SETTING UP ORACLE IN A CLIENT/SERVER TCP/IP ENVIRONMENT

Irma Fisher and Mary Ann Hale *
Internal Revenue Service

Abstract

This paper will explore configuring ethernet on a Unix system. We will discuss general ethernet setup, including affected system files and examples of their contents. We will introduce users to network commands for remote system access.

System files used specifically by Oracle to facilitate running SQL*Net TCP/IP will be examined. We will explain how to define the TCP/IP device driver for client/server access. We will compare and contrast setting the TWO_TASK environment variable, using "sqlnet" aliases, or specifying the driver during logon. We will discuss "orasrv" and the "tcpctl" utility. This will include examples of log files and statistics.

Background

History of TCP/IP

In 1978, the International Organization for Standardization (ISO) developed the Open Systems Interconnection (OSI) model. This model divides the communications process into seven layers. This set standards that permit a wide variety in the design of computer hardware and software, but standardized output format of each layer to enable diverse hardware and software systems to communicate. Figure 1 shows how TCP/IP fits into this OSI model.

What is TCP/IP? — Transmission Control Protocol (TCP) / Internet Protocol (IP) is a set of rules used to exchange information over a communication channel.

Ethernet Network

Network architecture is a set of specifications, rules and guidelines according to which networking hardware and software is designed and constructed. An ethernet network is a multipoint architecture as illustrated in figure 2.

Figure 1
Illustration of TCP/IP in the ISO OSI Model

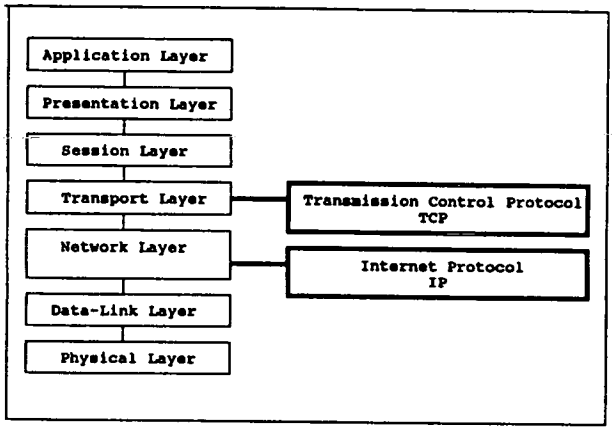
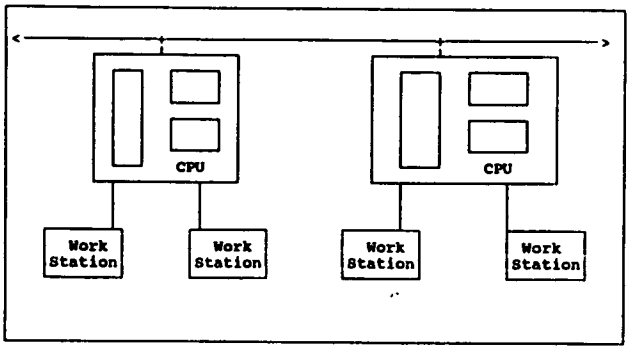


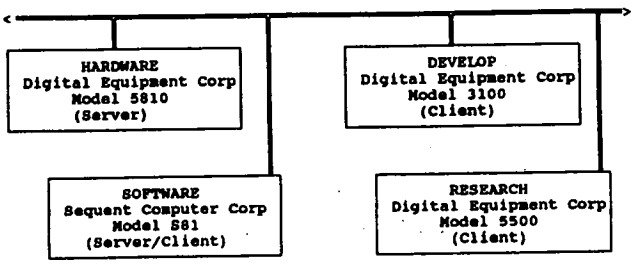
Figure 2
Example of Ethernet Network ArchitectureCable



We are assuming in this discussion that your choice of network architecture is multipoint ethernet and that your hardware will accommodate some form of ethernet connection. The specific hardware used will be the determining factor and a detailed description of the actual hardware installation will be avoided. At our site, we have four nodes on our ethernet network as shown in figure 3.

