SMTP-клиент

Индивидуальное задание

Разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее функции клиента протокола SMTP.
Основные возможности. Приложение должно реализовывать следующие функции:

1. Создание нового письма, включающего такие поля, как From (от- правитель), To (получатель), Subject (тема), Carbon copy (дополни- тельные адресаты), Blind copy (дополнительные скрытые адресаты), Body (текст)
2. Формирование всех необходимых заголовков письма, с тем, чтобы приёмная сторона не рассматривала данное письмо как спам.
3. Подключение к указанному SMTP-серверу и отсылка созданного письма
4. Подробное протоколирование соединения клиента с сервером

Поддерживаемые команды. Разработанное приложение должно реализовывать следующие команды протокола SMTP:

- HELO – передача серверу информации о домене пользователя
- MAIL FROM – передача серверу адреса отправителя письма
- RCPT TO – передача серверу адреса получателя письма
- DATA – передача серверу тела письма
- QUIT – завершение сеанса связи

Настройки приложения. Разработанное приложение должно обеспечивать настройку следующих параметров:

1. Собственное доменное имя для передачи в команде HELO
2. Адрес отправителя
3. IP-адрес или доменное имя сервера SMTP

Методика тестирования. Для тестирования приложения следует использовать почтовые серверы, имеющиеся в лаборатории, а также бесплатные почтовые серверы, имеющиеся в сети Internet ([http://www.mail.ru] (http://www.mail.ru), [http://www.yandex.ru] (http://www.yandex.ru), [http://www.rambler.ru] (http://www.rambler.ru), и т.п.). Средствами разработанного приложения осуществляется передача письма на указанные ящики электронной почты. Штатными клиентами электронной почты проверяется корректность доставки почты и правильность параметров письма.

Теоритические сведения протокола SMTP

Предоставленная информация соответсвует [RFC 5321 — Протокол SMTP] (http://rfc2.ru/5321.rfc/print)

Базовая структура

Когда у клиента SMTP имеется сообщение для передачи, он организует двухсторонний канал связи с сервером SMTP. Клиент SMTP отвечает за доставку почтовых сообщений одному или множеству серверов SMTP или возврат сообщения об отказе. После организации коммуникационного канала и согласования параметров клиент SMTP обычно инициирует почтовую транзакцию. Такая транзакция состоит из последовательности команд, задающих отправителя и получателя сообщения, а также передачи содержимого письма (включая все заголовки и прочие структуры). Если одно сообщение передается множеству адресатов, разумно передавать одну копию сообщения для всех получателей, доставка которым осуществляется на один хост или через один промежуточный транслятор. Сервер обеспечивает отклик на каждую полученную команду — отклик может показывать восприятие команды (в таких случаях ожидаются дополнительные команды), а также содержать сообщение о временной или постоянной ошибке. Команды, задающие отправителя или получателей, могут включать поддерживаемые сервером SMTP расширения, описанные ниже. Диалог между клиентом и сервером осуществляется поэтапно (команда — отклик — команда …), хотя можно использовать по взаимному согласию конвейерную обработку. После завершения передачи сообщения клиент может запросить разрыв соединения или инициировать следующую почтовую транзакцию. Кроме того, клиент SMTP может использовать соединение с сервером для доступа к дополнительному сервису типа проверки корректности почтовых адресов или получения адресов из списка рассылок.

Обзор процедур SMTP

В этом разделе приведены описания процедур, используемых в SMTP.

Инициирование сеанса

Сеанс SMTP инициируется, когда клиент соединяется с сервером и сервер отвечает соответствующим сообщением. Реализация сервера SMTP может включать идентификацию своих программ и сведения об их версии в отклик подтверждения соединения после кода 220. Протокол SMTP позволяет серверу формально отвергать транзакцию, не запрещая изначальные соединения: код 554 может возвращаться в открывающем сообщении взамен кода 220. Сервер, использующий такой вариант, должен по-прежнему ждать, пока клиент передаст команду QUIT перед закрытием соединения, а на любую мешающую команду следует возвращать отклик 503 bad sequence of commands (некорректная последовательность команд). Поскольку попытка организации SMTP-соединения с такими системами может приводить к

ошибке, серверу, возвращающему код 554, следует передавать вместе с кодом информацию, которая позволит передающей системе понять причину ошибки.

