

[2.6.18 PATCH]: Filesystem Event Reporter V4

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-09/msg09085.html>

- *From:* Yi Yang <yang.y.yi@xxxxxxxxxx>
 - *Date:* Sat, 30 Sep 2006 23:24:55 +0800
-

I always have no time to update this patch in the past several months, eventually I got some time to do it, compared to v2, new changes is:

- Use reference counter to synchronize rmmmod, also consider process migration from a CPU to another during of calling to raise_fsevent*
- Free all the skbs queued on fsevent_sock before releasing fsevent_sock
- Rework some inline functions in include/linux/fsevent.h

This patch implements a new feature — Filesystem Event Reporter, the user can monitor filesystem activities via it, currently, it can monitor access, write, utime, chmod, chown, chgrp, close, open, create, rename, unlink, mkdir, rmdir, mount, umount.

Every filesystem event includes tgid, uid, gid and name of the process which triggered this event, the operated file or directory name. Any application with root privilege can set its fsevent filter list in order to just get those events who concerns but won't take effects in other applications to get any fsevents they are interested in, a fsevent filter may be based on event mask, pid, uid and gid. There are three kinds of filter lists, they are pid filter list, uid filetr list and gid filter list, respectively, moreover, there is a fsevent mask used to control those fsevents which fail to match three filter lists, an application using fsevent can listen those fsevents it want to monitor and ignore those fsevents it hasn't interest in by set series of filters, there is a fsevent mask used to take effects on all the applications using fsevent, it can be set by sysctl and proc interface.

On every cpu, there is a fsevent queue, filesystem code path just puts the corresponding fsevent on current cpu, then it can continue to execute the followed code without any lock, so there is a low overhead, a work per cpu is responsible for processing all the fsevents on this cpu and send them to listener applications.

Filesystem Events Reporter is never a duplicate of inotify, inotify just concerns change on file or directory, Beagle uses it to watch file changes in order to regenerate index for it, inotify can't tell us who did that change and what is its process name, but filesystem events reporter can do these, moreover inotify's overhead is greater than filesystem events reporter, inotify needs compare inode with watched file or directories list to decide whether it should generate an

[2.6.18 PATCH]: Filesystem Event Reporter V4

inotify_event, some locks also increase overhead, filesystem event connector hasn't these overhead, it just generates a fsevent and send.

To be important, filesystem event reporter doesn't add any new system call, the user space application can make use of it by netlink socket, but inotify added several system calls, many events mechanism in kernel have used netlink as communication way with user space, for example, KOBJECT_UEVENT, PROC_EVENTS, to use netlink will make it more possible to unify events interface to netlink, the user space application can use it very easily.

```
fs/Kconfig | 10
fs/Makefile | 3
fs/fsevent.c | 653 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
fs/fsevent_hook.c | 47 +++
fs/namespace.c | 12
include/linux/fsevent.h | 190 +++++++++++++++++++++++++++++++++
include/linux/fsnotify.h | 36 ++
include/linux/netlink.h | 1
8 files changed, 952 insertions(+)
```

Signed-off-by: Yi Yang <yang.y.yi@xxxxxxxxxx>

```
--- a/fs/Kconfig.orig 2006-09-30 18:33:48.000000000 +0800
+++ b/fs/Kconfig 2006-09-28 14:11:32.000000000 +0800
@@ -427,6 +427,16 @@ config INOTIFY_USER
```

