DTCC

数/造/未/来

第十二届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2021

2021 年 10 月 18 日 - 20 日 | 北京国际会议中心

IT168.com ChinaUnix.net ITPUB

摸着Oracle过河 ----

大幅提升PostgreSQL性能分享

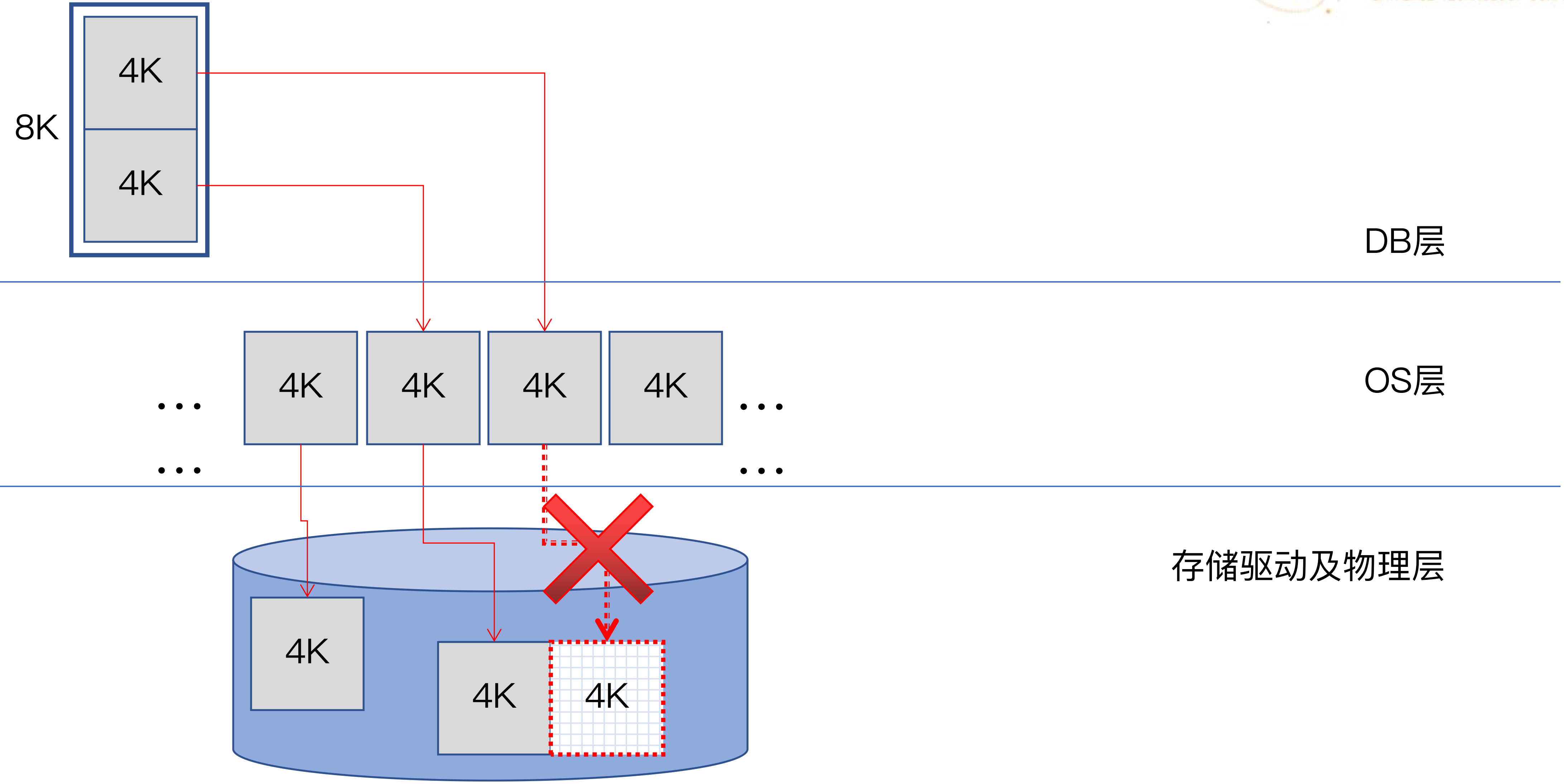
吕海波（VAGE）

主题：

优化PostgreSQL Partial Writes（页裂）体系

大幅提升整体性能

页分裂：部分写，Partial Writes



页分裂：部分写，Partial Writes

➢ 八仙过海　各显神通

- MySQL解决方案：Double Write
- Oracle解决方案：众说纷纭（高端硬件说、自有原子写说）
- PG解决方案：Full Page Writes（简称FPW）

