Remote Procedure Calls

CS 707

Motivation

- Send and Recv calls ⇔ I/O
- Goal: make distributed nature of system ***transparent*** to the programmer
- RPC provides procedural interface to distributed services

Figure 4.5 Request-reply communication.

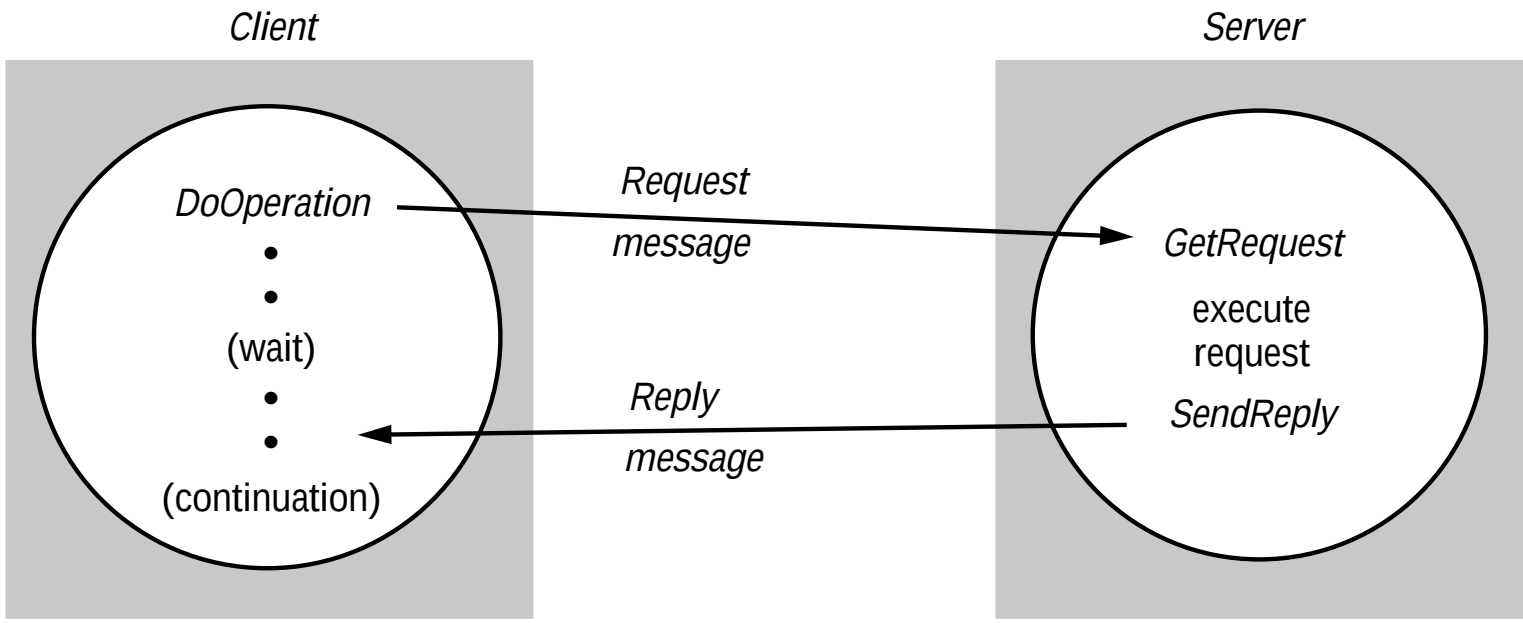
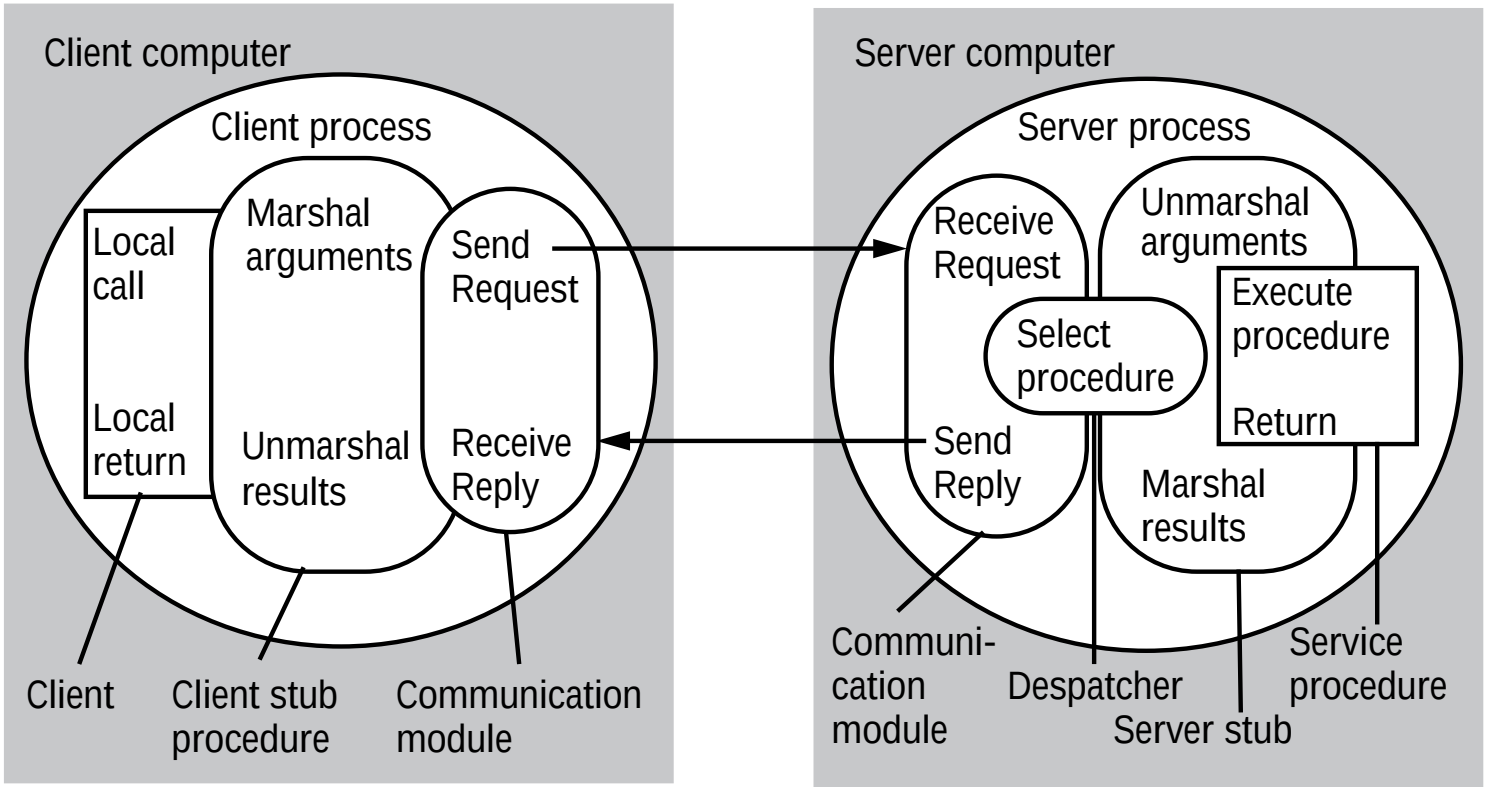
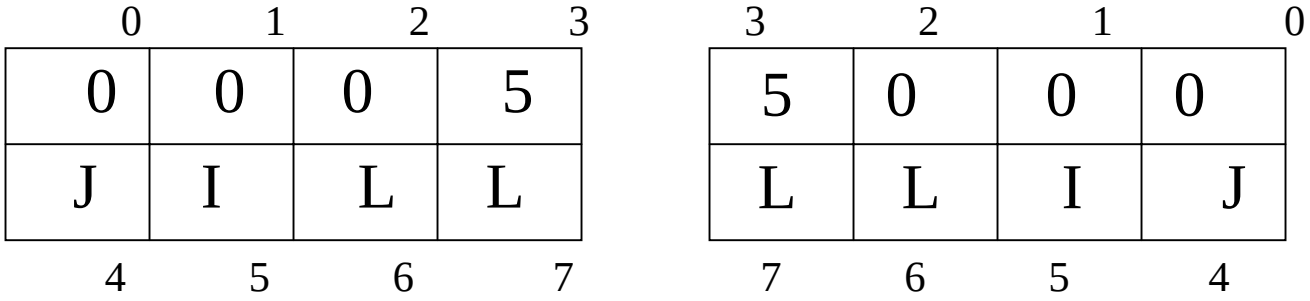


