7

Les processus d'Oracle 10*g*

Dans ce chapitre :

- les threads Oracle ;
- les threads indispensables ;
- les threads optionnels ;
- les threads utilisateurs.

Le chapitre 5, *Oracle 10*g *sous Windows*, a présenté le processus Oracle 10*g*. Ce chapitre est maintenant consacré aux threads du processus oracle.exe, qui assurent le fonctionnement d'une instance Oracle 10*g*. Les prochains chapitres vont présenter tour à tour les principaux composants d'une base Oracle 10*g* : fichiers, threads et espaces mémoire. Ici, nous entrons dans le détail du fonctionnement interne de chacun de ces threads, pour vous permettre de mieux comprendre son rôle, son impact sur le fonctionnement d'une instance et les performances de vos bases Oracle 10*g*.

Les threads indispensables, optionnels et utilisateur

Le fonctionnement d'une base Oracle est assuré par un ensemble de threads imbriqués qui réalisent de nombreuses actions. Pour plus de simplicité, nous avons regroupé les threads en trois familles : les indispensables, les optionnels et les threads utilisateur.

Les threads indispensables sont présents dès qu'une base Oracle fonctionne. Ils sont requis pour en assurer le fonctionnement minimal. Si l'un d'eux s'arrête, la base de données n'est plus opérationnelle.

D'autres threads peuvent être lancés pour assurer des fonctions complémentaires, comme la réplication ou la sauvegarde automatique de vos fichiers redo-log. Si l'un de ces threads optionnels n'est pas démarré, cela ne met pas en cause le fonctionnement global de la base de données. Seule la tâche assurée par ce threads optionnel ne sera pas réalisée.

Les threads utilisateur sont mis en œuvre dès qu'un utilisateur ouvre une connexion avec sa base Oracle. Leur existence et leur utilité sont très souvent méconnues.

Les threads indispensables

Pour un fonctionnement minimal, Oracle utilise 4 threads :

- DBWR (*Database Writer*) ;
- LGWR (*Log Writer*) ;
- CKPT (*Checkpoint*) ;
- PMON (*Process Monitor*) ;
- SMON (*System Monitor*).

La figure suivante décrit les liens inter-threads de l'architecture d'une instance Oracle 10*g*. Ce sont ces threads qui relient les fichiers de la base de données, la zone mémoire réservée à l'instance SGA (System Global Area), ainsi que la mémoire allouée à chaque connexion utilisateur PGA (Process Global Area)Nous commenterons largement cette figure tout au long de ce chapitre.

Ces threads correspondent à une instance unique. Comme présenté au chapitre 5, *Oracle 10*g *sous Windows*, on peut les visualiser à l'aide de l'*Oracle Administrative Assistant for Windows* ou encore depuis Oracle Enterprise Manager.
Dans ce chapitre, nous traitons chaque thread séparément. La tâche est difficile puisqu'ils sont très imbriqués et les dépendances mutuelles sont nombreuses.

Le threads DBWR

Le threads DBWR (*Database Writer* ou *Dirty Buffer Writer*) transfert les blocs de données modifiés ou « sales » (« dirty buffers ») du buffer mémoire de la SGA dans les fichiers disque de la base de données. Dès qu'un ordre SQL de type INSERT, UPDATE, DELETE intervient, il travaille prioritairement avec les buffers de données en mémoire, pour plus de performance. Les données non modifiées sont copiées dans la zone d'annulation (ou UNDO) de la SGA et les données modifiées dans la zone des buffers de données. La zone mémoire redo-log (destinée à être traitée par le Log Writer) est elle aussi modifiée.

En cas de validation de transaction (ordre COMMIT), le DBWR libère la place des enregistrements dans le Segment UNDO. S'il y a annulation de transaction (ou ROLLBACK), les enregistrements non modifiés lus dans le UNDO Segment supprimeront les enregistrements modifiés en buffer de données et les rétabliront en leur état initial.

