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 server ce utilizează diverse tehnologii ale programării 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 citeriile menționate. 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. 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++. 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 este 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.

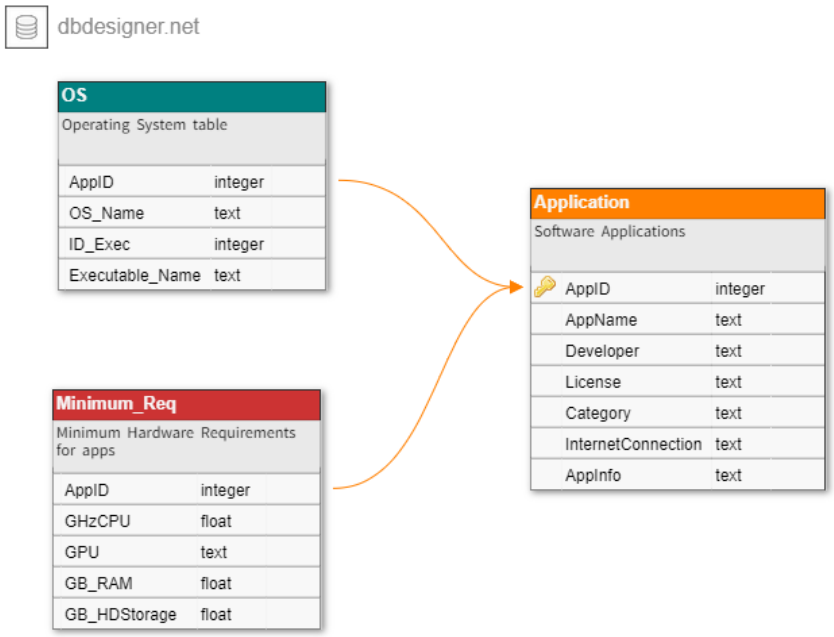
Pentru fiecare client conectat, serverul creează un proces copil astfel încât servirea lor să se facă în același timp. Serverul are rolul de a prelua date (comenzi) de la clienți, care mai apoi le prelucrează, iar informația prelucrată o trimite înapoi acestora.

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 următoarele tabele denumite astfel: Application, OS și Minimum_Req (fig.1) .

Fig. 1. Schema bazei de date

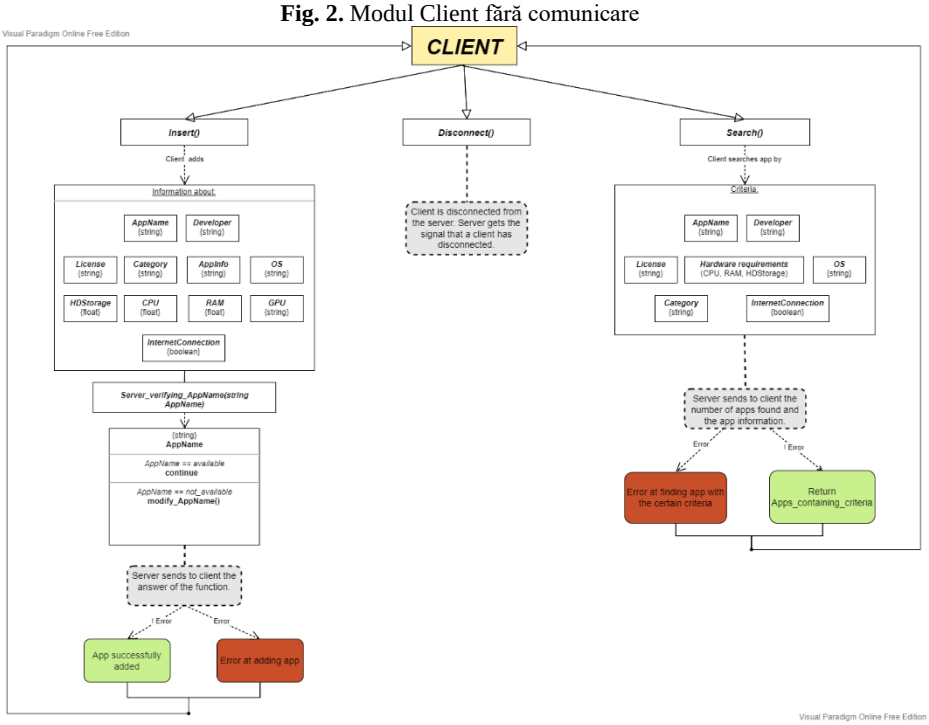


Atributele tabelei Application conțin informații despre aplicațiile introduse, cele ale tabelului OS reprezintă tipul sistemului de operare care poate rula aplicația respectivă, precum și numele executabilului, tabelul Minimum_Req conține cerințele hardware minimale pentru rularea aplicațiilor. Toate atributele trebuie sa aibă valori diferite de NULL (mai puțin ID_Exec și Executable_Name), altfel valorile necompletate de clienți sunt prelucrate în -1 pentru numere și “-” pentru șiruri de caractere.

Cele trei 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()* 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 in baza de date. În acest fel se verifica unicitatea aplicațiilor.

Dacă numele este valabil, clientul își continuă procesul de completarea a informațiilor. În caz de duplicare, clientul trebuie să aleagă alt nume reprezentativ. Câmpurile „nume aplicație”, „nume dezvoltator” și „nume executabil” (în cazul în care se doreș-

te adăugarea unuia în baza de date) sunt câmpuri obligatorii ce trebuie completate. Se va întreba clientul dacă dorește să adauge un executabil al aplicației ce rulează pe un anumit sistem de operare, în caz afirmativ, acesta va preciza numele. După aceea, numele fișierului va fi verificat dacă există sau nu. În caz de existență, fișierul este transmis către server și stocat cu nume specific „identificator.extensie" într-un folder numit „apps". Identificatorul asigură unicitatea executabilelor, acesta fiind incrementat la fiecare inserare în tabelul OS.

Dacă clientul alege funcția *Search()*, 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. În cazul în care clientul nu a adăugat niciun criteriu sau criteriul ales nu acoperă nicio aplicație, se va transmite câte un mesaj specific. Totodată, clientul poate descărca orice aplicație găsită după căutare, doar daca aceasta are în baza de date un ID_Exec diferit de NULL. În cazul în care nicio aplicație nu conține ID_Exec specific, clientul poate sări peste această etapă.

