ГЛАВА 5

Процессы Oracle

Мы подошли к рассмотрению последнего фрагмента архитектуры. Вы уже ознакомились с базой данных и набором образующих ее физических файлов. Память, используемая экземпляром Oracle, составляет одну его половину. Вторую половину архитектуры экземпляра образует набор *процессов*. Именно им посвящена настоящая глава.

Каждый процесс Oracle будет выполнять конкретную задачу или набор задач, и каждому из них для выполнения его задачи будет выделена внутренняя память (память PGA). В экземпляре Oracle существуют три обширных класса процессов.

- *Серверные процессы*. Эти процессы выполняют свои задачи в зависимости от запроса клиента. Мы уже до определенной степени ознакомились с выделенным и разделяемым сервером. Они относятся к серверным процессам.
- *Фоновые процессы*. Эти процессы запускаются при запуске базы данных и выполняют различные служебные задачи, такие как запись блоков на диски, поддержание оперативных журналов повторения, очистка после прерванных процессов, обслуживание AWR (Automatic Workload Repository — автоматический репозиторий рабочей нагрузки) и т.п.
- *Подчиненные процессы*. Эти процессы аналогичны фоновым, но выполняют дополнительную работу от имени либо фонового, либо серверного процесса.

Мы уже упоминали некоторые из этих процессов, такие как процесс записи блоков базы данных (`DBWn`) и процесс записи журналов (`LGWR`), а в этой главе функционирование каждого процесса и выполняемые им действия рассматриваются более подробно.

На заметку! Термин *процесс* в этой главе должен трактоваться как синоним термина *поток* в операционных системах, в средах которых СУБД Oracle реализована с помощью потоков (например, Windows). В контексте этой главы термин *процесс* покрывает как процессы, так и потоки. При работе с многозадачной реализацией Oracle, используемой в среде UNIX, термин *процесс* полностью отражает суть. В однозадачной реализации Oracle, применяемой в среде Windows, термин *процесс* в действительности означает *поток внутри процесса Oracle*. Так, например, когда речь идет о процессе `DBWn`, в среде Windows подразумевается поток `DBWn` внутри процесса Oracle.

Серверные процессы

Серверные процессы — это процессы, которые выполняют действия от имени сеанса клиента. Эти процессы обязательно принимают и реагируют на SQL-операторы, которые наши приложения отправляют базе данных.

В главе 2 мы кратко рассмотрели два типа подключений к Oracle.

- *Подключение посредством выделенного сервера*, при котором на сервере для подключения создается выделенный процесс. При этом между подключением к базе данных и серверным процессом или потоком существует однозначное соответствие.
- *Подключение посредством разделяемого сервера*, при котором множество сеансов используют общий пул серверных процессов, созданный и управляемый экземпляром Oracle. Подключение производится к диспетчеру базы данных, а не к процессу выделенного сервера, созданному специально для данного подключения.

На заметку! Важно понимать различие между подключением и сеансом в контексте терминологии Oracle. *Подключение* (connection) — это всего лишь физический путь передачи информации между процессом клиента и экземпляром Oracle (например, сетевое подключение пользователя к экземпляру). С другой стороны, *сеанс* (session) — это логический объект в базе данных, в котором процесс клиента может выполнять SQL-запросы и прочие задачи. С одним подключением может быть связано множество независимых сеансов, и эти сеансы могут существовать независимо от подключения. Вскоре это будет рассмотрено подробно.

Процессы как выделенного, так и разделяемого сервера решают одну и ту же задачу: они обрабатывают все передаваемые им SQL-запросы. При передаче базе данных запроса `SELECT * FROM EMP` процесс выделенного/разделяемого сервера осуществляет синтаксический разбор запроса и помещает его в разделяемый пул (или находит его в разделяемом пуле, если он уже там присутствует). При необходимости процесс создает план запроса и выполняет его, возможно отыскивая необходимые данные в буферном кэше или считывая данные с диска в буферный кэш.

Серверные процессы проделывают всю "черную" работу. Во многих случаях именно они являются в системе основными потребителями ресурсов процессора системы, поскольку они выполняют сортировку, суммирование и соединение — практически, все необходимые действия.

Подключения посредством выделенного сервера

В режиме выделенного сервера между подключением клиента и серверным процессом (или потоком в соответствующей ситуации) устанавливается соответствие "один к одному". При наличии на компьютере UNIX 100 подключений посредством выделенного сервера от их имени будут выполняться 100 процессов. Эта ситуация графически представлена на рис. 5.1.

Клиентское приложение будет содержать скомпонованные в него библиотеки Oracle. Эти библиотеки предоставляют API-интерфейсы, которые требуются для обмена информацией с базой данных. Такие API-интерфейсы знают, как отправлять запрос базе данных и обрабатывать возвращаемый курсор. Они также умеют объединять запросы в сетевые вызовы, которые выделенный сервер будет распаковывать. Эта часть программного обеспечения называется *Oracle Net* (Сеть Oracle), хотя в предшествующих версиях ее называли *SQL*Net* или *Net8*. Она представляет собой сетевое программное обеспечение/протокол, который СУБД Oracle использует для реализации модели "клиент-сервер" (даже многозвенная архитектура содержит программное обеспечение модели "клиент-сервер").

СУБД Oracle использует эту же архитектуру, даже если формально Oracle Net не принимает участия. То есть, эта двухпроцессная архитектура (которую также называют *двухзадачной*) применяется даже в ситуации, когда клиент и сервер установлены на одном и том же компьютере.

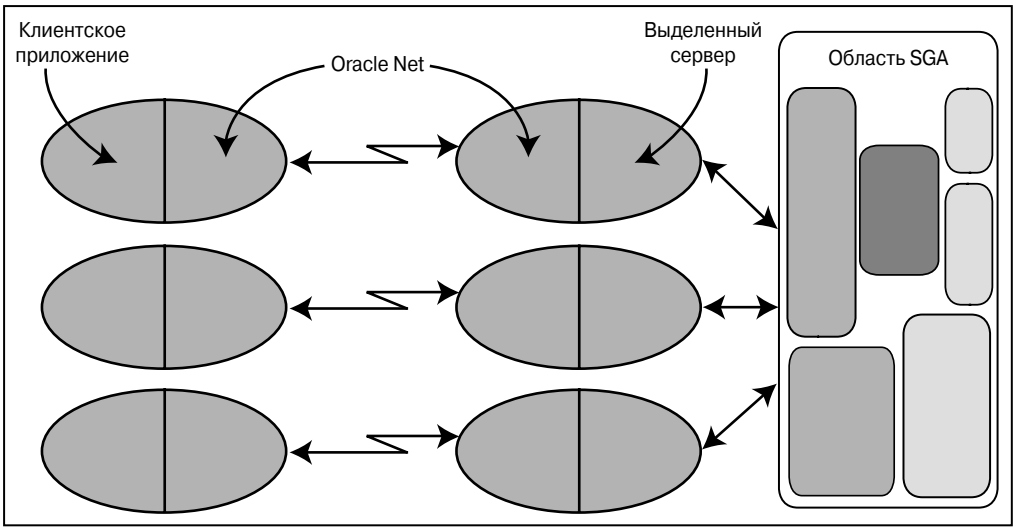

Рис. 5.1. Типичное подключение посредством выделенного сервера

Она предоставляет два преимущества.

- *Возможность удаленного выполнения.* Ситуация, когда клиентское приложение выполняется на другом компьютере, не том, где находится сама база данных, весьма естественна.
- *Обеспечение изоляции адресного пространства.* Серверный процесс имеет доступ по чтению и записи к области SGA. Если бы клиентский и серверный процессы были физически соединены между собой, ошибочный указатель в клиентском процессе легко мог бы повредить структуры данных в области SGA.

В главе 2 было показано, как эти выделенные серверы *порождаются*, или создаются, прослушивающим процессом Oracle. Все это здесь повторяться не будет. Вместо этого мы кратко рассмотрим, что происходит в тех случаях, когда прослушивающий процесс не задействован. Механизм во многом аналогичен использованному при наличии прослушивающего процесса, но теперь, вместо создания выделенного сервера прослушивающим процессом с помощью функции `fork()/exec()` в среде UNIX или с помощью функции взаимодействия между процессами (interprocess communication — IPC) в среде Windows, клиентский процесс создает его самостоятельно.

На заметку! Существует много разновидностей функций `fork()` и `exec()`, такие как `vfork()` и `execve()` и т.д. СУБД Oracle может использовать ту или иную функцию в зависимости от конкретной операционной системы и реализации, но сущность процесса при этом остается неизменной. Функция `fork()` создает новый процесс, являющийся клоном родительского процесса, и в среде UNIX этот способ создания нового процесса — единственно доступный. Функция `exec()` загружает в память образ новой программы поверх образа существующей программы, тем самым запуская новую программу. Таким образом, интерфейс SQL*Plus может *ветвиться* (скопировать самого себя), а затем *запустить* двоичный файл Oracle, перекрывая свою копию этой новой программой.

Создание этого родительского/дочернего процесса можно отчетливо наблюдать на компьютере UNIX при запуске клиента и сервера на одном компьютере:

```
$ sqlplus /

SQL*Plus: Release 11.2.0.1.0 Production on Sun Jan 24 07:28:13 2010

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

ops$tkyte%ORA11GR2> select a.spid dedicated_server,
  2 b.process clientpid
  3 from v$process a, v$session b
  4 where a.addr = b.paddr
  5 and b.sid = (select sid from v$mystat where rownum=1)
  6 /

DEDICATED_SERVER         CLIENTPID
------------------------ ------------------------
19168                    19167

ops$tkyte%ORA11GR2>
ops$tkyte%ORA11GR2> !/bin/ps -fp 19168 19167
UID        PID   PPID  C STIME  TTY   STAT TIME CMD
tkyte     19167  19166  0 07:30  pts/2 Ss+  0:00 /home/ora11gr2/app/.../bin/sqlplus
ora11gr2  19168  19167  0 07:30  ?     Ss   0:00 oracleorcl (DESCRIPTION=(LOCAL=...
```

Здесь с помощью запроса выясняется идентификатор процесса (PID), связанный с данным выделенным сервером (идентификатор SPID, полученный из представления `V$PROCESS` — это PID процесса операционной системы, который применялся во время выполнения этого запроса). Вывод `/bin/ps -fp` включает идентификатор родительского процесса (PPID) и показывает процесс выделенного сервера, 19168, как дочерний процесс процесса SQL*Plus: идентификатор процесса 19167.

Подключения посредством разделяемого сервера

Теперь подробнее рассмотрим процесс разделяемого сервера. Этот тип подключения требует использования Oracle Net, даже если клиент и сервер установлены на одном и том же компьютере — применение разделяемого сервера без прослушивающего процесса Oracle TNS невозможно. Как было сказано ранее, клиентское приложение будет подключаться к прослушивающему процессу Oracle TNS, а затем переадресовываться или передаваться диспетчеру. Диспетчер действует в качестве шлюза между клиентским приложением и процессом разделяемого сервера. Диаграмма архитектуры подключения к базе данных посредством разделяемого сервера приведена на рис. 5.2.

Как видите, клиентские приложения со скомпонованными в них библиотеками Oracle будут физически подключены к процессу диспетчера. Для любого конкретного экземпляра может быть сконфигурировано множество диспетчеров, но весьма часто только один диспетчер используют для сотен — и даже тысяч — пользователей. Диспетчер отвечает лишь за прием входящих запросов от клиентских приложений и помещение их в очередь запросов в области SGA. Первый доступный процесс разделяемого сервера, который в целом не отличается от процесса выделенного сервера, выберет запрос из очереди и присоединит область UGA соответствующего сеанса (на рис. 5.2 эти области изображены прямоугольниками с меткой “C”). Разделяемый сервер обработает запрос и поместит любой полученный в результате этого вывод в очередь ответов. Диспетчер постоянно отслеживает очередь ответов на предмет наличия в ней результатов и передает их обратно клиентскому приложению. На уровне клиента невозможно определить, подключен он посредством выделенного или разделяемого сервера — для клиента оба

типа подключения выглядят одинаково. Различия между ними проявляются только на уровне базы данных.

