

Сервер протокола FTP, функционирующий в пассивном 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 файлов каталога
- CWD – смена текущего каталога сервера
- MKD – создание каталога
- RMD – удаление каталога
- DELE – удаление файла на сервере
- PASV – перевод сервера в пассивный режим
- 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.

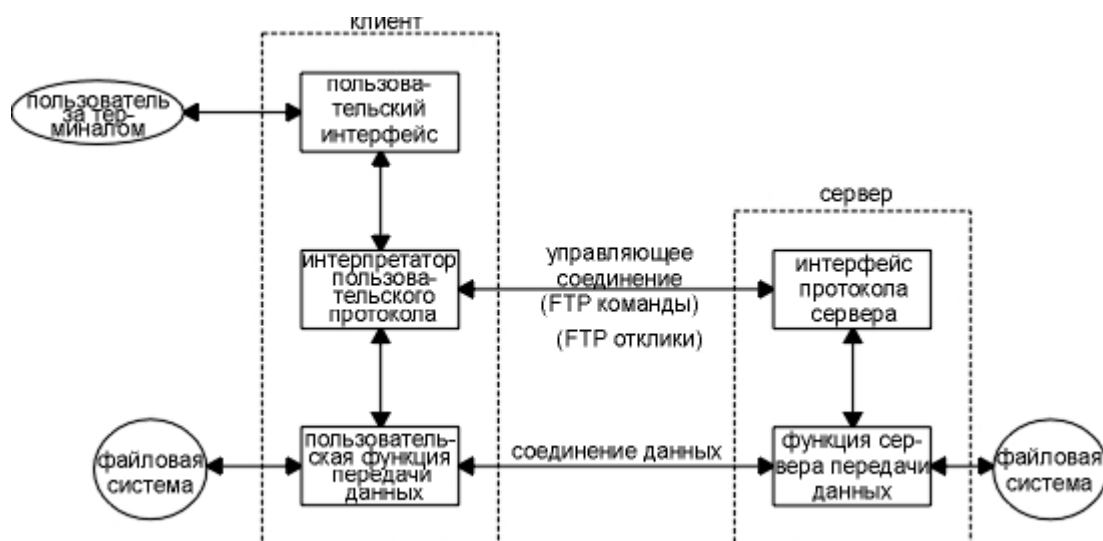
Введение

FTP это широко используемое приложение. Оно является стандартом Internet для передачи файлов. Необходимо различать передачу файлов, именно то, что предоставляет FTP, и доступ к файлам, что предоставляется такими приложениями как NFS (Network File System). Передача файлов заключается в копировании целого файла из одной системы в другую. Чтобы использовать FTP, необходимо иметь открытый бюджет на сервере, или можно воспользоваться так называемым анонимным FTP (anonymous FTP).

Протокол FTP

FTP 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 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. Команды состоят из 3 или 4 байт, а именно из заглавных ASCII символов, некоторые с необязательными аргументами. Клиент может отправить серверу более чем 30 различных FTP команд.

Реализованные в приложении команды приведены в описании задания, ниже приведем другие довольно часто встречающиеся команды:

- ABOR — Прервать передачу файла
- CDUP — Сменить директорию на вышестоящую.
- EPSV — Войти в расширенный пассивный режим. Применяется вместо PASV.
- HELP — Выводит список команд, принимаемых сервером.
- MDTM — Возвращает время модификации файла.
- NOOP — Пустая операция.
- PWD — Возвращает текущую директорию.
- REIN — Реинициализировать подключение.
- RNFR и RNT0 — Переименовать файл. RNFR — что переименовывать, RNT0 — во что.
- SIZE — Возвращает размер файла.
- SYST — Возвращает тип системы (UNIX, WIN, ...).

FTP отклики

Отклики состоят из 3-цифрных 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).

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.

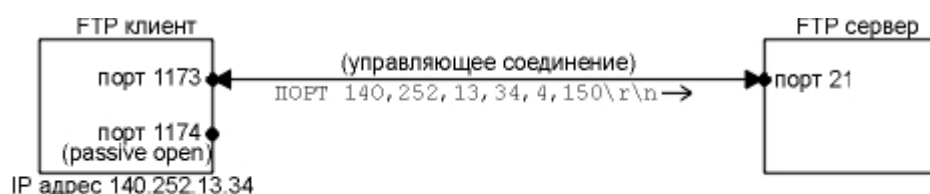
Управление соединением

Использовать соединение данных можно тремя способами.

