A Simple Application

Ron Burback

Lecture 16

The Application

- Simple math application
- Add two integer arrays

Steps needed

- Define the interface, idl file
- Write the client, easy
- Write the server
  - application production function, easy
  - Modify a template for the DCE interface
- Establish the name in the name space
- Build the client and server
- Evoke the server
- Evoke the client

The IDL: arithmetic.idl

```
/* FILE NAME: arithmetic.idl */
/* This Interface Definition Language file represents a basic arithmetic    */
/* procedure that a remote procedure call application can use.              */
[
uuid(C985A380-255B-11C9-A50B-08002B0ECEF1), version(1.0)         /* Universal Unique ID */
]
interface arithmetic                          /* interface name is arithmetic */
{
   const unsigned short ARRAY_SIZE = 10;/* an unsigned integer constant   */
   typedef long long_array[ARRAY_SIZE]; /* an array type of long integers */

   void sum_arrays ( /* The sum_arrays procedure does not return a value  */
      [in] long_array a,                     /* 1st parameter is passed in  */
      [in] long_array b,                     /* 2nd parameter is passed in  */
      [out] long_array c                     /* 3rd parameter is passed out */
   );
}
```

Run the IDL compiler

- generates
  - header used by client and server
  - client stub
    - Do nothing, just compile and link
  - server template
    - Have to modify to meet your needs

You get to write the client code

```
/* FILE NAME: client.c */
/* This is the client module of the arithmetic example. */
#include <stdio.h>
#include "arithmetic.h"    /* header file created by IDL compiler */

long_array a ={100,200,345,23,67,65,0,0,0,0};
long_array b ={4,0,2,3,1,7,5,9,6,8};

main ()
{
   long_array    result;
   int i;

   sum_arrays(a, b, result);           /* A Remote Procedure Call */
   puts("sums:");
   for(i = 0; i < ARRAY_SIZE; i++)
      printf("%ld\n", result[i]);
}
```

Easy! Just like a function call.

You get to modify the server template. Step 1

```
/* FILE NAME: server.c */
#include <stdio.h>
#include "arithmetic.h"               /* header created by the idl compiler */
#include "check_status.h"             /* header with the CHECK_STATUS macro */

main ()
{
   unsigned32           status;                   /* error status (nbase.h) */
   rpc_binding_vector_t *binding_vector; /*set of binding handles(rpcbase.h)*/
   unsigned_char_t      *entry_name; /*entry name for name service (lbase.h)*/
   char *getenv();

   rpc_server_register_if(     /* register interface with the RPC runtime */
      arithmetic_v0_0_s_ifspec,   /* interface specification (arithmetic.h) */
      NULL,
      NULL,
      &status                                               /* error status */
   );
   CHECK_STATUS(status, "Can't register interface\n", ABORT);
```

Tell the local dce run time about this service. arithmetic_v0_0_s_ifspec was built by IDL includes the UUID, version, entry point vector.

You get to modify the server template. Step 2

```
rpc_server_use_all_protseqs(             /* create binding information */
      rpc_c_protseq_max_reqs_default,   /* queue size for calls (rpcbase.h) , if out of threads*/
      &status
   );
   CHECK_STATUS(status, "Can't create binding information\n", ABORT);

   rpc_server_inq_bindings(   /* obtain this server's binding information */
      &binding_vector,
      &status
   );
   CHECK_STATUS(status, "Can't get binding information\n", ABORT);
```

Are we going to talk TCP? UDP? What Network Protocol?

rpc_server_inq_bindings gets back the binding vectors from the runtime. This takes into account the IP and Protocol sequence. The binding vector is then used in other calls to the runtime.

You get to modify the server template. Step 3

```
entry_name = (unsigned_char_t *)getenv("ARITHMETIC_SERVER_ENTRY");
   rpc_ns_binding_export(       /* export entry to name service database  */
      rpc_c_ns_syntax_default,     /* syntax of the entry name  (rpcbase.h) */
      entry_name,                  /* entry name for name service           */
      arithmetic_v0_0_s_ifspec,    /* interface specification (arithmetic.h)*/
      binding_vector,              /* the set of server binding handles     */
      NULL,
      &status
   );
   CHECK_STATUS(status, "Can't export to name service database\n", ABORT);

   rpc_ep_register(           /* register endpoints in local endpoint map */
      arithmetic_v0_0_s_ifspec,   /* interface specification (arithmetic.h) */
      binding_vector,             /* the set of server binding handles      */
      NULL,
      NULL,
      &status
   );
   CHECK_STATUS(status, "Can't add address to the endpoint map\n", ABORT);
```

We now advertise to the name space(CDS) our information.
We now establish a dynamic port from the end point mapper.

You get to modify the server template. Step 4

```
puts("Listening for remote procedure calls...");
   rpc_server_listen(                          /* listen for remote calls  */
      rpc_c_listen_max_calls_default, /*concurrent calls to server (rpcbase.h)*/
      &status
   );
   CHECK_STATUS(status, "rpc listen failed\n", ABORT);
}
```

The service is now ready and listening for request.
Each request is handled in a thread.
The runtime will call, using end EPV, the correct function.
This service will not return unless an out of bound interrupt occurs. For example, stop service from the dcecp.

Write the service.

```
/* FILE NAME: procedure.c */
/* An implementation of the procedure defined in the arithmetic interface. */
#include <stdio.h>
#include "arithmetic.h"           /* header file produced by IDL compiler */

void sum_arrays(a, b, c)    /* implementation of the sum_arrays procedure */
   long_array a;
   long_array b;
   long_array c;
   {
      int i;

      for(i = 0; i < ARRAY_SIZE; i++)
         c[i] = a[i] + b[i];    /* array elements are each added together */
   }
```

Easy!

Establish name in the CDS

Easy!

Done for you by the rpc_ns_binding_export call. You must have the correct ACL permissions

Build the client and server

```
# FILE NAME: Makefile
# Makefile for the arithmetic application
# definitions for this make file
CC=/usr/local/bin/gcc
CFLAGS=-g -D_REENTRANT -DSUNOS5
LIBS=-ldce -lnsl -lthread -lm -lsocket
INCL=-I.
IDLCMD=idl -v -keep all -cc_cmd "$(CC) -c  $(CFLAGS)"
SYS_TYPE=sun4m_53
APPL=arithmetic
# COMPLETE BUILD of the application
all:    local interface client server
#
# LOCAL BUILD of the client application to test locally
#
local:  interface client.c procedure.c
           $(CC)   -DLOCAL -o local_client client.c procedure.c
# remove object files so they do not interfere with a real build
           rm client.o procedure.o
#
# INTERFACE BUILD
#
interface:      $(APPL).h $(APPL)_cstub.o $(APPL)_sstub.o
$(APPL).h $(APPL)_cstub.o $(APPL)_sstub.o:      $(APPL).idl
           $(IDLCMD) $(APPL).idl
#
# CLIENT BUILD
#
client: $(APPL).h client.o $(APPL)_cstub.o
           $(CC) $(CFLAGS) -o client client.o $(APPL)_cstub.o $(LIBS)
#
# SERVER BUILD
#
server: $(APPL).h server.o procedure.o $(APPL)_sstub.o
           $(CC) $(CFLAGS) -o server server.o procedure.o $(APPL)_sstub.o $(LIBS)
```

Evoke the server

```
lunaris> setenv ARITHMETIC_ERVER_ENTRY /.:/arithmetic_serverhost
lunaris> ./server
Listening for remote procedure calls...
```

Evoke the client

```
lunaris> setenv RPC_DEFAULTENTRY /.:/arithmetic_serverhost
lunaris> ./client
sums:
104
200
347
26
68
72
5
9
6
8
lunaris>
```