*This paper was originally presented at the 1991 International Oracle User Week Conference in Miami, Florida, September 30 - October 4, 1991.

Figure 3



Cable

A cable connection will be necessary between each node on your network. The discussion of twisted pair vs. thin-wire vs. thick-wire will be left for another forum. At our site, we use thick-wire ethernet.

Network Utilities

The Transmission Control Protocol (TCP) in the transport layer provides:

- port to port connections;
- a reliable byte stream through: sequencing, retransmission, and checksum;
- flow control; and
- acknowledgements.

The Internet Protocol in the network layer provides:

- addressing (Class A,B,C);
- routing; and fragmentation and reassembly of packets.

Some Unix applications that use TCP/IP protocol are:

ftp — file transfer protocol;
telnet — network terminal protocol;
NFS — network file system;
lpr — remote printing; and
rsh, rexec — remote execution.

Some Unix utilities that use TCP/IP are:

- arp — address resolution protocol resolves internet addresses into ethernet address (physical addresses);
- finger user@host.domain;
- netstat -r;
- talk user@host.domain; and
- ping -l host.domain.

The reasons to use TCP/IP are:

- implement a user-grown set of specifications;
- large installed base (government and education);
- well known;
- has implementations for most existing platforms; and
- allows communication among heterogeneous systems.

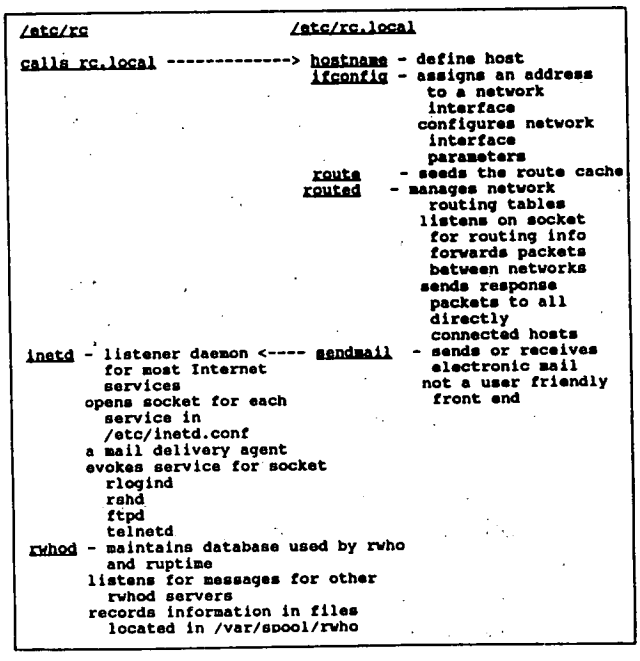
TCP/IP Protocol

There are two types of addressing methods: software-based and hardware-based. Ethernet networks are hardware-based or "peer to peer." These addresses are printed on the ethernet connection hardware and are unique to that interface.

The purpose of transmission protocols is to assure efficient, error-free data transfer by defining the format of data messages. TCP/IP is a common form of transmission protocol used within an ethernet network.

TCP/IP is initialized at "multi-user" startup time. The utilities and daemons are called from /etc/rc file (controls the reboot function and is generic) and /etc/rc.local (called from /etc/rc and is site specific). The sequence of events is shown in figure 4.

Figure 4
Sequence of Network Startup During Reboot



SQL*Net

SQL*Net is an Oracle utility/communication component that allows the exchange and sharing of information stored in different Oracle databases via a 3rd party network. It can be used to connect to any Oracle application and/or database on the network.

Introduction

Our technical advisor asked us to run/setup our systems using a client/server relationship, specifically SQL*Net, at first we were hesitant. Our knowledge in this area was limited. The need to set up the communications/ethernet (ie. tcp/ip, decnet) then learn, understand, and install Oracle's SQL*Net was somewhat intimidating at first.

After breaking the assignment down into individual tasks and defining what each task accomplishes, however, we soon relaxed and began to enjoy operating in the client/server environment.

Theory of Client/server Architecture

One theory behind client/server architecture is to share databases with many users while off-loading the processing. With SQL*Net the SQL statement is parsed on the client and then sent over ethernet to the server. The server processes the statement and returns only the data meeting the criteria or only the response indicating the successful or unsuccessful completion of the command.

A client/server architecture can be used for distributed processing as well as distributed databases. Distributed processing is the off-loading of the processing to many CPU's while accessing a common database. A distributed database is one that is accessed on many systems, making it look as though it is local. With SQL*Net, your system can be either a client, a server, or both.

Tasks/Steps/Actions

There are many components/tasks involved in setting up SQL*Net TCP/IP. We will look at an overview of these and then take a closer look and delve deeper into each step.

1. Setup communication/ethernet
   - *Hardware* — system boards, cabling;
   - *Software* — internet address,configure unix kernel, edit operating system files;
   - *Commands* — what applications/ commands provide access across the network?
2. Setup SQL*Net
   - verify access to all hosts on the network;
   - identify the socket for "orasrv;"
   - identify the available databases on the servers;
   - start "orasrv;"
   - setup "alias;" and
   - access the remote databases.

Steps to Set Up Ethernet

There are four main steps used to set up ethernet. This section lists the steps and provides a brief description of each.

1. **Modify the kernel**
   Change the kernel configuration file to add entries for the Ethernet controller and network-related pseudodevices. Then, regenerate the kernel to include these modifications. This change may include adding an entry for pseudoterminals "pty," etc.

   Some systems require the pseudodevice "ether" to be included in the kernel file, in order to include the Address Resolution Protocol module used in mapping between 48-bit Ethernet and 32-bit Internet addresses. This also needs to be done before regenerating the kernel.
2. **Regen the kernel**
   Regenerate the system kernel to include the above changes.
3. **Make the devices**
   Use the MAKEDEV command to create pseudo-terminals,if necessary, after checking the /dev directory to see if the files already exist.
4. **Edit the tty file**
   Create an entry in /etc/ttys for each pseudo-terminal "tty" file. TTY type should be "network" and no getty should be started.

Steps to Set Up TCP/IP

Three basic steps are included in the TCP/IP set up. Although there may be other specific steps on your platform, the following steps are generic BSD Unix.

1. Edit /etc/rc.local

Select a unique name for your system. Substitute that name for the word "myname" in the hostname command line at the beginning of the

file /etc/rc.local. Also, there should be a line in rc.local similar to the following, anywhere after the first line of the file:

```
/etc/ifconfig se0 `/bin/hostname` up arp
  -trailers
/etc/ifconfig lo0 localhost
/etc/ifconfig de0 host1 netmask 255.255.0.0.
```

This command configures the network hardware interface device 'se0' and 'de0' and a software loopback device 'lo0' on our platform.

2. *Edit System Files*

In order to communicate properly, all systems on the Ethernet should have identical copies of certain files. A description and/or explanation of each of these files and samples follow.

`/etc/hosts`

This file lists each system known on the network and its corresponding Internet address. Internet addresses consist of four bytes of data with some used to specify a network number and the remaining used to specify a system in that network. If none of your systems connect to the Internet, you can choose arbitrary Internet addresses for your systems.

Our system is a closed system; that is, there is no outside network access. For this reason, we have chosen arbitrary Internet addresses for our systems. See Figure 5.

Figure 5

Contents of /etc/hosts file.

Ethernet Address	System Name	Alias	
127.0.0.1	localhost	myhost	#std file entry
1.0.0.2	hardware	HARDWARE	
1.0.0.1	software	SOFTWARE	
1.0.0.3	develop	DEVELOP	
1.0.0.4	research	RESEARCH	

(Since each system is identified in this file in lower case and there is an alias in upper case, they may be specified either way.)

