Converting Local Procedures into Remote Procedures Example

This example illustrates one way to convert an application that runs on a single machine into one that runs over a network. For instance, a programmer first creates a program that prints a message to the console, as follows:

```
/*
 * printmsg.c: 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: couldn't print your
       message\n", argv[0]);
         exit(1);
    }
    printf("Message Delivered!\n");
    exit(0);
}
/*
 * Print a message to the console.
 * Return a boolean indicating whether the
 * message was actually printed.
 */
printmessage(msg)
    char *msg;
{
    FILE *f;
    f = fopen("/dev/console", "w");
    if (f == NULL) {
        return (0);
    }
    fprintf(f, "%s\n", msg);
    fclose(f);
    return(1);
}
```

The reply message follows:

```
example%  cc printmsg.c -o printmsg
example%  printmsg "Hello, there."
```

```
Message delivered!
example%
```

If the **printmessage** program is turned into a remote procedure, it can be called from anywhere in the network. Ideally, one would insert a keyword such as **remote** in front of a procedure to turn it into a remote procedure. Unfortunately, the constraints of the C language do not permit this. However, a procedure can be made remote without language support.

To do this, the programmer must know the data types of all procedure inputs and outputs. In this case, the **printmessage** procedure takes a string as input and returns an integer as output. Knowing this, the programmer can write a protocol specification in Remote Procedure Call language (RPCL) that describes the remote version of `PRINTMESSAGE`, as follows:

```
/*
 * msg.x: Remote message printing protocol
 */
program MESSAGEPROG {
   version MESSAGEVERS {
       int PRINTMESSAGE(string) = 1;
   } = 1;
} = 99;
```

Remote procedures are part of remote programs, so the above protocol declares a remote program containing the single procedure `PRINTMESSAGE`. This procedure was declared to be in version 1 of the remote program. No null procedure (procedure 0) is necessary, because the **rpcgen** command generates it automatically.

Conventionally, all declarations are written with capital letters.

The argument type is `string` and not `char *` because a `char *` in C is ambiguous. Programmers usually intend it to mean a null-terminated string of characters, but it could also represent a pointer to a single character or a pointer to an array of characters. In RPCL, a null-terminated string is unambiguously called a `string`.

Next, the programmer writes the remote procedure itself. The definition of a remote procedure to implement the `PRINTMESSAGE` procedure declared previously can be written as follows:

```
/*
 * msg_proc.c: implementation of the remote
 *procedure "printmessage"
 */
#include <stdio.h>
#include <rpc/rpc.h>   /* always needed  */
#include "msg.h"       /* msg.h will be generated by rpcgen */

/*
 * Remote version of "printmessage"
 */ int *
printmessage_1(msg)
    char **msg;
{

    static int result;  /* must be static! */
    FILE *f;
```

```
    f = fopen("/dev/console", "w");
    if (f == NULL) {
        result = 0;
        return (&result);
    }
    fprintf(f, "%s\en", *msg);
    fclose(f);
    result = 1;
    return (&result);
}
```

The declaration of the remote procedure `printmessage_1` in this step differs from that of the local procedure `printmessage` in the first step, in three ways:

- It takes a pointer to a string instead of the string itself. This is true of all remote procedures, which always take pointers to their arguments rather than the arguments themselves.
- It returns a pointer to an integer instead of the integer itself. This is also true of remote procedures, which generally return a pointer to their results.
- It has a `_1` appended to its name. Remote procedures called by the **rpcgen** command are named by the following rule: the name in the program definition (here `PRINTMESSAGE`) is converted to all lowercase letters, and an _ (underscore) and the version number are appended.

Finally, the programmers declare the main client program that will call the remote procedure, as follows:

```
/*
 * rprintmsg.c: remote version of "printmsg.c"
 */
#include <stdio.h>
#include <rpc/rpc.h>     /* always needed */
#include "msg.h"  /* msg.h will be generated by rpcgen */
main(argc, argv)
    int argc;
    char *argv[];
{
    CLIENT *cl;
    int *result;
    char *server;
    char *message;
    if (argc < 3) {
        fprintf(stderr,
        "usage: %s host message\en", argv[0]);
        exit(1);
    }

    /*
     * Save values of command line arguments
     */
    server = argv[1];
    message = argv[2];
    /*
     * Create client "handle" used for calling MESSAGEPROG on
     * the server designated on the command line.  We tell
     * the RPC package to use the "tcp" protocol when
     * contacting the server.
     */
    cl = clnt_create(server, MESSAGEPROG, MESSAGEVERS, "tcp");
    if (cl == NULL) {
         /*
```

```
            * Couldn't establish connection with server.
            * Print error message and die.
            */
            clnt_pcreateerror(server);
            exit(1);
      }
     /*
      * Call the remote procedure "printmessage" on the server
      */
      result = printmessage_1(&message, cl);
      if (result == NULL) {
          /*
           * An error occurred while calling the server.
           * Print error message and die.
           */
           clnt_perror(cl, server);
           exit(1);
      }
      /*
       * Okay, we successfully called the remote procedure.
       */
      if (*result == 0) {
          /*
           * Server was unable to print our message.
           * Print error message and die.
           */
      fprintf(stderr, "%s: %s couldn't print your message\n",
          argv[0], server);
          exit(1);
      }
      /*
       * The message got printed on the server's console
       */
       printf("Message delivered to %s!\n", server);
       exit(0);
}
```

Notes:

1. First a client handle is created using the Remote Procedure Call (RPC) library **clnt_create** routine. This client handle is passed to the stub routines that call the remote procedure.
2. The remote procedure `printmessage_1` is called exactly the same way as it is declared in the **msg_proc.c** program, except for the inserted client handle as the first argument.

The client program **rprintmsg** and the server program **msg_server** are compiled as follows:

```
example%  rpcgen msg.x
example%  cc rprintmsg.c msg_clnt.c -o rprintmsg
example%  cc msg_proc.c msg_svc.c -o msg_server
```

Before compilation, however, the **rpcgen** protocol compiler is used to perform the following operations on the **msg.x** input file:

- It creates a header file called **msg.h** that contains **#define** statements for `MESSAGEPROG`, `MESSAGEVERS`, and `PRINTMESSAGE` for use in the other modules.
- It creates a client stub routine in the **msg_clnt.c** file. In this case, there is only one stub routine, the

printmessage_1, which is referred to from the **printmsg** client program. The name of the output file for client stub routines is always formed in this way. For example, if the name of the input file is **FOO.x**, the client stub's output file would be called **FOO_clnt.c**.

- It creates the server program that calls printmessage_1 in the **msg_proc.c** file. This server program is named **msg_svc.c**. The rule for naming the server output file is similar to the previous one. For example, if an input file is called **FOO.x**, the output server file is named **FOO_svc.c**.