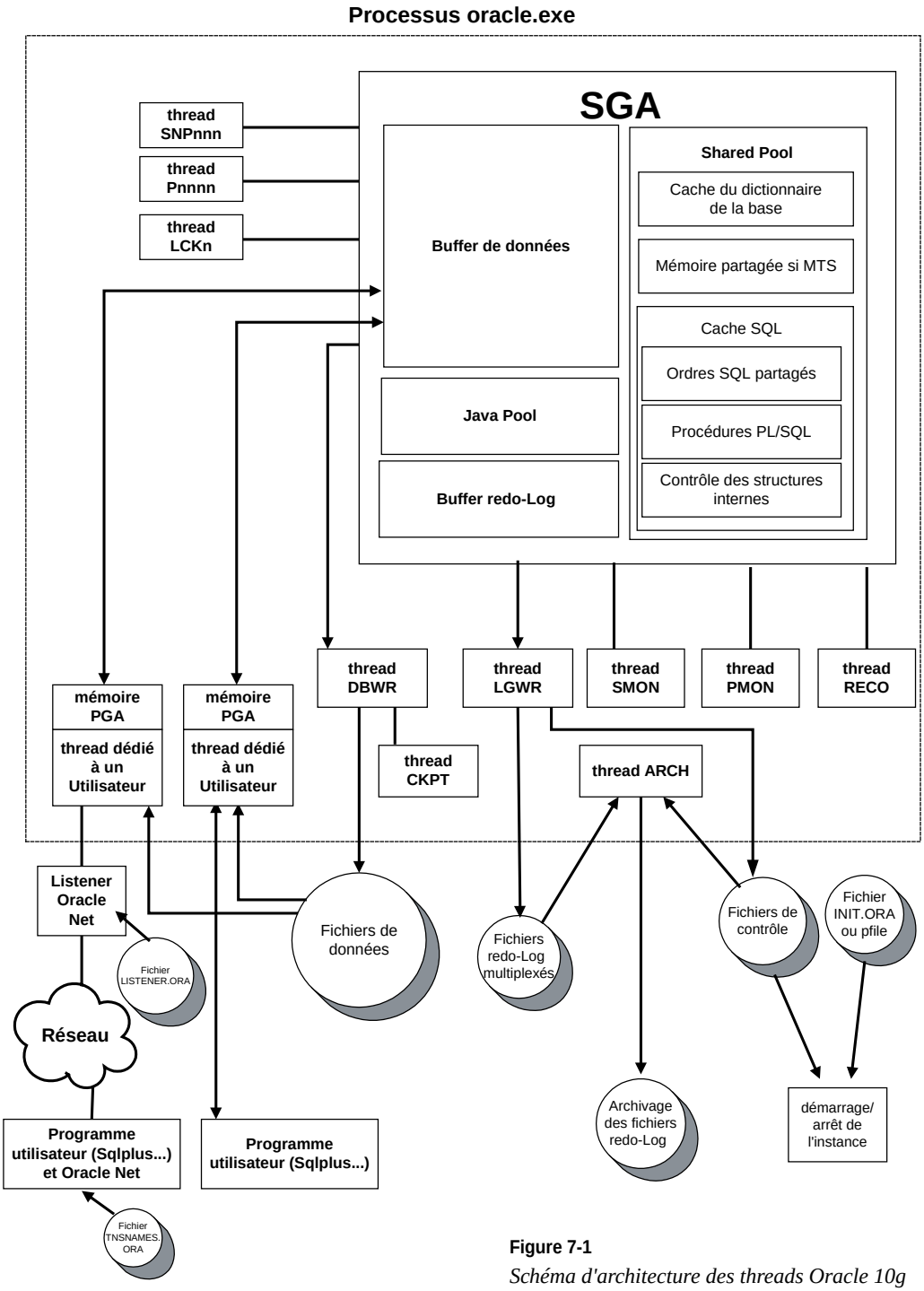


Figure 7-1
Schéma d'architecture des threads Oracle 10g

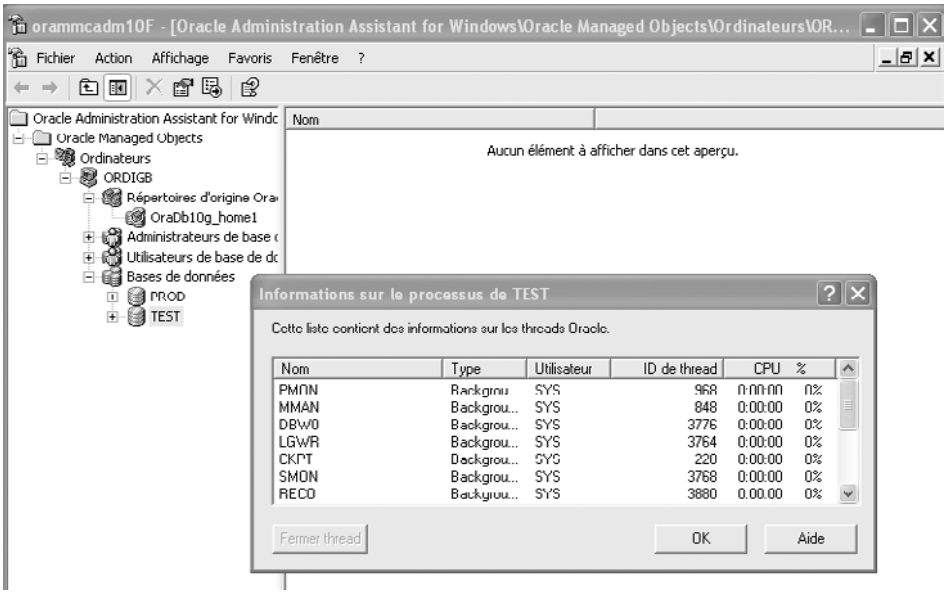


Figure 7-2
Les threads d'un processus

Le rôle de DBWR est de vérifier qu'il y a toujours suffisamment de buffers libres en mémoire. Comme le nombre de blocs existants en mémoire est fixe, chaque buffer modifié (ou « sale ») diminue le nombre de buffers libres disponibles. Dans ce cas, si un ordre SQL a besoin de lire des données depuis les fichiers de données pour les placer dans les buffers de données mémoire, un algorithme LRU (Least Recently Used) écrit les plus anciennes données modifiées sur le disque pour libérer de la place mémoire.

Pour optimiser les entrées/sorties disque, les écritures dans les fichiers de la base de données sont différées. La sécurité des transactions est assurée par les fichiers redo-log et le threads LGWR.

Le comportement de DBWR est contrôlé par le paramètre d'initialisation DB_WRITERS, qui permet de démarrer plusieurs threads DBWR, afin d'augmenter le taux d'écriture sur disque dans les systèmes très fortement sollicités. Nous vous conseillons de démarrer une base Oracle 10*g* avec un seul thread DBWR et d'en augmenter progressivement le nombre dans les systèmes très sollicités en entrées/sorties disque.

> Lors de tous les transferts entre la mémoire et les fichiers présents sur le disque, la plus petite unité de transfert est le « bloc Oracle » dont la taille est fixée lors de la création de chaque base de données. En effet, le système d'exploitation accède aux disques par des « unités élémentaires » qui lui sont propres. Ce bloc permet une cohérence entre les blocs manipulés par Oracle et les unités élémentaires manipulées par le système d'exploitation. Dans beaucoup de systèmes, l'unité d'échange de l'OS est 512 octets, et celle d'Oracle est un multiple de cette valeur: 2048, 4096, etc. Le chapitre 28, *Optimisation et performances traite de l'influence du bloc Oracle sur les performances*.

Les threads LGWR et CKPT

Les fichiers redo-log garantissent la préservation des données validées, même si la base est arrêtée brutalement (coupure électrique par exemple). Le thread LGWR a pour mission d'écrire toutes les informations utiles à cette sécurité dans les fichiers redo-log.

Dès qu'une transaction est validée, Oracle écrit les données modifiées à deux emplacements différents, de façon à pouvoir « repartir » si un problème matériel survient. La première copie vue précédemment est assurée par le Database Writer dans les fichiers contenant les données. Cette copie n'est par forcément immédiate : pour augmenter les performances et éviter des goulots d'étranglement, un délai d'écriture peut exister. Pour conserver l'intégralité des données présentes en mémoire mais en attente d'écriture sur disque, une seconde copie immédiate est assurée par le Log Writer dans les fichiers redo-log.

Dès qu'un COMMIT ou un ROLLBACK intervient, le thread LGWR (Log Writer) écrit immédiatement les données modifiées depuis la zone mémoire redo-log dans les fichiers redo-log, suivi de l'ordre de validation (COMMIT) ou d'annulation (ROLLBACK). Ainsi, toute modification validée ou annulée est immédiatement écrite sur le disque, puis la zone mémoire redo-log occupée est libérée.

À intervalles réguliers, toutes les données modifiées et présentes dans la SGA sont écrites dans les fichiers de la base de données par le thread DBWR. Cet événement se nomme un *checkpoint*. Le thread CKPT signale les checkpoints au thread DBWR et modifie l'ensemble des fichiers qui composent la base de données, pour que le numéro d'ordre du plus récent checkpoint soit inscrit en en-tête de fichier.

Le thread System Monitor

Le thread SMON (System Monitor) surveille la base de données lors de son démarrage puis au cours de son fonctionnement.

La principale fonction du thread SMON a lieu lors du démarrage de la base de données : il vérifie si le dernier arrêt a été correctement effectué. Si tel est le cas, il ne fait rien. Mais en cas d'arrêt brutal, il existe certainement des transactions en cours qui n'ont été ni validées, ni invalidées. SMON lit alors les informations contenues dans les UNDO segments (l'emplacement où sont conservées les données en attente de validation) puis les annule. En ce qui concerne les enregistrements validés, SMON récupère dans les fichiers redo-log ceux qui ont été modifiés (par un COMMIT ou un ROLLBACK) mais n'ont pas encore été écrits dans la base Oracle, et ce afin de les y insérer. Les autres enregistrements présents dans les fichiers redo-log, mais qui n'ont pas été validés ou annulés explicitement, sont alors annulés par SMON. Enfin, SMON libère toutes les ressources de la base de données (verrous, segments temporaires...).

Lorsqu'elle est en cours de fonctionnement, SMON surveille l'activité de la base. Il recycle les segments temporaires (utilisés par exemple lors des tris de requêtes SQL) devenus inutiles. Tous les tris ne pouvant avoir lieu en mémoire, c'est SMON qui assure ce rôle important : ainsi, les nouvelles transactions ayant besoin d'espace de tri disposent d'un maximum de place.

SMON vérifie aussi que des espaces libres subsistent dans les fichiers de la base de données. Dans ce cas, il essaie de les regrouper. En effet, même si vous disposez de beaucoup de place dans un fichier de la base, cet espace peut ressembler à du « gruyère ». Par exemple, il n'est pas possible de créer une table de 10 Mo, si le plus grand espace libre contigu est inférieur à cette taille (il existe des contournements). Cependant, les capacités de SMON étant limitées, vous devez parfois effectuer un Export/Import pour réorganiser physiquement le contenu de vos fichiers et regrouper l'ensemble des espaces libres en un seul espace contigu.

