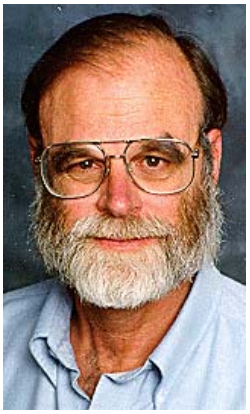

个人简介
专业打杂程序员
联系方式
新浪微博 腾讯微博

IT新闻:
苹果新Retina MacBook Pro（2014年中）开箱图+SSD简单测试 8分钟前
网吧里玩出的世界冠军 打场游戏赚了400万 10分钟前
Twitter收购深度学习创业公司Madbits 35分钟前

昵称：YY哥
园龄：7年2个月
粉丝：342
关注：2
+加关注

< 2009年3月 >

日	一	二	三	四	五	六
22	23	24	25	26	27	28
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

搜索
找找看
谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类
c/c++(9)
Linux相关(24)
MySQL(11)
Others(2)
Web技术(12)
数据结构与算法(15)
数据库技术(30)
系统相关(3)
云计算与虚拟化(3)

随笔档案
2014年7月 (4)

SQLite入门与分析(六)---再谈SQLite的锁

写在前面：SQLite封锁机制的实现需要底层文件系统的支持，不管是Linux，还是Windows，都提供了文件锁的机制，而这为SQLite提供了强大的支持。本节就来谈谈SQLite使用到的文件锁——主要基于Linux和Windows平台。

- Linux的文件锁

Linux 支持的文件锁技术主要包括建议锁（advisory lock）和强制锁（mandatory lock）这两种。此外，Linux 中还引入了两种强制锁的变种形式：共享模式强制锁（share-mode mandatory lock）和租借锁（lease）。在这里，主要讨论建议锁（advisory lock）。
建议锁并不由内核强制实行，也就是说如果有进程不遵守“游戏规则”，不检查目标文件是否已经由别的进程加了锁就往其中写入数据，那么内核是不会加以阻拦的。因此，建议锁并不能阻止进程对文件的访问，而只能依靠各个进程在访问文件之前检查该文件是否已经被其他进程加锁来实现并发控制。进程需要事先对锁的状态做一个约定，并根据锁的当前状态和相互关系来确定其他进程是否能对文件执行指定的操作。而强制锁是由内核强制采用的文件锁——由于内核对每个read()和write()操作都会检查相应的锁，所以会降低系统性能。
对于建议锁，Linux提供两种实现方式：锁文件（lock files）和记录锁（ record locking）。

(1)锁文件(lock files)
锁文件是最简单的对文件加锁的方法，每个需要加锁的数据文件都有一个锁文件（lock file）。当锁文件存在时，就认为该数据文件已经被加锁，别的进程不应该访问（但是你非要访问，Linux也不会阻止）。当锁不存在，进程就可以创建一个锁文件，然后访问相应的数据文件。只要创建锁的过程是原子的，就能保证某一时刻只有一个进程拥有该锁，这种方法保证某一时刻只有一个进程访问文件。
这种想法很简单，当一个进程想访问文件时，可以按如下方式对文件加锁：

```
fd = open("somefile.lck", O_RDONLY, 0644);
if (fd >= 0) {
    close(fd);
    printf("the file is already locked");
    return 1;
} else {
    /* the lock file does not exist, we can lock it and access it */
    fd = open("somefile.lck", O_CREAT | O_WRONLY, 0644");
    if (fd < 0) {
        perror("error creating lock file");
        return 1;
    }
    /* we could write our pid to the file */
    close(fd);
}
```

当一个进程处理完文件后，就可以调用unlink("somefile.lck")释放锁了——本质上是删除somefile.lck文件。
上面这段代码实际上存在竞争情况，原因在于if语句块不是原子性的，进入if语句块，内核可能调度别的进程运行。更好的方式如下：

```
fd = open("somefile.lck", O_WRONLY | O_CREAT | O_EXCL, 0644);
if (fd < 0 && errno == EEXIST) {
    printf("the file is already locked");
    return 1;
} else if (fd < 0) {
    perror("unexpected error checking lock");
    return 1;
}

/* we could write our pid to the file */
close(fd);
```

