## FTP сервер

### Цель работы

Научиться программно работать с файлами и файловой системой, читать, создавать, перемещать и передавать файлы по сети

### Задания для выполнения

Создать сервер, который предоставляет клиенту базовые возможности файлового менеджера по сети. Клиент после подключения к серверу должен иметь возможности просматривать список файлов и папок в рабочей директории сервера (рабочая директория - это специальная папка, к которой имеет доступ процесс сервера, но она отделена от парки с кодом сервера и от любых системных файлов), создавать и удалять в ней папки, создавать, копировать и переименовывать файлы. Также клиент может передать на сервер название и содержимое файла и сервер должен создать соответствующий файл в текущей директории. Кроме того, клиент может запросить содержимое любого файла и сервер должен передать его в ответ.

### Методические указания

Для выполнения этой работы вам пригодится код многопоточного сервера, который вы создавали на предыдущих работах. Также следует воспользоваться средствами стандартной библиотеки Python, а именно модулями: os, shutils, subprocess.

При проектировании сервера необходимо заранее определить формат запросов, которые может совершать пользователь. Так как он может выполнять разные действия, нужно предусмотреть, как это указывается в запросе. Фактически, вам нужно придумать набор команд. Причем, многие такие команды требуют передачи дополнительной информации. Рекомендуется использовать следующий набор команд:

1. Посмотреть содержимое папки;
2. Создать папку;
3. Удалить папку;
4. Удалить файл;
5. Переименовать файл;
6. Скопировать файл с клиента на сервер;
7. Скопировать файл с сервера на клиент;
8. Выход (отключение клиента от сервера);

Для примера вы можете использовать формат соответствующих команд из интерпретатора bash или придумать собственные названия.

Для начала работы необходимо создать просто сервер. Парадигма клиент-серверного взаимодействия подразумевает, что обычно клиент, присоединяясь к серверу, посылает ему сообщение-запрос. Сервер обрабатывает этот запрос и посылает ответ клиенту. После этого соединение закрывается. Это самая простая и распространенная схема. Мы будем использовать именно ее.

PORT = 9090

sock = socket.socket()

sock.bind(('', PORT))

sock.listen()

while True:

print("Слушаем порт", PORT)

conn, addr = sock.accept()

print(addr)

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

print(request)

response = process(request)

conn.send(response.encode())

sock.close()

Обратите внимание, что мы отделяем сетевую часть нашего приложения от логики обработки запроса. Это позволяет более просто развивать и модифицировать наше приложение.

Задание лабораторной заключается в написании сервера. Однако, для целей отладки напишем простой клиент. Он будет в вечном цикле читать команду со стандартного входа, направлять ее серверу, печатать ответ:

HOST = 'localhost'

PORT = 9090

while True:

sock = socket.socket()

sock.connect((HOST, PORT))

request = input('myftp@shell$ ')

sock.send(request.encode())

response = sock.recv(1024).decode()

print(response)

sock.close()

Конечно, вы можете воспользоваться наработками предыдущих лабораторных работ и использовать динамическое назначение порта, логирование в файл, выход по команде, авторизацию и многие другие возможности. Часть из них приведена в дополнительных заданиях.

Теперь мы можем написать функцию обработки запроса. Реализуем две самые простые функции нашего сервера:

def process(req):

if req == 'pwd':

return os.getcwd()

elif req == 'ls':

return '; '.join(os.listdir())

else:

return 'bad request'

Остальные функции попробуйте реализовать самостоятельно.

### Дополнительные задания:

1. Ограничьте возможности пользователя рамками одной определенной директории. Внутри нее он может делать все, что хочет: создавать и удалять любые файлы и папки. Нужно проследить, чтобы пользователь не мог совершить никаких действий вне пределов этой директории. Пользователь, в идеале, вообще не должен догадываться, что за пределами этой директории что-то есть.
2. Добавьте логирование всех действий сервера в файл. Можете использовать разные файлы для разных действий, например: подключения, авторизации, операции с файлами.
3. Добавьте возможность авторизации пользователя на сервере.
4. Добавьте возможность регистрации новых пользователей на сервере. При регистрации для пользователя создается новая рабочая папка (проще всего для ее имени использовать логин пользователя) и сфера деятельности этого пользователя ограничивается этой папкой.
5. Реализуете квотирование дискового пространства для каждого пользователя.
6. Реализуйте учётную запись администратора сервера.
7. Напишите отладочный клиент. Клиент должен подключаться к серверу и в автоматическом режиме тестировать корректность его работы. Используйте подход, аналогичный написанию модульных тестов. Клиент должен вывести предупреждающее сообщение, если сервер работает некорректно.