Project Three

Assigned: Tuesday, November 20th, 2001
Due: Wednesday, December 5th, 2001, midnight.

In this project, you will build a simple file transfer program on top of RPC (Remote Procedure Call Interface). You could also build it on top of your RDP protocol, if you prefer.

1 The File Transfer Program

Develop a simple file transfer application on top of RPC or your RDP protocol. You are required to provide both the client and the server code for the program. The server should be able to save and retrieve application files sent/requested by the client. The client should be able to send/request files from the server. You can pattern your application after TFTP (trivial file transfer program — man pages for TFTP are available). However, the server must also implement access permissions, and authenticate the client. For additional credit, you could also implement directory structures.

You should plan on designing a simple application with well-documented user functionality. At the very least, you should implement a `get`, `put`, and `ls`. Directory support would require a `pwd`, `mkdir`, and `cd`. Remember, there are also authentication, security, and access permission components.

You can limit yourself to implementing a personal secure file transfer program. In other words, you will not have to worry about honoring access permissions for multiple users, or maintaining multiple passwords/keys. Clients (and there can be multiple clients) will share the same file access permissions as the server, so you can use the standard Unix file permissions to implement file access permissions. In other words, if the server has permission to read and/or write a file, so does the client. (The functionality your code will provide is secure access to files on remote machines where the file system is not mounted).

This means that you do not necessarily have to create a separate temporary directory in /tmp in order to emulate a file system managed by your server. You could work in any directory, and read or write permission should be provided if the server has permission to read or write that directory and/or file. As extra credit, you can allow moving around in the directory hierarchy.

The client DOES need to be authenticated, however, to ensure that other users, for example, do not get remote access to your files. The server's response must also be authenticated to ensure that it has been sent by the server. The files must be transferred in a secure manner — in other words, privacy must be ensured. In order to help you with this part of the project, I have provided you with code that implements the DES algorithm, as well as routines that use cipher block chaining to encrypt and decrypt the data. The code is in `/u/cs257/projects/proj3/code/des.c`.

If you are implementing your application on top of RDP, use of DES should be pretty straightforward. If you are implementing your application using RPC, you will need to modify the xdr code generated for you in order to perform the encoding. (Please note that rerunning rpcgen will regenerate the `_xdr.c` file, so make sure to save any modifications you have made!).

2 Remote Procedure Call

Remote Procedure Call (RPC) is a popular mechanism for structuring distributed applications since it is based on the semantics of a local procedure call. Network File System (NFS) is one such application, which is integrated into many operating systems.

The caller of a function is the client, the callee the server. From a client's point of view, a remote procedure call looks just like an ordinary procedure call. From the server's point of view, being called by a remote procedure should be the same as being called by a local caller.

RPC relies on lower level networking mechanisms such as UDP/IP or TCP/IP, and implements client-server applications on top of them. Each RPC call sends a request to the server. When this request arrives, the server calls a dispatch routine, performs whatever service is requested, sends back the reply, and the procedure call returns to the client. Both client and server have "stubs" to invoke the network RPC library. The client stub is an interface between the client and the RPC library, and effectively hides the network from the caller. Similarly, the server stub hides the network from the server procedures that are to be invoked by remote clients.

SunRPC uses XDR (eXternal Data Representation) as the format of data sent across the network (you have seen an introduction to this through Assignment 3). The conversion process is called *marshaling*.

3 The rpcgen IDL Compiler

rpcgen is an Interface Definition Language compiler that takes an interface definition as its input and spits out the stub code and the XDR routines. The XDR routines can convert data in a stream into their network format and back to the local representation. **rpcgen** produces the interface between the application and the underlying networking mechanisms. It hides the representation and communication almost completely from you.

In order to use **rpcgen**, you need to learn the IDL it understands. A tutorial is available under `/u/csc257/projects/proj3`, as is an example. The IDL is very similar to C in its syntax and data types, with a few important differences.

rpcgen takes a .x file containing the interface definition as input. In this .x file, each RPC procedure is uniquely defined by a 32-bit program number and procedure number. The program number specifies a group of related remote procedures, each of which has a different procedure number. Each program also has a version number. Sun Microsystems as the developers of ONC (Open Network Computing Model of RPC) reserves the right to assign these program numbers. For instance, NFS uses program number 100003 and version number 2. For this project, just pick a program number less than 100000 and version number 1.

The RPC protocol is independent of the underlying transport. It can run on top of UDP or TCP. A command-line argument tells **rpcgen** which protocol is to be used. At runtime, a system daemon (`portmap`) assigns program numbers to available TCP/UDP port numbers dynamically. We'll run it on top of UDP. One disadvantage of this approach is that the data size cannot be larger than 8 KBytes. However, you can make the assumption that your code does NOT have to handle larger argument sizes.

The following is the list of files created by **rpcgen** from the interface definition in `foo.x`:

- `foo_clnt.c`: the client stub.
- `foo_svc.c`: the server stub.
- `foo.h`: the file containing definitions needed for the use of the interface. It should be included by both the client and the server modules.
- `foo_xdr.c`: the file converting procedure arguments and results into their network format and vice-versa.

Clients are linked with `foo_clnt.o` and `foo_xdr.o`. Servers are linked with `foo_svc.o` and `foo_xdr.o`.

4 Grading

This project is, once again, a *group* project (both for those taking the course as CS257 and for those taking the course as CS457) - groups can consist of one or two people. You are, as before, on your own to form your groups. If you form a two-person group, let me know who your partner is via e-mail. Part of your external documentation should be a discussion of who did what part of the work.

You should turn in

- your design document,
- the source code for your ftp server and client,
- a brief description of your experiments and test cases.

Be sure to include appropriate Makefiles and README files. Your Makefile should build the executable `ftp-server`, and the executable `ftp-client`. Multiple clients should be allowed.

To *handin* your assignment, you should -

- Put all the files relating to the assignment in one directory. Don't put anything not related to the assignment in that directory.
- Change to the directory and execute "/u/cs257/bin/TURN_IN .",
  or alternatively, "/u/cs257/bin/TURN_IN path_name", where path_name is the full path name of the directory where your project files are. This will copy all the files in the stated directory into the cs257 account. Don't execute this command until you are really done, because you cannot update the files once they have been created.