

## 9. SOCKETS

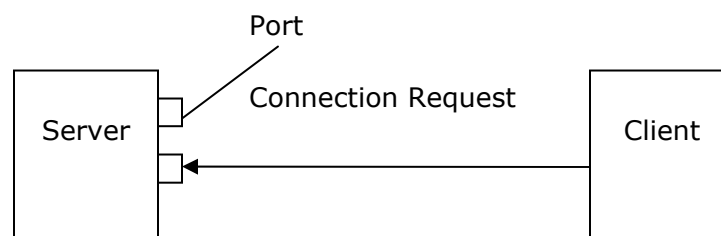
This chapter will begin with a basic description of sockets and proceed to describe their implementation in VSE.

### 9.1 Introduction to Sockets

Like most inter-process communication, sockets are based on the client server model. A socket is one end of a two-way connection between two programs running on a network. The program generally runs between two different terminals. It is run on the same terminal only for testing purposes. The two processes each establish their own socket.

*On the Server side:* A server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. A connection is established when a client connects successfully to a server and makes a request for information. The server can accept or refuse the connection, depending upon the number of clients that are allowed to communicate with the server at a time.

*On the Client side:* The client needs to know the hostname of the machine on which the server is running. To make a connection request, the client sends a request to a particular port on the server. If the connection is accepted, a socket is successfully created by the client and it can now communicate with the server by writing to or reading from the socket.



The client needs to know of the existence of and the address of the server, but the server does not need to know the address of - or even the existence of - the client prior to the connection being established.

## 9.1 Implementing sockets in VSE

Sockets are implemented in VSE by using the NETWORK library.

This library contains a module called CLNT\_SRV. The CLNT\_SRV contains several processes, the most commonly used of which are:

```
SERVER_OPEN
SERVER_ADD_A_CLIENT
CLIENT_OPEN
RECEIVE_RECORD
SEND_RECORD
```

The functionality of the rules is self explanatory.

The processes in the CLNT\_SRV module share a single resource definition which is given below:

```
TCPIP_INTERFACE
  1  HOST_ID                CHAR 80
  1  PAD                    REDEFINES HOST_ID
    2  TERM_HOST_ID        CHAR 40
    2  PAD                  CHAR 40
  1  RECORD_SIZE           INTEGER
  1  SOCKET                 INTEGER
  1  SERVER_PORT            INTEGER
  1  DESTINATION_ADDRESS   INTEGER
  1  TIME_OUT_IN_SECONDS   INTEGER
  1  RETURN_STATUS         STATUS      EROR   OK
  1  RESPONSE              CHAR 80
  1  PAD                    REDEFINES RESPONSE
    2  TERM_RESPONSE       CHAR 40
    2  PAD                  CHAR 40
```

Any VSE/GSS program that uses the NETWORK Library must use this interface because it is what the library expects.

The steps involved in establishing a socket on the client side are as follows:

- Create a socket. This is established by making a call to CLIENT\_OPEN.

```
CALL CLIENT_OPEN IN CLNT_SRV NETWORK USING TCP_IP_INTERFACE
```

Where TCP\_IP\_INTERFACE is a resource defined within your module. This call will read the values in the HOST\_ID and SERVER\_PORT variables and send a connection request to specified port on the server. If the connection succeeds, the client can proceed with sending and receiving data from the server. If the connection fails, a failure message is written to TERM\_RESPONSE.

- Send and receive data.

Data is sent by calling `SEND_RECORD`.

```
CALL SEND_RECORD IN CLNT_SRV NETWORK USING TCP_IP_BUFFER,  
SOCKET_INTERFACE
```

Data is received by calling `RECEIVE_RECORD`

```
CALL RECEIVE_RECORD IN CLNT_SRV NETWORK USING TCP_IP_BUFFER,  
SOCKET_INTERFACE
```

The steps involved in establishing a socket on the server side are as follows:

- Create a socket.

```
CALL SERVER_OPEN IN CLNT_SRV NETWORK USING SOCKET_INTERFACE
```

- Listen for connections from clients.

