Description of Oracle7 Wait Events and Enqueues

Anjo Kolk (akolk@us.oracle.com)

1.0 Introduction

Since Oracle7 version 7.0.12, there is a new wait interface that keeps track of the waits of all sessions/processes in an instance. The names of these views are **v$system_event, v$session_event** and **v$session_wait**.

The **v$session_wait** view shows the events that sessions have just completed waiting for or are currently waiting on. The **v$system_event** shows the total number of times all the sessions have waited on the events in that view. The **v$session_event** is similar to v$system_event, but is shows all the waits for events broken down by session.

For each event we document the name, wait time, parameters (p1, p2 and p3) and optionally advise on how to tune this event.

The following SQL statement will display all the events in an Oracle7 release:

```
select *
  from v$event_name;
```

Here follow the fixed view definitions for the views mentioned before in for Oracle7:

v$session_event

sid	number	Session id (v$session).
event	varchar2(64)	The name of the event that the session has been waiting for.
total_waits	number	The total number of waits for this event by this session.
total_timeouts	number	The total number of timeouts for this event by this session.
time_waited	number	The total time waited by this session for this event.
average_wait	number	The average wait (time_waited/total_waits).

v$session_wait

sid	number	Session id (v$session).
seq#	number	The sequence number of the wait for this session.
event	varchar2(64)	The name of the event that the session is waiting for or just completed waiting for (depends on state).
p1text	varchar2(64)	The name of parameter 1.
p1	number	The value of parameter 1.
p1raw (7.2)	raw(4)	The raw value of parameter 1.
p2text	varchar2(64)	The name of parameter 2.

p2	number	The value of parameter 2.
p2raw (7.2)	raw(4)	The raw value of parameter 2.
p3text	varchar2(64)	The name of parameter 3.
p3	number	The value of parameter 3
p3raw (7.2)	raw(4)	The raw value of parameter 3
wait_time	number	The time waited for this event.
seconds_in_wait (7.3)	number	The approximate wall clock time in seconds at the start of the wait.
state (7.2)	varchar2(19)	The following values are possible:

TABLE 1. The waitstates and their meaning.

State	Value	Meaning
WAITING	0	Session is currently waiting for this event.
WAITED UNKNOWN TIME	-2	Timing is not enabled.
WAITED SHORT TIME	-1	Session woke up in the same CPU clock tick as it went to sleep.
WAITED KNOWN TIME	> 0	Session waited 'wait_time'.

v$system_event

event	varchar2(64)	The name of the event.
total_waits	number	The total number of waits for this event (instance wide).
total_timeouts	number	Total number of time-outs for this event (instance wide).
time_waited	number	Total time waited for this event (instance wide).
average_wait	number	Average wait time (time_waited/total_waits).

These fixed views are based on fixed tables and normally do sorting (sort-merge joins). On systems with a lot of contention on the 'ST' enqueue[1], querying these tables may take a long time. Querying the underlying fixed tables may improve performance. Here follows a list of the underlying fixed tables:

TABLE 2. Mapping fixed views to fixed tables.

Fixed View	Fixed Table
v$session_wait	x$ksusecst, x$ksled
v$session_event	x$ksles, x$ksled
v$system_event	x$kslei, x$ksled

Remember these fixed tables are owned by SYS and can change without any notice from release to release. In Oracle7 version 7.3, sort segments can almost completely reduce contention on the ST enqueue for sorting (sort-merge join).

How to use these event views ?

The following query gives a quick and but not complete look at the overall system:

[1].\tSee chapter 2 and chapter 3 for more detail on the ST enqueue.

```
select count(*), event
   from v$session_wait
group by event;
```

Better is:

```
select count(*), event, max(seq#)
  from v$session_wait
 group by event;
```

The reason for the max(seq#) is that you can see if the sequence number changes. Every time a session waits for an event it will increment this sequence number. If the sequence number doesn't increment it means that one session is stuck in this event.

2.0 Oracle7 Events

In Oracle7 the following events are present:

Event Name	P1	P2	P3
DFS db file lock	file#		
DFS enqueue lock acquisition	name|mode	id1	id2
DFS enqueue lock handle	name|mode	id1	id2
DFS enqueue request cancellation	name|mode	id1	id2
DFS lock acquisition	type|mode	id1	id2
DFS lock convert	mode	options	
DFS lock handle	type|mode	id1	id2
DFS lock release	options		
DFS lock request cancellation	options		
Null event			
PL/SQL lock timer	duration		
SQL*Net break/reset to client	driver id	break?	
SQL*Net break/reset to dblink	driver id	break?	
SQL*Net message from client	driver id	#bytes	
SQL*Net message from dblink	driver id	#bytes	
SQL*Net message to client	driver id	#bytes	
SQL*Net message to dblink	driver id	#bytes	
SQL*Net more data from client	driver id	#bytes	
SQL*Net more data from dblink	driver id	#bytes	
SQL*Net more data to client	driver id	#bytes	
SQL*Net more data to dblink	driver id	#bytes	
WMON goes to sleep			
batched allocate scn lock request			

buffer busy waits	file#	block#	id
buffer deadlock	dba	class*10+mode	flag
buffer for checkpoint	buffer#	dba	state*10+mode
buffer latch	latch addr	chain#	
buffer read retry	file#	block#	
checkpoint completed			
checkpoint range buffer not saved			
client message	two-task?	driver-id	
control file parallel write	files	blocks	requests
control file sequential read	file#	block#	blocks
control file single write	file#	block#	blocks
conversion file read	file#	block#	blocks
db file parallel write	files	blocks	requests
db file scattered read	file#	block#	blocks
db file sequential read	file#	block#	blocks
db file single write	file#	block#	blocks
debugger command			
direct loader I/O/direct access I/O	descriptor address	first dba	last dba written/block cnt
dispatcher shutdown	waited		
dispatcher timer	sleep time		
dupl. cluster key	dba		
enqueue	name|mode	id1	id2
free buffer waits	file#	block#	set-id#
free global transaction table entry	tries		
free process state object			
inactive session	session#	waited	
index block split	rootdba	level	childdba
instance recovery	undo segment#		
instance state change	layer	value	waited
kcl bg acks	count	loops	
latch activity	address	number	process#
latch free	address	number	tries
library cache load lock	object address	lock address	10*mask+namespace
library cache lock	handle address	lock address	10*mode+namespace
library cache pin	handle address	pin address	10*mode+namespace
lock element cleanup	file#	block#	lenum
lock element waits	file#	block#	lenum
log buffer space			

log file parallel write	files	blocks	requests
log file sequential read	log#	block#	blocks
log file single write	log#	block#	blocks
log file space/switch			
log file switch (checkpoint incomplete)			
log file switch (archiving needed)			
log file switch (cleaning log file)			
log file switch (switch log file command)			
log file sync	buffer#		
log switch/archive	thread#		
on-going reading of SCN to complete			
parallel query create server	nservers	sleeptime	enqueue
parallel query dequeue wait	queue/reason	sleeptime	passes
parallel query qref latch	function	sleeptime	qref
parallel query server shutdown	nalive	sleeptime	loop
parallel query signal server	serial	error	nbusy
pending global transaction(s)	scans		
pipe get	handle address	buffer length	timeout
pipe put	handle address	record length	timeout
pmon timer	duration		
process startup	type	process#	waited
queue wait			
rdbms ipc message	timeout		
rdbms ipc message block			
rdbms ipc reply	from_process	timeout	
row cache lock	cache id	mode	request
scginq AST call			
secondary event	event #	wait time	
sequence# cache entry	entry#		
sequence# lock op	entry#		
single-task message			
smon timer	sleep time	failed	
timer in sksawat			
trace continue	delay time		
trace unfreeze			
trace writer I/O			

trace writer flush			
transaction	undo seg#| slot#	wrap#	count
undo segment extension	segment#		
undo segment recovery	segment#	tx flags	
undo segment tx slot	segment#		
virtual circuit status	circuit#	status	
write complete waits	file#	block#	id
writes stopped by instance recovery	by thread#	our thread#	

2.1 DFS db file lock

This will only show up for the DBWR in Parallel Server only. Each DBWR of every instance will hold a global lock on each file in shared mode. The instance that is trying to put the file off-line will escalate the global lock from shared to exclusive to signal to the other instances to synchronize their SGA's with the control file before it can be taken off-line. The name of this lock is **DF** (see chapter 3 for more information).

Wait time

1 Second, basically the DBWR is waiting in a tight loop for the other instances to downgrade to NULL mode. During this time the DBWR can't do other tasks like writing buffers.

Parameters

file#

Shows which file# is being brought off-line. The following query will show the name of the database file:

```
select *
  from v$datafile
 where file# = file#;
```

Advise

Don't try to off-line files during a peak time in a database, this can cause DBWR to fall behind and impact response times of transactions until the DBWR catches up again with the workload.

2.2 DFS enqueue lock acquisition

The Oracle7 kernel is waiting on the acquisition of a global enqueue. The operation is basically a get on a lock that we currently don't have. If mode is not 0, it is a convert operation.

Wait time

We are in a tight loop until acquisition is complete. Every loop we wait 0.5 seconds.

Parameters

name

The name of the lock can be determined by looking at the 2 high order bytes of P1 or P1RAW. The name will always be 2 characters (See Chapter 3). The following SQL statement will retrieve the lock name

```
select chr(bitand(p1,-16777216)/16777215)||
       chr(bitand(p1,16711680)/65535) "Lock"
  from v$session_wait
 where event = 'DFS enqueue lock acquisition';
```

The table below shows all the possible lock names[2].

TABLE 3. Enqueue names in Oracle7.

Name	Meaning
BL	Buffer Cache Management (PCM lock)
CF	Controlfile Transaction
CI	Cross Instance Call
CU	Bind Enqueue
DF	Data File
DL	Direct Loader
DM	Database Mount
DR	Distributed Recovery
DX	Distributed Transaction
FS	File Set
IN	Instance Number
IR	Instance Recovery
IS	Instance State
IV	Library Cache Invalidation
JQ	Job Queue
KK	Redo Log Kick
L[A-P]	Library Cache Lock
MM	Mount Definition
MR	Media Recovery
N[A-Z]	Library Cache Pin
PF	Password File
PI	Parallel Slaves
PR	Process Startup
PS	Parallel slave Synchronization
Q[A-Z]	Row Cache Lock
RT	Redo Thread
SC	System Commit number
SM	SMON synchronization
SN	Sequence Number
SQ	Sequence Enqueue
SR	Synchronous Replication

[2].\tThere was a bug in 7.0 and 7.1 that could cause the lock name to be corrupted. So if a weird name shows up, you are most likely running in this bug. It was supposed to be fixed in 7.1.6, but that is not true for all platforms.

SS	Sort Segment
ST	Space Management Transaction
SV	Sequence Number Value
TA	Transaction Recovery
TM	DML Enqueue
TS	Table Space (or Temporary Segment)
TT	Temporary Table
TX	Transaction
UL	User-defined Locks
UN	User Name
US	Undo segment Serialization
WL	Writing redo Log
XA	Instance Attribute Lock
XI	Instance Registration Lock

mode

The mode will be stored in the low order bytes of P1 or P1RAW. The mode will be one of the following values:

TABLE 4. Lock Modes

Mode	Meaning
1	Null mode
2	Sub-Share
3	Sub-Exclusive
4	Share
5	Share/Sub-Exclusive
6	Exclusive

With the following SQL statement one can find the name of the lock and the mode of the lock request:

```
select chr(bitand(p1,-16777216)/16777215)||
       chr(bitand(p1, 16711680)/65535) “Lock”,
       bitand(p1, 65535) “Mode”
  from v$session_wait
 where event = ‘DFS enqueue lock acquisition’;
```

id1

P2 or P2RAW will give the first identifier of the enqueue. See chapter 3 for the meaning of that identifier, it will depend on the name (P1).

id2

P3 or P3RAW will give the second identifier of the enqueue. See chapter 3 for the meaning of that identifier, it will depend on the name (P1).

Advise

So much depends on the on the name of the enqueue that you are trying to get. The following statistics from **v$sysstat** are related:

global lock gets (non async)

This statistics gets incremented when the get lock operation has finished and the operation was a get (mode is 0).

global lock get time

The time it took to complete the synchronous lock get.

global lock converts (non async)

This statistics gets incremented when the synchronous convert lock operation has finished.

global lock convert time

The time it took to complete the synchronous lock convert.

With the following SQL statement one can see the complete enqueue request that sessions are waiting for or have just waited for:

```
select chr(bitand(p1,-16777216)/16777215)||
       chr(bitand(p1, 16711680)/65535) “Lock”,
       bitand(p1,65535) “Mode”, p2 id1, p3 id2
  from v$session_wait
 where event = ‘DFS enqueue lock acquisition’;
```

And with the following SQL statement that needs to be executed on each instance one can find the current owner(s) of the enqueue that this session is waiting for:

```
select l.*
  from v$lock l, v$session_wait s
 where type = chr(bitand(p1, -16777216)/16777215)||
              chr(bitand(p1, 16711680)/65535)
   and event = ‘DFS enqueue lock acquisition’
   and lmode > 0;
```

There are also two internal tables that keep track of enqueues (local or global):

x$ksqrs

This fixed table shows all outstanding enqueues with an additional flag. It basically shows the same information as the v$lock table.

TABLE 5. ksqrsflg column of X$KSQRS fixed table.

KSQRSFLG	Meaning
0x01	Lock is global
0x02	Not free
0x04	User deadlock possible
0x08	Global lock in first release group
0x10	Global lock in last release group
0x20	non global lock value is invalid

x$ksqst

This fixed table shows the number of gets and waits on each enqueue resource identified by type in v$lock and name in the “DFS enqueue lock ...” events.

2.3 DFS enqueue lock handle

Oracle7 is waiting for the lock handle. The lock handle identifies a lock that is currently held. This lock handle is obtained from the lock state on the initial acquisition and may be used to reconstruct the state for conversion or release. This event is called if we are getting a lock and we want to know the lock handle (to identify the lock in future operations like conversions or release). The lock is maintained by the DLM.

Wait time

This happens in a tight loop. Waits are 0.5 seconds every time.

Parameters

name

See 'DFS enqueue lock acquisition' for a description.

mode

See 'DFS enqueue lock acquisition' for a description.

id1

See 'DFS enqueue lock acquisition' for a description.

id2

See 'DFS enqueue lock acquisition' for a description.

Advise

So much depends on the on the name of the enqueue that you are trying to get. See 'DFS enqueue lock acquisition' for a description.

2.4 DFS enqueue request cancellation

Oracle7 is trying to cancel a lock asynchronously. The reason to cancel is that there is an user interrupt (CTRL-C).

Wait time

The wait is 0.5 seconds in a tight loop, until the request is canceled.

Parameters

name

See 'DFS enqueue lock acquisition' for a description.

mode

See 'DFS enqueue lock acquisition' for a description.

id1

See 'DFS enqueue lock acquisition' for a description.

id2

See 'DFS enqueue lock acquisition' for a description.

Advise

See 'DFS enqueue lock acquisition' for a description.

2.5 DFS lock acquisition

The locks that show up under this event are only gotten in Parallel Server mode. The locks gotten under the event 'DFS enqueue lock acquisition' could also be gotten under normal (Exclusive mode Oracle) operation, they would show up under the event 'enqueues'. This operation can either be synchronous or asynchronous, it depends on the type of the lock. The synchronous operations will show under this event.

Wait time

Basically we keep on waiting until we have gotten the lock. There is a tight loop here we wait half a second (0.5 second) each time.

Parameters

type

See 'DFS enqueue lock acquisition' *name* for a description.

mode

See 'DFS enqueue lock acquisition' *mode* for a description.

id1

See 'DFS enqueue lock acquisition' *id1* for a description.

id2

See 'DFS enqueue lock acquisition' *id2* for a description.

Advise

The following statistics in the **v$sysstat** table are related to this event:

global lock gets (non async)

This statistics gets incremented when the synchronous get lock operation has finished and the operation was a get (mode is 0).

global lock get time

The time it took to complete this operation. This doesn't include the wait time for the ***global lock gets (async)***.

global lock gets (async)

This statistics gets incremented for the async lock operations

2.6 DFS lock convert

Oracle7 needs to convert an instance lock. The convert is a synchronous operation if this event shows up in the **v$session_wait** view, the convert is asynchronous if it doesn't show in the **v$session_wait** view. After a wait has completed there is a check for interrupts. When an interrupt occurs the lock convert is canceled.

Wait time

The wait is 0.5 second in a tight loop, until the convert completes

Parameters

mode

See 'DFS enqueue lock acquisition' *mode* for a description.

options

These options are options are internal options used by the Oracle7 kernel to DLM convert operations. These values are useful for debugging certain kind of operations. For example if one DFS lock convert operation shows a 0x10 and the next DFS lock convert operation shows a 0x20, one can assume that this is a convert on the SC global lock.

TABLE 6. DFS Lock Convert options.

Value	Meaning
0x01	This is a lock get (instead of lock convert)
0x02	Only for get: Lock Handle Not Needed
0x04	Only for get: Deadlock detection Resource Space

0x08	Get/Convert: Do Deadlock Detection
0x10	Read the Lock Value
0x20	Write the Lock value (modify)
0x40	read or write the lockvalue, as appropriate
0x80	This lock is in the first group to be released
0x100	This lock is in the last group to be released
0x200	Put the request in front of the convert queue
0x400	lock is process, as opposed to group, owned.

Advise

Mostly useful on debugging the type of resource and type of operation that is being waited on. One can see converts (on the SC instance lock[3]) by looking at the 0x10 and 0x20 flag in options parameter (P2/P2RAW).

2.7 DFS lock handle

The Oracle7 kernel needs to get the lock handle, so we are waiting for a currently pending lock operation to return a lock handle. There is a tight loop here until the lock handle is returned.

Wait time

The wait is 0.5 seconds every time.

Parameters

type

See 'DFS enqueue lock acquisition' for a description.

mode

See 'DFS enqueue lock acquisition' for a description.

id1

See 'DFS enqueue lock acquisition' for a description.

id2

See 'DFS enqueue lock acquisition' for a description.

Advise

This should normally be a short wait with very little time-outs. A large number of time-outs here could indicate a performance problem with the DLM (too many requests or remote operations could be too expensive).

2.8 DFS lock release

Oracle7 waits in this event while it is waiting for the DLM to release a lock. This operation is synchronous.

Wait time

The wait is 0.5 second in a tight loop, until the lock is released.

Parameters

[3].\tSee Chapter 3 for an overview of all Enqueues and Instance Locks.

options

The following values for options may be and'ed together to the value for P1:

TABLE 7. DFS Lock Release Options.

Value	Meaning
0x2	Write Lock Value
0x4	Async Callback on Error Only
0x8	Invalidate Lock Value if held X or SSX

Advise

If a session is hanging in this operation the problem would normally be with the DLM. Check the DLM to see if it is still working.

2.9 DFS lock request cancellation

Oracle7 is trying to cancel a lock synchronously. This can happen for many reasons, but the most common one will be the fact that Oracle7 received a BAST (Blocking ASynchronous Trap) while an AAST (Acquisition ASynchronous Trap) is pending. This will most likely be on the PCM locks.

Wait time

The wait is 0.5 seconds in a tight loop.

Parameters

options

This will most likely be 0.

Advise

Check if DLM is still working. A high number of lock request cancellations can be caused by a high number of 'lock element cleanup' time-outs. That could be caused by too much pinging (true or false) and LCK processes that are too busy. Add more LCK process or redistribute PCM locks (by changing the gc_* parameters) to reduce contention.

2.10 Null Event

The session is waiting without specifying an existing event.

Wait time

The wait time may vary depending on the context where it is used.

Parameters

No parameters.

Advise

This event should not be used and a bug should be filed if it is encountered.

2.11 PL/SQL lock timer

This event is called through the dbmslock.sleep procedure or userlock.sleep procedure. This event is most likely to come from user written stored procedures.

Wait time

The wait time is in hundredths of seconds and dependent on the user context where it is used.

Parameters

duration

The duration that the user specified in the dbms_lock.sleep or user_lock.sleep procedures.

Advise

Check user PL/SQL functions and procedures for use of these procedures.

2.12 SQL*Net break/reset to client

The server is sending a break or reset message to the client. The session running on the server is waiting for a reply from the client.

Wait time

The actual time it takes for the break or reset message to come back from the client.

Parameter

driver id

The value here is the value of the disconnect function of the driver that is currently being used.

break?

If the value for this parameter equals 0, it means a reset was send to the client. A non-zero value means that the break was sent to the client.

Advise

Check network delays and connections.

2.13 SQL*Net break/reset to dblink

Same as SQL*Net break/reset to client, but now the break/reset message is sent to another server process over a database link.

Wait time

The actual time it takes for the break or reset message to come back from the other server process.

Parameters

driver id

See ‘SQL*Net break/reset to client’.

break?

See ‘SQL*Net break/reset to client’.

Advise

Check network delays and connections.

2.14 SQL*Net message from client

The server process (foreground process) is waiting for a message from the client process to arrive.

Wait time

The time it took for a message to arrive from the client since the last message was send to the client.

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes received by the server (foreground process) from the client.

Advise

Check network delays and connections.

2.15 SQL*Net message from dblink

The server process (foreground process) is receiving messages over a database link from another server process.

Wait time

The time it took for a message to arrive from another server (foreground process) since a message was send to the other foreground process.

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes received by the server (foreground process) from another fore ground process over a database link.

Advise

Check network delays and connections.

2.16 SQL*Net message to client

The server (foreground process) is sending a message to the client.

Wait time

The actual time the ‘send’ takes.

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes send by the server process to the client.

Advise

Check network delays and connections.

2.17 SQL*Net message to dblink

The server process (foreground process) is sending a message over a database link to another server process.

Wait time

The actual time the ‘send’ takes.

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes send by the server process to another server process over a database link.

Advise

Check the network delays and connections.

2.18 SQL*Net more data from client

The previous operation was also a send to the client and the server is doing another send to the client.

Wait time

The time waited depends on the time it took to receive the data (including the waiting time).

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes received from the client.

Advise

Check the network delays and connections.

2.19 SQL*Net more data from dblink

The foreground process is expecting more data from a data base link.

Wait time

The total time it takes to the read the data from the database link (including the waiting time for the data to arrive).

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes received.

Advise

Check the network delays and connections.

2.20 SQL*Net more data to client

The previous operation to the client was also a ‘send’. The server process is sending more data/messages to the client.

Wait time

The actual time it took for the ‘send’ to complete.

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes that are being send to the client.

Advise

Check the network delays and connections.

2.21 SQL*Net more data to dblink

The previous operation over this database link was also a ‘send’. So the event indicates that the server is sending data over a database link again.

Wait time

The actual time it takes to send the data to the other server.

Parameters

driver id

See ‘SQL*Net break/reset to client’.

#bytes

The number of bytes are send over the database link to the other server process.

Advise

Check the network delays and connections.

2.22 WMON goes to sleep

This is the UNIX specific Wait Monitor, that can be used to reduce the number of system calls related to setting timers in Oracle. You need to set an init.ora parameter that enables the WMON process.

Wait time

Depends on the next timeout.

Parameters

None.

Advise

You need to set the init.ora parameter reduce_alarm (Port specific). The frequency of the timeout is controlled by _wakeup_timeout (specified in 1/100 sec., default is 100).

2.23 batched allocate scn lock request

The Oracle7 foreground is waiting on another process to allocate an System Commit Number (SCN). If the foreground timed out waiting on a process to get the SCN, it will get the SCN ourselves.

Wait time

The wait time is 1 second on the assumption that an SCN allocate should normally take a lot less than that. (If we wake up due to a post then we do not allow for possibility of a dead process).

Parameters

None.

Advise

Check for a high number of time-outs on this event. To many time-outs basically indicates that there may be to much congestion on the SC resource or DLM in general. The following statistics from **v$sysstat** are related:

next scns gotten without going to server

next scns gotten without going to dlm

Unnecessary process cleanup for SCN batching

calls to kcmgas

2.24 buffer busy waits

Wait until a buffer becomes available. This event happens because a buffer is either being read into the buffer cache by another session (and the session is waiting for that read to complete). Or the buffer is the buffer cache, but in a incompatible mode.

Wait time

Normal wait time is 1 second. If we have been waiting for an exclusive buffer the last wait then we wait 3 seconds this wait.

Parameters

file#

This is the file number of the data file that contains the block that Oracle7 needs to wait for. To find the name of this file enter:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the block number of the block that Oracle7 needs to wait for. The block number is relative to the start of the file. To find the object that this block belongs to enter:

```
select name, kind
  from ext_to_obj_view[4]
 where file#  = file#
   and lowb  <= block#
   and highb >= block#;
```

To determine the type of the block (segment header, free list group block, etc.):

```
select 'segment header'
  from dba_segments
 where dbafil = file#
   and dbablk = block#;

select 'freelist group'
  from dba_segments
 where dbafil = file#
   and block# between dbablk and dbablk + nfl;
etc.
```

id

Basically the buffer busy wait event is called from different places in the Oracle7 kernel. Each place in the kernel points to different reason:

TABLE 8. Buffer Busy Waits id's.

Id	Reason
0	A block is being read
1003	Block is being read, probably with undo information for rollback

[4].\tThis view is defined in ?/rdbms/admin/catparr.sql in Oracle7 Release 7.2. This is a view that gives a more accurate view of that the table **ext_to_obj**, but is slower.

1007	Trying to get a new block
1010	Trying to get a buffer in Share mode, but a modification has started on the buffer that has not yet been completed.
1012	A modification is happening on a SCUR or XCUR buffer, but has not yet completed
1012 (dup.)	Trying to access a CURRENT block, but a modification has started on the buffer that has not yet been completed.
1013	Block is being read by another session, so we have to wait until the read is completed. This may also occur after a buffer cache assumed deadlock. The kernel can't get a buffer in a certain amount of time and assumes a deadlock. There for it will read the CR version of the block.
1014	Block is being read by another session, so we have to wait until the read is completed.
1016	The session wants the block in SCUR or XCUR mode. If we are in discrete TX mode, wait for the first time and the second time escalate the block as a deadlock. In that case the **"exchange deadlocks"** system statistics is incremented and we yield the CPU for the event **buffer deadlock**.

Advise

The fixed view **v$waitstat** contains information on per block class basis:

```
select *
  from v$waitstat;
```

There is also an internal fixed view that shows the waits per file (**x$kcbfwait**):

```
select name, count
  from x$kcbfwait, v$datafile
 where indx + 1 = file#;
```

The file that has loads of waits should be checked for the objects in that file. The objects should be checked for the kind of operations that are performed on them (select, insert, update and delete). If there is any concurrent update, delete or insert taking place, check the number of freelists on your object. Also checking the **pctfree** and **pctused**. Setting a higher initial **initrans** could also help concurrency. A large **db_block_size** may increase the number of '*buffer busy waits*', because there is more information in the block that is accessed in parallel by many different processes.

To reduce buffer busy waits on:

data blocks	Change pctfree and/or pctused. Check for 'right-hand-indexes' (indexes that get inserted into at the same point by many processes). Up initrans. Reduce number of rows per block.
segment header	Use freelists or increase of number of freelists. Use freelist groups (even in single instance this can make a difference).
freelist blocks	Add more freelists. In case of Parallel Server make sure that each instance has its own freelist group(s).
undo header	Add more rollback segments (when running in Exclusive mode). Set transactions per rollback segment to 1 (Exclusive mode).
undo block	Consider making rollback segments larger in Exclusive mode or increase number of PCM locks in gc_rollback_locks (in Parallel Server Mode only).

2.25 buffer deadlock

This event will be hard to catch. We don't really wait on it. We only yield the CPU. So the chances of catching this event are pretty slim. This is not an application induced deadlock, but an assumed deadlock by the cache layer. The cache layer basically can't get a buffer in a certain mode within a certain amount of time.

Wait time

0 Seconds, we are only giving up the CPU (yield it) and we end up at the end of the CPU run queue again (Port Specific).

Parameters

class

The class of the block basically tells what the block contents is going to be used for. In the table below we list all the different classes that are possible in Oracle.

TABLE 9. Block Classes

Class	Block Type
1	Data Block
2	Sort Block
3	Save Undo
4	Segment header block
5	Save Undo Segment header block
6	Free List block
7 + (*n* * *2)*	Undo segment header block ((System) Rollback Segment Header)
8 + (*n* * 2)	Undo segment block

If *n* is 0, the class is pointing at the system rollback segment. The following query will display the class of the block:

```
select trunc(p2/10)
  from v$session_wait
 where event='buffer deadlock';
```

mode

The mode of the block tells in what mode the block is owned.

TABLE 10. Block/Buffer modes

Mode	Means
0	NULL
1	Current Share (SCUR)
2	Current Exclusive (XCUR)
3	Consistent Read (CR)
4	Consistent Read Examination mode (CRX)
5	Current, Exclusive new block (NEW)

The following query will display the block mode:

```
select mod(p2, 10)
  from v$session_wait
 where event='buffer deadlock';
```

flag

The flag points to the internal flags used by the Oracle7 kernel to get this block. They are for internal use only:

TABLE 11. Flags passed down with the buffer get.

Flag	Meaning
0x1	Block is being used in sequential scan only
0x2	Block not needed after release
0x4	Release block after operation
0x8	Get the block in exclusive mode
0x10	Don't log changes (temporary tables)
0x20	Called by sort which will get the block and modify it directly (Mode is New or Exclusive)
0x40	Sort is finished with the block, free the buffer
0x80	This is a block cleanout
0x100	Return a NULL ptr on a corrupt block rather than blowing off
0x200	Block will likely change soon.
0x400	Not used.
0x800	Internal flag
0x1000	Internal flag
0x2000	Flag is set when Oracle7 has to wait on block.
0x4000	User supplied buffer instead of buffer that is in buffer cache.

dba

This is the dba of the block that we deadlock on:

```
select name, kind
  from ext_to_obj_view
 where file#  = data_block_address_file(dba)[5]
   and lowb  <= data_block_address_block(dba)
   and highb >= data_block_address_block(dba);
```

Advise

We should be more concerned with the number of deadlocks over a given time period, rather than the characteristics of an individual wait. DBA could be important to see on which objects buffer deadlocks are happening.

[5].\tThe data_block_address_file procedure needs to be prefixed with the package name dbms_utility. The package is defined in ?/rdbms/admin/dbmsutil.sql. These procedures exists in Oracle version 7.2 or higher. Before 7.3 Production you may run into problem into bug 314564, add 'pragma restrict_references (data_block_address_file, WNDS, RNDS ...)' into dbmsutil.sql.

2.26 buffer for checkpoint

The buffer couldn't be checkpointed, because it is being modified by some process. This means that after the wait we will scan the whole buffer cache again. This could happen during a database close or after a user does a local checkpoint. During this situation the database can't be closed.

Wait time

1 Second.

Parameters

dba

The dba points to the dba of the block that is currently in the buffer. With the following SQL statement one can determine the name of the object the block belongs to.

```
select name, kind
  from ext_to_obj_view
 where file#  = data_block_address_file(dba)
   and lowb  <= data_block_address_block(dba)
   and highb >= data_block_address_block(dba);
```

It may also be important to check the number of versions in the cache of this buffer. A large number of versions indicates a hot buffer. If a large percentage of the buffers in the cache contain different versions of this block, it could indicate a host spot. Run the following query:

```
select count(*)
  from x$bh
 where dbafil = data_block_address_file(dba)
   and dbablk = data_block_address_block(dba);
```

state

State refers to the status of the buffer contents. It can be one of the following states:

TABLE 12. Buffer States.

State	Reason
0	free (FREE)
1	exclusive current (XCUR)
2	shared current (SCUR)
3	consistent read buffer (CR)
4	buffer is being read (READ)
5	media recovery (MREC)
6	instance recovery (IREC)

mode

Mode refers to the lock mode held by the users of the buffer. The mode can be one of the following:

TABLE 13. Buffer Lock Modes.

Mode	Meaning
0	NULL
1	Current Share
2	Current Exclusive

3	Consistent Read
4	Consistent Read Examination mode
5	Current, Exclusive New block.

buffer#

The buffer# is the index in the x$bh table:

```
select *
  from x$bh
 where indx = buffer#;
```

Advise

Some processes is still modifying the buffer. Find the name of the object and see which process/session is accessing this object. You can do this by doing a systemstate dump. You will find the buffer handle SO, which will point to the process SO. This happens mostly with a hot block that needs checkpointing. In normal circumstances, user processes will not wait under this event; rather they will wait for "log file space/switch". However during shutdown and user checkpoint, this event is used, in conjunction with "checkpoint completed", when a buffer needs to be written to disk. This event is most likely to occur when we have a hot block which needs checkpointing.

2.27 buffer latch

Waiting on the buffer hash chain latch. Primarily used in the dump routines.

Wait time

1 Second.

Parameters

latch addr

The virtual address in the SGA where this latch is located. You can find the name of this latch with the following command:

```
select *
  from v$latch a, v$latchname b
 where addr = latch addr
   and a.latch# = b.latch#;
```

chain#

This is the index into array of hash chains. When chain is 0xffffffff we are waiting on the LRU latch. Else chain is the index in the number of hash chains. Database buffers are basically hashed by their DBA to a particular hash chain. The number of hash chains is calculated by dividing the number of buffers by 4 and rounding up to the first prime number. So a thousand db_block_buffers, would mean 251 hash hash buckets. You can also dump the SGA in oradbx and find the variable kcbnhb (hash buckets) and kcbnbh (number of buffers). Dumping the SGA with the following SQL statement is also possible:

```
alter session
      set events 'immediate trace name heapdump level 10';
```

Or if you have access to x$ksppi:

```
select ksppinm, ksppivl
  from x$ksppi
 where ksppinm = '_db_block_hash_buckets';
```

This will display the number of hash buckets before it has been converted to a prime number.

Advise

This is only a problem for dumping trace information. So ignore this event unless a process is hanging on this event.

2.28 buffer read retry

This will only happen if the instance is mounted in shared mode (parallel server). During the read of the buffer the contents changed. This means that either the version number, dba or the incarnation and sequence number stored in the block didn't match any more or that the checksum on the block doesn't match the checksum in the block. So we will re-read the block (this may fail up to 3 times), then we will assume corruption and dump the corrupt block in the trace file.

Wait time

The wait time is the elapse time of the read.

Parameters

file#

This is the file that Oracle7 is trying to read the block from. With the following SQL statement one can determine the name of the file:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the number of the block that Oracle7 is currently reading. If the block number is not 1, use the following SQL statement to find the object that the block belongs to:

```
select name, kind
  from ext_to_obj_view
 where file#  = file#
   and lowb  <= block#
   and highb >= block#;
```

Advise

This basically could happen because there was some kind of media error. It could indicate a potential media problem. Or in Parallel Server one instance was reading a block, while another instance was writing the same block.

2.29 checkpoint completed

A session is waiting for checkpoint to complete. This could happen for example during a close database or a local checkpoint.

Wait time

5 Seconds.

Parameters

No parameters.

Advise

If this happens frequently, you may want to increase the time between checkpoints by checking the init.ora parameter db_block_checkpoint_batch:

```
select name, value, isdefault
  from v$parameter
 where name = 'db_block_checkpoint_batch'
```

The value should be large enough to take advantage of parallel writes. The DBWR uses a write batch that is calculated like this:

('*db_files*' * '*db_file_simultaneous_writes*')/2

The write_batch is also limited by two other factors:

- a port specific limit on the numbers of I/Os (compile time constant).
- 1/4 of the number of buffers in the SGA.

The **db_block_checkpoint_batch** is always smaller or equal to the **_db_block_write_batch**.

2.30 client message

This event indicates that the Oracle7 server (foreground process) is waiting for a message (via SQL*Net) from the client process. This means the Oracle7 server process is idle and waiting for a SQL*Net packet.

Wait time

The wait time can be indefinite.

Parameters

two-task?

0, if not running two-task. 1, if running two-task.

driver id

The address of the function that does the disconnect for the protocol selected.

Advise

Basically the foreground process is waiting for the client to send a message. If the client has send a message and the foreground process is still waiting on this event, check the network connection if running client/server.

2.31 control file parallel write

This happens while Oracle7 is writing physical blocks to all controlfiles. This happens when e.g.:

- Oracle7 starts a controlfile transaction (to make sure that the controlfiles are up to date if Oracle7 would crash before committing the controlfile transaction).
- When Oracle7 commits a transaction to a controlfile.
- When changing a generic entry in the controlfile, write the new value to all controlfiles.

Wait time

The wait time is the time it takes for all writes to all controlfiles to finish.

Parameters

files

This indicates the number of controlfiles that Oracle7 is writing in parallel too.

blocks

This indicates the number of controlfiles that Oracle7 is writing in parallel too.

requests

This indicates the number of controlfiles that Oracle7 is writing in parallel too.

Advise

Basically *files, blocks* and *requests* all will have the same value and it is the number of real requests that will be written in parallel.

2.32 control file sequential read

Reading from the controlfile. This happens in many cases, e.g. while:

- making a backup of the controlfiles.
- the shared info (between instances) from the controlfile.
- reading other blocks from the controlfiles.
- reading the header block.

Wait time

The wait time is the elapse time of the read.

Parameters

file#

This identifies the controlfile that Oracle7 is reading from and with the following SQL statement one can determine the name of the controlfile:

```
select *
  from x$kcccf
 where indx = file#
```

block#

Blocknumber in the controlfile from where we are starting to read. The blocksize is the as the physical blocksize of the port (normally 512 bytes, some UNIX ports have 1 Kilobytes or 2 Kilobytes).

blocks

The number of blocks that we are trying to read.

Advise

If the wait time is too long (more then average I/O speed), check if the controlfiles are not a disk that is too busy. This could really impact Parallel Server performance as some of the synchronization between instances is done through the controlfiles.

2.33 control file single write

This write is done to write the shared info of the controlfile to disk. This is atomic operation protected by an instance (CF) if Oracle7 is started shared, so that only one instance can do this for a Parallel Server database.

Wait time

The wait time is the elapse time of the write.

Parameters

file#

This identifies the controlfile that Oracle7 is currently writing too and with the following SQL statement one can determine the name of the statement:

```
select *
  from x$kcccf
 where indx = file#;
```

block#

Blocknumber in the controlfile from where we are starting to write. The blocksize is the as the physical blocksize of the port (normally 512 bytes, some UNIX ports have 1 Kilobytes or 2 Kilobytes).

blocks

The number of blocks that we are trying to read.

Advise

If the wait time is too long (more than average I/O speed), check if the controlfiles are not a disk that is too busy. This could really impact Parallel Server performance as some of the synchronization between instances is done through the controlfiles.

2.34 conversion file read

This event will showup during a the creation of a V7 controlfile as part of converting a database to V7 from V6.

Wait time

The wait time is the elapse time of the read.

Parameters

file#

This identifies the controlfile that Oracle7 is currently writing too and with the following SQL statement one can determine the name of the statement:

```
select *
  from x$kcccf
 where indx = file#;
```

block#

Blocknumber in the controlfile from where we are starting to read. The blocksize is the as the physical blocksize of the port (normally 512 bytes, some UNIX ports have 1 Kilobytes or 2 Kilobytes).

blocks

The number of blocks that we are trying to read.

Advise

This is a one time only operation.

2.35 db file parallel write

This will show up in the DBWR. It tells that the DBWR is doing a parallel write to ‘files’ and ‘blocks’ are written. ‘requests’ show the real number of I/Os that are being done. When the last I/O has gone to disk it will end the wait.

Wait time

Wait until all the I/Os are completed.

Parameters

files

This indicates the number of files that Oracle7 is writing too.

blocks

This indicates the total number of blocks to be written.

requests

This indicates the total number of I/O requests, which will be the same as *blocks*.

Advise

If the waiting is too long, it could indicate that there is some kind of I/O bottleneck on the system. Compare the wait time with other I/O times on the system. Also to speed the wait time it may be a good idea to put the database files on different devices, so that the write will happen in parallel to these devices. The total wait time should be divided by the number of devices. Also the internal write batch could be too big, try reducing the **_db_block_write_batch**.

2.36 db file scattered read

Same type of event as **"db file sequential read"**, except that Oracle7 will read multiple data blocks.

Wait time

The wait time is the actual time it takes to do all the I/Os.

Parameters

file#

This is the file# of the file that Oracle7 is trying to read from. The name of the file can be displayed with the following SQL command:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the starting block number in the file from were Oracle7 starts reading the blocks. The name of the object can be found with the following SQL statement:

```
select name, kind
  from ext_to_obj_view
 where file#  = file#
   and lowb  <= block#
   and highb >= block#;
```

blocks

This parameter specifies the number of blocks that Oracle7 is trying to read from the *file#* starting at *block#*.

Advise

This event is normally an indicator that a Full Table Scan is happening on the object identified before. Also the number of blocks that can be read is dependent on the init.ora parameter **db_multi_block_read_count**. There is a platform specific maximum to this parameter (normally it is 64K/db_block_size).

2.37 db file sequential read

This event shows a wait for a foreground process while doing a sequential read from the database. Also used for rebuilding the controlfile, dumping datafile headers and generally getting the database file headers.

Wait time

The wait time is the actual time it takes to do the I/O.

Parameters

file#

This is the file# of the file that Oracle7 is currently reading from. With the following SQL statement one can find the name of the file:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the number of the block that Oracle7 is currently writing. If the block number is not 1 (indicates Oracle7 File Header Block), use the following SQL statement to find the object that the block belongs to:

```
select name, kind
  from ext_to_obj_view
 where file#  = file#
   and lowb  <= block#
   and highb >= block#;
```

blocks

This is the number of blocks that Oracle7 is trying to read (should be 1).

Advise

The fileheader of a Oracle file is always block 1. The platform specific file header is block 0. If the *block#* is not 1, it points to a block in an object.

2.38 db file single write

This event is used to wait for the writing of the file headers.

Wait time

The wait time is the actual time it takes to do the I/O.

Parameters

file#

This is the file# of the file that Oracle7 is currently writing to. With the following SQL statement one can find the name of the file:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the number of the block that Oracle7 is currently writing. If the block number is not 1, use the following SQL statement to find the object that the block belongs to:

```
select name, kind
  from ext_to_obj_view
 where file#  = file#
   and lowb  <= block#
   and highb >= block#;
```

blocks

This is the number of blocks that Oracle7 is trying to write in *file#* starting at *block#*.

Advise

The fileheader of a Oracle7 file is always block 1. The platform specific header is block 0. If the block number is not 1, it points to a block in an object.

2.39 direct access I/O (direct loader I/O, pre 7.3)

During Direct Load operations the data is asynchronously written to the database files. At some stage Oracle7 needs to make sure that all outstanding asynchronous I/O have been flushed to disk. This can also happen if during a direct load no more slots are available to store outstanding load requests (a load request could consist of multiple I/Os).

Wait time

10 Seconds. Oracle7 will be posted by the completing asynchronous I/O. So will never wait the 10 seconds each time. Oracle7 waits in a tight loop until all outstanding I/Os have completed.

Parameters

descriptor address

This is a pointer to the I/O context of outstanding direct I/Os that we are currently waiting on.

first dba

The dba of the oldest I/O in the context referenced by descriptor address.

block cnt

Number of valid buffers in the context referenced by descriptor address.

Advise

Consider striping the database files over many different disks and placing them on a Raw Device (if running UNIX) to get a higher throughput. Also the stripsize is important. Too big could mean that all different I/O's still go to one single physical disk.

2.40 dispatcher shutdown

During shutdown immediate or normal, the shutdown process must wait for all the dispatchers to shutdown. As each dispatcher is signaled, we wait on this event until the requested dispatcher is no longer alive.

