Remote Procedure Call

Complex and Special RPCs

&

A Case Study on Sun RPC

1

Remote Procedure Call

Complex and Special RPCs

2

Complex RPC

- Complex RPC refers to a collection of unusual types of RPC calls
- For example, Long RPC is used in two different perspectives
  - RPCs Involving long-duration Calls
  - RPCs Involving long messages
- Another case of complex RPC is when there's a long gap between two successive calls

16 July 2024

3

Long duration RPC

- There are two broad approaches to handle such RPC calls
  - Periodic probing of the server by the client
  - Periodic generation of acknowledgment messages by the server

16 July 2024

4

1. Periodic Probing of Server

- After a client sends request to the server, it periodically sends probe packets to the server.
- Server is expected to acknowledge these probes.
- The message identifier of the original request message is included in each probe packet.

1. Periodic Probing of Server

- If the original request is lost, the server detects the same on receipt of a probe packet corresponding to that request message.
- The server then intimates the client that the request message corresponding to the probe packet has not been received.

1. Periodic Probing of Server

- On receipt of such a reply from the server, the client retransmits the original request.
- This allows the client to detect a server's crash or a link failure and hence to notify the corresponding user on an exception.

2. Periodic Acknowledgement

- If the server is not able to generate the next reply within an expected retransmission interval, then it spontaneously generates an acknowledgment.
- Thus, for a long-duration call, the server may have to generate several acknowledgments.

2. Periodic Acknowledgement

- If the client does not receive either the reply or an acknowledgment within a predetermined timeout period, it assumes that either the server has crashed or a link has failed.
- Thus, it notifies the concerned user on an exception condition.

16 July 2024

9

RPCs with Long Messages

- In some RPCs, the messages are too large to fit in a single packet.
  - As for example, in a file server, large volume of data may be transferred as input arguments to a write operation or as result of a read operation.
- Some RPC systems too are limited to small size packets.
  - The Sun Micro System‘s RPC is limited to 8 kilobytes per message.

16 July 2024

10

RPCs with Long Messages

- A way of handling RPCs of this category is to use multi-datagram messages.
- In this method, a long message is fragmented and transmitted in multiple packets.
- To improve performance, a single acknowledgment is used for all the packets of a message.

Callback RPC

- In callback RPC, a process may play the role of either a client or a server.
- In the usual RPC protocol, the caller and called processes have a client-server relationship.
- Unlike this, the callback RPC facilitates a peer-to-peer paradigm among the participating processes.

Callback RPC

- Consider a remotely initiated interactive applications that need input from user time to time or under special conditions.
- Say, the client process makes an RPC and during procedure execution, the server makes a callback RPC to the client process.

Callback RPC

- Now, client process takes necessary action and returns a reply for the callback RPC to the server process.
- On receiving this reply, the server resumes execution and finally replies to the initial RPC from the client.
- Server may even callback the client multiple times before returning the result of the initial call to the client.

Requirements for Callback RPC

- The ability for a server to call its client back requires the following:
  - Providing the server with the client's handle
  - Making the client process wait for the callback RPC
  - Handling callback deadlocks

Providing Client's Handle to Server

- The server must have the client's handle to call the client back.
- Typically, the client process uses a transient program number for the callback service
- A client exports the callback service by registering its program number with the binding agent.

Providing Client’s Handle to Server

- Next, the program number is sent as part of RPC request to the server.
- To make a callback RPC, the server initiates a normal RPC request to the client using given program number.
- Instead of having the client just send the server the program number, it could also send its handle, such as the port number.

16 July 2024

17

Providing Client’s Handle to Server

- The client's handle could then be used by the server to directly communicate with the client.

16 July 2024

18

Making the Client Process Wait

- The client process must be waiting for the callback so that it can process the incoming RPC request from the server
- It must be ensured that a callback RPC from the server is not mistaken as the reply of the RPC call made by the client process.

16 July 2024

19

Challenges for Transparency in RPC

- Inefficient Parameter Passing:
  - Unlike local procedure, a remote procedure is executed in an address space that is disjoint from the calling program's address space.
  - Hence, the remote procedure cannot have access to any variables or data values in the calling program's environment.
  - In absence of shared memory, it is meaningless to pass addresses in arguments, thus making call by reference useless.

16 July 2024

20

Challenges for Transparency in RPC

- Vulnerability:
  - RPCs are more vulnerable to failure than local procedure calls, as RPCs involve two different processes in disjoint address spaces and possibly a network between two different nodes.
  - The need for the ability to handle processor crashes and/or communication failures makes it even more difficult to obtain the same semantics for RPCs as for local procedure calls.