Figure 5.2 Stub procedures.



Issues in RPC

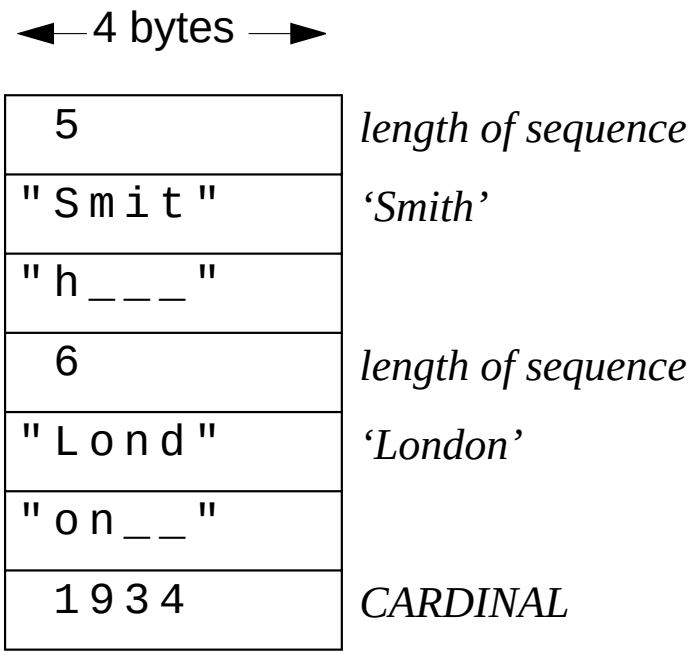
- Parameter Passing
  - marshalling
  - big endian vs little endian



