Today

- RPC

DCE

Distributed Computing Environment

- machine-independent data representation
- remote procedure protocol
- protocol compiler
- authentication services
- service naming/mapping facility
- network time
- distributed file system

Candidates:

- Sun's ONC (NIS, RPC/XDR, NTP, NFS)
- OSF DCE (Apollo's NCS, NIDL, NDR/ASN.1, LB, DTS, AFS)

RPC

Remote Procedure Call

- make network service request look like a function call
  local process invokes a remote procedure
- handle network details
- handle data representation
- presentation/session layers

RPC features

- parameter passing
- binding/locating
- transport protocols
- exception handling
- call semantics (once: exactly, at most, at least)
- data representation
- security
- performance

RPC history

- Xerox Courier ('81)
- Sun ONC RPC ('85)
- Apollo NCS ('85)
- OSF DCE ('91)

Sun RPC

Components

- protocol specs (RFC's)
- programming API (*-lrpcsvc*)
- data conversion (XDR)
- service mapping (**portmap**)
- protocol compiler (**rpcgen**)
- used by NIS, NFS, xtr/traffic

Sun RPC

Semantics

- single argument and result
- both passed by address
- provides XDR routines for argument and result
- supports UDP, TCP (TLI/XTI)
  UDP: max message is 8K, rpc takes care of timeout, retry, sequence
- server registers service number with **portmap**
- view with **rpcinfo -p**
- some RPC servers started by **inetd**
- client requests service port number from **portmap**

details

- services registered by number/version
- each procedure within a service has a number (NULL predefined)
- no pre-assigned transport ports (except **portmap**)

```
Program Numbers
      0x00000000 - 0x1fffffff  defined by Sun
      0x20000000 - 0x3fffffff  defined by user
      0x40000000 - 0x5fffffff  transient
      0x60000000 - 0xffffffff  reserved

/etc/inetd.conf
# RPC services syntax:
#  <rpc_prog>/<vers> <socket_type> rpc/<proto> <flags> <user> <pathname> <args>
rstatd/2-4      dgram   rpc/udp wait root /usr/etc/rpc.rstatd   rpc.rstatd
sprayd/1        dgram   rpc/udp wait root /usr/etc/rpc.sprayd   rpc.sprayd
```

Sun RPC

Several implementation levels

- command level (rusers hostname)
- subroutines to existing servers (rstatd, etherd)
- build your own, use **rpcgen**
- or manually with simple calls (*call-rpc()*)
- handle more details (*clnt_call()*)
- lower, mess with sockets, fork(), select()

rstatd

```
stats sun4
sun4    L1 =  0.0, L5 =  0.0, L15 =  0.0, ierr = 2610 oerr = 800 coll =  2.0%

/* stats.c  link with -lrpcsvc    stats h1 h2 h3 .... */
#include <stdio.h>
#include <rpc/rpc.h>
#include <rpcsvc/rstat.h>
main(argc, argv)
{
    struct statstime xstats;
    for (argv++, argc-- ; argc > 0 ; argv++, argc--) {
   if (getdata(*argv, &xstats)) printdata(*argv, &xstats);
   }
}
getdata(hn, sp)
char * hn; struct stats * sp;
{
    return(!rstat(hn, sp));
}
#define loadavg(sp, n) (((float) (sp)->avenrun[(n)]) / 256.0)
printdata(hostname, sp)
char * hostname; struct statstime * sp;
{
    fprintf(stdout, "%s\tL1 = %4.1f, L5 = %4.1f,
     L15 = %4.1f, ierr = %d oerr = %d coll = %4.1f%%\n",
   hostname, loadavg(sp, 0), loadavg(sp, 1), loadavg(sp, 2),
   sp->if_ierrors, sp->if_oerrors,(double)sp->if_collisions*100./
    (double)sp->if_opackets);
}
```


rpcgen

- protocol compiler
- generate XDR, client stubs, server main
- you provide procedure and message descriptions in **.x** file
- you provide client main that calls stubs
- you provide server stubs that do service

RPC Language

- extension of XDR defintion language
- C-like
- defined in a **.x** file
- data declaratons (constant, enumerations, structures, unions, typedef)
- supports simple, fixed-array, variable-array, and pointers
- procedure definitions

rpcgen example

dunigan/cs494-unp/rpc/

- rpc calls to get binary and ascii time from a remote
- you write message descriptor file, client main, and server stubs
- you write **date.x, rpcdate.c, date_proc.c** and makefile
- rpcgen produces *date.h date_clnt.c date_svc.c*

date.x

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

/*
 * Define 2 procedures:
 * bin_date_1() returns the binary time and date (no arguments).
 * str_date_1() takes a binary time and returns a human-readable string.
 */

program DATE_PROG {
    version DATE_VERS {
   long  BIN_DATE(void) = 1;  /* procedure number = 1 */
   string   STR_DATE(long) = 2;  /* procedure number = 2 */
    } = 1;           /* version number = 1 */
} = 0x31234567;            /* program number = 0x31234567 */
```


client: rpcdate.c

```
#include <stdio.h>
#include <rpc/rpc.h> /* standard RPC include file */
#include "date.h" /* this file is generated by rpcgen */
main(argc, argv)
int   argc;
char  *argv[];
{
   CLIENT     *cl;     /* RPC handle */
   char     *server;
   long     *lresult;   /* return value from bin_date_1() */
   char     **sresult;  /* return value from str_date_1() */

   if (argc != 2) {
      fprintf(stderr, "usage: %s hostname\n", argv[0]);
      exit(1);
   }
   server = argv[1];
   if ( (cl = clnt_create(server, DATE_PROG, DATE_VERS, "udp")) == NULL) {
      clnt_pcreateerror(server);
      exit(2);
   }
   if ( (lresult = bin_date_1(NULL, cl)) == NULL) {
      clnt_perror(cl, server);
      exit(3);
   }
   printf("time on host %s = %ld\n", server, *lresult);
```



```
   if ( (sresult = str_date_1(lresult, cl)) == NULL) {
      clnt_perror(cl, server);
      exit(4);
   }
   printf("time on host %s = %s", server, *sresult);

   clnt_destroy(cl);    /* done with the handle */
   exit(0);
}
```

server stubs

date_proc.c

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

long *
bin_date_1()
{
   static long timeval; /* must be static */
   long     time();     /* Unix function */

   timeval = time((long *) 0);

   return(&timeval);
}

char **
str_date_1(bintime)
long  *bintime;
{
   static char *ptr;    /* must be static */
   char     *ctime();   /* Unix function */

   ptr = ctime(bintime);      /* convert to local time */

   return(&ptr);     /* return the address of pointer */
}
```


building client/server

- use rpcgen to generate .h and server/client
- link with your modules and rpc library to create client and server

```
rpcgen date.x
  generates date_clnt.c date_svc.c date.h

cc -o date_svc date_svc.c date_proc.c -lrpcsvc

        cc -o rpcdate rpcdate.c date_clnt.c -lrpcsvc

  to run, start server:   date_svc
  start client:   rpcdate serverhostname
    time on host cetus1a = 941503427
    time on host cetus1a = Mon Nov  1 19:43:47 1999
```


makefile

```
BIN = date_svc rpcdate
GEN = date_clnt.c date_svc.c date.h
LIB = -lrpcsvc
CFLAGS = -I.. -O

all: $(BIN)

$(GEN):     date.x
      rpcgen date.x

date_svc:   date_proc.o date_svc.o
      $(CC) -o $@ date_proc.o date_svc.o $(LIB)

rpcdate:      rpcdate.o date_clnt.o
      $(CC) -o $@ rpcdate.o date_clnt.o $(LIB)

rpcdate.o: rpcdate.c date.h

date_proc.o:   date_proc.c date.h

clean:
      rm -f $(GEN) *.o $(BIN)
```


rpcgen: date.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/types.h>


#define DATE_PROG ((u_long)0x31234567)
#define DATE_VERS ((u_long)1)
#define BIN_DATE ((u_long)1)
extern long *bin_date_1();
#define STR_DATE ((u_long)2)
extern char **str_date_1();
```

rpcgen:date_clnt.c

```
#include <rpc/rpc.h>
#include "date.h"
static struct timeval TIMEOUT = { 25, 0 };
long *
bin_date_1(argp, clnt)
   void *argp;
   CLIENT *clnt;
{
   static long res;
   bzero((char *)&res, sizeof(res));
   if (clnt_call(clnt, BIN_DATE, xdr_void,
    argp, xdr_long, &res, TIMEOUT) != RPC_SUCCESS) {
      return (NULL);
   }
   return (&res);
}
char **
str_date_1(argp, clnt)
   long *argp;
   CLIENT *clnt;
{
   static char *res;
   bzero((char *)&res, sizeof(res));
   if (clnt_call(clnt, STR_DATE, xdr_long,
    argp, xdr_wrapstring, &res, TIMEOUT) != RPC_SUCCESS) {
      return (NULL);
   }
   return (&res);
}
```


rpcgen:server main

date_svc.c

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "date.h"
static void date_prog_1();
main()
{
   register SVCXPRT *transp;
   (void) pmap_unset(DATE_PROG, DATE_VERS);
   transp = svcudp_create(RPC_ANYSOCK);
   if (transp == NULL) {
      fprintf(stderr, "cannot create udp service."); exit(1);
   }
   if (!svc_register(transp, DATE_PROG, DATE_VERS, date_prog_1, IPPROTO_UDP)) {
      fprintf(stderr, "unable to register (DATE_PROG, DATE_VERS, udp).");
      exit(1);
   }
   transp = svctcp_create(RPC_ANYSOCK, 0, 0);
   if (transp == NULL) {
      fprintf(stderr, "cannot create tcp service."); exit(1);
   }
   if (!svc_register(transp, DATE_PROG, DATE_VERS, date_prog_1, IPPROTO_TCP)) {
      fprintf(stderr, "unable to register (DATE_PROG, DATE_VERS, tcp).");
      exit(1);
   }
   svc_run();
   fprintf(stderr, "svc_run returned"); exit(1);
```



```
static void
date_prog_1(rqstp, transp)
   struct svc_req *rqstp;
   register SVCXPRT *transp;
{
   union {
      long str_date_1_arg;
   } argument;
   char *result;
   bool_t (*xdr_argument)(), (*xdr_result)();
   char *(*local)();

   switch (rqstp->rq_proc) {
   case NULLPROC:
      (void) svc_sendreply(transp, xdr_void, (char *)NULL);
      return;
   case BIN_DATE:
      xdr_argument = xdr_void;
      xdr_result = xdr_long;
      local = (char *(*)()) bin_date_1;
      break;
   case STR_DATE:
      xdr_argument = xdr_long;
      xdr_result = xdr_wrapstring;
      local = (char *(*)()) str_date_1;
      break;
   default:
      svcerr_noproc(transp);
      return;
   }
```



```
   bzero((char *)&argument, sizeof(argument));
   if (!svc_getargs(transp, xdr_argument, &argument)) {
      svcerr_decode(transp);
      return;
   }
   result = (*local)(&argument, rqstp);
   if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
      svcerr_systemerr(transp);
   }
   if (!svc_freeargs(transp, xdr_argument, &argument)) {
      fprintf(stderr, "unable to free arguments");
      exit(1);
   }
   return;
}
```

no rpcgen

- you could generate all of this by hand
- might want finer control
- more likely, let rpcgen genrate stubs then you edit them
- see man entries for *svc_sendreply(), svc_getargs(), svc_run(), svc_register(), svcXXX_create()*

example 2

from Sun web document cs494-unp/rpc_rls

- remote directory listing (rls hostname directory)
- example of linked-list XDR data type
- rpcgen will produce a rls_xdr.c file
- worry about freeing up XDR storage

```
you write:  rls.x rls.c  rls_svc_proc.c

rpcgen creates rls_clnt.c rls_svc.c rls.h rls_xdr.c

start server: rls_svc
run client:  rls serverhost directory
   file1
   file2
   file3
```


