BÀI 4.
REMOTE PROCEDURE CALL (RPC)

Nội dung

- RPC là gì?
- Kiến trúc của RPC
- Java RMI
- MS RPC
- …

1. MÔ HÌNH RPC

3

Đặt vấn đề

- Làm thế nào để hai tiến trình client và server giao tiếp với nhau?
  - Sử dụng socket
  - Xây dựng giao thức hướng thông điệp (message-oriented protocol)

→Khó khăn?

  - Tổ chức thông điệp ở phía gửi như thế nào?
  - Xử lý thông điệp ở phía nhận
  - Xử lý trạng thái blocking
  - Không dễ dàng để triển khai
  - Khó có tính mở

→Làm cách nào client chỉ cần "gọi" các thủ tục của server giống như lời gọi trên tiến trình cục bộ?

→Remote Procedure Call (RPC)

4

RPC là gì?

- Là cơ chế giao tiếp giữa 2 tiến trình
- Thực hiện lời gọi thủ tục trên tiến trình khác giống như lời gọi thủ tục trong một tiến trình cục bộ
- Được xây dựng thành framework
- Dễ dàng phát triển các giao thức mới, phần mềm sụn (middleware)

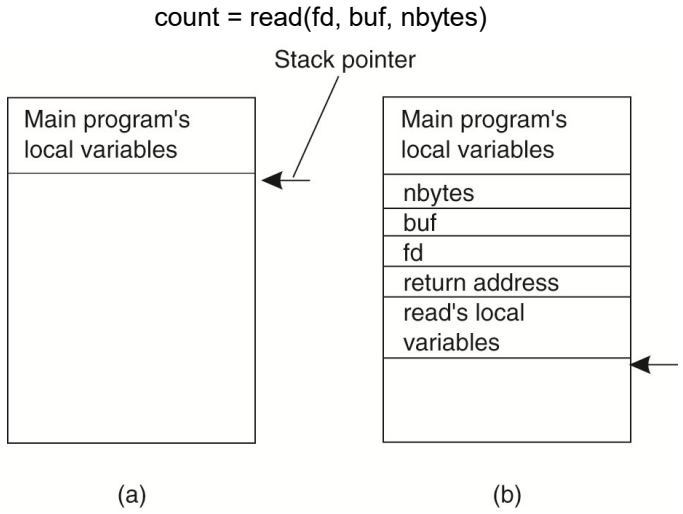
5

Nguyên lý cơ bản của RPC

- Tiến trình server: cung cấp một giao diện (interface) cho thủ tục/hàm để client gọi:
  - Tương tự thư viện lập trình API
- Tiến trình client:
  - Gọi thủ tục/hàm
  - Dừng chờ kết quả trả về
- Trao đổi dữ liệu giữa client và server thông qua tham số, giá trị trả về của hàm
- Trong suốt:
  - Cấu trúc thông điệp
  - Phân tán dịch vụ
  - Kiến trúc của hệ thống từ xa

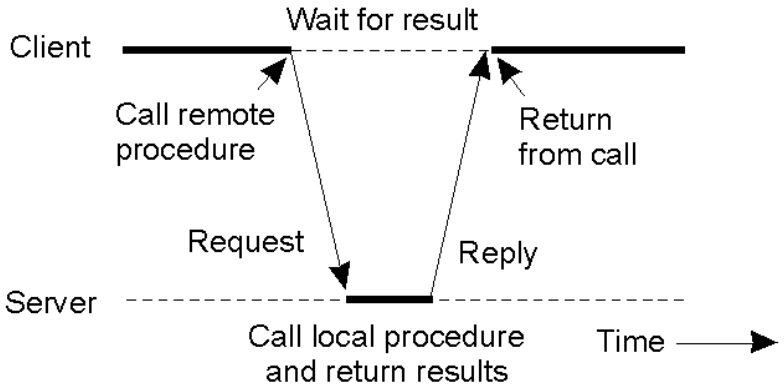
6

Lời gọi thủ tục trên tiến trình cục bộ



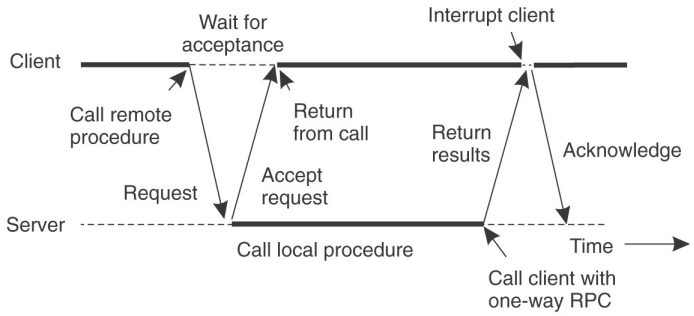
7

RPC



8

Asyncronous RPC



9

Lời gọi cục bộ và RPC

- Giả sử một thủ tục có khai báo nguyên mẫu:

```
bool verifyUser(char * userID, char *pass)
```

