# PSO proiect

# RPC- Remote procedure call

Documentație

Îndrumător: Realizat de:  
Slt. Vaman Adina Sd. sg. Simulescu Vlad  
Sd. sg. Crihană Dragoș  
Grupa C 113C

**Cuprins**

[1.Introducere 3](#_Toc156226968)

[**1.1 Definirea conceptului** 3](#_Toc156226969)

[**1.2** **Descriere generala a RPC** 3](#_Toc156226970)

[2. Arhitectura RPC 4](#_Toc156226971)

[**2.1 Componente principale** 4](#_Toc156226972)

[**2.2 Descrierea functionalitatii** 4](#_Toc156226973)

[**2.3 Functii** 5](#_Toc156226974)

[**2.3.1 base64()** 5](#_Toc156226975)

[**2.3.2 unbase64()** 6](#_Toc156226976)

[**2.3.3 hash\_djb2()** 6](#_Toc156226977)

[**2.3.4 ls()** 6](#_Toc156226978)

[**2.3.5 pwd()** 6](#_Toc156226979)

[**2.3.6 sum()** 6](#_Toc156226980)

[**2.3.7 shell()** 7](#_Toc156226981)

[**2.3.8 downloadFile()** 7](#_Toc156226982)

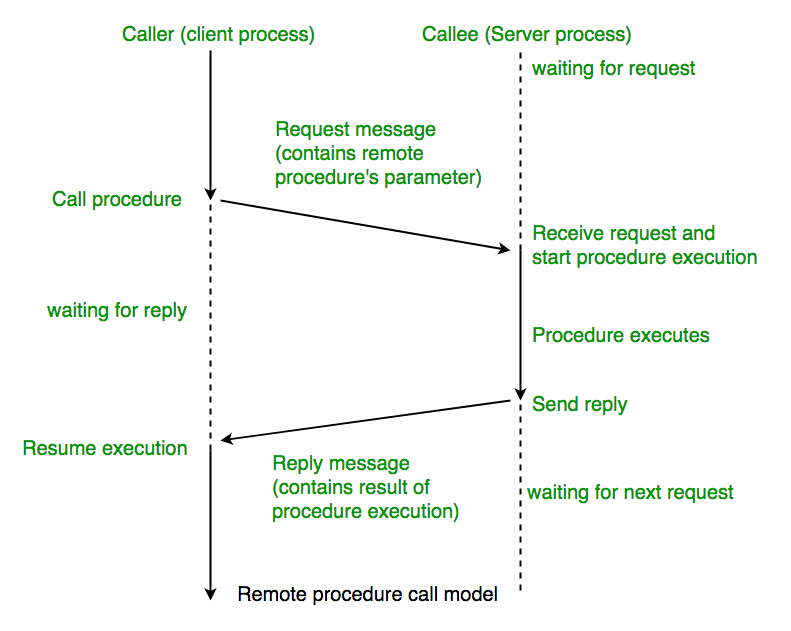
[**2.3.9 upload\_file()** 7](#_Toc156226983)

[3. Testare functionalitati 7](#_Toc156226984)

# 1.Introducere

## **1.1 Definirea conceptului**

* o aplicație RPC clasică va consta dintr-un client și un server. Aplicația client va executa prin intermediul serverului o procedură, serverul fiind cel care o execută și cel care trimite înapoi către client rezultatul.



## **1.2** **Descriere generala a RPC**

Acest proiect se va folosi pentru implementarea aplicației RPC de mai multe procese implementate pe VM-uri diferite: un proces administrator și alte procese client. Procesele client se vor ajuta de procesul server pentru realizarea anumitor funcții. Proiectul are drept scop posibilitatea de a crea un program care să se folosească de biblioteca creată de noi pentru a îndeplinii anumite funcționalități în acel program.

# 2. Arhitectura RPC

## **2.1 Componente principale**

Sistemul RPC este compus din doua componente principale:

* **Client**: Inițiază apelurile de functii la server.
* **Server**: Primește apelurile de functii de la clienți și execută acțiunile corespunzătoare.

