МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический университет

имени В.Ф. Уткина»

Кафедра ВПМ

**Лабораторная работа 3. Вычисления на основе взаимодействия сервисов.**

Выполнил ст. гр. 3413М

Зайцев В.О.

Проверил

Князьков П.А.

Рязань, 2023

Ссылка на github-репозиторий: <https://github.com/VladimirZaitsev21/distributed-systems>

Код тетради сервера xmlrpc\_server.ipynb:

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

import xmlrpc.client

import datetime

import pandas as pd

import pickle

import numpy as np

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc\_paths = ('/RPC2',)

server = SimpleXMLRPCServer(("localhost", 8080),

                            requestHandler=RequestHandler, allow\_none=True)

log\_server = xmlrpc.client.ServerProxy("http://localhost:8018")

# Добавление в лог через сервер

def add\_log(log\_line):

    try:

        log\_server.add\_log(log\_line)

    except ConnectionRefusedError:

        pass

# Тест

def ping():

    add\_log('ping')

    return True

server.register\_function(ping, 'ping')

# Время сервера

def now():

    add\_log('now')

    return datetime.datetime.now()

server.register\_function(now, 'now')

# Отображение строкового вида, типа и значений

def show\_type(arg):

    add\_log('type')

    return (str(arg), str(type(arg)), arg)

server.register\_function(show\_type, 'type')

# Сумма

def test\_sum(a, b):

    add\_log('sum')

    return a + b

server.register\_function(test\_sum, 'sum')

# Степень

def test\_pow(a, b):

    add\_log('pow')

    return a\*\*b

server.register\_function(test\_pow, 'pow')

def in\_black\_list(candidate):

    add\_log('in\_black\_list')

    frame = pd.read\_csv('bad\_boys2.csv', header=0, sep=',', encoding='utf8')

    is\_in\_black\_list = any(frame['Surname'] == candidate)

    boy = predicate\_to\_string(is\_in\_black\_list)

    return candidate + ": "+ boy

def predicate\_to\_string(predicate):

    if (predicate == True):

        return "bad\_boy"

    else:

        return "good\_boy"

server.register\_function(in\_black\_list, 'in\_black\_list')

def to\_binary(data\_arg):

    data = data\_arg.data

    return xmlrpc.client.Binary(data)

server.register\_function(to\_binary, 'to\_binary')

def inversion(data\_arg):

    add\_log('inversion')

    picture\_matrix = pickle.loads(data\_arg.data)

    height = picture\_matrix.shape[0]

    width = picture\_matrix.shape[1]

    colors = len(picture\_matrix.shape)

    if (colors == 3):

        for i in range(height):

            for j in range(width):

                picture\_matrix[i][j][0] = 255 - picture\_matrix[i][j][0]

                picture\_matrix[i][j][1] = 255 - picture\_matrix[i][j][1]

                picture\_matrix[i][j][2] = 255 - picture\_matrix[i][j][2]

    elif(colors == 2):

        for i in range(height):

            for j in range(width):

                picture\_matrix[i][j] = 255 - picture\_matrix[i][j]

    result = pickle.dumps(picture\_matrix)

    return xmlrpc.client.Binary(result)

server.register\_function(inversion, 'inversion')

def mirror(data\_arg):

    add\_log('mirror')

    picture\_matrix = pickle.loads(data\_arg.data)

    height = picture\_matrix.shape[0]

    width = picture\_matrix.shape[1] - 1

    for i in range(height):

        for j in range(width // 2):

            original = picture\_matrix[i][j].copy()

            picture\_matrix[i][j] = picture\_matrix[i][width - j].copy()

            picture\_matrix[i][width - j] = original

    result = pickle.dumps(picture\_matrix)

    return xmlrpc.client.Binary(result)

server.register\_function(mirror, 'mirror')

def picture\_binarisation(data\_arg, threshold):

    add\_log('picture\_binarisation')

    picture\_matrix = pickle.loads(data\_arg.data)

    height = picture\_matrix.shape[0]

    width = picture\_matrix.shape[1]

    colors = len(picture\_matrix.shape)

    for i in range(height):

        for j in range(width):

            if(colors == 3):