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

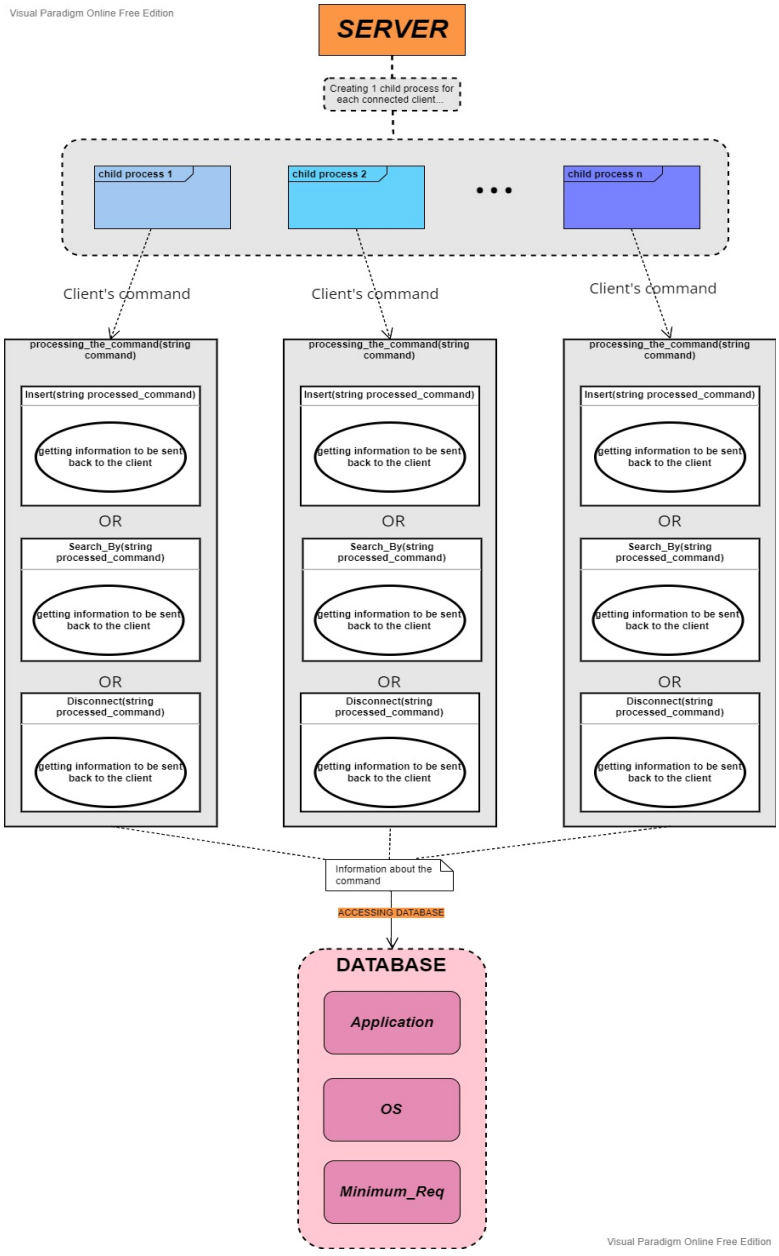


Fig. 3. Modul Server fără comunicare

Concepte implicate

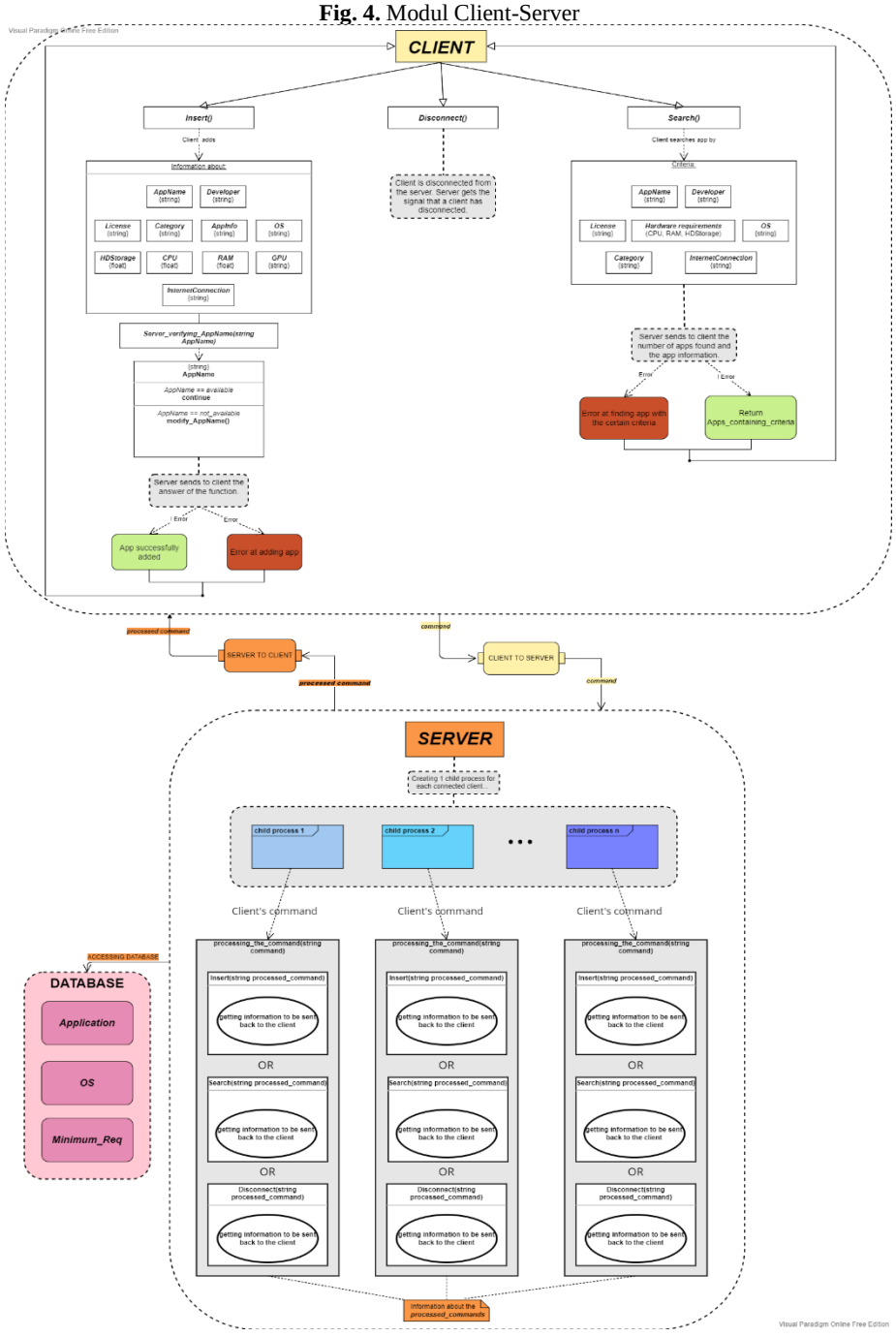
Serverul are ca funcționalitate răspunderea la comenzile trimise de client prin canalul de comunicare. Acesta primește mesajele și folosește interogările SQL pentru a prelucra informațiile care urmează a fi trimise.

