УДК 004.416.6

Логачев М.А.
студент
4 курс, факультет «Институт прикладной информатики, математики и физики»
ФГБОУ ВО «Армавирский государственный педагогический университет»
Научный руководитель: Карабут Н.В.
Старший преподаватель
Logachev M.A.
student
4 course, faculty " Institute of Applied Informatics, Mathematics and Physics"
FGBOU VO "Armavir State Pedagogical University"
Russia, Armavir
Scientific adviser: Karabut N.V.
Senior lecturer

РАЗРАБОТКА FTP ПРОТОКОЛА СРЕДСТВАМИ ЯЗЫКА PYTHON

DEVELOPMENT FTP PROTOCOL BY MEANS OF PYTHON LANGUAGE

Аннотация

В настоящее время скорость интернет соединения повсеместно растет, что неуклонно сказывается на постепенном отказе от подключения нескольких компьютеров физическими проводами, на смену этого подхода приходят программы, работающие по сети, не требующих физического подключения нескольких машин проводами.

Annotation

At present, the speed of Internet connection is growing everywhere, which steadily affects the gradual refusal to connect several computers with physical wires, this approach is replaced by programs running on the network that do not require physical connection of several machines with wires.

Ключевые слова

Программа, сервер, компьютер, клиент-серверная архитектура, Питон, сокет.

Keyword

Ftp-protocol, ftp-client, ftp-server, klient-server architecture, Python, socket.

FTP-протокол

File transfer protocol (Протокол передачи файлов) был создан в начале семидесятых годов. В то время жестко стоял вопрос о разработке технологии передачи файлов с одного компьютера на другой. С момента создания и по сей день, ftp-протокол неоднократно изменялся в лучшую сторону.

По необходимости, программа, созданная на основе ftp-протокола может предоставлять пользователю простой доступ к удаленному серверу, передавая на сервер команды с помощью текстовой консоли, беря на себя только работу по пересылке команд, введенных пользователем и запрашиваемых файлов. Во втором сценарии использования программа-клиент отображает файлы на удаленном сервере так, будто бы они являются частью файловой системы пользователя.

На сегодняшний день эти сценарии очень размыты, современные фтп-программы дают практически неограниченный доступ к компьютеру, который выступает в роли сервера.

В самом простейшем случае фтп-программа выступает как эмулятор файловой системы удаленной машины, с этой файловой системой можно совершать все привычные пользователю функции: копировать и удалять файлы из сервера, создавать файлы, переименовывать.

Разработка серверной части

Сначала создается класс сервера, который выступит каркасом серверной части программы.

Для собственных нужд клиент-серверная логика фтп-протокола была немного изменена - после подключения клиентов к серверу, именно сервер получает доступ над клиентскими машинами.

```
class FileServer:
    def __init__(self, ip, port):
        # server socket for command
        self.sock = socket(AF_INET, SOCK_STREAM)
        self.sock.bind(('', port))
        self.sock.listen(1)

        # serversocket for download files
        self.sockd = socket(AF_INET, SOCK_STREAM)
        self.sockd.bind(('', 6667))
        self.sockd.listen(1)
```

В методе __init__ происходит инициализация класса, сервер принимает айпи адрес и порт, на котором будет прослушивать входящие соединения. Создается объект socket, параметрами которого является тип айпи, в нашем случае это AF_INET, который представляет собой айпи вида ipv4, вторым параметром является тип соединения, константа SOCK_STREAM подходит для большинства случаев использования. Метод bind связывает сокет с выбранным айпи и портом. После

вызывается метод listen, параметром которого является число максимально допустимых соединений.

В __init__ методе нашего серверного класса инициализируется два сокет объекта, по одному соединению будут передаваться текстовые команды, а по второму будут передаваться файлы по необходимости.

Основной сложностью в разработке серверной части является алгоритм трансляции команд пользователя в системные команды, сложность заключается в том, чтобы используя привычные linux команды, язык программирования Python возвращал корректный ответ.

