1. Напишите программу, которая создает нить. Родительская и вновь созданная нити должны распечатать десять строк текста.

Что происходит:

main вызывает print\_2 в отдельном потоке и сам вызывает print\_1 в себе

print\_2 вызывает print\_1 в отдельном потоке и сам вызывает print\_1 в себе

получается так называемая нить потоков

"""

import threading

*def* print\_1(*t\_name*: str):

    for v in range(10):

        print(*f*"[Поток {*t\_name*}] {v}")

*def* print\_2():

    t = threading.Thread(*target*=print\_1, *args*=("Thread 3",))

    t.start()

    print\_1("Thread 2")

*def* main():

    t = threading.Thread(*target*=print\_2())

    t.start()

    print\_1("MAIN 1")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Напишите простой эхо-сервер, использующий неблокирующие сокеты и клиент к нему.

Server.py

import socket

*def* main():

    # Создаем сокет

    server\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    # Назначаем на localhost

    server\_sock.bind(("", 9095))

    # Устанавлитваем кол-во соединений

    server\_sock.listen(1)

    # Ждем клиента

    conn, addr = server\_sock.accept()

    # Делаем сокет не блокируемым

    conn.setblocking(False)

    while True:

        try:

            # Пытаемся получить данные, если они не готовы - ждём

            data = conn.recv(1024)

            break

        except socket.error:

            # Ждем

            print("Ожидаем..")

    # Декодируем, переворачиваем ответ и кодируем заново (или можно просто вернутьзапрос клиента)

    conn.send(data.decode()[::-1].encode())

    # Закрываем соединение

    conn.close()

    # Закрываем сокет

    server\_sock.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Client.py

import socket

*def* main():

    # Создаём сокет

    sock = socket.socket()

    # Подключаемся к localhost

    sock.connect(("", 9095))

    # Вводим данные для отправки на сервер

    data\_input = input("Введите какие-либо данные -> ")

    # Отправляем данные

    sock.send(data\_input.encode())

    # Получаем ответ

    data = sock.recv(1024)

    # Выводим на экран

    print(*f*"Полученный ответ от сервера: {data.decode()}")

    # Закрываем сокет

    sock.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Напишите простой многопоточный загрузчик URL. Список URL скрипт принимает как аргументы командной строки.

import sys

import threading

import requests

# Словарь с ассоциацией между url и данными

responses\_dict = {}

*def* loader(*url*: str):

    """Метод для загрузки ответов по URL"""

    # Получаем данные по URL

    r = requests.get(*url*)

    # Добавляем данные в словарь

    responses\_dict[*url*] = r.text

*def* main():

    if sys.argv == 1:

        print(

            "Необходимо передать URL в аргументах при запуске скрипта (например, python3 task.py https://ya.ru"

        )

        return

    # Получаем список URL

    links\_list = sys.argv

    # Удаляем 1-й параметр т.к. это название скрипта

    links\_list.pop(0)

    # Для каждой ссылки запускаем отдельный поток

    for link in links\_list:

        t = threading.Thread(*target*=loader, *args*=(link,))

        t.start()

    # Ждем, пока все потоки отработают

    while threading.active\_count() > 1:

        pass

    for key, value in responses\_dict.items():

        print(*f*"URL: {key}\nОтвет:\n{value}\n")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Реализуйте простой HTTP-клиент. Он принимает один параметр командной строки - URL. Клиент делает запрос по указанному URL и выдает тело ответа на терминал как текст.

--

Еще проще сделать через requests, но здесь, кажется, нужно через urllib (или вообще сокеты?)

Можно также сделать через os.system(curl ...)

Кароч неоднозначное задание

"""

import sys

import urllib.request

*def* main():

    # Если аргумент в виде URL не был принят - вводим его

    if len(sys.argv) != 2:

        url\_input = input("Введите какой-либо URL -> ")

    else:

        url\_input = sys.argv[1]

    # Открываем соединение

    web\_request = urllib.request.urlopen(url\_input)

    # получаем код результата и выводим его

    print(*f*"Код результата: {web\_request.getcode()}")

    # Вывод данных, которые получили

    data = web\_request.read()

    print(data)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Напишите программу, которая вычисляет число Пи при помощи ряда Эйлера. Количество потоков программы должно определяться параметром командной строки.