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 (), Disconnect() și accesează la necesitate baza de date pentru a da clientului răspunsul. In cazul in care exista erori de comunicare client-server, serverul inchide automat conexiunea.

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 clienți 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

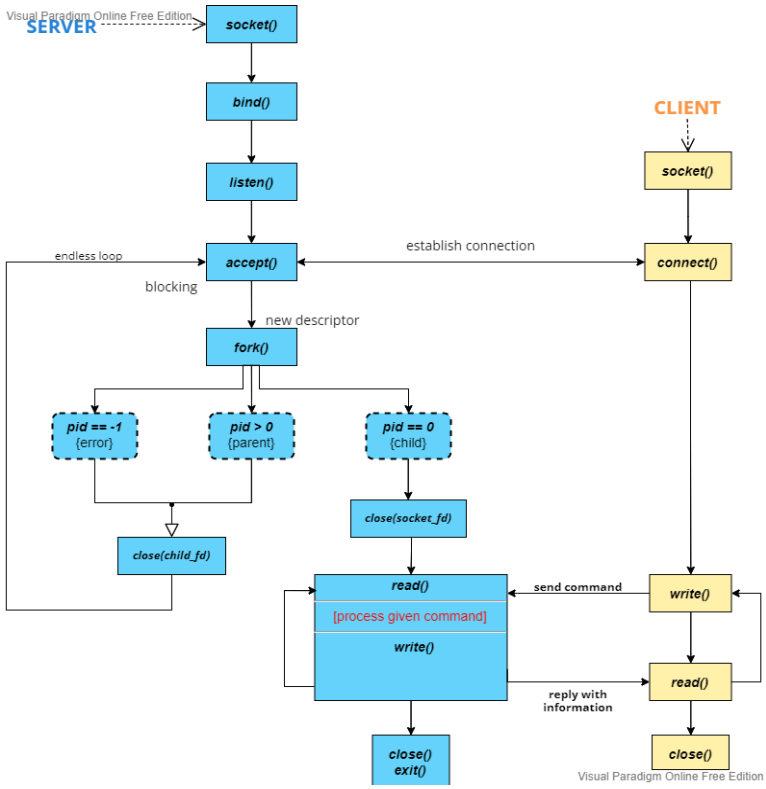


Fig. 5. Diagrama TCP Concurent

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.

```
void sendingCommand_CLIENT(int sd, int bytes, string com-
mand)
{
  if (write (sd, &bytes, sizeof(int)) <= 0)
  {
    errorHandling("[ERROR] Error at writting num bytes
for server.\n");
  }
  if (write (sd, command.c_str(), bytes) <= 0)
  {
    errorHandling("[ERROR] Error at writting command for
server.\n");
  }
}
```

Modul de primire a mesajelor de la client la server

```
string readingCommand_SERVER(int client)
{
  int bytes_sent = numBytesSent(client);
  char information[bytes_sent];
  bzero (information, bytes_sent);
  if (read (client, information, bytes_sent) <= 0)
  {
    cout << "[server] Error at reading command from cli-
ent. Client disconnected.\n";
    return "ERROR!";
  }
  string info = information;
  return info;
}
```

