The Oracle Background Processes for Sybase Pros

By Mich Talebzadeh

September 2006

In this article we will be looking at the Oracle background processes and what they mean.

Mich Talebzadeh is a consultant and a technical architect who has worked with Sybase and other databases since the early 1990s. He is the co-author of the book "Sybase Transact SQL Programming Guidelines and Best Practices" and the author of the forthcoming book "Oracle and Sybase, Concepts and Contrasts". Mich can be reached at mich@peridale.co.uk.

The Components of an Oracle Instance

Oracle memory usage can be broken down into two basic types, namely private and shared. Private memory is used only by a single process. In contrast, shared memory is used by more than one process. **An Oracle instance consists of shared memory and a collection of background processes.** The shared memory part is obtained from the shared memory on host and consists of the shared memory, private memory and the process and parameter overheads as follows:

Total memory used by Oracle = System Global Area (SGA) + Aggregate Program Global Area (PGA) + 40MB of Oracle process size + the defaults for empty parameters (1MB)

The 40MB is required for the Oracle process and around 1MB is allocated to the empty initialisation parameters of Oracle. So effectively Oracle memory is consumed by the SGA and the aggregate PGA. The largest segment of shared memory with Oracle is usually the SGA. SGA has various components. Internally within Oracle the size of SGA is controlled by two parameters **SGA_MAX_SIZE** and **SGA_TARGET**. The parameter SGA_TARGET is new in Oracle 10g. PGA is the private memory used by a single process. The upper limit on PGA is set by the initialization parameter **PGA_AGGREGATE_TARGET**.

The Oracle Background Processes

The Oracle background processes and the server processes share the SGA. When you start up an Oracle instance, the background processes are created from the Oracle binary. *Background processes, as the name says, are processes running behind the scene and are meant to perform certain maintenance activities or deal with abnormal conditions arising in the lifetime of the instance*. As we will see, each background process is meant for a specific purpose and its role is well defined. Throughout this book I will use examples of my Oracle 10g on Linux. My oracle instance is called "mydb". Oracle calls this name **SID**. SID is the site identifier for an Oracle instance. On a UNIX host, multiple Oracle instances could co-exit (much like multiple ASE servers). Additionally, it is worth mentioning that these days, a majority of beefy data servers have a combination of Oracle and Sybase servers, running on the same host.

When you create an Oracle instance, a so-called magic key is created by hashing together the UNIX environment variables $ORACLE_HOME and $ORACLE_SID. On a given host, this magic key is unique (since you cannot have two instances with the same name and the same version running on the same host). Oracle uses this unique key to attach this instance to the relevant shared memory segment. Otherwise if either of these variables is set wrongly, the instance cannot be started.

Oracle can use either the "process based" or the "thread based (otherwise known as Shared Server or Multi-Threaded Server MTS)" architectures. On UNIX platform the default behaviour of Oracle is processed based. That is all of the processes are operating system processes and not threads. On Windows, Oracle uses a thread based model within a single process. On UNIX you can change from process based to thread based/shared server mode. However you will need to configure Oracle Net Services accordingly. This will be explained later.

I will be doing a fast track to connecting to an Oracle instance here. This process will be explained later. You can use an Oracle client utility called SQL*Plus (it is much like isql in Sybase) to connect to, start and stop an Oracle instance. Assuming that you have set up the environment variables ORACLE_BASE and ORACLE_HOME (read $SYBASE in ASE) correctly and of course the PATH, you will specify which Oracle instance you want to connect to by setting the environment variable ORACLE_SID (read $DSQUERY in ASE). For now to start an Oracle instance, you log in to the host that Oracle is running with the UNIX login "Oracle" (or anyone else who happens to be a member of UNIX "dba" group). Oracle allows you to connect to an idle instance using the OS authentication method without knowing the relevant "sys" user password. "sys" user is the most powerful user in Oracle (equivalent to ASE's "sa" login/user). As an example I have set up the above environment variables in my Linux host as follows:

```
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/10.1.0/db_1
ORACLE_SID=mydb
```