                if(picture\_matrix[i][j][0] > threshold and picture\_matrix[i][j][1] > threshold and picture\_matrix[i][j][2] > threshold):

                    picture\_matrix[i][j][0] = 255

                    picture\_matrix[i][j][1] = 255

                    picture\_matrix[i][j][2] = 255

                else:

                    picture\_matrix[i][j][0] = 0

                    picture\_matrix[i][j][1] = 0

                    picture\_matrix[i][j][2] = 0

            elif(colors == 2):

                if(picture\_matrix[i][j] > threshold):

                    picture\_matrix[i][j] = 255

                else:

                    picture\_matrix[i][j] = 0

    result = pickle.dumps(picture\_matrix)

    return xmlrpc.client.Binary(result)

server.register\_function(picture\_binarisation, 'picture\_binarisation')

def in\_black\_list\_by(surname, name, patronym, birth):

    add\_log('in\_black\_list\_by')

    frame = pd.read\_csv('bad\_boys2.csv', header=0, sep=',', encoding='utf8')

    is\_in\_black\_list = any((frame['Surname'] == surname) & (frame['Name'] == name) & (frame['Patronym'] == patronym) & (frame['Birth'] == birth))

    boy = predicate\_to\_string(is\_in\_black\_list)

    return surname + ": " + boy

server.register\_function(in\_black\_list\_by, 'in\_black\_list\_by')

print ("Listening on port 8080...")

server.serve\_forever()

Код тетради клиента xmlrpc\_client.ipynb:

In [1]:

*# xmlrpc\_client.ipynb*

# xmlrpc\_client.ipynb

import xmlrpc.client

import pickle

import pandas as pd

server = xmlrpc.client.ServerProxy("http://localhost:8080")

stats\_server = xmlrpc.client.ServerProxy("http://localhost:8018")

print ('Ping', server.ping())

print ('Server datetime', server.now())

print ('View, type, value:', server.type(2))

print ('View, type, value:', server.type(2.))

print ('View, type, value:', server.type('My string'))

print ('View, type, value:', server.type("My string"))

print ('View, type, value:', server.type([1,2,3]))

print ('View, type, value:', server.type(["one", "two", "three"]))

print( 'Sum 2 + 3 :', server.sum(2, 3))

print( 'Sum 2 ^ 3 :', server.pow(2, 3))

Out[1]:

Ping True

Server datetime 20231023T19:04:48

View, type, value: ['2', "<class 'int'>", 2]

View, type, value: ['2.0', "<class 'float'>", 2.0]

View, type, value: ['My string', "<class 'str'>", 'My string']

View, type, value: ['My string', "<class 'str'>", 'My string']

View, type, value: ['[1, 2, 3]', "<class 'list'>", [1, 2, 3]]

View, type, value: ["['one', 'two', 'three']", "<class 'list'>", ['one', 'two', 'three']]

Sum 2 + 3 : 5

Sum 2 ^ 3 : 8

In [2]:

*#отображение монохромного изображения*

from PIL import Image # Работа с изображением

from pylab import \* # Отображение

import xmlrpc.client

# Режим отображения внутри ноутбука

%matplotlib inline

# img = Image.open('11.bmp')

img = Image.open('Jellyfish.jpg')

img\_arr = array(img)

# Исходное изображение

gray()

imshow(img\_arr)

Out[2]:

<matplotlib.image.AxesImage at 0x1de5a56dfd0>



In [3]:

# Тест бинарной передачи данных

#pimg = img\_arr.dumps()

pimg = pickle.dumps(img\_arr) # универсально

img\_bin = xmlrpc.client.Binary(pimg)

#img\_bin2 = server.to\_binary(img\_bin)

img\_bin2 = server.to\_binary(img\_bin)

#img\_arr2 = np.loads(img\_bin2.data)

img\_arr2 = pickle.loads(img\_bin2.data) # универсально

# Изображение после возрата с сервера

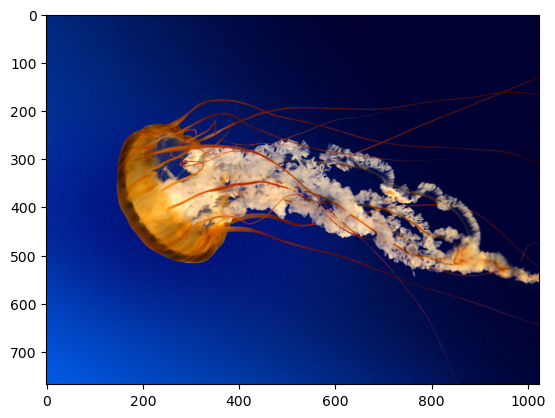
imshow(img\_arr2)