页分裂：部分写，Partial Writes

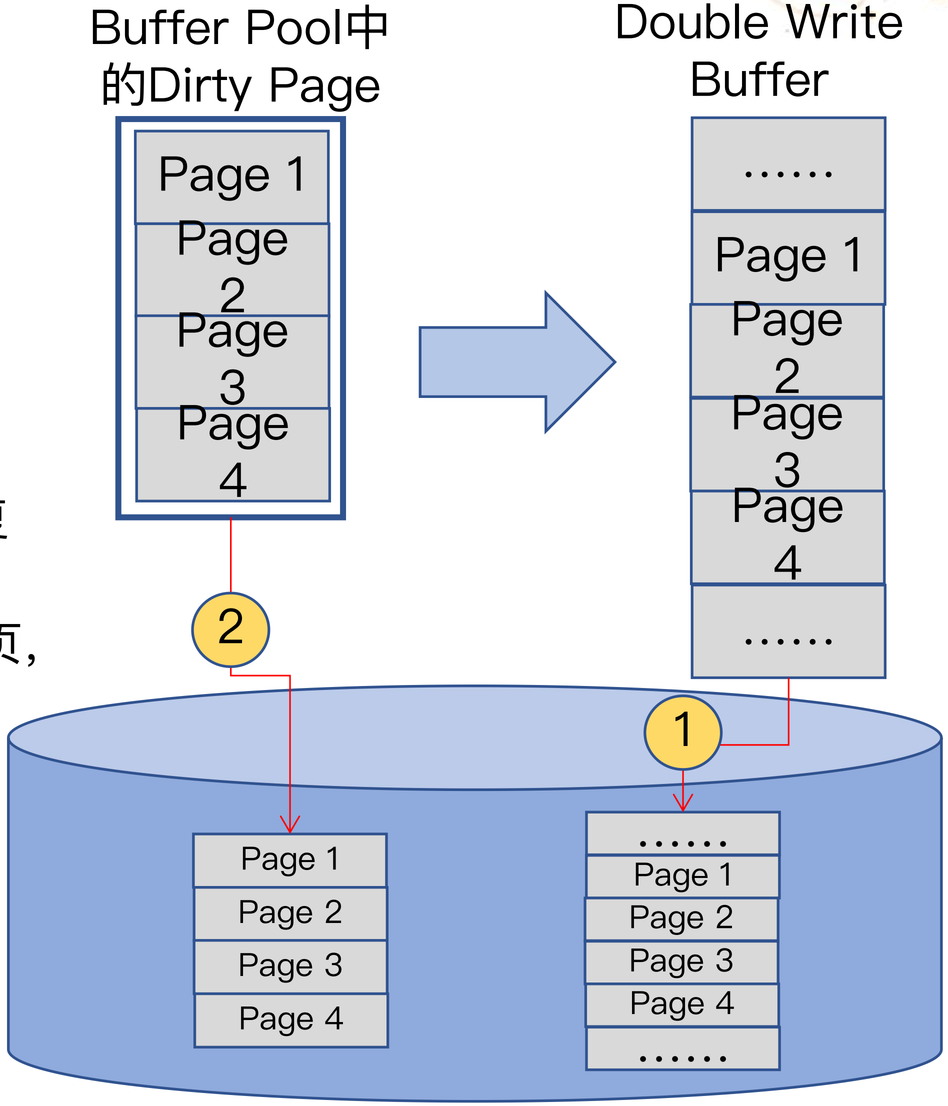
➢ MySQL Double Write

先写Double Write Buffer

再写Buffer Pool中的脏页

脏页写失败了，使用Double Write中的页进行恢复

Double Write写失败了，不写Buffer Pool中的脏页，脏页直接丢弃，依赖BinLog恢复。



页分裂：部分写，Partial Writes

➢ MySQL Double Write

先写Double Write Buffer

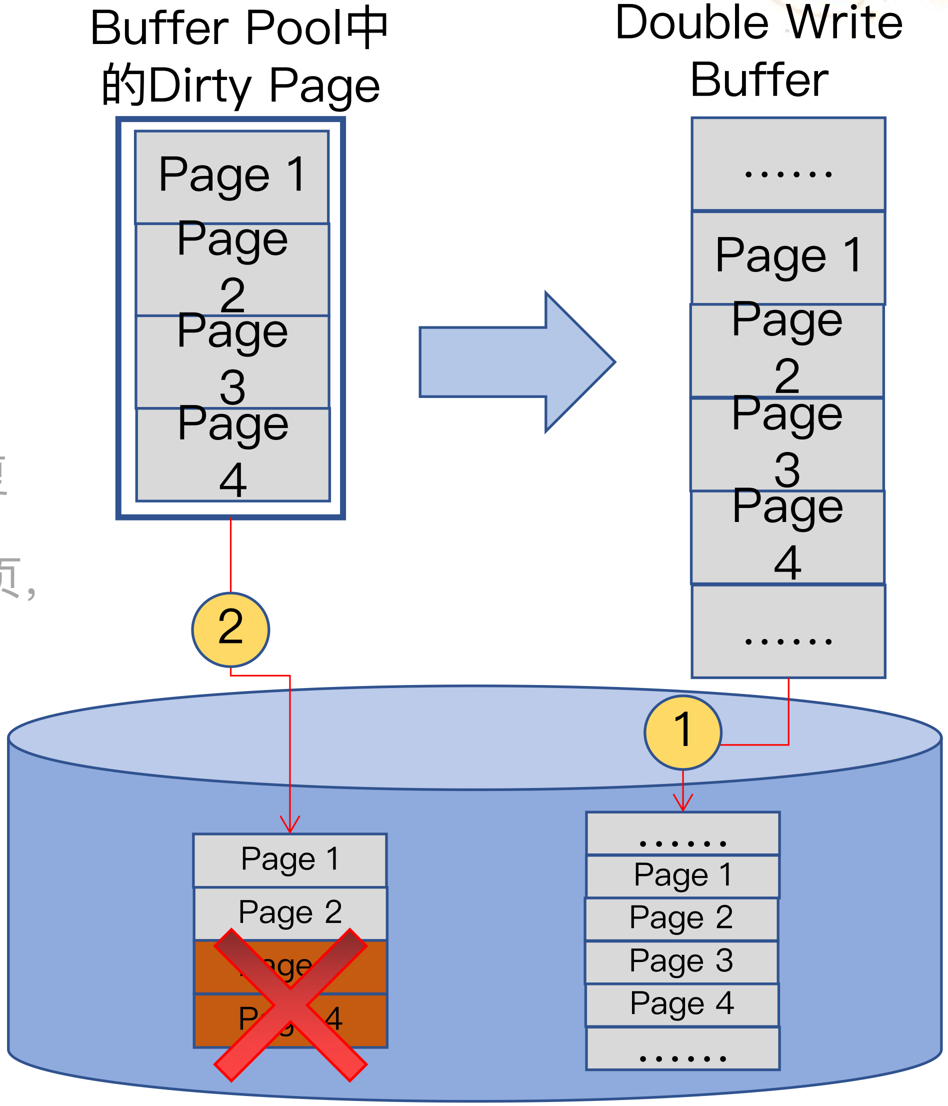
再写Buffer Pool中的脏页

脏页写失败了，使用Double Write中的页进行恢复

Double Write写失败了，不写Buffer Pool中的脏页，脏页直接丢弃，依赖BinLog恢复。

为什么MySQL必须要依赖双写？

为什么不能使用BinLog进行恢复？



页分裂：部分写，Partial Writes

➢ MySQL Double Write

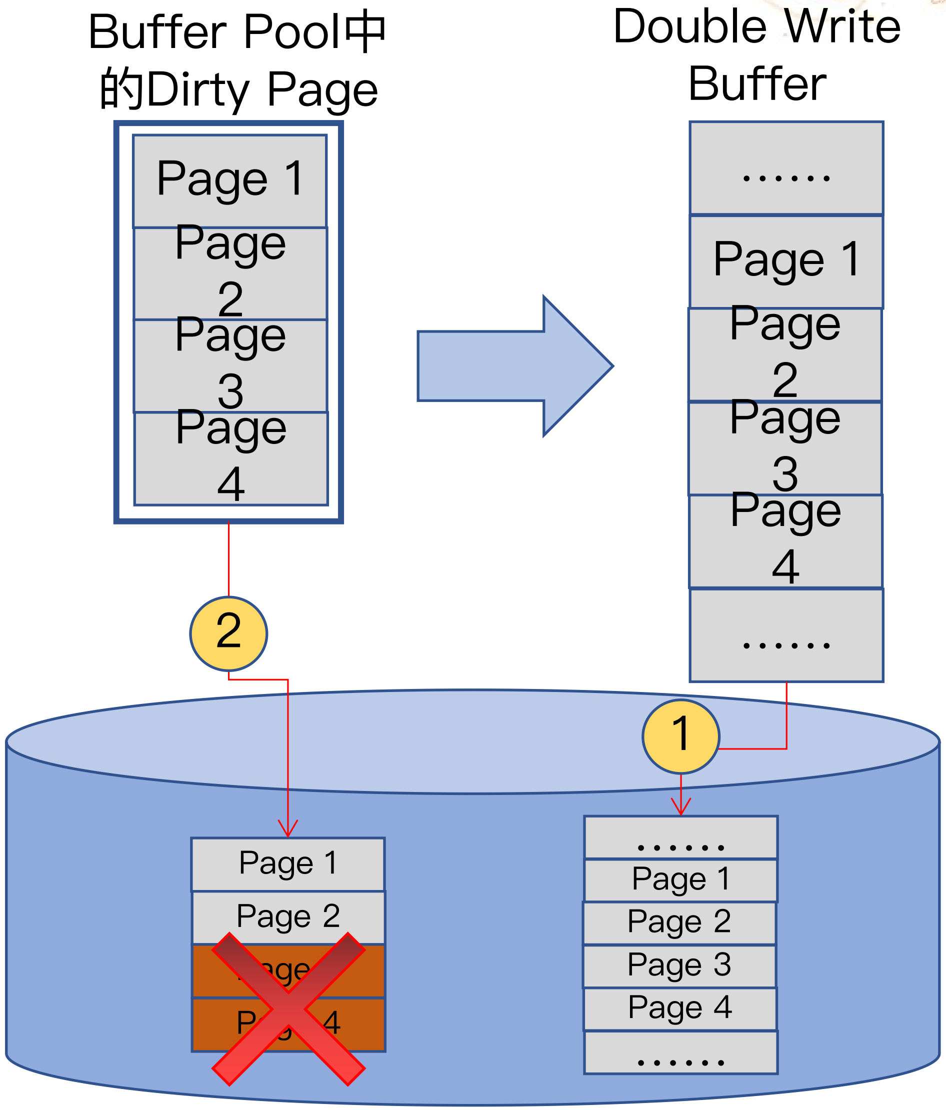
以Insert为例：

脏页中是Insert的新行，

BinLog记录的是SQL（逻辑日志），

使用BinLog恢复相当于再次执行Insert，

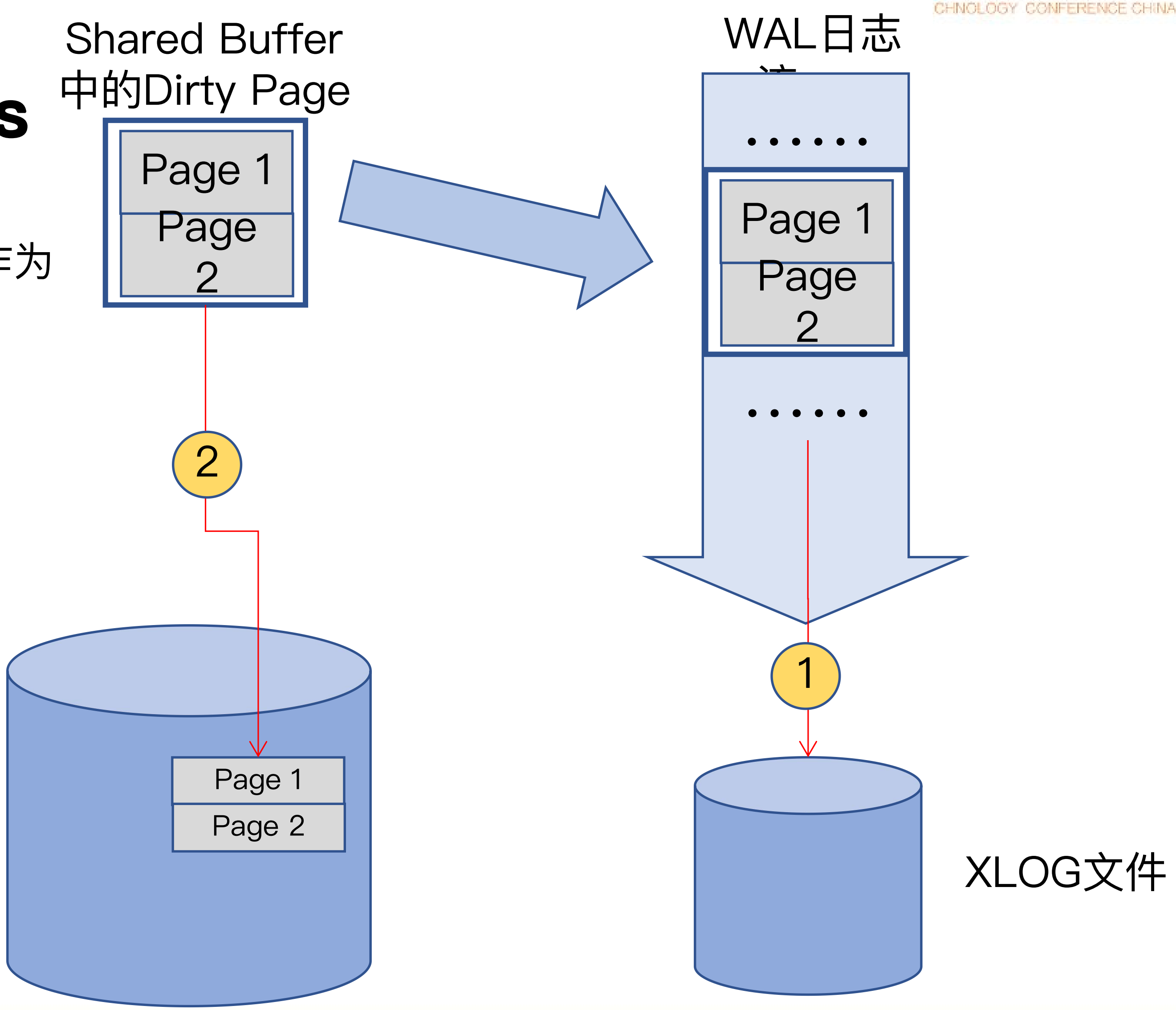
因此，



页分裂：部分写，Partial Writes

- **PostgreSQL Full Page Writes**

每次Checkpoint后，首次修改页，拷贝整页到日志中作为备份。

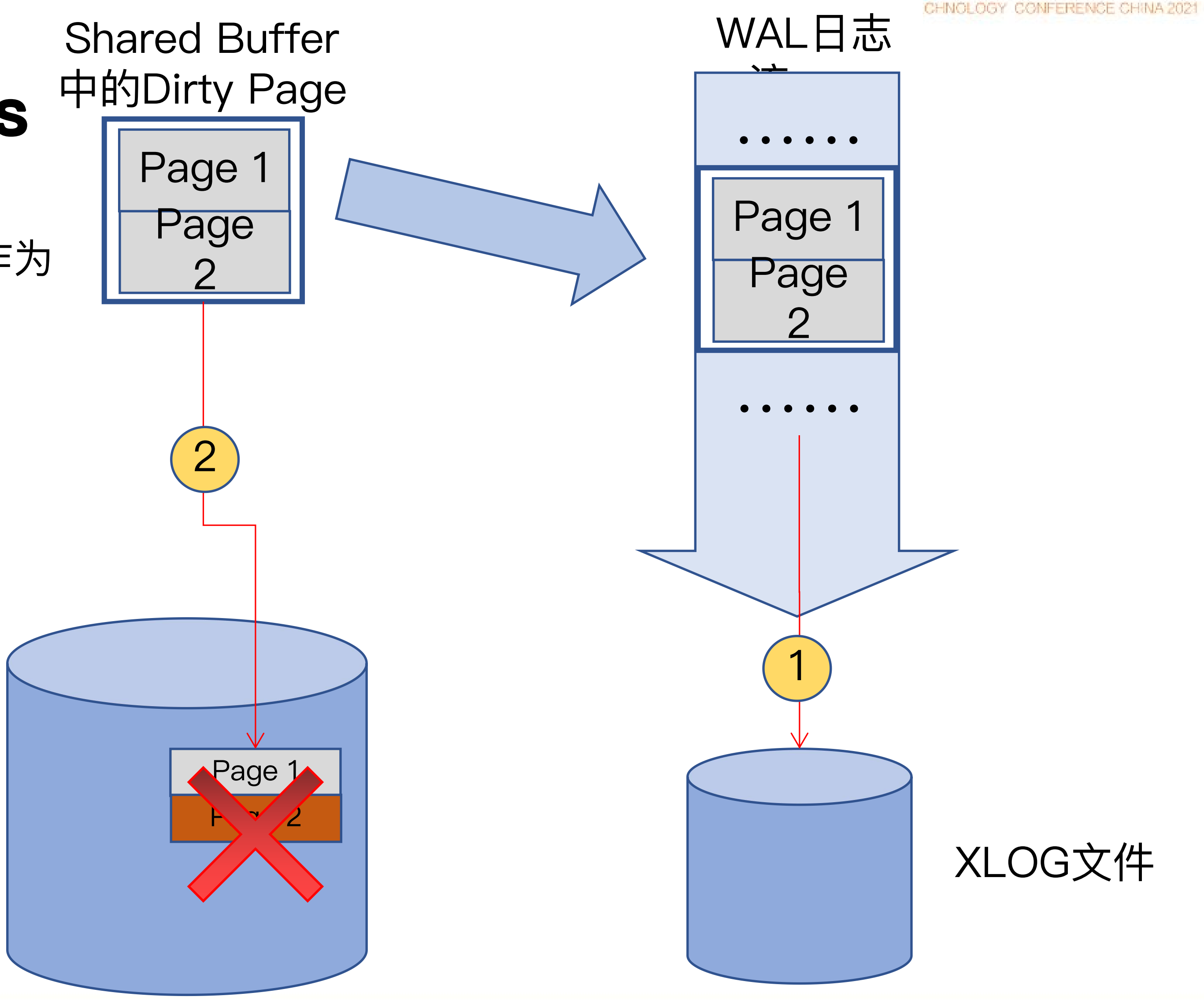


页分裂：部分写，Partial Writes

➢ PostgreSQL Full Page Writes

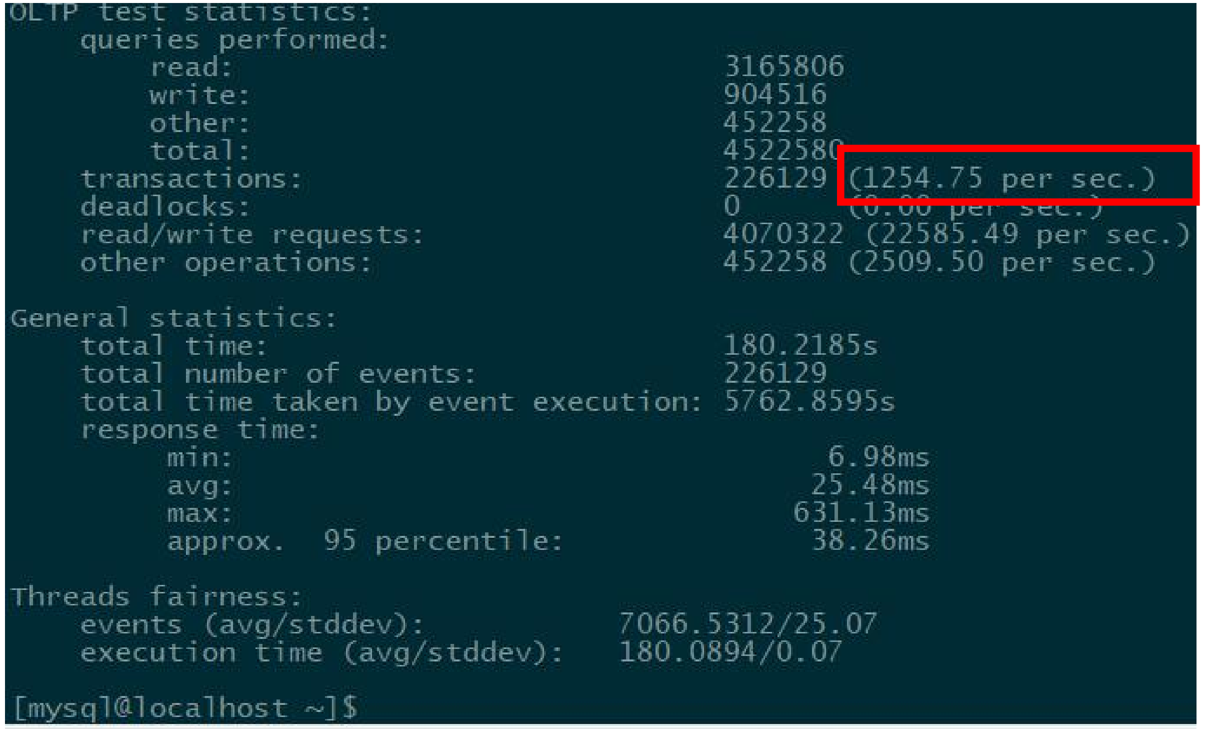
每次Checkpoint后，首次修改页，拷贝整页到日志中作为备份。

出现Partial Writes，以此备份为基础，对页进行恢复。

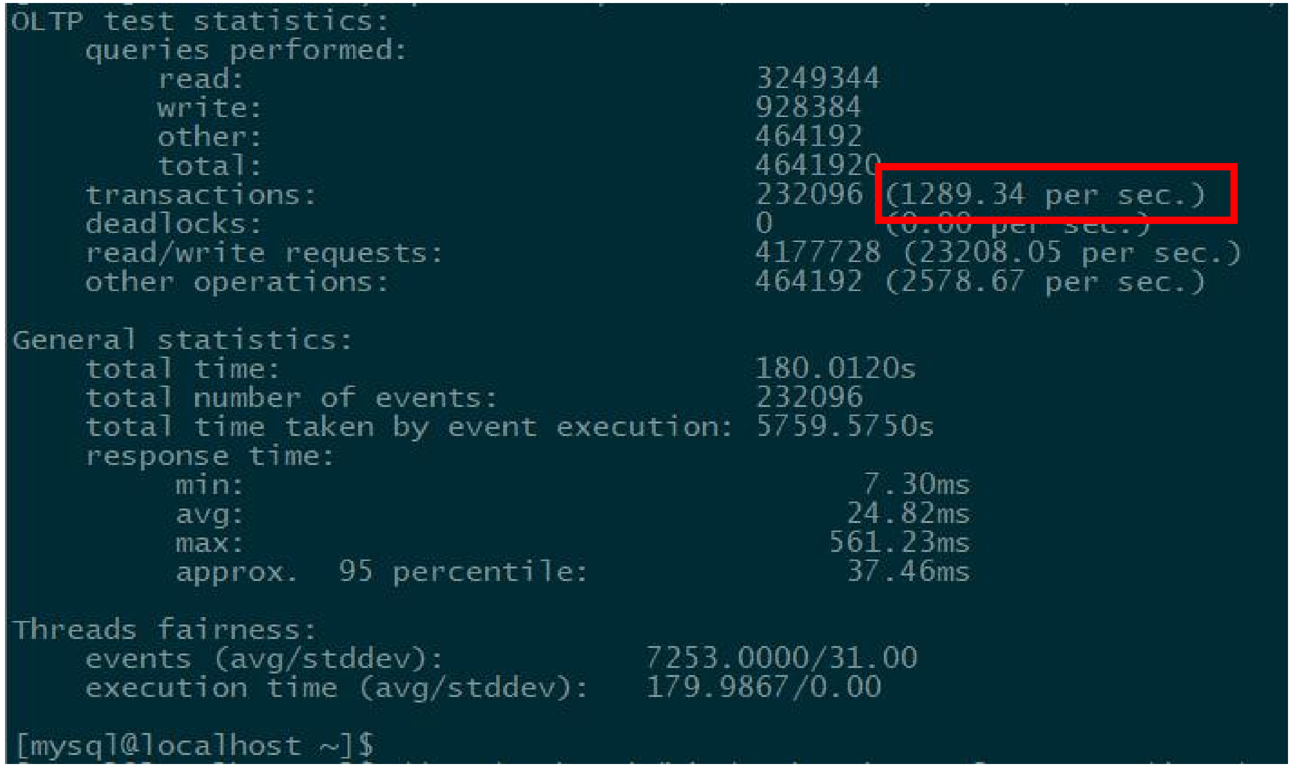


页分裂：部分写，Partial Writes

➢ 性能影响对比，MySQL Double Write性能影响：



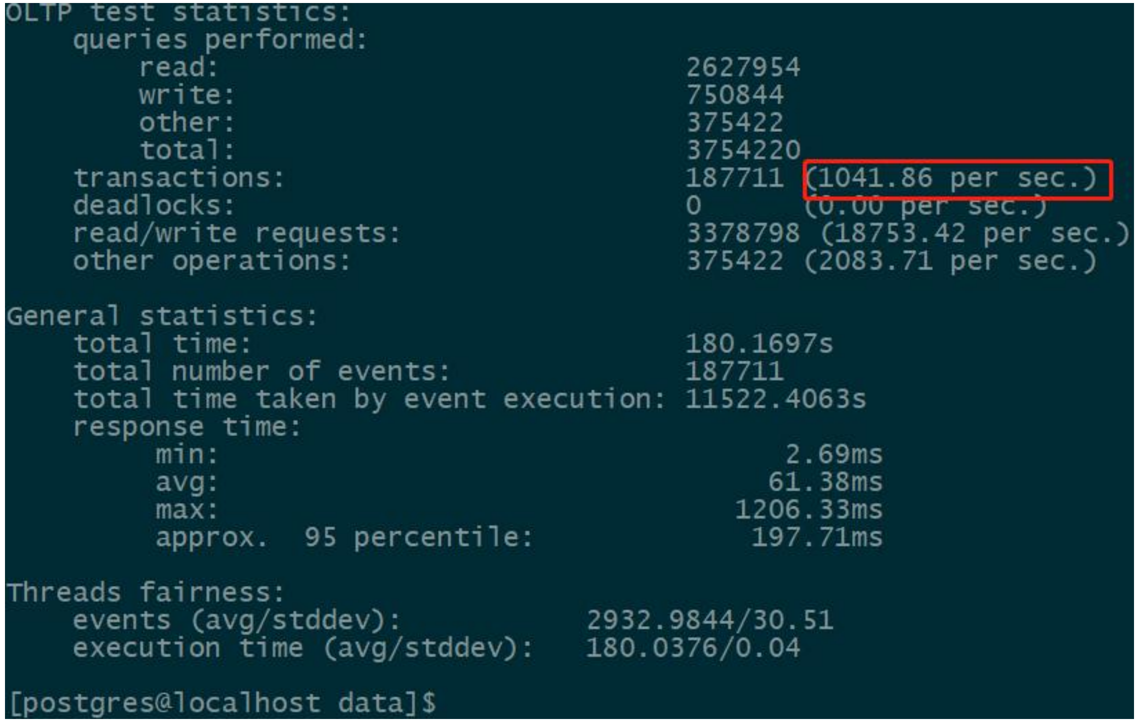
```
OLTP test statistics:
    queries performed:
        read:                            3165806
        write:                           904516
        other:                           452258
        total:                           4522580
    transactions:                        226129 (1254.75 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 4070322 (22585.49 per sec.)
    other operations:                    452258 (2509.50 per sec.)

General statistics:
    total time:                          180.2185s
    total number of events:              226129
    total time taken by event execution: 5762.8595s
    response time:
         min:                                  6.98ms
         avg:                                 25.48ms
         max:                                631.13ms
         approx.  95 percentile:              38.26ms

Threads fairness:
    events (avg/stddev):           7066.5312/25.07
    execution time (avg/stddev):   180.0894/0.07

[mysql@localhost ~]$
```



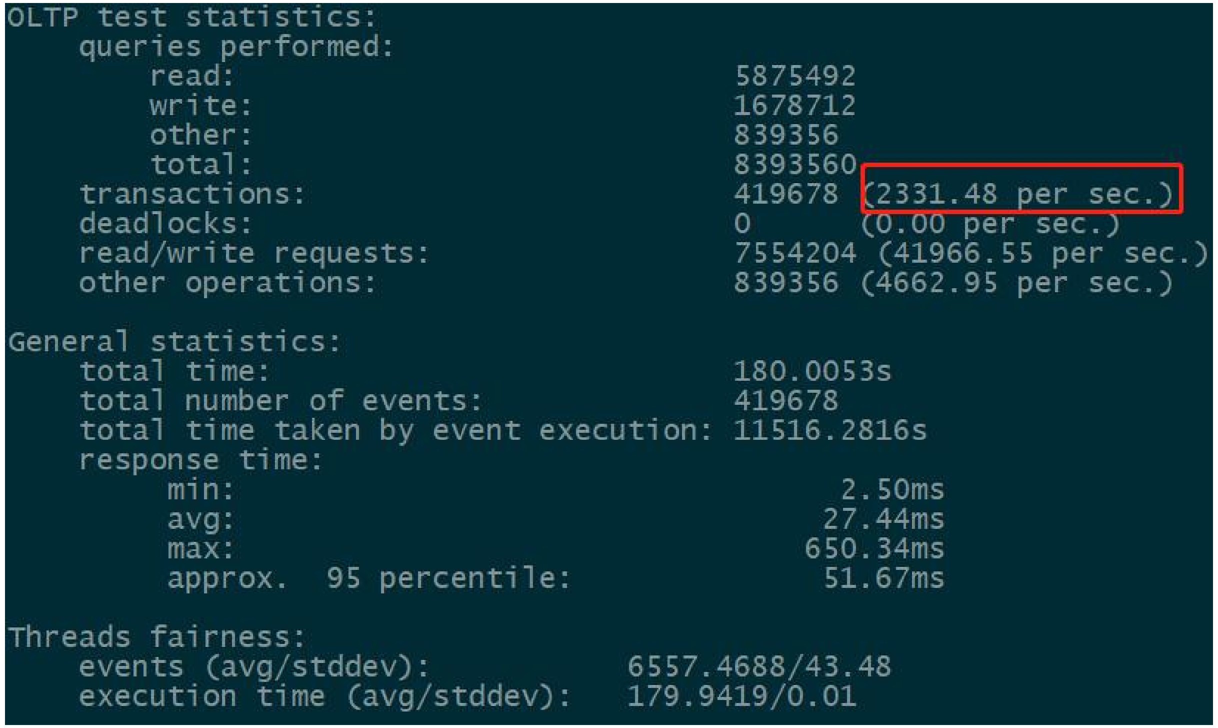
```
OLTP test statistics:
    queries performed:
        read:                            3249344
        write:                           928384
        other:                           464192
        total:                           4641920
    transactions:                        232096 (1289.34 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 4177728 (23208.05 per sec.)
    other operations:                    464192 (2578.67 per sec.)

General statistics:
    total time:                          180.0120s
    total number of events:              232096
    total time taken by event execution: 5759.5750s
    response time:
         min:                                  7.30ms
         avg:                                 24.82ms
         max:                                561.23ms
         approx.  95 percentile:              37.46ms

Threads fairness:
    events (avg/stddev):           7253.0000/31.00
    execution time (avg/stddev):   179.9867/0.00

[mysql@localhost ~]$
```

页分裂：部分写，Partial Writes

- ➢ **性能影响对比，PostgreSQL Full Page Writes性能影响：**



```
OLTP test statistics:
    queries performed:
        read:                            2627954
        write:                           750844
        other:                           375422
        total:                           3754220
    transactions:                        187711 (1041.86 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 3378798 (18753.42 per sec.)
    other operations:                    375422 (2083.71 per sec.)

General statistics:
    total time:                          180.1697s
    total number of events:              187711
    total time taken by event execution: 11522.4063s
    response time:
         min:                                  2.69ms
         avg:                                 61.38ms
         max:                               1206.33ms
         approx.  95 percentile:             197.71ms

Threads fairness:
    events (avg/stddev):           2932.9844/30.51
    execution time (avg/stddev):   180.0376/0.04

[postgres@localhost data]$
```



```
OLTP test statistics:
    queries performed:
        read:                            5875492
        write:                           1678712
        other:                           839356
        total:                           8393560
    transactions:                        419678 (2331.48 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 7554204 (41966.55 per sec.)
    other operations:                    839356 (4662.95 per sec.)

General statistics:
    total time:                          180.0053s
    total number of events:              419678
    total time taken by event execution: 11516.2816s
    response time:
         min:                                  2.50ms
         avg:                                 27.44ms
         max:                                650.34ms
         approx.  95 percentile:              51.67ms

Threads fairness:
    events (avg/stddev):           6557.4688/43.48
    execution time (avg/stddev):   179.9419/0.01
```

页Partial Writes模拟

走　两　步

页Partial Writes模拟

- 拦截系统调用，修改传入OS内核的参数

页Partial Writes模拟

- 拦截系统调用，修改传入OS内核的参数
- Linux，systemtap

页Partial Writes模拟：MySQL篇

- I/O相关系统函数取决于innodb_use_native_aio参数

on：SyS_io_submit

off：pwrite

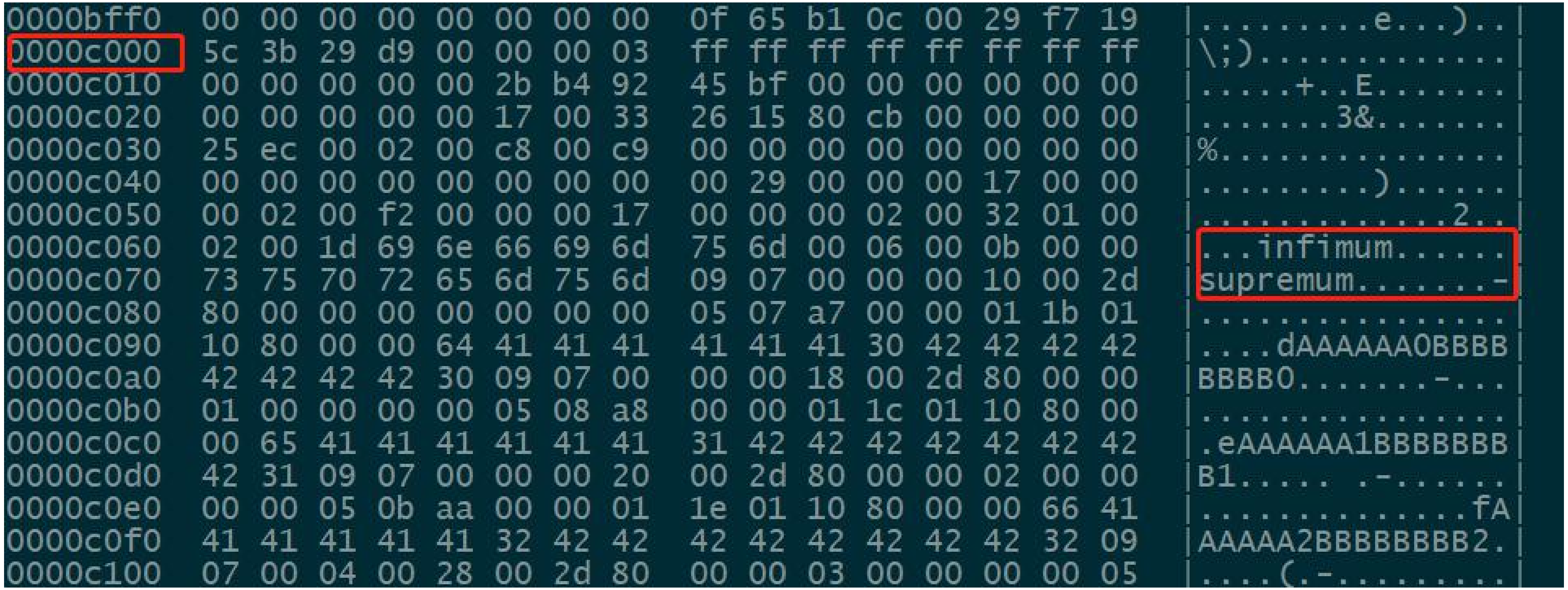
页Partial Writes模拟：MySQL篇

在MySQL中，创建测试表，插入200行数据：

```
CREATE TABLE `vage` (
  `id1` int(11) NOT NULL,
  `id2` int(11) DEFAULT NULL,
  `c1` varchar(100) DEFAULT NULL,
  `c2` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8

delimiter $$
CREATE PROCEDURE vage_init(in nums int)
BEGIN
  declare done int default 0;
  declare i int;
  declare v_id1 int;

  set i=0;
  while i<=nums DO
      insert into vage values(i, i+100, concat('AAAAAA', i), concat('BBBBBBBB', i));
      set i=i+1;
  end while;
  commit;
END$$
delimiter ;

call vage_init(200);
```

页Partial Writes模拟：MySQL篇

MySQL是从上到下使用页中空间，因此数据行要多一些，要保证数据超过4K：



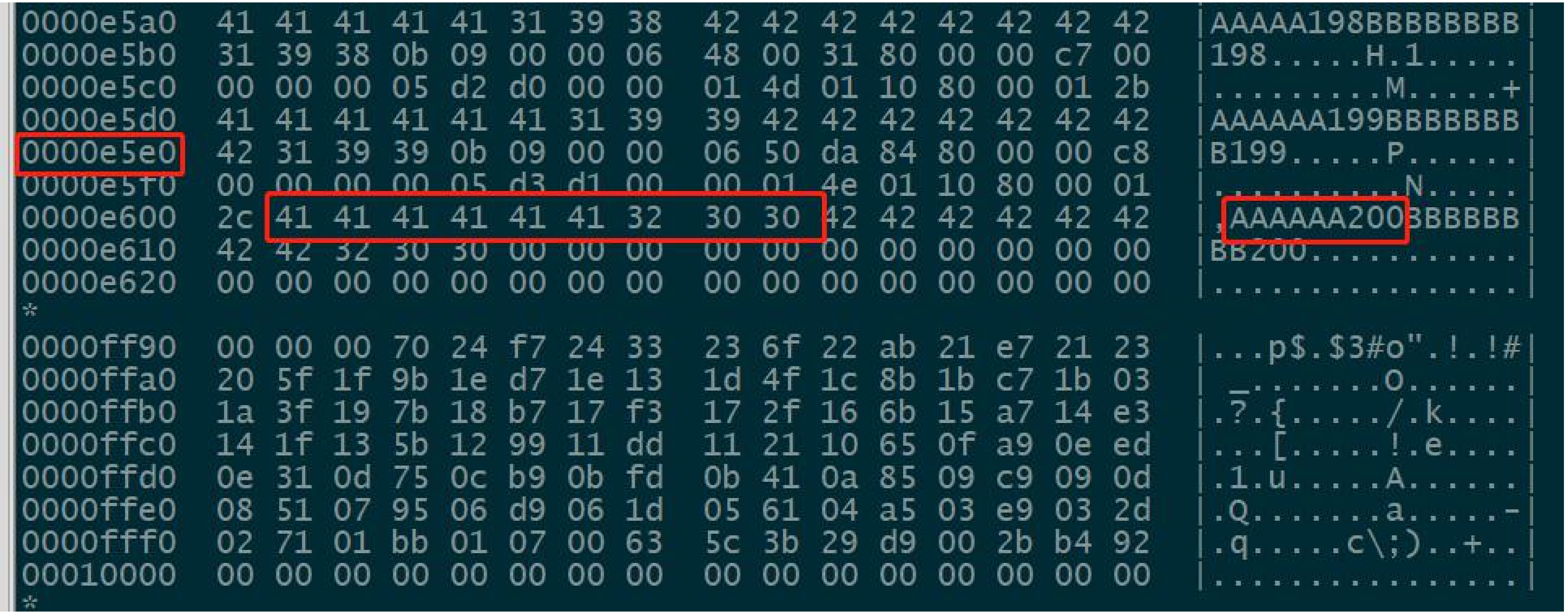
```
0000bff0  00 00 00 00 00 00 00 00  0f 65 b1 0c 00 29 f7 19  |.........e...)..|
0000c000  5c 3b 29 d9 00 00 00 03  ff ff ff ff ff ff ff ff  |\;).............|
0000c010  00 00 00 00 00 2b b4 92  45 bf 00 00 00 00 00 00  |.....+..E.......|
0000c020  00 00 00 00 00 17 00 33  26 15 80 cb 00 00 00 00  |.......3&.......|
0000c030  25 ec 00 02 00 c8 00 c9  00 00 00 00 00 00 00 00  |%...............|
0000c040  00 00 00 00 00 00 00 00  00 29 00 00 00 17 00 00  |.........)......|
0000c050  00 02 00 f2 00 00 00 17  00 00 00 02 00 32 01 00  |.............2..|
0000c060  02 00 1d 69 6e 66 69 6d  75 6d 00 06 00 0b 00 00  |...infimum......|
0000c070  73 75 70 72 65 6d 75 6d  09 07 00 00 00 10 00 2d  |supremum.......-|
0000c080  80 00 00 00 00 00 00 00  05 07 a7 00 00 01 1b 01  |................|
0000c090  10 80 00 00 64 41 41 41  41 41 41 30 42 42 42 42  |....dAAAAAA0BBBB|
0000c0a0  42 42 42 42 30 09 07 00  00 00 18 00 2d 80 00 00  |BBBB0........-..|
0000c0b0  01 00 00 00 00 05 08 a8  00 00 01 1c 01 10 80 00  |................|
0000c0c0  00 65 41 41 41 41 41 41  31 42 42 42 42 42 42 42  |.eAAAAAA1BBBBBBB|
0000c0d0  42 31 09 07 00 00 00 20  00 2d 80 00 00 02 00 00  |B1..... .-......|
0000c0e0  00 00 05 0b aa 00 00 01  1e 01 10 80 00 00 66 41  |..............fA|
0000c0f0  41 41 41 41 41 32 42 42  42 42 42 42 42 42 32 09  |AAAAA2BBBBBBBB2.|
0000c100  07 00 04 00 28 00 2d 80  00 00 03 00 00 00 00 05  |....(.-.........|
```

使用“dd if=vage.ibd bs=16384 | hexdump –C |more”观察表对应文件，从0xc000开始，是表数据对应的页（之前数据是表头）。

