Subject: Re: [RFC][PATCH][cryo] Save/restore state of unnamed pipes
Posted by Matt Helsley on Wed, 18 Jun 2008 22:38:09 GMT
View Forum Message <> Reply to Message

On Wed, 2008-06-18 at 14:56 -0700, sukadev@us.ibm.com wrote:
> Matt Helsley [matthltc@us.ibm.com] wrote:
> |
> | > 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().
>
> I agree. fd1 could be an hither-to-unseen fd from the 'orig-fd-set'.
>
> |
> | > (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.
>
> Ok I see what you are saying.
>
> The assumption I have is that we would process the fds from 'orig-fd-set'
> in ascending order. Its good to confirm that assumption now :-)

OK, but wouldn't that violate the "pipes first" ordering we discussed?

> proc_readfd_common() seems to return the fds in ascending order (so
> readdir() of "/proc/pid/fd/" would get them in ascending order - no ?)
>
> If we process 'orig-fd-set' in order and suppose we create the pipe for
> the smaller of the two fds (could be the write-side). Then the other side
> of the pipe would either not collide with an existing fd or that
> fd would not be in the 'orig-fd-set' (in the latter case it would
> be safe for dup2() to close).

Hmm, I don't see how that solves the problem.
Example:

Suppose fds 0-10 are already "restarted", 11 and 12 are the read and
write ends of a pipe we need to restart.

Now restart does :

```
int pipefds[2];

pipe(pipefds); /*
  * kernel is allowed to return pipefds[0] == 12 and
   * pipefds[1] == 11
  */

dup2(pipefds[0], 11); /* closes pipefds[1]! */
dup2(pipefds[1], 12);
```

So even though we're processing the fds in the orig-fd-set in order, and
regardless of how we completed restarting the earlier fds, we'd still
have this problem. Note that if the write end of the pipe should be fd
200 you'd still have this problem. The swap case is the nastiest though
because no matter whicg order we do the dup2() calls we'd break our
shiny new pipe().

move_fds() solves the problem by brute force checking and dup()'ing (not dup2) any problematic fds to new, guaranteed-safe-for-the-moment, fds.

Cheers,
-Matt

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