If unsure, say Y.

```
+config FS_EVENTS
+ tristate "Report filesystem events to userspace"
+ ---help---
+ Provide a facility that reports filesystem events to userspace. The
+ reported event include access, write, utime, chmod, chown, chgrp,
+ close, open, create, rename, unlink, mkdir, rmdir, mount, umount.
+
+ The user can set filesystem events filter to filter its events, so
+ that he just get those events he concerns.
```

```
config QUOTA
bool "Quota support"
help
--- a/fs/Makefile.orig 2006-09-30 18:33:48.000000000 +0800
+++ b/fs/Makefile 2006-09-28 14:31:39.000000000 +0800
@@ -14,6 +14,9 @@ obj-y := open.o read_write.o file_table.
```

```
obj-$(CONFIG_INOTIFY) += inotify.o
obj-$(CONFIG_INOTIFY_USER) += inotify_user.o
+obj-$(CONFIG_FS_EVENTS) += fsevent.o
+fseventbase-$(CONFIG_FS_EVENTS) := fsevent_hook.o
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+obj-y += $(fseventbase-y) $(fseventbase-m)
obj-$(CONFIG_EPOLL) += eventpoll.o
obj-$(CONFIG_COMPAT) += compat.o compat_ioctl.o

--- /dev/null 2006-06-16 21:07:58.000000000 +0800
+++ b/fs/fsevent.c 2006-09-30 18:21:26.000000000 +0800
@@ -0,0 +1,653 @@
+/*
+ * fsevent.c
+ *
+ * 2006 Copyright (c) Yi Yang <yang.y.yi@xxxxxxxx>
+ * All rights reserved.
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
+ */
+
+#include <linux/kernel.h>
+#include <linux/module.h>
+#include <linux/skbuff.h>
+#include <linux/netlink.h>
+#include <linux/moduleparam.h>
+#include <linux/fsevent.h>
+#include <linux/skbuff.h>
+#include <net/sock.h>
+#include <linux/list.h>
+#include <linux/percpu.h>
+#include <linux/cpu.h>
+#include <linux/kthread.h>
+#include <linux/notifier.h>
+#include <linux/compiler.h>
+#include <linux/workqueue.h>
+#include <linux/sysctl.h>
+#include <linux/mutex.h>
+
+#define FSEVENT_MASK_CTL_NAME -2
+
+static DEFINE_PER_CPU(struct sk_buff_head, fsevent_send_queue);
+static DEFINE_PER_CPU(struct work_struct, fsevent_work);
+
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+typedef struct pid_filter {
+ pid_t pid;
+ u32 mask;
+ struct list_head list;
+} pid_filter;
+
+typedef struct uid_filter {
+ uid_t uid;
+ u32 mask;
+ struct list_head list;
+} uid_filter;
+
+typedef struct gid_filter {
+ gid_t gid;
+ u32 mask;
+ struct list_head list;
+} gid_filter;
+
+typedef struct fsevent_listener {
+ pid_t pid;
+ struct list_head pid_filter_list_head;
+ struct list_head uid_filter_list_head;
+ struct list_head gid_filter_list_head;
+ u32 mask;
+ struct list_head list;
+} listener;
+
+/* The netlink socket. */
+static struct sock * fsevent_sock = NULL;
+static LIST_HEAD(listener_list_head);
+static DEFINE_MUTEX(listener_list_mutex);
+
+static atomic_t fsevent_count = ATOMIC_INIT(0);
+static int fsevent_burst_limit = 100000;
+static unsigned long fsevent_ratelimit = 5 * HZ;
+static unsigned long last = 0;
+static int fsevent_sum = 0;
+static u32 fsevents_mask = FSEVENT_MASK;
+static atomic_t fsevent_listener_num = ATOMIC_INIT(0);
+static int exit_flag = 0;
+static struct ctl_table_header * fsevent_mask_ctlhdr = NULL;
+
+extern int * get_fsevent_refcnt(void);
+extern void put_fsevent_refcnt(void);
+extern int per_cpu_fsevent_refcnt(int cpu);
+extern void init_fsevent_refcnt(int cpu);
+extern void init_missed_fsevent_refcnt(int cpu);
+extern atomic_t * per_cpu_missed_refcnt(int cpu);
+
+static inline void get_seq(__u32 *ts, int *cpu)
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+{
+ *ts = atomic_inc_return(&fsevent_count);
+ *cpu = smp_processor_id();
+}
+
+static void append_string(char **dest, const char *src, size_t len)
+{
+ strncpy(*dest, src, len);
+ (*dest)[len] = '\0';
+ *dest += len + 1;
+}
+
+static inline int filter_fsevent(u32 filter_mask, u32 event_mask)
+{
+ event_mask &= FSEVENT_MASK;
+ event_mask &= filter_mask;
+ if (event_mask == 0) {
+ return -1;
+ }
+ return 0;
+}
+
+static int filter_fsevent_all(u32 * mask)
+{
+ int ret = 0;
+
+ (*mask) &= FSEVENT_MASK;
+
+ if (((*mask) & FSEVENT_ISDIR) == FSEVENT_ISDIR)
+ && ((fsevents_mask & FSEVENT_ISDIR) == 0) {
+ ret = -1;
+ goto out;
+ }
+
+ (*mask) &= fsevents_mask;
+ if ((*mask) == 0) {
+ ret = -1;
+ }
+
+out:
+ return ret;
+}
+
+static void fsevent_send(struct sk_buff * skb)
+{
+ struct sk_buff_head * head = &get_cpu_var(fsevent_send_queue);
+ skb_queue_tail(head, skb);
+ schedule_work(&per_cpu(fsevent_work, smp_processor_id()));
+ put_cpu_var(fsevent_send_queue);
+}
+
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+int __raise_fsevent(const char * oldname, const char * newname, u32 mask)
+{
+ struct fsevent *event;
+ int namelen = 0;
+ char * nameptr = NULL;
+ unsigned int size;
+ struct nlmsg_hdr * nlhdr;
+ struct sk_buff * skb = NULL;
+
+ if (filter_fsevent_all(&mask) != 0)
+ return -1;
+
+ if (atomic_read(&fsevent_listener_num) <= 0)
+ return -1;
+
+ if (time_after(jiffies, last)) {
+ last = jiffies + fsevent_ratelimit;
+ fsevent_sum = 0;
+ }
+ else {
+ if (fsevent_sum > fsevent_burst_limit)
+ return -1;
+ fsevent_sum++;
+ }
+
+ namelen = strlen(current->comm) + strlen(oldname) + 2;
+ if (newname)
+ namelen += strlen(newname) + 1;
+
+ size = NLMSG_SPACE(sizeof(struct fsevent) + namelen);
+
+ skb = alloc_skb(size, GFP_KERNEL);
+ if (!skb)
+ return -1;
+
+ nlhdr = NLMSG_PUT(skb, 0, 0, NLMSG_DONE, size - sizeof(*nlhdr));
+ event = NLMSG_DATA(nlhdr);
+
+ get_seq(&(nlhdr->nlmsg_seq), &event->cpu);