SMON intervient aussi dans le fonctionnement en mode Parallel Server ou cluster. Il vérifie que l'ensemble des bases de données composant le cluster sont actives.

SMON peut aussi être appelé par d'autres threads. Ainsi, si le thread Database Writer a besoin d'espace disque pour effectuer un tri, il demande à SMON s'il peut en libérer.

Le fonctionnement de SMON est automatique : aucune action de l'administrateur de la base de données n'est requise. C'est l'un des points forts d'Oracle par rapport à ses concurrents.

Si SMON s'arrête, il faut redémarrer l'instance Oracle.

Le thread Monitor

Le thread PMON (*Process Monitor*) nettoie les transactions défaillantes, comme celle d'un poste distant arrêté brutalement durant une transaction. Ce nettoyage libère les zones mémoire allouées, supprime les verrous posés par les transactions et annule les ressources affectées aux threads de la transaction.

Le rôle de PMON est très important si vous utilisez un système comportant de nombreux utilisateurs ou encore si vous effectuez des requêtes lourdes. Chaque connexion à une base Oracle consomme quelques mégaoctets de mémoire et du temps processeur. Si un utilisateur arrête brutalement son PC au cours d'une longue requête SQL, il peut ainsi bloquer inutilement un ensemble de ressources, si PMON ne détecte pas et ne nettoie pas les transactions anormalement interrompues. Ce sont des threads comme PMON qui font la différence entre des systèmes qu'il faut fréquemment « rebooter » et d'autres, stables dans le temps.

Si PMON s'arrête, il faut redémarrer l'instance Oracle. Dans ce cas, SMON annulera (ordre ROLLBACK) toutes les transactions en attente de validation qu'il trouvera dans les buffers de données.

Les threads optionnels

Les threads détachés optionnels sont les suivants :

- Listener ou listener Net ;
- ARCH ou Archiver ;
- RECO ou Recover ;
- SNPnn ou Snapshot ;

- Dnnnn et Snnnn Dispatcher et Server ;
- Pnnnn ou Parallel Query Slave.

Le processus listener Oracle Net

Le listener Oracle Net n'est généralement pas compris dans la liste des processus indispensables au fonctionnement d'Oracle. Pourtant, il doit être lancé pour permettre d'établir des connexions client-serveur avec la base de données.

Son lancement n'est pas lié à celui de la base de données et il possède ses propres fichiers de configuration. Son fonctionnement sous forme d'un service Windows est détaillé au chapitre 15, *Oracle Net, le middleware Oracle*.

Le thread ARCH

Le thread ARCH (Archiver) n'existe que si vous utilisez la base de données en mode ARCHIVELOG. Il est responsable de la copie des fichiers redo-log, lorsqu'ils sont saturés, vers leur destination de stockage (disque ou bande). Il faut bien distinguer le thread ARCH responsable de la copie, de l'ordre à donner au niveau de la base de données pour lui indiquer qu'elle est en mode ARCHIVELOG et que les redo-log seront conservés en historique. Ce sont deux étapes séparées.

> Il est très dangereux de se fier à la seule présence du thread ARCH pour en conclure que la base est en mode ARCHIVELOG.

Si la destination de stockage est saturée et ne peut accueillir de nouveaux fichiers, la base de données se bloque. En effet, Oracle considère qu'il vaut mieux bloquer une base de données active plutôt que renoncer à un moyen de sauvegarde déterminant sans en avertir l'administrateur. Lorsqu'elle se bloque, la base de données est « silencieuse ». Seules de nouvelles connexions ou l'exploration des fichiers d'alerte de la base de données signalent cette situation. En revanche, si l'on place la base en mode ARCHIVELOG sans démarrer le thread ARCH, la base de données se bloquera dès qu'elle aura besoin « d'historiser » un fichier redo-log saturé. Un message sera alors écrit dans le fichier d'alerte.

Pour démarrer le thread ARCH, le paramètre d'initialisation ARCHIVE_LOG_START doit avoir pour valeur TRUE et la base de données doit être en mode ARCHIVELOG.

Le thread RECO

Le thread RECO (Recover) intervient uniquement si vous utilisez la réplication ou base de données distribuée. Dans ce cas, il faut que l'option « distributed database » soit installée dans la base.

> La liste des options disponibles installées dans la base de données s'affiche lors de chaque connexion à SQL*Plus. Vous pouvez aussi interroger la vue V$VERSION.

Le thread RECO est lancé si le paramètre d'initialisation de l'instance DISTRIBUTED _TRANSACTIONS est supérieur à zéro (pour les bases de données distribuées, sa valeur par défaut est 32).

Le thread SNPnn

Le thread SNPnn (*Snapshot*, *nn* représentant un nombre entier) était initialement utilisé pour le traitement des snapshots ou clichés. Lorsqu'une demande de cliché est placée en file d'attente dans la base de données, le thread SNPnn est chargé de le traiter.

Un thread SNPnn défaillant empêche le traitement des clichés et génère un message dans le fichier d'alerte. Le fonctionnement global de la base de données n'est pas affecté par l'arrêt d'un thread SNPnn.