O_EXCL标志保证open()创建锁文件的过程是原子性的。
注意以下几点：
1、任何时刻只有一个进程可以拥有锁。
2、O_EXCL标志只对本志文件系统是可靠的，对于网络文件系统并不能很好的支持。
3、锁仅仅只是建议性的。
4、如果一个持有锁的进程不正常结束，锁文件仍然存在。如果加锁进程的pid存储在锁文件中，其它进程可以检查锁进程是否存

2014年3月 (1)
2013年9月 (1)
2013年8月 (1)
2013年2月 (1)
2012年11月 (4)
2012年1月 (1)
2011年12月 (1)
2011年10月 (1)
2011年3月 (1)
2010年9月 (1)
2010年8月 (1)
2010年7月 (3)
2010年6月 (2)
2010年5月 (7)
2010年4月 (1)
2010年3月 (1)
2010年1月 (1)
2009年12月 (2)
2009年10月 (2)
2009年9月 (14)
2009年8月 (4)
2009年6月 (14)
2009年5月 (3)
2009年4月 (1)
2009年3月 (3)
2009年2月 (11)
2008年10月 (7)
2008年8月 (5)
2008年7月 (1)
2008年6月 (2)
2008年5月 (2)
2008年4月 (5)

kernel

kernel中文社区
LDN
The Linux Document Project
The Linux Kernel Archives

manual

cppreference
gcc manual
mysql manual

sites

Database Journal
Fedora镜象
highscalability
KFUPM ePrints
Linux docs
Linux Journal
NoSQL
SQLite

技术社区

apache
CSDN
IBM-developerworks
lucene中国
nutch中国
oldlinux
oracle's forum

最新评论

1. Re:理解MySQL——架构与概念
我试验了下.数据 5 9 10 13 18begin;select * from asf_execution where num> 5 and num 5 and INSTANCE_ID_<18 lock in share mode;会有 1.行锁 2.间隙所 [5 18)插

在，当它结束时就可以删除锁。但是，在检查的时候，如果pid被其它进程使用了，此时就无能为力了。

(2)记录锁（Record Locking）
为了克服锁文件的缺点，System V和BSD4.3引入了记录锁，相应的系统调用为lockf()和flock()。而POSIX对于记录锁提供了另外一种机制，其系统调用为fcntl()。Linux提供三种接口，在这里仅讨论POSIX的接口。
记录锁和锁文件有两个很重要的区别：首先，记录锁可以对文件的任何一部分加锁——这对于DBMS这样的应用程序，有极大的帮助，SQLite当然没有放过这样的好处。其次，记录锁的另一个优点就是它由内核持有，而不是文件系统持有。当进程结束时，所有的锁也随之释放。
和锁文件一样，POSIX锁也是建议性的。记录锁有两种锁：读锁（read locks）和写锁（write locks）。读锁也就是共享锁（shared lock），写锁也就是排它锁（exclusive lock）。对于一个记录，只能有一个进程持有写锁，读锁不能存在。
对于一个进程本身而言，多个锁绝不会冲突。如果一个进程对文件的200-250字节持有读锁，然后对200-225字节数据加写锁，是会成功的。此时，200-225为写锁，而226-250字节数据为读锁，该规则主要是防止进程本身发生死锁（尽管多进程之间仍然可能发生死锁）。

POSIX锁通过fcntl()系统来实现，如下：

#include <fcntl.h>

int fcntl(int fd, int command, long arg);

arg为指向flock结构体的指针：

```
#include <fcntl.h>

struct flock {
    short l_type;
    short l_whence;
    off_t l_start;
    off_t l_len;
    pid_t l_pid;
};
```

在 flock 结构中，l_type 用来指明创建的是共享锁还是排他锁，其取值有三种：F_RDLCK（共享锁）、F_WRLCK（排他锁）和 F_UNLCK（删除之前建立的锁）；l_pid 指明了该锁的拥有者；l_whence、l_start 和l_end 这些字段指明了进程需要对文件的哪个区域进行加锁，这个区域是一个连续的字节集合。因此，进程可以对同一个文件的不同部分加不同的锁。l_whence 必须是 SEEK_SET、SEEK_CUR 或 SEEK_END 这几个值中的一个，它们分别对应着文件头、当前位置和文件尾。l_whence 定义了相对于 l_start 的偏移量，l_start 是从文件开始计算的。

可以执行的操作包括：

* F_GETLK：进程可以通过它来获取通过 fd 打开的那个文件的加锁信息。执行该操作时，lock 指向的结构中就保存了希望对文件加的锁（或者说要查询的锁）。如果确实存在这样一把锁，它阻止 lock 指向的 flock 结构所给出的锁描述符，则把现存的锁的信息写到 lock 指向的 flock 结构中，并将该锁拥有者的 PID 写入 l_pid 字段中，然后返回；否则，就将 lock 指向的 flock 结构中的 l_type 设置为 F_UNLCK，并保持 flock 结构中其他信息不变返回，而不会对该文件真正加锁。
* F_SETLK：进程用它来对文件的某个区域进行加锁（l_type的值为 F_RDLCK 或 F_WRLCK）或者删除锁（l_type 的值为F_UNLCK），如果有其他锁阻止该锁被建立，那么 fcntl() 就出错返回
* F_SETLKW：与 F_SETLK 类似，唯一不同的是，如果有其他锁阻止该锁被建立，则调用进程进入睡眠状态，等待该锁释放。一旦这个调用开始了等待，就只有在能够进行加锁或者收到信号时才会返回

需要注意的是，F_GETLK 用于测试是否可以加锁，在 F_GETLK 测试可以加锁之后，F_SETLK 和 F_SETLKW 就会企图建立一把锁，但是这两者之间并不是一个原子操作，也就是说，在 F_SETLK 或者 F_SETLKW 还没有成功加锁之前，另外一个进程就有可能已经插进来加上了一把锁。而且，F_SETLKW 有可能导致程序长时间睡眠。还有，程序对某个文件拥有的各种锁会在相应的文件描述符被关闭时自动清除，程序运行结束后，其所加的各种锁也会自动清除。

- Windows中的文件锁

Windows中的锁都是强制锁(mandatory locks)，Windows中的共享文件通过以下几个机制来管理：
(1) 通过共享访问控制方式，应用程序可以指定整个文件进行共享读，写或者删除。
(2) 通过字节范围锁（byte range locks）可以对文件的某一部分进行读写访问。
(3) Windows文件系统不允许正在执行的文件被打开用来进行写或删除操作。
文件的共享方式由WIN32 API中的打开文件函数CreateFile()中的sharing mode参数确定：

```
HANDLE CreateFile(
  LPCTSTR lpFileName,
  DWORD dwDesiredAccess,
  DWORD dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD dwCreationDisposition,
```

入INSERT I.....
--麒麟飞
2. Re:理解MySQL——架构与概念
例1-5
insert into t(i) values(1);
这句话应该是可以插入的.
不会被阻塞
--麒麟飞
3. Re:理解MySQL——架构与概念
注：SELECT ... FOR UPDATE仅在自动提交关闭(即手动提交)时才会对元组加锁，而在自动提交时，符合条件的元组不会被加锁。

这个是错误的.自动提交的,也会尝试获取排它锁.
你可以试验下.
--麒麟飞
4. Re:浅谈mysql的两阶段提交协议
YY哥 偶像啊!细腻文笔 配有说服力的代码和图 我崇拜你 !!!
之前sqlite的深入分析帮了我大忙..现在做mysql相关 有来你的博客找东西 哈哈哈!!
--hark.perfe
5. Re:(i++)+(i++)与(++i)+(++i)
@arrowcat
这类语句本身没什么意义，但是楼主思考的角度让我豁然开朗。
--HJWAJ


```
  DWORD dwFlagsAndAttributes,
  HANDLE hTemplateFile
);
```

dwShareMode的取值通常为：
FILE_SHARE_DELETE：
Enables subsequent open operations on an object to request delete access.
Otherwise, other processes cannot open the object if they request delete access.
If this flag is not specified, but the object has been opened for delete access, the function fails.
FILE_SHARE_READ：
Enables subsequent open operations on an object to request read access.
Otherwise, other processes cannot open the object if they request read access.
If this flag is not specified, but the object has been opened for read access, the function fails.
FILE_SHARE_WRITE：
Enables subsequent open operations on an object to request write access.
Otherwise, other processes cannot open the object if they request write access.
If this flag is not specified, but the object has been opened for write access, the function fails.

具体的实现函数：

```
BOOL LockFile(
  HANDLE hFile,
  DWORD dwFileOffsetLow,
  DWORD dwFileOffsetHigh,
  DWORD nNumberOfBytesToLockLow,
  DWORD nNumberOfBytesToLockHigh
);
```

- SQLite封锁机制的几个注意点

SQLite的lock byte的定义如下：

```
#define PENDING_BYTE      0x40000000  /* First byte past the 1GB boundary */
#define RESERVED_BYTE     (PENDING_BYTE+1)
#define SHARED_FIRST      (PENDING_BYTE+2)
#define SHARED_SIZE       510
```

(1)PENDING_BYTE为何设置为0X4000 0000（1GB）？
在Windows文件中，被加锁的区域不要求有数据，并且它会阻止所有的进程写文件的该区域,包括第一个持有该锁的进程.为了防止出现由于对含有mandatory lock的页面进行读写操作而出现错误(这在Windows中是不允许的),SQLite完全忽略包含pending byte的页面,所以pending byte在数据库文件上产生一个"文件洞"。PENDING_BYTE设置得那么高,则大部分数据库文件不会遇到由于PENDING_BYTE产生"文件洞"引起的空间损失(除非文件特别大,超过1GB)。

(2)对于Windows来说，文件中加锁的区域不能重叠，为了使两个读进程可以同时访问文件，对于SHARED LOCK选择一个SHARED_FIRST——SHARED_FIRST+ SHARED_SIZE范围内的随机数，所以有可能两个进程取得一样的lock byte，所以对于Windows，SQLite的并发性就受到限制。

分类: 数据库技术

YY哥
关注 - 2
粉丝 - 342
+加关注

« 上一篇：SQLite入门与分析(五)---Page Cache之并发控制
» 下一篇：SQLite入门与分析(七)---浅谈SQLite的虚拟机

posted @ 2009-03-10 21:54 YY哥 阅读(6208) 评论(4) 编辑 收藏

评论列表

#1楼 2009-03-10 22:07 Soli

Not bad.

先评论再细看。

支持(0) 反对(0)

#2楼[楼主] 2009-03-10 22:09 YY哥

@Soli
谢谢支持~

支持(0) 反对(0)

#3楼 2009-03-10 22:53 ajax+u[未注册用户]

我只关心sqlite 虚拟机 希望能讲到

#4楼 2009-04-21 18:18 lgl_newbie[未注册用户]

@arrowcat
先感谢lz的分享，经过你的分析我对sqlite有了更深的认识。我现在在做一个项目，用到了sqlite，具体是在linux下用C语言操作sqlite。现在就是解决不了多进程访问sqlite，貌似用到sqlite3_busy_handler()和sqlite3_busy_timeout()函数，但是具体不咋会用这俩函数，能不能具体结合例子讲讲这两个函数的使用方法。谢谢！