Wait time

1 Second.

Parameters

waited

Gives the cumulative wait time. After 5 minutes, we write to the alert and trace files to indicate that there might be a problem.

Advise

Check the alert.log file and trace files, to see if there is a problem. Optionally kill the dispatcher processes by hand.

2.41 dispatcher timer

This basically means that the dispatcher is idle and waiting for some work to arrive.

Wait time

60 Seconds.

Parameters

sleep time

This is the intended sleep time, dispatcher will return sooner to work if it is posted by either data arriving on the network or by a post from a shared server process to send data back to the client.

Advise

If there are a lot of time-outs on this event, it basically means that the dispatcher is not very busy. If on the other hand the number of time-outs is low, it could mean that the dispatchers are (very) busy.

2.42 dupl. cluster key

It is possible for a race condition to occur when creating a new cluster key. If it is found that another process has put the cluster key into the data/index block, then we wait and retry. The retry should then find a valid cluster key.

Wait time

1/100 Second.

Parameters

dba

This is the dba of the block that we are trying to insert a cluster key into:

```
select name, kind
  from ext_to_obj_view
 where file#  = data_block_address_file(dba)
   and lowb  <= data_block_address_block(dba)
   and highb >= data_block_address_block(dba);
```

Advise

This shouldn't happen that often and if it does it only indicates some contention during insert into a cluster. To reduce the contention make sure that different processes insert with different key ranges (partition).

2.43 enqueue

Oracle7 is waiting for a local (vs. global) enqueue (vs. latch). The wait is dependent on the name of the enqueue (See Chapter 3).

Wait time

Depends on name.

Parameters

name

The name of enqueue is always two characters. These enqueues are used internally by Oracle7 to synchronize access to internal resources. Look in table 4 for an overview of possible names (DFS lock enqueue acquisition). With the following SQL statement one can find the name of the enqueue that the sessions are trying to get:

```
select chr(bitand(p1, -16777216)/16777215)||
       chr(bitand(p1,16711680)/65535) Enqueue
  from v$session_wait
 where event = 'enqueue';
```

mode

The mode is the requested mode by the session. See table 3 for an overview of all possible lock modes. With the following SQL statement one can determine the lock mode:

```
select bitand(p1, 65535)
  from v$session_wait
 where event = 'enqueue';
```

id1

P2 will give the first identifier of the enqueue. See chapter 3 for the meaning of that identifier, it will depend on the name of the enqueue.

id2

P3 will give the second identifier of the enqueue. See chapter 3 for the meaning of that identifier, it will depend on the name of the enqueue.

Advise

enqueue conversions

enqueue deadlocks

enqueue releases

enqueue requests

enqueue time-outs

Latch: enqueue_resources

enqueue_hash

int.ora: enqueues

_enqueue_locks

_enqueue_hash

_enqueue_debug_multi_instance

2.44 free buffer waits

This will happen if:

- All buffer gets have been suspended. This could happen when a file was read-only and is now read-write. All the existing buffers need to be invalidated since they are not linked to lock elements (needed when mounted parallel (shared)). So cache buffers are not assigned to dba's until the invalidation is finished.
- This could also happen when you need a buffer from the SGA for a block in CR, READING or any of the recovery modes. Basically you will post the DBWR to make some free buffers.
- This also happens after inspecting '*free buffers inspected*' buffers. If no free buffer was found, Oracle7 waits for one second and will try to get the buffer again (depends on the context).

Wait time

1 Second.

Parameters

file#

This the file in which Oracle7 is trying to get a free block. The name of the file can be determined with the following SQL statement:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the block# in the file that Oracle7 is trying to read into a buffer that currently is not available. With the following SQL statement one can determine to which object the block belongs:

```
select name, kind
  from ext_to_obj_view
 where file# = file#
   and lowb <= block#
   and highb >= block#;
```

Advise

The DBWR is not writing enough buffers to disk. Make sure that the I/O load is evenly distributed across all disks, use O/S monitor tools or look at v$filestat:

```
select name, phyrds, phywrts
  from v$filestat a, v$datafile b
 where a.file# = b.file#
```

Also look for files that have full table scans:

```
select name, phyrds, phyblkrd, phywrts
  from v$filestat a, v$datafile b
 where a.file# = b.file#
   and phyrds != phyblkrd
```

Check your application to make sure that is what you intended and that you don't miss an index. Always try to use the O/S striping software to distribute database files over as many disks as one can.

Disk Sorts are known to cause a flood of dirty buffers that will need to written out. It is important to stripe the datafiles belonging to the TEMP tablespace to over many different disks. Also during a checkpoint the SORT blocks are not checkpointed and they are thus only written in the normal write batch. If the write batch is full with SORT blocks, the other dirty blocks can't be written and foregrounds will have to wait for free buffers.

2.45 free global transaction table entry

The Oracle7 kernel is waiting for a free slot in the global transaction table (used by the Distributed Database Option). It will wait for 1 second and try again.

Wait time

1 Second.

Parameters

tries

The number of times the Oracle7 kernel tried to find a free slot in the global transaction table.

Advise

Look at the init.ora parameter 'distributed_transactions', consider a larger number:

```
select name, value, isdefault
  from v$parameter
 where name = 'distributed_transactions'
```

2.46 free process state object

Used during the creation of a process. Oracle7 will scan the process table and look for a free process slot. If none can be found PMON is posted to check if all the processes currently in the process table are still

alive. If there are dead processes, PMON will clean them up and make the process slot available to new processes. The waiting process will then rescan the process table to find the new slot.

Wait time

1 Second.

Parameters

None.

Advise

If this happens often, one should consider increasing the size of the process table (**processes** init.ora parameter).

2.47 inactive session

This event is used for two purposes:

- Switching sessions

  If a time-out period has been specified, then wait that amount of time for the session to be detached from another process.

- Killing sessions

  From either 'kill session' or internal request. Having posted a session that it should kill itself, wait for up to 1 minute for the session to die.

Wait time

1 Second.

Parameters

session#

This is the number of the inactive session. One can find the session with the following SQL statement:

```
select *
  from v$session
 where sid = session#;
```

waited

This is the total amount of time Oracle7 has waited for this session to die.

Advise

If the session can't be killed an Oracle error 31 will be signalled. A process can only be killed or switched if the user call that it was executing has finished.

2.48 index block split

While trying to find an index key in an index block, Oracle7 noticed that the index block was being split. Oracle7 will wait for the split to finish and try to find the key again.

Wait time

Oracle7 will yield the CPU, so there is no actual waiting time here.

Parameters

rootdba

This is the block that Oracle7 is trying to split. With the following SQL statement one can find the name of the index that this block belongs to:

```
select name, kind
  from ext_to_obj_view
 where file#  = data_block_address_file(rootdba)
   and lowb  <= data_block_address_block(rootdba)
   and highb >= data_block_address_block(rootdba);
```

level

This is the level of the block that Oracle7 is trying to split in the index. The leaf blocks are level 0. If the level is > 0, it is a branch block. (The root block can be considered a special branch block).

childdba

This is the new block that will contain half of the previous block(rootdba). One can also check to see if this childdba is part of the object:

```
select name, kind
  from ext_to_obj_view
 where file#  = data_block_address_file(childdba)
   and lowb  <= data_block_address_block(childdba)
   and highb >= data_block_address_block(childdba);
```

Advise

The amount of time waited for this event is not as important as the number of times that one has to wait. Always check the name of the index and check in the application to see what kind of operations are happening.

2.49 instance recovery

Sessions are waiting for SMON to finish instance or transaction recovery. It is also used in 7.3 to perform sort segment cleanup.

Wait time

The wait time may vary and basically depends on the amount of recovery to do, but the timeout per wait is 1 second in loop until the recovery is done or the

Parameters

undo segment#

If this is 0, SMON is most likely performing instance recovery. If P1 > 0, one can find the undo segment by performing the following query:

```
select *
  from v$rollstat
 where usn = undo segment#;
```

Advise

This can be instance recovery on the cache or the transactions. The amount of recovery depends on the amount of redo (determined by checkpoint frequency) and outstanding undo (determined by number of active transaction at time of failure).

2.50 instance state change

Wait for SMON to enable or disable cache or transaction recovery. This normally happens during alter database open/close.

Wait time

The amount of time depends on the amount of time the action takes (amount of recovery to do).

Parameter

layer

This value can be 1 or 2. If 1, it means that the transaction layer wants transaction recovery to be performed. If 2, it means that cache recovery will be performed.

value

This value can be 0 (disable) or 1 (enable).

waited

This is the number of seconds waited so far.

Advise

The instance state change are the result of a alter database command (like open, close etc.).

2.51 kcl bg acks

Basically you are waiting for the background LCK process(es) to finish what they are doing.

- lock recovery
- initializing the locks (start-up)
- finalizing the locks (shutdown).

Wait time

10 Seconds.

Parameters

count

The count is the number of LCK processes that have finished.

loops

The loops is the number times the process had to wait for the LCK processes to finish what they were doing.

Advise

Check the number of gc_lck_procs:

```
select value
  from v$parameter
 where name = 'gc_lck_procs'
```

2.52 latch activity

This event is used as part of the process of determining whether a latch needs to be cleaned up.

Wait time

5/100 to 1/10 of a second.

Parameters

address

This is the address of the latch that is being checked.

number

This is the latch number of the latch that has activity:

```
select *
  from v$latchname
 where latch# = number;
```

process#

If this is 0, it is the first phase of the in-flux tests.

Advise

Check for dead processes. There could also be an issue that processes get preempted a lot by processes trying to acquire a latch.

2.53 latch free

The process is waiting for a latch that is currently busy (held by another process).

Wait time

The wait time increases exponential and doesn't include spinning on the latch (active waiting). The maxi-mum wait time also depends on the number of latches that the Oracle7 process is holding. There is an exponential back off up to 2 seconds.

Parameters

address

This is the address of the latch that the process is waiting for.

number

This is the latch number that indexes in the v$latchname view:

```
select *
  from v$latchname
 where latch# = number;
```

tries

This is basically a counter that counts the number of times we tried to get the latch (slow with spinning) and the process has to sleep.

Advise

Depending on the name of the latch, there is a different advise to give:

KCL freelist latch

This latch protects the PCM lock element freelist in Fine Grain Locking. There is one latch for the one freelist. Only Lock Elements that have the free bit set are on the freelist. The free bit gets set when a buffer gets unlinked from the Lock Element. Only the releasable locks are put on the freelist.

KCL name table latch

This latch protects the name translation for the Lock Element. The name translation uses the Data Block Address and the Class of the block to hash to a Lock Element. If the Lock Element is used for Fine Grain or DBA locking, the name of the lock will be the DBA of the block. For Hashed locking the name will be Lock Element Index.

NLS data objects

This latch protects the loading and unloading of NLS data in the SGA. The NLS data is cached in the SGA.

archive control

The archive control latch protects the archive destination string in the SGA. If a session wants to update or read the archive destination, it will get this latch to make sure that the session gets a consistent view.

cache buffer handles

This latch protects the State Objects that are needed by a process to make a change to a block/buffer. Before the change a buffer handle is acquired either from the process' private pool or the global pool if none are present. The access to the global pool is protected by this latch.

cache buffer chains

Before modifying a buffer the foregrounds need to get a latch to protect the buffer from being updated by multiple users. Several buffers will get hashed to the same latch. The buffers get hashed to a cache buffer chain latch depending on the DBA of the block that is in the buffer.

cache buffer lru chain

Protects the Least Recently Used list of cache buffers. For moving buffers around on this list the Oracle7 kernel needs to get this latch.

cache protection latch

During the cloning of a buffer the buffer header is atomically updated to reflect the new position in the buffer cache. Only one buffer at the time may be cloned, as it is protected by this latch. A lot of contention on this latch would indicate that there is a lot of concurrent cloning going on.

cached attr list

This latch protects a cache of attributes that are only used in Parallel Server. The attributes that are cached are for example the instance number. It also keeps track of the instance map (which instances are up and running). By caching this information it becomes available (relatively) to whole instance.

cost function

The cost function latch protects the resource limit table in the SGA. If multiple sessions would execute the alter resource cost statement , the new cost values would need to be protected when copied into the SGA.

dml lock allocation

This latch protects the list of State Objects (dml locks). Every time a transaction modifies a table, a DML lock is gotten and released when the change is committed. The number of State Objects for dml locks is determined by the init.ora parameter **dml_locks**.

enqueue hash chains

_enqueue_hash_chain_latches (default number of CPU's or cpu_count). Protects the individual hash buckets

enqueues

This latch is used to add and remove the enqueue State Object to a parent Object State. It is also used if there are no free state objects on enqueue hash chain. The hash chain is calculated by:

((type[0] << 5) + type[1] + id1 + id2) % # buckets.

error message lists

Used by the Parallel Query Option to signal errors to the Query Coordinator.

global transaction

This latch is used to control the setting of a savepoint in a transaction control block (stored in the SGA).

global tx freelist

Only used when the Distributed Database Option is enabled.

global tx hash mapping

Only used when the Distributed Database Option is enabled.

instance latch

Only used when the Parallel Server Option is enabled. This latch is used to protect the freelist of Lock Context State Objects. And it is used to protect the count of LCK processes that are participating in a startup, shutdown or recovery phase.

latch wait list

Certain latches that are held a long time have a waiting list. This waiting list is protected by a single latch for all latches.

library cache

library cache load lock

library cache pin

list of block allocation

This is the latch that proctects the pool of State Objects for “cleanout on commit”.

lock element parent latch

messages

There is a pool of message blocks in the SGA. This pool is controlled by the init.ora parameter **_messages**. If a process needs a message it will get the latch, search a linked list for a free message block, unlink this message block , put this message on the message queue for the intended process and then release this latch. The event **rdbms ipc message** indicates that a process is waiting for a message to arrive on his message queue. The event **rdbms ipc message reply** is used to indicate that a message is expected back by the sending process.

modify parameters values

This latch protects the modification of changeable/dynamic init.ora parameters.

multiblock read objects

This latch protects the allocation and release of multi block read objects from the SGA.

parallel query alloc buffer

parallel query stats

process allocation

process queue

process queue reference

query server freelists

query server process

redo allocation

The redo allocation latch is used to allocate the redo buffers in the redo log buffer. It is also gotten we Oracle7 needs to advance to SCN as part of a checkpoint. This way we can make sure that nobody can use the new SCN number.

redo copy

When a redo buffer has been allocated and the size of the redo is larger than **log_small_entry_max_size**, the kernel will allocate a redo copy latch. The number of redo copy latches is controlled by the init.ora parameter **log_simultaneous_copies** (defaults to the number of CPUs). The LGWR will get all the redo copy latches before it will write redo to make sure that nobody is writing in redo buffers that are currently written.

row cache objects

This latch protects the access of the datadictionary cache in the SGA. When loading, referencing and freeing objects in the datadictionary cache you need to get this latch.

sequence cache

This latch protects the lists of State Objects that are used for sequence numbers. Allocating from and Releasing to this freelist is protected with this latch.

session allocation

This latch is used for a lot of things. It protects the session State Object freelist. The size of this freelist is determined by the init.ora parameter **sessions**. It also controls the updating of the systemwide statistics (v$sysstat). Licensing information is also controlled by this latch. The allocation of **calls** is also protected by this latch.

session idle bit

Every time the session starts to execute a call, the active bit will be set by the session and when the session finishes the call, it will clear the active bit. The status field in **v$session** will change from ACTIVE to INACTIVE (protected by this latch).

session switching

This latch protects the switching from sessions in XA.

shared pool

sort extent pool

This latch protects the sort extents in the SGA. There is a cache of extents and descriptors. Access to these is protected by this latch. This latch will get used if a tablespace is marked temporary.

system commit number

This latch protects the SCN in the SGA. Any reference to a SCN (Recent, Current) means that you need to get this latch first.

trace latch

This latch protect the Trace State Objects in the SGA. The number of Trace State Objects is equal to the number of **processes**.

transaction allocation

This latch protects the pool of transaction state objects in the SGA. The number of Transaction State Objects is determined by the init.ora parameter **transactions**.

undo global data

This latch serializes the access to the Undo (aka Rollback) segment information in the SGA. Every time a session wants to know about the state of the Undo Segments, it has to get this latch.

user lock

This latch protects the user. It makes sure that a user can't be dropped while logging on or while logged on.

virtual circuit buffers

One latch per session

virtual circuit queues

This latch serializes the access to queues. Each dispatcher process has its own queue. There are a total of mts_max_dispatcher + 2 queues. Basically the dispatcher will put a virtual circuit on a queue that is going to be picked up by a shared server process.

virtual circuits

This latch protects the pools of Virtual Circuit State Objects. If the Dispatcher allocates a Virtual Circuit, it is incoming. If a Shared Server process allocates a Virtual Circuit, it is outgoing. Also the ownership of the Virtual State Circuit gets moved from the Dispatcher to Shared Server and vice-versa.

2.54 library cache load lock

Oracle7 tries to find the load lock for the database object so that it can load the object. The load lock is always gotten in Exclusive mode, so that no other process can load the same object. If the load lock is busy the session will wait on this event until the lock becomes available.

Wait time

3 Seconds or 1 second if PMON.

Parameters

object address

Address of the object being loaded.

lock address

Address of load lock being used. This is not the same thing as a latch or an enqueue, it is basically a State Object.

mask

Indicates which data pieces of the object that needs to be loaded.

TABLE 14.

Mask	Meaning
0x01	object/ datablock 0, object
0x02	data block 1, source
0x04	data block 2, diana
0x08	data block 3, pcode
0x10	data block 4, machine-dependent pcode
0x20	data block 5, error
0x40	data block 6, SQL context (cursor)
0x80	data block 7

namespace

TABLE 15.

Namespace	Meaning
0	Cursor

1	table/view/sequence/synonym/class/set/procedure/ function/package
2	body (e.g. package body)
3	trigger
4	index
5	cluster
6	object
7	pipe
127	no namespace

Advise

?

2.55 library cache lock

The library cache lock controls the concurrency between clients of the library cache by acquiring a lock on the object handle so that one client can prevent other clients from accessing the same object or the client can maintain a dependency for a long time (no other client can change the object). This lock is also gotten to locate an object in the library cache.

Wait time

Parameters

handle address

lock address

mode

namespace

Advise

x$kgllk

2.56 library cache pin

The library cache pin manages library cache concurrency. Pinning an object causes the heaps to be loaded into memory. If a client wants to modify or examine the object, the client must acquire a pin after the lock.

Wait time

Parameters

handle address

pin address

mode

namespace

Advise

x$kglpn

2.57 lock element cleanup

A PCM lock upconvert was issued but hasn't finished yet. If the upconvert has finished the lock context will be cleaned up. Before a lock element can be converted, a lock context (State Object) is allocated by the foreground. After the convert has completed (or is canceled) the lock context will be cleaned up.

Wait time

2 Seconds.

Parameters

file#

Identifies the file in which the block resides that the session is trying to get a PCM lock in a certain mode (either S or X) for. With the following SQL statement one can determine the name and the location of the datafile:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the block that the session wants a PCM lock for. With the following SQL statement one can find out to which object this block belongs:

```
select name, kind
  from ext_to_obj_view
 where file#  = file#
   and lowb  <= block#
   and highb >= block#;
```

lenum

Is used as an index into the lock element table:

```
select addr
  from x$le
 where indx = lenum;
```

We can also join x$le.addr to x$bh.le_addr to find the buffer header in the cache:

```
select a.*
  from x$bh a, x$le b
 where a.le_addr = b.addr;
```

Advise

?

2.58 lock element waits

Basically we are waiting to make sure that there is no activity on the lock element (any lock acquire or release or invalidation is in progress), before we continue.

Wait time

1 Second.

Parameters

file#

Identifies the file in which the block resides that the session is trying to get a PCM lock in a certain mode (either S or X) for. With the following SQL statement one can determine the name and the location of the datafile:

```
select *
  from v$datafile
 where file# = file#;
```

block#

This is the block that the session wants a PCM lock for. With the following SQL statement one can find out to which object this block belongs:

```
select name, kind
  from ext_to_obj_view
 where file#  = file#
   and lowb  <= block#
   and highb >= block#;
```

lenum

Is used as an index into the lock element table:

```
select addr
  from x$le
 where indx = lenum;
```

We can also join x$le.addr to x$bh.le_addr to find the buffer header in the cache:

```
select a.*
  from x$bh a, x$le b
 where a.le_addr = b.addr;
```

Advise

If a large number of waits show up on this event, it means that a lot of pinging is occurring either TRUE or FALSE. Check to see on what object a lot of pinging is going on and try to either partition or redistribute PCM locks.

2.59 log buffer space

Waiting for space in the log buffer because we are writing stuff into the log buffer faster than lgwr can write it out. Consider making the log buffer bigger if it is small, or moving the log files to faster disks such as striped disks

Wait time

1 Second normally, but 5 seconds if waiting for a Switch Logfile to complete.

Parameters

No Parameters.

Advise

Consider making the log buffer bigger if it is small, or moving the log files to faster disks such as striped disks

2.60 log file parallel write

Writing redo records to the redo log files from the log buffer.

Wait time

Time it takes for the I/Os to complete. Even though they are written in parallel, one needs to wait for the last I/O to be on disk before the parallel write is complete.

Parameters

files

Number of files written to.

blocks

Number of blocks to be written.

requests

Number of I/O requests.

Advise

Check the average I/O time in v$system_event for this event. When the I/O time is too high check to see if the disk that contains the log files is too busy.

2.61 log file sequential read

Waiting for the read from this logfile to return. This used to read redo records from the log file.

Wait time

Time it takes to complete the physical I/O (read).

Parameters

log#

This is the relative sequence number within a log group or all the log files (only when dumping the logfiles).

block#

Is the offset into the logfile in a port specific physical blocksize.

blocks

The number of blocks read.

Advise

?

2.62 log file single write

Waiting for the write to this logfile to complete. This is used to write the header of the logfile. This done during adding a log file member and when the writing sequence numbers are being advanced.

Wait time

Time it takes for the physical I/O (write) to complete.

Parameters

log#

This is the number of the group/log that Oracle7 is currently writing too:

```
select *
  from v$log a, v$logfile b
 where a.group# = b.group#
   and a.thread# = b.thread#
   and a.group# = log#;
```

block#

Is the offset into the logfile in a port specific physical blocksize.

blocks

The number of blocks written.

Advise

If this happens to often consider running with larger redo log files (does this make sense?).

2.63 log file space/switch

Oracle7 is waiting for space on the disk to write the redo from log buffer. We are not waiting for space in the log buffer or for the LGWR to finish a write. The only way to get more disk space is to do a log switch. If the user is waiting for this event it means that there is not enough space in the buffers, but enough in the files. LGWR will need to write the current buffer, before the user can proceed. Used as part of the 'alter system archive log change <scn>' command.

Wait time

Wait for up to 10 seconds.

Parameters

No Parameters.

Advise

You may want to increase the log_buffer init.ora parameter:

```
select value
  from v$parameter
 where name = 'log_buffer';
```

The value returned is in bytes. Then look at the total redo size and the number of commits:

```
select a.value / decode(b.value, 0, 1, b.value)
   from v$sysstat a, v$sysstat b
  where a.name = 'redo size'
    and b.name = 'user commits';
```

The value returned is the average redo size per commit. The next thing to find out is the number commits per second and multiply that with the average redo size. That would be your max log_buffer size to hold all transactions that are simultaneously active.

2.64 log file switch (checkpoint incomplete)

Waiting for a log switch because we cannot wrap into the next log because the checkpoint for that log has not completed. To speedup checkpoint consider making the buffer cache smaller, or increasing **db_block_checkpoint_batch**, or adding more DBWR processes.

Wait time

1 Second.

Parameters

No Parameters.

Advise

2.65 log file switch (archiving needed)

Waiting for a log switch because the log we will be switching into has not been archived yet. Check the alert file to make sure that archiving has not stopped due to a failed archive write. To speedup archiving consider addint more archive processes or putting the archive files on striped disks.

Wait time

1 Second.

Parameters

No Parameters.

Advise

2.66 log file switch (clearing log file)

Waiting for a log switch because log is being cleared due to a clear logfile command or implicit clear logfile executed by recovery.

Wait time

1 Second.

Parameters

No Parameters.

Advise

2.67 log file switch (switch logfile command)

Waiting for log switch because of user command to switch logfiles.

Wait time

1 Second.

Parameters

No Parameters.

Advise

2.68 log file switch (full log file)

Waiting for log switch because current log is full and lgwr needs to complete writing to current log and open the new log.

Wait time

1 Second normally, but 5 seconds it waiting for a Switch Logfile to complete.

Parameters

No Parameters.

Advise

2.69 log file sync

When a user session commits (or rollsback), the sessions redo information needs to be flushed to the redo logfile. The user session will post the LGWR to write the log buffer to the redo log file. When the LGWR has finished that it will post the user session.

Wait time

The wait time includes the writing of the log buffer and the post.

Parameters

buffer#

This is the buffer# inside the log buffer. The log buffer consists of smaller buffers (of physical block size).

Advise

Speed up the LGWR. Look at the average Redo Size per Commit.

2.70 log switch/archive

Used as part of the 'alter system archive log change <scn>' command. Oracle7 is basically waiting for the current log from an open thread other than our own to be archived.

Wait time

Wait for up to 10 seconds.

Parameters

thread#

This is the thread number of the thread that is currently archiving its current log. The following SQL statement will give more info on the thread:

```
select *
  from v$thread
 where thread# = thread#;
```

Advise

What to do ?

2.71 on-going reading of SCN to complete

Wait time

Parameters

Advise

2.72 parallel query create server

Used when creating/starting a Parallel Query Slave (P???).

Wait time

The time it takes to start all the requested Parallel Query Slaves.

Parameters

nservers

This is the number of Parallel Query Slaves that are being started.

sleeptime

Time it takes to get the processes started. (Minimum wait time)

enqueue

?

Advise

It will be cheaper to have a minimum number of slaves hanging around instead of starting and shutting down all the time. Consider setting **parallel_min_servers** to a higher value.

2.73 parallel query dequeue wait

The process is waiting for a message during a parallel execute

Wait time

The wait time depends on how quickly the message arrives. So wait times may vary, but it will normally be a short period of time.

Parameters

queue/reason

Before 7.2, it was queue which indicated the process queue to dequeue. With 7.2 and higher it shows the reason for dequeueing.

TABLE 16. Dequeue Reasons.

Reason	Meaning
	KXFP KXFPRDP DEQUEUE
	KXFP KXFPG1SG DEQUEUE
	KXFP CREDIT (GEB) DEQUEUE
	KXFP CREDIT (FREE) DEQUEUE
	KXFP CREDIT (ENQ) DEQUEUE
	KXFP KXFPQRTST DEQUEUE

sleeptime

loop

Advise

2.74 parallel query qref latch

Each Parallel Query Process has a parallel query qref latch, which needs to be acquired before the queue buffers can be manipulated.

Wait time

Wait up to 1 second.

Parameters

function

sleeptime

qref

Advise

2.75 parallel query server shutdown

During normal or immediate shutdown the Parallel Query Slaves are posted to shutdown cleanly. If after 10 seconds any Parallel Query Slave are still alive, they are just killed.

Wait time

Wait up to 1/2 second.

Parameters

nalive

This is the number of Parallel Query Slaves that are still alive.

sleeptime

The total sleeptime since Oracle7 started to wait on this event.

loop

The number of times Oracle7 has waited for this event.

Advise

?

2.76 parallel query signal server

This will only happen in Exclusive mode. The Query Coordinator is signalling the Query Slaves that an error has happened.

Wait time

Parameters

serial

error

nbusy

Advise

2.77 pending global transaction(s)

Should only happen during testing, wait for pending transactions to clear.

Wait time

30 Seconds.

Parameters

scans

Number of times the Oracle7 kernel has scanned the pending_trans$ table. Every time it will wait for 30 seconds (it will scan 20 times), and every time it will do a open, fetch and close cursor.

Advise

Event 10244.

2.78 pipe get

Oracle7 is waiting for a message to be received on the pipe or for the pipe timer to expire.

Wait time

There is a 5 second wake up (check) and the pipe timer set by the user.

Parameters

handle address

buffer length

timeout

This the pipe timer set by the user.

Advise

?

2.79 pipe put

Oracle7 is waiting for the pipe send timer to expire or for space to be made available in the pipe.

Wait time

There is the 5 second wakeup (check) and the user supplied timeout value.

Parameters

handle address

record length

timeout

This is the pipe timer set by the user.

Advise

?

2.80 pmon timer

This is main wait event for PMON. When PMON is idle it is waiting on this event.

Wait time

Up to 3 seconds, if not posted before.

Parameters

duration

This is the actual amount that the PMON process is trying to sleep. This parameter is only present for Oracle7 versions after 7.2.

Advise

?

2.81 process startup

Wait for a Multi Threaded Server (Shared Server), Dispatcher or other background process to start.

Wait time

Wait up to 1 second for a background process to start. If timed out, then re-wait until 5 minutes have passed and signal an error. If the process has started, it will acknowledge this.

