CprE 450/550x
Distributed Systems and Middleware

Inter-process Communication

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu
February 9, 2004

Readings for Today's Lecture

- References
  - **Chapter 2 of "Distributed Systems: Principles and Paradigms"**

3

Remote Procedure Call

- Allow programs to call procedures located on other machines
  - Information can be transported from the caller to the callee in the parameters and come back in the procedure result.
  - No message passing is visible to the programmer
- Problems:
  - Different machines
  - Different address spaces
  - Both machines can crash and each of the possible failures causes different problems.

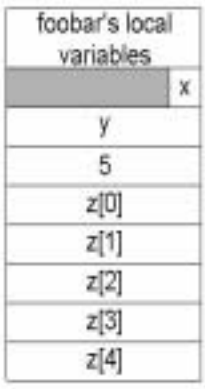
4

Parameter Specification and Stub Generation

a) A procedure
b) The corresponding message.

```
foobar( char x; float y; int z[5] )
{
  ....
}
```

(a)



(b)

5

Parameter Passing

- Passing value parameters
  - Parameter marshaling
- Passing reference parameters
  - How are pointers (i.e., references) passed?
  - A pointer is meaningful only within the address space of the process being used
  - Solutions:
    1. Forbid pointers and reference parameters
    2. Case: Pointer to an array of characters.

    If the size of the array is known, one strategy is to copy the array into a message and send it to the server.

    Call-by-reference is replaced by copy/restore.
  - Although we can handle pointers to simple arrays and structures, we still cannot handle the most general case of a pointer to an arbitrary data structures (complex graph).

6

DCE RPC

- RPC have been widely adopted as the basis of middleware and distributed systems.
- Distributed Computing Environment (DCE)
  - Developed by Open Software Foundation (OSF)
  - DCE is a middleware system in that it is designed to execute as a layer of abstraction between existing (network) operating systems and distributed applications.
  - The programming model underlying all of DCE is the client-server model.
  - A number of services that form part of DCE itself:
    - Distributed file service
    - Directory service
    - Access control
    - Distributed time service
- DCE RPC is good representative of RPC systems, though it is not as popular as SUN RPC. But it has been adopted in Microsoft's base system for distributed computing.

7

Goals of DCE RPC

- Make it possible for a client to access a remote service by simply calling a local procedure
- Make it possible to have a large of volumes of existing codes run in a distributed environment with few, if any, changes.
- Hide the details from the clients and servers

DCE semantic options:

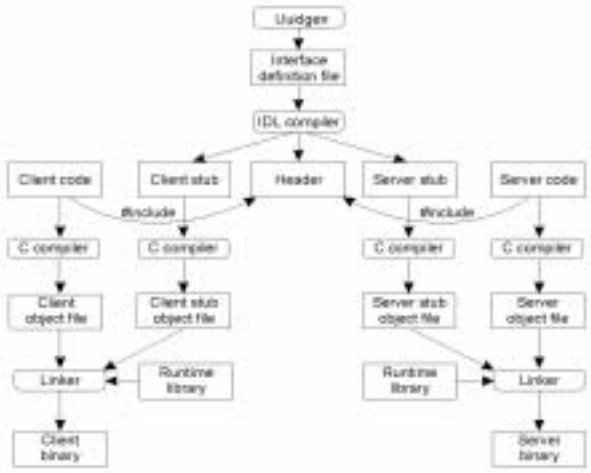
•At-most-once operation: No call is carried over more than once even in the face of system crashes

•Idempotent: repeated multiple times without harm

Can mark a remote procedure as idempotent

8

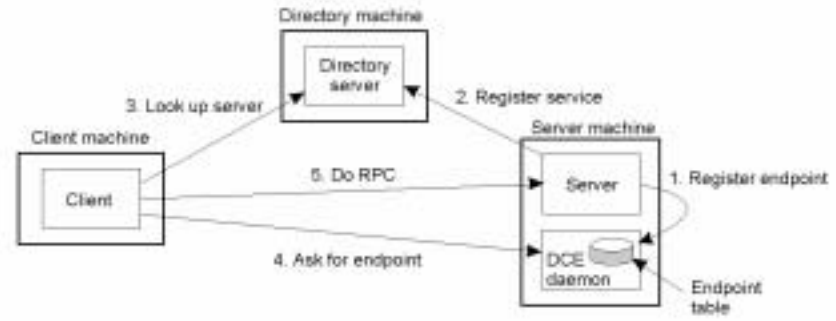
Writing a Client and a Server



- The steps in writing a client and a server in DCE RPC.

9

Binding a Client to a Server



- Client-to-server binding in DCE.

10

Example (local procedure)

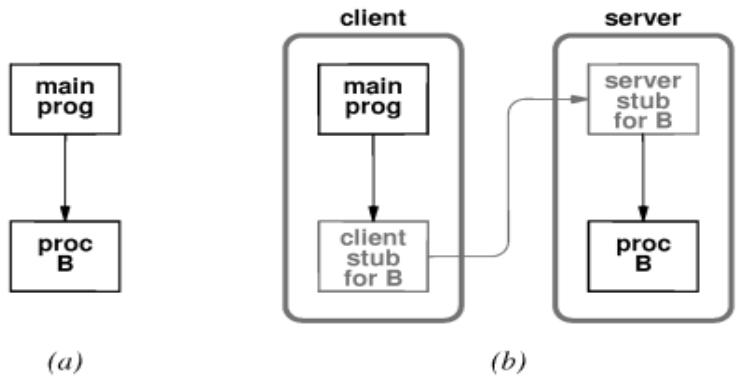
```
int f(x,y) {
   x = x+1; y = y+2;
   return(x+y);
}

main( )
{
   int a,b,c; a = 0; b = 1;
   c = f(a,b);
   printf("a = %d, b = %d, c = %d \n", a,b,c);
}
```

Parameter Passing Mechanism and Value Printed:

- Call-by-value: a = 0, b = 1, c = 4
- Call-by-reference: a = 1, b = 3, c = 4
- Call-by-copy/restore: a = 1, b = 3, c = 4

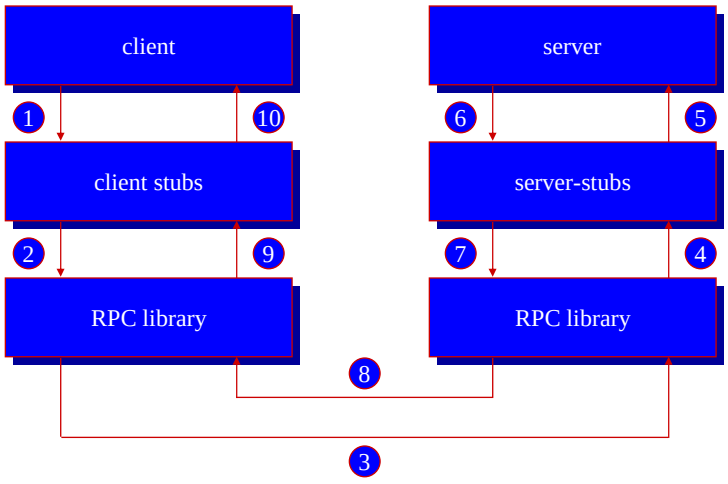
11

Client and Server Stubs



12

Structure of an RPC Call



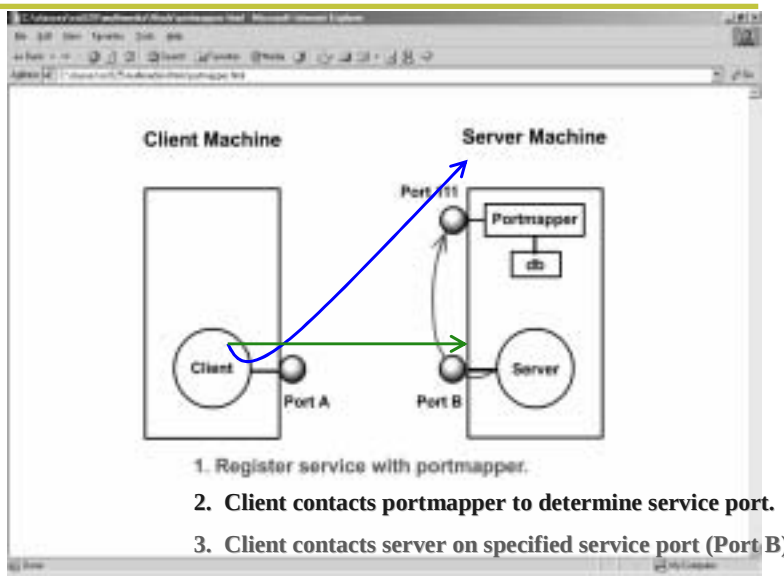
13

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

14

RPC Interaction



15

RPC Implementation

Establishing an RPC Session

1. The server **registers** its services (procedures) with the portmapper.
2. The client **contacts the portmapper** to determine if the requested service (procedure) is available; and if so, on which port.
3. The client **contacts the server** to initiate service.