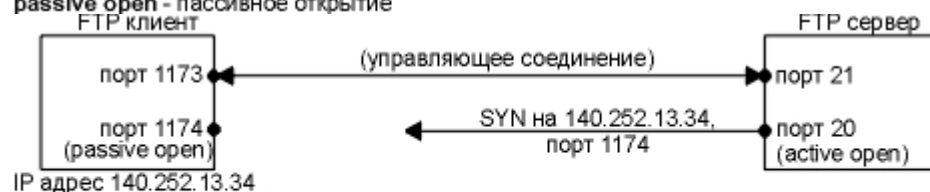
- Отправка файлов от клиента к серверу.
- Отправка файлов от сервера к клиенту.
- Отправка списка файлов или директорий от сервера к клиенту.

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 $4 \times 256 + 150 = 1174$.

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.

Информация взята из RFC 959 <https://tools.ietf.org/html/rfc959> (перевод на русский язык <http://www.soslan.ru/tcp/tcp27.html>)

Архитектура приложения

Реализуемые команды

- 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
- PASV – перевод сервера в пассивный режим
- RETR – посылка файла клиенту
- STOR – запись полученного от клиента файла
- TYPE – задание режима передачи данных
- QUIT – удаление всех помеченных сообщений и завершение сеанса
- SYST – Возвращает тип системы (UNIX, WIN, ...).
- FEAT – Запрашивает информацию о поддерживаемых расширениях FTP.

- CDUP – Сменить директорию на вышестоящую.
- PWD – Возвращает текущую директорию.

Классы и методы

В данном приложении реализовано 2 класса: FTPserver и FTPserverThread. Класс FTPserver является главным, он в свою очередь запускает экземпляр класса FTPserverThread, который и предоставляет всю функциональность FTP сервера.

Класс FTPserver

Методы:

- run(self) - основной метод, который создает новый экземпляр класса FTPserverThread.
- stop(self) - закрывает открытый сокет.

Класс FTPserverThread

Методы:

- run(self) - основной метод. В нем в начале загружается список пользователей, затем отсылается приветственное сообщение клиенту и осуществляется обработка приходящих команд.
- load_users(self) - загрузка списка пользователей и их паролей
- check_user(self,username) - проверка пользователя
- check_pass(self,password) - проверка пароля
- start_datasock(self) - открытие соединения для передачи данных
- stop_datasock(self) - закрытие соединения для передачи данных
- toListItem(self,fn) - преобразование имени файла в стандартный вид для выдачи по команде LIST

Далее следуют функции, обрабатывающие согласно протоколу одноименные команды от клиента:

