【MACLEAN LIU 技术分享】深入理解 ORACLE 中 MUTEX 的内部原理

BY MACLEAN.LIU

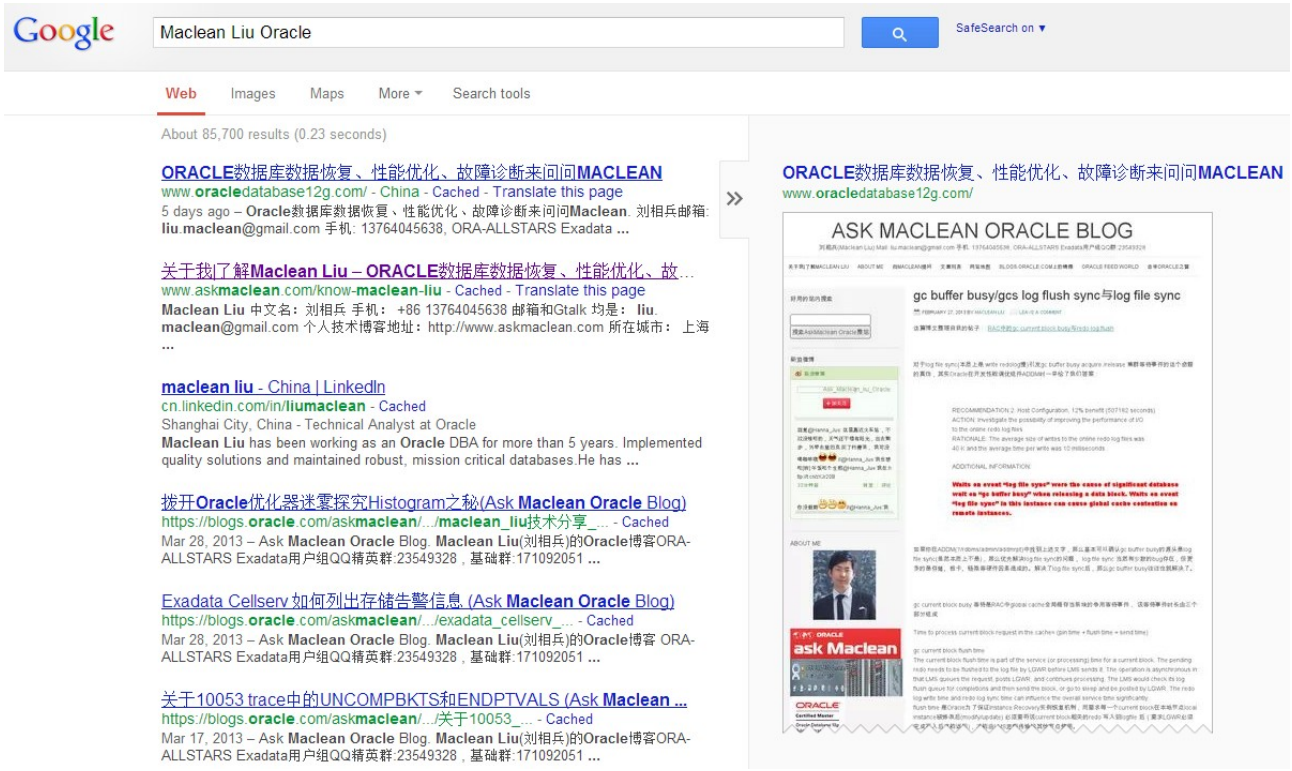
liu.maclean@gmail.com

www.askmaclean.com

About Me

- **Email:liu.maclean@gmail.com**
- **Blog:**www.askmaclean.com
- **Oracle Employee**
- **Oracle Certified Database Administrator Master 10g and 11g**
- **Over 7 years experience with Oracle DBA technology**
- **Over 8 years experience with Linux technology**
- **Member Independent Oracle Users Group**
- **Member All China Users Group**
- **Presents for advanced Oracle topics: RAC, DataGuard, Performance Tuning and Oracle Internal.**

How to Find ME?



了解 Oracle Mutex

虽然 Mutex 中文翻译为互斥锁，但为了和 OS mutex 充分的区别，所以我们在本文里称 Oracle Mutex 为 Mutex。

Oracle 中的 mutex，类似于 Latch，是一种低级的串行机制，用以控制对 SGA 中部分共享数据结构的访问控制。 Oracle 中的串行机制有不少，引入它们的目的是避免一个对象出现下述现象：

- 当某些进程在访问该对象时，该资源被重新分配
- 当某些进程在修改它时，被其他进程读取
- 当某些进程在修改它时，被其他进程修改
- 当某些进程在读取它时，被其他进程修改

不同于 Latch，Mutex 的使用更灵活，用途更多，例如:

- 哪些需要被 mutex 保护的共享数据结构可以有自己独立的 mutex，即一个对象拥有自己独立的 mutex，不像 Latch 往往一个需要保护大量对象，举例来说，每一个父游标有其对应的 mutex， 而每一个子游标也有其对应的 mutex
- 每一个数据结构可能有一个或多个 mutex 保护，每一个 mutex 负责保护其结构的不同部分
- 当然一个 mutex 也可以用来保护多于一个的数据结构