+ ktime_get_ts(&event->timestamp);
+ event->type = mask;
+ event->pid = current->tid;
+ event->uid = current->uid;
+ event->gid = current->gid;
+ nameptr = event->name;
+ event->pname_len = strlen(current->comm);
+ append_string(&nameptr, current->comm, event->pname_len);
+ event->fname_len = strlen(oldname);
+ append_string(&nameptr, oldname, event->fname_len);
+ event->len = event->pname_len + event->fname_len + 2;
+ event->new_fname_len = 0;
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ if (newname) {
+ event->new_fname_len = strlen(newname);
+ append_string(&nameptr, newname, event->new_fname_len);
+ event->len += event->new_fname_len + 1;
+ }
+ fsevent_send(skb);
+ return 0;
+
+nlmsg_failure:
+ kfree_skb(skb);
+ return -1;
+}
+
+static int fsevent_ack(enum fsevent_type type, pid_t pid, u32 seq)
+{
+ struct fsevent *event;
+ unsigned int size;
+ struct sk_buff * skb = NULL;
+ struct nlmsg_hdr * nlhdr = NULL;
+
+ size = NLMSG_SPACE(sizeof(struct fsevent));
+
+ skb = alloc_skb(size, GFP_KERNEL);
+ if (!skb)
+ return -ENOMEM;
+
+ nlhdr = NLMSG_PUT(skb, 0, seq, NLMSG_DONE, size - sizeof(*nlhdr));
+ event = NLMSG_DATA(nlhdr);
+
+ ktime_get_ts(&event->timestamp);
+ event->cpu = -1;
+ event->type = type;
+ event->pid = 0;
+ event->uid = 0;
+ event->gid = 0;
+ event->len = 0;
+ event->pname_len = 0;
+ event->fname_len = 0;
+ event->new_fname_len = 0;
+ event->err = 0;
+
+ NETLINK_CB(skb).dst_group = 0;
+ NETLINK_CB(skb).dst_pid = pid;
+ NETLINK_CB(skb).pid = 0;
+
+ return (netlink_unicast(fsevent_sock, skb, pid, MSG_DONTWAIT));
+
+nlmsg_failure:
+ kfree_skb(skb);
+ return -1;
+}
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+
+static void set_fsevent_mask(u32 * to_mask, u32 from_mask, int mode)
+{
+ if (mode == FSEVENT_FILTER_IGNORE)
+ (*to_mask) &= ~(from_mask);
+ else if (mode == FSEVENT_FILTER_LISTEN)
+ (*to_mask) |= from_mask;
+}
+
+#define DEFINE_FILTER_FIND_FUNC(type, key) \
+ type * find_##type(struct list_head * head, key##_t id) \
+ { \
+ int alloc_flag = 1; \
+ type * entry = NULL; \
+ \
+ list_for_each_entry(entry, head, list) { \
+ if (entry->key == id) { \
+ alloc_flag = 0; \
+ break; \
+ } \
+ } \
+ \
+ if (alloc_flag == 1) { \
+ entry = (type *)kmalloc(sizeof(type), GFP_KERNEL); \
+ if (entry == NULL) \
+ return NULL; \
+ memset(entry, 0, sizeof(type)); \
+ entry->key = id; \
+ list_add_tail(&(entry->list), head); \
+ } \
+ return entry; \
+ } \
+
+DEFINE_FILTER_FIND_FUNC(pid_filter, pid)
+
+DEFINE_FILTER_FIND_FUNC(uid_filter, uid)
+
+DEFINE_FILTER_FIND_FUNC(gid_filter, gid)
+
+DEFINE_FILTER_FIND_FUNC(listener, pid)
+
+static int set_fsevent_filter(struct fsevent_filter * filter, pid_t pid)
+{
+ enum fsevent_type type;
+ u32 mask = 0;
+ int control = 0;
+ listener * listenerp = NULL;
+ pid_filter * pfilter = NULL;
+ uid_filter * ufilter = NULL;
+ gid_filter * gfilter = NULL;
+ int ret = 0;
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+
+ mask = filter->mask;
+ control = filter->control;
+ type = filter->type;
+ mask &= FSEVENT_MASK;
+ if (mask == 0) {
+ ret = -1;
+ goto out;
+ }
+
+ mutex_lock(&listener_list_mutex);
+ listenerp = find_listener(&listener_list_head, pid);
+ if (unlikely(listenerp == NULL)) {
+ mutex_unlock(&listener_list_mutex);
+ return -1;
+ }
+
+ if (!(listenerp->pid_filter_list_head.next)) {
+ INIT_LIST_HEAD(&(listenerp->pid_filter_list_head));
+ INIT_LIST_HEAD(&(listenerp->uid_filter_list_head));
+ INIT_LIST_HEAD(&(listenerp->gid_filter_list_head));
+ }
+
+ if ((type & FSEVENT_FILTER_ALL) == FSEVENT_FILTER_ALL) {
+ if (control == FSEVENT_FILTER_REMOVE) {
+ atomic_dec(&fsevent_listener_num);
+ list_del(&(listenerp->list));
+ kfree(listenerp);
+ } else
+ set_fsevent_mask(&(listenerp->mask), mask, control);
+ } else if ((type & FSEVENT_FILTER_PID) == FSEVENT_FILTER_PID) {
+ pfilter = find_pid_filter(&(listenerp->pid_filter_list_head),
+ filter->id.pid);
+ if (unlikely(pfilter == NULL))
+ return -1;
+
+ if (control == FSEVENT_FILTER_REMOVE) {
+ list_del(&(pfilter->list));
+ kfree(pfilter);
+ } else
+ set_fsevent_mask(&(pfilter->mask), mask, control);
+ } else if ((type & FSEVENT_FILTER_UID) == FSEVENT_FILTER_UID) {
+ ufilter = find_uid_filter(&(listenerp->uid_filter_list_head),
+ filter->id.uid);
+ if (unlikely(ufilter == NULL))
+ return -1;
+
+ if (control == FSEVENT_FILTER_REMOVE) {
+ list_del(&(ufilter->list));
+ kfree(ufilter);
+ } else
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ set_fsevent_mask(&(ufilter->mask), mask, control);
+ } else if ((type & FSEVENT_FILTER_GID) == FSEVENT_FILTER_GID) {
+ gfilter = find_gid_filter(&(listenerp->gid_filter_list_head),
+ filter->id.gid);
+ if (unlikely(gfilter == NULL))
+ return -1;
+
+ if (control == FSEVENT_FILTER_REMOVE) {
+ list_del(&(gfilter->list));
+ kfree(gfilter);
+ } else
+ set_fsevent_mask(&(gfilter->mask), mask, control);
+ }
+ mutex_unlock(&listener_list_mutex);
+ ret = 0;
+
+out:
+ fsevent_ack(type, pid, 0);
+ return(ret);
+}
+
+static listener * find_fsevent_listener(pid_t pid)
+{
+ listener * listenerp = NULL;
+ mutex_lock(&listener_list_mutex);
+ list_for_each_entry(listenerp, &listener_list_head, list) {
+ if (listenerp->pid == pid) {
+ mutex_unlock(&listener_list_mutex);
+ return listenerp;
+ }
+ }
+ mutex_unlock(&listener_list_mutex);
+ return NULL;
+}
+
+static void cleanup_dead_listener(listener * x)