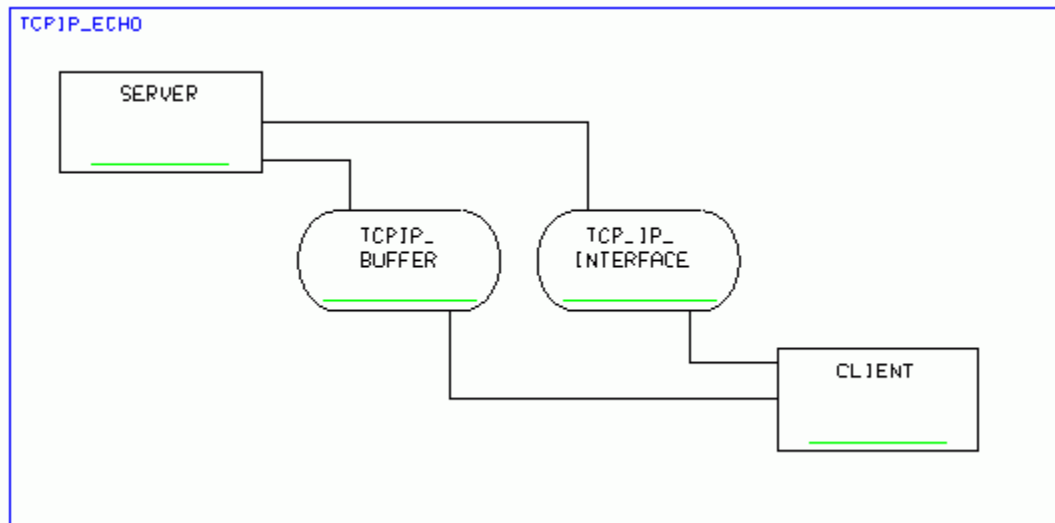
```
CALL SERVER_ADD_A_CLIENT IN CLNT_SRV NETWORK USING SOCKET_INTERFACE
```

- Once a client establishes a connection, data can be sent and received using the same commands used by the client.

## 9.2 Example Program – TCP/IP Echo

This simple program creates a client server connection. The client asks the user for data which is sent to the server, which echoes it back again to the client. Both the client and the server display the data on the console.

The program requires one module that contains two resources and two processes. The module, called TCPIP\_ECHO is illustrated below:



The code for the various resources and processes is outlined below:

### The TCP\_IP\_INTERFACE resource:

```
TCPIP_INTERFACE
  1  HOST_ID                CHAR 80
  1  PAD                    REDEFINES HOST_ID
    2  TERM_HOST_ID        CHAR 40
    2  PAD                  CHAR 40
  1  RECORD_SIZE           INTEGER
  1  SOCKET                 INTEGER
  1  SERVER_PORT           INTEGER
  1  DESTINATION_ADDRESS   INTEGER
  1  TIME_OUT_IN_SECONDS  INTEGER
  1  RETURN_STATUS         STATUS    EROR    OK
  1  RESPONSE              CHAR 80
  1  PAD                    REDEFINES RESPONSE
    2  TERM_RESPONSE       CHAR 40
    2  PAD                  CHAR 40
```

## The TCPIP\_BUFFER resource:

The TCPIP\_BUFFER resource is used to hold the data that is sent and received from the socket. It cannot be incorporated into the SOCKET\_INTERFACE resource because both resources are used when sends and receives are invoked.

```
MESSAGE CHAR 24
    ALIAS END_INTERACTION VALUE 'END', 'E', 'QUIT', 'Q',
                                'End', 'Quit',
                                'end', 'e', 'quit', 'q'
```

## The SERVER process:

```
SERVER
    MOVE 5555 TO SERVER_PORT
    DISPLAY ' Echo Server ', SERVER_PORT
    DISPLAY ' '
    EXECUTE TCPIP_RECEIVE_CONNECTION

    MOVE 24 TO RECORD_SIZE

    EXECUTE RECEIVE
    EXECUTE DISP_REPLY_AND_RECEIVE
        UNTIL MESSAGE IS AN END_INTERACTION

    DISPLAY ' '

DISP_REPLY_AND_RECEIVE
    DISPLAY ' Received Message ', MESSAGE
    EXECUTE REPLY
    EXECUTE RECEIVE

RECEIVE
    MOVE SPACES TO TCPIP_BUFFER
    EXECUTE TCPIP_READ

REPLY
    EXECUTE TCPIP_WRITE

*** ----- ***
TCPIP_RECEIVE_CONNECTION
*** MOVE TO SERVER_PORT
CALL SERVER_OPEN IN CLNT_SRV NETWORK
    USING SOCKET_INTERFACE
IF RETURN_STATUS IS EROR
    DISPLAY ' SERVER_OPEN ERROR ', TERM_RESPONSE
    DISPLAY ' SERVER_PORT ', SERVER_PORT
    EXIT THIS RULE .
*** MOVE SOCKET TO LISTEN_SOCKET

*** MOVE LISTEN_SOCKET TO SOCKET
CALL SERVER_ADD_A_CLIENT IN CLNT_SRV NETWORK
    USING SOCKET_INTERFACE
IF RETURN_STATUS IS A EROR
```

```

        DISPLAY ' SERVER_ADD_A_CLIENT ', TERM_RESPONSE
        DISPLAY ' SOCKET ', SOCKET
        EXIT THIS RULE .

    *** MOVE SOCKET TO CONNECTION

TCPIP_READ
    *** MOVE TO SOCKET
    *** MOVE TO RECORD_SIZE
    CALL RECEIVE_RECORD IN CLNT_SRV NETWORK
        USING TCPIP_BUFFER, SOCKET_INTERFACE
    IF RETURN_STATUS IS A ERROR
        DISPLAY ' RECEIVE_RECORD ERROR ', TERM_RESPONSE
        DISPLAY ' SOCKET ', SOCKET
        DISPLAY ' RECORD_SIZE ', RECORD_SIZE
        EXIT THIS RULE .
    *** DISPLAY ' RECEIVED ', TCPIP_BUFFER

TCPIP_WRITE
    *** MOVE TO SOCKET
    *** MOVE TO RECORD_SIZE
    *** MOVE TO TCPIP_BUFFER
    CALL SEND_RECORD IN CLNT_SRV NETWORK
        USING TCPIP_BUFFER, SOCKET_INTERFACE
    IF RETURN_STATUS IS A ERROR
        DISPLAY ' SEND_RECORD ERROR ', TERM_RESPONSE
        DISPLAY ' SOCKET ', SOCKET
        DISPLAY ' RECORD_SIZE ', RECORD_SIZE
        EXIT THIS RULE .