理论上 mutex 即可以存放在其保护的结构本身中(其实是嵌入在结构里)，也可以存放在其他地方。 一般情况下 Mutex 是在数据结构需要被保护时动态创建出来的。 如是嵌在需要保护结构体内的 mutex，则当所依附的数据结构被清理时 该 mutex 也将被摧毁。

Mutex 带来的好处

虽然 mutex 和 latch 都是 Oracle 中的串行机制，但是 mutex 具有一些 latch 没有的好处

更轻量级且更快

Mutex 作为 Latch 的替代品，具有更快速获得，更小等优势。 获取一个 mutex 进需要大约 30~35 个指令， 而 Latch 则需要 150~200 个指令。一个 mutex 结构的大小大约为 16 bytes，而在 10.2 版本中一个 latch 需要 112 个 bytes，在更早的版本中是 200 个 bytes。 从 200 个 bytes 精简到 112 个是通过减少不必要的统计指标 SLEEP1~SLEEP11、WAITERS_WOKEN, WAITS_HOLDING_LATCH 等从而实现的。今后我们将看到更多关于 Latch 的代码优化。

减少伪争用

典型情况下一个 Latch 保护多个对象。 当一个 Latch 保护多个热对象时，并行地对这些对象的频繁访问让 latch 本身变成性能的串行点。 这也就是我们此处说的伪争用点， 因为争用是发生在这个串行保护的机制上，而不是进程去访问的对象本身。与 latch 不同， 使用 mutex 的情况下 Oracle 开发人员可以为每一个要保护的数据结构创建一个独立的 mutex。 这意味着 Latch 的那种伪争用将大大减少，因为每一个对象均被自己独立拥有的 mutex 保护

Mutex 在一些地方替代了 latch 和 PIN

一个 Mutex 可供多个 Oracle 进程并行地参考，反过来说进程们可以以 S(Shared 共享) mode 模式参考一个 Mutex。以 S mode 一起共享参考这个 mutex 的进程的总数成为参考总数 reference count。Mutex 自身结构中存放了这个 ref count 的数据。另一方面，mutex 也可以被以 X (Exclusive)mode 排他模式被仅有一个进程所持有 Held。

Mutex 有 2 种用途，一方面他们可以充当维护必要串行机制的结构，如同 latch 那样； 同时也可以充当 pin，避免对象被 age out。

举例来说，mutex 结构中包含的 ref count 信息可以用作替代 library cache pin。 在 mutex 充当 cursor pin 之前，当一个进程要执行=>pin 一个 cursor 时需要做的是针对性地创建 library cache pin 和删除这个 library cache pin(均为 S mode)。mutex 充当 cursor pin 之后，进程只需要增加和减少 mutex 上的 ref count 即可。
当某一个进程首次解析一个游标 Cursor，他将临时创建并移除一个 library cache pin，但是该进程后续的解析或执行进要求增加或者减少 ref count。注意在这个增加/减少 ref count 的过程中无需 acquire latch，这是因为 mutex 自身能起到限制串行访问修改 ref count 的作用。 当一个进程要移除自己的 mutex pin 时，它减少 ref count，同样的无需 acquire 任何 latch。

Mutex 和 Latch 的交互

Latches 和 Mutex 是独立的串行机制， 举例来说一个进程可以同时持有 latch 和 mutex。 在进程异常 dead 的情况下，一般 latch 要比 Mutex 更早被 PMON 清理。 一般情况下不存在 mutex 的死锁。 不像 latch，在早期版本例如 9i 之前我们经常遇到 latch 死锁的问题。

Mutex 的用途

在版本 10.2 中仅仅有 KKS 这个内核层是 mutex 的客户，KKS 意为 Kernel Kompile Shared， 它是 Library Cache 中的 shared cursor 游标共享部分层次的代码。 在之后的版本中，ORACLE 开发部门更多地使用了 Mutex，不局限于 KKS。

KKS 游标共享如何使用 Mutex

kks 使用 mutex 以便保护对于下述基于 parent cursor 父游标和子游标 child cursor 的一系列操作。

对于父游标 parent cursor 的操作：

- •基于发生的不同操作，对应不同的等待事件：
  - •在某个父游标名下创建一个新的游标 ==> cursor:mutex X
  - •检查一个父游标 ==> cursor:mutex S
  - •绑定值捕获 ==> cursor:mutex X
- •保护父游标的 mutex 嵌入在父游标结构内
- • 针对父游标 parent cursor 的 Mutex 类型为'Cursor Parent' (kgx_kks2).
- • 针对父游标 parent cursor 的 Mutex 等待事件均为' Cursor: mutex *'的形式

针对游标统计信息的操作

- •基于对不同的游标统计信息的操作有不同的等待事件：
  - •构造，更新游标相关的统计信息 ==> cursor:mutex X
  - •检测游标相关的统计信息，例如访问 V$SQLSTATS ==> cursor : mutex S
- •相关的游标可能在父游标中，也可能在游标统计信息相关的 hash table 上
- •针对游标统计信息的 Mutex 类型为 Cursor Stat (kgx_kks1)
- • 针对游标统计信息的 Mutex 等待事件均为' Cursor: mutex *'的形式

Mutex 是如何替代 library cache pin 来保护 cursor heap 的?

- •传统的'library cache pin'在 10.2.0.2 之后默认被取代， 此处 PIN 被 Mutex 及其 ref count 取代。 当进程执行游标语句时或者需要 PIN，或者需要 hard parse 一个子游标 heap。
- •在版本 10.2.0.1 中， 使用 mutex 部分代码替代 PIN 的功能默认是不激活的，实际上这取决于隐藏参数_KKS_USE_MUTEX_PIN，在 10.2.0.2 之后_KKS_USE_MUTEX_PIN 默认为 TRUE。 换而言之在版本 10.2 中我们还是可以关闭 KKS 使用 MUTEX 替代 PIN 保护 CURSOR 的， 但是在版本 11g 中则几乎无法关闭 MUTEX。 注意 10.2 中仅当 KKS 真正使用 MUTEX 时，library cache pin 不再用作 cursor pin。
- •基于对不同的游标统计信息的操作有不同的等待事件：
  - •为执行某个 SQL 而 PIN 一个游标 Cursor ==>Cursor: Pin S Wait on X
  - •当执行一个游标而 PIN Cursor，而该 Cursor 正被其他进程以 S mode 检测 ==> cursor:pin S
- •当试图重建一个游标 Cursor ==> Cursor: pin X 该等待事件一般不太会看到，因为当一个游标正被执行，且其需要重建时会有另一个游标被创建
- •保护游标的 mutex 嵌入在游标结构内
- •Mutex 类型为'Cursor Pin' (kgx_kks3)
- •等待事件均为 'cursor: pin *'的形式

KKS 使用 MUTEX 情况下 SQL 语句的 解析与执行的收益

在版本 10.2 中， 以下是几个 SQL 解析与执行从 MUTEX 哪里获得主要收益：

- •在某个父游标下构建一个新的子游标

- •首先这种构建新子游标的操作更廉价了， 当时 Maclean 仍要告诫你 一个父游标下过多的子游标仍不是一件好事情

•对父游标的检测

- • 在找到一个合适的游标并执行前，父游标需要被适当检测。 对父游标的这种检测目前也使用 mutex 来保护了，所以这种检测更的成本更低了

•对于已经加载在 Library Cache 中的 SQL 语句重复执行

- •常规情况下，当一个进程要执行 SQL 游标前总是必须要先 pin 它
- •不使用 MUTEX 的情况：若游标处于 OPEN 状态下以便今后的重复执行，且参数 cursor_space_for_time(CSFT 目前已不推荐使用该参数)为 TRUE，则每一次重复执行可以不需要 library cache pin。 若游标处于 OPEN 状态下但是 cursor_space_for_time=false，则进程在重复执行 SQL 游标前总是要先拿 library cache pin
- •使用 MUTEX 的情况: 相反，若使用 mutex 来替代 library cache pin 时，则无需关心 cursor_space_for_time 。 仅第一个进程需要做一个 PIN，其他后续进程都只需要简单地在对应保护 cursor heap 的 mutex 上拿一个共享 reference 。

查询 SQL 统计信息

通过 V$SQLSTAT 视图(本质上是 X$KKSSQLSTAT)访问 SQL 统计信息时，其所需要的 CPu 和获取的 Latch 数量要远远少于访问其他 V$SQL 视图。 在早期版本中, 并行地访问 V$SQL 或者 V$SQLAREA 视图会造成 library cache latch 的争用。

Mutex 的统计信息

下面是一个 AWR 中的 Mutex Sleep Statistics， 这些数据主要来源于 V$MUTEX_SLEEP 视图。

Mutex Sleep Summary

- •ordered by number of sleeps desc

Mutex Type	Location	Sleeps	Wait Time (ms)
Library Cache	kglhdgh1 64	2,356	0
Library Cache	kglpnal2 91	2,345	0
Cursor Pin	kkslce [KKSCHLPIN2]	2,084	0
Library Cache	kglpin1 4	956	0
Library Cache	kglhdgn2 106	784	0
Library Cache	kglpndl1 95	691	0
Library Cache	kglpnal1 90	605	0
Library Cache	kgllkdl1 85	580	0
Library Cache	kgllkal1 80	404	0
Library Cache	kglllal3 111	282	0

Library Cache	kglllal1 109	218	0
Library Cache	kglhdgn1 62	163	0
Library Cache	kgllldl2 112	156	0
Library Cache	kgllkc1 57	105	0
Library Cache	kglget2 2	100	0
Library Cache	kglini1 32	53	0
Library Cache	kglget1 1	31	0
Cursor Pin	kksLockDelete [KKSCHLPIN6]	22	0
Library Cache	kgllkal3 82	18	0
Library Cache	kglUnsetHandleReference 120	10	0
Cursor Pin	kksxsccmp [KKSCHLPIN5]	10	0
Library Cache	kglobld1 75	8	0
Cursor Pin	kksfbc [KKSCHLPIN1]	8	0
Library Cache	kglUpgradeLock 119	7	0
Library Cache	kglhdgc1 102	2	0
Cursor Pin	kksfbc [KKSCHLFSP2]	2	0
Library Cache	kgldtin1 42	1	0
Library Cache	kglhbh1 63	1	0
Library Cache	kgllkal5 84	1	0
Library Cache	kglrdtin1 44	1	0
Cursor Parent	kkscsPruneChild [KKSPRTLOC35]	1	0

