Profiling the logwriter and database writer

Frits Hoogland

Collaborate 2014, Las Vegas

```
This is the font size used for showing screen output. Be sure this is readable for you.
```

This is the font used to accentuate text/console output. Make sure this is readable for you too!

`whoami`

- Frits Hoogland
- Working with Oracle products since 1996

- Blog: http://fritshoogland.wordpress.com
- Twitter: @fritshoogland
- Email: frits.hoogland@enkitec.com
- Oracle ACE Director
- OakTable Member

enkitec

- Early bird discount expires April 15, 2014.
- Quick links:
  - Home: www.enkitec.com/e4
  - Registration: http://www.enkitec.com/e4/register
  - Location: http://www.enkitec.com/e4/location
  - Training Days Following E4: http://www.enkitec.com/e4/training-days

Goals & prerequisites

- Goal: learn about typical behaviour of both lgwr and dbwr, both visible (wait events) and inner-working.

- Prerequisites:
  - Understanding of (internal) execution of C programs.
  - Understanding of Oracle tracing mechanisms.
  - Understanding of interaction between procs. with the Oracle database.

Test system

- The tests and investigation is done in a VM:
  - Host: Mac OSX 10.9 / VMWare Fusion 6.0.2.
  - VM: Oracle Linux x86_64 6u5 (UEK3 3.8.13).
  - Oracle Grid 11.2.0.4 with ASM/External redundancy.
  - Oracle database 11.2.0.4.
  - Unless specified otherwise.

Logwriter, concepts guide

- From the concepts guide:
  - The lgwr manages the redolog buffer
  - The lgwr writes all redo entries that have been copied in the buffer since the last time it wrote when:
    - User commits.
    - Logswitch.
    - Three seconds since last write.
    - Buffer 1/3th full or 1MB filled.
    - dbwr must write modified (‘dirty’) buffers.

Logwriter, idle

- The general behaviour of the log writer can easily be shown by putting a 10046/8 on lgwr:

```
SYS@v11204 AS SYSDBA> @who
…
130,1,@1  2243    oracle@ol65-ora11204-asm.local (LGWR)
…
SYS@v11204 AS SYSDBA> oradebug setospid 2243
Oracle pid: 11, Unix process pid: 2243, image: oracle@ol65-ora11204-asm.local (LGWR)
SYS@v11204 AS SYSDBA> oradebug unlimit
Statement processed.
SYS@v11204 AS SYSDBA> oradebug event 10046 trace name context forever, level 8;
Statement processed.
```

Logwriter, idle

- The 10046/8 trace shows:

```
*** 2013-12-18 14:12:32.479
WAIT #0: nam='rdbms ipc message' ela= 2999925 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372352479352

*** 2013-12-18 14:12:35.479
WAIT #0: nam='rdbms ipc message' ela= 3000075 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372355479531

*** 2013-12-18 14:12:38.479
WAIT #0: nam='rdbms ipc message' ela= 2999755 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372358479381

*** 2013-12-18 14:12:41.479
WAIT #0: nam='rdbms ipc message' ela= 3000021 timeout=300 p2=0 p3=0 obj#=-1
tim=1387372361479499
```

Logwriter, idle

- "rdbms ipc message" indicates a sleep/idle event
  - There isn't an indication lgwr writes something:

```
semtimedop(327683, {{15, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)
getrusage(RUSAGE_SELF, {ru_utime={0, 84000}, ru_stime={0, 700000}, ...}) = 0
times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
times({tms_utime=8, tms_stime=70, tms_cutime=0, tms_cstime=0}) = 431286151
semtimedop(327683, {{15, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)
…etc…
```

Logwriter, idle

- It does look in the /proc filesystem to the ‘stat’ file of a certain process:

```
open("/proc/2218/stat", O_RDONLY)       = 21
read(21, "2218 (oracle) S 1 2218 2218 0 -1"..., 999) = 209
close(21)
```

- It does so every 20th time (20*3)= 60 sec.
- The PID is PMON.

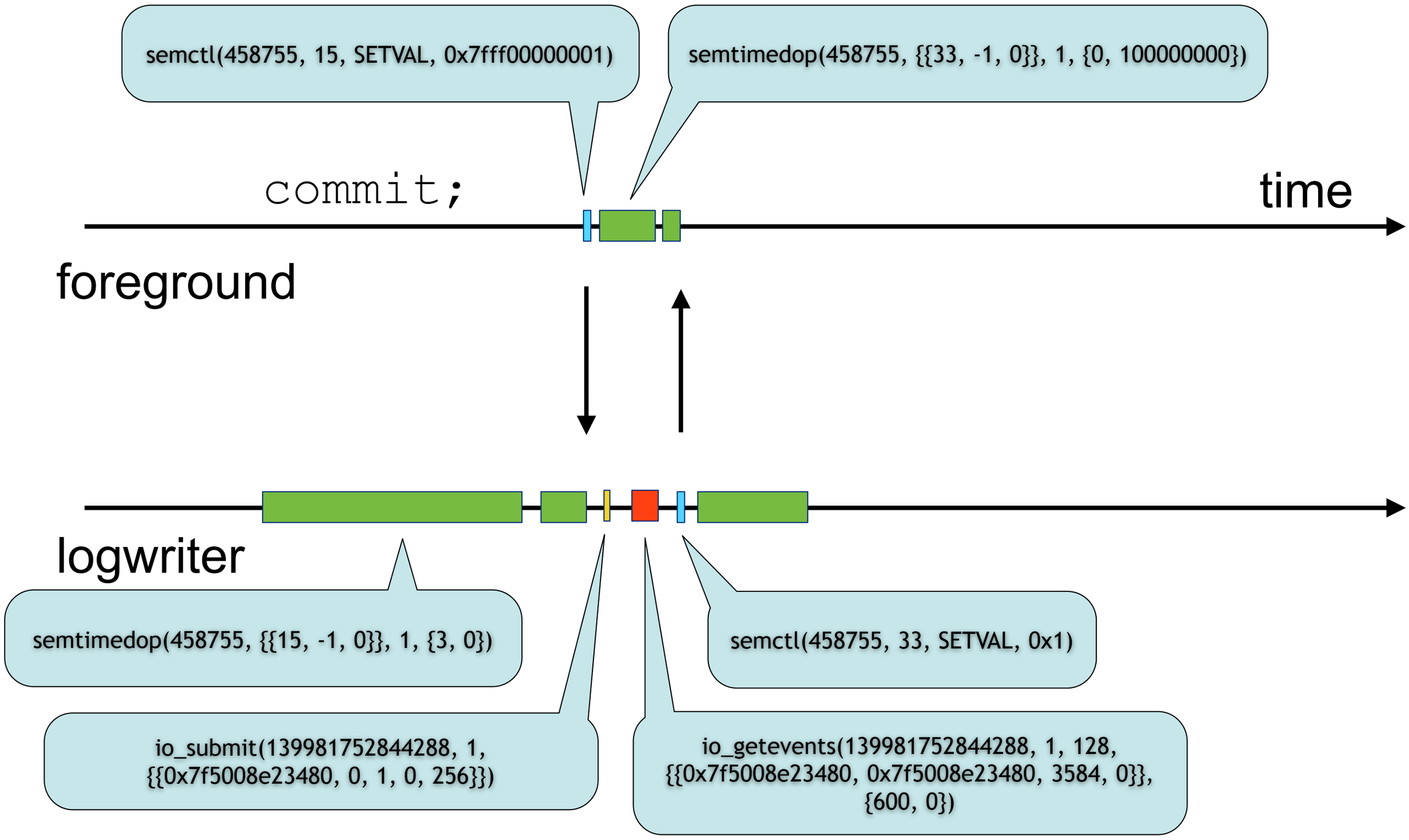
Logwriter, idle

- Recap:
  - In an idle database.
  - The lgwr sleeps on a semaphore for 3 seconds.
    - Then wakes up, and sets up the semaphore/sleep again.
    - Processes sleeping on a semaphore do not spend CPU
  - Every minute, lgwr reads pmon's process stats.
  - lgwr doesn’t write if there’s no need.
- But what happens when we insert a row of data, and commit that?

