**RPC - Remote procedure call between different systems**

****

Îndrumător: Realizat de:

Lt. Avram Dan Sd. Sg. Pantelimon Iuliana Larisa

Sd. Sg. Bordei Alin Viorel

Grupa **C113A**

**CUPRINS**

[1.1 Scopul proiectului: 3](#_Toc150460652)

[1.2 Lista definițiilor: 3](#_Toc150460653)

[1.3 Descrierea generală a aplicației 3](#_Toc150460654)

[1.4 Diagrama sistemului 4](#_Toc150460654)

[1.5 Modul de functionare 5](#_Toc150460654)

[2.1. Componentele sistemului: 6](#_Toc150460656)

2.2. Descrierea procesului administrator 6

[2.3. Descrierea proceselor worker: 6](#_Toc150460657)

2.4. User-ul si apelarea functiilor de catre acesta: 7

[3.1. Testarea conexiunii procesului worker la procesul administrator: 10](#_Toc150460663)

**Capitolul 1 –Introducere**

## Scopul proiectului:

Scopul acestui proiect este să implementeze și să faciliteze comunicarea între sistemele de calcul distribuite prin intermediul tehnologiei Remote Procedure Call (RPC). Prin utilizarea RPC, dorim să oferim utilizatorilor o modalitate eficientă și transparentă de a apela funcții sau proceduri situate la distanță, fără a fi nevoie să se preocupe de detalii complexe legate de comunicare și integrare între sisteme.

## Lista definițiilor:

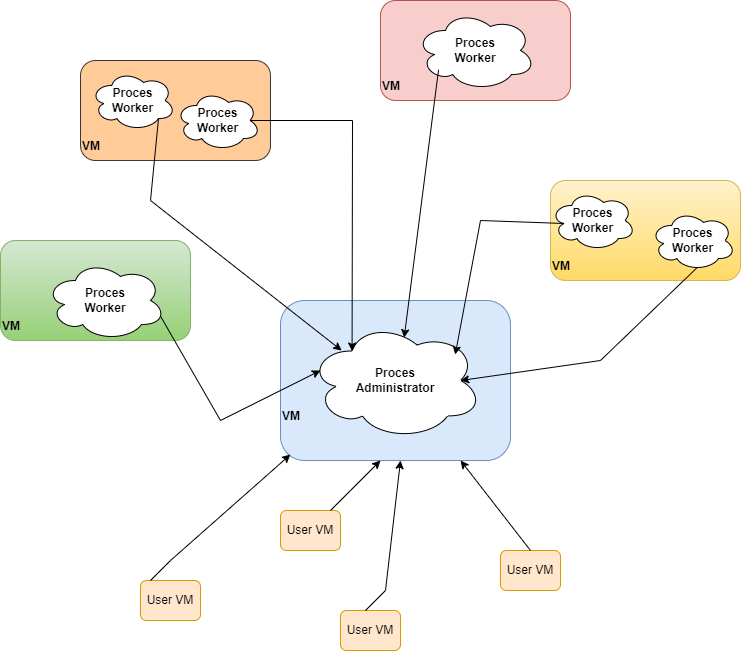
Remote Procedure Call (RPC) este o tehnologie de comunicare între procese care permite invocarea eficientă a funcțiilor sau procedurilor situate pe un alt sistem, precum și transferul de date asociat. Scopul principal al RPC este să ofere o metodă transparentă pentru apelarea funcțiilor sau procedurilor aflate la distanță, ca și cum acestea ar fi locale.

## Descrierea generală a aplicației

Aplicația permite dezvoltatorilor să definească funcții sau servicii pe un sistem, iar apoi să le apeleze de pe un alt sistem, ca și cum acestea ar fi locale. RPC se ocupă de detalii precum serializarea datelor, transmiterea acestora între sisteme și gestiunea conexiunilor de rețea, abstractizând aceste aspecte pentru a oferi o experiență de dezvoltare fluidă.

Prin implementarea acestei aplicații RPC, se urmărește simplificarea procesului de creare a aplicațiilor distribuite, reducerea complexității asociate comunicării între sisteme și creșterea eficienței dezvoltării. Aplicația oferă un cadru integrat pentru construirea unor sisteme distribuite scalabile, flexibile și ușor de gestionat.