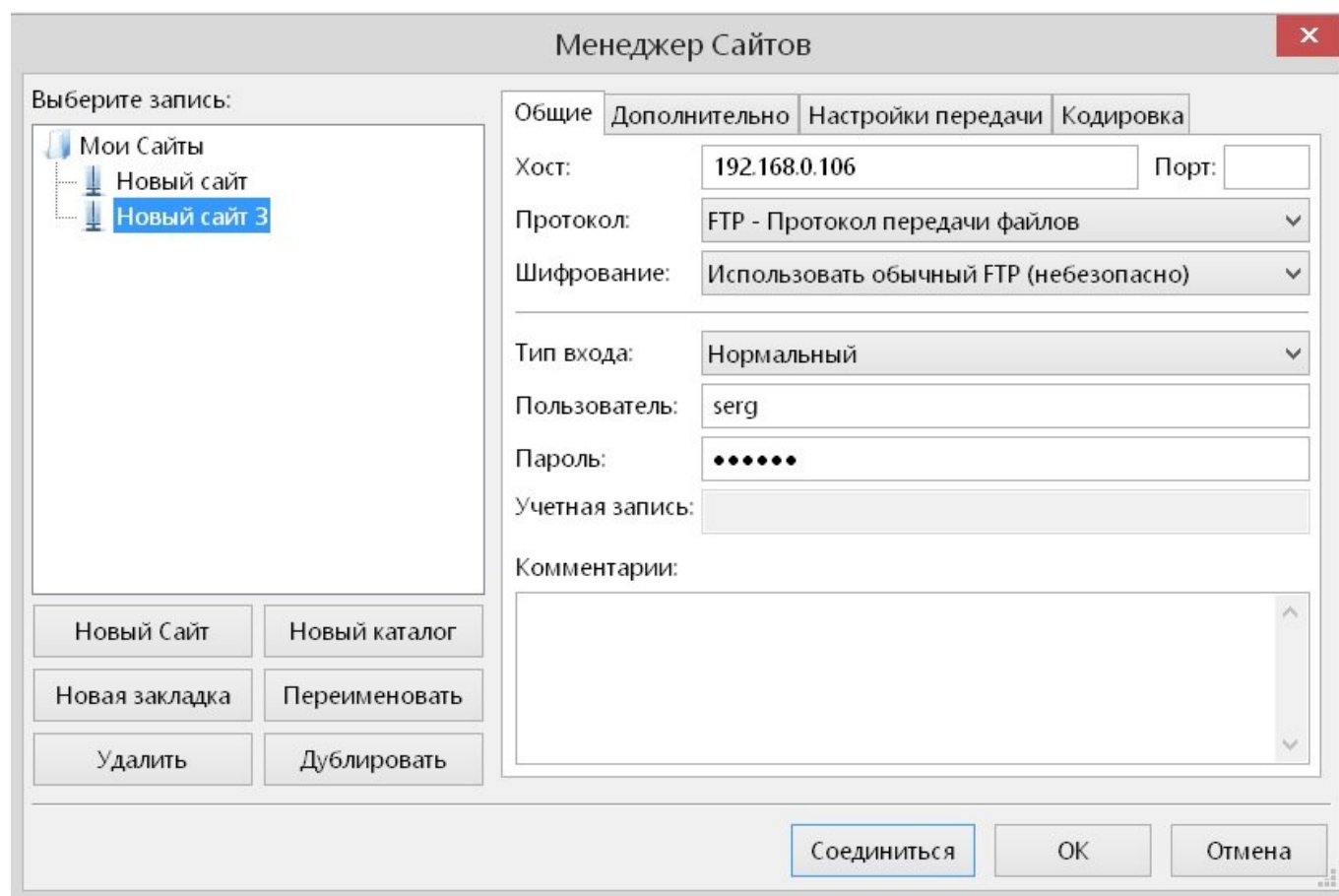
- SYST(self,cmd):
- USER(self,cmd):
- FEAT(self,cmd):
- PASS(self,cmd):
- QUIT(self,cmd):
- TYPE(self,cmd):
- CDUP(self,cmd):
- PWD(self,cmd):
- CWD(self,cmd):
- PASV(self,cmd):
- LIST(self,cmd):
- NLST(self,cmd):

- MKD(self,cmd):
- RMD(self,cmd):
- DELE(self,cmd):
- RETR(self,cmd):
- STOR(self,cmd):

Данное приложение осуществляет возможность подключения нескольких клиентов к серверу FTP. Это реализуется за счет многопоточности: для каждого клиента создается свой поток и открывается свой порт. Таким образом команды от различных клиентов не перемешиваются. Взаимодействие осуществляется согласно протоколу.

Тестирование

Работа FTP сервера была протестирована с помощью FTP-клиента FileZilla. Натстройки соединения:



Менеджер Сайтов

Выберите запись:

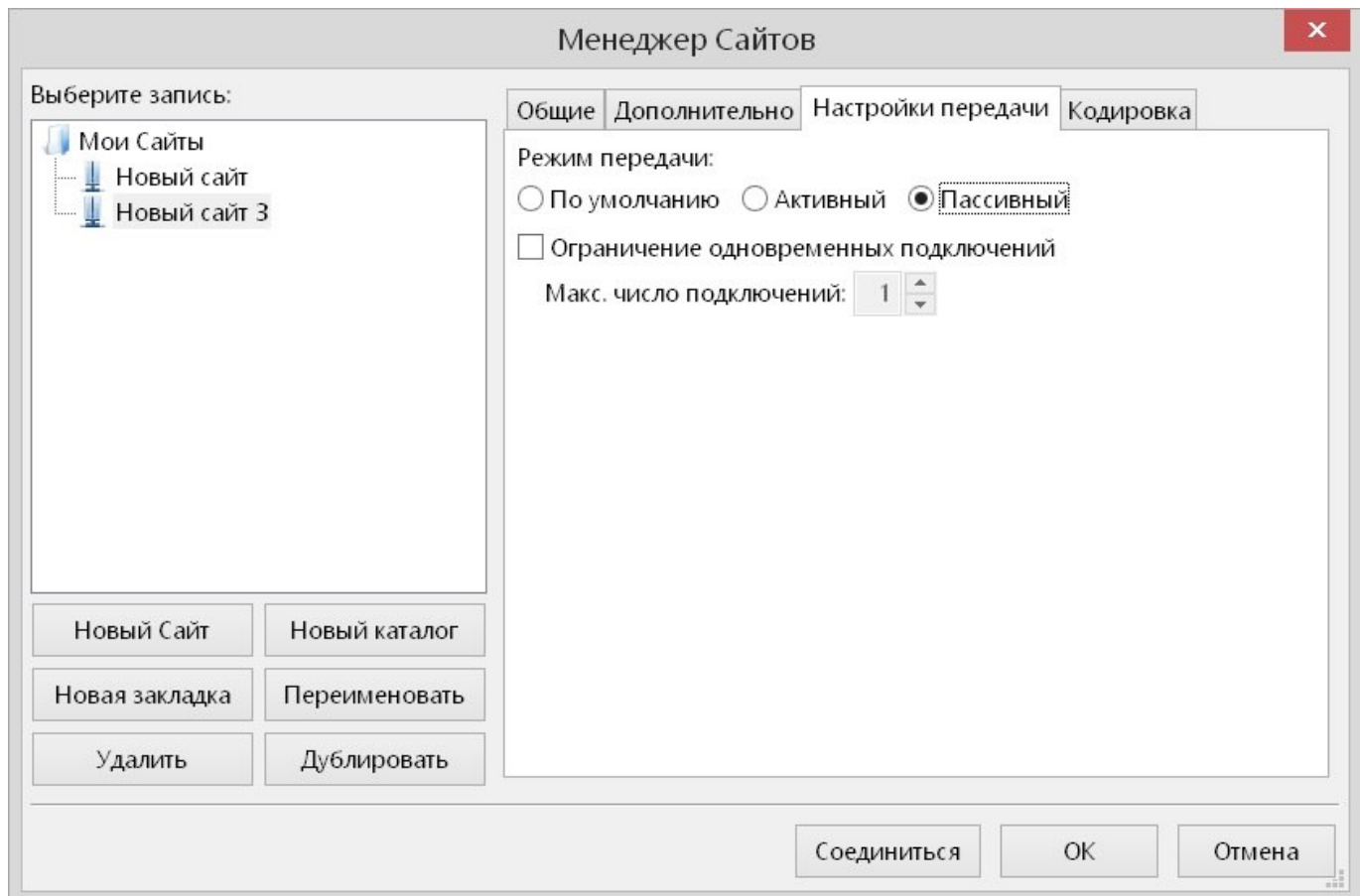
- Мои Сайты
 - Новый сайт
 - Новый сайт 3

Новый Сайт Новый каталог
 Новая закладка Переименовать
 Удалить Дублировать

Общие Дополнительно Настройки передачи Кодировка

Хост: 192.168.0.106 Порт:
 Протокол: FTP - Протокол передачи файлов
 Шифрование: Использовать обычный FTP (небезопасно)
 Тип входа: Нормальный
 Пользователь: serg
 Пароль:
 Учетная запись:
 Комментарии:

Соединиться ОК Отмена



Пример взаимодействия клиента и сервера:

```
Статус: Соединяюсь с 192.168.0.106:21...
Статус: Соединение установлено, ожидание приглашения...
Ответ: 220 Welcome!
Команда: USER anonymous
Ответ: 331 OK.
Команда: PASS *****
Ответ: 230 OK.
Команда: SYST
Ответ: 215 UNIX Type: L8
Команда: FEAT
Ответ: 211 OK.
Статус: Сервер не поддерживает символы не ASCII.
Статус: Соединение установлено
Статус: Получение списка каталогов...
Команда: PWD
Ответ: 257 "D:/common_linux/"
Команда: TYPE I
Ответ: 200 Binary mode.
Команда: PASV
Ответ: 227 Entering Passive Mode (192,168,0,106,211,46).
Команда: LIST
Ответ: 150 Here comes the directory listing.
Ответ: 226 Directory send OK.
Статус: Список каталогов "D:\common_linux" извлечен
```


