Progress Based Socket Server
Via XINETD

Scott Auge
sauge@amduus.com scott_auge@yahoo.com

ABSTRACT

There are times when one wants to reach a TCP port on a machine running a Progress client and interact with that client through a protocol on that port. This document describes how to do so on a Linux computer.

THEORY OF OPERATION

This method of setting up a service to interact with via a TCP port is done with the xinetd super-server.

One sets up `xinetd` to listen for connections on a given port number. When that happens, the `xinetd` super server will start off a sub-server dedicated to interaction through `stdin` and `stdout`.

We capture `stdin` and write out to `stdout` in a specialized set of routines that wrap the functionality the programmer wishes for the Progress based server.

SETTING UP XINETD

One first needs to create an entry in the `/etc/services` file to name the service you wish to call your functionality.

```
tstclib   20015/tcp # Progress based protocol server
```

Next we need to set up an entry in the `/etc/xinetd.d` directory to tell `xinetd` what to do when this port is tickled by a client. This should have the same name as your service so in this case it would be called `tstclb`.

```
# Test service io to progress
service tstclib
{
        socket_type     =  stream
        protocol        =  tcp
        wait            =  no
        user            =  root
```

```
        server          =  /bin/bash
        server_args     =  /appl/srvr/script/answer.bash
        #only_from       =  127.0.0.1
        log_on_failure  += USERID
        disable         =  no
}
```

You can see, that when the port is tickled, `xinetd` is going to start up a `bash` shell and then run the script `/appl/srvr/script/answer.bash`. This script is what starts the progress client and allows you to set up the PROPATH, TERM[1], DLC, etc. You will need to use absolute paths.

Next, as root, restart `xinetd`:

```
/etc/init.d/xinetd restart
```

As noted, you will need a shell script to start up the program – here is an example in `/scripts/answer.bash`:

```
export DLC=/prg/101a
export PROPATH=/appl/srvr/src

$DLC/bin/_progres -b -p answer.p
```

At this point, xinetd[2] will be set up to answer a call by a client on a port and allow Progress to interact with the client.

DISCUSSION OF PROGRAMMING YOUR SERVER

The program answer.p provides a simple example of a server that can perform tasks asked of it over the socket connection. These tasks include:

- Displaying the current time
- Displaying the environment the server is running under
- Executing a command
- Disconnecting

The purpose is to provide examples the programmer can use for executing 4GL code, showing interaction with the shell is possible remotely, and how to send commands that might include arguments to the server.

Using answer.p to show time

Here is a simple sequence for showing the time from a telnet into the server. The server reactions are colored blue while the client protocol commands are in red:

1 This really expects to interact with a program using a telnet client library. You will needto inform the shell of any parameters it should work under if not.

2 xinetd is available on many linux systems, AIX, and Solaris (there may be others.)

```
osxlaptop:~ scottauge$ telnet www 20015
Trying 192.168.1.36...
Connected to www.gateway.2wire.net.
Escape character is '^]'.
100 Ready
time
101 Action Received: time
12:03:53
100 Ready
stop
101 Action Received: stop
110 Bye
Connection closed by foreign host.
```

When the system is ready to receive a command, it prompts with a `100 Ready` message.

We then send in the command `time` and then the system replies with a reflection of what it is going to do, then shows the time and then the ready message for the next command.

Using the command processor in answer.p

This is a dangerous command to have as part of your server's functionality. I show it only to show it is possible should you want a tool that starts and stops processes.

Here we are going to run the ls command. The first time will generate an error – this shows what happens when an error occurs. Next we will have it actually work. The server is in blue and the client commands are in red:

```
osxlaptop:~ scottauge$ telnet www 20015
Trying 192.168.1.36...
Connected to www.gateway.2wire.net.
Escape character is '^]'.
100 Ready
cmd ls -l /tmp/tmp*
101 Action Received: cmd ls -l /tmp/tmp*
ls: /tmp/tmp*: No such file or directory
100 Ready
cmd ls -l /tmp/tpl*
101 Action Received: cmd ls -l /tmp/tpl*
-rw-r--r-- 1 wwwrun www   613 Sep 15 19:36 /tmp/tpl-default-
cc_index-b7186f29ed2121959c9b7e306113a80b.php
-rw-r--r-- 1 wwwrun www  4058 Sep 15 19:36 /tmp/tpl-default-
cc_menu-b7186f29ed2121959c9b7e306113a80b.php
100 Ready
stop
101 Action Received: stop
110 Bye
```

```
Connection closed by foreign host.
```

Discussion of the answer.p program

How is all this done on the Progress side? Here is a listing of the answer.p program – nothing super complicated about it!

```
DEFINE VARIABLE cClientCmd AS CHARACTER NO-UNDO.

{clib.i}

REPEAT:

  RUN PutClient ("100 Ready").
  ASSIGN cClientCmd = "".
  RUN GetClient (OUTPUT cClientCmd).

  RUN PutClient ("101 Action Received: " + cClientCmd).

  /*****************************************************/
  /* We use begins in case command has arguments       */
  /*****************************************************/

  IF cClientCmd BEGINS "STOP" THEN LEAVE.

  IF cClientCmd BEGINS "TIME" THEN RUN HandleTime.

  IF cClientCmd BEGINS "env" THEN RUN HandleEnv.

  IF cClientCmd BEGINS "CMD" THEN RUN HandleCmd (cClientCmd).

END. /* REPEAT */

RUN PutClient ("110 Bye").

QUIT.

/********************** Command Handlers *********************/
PROCEDURE HandleTime:

  RUN PutClient ( STRING (TIME, "HH:MM:SS")).

END.

PROCEDURE HandleEnv:

  DEFINE VARIABLE cLine AS CHARACTER NO-UNDO.

  INPUT THROUGH set.

  REPEAT:

    IMPORT UNFORMATTED cLine.
    RUN PutClient (cLine).

  END.
```

```
  INPUT CLOSE.

END. /* PROCEDURE HandleEnv */

PROCEDURE HandleCmd:

  DEFINE INPUT PARAMETER cClientCmd AS CHARACTER NO-UNDO.

  DEFINE VARIABLE cCmd AS CHARACTER NO-UNDO.
  DEFINE VARIABLE cLine AS CHARACTER NO-UNDO.

  ASSIGN cCmd = SUBSTRING(cClientCmd, 4, LENGTH(cClientCmd)).

  INPUT THROUGH VALUE(cCmd).

  REPEAT:

    IMPORT UNFORMATTED cLine.
    RUN PutClient (cLine).

  END.

  INPUT CLOSE.

END. /* PROCEDURE HandleCmd */
```

It should be pretty obvious on how you can use it to create records and look up records in a database, issue commands to the operating system, and even create peer to peer communications.

SPECIAL CONSIDERATIONS

There are a little bit of C coding involved in this functionality. You can find it in the `/lib` directory including the `Makefile` to generate it.

If you need to recompile, simply enter:

```
make clean
make all
```

and the new .so should be generated for you.

You will need to reference this .so in the `src/clib.i` file.

The coding should be portable across all UNIX/LINUX OS's with the gcc compiler.