Parameter Passing

- Passing arrays
  - in C by reference
- Canonical forms
  - SUN XDR, Xerox Courier
- Passing pointers?

Figure 4.1 XDR message.



The message is: 'Smith', 'London', 1934

Figure 4.6 Request-reply message structure.

Figure 4.7 RPC protocols.

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

Handling Failures

- Types of failure
  - client unable to locate server
  - request message lost
  - reply message lost
  - server crashes after receiving a request
  - client crashes after sending a request

Handling failures

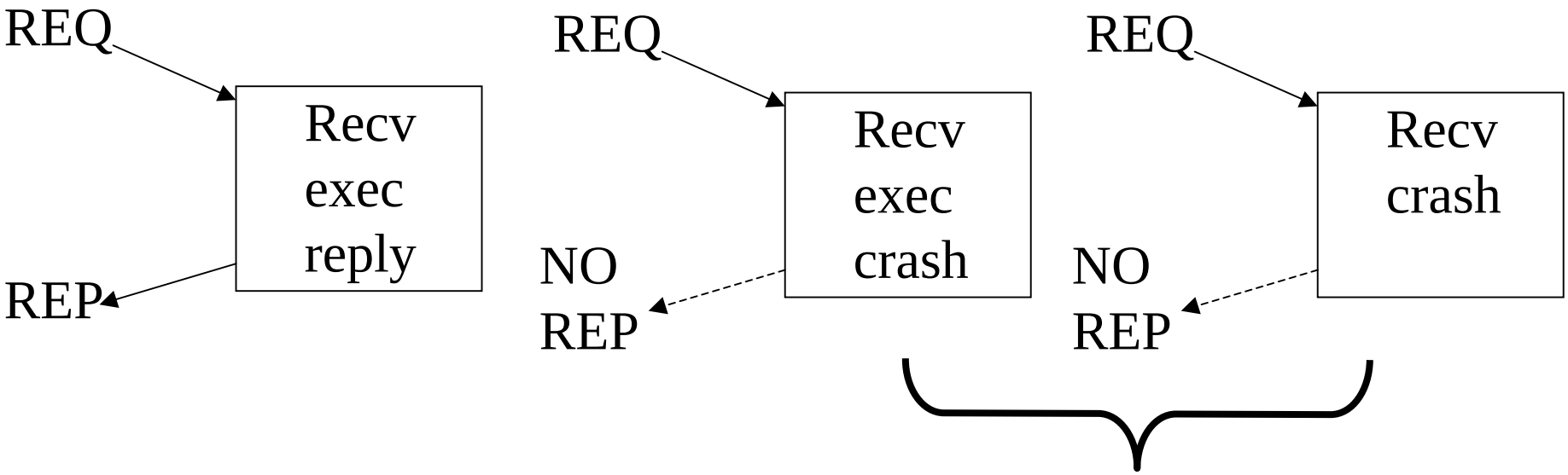
- Client cannot locate server
  - Reasons
    - server crashed
    - client compiled using old version of server interface
  - RPC system must be able to report error to client
  - loss of transparency

Handling failures

- Lost request message
  - retransmit a fixed number of times
- Lost reply message
  - client retransmits request
  - server choices
    - filter duplicates ⇒ hold on to results until ACK
    - re-execute procedure ⇒ ***service should be idempotent so that it can be safely repeated***

Handling failures

- Server crashes



Handling failures

- Server crashes
  - at least once (keep trying till server comes up)
  - at most once (return immediately)
  - *exactly once impossible to achieve*
- ONC RPC (Sun) uses at least once semantics if a RPC is successful and “zero or more” semantics if call fails

Delivery guarantees			*RPC call semantics*
Retry request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Handling failures

- Client crashes
  - if a client crashes before RPC returns, we have an “orphan” computation at server
  - wastes resources, could also start other remote computations
  - orphan detection
    - reincarnation (client broadcasts new “epoch” when it comes up)
    - expiration (RPC has fixed amount of time T to do work)

Binding

- Dynamic
  - Servers
    - register service with binder
    - withdraw
  - Client
    - lookup address of service
  - SUN RPC portmapper runs on every host

Figure 5.3 Binder Interface.