+{
+ pid_filter * p = NULL, * pq = NULL;
+ uid_filter * u = NULL, * uq = NULL;
+ gid_filter * g = NULL, * gq = NULL;
+
+ if (x == NULL)
+ return;
+
+ list_del(&(x->list));
+
+ list_for_each_entry_safe(p, pq, &(x->pid_filter_list_head), list) {
+ list_del(&(p->list));
+ kfree(p);
+ }
+}
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ list_for_each_entry_safe(u, uq, &(x->uid_filter_list_head), list) {
+ list_del(&(u->list));
+ kfree(u);
+ }
+
+ list_for_each_entry_safe(g, gq, &(x->gid_filter_list_head), list) {
+ list_del(&(g->list));
+ kfree(g);
+ }
+
+ kfree(x);
+}
+
+static void fsevent_recv(struct sock *sk, int len)
+{
+ struct sk_buff *skb = NULL;
+ struct nlmsg_hdr *nlhdr = NULL;
+ struct fsevent_filter *filter = NULL;
+ pid_t pid;
+ int inc_flag = 0;
+
+ if (exit_flag == 1)
+ return;
+
+ while ((skb = skb_dequeue(&sk->sk_receive_queue)) != NULL) {
+ skb_get(skb);
+ if (skb->len >= FSEVENT_FILTER_MSGSIZE) {
+ nlhdr = (struct nlmsg_hdr *)skb->data;
+ filter = NLMSG_DATA(nlhdr);
+ pid = NETLINK_CREDS(skb->pid);
+ if (find_fsevent_listener(pid) == NULL)
+ inc_flag = 1;
+ if (set_fsevent_filter(filter, pid) == 0) {
+ if (inc_flag == 1)
+ atomic_inc(&fsevent_listener_num);
+ }
+ }
+ }
+ kfree_skb(skb);
+ }
+}
+
+#define DEFINE_FILTER_MATCH_FUNC(filtertype, key) \
+ static int match_##filtertype(listener * p, \
+ struct fsevent * event, \
+ struct sk_buff * skb) \
+ { \
+ int ret = 0; \
+ filtertype * xfilter = NULL; \
+ struct sk_buff * skb2 = NULL; \
+ struct list_head * head = &(p->key##_filter_list_head); \
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ list_for_each_entry(xfilter, head, list) { \
+ if (xfilter->key != event->key) \
+ continue; \
+ ret = filter_fsevent(xfilter->mask, event->type); \
+ if (ret != 0) \
+ return -1; \
+ skb2 = skb_clone(skb, GFP_KERNEL); \
+ if (skb2 == NULL) \
+ return -1; \
+ NETLINK_CB(skb2).dst_group = 0; \
+ NETLINK_CB(skb2).dst_pid = p->pid; \
+ NETLINK_CB(skb2).pid = 0; \
+ return (netlink_unicast(fsevent_sock, skb2, \
+ p->pid, MSG_DONTWAIT)); \
+ } \
+ return -1; \
+ } \
+
+DEFINE_FILTER_MATCH_FUNC(pid_filter, pid)
+
+DEFINE_FILTER_MATCH_FUNC(uid_filter, uid)
+
+DEFINE_FILTER_MATCH_FUNC(gid_filter, gid)
+
+#define MATCH_XID(key, listenerp, event, skb) \
+ ret = match_##key##_filter(listenerp, event, skb); \
+ if (ret == 0) { \
+ kfree_skb(skb); \
+ continue; \
+ } \
+ do {} while (0) \
+
+static int fsevent_send_to_process(struct sk_buff * skb)
+{
+ listener * p = NULL, * q = NULL;
+ struct fsevent * event = NULL;
+ struct sk_buff * skb2 = NULL;
+ int ret = 0;
+
+ event = (struct fsevent *) (skb->data + sizeof(struct nlmsghdr));
+ mutex_lock(&listener_list_mutex);
+ list_for_each_entry_safe(p, q, &listener_list_head, list) {
+ MATCH_XID(pid, p, event, skb);
+ MATCH_XID(uid, p, event, skb);
+ MATCH_XID(gid, p, event, skb);
+
+ if (filter_fsevent(p->mask, event->type) == 0) {
+ skb2 = skb_clone(skb, GFP_KERNEL);
+ if (skb2 == NULL)
+ return -1;
+ NETLINK_CB(skb2).dst_group = 0;
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ NETLINK_CB(skb2).dst_pid = p->pid;
+ NETLINK_CB(skb2).pid = 0;
+ ret = netlink_unicast(fsevent_sock, skb2,
+ p->pid, 0);
+ if (ret == -ECONNREFUSED) {
+ atomic_dec(&fsevent_listener_num);
+ cleanup_dead_listener(p);
+ }
+ }
+ }
+ }
+ mutex_unlock(&listener_list_mutex);
+ return ret;
+ }
+
+static void fsevent_commit(void * unused)
+{
+ struct sk_buff * skb = NULL;
+ int * refcnt, missed_refcnt;
+ int cpuid;
+
+
+ while((skb = skb_dequeue(&get_cpu_var(fsevent_send_queue)))
+ != NULL) {
+ fsevent_send_to_process(skb);
+ kfree_skb(skb);
+ put_cpu_var(fsevent_send_queue);
+ }
+
+ if (exit_flag == 1) {
+ refcnt = get_fsevent_refcnt();
+
+ if (*refcnt <= 0) {
+ *refcnt = -1;
+ put_fsevent_refcnt();
+ return;
+ }
+ cpuid = smp_processor_id();
+ missed_refcnt = atomic_read(per_cpu_misssed_refcnt(cpuid));
+ if (missed_refcnt == *refcnt)
+ {
+ *refcnt = -1;
+ printk("cpu#%d: missed refcnt = %d\n", cpuid, missed_refcnt);
+ }
+ else {
+ if (missed_refcnt != 0) {
+ printk("cpu#%d: refcnt = %d, missed refcnt = %d\n", cpuid, *refcnt, missed_refcnt);
+ }
+ }
+ put_fsevent_refcnt();
+ }
+ }
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+
+static struct ctl_table fsevent_mask_sysctl[] = {
+ {
+ .ctl_name = FSEVENT_MASK_CTL_NAME,
+ .procname = "fsevent_mask",
+ .data = &fsevents_mask,
+ .maxlen = sizeof(u32),
+ .mode = 0644,
+ .proc_handler = &proc_dointvec,
+ },
+ { .ctl_name = 0 }
+};
+
+static struct ctl_table fs_root_sysctl[] = {
+ {
+ .ctl_name = CTL_FS,
+ .procname = "fs",
+ .mode = 0555,
+ .child = fsevent_mask_sysctl,
+ },
+ { .ctl_name = 0 }
+};
+
+static int __init fsevent_init(void)
+{
+ int cpu;
+ struct sk_buff_head * listptr;
+ struct work_struct * workptr;
+
+ fsevent_sock = netlink_kernel_create(NETLINK_FSEVENT, 0,
+ fsevent_recv, THIS_MODULE);
+ if (!fsevent_sock)
+ return -EIO;
+ for_each_possible_cpu(cpu) {
+ init_fsevent_refcnt(cpu);
+ init_missed_fsevent_refcnt(cpu);
+ listptr = &per_cpu(fsevent_send_queue, cpu);
+ skb_queue_head_init(listptr);
+ workptr = &per_cpu(fsevent_work, cpu);
+ INIT_WORK(workptr, fsevent_commit, NULL);
+ }
+
+ fsevent_mask_ctlhdr = register_sysctl_table(fs_root_sysctl, 0);
+ if (fsevent_mask_ctlhdr == NULL)
+ return -ENOMEM;
+
+ __raise_fsevent = ___raise_fsevent;
+
+ return 0;
+}
+
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+static void __exit fsevent_exit(void)
+{
+ listener * p = NULL, * q = NULL;
+ int cpu;
+ int wait_flag = 1;
+ struct sk_buff * skb = NULL;
+
+ fsevents_mask = 0;
+ exit_flag = 1;
+
+ while (wait_flag == 1) {
+ wait_flag = 0;
+ for_each_possible_cpu(cpu) {
+ if (per_cpu_fsevent_refcnt(cpu) >= 0) {
+ wait_flag = 1;
+ schedule_work(&per_cpu(fsevent_work, cpu));
+ }
+ }
+ flush_scheduled_work();
+ }
+ __raise_fsevent = 0;
+
+ while ((skb = skb_dequeue(&fsevent_sock->sk_receive_queue)) != NULL) {
+ kfree_skb(skb);
+ }
+ while ((skb = skb_dequeue(&fsevent_sock->sk_write_queue)) != NULL) {
+ kfree_skb(skb);
+ }
+ printk("sk_rmem_alloc=%d, sk_wmem_alloc=%d\n",
+ atomic_read(&fsevent_sock->sk_rmem_alloc),
+ atomic_read(&fsevent_sock->sk_wmem_alloc));
+ atomic_set(&fsevent_sock->sk_rmem_alloc, 0);
+ atomic_set(&fsevent_sock->sk_wmem_alloc, 0);
+ sock_release(fsevent_sock->sk_socket);
+ mutex_lock(&listener_list_mutex);
+ list_for_each_entry_safe(p, q, &listener_list_head, list) {
+ cleanup_dead_listener(p);
+ }
+ mutex_unlock(&listener_list_mutex);
+ unregister_sysctl_table(fsevent_mask_ctlhdr);
+ }
+
+ module_init(fsevent_init);
+ module_exit(fsevent_exit);
+
+ MODULE_LICENSE("GPL");
+ MODULE_AUTHOR("Yi Yang <yang.y.yi@xxxxxxxxxx>");
+ MODULE_DESCRIPTION("File System Events Reporter");
+--- /dev/null 2006-06-16 21:07:58.000000000 +0800
+++ b/fs/fsevent_hook.c 2006-09-30 16:20:54.000000000 +0800
@@ -0,0 +1,47 @@
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+#include <linux/fsevent.h>
+
+int (* __raise_fsevent)
+(const char * oldname, const char * newname, u32 mask) = 0;
+EXPORT_SYMBOL(__raise_fsevent);
+
+static DEFINE_PER_CPU(int, raise_fsevent_refcnt);
+static DEFINE_PER_CPU(atomic_t, missed_fsevent_refcnt);
+
+int * get_fsevent_refcnt(void)
+{
+ return &get_cpu_var(raise_fsevent_refcnt);
+}
+EXPORT_SYMBOL(get_fsevent_refcnt);
+
+void put_fsevent_refcnt(void)
+{
+ put_cpu_var(raise_fsevent_refcnt);
+}
+EXPORT_SYMBOL(put_fsevent_refcnt);
+
+int per_cpu_fsevent_refcnt(int cpu)
+{
+ return per_cpu(raise_fsevent_refcnt, cpu);
+}
+EXPORT_SYMBOL(per_cpu_fsevent_refcnt);
+
+void init_fsevent_refcnt(int cpu)
+{
+ int * refcnt = &per_cpu(raise_fsevent_refcnt, cpu);
+ *refcnt = 0;
+}
+EXPORT_SYMBOL(init_fsevent_refcnt);
+
+void init_missed_fsevent_refcnt(int cpu)
+{
+ atomic_t * missed_refcnt = &per_cpu(missed_fsevent_refcnt, cpu);
+
+ atomic_set(missed_refcnt, 0);
+}
+EXPORT_SYMBOL(init_missed_fsevent_refcnt);
+
+atomic_t * per_cpu_missed_refcnt(int cpu)
+{
+ return &per_cpu(missed_fsevent_refcnt, cpu);
+}
+EXPORT_SYMBOL(per_cpu_missed_refcnt);
+--- a/fs/namespace.c.orig 2006-09-30 18:33:48.000000000 +0800
+++ b/fs/namespace.c 2006-09-28 14:11:32.000000000 +0800
@@ -24,6 +24,7 @@
#include <linux/mount.h>
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
#include <asm/uaccess.h>
#include <asm/unistd.h>
+#include <linux/fsevent.h>
#include "pnode.h"

extern int __init init_rootfs(void);
@@ -656,6 +657,13 @@ asmlinkage long sys_umount(char __user *
goto dput_and_out;

retval = do_umount(nd.mnt, flags);
+
+ if (retval == 0) {
+ char * tmp = getname(name);
+ raise_fsevent_umount(tmp);
+ putname(tmp);
+ }
+
dput_and_out:
path_release_on_umount(&nd);
out:
@@ -1577,6 +1585,10 @@ asmlinkage long sys_mount(char __user *
retval = do_mount((char *)dev_page, dir_page, (char *)type_page,
flags, (void *)data_page);
unlock_kernel();
+
+ if (retval == 0)
+ raise_fsevent_mount((char *)dev_page, dir_page);
+
free_page(data_page);

out3:
--- /dev/null 2006-06-16 21:07:58.000000000 +0800
+++ b/include/linux/fsevent.h 2006-09-30 17:03:00.000000000 +0800
@@ -0,0 +1,190 @@
+/*
+ * fsevent.h - filesystem events reporter
+ *
+ * Copyright (C) 2006 Yi Yang <yang.y.yi@xxxxxxxxxx>
+ * Based on cn_proc.h by Matt Helsley, IBM Corp
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
+ */
+
+#ifndef LINUX_FSEVENT_H
+#define LINUX_FSEVENT_H
+
+#include <linux/types.h>
+#include <linux/time.h>
+#include <linux/netlink.h>
+
+enum fsevent_type {
+ FSEVENT_ACCESS = 0x00000001, /* File was accessed */
+ FSEVENT_MODIFY = 0x00000002, /* File was modified */
+ FSEVENT_MODIFY_ATTRIB = 0x00000004, /* Metadata changed */
+ FSEVENT_CLOSE = 0x00000008, /* File was closed */
+ FSEVENT_OPEN = 0x00000010, /* File was opened */
+ FSEVENT_MOVE = 0x00000020, /* File was moved */
+ FSEVENT_CREATE = 0x00000040, /* File was created */
+ FSEVENT_DELETE = 0x00000080, /* File was deleted */
+ FSEVENT_MOUNT = 0x00000100, /* File system is mounted */
+ FSEVENT_UMOUNT = 0x00000200, /* File system is unmounted */
+
+ /* The following definitions are command types for fsevent filter
+ * or acknowledge types of the corresponding commands
+ */
+ FSEVENT_FILTER_ALL = 0x08000000, /* For all events */
+ FSEVENT_FILTER_PID = 0x10000000, /* For some process ID */
+ FSEVENT_FILTER_UID = 0x20000000, /* For some user ID */
+ FSEVENT_FILTER_GID = 0x40000000, /* For some group ID */
+
+ FSEVENT_ISDIR = 0x80000000 /* It is set for a dir */
+};
+
+#define FSEVENT_MASK 0x800003ff
+
+typedef unsigned long fsevent_mask_t;
+
+enum filter_control {
+ FSEVENT_FILTER_LISTEN = 1, /* Listen fsevents mask defines*/
+ FSEVENT_FILTER_IGNORE, /* Ignore fsevents mask defines*/
+ FSEVENT_FILTER_REMOVE, /* Remove a given filter */
+};
+
+struct fsevent_filter {
+ /* filter type, it just is one of them