rls.x

```
const MAXNAMELEN = 255;
typedef string nametype<MAXNAMELEN>;   /* a directory entry */
typedef struct namenode *namelist;  /* a link in the listing */
struct namenode {
   nametype name;    /* name of directory entry */
   namelist pNext;      /* next entry */
};
/*
 * the result of a READDIR operation
 */
union readdir_res switch (int errno) {
case 0:
   namelist list;    /* no error: return directory listing */
default:
   void;       /* error occurred: nothing else to return */
};
/*
 * the directory program definition
 */
program DIRPROG {
   version DIRVERS {
      readdir_res
      READDIR(nametype) = 1;
   } = 1;
} = 0x20000001;
```


client: rls.c

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "rls.h"
main (argc, argv)
{
   CLIENT *cl;
   char *server;
   char *dir;
   readdir_res *result;
   namelist nl;
    cl = clnt_create(server, DIRPROG, DIRVERS, "tcp");
    if (cl == NULL) {
      clnt_pcreateerror(server);
      exit(1);
   }
   result = readdir_1(&dir, cl);
   if (result == NULL) {
      clnt_perror(cl, server);
      exit(1);
   }
   if (result->errno != 0) {
      errno = result->errno;
      perror(dir);
      exit(1);
   }
   for (nl = result->readdir_res_u.list; nl != NULL; nl = nl->pNext) {
      printf("%s\n", nl->name); }
}
```

server stubs

rls_svc_proc.c

```
...
readdir_res *
readdir_1(dirname)
   nametype *dirname;
{
   namelist nl;
   namelist *nlp;
   static readdir_res res; /* must be static! */
   static DIR *dirp = NULL;/* static to see if we've been called before */
   struct direct *d;
   dirp = opendir(*dirname);
   if (dirp == NULL) {
      res.errno = errno;
      return(&res);
   }
   if (dirp) xdr_free(xdr_readdir_res, &res);
   nlp = &res.readdir_res_u.list;
   while (d = readdir(dirp)) {
      nl = *nlp = (namenode *)malloc(sizeof(namenode));
      nl->name = strdup(d->d_name);
      nlp = &nl->pNext;
   }
   *nlp = NULL;
   res.errno = 0;
   closedir(dirp);
   return(&res);
}
```


makefile

```
BIN = rls rls_svc
GEN = rls_clnt.c rls_svc.c rls.h rls_xdr.c
LIB = -lrpcsvc
CFLAGS = -I.. -O -DRPC_SVC_FG
# for solaris to disable background fork -DRPC_SVC_FG

all:    $(BIN)

$(GEN): rls.x
        rpcgen rls.x

rls:    rls.o rls_clnt.o rls_xdr.o
        $(CC) -o $@ rls.o rls_clnt.o rls_xdr.o $(LIB)

rls_svc:        rls_svc_proc.o rls_svc.o rls_xdr.o
                $(CC) -o $@ rls_svc_proc.o rls_svc.o rls_xdr.o  $(LIB)
rls.o:  rls.c rls.h

rls_svc_proc.o: rls_svc_proc.c rls.h

clean:
        rm -f $(GEN) *.o $(BIN)
```


rpcgen: rls.h

```
#include <rpc/types.h>
#define MAXNAMELEN 255
typedef char *nametype;
bool_t xdr_nametype();
typedef struct namenode *namelist;
bool_t xdr_namelist();
struct namenode {
   nametype name;
   namelist pNext;
};
typedef struct namenode namenode;
bool_t xdr_namenode();
struct readdir_res {
   int errno;
   union {
      namelist list;
   } readdir_res_u;
};
typedef struct readdir_res readdir_res;
bool_t xdr_readdir_res();

#define DIRPROG ((u_long)0x20000001)
#define DIRVERS ((u_long)1)
#define READDIR ((u_long)1)
extern readdir_res *readdir_1();
```


rpcgen: rls_xdr.c