Инициирование клиента

После того, как сервер передал приглашающее сообщение (приветствие) и клиент получил его, последний обычно передает серверу команду EHLO, идентифицирующую клиента. В дополнение к открытию сеанса использование EHLO показывает, что клиент способен работать с расширенным сервисом и запрашивает у сервера список поддерживаемых им расширений. Старые системы SMTP, не способные поддерживать расширения сервиса, и современные клиенты, которым не требуется расширенный сервис в инициируемом почтовом сеансе, могут использовать HELO взамен EHLO. Для серверов недопустимо возвращать расширенные отклики в стиле EHLO в ответ на команду HELO. Для конкретной попытки соединения, если сервер возвращает отклик command not recognized (команда не распознана) на команду EHLO, клиенту следует начать процесс заново и передать команду HELO. Хост, передающий команду EHLO, идентифицирует в ней себя; команду можно интерпретировать как «Hello, I am » (Привет, я домен …), а для случая EHLO — «and I support service extension requests» (и я поддерживаю расширения …).

Почтовые транзакции

Почтовая транзакция SMTP состоит из трех этапов. Началом транзакции служит команда MAIL, дающая идентификацию отправителя. После этого следует одна или несколько команд RCPT, указывающих получателей сообщения. Последний этап транзакции начинается командой DATA, которая инициирует передачу почтовых данных и завершается индикатором end of mail, который также подтверждает транзакцию.

Первым этапом транзакции является команда MAIL.
MAIL FROM: [SP]
Эта команда говорит получателю SMTP о начале новой почтовой транзакции и сбрасывает все таблицы состояний и буферы, включая любые данные получателя или почтовые данные. Часть (обратный путь) первого или единственного аргумента команды содержит название почтового ящика отправителя (между скобками < и >), которое может использоваться для передачи отчетов об ошибках. Восприняв команду, сервер SMTP возвращает отклик 250 OK. Если указанный почтовый ящик по каким-то причинам неприемлем, сервер должен возвратить отклик, показывающий временной тип отказа — постоянная (т. е., повторится при повторе команды клиентом) или временная (т. е., адрес клиента может быть принят при следующем вызове) ошибка.

Вторым этапом транзакции является команда RCPT. Данный этап может повторяться много раз.
RCPT TO: [SP]
Первый или единственный аргумент этой команды включает прямой путь forward-path (обычно имя почтового ящика и домена, обязательно заключенные в скобки <>), идентифицирующий получателя. Восприняв команду, сервер SMTP возвращает

отклик 250 OK и сохраняет прямой путь. Если известно, что почта не может быть доставлена адресату, сервер SMTP возвращает отклик 550, обычно сопровождаемый строкой типа “no such user - ” с именем почтового ящика, для которого невозможна доставка (возможны также другие обстоятельства и коды возврата).

Третьим этапом транзакции является команда DATA (или соответствующая команда протокольного расширения). DATA Восприняв команду, сервер SMTP возвращает промежуточный отклик 354 Intermediate и рассматривает все последующие строки, вплоть (но не включая) до индикатора завершения почтовых данных, как текст сообщения. При успешном приеме всего текста сервер сохраняет полученные данные и возвращает отправителю отклик 250 OK. Протокол SMTP использует для обозначения конца почтовых данных точку в пустой строке. Индикатор завершения почтовых данных также подтверждает почтовую транзакцию и говорит серверу SMTP, что нужно обрабатывать сохраненные пользовательские и почтовые данные. Восприняв данные, сервер SMTP возвращает отклик 250 OK. Сбой при обработке команды DATA может происходить только на двух этапах обмена данными. Если команды MAIL и RCPT не были введены или были отвергнуты, сервер может возвращать отклик command out of sequence (503) или no valid recipients (554 — нет корректных получателей) в ответ на команду DATA. При получении одного из таких откликов (или любого отклика 5yz) для клиента недопустима передача данных серверу (точнее, передача данных недопустима, пока не будет получен отклик 354). Если команда воспринята и передан отклик 354, невыполнение команды DATA может быть связано только с неполнотой почтовой транзакции (например, не указан адресат), недоступностью ресурсов (включая и неожиданнуюнедоступность сервера) или отказом сервера от обработки сообщения в соответствии с заданной политикой или по иным причинам.
Команды почтовых транзакций должны использоваться в приведенном выше порядке.

Порядок команд

Сеанс, который будет включать почтовую транзакцию, должен быть сначала инициализирован командой EHLO. Серверам SMTP следует воспринимать без инициализации команды, не использующие почтовых транзакций. Команда EHLO может вводиться клиентом в действующем сеансе. При первом использовании команды в данной сессии сервер SMTP должен очистить все буферы и сбросить состояние, как при получении команды RSET. Если команда EHLO неприемлема для сервера SMTP, он должен возвращать отклик 501, 500, 502 или 550. Сервер SMTP должен сохранять после передачи таких откликов состояние, которое было до получения команды EHLO. Клиент SMTP должен (по возможности) предоставлять в параметрах команд EHLO первичное доменное имя (не CNAME или MX) своего хоста.
Команда MAIL начинает почтовую транзакцию. После начала транзакции последняя включает начальную команду, одну или несколько команд RCPT и команду DATA, вводимые в указанном порядке. Почтовая транзакция прерывается командой RSET, новой командой EHLO или командой QUIT. В сеансе может происходить множество последовательных транзакций или не быть транзакций вообще. Недопустимо

передавать команду MAIL, если почтовая транзакция уже открыта, т. е., эту команду можно передавать только при отсутствии в сеансе продолжающейся почтовой транзакции — предыдущая транзакция должна быть завершена успешным выполнением команды DATA или прервана командой RSET или новой командой EHLO. Если аргумент начинающей транзакцию команды неприемлем, должен возвращаться отклик 501 и сервер SMTP должен сохранять свое состояние. Если в сеансе нарушается порядок команд в такой степени, что это препятствует их выполнению сервером, последний должен возвратить отклик 503, сохраняя свое состояние. Последней командой сеанса должна быть команда QUIT. Клиентам следует использовать команду QUIT для разрыва соединения даже в тех случаях, когда команда организации сеанса не была передана и воспринята.

Расширения протокола

AUTH – аутентификация и шифрование

AUTH mechanism [initial-response]

Аргументы:

- mechanism - строка идентифицирующая SASL-механизм аутентификации;
- initial-response - опциональный base64-кодированный ответ.

После успешного выполнения команды AUTH, выполнить её в данном сеансе повторно уже нельзя. После успешного завершения команды AUTH, сервер ДОЛЖЕН (MUST) отклонять любые дальнейшие команды AUTH с кодом ответа 503.

Команда AUTH недопустима в процессе выполнения mail-транзакции.

Команда AUTH сообщает серверу механизм аутентификации. Если сервер поддерживает запрашиваемый механизм аутентификации, то он выполняет аутентификационный протокольный обмен, для того чтобы установить подлинность и идентифицировать пользователя. Опционально сервер также договаривается об уровне безопасности. В случае если запрашиваемый механизм аутентификации не поддерживается, сервер отклоняет команду с кодом ответа 504.
Аутентификационный протокольный обмен состоит из серии запросов сервера и ответов клиента зависящих от механизма аутентификации. При получении команды аутентификации от клиента, сервер отправляет клиенту ответ с кодом 334 (ответ о готовности) и текстовой частью содержащей BASE64-закодированную строку. Ответ клиента состоит из BASE64-закодированной строки. Если клиент желает отменить аутентификационный обмен, он должен послать строку с единственным символом “*”. Если сервер получает такой ответ, он ДОЛЖЕН отменить команду аутентификации AUTH и выдать ответ с кодом 501.
В случае если сервер не может декодировать BASE64-аргумент, он отклоняет команду AUTH с кодом ответа 501. Если сервер отклоняет утентификационные данные, то ему СЛЕДУЕТ отклонить команду AUTH с кодом ответа 535. При успешном завершении клиентом аутентификационного обмена, SMTP сервер

отвечает кодом ответа 235.

Архитектура приложения

Команды клиента и ответы сервера

Команда	Описание	Положительный ответ сервера
HELO [домен]	Передает серверу домен отправителя	250
AUTH LOGIN	Авторизация на сервере. Сервер проверяет, зарегистрирован ли пользователь	334 – ответ на команду. Далее клиент должен послать закодированный логин. 334 – ответ на логин. Далее клиент должен послать закодированный пароль. 235 – авторизация пройдена успешно.
MAIL FROM [адрес]	Передает серверу адрес отправителя	250
RCPT TO [адрес]	Передает серверу адрес получателя	250
DATA [текс письма]	Передается письмо, со всеми заголовками и полями. Конец письма – точка в пустой строке	354 – после отправки DATA. 250 – после отправки письма.
QUIT	Разрыв соединения	221

Команда DATA имеет собственные заголовки, они добавляются для передачи более полной информации о сообщении. К тому же, отсутствие заголовков помогает приемной стороне идентифицироть спам. В нашей программе добавим следующие заголовки: From, to, Date, Reply-To, Subject, Content-Transfer-Encoding, Content-Type, BCC, CC. Каждое поле заголовка содержит имя, после которого следует двоеточие, а затем следует значение этого поля. Тело - это содержимое сообщения от отправителя к получателю. RFC 822 определяет тело сообщения как текстовые строки в формате NVT ASCII. Когда происходит передача с использованием команды DATA, заголовки передаются первыми, за ними следует пустая строка и затем следует тело сообщения.

Команда	Описание
From: [ ]	Адрес отправителя
To: [ ]	Адрес получателя
Reply-To: [ ]	Адрес, на который следует посылать ответ
Subject: [ ]	Тема сообщения
Content-Transfer-Encoding: base64	Этот заголовок относится к MIME, стандартному методу помещения в письмо нетекстовой информации. Он не имеет никакого отношения к доставке почты, отвечает только за то, как программа-получатель интерпретирует содержимое сообщения.
Content-	

Type: text/plain; charset=ISO-8859-1	Еще один MIME-заголовок, сообщающий почтовой программе о типе данных, хранящихся в сообщении.
BCC: [ ]	Имена и адреса получателей письма, чьи адреса не следует показывать другим получателям
CC: [ ]	Имена и адреса вторичных получателей письма, к которым направляется копия
BODY	Тело письма отделяется от заголовка пустой строкой, а заканчивается строчкой, состоящей из единственной точки (и символа перевода строки)

В приложении реализованы основные команды SMTP- протокола:

- HELO
- MAIL FROM
- RCPT TO
- DATA

Возможно использваоние команды EHLO вместо HELO, но в этом нет необходимсоти.
Команда AUTH LOGIN используется для аутентификации клиента на сервере. Без этой команды не удалось подключиться к серверу, поэтому было принято решение в её использовании.
Использование команды DATA без дополнительных заголовков приводит к тому, что сервер отмечает сообщения как спам. Во избежания этого используются дополнительные заголовки, тогда сообщения приходят корректно.

Классы и методы

Файл smtp.py.
В файле определен класс smtpClient. В классе имеются следующие поля:

- connect - тип boolean, изначально False. True, если клиент подключился к серверу.
- auth - тип boolean, изначально False. True, если клиент аутентифицировался.
- set - множество получателей письмо, множество формируется из полей пакета RCPT TO, BCC, CC.
- sock - сокет для отправления и према сообщений.

В классе реализованы функции:

- sendCommandToServer(self,com) - отправление команды на сервер
- encrBase64(self,string) - шифрование алгоритмом base64 при аутентификации
- comConnect(self,string,port) - подключение к серверу
- helo(self,string) - реализация команды HELO

- setAuth(self,login,password) - реализация команды AUTH (аутентификация)
- sendFromTo(self,com,string) - реализация команд MAIL FROM и RCPT TO (указание получателей и отправителя)
- bccCcRcpt(self,string), def sendUsers(self,rcptTo) - разбор строк адресов получателей и BCC, CC адресов. Если адрес корректен, он добавляется в множество set.
- data(self,mailFrom, rcptTo, ccString, bccString, subject, message) - формирование текста сообщения с различными заголовками
- sendData(self,data) - реализация команды DATA
- sendMessage- отправление текста сообщения
- quit(self) - - реализация команды QUIT (выход)
- rset(self) - сброс текущей команды и обрыв соединения

Файл view.py.
Файл определяет взаимодействие приложения с клиентом через консоль. В нем создается объект класса smtpClient. После чего в бесконечном цикле считывается команда клиента и выполняются необходимые операции, если команда корректна. Иначе, выводится сообщение о некорректности команд. Корректные команды:

- connect- установление соединения
- quit- разрыв соединения
- send- отправка сообщения
- exit-выход из программы

Файлы mock_sock.py, test.py
Файлы тестируют приложение. Их описание представлено ниже (в подглаве “Тестирование”).

Дизайн приложения

При запуске приложения клиент видит на экране следующее сообщение:

Введите команду: connect- установление соединения
quit- разрыв соединения
send- отправка сообщения
exit-выход из программы

При вводе команды connect клиента просят указать название сервера, свой логин и пароль. При удачно выполненных операциях соединение считается установленным. Иначе - не установленным.

При вводе команды send анализируется соединение. Если оно не установлено, клиента оповещают о невозможности отправить сообщение без установления соединения с сервером. Если соединение установлено клиент поочередно должен ввести ответы на следующие сообщения:

Введите адрес отправителя
Введите адрес получателей через запятую (,)
Введите bcc адреса через запятую (,)

Введите cc адреса через запятую (,)
Введите тему сообщения
Введите сообщение

После чего происходит попытка отправить сообщение с указанными параметрами и выводится сообщение об удачной или неудачной попытке отправки сообщения по указанным адресам.

При вводе команды quit также анализируется соединение. Если соединение установлено, оно сбрасывается, и выводится сообщение об удачной операции quit. Если соединение не установлено, выводится сообщение о невозможности разрыва неустановленного соединения.

При вводе команды exit происходит выход из программы.

Тестирование

За тестирование отвечают файлы mock_sock.py, test.py.
Файл mock_sock.py
Файл необходим для написания unit тестов. Имеется 2 класса

- MockSocketTrue - класс, переопределеный от класса socket (пакета socket). Переопределяет операции connect, send, recv.
  Всегда возвращает верные коды ответов сервера в зависимости от отправляемой команды. Таким образом, можно проверить работу методов при верных ответах от сервера.
- MockSocketFalse - класс, переопределеный от класса socket (пакета socket). Переопределяет операции connect, send, recv.
  Всегда возвращает код ответа 500, что является ошибочным кодом. Таким образом, можно проверить работу методов при ошибочных кодах ответа сервера.

Файл test.py
Файл имеет несколько классов, классы ориентированы на проверку определенных команд.

Unit тесты. Создается объект класса MockSocketTrue(MockSocketFalse) и проверяется работа объекта SmtpClient при верных(или неверных) кодах ответа сервера. При этом не происходит непосредственное подключение к серверу, переопределенные сокеты являются оболочкой.
Классы:

- TestMockSocketFalse
- TestMockSocketTrue

Интеграционные тесты. Создаются объекты классов socket (пакета socket) и SmtpClient. Происходит непосредственное подключение к серверу, после чего, задавая определенные параметры входных значений, проверяется работа отдельных методов класса SmtpClient.
Классы:

- TestSmtpConnect - тестирует команды установления соединения (методы connect, helo)
- TestAuth - тестирует команды аутентификации и выхода (методы setAuth, quit)
- TestSend - тестирует команды установления адресов отправителя и получателей, отправления сообщения и сброс текущей команды с разрывом соединения (методы sendFromTo, sendMessage, rset)

С помощью бибилиотеки coverage было проанализировано тестовое покрытие. Анализируются только файлы, содержащие тестируемые и вспомогательные функции - test.py и mock_sock.py. Общее тестовое покрытие составляет 98% (почти 100%), это говорит о прохождении почти всех строк приложения, а значит проверке

различных возникающих ситуаций.
Файл view.py отвечает за взаимодейтсвие приложения с клиентом, он не содежит отдельных методов, которые можно было бы проверить с помощью интеграционных методов. Поэтому для решения этой задачи использовалось ручное тестирование, с его помощью была проверена не только функциональность методов, но и их взаимодействие между собой.

Сервер протокола FTP, функционирующий в активном режиме

Индивидуальное задание

Задание: разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее функции FTP-сервера.

Основные возможности. Приложение должно реализовывать следующие функции:

1. Хранение идентификационной и аутентификационной информации нескольких пользователей
2. Поддержка анонимного входа (пользователь anonymous)
3. Обработка подключения клиента
4. Выдача по запросу клиента содержимого каталога
5. Навигация по системе каталогов
6. Создание нового каталога
7. Удаление каталога
8. Посылка по запросу клиента содержимого указанного файла
9. Приём по запросу клиента содержимого указанного файла
10. Удаление указанного файла
11. Протоколирование соединения сервера с клиентом

Поддерживаемые команды. Разработанное приложение должно реализовывать следующие команды протокола FTP:

- USER – получение от клиента идентификационной информации пользователя
- PASS – получение от клиента пароля пользователя
- LIST – отправка клиенту расширенной информации о списке файлов каталога
- NLST – отправка клиенту сокращённой информации о списке фай- лов каталога
- CWD – смена текущего каталога сервера
- MKD – создание каталога
- RMD – удаление каталога
- DELE – удаление файла на сервере
- PORT – получение параметров сокета клиента (адреса и порта), осуществляющего приём и передачу данных
- RETR – посылка файла клиенту
- STOR – запись полученного от клиента файла
- TYPE – задание режима передачи данных
- QUIT – удаление всех помеченных сообщений и завершение сеанса

Настройки приложения. Разработанное приложение должно обеспечивать:

1. настройку номера порта сервера (по умолчанию – 21)

2. настройку корневого каталога сервера для каждого пользователя

Методика тестирования. Для тестирования приложения следует использовать стандартные FTP-клиенты, установленные в лаборатории (Mozilla Firefox, MS Explorer, Far, Total Commander). С помощью имеющихся клиентов протокола FTP осуществляется подключение к серверу с различной аутентификационной информацией. В процессе тестирования проверяются основные возможности сервера по передаче, приёму, удалению файлов, навигации по файловой системе, функции по работе с каталогами.

Теоритические сведения протокола FTP

Предоставленная информация соответсвует [RFC 959 [Postel and Reynolds 1985]-протокол FTP.] (http://www.soslan.ru/tcp/tcp27.html)

FTP отличается от других приложений тем, что он использует два TCP соединения для передачи файла.

1. Управляющее соединение устанавливается как обычное соединение клиент-сервер. Сервер осуществляет пассивное открытие на заранее известный порт FTP (21) и ожидает запроса на соединение от клиента. Клиент осуществляет активное открытие на TCP порт 21, чтобы установить управляющее соединение. Управляющее соединение существует все время, пока клиент общается с сервером. Это соединение используется для передачи команд от клиента к серверу и для передачи откликов от сервера. Тип IP сервиса для управляющего соединения устанавливается для получения “минимальной задержки”, так как команды обычно вводятся пользователем.
2. Соединение данных открывается каждый раз, когда осуществляется передача файла между клиентом и сервером. Тип сервиса IP для соединения данных должен быть “максимальная пропускная способность”, так как это соединение используется для передачи файлов.

Команды и отклики FTP

Команды и отклики передаются по управляющему соединению между клиентом и сервером в формате NVT ASCII. В конце каждой строки команды или отклика присутствует пара CR, LF.
Отклики состоят из 3-циферных значений в формате ASCII, и необязательных сообщений, которые следуют за числами. Подобное представление откликов объясняется тем, что программному обеспечению необходимо посмотреть только цифровые значения, чтобы понять, что ответил процесс, а дополнительную строку может прочитать человек. Поэтому пользователю достаточно просто прочитать сообщение (причем нет необходимости запоминать все цифровые коды откликов). Каждая из трех цифр в коде отклика имеет собственный смысл. Ниже показаны значения первых и вторых цифр в коде отклика. Третья цифра дает дополнительное объяснение сообщению об ошибке.

- 1yz Положительный предварительный отклик. Действие началось, однако необходимо дождаться еще одного отклика перед отправкой следующей команды.
- 2yz Положительный отклик о завершении. Может быть отправлена новая команда.
- 3yz Положительный промежуточный отклик. Команда принята, однако необходимо отправить еще одну команду.
- 4yz Временный отрицательный отклик о завершении. Требуемое действие не произошло, однако ошибка временная, поэтому команду необходимо повторить

позже.
- 5yz Постоянный отрицательный отклик о завершении. Команда не была воспринята и повторять ее не стоит.
- x0z Синтаксическая ошибка.
- x1z Информация.
- x2z Соединения. Отклики имеют отношение либо к управляющему, либо к соединению данных.
- x3z Аутентификация и бюджет. Отклик имеет отношение к логированию или командам, связанным с бюджетом.
- x4z Не определено.
- x5z Состояние файловой системы.

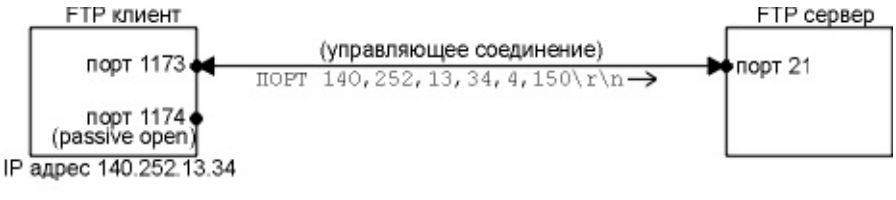
Управление соединением

Использовать соединение данных можно тремя способами.

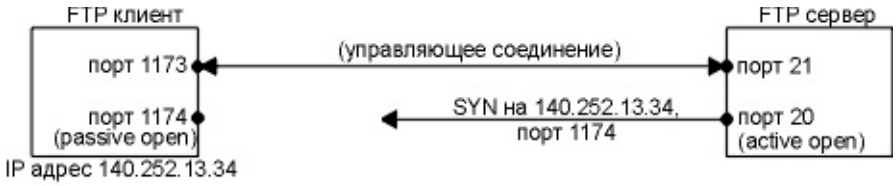
- Отправка файлов от клиента к серверу.
- Отправка файлов от сервера к клиенту.
- Отправка списка файлов или директорий от сервера к клиенту.

FTP сервер посылает список файлов по соединению данных, вместо того чтобы посылать многострочные отклики по управляющему соединению. При этом появляется возможность избежать любых ограничений в строках, накладывающихся на размер списка директории. Конец файла обозначает закрытие соединения данных. Из этого следует, что для передачи каждого файла или списка директории требуется новое соединение данных. Обычная процедура FTP в активном режиме выглядит следующим образом:

1. Создание соединения данных осуществляется клиентом, потому что именно клиент выдает команды, которые требуют передать данные (получить файл, передать файл или список директории).
2. Клиент обычно выбирает динамически назначаемый номер порта на хосте клиента для своего конца соединения данных. Клиент осуществляет пассивное открытие с этого порта.
3. Клиент посылает этот номер порта на сервер по управляющему соединению с использованием команды PORT.
4. Сервер принимает номер порта с управляющего соединения и осуществляет активное открытие на этот порт хоста клиента. Сервер всегда использует порт 20 для соединения данных.



где,
passive open - пассивное открытие



где,
passive open - пассивное открытие
active open - активное открытие

На первом рисунке показано состояние соединений, пока осуществляется шаг номер 3. Мы предполагаем, что динамически назначаемый порт клиента для управляющего соединения имеет номер 1173, а динамически назначаемый порт клиента для соединения данных имеет номер 1174. Команда, посылаемая клиентом - PORT, а ее аргументы это шесть десятичных цифр в формате ASCII, разделенные запятыми. Четыре первых числа - это IP адрес клиента, на который сервер должен осуществить активное открытие (140.252.13.34 в данном примере), а следующие два - это 16-битный номер порта. Так как 16-битный номер порта формируется из двух цифр, его значение в этом примере будет 4 x 256 + 150 = 1174.
На втором рисунке показано состояние соединений, когда сервер осуществляет активное открытие на конец клиента соединения данных. Конечная точка сервера это порт 20.

