Understanding Non-Blocking I/O

Vaidik Kapoor
github.com/vaidik

EuroPython 2015

High Level Overview

- What is Non Blocking I/O?
- Understanding by examples
- Why should you care?
- Disclaimer: a rather beginner level introduction to the topic

Who am I?

1. Pythonista for about 4 years
2. Infrastructure Engineer at Wingify (responsible for all things systems and operations)
3. Based out of New Delhi, India
4. Social networks:
    a. github.com/vaidik
    b. twitter.com/vaidikkapoor

Some Background

1. Started out as a web developer and moved down the stack
2. Encountered Gevent along the journey
3. Always wondered - how does this thing really work
4. Nobody talks about it

Non-Blocking I/O

OR

What is blocking?

What is Blocking?

A function or a code-block is blocking if it has to wait for anything to complete.

Blocking

1. A blocking function is capable of delaying execution of other tasks, especially those that are independent
    a. In case of a server, other requests may get blocked
    b. In case of a worker consuming tasks from a queue, other independent tasks may get delayed
2. The overall system is not able to progress

I/O

At least for today's applications (not exhaustive):

1. **Dealing with the network**
2. Reading from or writing to disk
3. Operations on Pipe
4. Basically, any kind of operation on a file descriptor (in *NIX terminology).

Non-Blocking I/O

Dealing with I/O in a way so that execution does not get delayed because of it.

Server / Client

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))

data = 'foobar\n' * 10 * 1024 * 1024  # ~ 70 MB of data
assert sock.send(data) == len(data)  # True
```

Server / Client

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))

data = 'foobar\n' * 10 * 1024 * 1024  # ~ 70 MB of data
assert sock.send(data) == len(data)  # True
```

Server / Client

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))

data = 'foobar\n' * 10 * 1024 * 1024  # ~ 70 MB of data
assert sock.send(data) == len(data)  # True
```

Server / Client

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))

data = 'foobar\n' * 10 * 1024 * 1024  # ~ 70 MB of data
assert sock.send(data) == len(data)  # True
```

Server / Client

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))

data = 'foobar\n' * 10 * 1024 * 1024  # ~ 70 MB of data
assert sock.send(data) == len(data)  # True
```

```
$ time python example1-client.py
python example1.1-client.py  0.05s user 0.08s system
0% cpu 45.050 total
```

Non-Blocking Network I/O in Python

At the most basic level, it's all about:

```
$ pydoc socket.socket.setblocking
socket.socket.setblocking = setblocking(...) unbound socket._socketobject method
     setblocking(flag)

     Set the socket to blocking (flag is true) or non-blocking (false).
     setblocking(True) is equivalent to settimeout(None);
     setblocking(False) is equivalent to settimeout(0.0).
```

Server / Client v2

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))
sock.setblocking(0)

data = 'foobar\n' * 10 * 1024 * 1024  # 70 MB of data
sent = sock.send(data)
assert sent == len(data), '%s != %s' % (sent, len(data))
```

```
$ time python example2-client.py
Traceback (most recent call last):
  File "example2-client.py", line 9, in <module>
    assert sent == len(data), '%s != %s' % (sent,
len(data))
AssertionError: 457816 != 73400320

python example2-client.py  0.06s user 0.06s system
89% cpu 0.136 total
```

Server / Client v3

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import errno
import select
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))
sock.setblocking(0)

data = 'foobar\n' * 1024 * 1024
data_size = len(data)
print 'Bytes to send: ', len(data)

total_sent = 0
while len(data):
    try:
        sent = sock.send(data)
        total_sent += sent
        data = data[sent:]
        print 'Sending data'
    except socket.error, e:
        if e.errno != errno.EAGAIN:
            raise e
        print 'Blocking with', len(data), 'remaining'

assert total_sent == data_size  # True
```

Server / Client v4

```python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.bind(('localhost', 1234))
sock.listen(5)

try:
    while True:
        conn, info = sock.accept()

        data = conn.recv(1024)
        while data:
            print data
            data = conn.recv(1024)
except KeyboardInterrupt:
    sock.close()
```

```python
import errno
import select
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 1234))
sock.setblocking(0)

data = 'foobar\n' * 1024 * 1024
data_size = len(data)
print 'Bytes to send: ', len(data)

total_sent = 0
while len(data):
    try:
        sent = sock.send(data)
        total_sent += sent
        data = data[sent:]
        print 'Sending data'
    except socket.error, e:
        if e.errno != errno.EAGAIN:
            raise e
        print 'Blocking with', len(data), 'remaining'
        select.select([], [sock], []) # This blocks

assert total_sent == data_size  # True
```

Understanding select()

- A system call for monitoring events on file descriptors
- select.select() just wraps the select syscall
  - It does make things much simpler than C
  - If you can understand this, then working with the C API would be much simpler

Understanding select()

```
select.select = select(...)
    select(rlist, wlist, xlist[, timeout]) -> (rlist, wlist, xlist)
```

- Takes three sets of fds for monitoring them for reading, writing and exceptions
- Returns three sets with fds that are ready to be read from, written to or handled for exception

Client v5

```
 7 def other_task():
 8     i = 0
 9     while i < 500:
10         i += 1
11         time.sleep(0.01)
12         print i
13         yield
14
15
16 def send_data_task(port, data):
17     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18     sock.connect(('localhost', port))
19     sock.setblocking(0)
20
21     data = (data + '\n') * 1024 * 1024
22     print 'Bytes to send: ', len(data)
23
24     total_sent = 0
25     while len(data):
26         try:
27             sent = sock.send(data)
28             total_sent += sent
29             data = data[sent:]
30             print 'Sending data'
31         except socket.error, e:
32             if e.errno != errno.EAGAIN:
33                 raise e
34
35             # monitor this socket
36             yield ('write', sock)
37
38     print 'Bytes sent: ', total_sent
```

```
41 if __name__ == '__main__':
42     tasks = [
43         other_task(),
44         send_data_task(port=1234, data='foo'),
45     ]
46
47     fds = dict(write={}, read={})
48     while len(tasks) or len(fds['write']) or len(fds['read']):
49         pending_tasks = []
50
51         for task in tasks:
52             try:
53                 val = next(task)
54                 if val is None:
55                     pending_tasks.append(task)
56                 else:
57                     fds[val[0]][val[1]] = task
58             except StopIteration:
59                 pass
60
61         if len(fds['write'].keys()) or len(fds['read'].keys()):
62             readable, writeable, exceptional = select.select(
63                 fds['read'].keys(), fds['write'].keys(), [], 0)
64
65             for readable_sock in readable:
66                 pending_tasks.append(fds['read'][fd])
67                 del fds['read'][fd]
68             for fd in writeable:
69                 pending_tasks.append(fds['write'][fd])
70                 del fds['write'][fd]
71
72         tasks = pending_tasks
```

select and family

1. Other implementations for monitoring file descriptors:
    a. poll - Unix/Linux
    b. epoll - Linux
    c. kqueue - BSD
2. The de-facto today - epoll and kqueue.

One library to rule them all

1. libevent
2. libev
3. libuv
4. more?

In Python World (Libraries)

1. Gevent
    a. Greenlet based
    b. C extension
    c. Probably the easiest to start with for all practical purposes
2. Eventlet
    a. Greenlet based
    b. Pure Python

In Python World (Frameworks)

1. Twisted
    a. Mainloop is called Reactor
    b. Almost all commonly used protocols implemented
    c. Pure Python
    d. Not very-well suited for web apps
2. Tornado
    a. Mainloop is called IOLoop
    b. Pure Python
    c. More focussed for writing webapps

In Python World (Frameworks)

1. asyncio

Questions?