Implementing Remote Procedure Calls

Presented by Xingbo Gao
COP6614 Operating Systems Techniques
9/15/2005

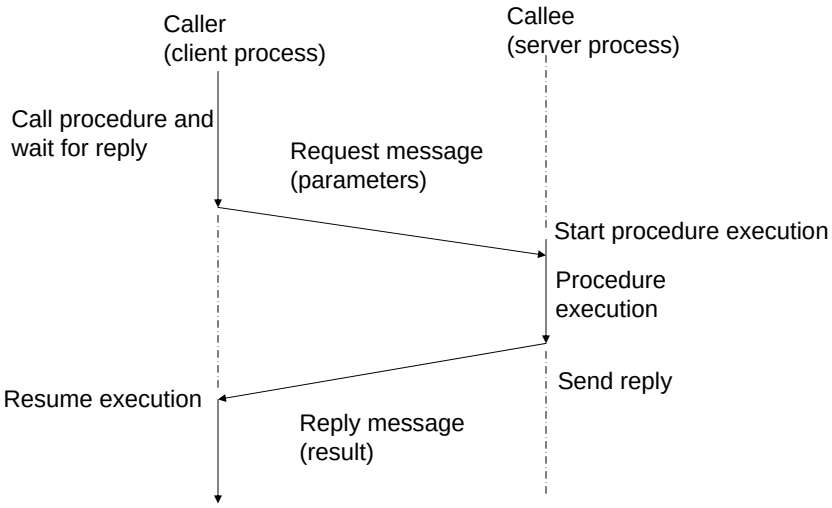
Outline

- RPC Introduction
- PRC Structure
- Client-Server Binding
- Communication Protocols
- Case Study: Sun RPC
- Conclusion

RPC Introduction
- History and Design Motivation

- Procedure calls are well-known and well-understood
- Message passing emerged in mid 80s
- Socket programming also appeared in the beginning of 80s in BSD 4.x
- Difficult to develop distributed applications
- 1981 Nelson's doctor dissertation extensively examined the design possibilities
- 1984 BIRRELL and NELSON developed the initial RPC packages at Xerox

