AIX

Developing a Client/Server Application Without a Client

By Richard Redpath

This article discusses developing a server application without having to write a client application. It also discusses some tips for developing maintainable client/server applications.

Most developers assume that two programs must be written simultaneously to develop a client/server application. Using `telnet` as a surrogate client can remove the requirement for dual development. Providing `telnet` support allows the developer to delay writing a client application until the server application is sufficiently mature.

The four areas to be considered are design, integrity, serviceability, and security.

Design

Programming for client/server computing differs from traditional programming because the programs are distributed. If developers understand the client/server model, they can ensure a clear and concise programming design that can be both easily read and maintained.

There are two categories of client/server flow control: linear and non-linear. The client is linear—similar to a traditional application waiting on a resource, such as reading data from a file. A client reading a socket from a server uses linear flow control.

The non-linear server is an N-way branch system that provides services to clients. The design for an N-way system of service is simple. The five procedures contained in a complete server model are as follows:

1. **Options**: Parse or set the options that are provided by the server.
2. **Daemonize**: Perform system requirements for the server to fit the system model. The server options will direct this level.
3. **Servicesetup**: Perform actions for service setup.
4. **Servicerequest**: Monitor a service request from a client.
5. **Processrequest**: Process the request independently.

For an N-way branch system, the server responds to a request in a forked process or thread. A thread may have an advantage over a forked process. Since it is a lightweight process, it allows access to information across service boundaries.

Figure 1 shows the main body of code for the sample server discussed in this article.

```
typedef struct {
     struct in_addr sin_addr;
     int            clientfd;
   } CLIENTINFO;
void main(int    argc,
          char **argv)
{CLIENTINFO *client;
 options(argc,argv);
 daemonize();
 servicesetup();
 for (;;){
    client=servicerequest();
    processrequest(client);
   }
}
```

Figure 1. Main body of server code

Nov. '93
p.13

Richard Redpath

Integrity

The following discussion shows how to provide the functions of the five procedures with integrity.

The options procedure. In this example, the server supports the following options:

- **-level**: Prevents the server from becoming a session leader, which helps the programmer develop the server
- **-port**: Sets the port address
- **-trace**: Turns trace on

The daemonize procedure. The application orphans the server application to the `init` process to provide for zombie cleanup. The unneeded file descriptors are closed according to the level set by the programmer (the default is above `stderr`). In addition, the current directory is changed to the root directory to avoid an `unmount` condition. The `umask` value for the running process is set to zero to ensure that anyone can read a file. Finally, `setsid()` makes the process a session leader to remove the controlling terminal. (As a session leader, there is no controlling terminal; hence, the controlling terminal should be removed in that situation.)

The servicesetup procedure. The `setsockopt SO_REUSEADDR` ensures that local addresses and ports do not duplicate foreign addresses and ports. This can be invaluable during development because it avoids the appearance of a non-responding server trying to establish a socket bind.

The servicerequest procedure. The socket accepts a connection to the client request. Memory is allocated to track the client file descriptor and client address. The client address can be used by future functions provided by the server.

The processrequest procedure. A forked orphan process is created to service the client, and the initial parent client file descriptor is closed. The system cleans up the orphan process after servicing the client request. After establishing an independent process connection with the client, via a forked process, the server can now handle the request from the client.

Serviceability

A `telnet` connection makes it easy to service a server. Using `telnet`, the system administrator can connect to the running server and verify that it is running properly. The system administrator can turn on trace facilities or dump log information (with the appropriate command set added). The `serveclient` function not only serves client requests, but also recognizes the string `telnet` if it is provided as the first command upon connection. To connect to the sample server, the system administrator must know the address and port of the server. Figure 2 shows a sample excerpt of the `telnet` command.

Dialog if the Server is Running

```
$telnet localhost 5100
Trying...
Connected to 9.67.225.54
Escape character is '¬]'
telnet
Yes, I support telnet connect
```

Dialog if the Server is Not Running

```
$telnet localhost 5100
Trying...
telnet: connect: A remote host refused an attempted connectio
operation.
```

Figure 2. Example dialogs when using telnet to communicate with a server

Security

System administrators should require a password for `telnet` access to the server if connecting and issuing commands to the server can lead to unsecure operations.

Sample Server Code

Complete source code for a sample server (file `9402P55`) is available on IBM's Electronic Vendor Information Service (EVIS). See page 47 for instructions on downloading this file. The server accepts the following commands by `telnet`: `cookie`, `fortune`, `telnet`. Each command produces a response to the user.

In general, `telnet` can be used to develop a server without requiring developers to code both the client and server. The `telnet` support adds to the serviceability of a server. With the appropriate command set implemented in the server, `telnet` facilitates the ability to turn on trace facilities or dump log information.

Richard Redpath, *IBM Corporation, 11000 Regency Park, Cary, NC 27513. Mr. Redpath is an advisory software engineer in the Software Library Development group. He is responsible for the user interface design and cross-platform architecture for the Configuration Management Version Control (CMVC) library. Mr. Redpath has a BS in Applied Mathematics in Computer Science from Carnegie-Mellon University.*