University of Pittsburgh
Computer Science Department

Course : CS - 2510 – Advanced Computer Operating Systems
Term : Fall 2016
Meetings : TTh 11:00 a.m. -12:15 p.m.
Classroom : 5313 Sennott Square
Instructor : Dr. Taieb Znati
Phone : (412) 624-8417

Project 1 – Simple Remote Procedure Call

I. Introduction

Remote Procedure Call (RPC) is a linguistic approach based on a fundamental concept, procedure call. RPC allows distributed programs to be written in the same conventional style as for centralized computer systems. The idea is based on the observation that in the client-server interaction, the client sends a request and then blocks until the remote server sends a reply. This behavior is similar to the classical interaction between main programs and procedures.

The development and implementation of a remote procedure call package requires addressing issues related to *parameter passing*, **binding, exception handling, call semantics, performance** and ***data representation***. The purpose of this project is to design and implement a **Simple Remote Procedure Call** (SRPC) package.

II. SRPC Model

The SRPC consists of a **Stub_Gen**(), a **Port Mapper**, a **Client Stub** and a **Server Stub**. The Stub_Gen(), a simple compiler that compiles the definition of a remote procedure interface, described in a specification file, and generates the **Client Stub** and the **Server Stub**. This procedure is illustrated in Figure 1.

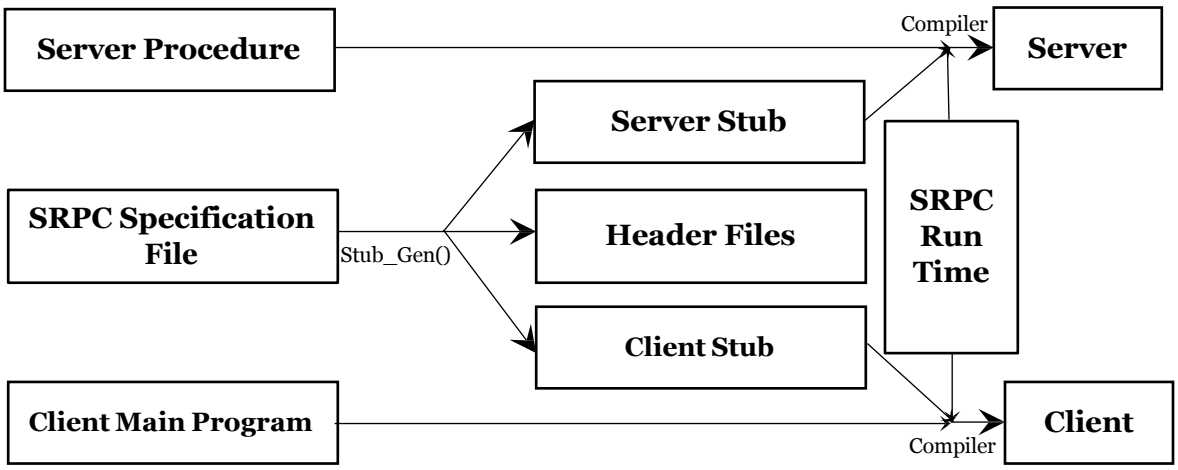

Figure 1- SRPC Execution Environment

The main function of the stubs is to carry the remote execution of the procedure and provide the user with the result. The mapper maintains a table of the names and locations of all currently exportable services, together with the version of the interface. Notice that the same service may run in several computers and we refer to these as *instances* of the service. The SRPC specification file has the following format:

program PROG_Name {

version PROG_Version {

type proc1 (param1 **type**, ..., param$_k$ **type,** …)=1;

type proc2 (param1 **type**, ..., param$_k$ **type,** …)=2;

…

}=1

}=1

Each procedure declared in the specification file is assigned a unique identifier. Each program is assigned a version number and a program number. Procedure **0**, referred as the null procedure, is defined for each program version and used to verify reachability of the server.

III. SRPC Binding and Execution

The SRCP Server stub creates (UDP/TCP) socket and binds any local port to the socket. The Server then calls a function in the SRPC library, **register_service**(), to register its program numbers and versions with the port mapper. The mapper is started as a daemon at "system" startup. The port mapper stores the program number and program version. Upon confirmation

of the binding operation, the server moves to wait for client requests. The binding process is described in Figure 2. Notice that servers can withdraw their own instances of services by notifying the port mapper.

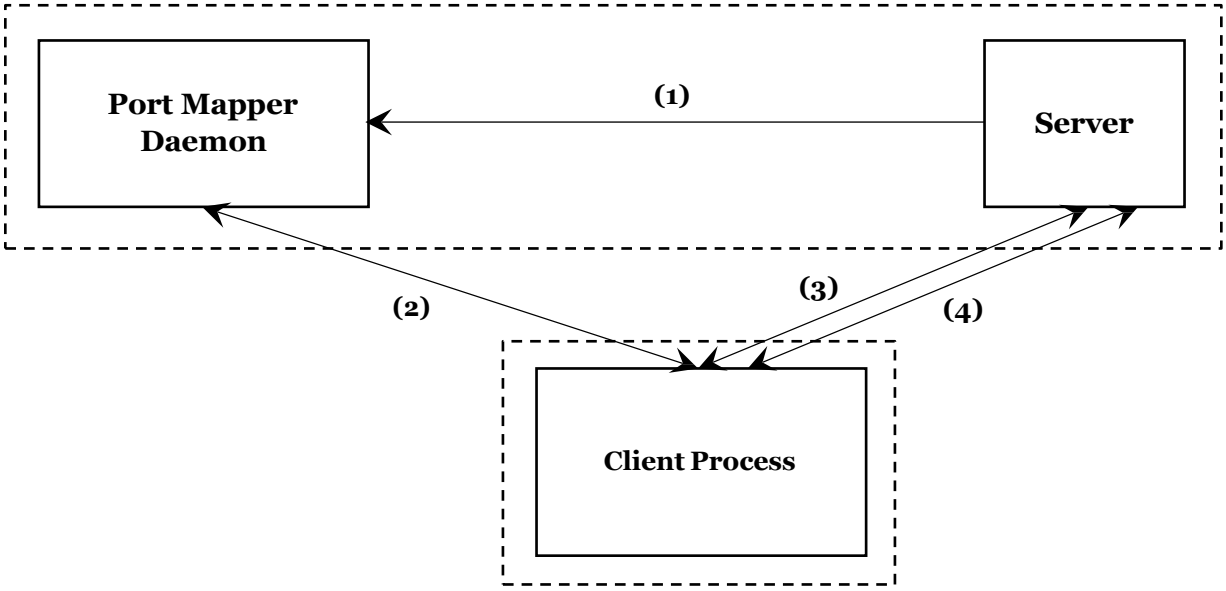

Figure 2- SRCP Binding and Execution

The client invokes procedures proc subi (arg1, ..., argn). This function must be defined in the SRCP client stub. The client also specifies the program number, the version number and the protocol (TCP/UDP). This call invokes the client stub procedure. The SRCP client stub contacts the port mapper to collect TCP/UDP port number and IP address of of the server.

The client stub places an identification of the target procedure into the message and **marshals** the arguments of the procedure to be executed. It then invokes transport layer routines to transmit the message to the server. Stub Marshalling involves **packing** the parameters into a network buffer on one end and unpacking them on the other. The assembling of the parameters must be done in a form suitable for transmission and understood by both stubs. Upon receiving the request, the server stub **unpacks** the parameters from the received message, identifies the procedure to be executed and makes a "normal" call to invoke the procedure, passing the parameters in the standard way.

Upon completion of the procedure execution, the server-stub packs the results into a message and invokes the server transport routines to transmit the message containing the results to the client stub. The client stub unpacks the results and returns them to the client user.

