Forking subprocesses in Perl unit tests stops prove; Test::Harness exiting

Asked 16 years, 2 months ago Modified 3 years, 6 months ago Viewed 2k times

14

I have been trying to use the Perl utility/module "prove" as a test harness for some unit tests. The unit tests are a little more "system" than "unit" as I need to fork off some background processes as part of the test, Using the following...

```
sub SpinupMonitor{
   my $base_dir = shift;
   my $config = shift;

   my $pid = fork();
   if($pid){
      return $pid;
   }else{
      my $cmd = "$base_dir\/..\/bin\/monitor_real.pl -config $config -test";
      close STDOUT;

      exec ($cmd) or die "cannot exec test code [$cmd]\n";
   }
}

sub KillMonitor{

   my $pid = shift;

   print "Killing monitor [$pid]\n";
   kill(1,$pid);
}
```

However for some reason when I have my .t file spin up some extra processes it causes the test harness to hang at the end of the first .t file after all the tests have finished, rather than going on to the next file, or exiting if there is only one.

At first I wondered if it might be because I was killing of my sub-processes and leaving them defunct. So I added..

```
$SIG{CHLD} = \&REAPER;
sub REAPER {
   my $pid = wait;
   $SIG{CHLD} = \&REAPER;
}
```

To the code. But that doesn't help. In fact on closed examination it turns out that my perl test file has exited and is now a defunct process and it is the prove wrapper script

that has not reaped its child. In fact when I added a die() call at the end of my test script I got...

```
# Looks like your test died just after 7.
```

So my script exited but for some reason the harness isn't unraveling.

I did confirm that it is definitely my sub-processes that are upsetting it as when I disabled them while the tests failed the harness exited properly.

Is there anything I am doing wrong with the way I am starting up my processes that might upset the harness in some way?

perl testing fork die perl-prove

Share Improve this question Follow

edited Jun 1, 2021 at 17:22

brian d foy
132k ● 31 ● 211 ● 604

asked Oct 8, 2008 at 16:07

Vagnerr
2,997 ● 4 ● 35 ● 46

2 Answers

Sorted by: Highest score (default)

12

Note that you don't test whether `fork()` failed. You need to make sure `$pid` is defined before assuming that "false" means "child."

Because your `$cmd` contains shell metacharacters (spaces), Perl is actually using a shell when you call `exec()`. While your monitor is running, there's (1) Perl, (2) a child `sh -c`, and (3) a grandchild Perl running `monitor_real.pl`. What that means in particular is when you call `KillMonitor`, you're only killing the shell (because that's the PID you have) and not the monitor.

You might also be interested in How do I fork a daemon process? from the Perl FAQ.

Share Improve this answer Follow

answered Oct 8, 2008 at 16:32

Kyle
867 ● 5 ● 5

8

I'm assuming that all your kids have exited before you leave your test? Because otherwise, it may be hanging on to STDERR, which may confuse prove. If you could close STDERR, or at least redirect to a pipe in your parent process, that may be one issue you're having.

Besides that, I'd also point out that you don't need to escape forward slashes, and if you're not using shell metacharacters (spaces are not metacharacters to perl - think "`*?{}()`"), you should be explicit and create a list:

```
use File::Spec;
my @cmd = File::Spec->catfile($basedir,
                              File::Spec->updir(),
                              qw(bin monitor_real.pl)
                             ),
          -config => $config,
          -test   =>;

close STDOUT;
close STDERR;

exec (@cmd) or die "cannot exec test code [@cmd]\n";
```

Share Improve this answer Follow

answered Oct 8, 2008 at 21:07

Tanktalus
22.3k ● 5 ● 43 ● 70

Thanks Tanktalus it turns out all I needed was to close STDERR as well as STDOUT. :-)
– Vagnerr Oct 8, 2008 at 23:21