

---

Subject: [RFC][PATCH][cryo] Save/restore state of unnamed pipes  
Posted by [Sukadev Bhattiprolu](#) on Tue, 17 Jun 2008 21:29:50 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

>From fd13986de32af31621b1badbcf7bfb5626648e0e Mon Sep 17 00:00:00 2001  
From: Sukadev Bhattiprolu <sukadev@linux.vnet.ibm.com>  
Date: Mon, 16 Jun 2008 18:41:05 -0700  
Subject: [PATCH] Save/restore state of unnamed pipes

Design:

Current Linux kernels provide ability to read/write contents of FIFOs using /proc. i.e 'cat /proc/pid/fd/read-side-fd' prints the unread data in the FIFO. Similarly, 'cat foo > /proc/pid/fd/read-side-fd' appends the contents of 'foo' to the unread contents of the FIFO.

So to save/restore the state of the pipe, a simple implementation is to read the from the unnamed pipe's fd and save to the checkpoint-file. When restoring, create a pipe (using PT\_PIPE()) in the child process, read the contents of the pipe from the checkpoint file and write it to the newly created pipe.

Its fairly straightforward, except for couple of notes:

- when we read contents of '/proc/pid/fd/read-side-fd' we drain the pipe such that when the checkpointed application resumes, it will not find any data. To fix this, we read from the 'read-side-fd' and write it back to the 'read-side-fd' in addition to writing to the checkpoint file.
- there does not seem to be a mechanism to determine the count of unread bytes in the file. Current implmentation assumes a maximum of 64K bytes (PIPE\_BUFS \* PAGE\_SIZE on i386) and fails if the pipe is not fully drained.

Basic unit-testing done at this point (using tests/pipe.c).

TODO:

- Additional testing (with multiple-processes and multiple-pipes)
- Named-pipes

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

---  
cr.c | 215 +++  
1 files changed, 203 insertions(+), 12 deletions(-)

diff --git a/cr.c b/cr.c  
index 5163a3d..0cb9774 100644

```

--- a/cr.c
+++ b/cr.c
@@ -84,6 +84,11 @@ typedef struct fdinfo_t {
    char name[128]; /* file name. NULL if anonymous (pipe, socketpair) */
} fdinfo_t;

+typedef struct fifoinfo_t {
+ int fi_fd; /* fifo's read-side fd */
+ int fi_length; /* number of bytes in the fifo */
+} fifofdinfo_t;
+
+typedef struct memseg_t {
    unsigned long start; /* memory segment start address */
    unsigned long end; /* memory segment end address */
@@ -468,6 +473,128 @@ out:
    return rc;
}

+static int estimate_fifo_unread_bytes(pinfo_t *pi, int fd)
+{
+ /*
+ * Is there a way to find the number of bytes remaining to be
+ * read in a fifo ? If not, can we print it in fdinfo ?
+ *
+ * Return 64K (PIPE_BUFS * PAGE_SIZE) for now.
+ */
+ return 65536;
+}
+
+static void ensure_fifo_has_drained(char *fname, int fifo_fd)
+{
+ int rc, c;
+
+ rc = read(fifo_fd, &c, 1);
+ if (rc != -1 && errno != EAGAIN) {
+ ERROR("FIFO '%s' not drained fully. rc %d, c %d "
+ "errno %d\n", fname, rc, c, errno);
+ }
+
+}
+
+static int save_process_fifo_info(pinfo_t *pi, int fd)
+{
+ int i;
+ int rc;
+ int nbytes;
+ int fifo_fd;
+ int pbuf_size;