Mutex 的类型 Mutex Type

Mutex 的类型其实就是 mutex 对应的客户的名字， 在版本 10.2 中基本只有 KKS 使用 Mutex，所以仅有 3 种:

- •Cursor Stat (kgx_kks1)
- •Cursor Parent (kgx_kks2)
- •Cursor Pin (kgx_kks3)

在版本 11g 中扩展了对 Mutex 的使用，在 Library Cache 的 HASH BUCKET 中嵌入了 mutex 以保护 hash bucket，所以多了一种 mutex type : Library Cache

哪些代码函数会申请 Mutex?

Oracle 中哪些代码函数会申请 Mutex? 例如 KKSFBC 等，其实很像 V$LATCH 的 location 列

10.2 中最常见的下面的几个函数

`kkspsc0` -负责解析游标 - 检测我们正在解析的游标是否有对象的 `parent cursor heap 0` 存在

`kksfbc` - 负责找到合适的子游标 或者创建一个新的子游标

`kksFindCursorstat`

11g 开始有大量函数需要用到 Mutex 了

```
SQL> select location from X$MUTEX_SLEEP_HISTORY;
LOCATION
_____________________-
kkslce [KKSCHLPIN2]
kksfbc [KKSCHLFSP2]
kglhdgn2 106
kglpin1 4
kglhdgn2 106
kglllal1 109
kglpin1 4
kglpndl1 95
kglpin1 4
kglpin1 4
kksfbc [KKSCHLFSP2]
kglhdgn1 62
kglpnal1 90
kglllal3 111
kglpnal1 90
kglpnal1 90
kglget2 2
kglllal3 111
kglget2 2
kglobld1 75
kkslce [KKSCHLPIN2]
kglpndl1 95
kglpndl1 95
kglpin1 4
kkslce [KKSCHLPIN2]
kglpin1 4
kglget2 2
kglllal1 109
kgllkc1 57
kglget2 2
kglpnal1 90
kglpin1 4
kglpin1 4
kglpin1 4
kgllkdl1 85
kglllal3 111
kgllldl2 112
kglpin1 4
kglpndl1 95
kkslce [KKSCHLPIN2]
```

```
kksLockDelete [KKSCHLPIN6]
kglpndl1 95
kkslce [KKSCHLPIN2]
kglpnal1 90
kglpin1 4
kglpin1 4
kgllldl2 112
kgllkdl1 85
kglpin1 4
kglhdgn2 106
kglhdgn2 106
kksLockDelete [KKSCHLPIN6]
kglhdgn1 62
```

Mutex 的 Get 和 Sleep

当一个 Mutex 被申请时， 一般称为一个 get request。 若初始的申请未能得到授权， 则该进程会因为此次申请而进入到 255 次 SPIN 中(255 是在代码中用常量写死的)，每次 SPIN 循环迭代过程中该进程都会去看看 Mutex 被释放了吗。
若该 Mutex 在 SPIN 之后仍未被释放，则该进程针对申请的 mutex 进入对应的 mutex wait 等待事件中。实际进程的等待事件和等待方式由 mutex 的类型锁决定，例如 Cursor pin、Cursor Parent。 举例来说，这种等待可能是阻塞等待，也可以是 sleep。
但是请注意在 V$MUTEX_SLEEP_*视图上的 sleep 列意味着等待的次数。相关代码函数在开始进入等待时自加这个 sleep 字段。
等待计时从进程进入等待前开始计算等待时间， 当一个进程结束其等待，则等待的时间加入都总和 total 中。 该进程再次尝试申请之前的 Mutex，若该 Mutex 仍不可用，则它再次进入 spin/wait 的循环。

V$MUTEX_SLEEP_HISTORY 视图的 GETS 列仅在成功申请到一个 Mutex 时才增加。

短期持有一个 **mutex**: spin 循环 255 次一般可以有效以 S mode 获得一个 mutex, 前提是该 Mutex 已经被以 S mode 持有。 简单来说若有 2 个进程同时以 S mode 去申请一个 Mutex，则稍晚的一个申请者需要进入 SPIN 并等稍早一点的申请者完成它的例如创建针对该 mutex 的一个 reference 的操作，但这都是非常迅速的操作。

长期持有一个 **Mutex**: 如若一个 Mutex 已经被某进程以 X mode 持有， 则往往有其进程以 SHRD 模式去申请该 mutex 时仍发现该 mutex 以 X mode 被其他进程所持有，则往往这个 EXCL 持有是 LONG_EXCL(可以通过 SSD DUMP 发现)，则后续的申请者往往要进入 spin 循环，甚至需要等待

