

Remote Procedure Call (RPC)

- Paradigms in building distributed applications
 - The RPC model
 - Primitives
 - Issues
 - Case study: Sun RPC
 - *Reading: Coulouris, Chapter 5*
-

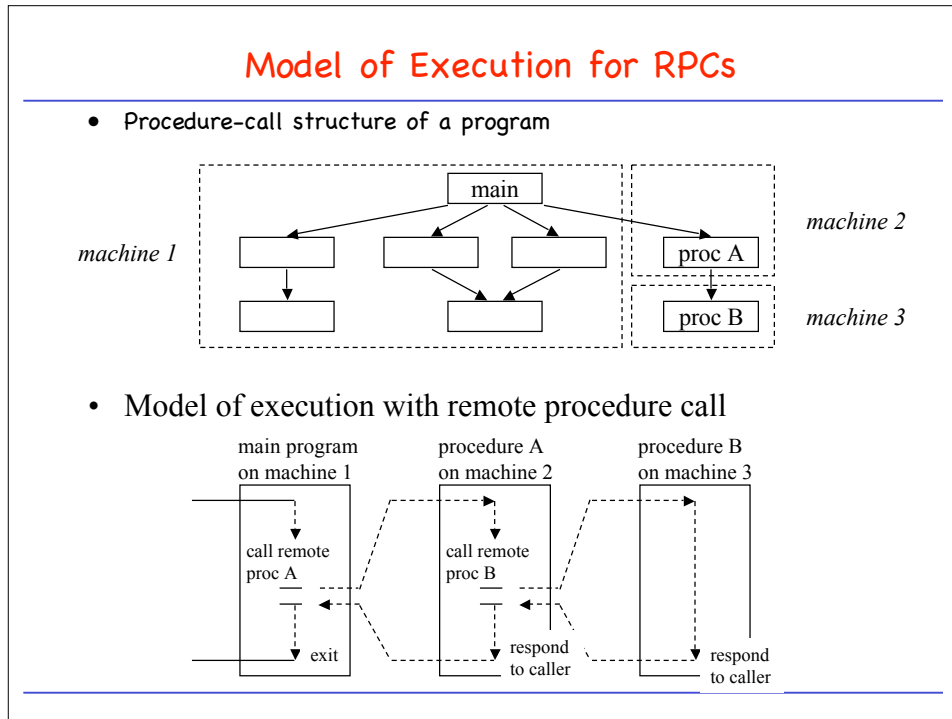
Building Distributed Programs: Two Paradigms

Paradigms:

- Communication-Oriented Design
 - Start with communication protocol
 - Design message format and syntax
 - Design client and server components by specifying how they react to incoming messages
- Application-Oriented Design
 - Start with application
 - Design, build, test conventional implementation
 - Partition program

Problems:

- Protocol-design problems
 - Application components as finite-state machines !?
 - Focus on communication instead of application!
 - Concurrency
-



RPC Properties

- Uniform call structure
 - Type checking
 - Full parameter functionality
 - Distributed binding
 - Recovery of orphan computations
-

RPC Primitives

- Invocation at caller side

```
call service (value_args; result_args);
```

- Definition at server side

- declaration

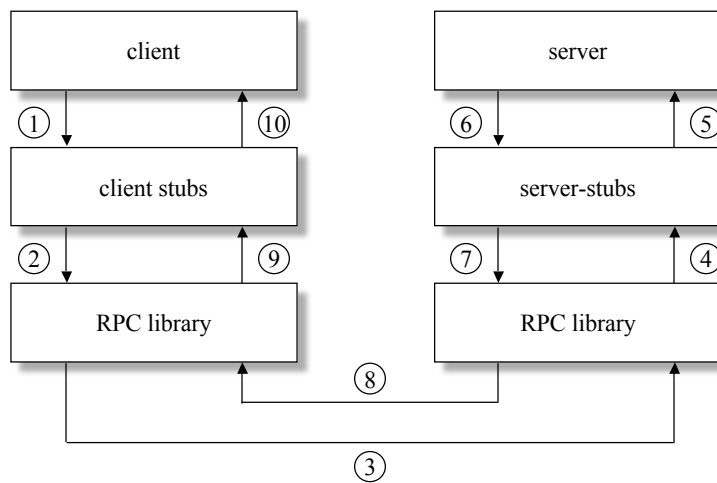
```
remote procedure service (in value_pars;
                           out result_pars);
```

```
begin body end;
```

