ECES 338 Assignment #7

Due: April 9, 1999

Spring 99, Ozsoyoglu, G., 100 points

In this assignment, you will implement the Printer Daemon Problem (from Assignment 4, Question #3) using SOLARIS remote procedure calls (RPCs). Assume that four user processes run on different "client" machines. And, the "server" machine provides allocation/deallocation services for three printers by executing the procedures "GetPrinter" and "ReleasePrinter", called by RPCs.

Create four concurrently running user (client) processes, each running on a different machine. A user process is in a loop "Compute-RequestAPrinter-ReleaseThePrinter": occasionally, requesting the allocation of a printer (e.g., the GetPrinter RPC), waiting and getting a printer allocated, and later releasing the printer (e.g., the ReleasePrinter RPC). The user processes have no priorities, and are serviced on a first-come-first-serve basis.

Test your program with a long run (making sure that you have cases in which all four user processes request a printer at about the same time), script and turn in the output as well as your source code. Make sure that you start the server (that runs procedures GetPrinter and ReleasePrinter) before starting the clients (i.e., the user processes). The user processes should print their actions with the machine name and the time attached.

There are three programs you have to write.

1) Definition of the RPC protocol in the *application.x* file (say, PrDmn.x).
2) The service program in *application.s.c* file (say, server.c). This program contains the actual procedure(s) to be invoked by the server dispatch routine in the compiler-generated stub. Please note that you can define and use several remote procedures to be called with RPCs using a single pair of client/server stubs.
3) Four different versions of the client program *application.c.c* file (say, client1.c, client2.c, client3.c, and client4.c). The client code should create a server handle, the attachment to the appropriate remote services of servers. In SOLARIS, this attachment is performed by the clnt_create system call.

Application development with RPCGEN: The protocol definition file *PrDmn.x* is first processed by RPCGEN (by issuing the command "RPCGEN PrDmn.x"), the RPC stub generation compiler. RPCGEN generates:

a) XDR wrapper routines in *PrDmn_xdr.c* (which are bidirectional filters used by both the client and the server),
b) the associated common "#include"s in *PrDmn.h*,
c) client stub in *PrDmn_clnt.c*,
d) server stub in *PrDmn_svc.c*. Please note that the server stub has a main. Thus, the remote procedure that you define in server.c should not have a main since it will be linked with the server stub.

The Protocol Definition Language: This is the language used for automated stub generation. The code will be stored in the *PrDmn.x* protocol definition file. Here is an example of a program definition:

```
program APPLICATION_PROGRAM {  /* Can specify multiple servers */
```

```
    version APPLICATION_VERSION        {
            struct output-parameter1 REMOTE-PROCEDURE1 (input-parameter1) = 1;
     } = 1;       /* RPC program version number (used for bookkeeping) */
} =0x20fff100  /* program number (used for bookkeeping) */
```

Please note that the client RPC call *must* have as the last parameter the server handle (created by the clnt_create system call) as a call-by-reference parameter. This parameter is only used by client and server stubs, not by the application. Also, all remote procedures should have the suffix _1 (as the version number) in their names.

At the Client: A client is required to maintain a unique structure (pointed to by a server handle) for each client/server connection. And, the "#include"s that you need to use in, say, *client1.c* are <rpc/rpc.h> and "PrDmn.h".

The RPC system calls that you need to use are clnt_create (you should use the UDP protocol (as opposed to TCP) in the specification) and clnt_pcreateerror and clnt_destroy.

At the Server: The server procedures that you write will be in *server.c* . The "#include" that you need to use in *server.c* is "PrDmn.h".

To compile and run RPCGEN: You can compile the protocol definition with the following command:

```
% rpcgen PrDmn.x
```

Then, RPCGEN produces a client stub, PrDmn_clnt.c, and a server stub, PrDmn_svc.c. It also produces any necessary XDR filters and a header file to be included by client and server applications and stubs. In this example, one XDR filter, PrDmn_xdr.c, and the header file, PrDmn.h, are generated.

You can compile the client code for, say, client1 and link it with the client stub and the XDR filter generated by RPCGEN using the command

```
% cc -o client1 client1.c PrDmn_clnt.c PrDmn_xdr.c -lnsl
```

You can compile the server code and link it with the server stub and the XDR filter generated by RPCGEN using the command

```
% cc -o server PrDmn_svc.c PrDmn_xdr.c  server.c -lnsl
```

To start remote processes: You can login to five different machines, and start (first) the server, and (then) the four clients. Or, you can use the remote shell command to start the server and the clients. A shell script to launch the server on the remote host cerne is

```
rsh -n cerne server &
```

Note: The RPC coverage in the Unix textbook, recitations and the class, together with man pages of the UNIX OS, should be sufficient for this assignment. Also, the book "Power Programming with RPCs" by J. Bloomer, O'Reilly & Associates, 1991 is an excellent source on RPCs, and has extensive RPC examples.