



9

Process Architecture

This chapter discusses the processes in an Oracle database system and the different configurations available for an Oracle system.

This chapter contains the following topics:

- [Introduction to Processes](#)
- [Overview of User Processes](#)
- [Overview of Oracle Processes](#)
- [Shared Server Architecture](#)
- [Dedicated Server Configuration](#)
- [The Program Interface](#)

Introduction to Processes

All connected Oracle users must run two modules of code to access an Oracle database instance.

- **Application or Oracle tool:** A database user runs a database application (such as a precompiler program) or an Oracle tool (such as SQL*Plus), which issues SQL statements to an Oracle database.
- **Oracle database server code:** Each user has some Oracle database code executing on his or her behalf, which interprets and processes the application's SQL statements.

These code modules are run by processes. A **process** is a "thread of control" or a mechanism in an operating system that can run a series of steps. (Some operating systems use the terms **job** or **task**.) A process normally has its own private memory area in which it runs.

Multiple-Process Oracle Systems

Multiple-process Oracle (also called **multiuser Oracle**) uses several processes to run different parts of the Oracle code and additional processes for the users—either one process for each connected user or one or more processes shared by multiple users. Most database systems are multiuser, because one of the primary benefits of a database is managing data needed by multiple users at the same time.

Each process in an Oracle instance performs a specific job. By dividing the work of Oracle and database applications into several processes, multiple users and applications can connect to a single database instance simultaneously while the system maintains excellent performance.

Types of Processes

The processes in an Oracle system can be categorized into two major groups:

- User processes run the application or Oracle tool code.
- Oracle processes run the Oracle database server code. They include server processes and background processes.

The process structure varies for different Oracle configurations, depending on the operating system and the choice of Oracle options. The code for connected users can be configured as a dedicated server or a shared server.

With dedicated server, for each user, the database application is run by a different process (a user process) than the one that runs the Oracle database server code (a dedicated server process).

With shared server, the database application is run by a different process (a user process) than the one that runs the Oracle database server code. Each server process that runs Oracle database server code (a **shared server process**) can serve multiple user processes.

[Figure 9-1](#) illustrates a dedicated server configuration. Each connected user has a separate user process, and several background processes run Oracle.

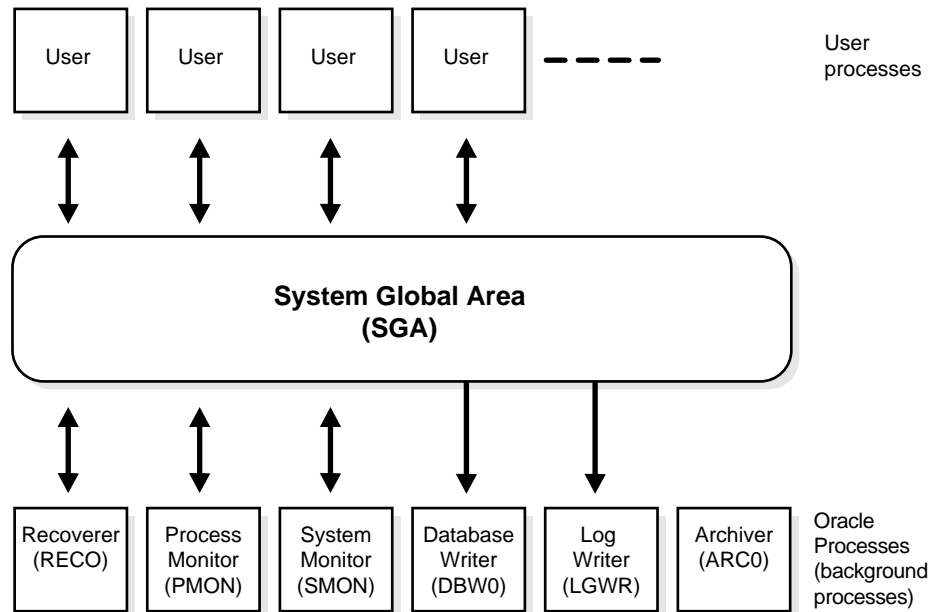
Figure 9–1 An Oracle Instance

Figure 9–1 can represent multiple concurrent users running an application on the same machine as Oracle. This particular configuration usually runs on a mainframe or minicomputer.

See Also:

- ["Overview of User Processes"](#) on page 9-4
- ["Overview of Oracle Processes"](#) on page 9-4
- ["Dedicated Server Configuration"](#) on page 9-22
- ["Shared Server Architecture"](#) on page 9-16
- Your Oracle operating system-specific documentation for more details on configuration choices

Overview of User Processes

When a user runs an application program (such as a Pro*C program) or an Oracle tool (such as Enterprise Manager or SQL*Plus), Oracle creates a **user process** to run the user's application.

Connections and Sessions

Connection and **session** are closely related to **user process** but are very different in meaning.

A **connection** is a communication pathway between a user process and an Oracle instance. A communication pathway is established using available interprocess communication mechanisms (on a computer that runs both the user process and Oracle) or network software (when different computers run the database application and Oracle, and communicate through a network).

A **session** is a specific connection of a user to an Oracle instance through a user process. For example, when a user starts SQL*Plus, the user must provide a valid user name and password, and then a session is established for that user. A session lasts from the time the user connects until the time the user disconnects or exits the database application.

Multiple sessions can be created and exist concurrently for a single Oracle user using the same user name. For example, a user with the user name/password of SCOTT/TIGER can connect to the same Oracle instance several times.

In configurations without the shared server, Oracle creates a server process on behalf of each user session. However, with the shared server, many user sessions can share a single server process.

See Also: ["Shared Server Architecture"](#) on page 9-16

Overview of Oracle Processes

This section describes the two types of processes that run the Oracle database server code (server processes and background processes). It also describes the trace files and alert file, which record database events for the Oracle processes.

Server Processes

Oracle creates **server processes** to handle the requests of user processes connected to the instance. In some situations when the application and Oracle operate on the same machine, it is possible to combine the user process and corresponding server

process into a single process to reduce system overhead. However, when the application and Oracle operate on different machines, a user process always communicates with Oracle through a separate server process.

Server processes (or the server portion of combined user/server processes) created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application
- Read necessary data blocks from datafiles on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA
- Return results in such a way that the application can process the information

Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle system uses some additional Oracle processes called **background processes**.

An Oracle instance can have many background processes; not all are always present. There are numerous background processes. The background processes in an Oracle instance can include the following:

- Database Writer Process (DBWn)
- Log Writer Process (LGWR)
- Checkpoint Process (CKPT)
- System Monitor Process (SMON)
- Process Monitor Process (PMON)
- Recoverer Process (RECO)
- Job Queue Processes
- Archiver Processes (ARCn)
- Queue Monitor Processes (QMNn)
- Other Background Processes

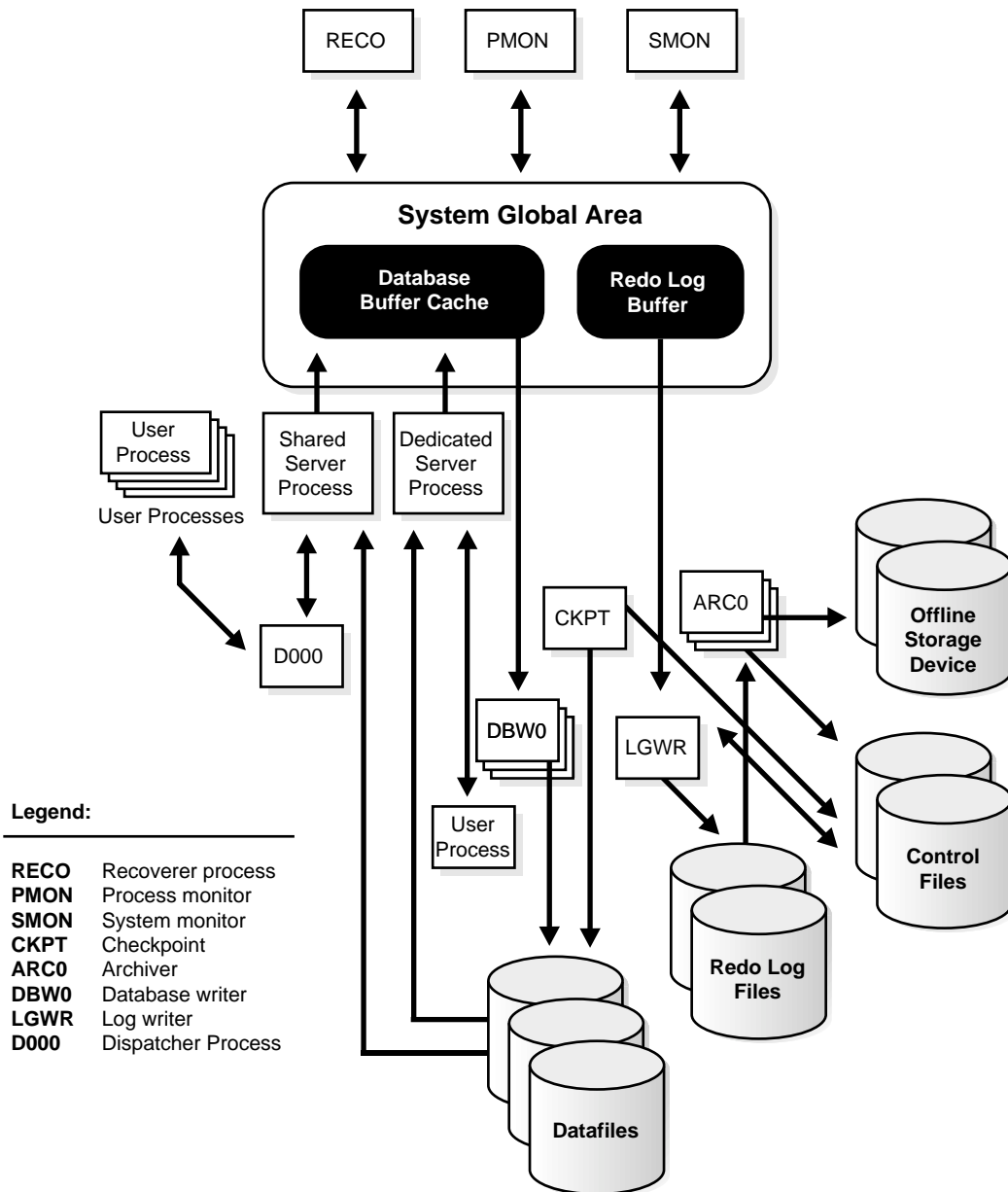
On many operating systems, background processes are created automatically when an instance is started.

