
Subject: [PATCH 2/2] signal c/r: implement /proc/pid/sig writing

Posted by [serue](#) on Fri, 08 Jun 2007 15:54:58 GMT

[View Forum Message](#) <> [Reply to Message](#)

>From 0fc34dcb54fe80ea720225825ad1dcb1b847d8ab Mon Sep 17 00:00:00 2001

From: Serge E. Hallyn <serue@us.ibm.com>

Date: Thu, 17 May 2007 17:28:07 -0400

Subject: [PATCH 2/2] signal c/r: implement /proc/pid/sig writing

Allow restore through writes to /proc/pid/sig/*, except for the waiters file. Waits must be restored by the waiter actually running wait.

To test writes to the action file I use the included handlertest.c.

Start it up in one terminal as

`./handlertest 1`

It should print "using handler 1"

in another terminal

`pid=`ps -ef | grep handlertest | grep -v grep | awk '{ print $2 }``

`cat /proc/$pid/sig/action > action`

`kill -USR1 $pid`

handlertest should print "handler 1 called"

You can repeat this with

`./handlertest 2`

to make sure it does what you expect with the second signal handler.

Restart handlertest with a different signal handler:

`./handlertest 2`

It prints "using handler 2"

Overwrite it's sigactions from another terminal

`pid=`ps -ef | grep handlertest | grep -v grep | awk '{ print $2 }``

`cat action > /proc/$pid/sig/action`

and send it a USR1

`kill -USR1 $pid`

and it should say "handler 1 called".

```
=====
handlertest.c
=====
```

```
void handler1(int sig)
{
    printf("handler 1 called\n");
}
```

```
void handler2(int sig)
{
```

```

    printf("handler 2 called\n");
}

int main(int argc, char *argv[])
{
    int hnum;
    __sighandler_t h = handler1;

    if (argc<2) {
        printf("usage: %s [1|2]\n", argv[0]);
        exit(2);
    }
    hnum = atoi(argv[1]);
    if (hnum == 1) {
        printf("using handler 1\n");
    } else {
        h = handler2;
        printf("using handler 2\n");
    }
    signal(SIGUSR1, h);

    sleep(100);
}

```

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

```

fs/proc/base.c      | 97 ++++++-----
include/linux/signal.h | 2 +
kernel/signal.c      | 218 ++++++-----
3 files changed, 313 insertions(+), 4 deletions(-)

```

diff --git a/fs/proc/base.c b/fs/proc/base.c

index 8e0dd7a..22817bb 100644

--- a/fs/proc/base.c

+++ b/fs/proc/base.c

```

@@ -1701,9 +1701,104 @@ static ssize_t proc_pid_sig_read(struct file * file, char __user * buf,
    return length;
}

```

```

+struct proc_pid_sig_partial {

```

```

+ char *buf;

```

```

+ loff_t pos;

```

```

+ size_t count;

```

```

+};

```

```

+

```

```

+static struct proc_pid_sig_partial *make_partial(char *page, int start, int total_count)

```

```

+{

```

```

+ struct proc_pid_sig_partial *partial;

```

```

+
+ partial = kzalloc(sizeof(*partial), GFP_KERNEL);
+ if (!partial)
+ return NULL;
+ partial->buf = page;
+ partial->pos = start;
+ partial->count = total_count;
+ return partial;
+}
+
+static ssize_t proc_pid_sig_write(struct file * file, const char __user * buf,
+    size_t count, loff_t *ppos)
+{
+ struct dentry * dentry = file->f_path.dentry;
+ struct inode * inode = dentry->d_inode;
+ char *page;
+ ssize_t length;
+ struct task_struct *task = get_proc_task(inode);
+ struct proc_pid_sig_partial *partial;
+ int partial_size = 0, total_count;
+
+ length = -ESRCH;
+ if (!task)
+ goto out_no_task;
+
+ partial = file->private_data;
+ file->private_data = NULL;
+ if (partial)
+ partial_size = partial->count - partial->pos;
+ total_count = count + partial_size;
+
+ printk(KERN_NOTICE "%s: 1\n", __FUNCTION__);
+ length = -ENOMEM;
+ page = kmalloc(total_count, GFP_USER);
+ if (!page)
+ goto out_put_task;
+ printk(KERN_NOTICE "%s: 2\n", __FUNCTION__);
+
+ if (partial) {
+ memcpy(page, partial->buf + partial->pos, partial_size);
+ kfree(partial->buf);
+ kfree(partial);
+ }
+
+ printk(KERN_NOTICE "%s: 3\n", __FUNCTION__);
+ length = -EFAULT;
+ if (copy_from_user(page+partial_size, buf, count))
+ goto out_free_page;