## **2.2 Descrierea functionalitatii**

* **Serverul-** rulează pe un VM diferit de cel pe care rulează codul clientului. Acesta va avea la dispoziție un tabel de funcții pe care le poate îndeplinii. La pornirea procesului server se va deschide un socket pentru comunicare, care va aștepta request-uri de la clienți. Se va crea un thread principal care are drept scop ascultarea de cereri și distriburiea lor către thread-uri secundare spre executare. Tot la pornirea procesului pe lângă crearea unui thread principal vor mai fi create un anumit număr prestabilit de thread-uri secundare care vor avea ca scop executarea efectivă a funcțiilor. Fiecarui thread secundar i se va asigna o cerere primită de la un client. În momentul în care numărul de cereri depășește thread-urile libere, numărul acestora se va suplimenta. Acesta se va ocupa de următoarele:
  + thread-ul așteaptă primirea unei cerințe( sub formă de pachet)
  + deserializarea pachetului primit de la client
  + interpretarea cererii: se va verifica în primul rând dacă cererea formată dintr-o funcție și parametrii acesteia există în tabela cu funcții pe care serverul le poate executa și dacă parametrii primiți sunt corespunzători; în cazul în care se găsesc greșeli la aceasta, se va serializa un mesaj de eroare într-un pachet, acesta fiind trimis clientului
  + în cazul în care funcția există și parametrii primiți sunt valizi se va continua prin executarea ei și generarea unui rezultat
  + rezultatul va fi serializat într-un pachet și transmis către client
  + thread-ul se eliberează și așteaptă asignarea unei cerințe noi
* **Clientul-** codul clientului va avea posibilitatea apelării unei funcții prin intermediul unei biblioteci, aceasta ocupânduse de:
  + deschiderea unui socket pentru comunicarea cu serverul
  + serializarea funcției într-un pachet ce va fi transmis serverului
  + trimiterea pachetului către server
  + așteptarea pachetului ce conține răspunsul
  + deserializarea pachetului ce conține răspunsul
  + interpretarea răspunsului primit

Mesajele trimise de client catre server sunt impachetate cu ajutorul urmatorului protocol:

sum () x01firstNr,secondNr  
pwd() x02  
ls() x03firstArg,secArg  
base64() x04mesaj  
unbase64() x06mesaj  
hash\_djb2() x07mesaj  
downloadFile() x08  
upload\_file() x09  
commandExec() x10command

## **2.3 Functii**

* Funții pe care serverul să le poată îndeplinii:
  + Funcții de codificare și decodificare
  + Hash
  + Funcții matematice
  + Funcții pentru managementul fișierelor
  + Funcții pentru încărcare și descărcare de fișiere text
  + Funcții pentru simularea unui terminal

### **2.3.1 base64()**

Funcția **base64** primește un șir de caractere și îl convertește într-o reprezentare Base64. Codificarea Base64 este utilizată pentru a transforma octeții binari în caractere ASCII, oferind astfel o modalitate eficientă de reprezentare a datelor într-un format ușor de gestionat.

### **2.3.2 unbase64()**

Funcția **unbase64** primește un șir de caractere codificate în Base64 și îl decodează, returnând șirul de octeți original. Decodificarea Base64 este utilizată pentru a reveresa operația de codificare, transformând caracterele ASCII codificate înapoi în octeți binari.

### **2.3.3 hash\_djb2()**

Funcția **hash\_djb2** implementează algoritmul de dispersie DJB2 (Dan Bernstein) și este utilizată pentru a calcula un hash unic pe baza unui șir de caractere. Algoritmii de dispersie sunt adesea utilizați pentru a asigna rapid o cheie sau un identificator unui set de date, facilitând astfel căutarea sau indexarea acestora

### **2.3.4 ls()**

Funcția **ls** implementează o comandă simplă de listare a fișierelor și directoarelor într-un director specificat. Această funcție primește ca parametri un descriptor de socket și un șir de caractere care conține parametrii pentru comanda **ls**. Funcția utilizează **fork** pentru a crea un proces copil care va executa comanda **ls** și va redirecționa ieșirea standard și erorile standard către descriptorul de socket al clientului.

### **2.3.5 pwd()**

Funcția **pwd** implementează o comandă pentru a obține directorul de lucru curent al procesului. Această funcție utilizează **fork** pentru a crea un proces copil care va executa comanda **sh -c pwd** și va redirecționa ieșirea standard și erorile standard către descriptorul de socket al clientului.

### **2.3.6 sum()**

Funcția **sum** implementează o funcționalitate de adunare a două numere. Funcția primește un șir de caractere (**pars**) care conține două numere separate prin virgulă și întoarce un șir de caractere care reprezintă rezultatul adunării acestor două numere.

### **2.3.7 shell()**

Shell începe prin a citi linii de comandă de la utilizator și le parsează în argumente. Apoi, determină tipul comenzii (simplă, cu redirecționare sau de atribuire a variabilelor de mediu) și le execută în consecință, utilizând procese separate acolo unde este necesar. Variabilele de mediu sunt gestionate pentru a înlocui corespunzător valorile în linia de comandă.

Shell-ul continuă să aștepte noi comenzi după fiecare execuție și se ocupă de procesarea lor în mod similar.

### **2.3.8 downloadFile()**

Funcția **downloadFile** realizează descărcarea in directorul ./download a unui fișier de la server.