Существует несколько вариантов

1 Вариант

import threading

# Начальная переменная для pi

pi = 0

*def* worker(*start*, *end*):

    global pi

    for num in range(*start*, *end*):

        if num % 2 == 1:

            pi += 1 / num \*\* 2

        else:

            pi -= 1 / num \*\* 2

*def* main():

    count = int(input("Введите кол-во потоков (1-100) -> "))

    # Рассчитываем, сколько элементов яда будет приходиться на 1 поток

    section\_length = 10000000 // count

    # Начало последнего запущенного потока

    start\_num = 1

    # Цикл по кол-ву потоков

    for i in range(count):

        t = threading.Thread(

*target*=worker,

*args*=(

                start\_num,

                start\_num + section\_length,

            ),

        )

        t.start()

        start\_num += section\_length

    # Ожидаем конца выполнения всех потоков

    while threading.active\_count() > 1:

        pass

    # См. тождество Фурье

    result = pow((pi \* 12), 0.5)

    print(*f*"Вычисленное значение Pi: {result}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

2 вариант

import threading

import time

pi = 1.0

*def* pi\_val(*i*, *iter*):

    global pi

    for num in range(*i* - *iter*, *iter*):

        if num % 2 == 1:

            pi += 1 / num \*\* 2

        else:

            pi -= 1 / num \*\* 2

*def* pi\_counter(*count*: int, *start\_time*: time):

    global pi

    # Интервалы

    inter = 100000 // *count*

    for i in range(2 + inter, 100000 + 1, inter):

        t = threading.Thread(

*target*=pi\_val,

*args*=(

                i,

                inter,

            ),

        ).start()

    pi = pow((pi \* 12), 0.5)

    time\_diff = time.time() - *start\_time*

    print(*f*"Вычисленное значение Pi: {pi}\nВремя вычисления: {time\_diff}")

*def* main():

    start\_time = time.time()

    pi\_counter(3, start\_time)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

3 вриант

Краткая форма

"""

import threading

pi = 1

*def* counter(*n*):

    global pi

    pi += 1 / (*n* \*\* 2)

*def* main():

    a = int(input("Введите кол-во потоков -> "))

    for i in range(a):

        threading.Thread(*target*=counter, *args*=(i + 2,)).start()

    print(pow((pi \* 6), 0.5))

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Дана функция calculate(x, y). Напишите программу, которая создает пул из 5 процессов и распределяет в этом пуле вычисление функции на промежутке х от 0 до 1 с шагом 0,1. у равняется 2 всегда.

import multiprocessing as mp

*def* calculate(*x*, *y*=2):

    """

    Функция calculate, умножает 2 числа

    На самом деле может делать что угодно, смысла в этом нет

    """

    return *x* \* *y*

*def* main():

    # Создаем пул процессов

    pool = mp.Pool(*processes*=5)

    # Создаем список аргументов x для функции calculate

    args\_list = [round(i \* 0.1, 2) for i in range(11)]

    # Получаем результат выполнения

    result\_list = list(pool.map(calculate, args\_list))

    # Выводим результаты на экран

    for arg, result in zip(args\_list, result\_list):

        print(*f*"x = {arg}, результат = {result}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Напишите программу, которая проверяет все числа от 0 на простоту и выводит простые числа на экран по мере нахождения. Числа должны проверяться в различных потоках (или процессах, по выбору студента) Программа должна работать до тех пор, пока ее не остановит пользователь.

import queue

import threading

# Наша очередь из чисел, которые надо обработать в worker

que = queue.Queue(10)

# Флаг остановки потоков

work\_flag = True

*def* worker(*thread\_num*):

    """

    Вычисление простого числа

    Вызывается в отдельном потоке

    :param thread\_num:

    :return:

    """

    while work\_flag:

        # Флаг того, что число простое

        is\_simple\_flag = True

        # Если очередь не пустая

        if not que.empty():

            # Число на проверку

            current\_number = int(que.get())

            for i in range(2, current\_number):

                # Если число не простое

                if current\_number % i == 0:

                    is\_simple\_flag = False

                    break

            if is\_simple\_flag:

                print(*f*"[Поток {*thread\_num*}] Число {current\_number} простое!")