Note that Oracle provides a shell routine *oraenv* for this purpose. Once you are there you just type the following at the operating system level:

```
oracle@linux:/home/oracle% ${ORACLE_HOME}/bin/sqlplus "/ as
sysdba"

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Jul 17 08:47:45 2005

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

Connected to an idle instance.

idle>
```

The quotes in sqlplus "/ as sysdba" are important. SYSDBA is a special database role that gives you the necessary privileges to perform administration functions. Now to start the instance you can do the following:

```
idle> startup
ORACLE instance started.

Total System Global Area 1526726656 bytes
Fixed Size                   779256 bytes
Variable Size             384834568 bytes
Database Buffers         1140850688 bytes
Redo Buffers                 262144 bytes
Database mounted.
Database opened.
```

To stop the instance you issue the following command

```
sys@MYDB.MICH.LOCAL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
```

You can log in to a running Oracle instance by issuing the same connection command:

```
oracle@linux:/home/oracle% sqlplus "/ as sysdba"

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Jul 17 09:05:53 2005

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

sys@MYDB.MICH.LOCAL>
```

What does connect / as sysdba mean?

This syntax is used to connect to the Oracle database with very powerful privileges. In particular, it allows the user to stop and start the instance.

When it is used, Oracle does not check the password held for the user in the database's data-dictionary but instead checks that the current operating system userid is defined in the "dba" group (if the OS is UNIX) or ORA_DBA group (if the OS is Windows NT/2000). If the user is in this group, then the user is allowed to connect. If the user is not in this group, an error message *ORA-01031: insufficient privileges* error message, or an *Enter password* prompt is displayed.

The reason the data dictionary password is not used is that Oracle needs some mechanism of checking that the user is sufficiently authorized to start the database even when the database is down. Being a member of this OS group is sufficient proof that one is authorized to stop and start the instance, since only the system administrator (*root* on UNIX or *Administrator* on Windows) can add users to this group.

The **connect / as sysdba** technique can therefore be used as a method of logging in as SYS or SYSTEM when one has forgotten both their passwords: Use **connect / as sysdba** to connect as SYS, then change the SYS and SYSTEM passwords.

The statement:

```
connect anything/anything as sysdba
```

has exactly the same effect as **connect / as sysdba**, ie. it logs you in as SYS with stop and start database privileges. The supplied userid and password are ignored. No password prompt is given providing the current operating system userid is in the appropriate group.

The statement:

```
connect sys as sysdba
```

Or indeed, **connect *anything* as sysdba** causes an **Enter password** prompt to be issued (even if the operating system userid is in the appropriate group), and the password held in the data-dictionary must then be entered to complete the login. This is probably a bug: I suspect the command parser detects that there is no / in the command and therefore that no password has been supplied, before it recognizes that no password is, in fact, necessary because *as sysdba* syntax has been used). If the database is up, the valid password for the user must be entered. I suspect this is the case, but have not tested it yet. If the database is down, then whatever is entered, as the password will be rejected because Oracle will be unable to validate it.

Now I connect to my idle Oracle instance. As mentioned a user must connect as SYSDBA or as SYSOPER privileges if he/she wants to perform administrator tasks. I connect to Oracle on my UNIX host using operating system authentication process.

```
oracle@linux:/home/oracle% sqlplus "/ as sysdba"

SQL*Plus: Release 10.1.0.3.0 - Production on Fri Apr 22 11:05:12 2005

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

Connected to an idle instance.
>idle
```

At this stage there is nothing running on host with respect to my Oracle instance *mydb*. Let me issue a ***ps*** command on the host and see what is going on

```
oracle@linux:/home/oracle% ps -auxww | grep oracle

root      7120  0.0  0.0  6740 1964 ?        S    09:55   0:00 sshd: oracle [priv]
oracle    7122  0.0  0.0  9704 2920 ?        R    09:55   0:00 sshd: oracle@pts/0
oracle    7123  0.0  0.0  1680  592 pts/0    S    09:55   0:00 -ksh
oracle    7143  0.0  0.0  4728 1576 pts/0    R    09:55   0:00 -sh
root     10246  0.0  0.0  6740 1964 ?        S    11:13   0:00 sshd: oracle [priv]
oracle   10248  0.1  0.0  9704 2920 ?        S    11:13   0:00 sshd: oracle@pts/1
oracle   10249  0.0  0.0  1684  588 pts/1    S    11:13   0:00 -ksh
oracle   10269  0.2  0.0  4688 1496 pts/1    S    11:13   0:00 -sh
oracle   10293  0.2  0.1 17152 5960 pts/1    S    11:13   0:00 sqlplus
oracle   10295  0.3  0.2 76356 7564 ?        S    11:13   0:00 oraclemydb
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
```

The only thing really relevant is my server connection to the idle instance. The server processes are named oracleSID (*oraclemydb* in this case). Next I will go and allocate resources (SGA) from shared memory and start the background processes, but do no not mount the database. As explained later “STARTUP NOMOUNT” command tells Oracle to read the initialization file, allocate SGA, start background processes and open the alert and trace files

```
idle> startup nomount
ORACLE instance started.

Total System Global Area 1476395008 bytes
Fixed Size                   779196 bytes
Variable Size             435166276 bytes
Database Buffers         1040187392 bytes
Redo Buffers                 262144 bytes
```

Let us have a look at the OS again and only display the Oracle background processes.

```
oracle@linux:/home/oracle% ps -auxww|grep ora_

oracle   10332  0.0  0.3 1523872 10884 ?     S    11:18   0:00 ora_pmon_mydb
oracle   10334  0.0  0.2 1523260 9300 ?      S    11:18   0:00 ora_mman_mydb
oracle   10336  0.0  0.3 1524704 10888 ?     S    11:18   0:00 ora_dbw0_mydb
oracle   10338  0.0  0.2 1523256 9336 ?      S    11:18   0:00 ora_lgwr_mydb
oracle   10340  0.0  0.2 1523256 10180 ?     S    11:18   0:00 ora_ckpt_mydb
oracle   10342  0.0  0.2 1523264 9336 ?      S    11:18   0:00 ora_smon_mydb
oracle   10344  0.0  0.2 1523256 9292 ?      S    11:18   0:00 ora_reco_mydb
oracle   10346  0.0  0.2 1523276 9284 ?      S    11:18   0:00 ora_cjq0_mydb
oracle   10348  0.0  0.2 1524440 9380 ?      S    11:18   0:00 ora_d000_mydb
oracle   10350  0.0  0.2 1523864 9104 ?      S    11:18   0:00 ora_s000_mydb
```

Note that anything with format “ora_<ProcessName>_<SID>” (and in this case “ora_<ProcessName>_mydb” such as “ora_pmon_mydb”) is an Oracle background process. They were started when I issued the command “STARTUP NOMOUNT”.There is an OS process for every Oracle background process. As the name says, these background processes run behind the scene and perform certain maintenance activities or deal with abnormal conditions arising in the lifetime of the instance.

SYBASE Notes These Oracle background processes are in essence similar to Sybase ASE internal processes as shown below:

```
1> sp_who
2> go
 fid    spid   status       loginame                       origname
hostname   blk_spid dbname                         cmd             block_xloid
 ------ ------ ------------ ------------------------------ ------------------------------
---------- -------- ------------------------------ --------------- -----------
      0      1 sleeping     NULL                           NULL
0        master                     CHKPOINT WRKR            0
      0      2 sleeping     NULL                           NULL
0        master                     DEADLOCK TUNE            0
      0      3 sleeping     NULL                           NULL
0        master                     ASTC HANDLER             0
      0      4 sleeping     NULL                           NULL
0        master                     ASTC HANDLER             0
      0      5 sleeping     NULL                           NULL
0        master                     ASTC HANDLER             0
      0      6 sleeping     NULL                           NULL
0        master                     ASTC HANDLER             0
      0      7 sleeping     NULL                           NULL
0        master                     CHECKPOINT SLEEP         0
      0      8 sleeping     NULL                           NULL
0        master                     HK WASH                  0
      0      9 sleeping     NULL                           NULL
0        master                     HK GC                    0
      0     10 sleeping     NULL                           NULL
0        master                     HK CHORES                0
      0     11 sleeping     NULL                           NULL
0        master                     PORT MANAGER             0
      0     12 sleeping     NULL                           NULL
0        master                     NETWORK HANDLER          0
      0     13 sleeping     NULL                           NULL
0        master                     NETWORK HANDLER          0
      0     15 sleeping     NULL                           NULL
0        master                     CHKPOINT WRKR            0
      0     19 sleeping     NULL                           NULL
0        master                     CHKPOINT WRKR            0
      0     20 sleeping     NULL                           NULL
0        master                     CHKPOINT WRKR            0
      0     21 sleeping     NULL                           NULL
0        master                     PLC FLUSHER              0
```

```
      0      41 sleeping      NULL                               NULL
0        master                          LOG WRITER                0
```

ASE internal processes have suid = 0 indicating that they are ASE's internal or background processes (please note that in this respect replication agent processes REP AGENT are also considered as background processes). They perform more and less the same function as Oracle background processes. Therefore, Oracle's BACKGROUND processes are equivalent to ASE's internal threads.

To get a list of all Oracle background processes you can use the following SQL command

```
select * from v$process where background = 1;
```

Similarly for Sybase you can try the following:

```
select * from master..sysprocesses where
suid = 0
go
```

Note that to execute a command in SQL*Plus you terminate it with a **";"** column. In ASE you do so by issuing a **"go"** command on a new line

Turning to Oracle, let us see briefly, what some of these background processes do:

- **PMON**
  The Process Monitor checks if a user process fails and if so, does all cleaning up of resources that the user process has acquired such as rolling back the uncommitted transactions and releasing locks. PMON also does service registration with the Oracle listener.

SYBASE Notes In ASE, for aborted threads, the rollback of uncommitted transactions is handled by the thread itself. That is the reason why threads can hang on even after being killed. The ASE internal threads *Port Manager* and *Network Handlers(s)* handle the entire network and connection related stuff.

- **MMAN**
  MMAN dynamically adjust the sizes of the SGA components. It is a new process added to Oracle 10g as part of automatic shared memory management.

SYBASE Notes There is no dynamic memory allocation in ASE yet. However, you can use *sp_sysmon with "cache wizard"* option to advise you of changes required to various ASE structures in order to achieve better performance

- **DBWR**
  The Database Writer writes *dirty blocks* from the *database* buffer to the datafiles. Dirty blocks need to be flushed out to disk to make room for new blocks in the cache. The checkpoint process (CKPT) uses DBWR to write dirty buffers to disk. This is normally referred to as *scattered or random writes*, because the dirty blocks to be written could be anywhere on the disk(s). You can configure more than one DBWR process up to 10. This depends on the number of CPUs allocated to the instance. To have more than one DBWR only make sense if each DBWR has been allocated its own list of blocks to write to disk. This is done through the initialization parameter DB_BLOCK_LRU_LATCHES. If this parameter is not set correctly, multiple DBWRs can end up contending for the same block list.

SYBASE Notes In ASE the role of Oracle's DBWR(s) and CKPT is performed by *Checkpoint Worker(s)* and *Checkpoint Sleep* threads. Checkpoint Worker thread performs the actual flushing of dirty pages from the cache to disk. Checkpoint sleep performs periodic checkpoints.

- **LGWR**
  The Log Writer writes the redo log buffer from the SGA to the *online redo log file*. LGWR does this every three seconds, whenever a user issues a commit, a checkpoint happens or whenever the redo log buffer is 1MB or more than 1/3 full. This means that there is no point in making the redolog buffer more than 3MB. Note that the logs are written via *sequential writes* as opposed to scattered writes.

SYBASE Notes In ASE under normal operations there is no single process like LGWR. ASE has multiple databases each with its own transaction log. ASE uses the concept of group commit to flush the log Cache to disk. Each user in ASE is allocated a chunk of cache from ASE memory called User Log Cache (ULC). When a thread generates a log records, the records are written to ULC. When the transaction commits or aborts or the ULC is full, ULC is flushed to the Log Cache. In databases with high transaction requirements, you can turn on a database option called "Asynchronous log service" or ALS. You can only turn on this option if you have four or more ASE engines. Turning on this option will start two ASE internal threads. These are *PLC Flusher* and *Log Writer*. The PLC Flusher flushes the User Log Cache to the Log Cache. The Log Writer performs a similar role to Oracle's LGWR by writing the Log Cache pages to disk (syslogs table).

It is important to understand the concept of scattered writes vis-à-vis sequential writes. Both Oracle and Sybase checkpoint processes deploy scattered or random writes. Typically, the dirty blocks or pages contain data from different tables etc. A write process may have to update data on different locations on different disks. These updates could contain data pertinent to tables, indexes and so forth. With the widespread use of Storage Area Networks (SAN) disks, the random writes are significantly faster than before. However, the random writes are slower than sequential writes performed by LGWR in Oracle and by means of "Group Commit" or Log Writer in ASE.

- **CKPT**

  The Checkpoint Process regularly initiates a checkpoint. A checkpoint process

  - Flushes the redo log buffer to redo log files by means of *LGWR*
  - Writes a checkpoint record to the redo log file
  - Uses *DBWR* to write *all* dirty blocks back to the datafiles, thus synchronizes the database.
  - Updates the file headers of the data files with information about the last checkpoint performed
  - Update the control files about the last checkpoint

`SYBASE Notes` In ASE the role of Oracle's DBWR(s) and CKPT is performed by *Checkpoint Worker(s)* and *Checkpoint Sleep* threads. Checkpoint Worker thread performs the actual flushing of dirty pages from the cache to disk. Checkpoint sleep performs periodic checkpoints.

- **SMON**

  The System Monitor carries out a `crash recovery` when a crashed instance is started up again. SMON cleans temporary segments. It also merges contiguous areas of free space in the datafiles, a process known as coalescing.

`SYBASE Notes` In ASE this role is performed by threads, all part of the *House Keeper (HK)* family. These are *HK Chores*, *HK Wash* and *HG GC (Garbage Collector)*.

- **RECO**

  The Distributed Transaction Recovery Process finds pending distributed transactions and resolves them. Pending distributed transactions are two-phase commit transactions involving multiple databases. The database that the transaction started is normally the coordinator. It will send request to other databases involved in two-phase commit if they are ready to commit. If a negative request is received from one of the other sites, the entire transaction will be rolled back. Otherwise, the distributed transaction will be committed on all sites. However, there is a chance that an error (network related or otherwise) causes the two-phase commit transaction to be left in pending state (i.e. not committed or rolled back!). It is the role of the RECO process to liaise with the coordinator to resolve the pending two-phase commit transaction. RECO will either commit or rollback this transaction.

`SYBASE Notes` In ASE, there are two types of two-phase commit transactions. Remember that in an ASE, there are multiple databases. First, there is the intra-server cross database two-phase commit. This type of two-phase commit is handled by the internal thread *ASTC Handler(s)*. Secondly, there is the two-phase commit between databases on remote Sybase servers. Remote Procedure Calls (RPCs), and Component Integration Services (CIS) such as proxy tables rely on this type of two-phase commit. In this case, ASE provides a system database called *sybsystemdb* that stores information about distributed transactions. These types of transactions in ASE are called SYB2PC (Sybase two-phase commit). *sybsystemdb* database has a table called *spt_committab*. This table stores information about and tracks the completion status of each two-phase commit transactions.

- **ARC**

  The Archiver process copies an online redo log file to another location when the redo log file is filled up. Archive log files are used for media recovery (in case of a hard disk failure and for maintaining an Oracle standby database via log shipping). There can be up to ten archiver processes. Archiver is only present if the database is running in archivelog mode and automatic archiving is enabled. LGWR process is responsible for starting multiple ARC processes when the workload increases. Unless archiver completes the copying of a redo log file, it is not released to LGWR for overwriting.

SYBASE Notes ASE does not have any internal threads for archiving. This is perhaps because ASE maintains multiple databases in the same server. In order to dump an ASE's database transaction log, the database should not have the option "truncate log on checkpoint" enabled (i.e. in Oracle's terminology it should be in archivelog mode). There are two ways of backing up transaction logs in an ASE database. The most common method is to write a shell script or use ASE Job Scheduler to dump the transaction log of the database to a specific location on a periodic basis. Additionally, ASE automatically executes a procedure called *sp_thresholdaction* when the number of free pages on the log segment falls below the last-chance threshold. This procedure has to be written by the DBA or the database owner (DBO). Within this procedure, you can specify that the transaction log to be dumped to a particular location. This is somehow similar to what the Oracle Archiver performs, although the implementation methods are different. Note that you cannot backup transaction log of system databases in ASE.

- **CJQ0**

  This is the Oracle's dynamic job queue coordinator. It periodically selects jobs that need to be run, scheduled by the Oracle job queue. The coordinator process dynamically spawns job queue slave processes (J000…J999) to run the jobs. These jobs could be PL/SQL statements or procedures on an Oracle instance. Please note that this is not a persistent process. It comes and goes

SYBASE Notes This process performs similar role to ASE's Job scheduler.

- **Dnnn**

  The Dispatcher Process is used in a shared server environment. Dnnn supports shared server configuration by allowing user processes to share a limited number of server processes. Shared server configuration is explained later. Oracle 10g comes with a single Dnnn process configured by default.

SYBASE Notes There is no equivalent dispatcher process in ASE. ASE architecture is different.

- **Snnn**

  The Shared Server Process is used in a shared server environment. Each shared server process serves multiple client requests in the shared server configuration. Shared server processes and dedicated server processes provide the same functionality, except shared server processes are not associated with a specific user process. That is a shared server process is not a shadow process.

SYBASE Notes At a simplistic level you can think of Oracle shared server processes as somehow equivalent to ASE engines. ASE engines like Oracle Snnn serve multiple clients.

There are other Oracle processes that are not covered here. For example, processes specifically dealing with Oracle Real Application Clusters (such as LMS - the Lock Manager Service, LMON – the Enqueue Service Monitor etc).The user is referred to the appropriate Oracle documentation.

These Oracle background processes are persistent processes that make up the instance and they will run until the instance is shutdown. Also note that these are processes, not programs. **There is only one Oracle program on the host, i.e. there is only one binary, named oracle.** It is just executed many times with different names!

Now my next step is to mount the database and open it.

```
> alter database mount;

Database altered.
```

When you mount a database, Oracle associates the started instance with the database. Oracle control files are opened and read. However no transactional verification such as rollback/recovery is carried out

Finally I go ahead and open the database.

```
> alter database open;

Database altered.
```

An open command opens data files, redo logs, it performs database consistency and auto recovery. At this stage, the database "mydb" is now ready to be used by all valid users.

I went through this long-winded approach to opening an Oracle database to show stages involved in bringing up and onlining an Oracle database. Otherwise you can just issue the command STARTUP to bring an Oracle database online as shown below:

```
oracle@linux:/home/oracle% sqlplus "/ as sysdba"

Connected to an idle instance.

idle> startup
ORACLE instance started.

Total System Global Area 1476395008 bytes
Fixed Size                   779196 bytes
Variable Size             435166276 bytes
Database Buffers         1040187392 bytes
Redo Buffers                 262144 bytes
Database mounted.
Database opened.
```

The Oracle Startup Process, a Summary

The steps that Oracle takes in starting up can be summarized as follows:

1. startup nomount

1.1. Reads the initialization file
1.2. Allocates the shared memory (SGA)
1.3. Starts the background processes
1.4. Opens the alert and trace files

2. alter database mount

2.1. Opens and reads the control files
2.2. Mounts the database and associates the started instance with the database

3. alter database open

3.1. Opens data files and redo logs
3.2. Performs database consistency and auto recovery