Les threads SNPnn sont contrôlés par les paramètres d'initialisation de l'instance : JOB_QUEUE_PROCESSUSES, JOB_QUEUE_INTERVAL et JOB_QUEUE_KEEP _CONNECTIONS (pour certains portages, JOB peut être remplacé par SNAPSHOT). JOB_QUEUE_PROCESSUSES détermine le nombre de threads SNPnn démarrés au lancement de l'instance. JOB_QUEUE_INTERVAL définit la fréquence de réveil des threads SNPnn qui vont scruter la file d'attente des clichés à diffuser. JOB_QUEUE _KEEP_CONNECTIONS conserve les liens avec des bases de données distantes (database links) ouvertes.

Les threads Dnnnn et Snnnn

Les threads Dnnnn (Dispatcher) et Snnnn (Server) avec *nnnn*, une suite de nombres entiers, n'interviennent que pour des systèmes ayant mis en place une configuration MTS (Multi-Thread Server). Les threads Dnnnn distribuent les demandes de connexion des postes client-serveur distants aux serveurs Snnnn.

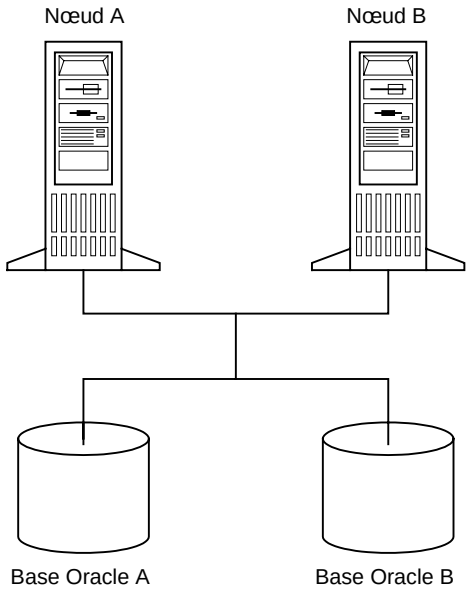
Dans un système MTS, le listener Net détermine si une demande de connexion d'un utilisateur distant doit utiliser un thread dédié ou partager un thread Snnnn avec d'autres utilisateurs.

Les threads Dnnnn et Snnnn sont contrôlés par les paramètres d'initialisation de l'instance : MTS_DISPATCHERS, MTS_MAX_DISPATCHERS, MTS_SERVER et MTS_MAX _SERVER. MTS_DISPATCHERS et MTS_SERVER déterminent respectivement le nombre de threads Dnnnn et Snnnn lancés au démarrage de l'instance. Le système s'allouant automatiquement de nouveaux threads Dnnnn et Snnnn, les deux autres paramètres fixent les seuils à ne pas dépasser.

Le thread Pnnnn

Le thread Pnnnn (*Parallel Query Slave*) avec *nnnn*, une suite de nombres entiers, est automatiquement lancé et arrêté si la base Oracle est utilisée dans un environnement Oracle *Real Application Clusters* (RAC) ou cluster de base de données. Dans cette configuration, des bases hébergées sur des machines différentes constituent un « tout » logique.

Figure 7-3
Principe d'un cluster de bases Oracle 10g



Ces configurations très élaborées, présentées au chapitre 6, *Les clusters Oracle*, sont destinées à des applications nécessitant une haute disponibilité ou de hautes performances.

Modes single-task et two-tasks

Suivant les systèmes d'exploitation, Oracle fonctionne en mode *single-task* ou en mode *two-tasks*. Ces modes de fonctionnement sont internes aux produits Oracle et n'ont aucune incidence sur votre façon de travailler ou de développer des applications.

En mode *single-task*, le noyau Oracle, le programme utilisateur et le programme interface qui les lient, fonctionnent dans le même thread. C'est le cas de VAX VMS (que l'on rencontre de moins en moins souvent) et de Windows : lorsque de nouveaux utilisateurs se connectent à la base Oracle, ils donnent naissance à des « threads » supplémentaires qui s'exécutent au sein du même processus.

En mode *two-tasks*, le noyau Oracle, le programme utilisateur et le programme interface qui les lient, fonctionnent dans des processus différents. C'est le cas de Linux et de l'ensemble des Unix : lorsque des utilisateurs supplémentaires se connectent à la base Oracle, ils engendrent de nouveaux processus, appelés *shadow processus*. Ils possèdent une « vie » indépendante de celle des autres processus.

L'approche est donc différente pour visualiser les caractéristiques (mémoire utilisée, pourcentage de CPU…) d'un processus utilisateur sous Windows et sous Linux. Dans le premier cas, il faut explorer les threads d'un processus, dans l'autre, il faut consulter les processus gérés par le système d'exploitation.