`/etc/hosts.equiv`

This is an equivalence list of hostnames from /etc/hosts that enables execution of "rsh" and "rlogin" without a password. Another name for this list is "trusted hosts." The file consists of trustworthy systems listed one per line. The user accessing the system from a trusted host is required to have an account on the remote system. This also reduces the level of security on this system. See Figure 6.

Figure 6

Contents of /etc/hosts.equiv file.

```
                #on system HARDWARE
software
develop
```

On system HARDWARE, users having an account can execute remote access commands while logged into SOFTWARE or DEVELOP. Even though a user logged into RESEARCH has an account on HARDWARE, they will be asked for a password before a remote access command will be executed on HARDWARE.

This equivalence of systems may be expanded or constrained by creating a .rhosts file in a user's home directory. Additional hosts that the user trusts can be listed in the file. For user root, /etc/hosts.equiv is ignored, and only /.rhosts file is used. On some systems, the tty for root login must be designated as secure in the tty file. This is an additional layer of security if individual users have access from trusted hosts, but root access needs to be restricted.

/etc/services and /etc/protocols These files don't need to be updated to set up a TCP/IP network, although /etc/services must be edited for SQL*Net.

/usr/hosts This file allows a user to log in to a remote host by typing only the name of the host if:

- /usr/hosts is included in the search path in user's .login or .profile, and
- The command /usr/hosts/MAKEHOSTS has been run by the System Administrator during initial network setup.

/usr/spool/mqueue This file contains the network syslog files. These syslog files document the daemon error messages and information from utilities that occurred during the day. These files are automatically maintained by cron.

/usr/spool/rwho This directory contains a file named whod.system-name for each system connected to your LAN. The data for the "rwho" and "ruptime" commands are saved in this directory. If you remove a system from your local network, you should remove that system's file from this directory.

3. Reboot

Reboot the system to allow all file changes to take effect.

Steps to Set Up SQL*Net

Now that ethernet is working on your system, we can look at how to set up the system for SQL*Net.

I am assuming that Oracle, including SQL*Net TCP/IP is installed on all systems involved. You need to know the ORACLE_HOME and ORACLE_SID environment variables and the host names of the systems. If SQL*Net TCP/IP was not installed at installation time, it needs to be installed before you can use it.

For the purpose of this paper we have configured 4 systems. The clients are DEVELOP and RESEARCH. The server is HARDWARE. SOFTWARE is a client and a server. Some of the following steps are needed on the client or the server only while others are needed on both. We have indicated where the command needs to be performed. In the case of a system acting as both a client and server, all commands need to be performed. In the examples, we have indicated the system using "%SYSTEM-NAME->" (i.e., %DEVELOP->), to show where in our network the command was executed.

An Oracle program "orasrv" makes SQL*Net TCP/IP possible. It runs on the server as an Oracle background process (although it is owned by root). Orasrv's purpose is to listen for connections from other systems and facilitate connection to the available Oracle databases. Orasrv can be started with various options including logging activities, debugging, and in-band or out-band breaks. We have found the defaults to be adequate.

1. /etc/hosts (CLIENT and SERVER)

Verify that /etc/hosts has the names of the systems you want to access. If not have your system administrator add them. See Figure 5.

2. telnet (CLIENT and SERVER)

Ensure the communications hardware (ethernet) and software (tcp/ip) are working. Use the "telnet" command (%telnet hostname) to access each machine you want in your network. You must have a password on the system you are trying to access using "telnet."

```
%DEVELOP-> telnet hardware
```

You can also use the "/etc/ping" command to ensure a system is communicating with yours.

```
%SOFTWARE-> /etc/ping develop
```

... the response should be

```
develop is alive
```

3. /etc/oratab (SERVER)

Specify the databases that are available for use to clients. Edit /etc/oratab. The installation script updates this file. If a new database becomes available, you will need to add it.

```
%SOFTWARE-> more /etc/oratab
   # /etc/oratab
   S:/w/oracle:Y


%HARDWARE-> more /etc/oratab
   # /etc/oratab
   H:/usr/oracle:Y
```

4. /etc/services (CLIENT and SERVER)

Identify the socket used for orasrv. Oracle recommends using socket 1525. Edit the /etc/services file. Add a line to include:

```
%HARDWARE-> grep orasrv /etc/services
   orasrv1525/tcporacle
```

where:

orasrv = the name of the service
1525 = the socket #
tcp = protocol used
oracle = alias

The socket # used should be the same # on all system on the network. If /etc/services does not exists, create one.

5. orasrv (SERVER)

Ensure suid bit is set for orasrv in $ORACLE_HOME/bin. Get a long list of the orasrv executable:

```
%SOFTWARE->ls -al $ORACLE_HOME/bin/orasrv
  -rwsr-xr-x 1 root  235456 Apr 25 11:41
     /w/oracle/bin/orasrv
```

The suid bit is identified by the 's' in the fourth position. If the suid bit is not set, set it:

```
%SOFTWARE-> chmod 4755 orasrv
```

Also ensure 'root' is the owner of orasrv:

```
%SOFTWARE-> chown root orasrv
```

Start the orasrv process on the server. You can start the orasrv process in various ways. In rc.local (A), using orasrv from the prompt with or without options (B), executing tcputl via tcpctl start (C).

(A) in rc.local

By having the proper entry in your /etc/rc.local, orasrv will be automatically started when your system is booted. Add

lines similar to the following in your /etc/rc.local file:

```
%SOFTWARE-> grep oracle /etc/rc.local
   ORACLE_HOME=/usr/oracle
   su -oracle -c $ORACLE_HOME/bin/orasrv
```

Using the above entry will start orasrv with default values. They are:

mapfile = /etc/oratab;
in-band breaks;
logging activities on;
debug activities off; and
log file = $ORACLE_HOME/tcp/log/orasrv.log.

(B) starting orasrv at the prompt manually

```
%SOFTWARE-> orasrv
```

This starts orasrv with the default values.

(C) tcputl and tcpctl

Tcpctl is a script that calls tcputl. Tcputl is an Oracle utility that controls and monitor orasrv. To start orasrv using the tcputl and tcpctl method enter

```
%HARDWARE-> tcpctl start
```

This uses the default values above. You can use the tcpctl script to gain statistics on orasrv and to display the version number. To do this enter

```
%HARDWARE-> tcpctl stat
tcputl: Status summary follows
Server is running:
   Started              : 16-JUN-91 05:15:26
   Last connection      : 20-JUN-91 13:36:08
   Total connections    : 942
   Total rejections     : 3
   Active subprocesses  : 7
   ORACLE SIDs          : H
   Default SID          : (null)
Logging mode is ENABLED
%HARDWARE-> tcpctl version
orasrv:Version 1.2.7.1.4-Production on Sun
Jun 16 05:15:25 1991 Copyright © Oracle
Corporation 1979, 1989. All rights reserved
```

You can also get statistics across the network. For instance you can get stats on the SOFTWARE system from the HARDWARE system.

```
%HARDWARE-> tcpctl stat @software
```

The output will look the same as the statistics above except the ORACLE SIDs will be S.

Tcpctl logs connections and other activity if the logging mode is enabled. An example of the contents of a log file from the hardware system is:

```
%HARDWARE-> more orasrv.log
SQL*Net TCP/IP Network Server
Started at 24-JUN-91 07:37:43 by oracle
LOGGING MODE IS ENABLED
Connection request from research at
   20-JUN-91 13:24:3
Connection request from develop at
   20-JUN-91 13:24:58
Connection request from research at
   20-JUN-91 13:26:23
Connection request from research at
   20-JUN-91 13:36:03
Connection request from develop at
   20-JUN-91 13:39:13
STATUS request from software at
   20-JUN-91 13:42:25
VERSION request from software at
   20-JUN-91 13:42:49
STATUS request from research at
   20-JUN-91 13:43:23
STOP request from hardware at 22-JUN-91
   17:12:02
SQL*Net TCP/IP Network Server
Shutdown at 22-JUN-91 17:12:02
```