Сервер всегда осуществляет активное открытие соединения данных. Обычно сервер также осуществляет активное закрытие соединения данных, за исключением тех случаев, когда клиент отправляет файл на сервер в потоковом режиме, который требует, чтобы клиент закрыл соединение (что делается с помощью уведомления сервера о конце файла).

Анонимный FTP

Существует невероятно популярная форма использования FTP. Она называется анонимный FTP (anonymous FTP). Если эта форма поддерживается сервером, она позволяет любому получить доступ к серверу и использовать FTP для передачи файлов. С помощью анонимного FTP можно получить доступ к огромному объему свободно распространяемой информации.
Чтобы использовать анонимный FTP, мы входим в систему с именем пользователя “anonymous”. Когда появляется приглашение ввести пароль, мы вводим наш адрес электронной почты.

Архитектура приложения

Команды клиента и ответы сервера

В представленном приложении реализованы не все команды FTP-сервера.
В процессе написания приложения было решено, что некоторые команды выполняют функции, которые для нашего FTP-сервера не столь важны, поэтому они не были реализованы. Команды, не реализованные в написанном FTP-сервере следующие:

- ABOR Отменяет выполнение предыдущей команды
- ADDM Добавляет элемент в физический файл
- ADDV Добавляет элемент в элемент переменной длины физического файла
- APPE Добавляет данные в указанный файл
- CRTL Создает библиотеку
- CRTP Создает физический файл
- CRTS Создает исходный физический файл
- DBUG Запускает или завершает трассировку сервера
- DLTL Удаляет библиотеку
- HELP Выдает информацию о командах сервера FTP
- MODE Задает формат передачи данных
- NOOP Проверяет, отвечает ли сервер
- RCMD Отправляет команду CL на сервер FTP
- REIN Перезапускает сеанс на сервере
- RNFR Задает файл, который должен быть переименован
- RNTO Задает новое имя файла
- SITE Отправляет на сервер необходимую информацию
- STAT Получает от сервера информацию о состоянии
- STOU Сохраняет данные на сервере без замены существующего файла
- STRU Задает структуру файла
- SYST Выводит имя операционной системы сервера
- TIME Задает значение тайм-аута для сервера FTP

Реализованные команды с кодами ответов сервера представлены в таблице ниже.

Команда	Собщение положительного ответа сервера	Код положительного ответа сервера
QUIT	Goodbye	221
TYPE [binary(ascii)]	Type set to I(A)	200
XMKD [name]	Folder [name] create	257
XRMD [name]	Directory [name] deleted	250
DELE[name]	File [name] deleted	250
AUTH [name]	Write the password	331
[password]	User [name] logged in	230
CWD	Current directory is [ ]	250

CWDUP	Current directory is [..]	250
PORT [ ]	PORT command succesful	200
LIST, NLST, RETR [name], STOR [name]	Accepted data connection	150
	[data]	
	Transfer complete	226

Формат файлов

user.txt
Сервер обращается к файлу ./user.txt для получения множества пар (login, пароль) зарегистрированных клиентов. Каждая строка файла отвечает за одного клиента. Строка состоит из логина и пароля клиента, разделенных запятой “,”.
Формат строки - *login,password*.

protocol.txt
Работа сервера протоколируется в файл ./protocol.txt. Пример файла:

*id_логин_команда_**количество клиентов***
0.578431_подключение___1
0.578431Nikitina[‘PORT’, ‘127,0,0,1,195,207’]1
0.578431Nikitina[‘LIST’]1
0.578431Nikitina[‘TYPE’, ‘A’]1
0.859663**_подключение___**2
0.859663anonymous[‘QUIT’]2
0.859663anonymousотключение1
0.578431Nikitina[‘QUIT’]1
***0.578431Nikitinaотключение**___0*

Первая строка - заголовки столбцов, далее в файл дописываются строки при любой активности подключенных клиентов.

- id - уникальный идентификатор клиента
- логин -логин клиента
- команда - команда, отправленная клиентом; подключение/отключение от сервера
- количество клиентов - количество клиентов, подключенных к серверу на данный момент.

Классы и методы

Класс User содержит основную информацию о каждом подключенном клиенте. Поля:

- dir - текущая директория. Первоначально ””
- user_id - уникальный идентификатор
- type - тип соединения(“binary”/“ascii”). Первоначально “binary”

- name - login
- password - пароль

Метод authUser(self,login,password) присваивает указанные name и password.

Класс Server содержит информацию о всех подключенных клиентах, отвечает за чтение файлов, работу с ОС. Поля:

- users - зарегистрированные клиенты. Множество состоит из пар login-пароль, считывается из файла ./user.txt.
- activeUsers - множество подключенных клиентов (объектов класса User)
- directory - корневая директория сервера

Методы:

- readFileUsers(self) - считывание файла ./user.txt и заполнение множества users парами login-пароль.
- hasUser(self,login,password) - определение наличия подключаемого клиента среди зарегистрированных. Если такой клиент есть или его login - anonymous возвращает True. Иначе - False.
- deleteFile(self,pathFile,userDir) - попытка удаления файла. Если адрес указан верен, файл удаляется, возвращает True. Иначе - False.
- deleteDir(self,pathFile,userDir) - попытка удаления папки со всем ее содержимым. Если адрес указан верен, папка удаляется, возвращает True. Иначе - False.
- mkDir(self,pathFile,userDir) - попытка создания новой папки. True/False.
- setPath(self,pathFile,userDir) - получение полного имени файла/папки .
- deleteUser(self,user_id) - удаление клиента из множества активных клиентов (при его отключении).
- auth(self,login,password,user_id) - присваивание определенному клиенту указанных login и пароль.
- newUser(self,user_id) - добавление клиента к множеству активных клиентов (при его подключении).
- setType(self,user_id,type) - изменение режима передачи (“binary”/“ascii”).
- serverList(self,flag,userDir) - получение списка входящих в указанную директорию папок и файлов.
- serverNlst(self,path,i) - получение расширенного списка входящих в указанную директорию папок и файлов (указывается дата создания, дата последнего изменения, размер файла/папки).
- serverSetDir(self,dir,userDir) - определение наличия указанной папки.
- writeProtocolFile(self,command,User) - запись в файл протоклирования новой строки, соответсвующей указанным команде и клиенту.

Класс SocketThread(threading.Thread). Объект этого класса создается на новое подключение клиента к серверу (реализуется многопоточность).
Поля:

- conn - IP-адрес
- addr - порт
- port -объект типа Boolean. True - открыт канал передачи данных, False - закрыт.
- id - уникальный идентификатор.

Методы:

- run (self) - основной метод. Создается новый объект класса User с указанными параметрами(login и пароль), уникальный идентификатор user_id генерируется случайным образом. Далее в бесконечном цикле считывается команда от клиента и выполняется неоходимая операция. После отключения клиента его сокет закрывается.
- setType(self,type) - реализует команду TYPE (измение типа соединения)
- mkDir(self,path) - реализует команду XMKD (создание директории)
- deleteDir(self,path)- реализует команду XRMD (удаление директории)
- delete(self,path)- реализует команду DELE (удаление файла)
- list(self,flag):- реализует команду LIST (получение списка директории)
- comPort(self,path)- реализует команду PORT(открытие соединения данных)
- cwdUp(self)- реализует команду CWD .. (переход в директорию выше)
- cwd(self,dir)- реализует команду CWD(переход в указанную директорию)
- pwd(self)- реализует команду XPWD (определение текущей директории)
- auth(self,login) -присваивают созданному объекту User указанных login и пароля.
- comRetr(self,file) - реализует команду RETR. В зависимости от типа соединения вызывается метод retrFileAscii или retrFileBinary.
- retrFileBinary(self,path) - отправление файла при типе соединения “binary”.
- retrFileAscii(self,path) - отправление файла при типе соединения “ascii”.
- comStor(self,file) - реализует команду STOR. В зависимости от типа соединения вызывается метод storeFileBinary или storeFileAscii.
- storeFileBinary(self,path) - получение файла при типе соединения “binary”.
- storeFileAscii(self,path)- получение файла при типе соединения “ascii”.

Основная нить. Многопоточность
Возможно подключение и работа сервера с несколькими клиентами. Сервер в главной нити создает слушающий сокет. При подключении каждого нового клиента создаются новый сокет и новая нить SocketThread,в которой сервер общается с клиентом согласно протоколу. После отключения клиента от сервера, нить SocketThread и сокет, соответсвующие этому клиенту закрываются. Таким образом, работа клиентов проходят параллельно и действия одного клиента никак не влияют на другого клиента.

Тестирование

Тестирование сервера осуществляется с помощью консоли операционой системы Windows. Подключимся к серверу командой *ftp localhost*, укажем логин, пароль клиента. Далее работа с сервером осуществляется, как с обычным FTP-сервером.

Отправляем поддерживаемые сервером команды и проверяем результат работы сервера, заметим, что все команды выполняются корректно.
Для тестирования механизма многопоточности, подключимся к серверу с различных консолей. Видим, что сервер со всеми клиентами работает в раздельном режиме, то есть команды, отправляемые одним клиентом, никак не влияют на работу другого клиента.