NetMap

Stefan Cristea Olaru
stefancristea27@gmail.com

UAIC Faculty of Computer Science

1 Introducere

NetMap reprezintă un sistem format dintr-un server și mai mulți clienți care rulează operațiuni simultane, furnizând informații despre mașinile virtuale (VM-uri) active pe un dispozitiv. Acest sistem poate crea o hartă care ilustrează mașinile virtuale și conexiunile dintre acestea.
Obiectivul central al acestui proiect constă în elaborarea unei metode eficiente pentru obținerea informațiilor referitoare la mașinile virtuale (VM-uri) aflate în prezent pe un dispozitiv, precum și în implementarea unei hărți grafice care să evidențieze conexiunile și relațiile complexe între aceste VM-uri. Prin intermediul acestei metode, se urmărește furnizarea unei platforme robuste și ușor de utilizat pentru monitorizarea și vizualizarea detaliată a infrastructurii de virtualizare, facilitând astfel o înțelegere mai profundă a interacțiunilor dintre mașinile virtuale.

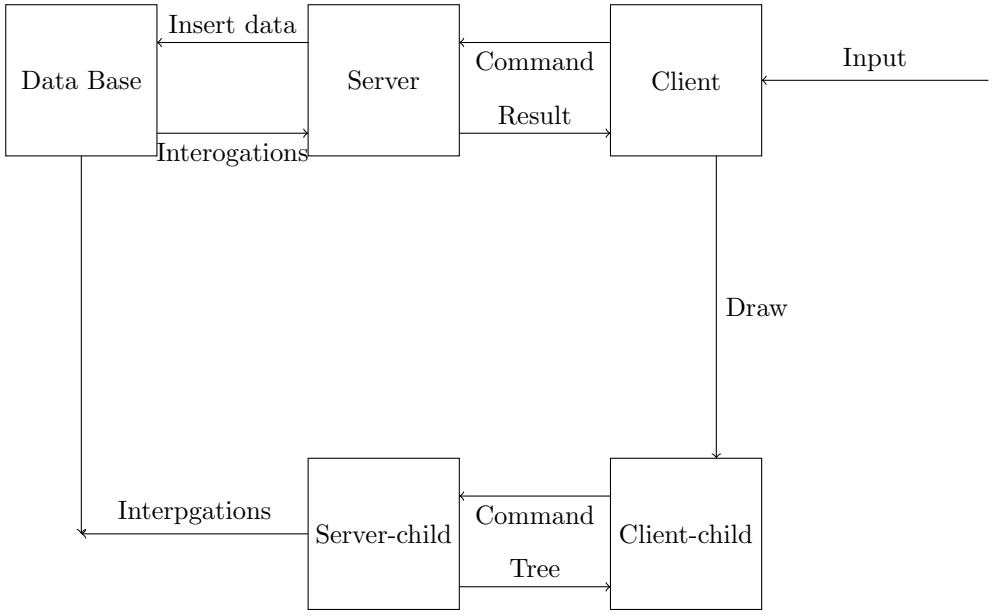
2 "Used Technology"

Protocolul de comunicare selectat între server și clienți este TCP. Alegerea acestui protocol este justificată de natura informațiilor transmise între client și server, și anume șiruri de caractere care reprezintă comenzi și parametri când sunt trimise de la client, precum și rezultatele executării acestor comenzi când sunt transmise de la server la client. Este esențial ca șirurile de caractere să ajungă intacte, iar protocolul TCP oferă această siguranță, asigurând transmiterea corectă și completă a datelor între cele două entități.
Serverul operează în mod concurent prin intermediul thread-urilor create la conectarea fiecărui client. Fiecare client este asociat cu un thread care așteaptă primirea unei comenzi sub forma unui șir de caractere din partea clientului. Odată ce șirul de caractere este primit, thread-ul asociat clientului execută comanda corespunzătoare și trimite rezultatul înapoi către client, tot sub forma unui șir de caractere.
Pentru a accesa informații despre mașina virtuală, am integrat biblioteca "libvirt/libvirt.h" în cadrul proiectului nostru. Am ales această bibliotecă datorită funcțiilor și structurilor sale, care sunt optimizate pentru a facilita conexiunea cu hipervizorul, interacțiunea cu o instanță specifică (domeniu), și obținerea detaliilor relevante despre acea instanță (VM), precum informații despre RAM, starea, nume, etc. Toate aceste informații sunt stocate într-o bază de date, accesibilă prin

intermediul aplicației noastre, și pentru implementarea acestei funcționalități am inclus suport pentru sqlite3.De asemenea am folosit SFML pentru partea grafica.

3 Structura Aplicatiei

Clientul trimite comenzi către server în formă de șiruri de caractere, iar serverul, după executarea funcției asociate, furnizează rezultatul înapoi către client. Datorită naturii concurente a serverului, mai mulți clienți pot executa comenzi simultan. Fiecare client este gestionat de un fir de execuție dedicat al serverului, asigurând astfel o execuție eficientă și concurentă a comenzilor provenite de la diverși clienți. Pentru a reprezenta grafic conexiunile, folosim un Copil-Client și un Server-Client, astfel încât să evităm interferența cu alte aspecte ale programului.