*def* adder():

    """

    Метод для добавления числа в очередь

    Отрабатывает в отдельном потоке

    """

    current\_number = 0

    while work\_flag:

        que.put(current\_number)

        current\_number += 1

*def* main():

    # Да, это флаг остановки всех потоков и он глобальный, что ты мне сделаешь я в другмо городе

    global work\_flag

    t = threading.Thread(*target*=adder)

    t.start()

    # Запускаем 10 воркеров, которые будут осуществлять работу

    for i in range(10):

        t = threading.Thread(*target*=worker, *args*=[i], *name*=*f*"Поток {i}")

        t.start()

    # В главном потоке ожидаем ввод, как ввели что-то - завершаем прогу

    \_ = input()

    work\_flag = False

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Напишите программу, которая обходит все файлы в директории, переданной ей как параметр и выводит на экран имена тех, чей размер задан как второй параметр. Реализовать рекурсивный обход поддиректорий.

У решения еще была папка с файлами: с папкой и двумя текстовыми документами

import os

import sys

*class* Trip:

    """

    Класс-путешествие по внутреннему миру ФС

    """

*def* \_\_init\_\_(*self*, *root\_dir*: str, *bytes\_size*: int):

*self*.root\_dir = *root\_dir*

*self*.bytes\_size = *bytes\_size*

*self*.processing(*self*.root\_dir)

*def* processing(*self*, *current\_dir*: str):

        # Текущий путь

        current\_path = os.path.abspath(*current\_dir*)

        print(*f*"Перешли в путь {current\_path}")

        # Цикл по каждому файлу в текущем пути

        for item in os.listdir(current\_path):

            # Текущий путь до файла

            current\_item\_path = os.path.abspath(*f*"{*current\_dir*}{os.sep}{item}")

            # Если это файл и размер совпадает, то выводим его

            if (

                os.path.isfile(current\_item\_path)

                and os.path.getsize(current\_item\_path) == *self*.bytes\_size

            ):

                print(*f*"[ФАЙЛ] {current\_item\_path}")

            # Если директория

            elif os.path.isdir(current\_item\_path):

                # print(f"[ДИРЕКТОРИЯ] {current\_item\_path}")

                # Вызываем рекурсивно self.processing для этой директории

*self*.processing(current\_item\_path)

*def* main():

    if len(sys.argv) != 3:

        file\_path\_input = input("Введите путь -> ")

        file\_size\_input = int(input("Введите размер файла для поиска (в байтах) ->"))

    else:

        file\_path\_input = sys.argv[1]

        file\_size\_input = sys.argv[2]

    # file\_path\_input = "check"

    # file\_size\_input = 555

    trip = Trip(file\_path\_input, file\_size\_input)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Напишите программу, которая выводит на экран список номеров открытых портов на данной машине. Использовать команду netstat.

--

Можно использовать команду netstat с флагами -atun или -l на ваш выбор

Python в данном случае выступает просто в виде обёртки

"""

import subprocess

*def* main():

    # Дергаем команду netstat через subprocess

    result = subprocess.getoutput("netstat -atun")

    print(result)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Напишите программу, которая копирует файл с удаленного хоста в текущую папку по SSH. Имя файла и адрес хоста принимать как параметры.

---

На самом деле за копирование отвечает команда scp, использование команды такое:

scp -r root@127.0.0.1:/root .

Скрипт Python выступает просто в виде обёртки

"""

import os

import sys

*def* main():

    # Если при запуске не были указаны параметры:

    if len(sys.argv) != 3:

        # Ввод первой части

        auth\_and\_ip = input(

            "Введите имя пользователя и ip адрес (например, root@127.0.0.1)\n->"

        )

        # Ввод названия файла

        file\_path = input(

            "Введите путь до файла для копирования с удалённого хоста (например, /root/test.txt)\n->"

        )

    # Если были указаны параметры

    else:

        auth\_and\_ip = sys.argv[1]

        file\_path = sys.argv[2]

    # Формирование итоговой строки

    command\_result = *f*"scp -r {auth\_and\_ip}:{file\_path} ."

