Prototyping Go's Select in Stackless Python with Stackless.py

1st draught

Andrew Francis

Montreal Python #13

April 26th, 2010

af.stackless@gmail.com

May 3rd, Notes

- Based on early feedback, I am going to change the structure of this talk somewhat.
- The title will change to:
  - *Prototyping Go's Select*
    for Stackless Python with Stackless.py

Purpose

- Illustrate how PyPy's stackless.py module can be used to prototype new Stackless Python features.
- The example is the Select statement in the Go Language.

What You Should Take Away?

- A basic knowledge of Stackless Python concepts and exposure to its API.
- The power of Python to quickly prototype concepts.
  - Sometimes a prototype illustrates the infeasibility of a concept.
  - “fearless programming” i., typeless, referenceless, clutterless ….

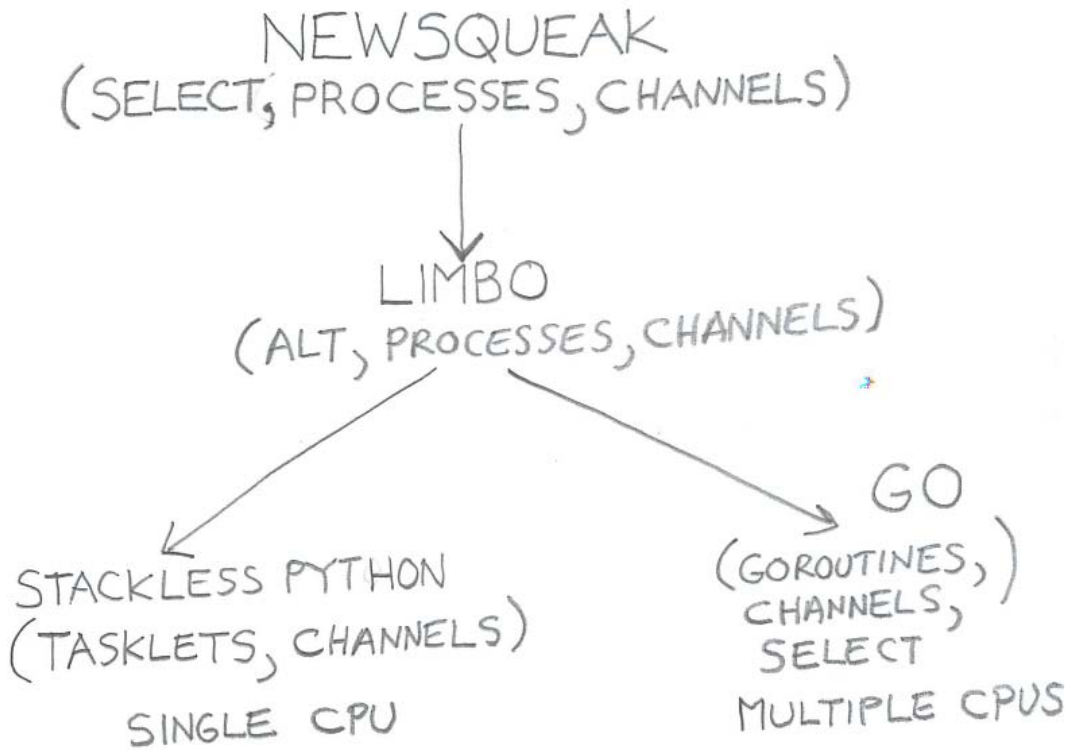
Motivation

- *Masha Rabinovich: Go's microthreads and channels are only partially exposed to the programmer, and in a very limited way. In order to make them usable you need select. In Stackless, you do not need select and in fact, there has been no demand for anything like it.*

Taken from Richard Tew's "Stuff What I Posted

http://posted-stuff.blogspot.com/2009/11/comparing-go-and-stackless-python.html

Why Go?


NEWSQUEAK
(SELECT, PROCESSES, CHANNELS)
LIMBO
(ALT, PROCESSES, CHANNELS)
STACKLESS PYTHON
(TASKLETS, CHANNELS)
SINGLE CPU
GO
(GOROUTINES, CHANNELS, SELECT)
MULTIPLE CPUS

Brief Overview

Section Two

Basics

- Tasklets
  - Light weight threads that execute in user space.
  - Low over head
  - Fast context switching
- Channels
  - Used for concurrency and communications.
  - Rendezvous semantics

