Linden Lab®

Eventlet

Asynchronous I/O with a synchronous attitude.

We learn to program like this...

```
name = input('Name: ')
age = input('Age: ')
print 'Hi %s!' % name
print 'According to my calculations...'
print 'You are', age, 'years old!'
```

... and we still build stuff that way ...

```
name = cursor.execute(
    'SELECT name FROM user WHERE id = %s',
    user_id).fetchone()[0]
groups = llsdhttp.get(
    'http://group-service/%s' % user_id)
print name, 'has', len(groups), 'groups.'
```

Linden Lab

The C10K Problem

http://www.kegel.com/c10k.html

"It's time for web servers to handle ten thousand clients simultaneously, don't you think?"

Concurrency models for network servers:

- Processes
- Threads
- Asynchronous I/O

Processes

- Pros
  - Simple
  - Safe (isolated address space)
  - IPC with pipes, sockets, signals, files, etc.
- Cons
  - Extremely heavy weight
    - address space
    - open files and other I/O information
    - register values
    - call stack (kernel and user space)
  - Non-deterministic
  - Context switch expensive

Threads

- Pros
  - Much smaller memory footprint compared to processes.
  - Somewhat cheaper to bootstrap
  - Convenience of a shared address space
- Cons
  - Still non-deterministic
  - Synchronization extremely hard to get right. (Deadlock)
  - Context switch is still expensive, still has to switch out call stack, register values, etc.

You are pretty much screwed.

Asynchronous I/O

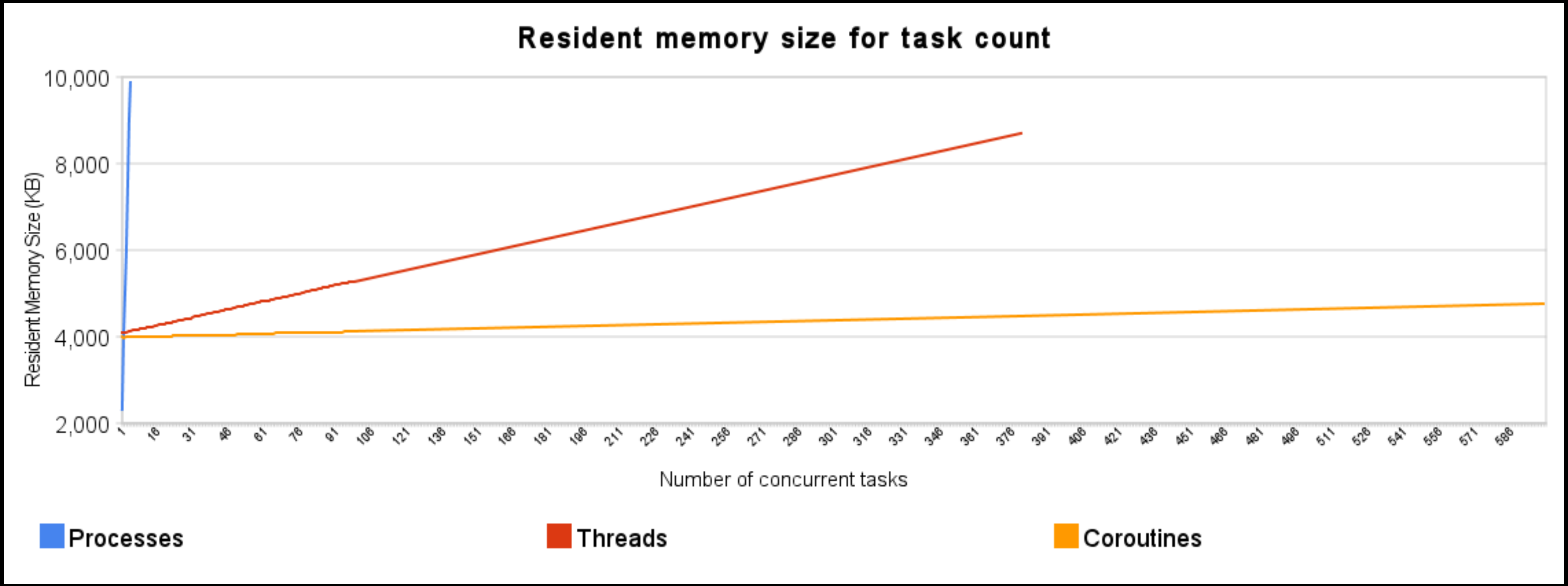
- Pros
  - Highly scalable, low memory requirements. Can handle thousands of concurrent requests.
    - haproxy
    - lighttpd
    - Twisted
  - Cheap context switch, just a function call
  - Deterministic
- Cons
  - Complicated programming model
  - Can't use existing libraries with blocking routines
  - Requires participation from the caller

Eventlet

Asynchronous I/O with Coroutines

Coroutines in Python with greenlet

```
>>> def counter(n):
...     p = greenlet.getcurrent().parent
...     p.switch()
...     for i in xrange(0, n):
...         p.switch(i)
>>> g = greenlet.greenlet(counter)
>>> g.switch(3)
>>> while True:
...     r = g.switch()
...     if r is None:
...         break
...     print r
1
2
3
>>>
```


Resident memory size for task count
10,000
8,000
6,000
4,000
2,000
Resident Memory Size (KB)
Number of concurrent tasks
1
16
31
46
61
76
91
106
121
136
151
166
181
196
211
226
241
256
271
286
301
316
331
346
361
376
391
406
421
436
451
466
481
496
511
526
541
556
571
586
Processes
Threads
Coroutines

API

eventlet.spawn(callable, *args, **kw)

```
>>> def worker(n):
...     print 'worker', n
>>> worker_list = []
>>> for i in range(3):
...     worker_list.append(
...         eventlet.spawn(worker, i))
>>> for worker in worker_list:
...     worker.wait()
worker 0
worker 1
worker 2
>>>
```

Linden Lab

exceptions work as you might expect

```
>>> def foo():
...     return n / 0
...
>>> try:
...     eventlet.spawn(foo, 5).wait()
... except ZeroDivisionError, e:
...     print 'Caught it!'
...
Caught it!
>>>
```

Linden Lab

... you can even throw them ...

```
>>> class StopWorking(Exception):
...     pass
...
>>> def worker():
...     try:
...         while True:
...             ... busy working away ...
...     except StopWorking:
...         ... perform any cleanup ...
...
>>> eventlet.spawn(worker).throw(StopWorking)
```

Linden Lab

eventlet.sleep(seconds=0)

```
>>> def f1():
...     print 'hi'
...     eventlet.sleep()
...     print 'bye'
...
>>> def f2():
...     print 'omg'
...     eventlet.sleep(0.5)
...     print 'no way!'
...
>>> for coro in map(eventlet.spawn, [f1, f2]):
...     coro.wait()
hi
omg
bye
no way!
>>>
```

Linden Lab

example usage of eventlet.sleep()

```
def waitpid(pid, options):
    if options & os_orig.WNOHANG != 0:
        return os.waitpid(pid, options)
    else:
        new_options = options | os.WNOHANG
        while True:
            rpid, status = \
                os.waitpid(pid, new_options)
            if status >= 0:
                return rpid, status
            eventlet.sleep(0.01)
```

eventlet.event.Event class

```
>>> e = eventlet.event.Event()
>>> eventlet.spawn_n(
...     lambda x: e.send(x + 1),
...     10)
>>> e.wait()
11
>>>
```

Linden Lab®

eventlet.queue.Queue class

```
>>> queue = eventlet.queue.Queue(maxsize=1)
>>> def consumer():
...     while True:
...         print queue.get(),
...
>>> _ = eventlet.spawn(consumer)
>>> queue.put('Hello,')
>>> queue.put('World!')
Hello, World!
>>>
```

Linden Lab

eventlet.semaphore.Semaphore

```
sem = \
    eventlet.semaphore.Semaphore()

def worker():
    with sem:
        print 'worker acquired'
        eventlet.sleep()
    print 'worker done'

w = eventlet.spawn_n(worker)
eventlet.sleep()
print 'main acquiring'
sem.acquire()
print 'main acquired!'
sem.release()
print 'main done!'
```

OUTPUT:

```
worker acquired
main acquiring
worker done
main acquired!
main done!
```

Linden Lab

eventlet.Timer

```
>>> with eventlet.timeout.Timeout(3):
...     try:
...         worker = eventlet.spawn(
...             lambda: eventlet.sleep(60))
...         worker.wait()
...     except eventlet.timeout.Timeout:
...         print 'timed out!'
...
timed out!
>>>
```

Linden Lab

eventlet.green package

non-blocking versions of modules
from the standard library.

eventlet.green provides...

- asynchat.py
- asyncore.py
- BaseHTTPServer.py
- CGIHTTPServer.py
- ftplib.py
- httplib.py
- os.py
- profile.py
- Queue.py
- select.py
- SimpleHTTPServer.py
- socket.py
- SocketServer.py
- ssl.py
- subprocess.py
- threading.py
- thread.py
- time.py
- urllib2.py
- urllib.py

Three choices for using eventlet.green

1. Import from eventlet.green package.

```
from eventlet.green import time
```

2. Use eventlet.import_patched()

```
feedparser = \
    eventlet.import_patched('feedparser')
```

Linden Lab

3. Use eventlet.monkey_patch()

To patch the standard library with all of eventlet.green:

```
eventlet.monkey_patch()
```

Or, just patch socket, select and time modules.

```
eventlet.monkey_patch(
    socket=True,
    select=True,
    time=True)
```

Linden Lab

Cooperative sockets

- Instead of blocking on accept(), read(), write(), etc, switch to mainloop.
- Main loop runs select(), poll(), epoll(), etc to switch back to suspended coroutine when I/O is ready.

```python
import eventlet

def handle(f):
    while True:
        line = f.readline()
        if not line:
            break
        f.write(line)
        f.flush()

server = eventlet.listen(('0.0.0.0', 6000))
pool = eventlet.GreenPool()
while True:
    new_sock, address = server.accept()
    pool.spawn_n(
        handle, new_sock.makefile('rw'))
```

Linden Lab

Examples

```python
import eventlet
from eventlet.green import urllib2, os

def fetch(url):
    resp = urllib2.urlopen(url)
    fd = open(
        os.basename(url),
        os.O_CREAT | os.O_WRONLY)
    while True:
        block = resp.read(8192)
        if not block:
            break
        os.write(fd, block)
    os.close(fd)
    return url

pool = eventlet.GreenPool(size=200)
urls = ['http://example.com/test.gif', ...]
for url in pool.imap(worker, hosts):
    print 'Fetched:', url
```

Linden Lab

```
import eventlet
from eventlet.green import subprocess

def worker(host_name):
    p = subprocess.Popen(
        ['ssh', host_name, '--', 'uptime'],
        stdout=subprocess.PIPE)
    output = p.communicate()[0].rstrip()
    return host_name, output

pool = eventlet.GreenPool(size=100)
hosts = (h.strip() for h in sys.stdin)

for host, output in pool.imap(worker, hosts):
    print '%s:' % host, output
```

Linden Lab

```
$ cat hostlist | ./uptime.py
host2.example.com: 01:54:53 up 80 days, 16:49,  0
users,  load average: 0.00, 0.00, 0.00
host9.example.com: 01:54:32 up 80 days, 16:49,  0
users,  load average: 0.16, 0.08, 0.14
... snip ...
host9000.example.com: 01:54:32 up 81 days,  4:
36,  0 users,  load average: 0.00, 0.00, 0.00
$ _
```

Linden Lab

```
import eventlet
from eventlet import wsgi

def application(env, start_response):
    if env['PATH_INFO'] != '/':
        start_response(
            '404 Not Found',
            [('Content-Type', 'text/plain')])
    start_response(
        '200 OK',
        [('Content-Type', 'text/plain')])
        return ['Hello, World!\r\n']

wsgi.server(
    eventlet.listen(('', 8090)),
    application)
```

Linden Lab

```
import MySQLdb
import eventlet.db_pool

pool = eventlet.db_pool.ConnectionPool(
    MySQLdb,
    host='mysql.example.com',
    user='user', passwd='passwd', db='mydb')

def worker():
    db_conn = pool.get()
    cursor = db_conn.cursor()
    cursor.execute(
        'UPDATE foo SET count = count + 1')

eventlet.spawn(worker).wait()
```

```python
@json_view
def command(request, body):
    if not request.method == 'POST':
        return \
            HttpResponseNotAllowed(['POST'])
    record_status(body)
    while True:
        update = \
            find_update(request.get_host())
        if update:
            break
        eventlet.sleep(3)
    return HttpResponse(
        update,
        mimetype='application/llsd+xml')
```

Linden Lab

```python
@json_view
def send_group_message(request, body):
    def send_message(user_id, msg):
        host, port = lookup_presence(user_id)
        send_message(user_id, host, port, msg)

    pool = eventlet.GreenPool(50)
    participants = \
        chat_sessions[body['session_id']]
    for user_id in participants:
        pool.spawn_n(
            send_message,
            user_id,
            body['message'])
    pool.waitall()
```