    # Вывод на экран

    print(*f*"Сформировали команду: '{command\_result}'")

    r = os.system(command\_result)

    # Если все норм

    if r == 0:

        print("Успешно скопировали файл")

    else:

        print("Что-то пошло не так при копировании файла")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Доп

11

#!/bin/python3

"""

Напишите программу, которая принимает как параметр имя исполняемого файла,

копирует его в директорию /usr/local/bin, регистриурует его в системе

как демона и запускает.

Предполагается использование systemd

Предположим, что скрипт имеет шебанг

"""

from shutil import copyfile

import sys

import os

# Необходимо запускать под sudo, пример запуска:

# sudo main.py myservice.service

*def* main():

    # Проверка на кол-во переданных аргументов

    if len(sys.argv) != 2:

        print("Введите имя исполняемого файла в виде аргумента!")

        return

    # Имя файла, которое принимаем

    file\_from = sys.argv[1]

    # Проверяем есть ли у файла право на запуск (x)

    is\_executable = os.access(file\_from, os.X\_OK)

    if not is\_executable:

        print("Переданный файл не является исполняемым!")

        return

    # Директория, куда копируем

    file\_to = *f*"/usr/local/bin/{file\_from}"

    # Само копирование

    copyfile(file\_from, file\_to)

    # Регистрация в качестве демона в системе и запуск

    os.system(*f*"systemctl start {file\_from}")

    os.system(*f*"systemctl enable {file\_from}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

12

"""

Модифицированный клиент/сервер (https://github.com/StudOborona/UnixSummer2021/tree/main/task2)

Когда сервер отсылает клиенту "Continue", то клиент повторяет запрос.

Если отсылает "Cancel", то клиент закрывает соединение

Тут реализация сервера

"""

# coding=utf-8

import socket

import random

# Словарь для рандомных ответов

random\_dict = {1: "Continue", 2: "Cancel"}

*def* main():

    # Создаем сокет

    server\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    # Назначаем на localhost

    server\_sock.bind(("127.0.0.1", 5000))

    # Устанавлитваем кол-во соединений

    server\_sock.listen(1)

    # Ждем клиента

    while True:

        conn, addr = server\_sock.accept()

        # Делаем сокет не блокируемым

        conn.setblocking(False)

        while True:

            try:

                # Пытаемся получить данные

                data = conn.recv(1024)

                break

            except socket.error:

                # Ждем

                print("Ожидаем..")

        message = data.decode()  # данные, которые приняли

        print(*f*"Получили сообщение от клиента: {message}")

        reply\_message = random\_dict[random.randint(1, 2)]

        conn.send(reply\_message.encode())

        print(*f*"Отправили ответное сообщение клиенту: {reply\_message}")

        # Закрываем соединение

        conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Тут реализация клиента

"""

# coding=utf-8

import socket

*def* main():

    cont\_flag = True

    while cont\_flag:

        # Создаём сокет

        sock = socket.socket()

        # Подключаемся к localhost

        sock.connect(("127.0.0.1", 5000))

        # Сообщение для сервера

        data\_input = "What must I do?"

        # Отправляем данные

        sock.send(data\_input.encode())

        # Получаем ответ

        data = sock.recv(1024)

        # Выводим на экран

        message = data.decode()

        print(*f*"Полученный ответ от сервера: {message}")

        # Если получили cancel, то завершаем работу

        if message == "Cancel":

            cont\_flag = False

        # Закрываем сокет

        sock.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

13

"""

Напишите TCP-клиент, подключающийся к веб-серверу google.com и сохраняющий на локальной машине код главной страницы этого сайта

"""

import socket

*def* main():

    host = "www.google.com"

    port = 80

    client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    client.connect((host, port))

    client.send('GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n'.encode())

    response = client.recv(4096)

    print(response)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

14

"""

Напишите программу, которая выводит файл /etc/passwd с удаленного хоста

"""

import os

import sys

*def* main():

    # Если при запуске не были указаны параметры:

    if len(sys.argv) != 2:

        # Ввод первой части

        auth\_and\_ip = input(

            "Введите имя пользователя и ip адрес (например, root@127.0.0.1)\n->"

        )

    # Если были указаны параметры

    else:

        auth\_and\_ip = sys.argv[1]

    # Формирование итоговой строки

    command\_result = *f*"ssh -p 22 {auth\_and\_ip} cat /etc/passwd"

