End-to-End Protocols

Class 17

Message-based Protocols
Remote Procedure Call

- RPCs make distributed computations look like local computations

- **Issues:**
  - Parameter passing
  - Identifying and accessing the remote procedure
  - Failure handling
RPC Concepts

- Generic Design
- Specific Case - Sun RPC
- Lightweight RPCs
- Remote Method Invocation (RMI)
  - Design issues
Distributed Program Design

• Communication-Oriented Design
  - Design protocol first
  - Build programs that adhere to the protocol

• Application-Oriented Design
  - Build application(s)
  - Divide programs up and add communication protocols
RPC Timeline

Could be at a remote location or on the same machine as the client

Client

Blocked

Request

Server

Blocked

Computing

Reply

Blocked
Generic RPC Design
 RPC Architecture

- Moving one or more procedure to a remote machine requires
  - additional code between the procedure call and the remote procedure
RPC Functions

- **Client side** - the new code must
  - *marshal arguments*
  - *translate them* to a machine-independent representation
  - *create an RPC CALL message*
  - *send the message* to the remote program
  - *wait for the results*, and
  - *translate the resulting values* back to the client’s native representation
RPC Functions (cont.)

- **Server side** - the new code must
  - *accept* an incoming RPC request
  - *translate arguments* to the server’s native data representation
  - *dispatch* the message to the appropriate *procedure/process*
  - *form a reply message* by translating values to the machine-independent data representation, and
  - *send* the result back to the client
RPC Architecture

- **RPC must**
  - keep the program structure intact
    - (including the interface between the original calling and called procedures)
  - isolate the code which handles RPC from the code that handles the application

- **RPC code is added in the form of two extra procedures (called *stub procedures*) that completely hide the communication details**
Stub Procedures

- Stubs use the same interface as the original call

- Adding stubs does not require a change to either the original called procedure or the calling procedure
“Ideal” 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
  - A.k.a. 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 exception conditions:
    • if last fragment arrives but message not complete
    • if timer LAST_FRAG expires
    • 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
Request/Reply (CHAN)

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

### Simple case

```
Client         Server
| Request      |
| ACK          |
| Reply        |
| ACK          |
```

### Implicit Acks

```
Client         Server
| Request 1    |
| Reply 1     |
| Request 2   |
| Reply 2     |
| ...         |
```
Remote Procedure deposit()

\[
\text{deposit (DavesAccount, $100)}
\]

- Always remember that you don't know how many times the remote procedure was run
  - The net can duplicate the request (UDP)
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
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
RPC Process-to-Process Architecture

- Call P
- Arguments
- Interface descriptor for procedure P
- Code
- Marshalled arguments
- Client stub
- Stub compiler
- Code
- Marshalled arguments
- RPC
- Message
- P
- Arguments
- Server stub
- RPC
Specific Type of RPC

SUN RPC
Specific RCP - SUNRPC

- One of the most widely used RPC systems
- Developed for use with NFS
- Built on top of UDP or TCP
  - TCP: stream is divided into records
  - TCP: return error if connection is terminated by server
  - UDP: max packet size < 8912 bytes
  - UDP: timeout plus limited number of retransmissions
- Multiple arguments marshaled into a single structure
- Semantics
  - At-least-once if reply received
  - At-least-zero if no reply
  - With UDP tries at-most-once
- Use SUN's eXternal Data Representation (XDR)
  - Big endian order for 32 bit integers, handle arbitrarily large data structures
XDR Formatting

- **Representation of base types**
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian

<table>
<thead>
<tr>
<th>Big-endian</th>
<th>(2)</th>
<th>(17)</th>
<th>(34)</th>
<th>(126)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00000010</td>
<td>00010001</td>
<td>00100010</td>
<td>01111110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Little-endian</th>
<th>(126)</th>
<th>(34)</th>
<th>(17)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01111110</td>
<td>00100010</td>
<td>00010001</td>
<td>00000010</td>
</tr>
</tbody>
</table>

Low address: (2) 00000010
High address: (126) 01111110
RPC Components and OSI Stack

**OSI Layers**

7. Application

6. Presentation

5. Session

4. Transport

3. Network

1-2. Physical/Data-link

- User Application
- XDR
- RPC
- TCP
- UDP
- IP
- Hardware Interface
- SunRPC
- UDP
- IP
- ETH
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 considered
  - integers
  - floats
  - strings
  - arrays
  - structs
Dynamic Port Mapping

- Servers typically do not use well known protocol ports

- Clients know the Program ID (and host IP address)

- RPC includes support for looking up the port number of a remote program
Port Lookup Service

- A port lookup service runs on each host that contains RPC servers
  - Port-mapper
  - Provides a central registry for RPC services
- Servers tell the port mapper what services they offer
- Clients ask port mapper for the port number corresponding to Remote Program ID
  - The portmapper is available on a well-known port (111)
- RPC servers register themselves with this service:
  - "I'm program 17 and I'm looking for requests on port 1736"
Port Mapper

- **Server start-up:** create port
- Server stub calls `svc_register` to register prog. #, version # with *local port mapper*
- Port mapper stores prog #, version #, and port

- **Client start-up:** call `clnt_create` to locate server port
- Upon return, client can call procedures at the server
Lightweight RPC
Lightweight RPCs

- Many RPCs occur between client and server on same machine
  - Need to optimize RPCs for this special case => use a lightweight RPC mechanism (LRPC)

- **Server** exports interface to remote procedures

- **Client** on same machine imports interface

- **OS kernel** creates data structures including an argument stack *shared* between **Server** and **Client**
Lightweight RPCs

• **RPC execution**
  - Push arguments onto stack
  - Trap to kernel
  - Kernel changes mem map of client to server address space
  - Client thread executes procedure (OS upcall)
  - Thread traps to kernel upon completion
  - Kernel changes the address space back and returns control to client

• **Called “doors” in Solaris**
Doors

- Which RPC to use? - run-time bit allows stub to choose between LRPC and RPC
Other Types of RPC
Other RPC Models

• Asynchronous RPC
  - Request-reply behavior often not needed
  - Server can reply as soon as request is received and execute procedure later

• Deferred-synchronous RPC
  - Use two asynchronous RPCs
  - Client needs a reply but can’t wait for it
    • server sends reply via another asynchronous RPC

• One-way RPC
  - Client does not even wait for an ACK from the server
  - Limitation: reliability not guaranteed (Client does not know if procedure was executed by the server).
Asynchronous RPC

(a) The interconnection between client and server in a traditional RPC
(b) The interaction using asynchronous RPC
Deferred Synchronous RPC

- A client and server interacting through two asynchronous RPCs
Remote Method Invocation (RMI)

• RPCs applied to objects, i.e., instances of a class
  - \textit{Class}: object-oriented abstraction
    - module with data and operations
  - Maintain separation between interface and implementation
  - Interface resides on one machine, implementation on another

• RMIIs support system-wide object references
  - Parameters can be object references
Distributed Objects

- When a client binds to a distributed object, load the interface ("proxy") into client address space
  - Proxy analogous to stubs
- Server stub is referred to as a skeleton
Proxies and Skeletons

• **Proxy: client stub**
  - Maintains server ID, endpoint, object ID
  - Sets up and tears down connection with the server
  - Serializes local object parameters
  - Can be downloaded/constructed on the fly

• **Skeleton: server stub**
  - Deserializes and passes parameters to server
  - Sends result to proxy
End of Class 17