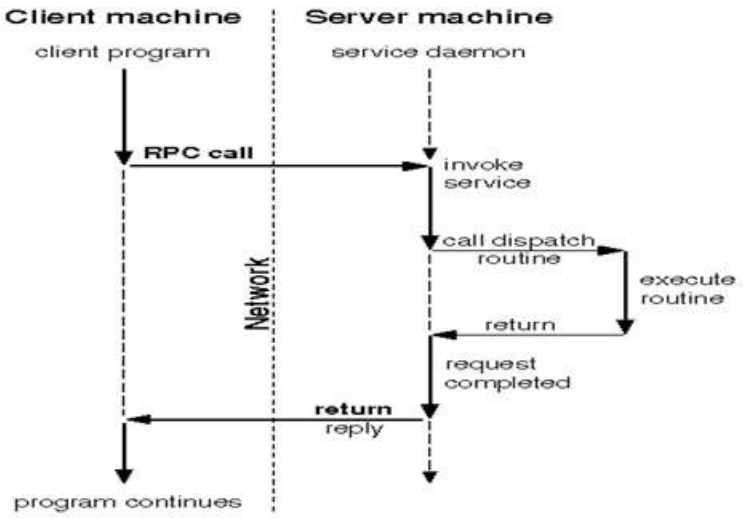
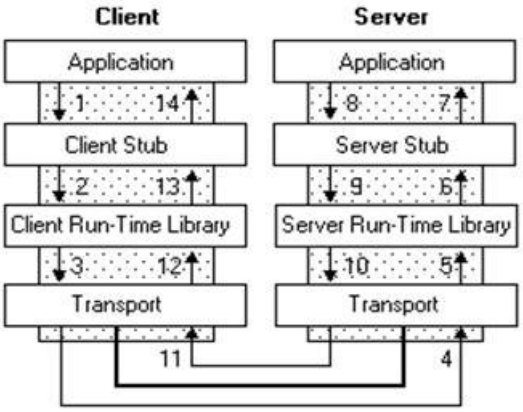
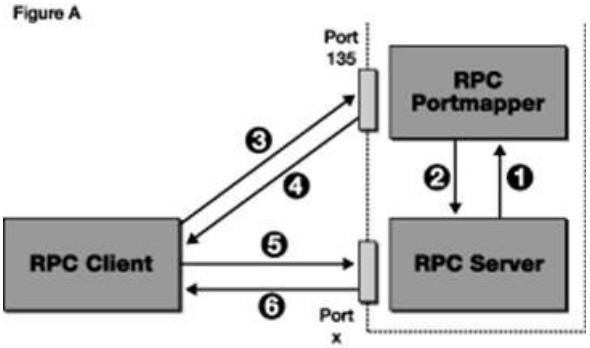
RPC (got these from web, but forgot to record website name)

Remote Procedure Call is a technique for building distributed systems, and client/server applications. It allows a program on one machine to call a subroutine on another machine.





http://www.winnetmag.com/Article/ArticleID/22206/22206.html



1 The RPC server starts and registers with the RPC Portmapper service. The RPC portmapper maps the RPC service

number to a port in the range 1024 to 65,535.

2. The portmapper returns the port to the RPC server. The first two steps are known as the RPC registration steps. The RPC client then wants to connect to the RPC server.

3. To find out the exact port on which the server is listening, the RPC client contacts the portmapper.

4. The portmapper then maps the RPC service number it received from the client to the server's port and returns the number to the client.

5. The RPC client connects to the server.

6. The server replies to the client.

Sun has control over program numbers **0x00000000** to **0x1fffffff**. The range **0x20000000** to **0x3fffffff** is for user use when testing new applications. Other ranges above are given out in blocks of 0x20000000.

Number Program Explanation

0x00100000 portmapper portmap sunrpc. Manages use of transport ports. Other Lookup services could be used here.

0x00100001 rstat_svc Remote statistics.

0x00100002 rusersd Remote users.

0x00100003 nfs Network file system.

….

RPC service is identified by its RPC program number, the version number and the transport address (i.e. the TCP or UDP port).

Port Mapper (PMAP version 2) is most often used as a lookup service where the server programs dynamically register their transport addresses (ports).

Client programs consult the lookup service vis the well-known port (111) and are given the temporary ports to use to access the server programs.

Port Mapper protocol **PMAP** uses TCP/UDP port 111 and allocates ports above 1024 to applications as and when the client requests it.

Now is the time to look at how to RPC!!!

This sample is largely taken from <http://it.becs.ac.in/rpc.pdf>
Note: I could not make Makefile work. And, I had to do *sudo apt-get install* things.

Procedures to write high level RPC.

Create ~.x file.

The ~.x file is shown below, "rpctest.x" for our example:

```
struct operands {               // This is a data structure that hops on the
int x;                          // internet.
int y;                          // This can be anything, array, string, float, ...
};
```

```
program SIM_PROG {              // This is to register server routines to the
version SIMP_VERSION {          // port mapper.
 int ADD(operands) = 1;         // Client give the program number 0x23456789 and
 int SUB(operands) = 2;         // the port mapper. Server looks at routine id
 } = 1;                         // ADD or SUB, then run it.
} = 0x23456789;
```

Run rpcgen (rpc compiler)

```
~/330/RPC$ rpcgen -C -a rpctest.x
```

This generates:
rpctest_client.c rpctest_clnt.c rpctest_server.c rpctest_svc.c rpctest_xdr.c

Do not modify:
rpctest_clnt.c rpctest_svc.c rpctest_xdr.c // These are for communications

We add our own code to rpctest_server.c and rpctest_client.c. The server program will have “insert server code here” and the client program will have sim_prog_1() where you add client code right before the call to the server.

OK, let's do rpctest_server.c

```
/*-----------------------------------------*/
/*        FILE rpctest_server.c            */
/*-----------------------------------------*/

int *
add_1_svc(operands *argp, struct svc_req *rqstp)    // *argp is an address variable
{                                                   // and used to point to address
        static int  result;                         // that stores struct operands.

        /*
         * insert server code here
         */

    /** my code begin **/

          printf("server running ADD\n");     // argp-> means, goto address pointed
          result = argp->x + argp->y;         // by argp and access x of operands.

    /** my code   end **/

        return &result;    // returning the location where computed value is stored
}


int *
sub_1_svc(operands *argp, struct svc_req *rqstp)
{
        static int  result;

        /*
         * insert server code here
         */
```

```
    /** my code begin **/
          printf("server running SUB\n");
          result = argp->x - argp->y;

    /** my code   end **/

        return &result;
}
```

So, only code I wrote is from “my code begin” to “my code end”.

Next, let's do rpctest_client.c

```
/*---------------------------------------*/
/*        FILE rpctest_client.c          */
/*---------------------------------------*/

/* FILE rpctest_client.c */

void
sim_prog_1(char *host)
{
 CLIENT *clnt;
 int  *result_1;
 operands  add_1_arg;
 int  *result_2;
 operands  sub_1_arg;

#ifndef DEBUG
 clnt = clnt_create (host, SIM_PROG, SIMP_VERSION, "udp");
 if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

/** My code here and there **/
   { int a, b;
   printf("Enter two integers a,b -> ");         // Do your own things here
   scanf("%d %d", &a, &b);                       // before you pass parameters
   add_1_arg.x = sub_1_arg.x = a;                // to remote procedure.
   add_1_arg.y = sub_1_arg.y = b;                // add_1_arg is not a pointer to
   }                                             // structure. It is a structure
                                                 // variable. Thus, '.' not '->'.

   result_1 = add_1(&add_1_arg, clnt);           //remote routine add_1() call
   if (result_1 == (int *) NULL) {               //&add_1_arg, give the address
          clnt_perror (clnt, "call failed");     //since address is expected.
   }                                             //'&' means address of.
   printf("sum = %d\n", *result_1);

   result_2 = sub_1(&sub_1_arg, clnt);           //remote routine sub_1() call
   if (result_2 == (int *) NULL) {
```

```
          clnt_perror (clnt, "call failed");
   }
   printf("dif = %d\n", *result_2);

#ifndef DEBUG
 clnt_destroy (clnt);
#endif   /* DEBUG */
}
```

After you did “rpcgen -C -a rpctest.x”, find out exactly where I added the code in rpctest_server.c and rpctest_client.c.

How to compile with gcc and run.

```
~/330/RPC$ gcc -C -g -DRPC_SVS_FG rpctest_client.c rpctest_clnt.c rpctest_xdr.c -o
test_client -lnsl

~/330/RPC$ gcc -C -g -DRPC_SVS_FG rpctest_server.c rpctest_svc.c rpctest_xdr.c -o
test_server -lnsl

If you get “unable to register” or “Port mapper failure  “, then do
sudo apt-get install rpcbind

If you get “Weak credential”, then do
sudo -i service portmap stop
sudo -i rpcbind -i -w
sudo -i service portmap start       // I forgot to do this, but it went fine.

to run:
~/330/RPC$./test_server
~/330/RPC$./test_client localhost
```