    # Вывод на экран

    print(*f*"Сформировали команду: '{command\_result}'")

    r = os.system(command\_result)

    # Если все норм

    if r == 0:

        print("Успешно вывели файл")

    else:

        print("Что-то пошло не так")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

15

"""

Напишите программу, которая выводит на экран все названия процессов,

выполняющихся в данный момент от имени текущего пользователя

"""

import os

import sys

*def* main():

    command = "ps -u $(whoami) -o cmd"

    # Вывод на экран

    print(*f*"Сформировали команду: '{command}'")

    r = os.system(command)

    # Если все норм

    if r == 0:

        print("Успешно вывели названия процессов")

    else:

        print("Что-то пошло не так")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

16

Test.html дан

"""

Напишите программу, которая выводит на экран все ссылки, которые содержатся в скачанном HTML документе

"""

*def* main():

    with open('test.html', 'r', *encoding*='utf-8') as f:

        text = f.read()

        s\_index = 6 + text.find("href=\"")

        while s\_index != 5:

            length = text[s\_index:].find("\"")

            if length > 0:

                print(text[s\_index:s\_index+length])

            elif length == -1:

                break

            text = text[s\_index+length+2:]

            s\_index = 6 + text.find("href=\"")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

17

"""

Напишите программу, которая создает четыре нити

исполняющие одну и ту же функцию. Эта функция должна распечатать последовательность

текстовых строк, переданных как параметр. Каждая из созданных нитей

должна распечатать различные последовательности строк

"""

import sys

import threading

# Список для хранения информации о строках, которые выводятся

my\_list = []

*def* my\_printer(*data*: str):

    """Вывод строки на экран"""

    print(*data*)

*def* print\_1(*data*: str):

    """Вывод + вывод в отдельном потоке"""

    # Стартуем новый поток с новой строкой

    t = threading.Thread(*target*=my\_printer, *args*=(my\_list[3],))

    t.start()

    # Выводим строку в текущем потоке

    my\_printer(*data*)

*def* print\_2(*data*: str):

    """Вывод + вывод в отдельном потоке, в котором вывод + вывод в отдельном потоке"""

    # Стартуем новый поток с новой строкой

    t = threading.Thread(*target*=print\_1, *args*=(my\_list[2],))

    t.start()

    # Выводим строку в текущем потоке

    my\_printer(*data*)

*def* main():

    """

    Вывод + вывод в отдельном потоке, в котором вывод + вывод в отдельном потоке,

    в котором вывод + вывод в отдельном потоке

    """

    # Глобальный список строк-аргументов

    global my\_list

    # Если не переданы параметры

    if len(sys.argv) != 5:

        print("Необходимо передать в качестве параметров 4 строки")

        return

    # Список аргументов

    my\_list = list(sys.argv)

    # Первый удаляем т.к. это имя скирпта

    my\_list.pop(0)

    # Стартуем новый поток с новой строкой

    t = threading.Thread(*target*=print\_2, *args*=(my\_list[1],))

    t.start()

    # Выводим строку в текущем потоке

    my\_printer(my\_list[0])

if \_\_name\_\_ == "\_\_main\_\_":

    main()

18

"""

Напишите программу, которая копирует все файлы из домашней директории текущего пользователя в папку

переданную ей как параметр

"""

import os

import shutil

import sys

from os.path import expanduser

*def* main():

    # Проверяем на то, передали ли мы аргумент

    if len(sys.argv) != 2:

        print("Пожалуйста, передайте название папки в качестве параметра")

        return

    # Новая директория в переменной

    new\_dir = sys.argv[1]

    print(*f*"Директория, переданная как параметр: {new\_dir}")

    # Проверяем на существование директории, если что - создаем

    if os.path.isdir(new\_dir):

        print(*f*"Директория {new\_dir} существует")

    else:

        os.mkdir(new\_dir)

        print(*f*"Директории {new\_dir} не существует")

    # Получаем домашнюю директорию пользователя

    home = expanduser("~")

    print(*f*"Домашняя директория пользователя: {home}")

    # Цикл по каждому файлу в домашнем каталоге пользователя

    for current\_file in os.listdir(home):

        # Полный путь файла (для os.path)

        currentfile\_fullname = os.path.join(home, current\_file)

        # Если это файл - копируем его

        if os.path.isfile(currentfile\_fullname):

            # Само копирование

            shutil.copy(currentfile\_fullname, new\_dir)

            print(*f*"Скопировали файл {currentfile\_fullname} в {new\_dir}")

if \_\_name\_\_ == '\_\_main\_\_':

    main()

19

"""

Реализуйте простой HTTP-клиент. Он принимает один параметр командной строки - URL.