上面我讲了 willing-to-wait 的 mutex， 实际上 mutex 的申请也可以是 nowait 的。进程以 nowait 申请 mutex 时不会进入 spin-cycle 也不 sleep，它只继续常规处理。 当一个 nowait get 失败时，将增加一次 miss，但是实际上 V$MUTEX_SLEEP_*中记录的 miss 不是这样的 miss， 视图中记录的 miss 是等待的次数， 对于真正的 miss 没有统计项。

Wait Time 等待时间
类似于 latch，spin time 不算做 mutex 的消耗时间，它只包含等待消耗的时间。

真正理解 Mutex 相关的等待

Mutex 数据结构中存放了 Holder id 持有者 ID， Ref Count，和其他 Mutex 相关的统计信息。 Holder id 对应于持有该 Mutex 的 session id (v$session.sid) 。 特别注意， Ref Count 是进程并发以 S mode 参考该 Mutex 的进程数量(如下文的演示)。
当一个 Mutex 被以 X mode 持有，则 Holder id 为对应持有该 mutex 的 session id，而 Ref Count 为 0。

每一个共享 S mode 持有者仅仅增加 mutex 上的 Ref Count。 可供大量 session 并发以 S mode 持有参考一个 Mutex。 但是注意更新 ref count 的操作是串行的， 这是为了避免错漏并维护 mutex 中正确的 ref count。

下面我们详细介绍一个执行游标过程中对 mutex share pin 的过程：

- •某进程以 SHRD 模式申请一个 Mutex，并尝试临时修改该 Mutex 的 Holder ID
- •若该 Mutex 正被他人更新，则该 session 会将 Holder id 设置为本 session 的 sid，之后该进程将增加 ref count，之后再清楚 mutex 上的 Holder id。简单来说 这个 Holder id 是真正做了并行控制的功能。 若该 Holder id 被设置了，则说明该 Mutex 要么被以 EXCL 模式持有，要么正有一个其他进程在以 S mode 申请该 Mutex 的过程中(例如更新 Ref Count)。 当更新 Ref Count 时临时设置 holder id 的目的就是为了实现避免其他进程并发更新该 Mutex 的机制。 通过这些例子说明了 , Mutex 既可以用作 Latch 并发控制， 也可用作 pin。
- •若 Holder id 已被设置，则申请进程将可能进入等待事件 。 例如若当前 Mutex 的持有者进程正以 X mode 更新该 Mutex，则申请者的等待事件应为”cursor: pin S on X” 。 而若当持有者 Holder 并不是”真的要持有” 该 Mutex，而仅仅是尝试更新其 Ref Count,则第二个进程将等在’ Cursor :pin S’等待事件上； 实际正在更新 Ref count 的操作时很快的，是一种轻微的操作。 当第一个进程正在更新 mutex，则后续的申请进程将进入 spin 循环中 255 次等待前者结束。 当 mutex 上不再有 Holder id 时(如前者的进程已经更新完 Ref Count)时， 则申请者进程将 Holder ID 设为自身的 SID，并更新 Ref Count，并清除 Holder id。 若在 255 次循环 SPIN 后 mutex 仍不被释放，则该进程进入等待并不再跑在 CPU 上。

Mutex 相关的等待事件

cursor: mutex * events 等待事件

- •cursor: mutex * events 等待事件用于 Cursor Parent 和 Cursor stats 类型的操作：
  - •‘cursor: mutex X’ ， 某个进程申请以 EXCL mode 持有 mutex 时进入该等待， 该 Mutex 要么正被其他进程以 SHRD 模式参考，这导致 X mode 的申请必须要等待直到 Ref count=0， 或者该 mutex 正被另一个进程以 X mode 持有。

- 相关操作要求以 EXCL X mode 持有 Mutex 的：
  - 在一个父游标下创建一个新的子游标
  - 捕获 SQL 中的绑定变量
  - 更新或构件 SQL 统计信息 V$SQLSTATS
- ‘Cursor: Mutex S’ ， 某个进程以 SHRD S mode 申请一个 Mutex， 而该 Mutex 要么被其他进程已 EXCL X mode 所持有，要么其他进程正在更新 mutex 上的 Ref Count。
  - 相关类型的操作一般是检测父游标或者 CURSOR 统计信息数据， 此外查询 V$SQLSTATS 也会造成 CURSOR statistics 被查询

‘cursor: pin * events’等待事件

该类等待事件一般是为了 pin 相关的子游标

- cursor: pin S 当一个进程以共享 pin 模式申请一个 Mutex，而不能立即获得时，进入 cursor: Pin S 等待事件。 Mutex Pin 是以共享类型的操作，例如执行一个游标。
  - 当一个进程等在 cursor: pin S 上，说明该进程在对一个共享的 mutex pin 参考或取消参考时，有其他的进程也正在为同样的 cursor heap 创建或者取消一个共享 Mutex pin。 实际上 cursor: pin S 等待事件应当很少见，因为更新共享 Mutex pin 的 reference 应当是很快的。 再重复一次，S mode 的 Mutex 可以被并发持有， 但是更新 Mutex 的 Ref Count 仍需要串行地处理 。一旦 reference count 被增加好，则后续进程将可以为同样的 cursor heap 增加 reference count。 因此此处 mutex 即可以扮演 Latch 的角色(串行控制 ref count 的更新)，又可以扮演 pin 的角色(ref count 本身)。
- ‘cursor: pin X’ 当一个进程需要以 EXCL X mode 获得 mutex 时， 这类需要 EXCL X 模式的串行操作包括:
  - 构建一个子游标
  - 某个进程已经以 X mode 持有该 Mutex
  - 一个或多个进程正在 reference 该 Mutex (shared mutex pin)
- ‘Cursor: pin S on X’ 最常见的等待事件， 进程为了共享操作例如执行 pin 游标而以 SHRD S mode 申请 mutex， 但是未立即获得。原因是该游标被其他进程以 EXCL X mode 持有了。

Mutex 的相关统计视图

`V$MUTEX_SLEEP`

shows the wait time, and the number of sleeps for each combination of mutex type and location.

Column	Datatype	Description
`MUTEX_TYPE`	`VARCHAR2(32)`	Type of action/object the mutex protects
`LOCATION`	`VARCHAR2(40)`	The code location where the waiter slept for the mutex

Column	Datatype	Description
SLEEPS	NUMBER	Number of sleeps for this MUTEX_TYPE and LOCATION
WAIT_TIME	NUMBER	Wait time in microseconds

V$MUTEX_SLEEP_HISTORY

displays time-series data. Each row in this view is for a specific time, mutex type, location, requesting session and blocking session combination. That is, it shows data related to a specific session (requesting session) that slept while requesting a specific mutex type and location, because it was being held by a specific blocking session. The data in this view is contained within a circular buffer, with the most recent sleeps shown.

Column	Datatype	Description
SLEEP_TIMESTAMP	TIMESTAMP(6)	The last date/time this MUTEX_TYPE and LOCATION was slept for by theREQUESTING_SESSION, while being held by the BLOCKING_SESSION.
MUTEX_TYPE	VARCHAR2(32)	Type of action/object the mutex protects
GETS	NUMBER	The number of times the mutex/location was requested by the requesting session while being held by the blocking session. GETS is only incremented once per request, irrespective of the number of sleeps required to obtain the mutex.
SLEEPS	NUMBER	The number of times the requestor had to sleep before obtaining the mutex
REQUESTING_SESSION	NUMBER	The SID of a session requesting the mutex
BLOCKING_SESSION	NUMBER	The SID of a session holding the mutex
LOCATION	VARCHAR2(40)	The code location where the waiter slept for the mutex
MUTEX_VALUE	RAW(4)	If the mutex is held in exclusive (X) mode, this column shows the SID of the blocking session, else it shows the number of sessions referencing the mutex in S mode.
P1	NUMBER	Internal use only
P1RAW	RAW(4)	Internal use only
P2	NUMBER	Internal use only
P3	NUMBER	Internal use only
P4	NUMBER	Internal use only
P5	VARCHAR2(64)	Internal use only

接着我们会在环境中模拟 cursor pin S wait on X 的场景，并通过 systemstate dump 和 v$mutex_sleep ,

v$mutex_sleep_history 等视图观察这一现象

```
session A:
SQL> select * from v$version;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 - 64bi
PL/SQL Release 10.2.0.5.0 - Production
CORE 10.2.0.5.0 Production
TNS for Linux: Version 10.2.0.5.0 - Production
NLSRTL Version 10.2.0.5.0 - Production
www.askmaclean.com
SQL> show parameter kks
SQL>
SQL> create table mac_kks tablespace users nologging as select * from
dba_objects;
Table created.
SQL> insert /*+ append */ into mac_kks select * from mac_kks;
77386 rows created.
SQL> commit;
Commit complete.
SQL> insert /*+ append */ into mac_kks select * from mac_kks;
154772 rows created.
SQL> commit;
Commit complete.
SQL> insert /*+ append */ into mac_kks select * from mac_kks;
309544 rows created.
SQL> commit;
Commit complete.
SQL> insert /*+ append */ into mac_kks select * from mac_kks;
619088 rows created.
SQL> commit;
Commit complete.
SQL> insert /*+ append */ into mac_kks select * from mac_kks;
1238176 rows created.
SQL> commit;
Commit complete.



SQL> oradebug setmypid
```

```
Statement processed.
SQL> oradebug tracefile_name
Statement processed.
SQL> alter table mac_kks add t2 char(2000) default 'MACLEAN';



session B:
SQL> oradebug setospid 32424
Oracle pid: 17, Unix process pid: 32424, image: oracle@vrh8.oracle.com (TNS
V1-V3)
SQL> oradebug suspend;
Statement processed.

session C:
select * from mac_kks where rownum=1; ==> hang
session D:
select * from mac_kks where rownum=1; ==> hang

session E:
SQL> select sid,event from v$session where wait_class!='Idle';
SID EVENT
------ ------------------------
141 SQL*Net message to client
145 library cache lock
149 cursor: pin S wait on X
159 log buffer space
SQL> oradebug setmypid
Statement processed.
SQL> oradebug dump systemstate 266;
Statement processed.
SQL> oradebug tracefile_name
/s01/admin/G10R25/udump/g10r25_ora_32537.trc
Object Names
~~~~~~~~~~~~
LOCK: handle=a7115ef0
Mutex 7fff7abadecf

KGX Atomic Operation Log 0x8d88a8d8
Mutex 0x954eaff8(145, 0) idn 7fff7abadecf oper EXCL
Cursor Pin uid 145 efd 0 whr 1 slp 0
opr=3 pso=0x97951af0 flg=0
pcs=0x954eaff8 nxt=(nil) flg=35 cld=0 hd=0xa7864b08 par=0x9523a9e0
```

ct=0 hsh=0 unp=(nil) unn=0 hvl=9595c3d8 nhv=1 ses=0xa8416738
hep=0x954eb078 flg=80 ld=1 ob=0x95ac6128 ptr=0x8fd90128 fex=0x8fd8f438
0x954eaff8(145, 0) ==> sid和 ref count
pso ==> parent state object
hd=0xa7864b08 ==>cursor 对应的handle address
par ==> 父游标的heap 0 pointer
ses=0xa8416738 ==》 一般 EXCL是才有 session address v$session.saddr
SID=145 对Mutex 0x954eaff8 oper EXCL以X mode Hold 该Mutex, SID=145 在等
SYS.MAC_KKS表的library cache lock, 该表被X mode pin和lock, 而解析SQL要求以S
mode lock该表
SID=149 对Mutex 0x954eaff8 申请 oper GET_SHRD, SID=149在等cursor: pin S wait
on X
KGX Atomic Operation Log 0x8db79798
Mutex 0x954eaff8(145, 0) idn 7fff7abadecf oper GET_SHRD
Cursor Pin uid 149 efd 0 whr 5 slp 13893
opr=2 pso=0x8e6bd518 flg=0
pcs=0x954eaff8 nxt=(nil) flg=35 cld=0 hd=0xa7864b08 par=0x9523a9e0
ct=0 hsh=0 unp=(nil) unn=0 hvl=9595c3d8 nhv=1 ses=0xa8416738
hep=0x954eb078 flg=80 ld=1 ob=0x95ac6128 ptr=0x8fd90128 fex=0x8fd8f438
SO: 0xa841bd18, type: 4, owner: 0xa830cf98, flag: INIT/-/-/0×00
(session) sid: 149 trans: (nil), creator: 0xa830cf98, flag: (80000041) USR/-
BSY/-/-/-/-/-
DID: 0001-0019-00000066, short-term DID: 0000-0000-00000000
txn branch: (nil)
oct: 0, prv: 0, sql: 0x8d88bf90, psql: (nil), user: 0/SYS
service name: SYS$USERS
O/S info: user: oracle, term: pts/5, ospid: 32510, machine: vrh8.oracle.com
program: sqlplus@vrh8.oracle.com (TNS V1-V3)
application name: sqlplus@vrh8.oracle.com (TNS V1-V3), hash value=543908804
waiting for ‘cursor: pin S wait on X’ wait_time=0, seconds since wait
started=0
idn=7abadecf, value=9100000000, where|sleeps=500003645
blocking sess=0x(nil) seq=13901
Dumping Session Wait History
for ‘cursor: pin S wait on X’ count=1 wait_time=0.266596 sec
idn=7abadecf, value=9100000000, where|sleeps=500003644
for ‘cursor: pin S wait on X’ count=1 wait_time=0.010679 sec
idn=7abadecf, value=9100000000, where|sleeps=500003643
for ‘cursor: pin S wait on X’ count=1 wait_time=0.010633 sec
idn=7abadecf, value=9100000000, where|sleeps=500003642
for ‘cursor: pin S wait on X’ count=1 wait_time=0.010843 sec
idn=7abadecf, value=9100000000, where|sleeps=500003641
for ‘cursor: pin S wait on X’ count=1 wait_time=0.011008 sec
idn=7abadecf, value=9100000000, where|sleeps=500003640

```
for 'cursor: pin S wait on X' count=1 wait_time=0.010406 sec


SO: 0xa8416738, type: 4, owner: 0xa830bfa8, flag: INIT/-/-/0×00
(session) sid: 145 trans: (nil), creator: 0xa830bfa8, flag: (80000041) USR/-
BSY/-/-/-/-/-
DID: 0001-0017-0000008E, short-term DID: 0000-0000-00000000
txn branch: (nil)
oct: 3, prv: 0, sql: 0x8d88bf90, psql: (nil), user: 0/SYS
service name: SYS$USERS
O/S info: user: oracle, term: pts/4, ospid: 32485, machine: vrh8.oracle.com
program: sqlplus@vrh8.oracle.com (TNS V1-V3)
application name: sqlplus@vrh8.oracle.com (TNS V1-V3), hash value=543908804
waiting for 'library cache lock' wait_time=0, seconds since wait started=165
handle address=a7115ef0, lock address=978def20, 100*mode+namespace=c9
blocking sess=0x(nil) seq=9
Dumping Session Wait History
for 'library cache lock' count=1 wait_time=3.297287 sec
handle address=a7115ef0, lock address=978def20, 100*mode+namespace=c9
for 'library cache lock' count=1 wait_time=2.930321 sec
handle address=a7115ef0, lock address=978def20, 100*mode+namespace=c9
for 'library cache lock' count=1 wait_time=2.930856 sec
handle address=a7115ef0, lock address=978def20, 100*mode+namespace=c9
for 'library cache lock' count=1 wait_time=2.930698 sec
handle address=a7115ef0, lock address=978def20, 100*mode+namespace=c9
for 'library cache lock' count=1 wait_time=2.931518 sec
handle address=a7115ef0, lock address=978def20, 100*mode+namespace=c9
SO: 0x978def20, type: 53, owner: 0xa8456058, flag: INIT/-/-/0×00
LIBRARY OBJECT LOCK: lock=978def20 handle=a7115ef0 request=S
call pin=(nil) session pin=(nil) hpc=0005 hlc=0000
htl=0x978defa0[0x8e4a8950,0x8e4a8950] htb=0x8e4a8950 ssga=0x8e4a7a88
user=a8416738 session=a8416738 count=0 flags=RES/[0010] savepoint=0x1f
LIBRARY OBJECT HANDLE: handle=a7115ef0 mtx=0xa7116020(0) lct=8 pct=10 cdp=0
name=SYS.MAC_KKS
hash=c066b9b9c6c80736a15f5ba325563fdb timestamp=04-14-2013 00:43:14
namespace=TABL flags=KGHP/TIM/SML/[02000000]
kkkk-dddd-llll=0000-0701-0201 lock=X pin=X latch#=3 hpc=0006 hlc=0004
lwt=0xa7115f98[0x978def50,0x978def50] ltm=0xa7115fa8[0xa7115fa8,0xa7115fa8]
pwt=0xa7115f60[0xa7115f60,0xa7115f60] ptm=0xa7115f70[0xa7115f70,0xa7115f70]
ref=0xa7115fc8[0xa7115fc8,0xa7115fc8] lnd=0xa7115fe0[0x9cc2c260,0xa79fd198]
LIBRARY OBJECT: object=95f0e5b0
type=TABL flags=EXS/LOC/UPD[0905] pflags=[0000] status=VALD load=0
DATA BLOCKS:
```

```
data# heap pointer status pins change whr
── ──────── ──────── ──────── ──- ── ─
0 a78d3840 95f0e708 I/P/A/-/- 0 NONE 00
8 957a6ad8 94461f88 I/P/A/-/- 1 UPDATE 00
Mutex 0x954eaff8 被 SID=145 oper EXCL 以 X mode Hold 该 Mutex
体现为 子游标 child cursor 被以 X mode pin
SO: 0x8e6bd518, type: 53, owner: 0xa841bd18, flag: INIT/-/-/0×00
LIBRARY OBJECT LOCK: lock=8e6bd518 handle=a7864b08 mode=N
call pin=(nil) session pin=(nil) hpc=0000 hlc=0000
htl=0x8e6bd598[0x8e542f60,0x979da4d0] htb=0x979da4d0 ssga=0x979d96c8
user=a841bd18 session=a841bd18 count=1 flags=CBK[0020] savepoint=0×0
LIBRARY OBJECT HANDLE: handle=a7864b08 mtx=0xa7864c38(0) lct=2 pct=3 cdp=0
namespace=CRSR flags=RON/KGHP/PN0/EXP/[10010100]
kkkk-dddd-llll=0000-0001-0001 lock=N pin=X latch#=3 hpc=0002 hlc=0002
lwt=0xa7864bb0[0xa7864bb0,0xa7864bb0] ltm=0xa7864bc0[0xa7864bc0,0xa7864bc0]
pwt=0xa7864b78[0xa7864b78,0xa7864b78] ptm=0xa7864b88[0xa7864b88,0xa7864b88]
ref=0xa7864be0[0x954eb320,0x954eb320] lnd=0xa7864bf8[0xa7864bf8,0xa7864bf8]
LIBRARY OBJECT: object=95ac6128
type=CRSR flags=EXS[0001] pflags=[0000] status=VALD load=0
DATA BLOCKS:
data# heap pointer status pins change whr
── ──────── ──────── ──────── ──- ── ─
0 8d81aa70 95ac6240 I/P/A/-/- 0 NONE 00
6 954eb0f0 8fd90128 I/P/A/-/E 0 NONE 00
并行执行 6 个 SQL 语句 并做 systemstate dump :

[oracle@vrh8 ~]$ grep SHRD /s01/admin/G10R25/udump/g10r25_ora_32716.trc |
grep 0×95987388
Mutex 0×95987388(0, 6) idn 7fff8f32fff6 oper SHRD
Mutex 0×95987388(0, 6) idn 7fff8f32fff6 oper SHRD
Mutex 0×95987388(0, 6) idn 7fff8f32fff6 oper SHRD
Mutex 0×95987388(0, 6) idn 7fff8f32fff6 oper SHRD
Mutex 0×95987388(0, 6) idn 7fff8f32fff6 oper SHRD
Mutex 0×95987388(0, 6) idn 7fff8f32fff6 oper SHRD
```