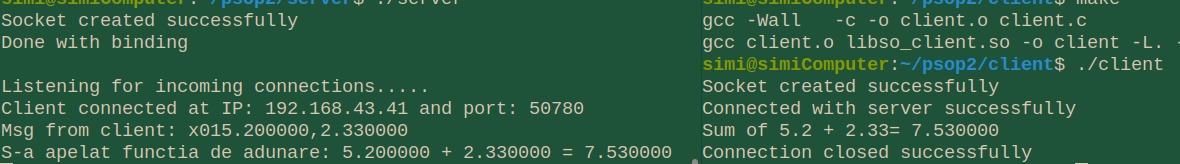
### **2.3.9 upload\_file()**

Funcția **upload\_file** realizează încărcarea unui fișier pe server.

# 3. Testare functionalitati

**Test 1:**

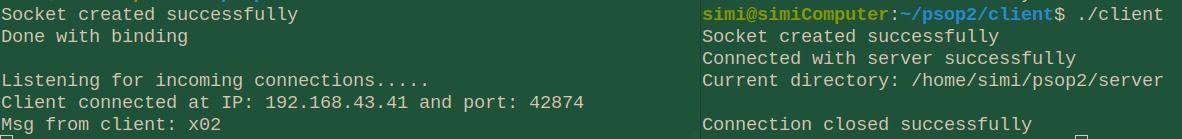
Input:

Output:

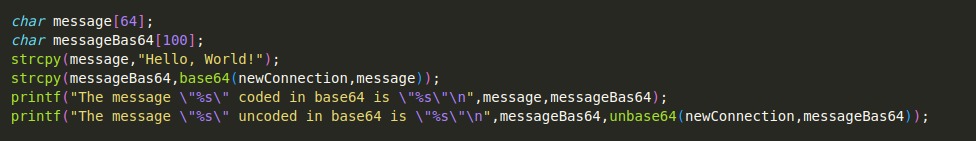
**Test 2:**

Input:

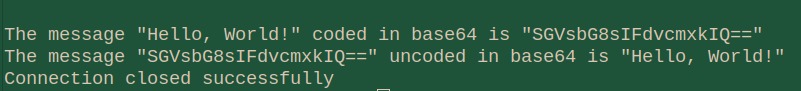


Output:

**Test 3:**

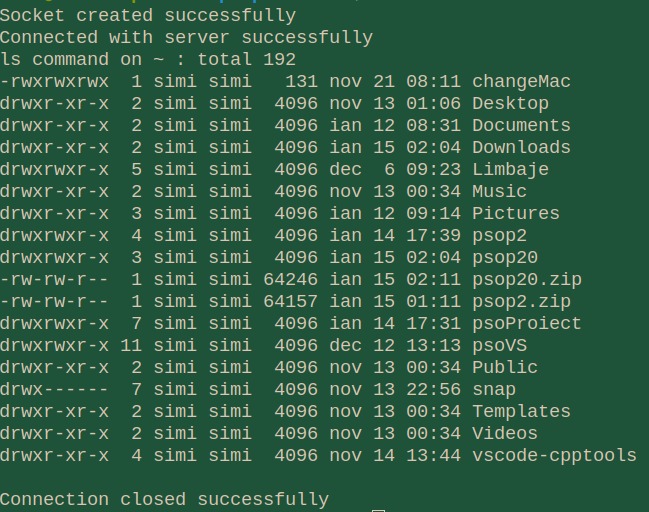
Input:

Output:

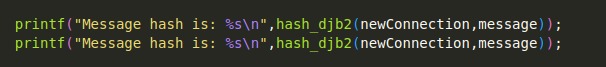


**Test 4:**

Input:

Output:

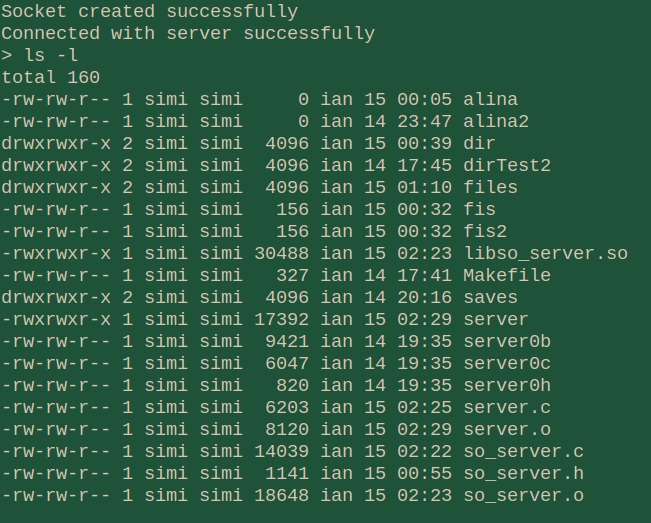
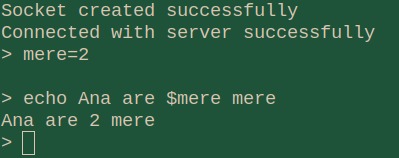
Test 5:

Input:

Output:

Test 6:

Input:

Output: