

*Посвящается AZ, NZ и TS: Twisted добился признания,
и мы с нетерпением ждем следующего поколения
разработчиков.*

– Моше Задка (Moshe Zadka)

Содержание

Об авторах	12
Благодарности	14
Введение	15
От издательства	16
Часть I. ОСНОВЫ	17
Глава 1. Введение в событийно-ориентированное программирование с помощью Twisted	18
Примечание о версиях Python	19
Событийно-ориентированное программирование – что это?.....	19
Множественные события.....	20
Мультиплексирование и демultipлексирование	22
Мультиплексор select.....	23
История, аналоги и назначение	23
Сокеты и select	24
События сокета – как, что и почему	25
Обработка событий	26
Цикл обработки событий с select	27
Управляемые событиями клиенты и серверы.....	29
Неблокирующий ввод/вывод.....	31
Знаем, когда нужно остановиться	31
Отслеживание состояния	32
Наличие информации о состоянии усложняет программы	35
Управление сложностью с помощью транспортов и протоколов	36
Реакторы: работа с транспортом.....	37
Транспорты: работа с протоколами	37
Игра в пинг-понг с протоколами и транспортами	38
Клиенты и серверы со своими реализациями протоколов и транспортов	42
Реакторы Twisted и протоколы с транспортами	43
Преимущества событийно-ориентированного программирования	44
Twisted и реальный мир.....	46
События и время.....	50
Повторение событий с LoopingCall	53
Интерфейсы событий в zope.interface.....	55

Управление потоком в событийно-ориентированных программах	57
Управление потоком в Twisted с помощью производителей и потребителей	58
Активные производители	59
Потребители	61
Пассивные производители	64
Итоги	64
Глава 2. Введение в асинхронное программирование с Twisted	66
Обработчики событий и их композиция	66
Что такое асинхронное программирование?	69
Заполнители для будущих значений	70
Асинхронная обработка исключений	72
Введение в Twisted Deferred	76
Обычные обработчики	76
Ошибки и их обработчики	77
Композиция экземпляров Deferred	80
Генераторы и inlineCallbacks	83
yield	83
send	84
throw	86
Асинхронное программирование с inlineCallbacks	87
Сопрограммы в Python	89
Сопрограммы с выражением yield from	90
Сопрограммы async и await	91
Ожидание завершения экземпляров Deferred	96
Преобразование сопрограмм в Deferred с помощью ensureDeferred	97
Мультиплексирование объектов Deferred	98
Тестирование объектов Deferred	102
Итоги	105
Глава 3. Создание приложений с библиотеками treq и Klein	107
Насколько важную роль играют эти библиотеки?	107
Агрегирование каналов	108
Введение в treq	109
Введение в Klein	112
Klein и Deferred	113
Механизм шаблонов Plating в Klein	115
Первая версия агрегатора каналов	117
Разработка через тестирование с использованием Klein и treq	123
Выполнение тестов на устанавливаемом проекте	123
Тестирование Klein с помощью StubTreq	126
Тестирование treq с помощью Klein	133

Журналирование с использованием twisted.logger.....	136
Запуск приложений Twisted с помощью twist.....	141
Итоги	144
Часть II. ПРОЕКТЫ	146
Глава 4. Twisted в Docker	147
Введение в Docker	147
Контейнеры.....	147
Образы контейнеров	148
runc и containerd	149
Клиент	149
Реестр	150
Сборка	150
Многоступенчатая сборка.....	151
Python в Docker	153
Варианты развертывания	153
В виртуальном окружении.....	157
В формате Pex	159
Варианты сборки	160
Один большой образ.....	160
Копирование пакетов wheel между этапами.....	161
Копирование окружения между этапами	161
Копирование файлов Pex между этапами.....	161
Автоматизация с использованием Dockerpy	161
Twisted в Docker	162
ENTRYPOINT и PID 1.....	162
Пользовательские плагины.....	162
NColony.....	162
Итоги	165
Глава 5. Использование Twisted в роли сервера WSGI	166
Введение в WSGI	166
PEP	167
Простой пример.....	168
Базовая реализация.....	170
Пример WebOb.....	172
Пример Pyramid	173
Начало	174
Сервер WSGI.....	174
Поиск кода.....	177
Путь по умолчанию	177
PYTHONPATH	177

setup.py	177
Почему Twisted?.....	178
Промышленная эксплуатация и разработка	178
TLS	179
Индикация имени сервера.....	180
Статические файлы	182
Модель ресурсов	182
Чисто статические ресурсы.....	183
Комбинирование статических файлов с WSGI	185
Встроенное планирование задач.....	186
Каналы управления	189
Стратегии параллельного выполнения.....	191
Балансировка нагрузки	191
Открытие сокета в режиме совместного использования	192
Другие варианты	195
Динамическая конфигурация.....	195
Приложение Pyramid с поддержкой A/B-тестирования.....	195
Плагин для поддержки AMP	197
Управляющая программа	200
Итоги	201

Глава 6. Tahoe-LAFS: децентрализованная файловая

система.....	202
Как работает Tahoe-LAFS	203
Архитектура системы	206
Как система Tahoe-LAFS использует Twisted.....	208
Часто встречающиеся проблемы	208
Инструменты поддержки выполнения в режиме демона	209
Внутренние интерфейсы FileNode	210
Интеграция интерфейсных протоколов	211
Веб-интерфейс	212
Типы файлов, Content-Type, /named/	214
Сохранение на диск.....	215
Заголовки Range.....	215
Преобразование ошибок на возвращающей стороне.....	216
Отображение элементов пользовательского интерфейса: шаблоны Nevow	217
Интерфейс FTP	218
Интерфейс SFTP.....	223
Обратная несовместимость Twisted API	223
Итоги	226
Ссылки	226

Глава 7. Magic Wormhole	227
Как это выглядит.....	228
Как это работает	229
Сетевые протоколы, задержки передачи, совместимость клиентов	231
Сетевые протоколы и совместимость клиентов.....	231
Архитектура сервера	234
База данных	235
Транзитный клиент: отменяемые отложенные операции	235
Сервер транзитной ретрансляции.....	238
Архитектура клиента.....	239
Отложенные вычисления и конечные автоматы, одноразовый наблюдатель.....	240
Одноразовые наблюдатели	243
Promise/Future и Deferred	244
Отсроченные вызовы, синхронное тестирование	247
Асинхронное тестирование с объектами Deferred	248
Синхронное тестирование с объектами Deferred.....	249
Синхронное тестирование и отсроченный вызов.....	252
Итоги	254
Ссылки.....	254
Глава 8. Передача данных в браузерах и микросервисах с использованием WebSocket	255
Нужен ли протокол WebSocket?	255
WebSocket и Twisted.....	256
WebSocket, из Python в Python	258
WebSocket, из Python в JavaScript	261
Более мощная поддержка WebSocket в WAMP.....	263
Итоги	269
Глава 9. Создание приложений с asyncio и Twisted	271
Основные понятия	271
Механизм обещаний	272
Циклы событий.....	273
Рекомендации.....	274
Пример: прокси с aiohttp и treq.....	277
Итоги	280
Глава 10. Buildbot и Twisted	282
История появления Buildbot	282
Эволюция асинхронного выполнения кода на Python в Buildbot	283
Миграция синхронных API	286
Этапы асинхронной сборки	287

Код Buildbot	287
Асинхронные утилиты	288
«Дребезг»	288
Асинхронные службы	288
Кеш LRU	291
Отложенный вызов функций	291
Взаимодействие с синхронным кодом	292
SQLAlchemy	292
requests	293
Docker	295
Конкурентный доступ к общим ресурсам	296
yield как барьер конкуренции	296
Функции из пула потоков не должны изменять общее состояние	297
Блокировки Deferred	298
Тестирование	298
Имитации	300
Итоги	300
Глава 11. Twisted и HTTP/2	301
Введение	301
Цели и задачи	303
Бесшовная интеграция	303
Оптимизация поведения по умолчанию	304
Разделение задач и повторное использование кода	305
Проблемы реализации	306
Что такое соединение? Ценность стандартных интерфейсов	306
Мультиплексирование и приоритеты	309
Противодавление	315
Противодавление в Twisted	317
Противодавление в HTTP/2	319
Текущее положение дел и возможность расширения в будущем	321
Итоги	322
Глава 12. Twisted и Django Channels	323
Введение	323
Основные компоненты Channels	325
Брокеры сообщений и очереди	325
Распределенные многоуровневые системы в Twisted	327
Текущее положение дел и возможность расширения в будущем	328
Итоги	329
Предметный указатель	330

Об авторах

Марк Уильямс (Mark Williams) работает в Twisted. В eBay и PayPal Марк Уильямс работал над высокопроизводительными веб-службами Python (более миллиарда запросов в день!), над обеспечением безопасности приложений и информации, а также переносом корпоративных Python-библиотек на Python.

Кори Бенфилд (Cory Benfield) – разработчик Python с открытым исходным кодом, активно участвует в сообществе Python HTTP. Входит в число основных разработчиков проектов Requests и urllib3 и является ведущим сопровождающим проекта Hyper – коллекции инструментов поддержки HTTP и HTTP/2 для Python, а также помогает в разработке Python Cryptographic Authority для PyOpenSSL.

Брайан Уорнер (Brian Warner) – инженер по безопасности и разработчик программного обеспечения, работавший в Mozilla на Firefox Sync, Add-On SDK и Persona. Один из основателей проекта Tahoe-LAFS – распределенной и защищенной файловой системы, разрабатывает средства безопасного хранения и связи.

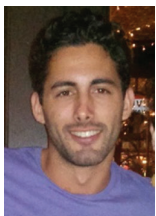
Моше Задка (Moshe Zadka) с 1995 года является участником сообщества открытого исходного кода, свой первый вклад в Python внес в 1998 году, один из основателей открытого проекта Twisted. Любит рассказывать о Twisted и Python и выступать на конференциях. Регулярно ведет блоги.

Дастин Митчелл (Dustin Mitchell) внес свой вклад в Buildbot. Является членом команды TaskCluster в Mozilla. Также работает в командах Release Engineering, Release Operations и Infrastructure.

Кевин Сэмюэл (Kevin Samuel) начал заниматься разработкой и преподаванием еще во времена, когда появилась версия Python 2.4. Работал в Восточной Европе, Северной Америке, Азии и Западной Африке. Тесно сотрудничает с командой Crossbar.io и является активным членом французского сообщества Python.

О технических рецензентах

Пьер Тарди (Pierre Tardy) – специалист по непрерывной интеграции в Renault Software Labs, в настоящее время является ведущим коммитером в Buildbot.



Джулиан Берман (Julian Berman) – разработчик программного обеспечения с открытым исходным кодом из Нью-Йорка. Автор библиотеки jsonschema для Python, периодически вносит вклад в экосистему Twisted, активный участник сообщества Python.

Шон Шоджи (Shawn Shojaie) живет в районе Калифорнийского залива, где работает инженером-программистом. Работал в Intel, NetApp. Сейчас работает в SimpleLegal, где создает веб-приложения для юридических фирм. В будние дни занимается разработкой с использованием Django и PostgreSQL, а в выходные вносит вклад в проекты с открытым исходным кодом, такие как django-pylint, и время от времени пишет технические статьи. Больше можно узнать на сайте shawnshojaie.com.

Том Мост (Tom Most) – Twisted-коммитер с десятилетним опытом разработки веб-служб, клиентских библиотек и приложений командной строки с использованием Twisted. Инженер-программист в телекоммуникационной отрасли. Сопровождает Afkak, клиента Twisted Kafka. Его адрес в интернете – freecog.net, а связаться можно по адресу twm@freecog.net.

Благодарности

Благодарю свою жену Дженнифер Задка (Jennifer Zadka), без чьей поддержки я бы не справился.

Благодарю своих родителей Якова и Пнине Задка (Yaacov and Pnina Zadka), которые хорошо меня учили.

Благодарю своего советника Яэля Каршону (Yael Karshon) за то, что он научил меня писать.

Благодарю Махмуда Хашеми (Mahmoud Hashemi) за вдохновение и поддержку.

Благодарю Марка Уильямса (Mark Williams) за то, что всегда был рядом со мной.

Благодарю Глифа Лефковица (Glyph Lefkowitz), который познакомил меня с Python, учил программировать и вообще быть хорошим человеком.

– *Моше Задка (Moshe Zadka)*

Благодарю Махмуда Хашеми (Mahmoud Hashemi) и Давида Карапетяна (David Karapetyan) за их отзывы. Благодарю Энни (Annie) за то, что была терпелива, пока я писал.

– *Марк Уильямс (Mark Williams)*

Введение

Twisted недавно отпраздновал свой шестнадцатый день рождения. За время своего существования он превратился в мощную библиотеку. За этот период на основе Twisted было создано несколько интересных приложений и многие из нас узнали много нового о том, как правильно использовать Twisted, как думать о сетевом коде и как создавать программы, основанные на событиях.

После ознакомления с вводными материалами, которые есть на сайте Twisted, часто можно услышать: «И что делать дальше? Как я могу узнать больше о Twisted?» На этот вопрос мы обычно отвечали встречным вопросом: «А что вы хотите сделать с Twisted?» В этой книге мы покажем некоторые интересные приемы использования Twisted.

Авторы данной книги использовали Twisted для разных целей и усвоили разные уроки. Мы рады представить все эти уроки, чтобы сообщество их знало и могло воспользоваться.

Вперед!

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Apress очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Часть I



ОСНОВЫ

Глава 1

Введение в событийно-ориентированное программирование с помощью Twisted

Twisted – это мощная, хорошо протестированная, развитая параллельная сетевая библиотека и фреймворк. Как мы в этой книге увидим далее, многие организации и отдельные люди при создании своих проектов эффективно использовали этот фреймворк на протяжении более чем десятка лет.

В то же время Twisted большой, сложный и старый. Его лексикон изобилует странными названиями, такими как «реактор», «протокол», «конечная точка» и «отложенный». Эти понятия описывают философию и архитектуру, сбивающую с толку как новичков, так и людей с многолетним опытом работы с Python.

Сетевые приложения, создаваемые с помощью Twisted, основываются на двух основных принципах программирования: *событийного программирования* и *асинхронного программирования*. Рост популярности языка программирования JavaScript и включение в стандартную библиотеку Python пакета `asyncio` привели к тому, что оба принципа стали основой фреймворка. Но ни один из принципов не занимает настолько доминирующего положения в программировании на Python, чтобы простого знания языка было достаточно для их освоения. Эти сложные темы доступны, пожалуй, только программистам со средним или высоким уровнем подготовки.

В этой и следующей главах представлена мотивация, лежащая в основе событийно-управляемого и асинхронного программирования, а затем показано, как Twisted использует эти принципы. Здесь закладывается основа для последующих глав, в которых рассматриваются реальные программы Twisted.

Сначала мы познакомимся с идеей событийно-ориентированного программирования без привязки к Twisted. Затем, получив представление о том, что определяет событийно-управляемое программирование, мы познакомимся с программными абстракциями в Twisted, помогающими разработчикам писать четкие и эффективные событийные программы. Попутно мы будем делать остановки, чтобы познакомиться с некоторыми уникальными частями этих абстракций, например *интерфейсами*, и исследуем документацию с их описанием на веб-сайте Twisted.

К концу этой главы вы освоите терминологию Twisted: протоколы, транспорты, реакторы, потребители и производители. Данные понятия образуют основу событийно-ориентированного программирования с Twisted. Знание этой основы необходимо для разработки полезного программного обеспечения с Twisted.

ПРИМЕЧАНИЕ О ВЕРСИЯХ PYTHON

Twisted поддерживает Python 2 и 3, поэтому все примеры кода в этой главе могут работать как с Python 2, так и с Python 3. Python 3 – это будущее, но одной из сильных сторон Twisted является его богатая история реализации протоколов; по этой причине важно, чтобы вы умели писать код, способный выполняться под управлением Python 2, даже если прежде вы никогда не писали его.

СОБЫТИЙНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ – ЧТО ЭТО?

Событие – это то, что заставляет управляемую событиями программу выполнить действие. Это широкое определение позволяет рассматривать многие программы как управляемые событиями. Рассмотрим, например, простую программу, которая в зависимости от ввода пользователя печатает либо Hello, либо World:

```
import sys
line = sys.stdin.readline().strip()
if line == "h":
    print("Hello")
else:
    print("World")
```

Появление строки в потоке стандартного ввода является событием. Наша программа приостанавливается на операторе `sys.stdin.readline()`, который просит операционную систему разрешить пользователю ввести законченную строку. Пока ввод не будет получен, программа дальше двигаться не будет. Когда операционная система прочитает очередной ввод и внутренние компоненты Python определяют, что строка завершена, оператор `sys.stdin`.

`readline()` возобновит работу программы, вернув ей полученные данные. Возобновление работы – это событие, толкающее нашу программу вперед. Поэтому даже такую простую программу можно представить как управляемую событиями.

МНОГОКРАТНЫЕ СОБЫТИЯ

Программа, обрабатывающая единственное событие и завершающая работу, не получает никаких преимуществ от событийно-ориентированного подхода. Программы, в которых одновременно может происходить несколько событий, имеют более естественную организацию для их обработки. Примером такой программы может быть графический интерфейс пользователя: в любой момент пользователь может нажать кнопку, выбрать пункт из меню, прокрутить текст и т. д.

Вот версия нашей предыдущей программы с графическим интерфейсом Tkinter:

```
from six.moves import tkinter
from six.moves.tkinter import scrolledtext
class Application(tkinter.Frame):
    def __init__(self, root):
        super(Application,self). __init__ (root)
        self.pack()
        self.helloButton = tkinter.Button(self,
                                          text="Say Hello",
                                          command=self.sayHello)
        self.worldButton = tkinter.Button(self,
                                          text="Say World",
                                          command=self.sayWorld)
        self.output = scrolledtext.ScrolledText(master=self)
        self.helloButton.pack(side="top")
        self.worldButton.pack(side="top")
        self.output.pack(side="top")
    def outputLine(self, text):
        self.output.insert(tkinter.INSERT, text+ '\n')
    def sayHello(self):
        self.outputLine("Hello")
    def sayWorld(self):
        self.outputLine("World")

Application(tkinter.Tk()).mainloop()
```

Эта версия нашей программы представляет пользователю две кнопки, каждая из которых может генерировать независимое событие (click) и отличается от предыдущей программы тем, что в предыдущей программе `sys.stdin.readline` может генерировать только одно событие: «готовность строки».

С этими событиями мы связали *обработчики событий*, которые вызываются нажатием любой кнопки. Кнопки Tkinter имеют свойство `command` со ссылкой

на обработчик, и вызывают этот обработчик при нажатии. Так, когда нажимается кнопка **Say Hello**, она генерирует событие, вынуждающее программу вызвать функцию `Application.sayHello`, которая, в свою очередь, выводит слово `Hello` в текстовое окно с прокруткой. То же происходит при нажатии кнопки **Say World**, которая вызывает функцию `Application.sayWorld` (см. рис. 1.1).



Рис. 1.1 ❖ Приложение Tkinter GUI
после нескольких нажатий кнопок **Say Hello** и **Say World**

Метод `tkinter.Frame.mainloop`, унаследованный нашим классом `Application`, ждет, пока связанная с ним кнопка сгенерирует событие, и запускает соответствующий обработчик. После выполнения каждого обработчика `tkinter.Frame.mainloop` переходит к ожиданию новых событий. Цикл ожидания событий и их передачи в соответствующие обработчики типичен для управляемых событиями программ и известен как *цикл событий*.

Вот основные понятия, лежащие в основе событийно-ориентированного программирования.

1. *События* показывают, что произошло то, на что должна реагировать программа. В обоих наших примерах события естественно соответствуют нажатиям кнопок, но, как мы увидим, они могут представлять все, что заставляет нашу программу выполнять какое-либо действие.
2. *Обработчики событий* определяют реакцию программы на события. Иногда обработчик события имеет вид простой инструкции, как вызов `sys.stdin.readline` в нашем примере. Но чаще всего он представлен функцией или методом, как в нашем примере `tkinter`.

3. *Цикл событий* ждет появления события и вызывает соответствующий обработчик. Не все управляемые событиями программы имеют цикл событий; в нашем первом примере `sys.stdin.readline` цикла обработки событий нет, потому что здесь происходит только одно событие. Однако большинство программ напоминают наш пример `tkinter` тем, что перед окончательным завершением обрабатывают много событий. Такие программы используют цикл событий.