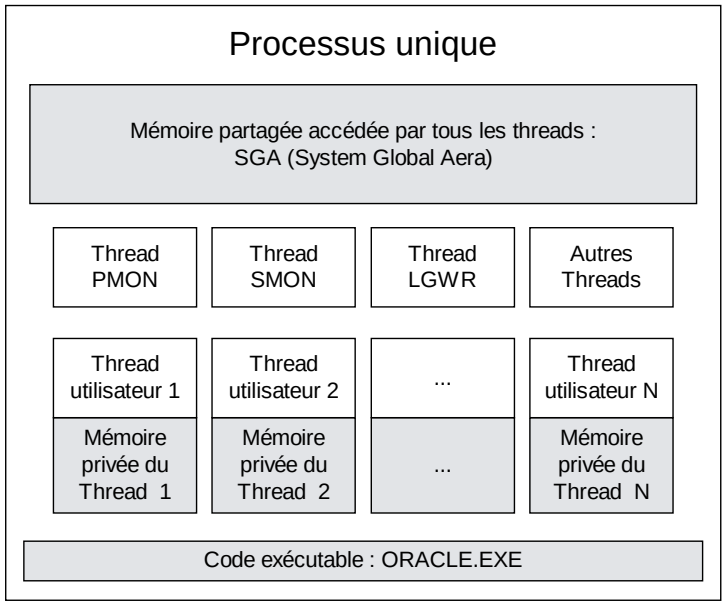


Figure 7-4

Principe du mode single-task : des threads au sein d'un processus

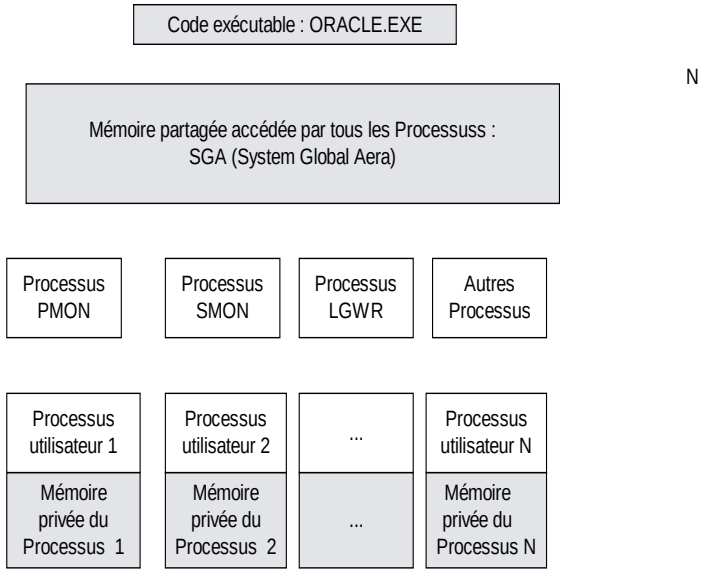


Figure 7-5

Principe du mode two-tasks : des processus coordonnés

Les threads utilisateur

Ce sont les plus méconnus, dans la mesure où ils sont intégralement gérés par la base Oracle. Leur rôle est fondamental. Dès que l'application demande l'accès à l'instance, ce sont eux qui vérifient que les données sont présentes en mémoire. Dans l'affirmative, le thread utilisateur traite vos ordres SQL. Dans la négative, les threads utilisateur lisent les fichiers de la base de données pour les « monter » en mémoire SGA, avant de traiter l'ordre SQL.

Ce n'est pas un thread permanent de l'instance qui assure la lecture des données contenues dans vos fichiers, mais le thread utilisateur créé lors de chaque connexion. Il assure le lien entre le programme de l'utilisateur, les threads de la base de données, la SGA et les fichiers qui composent la base.

Ce thread peut prendre plusieurs formes suivant l'architecture retenue. Étudions le thread utilisateur dans les cas suivants :

- configuration sans réseau ;
- configuration réseau en mode client-serveur ;
- configuration MTS Multi-Thread Server.

Configuration sans réseau

C'est le type de configuration la plus simple, celle que vous utilisez si la base de données et l'application sont situées sur la même machine. Dans ce cas, le thread utilisateur est en lien direct avec l'application.

C'est l'application qui demande à se connecter à l'instance créant le thread utilisateur. Il existe un thread utilisateur par application connectée à l'instance.

Chaque thread utilisateur est responsable de la lecture des données. Il donne naissance à la PGA (*Program Global Area* ou *Process Global Area*), zone mémoire privée allouée à chaque thread utilisateur.

Cette architecture, présente l'inconvénient de ne pas être extensible. Le programme utilisateur, son thread et la base de données se trouvent sur la même machine. Vous pouvez rapidement atteindre une saturation des ressources du système par accumulation de thread.

Les deux configurations suivantes permettent de lever ces limites, en permettant d'abord au programme utilisateur de fonctionner sur une autre machine (le client-serveur), en mettant ensuite en œuvre l'option Multi-Thread Server (MTS) pour limiter le nombre de threads dédiés à l'utilisateur sur le serveur de données.

