goless Documentation

Release 0.6.0

Rob Galanakis

July 11, 2014

Contents

CHAPTER 1

Intro

The goless library provides **Go** programming language semantics built on top of gevent, PyPy, or Stackless Python.

For an example of what **goless** can do, here is the Go program at https://gobyexample.com/select reimplemented with **goless**:

```
c1 = goless.chan()
c2 = goless.chan()

def func1():
    time.sleep(1)
    c1.send('one')
goless.go(func1)

def func2():
    time.sleep(2)
    c2.send('two')
goless.go(func2)

for i in range(2):
    case, val = goless.select([goless.rcase(c1), goless.rcase(c2)])
    print(val)
```

It is surely a testament to Go's style that it isn't much less Python code than Go code, but I quite like this. Don't you?

CHAPTER 2

Goroutines

The `goless.go()` function mimics Go's goroutines by, unsurprisingly, running the routine in a tasklet/greenlet. If an unhandled exception occurs in a goroutine, `goless.on_panic()` is called.

`goless.`**`go`**(*func*, **args*, ***kwargs*)

Run a function in a new tasklet, like a goroutine. If the goroutine raises an unhandled exception (*panics*), the `goless.on_panic()` will be called, which by default logs the error and exits the process.

Parameters

- **args** – Positional arguments to `func`.
- **kwargs** – Keyword arguments to `func`.

`goless.`**`on_panic`**(*etype*, *value*, *tb*)

Called when there is an unhandled error in a goroutine. By default, logs and exits the process.

CHAPTER 3

Channels

There are three types of channels available in `goless`. Use the `goless.chan()` function to create a channel. The channel implementations contain more thorough documentation about how they actually work.

`goless.`**`chan`**(*size=0*)
Returns a bidirectional channel.

A 0 or None size indicates a blocking channel (the `send` method will block until a receiver is available, and the `recv` method will block until a sender is available).

A positive integer value will return a channel with a buffer. Once the buffer is filled, the `send` method will block. When the buffer is empty, the `recv` method will block.

A negative integer will return a channel that will never block when the `send` method is called. The `recv` method will block if the buffer is empty.

Return type goless.channels.GoChannel

class `goless.channels.`**`GoChannel`**
A **Go**-like channel that can be sent to, received from, and closed. Callers should never create this directly. Always use `goless.chan()` to create channels.

`close`()
Closes the channel, not allowing further communication. Any blocking senders or receivers will be woken up and raise `goless.ChannelClosed`. Receiving or sending to a closed channel will raise `goless.ChannelClosed`.

`recv`()
Receive a value from the channel. Receiving will always block if no value is available. If the channel is already closed, `goless.ChannelClosed` will be raised. If the channel closes during a blocking `recv`, `goless.ChannelClosed` will be raised. (#TODO)

`send`(*value=None*)
Sends the value. Blocking behavior depends on the channel type. Unbufferred channels block if no receiver is waiting. Buffered channels block if the buffer is full. Asynchronous channels never block on send.

If the channel is already closed, `goless.ChannelClosed` will be raised. If the channel closes during a blocking `send`, `goless.ChannelClosed` will be raised. (#TODO)

class `goless.`**`ChannelClosed`**
Exception raised to indicate a channel is closing or has closed.

CHAPTER 4

The select function

Go's `select` statement is implemented through the `goless.select()` function. Because Python lacks anonymous blocks (*multiline lambdas*), `goless.select()` works like Go's reflect.Select function. Callers should create any number of `goless.case` classes that are passed into `goless.select()`. The function returns the chosen case, which the caller will usually switch off of. For example:

```
chan = goless.chan()
cases = [goless.rcase(chan), goless.scase(chan, 1), goless.dcase()]
chosen, value = goless.select(cases)
if chosen is cases[0]:
    print('Received %s' % value)
elif chosen is cases[1]:
    assert value is None
    print('Sent.')
else:
    assert chosen is cases[2]
    print('Default...')
```

Callers should never have to do anything with cases, other than create and switch off of them.

`goless.`**`select`**`(`**cases*`)`
: Select the first case that becomes ready. If a default case (`goless.dcase`) is present, return that if no other cases are ready. If there is no default case and no case is ready, block until one becomes ready.

  See Go's `reflect.Select` method for an analog (http://golang.org/pkg/reflect/#Select).

  Parameters **cases** – List of case instances, such as `goless.rcase`, `goless.scase`, or `goless.dcase`.

  Returns `(chosen case, received value)`. If the chosen case is not an `goless.rcase`, it will be None.

class `goless.`**`dcase`**
: The default case.

class `goless.`**`rcase`**`(`*chan*`)`
: A case that will `chan.recv()` when the channel is able to receive.

class `goless.`**`scase`**`(`*chan*, *value*`)`
: A case that will `chan.send(value)` when the channel is able to send.

CHAPTER 5

Exception Handling

Exception handling is a tricky topic and may change in the future. The default behavior right now is that an unhandled exception in a goroutine will log the exception and take down the entire process. This in theory emulates Go's `panic` behavior: if a goroutine panics, the process will exit.

If you are not happy with this behavior, you should patch *goless.on_panic* to provide custom behavior.

If you find a better pattern, create an issue on GitHub.

CHAPTER 6

Examples

The `examples/` folder contains a number of examples.

In addtion, there are many examples from http://gobyexample.com implemented via `goless` in the `tests/test_examples.py` file.

If there is an example you'd like to see, or an idiomatic Go example you'd like converted, please see *Contributing* below.

