Remote Procedure Call

Outline
Protocol Stack
Presentation Formatting

based on section 5.3 of Peterson & Davie’s book

RPC Timeline
Where RPC fits in OSI model

RPC Components

- Protocol Stack
  - fragments and reassembles large messages (BLAST)
  - synchronizes request and reply messages (CHAN)
  - dispatches request to the correct process (SELECT)

- Stubs

Programming Benefits

- The programming is easier since little or no network programming involved.
- If an unreliable protocol such as UDP is used, details like timeout and retransmission are handled by RPC package.
- The RPC library handles any required data translation for the arguments and return values.

Simple RPC Stack

TCP/IP Illustrated, Vol. I, Chap. 29
SunRPC

- IP implements BLAST-equivalent
  - except no selective retransmit

- SunRPC implements CHAN-equivalent
  - except not at-most-once

- UDP + SunRPC implement SELECT-equivalent
  - UDP dispatches to program (ports bound to programs)
  - SunRPC dispatches to procedure within program

SunRPC Header Format

- XID (transaction id) is similar to CHAN’s MID
- Server does not remember last XID it serviced
- Problem if client retransmits request while reply is in transit
Port Mapper

- Just another RPC program
- Listen to UDP port 111 and TCP port 111
- Provide server procedures:
  - `PMAPPROC_SET`: Called by an RPC server on startup to register a program no, version no, protocol with port no.
  - `PMAPPROC_UNSET`: Called by server to remove a previously registered mapping.
  - `PMAPPROC_GETPORT`: Called by an RPC client on start up to obtain the port no for a given program no, version no, and protocol.
  - `PMAPPROC_DUMP`: Returns all entries (program no, version no, protocol, and port no) in the port mapper database.

- Port mapper starts first (listen to 111) →
  RPC Server prog. starts (register with `PMAPPROC_SET`) →
  RPC Client prog. starts (`PMAPPROC_GETPORT`) →
  RPC Client sends an RPC call message

DCE-RPC

- Distributed Computing Environment (DCE) was defined by Open Software Foundation (OSF).
- DCE-RPC is the PRC protocol at the core of the DCE and CORBA (Common Object Request Broker Architecture).
- Run on top of UDP.
- Besides RPC, DCE also includes security services, LAN namespace services, and network time services. (Sun has a “Secure RPC” for authentication.)
Presentation Formatting

- Marshalling (encoding) application data into messages
- Unmarshalling (decoding) messages into application data
- Data types we consider
  - integers
  - floats
  - strings
  - arrays
  - structs
- Types of data we do not consider
  - images
  - video
  - multimedia documents

Difficulties

- Representation of base types
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian (e.g., 34,677,374)

```
Big-endian:
00000010 00010001 00100010 01111110
(2)        (17)       (34)         (126)

Little-endian:
01111110 00100010 00010001 00000010
(126)      (34)       (17)         (2)
```

- Compiler layout of structures
Taxonomy

• Data types
  – base types (e.g., ints, floats); must convert
  – flat types (e.g., structures, arrays); must pack
  – complex types (e.g., pointers); must linearize

• Conversion Strategy
  – canonical intermediate form
  – receiver-makes-right (an $N \times N$ solution)

Taxonomy (cont)

• Tagged versus untagged data

• Stubs
  – compiled
  – interpreted
eXternal Data Representation (XDR)

- Defined by Sun for use with SunRPC
- C type system (without function pointers)
- Canonical intermediate form
- Untagged (except array length)
- Compiled stubs

```c
#define MAXNAME 256;
#define MAXLIST 100;

struct item {
    int     count;
    char    name[MAXNAME];
    int     list[MAXLIST];
};

bool_t
xdr_item(XDR *xdrs, struct item *ptr)
{
    return(xdr_int(xdrs, &ptr->count) &&
            xdr_string(xdrs, &ptr->name, MAXNAME) &&
            xdr_array(xdrs, &ptr->list, &ptr->count,
                      MAXLIST, sizeof(int), xdr_int));
}
```

<table>
<thead>
<tr>
<th>Count</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>JOHNSON</td>
</tr>
<tr>
<td>4</td>
<td>497</td>
</tr>
<tr>
<td>9</td>
<td>852</td>
</tr>
<tr>
<td>7</td>
<td>205</td>
</tr>
</tbody>
</table>
**Abstract Syntax Notation One (ASN-1)**

- An ISO standard
- Essentially the C type system
- Canonical intermediate form
- Tagged
- Compiled or interpreted stubs
- BER: Basic Encoding Rules

\[(\text{tag, length, value})\]

<table>
<thead>
<tr>
<th>type</th>
<th>length</th>
<th>type</th>
<th>length</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>4</td>
<td>4-byte integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Network Data Representation (NDR)**

- Defined by DCE
- Essentially the C type system
- Receiver-makes-right (architecture tag)
- Individual data items untagged
- Compiled stubs from IDL
- 4-byte architecture tag

- IntegerRep
  - 0 = big-endian
  - 1 = little-endian
- CharRep
  - 0 = ASCII
  - 1 = EBCDIC
- FloatRep
  - 0 = IEEE 754
  - 1 = VAX
  - 2 = Cray
  - 3 = IBM

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntgrRep</td>
<td>CharRep</td>
<td>FloatRep</td>
<td>Extension 1</td>
<td>Extension 2</td>
<td></td>
</tr>
</tbody>
</table>