**linux 高效的文件系统事件监控 内核级解析方案 inotify**

[环境配置](https://www.lvtao.net/sort/config/) memory发布于June 27, 2013 标签: [Linux](https://www.lvtao.net/tag/linux/), [inotify](https://www.lvtao.net/tag/inotify/)

安装inotify-tools (http://inotify-tools.sourceforge.net) 下载源码包

wget http://github.com/downloads/rvoicilas/inotify-tools/inotify-tools-3.14.tar.gz

tar zxvf inotify-tools-3.14.tar.gz

cd inotify-tools-3.14

./configure --prefix=/usr && make && su -c 'make install'

其它的一些相关软件推荐 https://github.com/rvoicilas/inotify-tools/wiki#related-software

出现这个错误“/usr/local/bin/inotifywait: error while loading shared libraries: libinotifytools.so.0”可以采用以下办法解决：

ln -sv /usr/local/lib/libinotify\* /usr/lib/

ln -s /usr/local/lib/libinotifytools.so.0 /usr/lib64/libinotifytools.so.0

cp /usr/lib/libinotifytools.so.0 /usr/local/lib/

#!/bin/bash

#filename watchdir.sh

path=$1

/usr/local/bin/inotifywait -mrq --timefmt '%d/%m/%y/%H:%M' --format '%T %w %f' -e modify,delete,create,attrib $path

修改/etc/sysctl.conf，

增加fs.inotify.max\_user\_watches=10485760

sysctl -p

inotify报错upper limit on inotify watches reached  
在对一个大磁盘进行inotify监听时，爆出以上错误  
cat一下这个文件，默认值是8192，echo 8192000 > /proc/sys/fs/inotify/max\_user\_watches即可~  
Inotify 可监视的文件系统事件  
    IN\_ACCESS : 即文件被访问  
    IN\_MODIFY : 文件被 write  
    IN\_ATTRIB : 文件属性被修改，如 chmod、chown、touch 等  
    IN\_CLOSE\_WRITE : 可写文件被 close  
    IN\_CLOSE\_NOWRITE : 不可写文件被 close  
    IN\_OPEN : 文件被open  
    IN\_MOVED\_FROM : 文件被移走,如 mv  
    IN\_MOVED\_TO : 文件被移来，如 mv、cp  
    IN\_CREATE : 创建新文件  
    IN\_DELETE : 文件被删除，如 rm  
    IN\_DELETE\_SELF : 自删除，即一个可执行文件在执行时删除自己  
    IN\_MOVE\_SELF : 自移动，即一个可执行文件在执行时移动自己  
    IN\_UNMOUNT : 宿主文件系统被 umount  
    IN\_CLOSE : 文件被关闭，等同于(IN\_CLOSE\_WRITE | IN\_CLOSE\_NOWRITE)  
    IN\_MOVE : 文件被移动，等同于(IN\_MOVED\_FROM | IN\_MOVED\_TO)

注：上面所说的文件也包括目录  
Inotify内核版本支持  
从kernel 2.6.13开始,Inotify正式并入内核，RHEL5已经支持.  
看看是否有 /proc/sys/fs/inotify/目录，以确定内核是否支持inotify

#ls -l /proc/sys/fs/inotify/

total 0

-rw-r--r-- 1 root root 0 Oct 9 09:36 max\_queued\_events

-rw-r--r-- 1 root root 0 Oct 9 09:36 max\_user\_instances

-rw-r--r-- 1 root root 0 Oct 9 09:36 max\_user\_watches

inotify 的默认内核参数  
/proc/sys/fs/inotify/max\_queued\_events 默认值: 16384 该文件中的值为调用inotify\_init时分配给inotify instance中可排队的event的数目的最大值，超出这个值得事件被丢弃，但会触发IN\_Q\_OVERFLOW事件  
/proc/sys/fs/inotify/max\_user\_instances 默认值: 128 指定了每一个real user ID可创建的inotify instatnces的数量上限  
/proc/sys/fs/inotify/max\_user\_watches 默认值: 8192 指定了每个inotify instance相关联的watches的上限  
注意: max\_queued\_events 是 Inotify 管理的队列的最大长度，文件系统变化越频繁，这个值就应该越大  
如果你在日志中看到Event Queue Overflow，说明max\_queued\_events太小需要调整参数后再次使用.

