Remote Procedure Call Protocols
Local Procedure Calls

// Client code
...  
result = procedure(arg1, arg2);  
...

// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
...  
return(result);
}
Remote Procedure Calls (1)

// Client code
.
result = procedure(arg1, arg2);
.

// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
  .
  return(result);

  }


Remote Procedure Calls (2)

// Client code
... 
result = procedure(arg1, arg2);
... 

// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
... 
    return(result);
}
Block Diagram

Client

Application

Stub

RPC support code

Transport protocol

Server

Remote procedure

Stub

RPC support code

Transport protocol
ONC RPC

• Used with NFS
• eXternal Data Representation (XDR)
  – specification for how data is transmitted
  – language for specifying interfaces
Example

typedef struct {
    int    comp1;
    float  comp2[6];
    char   *annotation;
} value_t;

typedef struct {
    value_t    item;
    list_t     *next;
} list_t;

bool add(int key, value_t item);
bool remove(int key, value_t item);
list_t query(int key);
Placing a Call

```c
result = add(key, item);

bool add(int k, value_t v) {
    ...
    return(result);
}
```

marshall  

unmarshall

Wire
Returning From the Call

result = add(key, item);

char add(int k, value_t v) {
    ...
    return(result);
}

unmarshal

marshal

Wire
Marshalled Arguments

- int
- int
- float (1)
- float (2)
- float (3)
- float (4)
- float (5)
- float (6)
- string length
- string (1–4)
- string (5–8)

key
comp1
comp2
item
annotation
### Marshalled Linked List

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>array length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>value_t</td>
<td>next: 1</td>
</tr>
<tr>
<td>1</td>
<td>value_t</td>
<td>next: 2</td>
</tr>
<tr>
<td>2</td>
<td>value_t</td>
<td>next: 3</td>
</tr>
<tr>
<td>3</td>
<td>value_t</td>
<td>next: 4</td>
</tr>
<tr>
<td>4</td>
<td>value_t</td>
<td>next: 5</td>
</tr>
<tr>
<td>5</td>
<td>value_t</td>
<td>next: 6</td>
</tr>
<tr>
<td>6</td>
<td>value_t</td>
<td>next: -1</td>
</tr>
</tbody>
</table>
Reliability Explained …

• Exactly once semantics
  – each RPC request is handled exactly once on the server
RPC Exchanges

Client → Server: Request 1

Server → Client: Response 1

Client → Server: Request 2

Server → Client: Response 2
Uncertainty

Client → Request → Server

Client

Request

Client

Request

Server

Request

Server

Response
Idempotent Procedures

Client → Write File A Block 0 → Server

Client → (Ahem …) Write File A Block 0 → Server

Client → Done → Server

Client → Done → Server

At-least-once semantics
Non-Idempotent Procedures

Client \[\rightarrow\] Server
Transfer $1000 from Alice's Account to Bob's Account

Client \[\rightarrow\] Server
Done

Client \[\rightarrow\] Server
Do it again and again and again!

Client \[\rightarrow\] Server
Done

At-most-once semantics
Maintaining History

Client | Transfer $1000 from Alice’s Account to Bob’s | Server

Client | Done | Server

Client | Do it again! | Server

Client | Done (replay) | Server

At-most-once semantics
No History

Transfer $1000 from Alice’s Account to Bob’s

Client

Server

Done

Client

Server

CRASH!!

Client

Server

Do it again!

Client

Server

Sorry ...

Client

Server

At-most-once semantics
Making ONC RPC Reliable

• Each request uniquely identified by transmission ID (XID)
  – transmission and retransmission share same XID
• Server maintains duplicate request cache (DRC)
  – holds XIDs of non-idempotent requests and copies of their complete responses
  – kept in cache for a few minutes
Algorithm

Receive request

Duplicate?

Yes

Repeat original response

No

Perform request
Did It Work?

- No
Problem ...