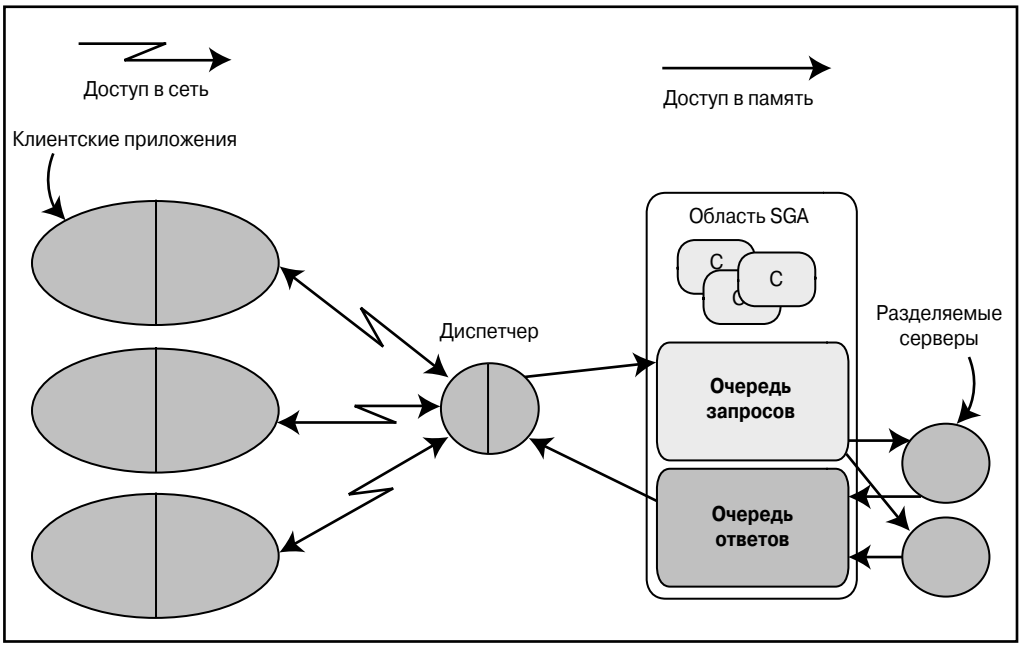

Рис. 5.2. Типичное подключение посредством разделяемого сервера

Резидентный пул соединений с базой данных (DRCP)

Резидентный пул соединений с базой данных (Database Resident Connection Pooling — DRCP) представляет собой дополнительный новый метод подключения к базе данных и установки сеанса. Он спроектирован как более эффективный метод организации пула соединений для интерфейсов приложений, которые не имеют встроенной поддержки организации пула соединений, таких как PHP — язык веб-сценариев общего назначения. DRCP — это смесь концепций выделенного и разделяемого сервера. Он наследует от разделяемого сервера концепцию организации пула серверных процессов, только процессы в пуле будут выделенными, а не разделяемыми серверами; от выделенного сервера она наследует концепцию собственно выделения.

При подключении к разделяемому серверу процесс разделяемого сервера совместно используется множеством сеансов, и один сеанс имеет тенденцию использовать множество выделенных серверов. В случае DRCP это не так; процесс выделенного сервера, выбранный из пула, становится выделенным для клиентского процесса на время жизни его сеанса. Если выполнить три оператора в отношении базы данных в сеансе на разделяемом сервере, высока вероятность, что эти три оператора будут выполнены тремя разными процессами разделяемого сервера. В случае использования DRCP те же самые три оператора будут выполнены выделенным сервером, назначенным из пула — и этот выделенный сервер будет находиться в распоряжении до тех пор, пока сеанс не вернет его обратно в пул. Таким образом, DRCP обладает средствами организации пула разделяемого сервера и характеристиками производительности выделенного сервера. Ниже мы еще сравним производительность выделенного сервера с производительностью разделяемого сервера.

Подключения и сеансы

Многие удивляются, когда узнают, что подключение — это не синоним сеанса. Большинство людей считают, что эти термины определяют одно и то же, но в действительности это не так. Подключение может иметь ноль, один или более установленных в нем сеансов. Каждый сеанс является отдельным и независимым, даже если они все совместно используют одно физическое подключение к базе данных. Фиксация транзакции, выполненная в одном сеансе, не оказывает влияния на остальные сеансы этого подключения. В действительности, каждый сеанс, использующий данное подключение, мог бы применять отдельные параметры идентификации пользователя!

В среде Oracle подключение представляет собой всего лишь физическую линию связи между клиентским процессом и экземпляром базы данных — чаще всего в этой роли выступает сетевое подключение. Подключение может производиться к процессу выделенного сервера или к диспетчеру. Как было сказано ранее, подключение может иметь ноль или более сеансов. То есть допускается существование подключения без соответствующих ему сеансов. Кроме того, сеанс *может иметь*, а *может и не иметь* подключение. С помощью усовершенствованных функциональных средств Oracle Net, таких как пул подключений, клиент может закрывать физическое подключение, оставляя сеанс невредимым (но бездействующим). Когда клиенту потребуется выполнить определенную операцию в этом *сеансе*, ему придется заново установить физическое *подключение*. Определим эти элементы подробнее.

- *Подключение (connection).* Подключение — это физическая линия связи между клиентом и экземпляром Oracle. Подключение устанавливается либо по сети, либо посредством механизма IPC. Как правило, подключение устанавливается между процессом клиента и выделенным сервером либо диспетчером. Однако диспетчер подключений Oracle (Connection Manager — CMAN) позволяет устанавливать подключения между клиентом и CMAN и между CMAN и базой данных. Подробное рассмотрение CMAN выходит за рамки этой книги, но определенные сведения об этом программном средстве можно найти в руководстве *Oracle Net Services Administrator's Guide* (Руководство администратора служб Oracle Net), которое доступно на сайте `http://otn.oracle.com`.
- *Сеанс (session).* Сеанс — это логический элемент, существующий в экземпляре. Каждый уникальный сеанс представлен *состоянием сеанса*, или коллекцией структур данных в памяти, которые представляют уникальный сеанс. Именно о нем думает большинство людей, имея в виду подключение к базе данных. Запросы SQL, фиксация транзакций и хранимые процедуры выполняются внутри сеанса на сервере.

Для наблюдения подключений и сеансов в действии, а также чтобы действительно удостовериться в возможности существования в подключении более одного сеанса, можно использовать SQL*Plus. Для этого мы просто применим команду `AUTOTRACE` и увидим, что есть два сеанса. Используя единственный процесс, мы создадим два сеанса, работающих через единственное подключение. Первый сеанс выглядит следующим образом:

```
ops$tkyte%ORA11G> select username, sid, serial#, server, paddr, status
  2 from v$session
  3 where username = USER
  4 /

USERNAME   SID   SERIAL#  SERVER    PADDR    STATUS
--------- ---- --------  --------- -------- --------
OPS$TKYTE   49      225   DEDICATED AE4CF614 ACTIVE
```

Непосредственно в данный момент существует один сеанс: единственный сеанс, подключенный посредством выделенного сервера. В столбце PADDR показан адрес единственного процесса выделенного сервера. Теперь просто включим функцию AUTOTRACE для получения статистических сведений об операторах, выполняемых в среде SQL*Plus:

```
ops$tkyte@ORA10G> set autotrace on statistics
ops$tkyte@ORA10G> select username, sid, serial#, server, paddr, status
  2 from v$session
  3 where username = USER
  4 /
USERNAME   SID  SERIAL#   SERVER    PADDR    STATUS
--------- ---- --------  --------- -------- --------
OPS$TKYTE   30     5476   DEDICATED 32BC2B84 INACTIVE
OPS$TKYTE   49      225   DEDICATED 32BC2B84 ACTIVE

Statistics
----------------------------------------------------------
        0 recursive calls
        0 db block gets
        0 consistent gets
        0 physical reads
        0 redo size
      831 bytes sent via SQL*Net to client
      419 bytes received via SQL*Net from client
        2 SQL*Net roundtrips to/from client
        0 sorts (memory)
        0 sorts (disk)
        2 rows processed
ops$tkyte@ORA10G> set autotrace off
```

Как видите, теперь система содержит два сеанса, но они оба используют единственный процесс выделенного сервера, о чем свидетельствует одно и то же значение PADDR. Ни одного нового процесса не было создано в операционной системе, и для обоих сеансов применяется единственный процесс — одно подключение. Обратите внимание, что один из сеансов (первоначальный) является активным (об этом свидетельствует значение ACTIVE в столбце STATUS (состояние)). Это вполне объяснимо: именно этот процесс выполняет запрос на отображение приведенной информации, поэтому, естественно, он активен. Но что собой представляет неактивный (INACTIVE) сеанс? Это сеанс AUTOTRACE. Его задача состоит в слежении за реальным сеансом и докладе о выполняемых им действиях.

В случае активизации функции AUTOTRACE в интерфейсе SQL*Plus при выполнении операций DML (INSERT, UPDATE, DELETE, SELECT и MERGE) среда SQL*Plus будет выполнять описанные ниже действия.

1. Она будет создавать новый сеанс, использующий текущее подключение, если второй сеанс еще не существует.
2. Она будет требовать от нового сеанса выполнения запроса представления V$SESSTAT для запоминания значений сеанса, в котором мы будем выполнять операцию DML. Это во многом аналогично функции, реализованной сценарием watch_stat.sql в главе 4.
3. Она будет выполнять операцию DML в исходном сеансе.
4. По завершении выполнения оператора DML среда SQL*Plus снова потребует от нового сеанса выполнения запроса представления V$SESSTAT и создания приведенного ранее отчета с отображением различий в статистических сведениях о сеансе, который выполнил операцию DML.

При отключении функции AUTOTRACE среда SQL*Plus прервет этот дополнительный сеанс, и он больше не будет отображаться в представлении V$SESSTAT. У читателей может возникнуть вопрос: "Почему SQL*Plus действует подобным образом?" Среда SQL*Plus выполняет эти действия по той же причине, по которой в главе 4 мы использовали второй сеанс SQL*Plus для отслеживания потребления памяти и временного пространства: сам мониторинг отнимает определенный объем памяти. Отображение статистических сведений в единственном сеансе привело бы к изменению этой информации. Если бы среда SQL*Plus использовала единственный сеанс для вывода отчета о количестве выполненных операций ввода-вывода, об объеме данных, переданных по сети, и о числе выполненных операций сортировки, то запросы, применяемые для получения этих сведений, также внесли бы свой вклад в статистическую информацию. Они могли бы выполнять сортировку, операции ввода-вывода, передачу данных по сети и т.п. Поэтому для получения корректных данных измерений необходимо использовать другой сеанс.

До сих пор мы имели дело с подключением с одним или двумя сеансами. Теперь было бы желательно воспользоваться SQL*Plus для просмотра информации подключения, не имеющего сеансов. Это достаточно легко сделать. Для этого в том же окне SQL*Plus, которое было использовано в предыдущем примере, достаточно ввести команду с дезориентирующим именем DISCONNECT:

```
ops$tkyte%ORA11GR2> disconnect
Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0
- Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

Формально эта команда должна была бы называться DESTROY_ALL_SESSIONS (уничтожить все сеансы), а не DISCONNECT (отключиться), поскольку в действительности она не выполняет физическое отключение от базы данных.

На заметку! Действительное отключение в SQL*Plus является "выход" (exit), поскольку для полного разрыва соединения нужно выйти из программы.

Однако мы все же закрыли все сеансы. Откроем еще один сеанс, используя при этом учетную запись какого-то другого пользователя, и инициируем запрос к базе данных (естественно, заменив OPS$TKYTE соответствующим именем учетной записи):

```
$ sqlplus / as sysdba
SQL*Plus: Release 11.2.0.1.0 Production on Wed May 12 14:08:16 2010
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

sys%ORA11GR2> select * from v$session where username = 'OPS$TKYTE';
no rows selected
строки не выбраны
```

Здесь можно убедиться в отсутствии каких-либо сеансов, но в наличии процесса — физического подключения (используется ранее показанное значение PADDR):

```
sys%ORA11GR2> select username, program
  2   from v$process
  3  where addr = hextoraw( '32BC2B84' );

USERNAME        PROGRAM
--------------  ------------------------------------------------
tkyte           oracle@localhost.localdomain (TNS V1-V3)
```

Итак, в данном случае мы имеем "подключение" без каких-либо связанных с ним сеансов. Можно использовать также не очень удачно названную команду CONNECT в SQL*Plus для создания нового сеанса в существующем процессе (вариант CREATE_SESSION (создать сеанс) лучше бы отражал сущность этой команды). Используя сеанс SQL*Plus, в котором было выполнено отключение, введем следующую команду:

```
ops$tkyte%ORA11GR2> connect /
Connected.
Подключено.

ops$tkyte%ORA11GR2> select username, sid, serial#, server, paddr, status
  2  from v$session
  3 where username = USER;

USERNAME        SID    SERIAL# SERVER    PADDR    STATUS
--------- --------- --------- --------- -------- --------
OPS$TKYTE        37       404 DEDICATED 32BC2B84 ACTIVE
```

Обратите внимание, что значение PADDR осталось неизменным — следовательно, мы используем то же самое физическое подключение. Но значение SID может быть другим (потенциально). "Потенциально" означает, что сеансу может быть назначен и тот же самый SID — это зависит от того, вошли ли другие пользователи в систему, пока мы были от нее отключены, и от доступности первоначального идентификатора SID.

На заметку! В среде Windows или других ОС, основанных на потоках, могут быть получены различные результаты — адрес процесса может меняться, поскольку вы подключаетесь к многопоточному процессу, а не к специализированному процессу, как это происходит в Unix.

До сих пор все тесты выполнялись с применением подключения посредством выделенного сервера. Поэтому значение PADDRR представляло адрес процесса выделенного сервера. А что происходит при использовании разделяемого сервера?

На заметку! Чтобы подключиться через разделяемый сервер, экземпляр базы данных нужно запустить с соответствующей настройкой. Освещение конфигурирования разделяемого сервера выходит за рамки этой книги, но эта тема подробно рассмотрена в руководстве *Oracle Net Services Administrator's Guide* (Руководство администратора сетевых служб Oracle).

Войдем в систему, используя разделяемый сервер, и выполним запрос в этом сеансе:

```
ops$tkyte%ORA11GR2> select a.username, a.sid, a.serial#, a.server,
  2       a.paddr, a.status, b.program
  3   from v$session a left join v$process b
  4     on (a.paddr = b.addr)
  5  where a.username = 'OPS$TKYTE'
  6  /

USERNAME  SID SERIAL# SERVER PADDR   STATUS  PROGRAM
--------- --- ------- ------ ------- ------- ------------------------------
OPS$TKYTE  49     239 SHARED 32BC20AC ACTIVE  oracle@localhost.localdomain(S000)
```

Наше подключение разделяемого сервера ассоциировано с процессом — запрос отображает соответствующее значение PADDR, и теперь можно воспользоваться представлением V$PROCESS для извлечения имени этого процесса. В данном случае мы видим, что он является разделяемым сервером, о чем свидетельствует S000.

Однако если для запроса той же информации воспользоваться другим окном SQL*PLus, оставив наш сеанс разделяемого сервера бездействующим, мы увидим нечто вроде:

```
sys%ORA11GR2> select a.username, a.sid, a.serial#, a.server,
  2        a.paddr, a.status, b.program
  3   from v$session a left join v$process b
  4     on (a.paddr = b.addr)
  5  where a.username = 'OPS$TKYTE'
  6  /

USERNAME  SID SERIAL# SERVER PADDR   STATUS  PROGRAM
--------- --- ------  ------ ------- ------- -------------------------------
OPS$TKYTE  49     239 NONE   32BC15D4 INACTIVE oracle@localhost.localdomain (D000)
```

Обратите внимание, что наш PADDR отличается, и имя процесса, с которыми мы связаны, также изменилось. Бездействующее соединение с разделяемым сервером теперь ассоциировано с диспетчером D000. Следовательно, имеется еще один метод для обзора множественных сеансов, указывающих на один процесс. Диспетчер может иметь сотни или даже тысячи сеансов, указывающих на него.

Интересная особенность соединений с разделяемым сервером состоит в том, что используемый нами процесс разделяемого сервера может меняться от вызова к вызову. Если я — единственный пользователь этой системы (а так было у меня в этих тестах), многократный запуск этого запроса от имени OPS$TKYTE должен производить один и тот же PADDR 32BC20AC снова и снова. Однако если открыть больше соединений разделяемого сервера и запускать их в разных сеансах, то можно заметить, что используемый разделяемый сервер варьируется.

Рассмотрим следующий пример. Я запрошу информацию о текущем сеансе, в том числе имя применяемого разделяемого сервера. Затем в другом сеансе разделяемого сервера я выполню длительную операцию (т.е. монополизирую этот разделяемый сервер). При повторном запросе базы данных об имени используемого разделяемого сервера, скорее всего, отобразится другой сервер (как если бы исходный сервер обслуживал другой сеанс). В следующем примере выделенный полужирным код представляет второй сеанс SQL*Plus, подключенный посредством разделяемого сервера:

```
ops$tkyte%ORA11GR2> select a.username, a.sid, a.serial#, a.server,
  2       a.paddr, a.status, b.program
  3  from v$session a left join v$process b
  4    on (a.paddr = b.addr)
  5 where a.username = 'OPS$TKYTE'
  6 /

USERNAME  SID SERIAL# SERVER PADDR   STATUS  PROGRAM
--------- --- ------  ------ ------- ------  -------------------------------
OPS$TKYTE  49     241 SHARED 32BC20AC ACTIVE  oracle@localhost.localdomain (S000)

scott%ORA11GR2> connect scott/tiger@orcl_ss
Connected.
Подключено.

scott%ORA11GR2> exec dbms_lock.sleep(20);
PL/SQL procedure successfully completed.
Процедура PL/SQL успешно выполнена.

ops$tkyte%ORA11GR2> select a.username, a.sid, a.serial#, a.server,
  2        a.paddr, a.status, b.program
  3   from v$session a left join v$process b
  4     on (a.paddr = b.addr)
  5  where a.username = 'OPS$TKYTE'
  6 /
```

```
USERNAME  SID SERIAL# SERVER PADDR   STATUS  PROGRAM
--------- --- ------- ------ ------- ------ -------------------------------
OPS$TKYTE  49     241  SHARED  32BC8D1C ACTIVE   oracle@localhost.localdomain (S001)
```

На заметку! Необходимо использовать учетную запись, имеющую право выполнения пакета DBMS_LOCK. Я выдал демонстрационному пользователю SCOTT привилегии на запуск пакета DBMS_LOCK следующим образом:

```
sys%ORA11GR2> grant execute on dbms_lock to scott;
```

Обратите внимание, что при первом выполнении запроса в качестве разделяемого сервера был использован процесс S000. Затем во втором сеансе мы выполнили отнимающий длительное время оператор, который монополизировал разделяемый сервер — в данном случае им оказался процесс S000. Задание передается первому же незанятому разделяемому серверу, и поскольку в данному случае никто другой не выдавал запрос на использование разделяемого сервера S000, команда DBMS_LOCK передается именно ему. Когда после этого я снова выполнил запрос в первом сеансе SQL*Plus, это задание было назначено процессу другого разделяемого сервера, поскольку разделяемый сервер S000 был занят.

Интересно отметить, что разбор запроса (который еще не возвращает никаких строк) мог бы выполниться разделяемым сервером S000, извлечение первой строки — сервером S0001, извлечение второй строки — сервером S0002, а закрытие курсора — сервером S0003. То есть отдельные операторы могут последовательно выполняться множеством разделяемых серверов.

Итак, в этом разделе мы убедились, что подключение — физическая линия связи между клиентом и экземпляром базы данных — может иметь ноль, два или более открытых для него сеансов. Один из возможных вариантов такого применения мы видели при использовании функции AUTOTRACE интерфейса SQL*Plus. Многие другие инструментальные средства задействуют эту возможность. Например, компонент Oracle Forms организует несколько сеансов одного подключения для реализации своих функций отладки. Функция многоуровневой прокси-аутентификации Oracle, обеспечивающая сквозную идентификацию пользователей от браузера до базы данных, интенсивно применяет концепцию одного подключения с несколькими сеансами. Но потенциально в каждом сеансе может быть задействована другая учетная запись пользователя. Мы удостоверились, что на протяжении некоторого интервала времени сеансы могут использовать множество процессов, особенно в среде с разделяемым сервером. Кроме того, в случае применения пула подключений Oracle Net сеанс может быть вообще не связан ни с одним процессом. Клиент будет разрывать соединение по истечении установленного интервала бездействия и незаметно для пользователя восстанавливать его при обнаружении активности.

Короче говоря, подключения и сеансы связаны между собой отношением типа “многие ко многим”. Однако наиболее распространенным вариантом, с которым большинство из нас встречается изо дня в день, является отношение “один к одному” между выделенным сервером и единственным сеансом.

Сравнение режимов выделенного, разделяемого сервера и DRCP

Прежде чем продолжить анализ остальных процессов, необходимо разобраться, чем обусловлено существование трех режимов подключения и когда один из них может быть более пригодным, чем остальные.

Когда следует использовать выделенный сервер

Как уже было сказано ранее, в режиме выделенного сервера между подключением клиента и серверным процессом имеется соответствие типа "один к одному". В настоящее время для всех SQL-приложений этот метод подключения к базе данных Oracle является наиболее распространенным. Он несложен в настройке, предоставляет простейший способ установки подключений и требует очень незначительного конфигурирования или вообще не нуждается в нем.

В связи с существованием однозначного соответствия можно не беспокоиться, что длительные транзакции заблокируют другие транзакции. Эти другие транзакции будут выполнены просто с помощью собственных выделенных процессов. Следовательно, этот метод подключения является единственным, к которому следует прибегать в среде, отличной от OLTP (On-Line Transaction Processing — оперативная обработка транзакций), в которой могут выполняться длительные транзакции. Конфигурация с выделенным сервером — рекомендуемая конфигурация Oracle, и она допускает значительное масштабирование. До тех пор, пока сервер располагает достаточным объемом аппаратных ресурсов (процессора и ОЗУ) для обслуживания стольких процессов выделенного сервера, в скольких нуждается система, выделенный сервер может обеспечивать поддержку тысяч одновременно работающих подключений.

Некоторые операции, такие как запуск или останов базы данных, должны выполняться только в режиме выделенного сервера. Поэтому каждая база данных будет содержать либо обе конфигурации, либо только конфигурацию с выделенным сервером.

Когда следует использовать разделяемый сервер

Установка и конфигурирование разделяемого сервера, хотя и не сложны, но все же требуют выполнения дополнительных действий по сравнению с установкой выделенного сервера. Однако основное различие между этими конфигурациями состоит не в их установке и настройке, а в режиме работы. При использовании выделенного сервера между клиентскими подключениями и серверными процессами имеется однозначное соответствие. При использовании разделяемого сервера подключения и процесс связаны отношением "многие к одному": множество клиентов связано с одним разделяемым сервером.

Как следует из его названия, разделяемый сервер — это разделяемый ресурс, чего нельзя сказать о выделенном сервере. Работая с разделяемым ресурсом, следует соблюдать осторожность, чтобы не монополизировать его на длительные периоды времени. Как уже было показано ранее, выдача простой команды `DBMS_LOCK.SLEEP(20)` в одном сеансе привело бы к монополизации процесса разделяемого сервера на 20 секунд. Монополизация ресурсов разделяемого сервера может создавать видимость "зависания" системы.

На рис. 5.2 изображены два разделяемых сервера. При наличии трех клиентов, которые все пытались выполнить 45-секундный процесс приблизительно в одно и то же время, два из них получили бы ответ за 45 секунд, а третий — через 90 секунд. Отсюда правило номер один применения разделяемого сервера: убедитесь, что все транзакции достаточно кратковременны. Они могут выполняться часто, но должны быть короткими (в соответствии с требованиями, предъявляемыми системами оперативной обработки транзакций). Если они длительны, это приведет к замедлению работы всей системы, обусловленному монополизацией разделяемых ресурсов несколькими процессами. В экстремальных ситуациях, если все разделяемые серверы будут заняты, система будет казаться зависшей всем пользователям, за исключением нескольких счастливчиков, которые монополизировали разделяемые серверы.

Еще одна интересная ситуация, которая может возникать при работе с разделяемым сервером — *искусственная взаимоблокировка*. При использовании разделяемого сервера ряд серверных процессов разделяются потенциально большим сообществом пользователей. Рассмотрим ситуацию с пятью разделяемыми серверами и сотней установленных сеансов пользователей. В такой системе максимум пять из этих сеансов пользователей могут быть активными в каждый конкретный момент времени. Предположим, что один из этих сеансов пользователей обновляет строку и не выполняет фиксацию транзакции. Пока этот пользователь бездействует и размышляет над своей транзакцией, пять других сеансов пользователей пытаются заблокировать эту же строку. Естественно, они оказываются заблокированными и будут вынуждены терпеливо дожидаться момента, когда эта строка станет доступной. Наконец, сеанс пользователя, который установил блокировку данной строки, предпринимает попытку зафиксировать свою транзакцию (тем самым, снимая блокировку со строки). Этот сеанс пользователя обнаружит, что все разделяемые серверы монополизированы пятью ожидающими сеансами. Возникает ситуация искусственной взаимоблокировки: владелец блокировки не получит в свое распоряжение разделяемый сервер, чтобы выполнить фиксацию транзакции, если только один из ожидающих сеансов не уступит свой разделяемый сервер. Если только для ожидающих сеансов не определен интервал истечения времени ожидания, они никогда не уступят свои разделяемые серверы (разумеется, можно потребовать от администратора, чтобы он "уничтожил" свой сеанс разделяемого сервера для выхода из этой тупиковой ситуации).

Таким образом, по описанным причинам разделяемый сервер подходит только для систем оперативной обработки транзакций (OLTP), характеризующихся короткими, часто выполняющимися транзакциями. В системе оперативной обработки транзакций транзакции выполняются за несколько миллисекунд — ни один процесс никогда не занимает более доли секунды. Разделяемый сервер совершенно не подходит для информационных хранилищ. В них могут выполняться запросы, которые занимают одну, две, пять и более минут. В режиме разделяемого сервера это было бы смертельно опасным. Если система на 90% является системой оперативной обработки транзакций и на 10% — не полностью таковой, в одном и том же экземпляре базы данных можно одновременно применять выделенные серверы и разделяемый сервер. Подобным образом можно радикально уменьшить количество серверных процессов для пользователей системы оперативной обработки транзакций и одновременно предотвратить монополизацию своих разделяемых серверов пользователями, которые не являются "чистыми пользователями системы OLTP". Кроме того, администратор базы данных может применять встроенный диспетчер ресурсов (Resource Manager) для еще более строгого управления использованием ресурсов.

Естественно, веской причиной применения разделяемого сервера является отсутствие другого выбора. Многие усовершенствованные функции подключения требуют использования разделяемого сервера. Если требуются пулы подключений Oracle Net, применение разделяемого сервера обязательно. Концентрация соединений нескольких баз данных также требует использования разделяемого сервера для этих подключений.

На заметку! Если приложение уже использует пул подключений (например, пул подключений J2EE), и размер пула подключений соответствующим образом определен, применение разделяемого сервера приведет только к снижению производительности. Пул подключений уже определен для поддержки всех подключений, которые могут быть одновременно открыты в любой конкретный момент времени — при этом желательно, чтобы каждое из этих подключений было прямым подключением к выделенному серверу. В противном случае пул подключений будет соединяться всего лишь с еще одним пулом подключений.

Потенциальные преимущества разделяемого сервера

Каковы же преимущества применения разделяемого сервера с учетом того, что следует соблюдать определенную осторожность в отношении типов транзакций, которым разрешено его использовать? В основном, разделяемый сервер предоставляет три преимущества: он уменьшает количество процессов/потоков операционной системы; он искусственно ограничивает уровень параллелизма; и он снижает объем памяти, требуемый в системе. Мы подробнее рассмотрим эти моменты в последующих разделах.

Уменьшение количества процессов/потоков операционной системы

В системе с несколькими тысячами пользователей попытка управления тысячами процессов должна быстро приводить к перегрузке операционной системы. В типичной системе только часть из тысяч пользователей одновременно активна в любой конкретный момент времени. Например, недавно мне пришлось работать с системой, которая обслуживала 5000 одновременно работающих пользователей. В каждый конкретный момент времени максимум 50 из них были активны. Эта система могла бы эффективно работать при организации 50 процессов разделяемых серверов, что позволило бы уменьшить количество процессов, которыми должна управлять операционная система, на два порядка. При такой конфигурации операционная система может в значительной степени избегать переключения контекста.

Искусственное ограничение уровня параллелизма

Мне, как человеку, который принимал участие во множестве сравнительных тестов, это преимущество совершенно очевидно. При выполнении сравнительных тестов администратора часто просят создавать максимальное количество пользователей вплоть до сбоя системы. Одним из результатов таких сравнительных тестов всегда является график зависимости числа транзакций от количества одновременно работающих пользователей (рис. 5.3).

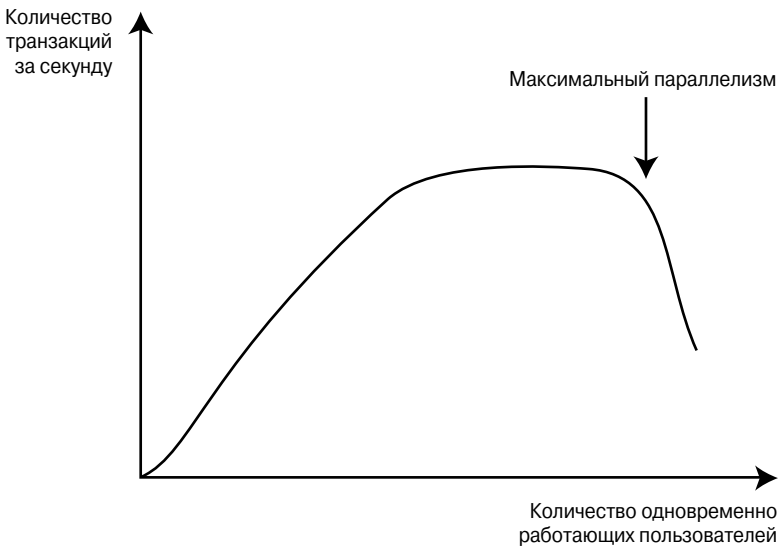

Рис. 5.3. Зависимость числа транзакций от количества одновременно работающих пользователей

Вначале с увеличением числа одновременно работающих пользователей количество транзакций увеличивается. Однако в определенный момент увеличение числа пользователей перестает приводить к увеличению количества транзакций, выполняемых за одну секунду — более того, кривая графика начинает снижаться. Это свидетельствует о достижении максимума пропускной способности системы, и теперь время отклика начинает возрастать (число транзакций, выполняемых за одну секунду, остается неизменным, но время отклика для конечных пользователей увеличивается). Дальнейший рост числа пользователей ведет к действительному снижению пропускной способности. Количество одновременно обслуживаемых пользователей, предшествующее этому снижению, соответствует максимальному уровню параллелизма, который допустим для данной системы. Повышение выше этого значения ведет к "затоплению" системы и образованию очередей на выполнение задач. Подобно ситуации затора на пропускном пункте, система больше не может справляться с нагрузкой. При этом не только существенно возрастает время отклика, но и пропускная способность системы может также снижаться, поскольку накладные расходы, обусловленные простым переключением контекста и разделением ресурсов между слишком большим числом потребителей, и сами требуют дополнительных ресурсов. Если ограничить максимальный уровень параллелизма значением, непосредственно предшествующим этому спаду графика, можно сохранить максимальную пропускную способность и минимизировать время отклика для большинства пользователей. Разделяемый сервер позволяет ограничить максимальный уровень параллелизма в системе этим значением.

В качестве аналогии этого процесса можно привести простую дверь. Ширина двери и ширина одного человека ограничивают количество людей, которые могут пройти через дверь за одну минуту. При низкой "нагрузке" никакой проблемы не существует. Однако с приходом новых людей начинает образовываться очередь (интервал загруженности процессора). Если через дверь стремится пройти множество людей, возникает эффект затора — настолько многие любезно произносят "после вас" и топчутся на месте, что пропускная способность падает до нуля. Каждому из людей приходится дожидаться подходящего момента для прохода. Использование очереди позволяет повысить пропускную способность. Кое-кому удается пройти через дверь почти без задержки, как если бы очередь отсутствовала, в то время как другим (попавшим в конец очереди) приходится дожидаться дольше всех, и они могут прийти к выводу, что образование очереди "было неудачной идеей". Но если оценить скорость прохода через дверь (включая человека, стоящего в очереди последним), становится ясно, что модель с применением очереди (разделяемый сервер) действует лучше "свободного" прохода (даже если все проявляют исключительную вежливость, не говоря уже о ситуации открытия двери в день распродажи, когда многие грубо толкаются, пытаясь попасть внутрь).

Снижение требований к системной памяти

Одна из наиболее часто приводимых причин использования разделяемого сервера следующая: он снижает объем требуемой памяти. Да, это так, но экономия не столь значительна, как можно было бы думать, особенно в случае применения новых средств автоматического управления памятью PGA, описанных в главе 4, когда рабочие области выделяются процессу, используются и освобождаются — причем их размеры изменяются в соответствии с параллельной рабочей нагрузкой. Таким образом, этот аргумент был *более весомым* в предшествующих версиях Oracle, но в настоящее время он утратил свою актуальность. Кроме того, следует помнить, что при использовании разделяемого сервера область UGA выделяется в области SGA. Это означает, что при переходе к применению разделяемого сервера необходимо точно определить предполагаемую потребность в памяти UGA и посредством параметра `LARGE_POOL_SIZE` выделить соответствующий объем в области SGA.

Таким образом, как правило, потребность в памяти SGA для конфигурации с разделяемым сервером очень велика. Обычно эта память должна быть выделена заранее и, следовательно, может использоваться только экземпляром базы данных.

На заметку! При использовании области SGA изменяемого размера ее можно увеличивать и уменьшать с течением времени, но в большинстве случаев она будет "принадлежать" экземпляру базы данных и будет недоступной для других процессов.

В ситуации с выделенным сервером ситуация совершенно меняется, поскольку любой пользователь может работать с любой областью памяти, не выделенной SGA. Но если в случае применения разделяемого сервера размер области SGA значительно больше из-за выделяемой в ней области UGA, то каким образом обеспечивается экономия памяти? Она обусловлена наличием множества распределенных областей PGA. Каждый выделенный/разделяемый сервер имеет область PGA. В ней хранится информация о процессе. Это и области сортировки, и области хеширования, и другие структуры, связанные с процессом. Именно эту память необходимо освобождать в системе за счет организации разделяемого сервера. При переходе от использования 5000 выделенных серверов к 100 разделяемым серверам экономия памяти будет равна суммарному размеру 4900 более не нужных областей PGA (за исключением их областей UGA).

DRCP

А как насчет DRCP — нового средства Oracle 11*g*? Оно обладает множеством преимуществ, таких как сокращение процессов (используется пул) — возможная экономия памяти без отрицательных последствий. Здесь нет шанса столкнуться с искусственной взаимоблокировкой; например, сеанс, удерживающий блокировку на ресурсе в приведенном выше примере, будет иметь собственный выделенный сервер, *назначенный* ему из пула, и этот сеанс сможет в конечном итоге снять эту блокировку. Он не поддерживает многопоточность разделяемого сервера; когда клиентский процесс получает выделенный сервер из пула, он *владеет* этим процессом до тех пор, пока клиентский процесс не отпустит его. Таким образом, это лучше подходит для клиентских приложений, которые часто подключаются, выполняют некоторую относительно небольшую работу и отключаются — и так снова и снова; короче говоря, он предназначен для клиентских процессов, API-интерфейс которых не предусматривает собственной поддержки эффективного пула соединений.

Заключительные соображения по поводу выделенного/разделяемого сервера

Если система не перегружена или по какой-либо особой причине должен применяться разделяемый сервер, вероятно, лучше всего использовать выделенный сервер. Его легко установить (фактически, он вообще не требует специальной установки), и он проще в настройке.

На заметку! При использовании подключений посредством разделяемого сервера трассировочная информация (вывод `SQL_TRACE=TRUE`) может быть распределена по множеству отдельных файлов трассировки; в этом случае восстановление транзакций, выполненных в ходе данного сеанса, значительно усложняется. С появлением пакета `DBMS_MONITOR` в Oracle Database 10*g* большая часть сложностей исчезла, хотя некоторые и остались.

При наличии очень большого сообщества пользователей, когда заведомо *известно*, что развертывание приложения будет выполняться посредством разделяемого сервера, настоятельно рекомендуется выполнять *разработку и тестирование* с использованием

разделяемого сервера. Разработка с применением только выделенного сервера без тестирования на разделяемом сервере увеличит вероятность возникновения сбоя. Создайте в системе наиболее жесткие условия, проведите сравнительные тесты ее функционирования и удостоверьтесь, что приложение будет правильно работать при использовании разделяемого сервера. То есть убедитесь, что приложение не монополизирует разделяемые серверы на слишком длительное время. Этот недостаток значительно легче устранить во время разработки, чем во время развертывания. Для преобразования длительного процесса в заведомо короткий можно воспользоваться таким средством, как AQ (Advanced Queuing — Усовершенствованная организация очередей), но эта возможность должна быть *запрограммирована* внутрь приложения. Подобные задачи лучше выполнять во время разработки. Кроме того, исторически так сложилось, что для подключения через разделяемый сервер и подключений посредством выделенного сервера доступны различные наборы функциональности. Например, мы уже рассматривали отсутствие возможности автоматического управления памятью PGA в Oracle 9*i*, но в прошлом даже такие важные функции, как хеш-соединения двух таблиц, были недоступны для подключений через разделяемый сервер. (Хеш-соединения доступны в Oracle 9*i* и последующих версиях с разделяемым сервером!)

Фоновые процессы

Экземпляр Oracle образован двумя компонентами: памятью SGA и фоновыми процессами. Фоновые процессы выполняют рутинные задачи по обслуживанию, необходимые для поддержания базы данных в рабочем состоянии. Например, существует процесс, который обслуживает кэш буферов блоков, по мере необходимости записывая блоки в файлы данных. Другой процесс отвечает за копирование файла оперативного журнала повторения при его заполнении в каталог архива. Еще один процесс выполняет очистку за прерванными процессами и т.д. Каждый из этих процессов в достаточной степени специализирован для решения своей задачи, но работает с учетом всех остальных процессов. Например, когда процесс, отвечающий за запись журнальных файлов, заполнит один журнал и перейдет к следующему, он сообщит процессу, отвечающему за архивирование заполненного журнального файла, о наличии задачи, которую нужно выполнить.

Существует представление `V$`, которое можно использовать для получения информации обо всех возможных фоновых процессах Oracle и выяснения того, какие из них в данный момент используются системой:

```
ops$tkyte%ORA11GR2> select paddr, name, description
  2 from v$bgprocess
  3 order by paddr desc
  4 /

PADDR      NAME   DESCRIPTION
--------   ----   --------------------------------------------------
32BC8D1C   VKRM   Virtual sKeduler for Resource Manager
32BC8244   SMCO   Space Manager Process
32BC4C0C   CJQ0   Job Queue Coordinator
32BC365C   QMNC   AQ Coordinator
...
00         ACFS   ACFS CSS
00         XDMG   cell automation manager
00         XDWK   cell automation worker actions

295 rows selected.
295 строк выбрано.
```

В этом представлении строки, в которых значение адреса PADDR отличается от 00, являются процессами (потоками), сконфигурированными и выполняющимися в системе.

Существуют два класса фоновых процессов: специализированные для выполнения конкретных задач (подобные описанным) и предназначенные для выполнения ряда других задач (служебные процессы). Например, существует доступный через пакет DBMS_JOB служебный фоновый процесс, предназначенный для обработки внутренних очередей заданий. Этот процесс следит за очередями заданий и запускает все помещенные в них задания. Во многих отношениях он подобен процессу выделенного сервера, но не обладает клиентским подключением. Мы рассмотрим каждый из этих фоновых процессов, начиная с тех, которые предназначены для решения конкретной задачи, а затем перейдем к рассмотрению служебных процессов.

Специализированные фоновые процессы

Количество, имена и типы специализированных фоновых процессов варьируется в зависимости от выпуска. *Типичный набор* фоновых процессов Oracle, имеющих конкретное назначение, показан на рис. 5.4.

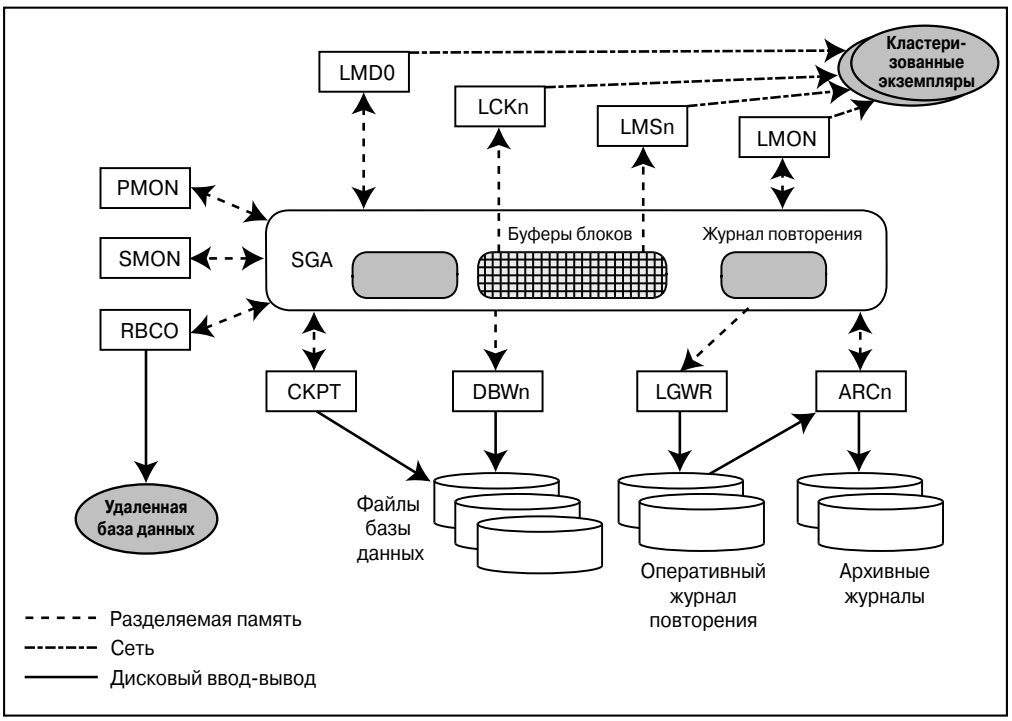

Рис. 5.4. Специализированные фоновые процессы

Например, когда в Oracle Database 11*g* Release 2 база данных стартует с использованием минимального набора параметров init.ora:

```
ops$tkyte%ORA11GR2> !cat /tmp/pfile
*.compatible='11.2.0.0.0'
*.control_files='/home/ora11gr2/app/ora11gr2/oradata/orcl/control01.ctl',
 '/home/ora11gr2/app/ora11gr2/flash_recovery_area/orcl/control02.ctl'
*.db_block_size=8192
```

```
*.db_name='orcl'
*.memory_target=314572800
*.undo_tablespace='UNDOTBS1'
```

запускается около 17 фоновых процессов:

```
ops$tkyte%ORA11GR2> select paddr, name, description
  2 from v$bgprocess
  3 where paddr <> '00'
  4 order by paddr desc
  5 /

PADDR      NAME  DESCRIPTION
--------   ----  -------------------------------------------
32AF0E64   CJQ0  Job Queue Coordinator
32AEF8B4   QMNC  AQ Coordinator
32AEE304   MMNL  Manageability Monitor Process 2
32AED82C   MMON  Manageability Monitor Process
32AECD54   RECO  distributed recovery
32AEC27C   SMON  System Monitor Process
32AEB7A4   CKPT  checkpoint
32AEACCC   LGWR  Redo etc.
32AEA1F4   DBW0  db writer process 0
32AE971C   MMAN  Memory Manager
32AE8C44   DIA0  diagnosibility process 0
32AE816C   PSP0  process spawner 0
32AE7694   DBRM  DataBase Resource Manager
32AE6BBC   DIAG  diagnosibility process
32AE60E4   GEN0  generic0
32AE560C   VKTM  Virtual Keeper of TiMe process
32AE4B34   PMON  process cleanup

17 rows selected.
```

Если использовать тот же `init.ora`, заменив только `memory_target` на `sga_target` и `pga_aggregate_target`, в Oracle Database 10*g* Release 2 вы увидите всего 12 процессов:

```
ops$tkyte %ORA10GR2> select paddr, name, description
  2 from v$bgprocess
  3 where paddr <> '00'
  4 order by paddr desc
  5 /

PADDR      NAME  DESCRIPTION
--------   ----  -------------------------------------------
23D27AC4   CJQ0  Job Queue Coordinator
23D27508   QMNC  AQ Coordinator
23D26990   MMNL  Manageability Monitor Process 2
23D263D4   MMON  Manageability Monitor Process
23D25E18   RECO  distributed recovery
23D2585C   SMON  System Monitor Process
23D252A0   CKPT  checkpoint
23D24CE4   LGWR  Redo etc.
23D24728   DBW0  db writer process 0
23D2416C   MMAN  Memory Manager
23D23BB0   PSP0  process spawner 0
23D235F4   PMON  process cleanup

12 rows selected.
```

Обратите внимание, что при запуске экземпляра могут запускаться не все из этих процессов, но большинство из них будет присутствовать. Процесс ARCn (архиватор) будет выполняться только в режиме ARCHIEVELOG при включенной функции автоматического архивирования. Процессы LMD0, LCKn, LMON и LMSn (подробнее они рассматриваются чуть ниже) выполняются только при использовании средства Oracle RAC — конфигурации Oracle, которая позволяет множеству экземпляров на различных компьютерах одного кластера монтировать и открывать одну и ту же физическую базу данных.

Итак, на рис. 5.4 изображены процессы, которые можно "увидеть", запустив экземпляр Oracle и смонтировав и открыв базу данных. Например, после запуска экземпляра базы данных моя система Linux содержит следующие процессы:

```
$ ps -aef | grep ora_....._$ORACLE_SID | grep -v grep
ora11gr2 21646      1 0 09:04 ?        00:00:00 ora_pmon_orcl
ora11gr2 21648      1 0 09:04 ?        00:00:00 ora_vktm_orcl
ora11gr2 21652      1 0 09:04 ?        00:00:00 ora_gen0_orcl
ora11gr2 21654      1 0 09:04 ?        00:00:00 ora_diag_orcl
ora11gr2 21656      1 0 09:04 ?        00:00:00 ora_dbrm_orcl
ora11gr2 21658      1 0 09:04 ?        00:00:00 ora_psp0_orcl
ora11gr2 21660      1 0 09:04 ?        00:00:00 ora_dia0_orcl
ora11gr2 21662      1 0 09:04 ?        00:00:00 ora_mman_orcl
ora11gr2 21664      1 0 09:04 ?        00:00:00 ora_dbw0_orcl
ora11gr2 21666      1 0 09:04 ?        00:00:00 ora_lgwr_orcl
ora11gr2 21668      1 0 09:04 ?        00:00:00 ora_ckpt_orcl
ora11gr2 21670      1 0 09:04 ?        00:00:00 ora_smon_orcl
ora11gr2 21672      1 0 09:04 ?        00:00:00 ora_reco_orcl
ora11gr2 21674      1 0 09:04 ?        00:00:00 ora_mmon_orcl
ora11gr2 21676      1 0 09:04 ?        00:00:00 ora_mmnl_orcl
ora11gr2 21678      1 0 09:04 ?        00:00:00 ora_d000_orcl
ora11gr2 21680      1 0 09:04 ?        00:00:00 ora_s000_orcl
ora11gr2 21698      1 0 09:05 ?        00:00:00 ora_qmnc_orcl
ora11gr2 21712      1 0 09:05 ?        00:00:00 ora_cjq0_orcl
ora11gr2 21722      1 0 09:05 ?        00:00:00 ora_q000_orcl
ora11gr2 21724      1 0 09:05 ?        00:00:00 ora_q001_orcl
ora11gr2 21819      1 0 09:10 ?        00:00:00 ora_smco_orcl
ora11gr2 21834      1 0 09:10 ?        00:00:00 ora_w000_orcl
ora11gr2 22005      1 0 09:18 ?        00:00:00 ora_q002_orcl
ora11gr2 22056      1 0 09:20 ?        00:00:00 ora_j000_orcl
ora11gr2 22058      1 0 09:20 ?        00:00:00 ora_j001_orcl
ora11gr2 22074      1 0 09:21 ?        00:00:00 ora_m000_orcl
```

Обратите внимание на используемое этими процессами соглашение по именованию. Имя процесса начинается с ora_. За ними следуют четыре символа, представляющие действительное имя процесса, а потом — _ora10g. Так получилось, что мой ORACLE_SID (идентификатор сайта) — ora10g. В системе UNIX применение этого соглашения по именованию позволяет без труда идентифицировать фоновые процессы Oracle и связать их с конкретным экземпляром (в системе Windows простого способа получения этой информации не существует, поскольку фоновые задачи представляют собой потоки внутри большого единого процесса). Наиболее интересная, но не сразу заметная в приведенном коде особенность — то, что все эти процессы *в действительности представляют собой одну и ту же двоичную исполняемую программу, а не отдельные исполняемые файлы для каждой "программы"*. Как бы тщательно вы не выполняли поиск, вы нигде на диске не обнаружите двоичный исполняемый файл ora_pmon_orcl. Вы не найдете и файл ora_lgwr_orcl или ora_reco_orcl. В действительности все эти процессы представляют файл oracle (таково имя запускаемого двоичного исполняемого файла). Просто

во время запуска экземпляра они назначают себе псевдонимы, облегчающие идентификацию процессов. На платформе UNIX такой подход обеспечивает эффективное совместное использование значительной части объектного кода. В среде Windows данное обстоятельство не представляет такого интереса, поскольку эти компоненты являются лишь потоками внутри процесса и, следовательно, они — части одного большого двоичного исполняемого файла.

Теперь рассмотрим функции, выполняемые каждым из *главных* процессов, начиная с основных фоновых процессов Oracle. Полный список возможных фоновых процессов и краткое описание выполняемых ими функций вы найдете в руководстве *Oracle Server Reference Manual* по адресу `http://otn.oracle.com`.

PMON: монитор процессов

Этот процесс отвечает за очистку после аварийного разрыва подключений. Например, если выделенный сервер выдает сбой или останавливается по какой-либо причине, процесс `PMON` отвечает за исправление ситуации (восстановление или отмену работы) и освобождение ресурсов. `PMON` будет инициировать откат незафиксированных транзакций, снятие блокировок и освобождение ресурсов SGA, выделенных давшему сбой процессу.

Помимо очистки за разорванными подключениями, `PMON` овечает за мониторинг остальных фоновых процессов Oracle и перезапускает их при необходимости (если это возможно). Если разделяемый сервер или диспетчер выдает сбой (терпит аварию), `PMON` вступает в действие и перезапускает другой процесс (после выполнения очистки за отказавшим процессом). `PMON` будет наблюдать за всеми процессами Oracle и либо перезапускать их, либо прекращать работу экземпляра. Например, в случае сбоя процесса записи журнала базы данных, `LGWR`, имеет смысл прервать работу экземпляра. Эта ошибка является весьма серьезной, и при ее возникновении наиболее безопасный образ действия — немедленное прекращение работы экземпляра и предоставление программе восстановления возможности исправления данных. (Следует отметить, что подобная ситуация встречается очень редко, и о ней нужно немедленно сообщать службе поддержки Oracle Support.)

Еще одна задача, выполняемая процессом `PMON` для экземпляра — его регистрация с помощью прослушивающего процесса Oracle TNS. Во время запуска экземпляра процесс `PMON` запрашивает известный ему адрес порта, если только не получает другое указание, для выяснения того, запущен ли прослушивающий процесс. Известным и используемым СУБД Oracle по умолчанию портом является порт с номером 1521. А что происходит, если прослушивающий процесс действует через какой-либо другой порт? В этом случае механизм остается таким же, за исключением того, что адрес прослушивающего процесса должен быть явно указан в параметре `LOCAL_LISTENER`. Если прослушивающий процесс работает во время запуска экземпляра базы данных, `PMON` связывается с прослушивающим процессом и передает ему необходимые параметры, такие как имя службы и показатели нагрузки экземпляра. Если же прослушивающий процесс не был запущен, `PMON` будет периодически пытаться связаться с ним, чтобы зарегистрироваться.

SMON: системный монитор

`SMON` — это процесс, который делает всю работу уровня системы. В то время как `PMON` обслуживает отдельные процессы, процесс `SMON` занят задачами системного уровня и служит своего рода сборщика мусора в базе данных. В числе прочих он выполняет перечисленные ниже задачи.

- *Очистка временного пространства.* С появлением настоящих временных табличных пространств задача очистки временного пространства стала менее трудоемкой, но не исчезла. Например, при построении индекса экстенты, выделенные для индекса во время создания, помечаются как TEMPORARY (временный). В случае прерывания сеанса CREATE INDEX по какой-либо причине процесс SMON должен очистить эти экстенты. Другие операции также создают временные экстенты, относящиеся к сфере ответственности процесса PMON.
- *Объединение свободного пространства.* При использовании табличных пространств, управляемых словарем, процесс SMON отвечает за объединение групп свободных, расположенных по соседству друг с другом экстентов табличного пространства в один большой свободный экстент. Это объединение выполняется только в табличных пространствах, управляемых словарем, для которых pctincrease в конструкции определения используемого по умолчанию хранилища данных установлено в ненулевое значение.
- *Восстановление активных транзакций с учетом недоступных файлов.* Эта задача аналогична решаемой во время запуска базы данных. В ходе ее выполнения процесс SMON восстанавливает неудачные транзакции, которые были пропущены во время восстановления экземпляра/состояния во время аварии из-за недоступности файла (файлов) процессу восстановления. Например, файл может размещаться на диске, который был недоступен или не был смонтирован. Когда файл станет доступным, процесс SMON восстановит его.
- *Восстановление экземпляра для отказавшего узла в среде RAC.* В конфигурации Oracle, использующей технологию RAC, в случае отказа экземпляра базы данных в данном кластере (т.е. при отказе компьютера, на котором действовал экземпляр) какой-то другой узел этого кластера откроет файлы журнала повторения отказавшего экземпляра и выполнит восстановление всех его данных.
- *Очистка OBJ$.* Здесь OBJ$ — низкоуровневая таблица словаря данных, которая содержит записи практически всех объектов (таблиц, индексов, триггеров, представлений и т.п.) базы данных. Во многих случаях она содержит записи удаленных объектов или "отсутствующих" объектов, используемых механизмом взаимозависимостей Oracle. Процесс SMON удаляет эти более не нужные строки.
- *Сжатие сегментов отката.* SMON будет выполнять автоматическое сжатие сегмента отката до оптимального размера, если он определен.
- *Отсоединение сегментов отката.* Администратор базы данных может *отсоединять*, или делать недоступным, сегмент отката, содержащий активные транзакции. Активные транзакции могут использовать этот отсоединенный сегмент отката. В этом случае в действительности сегмент отката оказывается не автономным (отсоединенным), а помеченным как "ожидающий отсоединения". Процесс SMON будет в фоновом режиме периодически предпринимать попытки отсоединить его, пока это не удастся.

Теперь читатели должны были получить представление о задачах, решаемых процессом SMON. Он выполняет также множество других действий, таких как сброс на диск статистической информации мониторинга, отображаемой в представлении DBA_TAB_MONITORING, запись информации преобразования временной метки, хранящейся в таблице SMON_SCN_TIME, и т.п. По истечении некоторого времени работы процесс SMON может занимать значительную долю ресурсов процессора, и это совершенно нормально. Периодически SMON "пробуждается" (самостоятельно или другими фоновыми процессами) для выполнения своих задач по наведению порядка в системе.

RECO: распределенное восстановление базы данных

Процесс RECO решает очень узкую задачу: он восстанавливает транзакции, оставшиеся в подготовленном состоянии вследствие аварии или разрыва подключения во время *двухфазной фиксации* (two-phase commit — 2PC). 2PC — распределенный протокол, который делает возможной автоматическую фиксацию изменений, влияющих на множество физически разделяемых баз данных. Прежде чем выполнить фиксацию, этот протокол предпринимает попытку максимально сузить временное окно возможного распределенного сбоя. При выполнении двухфазной фиксации в *N* базах данных одна из них — как правило (но не всегда) та, в которую клиент вошел вначале — будет играть роль координатора. Этот сайт будет запрашивать остальные *N*–1 сайтов об их готовности к выполнению фиксации. Каждый из *N*–1 сайтов сообщает YES (да) или NO (нет) о своем "состоянии готовности". Если любой из сайтов сообщает о своей неготовности (NO), производится откат всей транзакции. Если все сайты сообщают о готовности (YES), сайт-координатор рассылает сообщение о необходимости выполнения фиксации на каждом из остальных *N*–1 сайтов.

Если после того, как какой либо сайт сообщил о своей готовности к фиксации, но до получения от координатора директивы о необходимости действительного выполнения фиксации происходит сбой в сети или какая-то другая ошибка, транзакция становится *сомнительной распределенной транзакцией*. Протокол 2PC пытается ограничить временной интервал, в течение которого подобная ситуация возможна, но не может устранить его полностью. Если сбой происходит в конкретный момент времени в конкретном месте, выполнение транзакции становится обязанностью процесса RECO. Этот процесс попытается связаться с координатором транзакции для выяснения ее результата. До тех пор, пока он не выполнит это, транзакция будет оставаться в незафиксированном состоянии. Когда процессу RECO удастся снова связаться с координатором транзакции, он выполнит либо фиксацию, либо откат транзакции.

Следует отметить, что при наличии каких-то невыполненных транзакций и длительном периоде простоя фиксацию/откат этих транзакций можно осуществить вручную. Необходимость в выполнении этой операции может возникать в связи с тем, что сомнительные распределенные транзакции могут приводить к *блокированию процессов чтения процессами записи* — одна из ситуаций, которая может произойти в среде Oracle. В этом случае администратор базы данных может связаться с администратором другой базы данных и попросить его проверить состояние соответствующих сомнительных транзакций. Затем администратор базы данных может выполнить фиксацию или откат этих транзакций, освобождая процесс RECO от выполнения этой задачи.

CKPT: процесс выполнения контрольных точек

Вопреки своему названию, процесс выполнения контрольных точек не производит запись контрольных точек (контрольные точки рассматривались в главе 3, в разделе, посвященном журналам повторения) — в основном, это задача процесса DBWn. Процесс CKPT просто облегчает работу процессу записи контрольных точек, обновляя заголовки файлов данных. Раньше этот процесс считался необязательным, но, начиная с версии Oracle 8.0, он запускается всегда. Поэтому он всегда отображается в выводе команды ps в системе UNIX. Обычно принято считать, что обновление заголовков файлов данных информацией о контрольных точках — задача процесса LGWR. Но с увеличением количества файлов и размера базы данных в процессе ее эксплуатации эта дополнительная задача становится слишком большой обузой для LGWR. Если бы процесс LGWR был вынужден обновлять десятки, сотни или даже тысячи файлов, вполне вероятно, что сеансам пришлось бы очень долго дожидаться фиксации этих транзакций. CKPT освобождает LGWR от этой обязанности.

DBWn: процесс записи блоков базы данных

Процесс записи блоков базы данных (DBWn) — фоновый процесс, отвечающий за запись грязных блоков на диск. Этот процесс будет записывать грязные блоки из буферного кэша, как правило, чтобы освободить в нем место (подготовить буферы для считывания других данных) или чтобы продвинуть контрольную точку (перейти к последующей позиции в файле оперативного журнала повторения, с которой СУБД Oracle должна будет начинать чтение при восстановлении экземпляра после сбоя). Как было сказано в главе 3, при переключении журнальных файлов выдается сигнал о создании контрольной точки. СУБД Oracle нужно перейти к следующей контрольной точке, чтобы ей больше не требовался только что заполненный файл оперативного журнала повторения. Если это не будет сделано к моменту, когда потребуется снова использовать файл журнала повторения, отобразится сообщение вроде "checkpoint not complete" ("создание контрольной точки не завершено") и придется дожидаться завершения этой операции.

На заметку! Продвижение журнальных файлов — только один из множества возможных способов создания контрольных точек. Существуют инкрементные контрольные точки, управляемые такими параметрами, как FAST_START_MTTR_TARGET, и другие триггеры, которые вызывают запись грязных блоков на диск.

Как видите, производительность процесса DBWn может иметь очень большое значение. Если он записывает блоки в свободные буферы (буферы, которые могут повторно использоваться для кэширования других блоков) недостаточно быстро, это приведет к увеличению как количества, так и длительности периодов ожидания Free Buffer Waits (Ожидание появления свободного буфера) и Write Complete Waits (Ожидание окончания записи).

Базу данных можно сконфигурировать для использования более одного процесса DBWn — фактически, допускается иметь до 36 процессов DBWn (DBW0...DBW9, DBWa...DBWz). Большинство систем работают с одним процессом записи блоков базы данных, но крупные, многопроцессорные системы могут использовать более одного процесса. Обычно это делается для распределения рабочей нагрузки, связанной с поддержанием в чистом виде большого буферного кэша блоков в области SGA, посредством сброса грязных (модифицированных) блоков на диск.

В идеале для записи блоков на диск процесс DBWn использует асинхронный ввод-вывод. В этом случае DBWn собирает пакет блоков, которые необходимо записать, и передает их операционной системе. Процессу DBWn не приходится дожидаться, пока операционная система действительно запишет блоки. Вместо этого процесс "возвращается назад" и собирает следующий пакет, который должен быть записан. Когда операционная система завершает запись, она асинхронно уведомляет об этом процесс DBWn. В результате DBWn получает возможность работать значительно быстрее, чем если бы ему приходилось выполнять все действия последовательно. Позже, в разделе "Подчиненные процессы" вы узнаете, как применять подчиненные процессы ввода-вывода для имитации асинхронного ввода-вывода на платформах или в конфигурациях, которые его не поддерживают.

По поводу процесса DBWn остается привести еще одно замечание. Можно почти не сомневаться, что процесс DBWn будет записывать блоки по всему диску — он производит множество распределенных записей на диск. При выполнении обновления вы будете модифицировать индексные блоки, хранящиеся в различных областях диска, и блоки данных, которые также распределены по диску случайным образом. И напротив, процесс LGWR выполняет множество последовательных записей в журнал повторения. Это — одно из важных различий между названными процессами и одна из причин использования журнала повторения и процесса LGWR наряду с процессом DBWn.

Запись распределенных блоков осуществляется значительно медленнее записи последовательных. Имея в своем распоряжении грязные блоки буфера SGA и процесс LGWR, выполняющий объемные последовательные записи, которые могут воссоздавать эти грязные буферы, можно значительно повысить производительность системы. То, что процесс DBWn выполняет свою медленную работу в фоновом режиме, в то время как процесс LGWR запускает свои более быстрые операции в периоды ожидания пользователя, способствует увеличению общей производительности системы. Это действительно так, несмотря на то, что, возможно, СУБД Oracle приходится выполнять больше операций ввода-вывода, чем формально требуется (ей приходится выполнять запись в журнал и в файл данных). *Теоретически* запись в оперативный журнал повторения можно было бы пропустить, если бы во время фиксации вместо этого СУБД Oracle физически записывала бы измененные блоки на диск. На практике дело обстоит иначе: LGWR выполняет запись информации повторения в оперативные журналы повторения для каждой транзакции, а DBWn сбрасывает блоки базы данных на диск в фоновом режиме.

LGWR: процесс записи журналов

Процесс LGWR отвечает за сброс на диск содержимого буфера журнала повторения, расположенного в области SGA. Запись на диск производится в одном из следующих случаев:

- каждые три секунды;
- при выполнении фиксации любой транзакции;
- при заполнении буфера журнала повторения на одну треть или при наличии в нем 1 Мбайт буферизованных данных.

Поэтому наличие огромного (объемом в сотни мегабайт) буфера журнала повторения нецелесообразно — СУБД Oracle никогда не сможет задействовать его целиком. В отличие от распределенного ввода-вывода, который должен выполняться процессом DBWn, журналы записываются посредством последовательных операций записи. Выполнение таких пакетных операций записи значительно эффективнее выполнения множества распределенных операций записи в различные части файла. Это обстоятельство — одна из основных причин применения процесса LGWR и журналов повторения. Повышение эффективности за счет применения последовательного ввода-вывода для записи только изменившихся байтов превышает увеличение количества операций ввода-вывода. СУБД Oracle могла бы записывать блоки базы данных непосредственно на диск при выполнении фиксации, но это привело бы к распределенному вводу-выводу целых блоков, что выполнялось бы значительно медленнее последовательной записи изменений процессом LGWR.

ARCn: архивный процесс

Задача процесса ARCn — копирование файла оперативного журнала повторения в другой каталог при его заполнении процессом LGWR. Затем эти архивные файлы журналов повторения можно применять для восстановления с диска. В то время как оперативный журнал повторения используется для "исправления" файлов данных при сбое питания (при прерывании работы экземпляра), архивные журналы повторения служат для "исправления" файлов данных в случае аварии диска. При выходе из строя диска, содержащего файл данных, /d01/oradata/ora11g/system.dbf, можно обратиться к резервным копиям, созданным на прошлой неделе, восстановить эту старую копию файла, и затем потребовать от базы данных применения всех архивных и оперативных журналов повторения, сгенерированных с момента последнего резервного копирования. В результате этот файл окажется "связанным" с остальными файлами базы данных, и работу можно продолжить без утери каких-либо данных.

Как правило, процесс ARCn копирует файлы оперативного журнала повторения, по меньшей мере, в два дополнительных каталога (избыточность — основное средство предотвращения потери данных). Эти каталоги могут располагаться на дисках локального компьютера или, что более целесообразно, на случай серьезной аварии хотя бы один из них должен находиться на другом компьютере. Во многих случаях эти архивные файлы журналов повторения копируются другим процессом в какое-то третье хранилище, например, на магнитную ленту. Они могут также пересылаться на другой компьютер для применения к резервной базе данных — эту возможность в Oracle предусмотрена на случай аварии. Мы вскоре рассмотрим используемые при этом процессы.

DIAG: процесс диагностики

В прежних выпусках процесс DIAG присутствовал исключительно в среде RAC. Начиная с Oracle Database 11*g*, с его новым средством ADR (Advanced Diagnostic Repository — репозиторий расширенной диагностики), именно оно отвечает за мониторинг общей работоспособности экземпляра, и фиксирует информацию, необходимую для обработки сбоев экземпляра. Это касается как конфигураций с одним экземпляром, так и многоэкземплярных конфигураций RAC.

FBDA: процесс архивации ретроспективных данных

Процесс архивации ретроспективных данных (Flashback Data Archiver Process) является новым процессом в Oracle Database 11*g* Release 1 и последующих версиях. Это ключевой компонент нового средства архивирования ретроспективных данных — средства, позволяющего запросить данные в том виде, в каком они были в определенный момент времени в прошлом (например, запросить данные таблицы, как они выглядели год назад и т.д.). Такая возможность достигается благодаря поддержке хронологии изменений каждой строки в таблице со временем. Эта хронология, в свою очередь, поддерживается фоновым процессом FBDA. Данный процесс функционирует, выполняя свою работу вскоре после фиксации транзакций. Процесс FBDA читает данные отката, сгенерированные этой транзакцией, и откатывает изменения, выполненные ею. Затем он регистрирует эти откаченные (исходные значения) строки в архиве ретроспективных данных.

DBRM: процесс диспетчера ресурсов

Процесс диспетчера ресурсов (Database Resource Manager Process) реализует планы ресурсов, которые могут быть сконфигурированы для экземпляра базы данных. Он устанавливает эти планы ресурсов и выполняет различные операции, связанные с соблюдением/реализацией этих планов ресурсов. Диспетчер ресурсов позволяет администраторам базы данных иметь тонкий контроль над ресурсами, используемыми экземпляром базы данных, приложениями, имеющими доступ к базе данных, либо индивидуальными пользователями, обращающимися к базе данных.

GEN0: процесс выполнения общей задачи

Процесс выполнения общей задачи (General Task Execution Process), как следует из его названия, является потоком выполнения общей задачи. Его главная цель — разгрузить потенциально блокирующую обработку (которая может остановить выполнение) от некоторого другого процесса и выполнить его в фоне. Например, если главному процессу ASM требуется некоторая операция, блокирующая файл, но эта операция может быть безопасно выполнена в фоновом режиме (ASM может безопасно продолжать обработку перед завершением операции), то процесс ASM может запросить процесс GEN0 выполнить эту операцию и уведомить о ее завершении. Это похоже по природе на подчиненные процессы, описанные ниже.

Остальные специализированные процессы

В зависимости от используемых функциональных возможностей Oracle в системе могут действовать и другие специализированные процессы.

На заметку! Приложение F руководства по серверу (Server Reference Manual), доступное на `http://otn.oracle.com/`, содержит полный список фоновых процессов и их функций.

Большинство описанных ранее процессов присутствуют всегда — они стартуют при запуске экземпляра Oracle. (`ARCn` формально необязателен, но, по моему мнению, должен быть неотъемлемой частью любой производственной базы данных!) Описанные далее процессы не обязательны и будут действовать в системе только при использовании конкретных функциональных средств. Следующие процессы характерны только для экземпляра базы данных, в котором включена функция ASM (Automatic Storage Management — автоматическое управление пространством хранения), описанная в главе 3.

- *Процесс `ASMB` (Automatic Storage Management Background — фоновый процесс автоматического управления пространством хранения).* Процесс `ASMB` действует в экземпляре базы данных, который использует средство ASM. Он отвечает за обмен данными с экземпляром ASM, управляющим пространством хранения, снабжение экземпляра ASM обновленными статистическими сведениями и предоставление такта (heartbeat) экземпляру ASM, чтобы знать, действует ли он.
- *Процесс ребалансировки (`RBAL`).* Процесс `RBAL` также действует в экземпляре базы данных, использующем функции ASM. Этот процесс выполняет обработку запросов ребалансировки (запроса перераспределения) при добавлении/удалении дисков в и из группы дисков ASM.

Следующие процессы действуют в экземпляре Oracle RAC. Как уже упоминалось, RAC — это конфигурация Oracle, где несколько экземпляров, каждый из которых действует в отдельном узле (как правило, на отдельном физическом компьютере) кластера, могут монтировать и открывать одну базу данных. Это позволяет более чем одному экземпляру получать полный доступ по чтению и записи к одному набору файлов базы данных. Конфигурация RAC предназначена для решения двух целей.

- *Обеспечение высокой доступности.* При использовании Oracle RAC сбой одного узла/компьютера в кластере из-за программной, аппаратной или человеческой ошибки позволит остальным узлам продолжить функционирование. База данных останется доступной через другие узлы. При этом вычислительная мощность системы может несколько снизиться, но доступ к базе данных не будет утерян.
- *Обеспечение масштабируемости.* Вместо того чтобы приобретать все более мощные компьютеры для обеспечения все возрастающей нагрузки (такой подход называют *вертикальным масштабированием*), RAC позволяет наращивать ресурсы путем добавления компьютеров в кластер (этот подход носит название *горизонтального масштабирования*). Вместо того чтобы заменять 4-процессорный компьютер 8- или 16-процессорным, применение RAC позволяет просто добавить в систему еще один сравнительно дешевый 4-процессорный компьютер (или несколько).

Следующие процессы действуют только в среде RAC. В других системах они отсутствуют.

- *Процесс монитора блокировки (lock monitor — `LMON`).* Процесс `LMON` отслеживает все экземпляры в кластере на предмет выявления сбоя экземпляра. В случае его

выявления процесс восстанавливает глобальные блокировки, установленные отказавшим экземпляром. Этот процесс отвечает также за реконфигурирование блокировок и других ресурсов при удалении экземпляров или их добавлении в кластер (при их сбое и повторном подключении или при добавлении в кластер новых экземпляров в реальном времени).

- *Процесс демона диспетчера блокировки (lock manager daemon — LMD).* Процесс LMD обрабатывает запросы службы диспетчера блокировки, связанные с обслуживанием глобального кэша (поддержанием целостности буферов блоков в различных экземплярах). В основном, этот процесс играет роль посредника, отправляя запросы на предоставление ресурсов в очередь, которая обрабатывается процессами LMSn. Процесс LMD выполняет выявление/разрешение глобальных взаимоблокировок и отслеживает связанные с блокировкой паузы в глобальной среде.
- *Серверный процесс диспетчера блокировки (lock manager server — LMSn).* Как уже было сказано ранее, в среде RAC каждый экземпляр Oracle действует на отдельном компьютере в конкретном кластере, и все они обладают доступом по чтению и записи к одним и тем же файлам базы данных. Чтобы это было возможно, буферные кэши блоков области SGA должны быть согласованными друг с другом. Обеспечение этой согласованности — одно из основных назначений процесса LMSn. В предшествующих версиях OPS (Oracle Parallel Server — Параллельный сервер Oracle) это было реализовано посредством *выгрузки из кэша* (ping). То есть, если узлу в кластере требовалось согласованное по чтению представление блока, который был заблокирован другим узлом в монопольном режиме, обмен данными выполнялся посредством сброса на диск (блок оказывался выгруженным из кэша). Даже при чтении данных эта операция требовала большого объема ресурсов. Теперь, с появлением процесса LMSn, этот обмен данными осуществляется через очень быстрый обмен данными между кэшами по высокоскоростному подключению отдельного кластера. Один экземпляр может иметь до десяти процессов LMSn.
- *Процесс блокировки (LCK0).* По своему действию этот процесс подобен описанному ранее процессу LMD, но он обрабатывает запросы на предоставление всех других глобальных ресурсов, отличных от буферов блоков базы данных.

Следующие фоновые процессы общего назначения присутствуют в большинстве одиночных экземпляров или экземпляров RAC.

- *Процесс Spawner Process (PSP0).* Этот процесс отвечает за порождение (запуск/создание) различных фоновых процессов. Это процесс, создающий новые процессы/потоки для экземпляра Oracle. Он выполняет большую часть своей работы во время запуска экземпляра.
- *Виртуальный процесс-хранитель времени (VKTM).* Реализует согласованные, точные часы для экземпляра Oracle. Отвечает за представление времени в формате часов, минут и секунд (читабельном для человека), а также высокоточного таймера (который не обязательно применяет формат часов, минут и секунд, а скорее является высокоточным секундомером, отслеживающим очень малые временные промежутки), используемого для измерения длительностей и интервалов.
- *Процесс координатора управления пространством (SMC0).* Этот процесс является частью инфраструктуры управления. Он координирует работу средств упреждающего выделения пространства базы данных, таких как процесс, находящий место, которое может быть затребовано, и процесс, выполняющий это требование.

Служебные фоновые процессы

Эти фоновые процессы создаются исключительно по необходимости. Они предоставляют функциональные возможности, которые не требуются в повседневной работе базы данных, если только администратор сам не прибегнет к ним, например, создав очередь заданий, или не применит использующую их функцию, такую как новые диагностические возможности Oracle 10*g*.

Эти процессы будут отображаться в среде UNIX точно так же, как любые другие фоновые процессы — в выводе команды `ps`. Обратившись к результатам выполнения команды `ps` приведенным в начале раздела "Специализированные фоновые процессы", можно выяснить, что система содержит перечисленные ниже элементы.

- Сконфигурированные очереди заданий. Процесс `SJQ0` — это координатор очередей.
- Сконфигурированное программное обеспечение Advanced Queuing (AQ), о чем свидетельствует наличие `Q000` (процесс создания очереди AQ) и `QMNC` (процесс монитора AQ).
- Включенная функция автоматического управления памятью, о чем свидетельствует наличие процесса диспетчера памяти (Memory Manager; `MMAN`).
- Включенные функции управления/диагностики Oracle 10*g*, о чем свидетельствует наличие процессов монитора управляемости (Manageability Monitor; `MMON`) и облегченного монитора управляемости (Manageability Monitor Light; `MMNL`).

Рассмотрим различные процессы, которые могут создаваться в зависимости от используемых функциональных средств.

Процессы `CJ00` и `Jnnn`: очереди заданий

В первом выпуске Oracle 7.0 была предоставлена возможность репликации в форме объекта базы данных, называемого *моментальной копией* (или *моментальным снимком*). Очереди заданий служили внутренним механизмом обновления (или преобразования в актуальное состояние) этих снимков.

Процесс очереди отслеживал таблицу заданий, содержащую информацию о том, когда необходимо выполнить обновление различных снимков системы. В версии Oracle 7.1 эта возможность стала общедоступной посредством пакета базы данных `DBMS_JOB`. Таким образом, процесс, который в версии 7.0 был связан исключительно с обработкой снимков системы, в версии 7.1 и последующих превратился в "очередь заданий". С течением времени имена параметров управления поведением очереди заданий изменялись с `SNAPSHOT_REFRESH_INTERVAL` и `SNAPSHOT_REFRESH_PROCESSES` на `JOB_QUEUE_INTERVAL` и `JOB_QUEUE_PROCESSES`, и в настоящее время только параметр `JOB_QUEUE_PROCESSES` доступен для настройки пользователем.

В системе можно использовать до 1000 процессов очередей заданий с именами `J000`, `J001`, ... , `J999`. Эти процессы интенсивно применяются при репликации в качестве составной части процесса обновления материализованного представления. Потоковая репликация (новая функциональная возможность, появившаяся в Oracle9*i* Release 2) применяет для репликации функции AQ и, следовательно, не использует процессы очередей заданий. Разработчики также часто применяют очереди заданий для планирования разовых (фоновых) заданий или заданий по обслуживанию — например, для отправки в фоновом режиме сообщений электронной почты или фонового выполнения длительных пакетных процессов. Выполняя часть действий в фоновом режиме, можно создавать видимость значительно более быстрого выполнения длительных задач (нетерпеливому пользователю может казаться, что задание выполняется быстрее, хотя,

возможно, оно может быть еще не завершено). Это аналогично применению процессов LGWR и DBWn в СУБД Oracle — большую часть своих действий эти процессы проводят в фоновом режиме, поэтому нам не приходится дожидаться, пока они завершат все задания в реальном времени.

Процессы Jnnn (где nnn — число) во многом подобны процессу разделяемого сервера, но обладают некоторыми особенностями выделенного сервера. Они подобны разделяемому серверу в том смысле, что решают одну задачу за другой, но их способ управления памятью больше подобен способу, используемому выделенным сервером (их память UGA размещается в области PGA, а не в SGA). Каждый процесс очереди заданий в каждый конкретный момент времени будет выполнять только одно задание, затем другое и так до полного его завершения. Именно поэтому для одновременного выполнения нескольких заданий может требоваться несколько процессов. Не существует никакого метода организации заданий в потоки или предварительного их удаления. Если уж задание запущено, оно будет выполняться до полного завершения (или сбоя).

Вы заметите, что с течением времени процессы Jnnn появляются и исчезают — т.е., если система сконфигурирована с поддержкой 1000 процессов, то не все 1000 процессов будут запущены вместе с запуском базы данных. Скорее всего, вначале будет запущен только один процесс — координатор очереди заданий (CJQ0), который будет стартовать необходимые процессы Jnnn в соответствии с информацией, записанной в таблице очереди заданий. По мере выполнения своих заданий и при отсутствии новых, процессы Jnnn начнут производить выход — исчезать. Так, если большинство заданий будет запланировано на 2 часа ночи, когда за консолью управления нет ни одного человека, эти процессы Jnnn может никто в действительности и не "увидеть".

QMNC и Qnnn: расширенные очереди

Процесс QMNC играет для таблиц AQ ту же роль, которую процесс CJ00 играет для таблицы заданий. Он отслеживает расширенные очереди и оповещает процессы *извлечения из очереди*, которые ожидают сообщения, о том, что сообщение стало доступным. Процессы QMNC и Qnnn отвечают также за *распространение* в очереди — т.е. за возможность перемещения сообщения, которое было добавлено в очередь в одной базе данных, в очередь в другой базе данных для его последующего извлечения из очереди.

Процессы Qnnn играют для процесса QMNC ту же роль, которую процессы Jnnn играют для процесса CJQ0. Процесс QMNC уведомляет их о задаче, которая должна быть выполнена, а они выполняют эту задачу.

Процессы QMNC и Qnnn — необязательные фоновые процессы. Параметр AQ_TM_PROCESSES определяет создание до десяти таких процессов с именами Q000, ..., Q009 и единственного процесса QMNC. Если значение параметра AQ_TM_PROCESSES установлено в 0, в системе не будет ни одного процесса QMNC или Qnnn. В отличие от процессов Jnnn, используемых очередями заданий, процессы Qnnn являются постоянными. Если значение параметра AQ_TM_PROCESSES установлено в 10, при запуске базы данных в ней будут созданы десять процессов Qnnn и процесс QMNC, которые будут существовать в течение всего времени жизни экземпляра.

EMNn: процессы монитора событий

Процесс EMNn — составная часть архитектуры AQ. Он служит для уведомления клиентов очереди о сообщениях, которые могут их интересовать. Это уведомление производится асинхронно. Доступны функции OCI (Oracle Call Interface — интерфейс уровня вызовов Oracle), позволяющие зарегистрировать обратный вызов для уведомления о сообщении. Обратный вызов — это функция программы OCI, которая будет автоматически вызываться при каждом появлении интересующего сообщения в очереди. Фоновый процесс EMNn служит для уведомления подписчика. Он автоматически запускается при

выдаче для экземпляра первого уведомления. Затем приложение может выдать явную команду `message_receive(dequeue)` для получения сообщения.

MMAN: диспетчер памяти

Этот процесс появился в Oracle 10*g* и присутствует в последующих версиях. Он применяется функцией автоматической настройки размера области SGA. Процесс `MMAN` координирует установку и изменение размеров компонентов разделяемой памяти (буферного пула по умолчанию, а также разделяемого и большого пулов и пула Java).

MMON, MMNL и Mnnn: мониторы управляемости

Эти процессы используются для заполнения автоматического хранилища информации о рабочей нагрузке (Automatic Workload Repository — AWR) — нового функционального средства Oracle 10*g*. Процесс `MMNL` записывает статистические сведения из области SGA в таблицы базы данных в соответствии с установленным расписанием. Процесс `MMON` служит для автоматического обнаружения проблем с производительностью и реализации новых функций самонастройки. Процессы `Mnnn` аналогичны процессам `Jnnn` или `Qnnn` применительно к очередям заданий. Процесс `MMON` будет требовать от этих подчиненных процессов, чтобы они выполняли задачу от его имени. По своей природе процессы `Mnnn` временны — они будут создаваться и прекращать свое существование по мере необходимости.

CTWR: процесс отслеживания изменений

Это новый, необязательный процесс СУБД Oracle 10*g*. Он отвечает за обслуживание нового файла отслеживания изменений, который был описан в главе 3.

RVWR: процесс записи данных восстановления

Этот еще один новый необязательный процесс СУБД Oracle 10*g* обеспечивает хранение *предшествующих* образов блоков в области Flash Recovery Area (область для быстрого восстановления, описанная в главе 3), которая используется при выполнении команды `FLASHBACK DATABASE`.

DMnn/DWnn: ведущие/рабочие процессы Data Pump

Data Pump — это новое средство, добавленное в версию Oracle 10*g* Release 1. Оно было спроектировано как полная замена старого процесса экспорта/импорта. Data Pump выполняется целиком на сервере и API-интерфейсом для взаимодействия с ним служит PL/SQL. Поскольку Data Pump выполняется на сервере, на сервер была добавлена поддержка выполнения различных операций Data Pump. Ведущий процесс Data Pump (`DMnn`) собирает весь ввод от клиентских процессов (принимающих ввод через API-интерфейс) и затем координирует рабочие процессы (`DWnn`), которые выполняют реальную работу; процессы `DMnn` управляют реальной обработкой метаданных и данных.

Прочие служебные фоновые процессы

Итак, является ли приведенный перечень процессов исчерпывающим? Безусловно, нет. Например, технология Oracle Data Guard (Защита данных Oracle) предполагает использование набора процессов, облегчающих доставку информации повторения транзакций из одной базы данных в другую и ее применение (подробнее это описано в документе *Data Guard Concepts and Administration Guide* (Концепции защиты данных и руководство по администрированию), выпущенном Oracle). Существуют также процессы применения и перехвата потоков Streams. Однако приведенный перечень охватывает большинство наиболее часто встречающихся фоновых процессов.

Подчиненные процессы

Теперь можно приступить к рассмотрению последнего класса процессов Oracle — *подчиненных* процессов. Существуют два типа подчиненных процессов Oracle: подчиненные процессы ввода-вывода и подчиненные процессы параллельных запросов.

Подчиненные процессы ввода-вывода

Подчиненные процессы ввода-вывода служат для эмуляции асинхронного ввода-вывода для тех систем или устройств, которые его не поддерживают. Например, лентопротяжные устройства (которые славятся медленной работой) не поддерживают асинхронный ввод-вывод. Используя подчиненные процессы ввода-вывода, для лентопротяжных устройств можно имитировать действия, которые обычно операционная система выполняет для дисководов. Подобно тому, как это выполняется при действительно асинхронном вводе-выводе, процесс, осуществляющий запись на устройстве, объединяет большой объем данных в пакет, а затем передает его для записи. После того, как данные успешно записаны, записывающий процесс (в данном случае подчиненный процесс ввода-вывода, а *не* операционная система) сигнализирует об этом процессу, инициировавшему запись, который удаляет этот пакет из своего списка данных, предназначенных для записи. Это позволяет существенно увеличить пропускную способность, поскольку дожидаться завершения работы медленного устройства приходится подчиненным процессам ввода-вывода, в то время как вызывающий их процесс занят другой важной задачей сбора данных для следующей записи.

Подчиненные процессы ввода-вывода используются рядом компонентов Oracle. Процессы `DBWn` и `LGWR` могут их применять для эмуляции асинхронного ввода-вывода, а процесс `RMAN` будет их использовать при записи на магнитную ленту.

Для управления использованием подчиненных процессов ввода-вывода служат два параметра.

- `BACKUP_TAPE_IO_SLAVES`. Этот параметр указывает, использует ли RMAN подчиненные процессы ввода-вывода для резервирования, копирования или восстановления данных с магнитной ленты. Поскольку этот параметр предназначен для работы с *ленточными* накопителями, а ленточные накопители доступны только одному процессу в каждый конкретный момент времени, его значение является булевским, а не количеством используемых подчиненных процессов, как можно было бы предположить. Процесс RMAN запустит столько подчиненных процессов, сколько требуется для данного количества используемых физических устройств. Когда значение `BACKUP_TAPE_IO_SLAVES` установлено в `TRUE`, подчиненный процесс ввода-вывода используется для записи на ленточный накопитель или чтения с него. Если же оно установлено в `FALSE` (значение по умолчанию), подчиненные процессы ввода-вывода не применяются для выполнения резервного копирования. Вместо этого к ленточному накопителю будет обращаться процесс выделенного сервера, задействованный в резервировании.
- `DBWR_IO_SLAVES`. Этот параметр указывает количество подчиненных процессов ввода-вывода, используемых процессом `DBW0`. Процесс `DBW0` и его подчиненные процессы всегда выполняют запись грязных блоков буферного кэша на диск. По умолчанию значение этого параметра установлено в `0`, и подчиненные процессы ввода-вывода не используются. Обратите внимание, что при указании для этого параметра ненулевого значения процессы `LGWR` и `ARCH` будут применять также собственные подчиненные процессы ввода-вывода — они могут использовать до четырех подчиненных процессов ввода-вывода.

Подчиненные процессы ввода-вывода процесса DBWR отображаются с именем I1nn, а подчиненные процессы ввода-вывода процесса LGWR — с именем I2nn.

Pnnn: серверы выполнения параллельного запроса

В Oracle 7.1.6 была введена функциональная возможность параллельного запроса базы данных. Она позволяет создавать для SQL-оператора, такого как SELECT, CREATE TABLE, CREATE INDEX, UPDATE и т.п., план выполнения, состоящий из *множества* планов выполнения, которые могут быть сделаны одновременно. Результаты выполнения всех этих планов объединяются в один общий результат. Цель такого подхода — сокращение времени выполнения операции по сравнению с последовательным ее выполнением. Например, предположим, что существует действительно большая таблица, распределенная по десяти различным файлам. Предположим также, что в нашем распоряжении имеется 16 процессоров и нужно выполнить какой-либо особый запрос этой таблицы. Вероятно, имеет смысл разбить план выполнения на 32 небольших фрагмента и действительно в полной мере задействовать вычислительные возможности компьютера, вместо того чтобы применять только один процесс для последовательного чтения и обработки всех данных.

При использовании параллельного запроса создаются процессы Pnnn — это собственно серверы выполнения параллельного запроса. Во время обработки параллельного оператора серверный процесс известен под названием *координатора параллельного запроса*. Его название не изменяется на уровне операционной системы, но в документации по параллельным запросам он упоминается именно так.

Резюме

Мы рассмотрели файлы, используемые СУБД Oracle — начиная с простого, но важного файла параметров, и заканчивая файлами данных, журнальными файлами и т.д. Вы подробно ознакомились со структурами данных, используемыми Oracle как в серверных процессах, так и в области SGA. В главе было показано, что различные конфигурации серверов, такие как подключение в режиме выделенного и разделяемого сервера, оказывают решающее влияние на использование памяти системой. И, наконец, мы рассмотрели процессы (или потоки — в зависимости от конкретной операционной системы), позволяющие СУБД Oracle решать стоящие перед ней задачи. Теперь можно приступать к исследованиям реализации ряда других функциональных средств Oracle, таких как блокировка, средства управления параллельным доступом и транзакции.