6. checkTCP (SERVER)

Test SQL*Net TCP/IP using checkTCP -a. The "-a" flag will run through all the tests. Each test can be run separately. The tests are:

- verify orasrv is installed;
- check the socket used for orasrv;
- verify the presence of /etc/hosts file;
- list the available databases based on the info in /etc/oratab;
- check to see if orasrv is started at system bootup; and
- verify the tcp/ip driver is linked to the Oracle kernel.

Logging on Using SQL*Net TCP/IP

The heart and sole for access to a remote system is the SQL*Net TCP/IP connect string. The connect string specifies the driver to use, the remote system to access, and which database on the remote system to access.

The syntax for the connect string is

```
DRIVER:remote system:ORACLE_SID,buffer_size
```

The parameters in the connect strings are delimited by ':'.

In our case the DRIVER is TCP/IP which is always specified by 'T'.

The remote system is a system found in the /etc/hosts file that has been configured as a serv-

er and is running orasrv. For our example, it is 'hardware' or 'software'.

The remote database is specified in the /etc/oratab file on the remote system. Use the ORACLE_SID for this field — 'H' for the 'hardware' system, 'S' for the 'software' system.

There is also a database parameter for the buffer size. We let this parameter default. The default value is 4096.

Our servers, databases, and connect strings are:

server	*ORACLE_SID*	*connect string*
hardware	H	T:hardware:H
software	S	T:software:S.

Options

We use three main options to access remote databases. There are variations to each. The main difference is the first method specifies the TCP/IP driver (see option 1), the second uses alias' to specify the TCP/IP driver (see option 2) and the third method uses the TWO_TASK environment variable (see option 3).

Option 1 — Specify the TCP/IP Driver

You can specify the SQL*Net TCP/IP driver on the command line when calling up SQL*Plus, SQL*Forms, etc (A); when prompted for your username (B); or with the connect command in sqlplus (C).

(A) command line

To access SQL*Plus on the HARDWARE system using the command line enter:

```
%RESEARCH-> sqlplus scott/tiger@T:hardware:H
```

where:

sqlplus = the Oracle utility/application program to run;

scott = your oracle login on the remote system;

tiger = your oracle password on the remote system;

@T:hardware:H = SQL*Net TCP/IP connect string,

@ = delimiter to begin the connect string,

T = use the TCP/IP driver,

hardware = hostname of system to access, and

H = ORACLE_SID identifying which database on the remote system to access.

You must enter the complete logon command with no spaces and no return.

One problem with accessing using the command line method is that your oracle login name and password will show when a unix user does a 'w' command or the 'ps ax' command.

The same syntax can be used to access SQL*Forms, SQL*Calc, or any Oracle application program that accepts username and password on the command line. For example, to access SQL*Forms on the SOFTWARE system, enter:

```
%HARDWARE-> sqlforms scott/tiger@T:software:S
```

(B) prompted

To prevent your oracle login name and password from showing, you can call up the Oracle application program without inputting your login name/password by using:

```
%DEVELOP-> sqlplus -or-
%DEVELOP-> sqlforms
```

When prompted for your user-name, enter your login name/password@connect string. Remember to enter it with no spaces and no return until you have entered the complete logon.

```
For the HARDWARE system, enter ->
     scott/tiger@T:hardware:H
For the SOFTWARE system, enter ->
      scott/tiger@T:software:S
```

(C) connect command

You can also use the 'connect' command from within the local database or from within a remote database if you want to connect to the other remote database.

Fromwithin the local RESEARCH database to access the SOFTWARE database enter:

```
SQL> connect scott/tiger@T:software:S
```

This will log you off the RESEARCH database and log you onto the SOFTWARE system database. The same command will work from the HARDWARE system.

