CHAPTER 2

Architecture Overview

Oracle is designed to be a very portable database – it is available on every platform of relevance. For this reason, the physical architecture of Oracle looks different on different operating systems. For example, on a UNIX operating system, you will see Oracle implemented as many different operating system processes, virtually a process per major function. On UNIX, this is the correct implementation, as it works on a multi-process foundation. On Windows however, this architecture would be inappropriate and would not work very well (it would be slow and non-scaleable). On this platform, Oracle is implemented as a single, threaded process, which is the appropriate implementation mechanism on this platform. On IBM mainframe systems running OS/390 and zOS, the Oracle operating system-specific architecture exploits multiple OS/390 address spaces, all operating as a single Oracle instance. Up to 255 address spaces can be configured for a single database instance. Moreover, Oracle works together with OS/390 **W**ork**L**oad **M**anager (*WLM*) to establish execution priority of specific Oracle workloads relative to each other, and relative to all other work in the OS/390 system. Even though the physical mechanisms used to implement Oracle from platform to platform vary, the architecture is sufficiently generalized enough that you can get a good understanding of how Oracle works on all platforms.

In the next four chapters, we will look at the three major components of the Oracle architecture:

* In the next chapter, we will cover *files* – we will go through the set of five files that make up the database and the instance. These are the *parameter*, *data*, *temp*, *control* and *redo log* Files. We will also cover other types of files including trace, alert, dump (dmp), datapump, and simple flat files. We will also discuss the impact Automatic Storage Management (ASM) has on our files.
* Then, we will cover the Oracle *memory structures*, referred to as the **S**ystem **G**lobal **A**rea (*SGA*) and **P**rocess **G**lobal **A**rea (*PGA*) – we will go through the relationships between the SGA, PGA, and UGA. Here we also go through the Java pool, shared pool, large pool and various other SGA components

- Lastly, we will cover Oracle's *physical processes or threads* – we will go through the three different types of processes that will be running on the database: *server* processes, *background* processes, and *slave* processes.

It was hard to decide which of these components to cover first. The processes use the SGA, so discussing the SGA before the processes might not make sense. On the other hand, by discussing the processes and what they do, I'll be making references to the SGA. The two are very tied together. The files are acted on by the processes and would not make sense without understanding what the processes do. What I will do is define some terms and give a general overview of what Oracle looks like (if you were to draw it on a whiteboard) and then we'll get into some of the details.

The Server

There are two terms that, when used in an Oracle context, seem to cause a great deal of confusion. These terms are 'instance' and 'database'. In Oracle terminology, the definitions would be:

* *Database* – A collection of physical operating system files. When using Oracle 10*g* Automatic Storage Management (ASM), the database may not appear as individual separate files in the operating system, but the concept remains the same.
* *Instance* – A set of Oracle processes and an SGA. A database instance can exist without any disk storage whatsoever. It might not be the most useful thing in the world – but thinking about it that way will definitely help draw the line between the *instance* and the *database*.

The two are sometimes used interchangeably, but they embrace very different concepts. The relationship between the two is that a database may be *mounted* and *opened* by many instances. An instance may mount and open a single database at any point in time. In fact, it is true to say that an instance will mount and open at most a single database in its entire lifetime! We'll see an example of that in a moment.

Confused even more? Here are some examples that should help clear it up. An instance is simply a set of operating system processes and some memory. They can operate on a database, a database just being a collection of files (data files, temporary files, redo log files, control files). At any time, an instance will have only one set of files associated with it. In most cases, the opposite is true as well; a database will have only one instance working on it. In the special case of Oracle Real Application Clusters (RAC), an option of Oracle that allows it to function on many computers in a clustered environment, we may have many instances simultaneously mounting and opening this one database. This gives us access to this single database from many different computers at the same time. Oracle RAC provides for extremely highly available systems and has the potential to architect extremely scalable solutions.

Let's take a look at a simple example. I've just installed Oracle 10*g* version 10.1.0.3 on my machine. I did a software only installation – no starter databases, nothing, just the software. I'm in the "dbs" directory (on Windows, this would be the "database" directory) and there are no init.ora, no spfiles (Stored Parameter files will be discussed in detail below):

```
[ora10g@localhost dbs]$ pwd
```

```
/home/ora10g/dbs
[ora10g@localhost dbs]$ ls -l
total 0
[ora10g@localhost dbs]$ ps -aef | grep ora10g
ora10g    4173  4151  0 13:33 pts/0    00:00:00 -su
ora10g    4365  4173  0 14:09 pts/0    00:00:00 ps -aef
ora10g    4366  4173  0 14:09 pts/0    00:00:00 grep ora10g
[ora10g@localhost dbs]$ ipcs -a

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

[ora10g@localhost dbs]$ sqlplus / as sysdba

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Dec 19 14:09:44 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Connected to an idle instance.

SQL>
```

The PWD command shows the current working directory (this example was performed on a Linux based computer). We are in the dbs directory and the ls –l shows it is "empty". Using the ps command, we can see all processes being run by the user ora10g (the Oracle software owner in this case) – there are no Oracle database processes whatsoever at this point. I then use the ipcs command, a UNIX command that is used to show inter process communication devices like shared memory, semaphores and the like. Currently – there are none in use on this system at all. I then startup sqlplus – Oracle's command line interface and connect "as sysdba" – as the account that is allowed to do virtually anything in the database. The connection is successful and sqlplus reports we are connected to an idle instance:

```
SQL> !ps -aef | grep ora10g
ora10g    4173  4151  0 13:33 pts/0    00:00:00 -su
ora10g    4368  4173  0 14:09 pts/0    00:00:00 sqlplus   as sysdba
ora10g    4370     1  0 14:09 ?        00:00:00 oracleora10g
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
ora10g    4380  4368  0 14:14 pts/0    00:00:00 /bin/bash -c ps -aef | grep ora10g
ora10g    4381  4380  0 14:14 pts/0    00:00:00 ps -aef
ora10g    4382  4380  0 14:14 pts/0    00:00:00 grep ora10g

SQL> !ipcs -a

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
```

```
------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages


SQL>
```

Our "instance" right now consists solely of that Oracle server process in bold – there is no shared memory allocated yet, no other processes. Let's try to start the instance now:

```
SQL> startup
ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file '/home/ora10g/dbs/initora10g.ora'
SQL>
```

That is the sole file that must exist in order to startup an instance – we need either a "parameter file" (a simple flat file described in more detail below) or a stored parameter file. We'll create one now and put into it the minimal information we need to actually start a database instance:

```
SQL> !echo db_name = ora10g > /home/ora10g/dbs/initora10g.ora

SQL> startup nomount
ORACLE instance started.

Total System Global Area  113246208 bytes
Fixed Size                   777952 bytes
Variable Size              61874464 bytes
Database Buffers           50331648 bytes
Redo Buffers                 262144 bytes
SQL> !ps -aef | grep ora10g
ora10g    4173  4151  0 13:33 pts/0    00:00:00 -su
ora10g    4368  4173  0 14:09 pts/0    00:00:00 sqlplus   as sysdba
ora10g    4404     1  0 14:18 ?        00:00:00 ora_pmon_ora10g
ora10g    4406     1  0 14:18 ?        00:00:00 ora_mman_ora10g
ora10g    4408     1  0 14:18 ?        00:00:00 ora_dbw0_ora10g
ora10g    4410     1  0 14:18 ?        00:00:00 ora_lgwr_ora10g
ora10g    4412     1  0 14:18 ?        00:00:00 ora_ckpt_ora10g
ora10g    4414     1  0 14:18 ?        00:00:00 ora_smon_ora10g
ora10g    4416     1  0 14:18 ?        00:00:00 ora_reco_ora10g
ora10g    4418     1  0 14:18 ?        00:00:00 oracleora10g
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
ora10g    4419  4368  0 14:18 pts/0    00:00:00 /bin/bash -c ps -aef | grep ora10g
ora10g    4420  4419  0 14:18 pts/0    00:00:00 ps -aef
ora10g    4421  4419  0 14:18 pts/0    00:00:00 grep ora10g

SQL> !ipcs -a

------ Shared Memory Segments --------
```

```
key        shmid      owner      perms      bytes      nattch     status
0x99875060 458760     ora10g     660        115343360  8

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xf182650c 884736     ora10g     660        34

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

SQL>
```

Now, we have what I would call "an instance" – the processes needed to actually run a database are all there – such as PMON (process monitor), LGWR (LoG WRiter) and so on – we'll cover them in much more detail later. Additionally, ipcs is for the first time reporting the use of shared memory and semaphored, two important interprocess communication devices on UNIX. Note we have no "database" yet – we have a name of a database, but no database whatsoever. It we try to "mount" this database – it would fail because it quite simply does not yet exist. Let's create it – I've been told that creating an Oracle database involves quite a few steps – but lets see:

```
SQL> create database;
Database created.

 SQL> select name from v$datafile;

NAME
--------------------------------------------------------------------------------
/home/ora10g/dbs/dbs1ora10g.dbf
/home/ora10g/dbs/dbx1ora10g.dbf

SQL> select member from v$logfile;

MEMBER
--------------------------------------------------------------------------------
/home/ora10g/dbs/log1ora10g.dbf
/home/ora10g/dbs/log2ora10g.dbf

SQL> select name from v$controlfile;

NAME
--------------------------------------------------------------------------------
/home/ora10g/dbs/cntrlora10g.dbf

SQL>
```

That is actually all there is to creating a database. In the real world however, we would use a slightly more complicated form of the CREATE DATABASE command – to tell Oracle where to put the log files, datafiles, control files and so on. But here we now have a fully operational database (we would need to run $ORACLE_HOME/rdbms/admin/catalog.sql and other catalog scripts to get the rest of the data dictionary we use everyday built – but we have a database here). Oracle used defaults to

put everything together and created a database as a set of persistent files. If we close this database and try to open it again, we will discover:

```
SQL> alter database close;
Database altered.

SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-16196: database has been previously opened and closed
```

That shows that an instance can mount and open at most one database in it's life. We must discard this instance and create a new one in order to open this or any other database.

To recap –

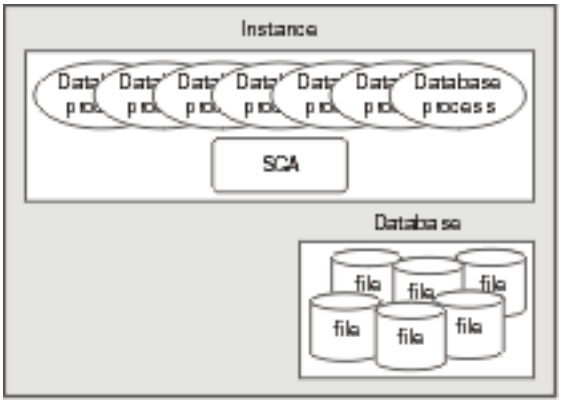
* an *instance* is a set of processes and memory
* a *database* is a collection of data stored on disk
* an instance can mount an open a single database, ever
* a database may be mounted and opened by one or more instances (using RAC)

In most cases, there is a one-to-one relationship between an instance and a database. This is how the confusion surrounding the terms probably arises. In most peoples' experience, a database is an instance, and an instance is a database.

In many test environments, however, this is not the case. On my disk, I might have, five separate databases. On the test machine, I have Oracle installed once. At any point in time there is only one instance running, but the database it is accessing may be different from day to day or hour to hour, depending on my needs. By simply having many different configuration files, I can mount and open any one of these databases. Here, I have one 'instance' but many databases, only one of which is accessible at any point in time.

So now when someone talks of the instance, you know they mean the processes and memory of Oracle. When they mention the database, they are talking of the physical files that hold the data. A database may be accessible from many instances, but an instance will provide access to exactly one database at a time.

Now we might be ready for an abstract picture of what Oracle looks like:



Insert 2433f0201scrap.gif

Figure 2-1. Insert figure caption here.

In its simplest form, this is it. Oracle has a large chunk of memory call the SGA where it will: store many internal data structures that all processes need access to; cache data from disk; cache redo data before writing it to disk; hold parsed SQL plans, and so on. Oracle has a set of processes that are 'attached' to this SGA and the mechanism by which they attach differs by operating system. In a UNIX environment, they will physically attach to a large shared memory segment – a chunk of memory allocated in the OS that may be accessed by many processes concurrently. Under Windows, they simply use the C call malloc() to allocate the memory, since they are really threads in one big process. Oracle will also have a set of files that the database processes/threads read and write (and Oracle processes are the only ones allowed to read or write these files). These files will hold all of our table data, indexes, temporary space, redo logs, and so on.

If you were to start up Oracle on a UNIX-based system and execute a ps (process status) command, you would see that many physical processes are running, with various names. We saw an example of that above when we saw "pmon", "smon" and the like. I will cover what each of these processes are, but they are commonly referred to as the *Oracle background processes*. They are persistent processes that make up the instance and you will see them from the time you start the database, until the time you shut it down. It is interesting to note that these are processes, not programs. There is only one Oracle program on UNIX; it has many personalities. The same 'program' that was run to get ora_pmon_ora10g, was used to get the process ora_ckpt_ora10g. There is only one binary, named simply oracle. It is just executed many times with different names. On Windows XP Pro, using the pslist tool (http://www.sysinternals.com/ntw2k/freeware/pslist.shtml), I will find only one process, Oracle.exe. Again, on NT there is only one binary program. Within this process, we'll find many threads representing the Oracle background processes. Using pslist (or any of a number of tools) we can see these threads:

```
C:\Documents and Settings\tkyte>pslist oracle

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for ORACLE-N15577HE:

Name                Pid Pri Thd  Hnd    Priv        CPU Time    Elapsed Time
oracle             1664   8  19  284 354684     0:00:05.687     0:02:42.218
```

Here we can see there are 19 threads (Thd in the display) contained in the single Oracle process. These threads represent what were processes on UNIX – they are the pmon, arch, lgwr, and so on bits of Oracle. We can use pslist to see more details about each:

```
C:\Documents and Settings\tkyte>pslist -d oracle

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Thread detail for ORACLE-N15577HE:
```

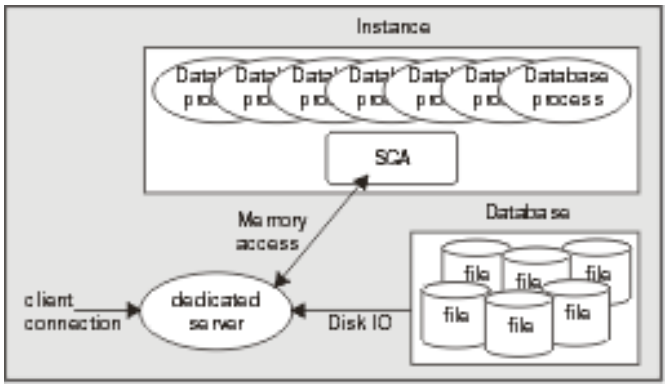
oracle 1664:

Tid	Pri	Cswtch	State	User Time	Kernel Time	Elapsed Time
1724	9	148	Wait:Executive	0:00:00.000	0:00:00.218	0:02:46.625
756	9	236	Wait:UserReq	0:00:00.000	0:00:00.046	0:02:45.984
1880	8	2	Wait:UserReq	0:00:00.000	0:00:00.000	0:02:45.953
1488	8	403	Wait:UserReq	0:00:00.000	0:00:00.109	0:02:10.593
1512	8	149	Wait:UserReq	0:00:00.000	0:00:00.046	0:02:09.171
1264	8	254	Wait:UserReq	0:00:00.000	0:00:00.062	0:02:09.140
960	9	425	Wait:UserReq	0:00:00.000	0:00:00.125	0:02:09.078
2008	9	341	Wait:UserReq	0:00:00.000	0:00:00.093	0:02:09.062
1504	8	1176	Wait:UserReq	0:00:00.046	0:00:00.218	0:02:09.015
1464	8	97	Wait:UserReq	0:00:00.000	0:00:00.031	0:02:09.000
1420	8	171	Wait:UserReq	0:00:00.015	0:00:00.093	0:02:08.984
1588	8	131	Wait:UserReq	0:00:00.000	0:00:00.046	0:02:08.890
1600	8	61	Wait:UserReq	0:00:00.000	0:00:00.046	0:02:08.796
1608	9	5	Wait:Queue	0:00:00.000	0:00:00.000	0:02:01.953
2080	8	84	Wait:UserReq	0:00:00.015	0:00:00.046	0:01:33.468
2088	8	127	Wait:UserReq	0:00:00.000	0:00:00.046	0:01:15.968
2092	8	110	Wait:UserReq	0:00:00.000	0:00:00.015	0:01:14.687
2144	8	115	Wait:UserReq	0:00:00.015	0:00:00.171	0:01:12.421
2148	9	803	Wait:UserReq	0:00:00.093	0:00:00.859	0:01:09.718

And if we were to log into this database using a “dedicated server”, we would see a new process get created just to service us:

```
C:\Documents and Settings\tkyte>sqlplus tkyte/tkyte

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Dec 19 15:41:53 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options

tkyte@ORA10G> host pslist oracle

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for ORACLE-N15577HE:

Name                Pid Pri Thd  Hnd    Priv        CPU Time    Elapsed Time
oracle             1664   8  20  297 356020     0:00:05.906     0:03:21.546
tkyte@ORA10G>
```

Now you can see there are 20 threads – instead of 19, the extra thread is our dedicated server process (more information on what a dedicated server process is exactly is coming shortly). When we log out, it will go away. On UNIX, I would probably see another process get added to the list of oracle processes running. This brings us to the next iteration of the diagram. The previous diagram gave a conceptual depiction of what Oracle would look like immediately after starting. Now, if we were to connect to Oracle in its most commonly used configuration, we would see something like:



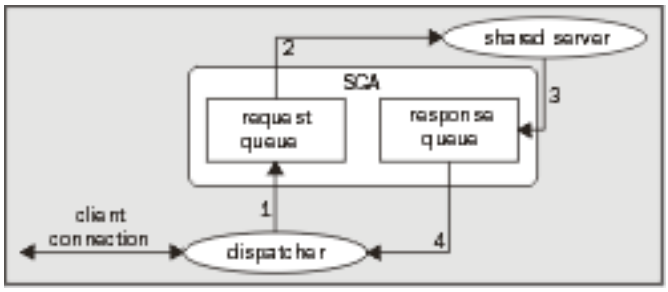
Insert 2433f0202scrap.gif

Figure 2-2. Insert figure caption here.

Typically, Oracle will create a new process for me when I log in. This is commonly referred to as the *dedicated server* configuration, since a server process will be dedicated to me for the life of my session. For each session, a new dedicated server will appear in a one-to-one mapping. My client process (whatever program is trying to connect to the database) will be in direct contact with this dedicated server over some networking conduit, such as a TCP/IP socket. It is this server that will receive my SQL and execute it for me. It will read data files, it will look in the database's cache for my data. It will perform my update statements. It will run my PL/SQL code. Its only goal is to respond to the SQL calls that I submit to it.

Oracle may also be accepting connections in a manner called shared server (formally known as Multi-Threaded Server or simply MTS) in which we would *not* see an additional thread created, or a new UNIX process appear for each user connection. In shared server, Oracle uses a pool of 'shared processes' for a large community of users. Shared servers are simply a connection pooling mechanism. Instead of having 10000 dedicated servers (that's a lot of processes or threads) for 10000 database sessions, shared server would allow me to have a small percentage of this number of processes/threads, which would be (as their name implies) shared by all sessions. This allows Oracle to connect many more users to the database than would otherwise be possible. Our machine might crumble under the load of managing 10000 processes, but managing 100 or 1000 processes would be doable. In shared server mode, the shared processes are generally started up with the database, and would just appear in the ps list.

A big difference between shared and dedicated server connections is that the client process connected to the database never talks directly to a shared server, as it would to a dedicated server. It cannot talk to a shared server since that process is in fact shared. In order to share these processes we need another mechanism through which to 'talk'. Oracle employs a process (or set of processes) called *dispatchers* for this purpose. The client process will talk to a dispatcher process over the network. The dispatcher process will put the client's request into a request queue in the SGA (one of the many things the SGA is used for). The first shared server that is not busy will pick up this request, and process it (for example, the request could be UPDATE T SET X = X+5 WHERE Y = 2. Upon completion of this command, the shared server will place the response in a response queue. The dispatcher process is monitoring this queue and upon seeing a result, will transmit it to the client. Conceptually, the flow of an shared server request looks like this:



Insert 2433f0203scrap.gif

Figure 2-3. Insert figure caption here.

The client connection will send a request to the dispatcher. The dispatcher will first place this request onto the request queue in the SGA (1). The first available shared server will dequeue this request (2) and process it. When the shared server completes, the response (return codes, data, and so on) is placed into the response queue (3) and subsequently picked up by the dispatcher (4), and transmitted back to the client.

As far as the developer is concerned, there is no difference between a shared server connection and a dedicated server connection. So, now that we understand what dedicated server and shared server connections are, this begs the questions, 'How do we get connected in the first place?' and 'What is it that would start this dedicated server?' and 'How might we get in touch with a dispatcher?' The answers depend on our specific platform, but in general, it happens as described below.

We will investigate the most common case – a network based connection request over a TCP/IP connection. In this case, the client is situated on one machine, and the server resides on another machine, the two being connected on a TCP/IP network. It all starts with the client. It makes a request to the Oracle client software to connect to database. For example, you issue:

```
[tkyte@localhost tkyte]$ sqlplus scott/tiger@ora10g.localdomain

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Dec 19 16:16:41 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options

scott@ORA10G>
```

Here the client is the program SQL*PLUS, scott/tiger is my username and password, and ora10g.localdomain is a TNS service name. TNS stands for **T**ransparent **N**etwork **S**ubstrate and is 'foundation' software built into the Oracle client that handles our remote connections – allowing for peer-to-peer communication. The TNS connection string tells the Oracle software how to connect to the remote database. Generally, the client software running on your machine will read a file called TNSNAMES.ORA. This is a plain text configuration file commonly found in the [ORACLE_HOME]\network\admin directory that will have entries that look like:

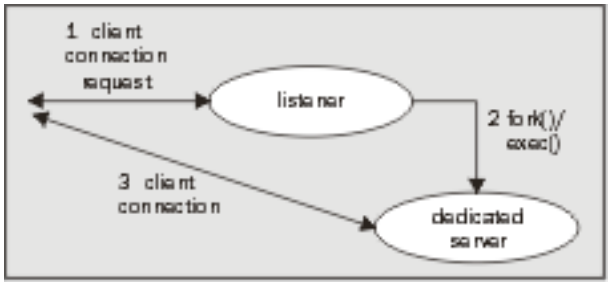
```
ORA10G.LOCALDOMAIN =
  (DESCRIPTION =
```

```
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost.localdomain)(PORT = 1521))
  )
  (CONNECT_DATA =
    (SERVICE_NAME = ora10g)
  )
)
```

It is this configuration information that allows the Oracle client software to turn ora10g.localdomain into something useful – a hostname, a port on that host that a 'listener' process will accept connections on, the *service name* of the database on the host to which we wish to connect, and so on. There are other ways that this string, ora10g.localdomain, could have been resolved. For example it could have been resolved using Oracle Internet Directory (OID), which is a distributed LDAP (Lightweight Directory Access Protocol) server for the database, similar in purpose to DNS for hostname resolution. However, use of the TNSNAMES.ORA is common in most small to medium installations where the number of copies of such a configuration file is manageable.