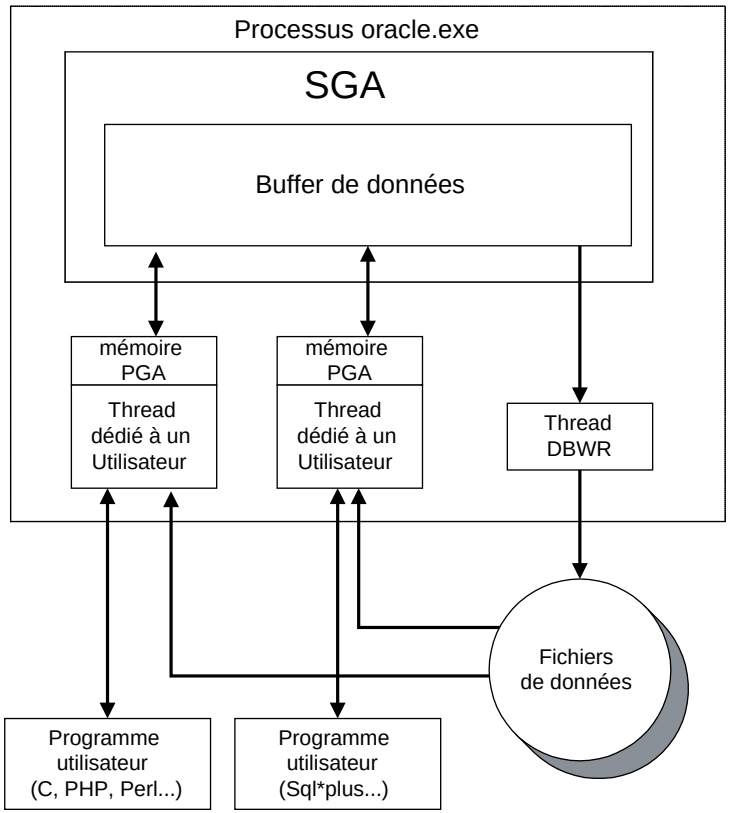


Figure 7-6

Thread utilisateur sans réseau

Configuration réseau en mode client-serveur

En mode client-serveur, le programme utilisateur est situé sur une machine (le client) et la base de données sur le serveur. Le lien entre ces deux programmes est assuré par le middleware de communication Oracle, traité au chapitre 15, *Oracle Net, le middleware Oracle*.

Ce mode introduit le thread Listener Oracle Net qui fonctionne en résident sur le serveur : il est en attente d'une demande de connexion en provenance d'un client. Dès qu'une demande intervient, le Listener Oracle Net crée le thread utilisateur et gère tous les échanges entre le poste client et le serveur.

Cette architecture offre une grande souplesse. Des programmes utilisateur fonctionnant sous des OS différents peuvent accéder à des données situées dans une même base. De plus, en déportant le programme utilisateur sur une autre machine, vous pouvez équilibrer les charges entre vos ordinateurs.

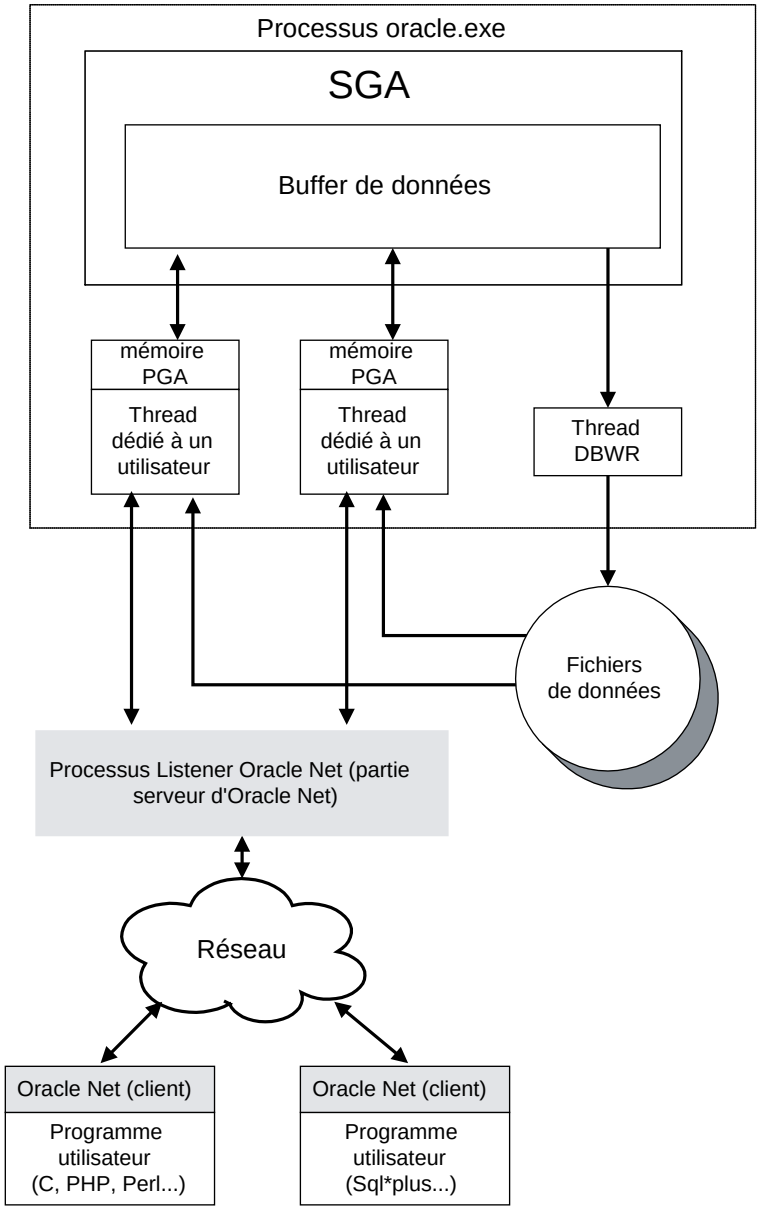


Figure 7-7

Thread utilisateur en mode client-serveur

Configuration MTS (Multi-Thread Server)

Dans les deux configurations précédentes, il y a toujours un thread utilisateur créé par un programme client. Ces threads utilisateur consomment non seulement de la puissance machine (CPU), mais aussi de la mémoire. Pour augmenter le nombre de connexions acceptées par un serveur, la configuration MTS (*Multi-Thread Server*) propose une solution de multiplexage des threads utilisateur.

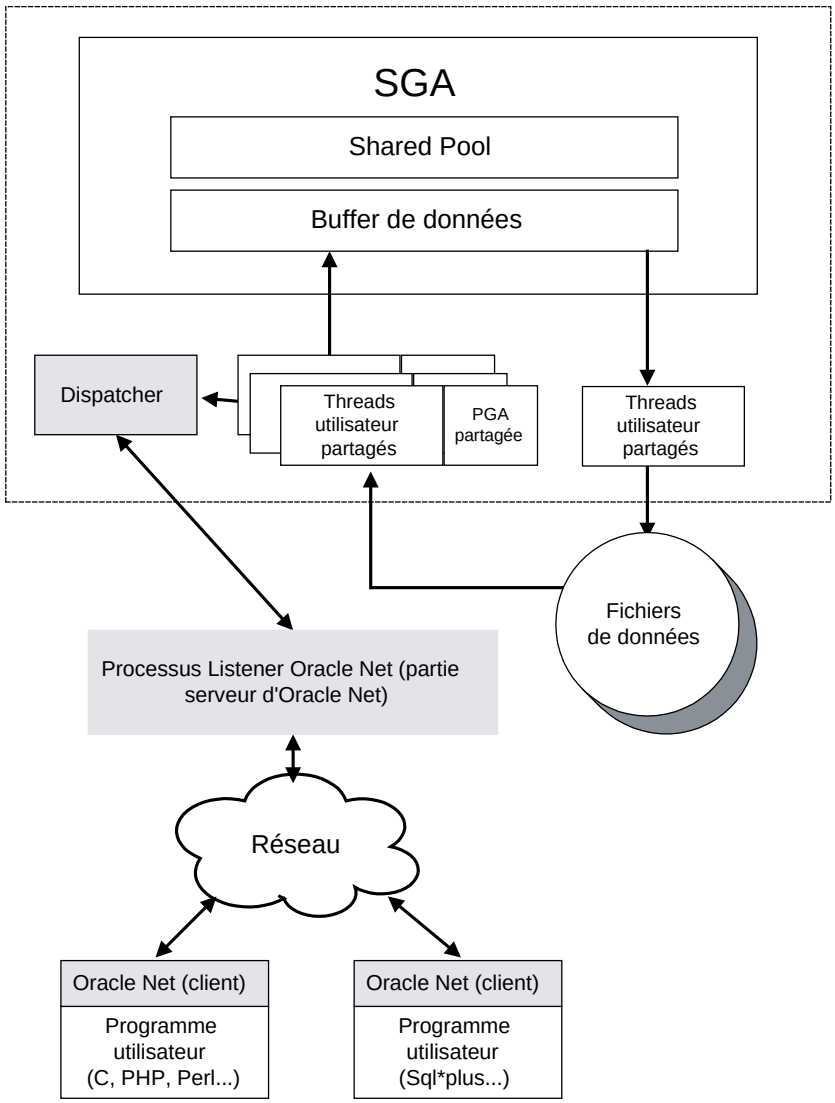


Figure 7-8

Threads utilisateur en configuration MTS (Multi-Thread Server)

Cette configuration économise les ressources serveur grâce à la présence des dispatchers et des threads utilisateur multiplexés. À l'inverse, certaines opérations comme le tri des données sont maintenant partagées en SGA, la taille de la Shared Pool augmente alors fortement.

> Oracle préconise d'utiliser la configuration MTS à partir de 100-150 utilisateurs simultanés.

Résumé

Ce chapitre a détaillé le fonctionnement et l'interaction des différents threads pouvant exister autour d'une base de données Oracle 10*g*. Après une présentation générale, les threads ont été classés en trois familles : les indispensables présents dans chaque base Oracle 10*g* en fonctionnement, les optionnels qui peuvent être mis en place suivant les options retenues et enfin les threads utilisateur.

La connaissance de ces threads permet de mieux comprendre le fonctionnement interne d'Oracle 10*g* afin d'en tirer le meilleur parti et de déterminer quels leviers actionner pour en optimiser le fonctionnement.