SOCKET Programming with REXX/VSE

BY LEO J. LANGEVIN

A couple of months ago, I wrote about the simplicity of socket programming. Since then, several readers have written to me, asking about the REXX example that I provided. To them, this made a lot of sense for people who don't work with UNIX. Certainly, you can use either the masochistic BSD method or the simplistic VSE method. Obviously, there's a lot of interest in this area, since we are starting to see several vendors who are writing TCP/IP applications. Several years ago, I wrote in *DOS/VSE/SP Guide for Systems Programmers* that if you see a lot of people doing the same thing, then it can't be that hard to do.

Well, some folks don't want to spend the time writing assembler language applications and others don't want to bother with COBOL. Granted, there's still PL/I and C, but what about a simple language? What about REXX/VSE? Can you think of a simpler or faster way to crank out code? With REXX/VSE you can create a TCP/IP application in a short period of time. By using the simple functions of TCP/IP for VSE's REXX interface you can write your own clients or servers in REXX!

Imagine tossing together an FTP client in a few minutes that allows you to check out the responses and take some sort of action (e.g., test if DIRLIST indicates that a file was not found). Or, what about using REXX to extract lines of a report and invoke LPR to print them at a remote site? Even better, a full-blown server that uses TCP or UDP to return responses to simple PC programs!

Imagine tossing together an FTP client in a few minutes that allows you to check out the responses and take some sort of action (e.g., test if DIRLIST indicates that a file was not found).

SO HOW DOES THIS WORK?

REXX/VSE consists of a half-dozen necessary function calls:

- OPEN
- CLOSE
- SEND
- RECEIVE
- ABORT
- STATUS

I can't imagine a simpler approach. If you think of SEND, RECEIVE and ABORT as WRITE, READ and CANCEL, then you can manipulate datagrams (a unit of transmission) the same way that you manipulate files. The purpose of the STATUS command is for a REXX/VSE server to check if a connection has been made.

Before we look at how to code our REXX/VSE socket application, let's look at some of the constants and return codes, and how the functions are actually coded.

SPECIAL CONSTANTS

When you code a REXX/VSE socket program, up to five different variable names will be set when you issue a SOCKET() function:

- **handle** - an output field that contains a special pointer that is generated by an OPEN. This value will be used in subsequent calls.
- **buffer** - an output field that contains the data that was read from a RECEIVE request.
- **errmsg** - if an error occurs, *Rexx Sockets* will put the message in this variable instead of outputting it to SYSLOG or SYSLST. The exception to this is when the setting of *errmsg* fails; then the message will be routed to SYSLOG.
- **foip** - for a SERVER, after a connection occurs, the 15-byte IP-address in the format of 'nnn.nnn.nnn.nnn' will be contained in this field.
- **foport** - for a SERVER or a CLIENT, after a connection occurs, the 1- to 5-byte port number will be contained in this field.

RETURN CODES

The return code from these calls is consistent for all but the STATUS request:

0 =	Request was successful
4 =	Timeout occurred
8 =	Error
12 =	Foreign IP address is not available
16 =	Local TCP/IP system is down
20 =	Unregistered version

CODING THE REXX FUNCTIONS

First, Figure 1 presents the format of each call. As you can see from this example, the OPEN function has the largest number of parameters; however, only the first three are required for a server and the first five are required for a client. The "handle" is returned from the OPEN call in the *handle* variable and must be used as input to every other call. The handle is, in actuality, a pointer to a block of storage. If you want to open multiple ports you will need to allocate multiple handles and save them in different names for later use. Following are the required fields:

rc - return code: It's important to check this. A return code of zero is good. However, the STATUS call will return a binary value based on the connection. If no connection has been requested, then it is zero. If a connection has occurred, then the value is one. Basically, it's the ECB posted byte that's returned.

If you think of SEND, RECEIVE and ABORT as WRITE, READ and CANCEL, then you can manipulate datagrams (a unit of transmission) the same way that you manipulate files.

type - the connection type: This may be one of the following: TCP, UDP, TELNET, CLIENT, FTP, RAW or CONTROL. What's nice about CLIENT is that you can simulate any client function that is provided with TCP/IP for VSE.