- Lời gọi cục bộ: trả về **`false`** hoặc **`true`**
- Sử dụng socket và giao thức hướng thông điệp: thực hiện như thế nào?
- RPC: trong suốt
  - Giá trị trả về: **`false`** hoặc **`true`**
  - ?
  - I don't know

10

Hạn chế của RPC

- Lời gọi cục bộ:
  - Chia sẻ/đồng bộ trạng thái: nếu thủ tục được gọi (callee) lỗi → thủ tục gọi (caller) lỗi
  - Call semantic luôn là exactly once
- RPC: không chia sẻ/đồng bộ trạng thái:
  - Lỗi chỉ xảy ra một phía
  - Lỗi trong quá trình truyền tin
- Các khả năng khi có lỗi:
  - Thủ tục không thực thi
  - Thủ tục thực thi 1 lần
  - Thủ tục thực thi nhiều lần
  - Thủ tục thực thi một phần

→ Giải quyết?

11

Giải quyết

- Các thao tác:
  - Gửi lại thông điệp yêu cầu
  - Kiểm tra lặp thông điệp
  - Truyền lại thông điệp đáp ứng
- “at least once”:
  - Client: gửi lại thông điệp yêu cầu cho tới khi nhận được đáp ứng
  - Server: thực thi lại thủ tục mỗi khi có thông điệp yêu cầu, stateless
  - Áp dụng trong trường hợp nào?

12

“at most once”

- Client: gửi lại thông điệp yêu cầu cho tới khi nhận được đáp ứng
- Server: chỉ xử lý yêu cầu 1 lần, truyền lại kết quả khi cần:
  - Phát hiện lặp thông điệp
  - Giữ câu trả lời trong bộ đệm
- Đảm bảo server nếu thực thi thủ tục thì chỉ thực thi 1 lần
- Yêu cầu: nếu có lỗi thì thủ tục không thực thi bất cứ phần nào

13

Thực hiện “at most once”

- Mất thông điệp yêu cầu
- Mất thông điệp trả lời
- Thời gian thực thi thủ tục quá dài
- Server lỗi sau khi thực thi và thông điệp bị mất
- Server lỗi khi thực thi

14

Các cơ chế của RPC

- Bằng cách nào client biết thủ tục nào cần gọi và danh sách tham số?
- Bằng cách nào client truyền lời gọi thủ tục tới server thông qua hệ thống mạng
- Server xử lý yêu cầu như thế nào?
  - Cổng ứng dụng nhận được yêu cầu
  - Xác định thủ tục được gọi
  - Xác định tham số truyền
- Client xử lý thông điệp trả lời như thế nào?

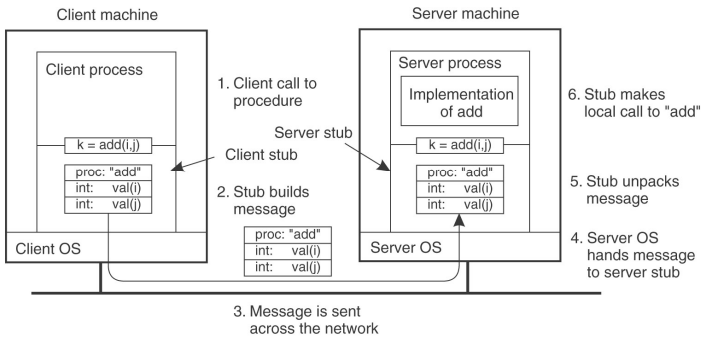
→Xây dựng thành phần stub ở 2 phía

→Thành phần gọi/được gọi: skeleton

15

Stub

- Client-side stub: thủ tục được client coi như là thủ tục được gọi
- Server-side stub: thủ tục được server coi như là thủ tục gọi



