client] %s\n", information);
      }
    }
  }
```

COD SERVER (EXECUTABIL ȘI COMPILABIL)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>

#define PORT 2024

extern int errno; // codul de eroare returnat de anumite apeluri

void sighandler()
{
  while(waitpid(-1, NULL, WNOHANG) > 0);
}

int main ()
{
  struct sockaddr_in server;  // structura folosita de server
  struct sockaddr_in from;
  int sd;          //descriptorul de socket

  if (signal (SIGCHLD, sighandler) == SIG_ERR)
  {
    perror ("signal()");
    return 1;
  }

  /* crearea unui socket */
  if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
    {
      perror ("[server]Eroare la socket().\n");
      return errno;
    }

  /* pregatirea structurilor de date */
  bzero (&server, sizeof (server));
  bzero (&from, sizeof (from));

  /* umplem structura folosita de server */
  /* stabilirea familiei de socket-uri */
```

```
  server.sin_family = AF_INET;
/* acceptam orice adresa */
  server.sin_addr.s_addr = htonl (INADDR_ANY);
/* utilizam un port utilizator */
  server.sin_port = htons (PORT);

/* atasam socketul */
if (bind (sd, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)
  {
    perror ("[server]Eroare la bind().\n");
    return errno;
  }

/* punem serverul sa asculte daca vin clienti sa se conecteze */
if (listen (sd, 5) == -1)
  {
    perror ("[server]Eroare la listen().\n");
    return errno;
  }

/* servim in mod concurent clientii... */
while (1)
{
  int client;
  int length = sizeof (from);

  printf ("[server] Asteptam la portul %d...\n",PORT);
  fflush (stdout);

  /* acceptam un client (stare blocanta pana la realizarea conexiunii) */
  client = accept (sd, (struct sockaddr *) &from, &length);

  /* eroare la acceptarea conexiunii de la un client */
  if (client < 0)
  {
    perror ("[server] Eroare la accept().\n");
    continue;
  }


  if(fork() == 0)  // child process
  {
    close(sd);
    char command[100];
```

```
while(1)
{
 /* s-a realizat conexiunea, se astepta mesajul */
 bzero (command, 100);
 printf ("[server] Waiting client to write command...\n");
 fflush (stdout);

 /* Citire bytes si comanda de la client*/
 int bytes_sent;  //bytes trimisi de client
 if (read (client, &bytes_sent, sizeof(int)) < 0)
 {
    perror ("[server] Error at reading num bytes from client.\n");
    close(client); /* inchidem conexiunea cu clientul */
    break;  /* continuam sa ascultam */
 }

 if (read (client, command, bytes_sent) < 0)
 {
    perror ("[server] Error at reading command from client.\n");
    close(client);  /* inchidem conexiunea cu clientul */
    break;  /* continuam sa ascultam */
 }

 command[strlen(command)] = '\0';
 printf ("[server] Command sent by client: \"%s\"\n", command);


 /*pregatim mesajul de raspuns */
 if(strcmp(command,"Disconnect") == 0)
 {
  printf ("[server] A client has disconnected from server. \n");
  close (client); /* inchidem conexiunea cu clientul */
  exit(1);
 }
 else
 if(strcmp(command,"Insert") == 0)
 {
  printf ("[server] Client wants to add an app. \n");

  char information[100];
  bzero(information,100);
  strcpy(information, "[testing] Application has been added successfully.");

  printf ("[server] Sending back information... \n");
```

```
  int bytes = strlen(information) + 1;  // bytes de trimis la client

  if (write (client, &bytes, sizeof(int)) <= 0)
  {
     perror ("[server] Error at writting num bytes for client.\n");
     break; /* continuam sa ascultam */
  }

  // trimitere comanda la server
  if (write (client, information, bytes) <= 0)
  {
     perror ("[server] Error at writting command for client.\n");
     break; /* continuam sa ascultam */
  }
  else
     printf("[server] Client has received the message.\n\n");
}
else
if(strcmp(command,"Search") == 0)
{
  printf ("[server] Client wants to add an app. \n");

  char information[100];
  bzero(information,100);
  strcpy(information, "[testing] Applications have been found.");

  printf ("[server] Sending back information... \n");

  int bytes = strlen(information) + 1;  // bytes de trimis la client

  if (write (client, &bytes, sizeof(int)) <= 0)
  {
     perror ("[server] Error at writting num bytes for client.\n");
     break; /* continuam sa ascultam */
  }

  // trimitere comanda la server
  if (write (client, information, bytes) <= 0)
  {
     perror ("[server] Error at writting command for client.\n");
     break; /* continuam sa ascultam */
  }
  else
     printf("[server] Client has received the message.\n\n");
}
```

```
    }
    /* am terminat cu acest client, inchidem conexiunea */
    close (client);
    exit(1);
   }
   // parent process or error at fork() :
   close(client);
   }
}
```

4 Concluzii

Soluția aplicației poate fi îmbunătățită prin adăugarea unui algoritm de verificare a informațiilor introduse la diverse atribute. Acest algoritm ar trebui să facă verificarea dintre o bază de date deja existentă cu informații corecte si informația care urmează a fi introdusă. De exemplu, pentru atributul OS din tabela OS_DB, s-ar putea verifica dacă acel sistem de operare introdus este corect, adică clientul să nu introducă un șir de caractere aleatorii (precum Operating System: abcdefg).

Totodată, se poate ține cont de ușurarea completării informațiilor despre aplicație și trimiterea către server un mesaj concatenat cu toate cerințele hardware/software, in locul trimiterii rând pe rând a fiecărei cerințe.

În ceea ce privește folosirea tehnologiei de comunicare, un server UDP ar fi mai potrivit dacă s-ar dori o viteză mai mare de transmitere a mesajelor. Performața client-server crește, dar integritatea mesajelor nu este întotdeauna garantată.

În ceea ce privește esteticul clientului, soluția s-ar putea îmbunătăți prin folosirea unei biblioteci grafice care îmbunătățește vizual si ușurează folosirea comenzilor.

5 Bibliografie

1. Site-ul materiei, https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php.
2. Calancea Georgiana, Laboratories, https://profs.info.uaic.ro/~georgiana.calancea/laboratories.html.
3. Kristina Perunicic, „TCP vs UDP: Understanding the Difference”, https://www.vpnmentor.com/blog/tcp-vs-udp/.
4. Ramon Nastase, „Ce este Protocolul TCP si cum Functioneaza?”, https://ramonnastase.ro/blog/ce-este-protocolul-tcp-si-cum-functioneaza/.
5. TCP Concurrent Server, One Child per Client, https://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch30lev1sec5.html.
6. Unix Socket - Server Examples: Handle Multiple Connections, https://www.tutorialspoint.com/unix_sockets/socket_server_example.htm.
7. Vittorio Triassi, „SQL using C/C++ and SQLite”, https://www.geeksforgeeks.org/sql-using-c-c-and-sqlite/.

8. Site-ul materiei Baze de date, https://profs.info.uaic.ro/~bd/wiki/index.php/Laborator_1.
9. SQLite homepage, https://www.sqlite.org/index.html
10. SQLite, „Appropriate Uses For SQLite”, https://www.sqlite.org/whentouse.html.
11. StackOverflow, „Detecting TCP Client Disconnect”, https://stackoverflow.com/questions/283375/detecting-tcp-client-disconnect
12. „Using TCP keepalive under Linux”, https://tldp.org/HOWTO/TCP-Keepalive-HOWTO/usingkeepalive.html
13. „To detect an absent peer”, https://holmeshe.me/network-essentials-setsockopt-SO_KEEPALIVE/