Реализовано это с помощью использования объекта, который транслирует привычные linux команды в строки на языке python.

```
cmdtable = {
   'linux': {'nochoise': {'ls': 'os.listdir()',
                          'pwd': 'os.getcwd()',
                          'exit': "sys.exit()",
                          '../': "os.chdir('../')"},
             'choise': {'cd': "os.chdir(r'{}')",
                        'fullpath': "os.path.abspath(r'{}')",
                        'download': "load {}"}
             }

}
```

Таблица отражений представляет собой объект, первым элементом которого является дочерний объект с названием linux, что означает, что команды переводятся с команд linux в язык python. Данный метод хорош тем, что впоследствии стоит только добавить объект windows, в котором будет описана трансляция команд с windows в язык Python.

Эту таблицу использует метод parsecmd.

```
def parsecmd(self):
```

```
cmd = input('\nGive next cmd - ').strip()
output = None
while not output:
    if self.linux['nochoise'].get(cmd, 0):
        output = self.linux['nochoise'][cmd]
    elif self.linux['choise'].get(cmd, 0):
        path = input('.. path - ')
        output = self.linux['choise'][cmd].format(path.strip())
return output
```

Он получает от пользователя команду с помощью вызова input(), далее команда переводится на язык python. В последствии эта команда отправляется на клиентскую часть.

Основная часть по организации логики заключена в методе run().

```
def run(self):
    print('Сервер начинает работу', os.getpid())
    while True:
        newsock, addr = self.sock.accept()
        _thread.start_new_thread(self.serverdownload, ())
        print('Have new connection from,', addr)
        while True:
            cmd = self.parsecmd()

            newsock.send(cmd.encode())
            data = newsock.recv(1024)
            try:
                pprint(json.loads(data.decode()))
            except json.decoder.JSONDecodeError:
                print(data.decode())
```

Этот метод делит программу фактически на два потока с помощью вызова thread.start_new_thread(). В первом потоке программа обменивается текстовыми командами, во втором случае файлами.

В метод создания нового потока передается функция serverdownload(), которая будет отвечать за передачу файлов.

```
def serverdownload(self):
    while True:
        downloadsocker, addr = self.sockd.accept()
        namefile, filesize = downloadsocker.recv(1024).decode().split()

        with open(namefile, 'wb') as file:
            while True:
                bytes = downloadsocker.recv(1024)
                if not bytes:
                    break
                file.write(bytes)
        print('\nDownload success - '
              'was downloaded {:.2%} mb'.format(float(filesize)))
```

Разработка клиентской части

По сравнению с серверной версией, клиентская часть является более легковесным решением относительно количества кода. В обязанности клиентской части входит получение команд от сервера и формирование ответа.

Для клиентской версии так же создается класс Client с методом __init__ в котором есть отличия от его серверной версии.

```
def __init__(self, ip, port):
    self.sock = socket(AF_INET, SOCK_STREAM)
    self.sock.connect(('localhost', 6666))
```

Метод получает на вход айпи сервера и порт для связи, однако вместо ожидания новых подключений, клиент производит подключение с помощью метода connect().

Как и у класса сервера, у клиента есть метод run() организующий всю алгоритмическую работу.

```
def run(self):
    while True:
        cmd = self.sock.recv(1024)
        if 'load' in cmd.decode():
            _thread.start_new_thread(self.download,
                                     (cmd.decode().split()[-1],))
            self.sock.send(b'\ndownload started')
        else:
            try:
                action = eval(cmd.decode())
                if action is not None:
                    self.sock.send(json.dumps(action).encode())
                else:
                    self.sock.send(b'No feedback')
            except Exception:
                self.sock.send(b'Smth went wrong!')
                traceback.print_exc()
                self.run()
```

В этом методе присланная сервером команда декодируется из байтового представления с помощью метода decode(). Данная команда представляет собой строку на языке Python, благодаря тому, что этот язык программирования транслируемый, существует возможность выполнять исполнять различный код непосредственно во время работы программы. За это отвечает функция eval(), она принимает на вход корректный по

синтаксису Python-код и выполняет его. Таким образом мы можем исполнять команды на клиенте, полученные от сервера.

Библиографический список

1. Буйначев С.К. Основы программирования на языке Python [Электронный ресурс] : учебное пособие / С.К. Буйначев, Н.Ю. Боклаг. — Электрон. текстовые данные. — Екатеринбург: Уральский федеральный университет, ЭБС АСВ, 2014. — 92 с.

2. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.

3. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

4. Сузи Р.А. Язык программирования Python [Электронный ресурс] / Р.А. Сузи. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 350 с.