- rendezvous statement

```
accept service (in value_pars;
                out result_pars) -> body;
```

Structure of an RPC Call

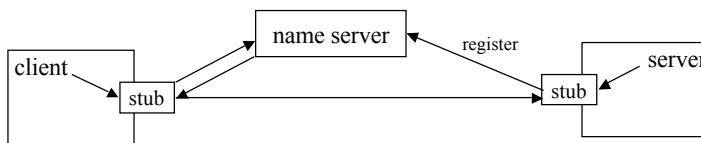


RPCs: Issues

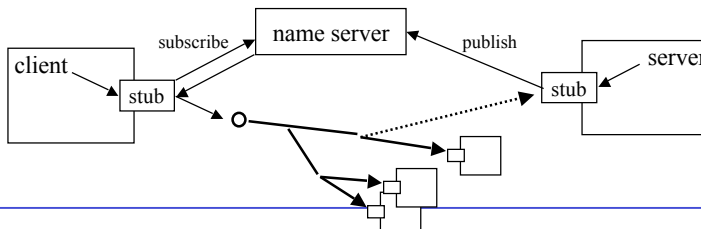
- Parameter passing
 - value parameters
 - reference parameters?
- Marshalling
 - simple data types
 - complex data structures
- Exception handling
 - language dependent
 - need to deal with asynchronous events

Locating Servers

- Broadcast requests
 - broadcast call and process incoming replies
- Name servers
 - server registers with name server



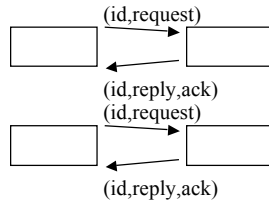
- Combination: publish/subscribe



Communication Protocols for RPC

- Reliable protocols: e.g. TCP
- Unreliable datagram protocols: e.g. UDP
- Specifically designed protocols: Example

Simple Call



Client times out and retransmits request.

Three cases:

- request lost
- server still executing
- ack lost

Complicated Call

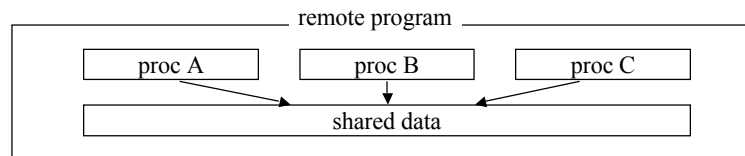
- long gaps between requests
 - acknowledge each message transmission separately
 or
 - periodically send “I-am-alive” message and use simple-call scheme.
- long messages (don't fit into packet)
 - segment message
 - segment-relative seq #'s
 - retransmission scheme for segments

RPC in Heterogeneous Environments

- Compile-time support
- Binding protocol
- Transport protocol
- Control protocol
- Data representation

Case Study: SUN RPC

- Defines format for messages, arguments, and results.
- Uses UDP or TCP.
- Uses XDR (eXternal Data Representation) to represent procedure arguments and header data.
- Compiler system to automatically generate distributed programs.
- Remote execution environment: remote program.



- Mutually exclusive execution of procedure in remote program.

Identifying Remote Programs and Procedures

- Conceptually, each procedure on a computer is identified by pair :
 - $(prog, proc)$
 - *prog*: 32-bit integer identifying remote program
 - *proc*: integer identifying procedure
- Set of program numbers partitioned into 8 sets.

0x00000000 - 0x1ffffff	assigned by SUN
0x20000000 - 0x3ffffff	assigned by local system manager
0x40000000 - 0x5ffffff	temporary
0x60000000 - 0xfffffff	reserved

- Multiple remote program versions can be identified:
 - $(prog, version, proc)$

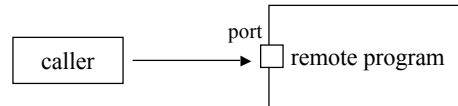
Example RPC Program Numbers

<u>name</u>	<u>assigned no</u>	<u>description</u>
portmap	100000	port mapper
rstatd	100001	rstat, rup, perfmeter
rusersd	100002	remote users
nfs	100003	network file system
ypserv	100004	yp (NIS)
mountd	100005	mount, showmount
dbxd	100006	DBXprog (debug)
ypbind	100007	NIS binder
walld	100008	rwall, shutdown
yppasswdd	100009	yppasswd

