**Code:**

*x-file:*

struct nums {

int x;

int y;

};

program ADD\_PROG {

version ADD\_VERS {

int add(nums)=1;

}=1;

}=0x12341234;

*Client:*

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include <memory.h> /\* for memset \*/

#include "add.h"

/\* Default timeout can be changed using clnt\_control() \*/

static struct timeval TIMEOUT = { 25, 0 };

int \*

add\_1(nums \*argp, CLIENT \*clnt)

{

static int clnt\_res;

memset((char \*)&clnt\_res, 0, sizeof(clnt\_res));

if (clnt\_call (clnt, add,

(xdrproc\_t) xdr\_nums, (caddr\_t) argp,

(xdrproc\_t) xdr\_int, (caddr\_t) &clnt\_res,

TIMEOUT) != RPC\_SUCCESS) {

return (NULL);

}

return (&clnt\_res);

}

*Server:*

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include "add.h"

#include <stdio.h>

#include <stdlib.h>

#include <rpc/pmap\_clnt.h>

#include <string.h>

#include <memory.h>

#include <sys/socket.h>

#include <netinet/in.h>

#ifndef SIG\_PF

#define SIG\_PF void(\*)(int)

#endif

static void

add\_prog\_1(struct svc\_req \*rqstp, register SVCXPRT \*transp)

{

union {

nums add\_1\_arg;

} argument;

char \*result;

xdrproc\_t \_xdr\_argument, \_xdr\_result;

char \*(\*local)(char \*, struct svc\_req \*);

switch (rqstp->rq\_proc) {

case NULLPROC:

(void) svc\_sendreply (transp, (xdrproc\_t) xdr\_void, (char \*)NULL);

return;

case add:

\_xdr\_argument = (xdrproc\_t) xdr\_nums;

\_xdr\_result = (xdrproc\_t) xdr\_int;

local = (char \*(\*)(char \*, struct svc\_req \*)) add\_1\_svc;

break;

default:

svcerr\_noproc (transp);

return;

}

memset ((char \*)&argument, 0, sizeof (argument));

if (!svc\_getargs (transp, (xdrproc\_t) \_xdr\_argument, (caddr\_t) &argument)) {

svcerr\_decode (transp);

return;

}

result = (\*local)((char \*)&argument, rqstp);

if (result != NULL && !svc\_sendreply(transp, (xdrproc\_t) \_xdr\_result, result)) {

svcerr\_systemerr (transp);

}

if (!svc\_freeargs (transp, (xdrproc\_t) \_xdr\_argument, (caddr\_t) &argument)) {

fprintf (stderr, "%s", "unable to free arguments");

exit (1);

}

return;

}

int

main (int argc, char \*\*argv)

{

register SVCXPRT \*transp;

pmap\_unset (ADD\_PROG, ADD\_VERS);

transp = svcudp\_create(RPC\_ANYSOCK);

if (transp == NULL) {

fprintf (stderr, "%s", "cannot create udp service.");

exit(1);

}

if (!svc\_register(transp, ADD\_PROG, ADD\_VERS, add\_prog\_1, IPPROTO\_UDP)) {

fprintf (stderr, "%s", "unable to register (ADD\_PROG, ADD\_VERS, udp).");

exit(1);

}

transp = svctcp\_create(RPC\_ANYSOCK, 0, 0);

if (transp == NULL) {

fprintf (stderr, "%s", "cannot create tcp service.");

exit(1);

}

if (!svc\_register(transp, ADD\_PROG, ADD\_VERS, add\_prog\_1, IPPROTO\_TCP)) {

fprintf (stderr, "%s", "unable to register (ADD\_PROG, ADD\_VERS, tcp).");

exit(1);

}

svc\_run ();

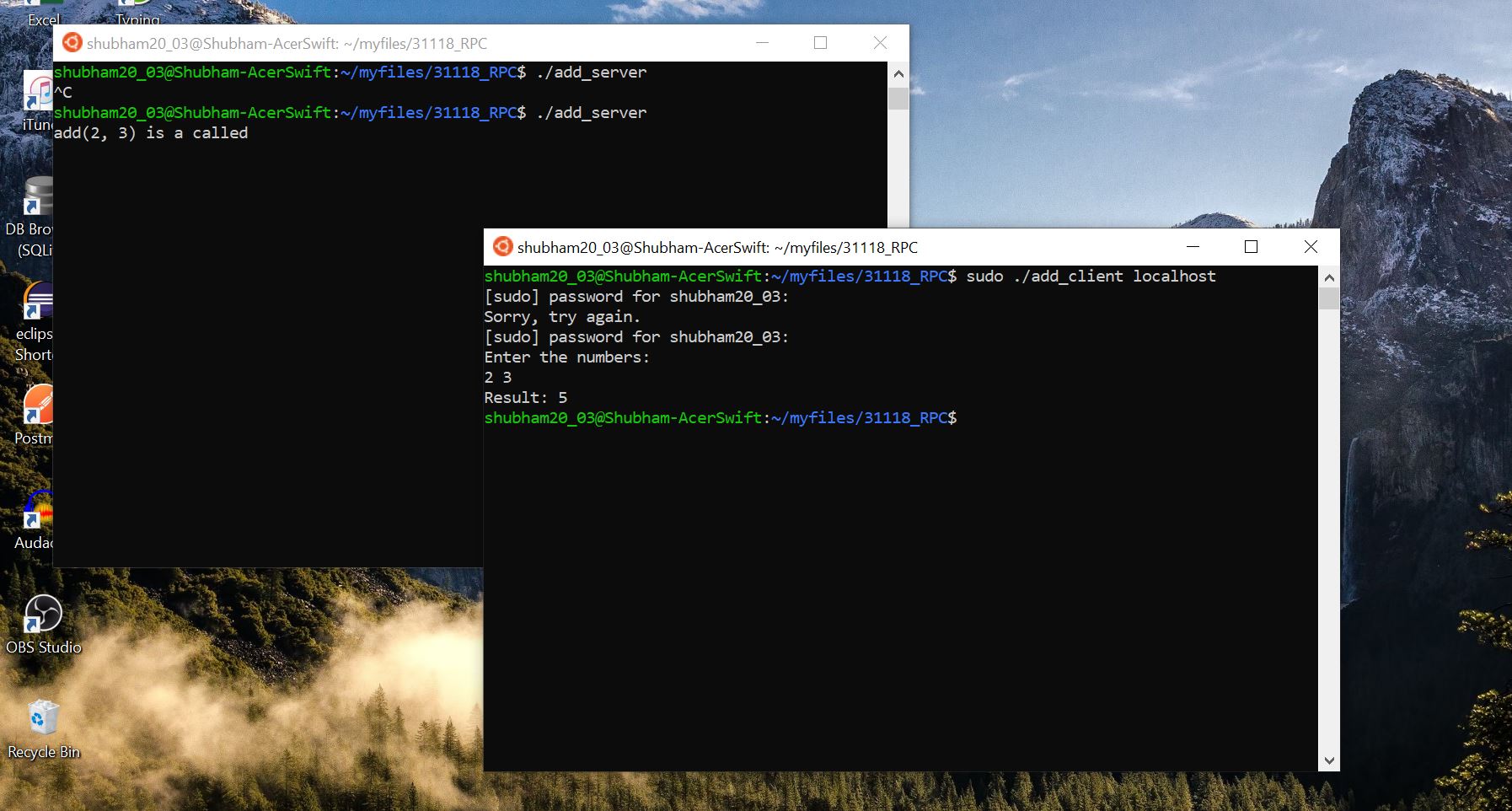
fprintf (stderr, "%s", "svc\_run returned");

exit (1);

/\* NOTREACHED \*/

}

**Output:**

****