Logwriter, commit

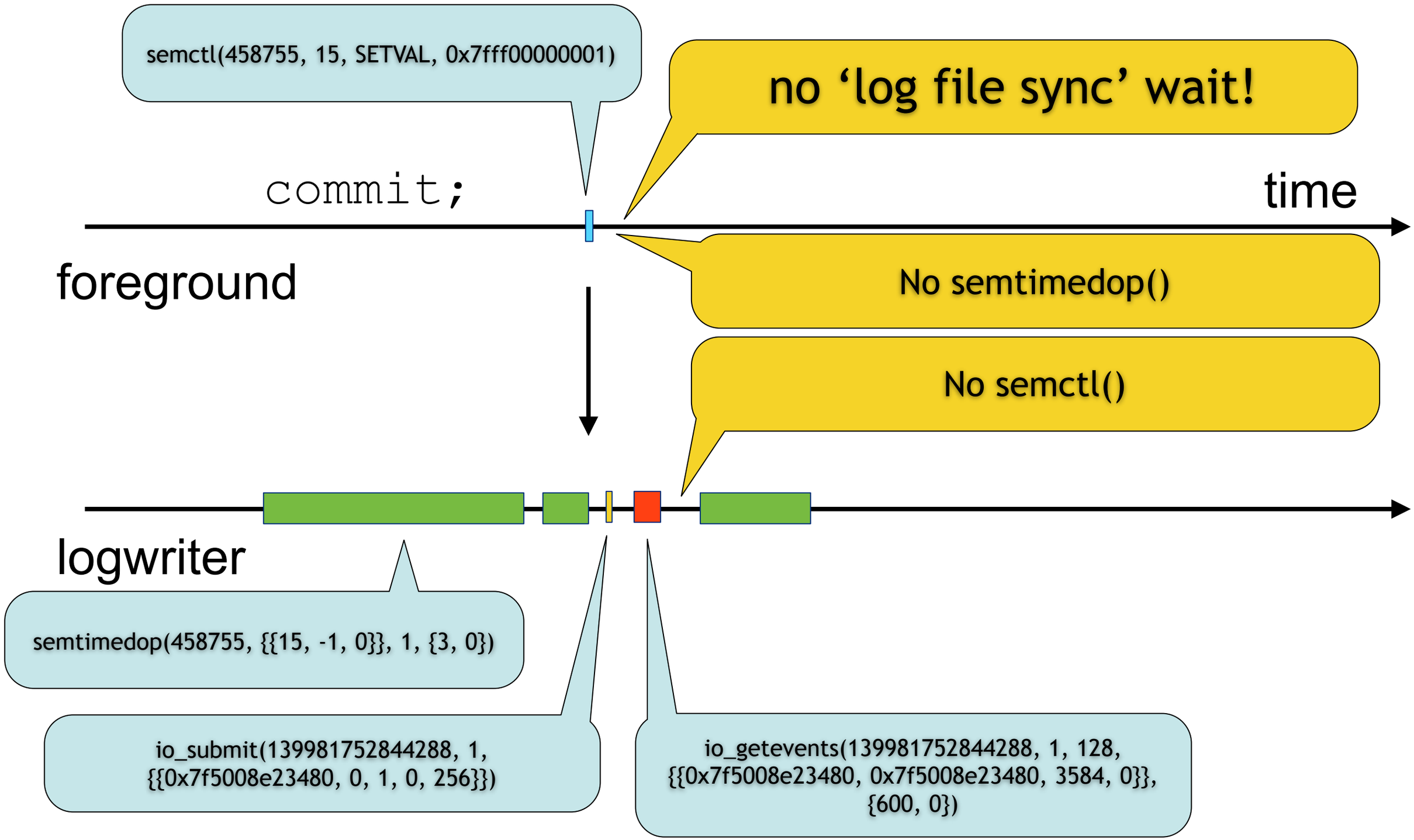
```
TS@//localhost/v11204 > insert into t values ( 1, 'aaaa', 'bbbb' );

1 row created.

TS@//localhost/v11204 > commit;

Commit complete.
```

Logwriter, commit - expected



enkitec

Logwriter, commit - actual



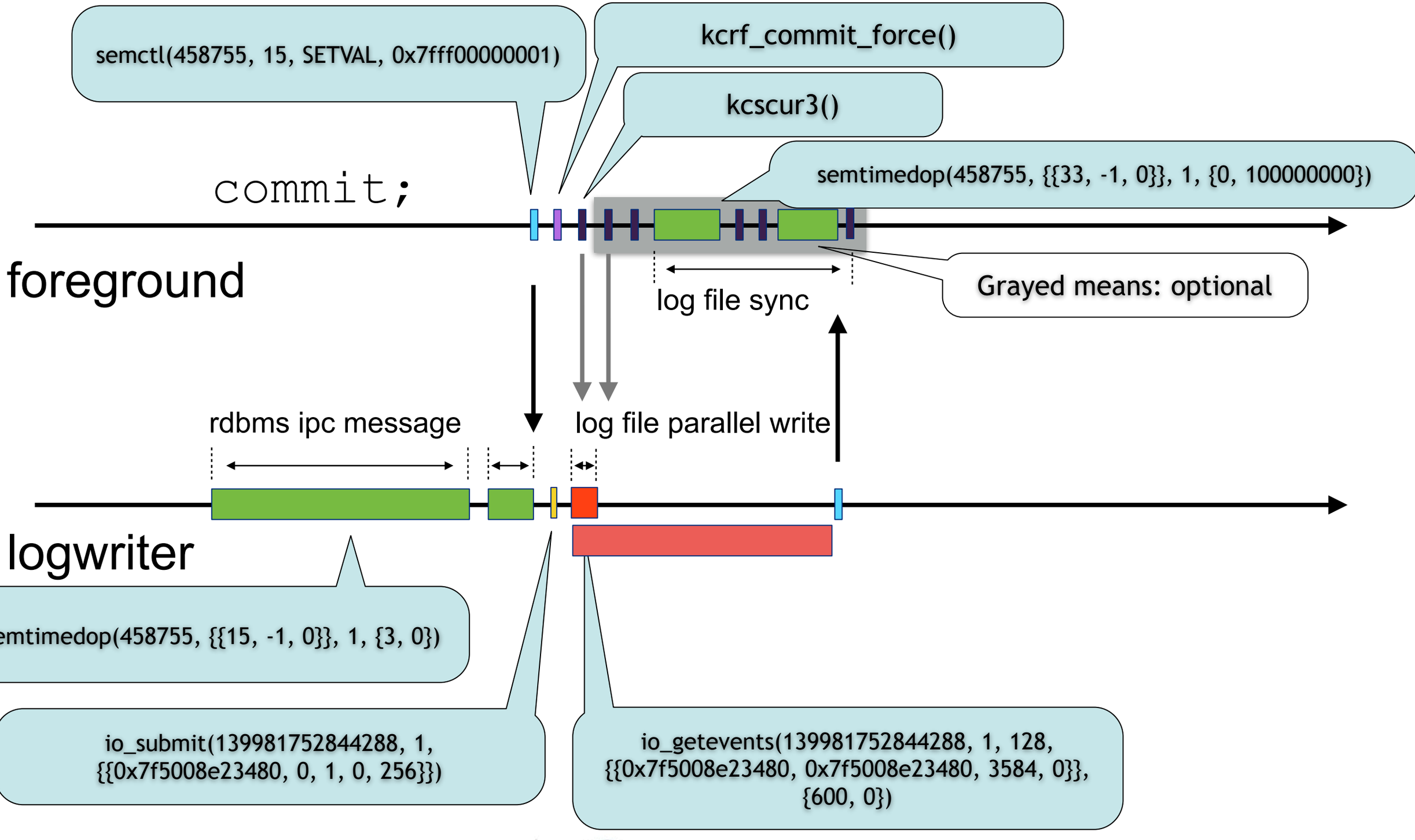
Logwriter, commit

- Investigation shows:
  - Foreground scans log writer progress up to 3 times.
    - kcrf_commit_force() > kcscur3()
  - If its data* in the redo log buffer is not written:
    - It notifies the lgwr that it is going to sleep on a semaphore.
    - semtimedop() for 100ms, until posted by lgwr.
  - If its data* has been written:
    - No need to wait on it.
    - No ‘log file sync’ wait.