Communication Semantics

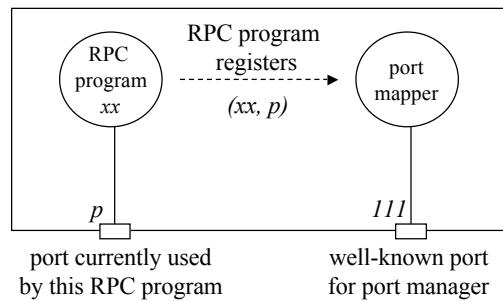
- TCP or UDP ?
- Sun RPC semantics defined as function of underlying transport protocol.
 - RPC on UDP: calls can be lost or duplicated.
- *at-least-once* semantics if caller receives reply.
- *zero-or-more* semantics if caller does not receive reply.
- Programming with *zero-or-more* semantics: *idempotent* procedure calls.
- Sun RPC retransmission mechanism:
 - non-adaptive timeouts
 - fixed number of retransmissions

Remote Programs and Protocol Ports



program_id vs. port_id
(32 bit) (16 bit)

- Dynamic port mapping: *RPC port mapper*



Sun RPC Message Format: XDR Specification

```

enum msg_type { /* RPC message type constants */
    CALL = 0;
    REPLY = 1;
};
  
```

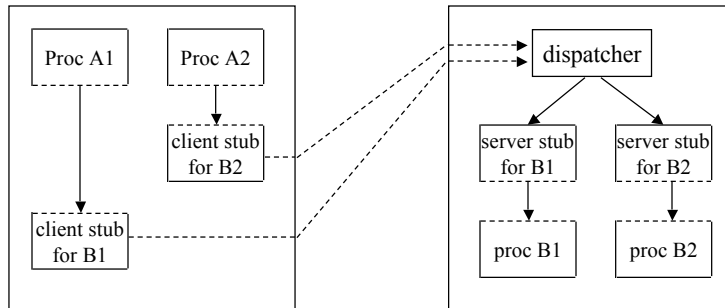
```

struct rpc_msg { /* format of a RPC message */
    unsigned int mesgid; /* used to match reply to call */
    union switch (msg_type mesgt) {
        case CALL : call_body cbody;
        case REPLY: reply_body rbody;
    } body;
};
  
```

```

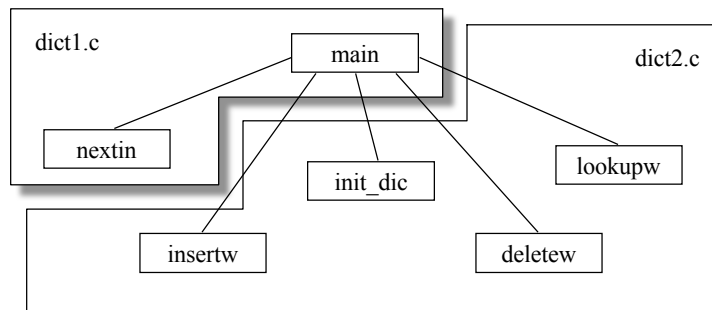
struct call_body { /* format of RPC CALL */
    u_int rpcvers; /* which version of RPC? */
    u_int rprog; /* remote program number */
    u_int rprogvers; /* version number of remote prog */
    u_int rproc; /* number of remote procedure */
    opaque_auth cred; /* credentials for called auth. */
    opaque_auth verf; /* authentication verifier */
    /* ARGS */
};
  
```


Message Dispatch for Remote Programs



Creating Distributed Applications with Sun RPC Example: Remote Dictionary Using `rpcgen`

- Procedure call structure:



Procedures should execute on the same machines as their resources are located.

Specification for rpcgen

Specify:

- constants
- data types
- remote programs, their procedures, types of parameters

```

/* rdict.x */
/* RPC declarations for dictionary program */
const MAXWORD = 50;
const DICTSIZ = 100;
struct example { /* unused; rpcgen would */
    int exfield1; /* generate XDR routines */
    char exfield2; /* to convert this structure.*/
};

/* RDICTPROG: remote program that provides
insert, delete, and lookup */

program RDICTPROG { /* name (not used) */
    version RDICTVERS { /* version declarat.*/
        int INITW(void) = 1; /* first procedure */
        int INSERTW(string) = 2; /* second proc.... */
        int DELETEW(string) = 3;
        int LOOKUP(string) = 4;
    } = 1; /* version definit.*/
} = 0x30090949; /* program no */
/* (must be unique)*/
    
```

Program Generation