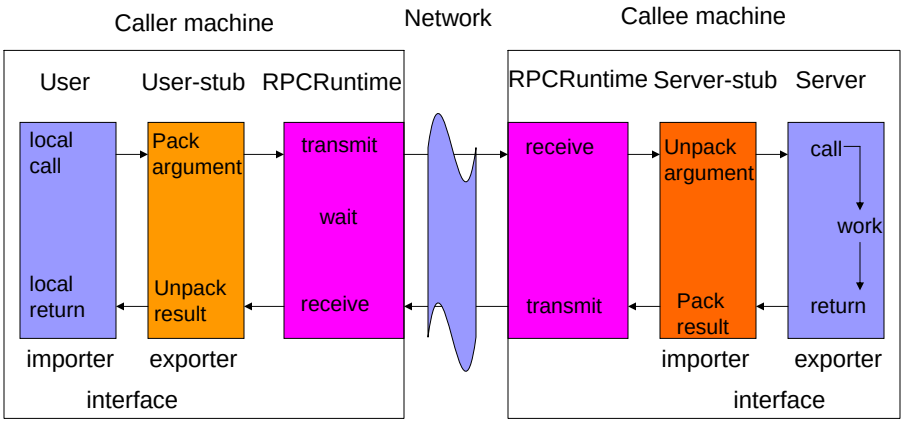
RPC Model



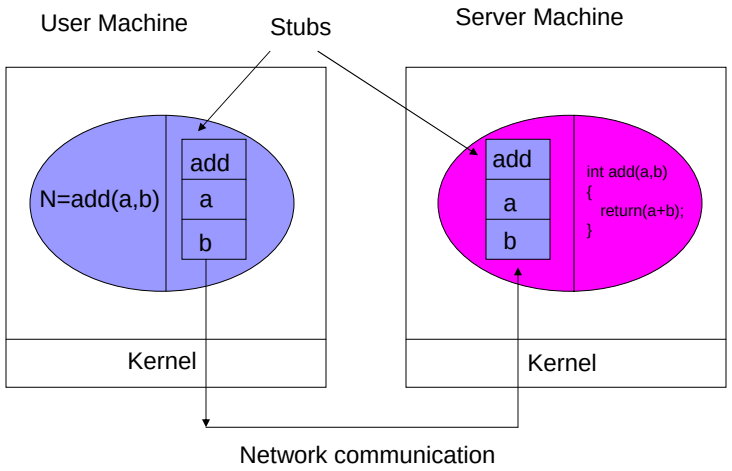
RPC advantages

- Simple call syntax
- Similar semantics to local procedure calls
- Well-defined interface
- Ease of use
- Generality
- Efficiency: simple and quick
- Works on different machines and a single machine

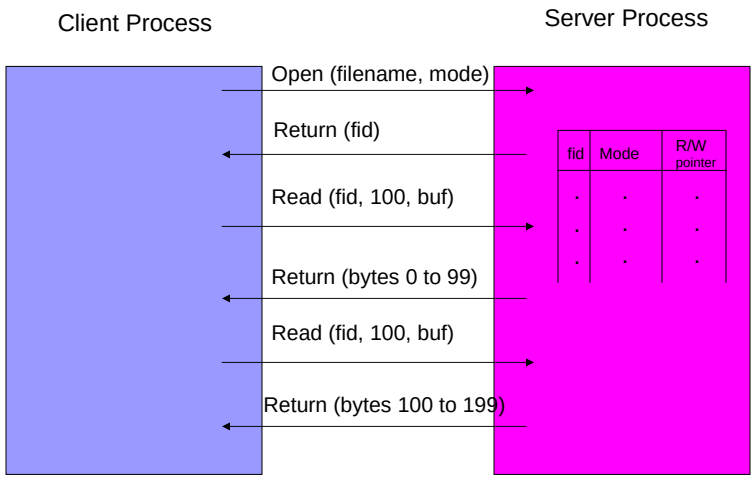
Components and Call interactions



Example One (remote computing)



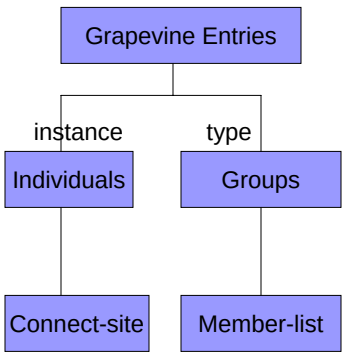
Example Two (stateful file server)



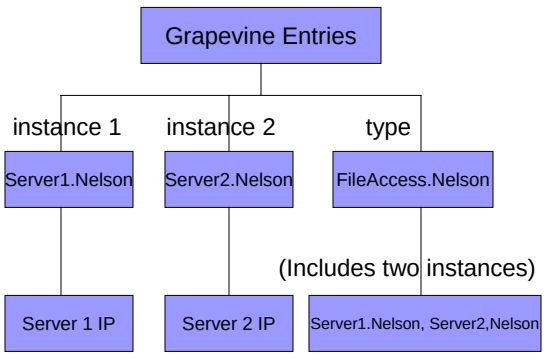
Client-Server Binding

- Naming: how does a client specify what he wants to be bound to?
- Location: how does a caller determine the machine address of the callee and specify to the callee the procedure to be invoked?

Client-Server Binding (contd)

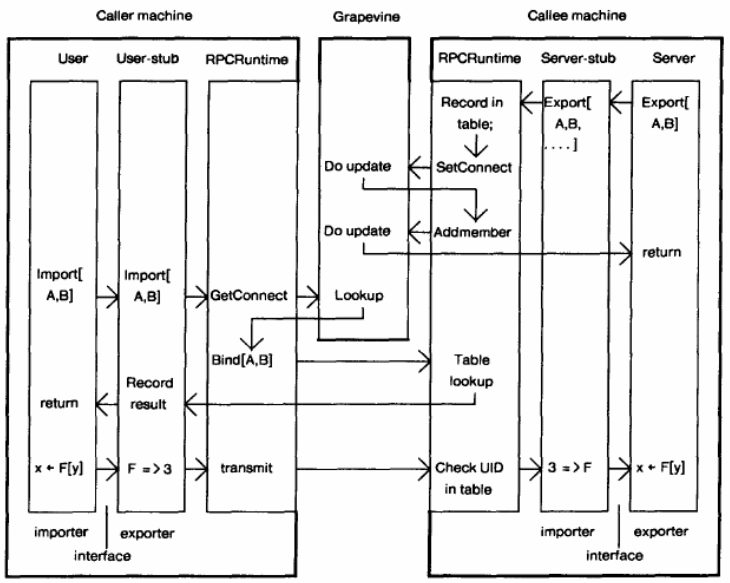


Grapevine Structure



Example: one type + two instances

Sequence of events in binding and a subsequent call



Binding Scheme Discussion

- Independent data structures in the exporting machine
- Unique identifier for each import/export interface
- Access controls on Grapevine database for two-way authentication
- Supported dynamic binding by specifying only the type of the interface and not its instance
- Allowed an importer to bind to multiple exporting machine: useful in some open-ended multimachine algorithms

Communication Protocol

- Aimed at minimizing the elapsed real-time for a call
- Strove to minimize the server load by substantial numbers of users
- Byte stream protocols (e.g. socket) were targeted at bulk data transfers, incurring a high cost for connection setup and tear-down
- Large data transferring protocols also required maintenance of substantial state information during a connection
- RPC send/reply messages were usually small (one or two packets), it needed a special-designed, efficient transport layer protocol

Simple Calls

- Call packet contains an identifier, desired procedure data and the arguments
- Result packet contains the same identifier and the results data
- A packet is retransmitted if no acknowledgement received; no new calls initiated until getting results back
- Call identifier containing calling machine identifier (address), calling process identifier and a sequence number could eliminate duplicate call packets

Simple Calls (contd)

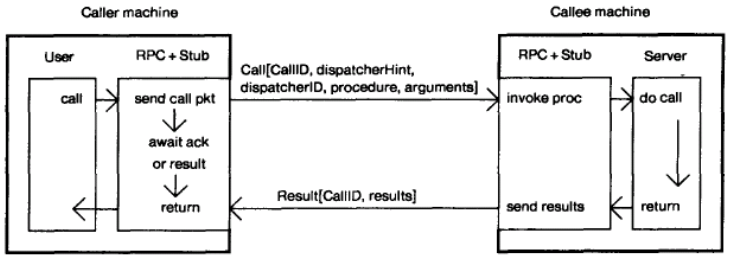
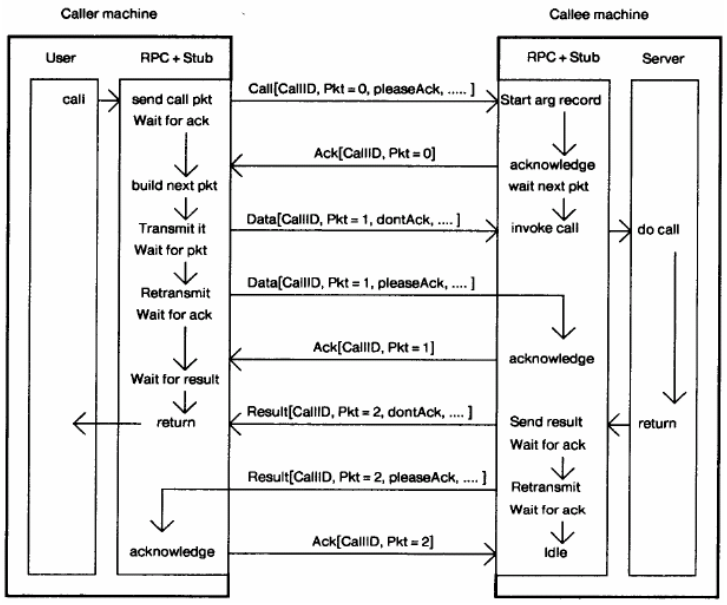


Fig. 3. The packets transmitted during a simple call.

Complicated Calls

- Explicit acknowledgements are used to handle lost packets, long duration calls and long intervals between calls
- Caller periodically sends *probe* packet expecting the callee to acknowledge
- Detects communication failures only, but couldn't detect deadlocked callee, keeping RPC semantics similar to local procedure call
- Large arguments or results are sent in multiple packets, with the last one requesting explicit acknowledgement

Complicated Calls (contd)



Exception Handling

- Exceptions called signals.
- An exception packet replaces a result packet if an exception happens
- RPCRuntime on the caller machine handles the exception packet and activates the catch phase if any
- Callee machine gets notified to resume or unwind the procedure activations

Optimizations

- Maintains a pool of idle server processes on the callee machines; excess server processes kill themselves upon completion
- The successive new calls would be dispatched to the server process that handled the previous call in the same calling activity
- In simple calls no processes created; typically only four process swaps in each call

Sun RPC case study

- Best known UNIX PRC
- Designed for Network File System (NFS)
- XDR – Interface Definition Language
  - Interface name: program number, version number
  - Procedure identifier: procedure number

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

program number, version number plus procedure number uniquely identify a remote procedure

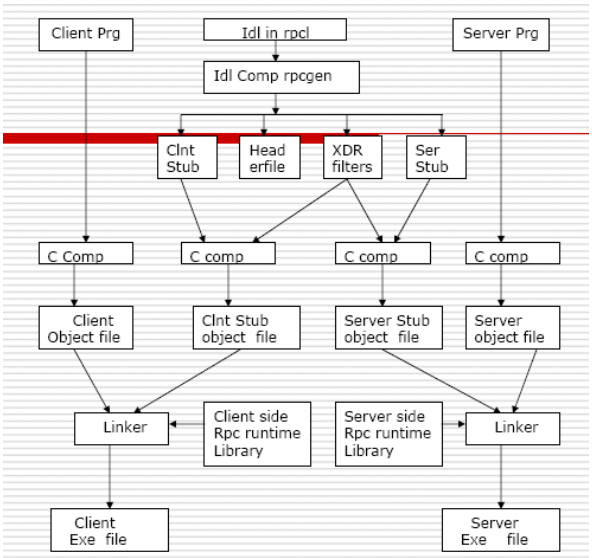
Sun RPC case study (contd)

- rpcgen – generator of RPC components
  - Client stub procedure
  - Server stub procedure
  - Server main procedure
  - Dispatcher
  - Marshalling and unmarshalling procedures: used by client and server stub procedures

An example using Sun RPC

1. Write add.x specification file for a remote procedure interface
2. Call rpcgen: *rpcgen add.x*
   - add_svc.c: server stub
   - add_clnt.c: client stub
   - add_xdr.c: xdr filters file: (un)marshalling
3. Create client executable:
   gcc –o client client.c add_xdr.c add_client.c –lrpcsvc –lnsl
4. Create server executable:
   gcc –o server server.c add_xdr.c add_svc.c –lrpcsvc –lnsl
5. Start server process: *server &*
6. Make a remote procedure call:
   client hostname 100 250
7. Or make a local procedure call:
   client localhost 100 250

The steps in creating an RPC application in Sun RPC



Sun RPC Binding

- Binding – *portmapper*
  - Server: register ((program number, version number), port number)
  - Client: request port number by (program number, version number)

Sun RPC Security

- Unix-style authentication
  - Each request contains the credentials of the user, e.g. uid and gid of the user
  - Access control according to the credential information
- DES-style authentication
  - Data Encryption Standard (DES) used for encryption
  - Referred as secure RPC

Conclusion

- Easy to write distributed application with RPC
- RPC is efficient and has low costs
- Simple and similar semantics to local procedure calls
- Secure and reliable

References

- Andrew D. Birrell, Brue Jay Nelson, *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, 1984
- Pradeep K Sinha, *Distributed Operating Systems Concepts and Design*, IEEE Press, 1997