```

The server initializes the SERVER\_PORT to 5555. But, any port which is not a reserved port can be used by the program.

Error messages containing the TERM\_RESPONSE, HOST\_ID and SERVER\_PORT are printed whenever a connection or a read/write operation fails.

### The CLIENT process:

```

CLIENT
    MOVE 5555 TO SERVER_PORT
    MOVE 'local host' TO HOST_ID
    DISPLAY ' Echo Client ', HOST_ID, SERVER_PORT
    DISPLAY ' '
    EXECUTE TCPIP_CONNECT

    MOVE 24 TO RECORD_SIZE

    EXECUTE SEND
    EXECUTE RECEIVE_DISP_AND_SEND
        UNTIL MESSAGE IS AN END_INTERACTION

    DISPLAY ' '

```

```
RECEIVE_DISP_AND_SEND
  EXECUTE RECEIVE
  DISPLAY ' Received message ', MESSAGE
  EXECUTE SEND
```

```
RECEIVE
  MOVE SPACES TO TCPIP_BUFFER
  EXECUTE TCPIP_READ
```

```
SEND
  DISPLAY ' Please enter a message. '
  ACCEPT MESSAGE
  EXECUTE TCPIP_WRITE
```

```
*** ----- ***
```

```
TCPIP_CONNECT
  *** MOVE TO HOST_ID
  *** MOVE TO SERVER_PORT
  CALL CLIENT_OPEN IN CLNT_SRV NETWORK
  USING SOCKET_INTERFACE
  IF RETURN_STATUS IS AN EROR
    DISPLAY ' CLIENT_OPEN ERROR ', TERM_RESPONSE
    DISPLAY ' HOST_ID ', HOST_ID
    DISPLAY ' SERVER_PORT ', SERVER_PORT
  EXIT THIS RULE .
```

```
*** MOVE SOCKET TO CONNECTION
```

```
TCPIP_READ
  *** MOVE TO SOCKET
  *** MOVE TO RECORD_SIZE
  CALL RECEIVE_RECORD IN CLNT_SRV NETWORK
  USING TCPIP_BUFFER, SOCKET_INTERFACE
  IF RETURN_STATUS IS A EROR
    DISPLAY ' RECEIVE_RECORD ERROR ', TERM_RESPONSE
    DISPLAY ' SOCKET ', SOCKET
    DISPLAY ' RECORD_SIZE ', RECORD_SIZE
  EXIT THIS RULE .
  *** DISPLAY ' RECEIVED ', TCPIP_BUFFER
```

```
TCPIP_WRITE
  *** MOVE TO SOCKET
  *** MOVE TO RECORD_SIZE
  *** MOVE TO TCPIP_BUFFER
  CALL SEND_RECORD IN CLNT_SRV NETWORK
  USING TCPIP_BUFFER, SOCKET_INTERFACE
  IF RETURN_STATUS IS A EROR
    DISPLAY ' SEND_RECORD ERROR ', TERM_RESPONSE
    DISPLAY ' SOCKET ', SOCKET
    DISPLAY ' RECORD_SIZE ', RECORD_SIZE
  EXIT THIS RULE .
```

The client initializes the HOST\_ID to the 'local host' in this program. But, it can be set to the IP address of the machine that it is trying to connect to.

Finally, after this module is prepared, it should be used in a task. This program will have two tasks, called ECHO\_S and ECHO\_C.

The control specifications of each task will indicate the process in the module that is to be executed.

### **ECHO\_S:**

```
CONTROL SECTION
  TITLE, Echo Server

MODULE SECTION
  SERVER

END
```

### **ECHO\_C:**

```
CONTROL SECTION
  TITLE, Echo Client

MODULE SECTION
  CLIENT

END
```

The server should be run first. After which the client is run. The client will prompt the user for messages and the entries are sent to the server and echoed back.

```
H:\VSE\ECHO_C.EXE
Echo Client 131.183.20.211
5555

Please enter a message.
> This is a test.

Received message This is a test.
Please enter a message.
> Sockets in USE.

Received message Sockets in USE.
Please enter a message.
> Simple echo server.

Received message Simple echo server.
Please enter a message.
> quit
```

Any of the commands entered in TCP\_IP\_BUFFER should be used to quit the program – quit, end, e, q etc.

```
H:\VSE\ECHO_S.EXE
Echo Server 5555

Received Message This is a test.
Received Message Sockets in USE.
Received Message Simple echo server.
```

\*\*\*