Systems Programming in Python - Week 6

Asynchronous Network I/O

Brian Dorsey
brian@dorseys.org

why do we care?

if you need lots of
simultaneous connections

chat
service proxies
slow clients
anything that takes a while

big topic

&

I’m a newbie

you’ve been warned

the solutions in this talk are only effective if your applicaion is I/O bound

```
CPU bound problems will need different solutions, processes, multiple computers, etc...

                    of course, once you try to get them all talking to each other...
                                                you have network I/O bound problems again. :)
```

telephone analogy

- answer
- ask someone to get get info (they walk across the building and back)
- relay answer
- hang up

HERE WE GO

blocking I/O

we’ve got a fantastic phone system, it can handle thousands of incoming calls, easily
we have 1 person answering the phones

what if you need to handle more than one client?

threads
(& thread pools)

Django, CherryPy, etc
(mod_wsgi uses processes
and threads)

```
1 thread = 1 person answering phones, 10 people in our phone pool
```

GIL?

(or… the internet says Python can’t do real threading)

We’re I/O bound today, so no worries.

this scales up pretty darn well

remember my apache bench stats last quarter?

except when it doesn’t

what if we get more traffic than we can handle?

just keep adding people to answer phones

as long as each conversation is short, we're OK

what if we need to ask a question which takes a while to answer?

as the conversations get longer, we're tying up resources even when we're not actively working

and eventually, we run out of space for more people to answer the phones

demo

how many threads can I make?

c10k

10,000 simultaneous connections

“It's time for web servers to handle ten thousand clients simultaneously, don't you think? After all, the web is a big place now.

And computers are big, too. You can buy a 1000MHz machine with 2 gigabytes of RAM and an 1000Mbit/sec Ethernet card for $1200 or so. Let's see - at 20000 clients, that's 50KHz, 100Kbytes, and 50Kbits/sec per client. It shouldn't take any more horsepower than that to take four kilobytes from the disk and send them to the network once a second for each of twenty thousand clients. <...> So hardware is no longer the bottleneck.”

Note the dated specs... this was said a *while* ago! http://www.kegel.com/c10k.html

why does this happen?

slow clients

maybe even maliciously slow?

nature of the problem

(think chat applications, or messaging servers)

something else is slow, and you need to wait on it

networks are always ‘slow’, so DB and other service requests add up

until recently, most of us didn't need to worry about this stuff

real time web, chat, events, live updates are getting more and more common

soon they'll be expected everywhere

have you used chat in gmail?

Asynchronous I/O

ignoring:

hardware interrupts

I/O callback functions

I/O completion ports

http://en.wikipedia.org/wiki/Asynchronous_I/O

processes

threads

(OS threads)

user threads

(lightweight threads)

select loop

- give list of file descriptors, block until one is ready
- slows down if hundreds or thousands of file descriptors

epoll, kqueue

- same as select loop, but only returns the file descriptors which are ready
- fast even with many thousands of (idle) file descriptors

these work only as long as we're blocking on I/O

analogy - if the operator needs to do work themselves, all calls are slowed down

for both select and epoll,
we have a main event loop

being programmers, the next step is to add abstraction layers

loop with inline code?

a reactor which owns the loop and calls methods on our objects?

register callbacks with the loop to handle each connection?

use coroutine magic to write code which looks like it blocks, but really hands execution off to the loop with a way to jump back when ready?

Yes.

all of those and more

wait, what's a coroutine?

subroutines are called, then exit

coroutines call each other

Python generators are limited coroutines

Python options

there are far, far

too many to cover

remember the echo server?

this is blocking I/O

```
1 operator answering the phones, getting answers, etc
```

aysyncore

- stdlib
- handles the select loop for you
- you subclass an object, create handers to do the work

http://docs.python.org/library/asyncore.html

twisted

- the original async framework in Python
- large, efficient, steep learning curve
- twisted
  - callback style programming, with deferreds to keep things clean
  - if you already understand JavaScript callbacks and jQuery deferreds, you won't be confused

http://twistedmatrix.com/trac/

gevent

- best of both worlds: "What you get is all the performance and scalability of an event system with the elegance and straightforward model of blocking IO programing."
- uses greenlets to cooperatively swap state and switch between functions

http://www.gevent.org/

Wrap up

if you have a c10k problem,
you probably need
async I/O

async I/O is really just:

do one thing at a time, very quickly

there are a bunch of ways to implement the async part, with different trade offs

there are a bunch of abstractions to make it easier to understand async code

this whole talk is only relevant when you're I/O bound

Questions?