*# Исходное изображение*

imshow(img\_arr)

Out[3]:

<matplotlib.image.AxesImage at 0x1de5a7d9a90>



In [4]:

# Инверсия цвета изображения через сервер

def inv\_color(img\_arr\_in):

    pimg = pickle.dumps(img\_arr\_in)

    img\_bin = xmlrpc.client.Binary(pimg)

    #img\_bin2 = server.inversion(img\_bin)

    img\_bin2 = server.inversion(img\_bin)

    img\_arr\_out = pickle.loads(img\_bin2.data)

    return img\_arr\_out

img\_arr\_inv = inv\_color(img\_arr)

# Изображение с инверсией цвета

imshow(img\_arr\_inv)

Out[4]:

<matplotlib.image.AxesImage at 0x1de5a880c50>



#Зеркальное отображение

pimg = pickle.dumps(img\_arr)

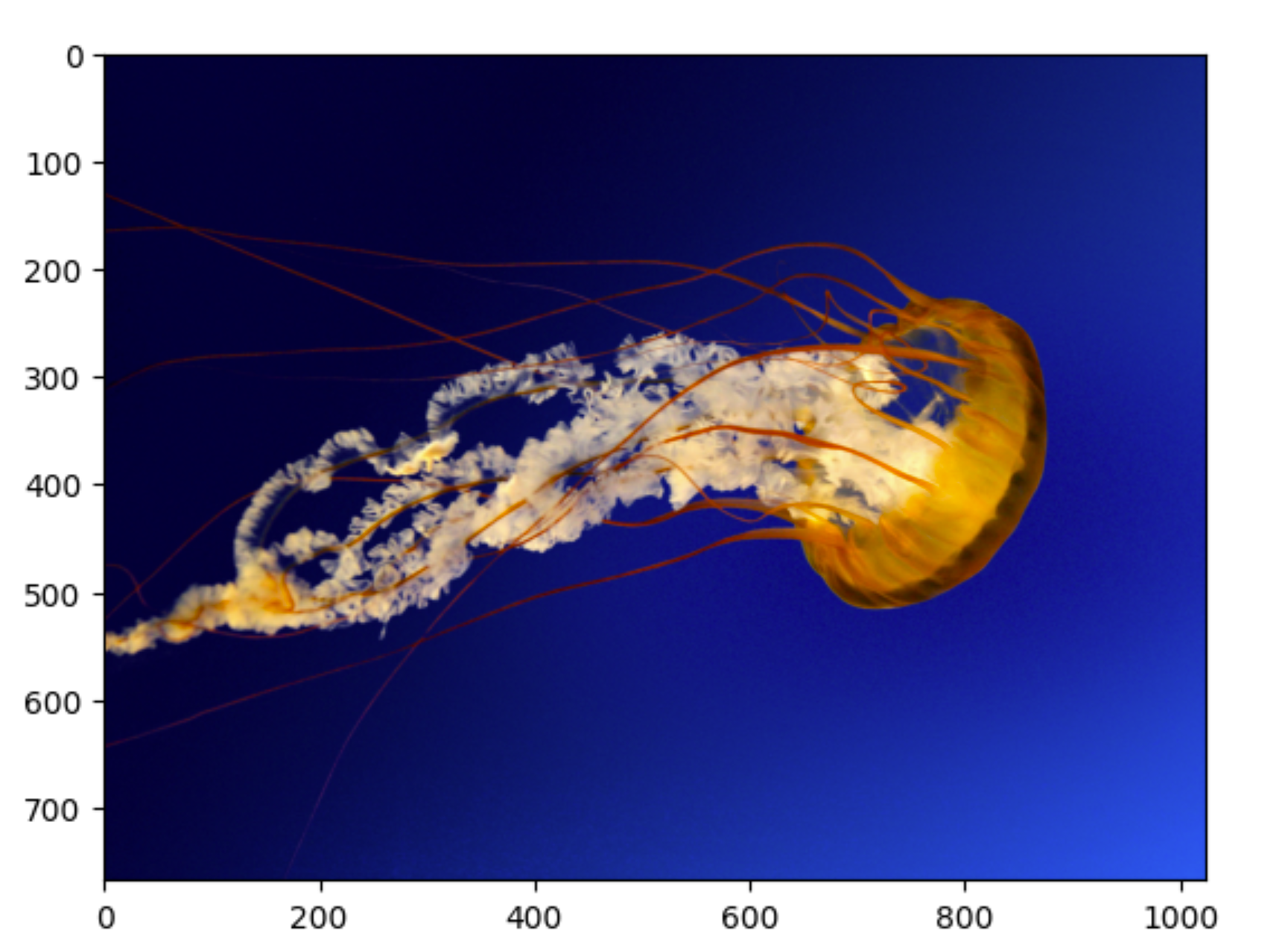
img\_bin = xmlrpc.client.Binary(pimg)