Приложение №1. Код программы.

```
import os,socket,threading,time,sys, logging

allow_delete = False
local_ip = socket.gethostbyname(socket.gethostname())
local_port = 21
currdir='D:/common_linux/'

class FTPserverThread(threading.Thread):
    def __init__(self,conn,addr):
        self.conn=conn
        self.addr=addr
        self.basewd=currdir
        self.cwd=self.basewd
        self.rest=False
        threading.Thread.__init__(self)
        self.mode = ''
        self.users = {}
        self.user = ''
        logging.basicConfig(format = u'%(levelname)-8s [%s] %(message)s',
level = logging.DEBUG, filename = u'mylog.log')

    def run(self):
        self.load_usrs()
        self.conn.send(bytes('220 Welcome!\r\n', 'UTF-8'))
        while True:
            cmd=self.conn.recv(256)
            logging.info(cmd)
            if not cmd: break
            else:
                pr1 = cmd[:4].decode('UTF-8')
                pr2 = pr1.strip()
                pr = pr2.upper()
                func = getattr(self, pr)
                func(cmd)

    def load_usrs(self):
        f = open('users.cfg','r')
        for line in f:
            self.users[line.split(" ")[0]] = line.split(" ")[1]

    def check_user(self,usrnm):
        for i in self.users:
            if i == usrnm:
                self.user = usrnm
                return True
        return False

    def check_pass(self,paswd):
        if self.users[self.user] == paswd:
            return True
        else:
            return False
```

```

def SYST(self,cmd):
    self.conn.send(bytes('215 UNIX Type: L8\r\n', 'UTF-8'))
def USER(self,cmd):
    if cmd.decode('UTF-8').split(" ")[1].split("\r\n")[0] == 'anonymous':
        self.user = 'anonymous'
        self.conn.send(b'331 OK.\r\n')
    elif self.check_user(cmd.decode('UTF-8').split(" ")[1].split("\r\n")[0]):
        self.conn.send(b'331 OK.\r\n')
    else:
        self.conn.send(b'530 Login is incorrect\r\n')

def FEAT(self,cmd):
    self.conn.send(bytes('211 OK.\r\n', 'UTF-8'))
def PASS(self,cmd):
    if self.user == 'anonymous':
        self.conn.send(b'230 OK.\r\n')
    else:
        if self.check_pass(cmd.decode('UTF-8').split("
") [1].split("\r\n") [0]+'\\n'):
            self.conn.send(b'230 OK.\r\n')
        else:
            self.conn.send(b'530 Password is incorrect\r\n')

def QUIT(self,cmd):
    self.conn.send(bytes('221 Goodbye.\r\n', 'UTF-8'))
def TYPE(self,cmd):
    self.mode=cmd[5]
    self.conn.send(bytes('200 Binary mode.\r\n', 'UTF-8'))

def CDUP(self,cmd):
    if not os.path.samefile(self.cwd,self.basewd):
        self.cwd=os.path.abspath(os.path.join(self.cwd,'..'))
    self.conn.send(bytes('200 OK.\r\n', 'UTF-8'))
def PWD(self,cmd):
    cwd = 'D:/common_linux/'
    cwd = self.cwd
    self.conn.send(bytes('257 \"%s\"\\r\\n' % cwd, 'UTF-8'))
def CWD(self,cmd):
    chwd=cmd[4:-2]
    print('__%s__' % chwd)
    nf = chwd.decode('UTF-8')
    print('new: %s' % nf)
    chwd = nf
    if chwd=='D:/':
        self.cwd=self.basewd
    elif chwd[0]=='/':
        self.cwd=os.path.join(self.basewd,chwd[1:])
    else:
        self.cwd=os.path.join(self.cwd,nf)
    self.conn.send(bytes('250 OK.\r\n', 'UTF-8'))

def PASV(self,cmd):
    self.servsock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    self.servsock.bind((local_ip,0))
    self.servsock.listen(1)
    ip, port = self.servsock.getsockname()
    print ('open', ip, port)