МУЛЬТИПЛЕКСИРОВАНИЕ И ДЕМУЛЬТИПЛЕКСИРОВАНИЕ

Способ ожидания появления событий в цикле существенно влияет на написание управляемых событиями программ, поэтому рассмотрим его более подробно. Вернемся к нашему примеру `tkinter` с двумя кнопками. Цикл событий внутри `mainloop` должен ждать, пока пользователь не нажмет одну из двух кнопок.

Простая реализация цикла может выглядеть следующим образом:

```
def mainloop(self):
    while self.running:
        ready = [button for button in self.buttons if button.hasEvent()]
        if ready:
            self.dispatchButtonEventHandlers(ready)
```

Цикл `mainloop` в ожидании нового события постоянно *опрашивает* каждую кнопку и запускает обработчики только для имеющих готовое событие. Когда событий нет, программа никаких действий не выполняет, так как никаких действий, требующих ответа, предпринято не было. В течение периодов бездействия управляемая событиями программа свое выполнение должна приостанавливать.

Цикл `while` в нашем примере `mainloop` приостанавливает работу программы до тех пор, пока не будет нажата одна из кнопок и не понадобится вызвать функцию `sayHello` или `sayWorld`. Если пользователь не сможет щелкнуть мышью со сверхъестественной скоростью, большая часть времени цикла будет тратиться на проверку кнопок, которые не были нажаты. Это называется *активным ожиданием*.

Активное ожидание, подобное этому, приостанавливает общее выполнение программы до тех пор, пока один из ее источников событий не сообщит о том, что событие произошло. Поэтому активного ожидания достаточно для приостановки цикла событий.

Внутренний генератор списков, управляющий активным ожиданием, задает важный вопрос: что-нибудь произошло? Ответ помещается в переменную `ready` и имеет вид списка со всеми кнопками, которые были нажаты. Истинность переменной `ready` определяет ответ на вопрос: если переменная `ready` ничего не содержит (имеет ложное значение), значит, кнопки не нажи-

мались и ничего не произошло. Если переменная имеет истинное значение, следовательно, событие произошло и по меньшей мере одна из кнопок была нажата.

Генератор списков, формирующий значение для `ready`, объединяет множество событий в одном месте. Этот процесс известен как *мультиплексирование*. Обратный процесс разделения списка на отдельные события называется *демультиплексированием*. Генератор списка объединяет (мультиплексирует) все наши кнопки в переменной `ready`, тогда как метод `dispatchButtonEventHandlers` демультиплексирует (перенаправляет сигнал с одного из информационных входов на один из информационных выходов) их, вызывая обработчик каждого события отдельно.

Теперь мы можем уточнить наше понимание циклов событий, точно описав, как происходит ожидание событий.

- *Цикл событий* ожидает появления событий, мультиплексируя их источники в один список. Если получился непустой список, цикл событий демультиплексирует его и для каждого события вызывает соответствующий обработчик.

Наш мультиплексор `mainloop` тратит большую часть своего времени на опрос кнопок, которые не были нажаты. Но не все мультиплексоры настолько неэффективны. В `tkinter.Frame.mainloop` используется аналогичный мультиплексор, который опрашивает все виджеты, если операционная система не предлагает более эффективного способа. Мультиплексор `mainloop`, чтобы повысить свою эффективность, использует тот факт, что компьютеры могут проверять виджеты GUI быстрее, чем с ними может взаимодействовать человек, и вставляет вызов `sleep`, приостанавливающий программу на несколько миллисекунд. Это позволяет программе часть времени в цикле активного ожидания вообще не выполнять никаких операций и за счет небольшой задержки экономить процессорное время и электроэнергию.

Хоть Twisted может интегрироваться с графическими пользовательскими интерфейсами и фактически имеет специальную поддержку `tkinter`, в его основе лежит сетевой движок. В сети основными источниками событий являются *сокеты*, а не кнопки, и операционные системы предлагают эффективные механизмы для мультиплексирования событий сокетов. Цикл событий в Twisted использует эти механизмы. Чтобы понять подход Twisted к событийному программированию, нужно разобраться, как взаимодействуют сокеты с этими механизмами мультиплексирования.

МУЛЬТИПЛЕКСОР SELECT

История, аналоги и назначение

Мультиплексор `select` поддерживается большинством операционных систем. Свое имя «select» (выбор) этот мультиплексор получил из-за своей способно-

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru