

```
1  def read_message(socket):
2      uname = socket.recv(NAME_SIZE)
3      res = Result(uname)
4      input_msg = socket.recv(INPUT_MSG_SIZE)
5      while "BYE!" != input_msg:
6          res.add_message(input_msg)
7          input_msg = socket.recv(INPUT_MSG_SIZE)
8
9      return res
```



## 10K problem

- <http://www.kegel.com/c10k.html>
- Много времени на переключение в ядро и назад
- Алгоритм шедулера в ядре не совсем подходит для таких задач
- На самом деле решение ближе к кооперативной многозадачности

## select + конечные автоматы

```
1  import select
2  r_rd , w_rd , e_rd = select.select(
3      read_fds , write_fds , err_fds , timeout=0)
4
5  pobj = select.poll()
6  pobj.register(fd , EVENT_TYPE)
7  ...
8
9  fd_end_evts = pobj.poll(timeout=0)
```

## Async libs

- переписываем наш код из хороших и понятных последовательных функций на ужасные конечные автоматы
- регистрируем обработчики событий функции или классы
- запускаем цикл обработки событий
- по мере продвижения обработки запроса одни функции регистрируют другие в качестве "продолжателей дела"
- базовые классы для разных протоколов способны облегчить ситуацию
- Но их на всех не хватит

```

1
2     def new_request(socket):
3         return read_name, Result()
4
5     def read_name(data, res):
6         if len(data) >= NAME_SIZE - len(res.name):
7             res.name += data[:NAME_SIZE - len(res.name)]
8             return read_message(data[NAME_SIZE - len(res.name):], res)
9         else:
10            res.name += data
11            return read_name, res
12
13    def read_message(data, res):
14        #....
15        if message == "BYE!":
16            save(res)
17            return CLOSE_CONN
18        else:
19            #....
20            return read_message, res
21
22    def conn_closed(res):
23        pass
24

```

```
25     io_loop.register(new_request, port=15678, proto=TCP)
26     io_loop.run()
```

```

1  from twisted.internet import reactor, protocol
2  from twisted.protocols import basic
3
4  class PubProtocol(basic.LineReceiver):
5      def __init__(self, factory):
6          self.factory = factory
7
8      def connectionMade(self):
9          self.factory.clients.add(self)
10
11     def connectionLost(self, reason):
12         self.factory.clients.remove(self)
13
14     def lineReceived(self, line):
15         for c in self.factory.clients:
16             c.sendLine("<{}>_{}".format(self.transport.getHost(), line))
17
18 class PubFactory(protocol.Factory):
19     def __init__(self):
20         self.clients = set()
21
22     def buildProtocol(self, addr):
23         return PubProtocol(self)
24

```



```
25 reactor.listenTCP(1025, PubFactory())  
26 reactor.run()
```

## Генераторы. Cogen

```
1  class SocketWrapper(object):
2      def __init__(self, real_socket):
3          self.real_socket = real_socket
4
5      def recv(self, sz):
6          return "READ", self.real_socket, sz
7
8      def send(self, data):
9          return "SEND", self.real_socket, data
10
11  def read_message(socket):
12      uname = yield socket.recv(NAME_SIZE)
13      res = Result(uname)
14      input_msg = yield socket.recv(INPUT_MSG_SIZE)
15      while "BYE!" != input_msg:
16          res.add_message(input_msg)
17          input_msg = yield socket.recv(INPUT_MSG_SIZE)
18
19  return res
```

## Генераторы. Twisted

```
1  from twisted.internet import reactor, protocol, defer, endpoints
2  from twisted.mail import imap4
3  from twisted.python import log, failure
4
5  @defer.inlineCallbacks
6  def main(username="alice", password="secret",
7           strport="ssl:host=example.com:port=993"):
8      endpoint = endpoints.clientFromString(reactor, strport)
9      factory = protocol.Factory()
10     factory.protocol = imap4.IMAP4Client
11     try:
12         client = yield endpoint.connect(factory)
13         yield client.login(username, password)
14         yield client.select('INBOX')
15         info = yield client.fetchEnvelope(imap4.MessageSet(1))
16         print info[1]['ENVELOPE'][1]
17     except:
18         log.err(failure.Failure(), "IMAP4_client_interaction_failed")
19     reactor.stop()
20
21 import sys
22 main(*sys.argv[1:])
```

23      `reactor.run()`

gevent, eventlet, ..

- <http://www.gevent.org/>
- greenlet (part of stackless python spin-off)
- libev inside (c)
- Хак для создания легковесных python потоков в пользовательском режиме
- Автоматически сохраняет состояние точки исполнения при запросе блокирующей операции и передает управление циклу обработки
- ОС пользовательского режима с кооперативной многозадачностью
- `from gevent import monkey; monkey.patch_socket()` и снова пишем нормальный код
- <http://nichol.as/benchmark-of-python-web-servers>

gevent, eventlet, ..

```
1  import gevent
2  from gevent import monkey
3  monkey.patch_all()
4
5  import urllib2
6
7  def print_head(url):
8      data = urllib2.urlopen(url).read()
9      print ( '%s: %s bytes: %r' % (url, len(data), data[:50]))
10
11  urls = [ 'http://www.google.com',
12          'http://www.yandex.ru',
13          'http://www.python.org' ]
14
15  jobs = [gevent.spawn(print_head, url) for url in urls]
16
17  gevent.joinall(jobs)
```

tornado

ДЗ

Написать RPC сервер и клиент к нему. API для сервера:

```
1 c = RPCServer(ip , port)
2
3 def func1(x,y,z):
4     return x + y ** z
5
6 # can register any amount of any functions
7 c.register(func1)
8 c.register(func2)
9
10 c.run()
```

API для клиента:

```
1 c = RPCClient(api , port)
2
3 print c.func1(1, 2, 3) # напечатает 9
```

`pickle`, `marshal`, `exec` и `eval` забанены. На всякий случай - все параметры и имя функции нужно передать на сервер, там вызвать зарегистрированную функцию, получить ее результат и отдать его на клиент. Если функция с запрошенным именем не зарегистрирована или при ее исполнении произошла ошибка - передать сообщение на клиент и выбросить там исключение. Необходимо поддерживать следующие типы данных: `str`, `int`, `list`. Данные передавать через `tcp` сокет.

Бонус:

- Поддерживать мапы и пользовательские классы
- сделать так, что-бы на клиенте можно было получить имена функций питоновским способом
- сделать так, что-бы клиентский объект имел поля-функции, полностью имитирующие функции на сервере. Для этого можно использовать `eval`, но нужно провести полную валидацию строки, которая будет передана в `eval`, для исключения возможности исполнения вредоносного кода при ее вызове.
- Сделать 3 варианта сервера - на потоках (на каждый запрос



порождать новый), на пуле потоков и на `gevent`, сравнить производительность этих решений и объемы ресурсов в зависимости от количества клиентов.