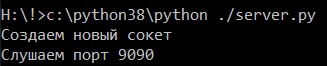
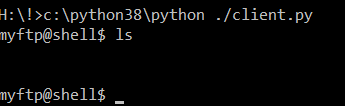
Пример работы с программой. Запускаем сервер:



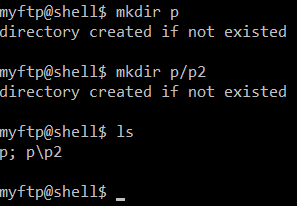
Запускаем клиент:



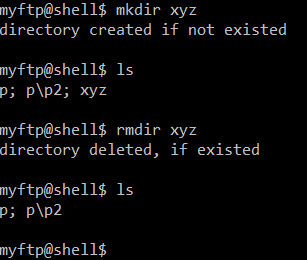
Покажем список файлов. Пока он пуст.



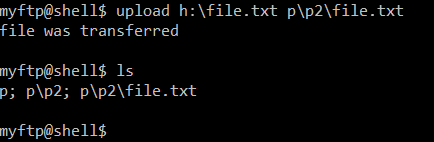
Создадим каталоги, и посмотрим снова:



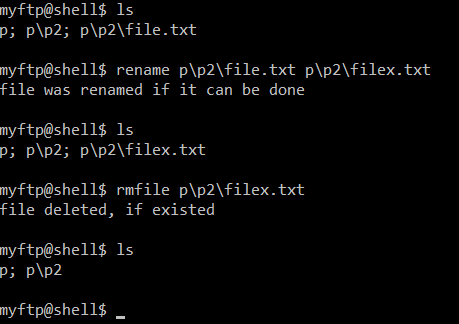
Создадим каталог, посмотрим на его наличие, а потом удалим:



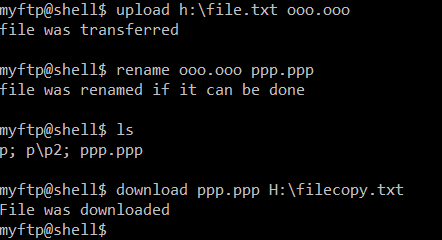
Закачаем файл на сервер:



Переименуем этот файл, а затем удалим:



Закачаем файл на сервер, потом переименуем его, и скачаем обратно, под другим именем:



Как работает клиент. Прежде всего, он устанавливает соединение на порту 9090:

# Устанавливаем соединение

HOST = 'localhost'

PORT = 9090

while True:

# Подключаемся к серверу

sock = socket.socket()

sock.connect((HOST, PORT))

Далее у пользователя просят ввести команду:

# Получаем команду

request = input('myftp@shell$ ')

Почти все команды можно просто передать на сервер. Но три команды особые. Во-первых, это команда закачки (upload). Проверим, что это она:

# Если это закачка

if request.startswith("upload "):

Если это закачка, то во-первых, нужно скрыть исходное имя файла (серверу не обязательно знать, где файл лежал изначально), а во-вторых, передать размер, и, собственно, данные файла. Поэтому сначала получим параметры команды:

# Получаем исходное имя

ref = request.split(" ")[1]

# Получаем имя на сервере

willbe = request.split(" ")[2]

Читаем данные файла:

# Читаем данные

myfile = open(ref, "r")

data = myfile.read()

myfile.close()

Получаем длину этих данных, и передаем на сервер: команду, имя файла, размер данных, и данные. Все разделяем пробелами.

# Получаем их длину

datasize = len(data)

# Отправляем на сервер имя файла, длину, и данные

request = "upload " + willbe + " " + str(datasize) + " " + data

Если мы скачиваем файл с сервера, то серверу не обязательно знать, куда мы потом положим этот файл. Получим параметры команды, но на сервер передадим лишь один. Второй сохраним в переменной xwillbe.

# Если это скачка

if request.startswith("download "):

# То серверу не нужно знать, куда мы сохраним файл

xref = request.split(" ")[1]

xwillbe = request.split(" ")[2]

# Передадим ему лишь имя желаемого файла

request = "download " + xref

Передаем запрос на сервер:

# Отправляем запрос

sock.send(request.encode())

Если команда была exit, выйдем из программы. Мы уже передали ее серверу, так что он отсоединит нас, но нам и самим нужно выйти:

# Если запрос - exit, выходим из клиента

if request == "exit":

break;

Получаем ответ сервера. Обратим внимание, только 8192 байта. Соответственно, если мы попытаемся передавать файлы длиннее, это не сработает. Но для нашего простого случая можно этим пренебречь:

# Получаем ответ

response = sock.recv(8192).decode()

Если ответа не пришло, напишем об этом:

# Если ответа нет

if not response:

print("No data was received")

else:

Если ответ пришел, и мы скачивали файл, то получим размер файла, и прочитаем его содержимое:

# Если мы скачивали

if request.startswith("download "):

# Получаем размер ответа

filesize = int(response.split(' ')[0])

# Получаем содержимое файла

sodfile = response[-filesize-1:]

Теперь сохраним его на файловой системе (имя файла сохранено в переменной xwillbe):

# Сохраняем его

myfile = open(xwillbe, "w")

myfile.write(sodfile)

myfile.close()

response = "File was downloaded"

Теперь осталось лишь напечатать ответ:

# Печатаем ответ

print(response)

Если же мы вышли из программы, то перед окончанием работы, закроем сокет:

# Закрываем сокет

sock.close()

Как работает сервер. Прежде всего, он устанавливает соединение на порту 9090:

PORT = 9090

os.chdir("DATA")

while True:

print("Создаем новый сокет")

sock = socket.socket()

sock.bind(('' , PORT))

sock.listen()

Далее мы принимаем входящие соединения, и определяем запросы:

while True:

print('Слушаем порт', PORT)

conn, addr = sock.accept()

print(addr)

request = conn.recv(1024).decode()

print(request)

Если это команда exit, просто выходим:

if request == "exit":

break;

В противном случае нужно обработать запрос (функцией process), и добавить один символ (перевод строки). Если этого не сделать, то, например, при выполнении команды ls, если нет ни одного файла, по соединению не будет передаваться ни одного байта, и клиент просто повиснет, ожидая хоть какого-нибудь ответа. С другой стороны, можно было бы при отсутствии какого-либо ответа отдавать, например, строку «No answer».

response = process(request) + "\n"

Далее нужно отправить результат обработки команды

conn.send(response.encode())

Если была команда exit (мы вышли из цикла while), то закроем сокет:

sock.close()

Разберем, как работает обработка команд – функция process. Если команда pwd – просто вернем текущий каталог:

def process(req):

if req == 'pwd':

return os.getcwd()

Если команда ls, то можно, конечно, воспользоваться os.listdir, как в примере, однако таким образом мы не увидим подкаталоги. Поэтому лучше воспользуемся рекурсивной функцией glob.iglob, соберем все результаты в список, и выведем на экран:

elif req == 'ls':

files = []

for filename in glob.iglob('\*\*', recursive=True):

files = files + [filename]

return '; '.join(files)

Все остальные команды требуют параметров, поэтому разделим команду по символу пробела, и проверим, два ли их:

else:

reqs = req.split(" ")

if len(reqs) != 2:

Если их НЕ два – это может быть команда rename (3 параметра – rename, имя1, имя2) или upload (тогда нам вообще передается целый файл, и там может быть все, что угодно, но меньше четырех параметров точно быть не может – upload, размер, имя файла, данные, в которых тоже могут встретиться пробелы). Поэтому проверяем, что их не меньше двух.

if len(reqs) < 2:

return 'bad request'

Если параметров не два, и не меньше двух, то очевидно, их больше двух (хотя бы три). Если это команда rename, переименовываем файл. При этом убираем любые две подряд идущие точки, чтобы пользователь не мог выйти за пределы нашего каталога.

elif reqs[0] == 'rename':

os.rename(reqs[1].replace('..', ''), reqs[2].replace('..', ''))

return 'file was renamed if it can be done'

Если это команда upload, то убираем все последовательности из двух точек в имени файла, а затем получаем размер файла – он будет третьим параметром, и берем данные с конца строки нужного размера:

elif reqs[0] == 'upload':

filename = reqs[1].replace('..', '')

filesize = int(reqs[2])

sodfile = req[-filesize-1:]

Записываем эти данные в указанный файл:

myfile = open(filename, "w")

myfile.write(sodfile)

myfile.close()

return 'file was transferred'

Если параметров было не менее трех, но команда не upload и не rename, ошибка:

else:

return 'bad request'

Если команда mkdir, создаем каталог:

elif reqs[0] == 'mkdir':

os.mkdir(reqs[1].replace('..', ''))

return 'directory created if not existed'

Если команда rmdir, удаляем каталог:

elif reqs[0] == 'rmdir':

os.rmdir(reqs[1].replace('..', ''))

return 'directory deleted, if existed'

Если команда rmfile, удаляем файл:

elif reqs[0] == 'rmfile':

os.remove(reqs[1].replace('..', ''))

return 'file deleted, if existed'

Если команда download, то читаем данные файла в переменную data:

elif reqs[0] == 'download':

ref = req.split(" ")[1].replace('..', '')

myfile = open(ref, "r")

data = myfile.read()

myfile.close()

Затем вычисляем длину данных, и передаем длину вместе с данными:

datasize = len(data)

request = str(datasize) + " " + data

return request

В противном случае (неизвестная команда) выдаем ошибку:

else:

return 'bad request'

Как можно улучшить данную программу:

- Мы читаем из сокета всегда 8192 байта. Этого может не хватить, если файл достаточно длинный. Правильнее было бы делить файл на части, и передавать по частям.

- Мы удаляем все вхождения символов .. из имен файлов, чтобы нельзя было написать, например, «файл ../../../ss.ss», выйдя из нашего каталога. Однако мы не обрабатываем, например, попытки доступа к файлу напрямую – H:\file.txt. В ОС Unix есть функция os.chroot, которая блокирует программу в одном, конкретно заданном каталоге, однако она не работает на ОС Windows. Поэтому в данном случае мы использовали просто os.chdir.