```

```

+ pid_t pid = pi->pid;
+ char fname[256];
+ fdinfo_t *fi = pi->fi;
+ char *pbuf;
+ fifofdinfo_t fifofdinfo;
+
+ write_item(fd, "FIFO", NULL, 0);
+
+ for (i = 0; i < pi->nf; i++) {
+ if (! S_ISFIFO(fi[i].mode))
+ continue;
+
+ DEBUG("FIFO fd %d (%s), flag 0x%x\n", fi[i].fdnum, fi[i].name,
+ fi[i].flag);
+
+ if (!(fi[i].flag & O_WRONLY))
+ continue;
+
+ pbuf_size = estimate_fifo_unread_bytes(pi, fd);
+
+ pbuf = (char *)malloc(pbuf_size);
+ if (!pbuf) {
+ ERROR("Unable to allocate FIFO buffer of size %d\n",
+ pbuf_size);
+ }
+ memset(pbuf, 0, pbuf_size);
+
+ sprintf(fname, "/proc/%u/fd/%u", pid, fi[i].fdnum);
+
+ /*
+ * Open O_NONBLOCK so read does not block if fifo has fewer
+ * bytes than our estimate.
+ */
+ fifo_fd = open(fname, O_RDWR|O_NONBLOCK);
+ if (fifo_fd < 0)
+ ERROR("Error %d opening FIFO '%s'\n", errno, fname);
+
+ nbytes = read(fifo_fd, pbuf, pbuf_size);
+ if (nbytes < 0) {
+ if (errno != EAGAIN) {
+ ERROR("Error %d reading FIFO '%s'\n", errno,
+ fname);
+ }
+ }
+ nbytes = 0; /* empty fifo */
+ }
+
+ /*
+ * Ensure FIFO has been drained.

```

```

+ *
+ * TODO: If FIFO has not fully drained, our estimate of
+ * unread-bytes is wrong. We could:
+ *
+ * - have kernel print exact number of unread-bytes
+ *   in /proc/pid/fdinfo/<fd>
+ *
+ * - read in contents multiple times and write multiple
+ *   fifobufs or assemble them into a single, large
+ *   buffer.
+ */
+ ensure_fifo_has_drained(fname, fifo_fd);
+
+ /*
+ * Save FIFO data to checkpoint file
+ */
+ fifofdinfo.fi_fd = fi[i].fdnum;
+ fifofdinfo.fi_length = nbytes;
+ write_item(fd, "fifofdinfo", &fifofdinfo, sizeof(fifofdinfo));
+
+ if (nbytes) {
+   write_item(fd, "fifobufs", pbuf, nbytes);
+ }
+ /*
+ * Restore FIFO's contents so checkpointed application
+ * won't miss a thing.
+ */
+ errno = 0;
+ rc = write(fifo_fd, pbuf, nbytes);
+ if (rc != nbytes) {
+   ERROR("Wrote-back only %d of %d bytes to FIFO, "
+         "error %d\n", rc, nbytes, errno);
+ }
+ }
+
+ close(fifo_fd);
+ free(pbuf);
+ }
+
+ write_item(fd, "END FIFO", NULL, 0);
+
+ return 0;
+}
+
+static int save_process_data(pid_t pid, int fd, lh_list_t *ptree)
+{
+   char fname[256], exe[256], cwd[256], *argv, *env, *buf;
+@@ -587,6 +714,8 @@ static int save_process_data(pid_t pid, int fd, lh_list_t *ptree)