16 July 2024

21

Challenges for Transparency in RPC

- Poor Response Time:
  - RPCs consume significantly more time (100-1000 times more) than local procedure calls.
  - This is mainly due to the involvement of a communication network in RPCs.
  - Hence, the applications using RPCs must also have the capability to handle the long delays that may possibly occur due to network congestion.

16 July 2024

22

Handling Deadlock in RPC

- Since a process may play the role of either a client or a server, deadlocks can occur.
- Consider a process X making an RPC call to a process Y. X waits for a reply from Y.
- In the meantime, Y makes an RPC call to another process Z and waits for a reply from Z.

16 July 2024

23

Handling Deadlock in RPC

- Now, say, Z makes an RPC call to X. So, Z waits for a reply from X.
- X cannot reply to Z's call until its request to Y is replied, and Y cannot process X's request until its call to Z is satisfied which is blocked by X.
- So X, Y, and Z are in deadlock.
- In RPC, care must be taken to handle deadlock.

16 July 2024

24

Remote Procedure Call

A Case Study on Sun RPC

25

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

16 July 2024

26

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

16 July 2024

27

SUN RPC: Stub Generation

- Sun RPC supports automatic stub generation, although users have the flexibility of writing the manual stubs.
- An application's interface definition is written in an IDL called RPC Language (RPCL).
- RPCL is an extension of the Sun XDR language, originally designed to specify external data representation.

16 July 2024

28

SUN RPC: Interface Definition

```
/*  File StatelessFS.x */
    const FILE_NAME_SIZE = 16
    const BUFFER_SIZE = 1024

    typedef string FileName<FILE_NAME_SIZE>;
    typedef long Position;
    typedef long Nbytes;

    struct Data {
          long n;
          char buffer(BUFFER_SIZE];
    };
```

16 July 2024

29

SUN RPC: Interface Definition

```
struct readargs {
      FIleName
      Position
      Nbytes
};
struct writeargs {
      FileName
      Position
      Data
};
program STATELESS_FS_PROG {
version STATELESS_FS_VERS {
      Data          READ (readargs) = 1;
      Nbytes        WRITE (writeargs) = 2;
} = 1 ;
} = 0x20000000;
```

30

SUN RPC: Stub Generation

- An interface definition contains:
  - a program number (0x20000000 in our example)
  - a version number of the service (1 in our example),
  - the procedures supported by the service (READ and WRITE in our example),
  - the input and output parameters along with their types for each procedure, and
  - the supporting type definitions.

SUN RPC: Stub Generation

- Three numbers, program number (STATELESS_FS_PROG), version number (STATELESS_FS_ VERS), and procedure number (READ or WRITE) uniquely identify a RP.
- READ and WRITE procedures are numbered as 1 and 2, respectively.
- Extension of .x is used by interface definition files (e.g., StatelessFS.x).

SUN RPC: *rpcgen* Compiler

- The IDL compiler in SUN RPC is called *rpcgen*
- From a .x file, *rpcgen* produces the following:
  - Header files (with .h extension)
  - XDR filter files (with _xdr.c suffix)
  - A client stub file (with _clnt.c suffix)
  - A server stub file (with _svc.c suffix)

16 July 2024

33

SUN RPC: Header File by *rpcgen*

- The header file contains
  - definitions of common constants and types defined in the .x definition file.
  - external declarations for all XDR marshaling and un-marshaling procedures that are automatically generated.
- Name of a header file is formed by taking the name of input .x file and adding .h suffix (e.g. StatelessFS.h).

16 July 2024

34

SUN RPC: Header File by *rpcgen*

- This header file is to be manually included in client and server program files using #include preprocessor directive.
- The header file built by *rpcgen* compiler is automatically included in client stub, server stub, and XDR filters files.

SUN RPC: XDR filters by *rpcgen*

- The procedures in the XDR filter files built by *rpcgen* are used by the client and server stub procedures for encryption/decryption.
- The name of this file is formed by taking the name of the .x definition file and adding _xdr.c suffix (e.g., StatelessFS_xdr.c).

SUN RPC: Client Stub by *rpcgen*

- A client stub file contains one stub procedure for each procedure defined in the .x definition file.
- A client stub procedure name is the name of the procedure given in the interface definition, converted to lowercase appended with an underscore and the version number.

SUN RPC: Client Stub by *rpcgen*

- In our example, the client stub procedure names for READ and WRITE procedures will be read_1 and write_1 respectively.
- The name of the client stub file is formed by taking the name of the .x file to *rpcgen* and adding a _clnt.c suffix to it. (e.g., StatelessFS_clnt.c).

SUN RPC: Server Stub by *rpcgen*

- Server stub file contains the main routine, the dispatch routine, and one stub procedure for each procedure defined in the interface definition file plus a null procedure.
- The dispatch routine dispatches incoming calls to the appropriate procedure.

16 July 2024

39

SUN RPC: Server Stub by *rpcgen*

- The dispatch routine is named by taking the .x program name to lowercase characters and appending an underscore followed by version number (e.g., stateless_fs_prog_1) .
- The name of the server stub file is formed by taking the base name of the .x input file and adding _svc. c suffix to it (e.g., StatelessFS_svc.c).

16 July 2024

40

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

16 July 2024

41

Implementing SUN RPC applications

- Application programmer manually writes the client and server programs
- Client program is compiled to get a client object file.
- Server program is compiled to get a server object file.
- The client stub file and the XDR filters file are compiled and linked to get a client stub object file

16 July 2024

42

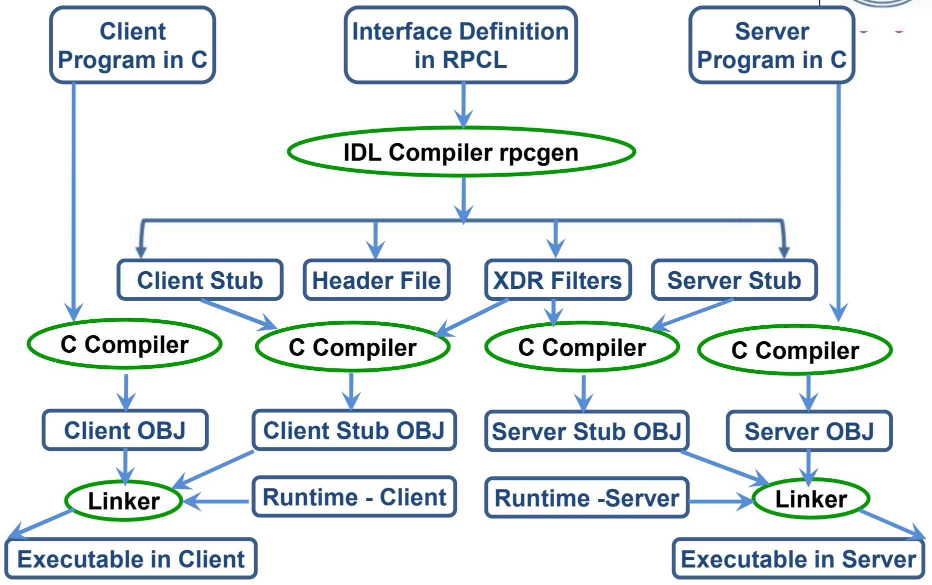
Implementing SUN RPC applications

- Server stub and XDR filter files are compiled and linked to get a server stub object file.
- Client object file, client stub object file, and client-side RPCRuntime library are linked together to get the client executable file.
- Server object file, server stub object file, and server-side RPCRuntime library are linked together to get the server executable file.

16 July 2024

43

Implementing SUN RPC Applications



44

SUN RPC: Implementing File Server

```
#include <Stdio.h>
#include <rpc.h>
#include "StatelessFS.h“

/* READ PROCEDURE •/
Data   *read_1 (args)
       readargs      •args;
{
   static Data          result;
   /* Statements for reading args.n bytes of data from
   the file args.filename starting from args.position, and
   for putting the data read in &result.buffer and the
   actual number of bytes read in result.n */
   Return (&result);
}
```

45

SUN RPC: Implementing File Server

```
/* WRITE PROCEDURE •/
Nbytes         •wnte_ 1 (args)
               writeargs •args;
{
   static Nbytes       result;
   /* Statements for writingargs.data.n bytes of data
   from the buffer &args.data.buffer into the file
   args.filename starting at position args.position */
   /* Statement for putting the actual number of bytes
   written in result */
   Return (&result);
}
```

46

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

16 July 2024

47

SUN RPC: Parameter Passing

- In Sun RPC, a called procedure can accept only one data argument and return only one result.
- Therefore, procedures requiring multiple parameters as input or as output must include them as components of a single structure.
- This is why structures *Data*, *readargs* and *writeargs* are defined.

16 July 2024

48

SUN RPC: Parameter Passing

- A Sun RPC call message has two parameters
  - the first is a pointer to the single argument of the remote procedure
  - the second is a pointer to a client handle.
    - read_result=read_1(&read_args, client_handle);
    - write_result=write_1(&write_args, client_handle);
- The return argument for a procedure is a pointer to the single result.

16 July 2024

49

SUN RPC: Parameter Passing

- The returned result must be declared as a static variable in the server program.
- Otherwise, the value of the returned result becomes undefined when the procedure returns.

16 July 2024

50

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

SUN RPC: Call Semantics

- Sun RPC supports at-least-once semantics.
- After sending a request message, the RPC Runtime library waits for a timeout period for the server to reply before retransmitting the request.
- The number of retries is the total time to wait divided by the timeout period.

SUN RPC: Call Semantics

- The default time to wait and the timeout period are 25 and 5 seconds respectively. These default values can be reset by the users.
- Eventually, if no reply is received from the server within the total time to wait, the RPC Runtime returns a timeout error.

16 July 2024

53

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

16 July 2024

54

SUN RPC: Client-Server Binding

- Sun RPC does not have a network-wide client-server binding.
- Each node has a local binding agent called port-mapper that maintains a database of mapping of all local services and their port numbers.
- The port-mapper runs at a well-known port number on every node.

SUN RPC: Client-Server Binding

- When a server starts up, it registers its program number, version number, and port number with the local port-mapper.
- When a client wants to do an RPC, it must first find out the port number of the server that supports the remote procedure.

SUN RPC: Client-Server Binding

- For this, the client makes a remote request to the port-mapper at the server's host, specifying the program number and version number.
- Thus, a client must specify host name of the server to import a service.
- In effect, this means that Sun RPC has no location transparency.

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

SUN RPC: Security

- SUN RPC supports three types of authentication models:
  - No Authentication: Default type
  - Unix Style Authentication: Restricted access to a service to a certain set of users based on user and group ids.
  - DES Style Authentication: User can access the service encrypted *with* DES

16 July 2024

59

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

16 July 2024

60

SUN RPC: Special Types of RPC

- SUN RPC supports some special types of RPCS including
  - Asynchronous RPC
  - Callback RPC
  - Broadcast RPC, and
  - Batch-mode RPC.

16 July 2024

61

SUN RPC: Special Types of RPC

- Asynchronous RPC is realized by setting the timeout value of an RPC to 0 and writing the server such that no reply is generated for the request.
- To facilitate callback RPC, the client registers the callback service using a transient program number with the local port-mapper.

16 July 2024

62

SUN RPC: Special Types of RPC

- The program number is then sent as part of the RPC request to the server.
- The server initiates a normal RPC request to the client using the given program number when it is ready to do the callback RPC.

16 July 2024

63

Case Study: SUN RPC

- Stub Generation
- Implementing SUN RPC applications
- Parameter Passing
- Call Semantics
- Client Server Binding
- Security
- Special Types of RPC
- Limitations of SUN RPC

16 July 2024

64

Case Study: Limitations of SUN RPC

- Sun RPC lacks location transparency as a client requires to specify the host name of the server when it imports a service interface.
- Sun RPC supports at-least-once or may-be call semantics, which may not be acceptable for many applications.

Case Study: Limitations of SUN RPC

- The IDL of Sun RPC does not allow a general specification of arguments and results.
- It allows only a single argument and a single result.
- This forces multiple arguments or return values to be packaged as a single structure.

Case Study: Limitations of SUN RPC

- Sun RPC is not transport independent and the transport protocol is limited to either UDP or TCP.
- In UDP, Sun RPC messages are limited to 8 kilobytes in length.
- Sun RPC does not have a network-wide client-server binding.

16 July 2024

67

Case Study: Limitations of SUN RPC

- A transport-independent version of Sun RPC, known as TI-RPC (transport-independent RPC), has been developed by Sun-Soft.
- TI-RPC provides a simple and consistent way in which transports can be dynamically selected depending upon user preference.

16 July 2024

68

Case Study: Limitations of SUN RPC

- We know that threads in client or server can improve performance of an RPC-based application.
- Sun RPC does not include any integrated facility for threads in the client or server, although Sun OS has a separate thread package.

16 July 2024

69

Thanks for your kind attention

Questions??

70