Subject: [PATCH 1/2] signal checkpoint: define /proc/pid/sig/
Posted by serue on Fri, 08 Jun 2007 15:54:37 GMT
View Forum Message <> Reply to Message

As I mentioned earlier, I don't know what sort of approach we want
to take to guide checkpoint and restart. I.e. do we want it to be
a mostly userspace-orchestrated affair, or entirely done in the
kernel using the freezer or some other mechanism in response to a
single syscall or containerfs file write?

If we wanted to do a lot of the work in userspace, here is a pair of
patches to read and restore signal information. It's entirely unsafe
wrt locking, bc i would assume that if we did in fact do c/r from
userspace, we would have some way of entirely pulling the task off
the runqueue while doing our thing...

Anyway, this is purely to start discussion.

thanks,
-serge

>From 30bbe322942e5ed86bad63861dad80595cd04063 Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Mon, 30 Apr 2007 16:22:44 -0400
Subject: [PATCH 1/2] signal checkpoint: define /proc/pid/sig/

Define /proc/<pid>/sig/ directory containing files to report
on a process' signal info.

Files defined:
 action: list signal action
 altstack: print sigaltstack location and size
 blocked: print blocked signal mask
 pending: print pending signals and siginfo
 shared_pending: print shared pending signals and siginfo
 waiters: list tasks wait4()ing on task PID.

TESTING:
In one terminal run 'forker' as compiled from forker.c below. In
another terminal, run 'sh checkforker.sh' also included below
which will send more signals to the forker, then check it's pending,
blocked, and shared_pending sig/ files.

==========
forker.c
==========

void runchild(void)

```
{
 printf("child exiting\n");
 exit(0);
}

int main()
{
 sigset_t blockset;
 sigset_t prevset;
 int pid;

 if (sigemptyset(&blockset) == -1) {
  perror("sigemptyset");
  return -1;
 }

 if (sigaddset(&blockset, SIGUSR1) == -1) {
  perror("sigaddset");
  return -1;
 }

 if (sigaddset(&blockset, SIGUSR2) == -1) {
  perror("sigaddset");
  return -1;
 }

 if (sigaddset(&blockset, SIGCHLD) == -1) {
  perror("sigaddset");
  return -1;
 }

 if (sigaddset(&blockset, SIGSEGV) == -1) {
  perror("sigaddset");
  return -1;
 }

 if (sigaddset(&blockset, SIGPOLL) == -1) {
  perror("sigaddset");
  return -1;
 }

 if (sigprocmask(SIG_SETMASK, &blockset, &prevset) == -1) {
  perror("sigprocmask");
  return -1;
 }

 pid = fork();
 if (pid < 0)
```

```
  perror("fork");
 if (pid == 0)
  runchild();
 sleep(30);
}
```

```
==========================================================================

==========================================================================
checkforker.sh
==========================================================================
p=`ps -ef | grep forker | head -1 | awk '{ print $2 }'`
kill -10 $p
kill -29 $p
kill -11 $p
echo pending
cat /proc/$p/sig/pending
echo blocked
cat /proc/$p/sig/blocked
echo shared pending
cat /proc/$p/sig/shared_pending
==========================================================================
```

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

```
---
 Documentation/filesystems/proc.txt |   59 +++++++++--
 fs/proc/base.c                     |   62 ++++++++++++
 include/linux/signal.h             |    1 +
 kernel/signal.c                    |  202 ++++++++++++++++++++++++++++++++++++++
 4 files changed, 312 insertions(+), 12 deletions(-)

diff --git a/Documentation/filesystems/proc.txt b/Documentation/filesystems/proc.txt
index 8756a07..1d74e19 100644
--- a/Documentation/filesystems/proc.txt
+++ b/Documentation/filesystems/proc.txt
@@ -133,6 +133,7 @@ Table 1-1: Process specific entries in /proc
  maps  Memory maps to executables and library files (2.4)
  mem  Memory held by this process
  root  Link to the root directory of this process
+ sig     Directory, which contains signal information
  stat  Process status
  statm  Process memory status information
  status  Process status in human readable form
@@ -188,16 +189,50 @@ Table 1-2: Contents of the statm files (as of 2.6.8-rc3)
  dt       number of dirty pages   (always 0 on 2.6)
 ..............................................................................

+Table 1-3: Contents of sig directory files
+..............................................................................
```

+ File Content
+ action List sigaction as "signum flags sigmask handler"
+ altstack List location and size of sigaltstack
+ blocked For each signal, print '1' if blocked, '0' otherwise
+ pending For each signal, print '1' if pending for this thread,
+ '0' otherwise
+ Also print additional information for certain given
+ pending signals. See 'pending' entry below for details.
+ shared_pending For each signal, print '1' if pending for all threads,
+ '0' otherwise.
+ waiters For each task doing wait on task PID, list the flags and
+ the waiting process' pid.
+.....
+
+The /proc/PID/sig/pending file prints additional pending signal information
+depending on the signal. See Table 1-4 for details by signal number.
+
+Table 1-4: Details of /proc/PID/sig/pending info for pending signals
+.....
+ Signal Information printed
+ SIGKILL Pid and uid of killing process
+
+ SIGCHLD Pid, uid, status, utime, and stime of exited child process
+
+ SIGILL
+ SIGFPE
+ SIGSEGV
+ SIGBUS Address and (if applicable to arch) trap number
+
+ SIGPOLL Band and fd
+.....
+
1.2 Kernel data

Similar to the process entries, the kernel data files give information about
the running kernel. The files used to obtain this information are contained in
-/proc and are listed in Table 1-3. Not all of these will be present in your
+/proc and are listed in Table 1-5. Not all of these will be present in your
system. It depends on the kernel configuration and the loaded modules, which
files are there, and which are missing.

-Table 1-3: Kernel info in /proc
+Table 1-5: Kernel info in /proc
.....
File Content
apm Advanced power management info
@@ -476,7 +511,7 @@ subdirectories. These are named ide0, ide1 and so on. Each of

these
directories contains the files shown in table 1-4.

-Table 1-4: IDE controller info in /proc/ide/ide?
+Table 1-6: IDE controller info in /proc/ide/ide?
.....
File Content
channel IDE channel (0 or 1)
@@ -490,7 +525,7 @@ controllers directory. The files listed in table 1-5 are contained in these
directories.

-Table 1-5: IDE device information
+Table 1-7: IDE device information
.....
File Content
cache The cache
@@ -532,12 +567,12 @@ the drive parameters:
1.4 Networking info in /proc/net

-The subdirectory /proc/net follows the usual pattern. Table 1-6 shows the
+The subdirectory /proc/net follows the usual pattern. Table 1-8 shows the
additional values you get for IP version 6 if you configure the kernel to
-support this. Table 1-7 lists the files and their meaning.
+support this. Table 1-9 lists the files and their meaning.

-Table 1-6: IPv6 info in /proc/net
+Table 1-8: IPv6 info in /proc/net
.....
File Content
udp6 UDP sockets (IPv6)
@@ -552,7 +587,7 @@ Table 1-6: IPv6 info in /proc/net
.....

-Table 1-7: Network info in /proc/net
+Table 1-9: Network info in /proc/net
.....
File Content
arp Kernel ARP table
@@ -676,10 +711,10 @@ The directory /proc/parport contains information about the parallel
ports of
your system. It has one subdirectory for each port, named after the port
number (0,1,2,...).

-These directories contain the four files shown in Table 1-8.
+These directories contain the four files shown in Table 1-10.

-Table 1-8: Files in /proc/parport
+Table 1-10: Files in /proc/parport
 .....
  File   Content
  autoprobe Any IEEE-1284 device ID information that has been acquired.
@@ -697,10 +732,10 @@ Table 1-8: Files in /proc/parport

 Information about the available and actually used tty's can be found in the
 directory /proc/tty.You'll find entries for drivers and line disciplines in
-this directory, as shown in Table 1-9.
+this directory, as shown in Table 1-11.

-Table 1-9: Files in /proc/tty
+Table 1-11: Files in /proc/tty
 .....
  File   Content
  drivers   list of drivers and their usage
diff --git a/fs/proc/base.c b/fs/proc/base.c
index a5fa1fd..8e0dd7a 100644
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -1681,6 +1681,66 @@ out_no_task:
  return ret;
 }

+/* /proc/<pid>/sig/ */
+static ssize_t proc_pid_sig_read(struct file * file, char __user * buf,
+   size_t count, loff_t *ppos)
+{
+ struct dentry *dentry = file->f_path.dentry;
+ struct inode * inode = dentry->d_inode;
+ char *p = NULL;
+ ssize_t length;
+ struct task_struct *task = get_proc_task(inode);
+
+ if (!task)
+ return -ESRCH;
+ length = task_read_procsig(task, (char*)dentry->d_name.name, &p);
+ put_task_struct(task);
+ if (length > 0)
+ length = simple_read_from_buffer(buf, count, ppos, p, length);
+ kfree(p);
+ return length;

```
+}
+
+static const struct file_operations proc_sig_attr_operations = {
+ .read  = proc_pid_sig_read,
+// .write  = proc_pid_sig_write,
+};
+
+
+static struct pid_entry sig_dir_stuff[] = {
+ REG("pending",           S_IRUGO|S_IWUGO, sig_attr),
+ REG("shared_pending",    S_IRUGO|S_IWUGO, sig_attr),
+ REG("blocked",           S_IRUGO|S_IWUGO, sig_attr),
+ REG("action",            S_IRUGO|S_IWUGO, sig_attr),
+ REG("waiters",           S_IRUGO|S_IWUGO, sig_attr),
+ REG("altstack",          S_IRUGO|S_IWUGO, sig_attr),
+};
+
+static int proc_sig_dir_readdir(struct file * filp,
+        void * dirent, filldir_t filldir)
+{
+ return proc_pident_readdir(filp,dirent, filldir,
+       sig_dir_stuff, ARRAY_SIZE(sig_dir_stuff));
+}
+
+static const struct file_operations proc_sig_dir_operations = {
+ .read  = generic_read_dir,
+ .readdir = proc_sig_dir_readdir,
+};
+
+static struct dentry *proc_sig_dir_lookup(struct inode *dir,
+    struct dentry *dentry, struct nameidata *nd)
+{
+ return proc_pident_lookup(dir, dentry,
+      sig_dir_stuff, ARRAY_SIZE(sig_dir_stuff));
+}
+
+static const struct inode_operations proc_sig_dir_inode_operations = {
+ .lookup  = proc_sig_dir_lookup,
+ .getattr = pid_getattr,
+ .setattr = proc_setattr,
+};
+
 #ifdef CONFIG_SECURITY
 static ssize_t proc_pid_attr_read(struct file * file, char __user * buf,
       size_t count, loff_t *ppos)
@@ -2006,6 +2066,7 @@ static const struct pid_entry tgid_base_stuff[] = {
 #ifdef CONFIG_TASK_IO_ACCOUNTING
  INF("io", S_IRUGO, pid_io_accounting),
```

```
 #endif
+ DIR("sig",        S_IRUGO|S_IXUGO, sig_dir),
 };

 static int proc_tgid_base_readdir(struct file * filp,
@@ -2286,6 +2347,7 @@ static const struct pid_entry tid_base_stuff[] = {
 #ifdef CONFIG_FAULT_INJECTION
  REG("make-it-fail", S_IRUGO|S_IWUSR, fault_inject),
 #endif
+ DIR("sig",        S_IRUGO|S_IXUGO, sig_dir),
 };

 static int proc_tid_base_readdir(struct file * filp,
diff --git a/include/linux/signal.h b/include/linux/signal.h
index 9a5eac5..6863543 100644
--- a/include/linux/signal.h
+++ b/include/linux/signal.h
@@ -243,6 +243,7 @@ struct pt_regs;
 extern int get_signal_to_deliver(siginfo_t *info, struct k_sigaction *return_ka, struct pt_regs *regs,
void *cookie);

 extern struct kmem_cache *sighand_cachep;
+extern int task_read_procsig(struct task_struct *p, char *name, char **value);

 /*
  * In POSIX a signal is sent either to a specific thread (Linux task)
diff --git a/kernel/signal.c b/kernel/signal.c
index 364fc95..a2a3ebe 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -2537,3 +2537,205 @@ void __init signals_init(void)
 {
  sigqueue_cachep = KMEM_CACHE(sigqueue, SLAB_PANIC);
 }
+
+/*
+ * print a sigset_t to a buffer.  Return # characters printed,
+ * not including the final ending '\0'.
+ */
+static int print_sigset(char *buf, sigset_t *sig)
+{
+ int i;
+
+ for (i=0; i<_NSIG; i++) {
+  if (sigismember(sig, i))
+   buf[i] = '1';
+  else
+   buf[i] = '0';
```

```
+ }
+ buf[_NSIG] = '\0';
+
+ return _NSIG;
+}
+
+static int print_sigset_alloc(char **bufp, sigset_t *sig)
+{
+ char *buf;
+
+ *bufp = buf = kmalloc(_NSIG+1, GFP_KERNEL);
+ if (!buf)
+  return -ENOMEM;
+
+ return print_sigset(buf, sig);
+}
+
+static int print_wait_info(char **bufp, struct signal_struct *signal)
+{
+ char *buf;
+ wait_queue_t *wait;
+
+ *bufp = buf = kmalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!buf)
+  return -ENOMEM;
+
+ spin_lock(&signal->wait_chldexit.lock);
+
+ list_for_each_entry(wait, &signal->wait_chldexit.task_list, task_list) {
+  struct task_struct *tsk = wait->private;
+
+  if (buf - *bufp +50 > PAGE_SIZE) {
+   spin_unlock(&signal->wait_chldexit.lock);
+   kfree(buf);
+   *bufp = NULL;
+   return -ENOMEM;
+  }
+  WARN_ON(wait->func != default_wake_function);
+  buf += sprintf(buf, "%u %d\n", wait->flags, tsk->pid);
+ }
+
+ spin_unlock(&signal->wait_chldexit.lock);
+
+ return buf-*bufp;
+}
+
+static int print_sigpending_alloc(char **bufp, struct sigpending *pending)
+{
```

```
+ int alloced=0;
+ char *buf, *p;
+ struct sigqueue *q;
+ struct siginfo *info;
+
+ alloced = PAGE_SIZE;
+ p = buf = kmalloc(alloced, GFP_KERNEL);
+ if (!buf)
+  return -ENOMEM;
+
+ p += print_sigset(buf, &pending->signal);
+ p += sprintf(p, "\n");
+
+ list_for_each_entry(q, &pending->list, list) {
+  info = &q->info;
+  if (p-buf+215 > alloced) {
+   int len=p-buf;
+   char *buf2;
+   alloced += PAGE_SIZE;
+   buf2 = kmalloc(alloced, GFP_KERNEL);
+   if (!buf2) {
+    kfree(buf);
+    return -ENOMEM;
+   }
+   memcpy(buf2, buf, alloced - PAGE_SIZE);
+   kfree(buf);
+   buf = buf2;
+   p = buf+len;
+  }
+
+  p += sprintf(p, "sig %d: user %d flags %d",
+   info->si_signo, (int)q->user->uid, q->flags);
+  p += sprintf(p, " errno %d code %d\n",
+   info->si_errno, info->si_code);
+
+  switch(info->si_signo) {
+  case SIGKILL:
+   p += sprintf(p, "  spid %d suid %d\n",
+    info->_sifields._kill._pid,
+    info->_sifields._kill._uid);
+   break;
+  /* XXX skipping posix1b timers and signals for now */
+  case SIGCHLD:
+   p += sprintf(p, "  pid %d uid %d status %d utime %lu stime %lu\n",
+    info->_sifields._sigchld._pid,
+    info->_sifields._sigchld._uid,
+    info->_sifields._sigchld._status,
+    info->_sifields._sigchld._utime,
```

```
+    info->_sifields._sigchld._stime);
+   break;
+  case SIGILL:
+  case SIGFPE:
+  case SIGSEGV:
+  case SIGBUS:
+#ifdef __ARCH_SI_TRAPNO
+   p += sprintf(p, "  addr %lu trapno %d\n",
+    (unsigned long)info->_sifields._sigfault._addr,
+    info->_sifields._sigfault._trapno);
+#else
+   p += sprintf(p, "  addr %lu\n",
+    (unsigned long)info->_sifields._sigfault._addr);
+#endif
+   break;
+  case SIGPOLL:
+   p += sprintf(p, "  band %ld fd %d\n",
+    (long)info->_sifields._sigpoll._band,
+    info->_sifields._sigpoll._fd);
+   break;
+  default:
+   p += sprintf(p, "  Unsupported siginfo for signal %d\n",
+    info->si_signo);
+   break;
+  }
+ }
+ *bufp = buf;
+ return p-buf;
+}
+
+static int print_sigaction_list(char **bufp, struct sighand_struct *sighand)
+{
+ struct k_sigaction *action;
+ int maxlen;
+ int i;
+ char *buf;
+
+ /* two unsigned longs (20 chars), one int (10 chars), a sigset_t, 3 spaces, plus a newline */
+ maxlen = 10 + 20*2 + _NSIG + 4;
+ /* and we have _NSIG of those entries, plus an ending \0 */
+ maxlen *= _NSIG+1;
+
+ *bufp = buf = kmalloc(maxlen, GFP_KERNEL);
+ if (!buf)
+  return -ENOMEM;
+
+ spin_lock(&sighand->siglock);
+ for (i=0; i<_NSIG; i++) {
```

```
+  action = &sighand->action[i];
+  buf += sprintf(buf, "%10d %20lu ", i, action->sa.sa_flags);
+  buf += print_sigset(buf, &action->sa.sa_mask);
+  buf += sprintf(buf, " %20lu\n", (unsigned long)action->sa.sa_handler);
+  BUG_ON(buf-*bufp > maxlen);
+ }
+ spin_unlock(&sighand->siglock);
+ return buf-*bufp;
+}
+
+int task_read_procsig(struct task_struct *p, char *name, char **value)
+{
+ 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 = print_sigpending_alloc(value, &p->pending);
+ } else if (strcmp(name, "blocked") == 0) {
+  /* not masking out blocked signals  yet */
+  ret = print_sigset_alloc(value, &p->blocked);
+ } else if (strcmp(name, "shared_pending") == 0) {
+  ret = print_sigpending_alloc(value, &p->signal->shared_pending);
+ } else if (strcmp(name, "waiters") == 0) {
+  ret = print_wait_info(value, p->signal);
+ } else if (strcmp(name, "action") == 0) {
+  ret = print_sigaction_list(value, p->sighand);
+ }  else if (strcmp(name, "altstack") == 0) {
+  *value = kmalloc(40, GFP_KERNEL);
+  ret = -ENOMEM;
+  if (*value) {
+   ret = sprintf(*value, "%lu %zd", p->sas_ss_sp, p->sas_ss_size);
+  }
+ }
+
+ return ret;
+}
--
1.5.1.1.GIT
```

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