Remote Procedure Calls

- Reference paper: “Implementing Remote Procedure Calls” in the paper
- Reference book chapter: Section 2.4 in “Distributed Operating Systems” by Andrew Tanenbaum

Communication Paradigms

- message passing
  - socket, bind, listen, accept, connect, read/write
- client/server
  - request/reply
  - naming
- remote procedure call

Remote Procedure Call

- Basic implementation and parameter marshalling
- Semantics in the presence of failure
- Implementation issues
- Performance issues

An Example

- count = read(fd, buf, bytes)
- what happens in normal procedure calls?
  - how does the stack change?
  - call-by-value or call-by-reference
- what happens in system calls?
- what should happen in RPCs?
  - client stub and server stub

Basic Steps

- client procedure calls the client stub in the normal way (doesn’t know it is a RPC stub)
- the client stub builds a message and traps to the kernel
- the kernel sends the message to the remote kernel
- the remote kernel gives the message to the server stub
- the server stub unpacks the parameters and calls the server procedure
- the server does the work and returns the result to the server stub
- the server stub packs it in a message and traps to the kernel
- the remote kernel sends the message to the client’s kernel
- the client’s kernel gives the message to the client stub
- the stub unpacks the result and returns to the caller

Parameter Marshalling

- simple example: sum(4,7)
- little endian and big endian
  - example: index(3, “abcd”)
  - solutions?
- data representation
  - what about arrays?
- network canonical forms
  - conversion and problems

Passing Pointers

- what are our choices?
  - no pointers or call by references
  - knowing the size of the array
  - knowing the type (input/output) of the pointer parameter and the size of the array
  - with DSM techniques
    - do you still want RPC if you have DSM?

Locating the Server

- Binding: use central registry server
- Each server registers “type+instance” of the service, lists all procedure calls supported
- Access control at the registry protects clients from rogue servers
- Server builds table including all exported RPC interfaces and interface ID
  - interface ID used to detect calls to old interfaces

RPC semantics of Failures

- client can’t locate the server
  - procedure returns error upon binding time
- request message lost
  - caller retransmits, callee uses request sequence number to identify duplicate requests
- reply message lost
  - callee retransmits, caller uses the corresponding request sequence number to identify duplicate replies

Server or Client Crashes

- Server crash
  - at least once
  - at most once
  - no guarantee
  - exactly once
- Client crash
  - how to kill orphan computation at server side?
    - client log RPC calls;
    - client asks server to kill orphans upon recovery;

Implementation Issues

- protocol choice
  - TCP or UDP?
  - UDP/IP or RPC specific protocol?
  - message size
- Birell&Nelson’s solution:
  - RPC specific protocol
  - using replies as acks of requests
  - client/caller probes server/callee periodically

Implementation

- acknowledgements
  - stop & wait versus blast
- flow control
  - network overrun
  - dealing with overruns
    - problems with chip
    - problems with limited buffering

Performance Issues

- where is time spent in RPC?

Performance Tuning

- copying overhead
  - best case: client memory ->network interface ->server memory
  - general case: client memory -> kernel buffer -> network interface->network buffer->kernel buffer->server address space
  - worst case: general case + copy of array to message + copy of array to stack + restore copy
  - copying speed: 500ns for 32-bit words
  - scatter-gather hardware and page remapping

Problem Areas for RPC

- global variables
  - example: errno
- variable size arrays
- pointer to complicated data structures
- pipes and standard input/outputs
  - client/server model not a good fit