NAME

MooseX::Daemonize::Core − A Role with the core daemonization features

VERSION

version 0.21

SYNOPSIS

```
package My::Daemon;
use Moose;

with 'MooseX::Daemonize::Core';

sub start {
    my $self = shift;
    # daemonize me ...
    $self->daemonize;
    # return from the parent,...
    return unless $self->is_daemon;
    # but continue on in the child (daemon)
}
```

DESCRIPTION

This is the basic daemonization Role, it provides a few methods (see below) and the minimum features needed to properly daemonize your code.

Important Notes

None of the methods in this role will exit the parent process for you, it only forks and detaches your child (daemon) process. It is your responsibility to exit the parent process in some way.

There is no PID or PID file management in this role, that is your responsibility (see some of the other roles in this distro for that).

ATTRIBUTES

is_daemon (is = rw, isa => Bool)>

This attribute is used to signal if we are within the daemon process or not.

no_double_fork (is = rw, isa => Bool)>

Setting this attribute to true will cause this method to not perform the typical double-fork, which is extra added protection from your process accidentally acquiring a controlling terminal. More information can be found above, and by Googling ‘‘double fork daemonize’’.

If you the double-fork behavior off, you might want to enable the *ignore_zombies*.

ignore_zombies (is = rw, isa => Bool)>

Setting this attribute to a true value will result in setting the `$SIG{CHLD}` handler to `IGNORE`. This tells perl to clean up zombie processes. By default, and for the most part you don’t *need* it, only when you turn off the double fork behavior (with the *no_double_fork* attribute) do you sometimes want this behavior.

dont_close_all_files (is = rw, isa => Bool)>

Setting this attribute to true will cause it to skip closing all the filehandles. This is useful if you are opening things like sockets and such in the pre-fork.

METHODS

daemon_fork (?%options)

This forks off the child process to be daemonized. Just as with the built in fork, it returns the child pid to the parent process, 0 to the child process. It will also set the is_daemon flag appropriately.

The `%options` argument remains for backwards compatibility, but it is suggested that you use the attributes listed above instead.

daemon_detach (?%options)

This detaches the new child process from the terminal by doing the following things.

The `%options` argument remains for backwards compatibility, but it is suggested that you use the attributes listed above instead.

Becomes a session leader

This detaches the program from the controlling terminal, it is accomplished by calling `POSIX::setsid`.

Performing the double-fork

See below for information on how to change this part of the process.

Changes the current working directory to "/"

This is standard daemon behavior, if you want a different working directory then simply change it later in your daemons code.

Clears the file creation mask.

Closes all open file descriptors.

See the *dont_close_all_files* attribute for information on how to change this part of the process.

Reopen STDERR, STDOUT & STDIN to /dev/null

This behavior can be controlled slightly though the `MX_DAEMON_STDERR` and `MX_DAEMON_STDOUT` environment variables. It will look for a filename in either of these variables and redirect `STDOUT` and/or `STDERR` to those files. This is useful for debugging and/or testing purposes.

NOTE

If called from within the parent process (the `is_daemon` flag is set to false), this method will simply return and do nothing.

daemonize (?%options)

This will simply call `daemon_fork` followed by `daemon_detach`.

The `%options` argument remains for backwards compatibility, but it is suggested that you use the attributes listed above instead.

meta()

The `meta()` method from Class::MOP::Class

STUFF YOU SHOULD READ

Note about double fork

Taken from <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/66012> in a comment entitled *The second fork _is_ necessary by Jonathan Bartlett*, it is not the definitive statement on the issue, but it's clear and well written enough so I decided to reproduce it here.

```
The first fork accomplishes two things - allow the shell to return,
and allow you to do a setsid().

The setsid() removes yourself from your controlling terminal. You
see, before, you were still listed as a job of your previous process,
and therefore the user might accidentally send you a signal. setsid()
gives you a new session, and removes the existing controlling terminal.

The problem is, you are now a session leader. As a session leader, if
you open a file descriptor that is a terminal, it will become your
controlling terminal (oops!). Therefore, the second fork makes you NOT
be a session leader. Only session leaders can acquire a controlling
terminal, so you can open up any file you wish without worrying that
it will make you a controlling terminal.

So - first fork - allow shell to return, and permit you to call setsid()

Second fork - prevent you from accidentally reacquiring a controlling
terminal.
```

That said, you don't always want this to be the behavior, so you are free to specify otherwise using the *no_double_fork* attribute.

Note about zombies

Doing the double fork (see above) tends to get rid of your zombies since by the time you have double forked your daemon process is then owned by the init process. However, sometimes the double-fork is more than you really need, and you want to keep your daemon processes a little closer to you. In this case you have to watch out for zombies, you can avoid then by just setting the *ignore_zombies* attribute (see above).

ENVIRONMENT VARIABLES

These variables are best just used for debugging and/or testing, but not used for actual logging. For that, you should reopen `STDOUT`/`STDERR` on your own.

MX_DAEMON_STDOUT

A filename to redirect the daemon `STDOUT` to.

MX_DAEMON_STDERR

A filename to redirect the daemon `STDERR` to.

DEPENDENCIES

Moose::Role, POSIX

INCOMPATIBILITIES

SEE ALSO

Proc::Daemon

This code is based **HEAVILY** on Proc::Daemon, we originally depended on it, but we needed some more flexibility, so instead we just stole the code.

COPYRIGHT AND LICENCE

Portions heavily borrowed from Proc::Daemon which is copyright Earl Hood.

SUPPORT

Bugs may be submitted through the RT bug tracker <https://rt.cpan.org/Public/Dist/Display.html?Name=MooseX-Daemonize> (or bug−MooseX−Daemonize AT rt DOT cpan DOT org <mailto:bug-MooseX-Daemonize AT rt DOT cpan DOT org>).

There is also a mailing list available for users of this distribution, at <http://lists.perl.org/list/moose.html>.

There is also an irc channel available for users of this distribution, at `#moose` on `irc.perl.org` <irc://irc.perl.org/#moose>.

AUTHORS

- Stevan Little <stevan DOT little AT iinteractive DOT com>
- Chris Prather <chris AT prather DOT org>

COPYRIGHT AND LICENCE