* 1. **Diagrama sistemului:**



# 

# 1.5 Modul de functionare:

# 

# Capitolul 2– Descrierea modului de functionare

## Componentele sistemului:

Sistemul va fi alcatuit din doua tipuri de procese. Un proces administrator unic si mai multe procese de tip worker existente pe mai multe vm-uri. Acestea sunt fisiere sursa in limbajul C. Procesele vor fi lansate in executie prin comanda in terminal si linkate cu bibliotecile statice aferente functiilor pe care le vor executa.

* 1. **Descrierea procesului administrator:**

La nivelul acestui proces se va realiza legatura dintre user si functiile apelate. Acesta se va conecta la procesele worker printr-un socket si la useri prin alt socket. Va avea in componenta thread-uri: unul va asculta socket-ul constant pentru a primi request-urile de la useri pe care le memoreaza intr-o coada, alt thread va asculta socket-ul prin care procesele worker se conecteaza la procesul administrator si le va memora intr-o coada. De asemenea, vom avea un thread care se va ocupa cu trimiterea request-urilor catre procesele worker. Requst-urile vor fi trimise sub forma de pachet:

**IDProces | NumeFunctie | Parametri**.

Rezultatul primit va fi trimis catre utilizator.

Varianta **ASINCRONA**:

Cat timp se asteapta rezultatul cererii utilizatorul nu va ramane blocat, i se va trimite un semnal ca request-ul este gata si se va reconecta pentru a primi pachetul fina.

**Varianta SINCRONA:**

User-ul va ramane blocat pana la terminarea executiei schimbului de pachete dintre procese si returnarea rezultatului.

In cazul existentei mai multor procese worker cu aceeasi functie, pentru executarea request-ului utilizatorului procesul care va prelua request-ul este cel care a intrat primul in coada de procese.

## Descrierea proceselor worker:

## Conectarea proceselor worker se va realiza prin socket. Procesul administrator va sta tot timpul deschis in asteptarea unor noi conectari. Procesul worker este executat din terminal cu adresa IP a procesului administrator si numarul portului ca argumente. Odata rulat primul lucru pe care il face acesta este sa trimita un pachet care ca contine:

## IDProces | Numarul Functiilor | Nume\_Functii.

## Dupa aceea asteapta request-urile din partea administratorului pe care le va executa astfel: la nivelul procesului worker vor exista 2 thread-uri –unul pentru a asculta request-urile , celalat pentru a apela functiile si a returna rezultatul – rezultatul va fi trimis sub forma de pachet:

## Rezultat

Exemple de functii pe care le vor executa procesele worker:

* Functie hash
* Determinarea frecventei unui cuvant dat intr-un fisier
* Criptarea unuor date folosind RSA
* Decriptarea unor date folosind RSA
* Eliminare duplicate dintr-un fisier
* Codarea unui sir folosind Base64
  1. **User-ul si apelarea functiilor de catre acesta:**

Intetionam sa apelam procesul administrator folosind Stubs:

Stubs-urile sunt componente software care acționează ca intermediari între aplicația client și aplicația server într-un sistem distribuit. Ele sunt responsabile pentru împachetarea parametrilor apelului la o funcție și transmiterea acestora către procesul administrator, precum și pentru dezambalarea rezultatelor și returnarea acestora către user.

Atunci când user-ul apelează o funcție aflată pe unul dintre procese, stub-ul local preia controlul. Acesta împachetează parametrii apelului într-un format standardizat pentru transmiterea către procesul administrat.

Este alcatuit din urmatoarele functii:

connectToServer(con param):

Această funcție este responsabilă de stabilirea unei conexiuni cu serverul utilizând adresa IP și numărul de port specificate în obiectul param. Mai jos este o descriere mai detaliată a pașilor pe care îi urmează funcția:

Se creează un socket utilizând socket(AF\_INET, SOCK\_STREAM, 0). Se verifică dacă socketul a fost creat cu succes, și în caz contrar, se afișează un mesaj de eroare.

Se configurează structura serv\_addr pentru a specifica adresa IP și portul serverului.

Se încearcă conectarea la server utilizând connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)). În caz de eroare, se afișează un mesaj de eroare și se încheie programul.