Now for the optional fields:

loport - this is the port number that the program will be using for itself.

foip - if set, it will indicate the IP address to connect to. If not set, REXX sockets will assume that this is a server and not a client.

foport - if set, this will indicate what port number to connect to at the remote side. If it is not set, then the LOPORT value will be used.

sysid - if set, this will indicate which TCP/IP for VSE partition to use. If this is not set, then "00" will be assumed.

timeout - if set, this will indicate the number of 1/300th of a second that will be used to indicate the maximum wait value before timing out. If not set, a default of 36000 (two minutes) will be used for clients.

asynch - if set to "N", which is the default, the OPEN will be performed

Figure 1: Format of Calls

```
rc=SOCKET(type,'OPEN',loport,foip,foport,sysid,timeout,asynch)
rc=SOCKET(handle,'CLOSE',timeout)
rc=SOCKET(handle,'ABORT',timeout)
rc=SOCKET(handle,'RECEIVE',timeout)
rc=SOCKET(handle,'SEND',data,timeout)
rc=SOCKET(handle,'STATUS')
```

Figure 2: REXX FTP Client

```
* $$ JOB DISP=D,CLASS=4,JNM=REXXFTP
* $$ LST DISP=D,CLASS=Q,JSEP=0,RBS=0
// JOB REXXFTP
* Cataloging the member
// EXEC LIBR
A S=PRD2.TCPIP
CATALOG REXXFTP.PROC REP=Y EOD=/(
FIP = arg(1)                         /* Get the Foreign IP address        */
/*
                              REXX FTP Client

               Copyright (c) 1998, Connectivity Systems, Inc.

                          Author: Leo J. Langevin
*/

/* ---------------------------- Main Program ---------------------------- */
call init                            /* Initialize the variables          */
call open                            /* Open the SOCKET port              */
call process                         /* Receive lines and send commands   */
call close                           /* Close the SOCKET port             */
exit 0                               /* Terminate the program             */
/* ---------------------------------------------------------------------- */

/* ---------------------------- INIT Routine ---------------------------- */
init:                                /* Start of routine                  */
x=ASSGN('STDOUT','SYSLST')           /* Display output on SYSLST          */
x=ASSGN('STDIN','SYSIPT')            /* Read data from SYSIPT             */
if FIP = '' then do                  /* If the user forget an IPaddr.     */
   say 'IP address is missing'       /*    then inform the user           */
   exit 20                           /*    terminate                      */
   end                               /* Otherwise we'll proceed           */
eof=1                                /* Init the indicator                */
max=900                              /* 3 second maximum timeout          */
sysid=C'00'                          /* default SYSID                     */
return                               /* Return to caller                  */
/* ---------------------------------------------------------------------- */

/* ---------------------------- OPEN Routine ---------------------------- */
open:                                /* Start of routine                  */
x  = SOCKET('FTP','OPEN',21,fip,21,sysid)                                 */
if x \= 0 then do                    /* If the OPEN failed...             */
   say errmsg                        /*    Display the error message      */
   call eatum                        /*    Consume all of SYSIPT          */
   exit 8                            /*    And exit this program          */
end                                  /* Otherwise...                      */
return                               /* Return to caller                  */
/* ---------------------------------------------------------------------- */

/* --------------------------- PROCESS Routine -------------------------- */
process:                             /* Start of routine                  */
do until eof=0                       /* Continue until no more SYSIPT     */
   call getlines                     /* Read display lines from server    */
   call getcommand                   /* Get data from SYSIPT              */
   if eof \= 0 then call send        /* If there's a command, send it     */
   end                               /* Repeat until EOF                  */
return                               /* Return to caller                  */
/* ---------------------------------------------------------------------- */

/* ---------------------------- CLOSE Routine --------------------------- */
close:                               /* Start of routine                  */
x = SOCKET(handle,'CLOSE')           /* Close the SOCKET                  */
if x \= 0 then do                    /* If it failed...                   */
   say errmsg                        /*    Display the problem            */
   call abort                        /*    Abort the connection           */
   x = SOCKET(handle,'CLOSE')        /*    Try a second close             */
   exit 8                            /*    And exit the program           */
end                                  /* If it didn't fail...              */
return                               /*    Return to caller               */
/* ---------------------------------------------------------------------- */

/* ---------------------------- SEND Routine ---------------------------- */
send:                                /* Start of routine                  */
x=SOCKET(handle,'SEND',command)      /* Send the command to the server    */
  if x \= 0 then do                  /* If it failed...                   */
     say errmsg                      /*    Tell us why                    */
     call abort                      /*    Abort the connection           */
```

Continued on next page.

Continued from last page.

```
     call close                     /*     Close the connection           */
     call eatum                     /*     Get rid of SYSIPT data         */
     exit 8                         /*     And terminate the program      */
     end                            /* If it worked...                    */
return                              /*    Return to caller                */
/* -------------------------------------------------------------------- */

/* ------------------------ GETCOMMAND Routine ------------------------ */
getcommand:                         /* Start of routine                   */
parse upper pull command            /* Get the FTP command from SYSIPT    */
eof=length(command)                 /* As well as the length              */
if eof > 50 then                    /* If the length is really big...     */
   command=substr(command,1,50)     /*     Truncate it for SYSOUT         */
say command                         /* Display the command                */
return                              /* And return to caller               */
/* -------------------------------------------------------------------- */

/* -------------------------- ABORT Routine --------------------------- */
abort:                              /* Start of routine                   */
x=SOCKET(handle,'ABORT')            /* Abort the connection               */
if x \= 0 then say errmsg           /* If it failed, tell the user        */
return                              /* Return to caller                   */
/* -------------------------------------------------------------------- */

/* -------------------------- EATUM Routine --------------------------- */
eatum:                              /* Start of routine                   */
do until eof = 0                    /* Do this until we read no more      */
  pull command                      /* Read a command from SYSIPT         */
  end                               /* And repeat                         */
/* -------------------------------------------------------------------- */

/* ------------------------- RECEIVE Routine -------------------------- */
receive:                            /* Start of routine                   */
x = SOCKET(handle,'RECEIVE',max)    /* Read data from the server          */
if x \= 0 then do                   /* If it failed...                    */
   say errmsg                       /*    Tell us what went wrong         */
   call abort                       /*    Abort the connection            */
   call close                       /*    Close the connection            */
   call eatum                       /*    Discard the rest of SYSIPT      */
   exit 8                           /*    And terminate the program       */
end                                 /* Otherwise...                       */
return                              /* Return to caller                   */
/* -------------------------------------------------------------------- */

/* ------------------------ GETLINES Routine -------------------------- */
getlines:                           /* Start of routine                   */
do forever                          /* Do until we find a match           */
  call receive                      /* Read another datagram              */
  say buffer                        /* Display it on SYSOUT               */
  loc=POS('Error:',buffer)          /* See if a connection failed         */
  if loc > 0 then exit 20           /* If the connection failed, exit     */
  loc=POS('Ready:',buffer)          /* Look for another keyword           */
  if loc > 0 then return            /* And if we found it, it's okay      */
  loc=POS('Enter Foreign User ID',buffer)
  if loc > 0 then return
  loc=POS('Enter Foreign Password',buffer)
  if loc > 0 then return
  loc=POS('Enter Foreign Account',buffer)
  if loc > 0 then return
  loc=POS('Enter Local User ID',buffer)
  if loc > 0 then return
  loc=POS('Enter Local Password',buffer)
  if loc > 0 then return
  loc=POS('Enter Local Account',buffer)
  if loc > 0 then return
  loc=POS('Service closing control connection',buffer)
  if loc > 0 then return
end
/* -------------------------------------------------------------------*/
/(
/*
// IF $RC EQ 0 THEN
// GOTO OK
* Catalog failed ! Execution bypassed
// GOTO EXIT
/. OK
* Executing the member
// LIBDEF *,SEARCH=(PRD2.TCPIP,PRD2.TCPIPCFG)
// EXEC REXX=REXXFTP,PARM='100.100.1.1'
LEO
LEO
LEO
LEO
DIR
QUIT
/*
/. EXIT
/&
* $$ EOJ
```

synchronously - this means that the OPEN will not complete until an OPEN from the other side completes. If set to "Y", then *asynchronous* processing will take place. This means that the OPEN will return control to the REXX program immediately without waiting for an OPEN from the other side to complete. This is especially useful when you are writing a SERVER program and want immediate control, even if there is no client attempting to connect to the program.

So now that we have all of this information, let's take a look at a concrete example. The source code for REXX FTP is provided in Figure 2. This is an FTP client that you can run in a batch partition. It will read FTP commands from SYSIPT, pass them to an FTP server, and display the response on SYSLST. Obviously, you can add whatever logic you want to perform specialized actions. Again, your only limit is your imagination. ts

NaSPA member Leo J. Langevin works for Connectivity Systems and is the lead developer for NFS for VSE as well as REXX Sockets for VSE. He has been involved with VSE since its inception. He can be reached at LEO@TCPIP4VSE.COM.