Figure 9-2 illustrates how each background process interacts with the different parts of an Oracle database, and the rest of this section describes each process.

See Also:

- *Oracle Real Application Clusters Administrator's Guide* for more information. Real Application Clusters are not illustrated in [Figure 9-2](#)
- Your operating system-specific documentation for details on how these processes are created

Figure 9-2 Background Processes of a Multiple-Process Oracle Instance



Database Writer Process (DBWn)

The **database writer process (DBWn)** writes the contents of buffers to datafiles. The DBWn processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9 and DBW_a through DBW_j) to improve write performance if your system modifies data heavily. These additional DBWn processes are not useful on uniprocessor systems.

When a buffer in the database buffer cache is modified, it is marked **dirty**. A **cold** buffer is a buffer that has not been recently used according to the least recently used (LRU) algorithm. The DBWn process writes cold, dirty buffers to disk so that user processes are able to find cold, clean buffers that can be used to read new blocks into the cache. As buffers are dirtied by user processes, the number of free buffers diminishes. If the number of free buffers drops too low, user processes that must read blocks from disk into the cache are not able to find free buffers. DBWn manages the buffer cache so that user processes can always find free buffers.

By writing cold, dirty buffers to disk, DBWn improves the performance of finding free buffers while keeping recently used buffers resident in memory. For example, blocks that are part of frequently accessed small tables or indexes are kept in the cache so that they do not need to be read in again from disk. The LRU algorithm keeps more frequently accessed blocks in the buffer cache so that when a buffer is written to disk, it is unlikely to contain data that will be useful soon.

The initialization parameter `DB_WRITER_PROCESSES` specifies the number of DBWn processes. The maximum number of DBWn processes is 20. If it is not specified by the user during startup, Oracle determines how to set `DB_WRITER_PROCESSES` based on the number of CPUs and processor groups.

The DBWn process writes dirty buffers to disk under the following conditions:

- When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers, it signals DBWn to write. DBWn writes dirty buffers to disk asynchronously while performing other processing.
- DBWn periodically writes buffers to advance the **checkpoint**, which is the position in the redo thread (log) from which instance recovery begins. This log position is determined by the oldest dirty buffer in the buffer cache.

In all cases, DBWn performs batched (multiblock) writes to improve efficiency. The number of blocks written in a multiblock write varies by operating system.

See Also:

- ["Database Buffer Cache"](#) on page 8-9
- *Oracle Database Performance Tuning Guide* for advice on setting `DB_WRITER_PROCESSES` and for information about how to monitor and tune the performance of a single `DBW0` process or multiple `DBWn` processes
- *Oracle Database Backup and Recovery Basics*

Log Writer Process (LGWR)

The **log writer process (LGWR)** is responsible for redo log buffer management—writing the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

LGWR writes one contiguous portion of the buffer to disk. LGWR writes:

- A commit record when a user process commits a transaction
- Redo log buffers
 - Every three seconds
 - When the redo log buffer is one-third full
 - When a `DBWn` process writes modified buffers to disk, if necessary

Note: Before `DBWn` can write a modified buffer, all redo records associated with the changes to the buffer must be written to disk (the **write-ahead protocol**). If `DBWn` finds that some redo records have not been written, it signals LGWR to write the redo records to disk and waits for LGWR to complete writing the redo log buffer before it can write out the data buffers.

LGWR writes synchronously to the active mirrored group of redo log files. If one of the files in the group is damaged or unavailable, LGWR continues writing to other files in the group and logs an error in the LGWR trace file and in the system alert

file. If all files in a group are damaged, or the group is unavailable because it has not been archived, LGWR cannot continue to function.

When a user issues a `COMMIT` statement, LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries. The corresponding changes to data blocks are deferred until it is more efficient to write them. This is called a **fast commit** mechanism. The atomic write of the redo entry containing the transaction's commit record is the single event that determines the transaction has committed. Oracle returns a success code to the committing transaction, although the data buffers have not yet been written to disk.

Note: Sometimes, if more buffer space is needed, LGWR writes redo log entries before a transaction is committed. These entries become permanent only if the transaction is later committed.

When a user commits a transaction, the transaction is assigned a **system change number (SCN)**, which Oracle records along with the transaction's redo entries in the redo log. SCNs are recorded in the redo log so that recovery operations can be synchronized in Real Application Clusters and distributed databases.

In times of high activity, LGWR can write to the redo log file using **group commits**. For example, assume that a user commits a transaction. LGWR must write the transaction's redo entries to disk, and as this happens, other users issue `COMMIT` statements. However, LGWR cannot write to the redo log file to commit these transactions until it has completed its previous write operation. After the first transaction's entries are written to the redo log file, the entire list of redo entries of waiting transactions (not yet committed) can be written to disk in one operation, requiring less I/O than do transaction entries handled individually. Therefore, Oracle minimizes disk I/O and maximizes performance of LGWR. If requests to commit continue at a high rate, then every write (by LGWR) from the redo log buffer can contain multiple commit records.

See Also:

- [Redo Log Buffer](#) on page 8-13
- ["Trace Files and the Alert Log"](#) on page 9-15
- *Oracle Real Application Clusters Deployment and Performance Guide* for more information about SCNs and how they are used
- *Oracle Database Administrator's Guide* for more information about SCNs and how they are used
- *Oracle Database Performance Tuning Guide* for information about how to monitor and tune the performance of LGWR

Checkpoint Process (CKPT)

When a checkpoint occurs, Oracle must update the headers of all datafiles to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBWn always performs that work.

The statistic **DBWR checkpoints** displayed by the `System_Statistics` monitor in Enterprise Manager indicates the number of checkpoint requests completed.

See Also: *Oracle Real Application Clusters Administrator's Guide* for information about CKPT with Real Application Clusters

System Monitor Process (SMON)

The **system monitor process (SMON)** performs recovery, if necessary, at instance startup. SMON is also responsible for cleaning up temporary segments that are no longer in use and for coalescing contiguous free extents within dictionary managed tablespaces. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online. SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

With Real Application Clusters, the SMON process of one instance can perform instance recovery for a failed CPU or instance.

See Also: *Oracle Real Application Clusters Administrator's Guide* for more information about SMON

Process Monitor Process (PMON)

The **process monitor (PMON)** performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.

PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener.

Like SMON, PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

Recoverer Process (RECO)

The **recoverer process (RECO)** is a background process used with the distributed database configuration that automatically resolves failures involving distributed transactions. The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction. When the RECO process reestablishes a connection between involved database servers, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved in-doubt transactions.

If the RECO process fails to connect with a remote server, RECO automatically tries to connect again after a timed interval. However, RECO waits an increasing amount of time (growing exponentially) before it attempts another connection. The RECO process is present only if the instance permits distributed transactions. The number of concurrent distributed transactions is not limited.

See Also: *Oracle Database Administrator's Guide* for more information about distributed transaction recovery

Job Queue Processes

Job queue processes are used for batch processing. They run user jobs. They can be viewed as a scheduler service that can be used to schedule jobs as PL/SQL statements or procedures on an Oracle instance. Given a start date and an interval, the job queue processes try to run the job at the next occurrence of the interval.

Job queue processes are managed dynamically. This allows job queue clients to use more job queue processes when required. The resources used by the new processes are released when they are idle.

Dynamic job queue processes can run a large number of jobs concurrently at a given interval. The job queue processes run user jobs as they are assigned by the CJQ process. Here's what happens:

1. The coordinator process, named CJQ0, periodically selects jobs that need to be run from the system `JOB$` table. New jobs selected are ordered by time.
2. The CJQ0 process dynamically spawns job queue slave processes (J000...J999) to run the jobs.
3. The job queue process runs one of the jobs that was selected by the CJQ process for execution. The processes run one job at a time.
4. After the process finishes execution of a single job, it polls for more jobs. If no jobs are scheduled for execution, then it enters a sleep state, from which it wakes up at periodic intervals and polls for more jobs. If the process does not find any new jobs, then it aborts after a preset interval.

