# 文件的access time

韦体东

## 三种文件时间

文件时间有三种，分别为：

1. Access - the last time the file was read 读文件会造成修改

st\_atime

Time when file data was last accessed. Changed by the

following functions: creat(), mknod(), pipe(), utime(2), and read(2).

2. Modify - the last time the file was modified (content has been modified) 写文件会造成修改

st\_mtime

Time when data was last modified. Changed by the fol-

lowing functions: creat(), mknod(), pipe(), utime(), and write(2).

3. Change - the last time meta data of the file was changed (e.g. permissions) - 写文件或者修改属性或造成修改

st\_ctime

Time when file status was last changed. time of last modification of file status information.Changed by the

following functions: chmod(), chown(), creat(),

link(2), mknod(), pipe(), unlink(2), utime(), and write().

## criticism of atime

mtime和ctime容易理解，大体来说，前者是写了文件，就会更新，后者是修改了文件属性，就会更新。

特别一点的是atime，虽说，应该是有read便会更新，但是想想：

1. 在只读文件系统上，如何更新文件属性？

2. 也许只是读一个小文件，却偏偏要伴随一次写操作(更新atime)，要知道，写操作一般都比读更耗时，这时候更新atime，是不是有点本末倒置？

这就是所谓的criticism of atime： <https://en.wikipedia.org/wiki/Stat_%28system_call%29#Criticism_of_atime>

## mount属性对atime的影响

鉴于criticism of atime问题，linux专门实现了如下mount属性，用于减少此问题影响。

1. strictatime - 严格执行POSIX标准，有read就会更新atime。

我的根分区之前是noatime挂载的，使用strictatime重现挂载：

修改为strictatime试试：

localhost / # mount -o remount,rw,strictatime,data=ordered /

先用touch重置一下所有的时间：

localhost / # touch a.txt; stat a.txt

File: 'a.txt'

Size: 5 Blocks: 8 IO Block: 4096 regular file

Device: fe02h/65026d Inode: 13 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-09 11:32:54.665000000 +0800

Modify: 2017-08-09 11:32:54.665000000 +0800

Change: 2017-08-09 11:32:54.665000000 +0800

执行读操作：

localhost / # cat a.txt

abcd

再次查看，发现atime更新了：

localhost / # stat a.txt

File: 'a.txt'

Size: 5 Blocks: 8 IO Block: 4096 regular file

Device: fe02h/65026d Inode: 13 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-09 11:32:58.593000000 +0800

Modify: 2017-08-09 11:32:54.665000000 +0800

Change: 2017-08-09 11:32:54.665000000 +0800

再次执行cat a.txt，atime会继续更新，strictatime模式，是严格执行POSIX标准的，性能也最差。

2. relatime - 只满足(atime <= mtime) or (atime <= ctime) or (now - last atime) > 24 hours 的时候才更新atime

我用remount方式添加relatime无效，找了一个本来就是relatime方式挂载的目录验证：

localhost run # cat /proc/mounts | grep /run

none /run tmpfs rw,nosuid,nodev,relatime,mode=755 0 0

先用touch重置一下所有的时间：

localhost run # touch a.txt; stat a.txt

File: 'a.txt'

Size: 0 Blocks: 0 IO Block: 4096 regular empty file

Device: 10h/16d Inode: 26181439 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 14:51:38.610000000 +0800

Modify: 2017-08-10 14:51:38.610000000 +0800

Change: 2017-08-10 14:51:38.610000000 +0800

构造条件（当前条件已经满足，因为mtime=atime=ctime），我准备构造mtime>=atime，write操作会导致mtime和ctime同时被更新，trunc方式打开文件也会导致mtime和ctime同时被更新：

localhost run # echo a >> a.txt

localhost run # stat a.txt

File: 'a.txt'

Size: 2 Blocks: 8 IO Block: 4096 regular file

Device: 10h/16d Inode: 26181439 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 14:51:38.610000000 +0800

Modify: 2017-08-10 14:51:59.250000000 +0800

Change: 2017-08-10 14:51:59.250000000 +0800

此时读取文件，mtime>=atime条件满足，atime被更新：

localhost run # cat a.txt

a

localhost run # stat a.txt

File: 'a.txt'

Size: 2 Blocks: 8 IO Block: 4096 regular file

Device: 10h/16d Inode: 26181439 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 14:52:15.146000000 +0800

Modify: 2017-08-10 14:51:59.250000000 +0800

Change: 2017-08-10 14:51:59.250000000 +0800

如上，此时atime大于其他时间，也不在过去24小时，任何条件都不满足，所以再次cat不会更新atime：

localhost run # cat a.txt

a

localhost run # stat a.txt

File: 'a.txt'