Клиент делает запрос по указанному URL и выдает тело ответа на терминал как текст. Если HTML то распечатываем без форматирования.

Вывод производится по мере того, как даннные поступают из HTTP соединения. Когда будет выведено более 25 строк, клиент должен продолжить прием данных, но должен остановить вывод и выдать приглашение "Press SPACE to scroll down".

"""

import sys

import urllib.request

from pynput import keyboard

*def* main():

    # Если аргумент в виде URL не был принят - вводим его

    if len(sys.argv) != 2:

        url\_input = input("Введите какой-либо URL -> ")

    else:

        url\_input = sys.argv[1]

    # Открываем соединение

    web\_request = urllib.request.urlopen(url\_input)

    print(type(web\_request))

    # Получаем код результата и выводим его

    print(*f*"Код результата: {web\_request.getcode()}")

    # Вывод данных, которые получили

    global i

    i = 0

    counter = 1

    mark = True

    while mark:

        while i < 25:

            if str(web\_request.readline()).strip() == "b''":

                print("----- END OF FILE -----")

                break

                return False

            else:

                line = web\_request.readline()

                print(str(counter) + " | " + str(line))

                counter += 1

                i += 1

        with keyboard.Events() as events:

            for event in events:

                if (event.key == keyboard.Key.space and str(type(event)) == "<class 'pynput.keyboard.Events.Press'>"):

                    i = 0

                    print("\n------------------\n\nВывод новых строк\n\n------------------")

                    break

                if event.key == keyboard.Key.esc:

                    return False

if \_\_name\_\_ == "\_\_main\_\_":

    main()

20

"""

Напишите программу, которая выводит список всех выполняющихся в текущий момент системных демонов.

Рядом с каждым вывести адрес его конфигурационного файла.

"""

import os

import subprocess

*def* main():

    # Команда для получения всех демонов, которые работают сейчас

    my\_command = "systemctl --type=service --state=active | awk '{print $1}'"

    # Создаем процесс с выполнением этой команды

    p = subprocess.Popen(my\_command, *stdout*=subprocess.PIPE, *shell*=True)

    # Получаем результат работы процесса

    out, err = p.communicate()

    # Формируем список демонов, которые присутствуют в системе

    daemon\_list = [d for d in out.decode().split("\n") if ".service" in d]

    # Для каждого демона выполняем команду systemctl show -p FragmentPath, которая выводит путь до его файла

    for d in daemon\_list:

        os.system(*f*"echo '\n{d}' && systemctl show -p FragmentPath {d} | sed 's/FragmentPath=//g'")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

21

Server.py

"""

Напишите TCP-сервер, принимающий от клиента имя файла и возвращающий содержание этого файла

Это сервер

"""

import os

import socket

from typing import Union

PORT = 8888

*class* FileProcessing:

    """Работа с файлами"""

*def* \_\_init\_\_(*self*) -> None:

*self*.sep = os.sep

        # Путь, где хранятся файлы

*self*.root\_path = "./server\_storage"

*def* ls(*self*) -> str:

        """Вывод содержимого директории на экран"""

        # Список файлов, которые есть в директории

        result\_list = []

        # Список всех файлов в текущей директории

        files\_list = os.listdir(*self*.root\_path)

        for current\_file in files\_list:

            current\_path = *self*.root\_path + *self*.sep + current\_file

            # Если это файл (не папка), то добавляем в результирующий список

            if os.path.isfile(current\_path):

                result\_list.append(current\_file)

        # Возвращаем результат в виде строки

        r = "\n".join(result\_list)

        return *f*"Содержимое {*self*.root\_path}:\n{r}"

*def* reader(*self*, *filename*: str) -> Union[None, str]:

        """Просмотр содержимого текстового файла"""