VI. Call Semantics

Every client request carries a unique transaction ID. The ID is initialized randomly. The Client functions test for matching of transaction ID before reporting results. The server keeps a cache indexed by (***ID***, ***prog #***, ***version #***) and containing responses. A cache lookup takes place before execution. If the request was previously executed, the server returns the response.

V. Implementation Requirement

As part of this project, you are required to develop:

1. The **RPC stubs**, both on the client and server sides. Your implementation should include:
    - On the client side, **marshaling** the parameters into a message. This includes converting the representation of the parameters into a "**standard format**", as defined by your implementation, and copying each parameter into the message. The client stub must also support **demarshaling** the return parameters, before the execution returns to the caller.
    - On the server side, handling of message requests by the stub, including demarshaling the parameters, executing the desired procedure, marshalling the return values into a message, and transmitting the message to client stub.
2. Development of a **TCP/IP socket-based communication protocol** to support message exchange between client and the server, including different message types and their associated semantics. The communication protocol should support the RPC semantics, described in this project.
3. Development of a **port mapper** to support dynamic binding, which allows the client to determine who to call and where the service resides. Dynamic binding allows the client to find the server at run time when the RPC is first made. The first time the client stub is invoked, it contacts the port mapper to determine the transport address at which the server resides.
4. Development of the matrix library procedures, including the null procedure.
5. The implementation of a dispatcher at a server side to support multiple client calls to different procedures of the scientific library. The dispatcher uses remote procedure number and program version to identify to forward each incoming procedure call to an appropriate stub.

VI. What you should submit

- **Part 1** – In this part, you are required to describe the model that you intend to implement for the SRPC mechanism. As part of the report, you are expected to

discuss the design and implementation issues in detail, and justify any design decisions you have made. The report should include a modular decomposition of each component of the system, a description of the interface between each pair of components, and a justification of your design choices. Careful thought must be given to the design to allow for new functionalities to be added to the system. More specifically, assuming the existence of communication primitives between remote hosts, you are to:

- Describe the main components of the SRPC package, namely the Stub_gen(), the port mapper, the client and server stubs, including the marshaling process and communication protocol for message exchange.
- Describe and show by a schematic diagram the sequence of events that occurs when a remote procedure call is invoked.
- Identify and describe the procedures used in the implementation of the SRPC model. A high level description of the procedure functions is sufficient.
- Explicitly, describe the mechanism used for the interaction between the different components of the model.
- Describe the general format of the Request/Reply messages exchanged among the remote hosts.

- **Part 2** – In this part, you are required to implement the SRPC mechanism using Linux sockets in the INET domain to program the communication primitives between the client and the server. Enclose a well-documented listing of the source code. The server must support a simple scientific library comprising:
  - **Matrix multiplication**: **Multiply**(A[n,m], B[m,l], C[n,l]),
  - **Vector sorting**: **Sort**(v_i , $0 \leq i \leq n-1$),
  - **Compute minimum**: **Min**(x_i , $0 \leq i \leq n-1$), and
  - **Compute maximum**: **Max**(x_i , $0 \leq i \leq n-1$).

VII. Important Dates

- **Part-1 Due Date** – Monday, Thursday, **October 17**, 2016. The report will be read and either approved or recommended for changes, promptly after its submission. A grade will also be assigned based on the efficiency and clear description of the implementation of the SRPC mechanism.

- **Final Submission Due Date** – The final report is due Wednesday, **October 31** 2016. All students are required to schedule an appointment for a demonstration of their projects. A spreadsheet with available time slots will be distributed.

The slots will be served FIFO, and collisions will be broken randomly. As part of the final report, you are expected to discuss the program design and implementation in detail, and justify the design decisions you have made. In a directory **accessible by both the instructor and the TA**, provide a well-documented **user manual**, a **make file**, if possible, and the **source code**.