Option 2 — Using Alias' to Specify the TCP/IP Driver

The logon process is simplified by the use of alias'. Alias' are defined in the file /etc/sqlnet or in $HOME/.sqlnet. Edit the /etc/sqlnet file to define system wide alias'. Edit your own $HOME/.sqlnet file to define alias' for your use only. When using an alias, the $HOME/.sqlnet file is read first.

```
%DEVELOP-> more /etc/sqlnet
hwT:hardware:H
swT:software:S
localP:
```

where:
hw = alias
sw = alias
local = alias
T:hardware:H = connect string for system 'hardware'
T:software:S = connect string for system 'software'
P: = connect string (specifies the Pipe Driver on the localsystem)

To use the alias you have the same choices as with option 1, on the command line (A); when prompted (B); or using 'connect' (C). The difference being instead of specifying the connect string you specify the alias.

(A) command line

To execute SQL*Plus and access the database on the HARDWARE system, use:

```
%RESEARCH-> sqlplus scott/tiger@hw
```

To execute SQL*Forms and access the database on the SOFTWARE system, use:

```
%HARDWARE-> sqlforms scott/tiger@sw
```

You will notice the local alias. You can also use the alias to access the local system using the pipe driver in this example,

```
%RESEARCH-> sqlplus scott/tiger@local
```

(B) prompted

```
%DEVELOP-> sqlplus
```

When prompted for username, enter:

```
for HARDWARE system --> scott/tiger@hw
for SOFTWARE system --> scott/tiger@sw
for local system  --> scott/tiger@local
```

(C) connect command

From within SQL*Plus on any database on any system, enter:

for HARDWARE system,

```
SQL> connect scott/tiger@hw
```

for SOFTWARE system,

```
SQL> connect scott/tiger@sw
```

for local system...

```
SQL> connect scott/tiger@local
```

The connect command logs you off the database you were accessing and logs you onto the database specified by the alias.

Option 3 — Using the TWO_TASK Environment Variable

By using the TWO_TASK environment variable you can access a remote host as though you are attached directly. Once we resolved our questions concerning TWO_TASK, we began to enjoy running in a client/server environment. The users of our systems soon became confident in their ability to access the remote database with minimal effort on their part.

When we first started using TCP/IP to access the remote database, our users were instructed to use option 1 or 2 above. They were apprehensive. However, once the TWO_TASK environment variable was set up and unix alias' were created to switch between databases, the users were amazed with the ease of accessing the servers on the network.

Using the TWO_TASK method when you execute an Oracle application program like SQL*Plus provides access to the system and database defined by the TWO_TASK variable without having to specify the connect string.

Setting Up TWO_TASK

As with the other options there are different ways to define the TWO_TASK environment variable. You can define it by typing it at the unix prompt (A), upon login using your .login or .profile (B), or you can create Unix alias' to define TWO_TASK and change from one database to another (C). We use the Unix C shell with a combination of (B) and (C).

(A) at the Unix prompt

Notice the use of the connect string. You can use the previously defined alias' in place of the connect string also. To define the TWO_TASK environment variable at the Unix prompt, you enter:

(in bourne shell) For access to the HARDWARE system,

```
TWO_TASK='T:hardware:H';export TWO_TASK
```

or using the alias created above in /etc/sqlnet or $HOME/.sqlnet,

```
TWO_TASK=hw;export TWO_TASK
```

(in C shell) For access to the SOFTWARE system,

```
setenv TWO_TASK 'T:software:S'
```

or using the alias created above,

```
setenv TWO_TASK sw
```

(B) in the .login or .profile

It is convenient to define the environment variables upon login to the system. You do this by editing the .login (for C shell) or .profile (for bourne shell). The entry in the file is the same as the one you type at the unix prompt as described above.