两百行数据，目前占据一个MySQL页，即从0xC000到0x10000。

页Partial Writes模拟：MySQL篇

MySQL是从上到下使用页中空间，因此数据行要多一些，要保证数据超过4K：

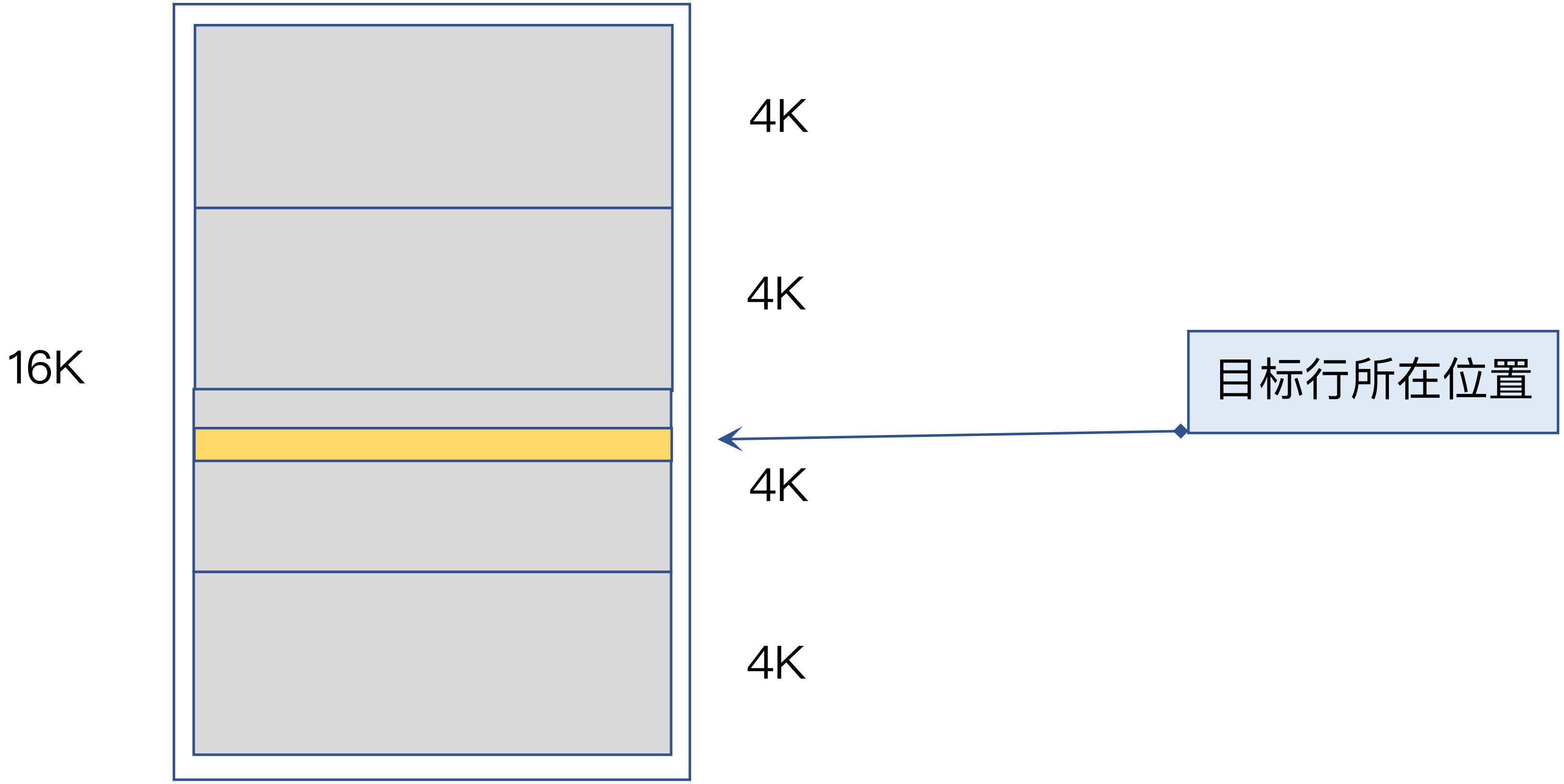


```
0000e5a0  41 41 41 41 41 31 39 38  42 42 42 42 42 42 42 42  |AAAAA198BBBBBBBB|
0000e5b0  31 39 38 0b 09 00 00 06  48 00 31 80 00 00 c7 00  |198.....H.1.....|
0000e5c0  00 00 00 05 d2 d0 00 00  01 4d 01 10 80 00 01 2b  |.........M.....+|
0000e5d0  41 41 41 41 41 41 31 39  39 42 42 42 42 42 42 42  |AAAAAA199BBBBBBB|
0000e5e0  42 31 39 39 0b 09 00 00  06 50 da 84 80 00 00 c8  |B199.....P......|
0000e5f0  00 00 00 00 05 d3 d1 00  00 01 4e 01 10 80 00 01  |..........N.....|
0000e600  2c 41 41 41 41 41 41 32  30 30 42 42 42 42 42 42  |,AAAAAA200BBBBBB|
0000e610  42 42 32 30 30 00 00 00  00 00 00 00 00 00 00 00  |BB200...........|
0000e620  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0000ff90  00 00 00 70 24 f7 24 33  23 6f 22 ab 21 e7 21 23  |...p$.$3#o".!.!#|
0000ffa0  20 5f 1f 9b 1e d7 1e 13  1d 4f 1c 8b 1b c7 1b 03  | _.......O......|
0000ffb0  1a 3f 19 7b 18 b7 17 f3  17 2f 16 6b 15 a7 14 e3  |.?.{...../.k....|
0000ffc0  14 1f 13 5b 12 99 11 dd  11 21 10 65 0f a9 0e ed  |...[.....!.e....|
0000ffd0  0e 31 0d 75 0c b9 0b fd  0b 41 0a 85 09 c9 09 0d  |.1.u.....A......|
0000ffe0  08 51 07 95 06 d9 06 1d  05 61 04 a5 03 e9 03 2d  |.Q.......a.....-|
0000fff0  02 71 01 bb 01 07 00 63  5c 3b 29 d9 00 2b b4 92  |.q.....c\;)..+..|
00010000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

最后一行，开始自0xe5e0这一行。我们以最后一行的c1列为目标，它在0xe601处。

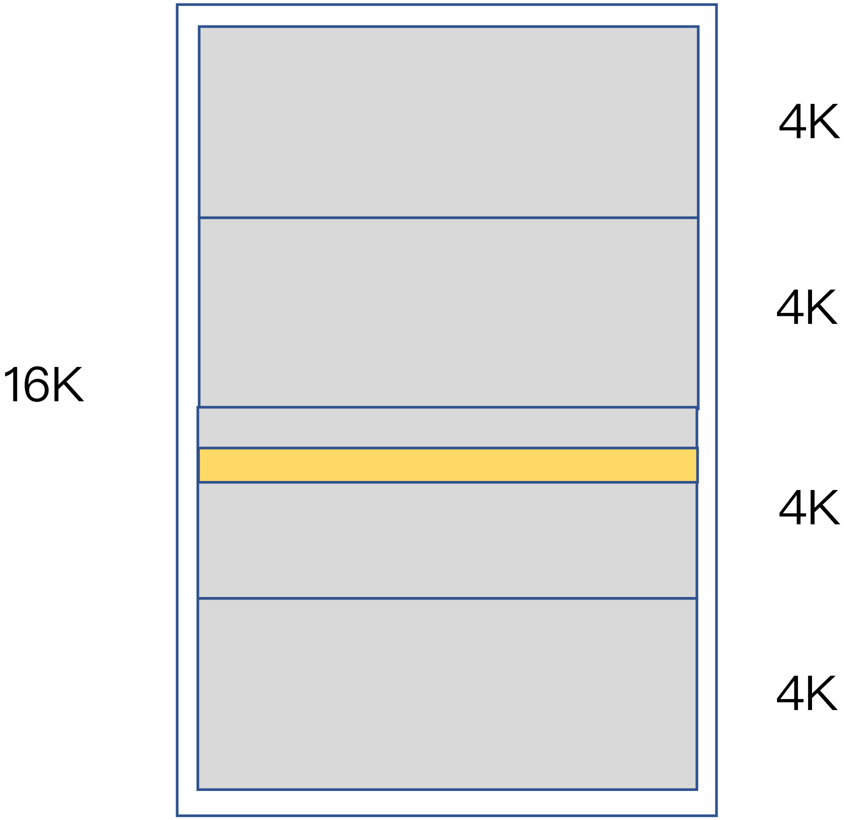
页Partial Writes模拟：MySQL篇

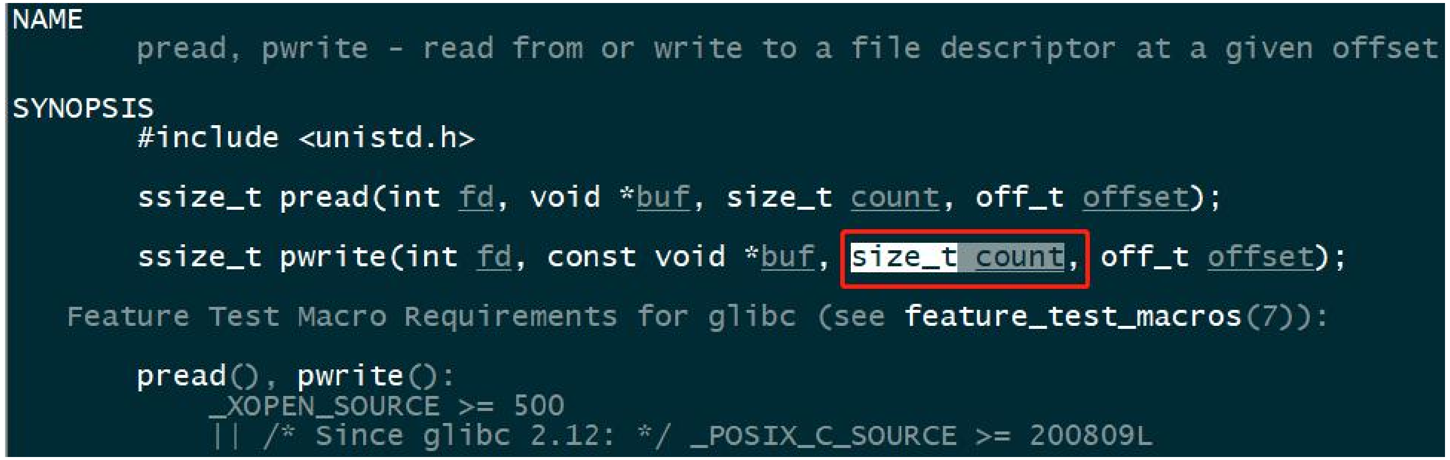
MySQL是从上到下使用页中空间，因此数据行要多一些，要保证数据超过4K：



、

页Partial Writes模拟：MySQL篇

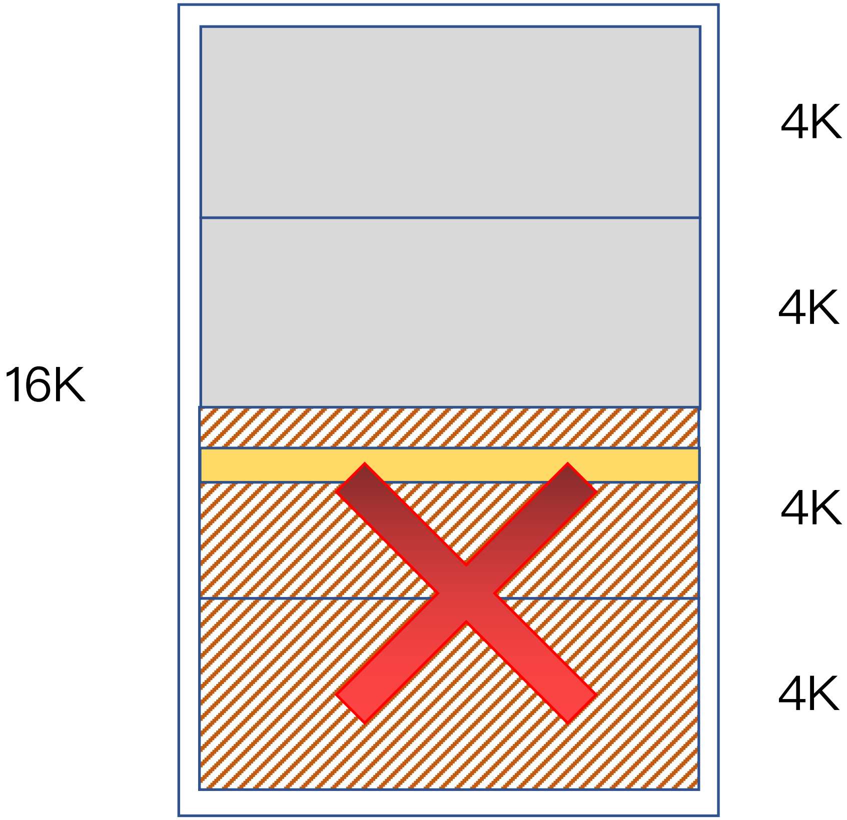


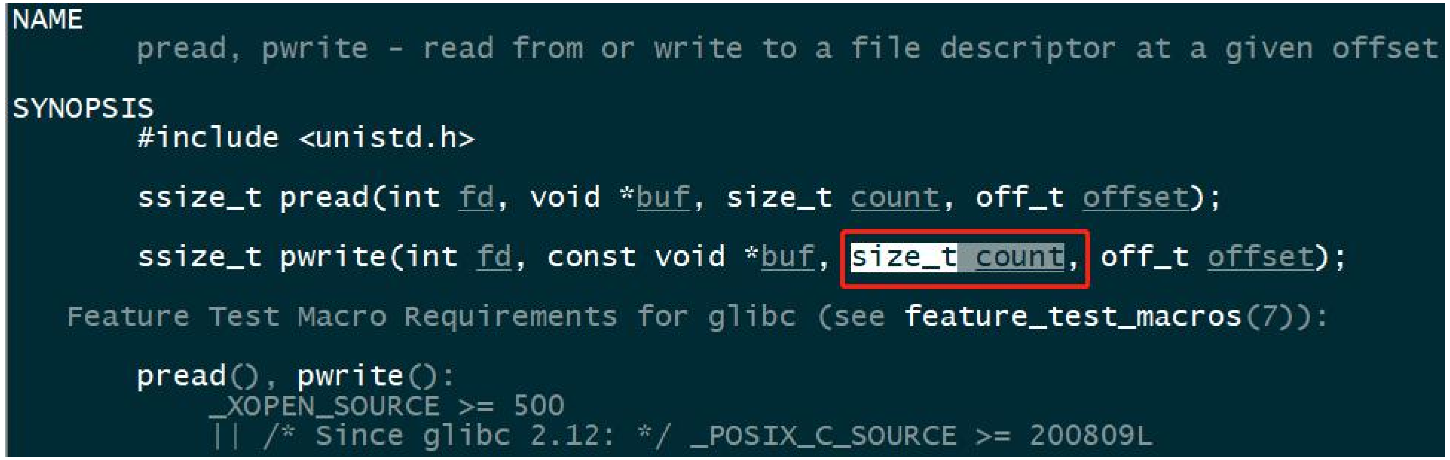


```
NAME
       pread, pwrite - read from or write to a file descriptor at a given offset

SYNOPSIS
       #include <unistd.h>

       ssize_t pread(int fd, void *buf, size_t count, off_t offset);

       ssize_t pwrite(int fd, const void *buf, size_t count, off_t offset);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       pread(), pwrite():
           _XOPEN_SOURCE >= 500
           || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
```

发出修改目标行的SQL，触发对目标页的写操作，拦截OS I/O系统调用，修改count参数。

页Partial Writes模拟：MySQL篇





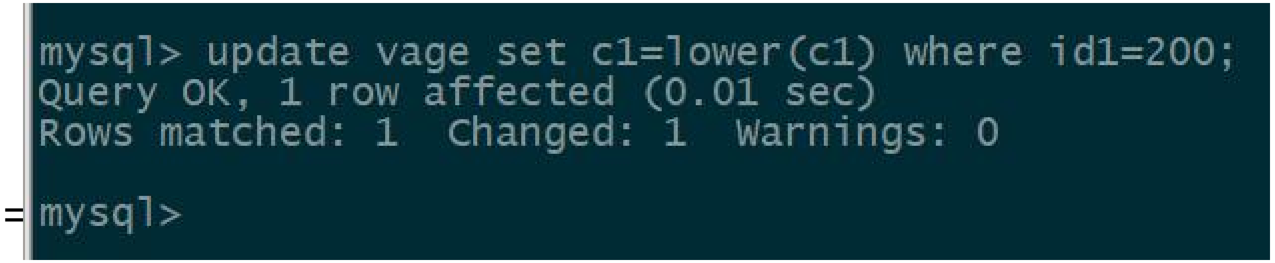
```
NAME
       pread, pwrite - read from or write to a file descriptor at a given offset

SYNOPSIS
       #include <unistd.h>

       ssize_t pread(int fd, void *buf, size_t count, off_t offset);

       ssize_t pwrite(int fd, const void *buf, size_t count, off_t offset);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       pread(), pwrite():
           _XOPEN_SOURCE >= 500
           || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
```

count参数原来是16384，将其改为8192。造成一个页，只写了一半的效果，也就是页的Partial Writes。
同时还要返回I/O错误，让MySQL知道，I/O并没有成功完成。因为突然的OS宕机，也是同样的情况：

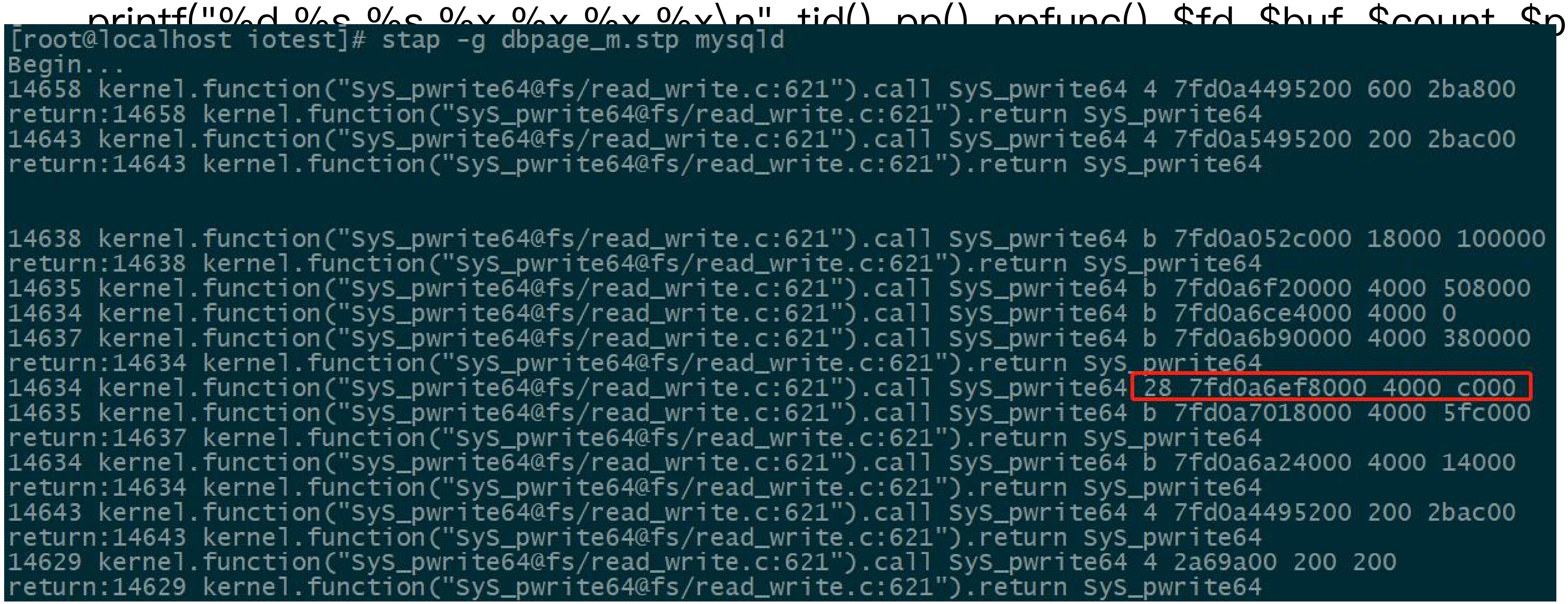
- 只完成部分I/O
- 最终I/O错误

页Partial Writes模拟：MySQL篇

```
……
probe syscall.pwrite
{
        if (execname() =
        {
                printf("%d %s %s %x %x %x %x\n", tid(), pp(), ppfunc(), $fd, $buf, $count, $pos)
        }
}
……
```



```
mysql> update vage set c1=lower(c1) where id1=200;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```



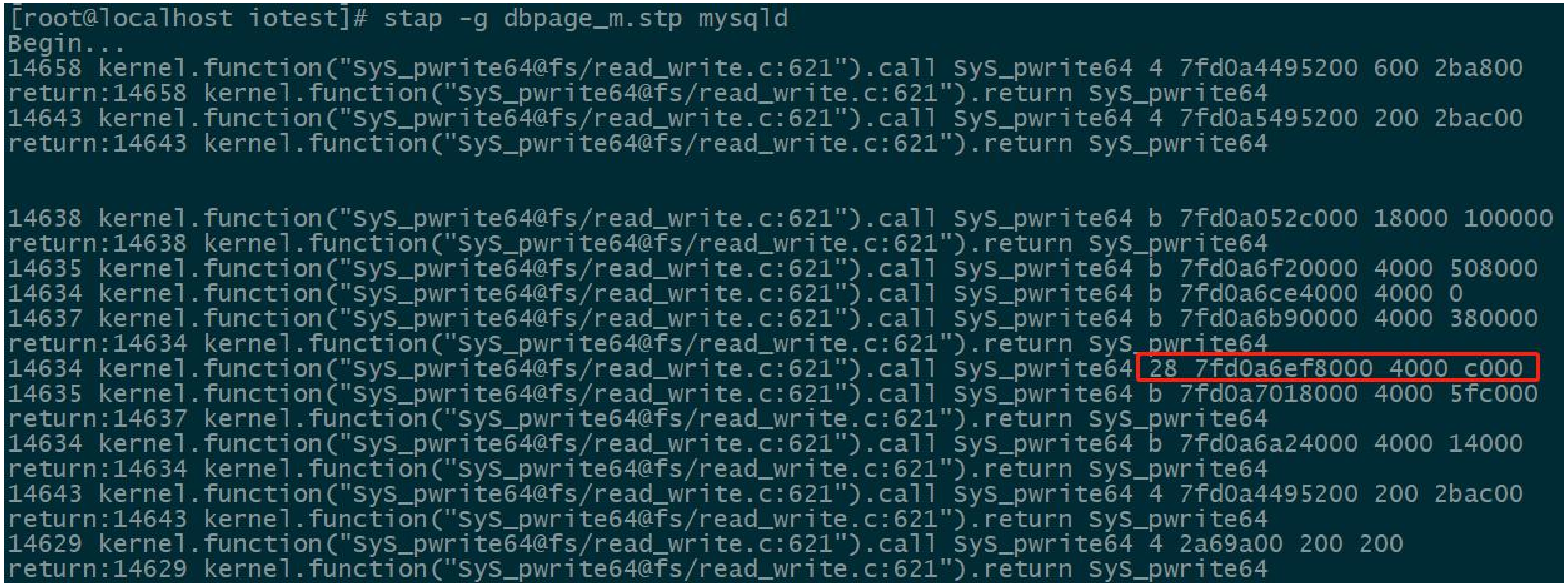
```
[root@localhost iotest]# stap -g dbpage_m.stp mysqld
Begin...
14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 600 2ba800
return:14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a5495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a052c000 18000 100000
return:14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6f20000 4000 508000
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6ce4000 4000 0
14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6b90000 4000 380000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 28 7fd0a6ef8000 4000 c000
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a7018000 4000 5fc000
return:14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6a24000 4000 14000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 2a69a00 200 200
return:14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

页Partial Writes模拟：MySQL篇

……

printf("%d %s %s %x %x %x %x\n", tid(), pp(), ppfunc(), $fd, $buf, $count, $pos)

……



```
[root@localhost iotest]# stap -g dbpage_m.stp mysqld
Begin...
14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 600 2ba800
return:14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a5495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a052c000 18000 100000
return:14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6f20000 4000 508000
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6ce4000 4000 0
14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6b90000 4000 380000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 28 7fd0a6ef8000 4000 c000
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a7018000 4000 5fc000
return:14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6a24000 4000 14000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 2a69a00 200 200
return:14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

目标表的更新在中间，fd是0x28（十进制为40），它是表对应的文件：vage.ibd。
可以在/proc/(mysqld进程号)/fd目录中确认此点。

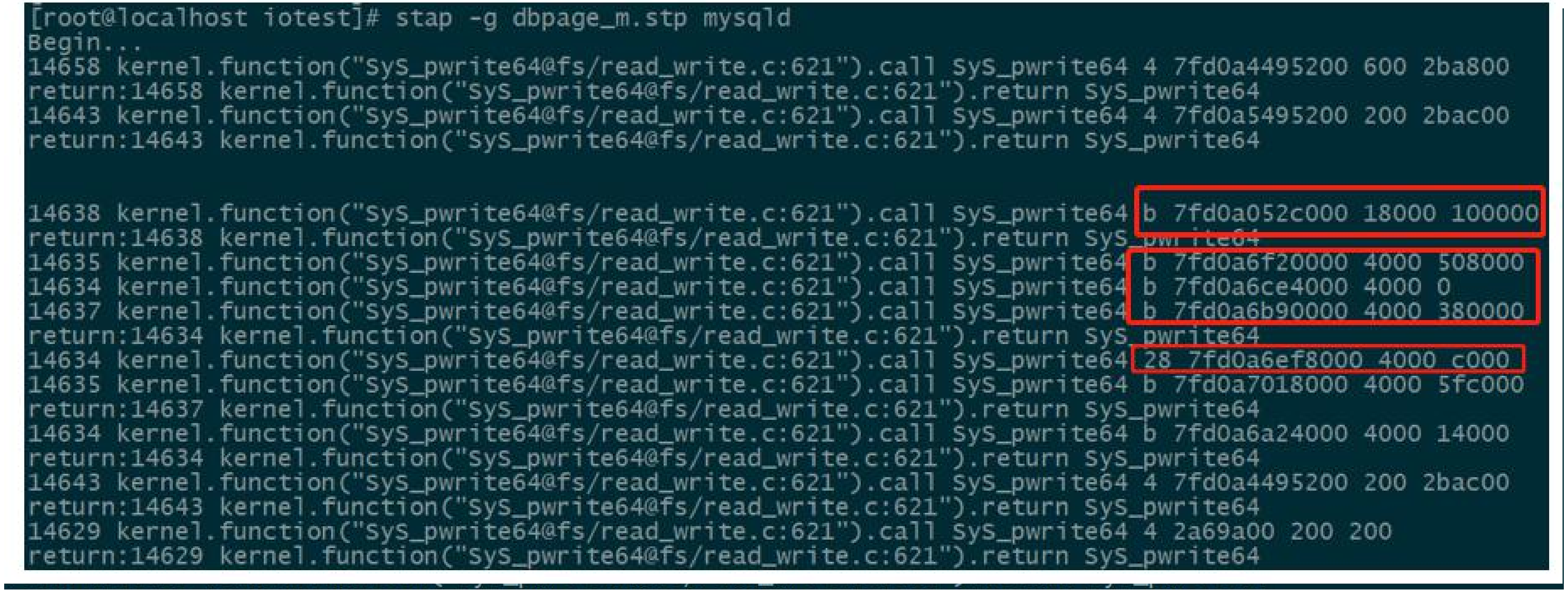
count参数为0x4000（16384），pos参数为0xc000。

页Partial Writes模拟：MySQL篇

……

```
printf("%d %s %s %x %x %x %x\n", tid(), pp(), ppfunc(), $fd, $buf, $count, $pos)
```

……



```
[root@localhost iotest]# stap -g dbpage_m.stp mysqld
Begin...
14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 600 2ba800
return:14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a5495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a052c000 18000 100000
return:14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6f20000 4000 508000
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6ce4000 4000 0
14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6b90000 4000 380000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 28 7fd0a6ef8000 4000 c000
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a7018000 4000 5fc000
return:14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6a24000 4000 14000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 2a69a00 200 200
return:14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

对0xb号文件（系统表对应文件：ibdata1）有4次写操作。其中第一次写0x18000（6个页）字节的，是写Double Write Buffer。可以使用dd 命令，读出偏移0x100,000的数据进行确认。

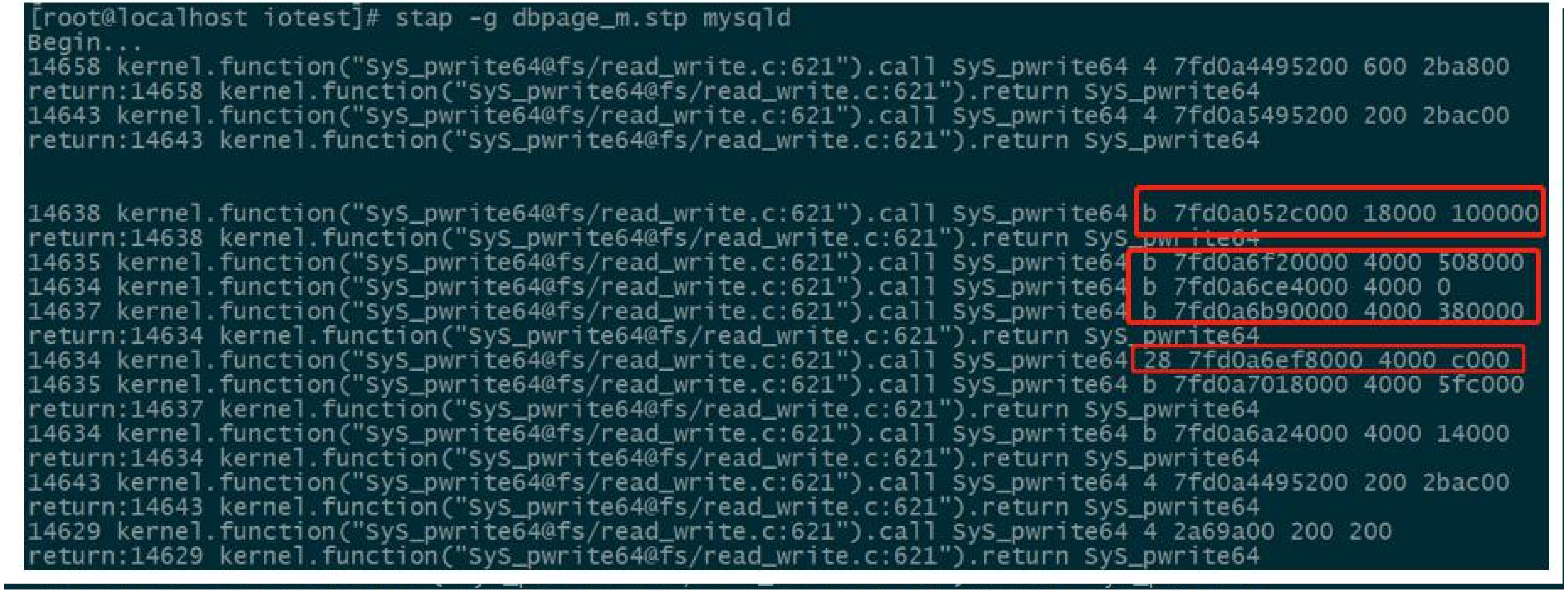
后面几个页，是UNDO和其他内容。

页Partial Writes模拟：MySQL篇

……

printf("%d %s %s %x %x %x %x\n", tid(), pp(), ppfunc(), $fd, $buf, $count, $pos)

……



```
[root@localhost iotest]# stap -g dbpage_m.stp mysqld
Begin...
14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 600 2ba800
return:14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a5495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a052c000 18000 100000
return:14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6f20000 4000 508000
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6ce4000 4000 0
14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6b90000 4000 380000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 28 7fd0a6ef8000 4000 c000
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a7018000 4000 5fc000
return:14637 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6a24000 4000 14000
return:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 200 2bac00
return:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 2a69a00 200 200
return:14629 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

双写相关I/O，仍然保证成功。我们的目标，拦截系统调用，修改参数，使表/索引页的I/O出现Partial Writes。

方法：修改count参数，造成部分写入成功。修改fd参数，使写16K的I/O最终报错。

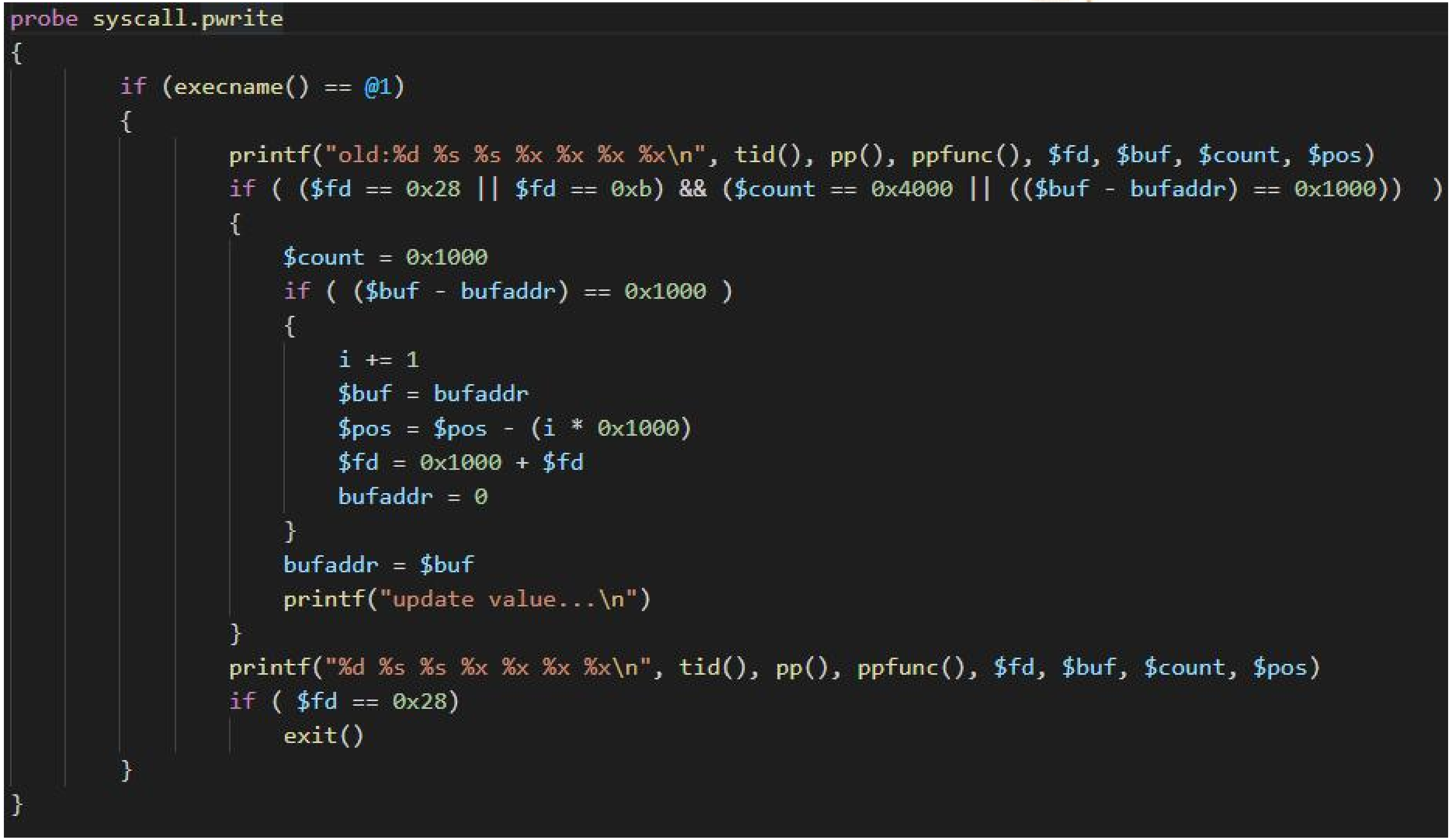
页Partial Writes模拟：MySQL篇

➢ 最终的脚本：

修改$count为0x1000，成功写为4K。

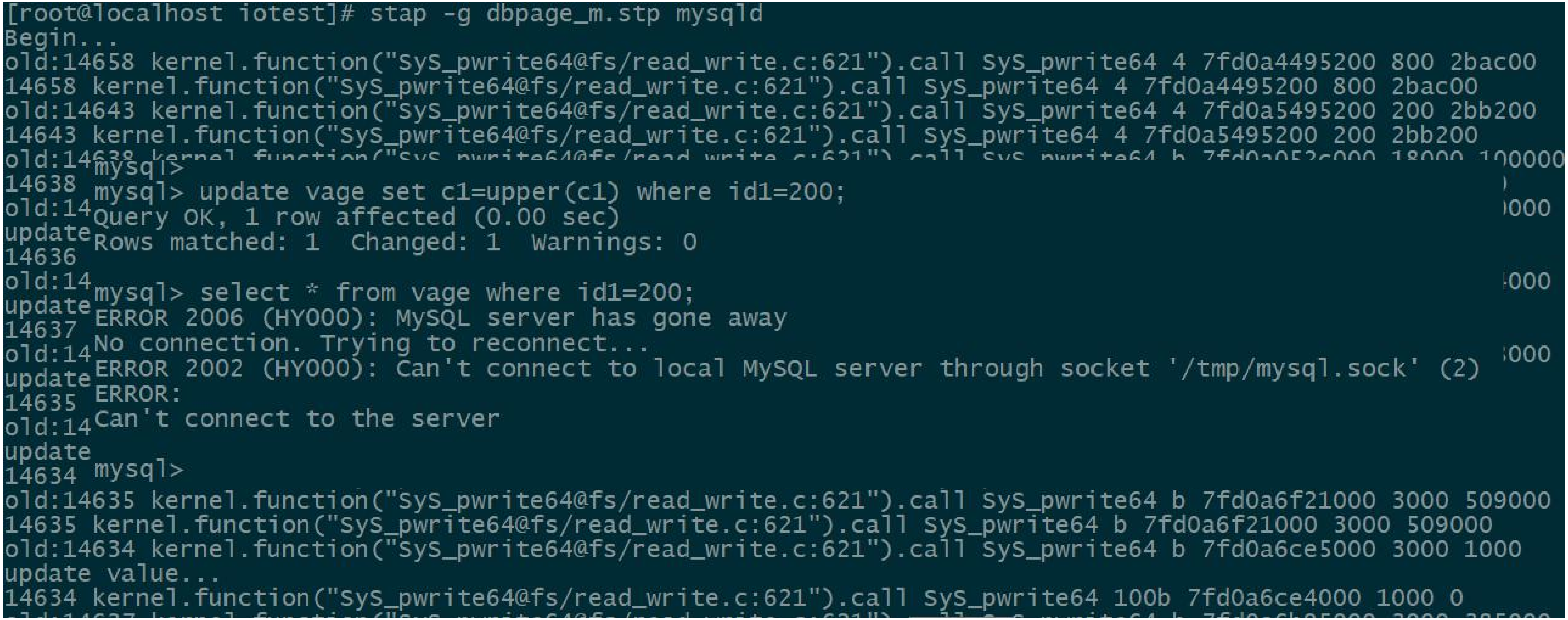
后面的脚本，是为了造成I/O错误。

前面的条件，是跳过双写相关的I/O，保证双写可以成功完成。



```
probe syscall.pwrite
{
        if (execname() == @1)
        {
                printf("old:%d %s %s %x %x %x %x\n", tid(), pp(), ppfunc(), $fd, $buf, $count, $pos)
                if ( ($fd == 0x28 || $fd == 0xb) && ($count == 0x4000 || (($buf - bufaddr) == 0x1000))  )
                {
                    $count = 0x1000
                    if ( ($buf - bufaddr) == 0x1000 )
                    {
                        i += 1
                        $buf = bufaddr
                        $pos = $pos - (i * 0x1000)
                        $fd = 0x1000 + $fd
                        bufaddr = 0
                    }
                    bufaddr = $buf
                    printf("update value...\n")
                }
                printf("%d %s %s %x %x %x %x\n", tid(), pp(), ppfunc(), $fd, $buf, $count, $pos)
                if ( $fd == 0x28)
                    exit()
        }
}
```

页Partial Writes模拟：MySQL篇



```
[root@localhost iotest]# stap -g dbpage_m.stp mysqld
Begin...
old:14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 800 2bac00
14658 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a4495200 800 2bac00
old:14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a5495200 200 2bb200
14643 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 7fd0a5495200 200 2bb200
old:14638 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a052c000 18000 100000
14638                                                                                                   )
old:14                                                                                                  )000
update
14636
old:14                                                                                                  000
update
14637
old:14                                                                                                  000
update
14635
old:14
update
14634
old:14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6f21000 3000 509000
14635 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6f21000 3000 509000
old:14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 b 7fd0a6ce5000 3000 1000
update value...
14634 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 100b 7fd0a6ce4000 1000 0
```

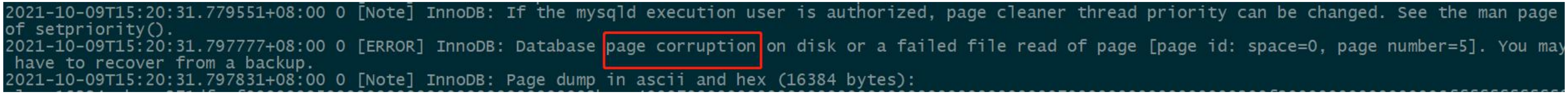
```
mysql>
mysql> update vage set c1=upper(c1) where id1=200;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from vage where id1=200;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)
ERROR:
Can't connect to the server

mysql>
```

页Partial Writes模拟：MySQL篇

脚本还存在一些问题，并不能使所有表/索引页的修改都是Partial Writes。

页Partial Writes模拟：MySQL篇

启动MySQL，可以看到页损坏报错，启动失败：



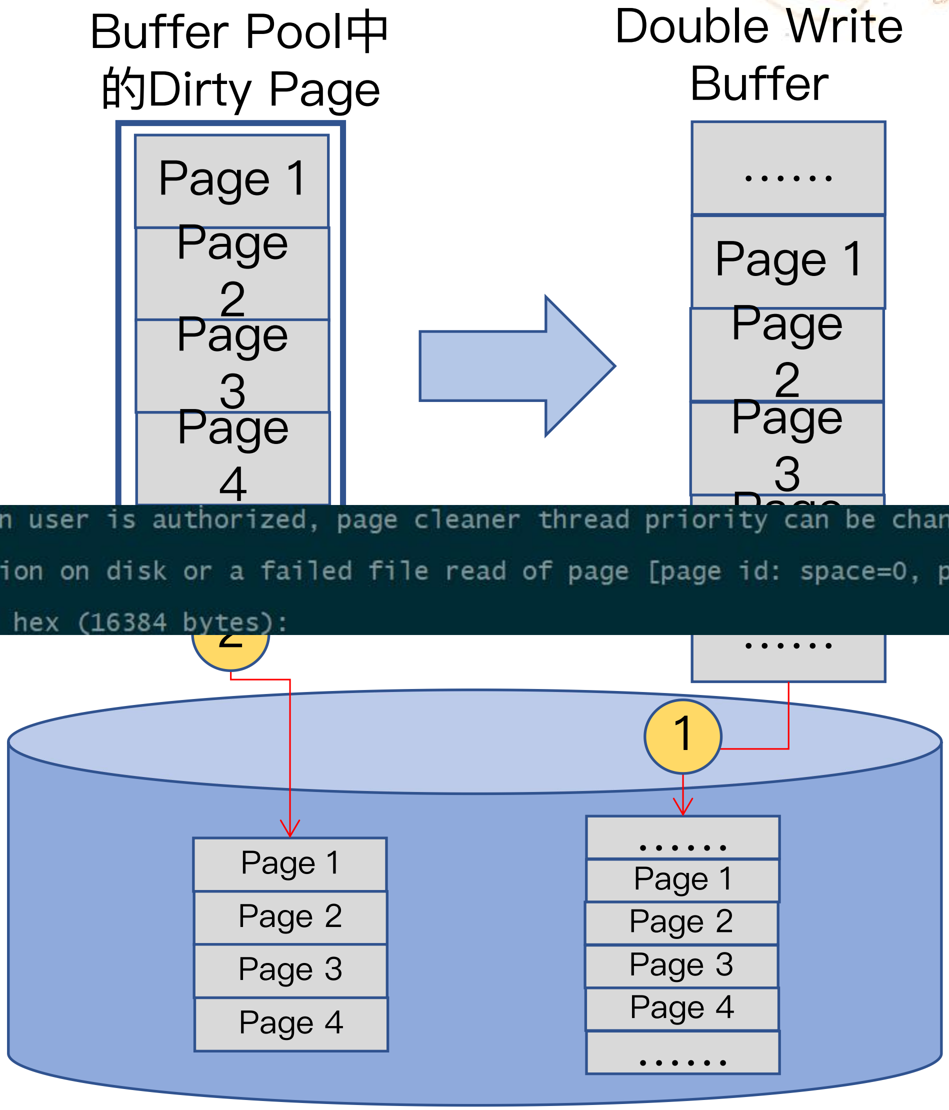
```
2021-10-09T15:20:31.779551+08:00 0 [Note] InnoDB: If the mysqld execution user is authorized, page cleaner thread priority can be changed. See the man page
of setpriority().
2021-10-09T15:20:31.797777+08:00 0 [ERROR] InnoDB: Database page corruption on disk or a failed file read of page [page id: space=0, page number=5]. You may
 have to recover from a backup.
2021-10-09T15:20:31.797831+08:00 0 [Note] InnoDB: Page dump in ascii and hex (16384 bytes):
```

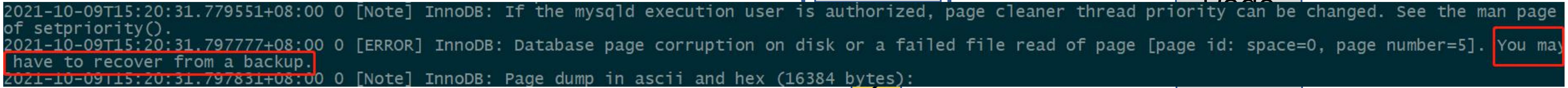
页Partial Writes模拟：MySQL篇总结

Double Write并没有像想像的那样，发挥它应有的作用。

备份很重要。双写只在部分情况下有效。

解决Partial Writes，终极方案，如myerror.log中所说：





```
2021-10-09T15:20:31.779551+08:00 0 [Note] InnoDB: If the mysqld execution user is authorized, page cleaner thread priority can be changed. See the man page
of setpriority().
2021-10-09T15:20:31.797777+08:00 0 [ERROR] InnoDB: Database page corruption on disk or a failed file read of page [page id: space=0, page number=5]. You may
 have to recover from a backup.
2021-10-09T15:20:31.797831+08:00 0 [Note] InnoDB: Page dump in ascii and hex (16384 bytes):
```

页Partial Writes模拟：Oracle篇

- I/O相关系统函数（和MySQL一样）

打开异步I/O：SyS_io_submit

未打开异步I/O：pwrite

页Partial Writes模拟：Oracle篇

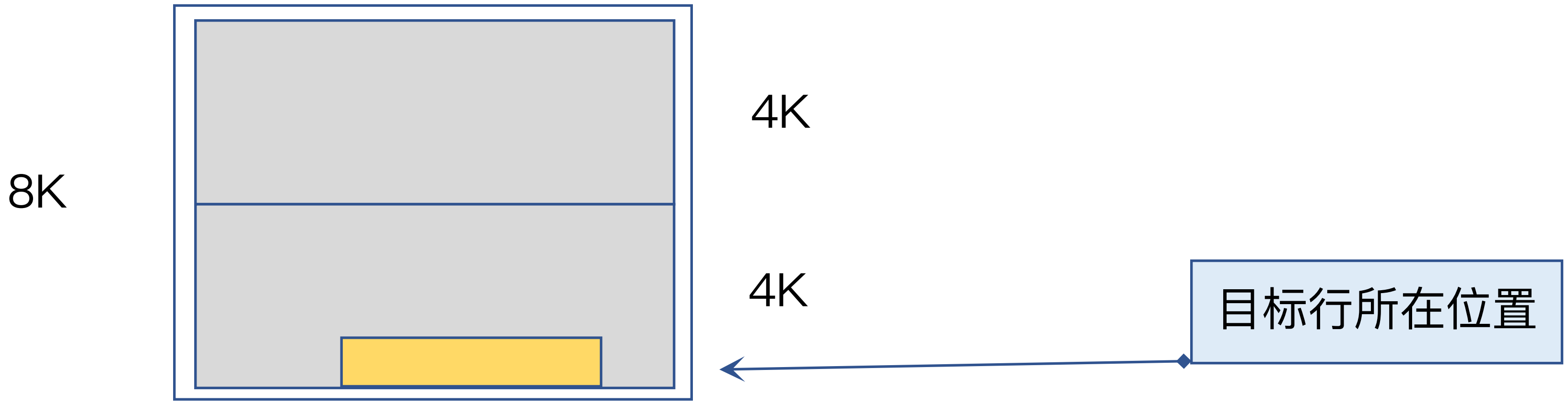
- 准备测试数据

```
create tablespace TPS_TEST datafile  '/u01/oradata/PROD/tps_test_01' size 128k;

create table u2.dtcc(id1 int primary key, c1 varchar2(30)) tablespace tps_test;

insert into u2.dtcc values(1, 'AAAAAAAA');

commit;
```

Oracle 是从下往上使用页空间，虽然只插入一行，但目标行在页的尾部。

页Partial Writes模拟：Oracle篇

- 准备测试数据

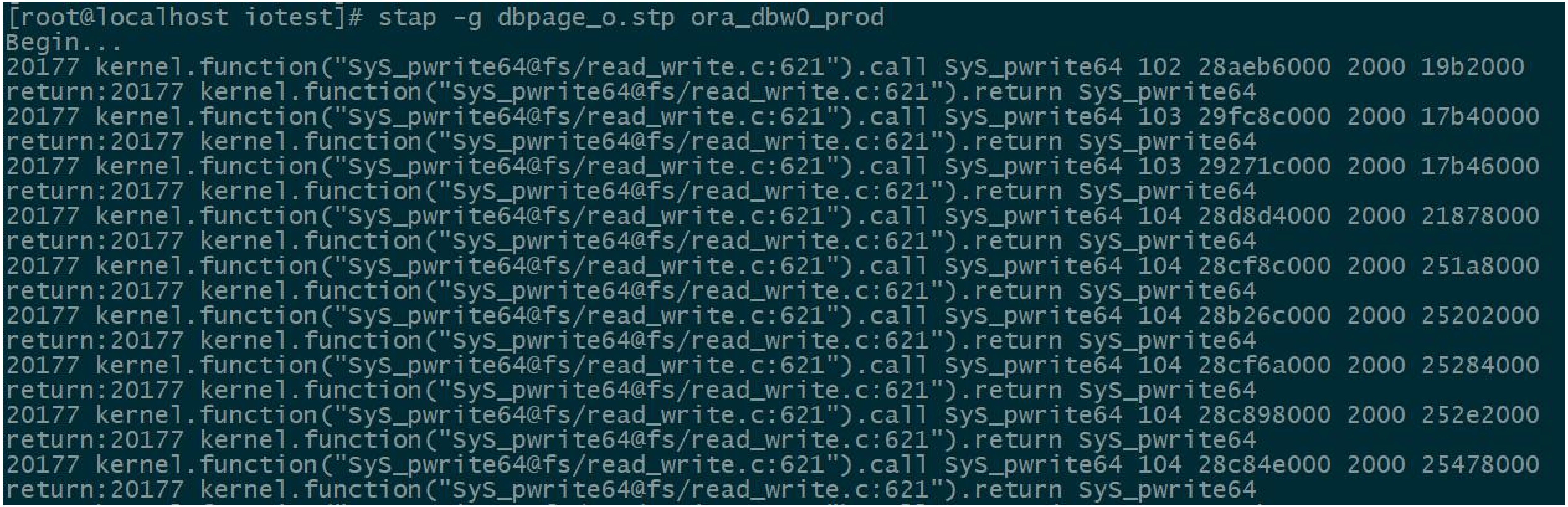


- 测试步骤

- 修改目标行，使目标块成为脏页。Oracle会修改块头部的SCN等信息，和块尾部的目标行。
- 手动发出检查点，触发DBWR进程写脏块。
- 拦截I/O系统调用，修改count参数，造成一个8K块，前4K写成功、后4K写失败。
- 观察、记录Oracle的处理过程。

页Partial Writes模拟：Oracle篇

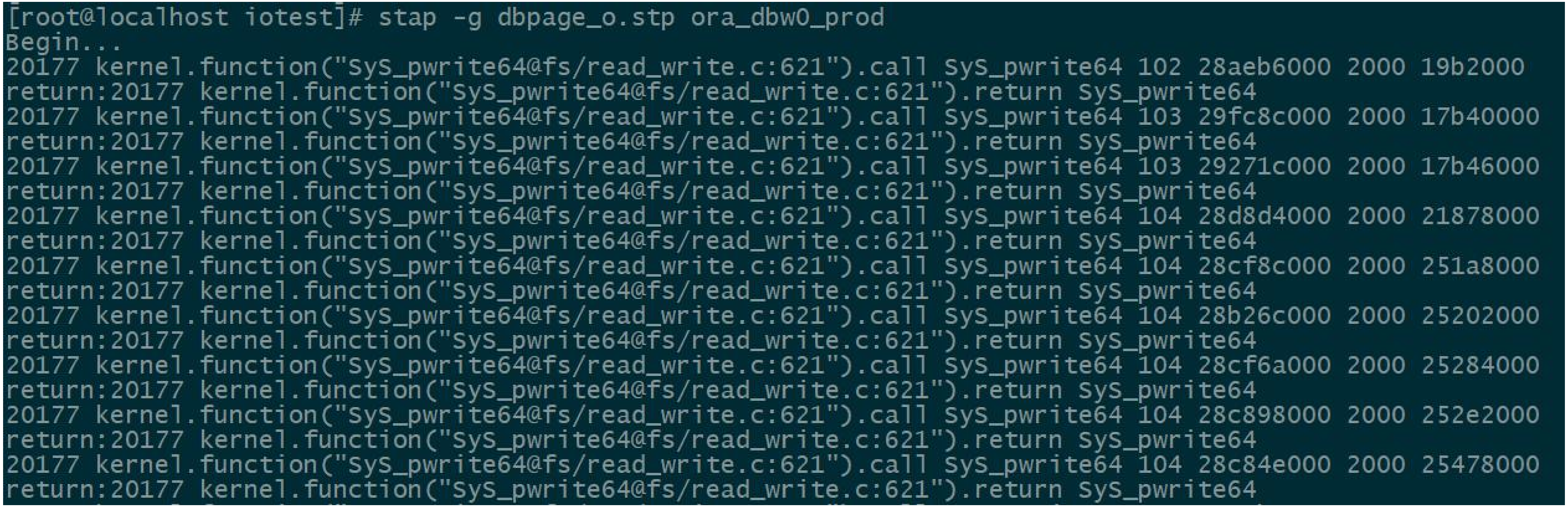
- 首先观察I/O:



```
[root@localhost iotest]# stap -g dbpage_o.stp ora_dbw0_prod
Begin...
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 102 28aeb6000 2000 19b2000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 103 29fc8c000 2000 17b40000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 103 29271c000 2000 17b46000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28d8d4000 2000 21878000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28cf8c000 2000 251a8000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28b26c000 2000 25202000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28cf6a000 2000 25284000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28c898000 2000 252e2000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28c84e000 2000 25478000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

页Partial Writes模拟：Oracle篇

- 首先观察I/O：

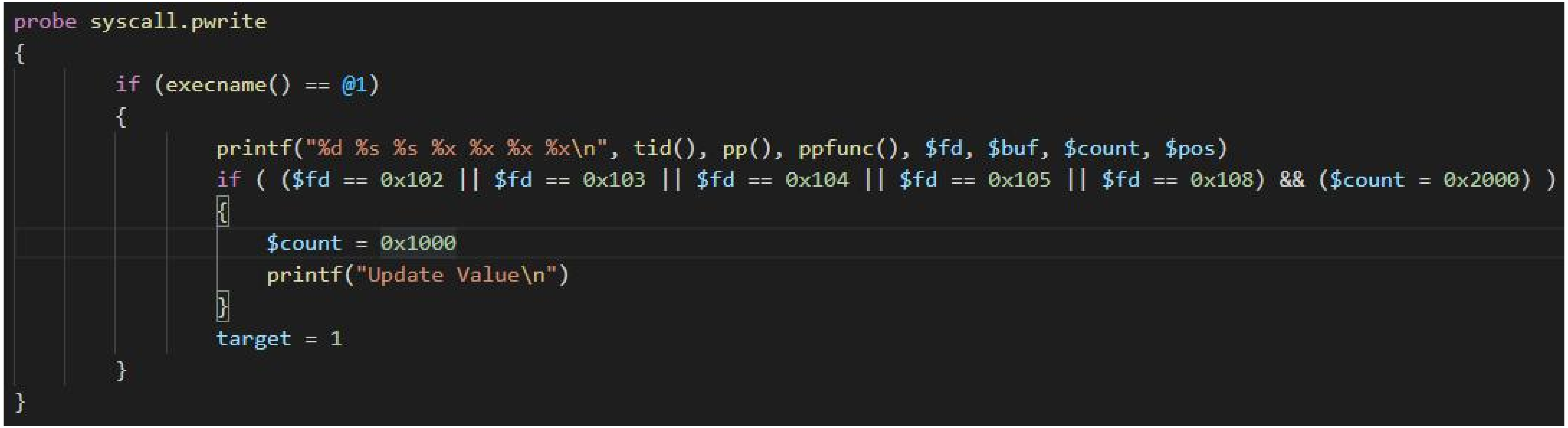


```
[root@localhost iotest]# stap -g dbpage_o.stp ora_dbw0_prod
Begin...
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 102 28aeb6000 2000 19b2000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 103 29fc8c000 2000 17b40000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 103 29271c000 2000 17b46000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28d8d4000 2000 21878000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28cf8c000 2000 251a8000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28b26c000 2000 25202000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28cf6a000 2000 25284000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28c898000 2000 252e2000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 104 28c84e000 2000 25478000
return:20177 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

0x102是system01.dbf，0x103是undotbs01.dbf，0x104是sysaux01.dbf。后面还有0x105，undotbs01_02.dbf文件。0x108，是目标文件tps_test_01。

页Partial Writes模拟：Oracle篇

- 开始测试

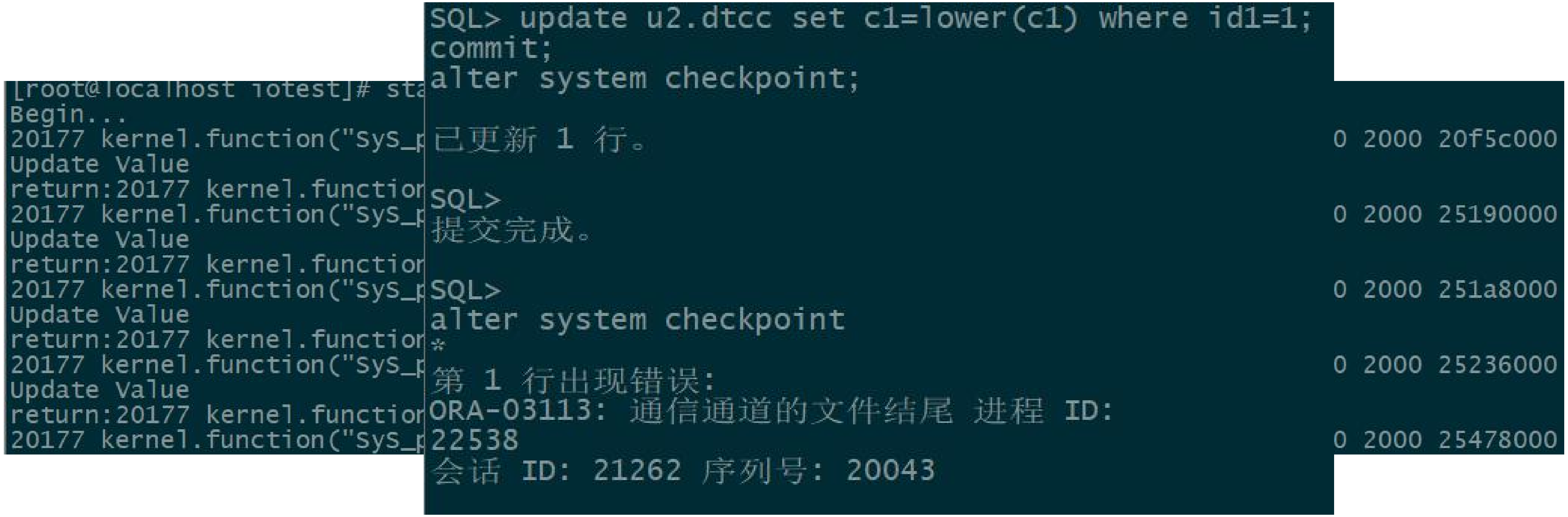


```
probe syscall.pwrite
{
        if (execname() == @1)
        {
                printf("%d %s %s %x %x %x %x\n", tid(), pp(), ppfunc(), $fd, $buf, $count, $pos)
                if ( ($fd == 0x102 || $fd == 0x103 || $fd == 0x104 || $fd == 0x105 || $fd == 0x108) && ($count = 0x2000) )
                {
                    $count = 0x1000
                    printf("Update Value\n")
                }
                target = 1
        }
}
```

对部分文件的$count参数进行修改。

页Partial Writes模拟：Oracle篇

- 开始测试

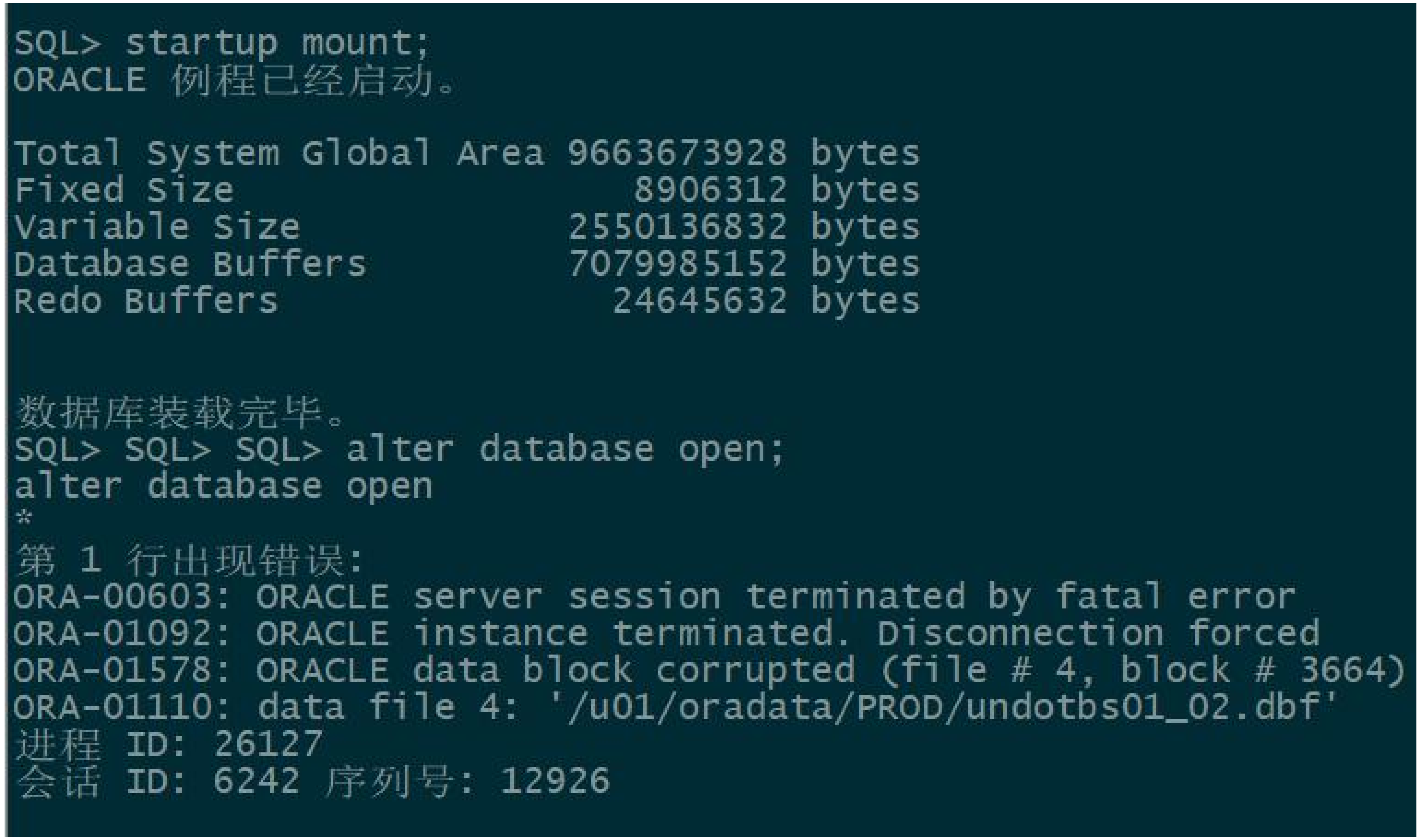


```
[root@localhost iotest]# sta
Begin...
20177 kernel.function("SyS_p                                             0 2000 20f5c000
Update Value
return:20177 kernel.function
20177 kernel.function("SyS_p                                             0 2000 25190000
Update Value
return:20177 kernel.function
20177 kernel.function("SyS_p                                             0 2000 251a8000
Update Value
return:20177 kernel.function
20177 kernel.function("SyS_p                                             0 2000 25236000
Update Value
return:20177 kernel.function
20177 kernel.function("SyS_p                                             0 2000 25478000
```

```
SQL> update u2.dtcc set c1=lower(c1) where id1=1;
commit;
alter system checkpoint;

已更新 1 行。

SQL>
提交完成。

SQL>
alter system checkpoint
*
第 1 行出现错误:
ORA-03113: 通信通道的文件结尾 进程 ID:
22538
会话 ID: 21262 序列号: 20043
```

页Partial Writes模拟：Oracle篇

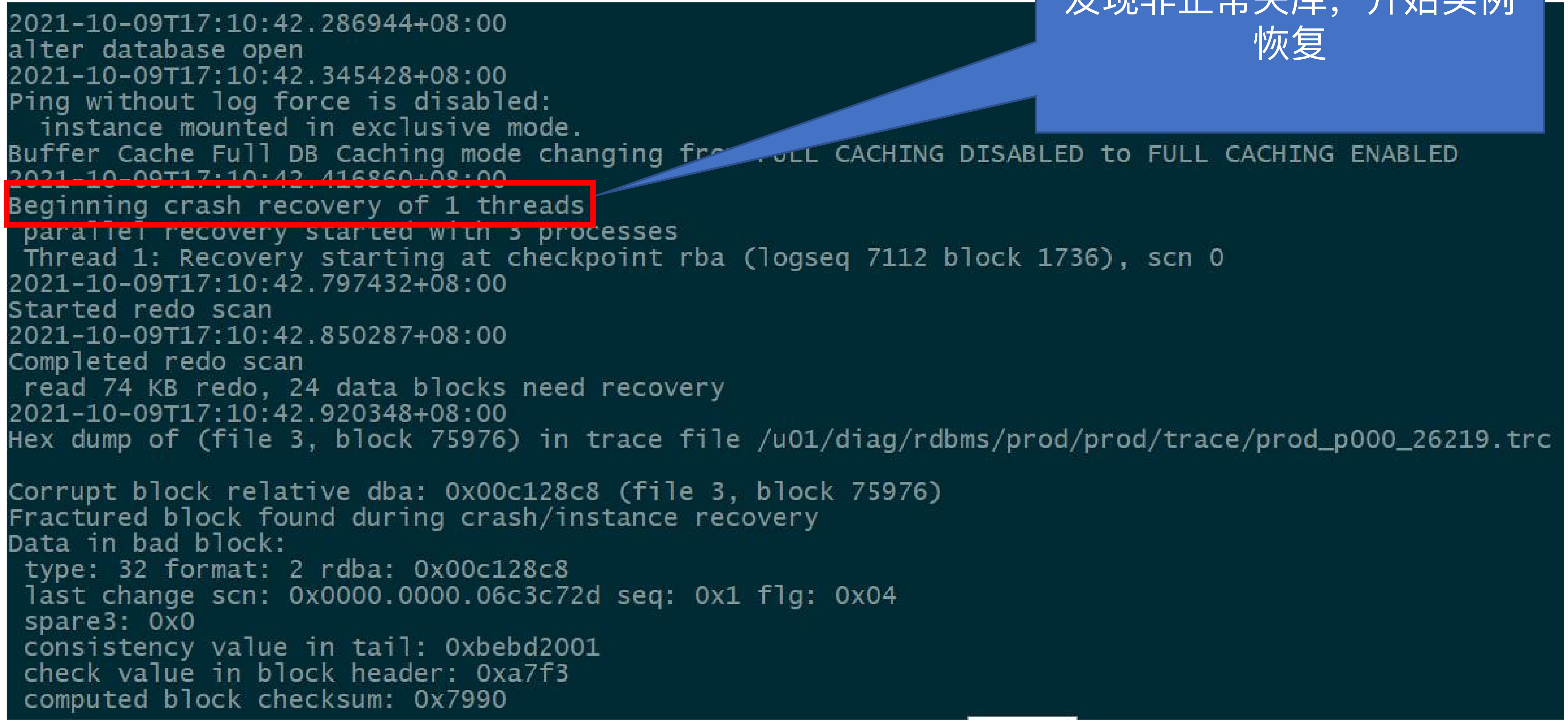
- 开始测试



```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area 9663673928 bytes
Fixed Size                  8906312 bytes
Variable Size            2550136832 bytes
Database Buffers         7079985152 bytes
Redo Buffers               24645632 bytes


数据库装载完毕。
SQL> SQL> SQL> alter database open;
alter database open
*
第 1 行出现错误:
ORA-00603: ORACLE server session terminated by fatal error
ORA-01092: ORACLE instance terminated. Disconnection forced
ORA-01578: ORACLE data block corrupted (file # 4, block # 3664)
ORA-01110: data file 4: '/u01/oradata/PROD/undotbs01_02.dbf'
进程 ID: 26127
会话 ID: 6242 序列号: 12926
```

页Partial Writes模拟：Oracle篇

- 开始测试



```
2021-10-09T17:10:42.286944+08:00
alter database open
2021-10-09T17:10:42.345428+08:00
Ping without log force is disabled:
  instance mounted in exclusive mode.
Buffer Cache Full DB Caching mode changing from FULL CACHING DISABLED to FULL CACHING ENABLED
2021-10-09T17:10:42.416860+08:00
Beginning crash recovery of 1 threads
 parallel recovery started with 3 processes
 Thread 1: Recovery starting at checkpoint rba (logseq 7112 block 1736), scn 0
2021-10-09T17:10:42.797432+08:00
Started redo scan
2021-10-09T17:10:42.850287+08:00
Completed redo scan
 read 74 KB redo, 24 data blocks need recovery
2021-10-09T17:10:42.920348+08:00
Hex dump of (file 3, block 75976) in trace file /u01/diag/rdbms/prod/prod/trace/prod_p000_26219.trc

Corrupt block relative dba: 0x00c128c8 (file 3, block 75976)
Fractured block found during crash/instance recovery
Data in bad block:
 type: 32 format: 2 rdba: 0x00c128c8
 last change scn: 0x0000.0000.06c3c72d seq: 0x1 flg: 0x04
 spare3: 0x0
 consistency value in tail: 0xbebd2001
 check value in block header: 0xa7f3
 computed block checksum: 0x7990
```

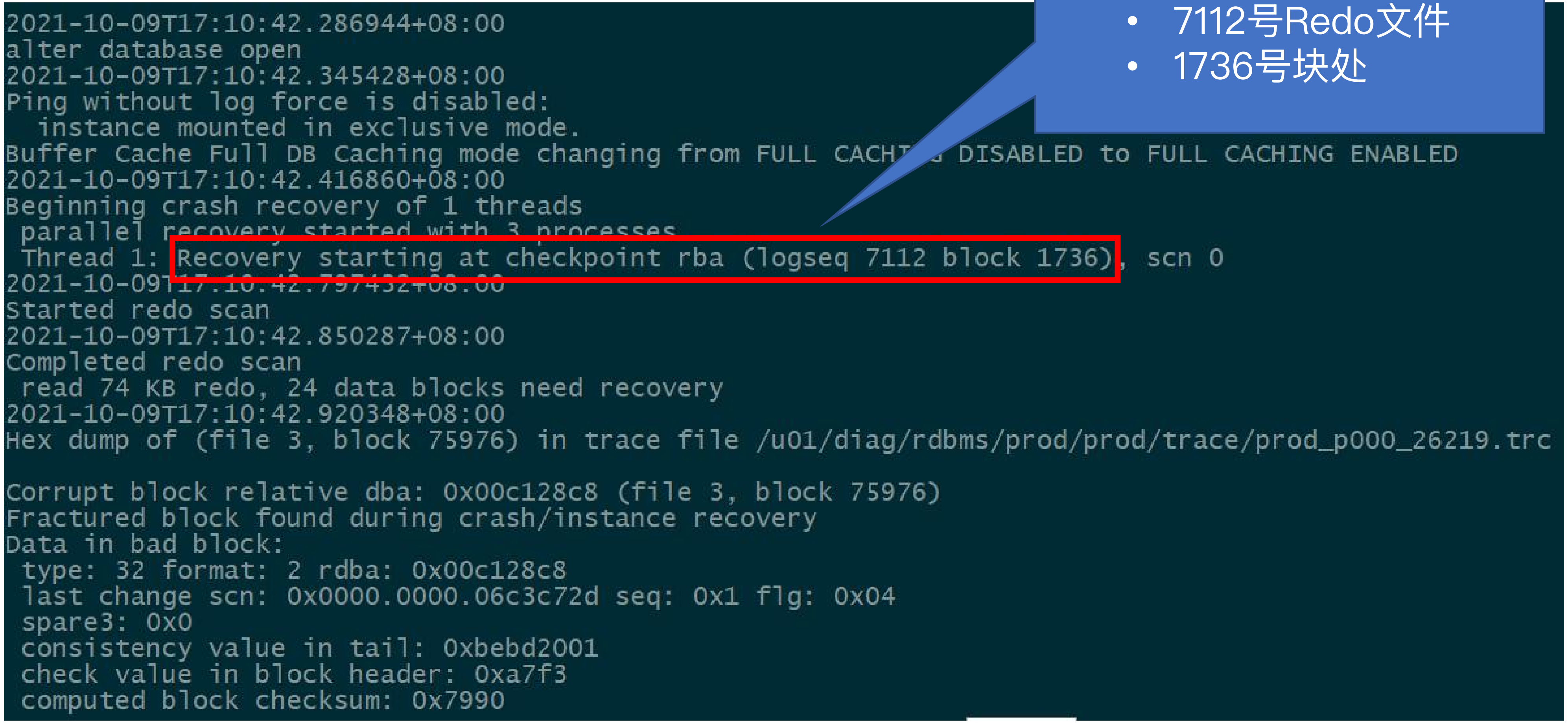
发现非正常关库，开始实例恢复

页Partial Writes模拟：Oracle篇

- 开始测试

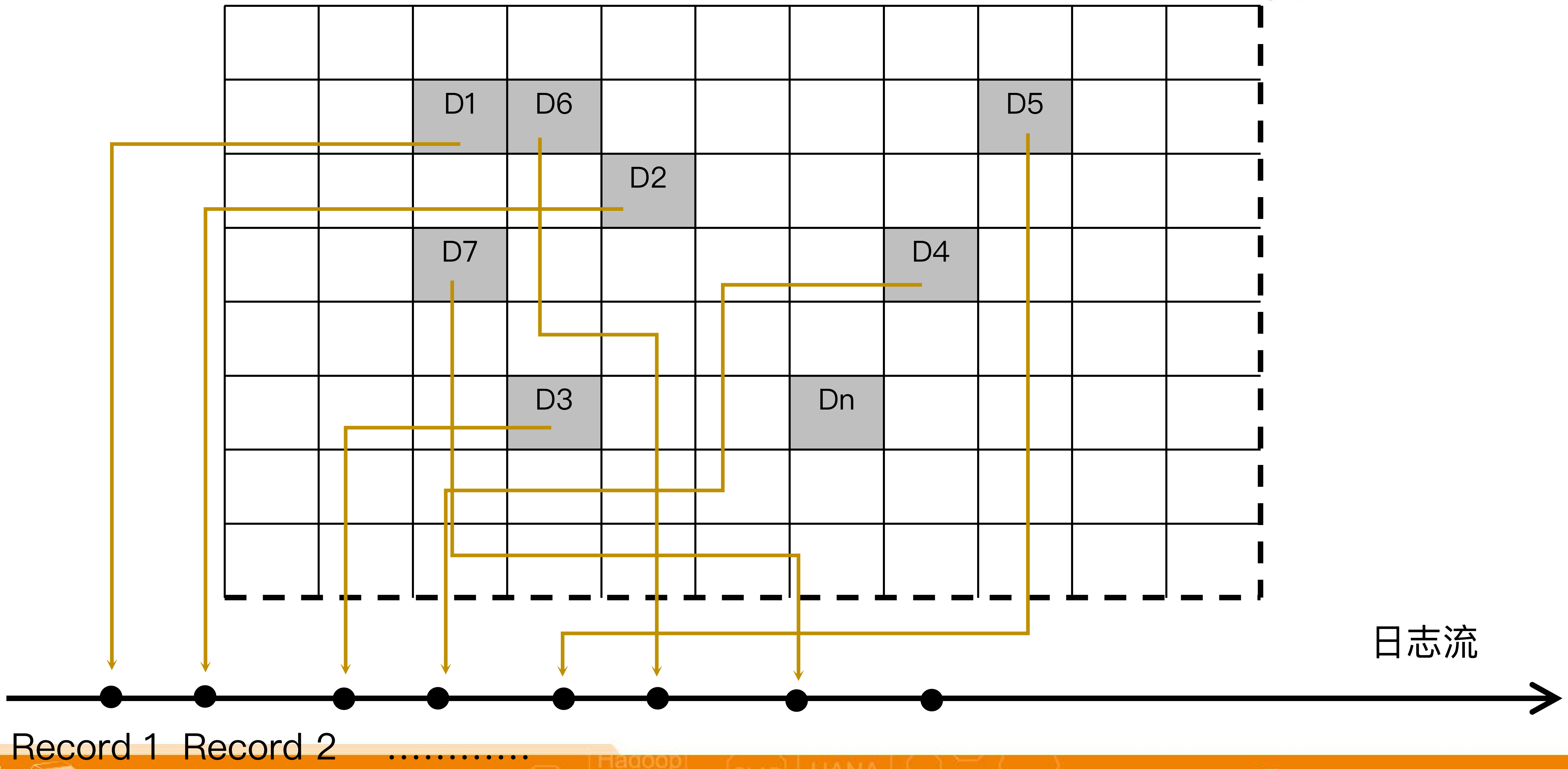
确定恢复的起点：

- 7112号Redo文件
- 1736号块处

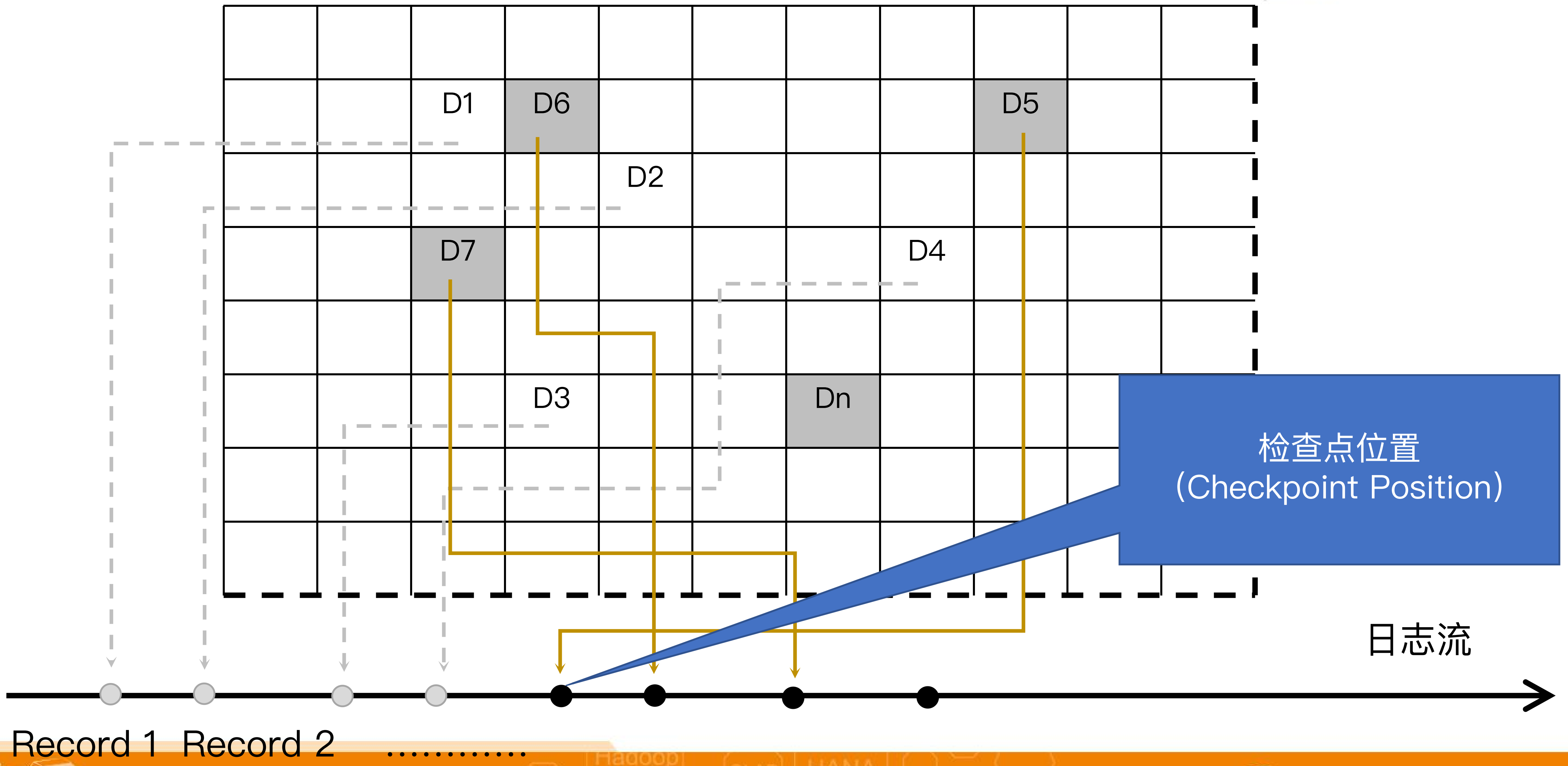


```
2021-10-09T17:10:42.286944+08:00
alter database open
2021-10-09T17:10:42.345428+08:00
Ping without log force is disabled:
  instance mounted in exclusive mode.
Buffer Cache Full DB Caching mode changing from FULL CACHING DISABLED to FULL CACHING ENABLED
2021-10-09T17:10:42.416860+08:00
Beginning crash recovery of 1 threads
 parallel recovery started with 3 processes
 Thread 1: Recovery starting at checkpoint rba (logseq 7112 block 1736), scn 0
2021-10-09T17:10:42.797432+08:00
Started redo scan
2021-10-09T17:10:42.850287+08:00
Completed redo scan
 read 74 KB redo, 24 data blocks need recovery
2021-10-09T17:10:42.920348+08:00
Hex dump of (file 3, block 75976) in trace file /u01/diag/rdbms/prod/prod/trace/prod_p000_26219.trc

Corrupt block relative dba: 0x00c128c8 (file 3, block 75976)
Fractured block found during crash/instance recovery
Data in bad block:
 type: 32 format: 2 rdba: 0x00c128c8
 last change scn: 0x0000.0000.06c3c72d seq: 0x1 flg: 0x04
 spare3: 0x0
 consistency value in tail: 0xbebd2001
 check value in block header: 0xa7f3
 computed block checksum: 0x7990
```

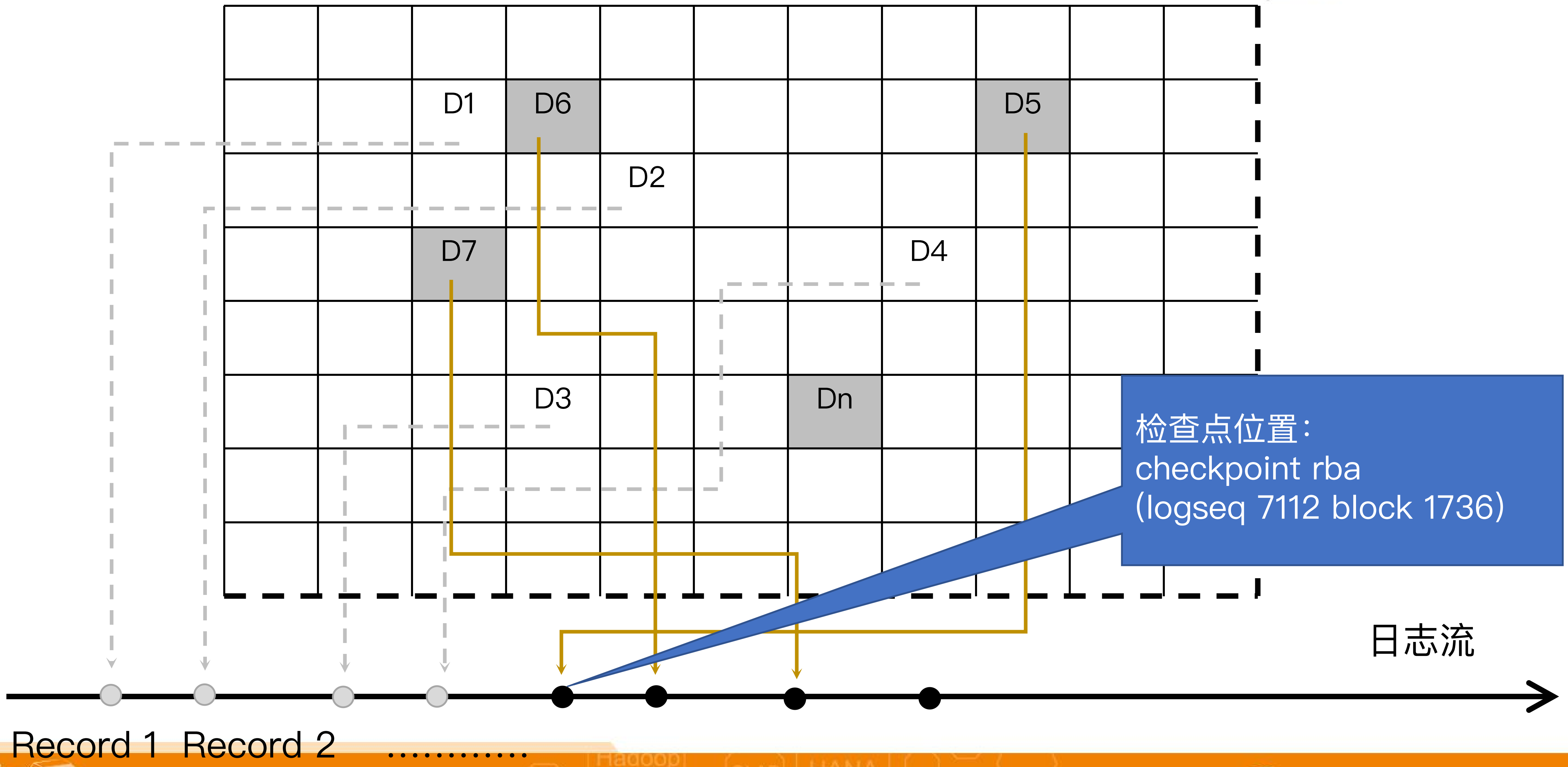
页Partial Writes模拟：Oracle篇



页Partial Writes模拟：Oracle篇

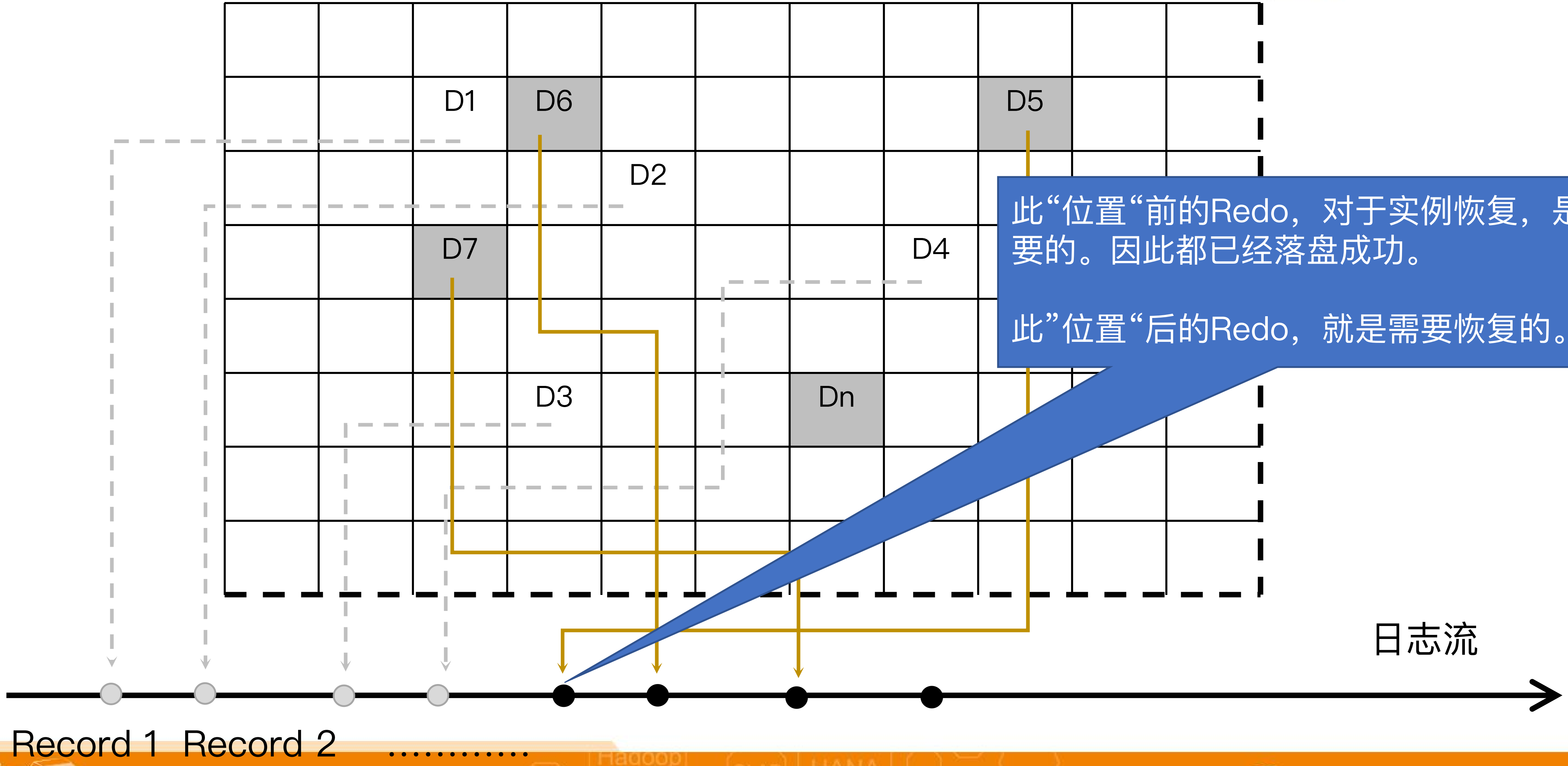


页Partial Writes模拟：Oracle篇



检查点位置：
checkpoint rba
(logseq 7112 block 1736)

页Partial Writes模拟：Oracle篇

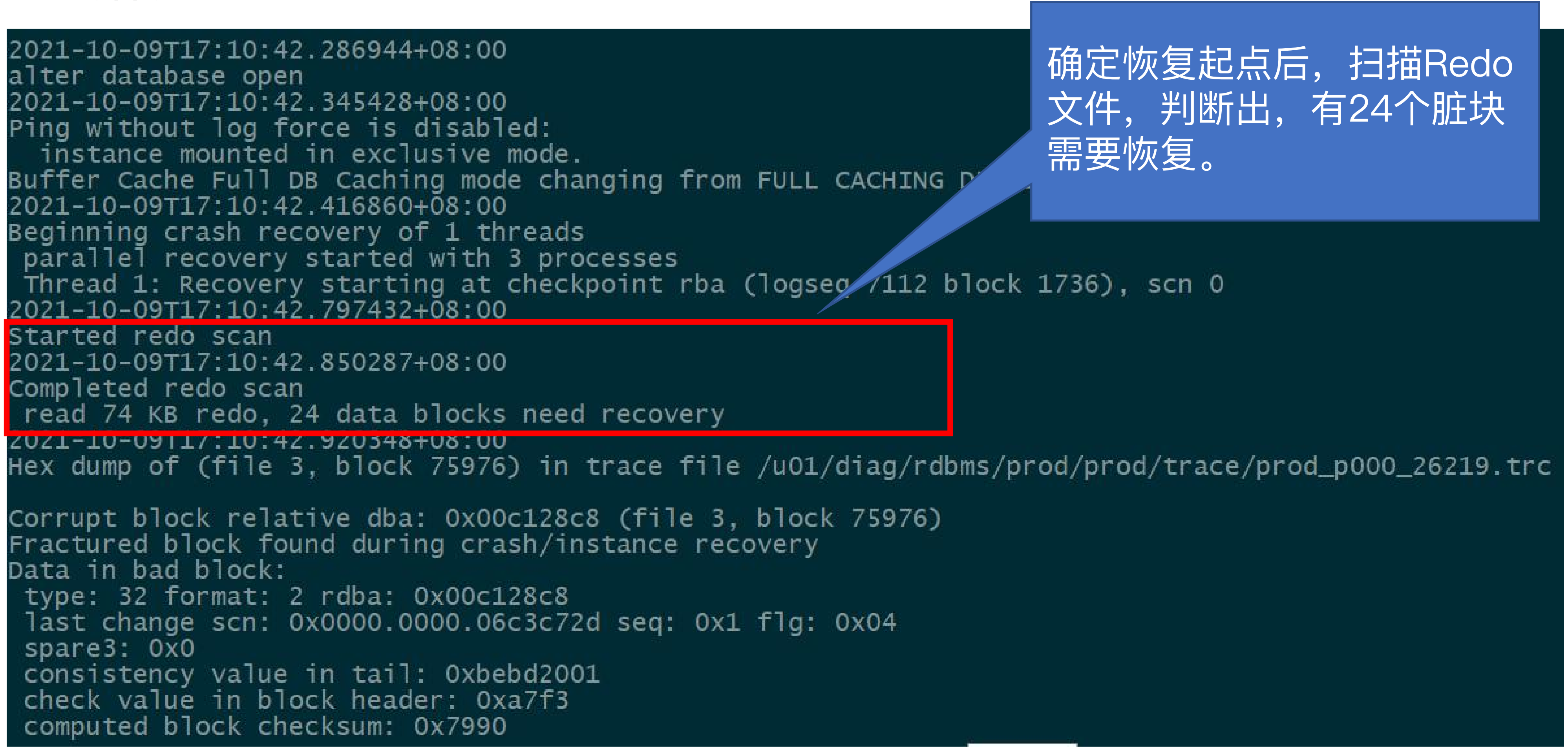


此“位置“前的Redo，对于实例恢复，是不需要的。因此都已经落盘成功。

此”位置“后的Redo，就是需要恢复的。

页Partial Writes模拟：Oracle篇

- 开始测试

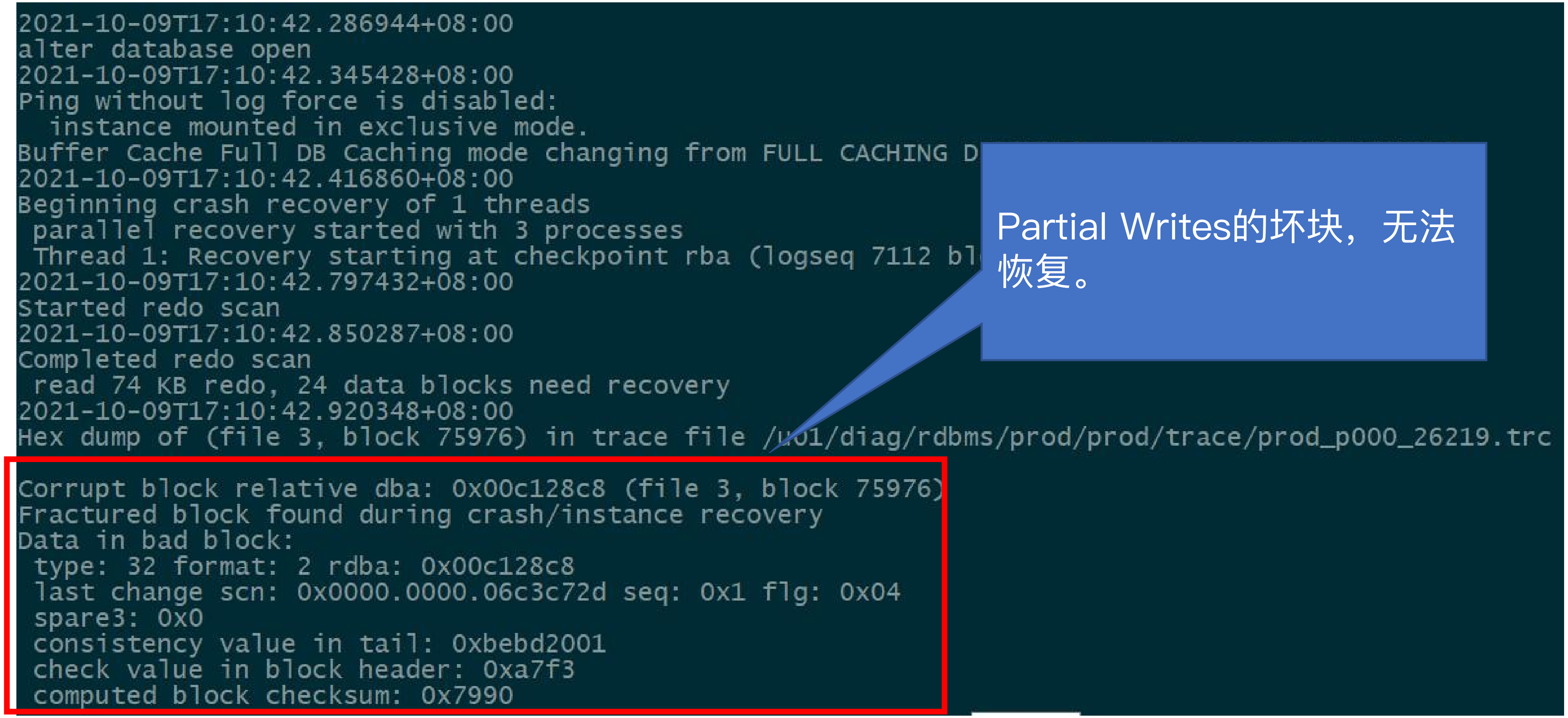


```
2021-10-09T17:10:42.286944+08:00
alter database open
2021-10-09T17:10:42.345428+08:00
Ping without log force is disabled:
  instance mounted in exclusive mode.
Buffer Cache Full DB Caching mode changing from FULL CACHING D
2021-10-09T17:10:42.416860+08:00
Beginning crash recovery of 1 threads
 parallel recovery started with 3 processes
 Thread 1: Recovery starting at checkpoint rba (logseq 112 block 1736), scn 0
2021-10-09T17:10:42.797432+08:00
Started redo scan
2021-10-09T17:10:42.850287+08:00
Completed redo scan
 read 74 KB redo, 24 data blocks need recovery
2021-10-09T17:10:42.920348+08:00
Hex dump of (file 3, block 75976) in trace file /u01/diag/rdbms/prod/prod/trace/prod_p000_26219.trc

Corrupt block relative dba: 0x00c128c8 (file 3, block 75976)
Fractured block found during crash/instance recovery
Data in bad block:
 type: 32 format: 2 rdba: 0x00c128c8
 last change scn: 0x0000.0000.06c3c72d seq: 0x1 flg: 0x04
 spare3: 0x0
 consistency value in tail: 0xbebd2001
 check value in block header: 0xa7f3
 computed block checksum: 0x7990
```

确定恢复起点后，扫描Redo文件，判断出，有24个脏块需要恢复。

页Partial Writes模拟：Oracle篇

- 开始测试



```
2021-10-09T17:10:42.286944+08:00
alter database open
2021-10-09T17:10:42.345428+08:00
Ping without log force is disabled:
  instance mounted in exclusive mode.
Buffer Cache Full DB Caching mode changing from FULL CACHING D
2021-10-09T17:10:42.416860+08:00
Beginning crash recovery of 1 threads
 parallel recovery started with 3 processes
 Thread 1: Recovery starting at checkpoint rba (logseq 7112 bl
2021-10-09T17:10:42.797432+08:00
Started redo scan
2021-10-09T17:10:42.850287+08:00
Completed redo scan
 read 74 KB redo, 24 data blocks need recovery
2021-10-09T17:10:42.920348+08:00
Hex dump of (file 3, block 75976) in trace file /u01/diag/rdbms/prod/prod/trace/prod_p000_26219.trc

Corrupt block relative dba: 0x00c128c8 (file 3, block 75976)
Fractured block found during crash/instance recovery
Data in bad block:
 type: 32 format: 2 rdba: 0x00c128c8
 last change scn: 0x0000.0000.06c3c72d seq: 0x1 flg: 0x04
 spare3: 0x0
 consistency value in tail: 0xbebd2001
 check value in block header: 0xa7f3
 computed block checksum: 0x7990
```

Partial Writes的坏块，无法恢复。

页Partial Writes模拟：Oracle篇

➢ 恢复的局限性

页（块）的损坏是千奇百怪的。数据又是十分重要的。

尝试用Redo在坏块基础上进行修复。坏块坏的可能近乎无限，你能想到的坏的情况是有限。

页Partial Writes模拟：Oracle篇

- 恢复的局限性

页（块）的损坏是千奇百怪的。数据又是十分重要的。

尝试用Redo在坏块基础上进行修复。坏块坏的可能近乎无限，你能想到的坏的情况是有限：

以有涯随无涯，殆已

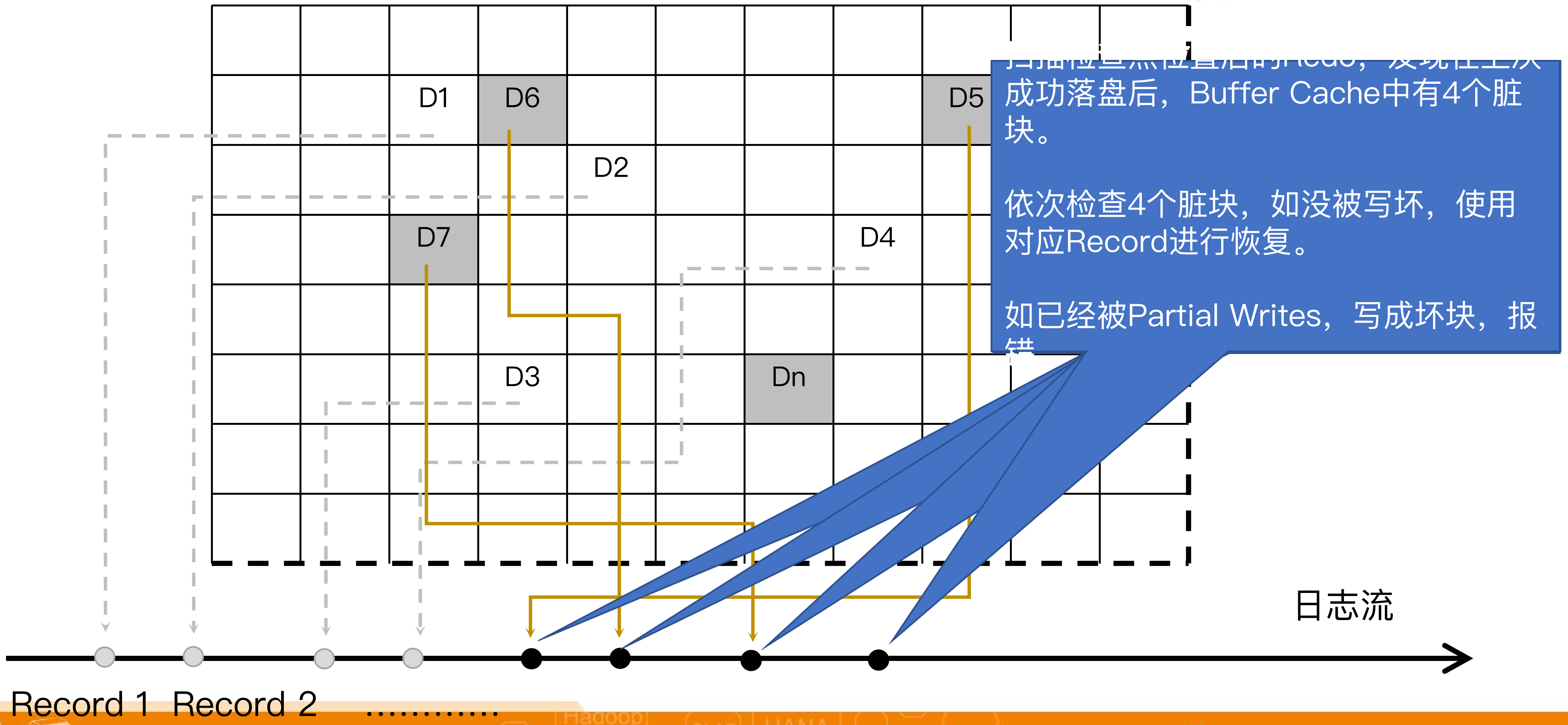
因此，数据库的方式都是，在前一份完好一致的、正确的数据基础上，应用日志（redo、xlog、binlog等），将数据推进到最近的时刻。

页Partial Writes模拟：Oracle篇总结

- Oracle篇总结

Oracle应对Partial Writes，依赖“检查“和介质恢复。

页Partial Writes模拟：Oracle篇总结



扫描检查点位置后的Redo，发现在上次成功落盘后，Buffer Cache中有4个脏块。

依次检查4个脏块，如没被写坏，使用对应Record进行恢复。

如已经被Partial Writes，写成坏块，报错

页Partial Writes模拟：PG篇

- I/O相关函数　：pwrite

- 准备测试数据:

```
create table dtcc2(id1 int primary key, id2 int, c1 varchar(30), c2 varchar(30)) ;

insert into dtcc2 values(1, 101, 'aaaaa1', 'aaaaaa1');
```

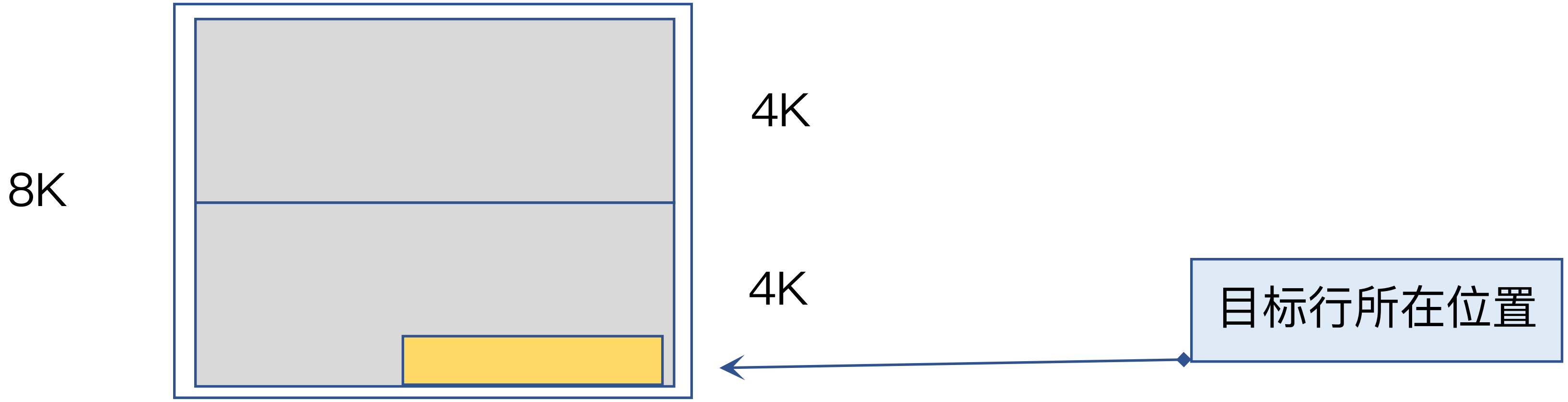
和Oracle一样，PG也是从下往上使用页空间，虽然只插入一行，但目标行在页的尾部。

如下方式观察表对应的文件:

cd PG的数据库目录/base/DATABASE_OID（Database_OID来自于pg_database视图）
hexdump –C TABLE_OID （Table_OID来自于pg_class视图）

页Partial Writes模拟：PG篇

- 准备测试数据



- 测试步骤：和Oracle方法类似
  - 修改目标行，使目标块成为脏页。
  - 手动发出检查点，触发CheckPoint进程写脏块。
  - 拦截I/O系统调用，修改count参数，造成一个8K块，前4K写成功、后4K写失败。
  - 观察、记录PG的处理过程。

页Partial Writes模拟：PG篇

- 观察I/O：

```
probe syscall.pwrite
{
        if (execname() == @1)
        {
                printf("old:%d %s %s %s %x %x %x %x\n", tid(), execname(), pp(), ppfunc(), $fd, $buf,
$count, $pos)
        }
}
```

- 开始测试

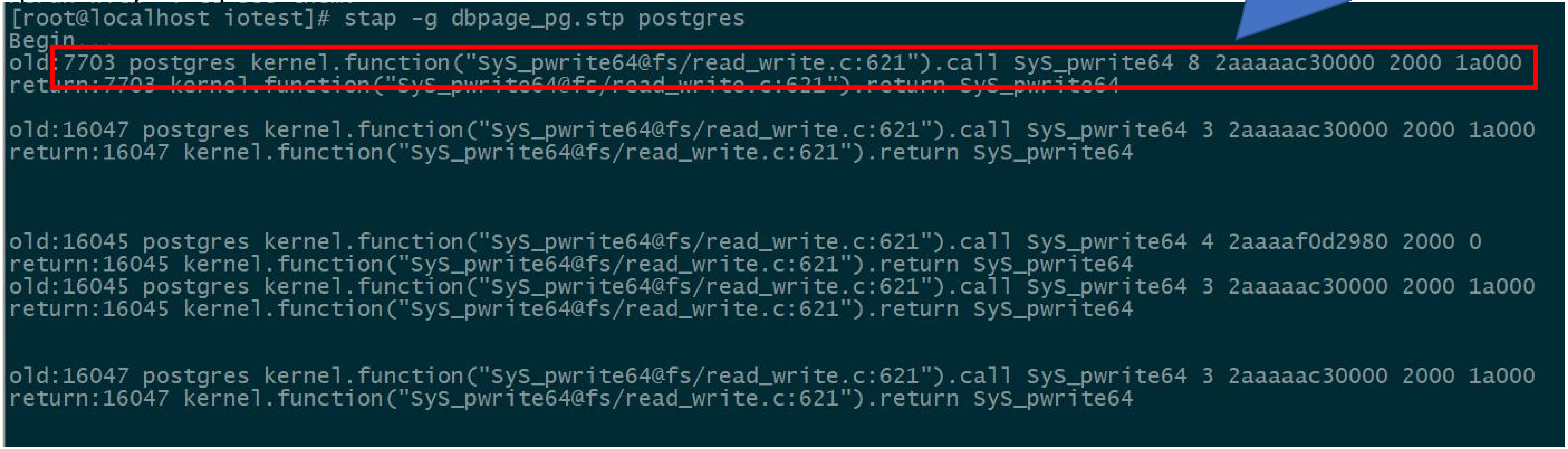
```
postgres=# update dtcc2 set c1=upper(c1) where id1=1;
UPDATE 1
postgres=# checkpoint;
```

页Partial Writes模拟：PG篇

- ➢ 观察I/O：

用户后台进程写WAL日志

probe syscall.pwrite



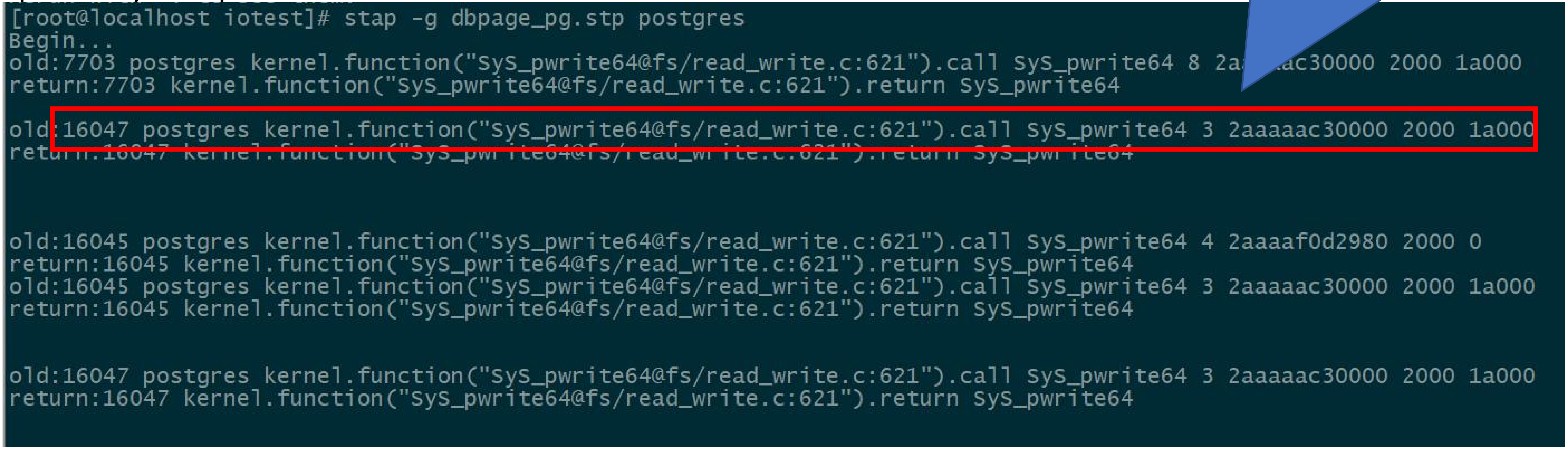
```
[root@localhost iotest]# stap -g dbpage_pg.stp postgres
Begin...
old:7703 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 8 2aaaaac30000 2000 1a000
return:7703 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64

old:16047 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16047 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


old:16045 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 2aaaaf0d2980 2000 0
return:16045 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
old:16045 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16045 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


old:16047 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16047 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

postgres=# checkpoint;

页Partial Writes模拟：PG篇

- 观察I/O：

probe syscall.pwrite



```
[root@localhost iotest]# stap -g dbpage_pg.stp postgres
Begin...
old:7703 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 8 2a    ac30000 2000 1a000
return:7703 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64

old:16047 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16047 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64



old:16045 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 2aaaaf0d2980 2000 0
return:16045 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
old:16045 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16045 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


old:16047 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16047 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

walwriter进程写WAL日志

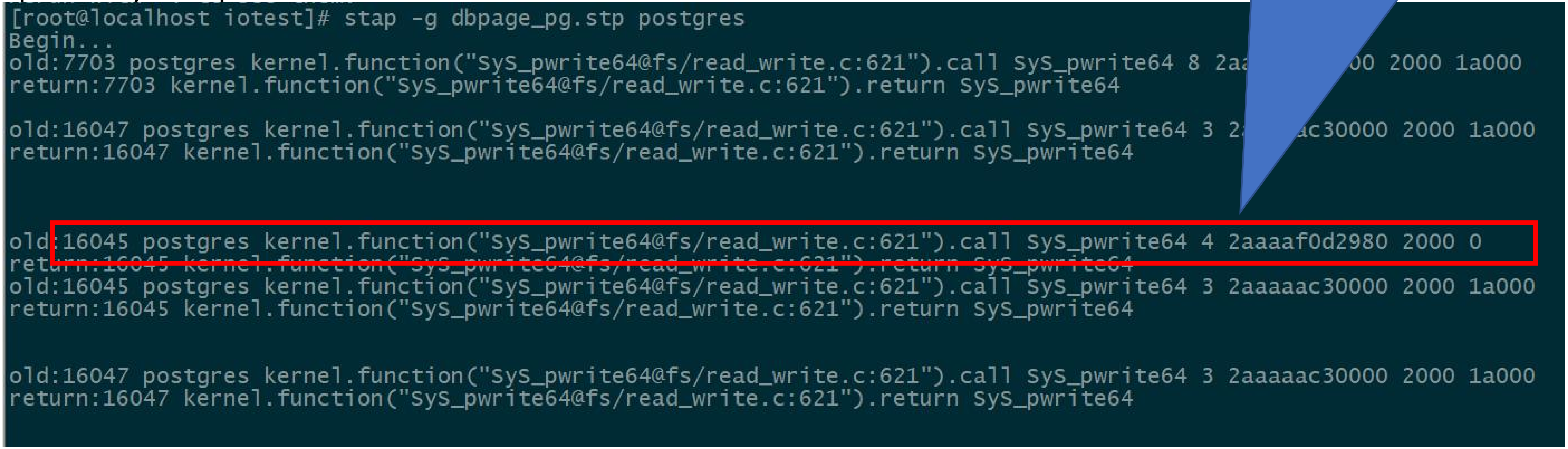
postgres=# checkpoint;

页Partial Writes模拟：PG篇

- 观察I/O：

Checkpoint进程写目标表对应的文件

probe syscall.pwrite

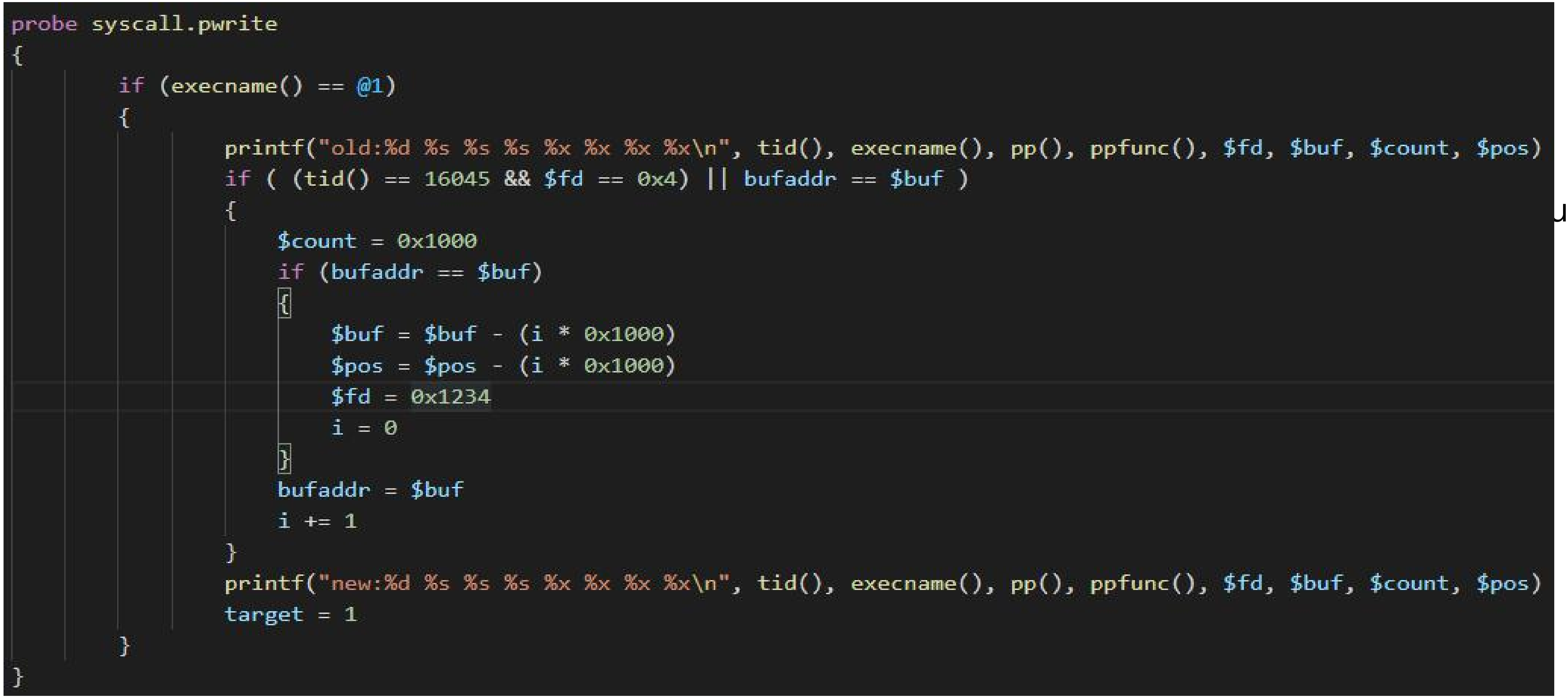


```
[root@localhost iotest]# stap -g dbpage_pg.stp postgres
Begin...
old:7703 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 8 2aa          00 2000 1a000
return:7703 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64

old:16047 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2     ac30000 2000 1a000
return:16047 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


old:16045 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 4 2aaaaf0d2980 2000 0
return:16045 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
old:16045 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16045 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64


old:16047 postgres kernel.function("SyS_pwrite64@fs/read_write.c:621").call SyS_pwrite64 3 2aaaaac30000 2000 1a000
return:16047 kernel.function("SyS_pwrite64@fs/read_write.c:621").return SyS_pwrite64
```

postgres=# checkpoint;

页Partial Writes模拟：PG篇

- 测试脚本：

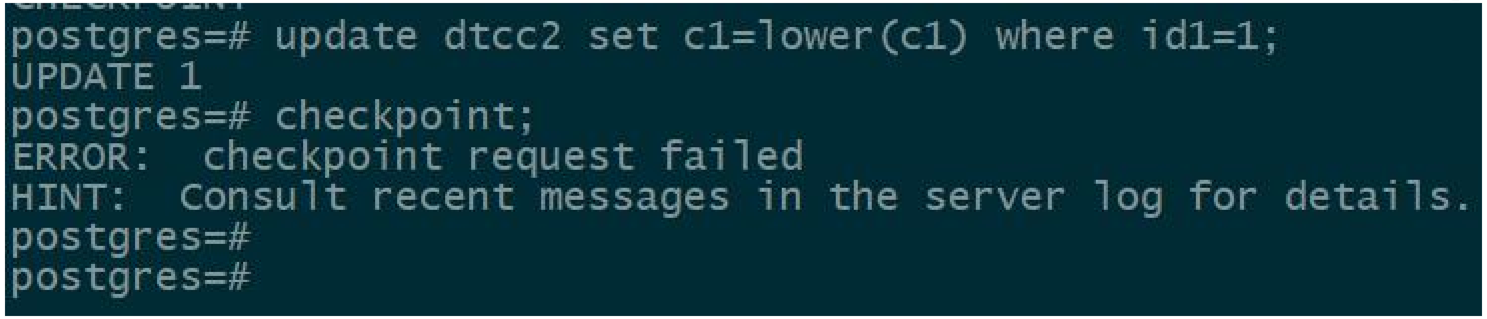


```
probe syscall.pwrite
{
        if (execname() == @1)
        {
                printf("old:%d %s %s %s %x %x %x %x\n", tid(), execname(), pp(), ppfunc(), $fd, $buf, $count, $pos)
                if ( (tid() == 16045 && $fd == 0x4) || bufaddr == $buf )
                {
                    $count = 0x1000
                    if (bufaddr == $buf)
                    {
                        $buf = $buf - (i * 0x1000)
                        $pos = $pos - (i * 0x1000)
                        $fd = 0x1234
                        i = 0
                    }
                    bufaddr = $buf
                    i += 1
                }
                printf("new:%d %s %s %s %x %x %x %x\n", tid(), execname(), pp(), ppfunc(), $fd, $buf, $count, $pos)
                target = 1
        }
}
```

uf,

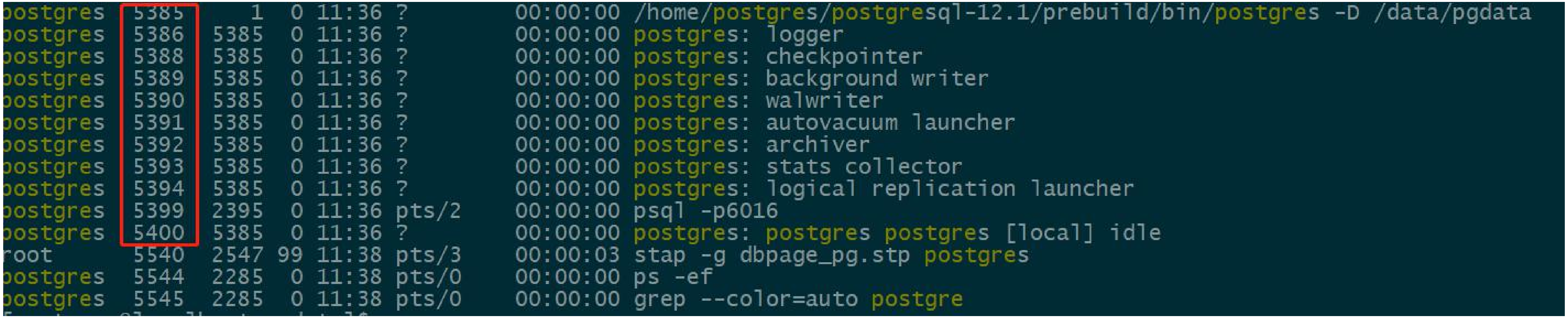
页Partial Writes模拟：PG篇

- 测试结果：



```
CHECKPOINT
postgres=# update dtcc2 set c1=lower(c1) where id1=1;
UPDATE 1
postgres=# checkpoint;
ERROR:  checkpoint request failed
HINT:  Consult recent messages in the server log for details.
postgres=#
postgres=#
```

I/O错误后数据库没有当掉，使用kill命令，kill掉所有PG的进程，模拟意外当库：



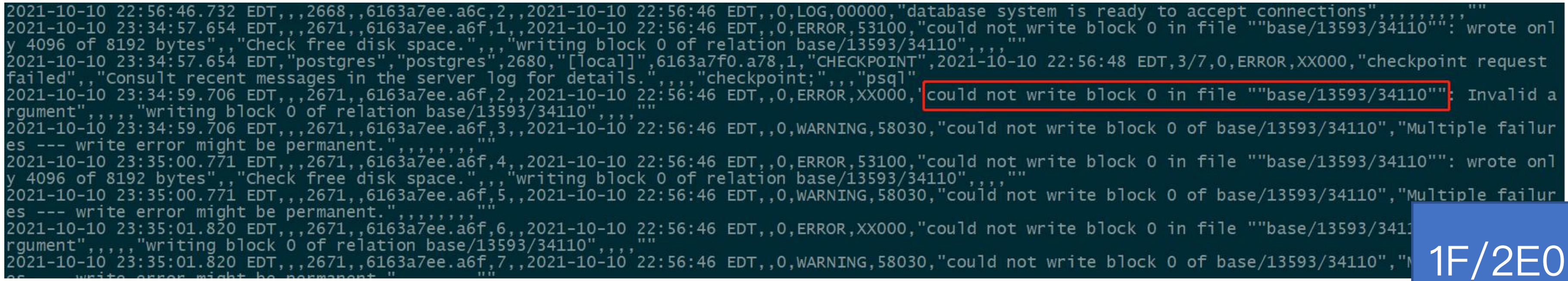
```
postgres   5385     1  0 11:36 ?        00:00:00 /home/postgres/postgresql-12.1/prebuild/bin/postgres -D /data/pgdata
postgres   5386  5385  0 11:36 ?        00:00:00 postgres: logger
postgres   5388  5385  0 11:36 ?        00:00:00 postgres: checkpointer
postgres   5389  5385  0 11:36 ?        00:00:00 postgres: background writer
postgres   5390  5385  0 11:36 ?        00:00:00 postgres: walwriter
postgres   5391  5385  0 11:36 ?        00:00:00 postgres: autovacuum launcher
postgres   5392  5385  0 11:36 ?        00:00:00 postgres: archiver
postgres   5393  5385  0 11:36 ?        00:00:00 postgres: stats collector
postgres   5394  5385  0 11:36 ?        00:00:00 postgres: logical replication launcher
postgres   5399  2395  0 11:36 pts/2    00:00:00 psql -p6016
postgres   5400  5385  0 11:36 ?        00:00:00 postgres: postgres postgres [local] idle
root       5540  2547 99 11:38 pts/3    00:00:03 stap -g dbpage_pg.stp postgres
postgres   5544  2285  0 11:38 pts/0    00:00:00 ps -ef
postgres   5545  2285  0 11:38 pts/0    00:00:00 grep --color=auto postgre
```

kill –9 5385 5386 5388 5389 5390 5391 5392 5393 5394 5400

页Partial Writes模拟：PG篇

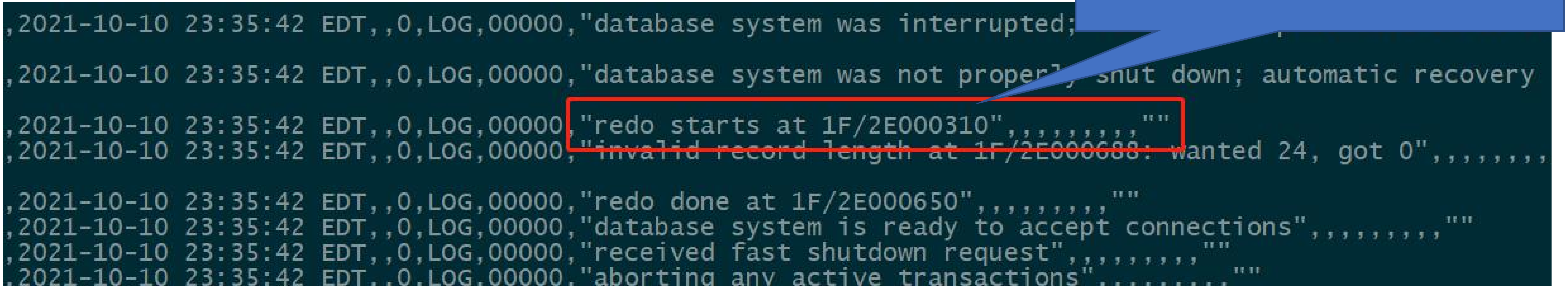
➢ 测试结果：

- 查看日志，有大量I/O错误：



```
2021-10-10 22:56:46.732 EDT,,,2668,,6163a7ee.a6c,2,,2021-10-10 22:56:46 EDT,,0,LOG,00000,"database system is ready to accept connections",,,,,,,,,""
2021-10-10 23:34:57.654 EDT,,,2671,,6163a7ee.a6f,1,,2021-10-10 22:56:46 EDT,,0,ERROR,53100,"could not write block 0 in file ""base/13593/34110"": wrote onl
y 4096 of 8192 bytes",,"Check free disk space.",,,,"writing block 0 of relation base/13593/34110",,,,""
2021-10-10 23:34:57.654 EDT,"postgres","postgres",2680,"[local]",6163a7f0.a78,1,"CHECKPOINT",2021-10-10 22:56:48 EDT,3/7,0,ERROR,XX000,"checkpoint request
failed",,"Consult recent messages in the server log for details.",,,,,"checkpoint;",,,,"psql"
2021-10-10 23:34:59.706 EDT,,,2671,,6163a7ee.a6f,2,,2021-10-10 22:56:46 EDT,,0,ERROR,XX000,"could not write block 0 in file ""base/13593/34110"": Invalid a
rgument",,,,,,"writing block 0 of relation base/13593/34110",,,,""
2021-10-10 23:34:59.706 EDT,,,2671,,6163a7ee.a6f,3,,2021-10-10 22:56:46 EDT,,0,WARNING,58030,"could not write block 0 of base/13593/34110","Multiple failur
es --- write error might be permanent.",,,,,,,,,""
2021-10-10 23:35:00.771 EDT,,,2671,,6163a7ee.a6f,4,,2021-10-10 22:56:46 EDT,,0,ERROR,53100,"could not write block 0 in file ""base/13593/34110"": wrote onl
y 4096 of 8192 bytes",,"Check free disk space.",,,,"writing block 0 of relation base/13593/34110",,,,""
2021-10-10 23:35:00.771 EDT,,,2671,,6163a7ee.a6f,5,,2021-10-10 22:56:46 EDT,,0,WARNING,58030,"could not write block 0 of base/13593/34110","Multiple failur
es --- write error might be permanent.",,,,,,,,,""
2021-10-10 23:35:01.820 EDT,,,2671,,6163a7ee.a6f,6,,2021-10-10 22:56:46 EDT,,0,ERROR,XX000,"could not write block 0 in file ""base/13593/3411
rgument",,,,,,"writing block 0 of relation base/13593/34110",,,,""
2021-10-10 23:35:01.820 EDT,,,2671,,6163a7ee.a6f,7,,2021-10-10 22:56:46 EDT,,0,WARNING,58030,"could not write block 0 of base/13593/34110","M
es --- write error might be permanent.",,,,,,,,,""
```

- 重新启动PG，PG自动进行恢复：



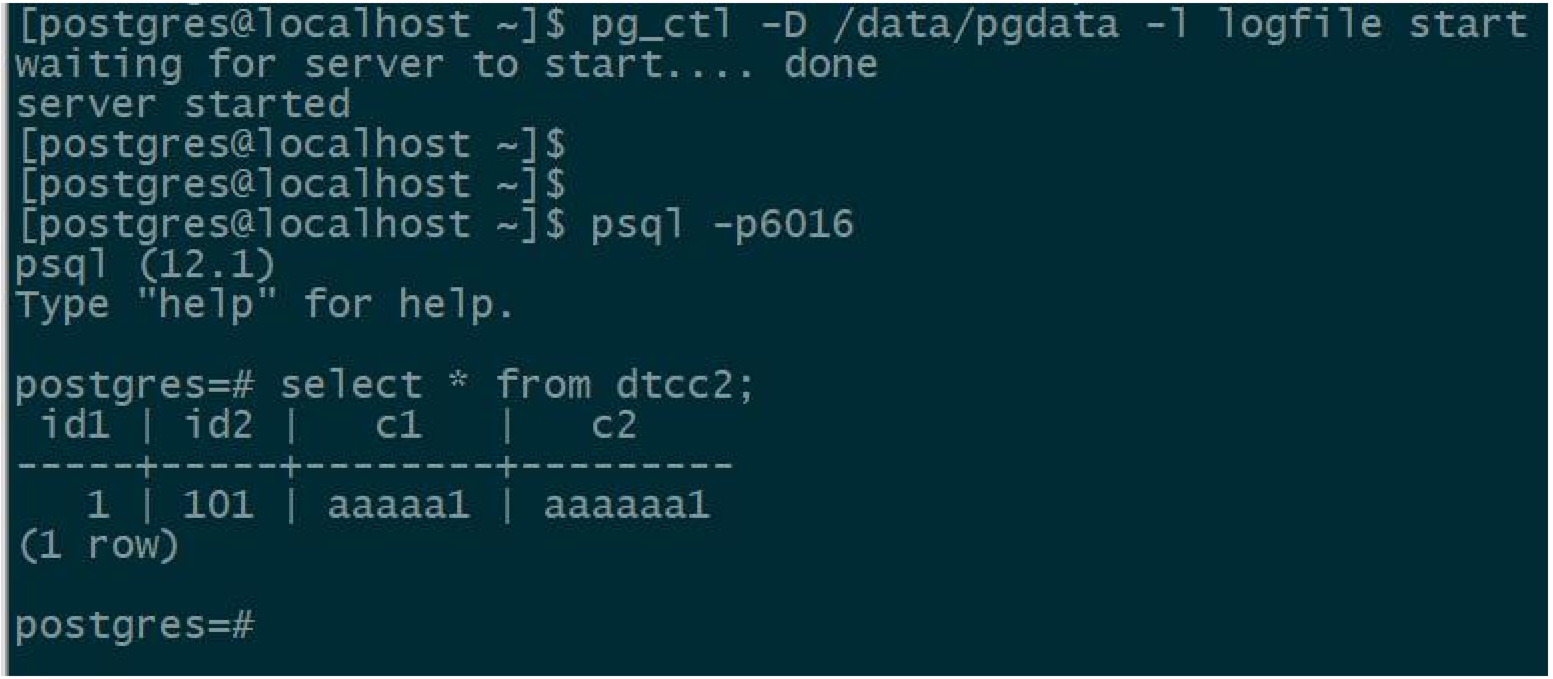
```
,2021-10-10 23:35:42 EDT,,0,LOG,00000,"database system was interrupted;

,2021-10-10 23:35:42 EDT,,0,LOG,00000,"database system was not properly shut down; automatic recovery

,2021-10-10 23:35:42 EDT,,0,LOG,00000,"redo starts at 1F/2E000310",,,,,,,,,""
,2021-10-10 23:35:42 EDT,,0,LOG,00000,"invalid record length at 1F/2E000688: wanted 24, got 0",,,,,,,,,

,2021-10-10 23:35:42 EDT,,0,LOG,00000,"redo done at 1F/2E000650",,,,,,,,,""
,2021-10-10 23:35:42 EDT,,0,LOG,00000,"database system is ready to accept connections",,,,,,,,,""
,2021-10-10 23:35:42 EDT,,0,LOG,00000,"received fast shutdown request",,,,,,,,,""
,2021-10-10 23:35:42 EDT,,0,LOG,00000,"aborting any active transactions",,,,,,,,,""
```

1F/2E000310，就是“检查点位置”

相关信息记录在PG的控制文件中

页Partial Writes模拟：PG篇

- 测试结果:

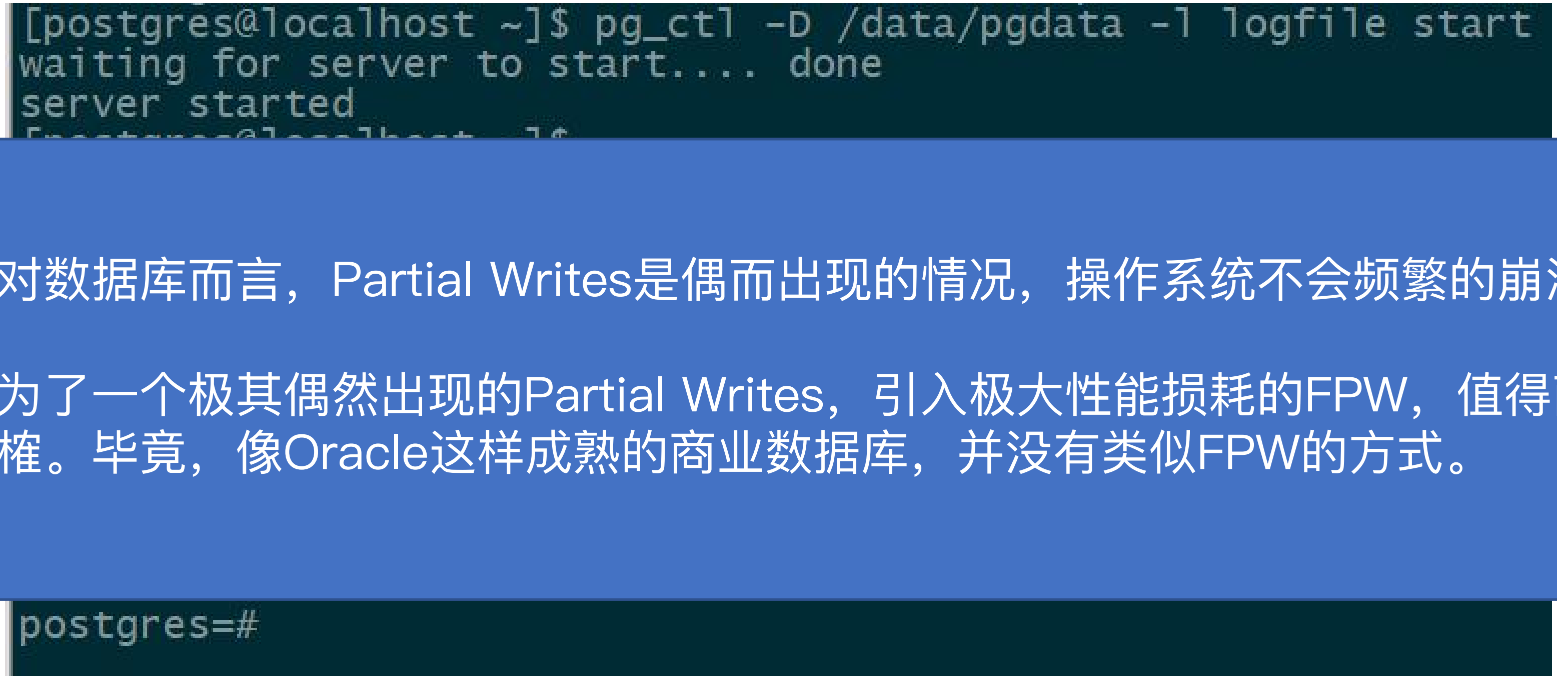


```
[postgres@localhost ~]$ pg_ctl -D /data/pgdata -l logfile start
waiting for server to start.... done
server started
[postgres@localhost ~]$
[postgres@localhost ~]$
[postgres@localhost ~]$ psql -p6016
psql (12.1)
Type "help" for help.

postgres=# select * from dtcc2;
 id1 | id2 |   c1   |   c2
-----+-----+--------+---------
   1 | 101 | aaaaa1 | aaaaaa1
(1 row)

postgres=#
```

再次启动数据库，一切正常，数据也没有丢失。最后的测试语句，就是将数据改为小写。

页Partial Writes模拟：PG篇

➢ 测试结果:



```
[postgres@localhost ~]$ pg_ctl -D /data/pgdata -l logfile start
waiting for server to start.... done
server started
[postgres@localhost ~]$
```

- 对数据库而言，Partial Writes是偶而出现的情况，操作系统不会频繁的崩溃。
- 为了一个极其偶然出现的Partial Writes，引入极大性能损耗的FPW，值得商榷。毕竟，像Oracle这样成熟的商业数据库，并没有类似FPW的方式。

```
postgres=#
```

再次启动数据库，一切正常，数据也没有丢失。最后的测试语句，就是将数据改为小写。

PG以巨大的性能代价（超30%性能下降），解决了Partial Writes问题。

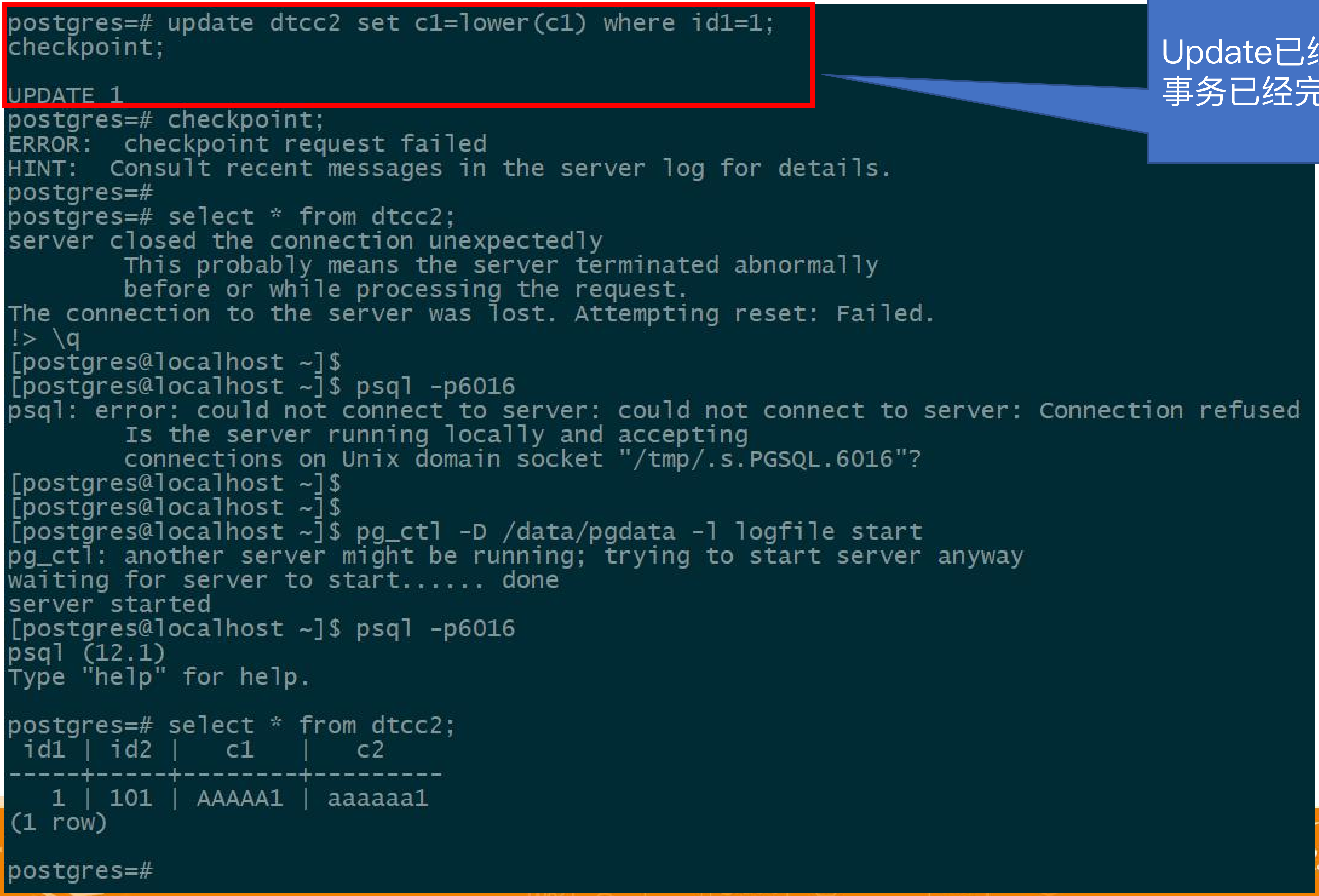
页Partial Writes模拟：PG篇 -- 关闭FPW的测试

➢ 测试结果：

测试步骤相同，唯一区别FPW参数设置为OFF：

- 执行脚本
- 发出Update SQL和Checkpoint命令
- kill 掉所有PG进程
- 重启数据库，观察结果

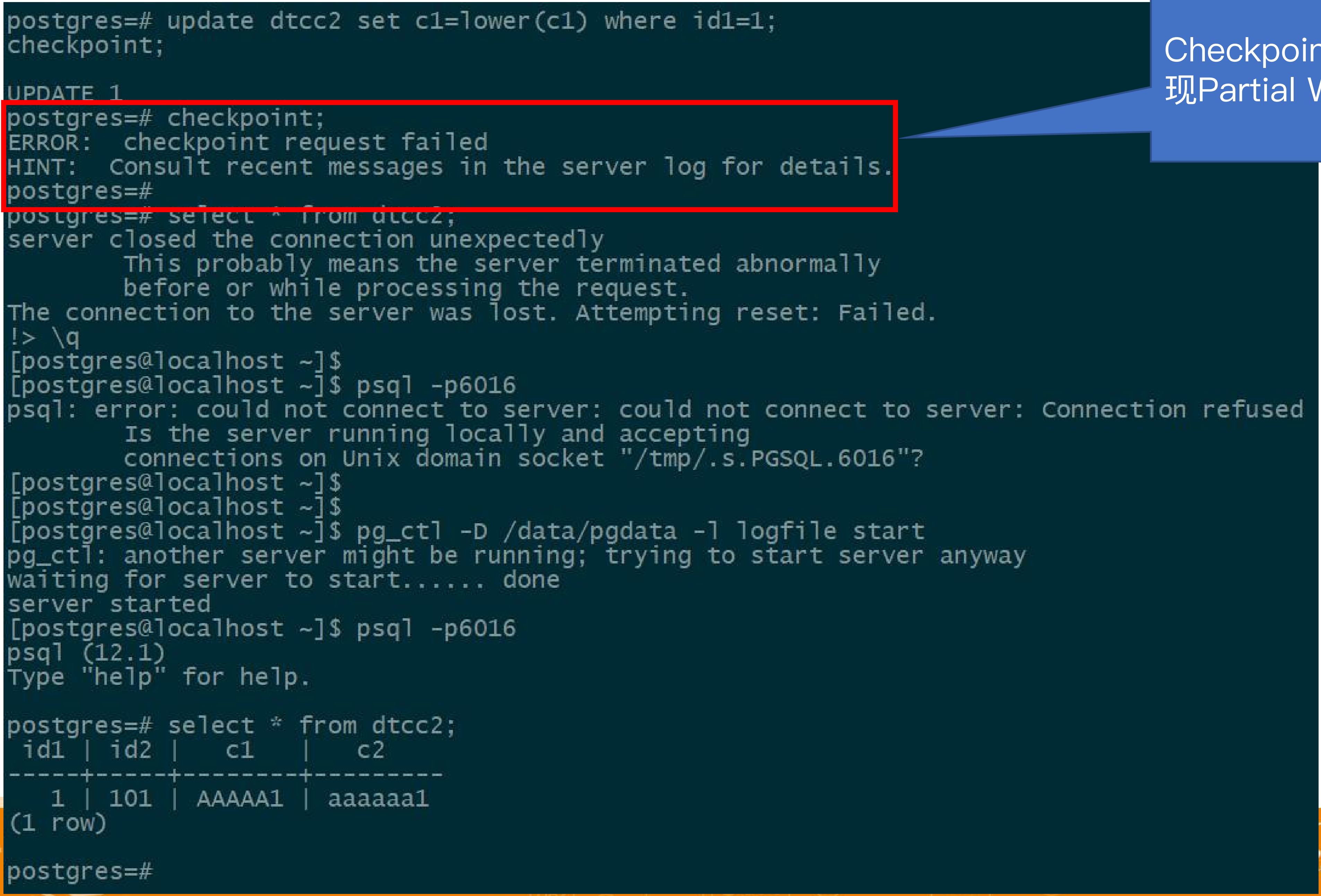
页Partial Writes模拟：PG篇 -- 关闭FPW的测试



```
postgres=# update dtcc2 set c1=lower(c1) where id1=1;
checkpoint;

UPDATE 1
postgres=# checkpoint;
ERROR:  checkpoint request failed
HINT:  Consult recent messages in the server log for details.
postgres=#
postgres=# select * from dtcc2;
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Failed.
!> \q
[postgres@localhost ~]$
[postgres@localhost ~]$ psql -p6016
psql: error: could not connect to server: could not connect to server: Connection refused
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.6016"?
[postgres@localhost ~]$
[postgres@localhost ~]$
[postgres@localhost ~]$ pg_ctl -D /data/pgdata -l logfile start
pg_ctl: another server might be running; trying to start server anyway
waiting for server to start...... done
server started
[postgres@localhost ~]$ psql -p6016
psql (12.1)
Type "help" for help.

postgres=# select * from dtcc2;
 id1 | id2 |   c1    |    c2
-----+-----+---------+----------
   1 | 101 | AAAAA1 | aaaaaa1
(1 row)

postgres=#
```

Update已经完成（隐含提交），
事务已经完成，C1列将被改为小写。

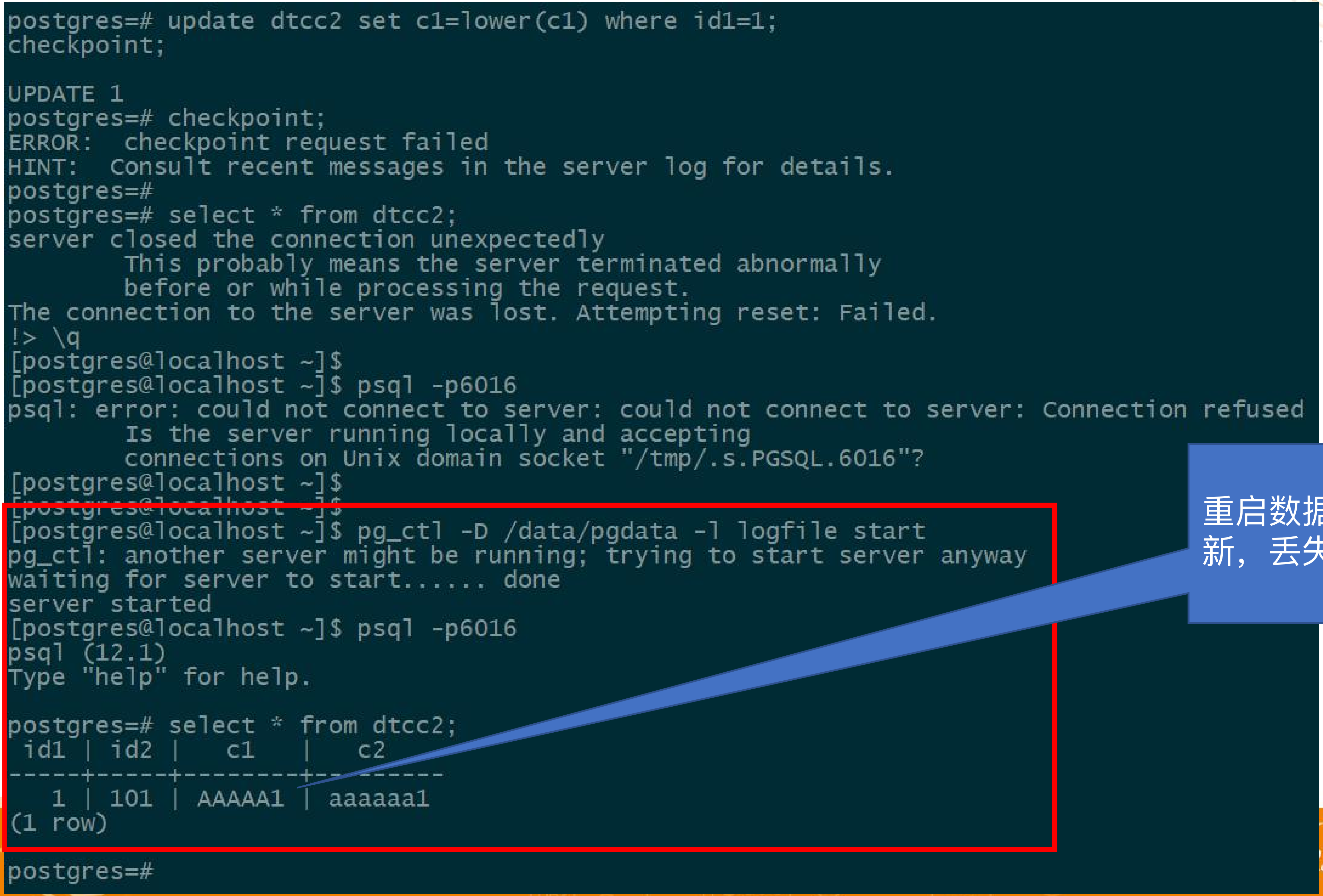
页Partial Writes模拟：PG篇 -- 关闭FPW的测试



```
postgres=# update dtcc2 set c1=lower(c1) where id1=1;
checkpoint;

UPDATE 1
postgres=# checkpoint;
ERROR:  checkpoint request failed
HINT:  Consult recent messages in the server log for details.
postgres=#
postgres=# select * from dtcc2;
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Failed.
!> \q
[postgres@localhost ~]$
[postgres@localhost ~]$ psql -p6016
psql: error: could not connect to server: could not connect to server: Connection refused
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.6016"?
[postgres@localhost ~]$
[postgres@localhost ~]$
[postgres@localhost ~]$ pg_ctl -D /data/pgdata -l logfile start
pg_ctl: another server might be running; trying to start server anyway
waiting for server to start...... done
server started
[postgres@localhost ~]$ psql -p6016
psql (12.1)
Type "help" for help.

postgres=# select * from dtcc2;
 id1 | id2 |   c1   |    c2
-----+-----+--------+---------
   1 | 101 | AAAAA1 | aaaaaa1
(1 row)

postgres=#
```

Checkpoint失败，写数据错误，出现Partial Writes。

页Partial Writes模拟：PG篇 -- 关闭FPW的测试



```
postgres=# update dtcc2 set c1=lower(c1) where id1=1;
checkpoint;

UPDATE 1
postgres=# checkpoint;
ERROR:  checkpoint request failed
HINT:  Consult recent messages in the server log for details.
postgres=#
postgres=# select * from dtcc2;
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Failed.
!> \q
[postgres@localhost ~]$
[postgres@localhost ~]$ psql -p6016
psql: error: could not connect to server: could not connect to server: Connection refused
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.6016"?
[postgres@localhost ~]$
[postgres@localhost ~]$
[postgres@localhost ~]$ pg_ctl -D /data/pgdata -l logfile start
pg_ctl: another server might be running; trying to start server anyway
waiting for server to start...... done
server started
[postgres@localhost ~]$ psql -p6016
psql (12.1)
Type "help" for help.

postgres=# select * from dtcc2;
 id1 | id2 |   c1    |   c2
-----+-----+---------+---------
   1 | 101 | AAAAA1  | aaaaaa1
(1 row)

postgres=#
```

重启数据库后，最后已经提交的更新，丢失。数据仍是大写’A’。

PostgreSQL Partial Writes解决方案的优化

➢ 为什么不能参照MySQL的Double Write（双写）

- 前文已经有测试，无法恢复所有Partial Writes场景
- 根据前文关掉FPW的测试，数据存在不一致的风险

结论：PG中关闭FPW引入双写，存在数据不一致的可能

➢ 参

- 报出错误
- 最终依赖基于备份的恢复

PostgreSQL Partial Writes解决方案的优化

➢ 摸着Oracle过河

- PG和Oracle使用一样的物理日志，备份恢复体系的原理一致
- Oracle在控制文件中记录每次检查点完成后的检查点位置，PG也有一模一样的机制。

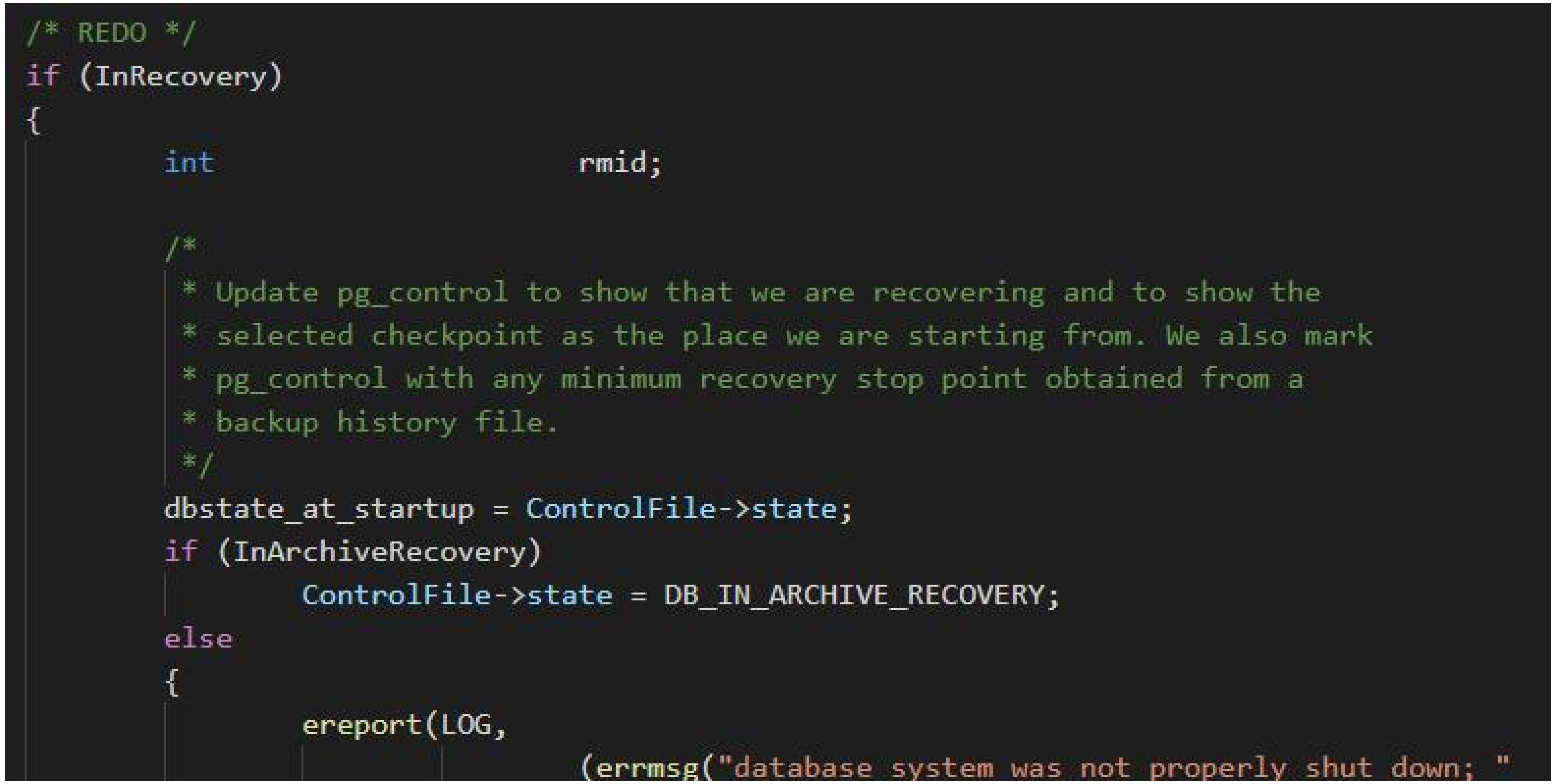
➢ 目标

- PG完成可以实现和Oracle一模一样的机制：在XLOG中，检查“检查点位置”后XLOG Record对应的块，如没被写坏，使用对应Record进行恢复。如已经被Partial Writes，写成坏块，报错。
- 安全的关闭FPW，（至少）提升30%性能。

Partial Writes解决方案的优化

➢ PG的解决方案：

- PG的恢复在StartupXLOG()函数中。（xlog.c文件）
- 在此if之后，就是PG的恢复流程。



```
/* REDO */
if (InRecovery)
{
        int                         rmid;

        /*
         * Update pg_control to show that we are recovering and to show the
         * selected checkpoint as the place we are starting from. We also mark
         * pg_control with any minimum recovery stop point obtained from a
         * backup history file.
         */
        dbstate_at_startup = ControlFile->state;
        if (InArchiveRecovery)
                ControlFile->state = DB_IN_ARCHIVE_RECOVERY;
        else
        {
                ereport(LOG,
                                (errmsg("database system was not properly shut down; "
```

Partial Writes解决方案的优化

➢ PG的解决方案：

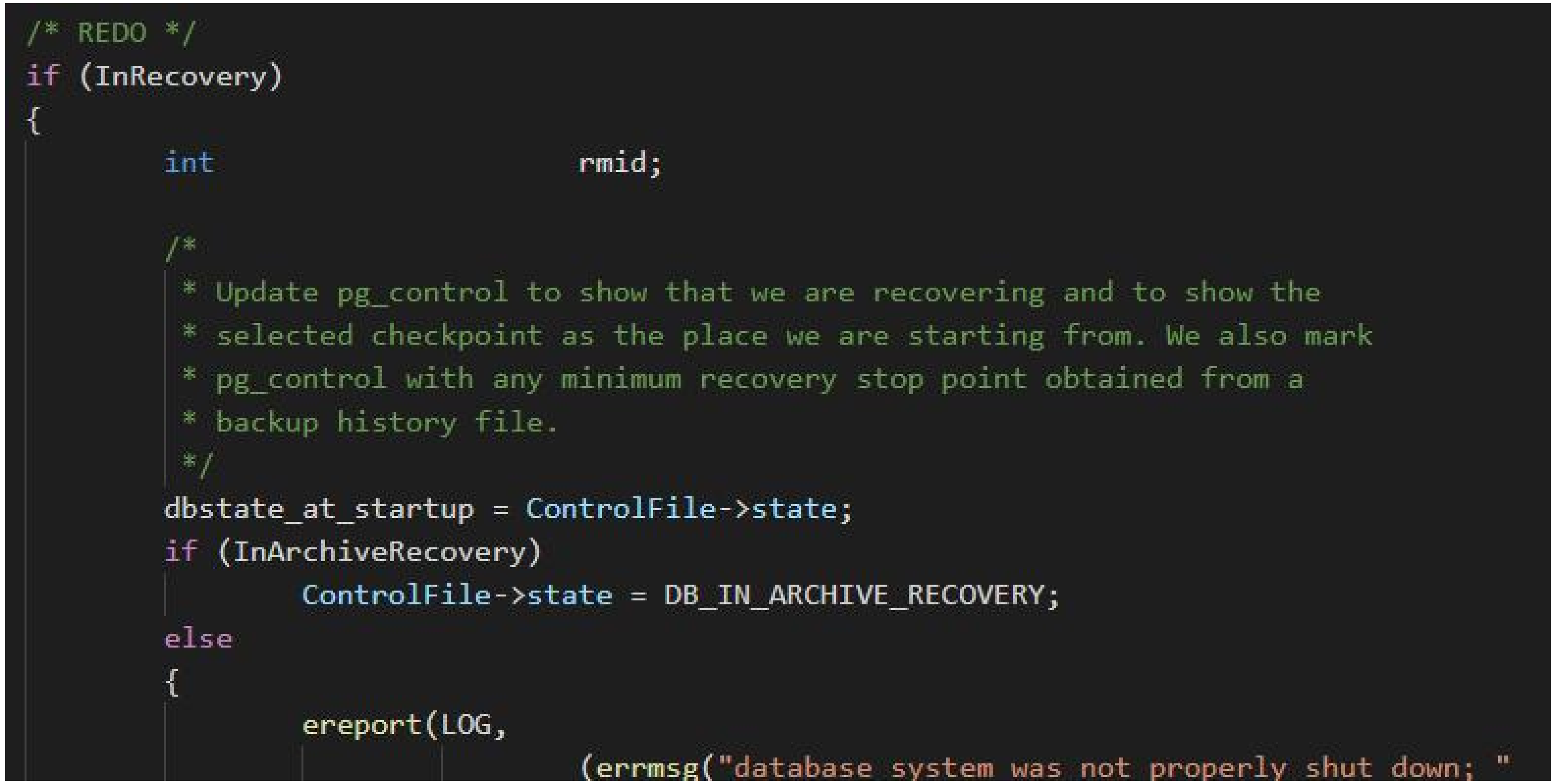
- PG的恢复在StartupXLOG()函数中。（xlog.c文件）

- 在此if之后，就是PG的恢复流程。

- 目标：

  并不直接修改StartupXLOG()

  复制它，另外做一个Non–FPW模块，



```
/* REDO */
if (InRecovery)
{
        int                     rmid;

        /*
         * Update pg_control to show that we are recovering and to show the
         * selected checkpoint as the place we are starting from. We also mark
         * pg_control with any minimum recovery stop point obtained from a
         * backup history file.
         */
        dbstate_at_startup = ControlFile->state;
        if (InArchiveRecovery)
                ControlFile->state = DB_IN_ARCHIVE_RECOVERY;
        else
        {
                ereport(LOG,
                                (errmsg("database system was not properly shut down; "
```

如果出现异常当库，先使用Non–FPW模块进行检查、恢复，之后再使用正常的PG启动数据库。

欢迎讨论 共同为国内开源社区建设贡献力量

VAGE_LV

THANKS