```
bool_t
xdr_nametype(xdrs, objp)
   XDR *xdrs;
   nametype *objp;
{
   if (!xdr_string(xdrs, objp, MAXNAMELEN)) {
      return (FALSE);
   }
   return (TRUE);
}

bool_t
xdr_namelist(xdrs, objp)
   XDR *xdrs;
   namelist *objp;
{
   if (!xdr_pointer(xdrs, (char **)objp, sizeof(struct namenode), xdr_namenode)) {
      return (FALSE);
   }
   return (TRUE);
}
```

```
bool_t
xdr_namenode(xdrs, objp)
   XDR *xdrs;
   namenode *objp;
{
   if (!xdr_nametype(xdrs, &objp->name)) {
      return (FALSE);
   }
   if (!xdr_namelist(xdrs, &objp->pNext)) {
      return (FALSE);
   }
   return (TRUE);
}

bool_t
xdr_readdir_res(xdrs, objp)
   XDR *xdrs;
   readdir_res *objp;
{
   if (!xdr_int(xdrs, &objp->errno)) {
      return (FALSE);
   }
   switch (objp->errno) {
   case 0:
      if (!xdr_namelist(xdrs, &objp->readdir_res_u.list)) {
         return (FALSE);
      }
      break;
   }
   return (TRUE);
}
```


rgcgen: client stubs

rls_clnt.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include "rls.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

readdir_res *
readdir_1(argp, clnt)
   nametype *argp;
   CLIENT *clnt;
{
   static readdir_res res;

   bzero((char *)&res, sizeof(res));
   if (clnt_call(clnt, READDIR, xdr_nametype,
     argp, xdr_readdir_res, &res, TIMEOUT) != RPC_SUCCESS) {
      return (NULL);
   }
   return (&res);
}
```


rpcgen:server main

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "rls.h"
static void dirprog_1();
main()
{
   register SVCXPRT *transp;
   (void) pmap_unset(DIRPROG, DIRVERS);
   transp = svcudp_create(RPC_ANYSOCK);
   if (transp == NULL) {
      fprintf(stderr, "cannot create udp service.");
      exit(1);
   }
   if (!svc_register(transp, DIRPROG, DIRVERS, dirprog_1, IPPROTO_UDP)) {
      fprintf(stderr, "unable to register (DIRPROG, DIRVERS, udp).");
      exit(1);
   }
   transp = svctcp_create(RPC_ANYSOCK, 0, 0);
   if (transp == NULL) {
      fprintf(stderr, "cannot create tcp service.");
      exit(1);
   }
   if (!svc_register(transp, DIRPROG, DIRVERS, dirprog_1, IPPROTO_TCP)) {
      fprintf(stderr, "unable to register (DIRPROG, DIRVERS, tcp).");
      exit(1);
   }
   svc_run();
}
```



```
static void
dirprog_1(rqstp, transp)
   struct svc_req *rqstp;
   register SVCXPRT *transp;
{
   union {
      nametype readdir_1_arg;
   } argument;
   char *result;
   bool_t (*xdr_argument)(), (*xdr_result)();
   char *(*local)();

   switch (rqstp->rq_proc) {
   case NULLPROC:
      (void) svc_sendreply(transp, xdr_void, (char *)NULL);
      return;
```

```
   case READDIR:
      xdr_argument = xdr_nametype;
      xdr_result = xdr_readdir_res;
      local = (char *(*)()) readdir_1;
      break;

   default:
      svcerr_noproc(transp);
      return;
   }
   bzero((char *)&argument, sizeof(argument));
   if (!svc_getargs(transp, xdr_argument, &argument)) {
      svcerr_decode(transp);
      return;
   }
   result = (*local)(&argument, rqstp);
   if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
      svcerr_systemerr(transp);
   }
   if (!svc_freeargs(transp, xdr_argument, &argument)) {
      fprintf(stderr, "unable to free arguments");
      exit(1);
   }
   return;
}
```


RPC review

- emulate function-call
- hide details of network
- hide details of hetergeneous machines
- several levels of implementation
- part of a DCE
- Sun RPC also has 3 authentication options: none, UNIX, DES

why not

- may not match application
- may need more control
- hard to debug/trace
- may not be supported
- too synchronous
- not good for high speed nets

next time

RPC implementation