16

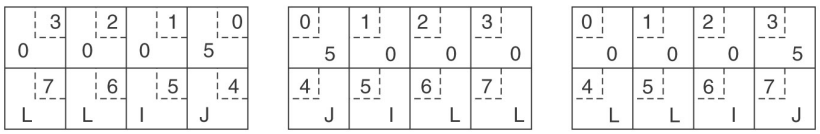
Marshalling và unmarshalling

- Marshalling: đóng gói dữ liệu vào thông điệp để gửi đi
- Unmarshalling: mở thông điệp nhận được và đọc dữ liệu
- Các vấn đề khi marshalling và unmarshalling:
  - Tham trị: Khác nhau về biểu diễn dữ liệu giữa máy cục bộ và máy từ xa
  - Tham biến/Tham chiếu: Không chia sẻ không gian địa chỉ bộ nhớ

17

Không đồng nhất biểu diễn dữ liệu

- Big-edian và little edian



(a) Pentium

(b) SPARC

(c) Chuyển đổi

- Biểu diễn số thực dấu phảy động
- Bảng mã ký tự

18

Truyền tham số theo kiểu tham chiếu

- Cấm sử dụng các tham chiếu

Hoặc

- Copy/Restore
  - Call by value: stub gửi đóng gói dữ liệu được trỏ bởi tham chiếu vào thông điệp gửi đi
  - Call by result: stub gửi cung cấp buffer chứa kết quả thực hiện
  - Call by value-result
  - Rất phức tạp với các kiểu dữ liệu có cấu trúc

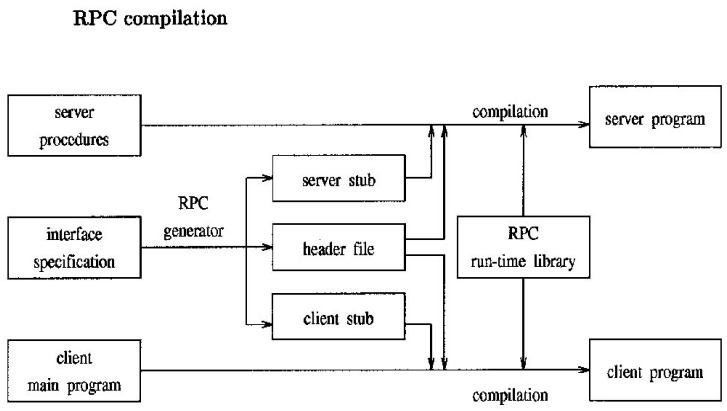
19

Interface Definition Languages

- Ngôn ngữ được sử dụng để định nghĩa giao diện ghép nối giữa client và server trong RPC
- Cần trình biên dịch riêng. Kết quả biên dịch:
  - File mã nguồn (C/C++/Java…)
  - Mã nguồn sử dụng để marshal các kiểu dữ liệu thành chuỗi byte
  - Stub routine

20

Quá trình biên dịch



21

Liên kết client-server

- Để thực hiện được lời gọi thủ tục RPC, cần thực hiện quá trình kết nối client tới server
- Server: export giao diện:
  - Định danh của thủ tục
  - Thông báo tới RPC runtime trạng thái sẵn sàng
- Client: import giao diện
  - RPC runtime thực hiện tìm kiếm thủ tục và thiết lập kết nối giữa client và server
- Được thực hiện trong suốt

22

2. MỘT SỐ RPC FRAMEWORK

2.1. SUN ONC RPC

Sun ONC RPC

- Sun ONC RPC(v1: RFC 1057, v2: RFC 5531)
- Được SUN thiết kế dành cho các hệ điều hành Unix, Linux, BSD, OS X
- Có thể sử dụng một số Toolkit để sử dụng trên nền tảng Windows
- Ngôn ngữ IDL: XDR(External Data Representation)
- Biên dịch file định nghĩa: **rpcgen**
- Sản phẩm tương tự: DCE RPC, Microsoft DCOM

25

Sun ONC RPC

- Bước 1: Tạo file định nghĩa với ngôn ngữ XDR(.x)
- Bước 2: Biên dịch file .x.