Client
write(data) xid=1
nfsd1
blocks

Client
write(data) xid=1 (retransmission)
nfsd2

Client
done
updates file

Client
write(newdata) xid=2
nfsd2

Client
done
updates file

Client
done
updates file
Solution

1. Receive request
2. Duplicate?
   - Yes: Original in progress?
     - Yes: Discard
     - No: Original successful?
       - Yes: Repeat original reply
       - No: Perform request
3. No: Perform request
4. Original in progress?
   - Yes: Discard
   - No: Original successful?
     - Yes: Repeat original reply
     - No: Idempotent?
       - Yes: Perform request
       - No: Perform request
Did It Work?

• Sort of …
• Works fine in well behaved networks
• Doesn’t work with “Byzantine” routers
  – programmed by your worst (and brightest) enemy
  – probably doesn’t occur in local environment
  – good approximation of behavior on overloaded Internet
• Doesn’t work if server crashes at inopportune moment (and comes back up)
Enter TCP

- Any improvement?
  - no ...

RPC
TCP
IP
What’s Wrong?

• The problem is the duplicate request cache (DRC)
  – it’s necessary
  – but when may cached entries be removed?
Session-Oriented RPC

Client 1

2-slot DRC for client 1

Client 2

3-slot DRC for client 2
DCE RPC

• Designed by Apollo and Digital in the 1980s
  – both companies later absorbed by HP
• Does everything ONC RPC can do, and more
• Basis for Microsoft RPC
typedef struct {
    double comp1;
    int comp2;
    long long comp3;
    char *annotation;
} value_t;

char add(int key, value_t value);
char remove(int key, value_t value);
int query(int key, int number, value_t values[]);
An Interface Specification

interface db {
    typedef struct {
        double   comp1;
        long     comp2;
        hyper    comp3;
        [string, ptr]
            ISO_LATIN_1
                *annotation;
    } value_t;

    boolean add(
        [in] long key,
        [in] value_t value
    );

    boolean remove(
        [in] long key,
        [in] value_t value
    );

    [idempotent] long query(
        [in] long key,
        [in] long number,
        [out, size_is(number)]
            value_t values[]
    );
}

An Interface Specification (notes continued)

interface db {
    typedef struct {
        double  comp1;
        long    comp2;
        hyper   comp3;
        [string, ptr]
            ISO_LATIN_1
                *annotation;
    } value_t;

    boolean add(
        [in]  long key,
        [in]  value_t value
    );

    boolean remove(
        [in]  long key,
        [in]  value_t value
    );

    [idempotent] long query(
        [in]  long key,
        [in]  long number,
        [out, size_is(number)]
            value_t values[]
    );

};
Representing an Array

| Length | Item 1 | Item 2 | …     | Item n |
Representing Pointers

sender

marshalled

receiver

on stack

on callee’s stack
Complications
Marshalling Unrestricted Pointers

A:

B:

C:

D:

E:

0 (A):

2

4

2 (B):

-1

6

4 (C):

6

8

6 (D):

8 (E):
Referring to Server State

Client

pointer

Server
Maintaining Client State on Servers

```c
interface trees {
    typedef [context_handle] void *tree_t;

    void create (
        [in] long value,
        [out] tree_t pine
    );

    void insert (
        [in] long value,
        [in, out] tree_t pine
    );
}
```
Unique Identifiers

\[
\text{uuid } (333A2276-0000-0000-0D00-008090C000000), \\
\text{version } (3.1)]
\]

\textbf{interface} vectorops \{

\textbf{small} inner \{

\textbf{in} \quad \textbf{long} \ \text{size},
\textbf{in, size_is} (size) \textbf{long} \ A[],
\textbf{in, size_is} (size) \textbf{long} \ B[],
\textbf{out} \quad \textbf{long} \ *\text{result}

\}

\}
UUIDs

- **low bits of time**: 32 bits
- **mid bits of time**: 16 bits
- **high bits of time**: 12 bits
- **version**: 4 bits
- **clock seq high**: 2 bits
- **clock seq low**: 6 bits
- **node address**: 48 bits
- **reserved**: 8 bits