Logwriter, commit

- Wait!!!
  - This (no log file sync) turned out to be an *edge case.*
    - I traced the kcrf_commit_force() and kcscur3() calls using breaks in gdb.
  - In normal situations, the wait will appear.
    - Depending on log writer and FG progress.
    - The semtimedop() call in the FG can be absent.
      - As a result, lgwr will not semctl()

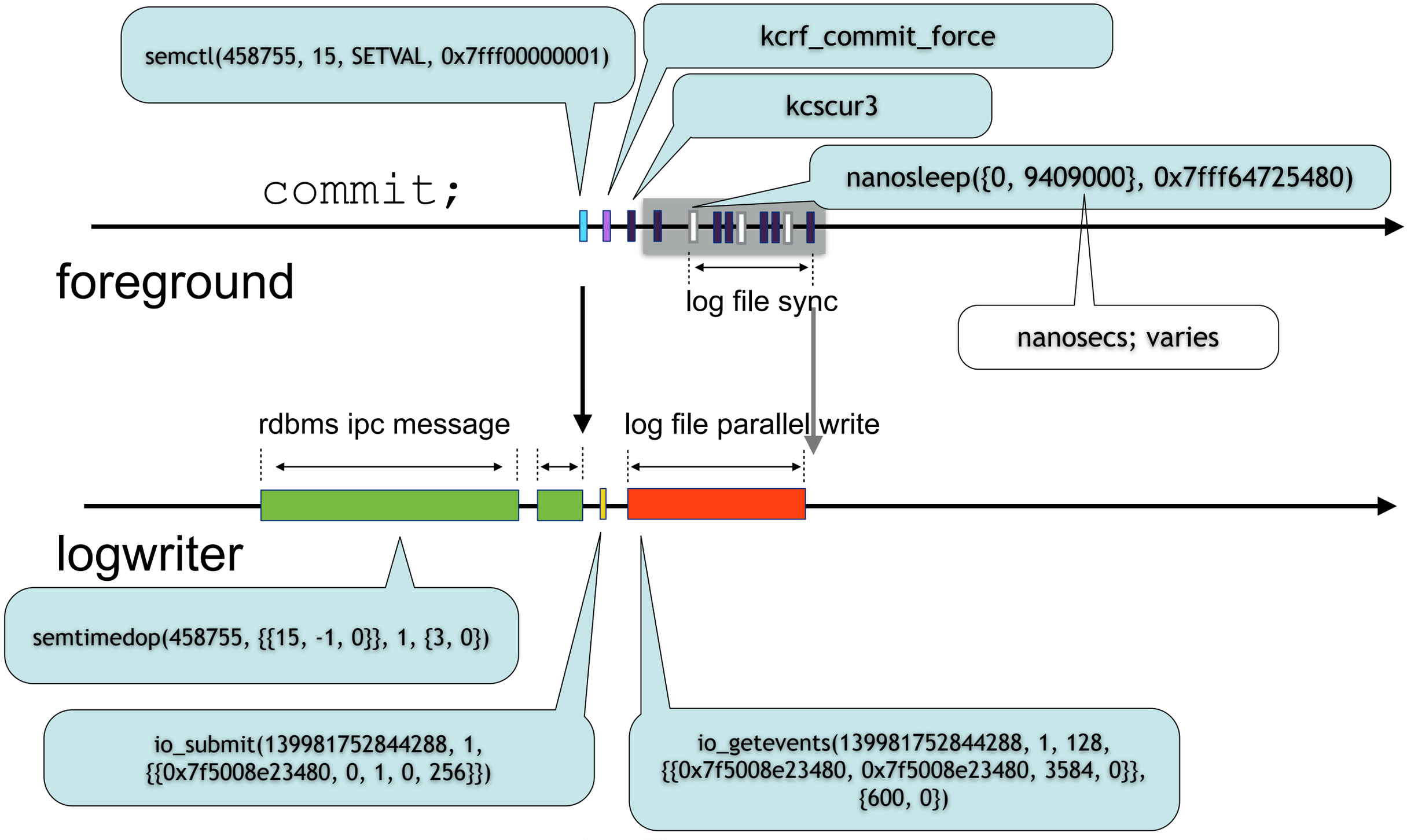
Logwriter, commit - post-wait



adaptive log file sync

- Feature of Oracle 11.2
  - Parameter '_use_adaptive_log_file_sync'
    - Set to FALSE up to 11.2.0.2
    - Set to TRUE starting from 11.2.0.3
    - Third value ‘POLLING_ONLY’
  - Makes Oracle adaptively switch between ‘post-wait’ and polling.
  - The log writer writes a notification in its logfile if it switches between modes (if param = ‘TRUE’)

Logwriter, commit - polling



Logwriter, post-wait vs. polling

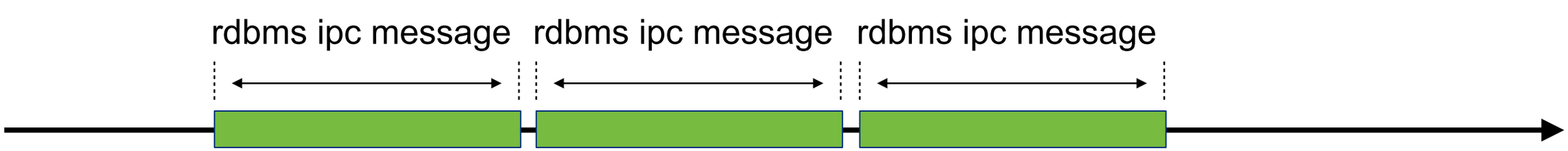
- No wait event ‘log file sync’ if:
  - Lgwr was able to flush the committed data before the foreground has issued kcscur3() 2/3 times in kcrf_commit_force().
- If not, the foreground starts a ‘log file sync’ wait.
  - If in “post-wait” mode (default), it will record it’s waiting state in the post-wait queue, sleep in semtimedop() for 100ms at a time, waiting to be posted by lgwr.
  - If in “polling” mode, it will sleep in nanosleep() for computed time*, then check lgwr progress, if lgwr write has progressed beyond its committed data SCN: end wait, else start sleeping in nanosleep() again.

Logwriter

- The main task of lgwr is to flush data in the logbuffer to disk.
  - The lgwr is idle when waiting on ‘rdbms ipc message’.
  - There are two main* indicators of lgwr business:
    - CPU time.
    - Wait event ‘log file parallel write’.

- The lgwr needs to be able to get onto the CPU in order to do process!