Dacă conexiunea este stabilită cu succes, funcția returnează descriptorul de fișier al socketului.

callfunction(char \*payload, con param):

Această funcție trimite un payload (comandă) către server și primește un răspuns sincron. Mai jos sunt pașii implicați:

Se stabilește o conexiune la server utilizând connectToServer(param).

Se trimite payload-ul către server utilizând send(sockfd, buffer, strlen(buffer) + 20, 0).

Se primește răspunsul de la server utilizând recv(sockfd, recBuffer, 1024, 0).

Se închide conexiunea utilizând shutdown(sockfd, 2) și close(sockfd).

Funcția returnează răspunsul primit de la server.

callfunction\_noblock(char \*payload, con param):

Această funcție trimite un payload (comandă) către server și primește un ID asincron pentru referință. Iată pașii implicați:

Se stabilește o conexiune la server utilizând connectToServer(param).

Se generează un ID de referință utilizând rand() și srand(time(NULL)).

Se construiește un nou payload care include ID-ul generat și payload-ul original.

Se trimite noul payload către server utilizând send(sockfd, buffer, strlen(buffer) + 50, 0).

Se închide conexiunea utilizând shutdown(sockfd, 2) și close(sockfd).

Funcția returnează ID-ul generat pentru cerere.

StergereDuplicate(char \*text, con param) și StergereDuplicate\_noblock(char \*text, con param):

Aceste funcții trimit o comandă către server pentru a șterge duplicatelor dintr-un text. Iată pașii implicați:

Se construiește un payload care conține comanda și textul specificat.

Funcția callfunction() sau callfunction\_noblock() este apelată cu payload-ul și parametrii specificați.

Funcțiile returnează rezultatul primit de la server sau ID-ul cererii, în funcție de funcția folosită.

Base64(char \*text, con param) și Base64\_noblock(char \*text, con param):

Aceste funcții trimit un text către server pentru a efectua o conversie Base64. Iată pașii implicați:

Se construiește un payload care conține comanda și textul specificat.

Funcția callfunction() sau callfunction\_noblock() este apelată cu payload-ul și parametrii specificați.

Funcțiile returnează rezultatul primit de la server sau ID-ul cererii, în funcție de funcția folosită.

DeterminaFrecventa(char \*text, char \*cuv, con param) și DeterminaFrecventa\_noblock(char \*text, char \*cuv, con param):

Aceste funcții trimit un text și un cuvânt de căutat către server pentru a determinfrecvența cuvântului în text. Iată pașii implicați:

Se construiește un payload care conține comanda, textul și cuvântul specificate.

Funcția callfunction() sau callfunction\_noblock() este apelată cu payload-ul și parametrii specificați.

Funcțiile returnează rezultatul primit de la server sau ID-ul cererii, în funcție de funcția folosită.

Encrypt(char \*text, con param):

Această funcție trimite un text către server pentru a efectua o criptare. Iată pașii implicați:

Se construiește un payload care conține comanda și textul specificat.

Funcția callfunction() este apelată cu payload-ul și parametrii specificați.

Funcția returnează rezultatul primit de la server.

Hash(char \*text, con param) și Hash\_noblock(char \*text, con param):

Aceste funcții trimit un text către server pentru a calcula hash-ul. Iată pașii implicați:

Se construiește un payload care conține comanda și textul specificat.

Funcția callfunction() sau callfunction\_noblock() este apelată cu payload-ul și parametrii specificați.

Funcțiile returnează rezultatul primit de la server sau ID-ul cererii, în funcție de funcția folosită.

getRet(int id, con param):

Această funcție primește un ID de referință și returnează rezultatul asociat cu cererea respectivă. Iată pașii implicați:

Se construiește un payload care conține comanda și ID-ul specificat.

Funcția callfunction() este apelată cu payload-ul și parametrii specificați.

Funcția returnează rezultatul primit de la server.

Închiderea conexiunii:

După ce ați terminat de utilizat conexiunea, asigurați-vă că închideți socketul utilizând shutdown(sockfd, 2) și close(sockfd).

# Capitolul 3–Testarea sitemului

## 3.1. Testarea conexiunii procesului worker la procesul administrator:

## -am testat functia de hash