#mirr\_img\_bin = server.mirror(img\_bin)

mirr\_img\_bin = server.mirror(img\_bin)

img\_arr = pickle.loads(mirr\_img\_bin.data)

imshow(img\_arr)



#Бинаризация изображения

pimg = pickle.dumps(img\_arr)

img\_bin = xmlrpc.client.Binary(pimg)

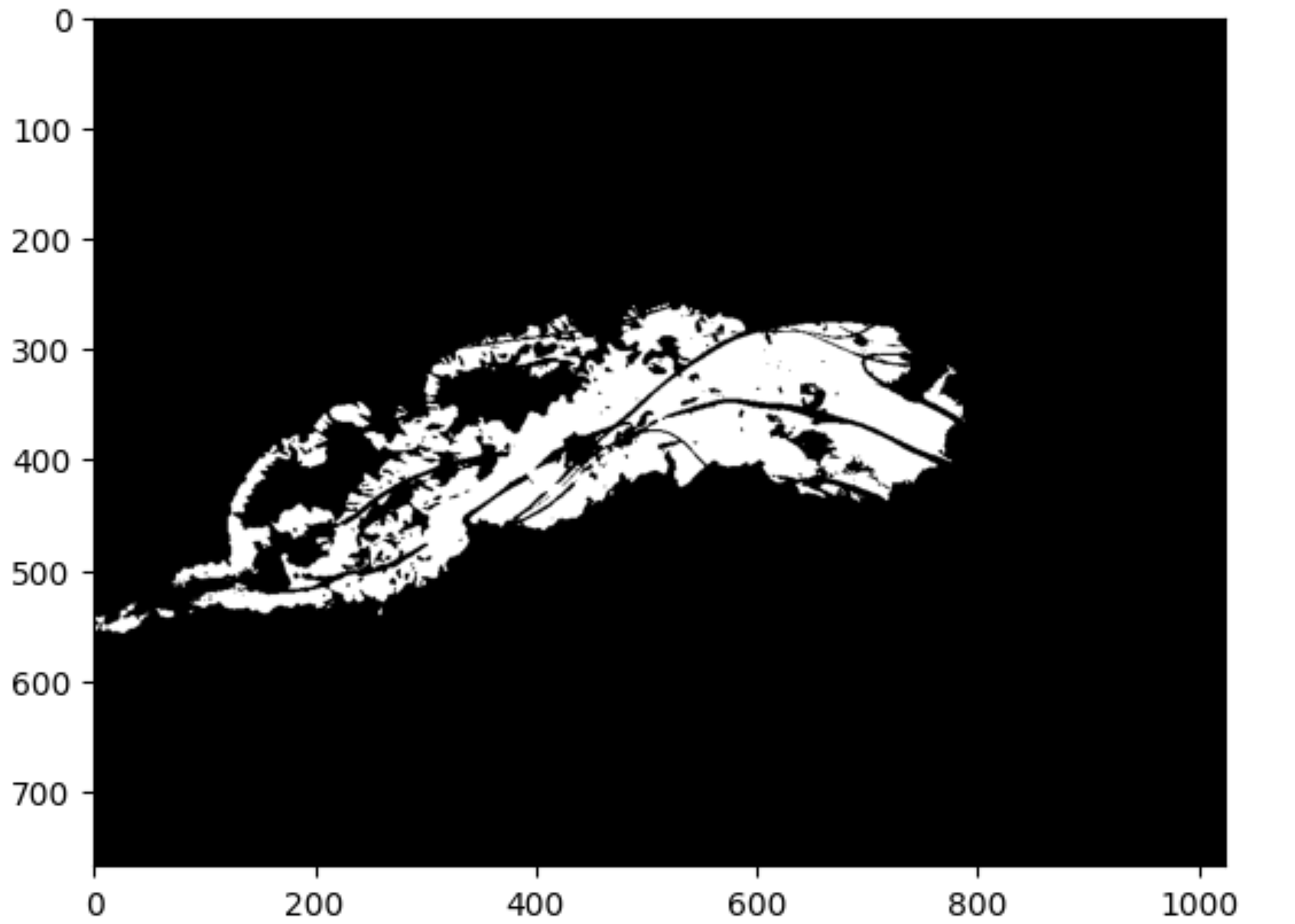
threshold = 100

#binary\_img\_bin = server.picture\_binarisation(img\_bin, porog)

binary\_img\_bin = server.picture\_binarisation(img\_bin, threshold)

binary\_img\_bin = pickle.loads(binary\_img\_bin.data)

imshow(binary\_img\_bin)



print(stats\_server.logs())

[['2023-10-10 18:01:57', 'ping'], ['2023-10-10 18:02:01', 'now'], ['2023-10-10 18:02:05', 'type'], ['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type'], ['2023-10-10 18:02:21', 'type'], ['2023-10-10 18:02:25', 'type'], ['2023-10-10 18:02:29', 'sum'], ['2023-10-10 18:02:34', 'pow'], ['2023-10-21 19:00:40', 'ping'], ['2023-10-21 19:00:44', 'now'], ['2023-10-21 19:00:49', 'type'], ['2023-10-21 19:00:53', 'type'], ['2023-10-21 19:00:57', 'type'], ['2023-10-21 19:01:01', 'type'], ['2023-10-21 19:01:05', 'type'], ['2023-10-21 19:01:09', 'type'], ['2023-10-21 19:01:13', 'sum'], ['2023-10-21 19:01:17', 'pow'], ['2023-10-23 19:04:44', 'ping'], ['2023-10-23 19:04:48', 'now'], ['2023-10-23 19:04:52', 'type'], ['2023-10-23 19:04:56', 'type'], ['2023-10-23 19:05:00', 'type'], ['2023-10-23 19:05:04', 'type'], ['2023-10-23 19:05:08', 'type'], ['2023-10-23 19:05:12', 'type'], ['2023-10-23 19:05:17', 'sum'], ['2023-10-23 19:05:21', 'pow'], ['2023-10-23 19:06:50', 'inversion'], ['2023-10-23 19:07:04', 'in\_black\_list'], ['2023-10-23 19:07:18', 'mirror'], ['2023-10-23 19:07:27', 'picture\_binarisation'], ['2023-10-23 19:07:41', 'in\_black\_list\_by']]

print(stats\_server.logs\_slice\_by\_operation('type'))

[['2023-10-10 18:02:05', 'type'], ['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type'], ['2023-10-10 18:02:21', 'type'], ['2023-10-10 18:02:25', 'type'], ['2023-10-21 19:00:49', 'type'], ['2023-10-21 19:00:53', 'type'], ['2023-10-21 19:00:57', 'type'], ['2023-10-21 19:01:01', 'type'], ['2023-10-21 19:01:05', 'type'], ['2023-10-21 19:01:09', 'type'], ['2023-10-23 19:04:52', 'type'], ['2023-10-23 19:04:56', 'type'], ['2023-10-23 19:05:00', 'type'], ['2023-10-23 19:05:04', 'type'], ['2023-10-23 19:05:08', 'type'], ['2023-10-23 19:05:12', 'type']]

print(stats\_server.logs\_slice\_by\_time('2023-10-10 18:02:09', '2023-10-10 18:02:21'))