Parameters

type

This number indicates the type of the background process:

TABLE 17. Type of background process

Type	Meaning
1	Regular background process
2	Multi-threaded server
3	Dispatcher

process#

This is the process number of the process being started. It is the Oracle7 process number which can be found in the v$process view:

```
select *
  from v$process
 where pid = process#;
```

waited

Cumulative time waited so for the process to start up.

Advise

If a lot of waits /timeouts show on this event, it could mean that the minimum number of processes (either Parallel Query Slaves, Dispatchers and Shared Servers), is set to low. Increase the corresponding init.ora parameters to set a higher minimum.

2.82 queue wait

The direct loader uses a queue of slots for managing buffers. These slots are used for read/write operations, and when a new slot is requested, but none are available, then we wait for wait. This wait will only occur if they underlying O/S is using Async I/O.

Wait time

Wait up to 1 second for slots to be freed in the circular direct loader I/O buffer queue.

Parameters

No Parameters.

Advise

Tune the Async I/O (O/S Parameters) and make sure that the files that are being loaded into are striped over different disks.

2.83 rdbms ipc message

Wait time

Parameters

timeout

Advise

2.84 rdbms ipc message block

This event basically means that all message blocks are in use.

Wait time

Wait up to 60 seconds.

Parameters

No Parameters.

Advise

If this happens often, try to increase the init.ora parameter messages.

2.85 rdbms ipc reply

This event is used to wait for a reply from one of the background processes.

Wait time

The wait time is specified by the user.

Parameters

from_process

This parameter only exists in release 7.1 and higher. It is the process that the waiting process is expecting a message from. With the following SQL statement one can get more information on that process:

```
select *
  from v$process
 where pid = from_process;
```

timeout

This the amount of time in seconds that this process will wait for a reply.

Advise

Check to see if the *from_process* is still running.

2.86 row cache lock

This event is used to wait for a lock on a data dictionary cache specified by "cache id". If one is running in shared mode (Parallel Server), the LCK0 is signalled to get the row cache lock for the foreground waiting on this event. The LCK0 process will get the lock asynchronously. In exclusive mode the foreground process will try to get the lock.

Wait time

In shared mode the foreground will wait 60 seconds for the LCK0 to get the lock, the foreground will wait in infinite loop until the lock has been granted (LCK0 will notify foreground).

In exclusive mode the foreground will wait 3 seconds, in a loop of 1000 iterations.

In either case PMON will wait for only 5 seconds.

Parameters

cache id

```
select *
  from v$rowcache
 where cache# = cache id;
```

mode

The mode the lock currently is held in.

request

The mode the lock is requested in.

Advise

If this event shows up a lot consider increasing the shared pool, so that more data dctionary can be cached.

2.87 scginq AST call

Called by Oracle7 to find the highest lock mode that is held on a resource.

Wait time

Wait up to 1/5th second, but keep on waiting until the NULL mode Acquisition AST has fired.

Parameters

No Parameters.

Advise

?

2.88 secondary event

Wait time

Parameters

event#

wait time

Advise

?

2.89 sequence# cache entry

Only in Parallel Server: wait for a sequence cache entry to become available.

Wait time

1/100th second.

Parameters

entry#

Advise

?

2.90 sequence# lock op

Only in Parallel Server: wait for the instance lock to be granted.

Wait time

Wait up to 5 seconds, but continue waiting for up to 15 minutes.

Parameters

entry#

Advise

?

2.91 single-task message

Wait time

Parameters

Advise

?

2.92 smon timer

This is the main idle event for SMON. SMON will be waiting on this event most of the time until it times out or is posted by another process.

Wait time

5 Minutes (300 seconds).

Parameters

sleeptime

This is the amount of time that SMON is trying wait on this event in seconds.

failed

This is the number of times SMON was posted or there was some kind of error.

Advise

?

2.93 timer in sksawat

Wait for Archiver async I/O to complete.

Wait time

1/100th Second.

Parameters

No Parameters.

Advise

?

2.94 trace continue

Wait time

Parameters

delay time

Advise

?

2.95 trace unfreeze

Wait time

Parameters

None.

Advise

?

2.96 trace writer I/O

Wait until a dirty buffer has been cleared by TRWR. This event will only be fired if the following init.ora parameters have been set:

_trace_archive_start = true

_trace_flushing = <number of buffers to keep clean>

Wait time

Parameters

None.

Advise

?

2.97 trace writer flush

Wait for to coordinate flushing of trace buffers. Processes are either waiting for the buffers to be flushed to disk or the TRWR process is sending a message to all other processes.

Wait time

? Seconds.

Parameters

None.

Advise

?

2.98 transaction

Wait for a blocking transaction to be rolled back. Continue waiting until the transaction has been rolled back.

Wait time

1 Second.

Parameters

undo seg#

```
select *
  from v$rollstat
 where usn = undo seg#
```

slot#

This is the slot# within the transaction table that is being used to store a transaction.

wrap#

Wrap or sequence number of the slot. For each new transaction increase this number.

count

Number of times that Oracle7 has waited on this transaction.

Advise

The waiting basically depends on the size of that transaction that is being rolled back. (*Can we see how much work was done and has to be rolled back?)*

2.99 undo segment extension

The undo segment is being extended or shrunk. The session has to wait until that has finished.

Wait time

1/100th second.

Parameters

segment#

```
select *
  from v$rollstat
 where usn = undo segment#;
```

The following query will show the affected transactions that have to wait until the rollback segment has been extended.

Advise

Don’t make rollback segments too small and also don’t always use the OPTIMAL clause. This will cause rollback segments to shrink and extent numerous times.

2.100 undo segment recovery

PMON is rolling back a dead transaction. Wait for that to finish.

Wait time

3 Seconds.

Parameters

segment#

This points to the name of the rollback segment within the v$rollstat view:

```
select *
  from v$rollstat
 where usn = undo segment#;
```

tx flags

These flags are the control flags for the transaction:

TABLE 18. Transaction control flags.

Flag	Meaning
0x1	TX has started storing collecting information
0x2	TX has forced the collecting information
0x4	Prepared TX needs distributed recovery
0x10	Rollback failed on this TX, SMON to recover
0x20	TX has rollback its updates

Advise

In Parallel Server mode the Global enqueue TA is acquired to recovery the transaction. If the TA lock can't be obtained wait on this event and try again.

2.101 undo segment tx slot

Wait for a transaction slot to become available within the selected rollback segment. Continue waiting until the slot is available.

Wait time

1 Second.

Parameters

segment#

```
select *
  from v$rollstat
 where usn = undo segment#
```

Advise

?

2.102 virtual circuit status

Wait time

Parameters

circuit#

```
select *
```

```
from x$kmcvc
where indx = circuit#;
```

status

Advise

2.103 write complete waits

The foreground is waiting for a buffer to be written. The write is caused by normal ageing or a cross instance call.

Wait time

1 Seconds.

Parameters

id

Id	Reason
1022	Trying to renew the buffer, but it is currently being written. The buffer is formatted in the cache for a new block (old block discarded).
1027	The buffer is in the middle of a write, wait max 1 second. Then rescan the cache.
1027 (dup)	Switch Current (same as previous).
1029	While trying to apply the changes to the block, the block is in the middle of a write. Wait until the block is available in a tight loop. Each wait is one second. The buffer being written is a current buffer.
1030	A block is marked as corrupted and then we wait until the block is on disk.
1030 (dup)	
1031	
1033	
1034	Cross Instance Write: This instance is trying to modify the block, while another instance is trying to get a current version. This instance waits max 1 second.
1035	Cross Instance Write: same as 1034, but the problem now is this happens during a hot backup.
1035 (dup)	Cross Instance Write:

file#

```
select *
  from v$datafile
 where file# = file#;
```

block#

```
select name, kind
  from ext_to_obj_view
```

```
where file#  = file#
  and lowb  <= block#
  and highb >= block#;
```

Advise

?

2.104 writes stopped by instance recovery

This event will block a foreground process until the DBWR of the instance is allowed to do a write again. This will only happen in Parallel Server. SMON will get the IR instance lock and then do a cross instance call to all the other instances which will update a variable in the SGA to indicate that writes have been stopped. Until SMON releases the IR instance lock, foregrounds will wait on this event.

Wait time

5 seconds in tight loop until the writes are allowed again.

Parameters

by thread#

The thread number that stopped the writes and is doing the recovery.

our thread#

The thread number of the current instance.

Advise

Make sure that the thread that is supposed to do instance recovery is really doing that.

3.0 Enqueues and Locks Names

3.1 BL, Buffer Cache Management

How Many Resources:

The total number of BL resources (or PCM resources) is determined by the following formula:

up to 7.3.1:

prime(gc_db_locks) + 3 + prime(gc_segments * 1.5) + prime(gc_segments * 1.5 * 5) + prime(gc_tablespaces) + prime(gc_save_rollback_locks) + (gc_rollback_segments * (gc_rollback_locks + 1))

after 7.3.1:

prime(gc_db_locks) + 3 + prime(gc_segments) + prime(gc_freelist_group) + prime(gc_tablespaces) + prime(gc_save_rollback_locks) + (gc_rollback_segments * (gc_rollback_locks + 1)) + gc_releasable_locks

How Many Locks:

The number of locks is determined by the number of instances running. For each instance running there will be one lock allocated for each BL resource.

How Many Users:

All processes, indirectly through LCK* processes

Who Uses:

LCK* processes

When and How Used:

When accessing database blocks. All acquired in NULL mode by each instance's LCK* processes when an instance is started. Converted up and down to different modes by LCK* processes in response to requests from user foregrounds and instance backgrounds. Each lock covers a portion of the physical database on disk, by file and block range, in accordance with the setting of 'gc_file_to_locks'.

Id1, Id2 Combination:

Lock Element Number, Block Class

Lock Value Block:

Thread number of last change and SCN.

Init.ora Parameters:

gc_db_locks, gc_segments, gc_tablespaces, gc_save_rollback_locks, gc_rollback_segments, gc_rollback_locks.

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Asynchronous.

3.2 CF, Controlfile Transaction

How Many Resources:

1 to serialize controlfile transactions, 1 to serialize reads and writes from shared information portion of controlfile, which is a special global notepad area of the controlfile.

How Many Locks:

1 Lock for each process that tries to perform a Control File operation.

How Many Users:

(depends)

Who Uses:

Any foreground or background doing a controlfile transaction

When Used:

- switching logfiles, held exclusive.
- updating checkpoint information for datafiles, held exclusive.
- opening a logfile for redo reading during recovery, held shared.
- getting information to perform archiving, held shared.
- performing crash recovery, held exclusive.
- performing instance recovery, held exclusive.
- performing media recovery, held exclusive.
- dumping logfile history, held shared.
- updating archiving information, then exclusive.
- creating a database, held exclusive.
- mounting a database, held shared.
- closing a database, held shared.
- adding a logfile or logfile member, held exclusive.

- dropping a logfile or logfile member, held exclusive.
- checking information about logfile group members, held shared.
- adding a new datafile, held exclusive.
- dropping a datafile, held exclusive.
- re-creating (as empty) an existing database file, held shared to find the file, held exclusive to perform the update to the file entry in the controlfile when the zeroing task is complete.
- identifying or re-identifying database files, held shared.
- setting a set of datafiles OFFLINE, held exclusive twice in a two step process.
- setting a set of datafiles ONLINE, held exclusive, and then again when the transaction subsystem completes applying save undo to the on-lined files, a two-step process.
- marking a set of datafiles READONLY, held exclusive twice in a two-step process.
- marking a set of READONLY datafiles READWRITE, held exclusive twice in a two-step process.
- formatting a new controlfile, i.e. CREATE CONTROLFILE, held exclusive.
- beginning a hot backup, held exclusive.
- ending a hot backup, held exclusive.
- checking to see, after a crash, whether datafiles are in hot backup mode, held shared, then exclusive.
- executing a ALTER DATABASE BACKUP CONTROLFILE TO TRACE, held shared.
- opening the database (controlfile), held exclusive by first instance to open, shared by subsequent instances.
- renaming datafiles or logfiles, held exclusive.
- marking a controlfile as valid and mountable, held exclusive.
- handling an error encountered in the controlfile, held exclusive.
- validating data dictionary entries against matching controlfile records for file entries, held exclusive.
- updating controlfile format after a software upgrade, held exclusive.
- scanning for log file info by log sequence #, held shared.
- finding the highest-in-use entry of a particular controlfile record type, held shared.
- getting information about the number of log entries and their lowest/highest sequence number and log file numbers, held shared.
- looking up a controlfile record matching a given filename, held shared.
- retrieving or updating a controlfile record, held shared for retrieval, exclusive for update.
- making a backup controlfile, held exclusive for the duration of the copy.
- dumping the contents of the controlfile during debugging, held shared.
- dumping contents of a current redo logfile during debugging, held shared.
- dumping contents of redo log headers, held shared.
- dumping contents of datafile headers, held shared.
- updating MAC security mode in Trusted Oracle, held exclusive.
- reading shared information from the controlfile, held exclusive until shared information is written back updated, or done looking at it. Shared information is read and updated infrequently and is used primarily for instances to agree on compatibility, modes of operation, and for system commit number housecleaning in instance recovery situations.

Id1, Id2 Combinations:

TABLE 19. Id1, Id2 Combinations for CF Resource.

Id1	Id2	Meaning
0	0	Serialize Control file actions
0	1	Shared Information Access.

Lock Value Block:

Init.ora parameters:

processes

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.3 CI, Cross-instance Call Invocation

How Many Resources:

44 Resources (4 per function, 11 Cross Instance Functions)

4/unique type of cross-instance call, acquired on the fly as a particular call is requested (call,parameter,queue,return). The four locks for each call are used as a mechanism to allow an RPC on a remote instance background process.

How Many Locks:

The number of locks depend on the number of concurrent processes that execute the Cross Instance Call.

How Many Users:

Potentially all foreground or backgrounds

Who Uses:

All foreground and backgrounds

When Used:

Used to invoke specific actions in background processes on a specific instance or all instances. Examples include checkpoint, log switch, shutting down, identifying or re-identifying datafiles, etc. All in all, there are a small number (10's) of predefined cross-instance call types.

Id1, Id2 Combination:

TABLE 20. Id1 for CI resource.

Id1	Meaning
0	Flush buffers for reuse as new class
1	LGWR checkpointing and Hot Backup
2	DBWR synchronization of SGA with control file
3	Log file add/drop/rename notification
4	Write buffer for CR read
5	Test Call
6	Invalidate KCK cache in all instances
7	Alter rollback segment optimal

8	Signal Query Servers/coordinator
9	Create Remote Parallel Query Server
10	Set Global Partitions
11	Stop Disk Writes
12	Drop Sort Segments
13	Release unused space from Sort Segments
14	Instance Recovery for Parallel operation Group
15	Validate parallel slave Lock Value
16	Check Transaction State Objects

TABLE 21. Id2 for CI resource.