16

XDR - eXternal Data Representation

- **XDR** is a universally used standard from Sun Microsystems used to represent data in a network canonical form.
- A set of conversion functions are used to encode and decode data; for example, **xdr_int( )** is used to encode and decode integers. Data is converted into a network canonical form (a standard form) to be presented in a meaningful format to the receiving host.
- Conversion functions exist for all standard data types. However, for complex structures, RPCGEN can be used to generate conversion routines.

17

Sun RPC Message Format: XDR Specification

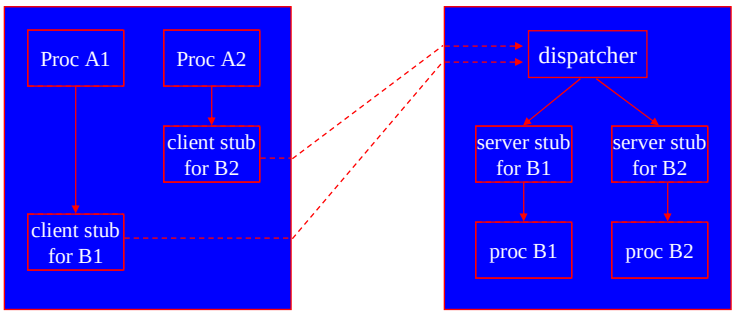
```
enum msg_type {  /* RPC message type constants */
      CALL  = 0;
      REPLY = 1;
};
```

```
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 */
};
```

18

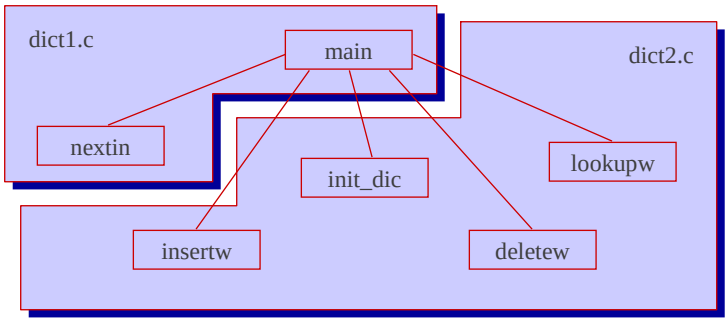
Message Dispatch for Remote Programs



19

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.

20

Specification for `rpcgen`

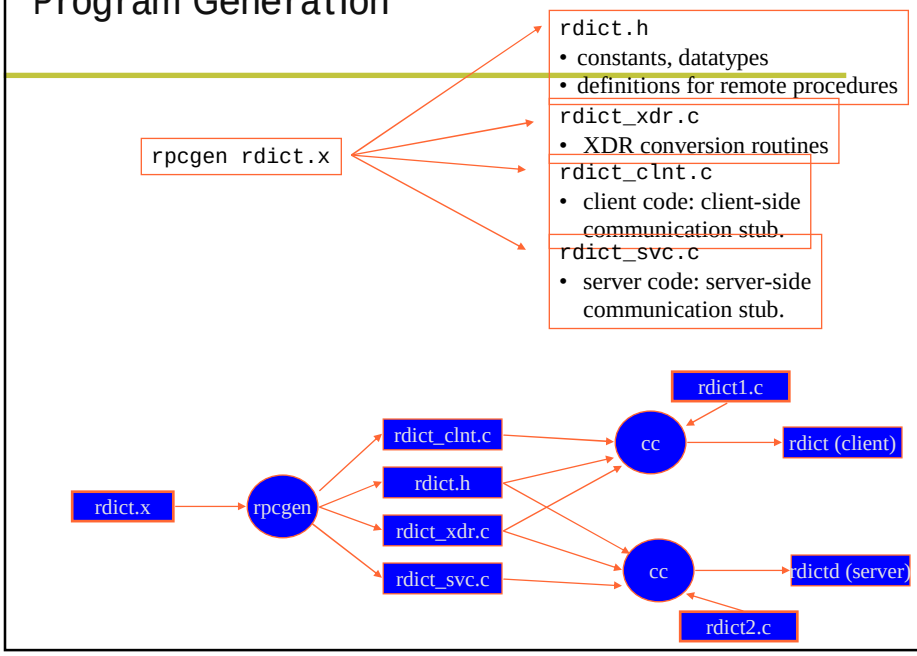
Specify:

- 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)*/
```

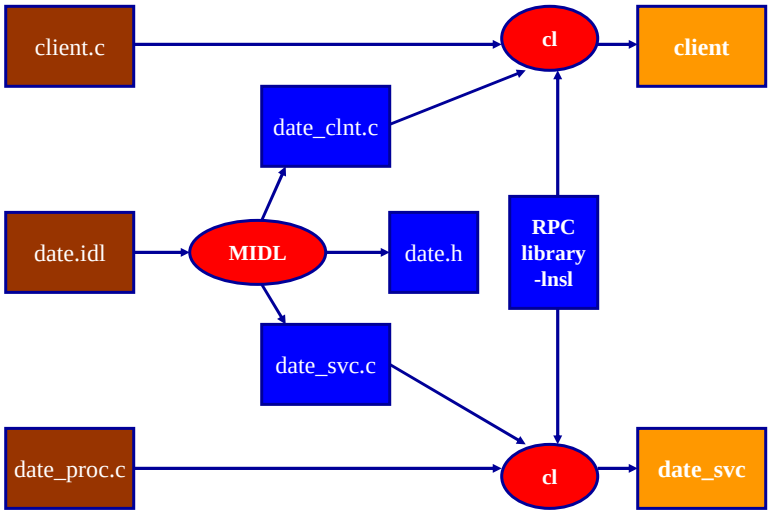
21

Program Generation



22

Windows (DCE) RPC Example



Date.x

```
/*
 * date.x - Specification of remote date, time, date and time service.
 */

/*
 * Define 1 procedure :
 * date_1() accepts a long and returns a string.
 */

program DATE_PROG {
   version DATE_VERS {
          string    DATE(long) = 1;     /* procedure number = 1 */
   } = 1;                                /* version number = 1   */
} = 0x31234567;                          /* program number       */
```


Client.c

```
main(int argc, char **argv)
{
   CLIENT  *cl;       /* RPC handle */
   char    *server;
   char    **sresult; /* return value from date_1() */
   char    s[MAX];    /* character array to hold output */
   long    response;  /* user response                  */
   long    *lresult;  /* pointer to user response       */

   if (argc != 2) {
      fprintf(stderr, "usage: %s hostname\n", argv[0]);
      exit(1);
   }
   server = argv[1];
   lresult = (&response);
   /*
    * Create the client "handle."
    */
   if ( (cl = clnt_create(server, DATE_PROG, DATE_VERS, "udp")) == NULL) {
      clnt_pcreateerror(server);
      exit(2);
   }
   response = get_response();
   while(response != 4) {
      if ((sresult = date_1(lresult, cl)) == NULL) {
         clnt_perror(cl, server);
         exit(3);
      }
            printf("  %s\n", *sresult);
            response = get_response();
   }
   clnt_destroy(cl);                     /* done with the handle */
   exit(0);
}
```

```
#include     <stdio.h>
#include     <string.h>
#include     <time.h>
#include     <sys/types.h>
#include     <rpc/rpc.h> /* standard RPC include file */
#include     "date.h"     /* this file is generated by rpcgen */

#define MAX 100
long get_response(void);
```

Client.c (cont.)

```
long get_response()
{
    long choice;

    printf("=============================================\n");
    printf("                    Menu: \n");
    printf("---------------------------------------------\n");
    printf("                1. Date\n");
    printf("                2. Time\n");
    printf("                3. Both\n");
    printf("                4. Quit\n");
    printf("---------------------------------------------\n");
    printf("              Choice (1-4):");
    scanf("%ld",&choice);
    printf("=============================================\n");
    return(choice);
}
```



```
/*
 * date_proc.c - remote procedures; called by server stub.
 */
#include <rpc/rpc.h>              /* standard RPC include file */
#include <time.h>
#include <sys/types.h>
#include "date.h"                 /* this file is generated by rpcgen */

#define MAX 100
/*
 * Return the binary date and time.
 */
char **
date_1(option)
long *option;
{
   struct tm *timeptr; /* Pointer to time structure     */
   time_t clock;       /* Clock value (in secs)         */
   static char *ptr;   /* Return string                 */
   static char err[] = "Invalid Response \0";
   static char s[MAX];

   clock = time(0);
   timeptr = localtime(&clock);

   switch(*option)
      {
      case 1: strftime(s,MAX,"%A, %B %d, %Y",timeptr);
              ptr=s;
              break;

      case 2: strftime(s,MAX,"%T",timeptr);
              ptr=s;
              break;

      case 3: strftime(s,MAX,"%A, %B %d, %Y - %T",timeptr);
              ptr=s;
              break;

      default: ptr=err;
               break;
      }
   return(&ptr);
}
```

Server.c

Any Questions?

See you next time.