PROCEDURE Register (serviceName:String; serverPort:Port; version:integer)
causes the binder to record the service name and server port of a service in its table, together with a version number.

PROCEDURE Withdraw (serviceName:String; serverPort:Port; version:integer)
causes the binder to remove the service from its table.

PROCEDURE LookUp (serviceName:String; version:integer): Port
the binder looks up the named service and returns its address(or set of addresses) if the version number agrees with the one stored in its table.

RPC protocols

- Connection-oriented or connection-less
  - connectionless has better performance on LANs
- UDP/IP or roll your own protocol
  - specialized protocol better but more effort
- large RPCs have to be broken up into multiple packets

Interface definition languages

- SUN XDR
- rpcgen (IDL compiler)
  - creates client and server stub procedures, header files, despatcher, server main procedure
  - stubs use XDR for marshalling and unmarshalling

Figure 5.1 Levels in the client software.

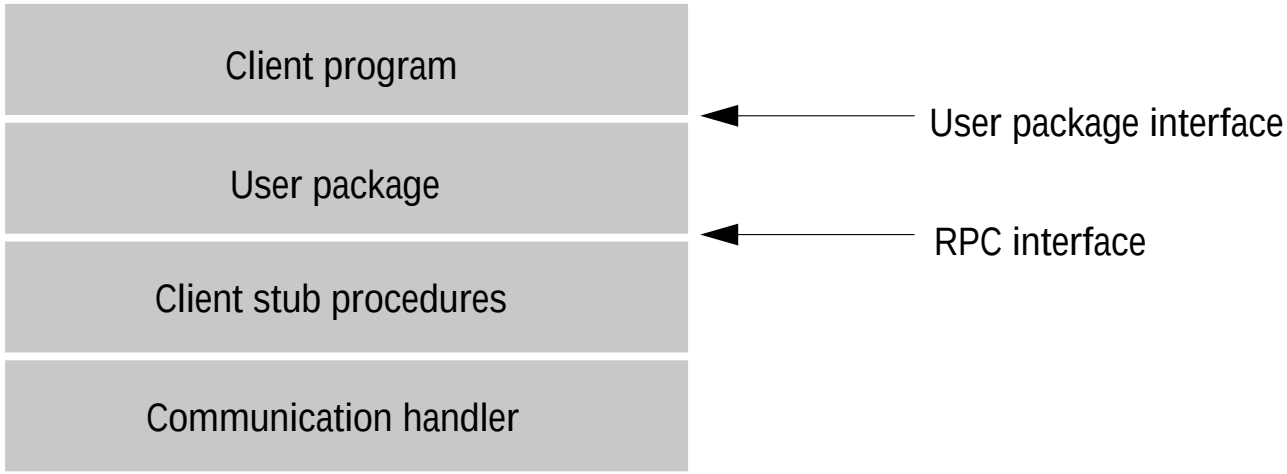


Figure 5.4 Files interface in Sun XDR.

```
 /* FileReadWrite service interface definition in file FileReadWrite.x
*/
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
        int length;
        char buffer[MAX];
};
struct writeargs {
        FileIdentifier f;
        FilePointer position;
        Data data;
};
struct readargs {
        FileIdentifier f;
        FilePointer position;
        Length length;
};

program FILEREADWRITE {
 version VERSION {
        void WRITE(writeargs)=1;
        Data READ(readargs)=2;
        }=2;
} = 9999;
```

Figure 5.5 C program for client in Sun RPC.

```
/* File : C.c - Simple client of the FileReadWrite service. */

#include <stdio.h>
#include <rpc/rpc.h>
#include "FileReadWrite .h"
main(int argc, char ** argv)
{
        CLIENT *clientHandle;
        char *serverName = "coffee";
        readargs a;
        Data *data;

        clientHandle= clnt_create(serverName, FILEREADWRITE,
              VERSION, "udp");        /* creates socket and a client handle*/
        if (clientHandle==NULL){
              clnt_pcreateerror(serverName); /* unable to contact server */
             exit(1);
        }
        a.f = 10;
        a.position = 100;
        a.length = 1000;
        data = read_2(&a, clientHandle);/* call to remote read procedure */
        •••
        clnt_destroy(clientHandle);              /* closes socket */
}
```

Figure 5.6 C program for server procedures in Sun RPC.

```
/* File S.c - server procedures for the FileReadWrite service */
#include <stdio.h>
#include <rpc/rpc.h>
#include"FileReadWrite.h"

void * write_2(writeargs *a)
{
 /* do the writing to the file */
}

Data * read_2(readargs * a)
{
        static Data result;      /* must be static */
        result.buffer  =  ...    /* do the reading from the file */
        result.length  =  ...    /* amount read from the file */
        return &result;
}
```