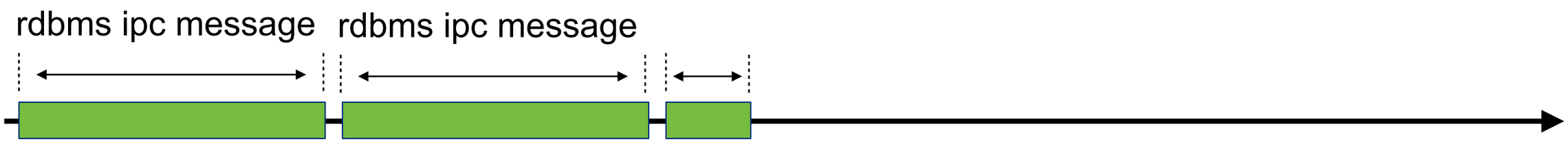
Logwriter - idle



logwriter

semtimedop(458755, {{15, -1, 0}}, 1, {3, 0})

Logwriter - idle



logwriter

Logwriter - idle

gwriter

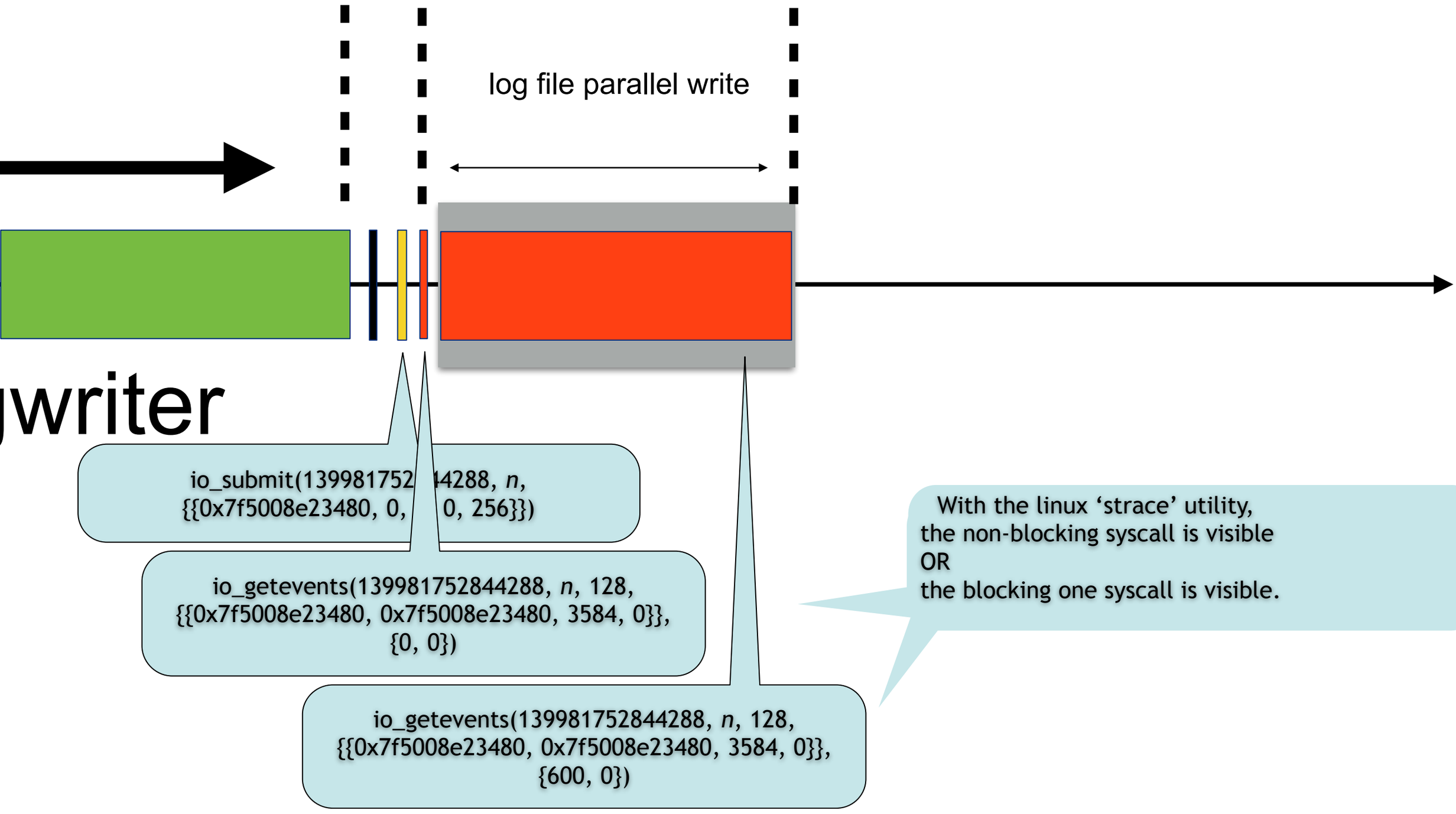
Idle mode latch gets:
‘messages’
‘mostly latch-free SCN’
‘lgwr LWN SCN’
‘KTF sga latch’
‘redo allocation’
‘messages’

Logwriter - writing

gwriter

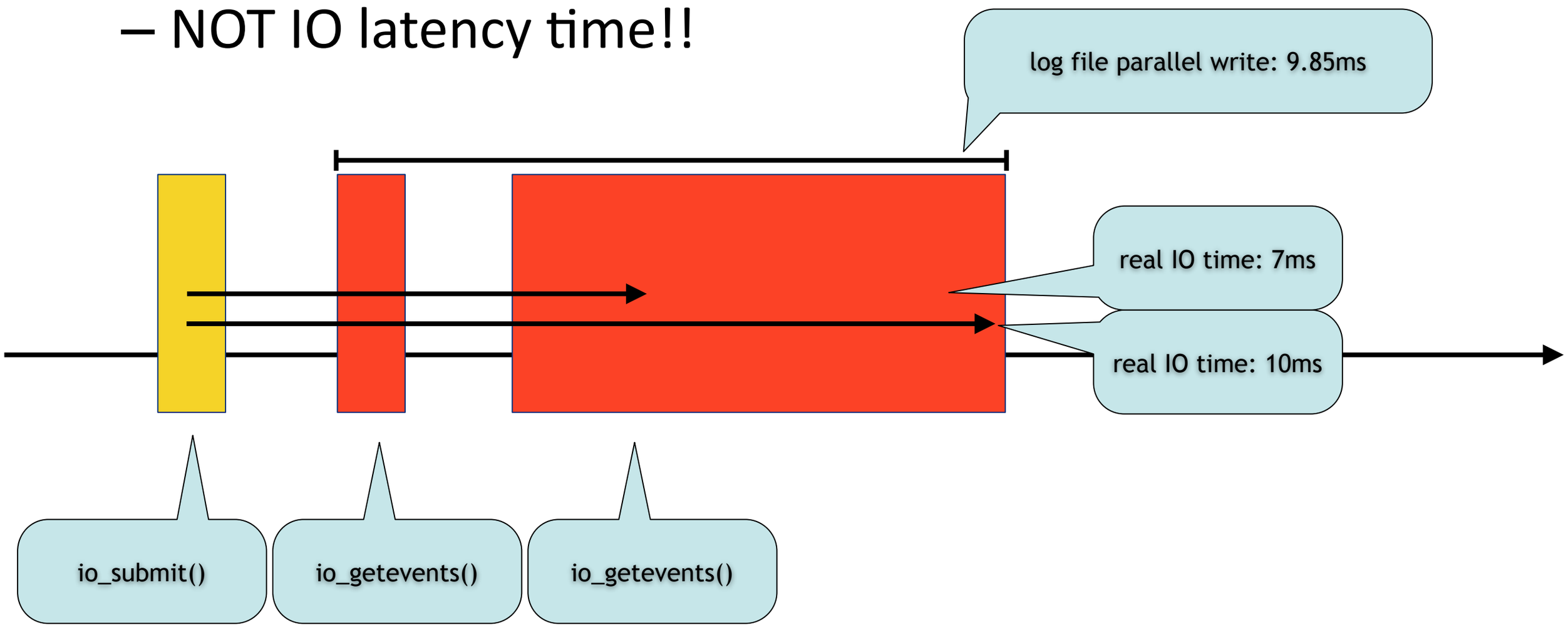
Write mode latch gets and frees:
‘messages’
‘mostly latch-free SCN’
‘lgwr LWN SCN’
‘KTF sga latch’
‘redo allocation’
‘messages’
‘redo writing’

Logwriter - writing



Logwriter - writing

- The event ‘log file parallel write’ is an *indicator* of IO wait time for the lgwr.
  - NOT IO latency time!!



Logwriter wait events

- rdbms ipc message
  - timeout: 300 (centiseconds; 3 seconds).
  - process sleeping ~ 3 seconds on semaphore.