        # Полный путь до файла

        current\_path = *self*.root\_path + *self*.sep + *filename*

        # Пытаемся прочитать

        try:

            with open(current\_path, "r") as file:

                return file.read()

        # Если файл не найден

        except FileNotFoundError:

            print(*f*"Файл {*filename*} не найден")

            return None

        # Если это не файл (мало ли)

        except IsADirectoryError:

            print(*f*"Файл {*filename*} является директорией")

            return None

*def* main():

    # Создаем сокет

    server\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    # Назначаем на localhost

    server\_sock.bind(("127.0.0.1", PORT))

    # Устанавливаем кол-во соединений

    server\_sock.listen(1)

    # экземпляр FileProcessing для работы с файлами

    file\_logic = FileProcessing()

    # Ждем клиента

    while True:

        conn, addr = server\_sock.accept()

        # Делаем сокет не блокируемым

        conn.setblocking(False)

        # Собираем данные

        while True:

            try:

                # Пытаемся получить данные

                data = conn.recv(4096)

                break

            except socket.error:

                # Ждем

                pass

        file\_name = data.decode()  # данные, которые приняли

        print(*f*"Получили сообщение от клиента: {file\_name}")

        # Пытаемся прочитать файл с таким именем

        reply\_message = file\_logic.reader(file\_name)

        # Если файла нет - отдаем список всех доступных файлов для чтения

        if reply\_message is None:

            reply\_message = file\_logic.ls()

        # Отправляем результат клиенту

        conn.send(reply\_message.encode())

        print(*f*"Отправили ответное сообщение клиенту: {reply\_message}")

        # Закрываем соединение

        conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Client.py

Это клиент

"""

import socket

PORT = 8888

*def* main():

    while True:

        # Создаём сокет

        sock = socket.socket()

        # Подключаемся к серверу

        sock.connect(("127.0.0.1", PORT))

        # Вводим данные для сервера

        data\_input = input("Введите название файла или exit для выхода ->")

        # Если это сообщение для выхода - выходим

        if data\_input == "exit":

            break

        # Отправляем данные

        sock.send(data\_input.encode())

        # Получаем ответ

        data = sock.recv(4096)

        # Выводим на экран

        message = data.decode()

        print(*f*"Полученный ответ от сервера: {message}")

        # Закрываем сокет

        sock.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

22

"""

Напишите программу, которая обходит все файлы  директории, переданные ей как параметр

и удаляет все с расширением .tmp, чей размер задан как второй параметр.

Реализовать рекурсивный обход поддиректорий.

"""

import os

import sys

*class* Finder:

    """Класс для работы с файлами"""

*def* \_\_init\_\_(*self*, *root\_dir*, *bytes\_size*):

        # Корневой каталог

*self*.root\_dir = *root\_dir*

        # Размер файла в байтах, который ищем

*self*.bytes\_size = *bytes\_size*

        # ЗАпуск поиска

*self*.main\_logic(*self*.root\_dir)

*def* main\_logic(*self*, *current\_dir*):

        # Текущий путь

        current\_path = os.path.abspath(*current\_dir*)

        print(*f*"Перешли в путь {current\_path}")

        # Цикл по каждому файлу в текущем пути

        for item in os.listdir(current\_path):

            # Текущий путь до конкретного файла

            buf\_path = os.path.abspath(*f*"{*current\_dir*}{os.sep}{item}")

            # Если это файл и размер с именем совпадают, то удаляем его

            if os.path.isfile(buf\_path) and os.path.getsize(buf\_path) == *self*.bytes\_size and ".tmp" in item:

                os.remove(buf\_path)

                print(*f*"Удалили файл {buf\_path}")

            # Если директория, то рекурсивно в ней ищем файлы и также их удаляем

            elif os.path.isdir(buf\_path):

*self*.main\_logic(buf\_path)

*def* main():

    if len(sys.argv) != 3:

        print("Пожалуйста, передайте два параметра.\nПервый - путь, где происходит поиск\nВторой - размер файла в "

              "байтах для удаления")

        return

    else:

        file\_path\_input = sys.argv[1]

        file\_size\_input = int(sys.argv[2])

    finder = Finder(file\_path\_input, file\_size\_input)

if \_\_name\_\_ == "\_\_main\_\_":

    main()