CHAPTER 7

Benchmarks

You can run benchmarks using the current Python interpreter and configured backend by running the following from the `goless` project directory:

```
$ python -m benchmark
```

Developers may run benchmarks locally and report them into the following table. The **Go** versions of the benchmarks are also run. The numbers are useful for relative comparisons only:

Table 7.1: Current goless Benchmarks

Platform	Backend	Benchmark	Time
go	gc	chan_async	0.00236
PyPy2	stackless	chan_async	0.03200
CPython2	stackless	chan_async	0.09000
PyPy2	gevent	chan_async	0.39600
CPython3	gevent	chan_async	0.91000
CPython2	gevent	chan_async	1.05000
go	gc	chan_buff	0.00235
PyPy2	stackless	chan_buff	0.03200
CPython2	stackless	chan_buff	0.10000
PyPy2	gevent	chan_buff	0.39600
CPython3	gevent	chan_buff	0.97000
CPython2	gevent	chan_buff	1.11000
go	gc	chan_sync	0.00507
PyPy2	stackless	chan_sync	0.05200
CPython2	stackless	chan_sync	0.10000
PyPy2	gevent	chan_sync	0.80000
CPython3	gevent	chan_sync	0.89000
CPython2	gevent	chan_sync	1.07000
go	gc	select	0.03031
PyPy2	stackless	select	0.06400
CPython2	stackless	select	0.28000
PyPy2	gevent	select	0.49200
CPython3	gevent	select	1.38000
CPython2	gevent	select	1.49000

Continued on next page

Table 7.1 – continued from previous page

Platform	Backend	Benchmark	Time
PyPy2	gevent	select_default	0.00800
PyPy2	stackless	select_default	0.00800
go	gc	select_default	0.02645
CPython2	stackless	select_default	0.14000
CPython3	gevent	select_default	0.15000
CPython2	gevent	select_default	0.20000

To regenerate this table, run:

```
$ python write_benchmarks.py
```

To print the table to stdout, run (notice the trailing – char):

```
$ python write_benchresults.py -
```

Assuming you have **Go** installed, you can run the benchmarks with:

```
$ go run benchmark.go
```

CHAPTER 8

Backends

There are two backends for concurrently available in `goless.backends`. Backends should only be used by `goless`, and not by any client code. You can choose between backends by setting the environment variable `GOLESS_BACKEND` to `"gevent"` or `"stackless"`. Otherwise, an appropriate backend will be chosen. If neither `gevent` or `stackless` are available, `goless` will raise an error when used (but will still be importable).

CHAPTER 9

Compatibility Details

The good news is that you probably don't need to worry about any of this, and goless works almost everywhere.

The bad news is, almost all abstractions are leaky, and there can be some nuances to compatibility. If you run into an issue where `goless` cannot create a backend, you may need to read the following sections.

9.1 PyPy

`goless` works under PyPy out of the box with the stackless backend, because PyPy includes a `stackless.py` file in its standard library. This appears to work properly, but fails the `goless` test suite. We are not sure why yet, as `stackless.py` does not have a real maintainer and the bug is difficult to track down. However, the examples and common usages seem to all work fine.

Using PyPy 2.2+ and the tip of gevent's GitHub repo (https://github.com/surfly/gevent), the gevent backend works and is fully tested.

9.2 Python 2 (CPython)

Using Python 2 and the CPython interpreter, you can use the gevent backend for `goless` with no problems. Under Python 2, you can just do:

```
$ pip install gevent
$ pip install goless
```

9.3 Python 3 (CPython)

Newer versions of gevent include Python 3 compatibility. To install gevent on Python3, you also **must** install Cython. So you can use thew following commands to install `goless` under Python3 with its gevent backend:

```
$ pip install cython
$ pip install git+https://github.com/surfly/gevent.git#gevent-egg
$ pip install goless
```

This works and is tested.

9.4 Stackless Python

All versions of Stackless Python (2 and 3) should work with goless. However, we cannot test with Stackless Python on Travis, so we only test with it locally. If you find any problems, *please* report an issue.

CHAPTER 10

goless and the GIL

`goless` does not address CPython's **Global Interpreter Lock** (**GIL**) at all. `goless` does not magically provide any parallelization. It provides Go-like semantics, but not its performance. Perhaps this will change in the future if the GIL is removed. Another option is PyPy's STM branch, which `goless` will (probably) benefit heartily.

CHAPTER 11

References

- **goless** on GitHub: https://github.com/rgalanakis/goless
- **goless** on Read the Docs: http://goless.readthedocs.org/
- **goless** on Travis-CI: https://travis-ci.org/rgalanakis/goless
- **goless** on Coveralls: https://coveralls.io/r/rgalanakis/goless
- The Go Programming Language: http://www.golang.org
- Stackless Python: http://www.stackless.com
- gevent: http://gevent.org/
- PyPy: http://pypy.org/
- Idiomatic Go Examples: http://gobyexample.com

CHAPTER 12

Contributing

I am definitely not a Go expert, so improvements to make things more idiomatic are very welcome. Please create an issue or pull request on GitHub: https://github.com/rgalanakis/goless

`goless` was created by a number of people at the PyCon 2014 sprints. Even a small library like `goless` is the product of lots of collaboration.

Maintainers:

- Rob Galanakis <rob.galanakis@gmail.com>
- Simon König <simjoko@gmail.com>
- Carlos Knippschild <carlos.chuim@gmail.com>

Special thanks:

- Kristján Valur Jónsson <sweskman@gmail.com>
- Andrew Francis <af.stackless@gmail.com>

CHAPTER 13

Miscellany

Coverage is wrong. It should be higher. The coverage module does not work properly with gevent and stackless.

CHAPTER 14

Indices and tables

- *genindex*
- *modindex*
- *search*