Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ
Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра дискретной математики и
информационных технологий

Изучение сетевых протоколов и возможностей их реализации на Python

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студентки 2 курса 271 группы
направления 09.04.01 «Информатика и вычислительная техника»
факультета компьютерных наук и информационных технологий
Мелешко Анастасии Юрьевны

Научный руководитель
к. ф.- м.н., доцент _____ В.А. Поздняков
подпись, дата

Зав. кафедрой
к. ф.- м.н., доцент _____ Л.Б. Тяпаев
подпись, дата

Саратов 2018

ВВЕДЕНИЕ Актуальность работы. На сегодняшний день сетевые технологии находятся на достаточно высоком уровне развития. Сетевые приложения стали неотъемлемой частью жизни миллионов людей — это серверы баз данных, сетевые игры, различные сетевые протоколы, web-серверы и другие. Поэтому как никогда актуальна разработка клиент-серверных приложений, с помощью которых можно решать множество различных задач. Сетевое приложение состоит из нескольких взаимодействующих частей, каждая из которых выполняет какую-то определенную законченную работу по решению прикладной задачи, причем каждая часть может выполняться и, как правило, выполняется на отдельном компьютере сети. Части распределенного приложения взаимодействуют друг с другом, используя сетевые службы и транспортные средства ОС. Как правило, компьютеры и программы не равноправны. Некоторые из них владеют ресурсами, другие имеют возможность обращаться к этим ресурсам. Компьютер или программа, управляющие ресурсом, называют сервером.

Для сетевого программирования на любом языке широко используются сокеты. Это один из интерфейсов межпроцессного взаимодействия, позволяющий разрабатывать клиент-серверные системы для локального или сетевого использования. Несмотря на то, что они часто используются для организации общения по сети, они так же могут использоваться как механизм общения между программами, выполняющимися на одном компьютере, принимая форму механизма взаимодействия между процессами.

Целью данной работы является разработка серверных приложений на основе методов многопоточности и асинхронности с использованием возможностей языка Python.

Для достижения данной цели необходимо выполнить следующие задачи:

1) Изучить основные сетевые протоколы.
2) Изучить возможности языка Python для сетевого программирования.

3) Рассмотреть основные методы программирования сокетов с использованием возможностей языка Python.
4) Применить данные методы программирования сокетов для реализации сетевых приложений, использующих сокеты TCP и UDP.
5) Рассмотреть методы многопоточности и асинхронности для программирования серверов на языке Python.

Работа выполнена на 63 страницах машинописного текста, состоит из введения, трех глав, заключения, содержит 27 рисунков, двух приложений, список литературных источников содержит 20 наименований.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ. В первой главе «Протоколы передачи данных» говорится о наиболее распространённой системе классификации сетевых протоколов, которой является эталонная модель OSI. Данная модель была разработана компанией ISO (International Organization for Standardization) в 1984 году как концептуальная модель стандартов для взаимодействия в сети оборудования и приложений различных производителей. В настоящее время модель OSI считается основной архитектурной моделью для межмашинных и межсетевых коммуникаций. Большинство используемых сегодня сетевых протоколов передачи информации имеют структуру, основанную на модели OSI. Модель OSI выделяет в процессе обмена информацией семь уровней, разделяя задачи, связанные с передачей информации между сетевыми машинами, на семь небольших, более управляемых групп задач. Затем задача или группа задач направляется каждому из семи слоев модели. Каждый уровень самодостаточен, поэтому задачи, направленные каждому слою, могут быть реализованы независимо, что позволяет совершенствовать решения, предлагаемые одним слоем, без ущерба для других.

Модель OSI, как и любая другая модель сетевой коммуникации, обеспечивает только концептуальную структуру для обмена данными между компьютерами, но сама модель не предлагает конкретных методов связи.

Фактический процесс общения осуществляется с помощью различных протоколов передачи данных. В контексте передачи данных протокол представляет собой формальный набор правил, условных обозначений и структуры данных, который определяет, как компьютеры и другие сетевые устройства обмениваются информацией по сети. Другими словами, протокол представляет собой стандартную процедуру и формат, которые должны понимать, принимать и использовать два устройства передачи данных, для осуществления процесса общения друг с другом.

Так же в данной главе были рассмотрены некоторые сетевые протоколы, такие как стек протоколов TCP/IP, протоколы UDP, HTTP, FTP, POP3, SMTP, TELNET, URL.

Во второй главе «Сетевые приложения на языке Python» говорится о методах программирования сетевых приложений на языке Python и о из применении для реализации сетевых приложений, использующих сокеты TCP и UDP. В Python для сетевого программирования широко используются сокеты. Сокет — это программный интерфейс для организации соединений между программами, которые могут выполняться на разных компьютерах в сети. Сокеты являются двунаправленными потоками данных, с их помощью программы могут отправлять и принимать данные. Для программиста сокеты принимают форму группы вызовов, доступных через библиотеку. Эти вызовы сокетов умеют пересылать байты между компьютерами, используя низкоуровневые механизмы, такие как сетевой протокол TCP управления передачей данных.

Сокетное программирование основано на архитектуре «клиент-сервер». Такая архитектура определяет общие принципы организации взаимодействия в сети, где имеются серверы, поставщики некоторых специфичных сервисов, и клиенты, потребители этих функций.

Основным интерфейсом сокетов в Python является стандартный библиотечный модуль socket. Модуль socket обеспечивает возможность выполнения операций с сокетами в любой системе (Windows, Mac OS, Linux,

Unix и т.д.), тем самым обеспечивая переносимый интерфейс сокетов. Кроме того, данный модуль поддерживает все стандартные типы сокетов и может использоваться как прикладной интерфейс доступа к сети и как универсальный механизм взаимодействий между процессами, выполняющимися на одном и том же компьютере.

Чтобы установить связь между компьютерами, программы на языке Python импортируют модуль socket, создают объект сокета и вызывают методы этого объекта для установления соединения, отправки и получения данных.

Существуют как клиентские, так и серверные сокеты. К серверным сокетам можно отнести: связывание сокета с хостами и портом и прослушивание определенного порта. Клиентский сокет производит подключение к серверу.

В третьей главе «Методы многопоточности и асинхронности» описаны методы многопоточности и асинхронности для программирования серверных приложений на языке Python, использующих сокет TCP.

Многопоточность – это самый простой и самый популярный метод выполнять более одного действия за раз. Суть многопоточного приложения заключается в том, что один или несколько клиентов обрабатываются в одном потоке, что очень удобно, особенно при использовании блокирующих сокетов, так как потоки работают параллельно друг другу и не останавливают выполнение основного потока.

В стандартную библиотеку Python входят два модуля работы с потоками:

- _tread – основной низкоуровневый интерфейс;
- threading – интерфейс более высокого уровня, основанный на объектах и классах. Внутри модуля threading используется _thread для реализации объектов, представляющих потоки и инструменты синхронизации.

Потоки в модуле threading реализуются с помощью класса Thread. Этот класс можно также использовать для запуска простых функций и вызываемых объектов других типов. Метод run класса Thread по умолчанию просто вызывает объект, переданный конструктору в аргументе target, со всеми дополнительными аргументами (args), которые по умолчанию являются пустым списком [18].

Передача простой функции:

```
t = threading.Thread(target=self.handler, args=(conn, addr))
t.daemon = True
t.start()
```

При использовании модуля threading программа не может завершиться, пока хотя бы один поток продолжает работу, если только это поток не был запущен как поток-демон. В таком случае программа завершиться, когда в ней останутся только потоки-демоны. При создании поток наследуют этот признак от потока, его породившего. Главный поток в программах на языке Python не может быть демоном, тогда как потоки, созданные без помощи этого модуля, считаются демонами. Чтобы переопределить признак, унаследованный по умолчанию, можно вручную установить атрибут daemon объекта потока.

Но существует другая техника, которая имеет практически все достоинства многопоточности, при этом, не используя множество потоков. Асинхронная модель построена на очереди событий. При возникновении некоторого события оно помещается в конец очереди. Модуль asyncore.py из стандартной библиотеки Python реализует модель обратного вызова, основанную на классах, в которой обратные вызовы для ввода и вывода переадресуются методам класса, уже реализованным циклом событий select. Таким образом, он позволяет строить серверы без потоков выполнения и ветвлений. Как и для любых других серверов этого типа, модуль asyncore лучше всего использовать в ситуациях, когда обслуживание клиента занимает короткий промежуток времени, то есть когда основная работа

связана с выполнением операций ввода-вывода, а не с вычислениями, так как в последнем случае необходимо использовать потоки или ветвление.

Метод многопоточности является простым и самым популярным методом, позволяющим за одну единицу времени выполнять несколько операций. Метод асинхронности лучше всего использовать в ситуациях, когда обслуживание клиента занимает небольшой промежуток времени, то есть при выполнении операций ввода—вывода, а не с вычислениями, так как в этом случае более производительным является метод многопоточности.

ЗАКЛЮЧЕНИЕ

В ходе проделанной научно-исследовательской работы были выполнены все поставленные задачи. Были рассмотрены основные сетевые протоколы, а так же изучены возможности языка Python для сетевого программирования.

Были описаны основные методы программирования сокетов для клиентских и серверных приложений с использованием возможностей языка Python. Данные методы программирования сокетов были применены при реализации клиент-серверных приложений, использующих сокеты TCP и UDP.

В ходе данной работы были так же рассмотрены методы многопоточности и асинхронности для программирования серверов на языке Python. На основе данных методов многопоточности и асинхронности мною были разработаны серверные приложения с помощью рассмотренных методов программирования сокетов на языке Python.