$rpcgen –a –C .x

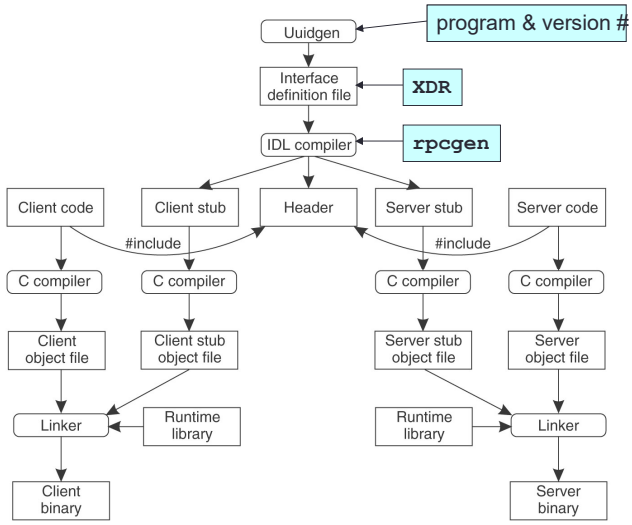
Các file thu được sau khi biên dịch:

- _clnt.c: client stub
- _svc.c: server stub
- _xdr: file XDR để marshalling
- .h: tệp tiêu đề định nghĩa các kiểu dữ liệu XDR
- Makefile.: file chỉ thị biên dịch
- _client.c -- client skeleton, chứa mã nguồn của ứng dụng client
- _server.c – server skeleton, chứa mã nguồn của ứng dụng server

- Bước 3: biên dịch thành chương trình client và server

26

Sun ONC RPC



27

File định nghĩa (.x)

- UUID: định danh cho thủ tục
  - 0 - 1fffffff : sử dụng bởi Sun
  - 20000000 - 3fffffff: sử dụng bởi người dùng
  - 40000000 - 5fffffff: sử dụng tạm thời
  - 40000000 – ffffffff: dự phòng

```
/* Định nghĩa kiểu dữ liệu của tham số
struct parameters {
    //Danh sách tham số
};
program APP_NAME {
    version APP_VERS {
        long PROCEDURE_1(parameters) = 1;
        string PROCEDURE_2(parameter) = 2;
    } = 1;   /* version */
} = UUID;
```

28

Sun ONC RPC – Kết nối

- Server đăng ký **`portmapper`** với HĐH:
  - Tên chương trình, phiên bản
  - Cổng dịch vụ
- Client: gọi hàm **clnt_create()**
  - Trả về: client handle để gọi thủ tục trên server

```
CLIENT *clnt_create(
       char *host,          //Địa chỉ server
       unsigned long prog, //Định danh chương trình
       unsigned long vers, //Phiên bản
       char *proto          //Giao thức tầng giao vận
);
```

29

2.2. JAVA RMI

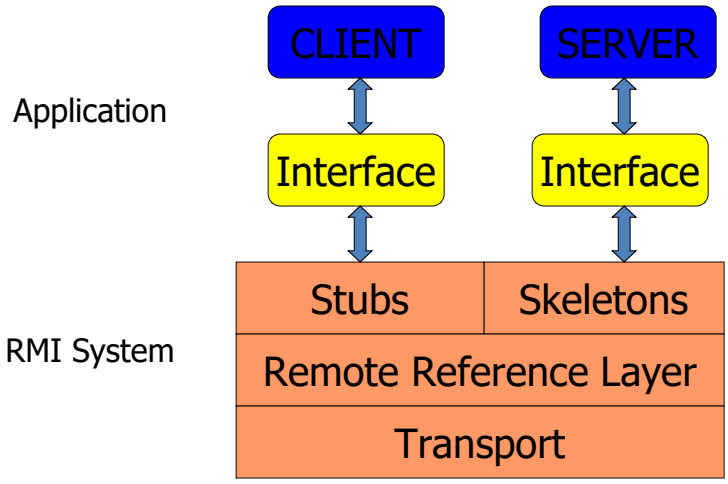
30

Java RMI

- Remote Method Invocation
- Ngôn ngữ Java không có cơ chế cho phương thức gọi từ xa
- 1995: Sun bổ sung và mở rộng thêm : Remote Method Invocation (RMI)
- RMI hoặc phương thức triệu gọi từ xa, là một middleware dựa trên Java.
- Cho phép phương thức của đối tượng Java ở một máy ảo Java (JVM) có thể triệu gọi từ một JVM khác
- Lưu ý: Không nhầm lẫn với cơ chế vào ra đối tượng Serializable

31

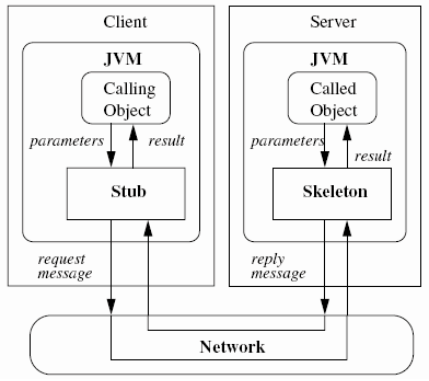
Kiến trúc Java RMI



32

Java RMI

- Mô hình hoạt động của phương thức triệu gọi phương thức từ xa trên RMI



33

Một số lớp thường dùng trong Java RMI

- Các gói java.rmi và java.rmi.Server
- Các lớp thường dùng là:
  - java.rmi.Naming
  - java.rmi.RMISecurityManager
  - java.rmi.RemoteException
  - java.rmi.Server.RemoteObject
  - java.rmi.Remote
  - java.rmi.Server.UnicastRemoteObject

34

Xây dựng ứng dụng với Java RMI

- Bước 1: Tạo giao diện của các phương thức cho phép gọi từ xa
- Bước 2: Định nghĩa các lớp triển khai
- Bước 3: Sử dụng công cụ **`rmic`** để dịch các lớp thành stub và skeleton
- Bước 4: Viết mã nguồn client và server
- Bước 5: Sử dụng công cụ **`rmiregistry`** khởi động dịch vụ đăng ký phương thức triệu gọi từ xa
- Bước 6: Thực thi client và server

35

2.3. CORBA

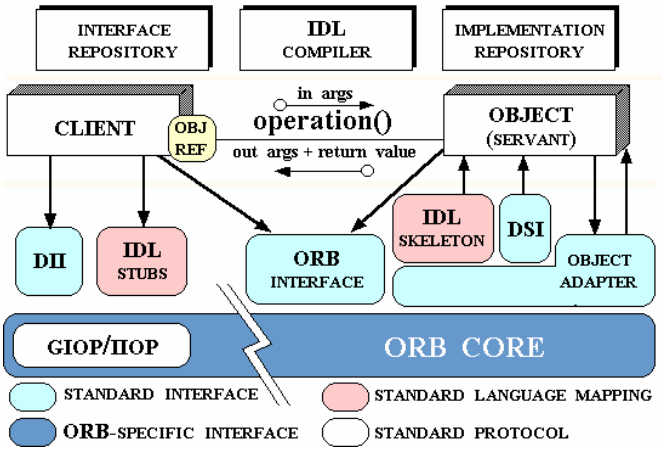
36

CORBA (Common Object Request Architecture)

- Cho phép các phần riêng biệt của ứng dụng viết bằng các ngôn ngữ khác nhau, chạy trên các máy khác nhau có thể phối hợp hoạt động như một ứng dụng duy nhất
- Phát triển bởi tổ chức OMG (Object Management Group)
- Hỗ trợ truy cập các đối tượng từ xa được phát triển trong nhiều ngôn ngữ trên một loạt các nền tảng.
- Cốt lõi của CORBA là ORB

37

Kiến trúc CORBA



38

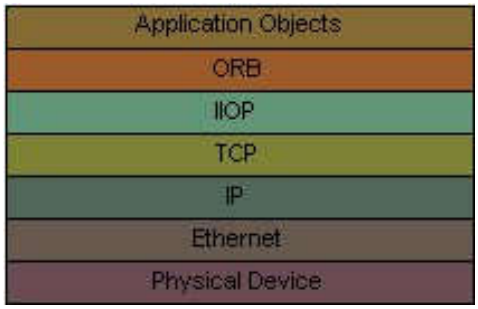
Kiến trúc CORBA: IDL

- IDL (Interface Definition Language)
- Xem như ngôn ngữ lập trình trung gian dùng chuẩn hóa các ngôn ngữ : C, C + +, Java, Ada, COBOL, Smalltalk, Objective C, LISP, Python
- Kiểu dữ liệu :
  - Kiểu cơ bản : long, short, string, float, …
  - Kiểu cấu trúc : struct, union, enum, sequence
  - Kiểu đối tượng tham chiếu
  - Các kiểu khác : Kiểu giá trị động
- Biên dịch đến ngôn ngữ đích
- Phát sinh thủ tục trên stub

39

Kiến trúc CORBA: GIOP/IIOP

- General Inter-ORB Protocol
- Internet Inter-ORB Protocol
- Hỗ trợ các dịch vụ khác nhau trên tầng giao giao vận, bao gồm: chuyển đổi dữ liệu, quản lý bộ nhớ đệm, quản lý dead clock, quản lý truyền thông.



40

Một số framework RPC cho Web

- XML RPC
- SOAP
- Web Services và WSDL(VD: Apache CFX)
- Microsoft .NET Remoting
- .NET Web Services
- Web Service
- AJAX
- REST

41