The initialization parameter `JOB_QUEUE_PROCESSES` represents the maximum number of job queue processes that can concurrently run on an instance. However, clients should not assume that all job queue processes are available for job execution.

Note: The coordinator process is not started if the initialization parameter `JOB_QUEUE_PROCESSES` is set to 0.

See Also: *Oracle Database Administrator's Guide* for more information about job queues

Archiver Processes (ARCn)

The **archiver process (ARCn)** copies redo log files to a designated storage device after a log switch has occurred. ARCn processes are present only when the database is in ARCHIVELOG mode, and automatic archiving is enabled.

An Oracle instance can have up to 10 ARCn processes (ARC0 to ARC9). The LGWR process starts a new ARCn process whenever the current number of ARCn processes is insufficient to handle the workload. The alert file keeps a record of when LGWR starts a new ARCn process.

If you anticipate a heavy workload for archiving, such as during bulk loading of data, you can specify multiple archiver processes with the initialization parameter `LOG_ARCHIVE_MAX_PROCESSES`. The `ALTER SYSTEM` statement can change the value of this parameter dynamically to increase or decrease the number of ARCn

processes. However, you do not need to change this parameter from its default value of 1, because the system determines how many ARC*n* processes are needed, and LGWR automatically starts up more ARC*n* processes when the database workload requires more.

See Also:

- ["Trace Files and the Alert Log"](#) on page 9-15
- *Oracle Database Backup and Recovery Basics*
- Your operating system-specific documentation for details about using the ARC*n* processes

Queue Monitor Processes (QMN*n*)

The **queue monitor process** is an optional background process for Oracle Streams Advanced Queuing, which monitors the message queues. You can configure up to 10 queue monitor processes. These processes, like the job queue processes, are different from other Oracle background processes in that process failure does not cause the instance to fail.

See Also: *Oracle Streams Advanced Queuing User's Guide and Reference*

Other Background Processes

There are several other background processes that might be running. These can include the following:

MMON performs various manageability-related background tasks, for example:

- Issuing alerts whenever a given metrics violates its threshold value
- Taking snapshots by spawning additional process (MMON slaves)
- Capturing statistics value for SQL objects which have been recently modified

MMNL performs frequent and light-weight manageability-related tasks, such as session history capture and metrics computation.

MMAN is used for internal database tasks.

RBAL coordinates rebalance activity for disk groups in an Automatic Storage Management instance. It performs a global open on Automatic Storage Management disks.

ORBn performs the actual rebalance data extent movements in an Automatic Storage Management instance. There can be many of these at a time, called ORB0, ORB1, and so forth.

OSMB is present in a database instance using an Automatic Storage Management disk group. It communicates with the Automatic Storage Management instance.

Trace Files and the Alert Log

Each server and background process can write to an associated **trace file**. When a process detects an internal error, it dumps information about the error to its trace file. If an internal error occurs and information is written to a trace file, the administrator should contact Oracle Support Services.

All filenames of trace files associated with a background process contain the name of the process that generated the trace file. The one exception to this is trace files generated by job queue processes (Jnnn).

Additional information in trace files can provide guidance for tuning applications or an instance. Background processes always write this information to a trace file when appropriate.

Each database also has an `alert.log`. The alert file of a database is a chronological log of messages and errors, including the following:

- All internal errors (ORA-600), block corruption errors (ORA-1578), and deadlock errors (ORA-60) that occur
- Administrative operations, such as the SQL statements `CREATE/ALTER/DROP DATABASE/TABLESPACE` and the Enterprise Manager or SQL*Plus statements `STARTUP, SHUTDOWN, ARCHIVE LOG, and RECOVER`
- Several messages and errors relating to the functions of shared server and dispatcher processes
- Errors during the automatic refresh of a materialized view

Oracle uses the alert file to keep a record of these events as an alternative to displaying the information on an operator's console. (Many systems also display this information on the console.) If an administrative operation is successful, a message is written in the alert file as "completed" along with a time stamp.

See Also:

- *Oracle Database Performance Tuning Guide* for information about enabling the SQL trace facility
- *Oracle Database Error Messages* for information about error messages

Shared Server Architecture

Shared server architecture eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. An idle shared server process from a shared pool of server processes picks up a request from a common queue, which means a small number of shared servers can perform the same amount of processing as many dedicated servers. Also, because the amount of memory required for each user is relatively small, less memory and process management are required, and more users can be supported.

A number of different processes are needed in a shared server system:

- A network listener process that connects the user processes to dispatchers or dedicated servers (the listener process is part of Oracle Net Services, not Oracle).
- One or more dispatcher processes
- One or more shared server processes

Shared server processes require Oracle Net Services or SQL*Net version 2.

Note: To use shared servers, a user process must connect through Oracle Net Services or SQL*Net version 2, even if the process runs on the same machine as the Oracle instance.

