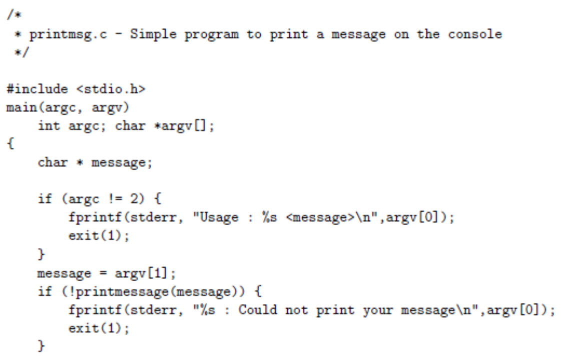
Writing a (Sun) RPC Application

Haodong Wang

A simple appplication



```
/*
 * printmsg.c - Simple program to print a message on the console
 */

#include <stdio.h>
main(argc, argv)
    int argc; char *argv[];
{
    char * message;

    if (argc != 2) {
        fprintf(stderr, "Usage : %s <message>\n",argv[0]);
        exit(1);
    }
    message = argv[1];
    if (!printmessage(message)) {
        fprintf(stderr, "%s : Could not print your message\n",argv[0]);
        exit(1);
    }
    printf("Message delivered\n");
    exit(0);
}

/*  Print a message on the console */
printmessage( char * msg)
{
    FILE *f;

    f = fopen("/dev/console","w");

    if (f == NULL) {
        return(0);
    }
    fprintf(f,"%s\n",msg);
    fclose(f);
    return(1);
}
```

Hook an Eterm to `/dev/console` with `Eterm -C`.

Run `./printmsg "testing"`

Output on the Eterm.

Want to provide a (trivial) service on a given system on the network. That service allows remote users to write to the local console on the server system.

Want procedural interface for clients who wish to use the service.

Use `rpcgen(1)` to structure both server and client code: `rpcgen` builds the RPC stub code.

We provide `msg.x`, protocol spec that `rpcgen` understands:

```
/*
 * msg.x - Remote message protocol file
 */

program MESSAGEPROG {
    version PRINTMESSAGEVERSION {
        int PRINTMESSAGE(string) = 1;
    } = 1;
} = 0x20000099;
```

RPC programs declared with following syntax:

```
program-definition:
   program program-ident {
      version-list
   } = value

version-list:
   version ;
   version ; version-list

version:
   version version-ident {
      procedure-list
   } = value

procedure-list:
   procedure ;
   procedure ; procedure-list

procedure
   type-ident procedure-ident ( type-ident ) = value
```

Remote program MESSAGEPROG

Version 1 of the remote program . . . multiple versions can coexist.

Remote procedure PRINTMESSAGE, taking a string as parameter, returning an int, is procedure 1 of the remote program.

`rpcgen` will generate name of procedure to be called remotely by taking name of procedure in .x file, converting it to lowercase, appending an underscore, and appending the version number. In our case, `printmessage_1()`, for a client call. For the server, `printmessage_1_svc()`.

All RPC services provide procedure 0 for pinging the program. `rpcgen` creates this automatically. Merely returns, without a value.

Remote program MESSAGEPROG is associated with the 32-bit int 0x20000099 (its handle to the RPC dispatcher).

0x00000000 - 0x1fffffff	:	Defined by SUN
0x20000000 - 0x3fffffff	:	Defined by users
0x40000000 - 0x5fffffff	:	Transient (Reserved for customer written application)
0x60000000 - 0xffffffff	:	reserved

The remotely-callable version of printmsg:

```
/*
 * msg_proc.c - RPC Version of printmsg.c
 */
#include <rpc/rpc.h>
#include <stdio.h>
#include "msg.h"

int *
printmessage_1_svc(msg, rqstp)  /* NOTE */
    char **msg;                 /* NOTE */
    struct svc_req *rqstp;      /* NOTE */
{
    FILE *f; static int result; /* NOTE */

    f = fopen("/dev/console","w");

    if (f == (FILE *) NULL) {
        result = 0;
        return(&result);        /* NOTE */
    }
    fprintf(f,"%s\n",*msg);
    fclose(f);
    result = 1;
    return(&result);            /* NOTE */
}
```

Note the four differences between locally-callable and remotely callable `printmsg()`.

1. (First) argument now a `char **`.
2. Additional argument, a `struct svc_req *`.
3. `result` is a `static int`.
4. Return address of `result`, not its value.

Only makes sense in context of `rpcgen`; i.e., the fact that this will be invoked via RPC is explicitly known to the programmer.

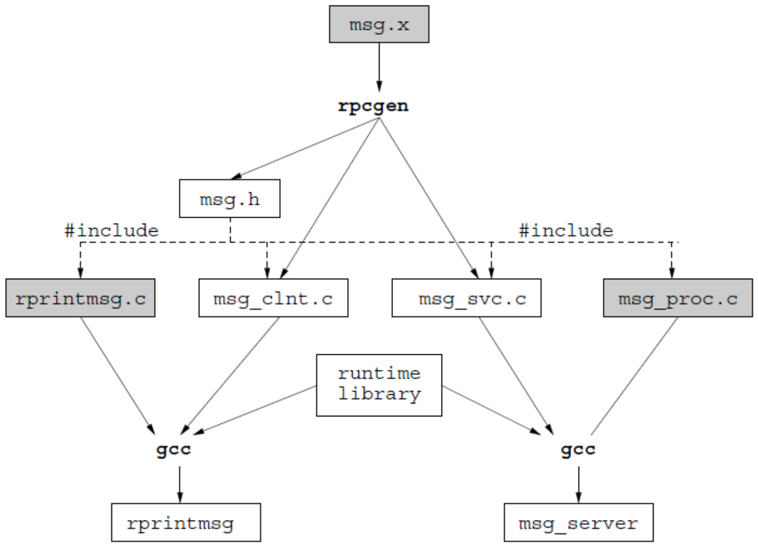
Client (caller) code:

```
/*
 * rprintmsg.c - Remote version of "printmsg.c"
 */
#include <rpc/rpc.h>
#include <stdio.h>
#include "msg.h"
int
main(argc, argv)
    int argc; char *argv[];
{
    CLIENT *c1;                 /* NOTE */
    int *result;
    char *server;
    char *message;

    if (argc != 3) {
        fprintf(stderr, "Usage : %s <host> <message>\n",argv[0]);
        exit(1);
    }
    server = argv[1];
    message = argv[2];

    /* Create the client handle used for calling MESSAGEPROG on the server */
    c1 = clnt_create(server, MESSAGEPROG, PRINTMESSAGEVERSION, "tcp");
    /* Check if connection is established to the server */
    if (c1 == NULL) {
        clnt_pcreateerror(server);
        exit(1);
    }
    /* Call the remote procedure */
    result = printmessage_1(&message, c1);
    if (result == NULL) {
        /* Error occurred while calling the server */
        clnt_perror(c1, server);
        exit(1);
    }
    /* RPC worked; check if the message was delivered */
    if (*result == 0) {
        fprintf(stderr,"%s : Could not print your message\n",argv[0]);
        exit(1);
    };
    /* Everything (RPC, message delivery) worked */
    printf("Message delivered to %s\n",server);
    exit(0);
}
```

Note substantial differences from local call.

`c1` is a `CLIENT *`, a client handle. Used to get at details of this RPC.

- `% rpcgen msg.x` produces `msg.h`, `msg_svc.c`, and `msg_clnt.c`
- `% gcc msg_proc.c msg_svc.c -o msg_server` produces the server
- `% gcc rprintmsg.c msg_clnt.c -o rprintmsg` produces a client capable of calling the `printmsg()` function remotely
- Run `msg_server` on server system; bring up a console Eterm on the server system; run `rprintmsg server_sys "test"`. Look for output message on Eterm.



msg.h (generated by rpcgen):

```
#ifndef _MSG_H_RPCGEN
#define _MSG_H_RPCGEN
#include <rpc/rpc.h>
#ifdef __cplusplus
extern "C" {
#endif

#define MESSAGEPROG 0x20000099
#define PRINTMESSAGEVERSION 1

#if defined(__STDC__) || defined(__cplusplus)
#define PRINTMESSAGE 1
extern  int * printmessage_1(char **, CLIENT *);
extern  int * printmessage_1_svc(char **, struct svc_req *);
extern int messageprog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);
#else /* K&R C */
#define PRINTMESSAGE 1
extern  int * printmessage_1();
extern  int * printmessage_1_svc();
extern int messageprog_1_freeresult ();
#endif /* K&R C */
#ifdef __cplusplus
}
#endif
#endif /* !_MSG_H_RPCGEN */
```

msg_clnt.c:

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */
#include <memory.h> /* for memset */
#include "msg.h"

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

int *
printmessage_1(char **argp, CLIENT *clnt)
{
        static int clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, PRINTMESSAGE,
                (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
                (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

msg_svc.c:

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

#include "msg.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
messageprog_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
        union {
                char *printmessage_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 PRINTMESSAGE:
		_xdr_argument = (xdrproc_t) xdr_wrapstring;
		_xdr_result = (xdrproc_t) xdr_int;
		local = (char *(*)(char *, struct svc_req *)) printmessage_1_svc;
		break;

	default:
		svcerr_noproc (transp);
		return;
	}
	memset ((char *)&argument, 0, sizeof (argument));
	if (!svc_getargs (transp, _xdr_argument, (caddr_t) &argument)) {
		svcerr_decode (transp);
		return;
	}
	result = (*local)((char *)&argument, rqstp);
	if (result != NULL && !svc_sendreply(transp, _xdr_result, result)) {
```

```
		svcerr_systemerr (transp);
	}
	if (!svc_freeargs (transp, _xdr_argument, (caddr_t) &argument)) {
		fprintf (stderr, "unable to free arguments");
		exit (1);
	}
	return;
}

int
main (int argc, char **argv)
{
	register SVCXPRT *transp;

	pmap_unset (MESSAGEPROG, PRINTMESSAGEVERSION);

	transp = svcudp_create(RPC_ANYSOCK);
	if (transp == NULL) {
		fprintf (stderr, "cannot create udp service.");
		exit(1);
	}
	if (!svc_register(transp, MESSAGEPROG, PRINTMESSAGEVERSION, messageprog_1, IPPROTO_UDP)) {
		fprintf (stderr, "unable to register (MESSAGEPROG, PRINTMESSAGEVERSION, 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, MESSAGEPROG, PRINTMESSAGEVERSION, messageprog_1, IPPROTO_TCP)) {
                fprintf (stderr, "unable to register (MESSAGEPROG, PRINTMESSAGEVERSION, tcp).");
                exit(1);
        }

        svc_run ();
        fprintf (stderr, "svc_run returned");
        exit (1);
        /* NOTREACHED */
}
```

See if the printmessage service is available:

```
/*
 * rprintping.c - Ping printmsg service on server
 */

#include <rpc/rpc.h>
#include <rpc/xdr.h>
#include <stdio.h>
#include "msg.h"

main(argc, argv)
    int argc; char *argv[];
{
    CLIENT *c1;
    int  result;
    char *server;

    if (argc != 2) {
        fprintf(stderr, "Usage : %s <host>\n",argv[0]);
        exit(1);
    }
    server = argv[1];

    /* Create the client handle used for pinging MESSAGEPROG on the server */
    c1 = clnt_create(server, MESSAGEPROG, PRINTMESSAGEVERSION, "udp");

    /* Check if connection is established to the server */
    if (c1 == NULL) {
        clnt_pcreateerror(server);
        exit(1);
    }

    /* Call the remote procedure */
    result = callrpc(server,MESSAGEPROG,PRINTMESSAGEVERSION,0,
                     xdr_void,NULL,xdr_void,NULL);
    if (result != 0) {
        /* Error occurred while calling the server */
        clnt_perror(c1, server);
        exit(1);
    }
    /* Everything (RPC, message delivery) worked */
    printf("Found printmsg service on %s\n",server);
    exit(0);
}
```