Id2	Meaning
1	Pass in Parameters
2	Invoke the call in background process
3	Foreground has not returned yet
4	Used to allocate the CI call
5	Used to queue up interested clients

Lock Value Block:

Init.ora Parameters:

processes

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.4 CU, Bind Enqueue

How Many Resources:

1/child cursor

How Many Locks:

1/child cursor/process.

How Many Users:

processes * open_cursors.

Who Uses:

All processes.

When Used:

This keeps us from hanging on the exclusive pin if somebody else sets the bind types and gets the shared-pin for the execute before we get the exclusive pin for bind (which would be unnecessary at that point). Once we get the enqueue, we'll check whether the cursor is still

unbound, in which case we'll verify that it was bound the way that we wanted it, or we'll go ahead and bind it ourselves.

Id1, Id2 Combination:

Child Object Handle, 0.

Lock Value Block:

Not Used.

Init.ora Parameters:

max_open_cursors

Scope:

Local Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.5 DF, Datafile

How Many Resources:

1/ on-line file

How Many Locks:

1 for each instance.

How Many Users:

1/ instance.

Who Uses:

DBWR's

When Used:

Obtained shared by DBWR in every instance as the instance comes up or upon request of a re-identify request. It is held shared for the life of the instance, and is used to notice changes like a file being off-lined on another instance. It is also held exclusive when bringing a file on-line.

Id1, Id2 Combination:

Always 0, File Number.

Lock Value Block:

Not Used.

Init.ora Parameters:

db_files

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.6 DL, Direct Loader Index Creation

How Many Resources:

1/ table being direct loaded

How Many Locks:

1 for each sqlldr process loading the table.

How Many Users:

1/ direct loader / object

Who Uses:

SQL*Loader in Direct-path mode

When Used:

When initializing to do a direct load of a table, held shared by every direct loader session loading that object for the duration of the load. Acquired and held shared exclusive when creating an index on the table.

Id1, Id2 Combination:

Object Number, Always 0.

Lock Value Block:

Not Used.

Init.ora Parameters:

processes.

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.7 DM, Database Mount

How Many Resources:

1 if mounted EXCLUSIVE, 5 if mounted SHARED

How Many Locks:

1 Lock per instance.

How Many Users:

Any processes trying to mount/open a database

Who Uses:

Any processes trying to mount/open a database

When Used:

When mounting or opening a database, if the database is being mounted exclusive, then one lock is acquired to ensure no other instances can successfully mount it; if the database is being mounted shared, all five locks are used to ensure that all processes attempting to mount or open on all instances have a consistent view of the mount or open state of the database.

Id1, Id2 Combination:

TABLE 22. Id1, Id2 Combination for DM Lock.

Id1	Id2	Meaning
0	0	Mount type. This is used to determine if the DB may be mounted shared or exclusive. It is always nowait. Only acquired by DBWR.
0	1	Mount gate. This is used to serialize checking for the first instance to mount. It is only acquired by foregrounds attempting to mount the database.
0	2	Hold mount. This is held to indicate that the db is mounted by some instance some where. Its lockvalue contains enough information to validate that the correct controlfiles are opened. The first foreground gets it shared. After the database is successfully mounted, the DBWR holds this lock shared until the DB is dismounted.
1	0	Open flag. This is used to verify that no other instance has the database open or is in process of opening it. It is similar to the hold open lock and is acquired and released at the same time. This is a local enqueue rather then an instance lock. This is separate of the hold open lock so that testing does not interfere with deciding if this is the first instance to open. If one instance was getting the hold lock just to see if the db was open any where, another instance might think the database was already opened.
1	1	Open gate. This functions identically to the mount gate except that it is used for database open.
1	2	Hold open. This functions identically to hold mount except that it is used for database open, and its lock value is not used.

Lock Value Block:

No.

Init.ora Parameters:

_db_no_mount_lock.

Scope:

Local and Global Enqueue (but also Global Lock).

Deadlock Sensitive:

Yes for the Local and Global Enqueue, no for the Global Lock.

Operation:

Synchronous.

3.8 DR, Distributed Recovery

How Many Resources:

1

How Many Locks:

1 per RECO process that tries to do Distributed Recovery.

How Many Users:

1/ instance

Who Uses:

RECO's

When Used:

Used to ensure that only one RECO is doing distributed transaction recovery actions at a time.

Id1, Id2 Combination:

Always 0, Always 0

Lock Value Block:

Not Used.

Init.ora Parameters:

distributed_transactions. (Distributed Database Option)

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.9 DX, Distributed TX

How Many Resources:

1

How Many Locks:

1 per process that tries to do a distributed transaction.

How Many Users:

1/ instance

Who Uses:

RECO's

When Used:

Used to ensure that only one process is using a distributed transaction slot at a time.

Id1, Id2 Combination:

Slot Number (of distributed transaction table), Always 0

Lock Value Block:

Not Used.

Init.ora Parameters:

distributed_transactions. (Distributed Database Option)

Scope:

Local Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.10 FS, File Set

How Many Resources:

2

How Many Locks:

Depends of the number of concurrent processes that tries to do these operations.

How Many Users:

Used by processes performing actions that require that the files that make up the database do not change the set of files it may be performing the action on.

Who Uses:

Used by processes performing actions that require that the files that make up the database do not change the set of files it may be performing the action on.

When Used:

The first lock is used to hold the set consistent for actions like dropping or renaming a datafile. The second is used to when validating controlfile information regarding files with information stored in the data dictionary.

Id1, Id2 Combination:

TABLE 23. Id1, Id2 Combination for FS Lock.

Id1	Id2	Meaning
0	0	Take offline
0	1	Do dictionary check.

Lock Value Block:

Not Used.

Init.ora Parameters:

None.

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.11 IN, Instance Number

How Many Resources:

1/ active instance

How Many Locks:

1/active instance

How Many Users:

1/ instance

Who Uses:

DBWR's

When Used:

Used to either obtain or ensure that the instance_number parameter used for freelist management, is unique for every instance accessing a database.

Id1, Id2 Combination:

Always 0, Instance Number.

Lock Value Block:

Not Used.

Init.ora Parameters:

instance_number

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.12 IR, Instance Recovery

How Many Resources:

1/database

How Many Locks:

at most 1/instance

How Many Users:

at most 1/ instance

Who Uses:

SMON's and processes attempting to open the database

When Used:

Used to ensure that only one SMON process or ALTER DATABASE OPEN is performing instance recovery at any given time.

Id1,Id2 Combination:

TABLE 24.

Id1	Id2	Meaning	Scope
0	0	To serialize instance recoverers.	Local and Global Enqueue
1	0	Used to verify if writes are allowed in the database.	Global Lock

Lock Value Block:

Not Used.

Init.ora Parameters:

None.

Scope:

Local and Global Enqueue and Global Lock.

Deadlock Sensitive:

Yes for the Local and Global Enqueue only.

Operation:

Synchronous.

3.13 IS, Instance State

How Many Resources:

3

How Many Locks:

Could range from 0 to the number of processes in an instance.

How Many Users:

All processes that want to mount/dismount or open/close a database. Also held by users who want to keep the instance in its current state.

Who Uses:

All processes that want to mount/dismount or open/close a database. Also held by users who want to keep the instance in its current state.

When Used:

Processes that wish to modify the instance state acquire the locks in exclusive mode. Processes that want to hold the instance in a particular mode acquire them shared.

Id1, Id2 Combination:

TABLE 25. Id1, Id2 Combination for IS Lock.

Id1	Id2	Meaning
0	0	Background Instance State Lock
0	1	Changing mount or open state of the database state on this instance.
0	3	Modify Cache State.

Lock Value Block:

Not Used.

Init.ora Parameters:

None.

Scope:

Local Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.14 IV, Library Cache Invalidation

How Many Resources:

1 / valid and existent database object that is currently cached in the library cache (valid and existent table, view, procedure, function, package, package body, trigger, index, cluster, synonym); cursors (SQL and PL/SQL), pipes, invalid or non-existent objects, and any other transient objects do not use this lock

How Many Locks:

1/per instance/object.

How Many Users:

All processes

Who Uses:

LCK? and the process which wants to invalidate objects cached across all instances

When Used:

When a valid and existent database object is brought into the library cache, LCK? is asked to acquire the lock in the correct mode (always S). It is held until the object becomes invalid or non-existent or is aged out of the library cache. This lock is used to invalidate objects cached across all instances. The process that wants to invalidate an object simply acquires the lock in X mode on the object causing all instances to invalidate their cached objects responding to BAST's and release their IV locks on their objects; then, the invalidating process immediately releases the X lock.

Id1, Id2 Combination:

Object Number, Timestamp.

Lock Value Block:

Not Used.

Init.ora Parameters:

None.

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.15 JQ, Job Queue

How Many Resources:

1 / job

How Many Locks:

1/job.

How Many Users:

All SNP process

Who Uses:

SNP? processes.

When Used:

To synchronize access to the job.

Id1, Id2 Combination:

0, job number.

Lock Value Block:

Not Used.

Init.ora Parameters:

job_queue_processes, snapshot_refresh_processes.

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.16 KK, Redo Log "Kick"

How Many Resources:

1/ redo thread, i.e. 1/instance in 7.x.

How Many Locks:

1/instance.

How Many Users:

1/ instance

Who Uses:

LGWR's

When Used:

Used to keep on-line redo logs archiving off of idle instances, while other active instances generate redo and archive on-line logs. The intent is to keeps archive streams from all participating instances close to each other in time, so that a set of archived logs for all redo streams/instances can easily be identified and managed for backup or recovery. Also used in executing the ALTER SYSTEM ARCHIVE LOG CURRENT command, which is used to cause all instances to archive their current logs.

Id1, Id2 Combination:

Always 0, Thread Number.

Lock Value Block:

Not Used.

Init.ora Parameters:

thread

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.17 L[A-P], Library Cache Lock

How Many Resources:

1 / database object referenced during parsing or compiling of SQL or PL/SQL statements (table, view, procedure, function, package, package body, trigger, index, cluster, synonym); released at the end of parse or compilation; cursors (SQL and PL/SQL areas), pipes and any other transient objects do not use this lock.

How Many Locks:

How Many Users:

All processes

Who Uses:

All processes

When Used:

When a database object (table, view, procedure, function, package, package body, trigger, index, cluster, synonym) is referenced during parsing or compiling of a SQL (DML/DDL) or PL/SQL statement, the process parsing or compiling the statement acquires the lock in the correct mode. It is held until the parse or compilation completes (for the duration of the parse call). In V6, these locks are known as parse or DDL locks except that in V6 these locks are held for the duration of the cursor (i.e., until the cursor is closed); in Oracle7, these locks are held only during the parse call.

Id1, Id2 Combination:

Lock Value Block:

Not Used.

Init.ora Parameters:

shared_pool

Scope:

Global Lock.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.18 MM, Mount Definition

How Many Resources:

1/ read-only database being mounted to the read-write database

How Many Locks:

How Many Users:

Any process trying to mount a read-only database under Trusted or any process asking for library cache cleanout of remote objects.

Who Uses:

Any process trying to mount a read-only database under Trusted or any process asking for library cache cleanout of remote objects.

When Used:

Only used by Trusted Oracle7 currently.

Id1, Id2 Combination:

Lock Value Block:

Init.ora Parameters:

Scope:

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.19 MR, Media Recovery

How Many Resources:

1+1/ file.

How Many Locks:

1 for each DBWR in each instance.

How Many Users:

Any proc. doing media recovery or ALTER DATABASE OPEN RESETLOGS

Who Uses:

Any proc. doing media recovery or ALTER DATABASE OPEN RESETLOGS

When Used:

Held shared by any process performing media recovery to prevent any other process from performing an ALTER DATABASE OPEN RESETLOGS. The file access locks are held shared by on-line users of a particular datafile to prevent recovery actions from taking place. They are held exclusive by processes performing recovery actions on that file.

Id1, Id2 Combination:

TABLE 26. Id1, Id2 Combinations for MR Lock

Id1	Id2	Meaning
0	0	Media Recovery Enqueue **X** - doing Reset Logs **S** - doing Recovery
file#	0	File Access Enqueue, used to coordinate recovery.

Lock Value Block:

Not Used.

Init.ora Parameters:

db_files

Scope:

Local and Global Enqueue

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.20 Q[A-Z], Row Cache

How Many Resources:

1/ row cache entry of any type (corresponds to dc_* params) (ts$, fet$, seg$, undo$, uet$, tsq$, file$, user$, obj$, tables, views, clusters, synonyms, user names, object ids, constraints, sequence cache entries, profiles, database links, histograms).

How Many Locks:

1 per instance.

How Many Users:

All processes

Who Uses:

All processes

When Used:

Used to keep row cache entries coherent across instances. Row cache entries are caches of recently used data dictionary information. The second part of the type [A-Z] is determined by the type of the parent cache (0=A, 1=B ... 25=Z)

Id1, Id2 Combinations:

Both values are being hashed from ??.

Lock Value Block:

Init.ora Parameters:

shared_pool

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.21 P[A-Z], Library Cache Pin (will be N[A-Z] in 7.1.3)

How Many Resources:

1 / database object that is currently cached in the library cache (table, view, procedure, function, package, package body, trigger, index, cluster, synonym); in the library cache, a database object is cached in 2 parts: “handle” and “object”; only when the “object” part is cached, this lock is held; cursors (SQL and PL/SQL areas), pipes and any other transient objects do not use this lock.

How Many Locks:

1 per process pinning the cursor.

How Many Users:

All processes

Who Uses:

Foregrounds.

When Used:

Used when pinning “object” parts of database objects (see above) in an instance’s library cache. Held until the “object” part is flushed or aged out (e.g., as the result of another object needing cache space, and the object to be flushed is not pinned or in use).

Id1, Id2 Combinations:

Lock Value Block:

Init.ora Parameters:
cursor_space_for_time.
Scope:
Global Lock.
Deadlock Sensitive:
No.
Operation:
Synchronous.

3.22 PF, Password File

How Many Resources:
1
How Many Locks:

How Many Users:
All processes either accessing or modifying the password file.
Who Uses:
All processes either accessing or modifying the password file.
When Used:
Acquired exclusive to read or update the password file.
Id1, Id2 Combination:

Lock Value Block:

Init.ora Parameters:

Scope:
a
Deadlock Sensitive:
No.
Operation:
Synchronous.

3.23 PI, Parallel Slaves

How Many Resources:
1/ parallel query, index, recovery process group
How Many Locks:

How Many Users:

All processes trying to add parallel slaves to a slave server group

Who Uses:

All processes trying to add parallel slaves to a slave server group

When Used:

To synchronize the creation of new parallel slave servers and their addition to a parallel slave group on behalf of a parallel query, a parallel index creation, or a parallel recovery session.

Id1, Id2 Combinations:

Lock Value Block:

Init.ora Parameters:

Scope:

Deadlock Sensitive:

No.

3.24 PR, Process Startup

How Many Resources:

1

How Many Locks:

1/process

How Many Users:

Any process starting up a background process or an MTS server.

Who Uses:

Any process starting up a background process or an MTS server.

When Used:

Acquired exclusive to serialize creation of background or MTS server processes.

Id1, Id2 Combinations:

0, 0.

Lock Value Block:

Not used.

Init.ora Parameters:

processes

Scope:

Local.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.25 PS, Parallel Slave Synchronization

How Many Resources:

1/ parallel slave server process

How Many Locks:

1/process

How Many Users:

All processes working on behalf of a parallel query, index creation, or recovery action.

Who Uses:

All processes working on behalf of a parallel query, index creation, or recovery action.

When Used:

Used to control execution flow of parallel slaves for parallel query, parallel index creation, and parallel recovery.

Id1, Id2 Combination:

Instance Id, Server Id

Instance Id << 16, Server Id

Lock Value Block:

process queue at remote site (pointer), flag (1 byte)(0x01=descriptor initialized, 0x02=Query Coordinator Queue, 0x80=Enqueue test bit), server id (1 byte), instance id (2 bytes), OS specific descriptor (8 bytes or less).

Init.ora Parameters:

min_parallel_servers, max_parallel_servers.

Scope:

Local, Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.26 RT, Redo Thread

How Many Resources:

Up to 12, one for LGWR, one for DBWR, and one for each configured LCK process [LCK0-LCK9]

How Many Locks:

How Many Users:

Up to 3-12/instance

Who Uses:

Each of the set above, and by processes testing to see if instance has failed.

When Used:

Used to detect the death of an instance after a BL lock is found to be invalid. Each is acquired exclusive by the associated process of each instance. If they can ever all be granted for a given set of processes associated with a particular instance, then instance failure can be inferred.

Id1, Id2 Combination:

TABLE 27.

Id1	Id2	Meaning
Thread	0	Mount Enqueue for that Thread
Thread	1	LGWR
Thread	2	DBWR
Thread	3	LCK0
Thread	4	LCK1
Thread	5	LCK2
Thread	6	LCK3
Thread	7	LCK4
Thread	8	LCK5
Thread	9	LCK6
Thread	10	LCK7
Thread	11	LCK8
Thread	12	LCK9

Lock Value Block:

Not Used.

Init.ora Parameters:

gc_lck_procs

Scope:

Local (Mount Queue only), Global

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.27 SC, System Commit Number

How Many Resources:

1

How Many Locks:

1/process

How Many Users:

All processes

Who Uses:

All processes

When Used:

The System Commit Number (SCN) is used to serialize time within a single instance, and across all instances. This lock resource caches the current value of the SCN, the value is incremented in response to many database event, but most notably COMMIT WORK. Access to the SCN lock value to get and store the SCN is batched on most cluster implementations, so that every process that needs a new SCN gets one and stores a new value back on one instance, before the SCN

lock is released so that it may be granted to another instance. Processes get the SC lock once and then use conversion operations to manipulate the lock value.

Id1, Id2 Combination:

TABLE 28.

Id1	Id2	Meaning
0	0	SCN Global lock
0	2	Local SCN batching enqueue

Lock Value Block:

System Commit Number, System Commit Number.

Init.ora Parameters:

processes

Scope:

Local (batching enqueue only), Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.28 SM, SMON

How Many Resources:

1

How Many Locks:

1 for each instance.

How Many Users:

1/ instance

Who Uses:

SMON's

When Used:

Used whenever an SMON wakes up to see if there is work to do. Acquired when SMON's are started in NULL mode, and converted to exclusive and back when checking for work to do, currently every five minutes.

Id1, Id2 Combinations:

Always 0, Always 0.

Lock Value Block:

Timestamp (4 bytes) in seconds of when SMON was run the last time.

Init.ora Parameters:

No.

Scope:

Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.29 SN, Sequence Number

How Many Resources:
1/ cached sequence
How Many Locks:

How Many Users:
All processes using sequences.
Who Uses:
All processes using sequences.
When Used:
When using sequence number generators.
Id1, Id2 Combinations:
Always 0, Object Number.
Lock Value Block:

Init.ora Parameters:
sequence_cached_entries
Scope:

Deadlock Sensitive:
No.
Operation:
Synchronous.

3.30 SQ, Sequence Number Enqueue

How Many Resources:
1/ cached sequence
How Many Locks:

How Many Users:
All processes using sequences.
Who Uses:
All processes using sequences.
When Used:
When using sequence number generators.
Id1, Id2 Combinations:
Always 0, Object Number.
Lock Value Block:

Init.ora Parameters:

Scope:

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.31 SR, Synchronized Replication

How Many Resources:

1/ cached sequence

How Many Locks:

How Many Users:

All processes using sequences.

Who Uses:

All processes using sequences.

When Used:

When using sequence number generators.

Id1, Id2 Combinations:

Always 0, Always 0.

Lock Value Block:

Init.ora Parameters:

Scope:

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.32 SS, Sort Segment

How Many Resources:

2/ tablespace

How Many Locks:

How Many Users:

All processes using sort segments.

Who Uses:

All processes using sert segments.

When Used:

When sorting.

Id1, Id2 Combinations:.

TABLE 29.

Id1	Id2	Meaning
tablespace id	1	Local enqueue, to notify SMON to request some action for tablespace identified by tablespace id.
tablespace id	2	Global enqueue, to release space for releasing space in tablespace identified by tablespace id.

Lock Value Block:

Init.ora Parameters:

Scope:

Enqueue (Local, Global)

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.33 ST, Space Management Transaction

How Many Resources:

1

How Many Locks:

How Many Users:

Any process allocating, freeing, or coalescing free space.

Who Uses:

Any process allocating, freeing, or coalescing free space.

When Used:

Used to serialize space transactions across all instances, which are executed within a transaction when space requires allocation or de-allocation. Also acquired by SMON or foreground processes when two or more physically adjacent free extents are coalesced back into one extent.

Id1, Id2 Combinations:

Always 0, Always 0.

Lock Value Block:

Init.ora Parameters:

Scope:

Local, Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.34 SV, Sequence Number Value

How Many Resources:

1/ cached sequence

How Many Locks:

How Many Users:

All processes using sequences.

Who Uses:

All processes using sequences.

When Used:

When using sequence number generators.

Id1, Id2 Combinations:

Always 0, Object Number.

Lock Value Block:

Init.ora Parameters:

Scope:

Local, Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.35 TA, Transaction Recovery

How Many Resources:

2 + 2/ in-use, non-SYSTEM rollback segment

How Many Users:

Any process attempting to do transaction recovery.

Who Uses:

LGWR's, SMON's and any process doing transaction recovery.

When Used:

An SMON acquires locks exclusive to do instance recovery. LGWR's hold the locks shared and/or exclusive for in-use/on-line rollbacks segments.Held exclusive by foreground processes when brining rollback segments on-line.

Id1, Id2 Combination:

TABLE 30. Id1, Id2 Combinations for TA lock.

Id1	Id2	Meaning

1	undo seg#	Instance lock on undo segment
2	0	Instance lock on instance existence
3	undo seg#	instance lock on undo segment
4	0	mutual-exclusive lock starting up
5	0	transaction enqueue

Lock Value Block:

Not Used.

Init.ora Parameters:

gc_rollback_segments, max_rollback_segments.

Scope:

Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.36 TM, DML Enqueue

How Many Resources:

1/database + 1/ table referenced.

How many Locks:

1/process accessing a table in (update, insert,delete).

How Many Users:

1/ instance + n/ table for n concurrent users of a table

Who Uses:

All processes when referencing tables *and* dml_locks > 0

When Used:

The instance lock is used when an instance attempts to mount the database to ensure that all participating instances either have dml_locks = 0 or dml_locks != 0, if not ORA-61 is returned and the mount attempt fails. The per table locks are acquired during the execution of a transaction when referencing a table with a DML statement so that the object is not dropped or altered during the execution of the transaction, if and only if the dml_locks parameter is non-zero.

Id1, Id2 Combinations:

TABLE 31.

Id1	Id2	Meaning
0	0	Check if the instances have dml_locks = 0
0	object#	Global lock on table identified by object#

Lock Value Block:

Not Used.

Init.ora Parameters:

dml_locks

Scope:

Local, Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.37 TS, Temporary Segment (also TableSpace)

How Many Resources:

2/ temporary segment (e.g. for sorts)

1/ tablespace being dropped or having a rollback segment created in it.

How Many Locks:

1 for each process doing one of operations below.

How Many Users:

All processes using temp space, DROP TABLESPACE, or CREATE ROLLBACK SEGMENT

Who Uses:

All processes using temp space, DROP TABLESPACE, or CREATE ROLLBACK SEGMENT

When Used:

The two temporary segment locks are used for two different purposes on each temp segment. The first use is to serialize the “high water mark” or where the highest allocated point of the segment is. The second is used to serialize creation, use, and deletion of a temp segment. It is acquired by LCK?. The tablespace enqueue is acquired either when dropping that tablespace or when creating a rollback segment in it. The purpose is to avoid deadlocks that can occur on resource in the row cache (dictionary cache).

Id1, Id2 Combinations:

TABLE 32.

Id1	Id2	Meaning
segment dba	0	Serialize access to the “high water mark”
segment dba	1	Create, use, delete temp segment
tablespace number	2	prevent deadlock during create rollback segment and create tablespace.

Lock Value Block:

Not Used.

Init.ora Parameters:

_bump_highwater_mark_count (7.3).

Scope:

Local, Global

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.38 TT, Temporary Table

How Many Resources:

1

How Many Locks:

1 per instance.

How Many Users:

Any process starting an instance.

Who Uses:

Any process starting an instance.

When Used:

Used after recovery of an instance at instance startup to allocate and initialize private and public rollback segments to the instance being started. Acquired exclusive for the duration of the operation.

Id1, Id2 Combinations:

Always 0, Always 0.

Lock Value Block:

Not Used.

Init.ora Parameters:

rollback_segments, gc_rollback_segments (if running in Parallel mode).

Scope:

Global Lock.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.39 TX, Transaction

How Many Resources:

1/ active transaction

How Many Locks:

1/transaction + 1/process waiting for a locked row by that transaction.

How Many Users:

1 + 1/ process waiting for something locked by this transaction

Who Uses:

All processes

When Used:

Acquired exclusive when a transaction initiates its first change and held until the transaction does a COMMIT or ROLLBACK. Also acquired exclusive by SMON when doing recovery (undo) of a transaction. Used as a queuing mechanism for processes waiting for something locks by a transaction's change to become available again.

Id1, Id2 Combinations:

undo segment number << 16 | slot, sequence

Lock Value Block:

Not Used.

Init.ora Parameters:

transactions

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.40 UL, User-defined Locks

How Many Resources:

As many as put in use by the DBMS_LOCK package.

How Many Locks:

1 per process acquiring a lock on a resource.

How Many Users:

As many as are using the DBMS_LOCK package to synchronize user applications between processes on multiple instances.

Who Uses:

Any application process using the DBMS_LOCK package.

When Used:

In accordance with the client applications use of the DBMS_LOCK package in PL/SQL.

Id1, Id2 Combinations:

Application Dependent, Application Dependent.

Lock Value Block:

Not Used.

Init.ora Parameters:

No.

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.41 UN, User Name

How Many Resources:

1/ active userid

How Many Locks:

How Many Users:

1/ session using a particular userid

Who Uses:

Processes logging in to the database or doing DROP USER.

When Used:

During login, locks are acquired shared to ensure that no other process does a DROP USER of that particular user's information in the dictionary. Acquired exclusive during DROP USER.

Id1, Id2 Combinations:

User Number, Always 0.

Lock Value Block:

Not Used.

Init.ora Parameters:

No.

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.42 US, Undo segment, Serialization

How Many Resources;

1 per rollback segment.

How Many Locks:

1 per user requesting access to a RBS.

How Many Users:

1 .. processes

Who Uses:

All foregrounds, SMON and PMON.

When Used:

To serialize DDL on a given undo segment (rollback segment). It serializes the following operations:

- CREATE ROLLBACK SEGMENT
- DROP ROLLBACK SEGMENT
- ALTER ROLLBACK SEGMENT ONLINE
- ALTER ROLLBACK OFFLINE
- ALTER ROLLBACK SEGMENT SHRINK
- ALTER ROLLBACK SEGMENT STORAGE
- Offlining PENDING OFFLINE RBS by SMON
- SMON - abortive offline cleanup.
- STARTUP.

Id1, Id2 Combination:

Undo Segment#, Always 0.

Lock Value Block:

No.

Init.ora Parameters:

gc_rollback_locks, rollback_segments.

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operations:

Synchronous.

3.43 WL, Being written Redo Log

How Many Resources:

1/ redo log being archived

How many Locks:

How Many Users:

1/ instance + any foreground sessions which are archiving

Who Uses:

ARCH's + any foreground sessions which are archiving

When Used:

Used to determine whether or not a logfile is available for archiving (it is unavailable if some other process is already archiving it). Acquired shared to do the test, and acquired exclusive by the process actually doing archiving.

Id1, Id2 Combinations:

Log Number, Always 0.

Lock Value Block:

Not Used.

Init.ora Parameters:

No.

Scope:

Local and Global Enqueue.

Deadlock Sensitive:

Yes.

Operation:

Synchronous.

3.44 XA, Instance Attribute Lock

How Many Resources:

1/ registered attribute

How many Locks:

1/registered attribute

How Many Users:

1/ instance + any foreground sessions which are archiving

Who Uses:

DBWR's hold the lock. Any processes interested in the atribute reads the lock.

When Used:

Used to store attribute values. Read when a process wants to know a certain atribute of another process.

Id1, Id2 Combinations:

.

Lock Value Block:

Init.ora Parameters:

No.

Scope:

Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.

3.45 XI, Instance Registration Lock

How Many Resources:

1/ instance + (total lock, bitmap lock(s), join lock, recover lock)

How many Locks:

How Many Users:

1/ instance + any foreground sessions which are archiving

Who Uses:

DBWR's hold the locks. Foregrounds and backgrounds use the locks to register the instances.

When Used:

Used to assign unique instance numbers to instances. Also used to notify instances about system reconfigurations (instances coming up or down).

Id1, Id2 Combinations:

.

Lock Value Block:

Init.ora Parameters:

No.

Scope:

Global.

Deadlock Sensitive:

No.

Operation:

Synchronous.