Subject: Re: [RFC][PATCH][cryo] Save/restore state of unnamed pipes
Posted by Matt Helsley on Wed, 18 Jun 2008 19:57:35 GMT
View Forum Message <> Reply to Message

On Wed, 2008-06-18 at 11:00 -0700, sukadev@us.ibm.com wrote:
> Matt Helsley [matthltc@us.ibm.com] wrote:
>
> | > |
> | > |
> | > | pipe(pipefds); /* returns 5 and 4 in elements 0 and 1 */
> | > | /* use fds after last_fd as trampolines for fds we want to create */
> | > | dup2(pipefds[0], last_fd + 1);
> | > | dup2(pipefds[1], last_fd + 2);
> | > | close(pipefds[0]);
> | > | close(pipefds[1]);
> | > | dup2(last_fd + 1, <orig pipefd[0]>);
> | > | dup2(last_fd + 2, <orig pipefd[1]>);
> | > | close(last_fd + 1);
> | > | close(last_fd + 2);
> | > |
> | > |
> | > | Which is alot more code but should work no matter which fds we get back
> | > | from pipe(). Of course this assumes the checkpointed application hasn't
> | > | used all of its fds. :(
> | > |
> | >
> | > This sounds like a good idea too, but we could use any fd that has not
> | > yet been used in the restart-process right ? It would break if all fds
> |
> | Yes, but we don't know which fd is available unless we allocate it
> | without dup2(). Here's how it could be done without last_fd (again,
> | dropping PT_FUNC notation):
> |
> | /*
> | * Move fds from src to dest. Useful for correctly "moving" pipe fds and
> | * other cases where we have a small number of fds to move to their
> | * original fd.
> | *
> | * Assumes dest_fds and src_fds are of the same, small length since
> | * this is O(num_fds^2).
> | *
> | * If num_fds == 1 then use plain dup2().
> | *
> | * Use this in place of multiple dup2() calls (num_fds > 1) unless you are
> | * absolutely certain the set of dest fds do not intersect the set of src fds.
> | * Does NOT magically prevent you from accidentally clobbering fds outside the
> | * src_fds array.
> | */

> | void move_fds(int *dest_fds, int *src_fds, const unsigned int num_fds)
> | {
> | int i;
> | unsigned int num_moved = 0;
> |
> | for (i = 0; i < num_fds; i++) {
> | int j;
> |
> | if (src_fds[i] == dest_fds[i])
> | continue; /* nothing to be done */
> |
> | /* src fd != dest fd so we need to perform:
> | dup2(src fd, dest fd);
> | but dup2() closes dest fd if it already exists.
> | This means we could accidentally close one of
> | the src fds. Avoid this by searching for any
> | src fd == dest fd and dup()'ing src fd to
> | a different fd so we can use the dest fd.
> | */
> | for (j = i + 1; j < num_fds; j++) /* This makes us O(N^2) */
> | if (dest_fds[i] == src_fds[j])
> | /*
> | * we're using an fd for something
> | * else already -- we need a trampoline
> | */
>
> So let me rephrase the problem.
>
> Suppose the checkpointed application was using fds in following
> "orig-fd-set"
>
> { [0..10], 18, 27 }
>
> where 18 and 27 are part of a pipe. For simplicity lets assume that
> 18 is the read-side-fd.

so orig_pipefd[0] == 18
and orig_pipefd[1] == 27

> We checkpointed this application and are now trying to restart it.
>
> In the restarted application, we would call
>
> dup2(fd1, fd2),
>
> where 'fd1' is some new, random fd and 'fd2' is an fd in 'orig-fd-set'
                        ^^^^^^ Even if they were truly random, this
does not preclude fd1 from having the same value as an fd in the

remaining orig-fd-set -- such as one of the two we're about to try and
restart with pipe().

> (say fd2 = 18).

fd1 = restarted_pipefd[0]
fd2 = restarted_pipefd[1]

In my example fd1 == 27 and fd2 == 18

> IIUC, there is a risk here of 'fd2' being closed accidentally while
> it is in use.

Yes, that's the risk.

> But, the only way I can see 'fd2' being in use in the restarted process
> is if _cryo_ opened some file _during_ restart and did not close. I ran

Both file descriptors returned from pipe() are in use during restart
and closing one of them would not be proper. Cryo hasn't "forgotten" to
close one of them -- cryo needs to dup2() both of them to their
"destination" fds. But if they have been swapped or if just one is the
"destination" of the other then you could end up with a broken pipe.

> into this early on with the randomize_va_space file (which was easily
> fixed).

This logic only works if cryo only has one new fd at a time. However
that's not possible with pipe(). Or socketpair(). In those cases one of
the two new fds could be the "destination" fd for the other. In that
case dup2() will kindly close it for you and break your new
pipe/socketpair! :)

That's why I asked if POSIX guarantees the read side file descriptor is
always less than the write side. If it does then the numbers can't be
swapped and maybe using your assumption that we don't have any other fds
accidentally left open ensures dup2() will be safe.

> Would cryo need to keep one or more temporary/debug files open in the
> restarted process (i.e files that are not in the 'orig-fd-set').

There's no need to keep temporary/debug files open that I can see. Just
a need to be careful when more than one new file descriptor has been
created before doing a dup2().

> If cryo does, then maybe it could open such files:
>
> - after clone() (so files are not open in restarted process), or

>
> - find the last_fd used and dup2() to that fd, leaving the
>   'orig-fd-set' all open/available for restarted process
>
> For debug, before each 'dup2(fd1, fd2)' we could 'fstat(fd2, &buf)'
> to ensure 'fd2' is not in use and error out if it is.

fstat() could certainly be useful for debugging dup2(). However it still doesn't nicely show us whether there are any fds we've leaked that we forgot about unless we fstat() all possible fds and then compare the set of existing fds to the orig-fd-set.

Cheers,
 -Matt

_____