```

```

+
+ length = task_write_procsig(task,
+     (char*)file->f_path.dentry->d_name.name,
+     (void*)page, total_count);
+
+ printk(KERN_NOTICE "%s: 4, length was %d, count %d totcnt %d\n", __FUNCTION__,
+     length, count, total_count);
+ if (length >= 0 && length < total_count) {
+     /* should i just allocate a new buffer for just the unread portion? */
+     file->private_data = make_partial(page, length+1, total_count);
+     if (!file->private_data)
+         length = -ENOMEM;
+     else
+         goto out_put_task; /* don't free page, partial points to it */
+ }
+
+out_free_page:
+ kfree(page);
+out_put_task:
+ put_task_struct(task);
+out_no_task:
+ if (length >= 0)
+     return count;
+ return length;
+}
+
+static int proc_pid_sig_release(struct inode *inode, struct file *file)
+{
+ struct proc_pid_sig_partial *partial = file->private_data;
+
+ if (partial) {
+     kfree(partial->buf);
+     kfree(partial);
+ }
+ return 0;
+}
+
+static const struct file_operations proc_sig_attr_operations = {
+ .read = proc_pid_sig_read,
+ // .write = proc_pid_sig_write,
+ .write = proc_pid_sig_write,
+ .release = proc_pid_sig_release,
+ };

```

```

diff --git a/include/linux/signal.h b/include/linux/signal.h
index 6863543..2e6d64c 100644
--- a/include/linux/signal.h

```

```

+++ b/include/linux/signal.h
@@ -244,6 +244,8 @@ extern int get_signal_to_deliver(siginfo_t *info, struct k_sigaction
*return_ka,

extern struct kmem_cache *sigband_cache;
extern int task_read_procsig(struct task_struct *p, char *name, char **value);
+extern int task_write_procsig(struct task_struct *p, char *name, void *value,
+ size_t size);

/*
 * In POSIX a signal is sent either to a specific thread (Linux task)
diff --git a/kernel/signal.c b/kernel/signal.c
index a2a3ebe..dc50e1b 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -2614,7 +2614,7 @@ static int print_sigpending_alloc(char **bufp, struct sigpending
*pending)

list_for_each_entry(q, &pending->list, list) {
info = &q->info;
- if (p-buf+215 > allocated) {
+ if (p-buf+221 > allocated) {
int len=p-buf;
char *buf2;
allocated += PAGE_SIZE;
@@ -2629,8 +2629,8 @@ static int print_sigpending_alloc(char **bufp, struct sigpending
*pending)
p = buf+len;
}

- p += sprintf(p, "sig %d: user %d flags %d",
- info->si_signo, (int)q->user->uid, q->flags);
+ p += sprintf(p, "sig %d: uid %d pid %d flags %d",
+ info->si_signo, (int)q->user->uid, info->si_pid, q->flags);
p += sprintf(p, " errno %d code %d\n",
info->si_errno, info->si_code);

@@ -2739,3 +2739,215 @@ int task_read_procsig(struct task_struct *p, char *name, char
**value)

return ret;
}
+
+static int set_sigset(sigset_t *sig, char *value, int size)
+{
+ int i;
+
+ if (size < _NSIG)

```

```

+ return -EINVAL;
+ sigemptyset(sig);
+ for (i=0; i<_NSIG; i++)
+   if (value[i] == '1')
+     sigaddset(sig, i);
+ return 0;
+}
+
+static char *next_eol(char *s, int maxlen)
+{
+   int len=0;
+   while (len<maxlen && *s!='\0' && *s !='\n')
+     s++, len++;
+   if (len < maxlen)
+     return s;
+   return NULL;
+}
+
+static int set_sigpending(struct task_struct *t, struct sigpending *pending,
+   char *value, int size)
+{
+   struct sigqueue *q;
+   int ret;
+   #ifdef __ARCH_SI_TRAPNO
+   int trapno;
+   #endif
+
+   if (set_sigset(&pending->signal, value, size))
+     return -EINVAL;
+
+   size -= _NSIG+1;
+   value += _NSIG+1;
+   while (size) {
+     int signum, uid, pid, flags, errno, code, status, fd;
+     unsigned long utime, stime, addr, band;
+     char *start, *eol = next_eol(value, size);
+
+     if (!eol)
+       return 0;
+
+     size -= (eol - value) + 1;
+     start = eol+1;
+     ret = sscanf(value, "sig %d: uid %d pid %d flags %d errno %d code %d\n",
+       &signum, &uid, &pid, &flags, &errno, &code);
+     if (ret != 6)
+       goto out;
+     q = __sigqueue_alloc(t, GFP_KERNEL, 1);
+     if (!q)