+ * FSEVENT_FILTER_ALL
+ * FSEVENT_FILTER_PID
+ * FSEVENT_FILTER_UID
+ * FSEVENT_FILTER_GID
+ */
+};
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ enum fsevent_type type; /* filter type */
+
+ /* mask of file system events the user listen or ignore
+ * if the user need to ignore all the events of some pid
+ * , gid or uid, he(he) must set mask to FSEVENT_MASK.
+ */
+ fsevent_mask_t mask;
+ union {
+ pid_t pid;
+ uid_t uid;
+ gid_t gid;
+ } id;
+
+ enum filter_control control;
+};
+
+struct fsevent {
+ __u32 type;
+ __u32 cpu;
+ struct timespec timestamp;
+ pid_t pid;
+ uid_t uid;
+ gid_t gid;
+ int err;
+ __u32 len;
+ __u32 pname_len;
+ __u32 fname_len;
+ __u32 new_fname_len;
+ char name[0];
+};
+
+#define FSEVENT_FILTER_MSGSIZE \
+ (sizeof(struct fsevent_filter) + sizeof(struct nlmsghdr))
+
+#ifdef __KERNEL__
+#if defined(CONFIG_FS_EVENTS) || defined(CONFIG_FS_EVENTS_MODULE)
+extern int (* __raise_fsevent)
+ (const char * oldname, const char * newname, u32 mask);
+
+extern int * get_fsevent_refcnt(void);
+extern void put_fsevent_refcnt(void);
+extern atomic_t * per_cpu_missed_refcnt(int cpu);
+
+static inline int _raise_fsevent
+ (const char * oldname, const char * newname, u32 mask)
+{
+ int ret;
+ int * refcnt = NULL;
+ int start_cpuid, end_cpuid;
+
+ refcnt = get_fsevent_refcnt();
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+ start_cpuid = smp_processor_id();
+ if ((*refcnt == -1) || (__raise_fsevent == 0)) {
+ put_fsevent_refcnt();
+ return -1;
+ }
+ (*refcnt)++;
+ put_fsevent_refcnt();
+ ret = __raise_fsevent(oldname, newname, mask);
+ refcnt = get_fsevent_refcnt();
+ end_cpuid = smp_processor_id();
+ if (start_cpuid != end_cpuid) {
+ printk("fsevent: process '%s' migration: cpu#%d -> cpu#%d.", current->comm, start_cpuid, end_cpuid);
+ atomic_inc(per_cpu_missed_refcnt(start_cpuid));
+ put_fsevent_refcnt();
+ return -1;
+ }
+ (*refcnt)--;
+ put_fsevent_refcnt();
+ return ret;
+}
+
+static inline void raise_fsevent(struct dentry * dentryp, u32 mask)
+{
+ if (dentryp->d_inode && (MAJOR(dentryp->d_inode->i_rdev) == 4))
+ return;
+ _raise_fsevent(dentryp->d_name.name, NULL, mask);
+}
+
+static inline void raise_fsevent_move(struct inode * olddir,
+ const char * oldname, struct inode * newdir,
+ const char * newname, u32 mask)
+{
+ _raise_fsevent(oldname, newname, mask);
+}
+
+static inline void raise_fsevent_create(struct inode * inode,
+ const char * name, u32 mask)
+{
+ _raise_fsevent(name, NULL, mask);
+}
+
+static inline void raise_fsevent_mount(const char * devname,
+ const char * mountpoint)
+{
+ _raise_fsevent(devname, mountpoint, FSEVENT_MOUNT);
+}
+
+static inline void raise_fsevent_umount(const char * mountpoint)
+{
+ _raise_fsevent(mountpoint, NULL, FSEVENT_UMOUNT);
+}
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
+#else
+static void raise_fsevent(struct dentry * dentryp, u32 mask)
+{}
+
+static void raise_fsevent_move(struct inode * olddir, const char * oldname,
+ struct inode * newdir, const char * newname, u32 mask)
+{}
+
+static void raise_fsevent_create(struct inode * inode,
+ const char * name, u32 mask)
+{}
+
+static void raise_fsevent_mount(const char * devname, const char * mountpoint)
+{}
+
+static void raise_fsevent_umount(const char * mountpoint)
+{}
+#endif /* CONFIG_FS_EVENTS || CONFIG_FS_EVENTS_MODULE */
+#endif /* __KERNEL__ */
+#endif /* LINUX_FSEVENT_H */
--- a/include/linux/fsnotify.h.orig 2006-09-30 18:33:48.000000000 +0800
+++ b/include/linux/fsnotify.h 2006-09-28 14:43:34.000000000 +0800
@@ -16,6 +16,7 @@
#include <linux/dnotify.h>
#include <linux/inotify.h>
#include <linux/audit.h>
+#include <linux/fsevent.h>

/*
 * fsnotify_d_instantiate – instantiate a dentry for inode
@@ -67,6 +68,8 @@ static inline void fsnotify_move(struct
if (source) {
inotify_inode_queue_event(source, IN_MOVE_SELF, 0, NULL, NULL);
}
+ raise_fsevent_move(old_dir, old_name, new_dir, new_name,
+ FSEVENT_MOVE | (isdir?FSEVENT_ISDIR:0));
audit_inode_child(new_name, source, new_dir);
}

@@ -79,6 +82,8 @@ static inline void fsnotify_nameremove(s
isdir = IN_ISDIR;
dnotify_parent(dentry, DN_DELETE);
inotify_dentry_parent_queue_event(dentry, IN_DELETE|isdir, 0, dentry->d_name.name);
+ raise_fsevent(dentry,
+ FSEVENT_DELETE | (isdir?FSEVENT_ISDIR:0));
}

/*
@@ -98,6 +103,7 @@ static inline void fsnotify_create(struc
inode_dir_notify(inode, DN_CREATE);
inotify_inode_queue_event(inode, IN_CREATE, 0, dentry->d_name.name,
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
dentry->d_inode);
+ raise_fsevent_create(inode, dentry->d_name.name, FSEVENT_CREATE);
audit_inode_child(dentry->d_name.name, dentry->d_inode, inode);
}
```

```
@@ -109,6 +115,8 @@ static inline void fsnotify_mkdir(struct
inode_dir_notify(inode, DN_CREATE);
inotify_inode_queue_event(inode, IN_CREATE | IN_ISDIR, 0,
dentry->d_name.name, dentry->d_inode);
+ raise_fsevent_create(inode, dentry->d_name.name,
+ FSEVENT_CREATE | FSEVENT_ISDIR);
audit_inode_child(dentry->d_name.name, dentry->d_inode, inode);
}
```

```
@@ -126,6 +134,8 @@ static inline void fsnotify_access(struc
dnotify_parent(dentry, DN_ACCESS);
inotify_dentry_parent_queue_event(dentry, mask, 0, dentry->d_name.name);
inotify_inode_queue_event(inode, mask, 0, NULL, NULL);
+ raise_fsevent(dentry, FSEVENT_ACCESS |
+ ((S_ISDIR(inode->i_mode))?FSEVENT_ISDIR:0));
}
```