Linden Lab

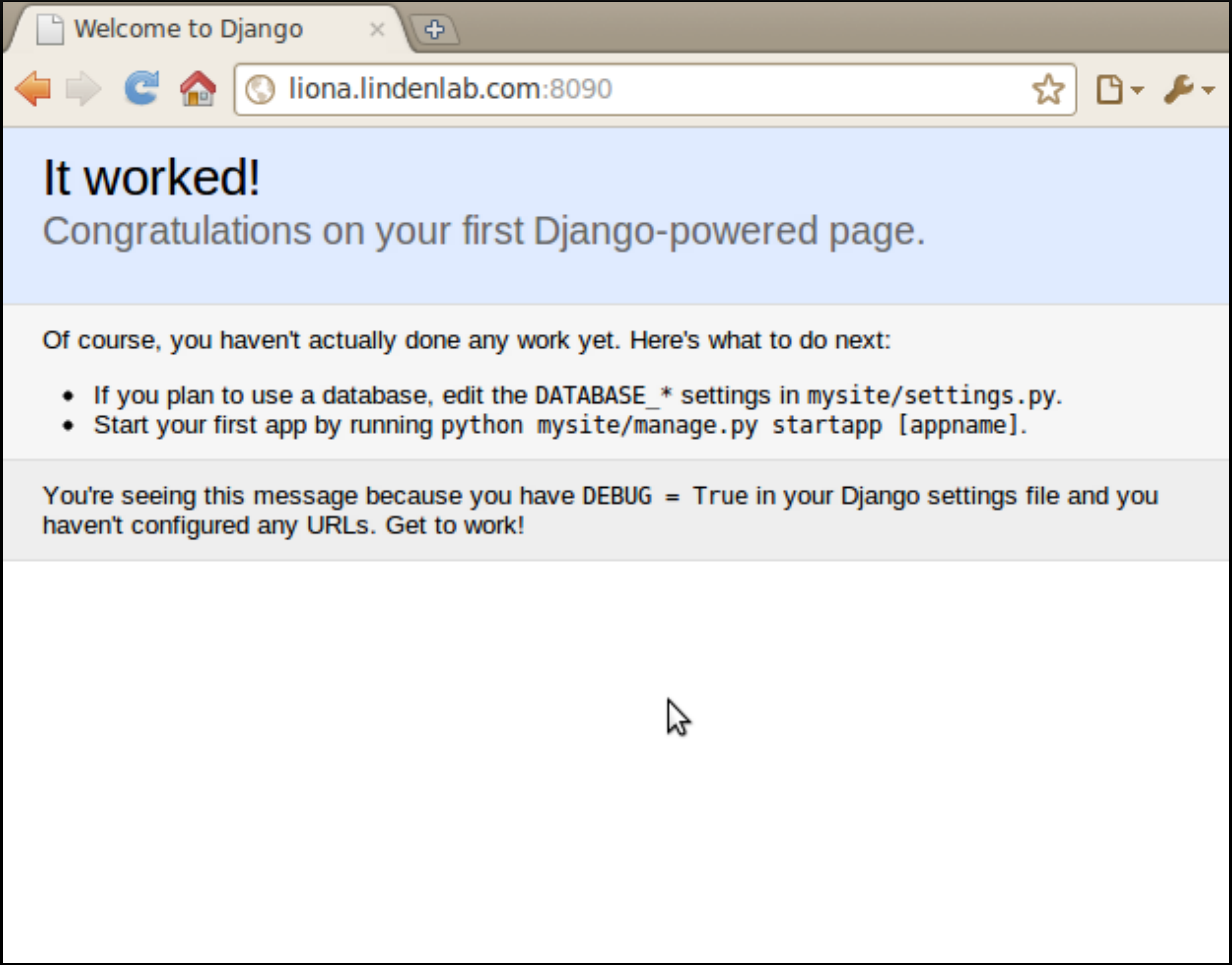
Spawning

Python WSGI Server

Spawning is neat!

```
$ pip install spawning
$ pip install django
$ django-admin startproject mysite
$ spawn \
    --factory=spawning.django_factory.config_factory \
    --port=8090 \
    --processes=4 \
    mysite.settings
(15997) wsgi starting up on http://0.0.0.0:8090/
(15999) wsgi starting up on http://0.0.0.0:8090/
(15996) wsgi starting up on http://0.0.0.0:8090/
(15998) wsgi starting up on http://0.0.0.0:8090/
```

Linden Lab



It worked!
Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Here's what to do next:

- If you plan to use a database, edit the `DATABASE_*` settings in `mysite/settings.py`.
- Start your first app by running `python mysite/manage.py startapp [appname]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

Linden Lab®

When should I use Eventlet?

- Your application does a lot of I/O
- You want deterministic context switching to simplify your code.
- You need to service 1000+ RPS on a single node without a super computer

When not to use Eventlet

- Your program is CPU bound (very little I/O wait)
- You depend on extension modules you can't change that do blocking I/O.
    - eventlet.tpool helps here, but it's not perfect.

How is it used at Linden?

- A bunch of web services
  - Agent presence service (16 master nodes, 500-1000 RPS each).
  - Asset upload system
  - L$ Service
  - Region conductor
  - Many, many others.
- Second Life Enterprise cluster upgrade system
- PyOGP (Python library for writing Second Life bots)
- Lots of misc. tools
  - Shout a message on all regions.
  - Deployment tools
  - others...

Linden Lab®

http://eventlet.net

Linden Lab®