```

```

+ goto out;
+ q->info.si_signo = signum;
+ q->info.si_errno = errno;
+ q->info.si_code = code;
+ q->info.si_uid = uid;
+ q->info.si_pid = pid;
+
+ switch(signum) {
+ /* XXX skipping posix1b timers and signals for now */
+ default: break;
+ case SIGKILL:
+ eol = next_eol(start, size);
+ if (!eol)
+ goto out;
+ size -= (eol - start) + 1;
+ ret = sscanf(start, " pid %d uid %d\n", &pid, &uid);
+ if (ret != 2)
+ goto out;
+ q->info._sifields._kill._pid = pid;
+ q->info._sifields._kill._uid = uid;
+ break;
+
+ case SIGCHLD:
+ eol = next_eol(start, size);
+ if (!eol)
+ goto out;
+ size -= (eol - start) + 1;
+ ret = sscanf(start, "pid %d uid %d status %d utime %lu stime %lu\n",
+ &pid, &uid, &status, &utime, &stime);
+ if (ret != 5)
+ goto out;
+ q->info._sifields._sigchld._pid = pid;
+ q->info._sifields._sigchld._uid = uid;
+ q->info._sifields._sigchld._status = status;
+ q->info._sifields._sigchld._utime = utime;
+ q->info._sifields._sigchld._stime = stime;
+ break;
+
+ case SIGILL:
+ case SIGFPE:
+ case SIGSEGV:
+ case SIGBUS:
+ eol = next_eol(start, size);
+ if (!eol)
+ goto out;
+ size -= (eol - start) + 1;
+#ifdef __ARCH_SI_TRAPNO
+ ret = sscanf(start, " addr %lu trapno %d\n", &addr, &trapno);

```

```

+   if (ret != 2)
+       goto out;
+   q->info._sifields._sigfault._trapno = trapno;
+ #else
+   ret = sscanf(start, " addr %lu\n", &addr);
+   if (ret != 1)
+       goto out;
+ #endif
+   q->info._sifields._sigfault._addr = (void __user *)addr;
+   break;
+
+ case SIGPOLL:
+   eol = next_eol(start, size);
+   if (!eol)
+       goto out;
+   size -= (eol - start) + 1;
+   ret = sscanf(start, " band %ld fd %d\n", &band, &fd);
+   if (ret != 2)
+       goto out;
+   q->info._sifields._sigpoll._band = band;
+   q->info._sifields._sigpoll._fd = fd;
+   break;
+ }
+ start = eol+1;
+ list_add_tail(&q->list, &pending->list);
+ }
+
+ out:
+ return size;
+ }
+
+ static int set_sigaction_list(struct sighand_struct *sighand, char *value,
+   int size)
+ {
+   struct k_sigaction *action;
+   char *eol;
+   int num, ret;
+   int origsize = size;
+   //char sa_mask[_NSIG+1];
+   char sa_mask[65];
+   unsigned long sa_flags, sa_handler;
+
+   while (size > 0 && (eol=next_eol(value, size))) {
+       sa_mask[64] = '\0';
+       ret = sscanf(value, "%d %lu %64s %lu\n",
+         &num, &sa_flags, sa_mask, &sa_handler);
+       if (ret != 4)
+           break;

```



```

+ if (num < 0 || num > _NSIG)
+ return -EINVAL;
+ action = &sigband->action[num];
+ action->sa.sa_flags = sa_flags;
+ set_sigset(&action->sa.sa_mask, sa_mask, _NSIG);
+ action->sa.sa_handler = (__sighandler_t) sa_handler;
+ size -= eol-value+1;
+ value = eol+1;
+ }
+
+ return origsize - size;
+}
+
+static int set_altstack(struct task_struct *p, char *value, int size)
+{
+ unsigned long sas_ss_sp;
+ size_t sas_ss_size;
+ int ret;
+
+ ret = sscanf(value, "%lu %zd\n", &sas_ss_sp, &sas_ss_size);
+ if (ret != 2)
+ return -EINVAL;
+ p->sas_ss_sp = sas_ss_sp;
+ p->sas_ss_size = sas_ss_size;
+
+ return size;
+}
+
+int task_write_procsig(struct task_struct *p, char *name, void *value,
+ size_t size)
+{
+ int ret;
+
+ if (current != p) {
+ ret = security_ptrace(current, p);
+ if (ret)
+ return ret;
+ }
+
+ ret = -EINVAL;
+
+ if (strcmp(name, "pending") == 0) {
+ /* not masking out blocked signals yet */
+ ret = set_sigpending(p, &p->pending, value, size);
+ } else if (strcmp(name, "shared_pending") == 0) {
+ ret = set_sigpending(p, &p->signal->shared_pending, value, size);
+ } else if (strcmp(name, "blocked") == 0) {
+ /* not masking out blocked signals yet */

```

```
+ ret = set_sigset(&p->blocked, value, size);
+ } else if (strcmp(name, "action") == 0) {
+ ret = set_sigaction_list(p->sighand, value, size);
+ } else if (strcmp(name, "altstack") == 0) {
+ ret = set_altstack(p, value, size);
+ #if 0
+ } else if (strcmp(name, "waiters") == 0) {
+ /* reset this at the waiter's restart using do_wait */
+ ret = set_wait_info(p->signal, value, size);
+ #endif
+ }
+
+ return ret;
+}
--
1.5.1.1.GIT
```

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