/*

```
@@ -142,6 +152,8 @@ static inline void fsnotify_modify(struc
dnotify_parent(dentry, DN_MODIFY);
inotify_dentry_parent_queue_event(dentry, mask, 0, dentry->d_name.name);
inotify_inode_queue_event(inode, mask, 0, NULL, NULL);
+ raise_fsevent(dentry, FSEVENT_MODIFY |
+ ((S_ISDIR(inode->i_mode))?FSEVENT_ISDIR:0));
}
```

/*

```
@@ -157,6 +169,8 @@ static inline void fsnotify_open(struct
inotify_dentry_parent_queue_event(dentry, mask, 0, dentry->d_name.name);
inotify_inode_queue_event(inode, mask, 0, NULL, NULL);
+ raise_fsevent(dentry, FSEVENT_OPEN |
+ ((S_ISDIR(inode->i_mode))?FSEVENT_ISDIR:0));
}
```

/*

```
@@ -175,6 +189,8 @@ static inline void fsnotify_close(struct
inotify_dentry_parent_queue_event(dentry, mask, 0, name);
inotify_inode_queue_event(inode, mask, 0, NULL, NULL);
+ raise_fsevent(dentry, FSEVENT_CLOSE |
+ ((S_ISDIR(inode->i_mode))?FSEVENT_ISDIR:0));
}
```