[['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type'], ['2023-10-10 18:02:21', 'type']]

print(stats\_server.logs\_slice\_by\_time2('2023-10-10 18:02:09', '2023-10-20 18:02:21', 'type'))

[['2023-10-10 18:02:05', 'type'], ['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type']]

print(stats2\_server.get\_all\_logs())

[[1, 'ping', '20231106T11:41:15', 2.029862403869629], [2, 'now', '20231106T11:41:21', 2.0377795696258545], [3, 'show\_type', '20231106T11:41:28', 2.0361692905426025], [4, 'show\_type', '20231106T11:41:34', 2.046144962310791], [5, 'show\_type', '20231106T11:41:40', 2.0334949493408203], [6, 'show\_type', '20231106T11:41:46', 2.0442824363708496], [7, 'show\_type', '20231106T11:41:52', 2.043067216873169], [8, 'show\_type', '20231106T11:41:58', 2.056852102279663], [9, 'test\_sum', '20231106T11:42:04', 2.049945592880249], [10, 'test\_pow', '20231106T11:42:11', 2.0591015815734863], [11, 'ping', '20231106T15:10:12', 2.030303478240967], [12, 'now', '20231106T15:10:18', 2.040510654449463], [13, 'show\_type', '20231106T15:10:25', 2.024599552154541], [14, 'show\_type', '20231106T15:10:31', 2.0422048568725586], [15, 'show\_type', '20231106T15:10:37', 2.0469741821289062], [16, 'show\_type', '20231106T15:10:43', 2.033388376235962], [17, 'show\_type', '20231106T15:10:49', 2.039318561553955], [18, 'show\_type', '20231106T15:10:55', 2.0387871265411377], [19, 'test\_sum', '20231106T15:11:01', 2.042252540588379], [20, 'test\_pow', '20231106T15:11:07', 2.0401558876037598]]

print(stats2\_server.get\_logs('ping', '2023-11-03 16:02:09', '2023-11-08 19:02:21', 5))

[[1, 'ping', '20231106T11:41:15', 2.029862403869629], [11, 'ping', '20231106T15:10:12', 2.030303478240967], [21, 'ping', '20231107T17:37:04', 2.0508627891540527]]

Код тетради xmlrpc\_proxy\_server.ipynb:

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

import xmlrpc.client

import datetime

import time

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc\_paths = ('/RPC2',)

server = SimpleXMLRPCServer(("localhost", 7070),

                            requestHandler=RequestHandler, allow\_none=True)

real\_server = xmlrpc.client.ServerProxy("http://localhost:8080")

db\_server = xmlrpc.client.ServerProxy("http://localhost:6060")

def ping():

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.ping()

    end = time.time()

    exec\_time = end - start

    db\_server.store('ping', now, exec\_time)

    return result

server.register\_function(ping, 'ping')

def now():

    when = datetime.datetime.now()

    start = time.time()

    result = real\_server.now()

    end = time.time()

    exec\_time = end - start

    db\_server.store('now', when, exec\_time)

    return result

server.register\_function(now, 'now')

def show\_type(arg):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.type(arg)

    end = time.time()

    exec\_time = end - start

    db\_server.store('show\_type', now, exec\_time)

    return result

server.register\_function(show\_type, 'type')

def test\_sum(a, b):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.sum(a, b)

    end = time.time()

    exec\_time = end - start

    db\_server.store('test\_sum', now, exec\_time)

    return result

server.register\_function(test\_sum, 'sum')

def test\_pow(a, b):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.pow(a, b)

    end = time.time()

    exec\_time = end - start

    db\_server.store('test\_pow', now, exec\_time)

    return result

server.register\_function(test\_pow, 'pow')

def in\_black\_list(candidate):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.in\_black\_list(candidate)

    end = time.time()

    exec\_time = end - start

    db\_server.store('in\_black\_list', now, exec\_time)

    return result

server.register\_function(in\_black\_list, 'in\_black\_list')

def to\_binary(data\_arg):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.to\_binary(data\_arg)

    end = time.time()

    exec\_time = end - start

    db\_server.store('to\_binary', now, exec\_time)

    return result

server.register\_function(to\_binary, 'to\_binary')

def inversion(data\_arg):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.inversion(data\_arg)

    end = time.time()