When an instance starts, the network listener process opens and establishes a communication pathway through which users connect to Oracle. Then, each dispatcher process gives the listener process an address at which the dispatcher listens for connection requests. At least one dispatcher process must be configured and started for each network protocol that the database clients will use.

When a user process makes a connection request, the listener examines the request and determines whether the user process can use a shared server process. If so, the

listener returns the address of the dispatcher process that has the lightest load, and the user process connects to the dispatcher directly.

Some user processes cannot communicate with the dispatcher, so the network listener process cannot connect them to a dispatcher. In this case, or if the user process requests a dedicated server, the listener creates a dedicated server and establishes an appropriate connection.

Oracle's shared server architecture increases the scalability of applications and the number of clients simultaneously connected to the database. It can enable existing applications to scale up without making any changes to the application itself.

See Also:

- ["Restricted Operations of the Shared Server"](#) on page 9-21
- *Oracle Net Services Administrator's Guide* for more information about the network listener

Dispatcher Request and Response Queues

A request from a user is a single program interface call that is part of the user's SQL statement. When a user makes a call, its dispatcher places the request on the **request queue**, where it is picked up by the next available shared server process.

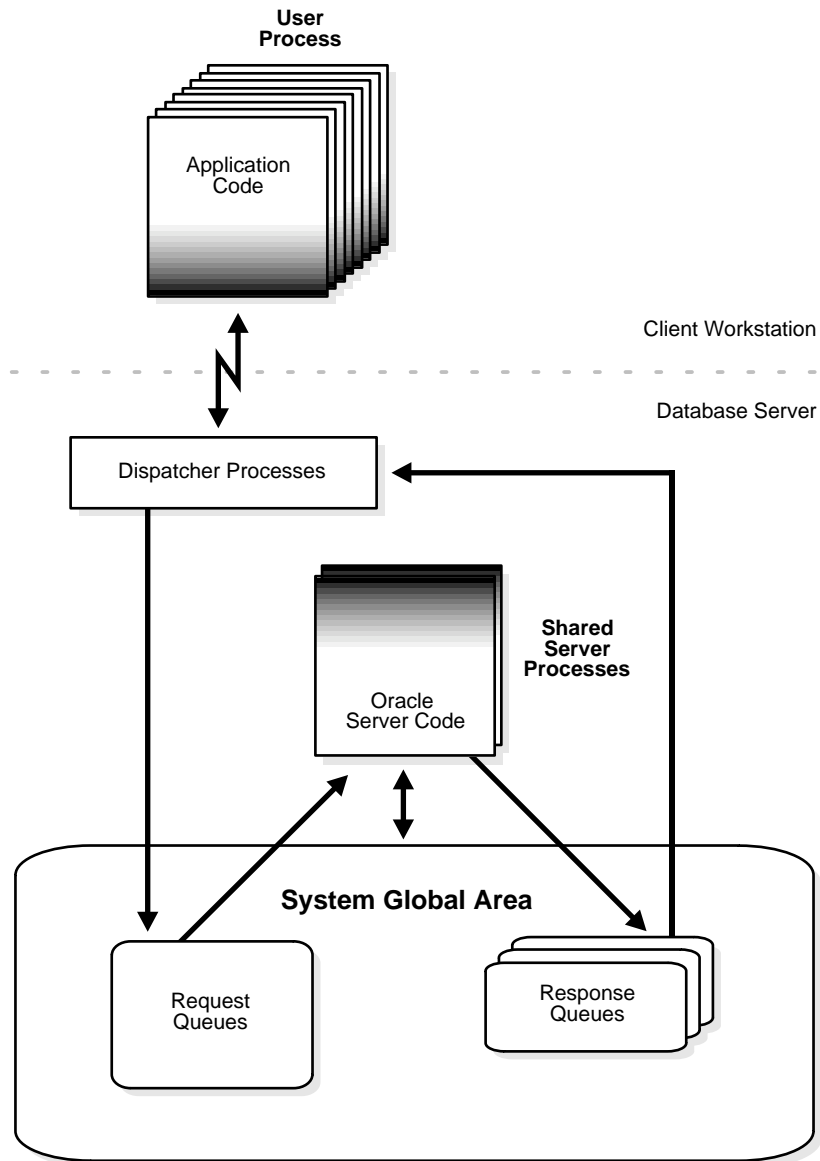
The request queue is in the SGA and is common to all dispatcher processes of an instance. The shared server processes check the common request queue for new requests, picking up new requests on a first-in-first-out basis. One shared server process picks up one request in the queue and makes all necessary calls to the database to complete that request.

When the server completes the request, it places the response on the calling dispatcher's **response queue**. Each dispatcher has its own response queue in the SGA. The dispatcher then returns the completed request to the appropriate user process.