inotifywait 仅执行阻塞，等待 inotify 事件。您可以监控任何一组文件和目录，或监控整个目录树（目录、子目录、子目录的子目录等等）  
在 shell 脚本中使用 inotifywait。  
inotifywatch 收集关于被监视的文件系统的统计数据，包括每个 inotify 事件发生多少次。

 shell脚本示例

vi /tmp/test.sh

#!/bin/bash

inotifywait -mrq --timefmt '%d/%m/%y %H:%M' --format '%T %w%f %e' --event modify,delete,create,attrib /home/admin | while read date time file event

do

case $event in

MODIFY|CREATE|MOVE|MODIFY,ISDIR|CREATE,ISDIR|MODIFY,ISDIR)

echo $event'-'$file

;;

MOVED\_FROM|MOVED\_FROM,ISDIR|DELETE|DELETE,ISDIR)

echo $event'-'$file

;;

esac

done

执行脚本，结果输出(这里测试删除了一个目录 rm -fr cronolog-1.6.2.bak)  
/tmp/test.sh  
DELETE-/home/admin/cronolog-1.6.2.bak/COPYING

我自己使用的一个实例

inotifywait -mrq --timefmt '%Y%m%d %H:%M:%S' --format '%T %e %w %f' -e modify,attrib,move,close\_write,create,delete /root/ >/home/inotify.log

C语言版

#include <unistd.h>

#include <sys/inotify.h>

#include <stdio.h>

#include <error.h>

#include <errno.h>

#include <string.h>

#define ERROR(text) error(1, errno, "%s", text)

struct EventMask {

int flag;

const char \*name;

};

int freadsome(void \*dest, size\_t remain, FILE \*file)

{

char \*offset = (char\*)dest;

while (remain) {

int n = fread(offset, 1, remain, file);

if (n==0) {

return -1;

}

remain -= n;

offset += n;

}

return 0;

}

int main(int argc, char \*argv[])

{

const char \*target;

if (argc == 1) {

target = ".";

} else {

target = argv[1];

}

EventMask event\_masks[] = {

{IN\_ACCESS , "IN\_ACCESS"} ,

{IN\_ATTRIB , "IN\_ATTRIB"} ,

{IN\_CLOSE\_WRITE , "IN\_CLOSE\_WRITE"} ,

{IN\_CLOSE\_NOWRITE , "IN\_CLOSE\_NOWRITE"} ,

{IN\_CREATE , "IN\_CREATE"} ,

{IN\_DELETE , "IN\_DELETE"} ,

{IN\_DELETE\_SELF , "IN\_DELETE\_SELF"} ,

{IN\_MODIFY , "IN\_MODIFY"} ,

{IN\_MOVE\_SELF , "IN\_MOVE\_SELF"} ,

{IN\_MOVED\_FROM , "IN\_MOVED\_FROM"} ,

{IN\_MOVED\_TO , "IN\_MOVED\_TO"} ,

{IN\_OPEN , "IN\_OPEN"} ,

{IN\_DONT\_FOLLOW , "IN\_DONT\_FOLLOW"} ,

{IN\_EXCL\_UNLINK , "IN\_EXCL\_UNLINK"} ,

{IN\_MASK\_ADD , "IN\_MASK\_ADD"} ,

{IN\_ONESHOT , "IN\_ONESHOT"} ,

{IN\_ONLYDIR , "IN\_ONLYDIR"} ,

{IN\_IGNORED , "IN\_IGNORED"} ,

{IN\_ISDIR , "IN\_ISDIR"} ,

{IN\_Q\_OVERFLOW , "IN\_Q\_OVERFLOW"} ,

{IN\_UNMOUNT , "IN\_UNMOUNT"} ,

};

int monitor = inotify\_init();

if ( -1 == monitor ) {

ERROR("monitor");