    exec\_time = end - start

    db\_server.store('inversion', now, exec\_time)

    return result

server.register\_function(inversion, 'inversion')

def mirror(data\_arg):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.mirror(data\_arg)

    end = time.time()

    exec\_time = end - start

    db\_server.store('mirror', now, exec\_time)

    return result

server.register\_function(mirror, 'mirror')

def picture\_binarisation(data\_arg, threshold):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.picture\_binarisation(data\_arg, threshold)

    end = time.time()

    exec\_time = end - start

    db\_server.store('picture\_binarisation', now, exec\_time)

    return result

server.register\_function(picture\_binarisation, 'picture\_binarisation')

def in\_black\_list\_by(surname, name, patronym, birth):

    now = datetime.datetime.now()

    start = time.time()

    result = real\_server.in\_black\_list\_by(surname, name, patronym, birth)

    end = time.time()

    exec\_time = end - start

    db\_server.store('in\_black\_list\_by', now, exec\_time)

    return result

server.register\_function(in\_black\_list\_by, 'in\_black\_list\_by')

print ("Listening on port 7070...")

server.serve\_forever()

Код тетради xmlprc\_stats2\_server.ipynb:

from xmlrpc.server import SimpleXMLRPCRequestHandler

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCServer

import sqlite3

from datetime import datetime

import pandas as pd

import numpy as np

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc\_paths = ('/RPC2',)

server = SimpleXMLRPCServer(("localhost", 6060),

                            requestHandler=RequestHandler, allow\_none=True)

db\_connection = sqlite3.connect('log.db')

db\_cursor = db\_connection.cursor()

is\_table\_created = False

def store(operation, time, duration):

    db\_cursor.execute(

        """

        CREATE TABLE IF NOT EXISTS logs

        (

            log\_id integer primary key autoincrement,

            operation TEXT,

            operation\_time timestamp,

            operation\_duration real

        );

        """

    )

    db\_connection.commit()

    log = (operation, str(time), duration)

    db\_cursor.execute("INSERT INTO logs(operation, operation\_time, operation\_duration) VALUES(?, ?, ?);", log)

    db\_connection.commit()

server.register\_function(store, 'store')

def get\_all\_logs():

    db\_cursor.execute('SELECT \* FROM logs')

    return db\_cursor.fetchall()

server.register\_function(get\_all\_logs, 'get\_all\_logs')

def get\_logs(operation, time\_left, time\_right, duration):

    time\_left = datetime.strptime(time\_left, '%Y-%m-%d %H:%M:%S')

    time\_right = datetime.strptime(time\_right, '%Y-%m-%d %H:%M:%S')

    time\_left = time\_left.strftime('%Y%m%dT%H:%M:%S')

    time\_right = time\_right.strftime('%Y%m%dT%H:%M:%S')

    query = "SELECT \* FROM logs"

    wheres = list()

    if operation is not None:

        wheres.append("operation = '" + operation + "'")

    if time\_left is not None and time\_right is not None:

        wheres.append("operation\_time > '" + time\_left + "' and operation\_time < '" + time\_right + "'")

    if duration is not None:

        wheres.append("operation\_duration < " + str(duration))

    joined\_wheres = " and ".join(wheres)

    db\_cursor.execute(query + ' where ' + joined\_wheres)

    return db\_cursor.fetchall()

server.register\_function(get\_logs, 'get\_logs')

print ("Listening on port 6060...")

server.serve\_forever()

По результатам данной работы были получены следующие навыки:

1. Работа с сервером: приобретение опыта в разработке и модификации функций сервера.

2. Обработка изображений: приобретение опыта в модификации функции для работы с монохромными изображениями, добавление функции бинаризации и разворота изображения.

3. Работа без использования библиотечных функций: приобретение навыков в реализации функционала без использования сторонних библиотек для инверсии, бинаризации и разворота изображений.

4. Приобрели навыки разработки сфокусированных на одной задаче сервисов.

5. Развили навыки работы с клиент-серверной системой, работающей через прокси-сервер

В данной работе было продемонстрировано, что использование сервис-ориентированной архитектуры вычислений является целесообразным.

Сервис-ориентированная архитектура позволяет эффективно организовать работу с различными функциями сервера, а также упростить их модификацию и расширение.