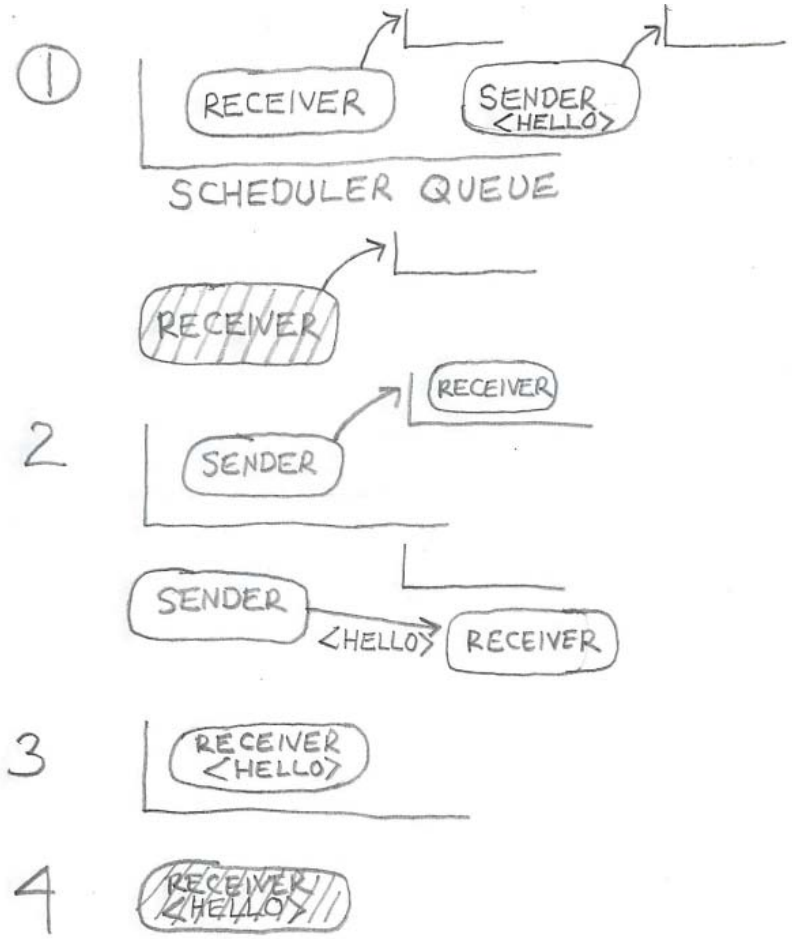
Stackless Example

```
import stackless

def reader(channel):
    print "entering reader "
    print channel.receive()
    print "exiting reader"


def writer(channel):
    print "entering writer"
    channel.send(100)
    print "exiting writer"


if __name__ == "__main__":
   ch = stackless.channel()
   stackless.tasklet(reader)(ch)
   stackless.tasklet(writer)(ch)
   stackless.run()
```

Go Example

```
package main

import fmt "fmt"

func reader(in chan int) {
     fmt.Printf("entering reader\n");
     x := <- in;
     fmt.Printf("->%d \n", x);
     fmt.Printf("exiting reader\n");
     in <- 1
}

func writer(out chan int) {
     fmt.Printf("entering writer\n");
     out <- 1;
     fmt.Printf("exiting writer\n");
}

func main() {
     var ch = make(chan int);
     go reader(ch);
     go writer(ch);
     <-ch;
}
```

How Channels Work



What Problem Does Select Solve?

```
while True:
  for ch in channels:
    message = ch.receive()
    stackless.tasklet(doSomething)(message)
```

```
Let us pretend ….
Ch[0] ready after T + 10
Ch[1] ready after T + 15
Ch[2] ready after T + 1
Ch[3] ready after T + 2
```

Go Select Example

```
for {
        select {
        case a := <- ch1:
             go doSomething(a);
        case b := <- ch2
             go doSomething(b);
        case c := <- ch3
             go doSomething(c);
        case d := <- ch4:
             go doSomething(d);
        }
     }
```

Another Go Concept – Checking to block

v, okay = <- ch

Okay := ch <- v

(taken from GoCourseDay 3)

Allows programmer to check if operation will block

Stackless has channel.balance

Philosophical Differences

- In Go, concurrency is a language feature. Stackless offers concurrency via classes.
- Go exposes much less of the underlying concurrency machinery to the programmer than Stackless.
- Different audiences?

Food For Thought

- In the WS-BPEL specification, the Pick activity waits on multiple events.
  - When an event is ready, the process resumes.
  - The remaining events are ignored.
- If the implementation uses tasklets, ignoring is easy
  - tasklet.kill(), tasklet.remove()
- How does Go do it?

A Stackless Python Work Around

- Select can be emulated in Stackless Python
  - Solution requires additional tasklets to block on selected channels
  - Andrew Dalke posted a solution see http://www.stackless.com/pipermail/stackless/2009-November/004392.html
  - Works but more complicated than having support in the scheduler

My Proposal

```
def eventManager(list):
    ev = stackless.eventHandler(self.list)
    while True:
       ch, operation, value =  ev.wait()
       if ch == self.channels[0] and \
          operation == SEND:
             doNetworkEvent(value)
       elif ch == self.channels[1] and \
            operation == SEND
               doWindowSystemEvent(value)
       ....
    return
```

Stackless.py

Section Three

Stackless.py Basics

- Implementation of the Stackless Python module included with PyPy.
- Stackless.py is written in Python!
- Runs on top of the coroutine and greenlet packages
- Can be used with PyPy interpreter (real slow)
- Compiled into pypy-c

Stackless.py Continued

- Unfortunately, stackless.py is buggy and hasn't been kept up to date.
  - Not a high priority for the PyPy Team.
  - We don't need stackless.py to be perfect
- A great way to understand Stackless Python concepts without the clutter of C!

A Trick

- Stackless.py along with the greenlet package can be used with Standard Python!
- This is how the PyPy team developed stackless.py!

Questions

- How closely does Stackless.py approximate C Stackless Python?
- Is it possible to implement Select without changing the C Stackless Python implementation?
- If not, what are the minimum number of changes needed to support Select as a Python module?

Stackless Python and Psyco

- The creator of Stackless Python, Christian Tismer also works on Psyco, a JIT extension.
- Psyco and Stackless are being integrated
  - Version already available
- Stackless Python will eventually become an Python extension module!
- All the more reason to write in Python!

Prototyping Select with Stackless.py

Section Four

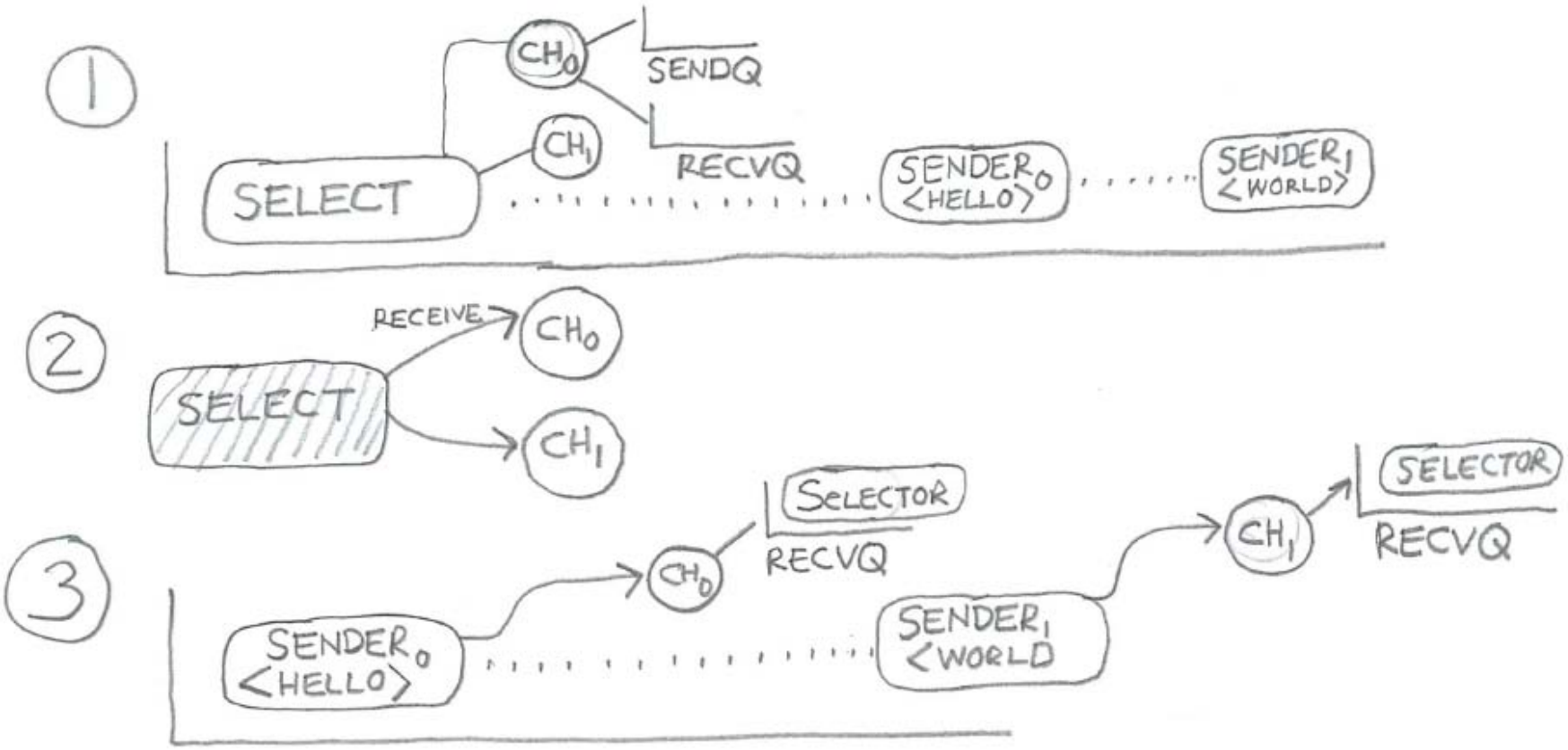
Strategy

- Follow the algorithms in Rob Pike's "The Implementation of Newsqueak."
- Reviewed Stackless C code.
- Reviewed Go chan.c and proc.c
- Ask questions in Golang nuts Google group and Stackless mailing lists.

The Select Algorithm

- Search list for a ready channel operation
  - In case of tie, randomly choose
- Put the coroutine on the all the channel queues.
- Suspend
- When rendezvous occurs
  - Remove reference from remaining channels
  - Schedule selected coroutine

Select Illustrated



Fundamental Problems

- Need way to add a tasklet to a channel queue without blocking.
- When a tasklet implementing a selector wakes up, how does the selector determine the channel and the operation?
- How to do selector data transfers?
  - The case of when the selector is blocked on a send().

Need Changes to C Implementation

- The aforementioned additions require changes to C Stackless Python.

More Info – The Scheduler API

- Need to use three new stackless methods
  - schedule_remove()
    - removes a tasklet from the scheduler queue.
  - schedule()
    - Allows the next runnable tasklet to execute
  - getcurrent()
    - Get the currently executing tasklet

Main Changes

- Created a new class – eventHandler that acts as the selector
  - __init__((channel, operation, value))
    - Create the selector set
  - wait()
    - Wait for an event to occur
  - update(self, source, target, operation, data)
    - Allows a channel to update the selector
    - Isolate changes to channel implementation

Main Changes Continued

- Added channel.addOperation(self, tasklet, operation, value)
  - Allows data to be added to the channel without an action taken (i.e., send/receive)

Event Wait

```
def wait(self):
        source = getcurrent()
        source._selector = self

        event = self._selectReadyAtRandom()
        if event == None:
           for event in self.newEventQueue:
               self.blockedChannels.append(event)
        else:
            event.getValue()
            return event.channel, event.operation, event.value

        self.blocked = True
        for event in self.blockedChannels:
            event.addToChannel(source)

        schedule_remove()
        schedule()

        readyEvent = self.readyEventQueue.pop(0)

        self.reset()

        return readyEvent.channel, readyEvent.operation, readyEvent.value
```

EventHandler update()

```
def update(self, channel, operation, source, target,  args):
    e = event(channel, -operation, source.tempval, True)
    self.readyEventQueue.append(e)
    for theEvent in self.blockedChannels:
      try:
        """
        remove the tasklet and restore the balance
        """
        theEvent.channel.queue.remove((target, theEvent.value))
        theEvent.channel.balance -= theEvent.operation
      except ValueError:
        pass
    return
```

Stackless.py channel_action

```
if cando:
        detected = False
        debug("CANDO")
        # communication 1): there is somebody waiting
        target, data = self.queue.popleft()
        if hasattr(target, "_selector"):
          debug("UPDATING")
          target._selector.update(self, d, source, target, arg)
          detected = True

        source.tempval, target.tempval = data, source.tempval

        target.blocked = 0
```

C Stackless Python

```
static PyObject *
generic_channel_action(PyChannelObject *self, PyObject *arg, int dir, int
stackless)
{
          PyThreadState *ts = PyThreadState_GET();
          PyTaskletObject *source = ts->st.current;
          PyTaskletObject *target = self->head;
          int cando = dir > 0 ? self->balance < 0 : self->balance > 0;
          int interthread = cando ? target->cstate->tstate != ts : 0;
          PyObject *retval;

          assert(abs(dir) == 1);

          TASKLET_SETVAL(source, arg);

          /* note that notify might release the GIL. */
          /* XXX for the moment, we notify late on interthread */
          if (!interthread)
                    NOTIFY_CHANNEL(self, source, dir, cando, NULL);

          if (cando) {
                    /* communication 1): there is somebody waiting */
                    target = slp_channel_remove(self, -dir);
                    /* exchange data */
                    TASKLET_SWAPVAL(source, target);
```

Observations

- Once I got the gist of the algorithms and stackless.py, things when quickly.
- A clear description helps!
  - Can also test against Go!
- Wasted the most time on premature optimizations while I was still learning!
- Unit tests helped immensely.
- And a second pair of eyes!

Conclusions

- Happy with the process
- Changes to C Stackless would be necessary.
- Now have a framework to do what-ifs and measure results.
  - Tearing down channels is mighty expensive ….
- Get input from other developers.
- If feasible, lets put the changes in C Python or compile into pypy-c!

Thank You!