Remote Procedure Call

Outline
- Protocol Stack
- Presentation Formatting

RPC Timeline

RCP Components
- Protocol Stack
  - BLAST: fragments and reassembles large messages
  - CHAN: synchronizes request and reply messages
  - SELECT: dispatches request to the correct process
- Stubs
Bulk Transfer (BLAST)

- Unlike AAL and IP, tries to recover from lost fragments
- Strategy
  - selective retransmission
  - aka partial acknowledgements

BLAST Details

- Sender:
  - after sending all fragments, set timer DONE
  - if receive SRR, send missing fragments and reset DONE
  - if timer DONE expires, free fragments

BLAST Details (cont)

- Receiver:
  - when first fragments arrives, set timer LAST_FRAG
  - when all fragments present, reassemble and pass up
  - four exceptional conditions:
    - if last fragment arrives but message not complete
      - send SRR and set timer RETRY
    - if timer LAST_FRAG expires
      - send SRR and set timer RETRY
    - if timer RETRY expires for first or second time
      - send SRR and set timer RETRY
    - if timer RETRY expires a third time
      - give up and free partial message
BLAST Header Format

- MID must protect against wrap around
- TYPE = DATA or SRR
- NumFrag indicates number of fragments
- FragMask distinguishes among fragments
  - if Type=DATA, identifies this fragment
  - if Type=SRR, identifies missing fragments

Request/Reply (CHAN)

- Guarantees message delivery
- Synchronizes client with server
- Supports at-most-once semantics

Simple case

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Request</td>
</tr>
<tr>
<td>Ack</td>
<td>Ack</td>
</tr>
</tbody>
</table>

Implicit Acks

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request 1</td>
<td>Request 2</td>
</tr>
<tr>
<td>Ack 1</td>
<td>Ack 2</td>
</tr>
</tbody>
</table>

CHAN Details

- Lost message (request, reply, or ACK)
  - set RETRANSMIT timer
  - use message id (MID) field to distinguish
- Slow (long running) server
  - client periodically sends “are you alive” probe, or
  - server periodically sends “I’m alive” notice
- Want to support multiple outstanding calls
  - use channel id (CID) field to distinguish
- Machines crash and reboot
  - use boot id (BID) field to distinguish
### CHAN Header Format

```c
typedef struct {
    u_short Type; /* REQ, REP, ACK, PROBE */
    u_short CID; /* unique channel id */
    int MID; /* unique message id */
    int length; /* length of message */
    int probs; /* high-level protocol */
} ChanHdr;
```

```c
typedef struct {
    u_char type; /* CLIENT or SERVER */
    u_char status; /* BUSY or IDLE */
    int timeout; /* timeout value */
    int retry; /* number of retries */
    int return_error; /* return value */
    int request; /* request message */
    int reply; /* reply message */
    Semaphore reply_sem; /* client semaphore */
    int mid; /* message id */
    int bid; /* boot id */
} ChanStat;
```

### Synchronous vs Asynchronous Protocols

- **Asynchronous interface**
  ```c
  send(Protocol llp, Msg *message)
  deliver(Protocol llp, Msg *message)
  ```

- **Synchronous interface**
  ```c
  call(Protocol llp, Msg *request, Msg *reply)
  upcall(Protocol llp, Msg *request, Msg *reply)
  ```

- CHAN is a hybrid protocol
  - synchronous from above: `call`
  - asynchronous from below: `deliver`

### Dispatcher (SELECT)

- Dispatch to appropriate procedure
- Synchronous counterpart to UDP
- Implement concurrency (open multiple CHANs)

- Address Space for Procedures
  - flat: unique id for each possible procedure
  - hierarchical: program + procedure number

---

*Spring 2002 CS 461*
Simple RPC Stack

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
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)

<table>
<thead>
<tr>
<th>Base Type</th>
<th>Big-endian</th>
<th>Little-endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>100000010</td>
<td>01111110</td>
</tr>
<tr>
<td>17</td>
<td>000100001</td>
<td>000000001</td>
</tr>
<tr>
<td>34</td>
<td>00100010</td>
<td>126</td>
</tr>
<tr>
<td>126</td>
<td>01111110</td>
<td>2</td>
</tr>
</tbody>
</table>

- 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 x N solution)
Taxonomy (cont)

- Tagged versus untagged data

<table>
<thead>
<tr>
<th>Type</th>
<th>len</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4</td>
<td>417892</td>
</tr>
</tbody>
</table>

- Stubs
  - compiled
  - interpreted

 stub Marshaled arguments
  
  stub Marshaled arguments

  Call Client
  
  NPC
  
  Message

  Interface
  
  Description for Procedure P

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Specified</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Code</td>
<td>Client</td>
</tr>
<tr>
<td>Code</td>
<td>Code</td>
<td>RPC</td>
</tr>
<tr>
<td></td>
<td>compiled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RPC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

```
<table>
<thead>
<tr>
<th>Count</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Int</td>
</tr>
<tr>
<td>3</td>
<td>Item6</td>
</tr>
<tr>
<td>1</td>
<td>Item4</td>
</tr>
<tr>
<td>2</td>
<td>Item5</td>
</tr>
<tr>
<td>3</td>
<td>Item3</td>
</tr>
<tr>
<td>4</td>
<td>Item2</td>
</tr>
<tr>
<td>5</td>
<td>Item1</td>
</tr>
<tr>
<td>6</td>
<td>Item0</td>
</tr>
</tbody>
</table>

console.log(xdr_item(xdr, ptr));
```
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

(tag, length, value)

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

Example representation: