

[RFC][PATCH] inotify 0.11.0

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-09/8586.html>

From: John McCutchan (ttb_at_tentacle.dhs.org)

Date: 09/29/04

To: linux-kernel@vger.kernel.org, gamin-list@gnome.org, rml@ximian.com, viro@parcellarce.linux.th

Date: Tue, 28 Sep 2004 18:28:09 -0400

Hello,

Here is release 0.11.0 of inotify. Attached is a patch to 2.6.8.1

--New in this version--

- remove timer (rml)
- fix typo (rml)
- remove check for dev->file_private (rml)
- redo find_inode (rml)
- use the bitmap functions (rml)
- modularization (rml)
- misc cleanup (rml,me)
- redo inotify_read (me)

John McCutchan

Release notes:

--Why Not dnotify and Why inotify (By Robert Love)--

Everyone seems quick to deride the blunder known as "dnotify" and applaud a replacement, any replacement, man anything but that current mess, but in the name of fairness I present my treatise on why dnotify is what one might call not good:

- * dnotify requires the opening of one fd per each directory that you intend to watch.
 - o The file descriptor pins the directory, disallowing the backing device to be unmounted, which absolutely wrecks havoc with removable media.
 - o Watching many directories results in many open file descriptors, possibly hitting a per-process fd limit.
- * dnotify is directory-based. You only learn about changes to

directories.

Sure, a change to a file in a directory affects the directory, but you are

then forced to keep a cache of stat structures around to compare things in order to find out which file.

- * dnotify's interface to user–space is awful.
 - o dnotify uses signals to communicate with user–space.
 - o Specifically, dnotify uses SIGIO.
 - o But then you can pick a different signal! So by "signals," I really meant you need to use real–time signals if you want to queue the events.
- * dnotify basically ignores any problems that would arise in the VFS from hard links.
- * Rumor is that the "d" in "dnotify" does not stand for "directory" but for "suck."

A suitable replacement is "inotify." And now, my tract on what inotify brings to the table:

- * inotify's interface is a device node, not SIGIO.
 - o You open only a single fd, to the device node. No more pinning directories or opening a million file descriptors.
 - o Usage is nice: open the device, issue simple commands via ioctl(), and then block on the device. It returns events when, well, there are events to be returned.
 - o You can select() on the device node and so it integrates with main loops like coffee mixed with vanilla milkshake.
- * inotify has an event that says "the filesystem that the item you were watching is on was unmounted" (this is particularly cool).
- * inotify can watch directories or files.
- * The "i" in inotify does not stand for "suck" but for "inode" -- the logical choice since inotify is inode–based.

--COMPLEXITY--

I have been asked what the complexity of inotify is. Inotify has 2 path codes where complexity could be an issue:

Adding a watcher to a device

This code has to check if the inode is already being watched by the device, this is O(1) since the maximum number of devices is limited to 8.

Removing a watch from a device

This code has to do a search of all watches on the device to find the watch descriptor that is being asked to remove.

This involves a linear search, but should not really be an issue because it is limited to 8192 entries. If this does turn in to a concern, I would replace the list of watches on the device with a sorted binary tree, so that the search could be done very quickly.

The calls to inotify from the VFS code has a complexity of $O(1)$ so inotify does not affect the speed of VFS operations.

--MEMORY USAGE--

The inotify data structures are light weight:

inotify watch is 40 bytes
inotify device is 68 bytes
inotify event is 272 bytes

So assuming a device has 8192 watches, the structures are only going to consume 320KB of memory. With a maximum number of 8 devices allowed to exist at a time, this is still only 2.5 MB

Each device can also have 256 events queued at a time, which sums to 68KB per device. And only .5 MB if all devices are opened and have a full event queue.

So approximately 3 MB of memory are used in the rare case of everything open and full.

Each inotify watch pins the inode of a directory/file in memory, the size of an inode is different per file system but lets assume that it is 512 bytes.

So assuming the maximum number of global watches are active, this would pin down 32 MB of inodes in the inode cache. Again not a problem on a modern system.

On smaller systems, the maximum watches / events could be lowered to provide a smaller foot print.

Keep in mind that this is an absolute worst case memory analysis. In reality it will most likely cost approximately 5MB.

--HOWTO USE--

Inotify is a character device that when opened offers 2 IOCTL's. (It actually has 4 but the other 2 are used for debugging)

INOTIFY_WATCH:

Which takes a path and event mask and returns a unique (to the instance of the driver) integer (wd [watch descriptor] from here on) that is a 1:1 mapping to the path passed.
What happens is inotify gets the inode (and ref's the inode)

for the path and adds a `inotify_watcher` structure to the inodes list of watchers. If this instance of the driver is already watching the path, the event mask will be updated and the original `wd` will be returned.

INOTIFY_IGNORE:

Which takes an integer (that you got from `INOTIFY_WATCH`) representing a `wd` that you are not interested in watching anymore. This will:

- send an `IGNORE` event to the device
- remove the `inotify_watcher` structure from the device and from the inode and unref the inode.

After you are watching 1 or more paths, you can read from the `fd` and get events. The events are `struct inotify_event`. If you are watching a directory and something happens to a file in the directory the event will contain the filename (just the filename not the full path).

— EVENTS —

`IN_ACCESS` – Sent when file is accessed.

`IN_MODIFY` – Sent when file is modified.

`IN_ATTRIB` – Sent when file is `chmod`'ed.

`IN_CLOSE` – Sent when file is closed

`IN_OPEN` – Sent when file is opened.

`IN_MOVED_FROM` – Sent to the source folder of a move.

`IN_MOVED_TO` – Sent to the destination folder of a move.

`IN_DELETE_SUBDIR` – Sent when a sub directory is deleted. (When watching parent)

`IN_DELETE_FILE` – Sent when a file is deleted. (When watching parent)

`IN_CREATE_SUBDIR` – Sent when a sub directory is created. (When watching parent)

`IN_CREATE_FILE` – Sent when a file is created. (When watching parent)

`IN_DELETE_SELF` – Sent when file is deleted.

`IN_UNMOUNT` – Sent when the filesystem is being unmounted.

`IN_Q_OVERFLOW` – Sent when your event queue has over flowed.

The `MOVED_FROM`/`MOVED_TO` events are always sent in pairs.

`MOVED_FROM`/`MOVED_TO`

is also sent when a file is renamed. The `cookie` field in the event pairs up `MOVED_FROM`/`MOVED_TO` events. These two events are not guaranteed to be successive in the event stream. You must rely on the `cookie` to pair them up. (Note, the `cookie` is not sent yet.)

If you aren't watching the source and destination folders in a `MOVE`.

You will only get `MOVED_TO` or `MOVED_FROM`. In this case, `MOVED_TO` is equivalent to a `CREATE` and `MOVED_FROM` is equivalent to a `DELETE`.

—KERNEL CHANGES—

`inotify` char device driver.

Linux-Kernel: [RFC][PATCH] inotify 0.11.0

Adding calls to `inotify_inode_queue_event` and `inotify_dentry_parent_queue_event` from VFS operations. Dnotify has the same function calls. The complexity of the VFS operations is not affected because `inotify_*_queue_event` is O(1).

Adding a call to `inotify_super_block_umount` from `generic_shutdown_superblock`

`inotify_super_block_umount` consists of this:
find all of the inodes that are on the super block being shut down,
sends each watcher on each inode the UNMOUNT and IGNORED event
removes the watcher structures from each instance of the device driver
and each inode.
unref's the inode.

—

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>