- log file parallel write
  - files: number of log file members.
  - blocks: total number of log blocks written.
  - requests: ?
    - I've seen this differ from the actual numer of IO requests.

Logwriter wait events

- Let's switch the database to synchronous IO.
  - Some platforms have difficulty with AIO (HPUX!)
  - Got to check if your config does use AIO.
    - Found out by accident that ASM+NFS has no AIO by default.
  - Good to understand what the absence of AIO means.

- If you can't use AIO today, you are doing it WRONG!

log file parallel write

disk_asynch_io=FALSE (no AIO)

```
kslwtbctx
semtimedop - 458755 semid, timeout: $62 = {tv_sec = 3, tv_nsec = 0}
kslwtectx -- Previous wait time: 584208: rdbms ipc message

pwrite64 - fd, size - 256,512

kslwtbctx
kslwtectx -- Previous wait time: 782: log file parallel write

kslwtbctx
semtimedop - 458755 semid, timeout: $63 = {tv_sec = 2, tv_nsec = 310000000}
kslwtectx -- Previous wait time: 2315982: rdbms ipc message
```

timeout = 3s
wait time = 0.584208s
=> semaphore is posted!

two sequential writes
(two logfiles)

the wait begins AFTER the write (!)
it's also suspiciously fast (0.8ms)

log file parallel write

disk_asynch_io=FALSE (no AIO)

Let's add 100ms to the IOs (shell sleep 0.1)

```
kslwtbctx
semtimedop - 458755 semid, timeout: $3 = {tv_sec = 2, tv_nsec = 900000000}
kslwtectx -- Previous wait time: 2905568: rdbms ipc message

pwrite64 - fd, size - 256,512
 >sleep 0.1
pwrite64 - fd, size - 256,512
 >sleep 0.1

kslwtbctx
kslwtectx -- Previous wait time: 545: log file parallel write
```

Two writes again.
In the break a sleep of 100ms is added.
This should make the timing at least 200'000

The timing is 545 (0.5ms): timing is off.

log file parallel write

- Conclusion:
  - For at least Oracle version 11.2.
  - When synchronous IO (pwrite64()) is issued.
    - disk_asynch_io = FALSE (ASM)
    - filesystemio_options != “setall” or “asynch”
  - The wait event does not time the IO requests.

- How about the other log writer wait events?

control file sequential read

disk_asynch_io=FALSE (no AIO)

```
kslwtbctx
pread64 - fd, size - 256,16384
 >sleep 0.1
kslwtectx -- Previous wait time: 100323: control file sequential read
```

This event is correctly timed.

control file parallel write

disk_asynch_io=FALSE (no AIO)

```
pwrite64 - fd, size - 256,16384
 >sleep 0.1
kslwtbctx
kslwtectx -- Previous wait time: 705: control file parallel write
```

This event is incorrectly timed!

log file single write

disk_asynch_io=FALSE (no AIO)

```
kslwtbctx
pwrite64 - fd, size - 256,512
 >sleep 0.1
kslwtectx -- Previous wait time: 104594: log file single write
```

This event is correctly timed.

Logwriter wait events logswitch

- Some of these waits typically show up during a logswitch.
  - This are all the waits which are normally seen:
    - os thread startup (semctl()-semtimedop())
    - control file sequential read (pread64())
    - control file parallel write (io_submit()-io_getevents())
    - log file sequential read (pread64())
    - log file single write (pwrite64())
    - KSV master wait (semctl() post to dbwr)

- This is with AIO enabled!

Logwriter, timeout message

- Warning:

```
Warning: log write elapsed time 523ms, size 2760KB
```

- Printed in logwriter tracefile (NOT alert.log)
- Threshold set with parameter:
  - _side_channel_batch_timeout_ms (500ms)

Logwriter: disable logging

- The “forbidden switch”: _disable_logging
  - Do not use this for anything else than tests!

- Everything is done the same — no magic
  - Except the write by the lgwr to the logfiles
  - No ‘log file parallel write’
  - Redo/control/data files are synced with shut normal

- A way to test if lgwr IO influences db processing

Logwriter: exadata

- How does this look like on Exadata?

Logwriter: exadata

The dbwr semaphore sleep.

```
kslwtbctx
semtimedop - 3309577 semid, timeout: $24 = {tv_sec = 2, tv_nsec = 970000000}
kslwtectx -- Previous wait time: 2973630: rdbms ipc message

$25 = "oss_write"
$26 = "oss_write"
$27 = "oss_write"
$28 = "oss_write"
```

The writes are issued here.
There is no io_submit like wait. This is not timed.

```
kslwtbctx
$29 = "oss_wait"
$30 = "oss_wait"
$31 = "oss_wait"
$32 = "oss_wait"
kslwtectx -- Previous wait time: 2956: log file parallel write

kslwtbctx
semtimedop - 3309577 semid, timeout: $33 = {tv_sec = 3, tv_nsec = 0}
kslwtectx -- Previous wait time: 3004075: rdbms ipc message
```

The wait is log file parallel write, identical to non-exadata.
It seems to wait for all issued IOs

Database writer

- From the Oracle 11.2 concepts guide:
  - The DBW*n* process writes dirty buffers to disk under the following conditions:
    - When a server process cannot find a clean reusable buffer after scanning a threshold of buffers, it signals DBW*n* to write. DBW*n* writes dirty buffers to disk asynchronously if possible while performing other processing.
    - DBW*n* periodically writes buffers to advance the checkpoint, which is the position in the redo thread from which instance recovery begins. The log position of the checkpoint is determined by the oldest dirty buffer in the buffer cache.

Database writer, idle

- The 10046/8 trace shows:

```
*** 2013-12-31 00:45:51.088
WAIT #0: nam='rdbms ipc message' ela= 3006219 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447151086891
```

```
*** 2013-12-31 00:45:54.142
WAIT #0: nam='rdbms ipc message' ela= 3005237 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447154140873

*** 2013-12-31 00:45:57.197
WAIT #0: nam='rdbms ipc message' ela= 3005258 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447157195828

*** 2013-12-31 00:46:00.255
WAIT #0: nam='rdbms ipc message' ela= 3005716 timeout=300 p2=0 p3=0 obj#=-1
tim=1388447160253960
```

Database writer, idle

- "rdbms ipc message" indicates a sleep/idle event
  - There isn't an indication dbw0 writes something:

```
semtimedop(983043, {{14, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)
getrusage(RUSAGE_SELF, {ru_utime={0, 31000}, ru_stime={0, 89000}, ...}) = 0
times({tms_utime=3, tms_stime=8, tms_cutime=0, tms_cstime=0}) = 431915044
times({tms_utime=3, tms_stime=8, tms_cutime=0, tms_cstime=0}) = 431915044
times({tms_utime=3, tms_stime=8, tms_cutime=0, tms_cstime=0}) = 431915044
semtimedop(983043, {{14, -1, 0}}, 1, {3, 0}) = -1 EAGAIN (Resource temporarily unavailable)
…etc…
```

Database writer, idle

- It does look in the /proc filesystem to the ‘stat’ file of a certain process:

```
open("/proc/2218/stat", O_RDONLY)       = 21
read(21, "2218 (oracle) S 1 2218 2218 0 -1"..., 999) = 209
close(21)
```

- It does so every 20th time (20*3)= 60 sec.
- The PID is PMON.

Database writer, idle

- Recap:
  - In an idle database.
  - The dbwr sleeps on a semaphore for 3 seconds.
    - Then wakes up, and sets up the semaphore/sleep again.
    - Processes sleeping on a semaphore do not spend CPU
  - Every minute, dbwr reads pmon's process stats.
  - dbwr doesn’t write if there’s no need.

Database writer, force write

- We can force the dbwr to write:
  - Dirty some blocks (insert a row into a table).
  - Force a thread checkpoint (alter system checkpoint).

* There are multiple ways, this is one of them.

Database writer, force write

10046/8 trace:

db file async I/O submit?!
It looks like the io_submit() call is instrumented for the dbwr!
But what does ‘requests=3’ mean for a single row update checkpoint?

elapsed time = 2.26 sec.
So the dbwr is posted!

```
WAIT #0: nam='rdbms ipc message'  ela= 2261867 timeout=300 p2=0 p3=0 obj#=-1
tim=1388716669735046
WAIT #0: nam='db file async I/O submit' ela= 0 requests=3 interrupt=0 timeout=0
obj#=-1 tim=1388716669735493
WAIT #0: nam='db file parallel write' ela= 21 requests=1 interrupt=0
timeout=2147483647 obj#=-1 tim=1388716669735566

*** 2014-01-03 03:37:50.465
WAIT #0: nam='rdbms ipc message' ela= 729110 timeout=73 p2=0 p3=0 obj#=-1
tim=1388716670464967
```

And the write, via the event ‘db file parallel write’.

dbwr, sql_trace + strace

- Let’s take a look at the Oracle wait events, together with the actual system calls.

- That is:
  - Setting a 10046/8 event for trace and waits.
  - Execute strace with ‘-e write=all -e all’

dbwr, sql_trace + strace

```
io_submit(140195085938688, 3, {{0x7f81b622ab10, 0, 1, 0, 256}, {0x7f81b622a8a0, 0,
1, 0, 256}, {0x7f81b622a630, 0, 1, 0, 256}}) = 3
write(13, "WAIT #0: nam='db file async I/O "..., 108) = 108
 | 00000  57 41 49 54 20 23 30 3a  20 6e 61 6d 3d 27 64 62  WAIT #0:  nam='db |
 | 00010  20 66 69 6c 65 20 61 73  79 6e 63 20 49 2f 4f 20   file as ync I/O  |
 | 00020  73 75 62 6d 69 74 27 20  65 6c 61 3d 20 31 20 72  submit'  ela= 1 r |
 | 00030  65 71 75 65 73 74 73 3d  33 20 69 6e 74 65 72 72  equests= 3 interr |
 | 00040  75 70 74 3d 30 20 74 69  6d 65 6f 75 74 3d 30 20  upt=0 ti meout=0  |
 | 00050  6f 62 6a 23 3d 2d 31 20  74 69 6d 3d 31 33 38 38  obj#=-1  tim=1388 |
 | 00060  39 37 37 36 35 31 38 30  34 32 36 31              97765180 4261     |
io_getevents(140195085938688, 1, 128, {{0x7f81b622ab10, 0x7f81b622ab10, 8192, 0},
{0x7f81b622a8a0, 0x7f81b622a8a0, 8192, 0}, {0x7f81b622a630, 0x7f81b622a630, 8192,
0}}, {600, 0}) = 3
write(13, "WAIT #0: nam='db file parallel w"..., 116) = 116
 | 00000  57 41 49 54 20 23 30 3a  20 6e 61 6d 3d 27 64 62  WAIT #0:  nam='db |
 | 00010  20 66 69 6c 65 20 70 61  72 61 6c 6c 65 6c 20 77   file pa rallel w |
 | 00020  72 69 74 65 27 20 65 6c  61 3d 20 35 38 20 72 65  rite' el a= 58 re |
 | 00030  71 75 65 73 74 73 3d 31  20 69 6e 74 65 72 72 75  quests=1  interru |
 | 00040  70 74 3d 30 20 74 69 6d  65 6f 75 74 3d 32 31 34  pt=0 tim eout=214 |
 | 00050  37 34 38 33 36 34 37 20  6f 62 6a 23 3d 2d 31 20  7483647  obj#=-1  |
 | 00060  74 69 6d 3d 31 33 38 38  39 37 37 36 35 31 38 30  tim=1388 97765180 |
 | 00070  34 35 37 39                                       4579              |
```

dbwr, sql_trace + strace

```
io_submit(140195085938688, 3, {{0x7f81b622ab10, 0, 1, 0, 256}, {0x7f81b622a8a0, 0,
1, 0, 256}, {0x7f81b622a630, 0, 1, 0, 256}}) = 3
write(13, "WAIT #0: nam='db file async I/O "..., 108) = 108
 | 00000  57 41 49 54 20 23 30 3a  20 6e 61 6d 3d 27 64 62  WAIT #0:  nam='db |
 | 00010  20 66 69 6c 65 20 61 73  79 6e 63 20 49 2f 4f 20   file as ync I/O  |
 | 00020  73 75 62 6d 69 74 27 20  65 6c 61 3d 20 31 20 72  submit'  ela= 1 r |
 | 00030  65 71 75 65 73 74 73 3d  33 20 69 6e 74 65 72 72  equests= 3 interr |
 | 00040  75 70 74 3d 30 20 74 69  6d 65 6f 75 74 3d 30 20  upt=0 ti meout=0  |
 | 00050  6f 62 6a 23 3d 2d 31 20  74 69 6d 3d 31 33 38 38  obj#=-1  tim=1388 |
 | 00060  39 37 37 36 35 31 38 30  34 32 36 31              97765180 4261     |
io_getevents(140195085938688, 1, 128, {{0x7f81b622ab10, 0x7f81b622ab10, 8192, 0},
{0x7f81b622a8a0, 0x7f81b622a8a0, 8192, 0}, {0x7f81b622a630, 0x7f81b622a630, 8192,
0}}, {600, 0}) = 3
write(13, "WAIT #0: nam='db file parallel w"..., 116) = 116
 | 00000  57 41 49 54 20 23 30 3a  20 6e 61 6d 3d 27 64 62  WAIT #0:  nam='db |
 | 00010  20 66 69 6c 65 20 70 61  72 61 6c 6c 65 6c 20 77   file pa rallel w |
 | 00020  72 69 74 65 27 20 65 6c  61 3d 20 35 38 20 72 65  rite' el a= 58 re |
 | 00030  71 75 65 73 74 73 3d 31  20 69 6e 74 65 72 72 75  quests=1  interru |
 | 00040  70 74 3d 30 20 74 69 6d  65 6f 75 74 3d 32 31 34  pt=0 tim eout=214 |
 | 00050  37 34 38 33 36 34 37 20  6f 62 6a 23 3d 2d 31 20  7483647  obj#=-1  |
 | 00060  74 69 6d 3d 31 33 38 38  39 37 37 36 35 31 38 30  tim=1388 97765180 |
 | 00070  34 35 37 39                                       4579              |
```

dbwr, sql_trace + strace

```
io_submit(140195085938688, 3, {{0x7f81b622ab10, 0, 1, 0, 256}, {0x7f81b622a8a0, 0,
1, 0, 256}, {0x7f81b622a630, 0, 1, 0, 256}}) = 3
write(13, "WAIT #0: nam='db file async I/O "..., 108) = 108
 | 00000  57 41 49 54 20 23 30 3a  20 6e 61 6d 3d 27 64 62  WAIT #0:  nam='db |
 | 00010  20 66 69 6c 65 20 61 73  79 6e 63 20 49 2f 4f 20   file as ync I/O  |
                                                 20 31 20 72  submit'  ela= 1 r |
                                                 74 65 7     equests=  3 interr |
                                                 74 3d 3     upt=0  ti meout=0  |
 | 00050  6f 62 6a 23 3d 2d 3  20  74 69 6d 3d 31 33 38 38  obj#=-1  tim=1388 |
 | 00060  39 37 37 36 35 31 38 30  34 32 36 31              97765180 4261     |
```

This is the MINIMAL number of requests to reap before successful.
(min_nr - see man io_getevents)

?

?

```
io_getevents(140195085938688, 1, 128, {{0x7f81b622ab10, 0x7f81b622ab10, 8192, 0},
{0x7f81b622a8a0, 0x7f81b622a8a0, 8192, 0}, {0x7f81b622a630, 0x7f81b622a630, 8192,
0}}, {600, 0}) = 3
write(13, "WAIT #0: nam='db file parallel w"..., 116) = 116
                                                 3d 27 64 62  WAIT #0:  nam='db |
                                                 6 6c 20 77   file pa rallel w |
                                                 8 20 72 65  rite' el a= 58 re |
 | 00030  71 75     73 74 73 3d 31  20 69 6e 74 65 72 7     quests=1  interru |
                                                 d 32 31 3   pt=0 tim eout=214 |
                                                 d 2d 31 20  7483647  obj#=-1  |
 | 00060  74 69 6d 3d 31 33 38 38  39 37 37 36 35 31 38 30  tim=1388 97765180 |
 | 00070  34 35 37 39                                       4579              |
```

The timeout for io_getevents() is set to 600 seconds.
struct timespec { sec, nsec }

Despite only needing 1 request, this call returned all 3.
This information is NOT EXTERNALISED (!!)

enkitec

dbwr, db file async I/O submit

- Let's take a look at the what the documentation says about "db file async I/O submit":

(That's right…nothing)

dbwr, db file async I/O submit

- My Oracle Support on “db file async I/O submit”:

'db file async I/O submit' when FILESYSTEMIO_OPTIONS=NONE

[Article ID 1274737.1]

How To Address High Wait Times for the 'Direct Path Write Temp ' Wait Event

[Article ID 1576956.1]

- Both don’t describe what this event is.
- 1st note is only for filesystemio_options=NONE and describes the event not being tracked prior to version 11.2.0.2.

dbwr, db file async I/O submit

- So the question is:
  - What DOES the event “db file async I/O submit” mean?
- The obvious answer is:
  - Instrumentation of the io_submit() call.
- *My* answer is:
  - Don’t know.
  - But NOT the instrumentation of io_submit().

dbwr, db file async I/O submit

- This is a trace of the relevant C functions:

```
kslwtbctx
kslwtectx -- Previous wait time: 236317: rdbms ipc message

io_submit - 3,45e5a000 - nr,ctx
kslwtbctx
kslwtectx -- Previous wait time: 688: db file async I/O submit

kslwtbctx
io_getevents - 1,45e5a000 - minnr,ctx,timeout: $3 = {tv_sec = 600, tv_nsec = 0}
skgfr_return64 - 3 IOs returned
kslwtectx -- Previous wait time: 9604: db file parallel write
```

Waiting on a semaphore to be posted.

io_submit() for 3 IOs

The begin of the wait starts AFTER the io_submit()?

io_getevevents() is properly timed. min_nr=1, got 3 IOs

dbwr, db file async I/O submit

- Trace with sleep 0.1 in the break on io_submit()

Waiting on a semaphore to be posted.

```
kslwtbctx
kslwtectx -- Previous wait time: 385794: rdbms ipc message

io_submit - 3,45e5a000 - nr,ctx
 > sleep 0.1
kslwtbctx
kslwtectx -- Previous wait time: 428: db file async I/O submit

kslwtbctx
io_getevents - 1,45e5a000 - minnr,ctx,timeout: $37 = {tv_sec = 600, tv_nsec = 0}
skgfr_return64 - 3 IOs returned
kslwtectx -- Previous wait time: 8053: db file parallel write
```

io_submit() for 3 IOs + sleep of 100’000

Wait time too low. io_submit() is not timed.

dbwr, db file parallel write

- Let’s look at the “db file parallel write” event.

dbwr, db file parallel write

- Description from the Reference Guide:

Correct

Correct…but only if AIO is enabled.

db file parallel write

This event occurs in the DBWR. It indicates that the DBWR is performing a parallel write to files and blocks. When the last I/O has gone to disk, the wait ends.

Incorrect

Wait Time: Wait until all of the I/Os are completed

Incorrect

Parameter
Description

Incorrect

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

interrupt:

Empty?

timeout: This indicates the timeout value in hundredths of a second to wait for the I/O completion.

Probably incorrect.
Or does a timeout of
2’147’483’647
/100/60/60/24=
248.55 days
Make sense to
anybody?

dbwr, db file parallel write

- Recap of previous traced calls:

```
kslwtbctx
kslwtectx -- Previous wait time: 236317: rdbms ipc message

io_submit - 3,45e5a000 - nr,ctx
kslwtbctx
kslwtectx -- Previous wait time: 688: db file async I/O submit

kslwtbctx
io_getevents - 1,45e5a000 - minnr,ctx,timeout: $3 = {tv_sec = 600, tv_nsec = 0}
skgfr_return64 - 3 IOs returned
kslwtectx -- Previous wait time: 9604: db file parallel write
```

So….how about severely limiting OS IO capacity and see what happens?

dbwr, db file parallel write

- Database writer — severely limited IO (1 IOPS)

```
io_submit - 366,45e5a000
 kslwtbctx
 kslwtectx -- Previous wait time: 1070: db file async I/O submit

 kslwtbctx
io_getevents - 100,45e5a000 - minnr,ctx,timeout: $7 = {tv_sec = 600, tv_nsec = 0}
 skgfr_return64 - 100 IOs returned
 kslwtectx -- Previous wait time: 109334845: db file parallel write

io_getevents - 128,45e5a000 - minnr,ctx,timeout: $8 = {tv_sec = 0, tv_nsec = 0}
io_getevents - 128,45e5a000 - minnr,ctx,timeout: $9 = {tv_sec = 0, tv_nsec = 0}

io_submit - 73,45e5a000 - nr,ctx
 kslwtbctx
 kslwtectx -- Previous wait time: 486: db file async I/O submit
```

366 IO requests are submitted onto the OS.

But only 100 IOs are needed to satisfy io_getevents()
Which it does in this case… leaving outstanding IOs

The dbwr starts issuing non-blocking calls to reap IOs!
It seems to be always 2 if outstanding IOs remain.
Minnr = # outstanding IOs, max 128.

dbwr, db file parallel write

- This got me thinking…
- The dbwr submits the IOs it needs to write.
- But it waits for a variable amount of IOs to finish.
  - Wait event ‘db file parallel write’.
  - Amount seems 33-25% of submitted IOs*
  - After that, 2 tries to reap the remaining IOs*
  - Then either submit again, DFPW until IOs reaped or back to sleeping on semaphore.

dbwr, db file parallel write

- This means ‘db file parallel write’ is not:
  - Physical IO indicator.
  - IO latency timing

- I’ve come to the conclusion that the blocking io_getevents call for a number of IOs of the dbwr is an IO limiter.
- …and ’db file parallel write’ is the timing of it.

dbwr, synchronous IO

- Let's turn AIO off again.
  - To simulate this, I've set disk_asynch_io to FALSE.
- And set a 10046/8 trace and strace on the dbwr.
- And issue the SQLs as before:
  - insert into followed by commit
  - alter system checkpoint

dbwr, synchronous IO

```
pwrite(256, "\6\242\0\0\207                                                \0"...,
8192, 8053121024) = 8192
pwrite(256, "\2\242\0\0\246\0\300\0hG\17\0\0\0\1\4\220T\0\0\3\0\r\0j
\3\0\0\203\0\17\17"..., 8192, 7443103744) = 8192
pwrite(256, "&\242\0\0\240\0\300\0hG
\17\0\0\0\2\4\372\216\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 8192, 7443054592) = 8192
write(11, "WAIT #0: nam='db file parallel w"..., 107) = 107
```

3 pwrite() calls. This is synchronous IO!

```
 | 00000  57 41 49 54 20 23 30 3a  20 6e 61 6d 3d 27 64 62  WAIT #0:  nam='db |
 |                                                 77   file pa rallel w |
 |                                                 65  rite' el a= 20 re |
 | 00030  71 75 65 73 74 73 3d 33  20 69 6e 74 65 72 72 75  quests=3  interru |
 | 00040  70 74 3d 30 20 74 69 6d  65 6f 75 74 3d 30 20 6f  pt=0 tim eout=0 o |
 | 00050  62 6a 23 3d 2d 31 20 74  69 6d 3d 31 33 38 39 38  bj#=-1 t im=13898 |
 | 00060  30 32 32 38 31 34 35 36  37 34 30                 02281456 740      |
write(11, "WAIT #0: nam='db file parallel w"..., 106) = 106
 | 00000  57 41 49 54 20 23 30 3a  20 6e 61 6d 3d 27 64 62  WAIT #0:  nam='db |
 |                                                 77   file pa rallel w |
 |                                                 71  rite' el a= 1 req |
 | 00030  75 65 73 74 73 3d 33 20  69 6e 74 65 72 72 75 70  uests=3  interrup |
 | 00040  74 3d 30 20 74 69 6d 65  6f 75 74 3d 30 20 6f 62  t=0 time out=0 ob |
 | 00050  6a 23 3d 2d 31 20 74 69  6d 3d 31 33 38 39 38 30  j#=-1 ti m=138980 |
 | 00060  32 32 38 31 34 35 38 34  39 32                    22814584 92       |
```

The db file parallel write wait event shows 3 requests!

But why a second db file parallel write wait event?

dbwr, synchronous IO

- There's no 'db file async I/O submit' wait anymore.
  - Which is good, because SIO has no submit phase.
- The 'db file parallel write' waits seem suspicious.
  - It seems like the wait for DFPW is issued twice.
  - Further investigation shows that it does.
    - My guess this is a bug in the sync. IO implementation.
- Let's look a level deeper and see if there's more to see.

dbwr, synchronous IO

This is clearly the semaphore being posted: timeout=3s,
wait time = 1239,2ms

```
kslwtbctx
semtimedop - 458755 semid, timeout: $18 = {tv_sec = 3, tv_nsec = 0}
kslwtectx -- Previous wait time: 1239214: rdbms ipc message

pwrite64 - fd, size - 256,8192
pwrite64 - fd, size - 256,8192
pwrite64 - fd, size - 256,8192

kslwtbctx
kslwtectx -- Previous wait time: 949: db file parallel write

kslwtbctx
kslwtectx -- Previous wait time: 650: db file parallel write

kslwtbctx
semtimedop - 458755 semid, timeout: $19 = {tv_sec = 1, tv_nsec = 620000000}
```

3 IO's *in serial* using pwrite().
The only possibility if there isn't AIO of course.

Two db file parallel write (which aren't parallel) for which
both the begin of the waits are started AFTER the IO (!!)

After the IOs are done, the dbwr continues sleeping.

dbwr, synchronous IO

- Let's do the same trick as done earlier:
  - In gdb, add "shell sleep 0.1" to the pwrite call.
  - This makes the execution of this call take 100ms longer.
  - To see if there's still some way Oracle times it properly.

dbwr, synchronous IO

```
kslwtbctx
semtimedop - 458755 semid, timeout: $23 = {tv_sec = 3, tv_nsec = 0}
kslwtectx -- Previous wait time: 92080: rdbms ipc message

pwrite64 - fd, size - 256,8192
 > shell sleep 0.1
pwrite64 - fd, size - 256,8192
 > shell sleep 0.1
pwrite64 - fd, size - 256,8192
 > shell sleep 0.1

kslwtbctx
kslwtectx -- Previous wait time: 478: db file parallel write

kslwtbctx
kslwtectx -- Previous wait time: 495: db file parallel write

kslwtbctx
semtimedop - 458755 semid, timeout: $24 = {tv_sec = 2, tv_nsec = 460000000}
```

The 3 IOs again, each sleeps in pwrite() for 100ms (0.1s)

Yet the 'db file parallel write' wait shows a waiting time of 478; which is 0.478ms: the timing is wrong.

dbwr, synchronous IO

- So, my conclusion on the wait events for the dbwr with synchronous IO:
  - The events are not properly timed
  - It seems like the wait for DFPW is issued twice.
  - Further investigation shows that it does.
    - My guess this is a bug in the sync. IO implementation.

dbwr: exadata

- How does this look like on Exadata?

dbwr: exadata

```
kslwtbctx
semtimedop - 3309577 semid, timeout: $389 = {tv_sec = 3, tv_nsec = 0}
kslwtectx -- Previous wait time: 1266041: rdbms ipc message
$390 = "oss_write"
$391 = "oss_write"
$392 = "oss_write"
$393 = "oss_write"
$394 = "oss_write"
$395 = "oss_write"

kslwtbctx
kslwtectx -- Previous wait time: 684: db file async I/O submit

kslwtbctx
$396 = "oss_wait"
$397 = "oss_wait"
kslwtectx -- Previous wait time: 2001: db file parallel write
$398 = "oss_wait"
$399 = "oss_wait"
$400 = "oss_wait"
$401 = "oss_wait"

semctl - 3309577,23,16 - semid, semnum, cmd
kslwtbctx
semtimedop - 3309577 semid, timeout: $402 = {tv_sec = 1, tv_nsec = 630000000}
kslwtectx -- Previous wait time: 1634299: rdbms ipc message
```

The dbwr semaphore sleep.

The writes are issued here.
This is not timed.

There is the db file async I/O submit.
Again, it doesn't seem to time any of the typical IO calls!

And there we got the db file parallel write.
It does seem to always time two oss_wait() calls…

But it looks for more IOs to finish, alike the trailing io_getevents() calls.
I am quite sure oss_wait() is a blocking call…

Conclusion

- Logwriter:
  - When idle, is sleeping on a semaphore/rdbms ipc message
  - Gets posted with semctl() to do work.
  - Only writes when it needs to do so.
  - Version 11.2.0.3: two methods for posting FGs:
    - Polling and post/wait.
    - Post/wait is default, might switch to polling.
    - Notification of switch is in log writer trace file.
    - Polling/nanosleep() time is variable.

Conclusion

- Logwriter:
  - Log file parallel write
    - AIO: two io_getevents() calls.
    - AIO: time waiting for all lgwr submitted IOs to finish.
      - **Not IO latency time!**
  - SIO: does not do parallel writes, but serial.
  - SIO: does not time IO.

Conclusion

- Logwriter:
  - Wait event IO timing:
    - All the '* parallel read' and '* parallel write' events do not seem to time IO correctly with synchronous IO.
    - All the events which cover single block IOs do use synchronous IO calls, even with asynchronous IO set.
  - Logwriter writes a warning when IO time exceeds 500ms in the log writer trace file.
  - _disable_logging *only* disables write to logs.

Conclusion

- Database writer:
  - When idle, is sleeping on a semaphore/rdbms ipc message
  - Gets posted with semctl() to do work.
  - Only writes when it needs to do so.
  - Since version 11.2.0.2, event ‘db file async I/O submit’:
    - Is not shown with synchronous I/O.
    - Shows the actual amount of IOs submitted.
    - Does not time io_submit()
    - Unknown what or if it times something.

Conclusion

- Database writer:
  - Event ‘db file parallel write’:
    - Shows the minimal number io_getevents() waits for.
    - The number of requests it waits for varies, but mostly seems to be ~ 25-33% of submitted IOs.
  - After the timed, blocking, io_getevents() call, it issues two non-blocking io_getevents() calls for the remaining non-reaped IOs, if any.
  - My current idea is the blocking io_getevents() call is an IO throttle mechanism.

Conclusion

- Database writer:
  - Event ‘db file parallel write’, with synchronous IO:
    - pwrite64() calls are issued serially.
      - These are not timed.
    - The event is triggered twice.

  - On exadata, two out of the total number of oss_wait() calls are timed with the event ‘db file parallel write’.

Q & A

Thanks & Links

- Enkitec
- Tanel Poder, Martin Bach, Klaas-Jan Jongsma, Jeremy Schneider, Karl Arao, Michael Fontana.
- http://www.pythian.com/blog/adaptive-log-file-sync-oracle-please-dont-do-that-again/
- http://files.e2sn.com/slides/Tanel_Poder_log_file_sync.pdf