```

```

}
write_item(fd, "END FD", NULL, 0);

+ save_process_fifo_info(pi, fd);
+
+ /* sockets */
+ write_item(fd, "SOCK", NULL, 0);
+ for (i = 0; i < pi->ns; i++)
@@ -839,6 +968,29 @@ int restore_fd(int fd, pid_t pid)
+ }
+ if (pfd != fdinfo->fdnum) t_d(PT_CLOSE(pid, pfd));
+ }
+ } else if (S_ISFIFO(fdinfo->mode)) {
+ int pipefds[2] = { 0, 0 };
+
+ /*
+ * We create the pipe when we see the pipe's read-fd.
+ * Just ignore the pipe's write-fd.
+ */
+ if (fdinfo->flag == O_WRONLY)
+ continue;
+
+ DEBUG("Creating pipe for fd %d\n", fdinfo->fdnum);
+
+ t_d(PT_PIPE(pid, pipefds));
+ t_d(pipefds[0]);
+ t_d(pipefds[1]);
+
+ if (pipefds[0] != fdinfo->fdnum) {
+ DEBUG("Hmm, new pipe has fds %d, %d "
+ "Old pipe had fd %d\n", pipefds[0],
+ pipefds[1], fdinfo->fdnum); getchar();
+ exit(1);
+ }
+ DEBUG("Done creating pipefds[0] %d\n", pipefds[0]);
+ }

+ /*
@@ -847,20 +999,8 @@ int restore_fd(int fd, pid_t pid)
+ ret = PT_FCNTL(pid, fdinfo->fdnum, F_SETFL, fdinfo->flag);
+ DEBUG("---- restore_fd() fd %d setfl flag 0x%x, ret %d\n",
+ fdinfo->fdnum, fdinfo->flag, ret);
-
-
+ free(fdinfo);
+ }
- if (1) {
- /* test: force pipe creation */

```

```

- static int first = 1;
- int pipe[2] = { 0, 0 };
- if (! first) return 0;
- else first = 0;
- t_d(PT_PIPE(pid, pipe));
- t_d(pipe[0]);
- t_d(pipe[1]);
- }
  return 0;
error:
  free(fdinfo);
@@ -1231,6 +1371,55 @@ int restore_sig(pid_t pid, struct sigaction *sigact, sigset_t *sigmask,
sigset_t
  return 0;
}

```

```

+int restore_fifo(int fd, pid_t pid)
+{
+ char item[64];
+ void *buf = NULL;
+ size_t bufsz;
+ int ret;
+ int fifo_fd;
+ char fname[64];
+ int nbytes;
+ fifofdinfo_t *fifofdinfo = NULL;
+
+ for(;;) {
+ ret = read_item(fd, item, sizeof(item), &buf, &bufsz);
+ DEBUG("restore_fifo() read item '%.12s'\n", item);
+ if ITEM_IS("END FIFO")
+ break;
+ else ITEM_SET(fifofdinfo, fifofdinfo_t);
+ else if ITEM_IS("fifobufs") {
+ DEBUG("restore_fifo() bufsz %d, fi_fd %d, length %d\n",
+ bufsz, fifofdinfo->fi_fd,
+ fifofdinfo->fi_length);
+
+ if (!fifofdinfo->fi_length)
+ continue;
+
+ sprintf(fname, "/proc/%u/fd/%d", pid,
+ fifofdinfo->fi_fd);
+
+ fifo_fd = open(fname, O_WRONLY|O_NONBLOCK);
+ if (fifo_fd < 0) {
+ ERROR("Error %d opening FIFO '%s'\n", errno,
+ fname);

```

```

+ }
+
+ errno = 0;
+ nbytes = write(fifo_fd, buf, bufsz);
+ if (nbytes != bufsz) {
+   ERROR("Error %d writing to FIFO '%s'\n",
+     errno, fname);
+ }
+ close(fifo_fd);
+ } else
+   ERROR("Unexpected item, '%s'\n", item);
+ }
+ DEBUG("restore_fifo() fd %d, len %d, got 'END FIFO'\n",
+   fifofdinfo->fi_fd, fifofdinfo->fi_length);
+ return 0;
+}
+
static int process_restart(int fd, int mode)
{
  char item[64];
@@ -1314,6 +1503,8 @@ static int process_restart(int fd, int mode)
  ptrace_set_thread_area(npid, ldt);
  if (cwd) PT_CHDIR(npid, cwd);
  restore_fd(fd, npid);
+ } else if (ITEM_IS("FIFO")) {
+   restore_fifo(fd, npid);
  } else if (ITEM_IS("SOCK")) {
    restore_sock(fd, npid);
  } else if (ITEM_IS("SEMUNDO")) {
--

```

1.5.2.5

---

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

---