(in bourne shell) for access to the SOFTWARE system,

```
TWO_TASK='T:software:S';export TWO_TASK
```

(in C shell) for access to the HARDWARE system,

```
setenv TWO_TASK 'T:hardware:H'
```

Edit the appropriate file (.login or .profile) and insert the line(s) above to define TWO_TASK.

(C) alias' in the .cshrc

If you are running with a C shell you can create Unix alias' to define the TWO_TASK environment variable as well as change it. Edit your .cshrc and insert a line similar to:

alias hard "setenv TWO_TASK 'T:hardware:H'"

alias soft "setenv TWO_TASK 'T:software:S'"

alias local "setenv TWO_TASK 'P:'"

using SQL*Net alias', insert a line similar to:

alias hard "setenv TWO_TASK hw"

alias soft "setenv TWO_TASK sw"

alias local "setenv TWO_TASK local"

When you enter 'hard' at the Unix prompt, your TWO_TASK is changed or set to the connect string 'T:hardware:H'. Once TWO_TASK has been defined, the access to Oracle application programs is the same as accessing the local database. Enter 'sqlplus', 'sqlforms', 'sqldba', etc at the prompt. Oracle looks for the TWO_TASK environment variable to identify the database to access. To call up SQL*Plus, enter:

```
%DEVELOP-> sqlplus
```

Enter your remote login and password that correspond with the TWO_TASK when prompted.

A Few Words About PRO*C

If you use the TWO_TASK environment variable to identify the database you want to access, you do not have to make any special changes to your Pro*C source code. I strongly recommend using TWO_TASK if you are using PRO*C.

Without TWO_TASK, you must define variables username, password, and db_string to connect to a remote database:

```
set username = 'scott';
set password = 'tiger';
set db_string = 'T:hardware:H';
```

where:
username = your remote oracle login name;
password = your remote password; and
db_string = the SQL*NET TCP/IP connect string.

The Pro*C program would connect to the database using the variables defined above by including a statement similar to:

```
EXEC SQL CONNECT :username IDENTIFIED BY
:password AT db_name USING :db_string;
```

Creating Database Links

The use of Database Links makes it possible to gain access to remote databases while connected to the local database. Database links are defined using the SQL 'create database link' command. They may be created as public or private. Any user with connect can use a public database link. To create a public database link 'sfwr' on the 'develop' system to access tables as 'scott' residing on the 'software' system, you would enter:

```
SQL> create public database link SFWR
     connect to SCOTT identified by TIGER
     using 'T:software:S';
```

To access the rows in the emp table enter:

```
SQL> select * from emp@sfwr;
```

Create a synonym to simplify this access and make the location of the actual table transparent by entering:

```
SQL> create public synonym emp for
scott.emp@sfwr;
```

Now you can access the emp table by entering:

```
SQL> select * from emp;
```

Defining a Default Host

Note: The SQL*Net TCP/IP manual discusses defining a default host for use by SQL*Net TCP/IP. This information is not completely accurate. On the Unix platform, you cannot define a default database on the server. This results in the inability to use default hosts when accessing Unix to Unix. It is supposed to work accessing Unix to VMS.

Summary

The benefits of operations in a client/server environment more than outweigh the detriments. Some of the benefits are:

- System performance is improved because user-interface cpu usage is offloaded and the MIPS available on clients are utilized;
- The number of backups is reduced;
- Database recovery time after a crash is reduced;
- Control of data is easier because there is only one copy to maintain;

- External database connections are transparent to users;
- Productivity is enhanced;
- Migration in hardware and software is more easily managed;
- Hardware can be scaled to adapt to changing applications; and
- Architecture is flexible.

The detriments of operating in a client/server environment revolve around communications. For example,

- if communications are lost, the client can appear hung, or
- if the server goes down, all clients are unusable.

Setting up and running Oracle in a client/server TCP/IP environment has been a pleasant experience. The key to keeping the access method simple is the use of the TWO_TASK environment variable. Although we had reservations at first, we highly recommend the use of both ethernet and SQL*Net to access data across a network.