Size: 2 Blocks: 8 IO Block: 4096 regular file

Device: 10h/16d Inode: 26181439 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 14:52:15.146000000 +0800

Modify: 2017-08-10 14:51:59.250000000 +0800

Change: 2017-08-10 14:51:59.250000000 +0800

Birth: -

我们再来看看，目录的atime：

localhost run # touch abc; stat abc

File: 'abc'

Size: 40 Blocks: 0 IO Block: 4096 directory

Device: 10h/16d Inode: 26508860 Links: 2

Access: (0755/drwxr-xr-x) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 16:13:47.670000000 +0800

Modify: 2017-08-10 16:13:47.670000000 +0800

Change: 2017-08-10 16:13:47.670000000 +0800

ls之类的命令，会修改atime

localhost run # ls abc; stat abc

File: 'abc'

Size: 40 Blocks: 0 IO Block: 4096 directory

Device: 10h/16d Inode: 26508860 Links: 2

Access: (0755/drwxr-xr-x) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 16:13:51.949000000 +0800

Modify: 2017-08-10 16:13:47.670000000 +0800

Change: 2017-08-10 16:13:47.670000000 +0800

3. nodiratime - 不更新目录的atime，但更新文件的atime

mount一个nodiratime的挂载目录：

Localhost / # mkdir nodir; mount -o rw,nodiratime,data=ordered /cfg /nodir; mount | grep /nodir

/cfg on /nodir type ext3 (rw,nodiratime,relatime,errors=continue,user\_xattr,acl,barrier=1,data=ordered)

奇怪的是，我虽然只指定了nodiratime，但是mount命令看到的却多了一个relatime，根据后面我们推测，应该是这个relatime管控文件，nodiratime只管目录。

创建目录和文件：

Localhost /nodir # mkdir abc; stat abc

File: abc

Size: 1024 Blocks: 2 IO Block: 1024 directory

Device: 701h/1793d Inode: 21738 Links: 2

Access: (2755/drwxr-sr-x) Uid: ( 0/ root) Gid: ( 50/ staff)

Access: 2017-08-09 16:21:37.000000000

Modify: 2017-08-09 16:21:37.000000000

Change: 2017-08-09 16:21:37.000000000

Localhost /nodir # echo a > abc/a.txt

Localhost /nodir # stat abc/a.txt

File: abc/a.txt

Size: 2 Blocks: 2 IO Block: 1024 regular file

Device: 701h/1793d Inode: 21739 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 50/ staff)

Access: 2017-08-09 16:21:48.000000000

Modify: 2017-08-09 16:21:48.000000000

Change: 2017-08-09 16:21:48.000000000

往目录里面新增了文件，类似于对目录进行了write，此时mtime和ctime都会更新：

Localhost /nodir # stat abc

File: abc

Size: 1024 Blocks: 2 IO Block: 1024 directory

Device: 701h/1793d Inode: 21738 Links: 2

Access: (2755/drwxr-sr-x) Uid: ( 0/ root) Gid: ( 50/ staff)

Access: 2017-08-09 16:21:37.000000000

Modify: 2017-08-09 16:21:48.000000000

Change: 2017-08-09 16:21:48.000000000

访问一下目录，发现目录的atime并没有更新，这点可对比上面针对relatime做的试验(relatime的时候，此时ls会导致目录atime更新)：

Localhost /nodir # touch abc; sleep 1; ls abc; stat abc

File: abc

Size: 1024 Blocks: 2 IO Block: 1024 directory

Device: 701h/1793d Inode: 21738 Links: 2

Access: (2755/drwxr-sr-x) Uid: ( 0/ root) Gid: ( 50/ staff)

Access: 2017-08-09 16:22:21.000000000

Modify: 2017-08-09 16:22:21.000000000

Change: 2017-08-09 16:22:21.000000000

而针对文件，我们读取一次文件，会更新atime，和relatime一样，此时再次cat不会更新atime，所以，我怀疑nodirtime后面跟了一个relatime属性，这个relatime是文件的更新标志：

Localhost /nodir # cat abc/a.txt

a

Localhost /nodir # stat abc/a.txt

File: abc/a.txt

Size: 2 Blocks: 2 IO Block: 1024 regular file

Device: 701h/1793d Inode: 21739 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 50/ staff)

Access: 2017-08-09 16:22:37.000000000

Modify: 2017-08-09 16:21:48.000000000

Change: 2017-08-09 16:21:48.000000000

为了验证这个想法，我们再找一个，不是relatime和nodiratime组合的，而是noatime和nodiratime组合的目录试验：

Localhost /sf/cfg # mount | grep /sf/cfg

/cfg2 on /sf/cfg type ext3 (rw,noatime,nodiratime,errors=continue,user\_xattr,acl,barrier=1,data=ordered)

ls后abc的atime并没有更新，说明，nodiratime生效：

Localhost /sf/cfg # mkdir abc; sleep 1; ls abc; stat abc

File: abc

Size: 1024 Blocks: 2 IO Block: 1024 directory

Device: 700h/1792d Inode: 21738 Links: 2

Access: (2755/drwxr-sr-x) Uid: ( 0/ root) Gid: ( 50/ staff)

Access: 2017-08-09 16:36:33.000000000

Modify: 2017-08-09 16:36:33.000000000

Change: 2017-08-09 16:36:33.000000000

读取文件，文件的atime并没有更新，说明noatime生效，和猜想吻合：

Localhost /sf/cfg # touch a.txt; sleep 1; cat a.txt; stat a.txt

File: a.txt

Size: 2 Blocks: 2 IO Block: 1024 regular file

Device: 700h/1792d Inode: 21739 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 50/ staff)

Access: 2017-08-09 16:36:47.000000000

Modify: 2017-08-09 16:36:47.000000000

Change: 2017-08-09 16:36:47.000000000

4. noatime - 根本就不更新atime，如果没有主动修改(例如touch)，atime就是文件创建的时候的时间

我找根分区试验，此分区就是noatime挂载的：

localhost / # mount | grep /dev/vda2

/dev/vda2 on / type ext3 (rw,noatime,data=ordered)

目录，无更新：

localhost / # rm -rf abc; mkdir abc; sleep 1; ls abc; stat abc

File: 'abc'

Size: 4096 Blocks: 8 IO Block: 4096 directory

Device: fe02h/65026d Inode: 1708785 Links: 2

Access: (0755/drwxr-xr-x) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 17:03:57.477000000 +0800

Modify: 2017-08-10 17:03:57.477000000 +0800

Change: 2017-08-10 17:03:57.477000000 +0800

文件，无更新：

localhost / # rm -f a.txt; touch a.txt; sleep 1; cat a.txt; stat a.txt

File: 'a.txt'

Size: 0 Blocks: 0 IO Block: 4096 regular empty file

Device: fe02h/65026d Inode: 13 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2017-08-10 17:07:17.990000000 +0800

Modify: 2017-08-10 17:07:17.990000000 +0800

Change: 2017-08-10 17:07:17.990000000 +0800

5. lazytime - 先缓存在内存中，等到一定时间或者sync操作的时候，一起刷新atime，这种方式兼容POSIX，只是有一定延迟

这功能在kernel 4.0才有，没有验证具体效果。

## 使用GDB验证

使用GDB一条一条命令调用，相对而言控制粒度要细一些，我通过类似测试，发现write操作会同时更新mtime和ctime，trunc方式打开会也是。下面记录几个验证步骤：

(gdb) call open("/var/lib/docker/image/overlay2/layerdb/sha256/b.txt", O\_WRONLY|O\_CREAT|O\_APPEND, 0666)

No symbol "O\_WRONLY" in current context.

宏不能再gdb里面使用，这几宏可以在头文件中搜索：

#define O\_RDONLY 00

#define O\_WRONLY 01

#define O\_RDWR 02

#define O\_CREAT 0100

#define O\_TRUNC 01000

#define O\_APPEND 02000

八进制的O\_WRONLY|O\_CREAT|O\_APPEND，转换为十六进制为0x441

八进制的O\_WRONLY|O\_CREAT|O\_TRUNC，转换为十六进制为0x241

(gdb) call open("/var/lib/docker/image/overlay2/layerdb/sha256/b.txt", 0x441, 0666)

$2 = 8

返回的是句柄号8，往其中写入5个字符：

(gdb) call write(8,"test",5)

$3 = 5

关闭句柄：

(gdb) call close(8)

$4 = 0

如果是读打开，验证读操作前，需要先为其分配内存，返回的是malloc地址：

(gdb) call (char\*)malloc(10)

$11 = 0x2306c70 ""

设置读偏移，并提供一个10字节的堆地址0x2306c70用于存储数据：

(gdb) call lseek(9,0,0)

$13 = 0

(gdb) call read(9, (char\*)0x2306c70, 10)

$14 = 4

2017/8/9