```
jax% rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100021    1   udp   1024  nlockmgr
    100021    3   udp   1024  nlockmgr
    100024    1   udp    991  status
    100024    1   tcp    993  status
 536871065    1   udp   1033
 536871065    1   tcp   1797
jax% ./rprintping localhost
Found printmsg service on localhost
jax% rpcinfo -u localhost 536871065
program 536871065 version 1 ready and waiting
jax% rpcinfo -t localhost 536871065
program 536871065 version 1 ready and waiting
jax% ./rprintmsg localhost help
Message delivered to localhost
jax% ps ax | grep msg_server | grep -v grep
 2521 pts/3    S      0:00 ./msg_server
jax% kill 2521
jax% ps ax | grep msg_server | grep -v grep
jax% rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100021    1   udp   1024  nlockmgr
    100021    3   udp   1024  nlockmgr
    100024    1   udp    991  status
    100024    1   tcp    993  status
 536871065    1   udp   1033
 536871065    1   tcp   1797
jax% rpcinfo -u localhost 536871065
rpcinfo: RPC: Unable to receive; errno = Connection refused
program 536871065 version 0 is not available
jax% rpcinfo -d 536871065 1
Sorry. You are not root
jax% su
Password:
[root@jax linux]# rpcinfo -d 536871065 1
[root@jax linux]# rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100021    1   udp   1024  nlockmgr
    100021    3   udp   1024  nlockmgr
    100024    1   udp    991  status
    100024    1   tcp    993  status
[root@jax linux]#
```

Slightly more complex example (from Stevens):

square.x:

```
struct square_in {           /* input arg */
  long arg1;
};

struct square_out {          /* retval */
  long res1;
};

program SQUARE_PROG {
  version SQUARE_VERS {
        square_out SQUAREPROC(square_in) = 1 ; /* procedure number 1 */
  } = 1;                                       /* version 1 */
} = 0x31230000;                                /* program number */
```

client.c

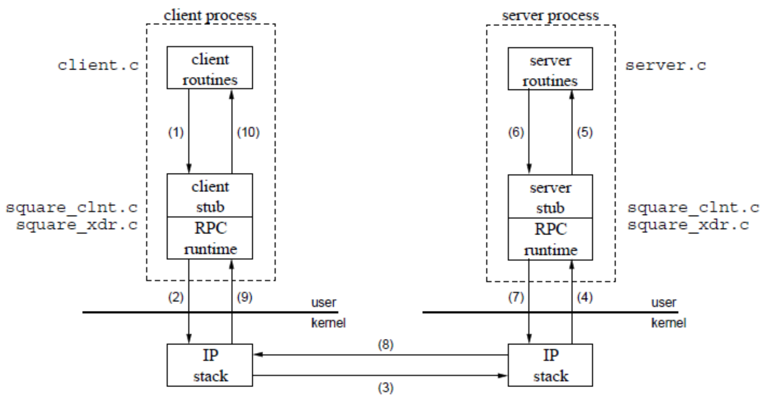
```
#include "square.h"
#include <rpc/rpc.h>
#include <stdio.h>

int
main(int argc, char **argv)
{
  CLIENT  *cl;
  square_in in; square_out *outp;

  if (argc != 3) {
    fprintf(stderr, "usage: sq_client hostname int_value\n"); exit(1);
  }

  cl = clnt_create(argv[1], SQUARE_PROG, SQUARE_VERS, "tcp");

  in.arg1 = atol(argv[2]);
  if ( (outp = squareproc_1(&in, cl)) == NULL) {
    fprintf(stderr, "sq_client: bad rpc\n"); exit(1);
  }

  printf("result=%ld\n", outp->res1);
  return(0);
}
```

server.c:

```
#include "square.h"
#include <rpc/rpc.h>
#include <stdio.h>

square_out *
squareproc_1_svc(square_in *inp, struct svc_req *rqstp)
{
  static square_out out;

  out.res1 = inp->arg1 * inp->arg1;
  return(&out);
}
```

Compiling:

```
jax% gcc -c client.c
jax% gcc -c square_clnt.c
jax% gcc -c square_xdr.c
jax% gcc -o client client.o square_clnt.o square_xdr.o
jax% gcc -c server.c
jax% gcc -c square_svc.c
jax% gcc -o server server.o square_svc.o square_xdr.o
```


client process
client routines
client.c
(1)
(10)
client stub
RPC runtime
square_clnt.c
square_xdr.c
(2)
(9)
user
kernel
IP stack
server process
server routines
server.c
(6)
(5)
server stub
RPC runtime
square_clnt.c
square_xdr.c
(7)
(4)
user
kernel
IP stack
(8)
(3)