4 Aspecte de Implementare

4.1 Threadu-uri

[language=C++,caption=Thread Creation,label=code:example]

if ((client = accept (sd, (struct sockaddr *) from,(socklen_t*)length)) ¡ 0)

perror ("[server]Eroare la accept().");
continue;

```
    /* s-a realizat conexiunea, se astepta mesajul */

    // int idThread; //id-ul threadului
// int cl; //descriptorul intors de accept

    td=(struct thData*)malloc(sizeof(struct thData));
td→idThread=i++;
td→cl=client;

    pthread_create(th[i], NULL, treat, td);
```

Secvența de mai sus generează un fir de execuție (thread) pentru fiecare client în momentul conectării acestuia, utilizând funcția pthread_create".

4.2 Conectarea la o masina virtuala

```
[language=C++,caption=VM connection,label=code:example]
virConnectPtr con=virConnectOpen("qemu:///system");

    if(con==nullptr)///conexiunea nu s-a realizat

printf("[server]Failed to open connection");
return -1;
;

    virDomainPtr vm;///domeniu dupa IP/ID

    if(strcmp(Type,"ID")==0)///ne conectam prin ID

int domainID = std::stoi(Ident);
vm=virDomainLookupByID(con,domainID);
else
if(strcmp(Type,"IP")==0)///ne conectam prin IP

in_addraddr;
if(inet_pton(AF_INET, Ident, addr) == 1)
vm = virDomainLookupByUUIDString(con, Ident);
else
virConnectClose(con); printf("[server]Couldnotfind VM"); return − 1;
else
if(strcmp(Type, "name") == 0)///neconectamprinnume
vm = virDomainLookupByName(con, Ident);
```

if(vm==nullptr)

virConnectClose(con);
printf("[server]Could not find VM");
return -1;

Secventa de mai sus contine functii si structuri specifice librariei "libvirt/libvirt.h" ,precum:
virConnectPtr con=virConnectOpen("qemu:///system") ce este folosita pentru a face conexiunea cu hypervisor-ul
virDomainPtr vm; este folosit pentru a avea acces la o masina virtuala
Acest "vm" poate sa fie cautat dupa
ID : vm=virDomainLookupByID(con,domainID);
IP : vm=virDomainLookupByUUIDString(con,Ident);
Nume : vm=virDomainLookupByName(con,Ident);

4.3 TCP

[language=C++,caption=TCP,label=code:example]
if (bind (sd, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)

perror ("[server]Eroare la bind().");
return errno;

if (listen (sd, 2) == -1)

perror ("[server]Eroare la listen().");
return errno;

if ((client = accept (sd, (struct sockaddr *) &from,(socklen$_t$*)&$length$)) < 0)
$perror$("[$server$]$Eroare\ la\ accept$().n"); $continue$;

In secventa de mai sunt folosite functii specifice TCP-ului :
bind() : este folosit pentru a asocia socket-ul cu o adresa si un port
listen() : este folosit pentru a astepta clientii ce vor sa se conecteze
accept() : este folosit pentru a accepta un client , si returneaza un socket descriptor dedicat clientului

4.4 Inserarea VM-urilo in baza de date

[language=C++,caption=SQLite3]
sprintf(stmt,"INSERT INTO vm$_list(id_save, name, CPU_number, CPU_time, RAM, state, interface_nr, load)VA$
$lastInsertId, list_vm_info[i]- > name, list_vm_info[i]- > CPU_number, list_vm_info[i]- >$
$CPU_time, list_vm_info[i]- > RAM, list_vm_info[i]- > state,$
$list_vm_info[i]- > interface_nr, list_vm_info[i]- > load);$
$rc = sqlite3_exec(db, stmt, 0, 0, errMsg);$
$if(rc! = SQLITE_OK)$
$printf("Errorinsertingsave : %s", errMsg); sqlite3_free(errMsg);$
$///interfaces$
$for(intj = 0; j < list_vm_info[i]- > interface_nr; j + +)$
$bzero(stmt, 1024 * sizeof(char)); sprintf(stmt, "INSERTINTOvm_interface(id_save, name, interface, IP)V$

Un exemplu de inserare a datelor pentru o mașină virtuală folosind SQLite3.

4.5 Scenarii reale de utilizare

Aplicatia poate fi folosita pentru a obtine informatii despre masini virtuale ,cat si pentru vizualizarea conexiunilor dintre ele.

5 Concluzii

In viitor, aplicatia poate fi imbunatatia prin adaugare altor functii si comenzi pentru executarea anumitor comenzi pe o anumita masina virtuala .

6 Referinte Bibliografice

"Reference Manual for libvirt"`https://libvirt.org/html/index.html#main-libvirt-apis`
"Computer Networks (UAIC)" `https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php`
"SFML Documentation" `https://www.sfml-dev.org/documentation/2.6.1/`
"Handling multiple clients on server with multithreading using Socket Programming in C/C++"
`https://www.geeksforgeeks.org/`