littleworkers Documentation

Release 0.3.1

Daniel Lindsley

February 23, 2016

author Daniel Lindsley

date 2011/11/10

version 0.3.1

license BSD

Little process-based workers to do your bidding.

Deliberately minimalist, you provide the number of workers to use & a list of commands (to be executed at the shell) & littleworkers will eat through the list as fast as it can.

CHAPTER 1

Topics

1.1 Tutorial

1.1.1 Quick Start

A simple setup looks like:

```python
from littleworkers import Pool

# Define your commands.
commands = [
    'ls -al',
    'cd /tmp && mkdir foo',
    'date',
    'echo "Hello There."',
    'sleep 2 && echo "Done."'
]

# Setup a pool. Since I have two cores, I'll use two workers.
lil = Pool(workers=2)

# Run!
lil.run(commands)
```

1.1.2 Philosophy

littleworkers shines when you just want to parallelize something without a lot of fuss & when you care more about the data/commands to be run.

- Tiny source
- Easy to queue a set of actions
- Works with any runnable commands
- Uses processes
- Non-blocking

Seriously, it's not a replacement for threading or multiprocessing if your application needs to share a ton of data with the children.

1.1.3 Extension

littleworkers was designed to be extended, so most customizations should be possible without forking the code. Instead, you should simple subclass `Pool` & extend/override the method. You can find the details of each method in the API docs.

1.1.4 Example Customizations

You want the stdout back:

```
import subprocess
from littleworkers import Pool


class MyPool(Pool):
    def __init__(self, *args, **kwargs):
        super(MyPool, self).__init__(*args, **kwargs)
        self.collected_output = []

    def create_process(self, command):
        logging.debug("Starting process to handle command '%s'." % command)
        return subprocess.Popen(command, shell=True, stdout=subprocess.PIPE)

    def remove_from_pool(self, pid):
        self.collected_output.append(self.pool[pid].stdout.read())
        return super(MyPool, self).remove_from_pool(pid)
```

You want to use a `Queue` instead of the default `list`:

```
from Queue import Queue, Empty
from littleworkers import Pool


class QueuePool(Pool):
    def __init__(self, *args, **kwargs):
        super(QueuePool, self).__init__(*args, **kwargs)
        self.commands = Queue()

    def prepare_commands(self, commands):
        for command in commands:
            self.commands.put(command)

    def command_count(self):
        return self.commands.qsize()

    def next_command(self):
        try:
            return self.commands.get()
        except Empty:
            return None
```

You want to setup a callback:

```
from littleworkers import Pool

codes = []

def track(proc):
    codes.append("%s returned status %s" % (proc.pid, proc.returncode))

commands = [
    'sleep 1',
    'busted_command --here',
    'sleep 1',
]
lil.run(commands, callback=track)
```

1.2 API

class `littleworkers.`**`Pool`**(*workers=1*, *debug=False*, *wait_time=0.1*)

The main pool object. Manages a set of specified workers.

Usage:

```
    commands = [
        'ls -al',
        'cd /tmp && mkdir foo',
        'date',
        'echo "Hello There."',
        'sleep 2 && echo "Done."'
    ]
    lil = Pool(workers=2)
    lil.run(commands)
```

Optionally accepts a `workers` kwarg. Default is 1.

Optionally accepts a `debug` kwarg. Default is False.

Optionally accepts a `wait_time` kwarg. Default is 0.1.

`add_to_pool`(*proc*)

Adds a process to the pool.

`busy_wait`()

A hook to control how often the busy-wait loop runs.

By default, sleeps for 0.1 seconds.

`command_count`()

Returns the number of commands to be run.

Useful as a hook if you use a different structure for the commands.

`create_process`(*command*)

Given a provided command (string or list), creates a new process to execute the command.

`inspect_pool`()

A hook for inspecting the pool's current status.

By default, simply makes a log message and returns the length of the pool.

`next_command`()

Fetches the next command for processing.

Will return `None` if there are no commands remaining (unless `Pool.debug = True`).

`prepare_commands`(*commands*)

A hook to override how the commands are added.

By default, simply copies the provided command `list` to the internal `commands` list.

`process_kwargs`(*command*)
A hook to alter the kwargs given to `subprocess.Process`.

Takes a `command` argument, which is unused by default, but can be used to switch the flags used.

By default, only specifies `shell=True`.

`remove_from_pool`(*pid*)
Removes a process to the pool.

Fails silently if the process id is no longer present (unless `Pool.debug = True`).

`run`(*commands=None*, *callback=None*)
The method to actually execute all the commands with the pool.

Optionally accepts a `commands` kwarg, as a shortcut not to have to call `Pool.prepare_commands`.

`set_callback`(*callback=None*)
Sets up a callback to be run whenever a process finishes.

If called with `None` or without any args, it will clear any existing callback.

CHAPTER 2

Requirements

- Python 2.6+ (may work with Python 2.5)

`littleworkers` is tested & works on Mac OS X/Linux/BSD. It may work on Windows (!) but is untested. Feedback welcome.

CHAPTER 3

Installation

You can install from PyPI using `pip` (or `easy_install` if you prefer broken, unmaintained software):

```
pip install littleworkers
```

The only dependencies are in Python's stdlib & the code is pure Python, so there's nothing to compile.

Testing

`littleworkers` is maintained with a passing test suite at all times. You should use **nose_** or similar tools to run the tests like:

```
nosetests tests.py
```

Output is currently pretty verbose, which will be fixed in the future.

CHAPTER 5

Contributions

Contributions are welcome & should be submitted as pull requests on **GitHub_**. The pull request must have:

- Only the code needed to add the feature or fix the bug (not several in one)
- Added tests to cover the change
- Internal docs in the form of docstrings
- If it changes the public API, it should include docs
- Must be BSD-licensed code

Python Module Index