/*

```
@@ -190,6 +206,8 @@ static inline void fsnotify_xattr(struct
```

[2.6.18 PATCH]: Filesystem Event Reporter V4

```
inotify_dentry_parent_queue_event(dentry, mask, 0, dentry->d_name.name);
inotify_inode_queue_event(inode, mask, 0, NULL, NULL);
+ raise_fsevent(dentry, FSEVENT_MODIFY_ATTRIB |
+ ((S_ISDIR(inode->i_mode))?FSEVENT_ISDIR:0));
}

/*
@@ -240,6 +258,24 @@ static inline void fsnotify_change(struc
inotify_dentry_parent_queue_event(dentry, in_mask, 0,
dentry->d_name.name);
}
+
+#ifdef CONFIG_FS_EVENTS
+ {
+ u32 fsevent_mask = 0;
+ if (ia_valid & (ATTR_UID | ATTR_GID | ATTR_MODE))
+ fsevent_mask |= FSEVENT_MODIFY_ATTRIB;
+ if ((ia_valid & ATTR_ATIME) && (ia_valid & ATTR_MTIME))
+ fsevent_mask |= FSEVENT_MODIFY_ATTRIB;
+ else if (ia_valid & ATTR_ATIME)
+ fsevent_mask |= FSEVENT_ACCESS;
+ else if (ia_valid & ATTR_MTIME)
+ fsevent_mask |= FSEVENT_MODIFY;
+ if (ia_valid & ATTR_SIZE)
+ fsevent_mask |= FSEVENT_MODIFY;
+ if (fsevent_mask)
+ raise_fsevent(dentry, fsevent_mask);
+ }
+#endif /* CONFIG_FS_EVENTS */
}

#ifdef CONFIG_INOTIFY /* inotify helpers */
--- a/include/linux/netlink.h.orig 2006-09-30 18:33:49.000000000 +0800
+++ b/include/linux/netlink.h 2006-09-28 14:11:32.000000000 +0800
@@ -21,6 +21,7 @@
#define NETLINK_DNRTMSG 14 /* DECnet routing messages */
#define NETLINK_KOBJECT_UEVENT 15 /* Kernel messages to userspace */
#define NETLINK_GENERIC 16
+#define NETLINK_FSEVENT 17 /* File system events to userspace */

#define MAX_LINKS 32

-
```

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