```

```

        self.conn.send(bytes('227 Entering Passive Mode (%s,%u,%u).\r\n' %
                               ('.'.join(ip.split('.')), port>>8&0xFF, port&0xFF), 'UTF-8'))

def start_datasock(self):
    self.datasock, addr = self.servsock.accept()
    print ('connect:', addr)

def stop_datasock(self):
    self.datasock.close()
    self.servsock.close()

def LIST(self,cmd):
    self.conn.send(bytes('150 Here comes the directory listing.\r\n','UTF-8'))
    print ('list:', self.cwd)
    self.start_datasock()
    for t in os.listdir(self.cwd):
        k=self.toListItem(os.path.join(self.cwd,t))
        self.datasock.send(bytes(k+'\r\n','UTF-8'))
    self.stop_datasock()
    self.conn.send(bytes('226 Directory send OK.\r\n','UTF-8'))

def NLST(self,cmd):
    self.LIST(cmd)

def toListItem(self,fn):
    st=os.stat(fn)
    fullmode='rwxrwxrwx'
    mode=''
    for i in range(9):
        mode+=((st.st_mode>>(8-i))&1) and fullmode[i] or '-'
    d=(os.path.isdir(fn)) and 'd' or '-'
    ftime=time.strftime(' %b %d %H:%M ', time.gmtime(st.st_mtime))
    return d+mode+' 1 user group '+str(st.st_size)+ftime+os.path.basename(fn)

def MKD(self,cmd):
    dn=os.path.join(self.cwd,cmd[4:-2])
    os.mkdir(dn)
    self.conn.send('257 Directory created.\r\n')

def RMD(self,cmd):
    dn=os.path.join(self.cwd,cmd[4:-2])
    if allow_delete:
        os.rmdir(dn)
        self.conn.send('250 Directory deleted.\r\n')
    else:
        self.conn.send('450 Not allowed.\r\n')

def DELE(self,cmd):
    fn=os.path.join(self.cwd,cmd[5:-2])
    if allow_delete:
        os.remove(fn)
        self.conn.send('250 File deleted.\r\n')
    else:
        self.conn.send('450 Not allowed.\r\n')

def RETR(self,cmd):
    fn=os.path.join(self.cwd,cmd[5:-2].decode('UTF-8'))

```

```

        print ('Downlowding:',fn)
        if self.mode=='I':
            fi=open(fn,'rb',encoding="UTF-8")
        else:
            fi=open(fn,'rb')
        self.conn.send(bytes('150 Opening data connection.\r\n','UTF-8'))
        if self.rest:
            fi.seek(self.pos)
            self.rest=False
        try:
            data= fi.read(1024)
        except Exception as e:
            print('Error: ',e)
        self.start_datasock()
        while data:
            self.datasock.send(data)
            data=fi.read(1024)
        fi.close()
        self.stop_datasock()
        self.conn.send(bytes('226 Transfer complete.\r\n','UTF-8'))

def STOR(self,cmd):
    fn=os.path.join(self.cwd,cmd[5:-2].decode())
    print ('Uplaoding:',fn)
    if self.mode=='I':
        fo=open(fn,'wb')
    else:
        fo=open(fn,'wb')
    self.conn.send(bytes('150 Opening data connection.\r\n','UTF-8'))
    self.start_datasock()
    while True:
        data=self.datasock.recv(1024)
        if not data: break
        fo.write(data)
    fo.close()
    self.stop_datasock()
    self.conn.send(bytes('226 Transfer complete.\r\n','UTF-8'))

class FTPserver(threading.Thread):
    def __init__(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.bind((local_ip,local_port))
        threading.Thread.__init__(self)

    def run(self):
        self.sock.listen(5)
        while True:
            fd, addr = self.sock.accept()
            th=FTPserverThread(fd,addr)
            th.start()

    def stop(self):
        self.sock.close()

if __name__=='__main__':
    ftp=FTPserver()
    ftp.start()

```

```
print ('On', local_ip, ':', local_port)
input('Enter to end...\n')
ftp.stop()
```