For example, in an order entry system each clerk's user process connects to a dispatcher and each request made by the clerk is sent to that dispatcher, which places the request in the request queue. The next available shared server process picks up the request, services it, and puts the response in the response queue. When a clerk's request is completed, the clerk remains connected to the dispatcher, but the shared server process that processed the request is released and available for other requests. While one clerk is talking to a customer, another clerk can use the same shared server process.

Figure 9-3 illustrates how user processes communicate with the dispatcher across the program interface and how the dispatcher communicates users' requests to shared server processes.

Figure 9-3 The Shared Server Configuration and Processes



Dispatcher Processes (Dnnn)

The **dispatcher processes** support shared server configuration by allowing user processes to share a limited number of server processes. With the shared server, fewer shared server processes are required for the same number of users. Therefore, the shared server can support a greater number of users, particularly in client/server environments where the client application and server operate on different machines.

You can create multiple dispatcher processes for a single database instance. At least one dispatcher must be created for each network protocol used with Oracle. The database administrator starts an optimal number of dispatcher processes depending on the operating system limitation and the number of connections for each process, and can add and remove dispatcher processes while the instance runs.

Note: Each user process that connects to a dispatcher must do so through Oracle Net Services or SQL*Net version 2, even if both processes are running on the same machine.

In a shared server configuration, a network listener process waits for connection requests from client applications and routes each to a dispatcher process. If it cannot connect a client application to a dispatcher, the listener process starts a dedicated server process, and connects the client application to the dedicated server. The listener process is not part of an Oracle instance; rather, it is part of the networking processes that work with Oracle.

See Also:

- ["Shared Server Architecture"](#) on page 9-16
- *Oracle Net Services Administrator's Guide* for more information about the network listener

Shared Server Processes (Snnn)

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. Instead, a shared server process serves any client request in the shared server configuration.

The PGA of a shared server process does not contain user-related data (which needs to be accessible to all shared server processes). The PGA of a shared server process contains only stack space and process-specific variables.

All session-related information is contained in the SGA. Each shared server process needs to be able to access all sessions' data spaces so that any server can handle requests from any session. Space is allocated in the SGA for each session's data space. You can limit the amount of space that a session can allocate by setting the resource limit `PRIVATE_SGA` to the desired amount of space in the user's profile.

Oracle dynamically adjusts the number of shared server processes based on the length of the request queue. The number of shared server processes that can be created ranges between the values of the initialization parameters `SHARED_SERVERS` and `MAX_SHARED_SERVERS`.

See Also:

- ["Overview of the Program Global Areas"](#) on page 8-19 for more information about the content of a PGA in different types of instance configurations
- [Chapter 20, "Database Security"](#) for more information about resource limits and profiles

Restricted Operations of the Shared Server

Certain administrative activities cannot be performed while connected to a dispatcher process, including shutting down or starting an instance and media recovery. An error message is issued if you attempt to perform these activities while connected to a dispatcher process.

These activities are typically performed when connected with administrator privileges. When you want to connect with administrator privileges in a system configured with shared servers, you must state in your connect string that you want to use a dedicated server process (`SERVER=DEDICATED`) instead of a dispatcher process.

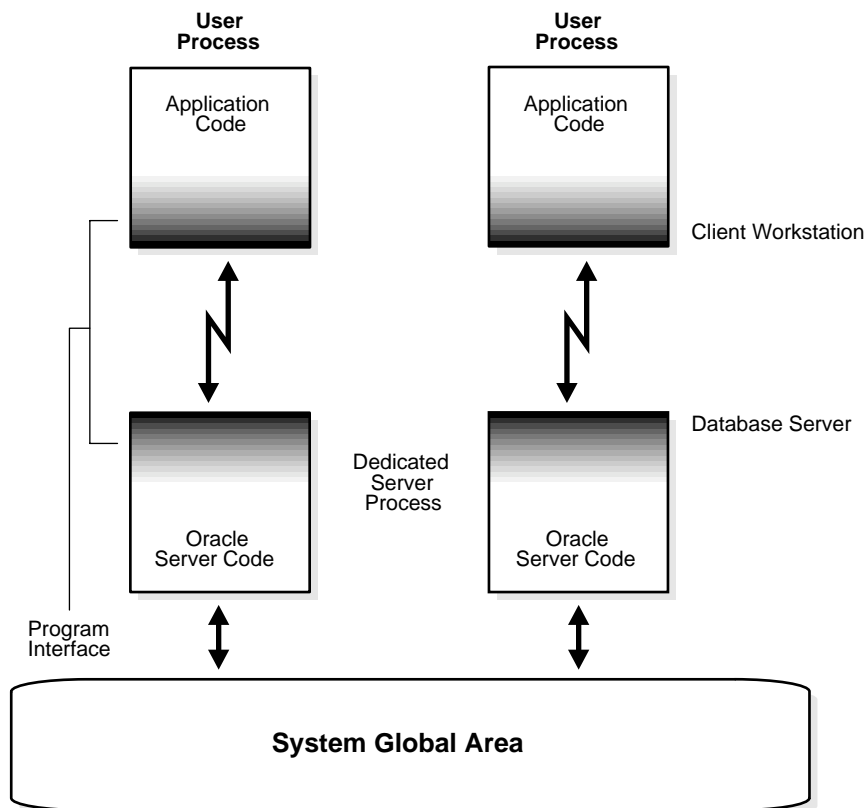
See Also:

- Your operating system-specific documentation
- *Oracle Net Services Administrator's Guide* for the proper connect string syntax

Dedicated Server Configuration

Figure 9-4 illustrates Oracle running on two computers using the dedicated server architecture. In this configuration, a user process runs the database application on one machine, and a server process runs the associated Oracle database server on another machine.

Figure 9-4 Oracle Using Dedicated Server Processes



The user and server processes are separate, distinct processes. The separate server process created on behalf of each user process is called a **dedicated server process** (or **shadow process**), because this server process acts only on behalf of the associated user process.

This configuration maintains a one-to-one ratio between the number of user processes and server processes. Even when the user is not actively making a database request, the dedicated server process remains (though it is inactive and can be paged out on some operating systems).

Figure 9-4 shows user and server processes running on separate computers connected across a network. However, the dedicated server architecture is also used if the same computer runs both the client application and the Oracle database server code but the host operating system could not maintain the separation of the two programs if they were run in a single process. UNIX is a common example of such an operating system.

In the dedicated server configuration, the user and server processes communicate using different mechanisms:

- If the system is configured so that the user process and the dedicated server process run on the same computer, the program interface uses the host operating system's interprocess communication mechanism to perform its job.
- If the user process and the dedicated server process run on different computers, the program interface provides the communication mechanisms (such as the network software and Oracle Net Services) between the programs.
- Dedicated server architecture can sometimes result in inefficiency. Consider an order entry system with dedicated server processes. A customer places an order as a clerk enters the order into the database. For most of the transaction, the clerk is talking to the customer while the server process dedicated to the clerk's user process remains idle. The server process is not needed during most of the transaction, and the system is slower for other clerks entering orders. For applications such as this, the shared server architecture may be preferable.

See Also:

- Your operating system-specific documentation
- *Oracle Net Services Administrator's Guide*

for more information about communication links

The Program Interface

The **program interface** is the software layer between a database application and Oracle. The program interface:

- Provides a security barrier, preventing destructive access to the SGA by client user processes

- Acts as a communication mechanism, formatting information requests, passing data, and trapping and returning errors
- Converts and translates data, particularly between different types of computers or to external user program datatypes

The **Oracle code** acts as a server, performing database tasks on behalf of an **application** (a client), such as fetching rows from data blocks. It consists of several parts, provided by both Oracle software and operating system-specific software.

Program Interface Structure

The program interface consists of the following pieces:

- Oracle call interface (OCI) or the Oracle runtime library (SQLLIB)
- The client or user side of the program interface (also called the **UPI**)
- Various **Oracle Net Services drivers** (protocol-specific communications software)
- Operating system communications software
- The server or Oracle side of the program interface (also called the **OPI**)

Both the user and Oracle sides of the program interface run Oracle software, as do the drivers.

Oracle Net Services is the portion of the program interface that allows the client application program and the Oracle database server to reside on separate computers in your communication network.

Program Interface Drivers

Drivers are pieces of software that transport data, usually across a network. They perform operations such as connect, disconnect, signal errors, and test for errors. Drivers are specific to a communications protocol, and there is always a default driver.

You can install multiple drivers (such as the asynchronous or DECnet drivers) and select one as the default driver, but allow an individual user to use other drivers by specifying the desired driver at the time of connection. Different processes can use different drivers. A single process can have concurrent connections to a single database or to multiple databases (either local or remote) using different Oracle Net Services drivers.

See Also:

- Your system installation and configuration guide for details about choosing, installing, and adding drivers
- Your system Oracle Net Services documentation for information about selecting a driver at runtime while accessing Oracle
- *Oracle Net Services Administrator's Guide*

Communications Software for the Operating System

The lowest-level software connecting the user side to the Oracle side of the program interface is the communications software, which is provided by the host operating system. DECnet, TCP/IP, LU6.2, and ASYNC are examples. The communication software can be supplied by Oracle, but it is usually purchased separately from the hardware vendor or a third-party software supplier.

See Also: Your Oracle operating system-specific documentation for more information about the communication software of your system