}

int watcher = inotify\_add\_watch(monitor, target, IN\_ALL\_EVENTS);

if ( -1 == watcher ) {

ERROR("inotify\_add\_watch");

}

FILE \*monitor\_file = fdopen(monitor, "r");

char last\_name[1024];

char name[1024];

/\* event:inotify\_event -> name:char[event.len] \*/

while (true) {

inotify\_event event;

if ( -1 == freadsome(&event, sizeof(event), monitor\_file) ) {

ERROR("freadsome");

}

if (event.len) {

freadsome(name, event.len, monitor\_file);

} else {

sprintf(name, "FD: %d\n", event.wd);

}

if (strcmp(name, last\_name) != 0) {

puts(name);

strcpy(last\_name, name);

}

/\* 显示event的mask的含义 \*/

for (int i=0; i<sizeof(event\_masks)/sizeof(EventMask); ++i) {

if (event.mask & event\_masks[i].flag) {

printf("\t%s\n", event\_masks[i].name);

}

}

}

return 0;

}

inotify相关参数：  
/proc/sys/fs/inotify/max\_queued\_events  #请求events数的最大值  
/proc/sys/fs/inotify/max\_user\_instances #每个user可创建的instances数量上限  
/proc/sys/fs/inotify/max\_user\_watches   #可监控的目录最大数

常用参数：  
--timefmt 时间格式  
%y年 %m月 %d日 %H小时 %M分钟  
--format 输出格式  
%T时间 %w路径 %f文件名 %e状态  
-m 始终保持监听状态，默认触发事件即退出。  
-r 递归查询目录  
-q 打印出监控事件  
-e 定义监控的事件，可用参数：  
open   打开文件 attrb  属性变更

access文件读取

modify文件更改。

attrib文件属性更改，如权限，时间戳等。

close\_write以可写模式打开的文件被关闭，不代表此文件一定已经写入数据。

close\_nowrite以只读模式打开的文件被关闭。

close文件被关闭，不管它是如何打开的。

open文件打开。

moved\_to一个文件或目录移动到监听的目录，即使是在同一目录内移动，此事件也触发。

moved\_from一个文件或目录移出监听的目录，即使是在同一目录内移动，此事件也触发。

move包括moved\_to和 moved\_from

move\_self文件或目录被移除，之后不再监听此文件或目录。

create文件或目录创建

delete文件或目录删除

delete\_self文件或目录移除，之后不再监听此文件或目录

unmount文件系统取消挂载，之后不再监听此文件系统。  
示例： inotifywait -mrq -e modify,create --timefmt '%y-%m-%d %H:%M:%S' --format '%T %f %e %w' /home/wwwroot/

inotify-tools提供两种工具，一是inotifywait，它是用来监控文件或目录的变化，二是inotifywatch，它是用来统计文件系统访问的次数  
**inotifywatch**

1、统计/home文件系统的事件

inotifywatch -v -e access -e modify -t 60 -r /home

**inotifywait参数说明**

语法：

inotifywait [-hcmrq] [-e ] [-t ] [--format ] [--timefmt ] [ ... ]

参数：

-h,–help

输出帮助信息

@

排除不需要监视的文件，可以是相对路径，也可以是绝对路径。

–fromfile

从文件读取需要监视的文件或排除的文件，一个文件一行，排除的文件以@开头。

-m, –monitor

接收到一个事情而不退出，无限期地执行。默认的行为是接收到一个事情后立即退出。

-d, –daemon

跟–monitor一样，除了是在后台运行，需要指定–outfile把事情输出到一个文件。也意味着使用了–syslog。

-o, –outfile

输出事情到一个文件而不是标准输出。

-s, –syslog

输出错误信息到系统日志

-r, –recursive

监视一个目录下的所有子目录。

-q, –quiet

指定一次，不会输出详细信息，指定二次，除了致命错误，不会输出任何信息。

–exclude

正则匹配需要排除的文件，大小写敏感。

–excludei

正则匹配需要排除的文件，忽略大小写。

-t , –timeout

设置超时时间，如果为0，则无限期地执行下去。

-e , –event

指定监视的事件。

-c, –csv

输出csv格式。

–timefmt

指定时间格式，用于–format选项中的%T格式。

–format

指定输出格式。

%w 表示发生事件的目录

%f 表示发生事件的文件

%e 表示发生的事件

%Xe 事件以“X”分隔

%T 使用由–timefmt定义的时间格式

**inotifywatch**

语法：

inotifywatch [-hvzrqf] [-e ] [-t ] [-a ] [-d ] [ ... ]

参数：

-h, –help

输出帮助信息

-v, –verbose

输出详细信息

@

排除不需要监视的文件，可以是相对路径，也可以是绝对路径。

–fromfile

从文件读取需要监视的文件或排除的文件，一个文件一行，排除的文件以@开头。

-z, –zero

输出表格的行和列，即使元素为空

–exclude

正则匹配需要排除的文件，大小写敏感。

–excludei

正则匹配需要排除的文件，忽略大小写。

-r, –recursive

监视一个目录下的所有子目录。

-t , –timeout

设置超时时间

-e , –event

只监听指定的事件。

-a , –ascending

以指定事件升序排列。

-d , –descending

以指定事件降序排列。  
2014-10-09更新

**inotify相关参数**

inotify定义了下列的接口参数，可以用来限制inotify消耗kernel memory的大小。由于这些参数都是内存参数，因此，可以根据应用需求，实时的调节其大小：

* /proc/sys/fs/inotify/max\_queued\_evnets表示调用inotify\_init时分配给inotify instance中可排队的event的数目的最大值，超出这个值的事件被丢弃，但会触发IN\_Q\_OVERFLOW事件。
* /proc/sys/fs/inotify/max\_user\_instances表示每一个real user ID可创建的inotify instatnces的数量上限。
* /proc/sys/fs/inotify/max\_user\_watches表示每个inotify instatnces可监控的最大目录数量。如果监控的文件数目巨大，需要根据情况，适当增加此值的大小。

根据以上在32位或者64位系统都可以执行：

echo 104857600 > /proc/sys/fs/inotify/max\_user\_watches

echo 'echo 104857600 > /proc/sys/fs/inotify/max\_user\_watches' >> /etc/rc.local

如果遇到以下错误：

inotifywait: error while loading shared libraries: libinotifytools.so.0: cannot open shared object file: No such file or directory

**解决方法：** 32位系统：ln -s /usr/local/lib/libinotifytools.so.0 /usr/lib/libinotifytools.so.0

64位系统：ln -s /usr/local/lib/libinotifytools.so.0 /usr/lib64/libinotifytools.so.0

**inotifywait命令使用**

#!/bin/bash

#filename watchdir.sh

path=$1

/usr/local/bin/inotifywait -mrq --timefmt '%d/%m/%y/%H:%M' --format '%T %w %f' -e modify,delete,create,attrib $path

**inotifywait命令参数**

* -m是要持续监视变化。
* -r使用递归形式监视目录。
* -q减少冗余信息，只打印出需要的信息。
* -e指定要监视的事件列表。
* --timefmt是指定时间的输出格式。
* --format指定文件变化的详细信息。

**可监听的事件**

|  |  |
| --- | --- |
| 事件 | 描述 |
| access | **访问**，读取文件。 |
| modify | **修改**，文件内容被修改。 |
| attrib | **属性**，文件元数据被修改。 |
| move | **移动**，对文件进行移动操作。 |
| create | **创建**，生成新文件 |
| open | **打开**，对文件进行打开操作。 |
| close | **关闭**，对文件进行关闭操作。 |
| delete | **删除**，文件被删除。 |