orphanage Documentation

Release 0.1.0

Jiangge Zhang

May 09, 2018

Contents

CHAPTER 1

Overview

Let child processes in Python suicide if they became orphans.

CHAPTER 2

Installation

```
pip install orphanage
```

Don't forget to put it in `setup.py` / `requirements.txt`.

CHAPTER 3

Usage

```
from orphanage import exit_when_orphaned

exit_when_orphaned()
```

CHAPTER 4

Motivation

Some application server softwares (e.g. Gunicorn) work on a multiple-process architect which we call the master-worker model. They must clean up the worker processes if the master process is stopped, to prevent them from becoming orphan processes.

In the gevent-integration scene, the worker processes of Gunicorn poll their `ppid` in an user thread (a.k.a greenlet) to be orphan-aware. But the user thread may be hanged once the master process crashed because of the blocked writing on a pipe, the communicating channel between master process and worker processes.

We want to perform this `ppid` polling in a real kernel thread. That is the intent of this library.

CHAPTER 5

Principle

This library spawns an internal thread to poll the `ppid` at regular intervals (for now it is one second). Once the `ppid` changed, the original parent process should be dead and the current process should be orphaned. The internal thread will send `SIGTERM` to the current process.

In the plan, the `prctl` & `SIGHUP` pattern may be introduced in Linux platforms to avoid from creating threads. For now, the only supported strategy is the `ppid` polling, for being portable.

CHAPTER 6

Alternatives

CaoE is an alternative to this library which developed by the Douban Inc. It uses `prctl` and a twice-forking pattern. It has a pure Python implementation without any C extension compiling requirement. If you don't mind to twist the process tree, that will be a good choice too.

CHAPTER 7

Contributing

If you want to report bugs or request features, please feel free to open issues on GitHub.

Of course, pull requests are always welcome.

CHAPTER 8

Table of Content

8.1 Development Guide

There is a `Makefile` for development in local environment. See available commands via invoking `make help`.

You will need to install pyenv and **tox_** globally in your local environment:

```
brew install pyenv tox
pyenv install 2.7.14  # and also
```

8.1.1 Requirement

The requirement changes of testing and document building environment need to be included in `requirements-test.in` and `docs/requirements.in`. You will need to invoke `make deps` to compile them into `requirements*.txt`.

8.1.2 Test

For running test in all supported Python versions, you will need pyenv:

```
# Enter the multi-version Python environment (2.7, 3.6, pypy2, pypy3)
pyenv shell 2.7.14:3.6.5:pypy2.7-5.10.0:pypy3.5-5.10.1

make test
```

For debugging, you may want to test in a specific Python version, such as 2.7:

```
tox -e py27                              # Default pytest options
tox -e py27 -- -vxs --log-cli-level=DEBUG # Custom pytest options
```

8.1.3 Package

For packaging a new distribution, `make dist` will be helpful. It assumes you are using macOS and the Docker for Mac has been installed and started also. The binary wheel packages for macOS (with your current ABI) and Linux (with manylinux API) will be present. Using pyenv and bumpversion is a good idea:

```
# Enter the multi-version Python environment (2.7, 3.6, pypy2, pypy3)
pyenv shell 2.7.14:3.6.5:pypy2.7-5.10.0:pypy3.5-5.10.1

bumpversion minor           # Commit and tag a new major/minor/patch release
make dist                   # Build release packages
make dist options="-b dev0" # Build pre-release packages
```

8.1.4 Clean up

You could clean up the workspace with `make clean`. It removes files which was ignored in the version control except the `.tox`.

8.1.5 Debugg C Extension

Debugging the C extension of Python needs different toolchains and skills. The `lldb` or `gdb` will be useful in that:

```
tox -e py27                  # Run test until it hangs
vim tests/_orphanage_poll.c  # Inspect the CFFI generated code
lldb --attach-pid=100001     # Attach to the target process
lldb> breakpoint set -f _orphanage_poll.c -l 434
lldb> continue
lldb> bt all
```

For unexpected crashing, the coredump will include useful information:

```
ulimit -c unlimited         # Turn on coredump in current shell
tox -e py27                 # Run test until it crashes
lldb --core /cores/cores.10 # Open the coredump named with its pid
lldb> bt all                # Print the backtrace
```

8.2 API Reference

For most users, please use the public API instead of the internal one.

8.2.1 Public API

orphanage.**exit_when_orphaned**()

Let the current process exit when it was orphaned.

Calling multiple times and calling-and-forking are both safe. But this is not a thread safe function. Never call it concurrently.

8.2.2 Internal API

class orphanage.poll.**Context**(*callbacks=None*, *suicide_instead=False*)
The context of orphans polling which acts as the CFFI wrapper.

> **Caution:** It is dangerous to use this class directly except you are familiar with the implementation of CPython and you know what you are doing clearly. It is recommended to use the *Public API* instead, for most users.

The context must be closed via *close()* or the memory will be leaked.

Parameters **callbacks** – Optional. The list of callback functions. A callback function will be passed one parameter, the instance of this context. Be careful, never invoking any Python built-in and C/C++ extended functions which use the `Py_BEGIN_ALLOW_THREADS`, such as `os.close` and all methods on this context, to avoid from deadlock and other undefined behaviors.

close()
Closes this context and release the memory from C area.

start()
Starts the polling thread.

stop()
Stops the polling thread.

Don't forget to release allocated memory by calling *close()* if you won't use it anymore.

trigger_callbacks()
Triggers the callback functions.

This method is expected to be called from C area.

orphanage.poll.**orphanage_poll_routine_callback**(*ptr*)
The external callback function of CFFI.

This function invokes the *Context.trigger_callbacks()* method.

Parameters **ptr** – The C pointer of context.

Returns `0` for nonerror calls.

orphanage.poll.**perror**(*description*)
Raises a runtime error from the specified description and `errno`.

orphanage.poll.**raise_for_return_value**(*return_value*)
Checks the return value from C area.

A runtime error will be raised if the return value is nonzero.

Python Module Index

o

Index