Now that the client software knows where to connect to, it will open a TCP/IP socket connection to the machine localhost.localdomain on the port 1521. If the DBA for our server has setup Net8, and has the listener running, this connection may be accepted. In a network environment, we will be running a process called the *TNS Listener* on our server. This listener process is what will get us physically connected to our database. When it receives the inbound connection request, it inspects the request and, using its own configuration files, either rejects the request (no such database for example, or perhaps our IP address has been disallowed connections to this host) or accepts it and goes about getting us connected.

If we are making a dedicated server connection, the listener process will create a dedicated server for us. On UNIX, this is achieved via a fork() and exec() system call (the only way to create a new process after initialization in UNIX is fork()). We are now physically connected to the database. On Windows, the listener process requests the database process to create a new thread for a connection. Once this thread is created, the client is 'redirected' to it, and we are physically connected. Diagrammatically in UNIX, it would look like this:

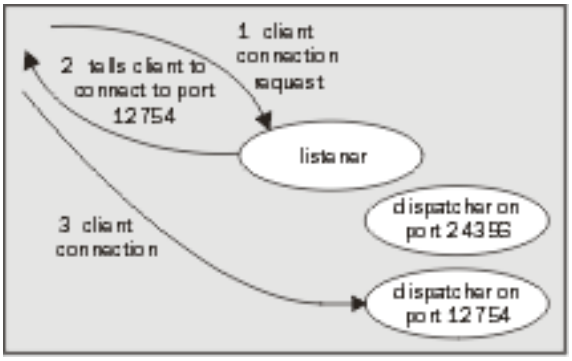


Insert 2433f0204scrap.gif

Figure 2-4. Insert figure caption here.

On the other hand, if we are making a shared server connection request, the listener will behave differently. This listener process knows the dispatcher(s) we have running on the database. As connection requests are received, the listener will choose a dispatcher process from the pool of available dispatchers. The listener will send back to the client the connection information describing how the client can connect to the dispatcher process. This must be done because the listener is running on a well-known hostname and port on that host, but the

dispatchers will be accepting connections on 'randomly assigned' ports on that server. The listener is aware of these random port assignments and picks a dispatcher for us. The client then disconnects from the listener and connects directly to the dispatcher. We now have a physical connection to the database. Graphically this would look like this:

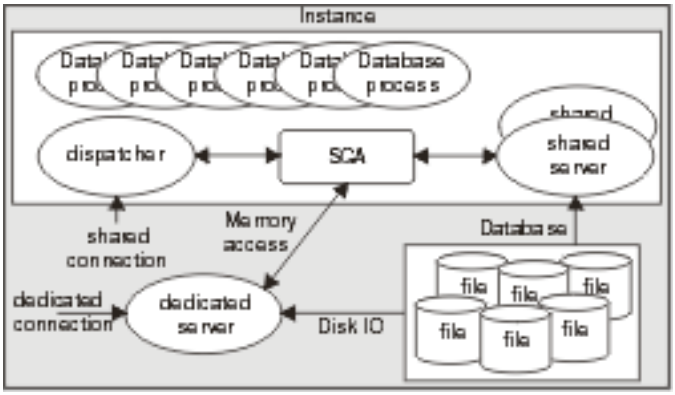


Insert 2433f0205scrap.gif

Figure 2-5. Insert figure caption here.

Summary

So, that is an overview of the Oracle architecture. Here, we have looked at what an Oracle instance is, what a database is, and how you can get connected to the database either through a dedicated server, or a shared server. The following diagram sums up what we've seen so far; showing the interaction between the a client using a shared server connection and a client using a dedicated server connection. It also shows that an Oracle instance may use both connection types simultaneously (in fact an Oracle database *always* supports dedicated server connections – even when configured for shared server):



Insert 2433f0206scrap.gif

Figure 2-6. Insert figure caption here.

Now we are ready to take a more in-depth look at the processes behind the server, what they do, and how they interact with each other. We are also ready to look inside the SGA to see what is in there, and what its purpose it. We will start by looking at the types of files Oracle uses to manage the data and the role of each file type.