Modul de transmitere a mesajelor de la server la client

```
void sendingInfo_SERVER(int client, string information)
{
  int bytes = information.length() + 1;
  if (write (client, &bytes, sizeof(int)) <= 0)
  {
    cout << "[server] Error at writting num bytes for
client. Client disconnected.\n";
    close(client);
    exit(1);
  }
  if (write (client, information.c_str(), bytes) <= 0)
  {
    cout << "[server] Error at writting command for cli-
ent. Client disconnected.\n";
    close(client);
    exit(1);
  }
  else
    cout << "[server] Client has received the messa-
ge.\n";
}
```

Modul de primire a mesajelor de la server la client

```
string readingInfo_CLIENT(int sd)
{
  int bytes_sent;
  if (read (sd, &bytes_sent, sizeof(int)) <= 0)
  {
    errorHandling("[ERROR] Error at reading num bytes
from server.\n");
    return "ERROR!";
  }
  char information[bytes_sent];
  bzero(information, bytes_sent);
  if (read (sd, information, bytes_sent) <= 0)
  {
    errorHandling("[ERROR] Error at reading message from
server.\n");
    return "ERROR!";
  }
  string str = information;
  return str;
}
```

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.
- Clientul nu primește număr de bytes/comandă de la server
  - În aceste două cazuri se va închide forțat conexiunea cu serverul.

Scenarii particulare de utilizare

- Clientul se deconectează brusc de la server (i.e. pierde conexiunea)
  - În acest caz, dacă la apelul read, valoarea de return este 0, respectiv -1, conexiunea dintre client și server se închide forțat, astfel serverul este anunțat că un client a pierdut conexiunea într-un mod neașteptat.
- Clientul nu completează numele aplicației/dezvoltatorului/executabilului pe care urmează să o/îl 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
  - Se trimite mesajul specific: "Found 0 programs. No criteria entered".
- Clientul nu scrie corect numele executabilului ce urmează a fi trimis către server
  - Se va folosi o funcție ce verifică existența fișierului cu numele introdus, înainte de a se trimite către server. În cazul în care fișierul este inexistent, serverul pri-

mește mesajul specific: „FILE_DOES_NOT_EXIST”, altfel serverul primește fișierul transmis de client.

- 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

- Searching existing AppName in db

```
bool verifyingExistingName(sqlite3* db, string appName)
{
    string sqlQuery = "SELECT AppName FROM Application WHERE
AppName=\"" + appName + "\";";
    string sqlResponse; // sql query response for search exis-
ting appName in database
    sqlResponse = selectQuery(db, sqlQuery);
    if(sqlResponse.empty())
        return false;

    string name = sqlResponse.substr(10,string::npos);
    name = name.substr(0, name.length()-1);
    if(name == appName)
        return true;
    return false;
}
```

- Insert Query Function

```
string insertQuery(sqlite3* db, string sqlQuery)
{
    char* SQL_errorMessage; // error message from sql query
    char data[maxi]; // callback argument
    string sqlQueryResult;
    sqlQueryResult.clear();
    data[0] = 0;
    int sqlExec = sqlite3_exec(db, sqlQuery.c_str(), callback,
data, &SQL_errorMessage);
    if (sqlExec != SQLITE_OK)
    {   sqlQueryResult = SQL_errorMessage;
        sqlite3_free (SQL_errorMessage);
        return sqlQueryResult;
    }
    else
    { return "[Inserting Query succeeded]";}
}
```

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_Name din tabela OS, s-ar putea verifica dacă acel sistem de operare introdus este corect, adică clientul să nu introducă un șir de caractere aleatoriu (precum Operating System: abcdefg).

O altă idee de îmbunătățire ar fi adăugarea unei funcții de verificare a corectitudinii extensiei executabilelor. Așadar, pentru fiecare sistem de operare, extensia executabilului să fie specifică SO (e.g. Windows – „.exe”, MacOS – „.app”, Linux – „.deb” etc.).

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. https://stackoverflow.com/questions/17598572/read-and-write-to-binary-files-in-c
12. https://www.linuxquestions.org/questions/programming-9/tcp-file-transfer-in-c-with-socket-server-client-on-linux-help-with-code-4175413995/
13. StackOverflow, How to use setsockopt(SO_REUSEADDR)? https://stackoverflow.com/questions/24194961/how-do-i-use-setsockoptso-reuseaddr