МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Рязанский государственный радиотехнический университет**

**имени В. Ф. Уткина»**

Факультет вычислительной техники

Кафедра вычислительной и прикладной математики

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к лабораторной работе №3 на тему

«**Вычисления на основе взаимодействия сервисов**»

Выполнил:

Студент группы 4413м

Болонин А.В.

Проверил:

доцент,

Князьков П.А.

Рязань 2024

**Цель работы:**

Получение практических навыков реализация взаимодействия сервисов.

**Практическая часть:**

В ходе выполнения лабораторной работы были написаны следующие программы:

**Код клиента:**

import xmlrpc.client

import pickle

import pandas as pd

server = xmlrpc.client.ServerProxy("http://localhost:8000", allow\_none=True)

server\_statys = xmlrpc.client.ServerProxy("http://localhost:7000", allow\_none=True)

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('View, type, value:', server.type((1,2,"3")))

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

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

Surname, Name, Patronym, Birth = map(str, input("Введите ФИО и дату рождения:").split())

list = []  #Петров Иван Иванович 22.03.1989

list.extend((Surname, Name, Patronym, Birth))

print('Плохой ли ты парень?', server.black\_list\_check(list))

#инверсия цвета

input\_file = 'Jellyfish.bmp'

output, header = server.color\_inversion(input\_file)

output = pickle.loads(output.data)

header = pickle.loads(header.data)

output\_file = 'invers-jelly.bmp'

with open(output\_file, 'wb') as f:

    f.write(header)

    f.write(output)

input\_file = 'xxxxx.bmp'

output, header = server.color\_inversion(input\_file)

output = pickle.loads(output.data)

header = pickle.loads(header.data)

output\_file = 'invers-xx.bmp'

with open(output\_file, 'wb') as f:

    f.write(header)

    f.write(output)

#бинаризация изображения

file\_name = 'Jellyfish.bmp'

threshold = int(input('Ведите значене порога бинаризации:'))

binarized\_data, header = server.binarize\_image(file\_name, threshold)

binarized\_data = pickle.loads(binarized\_data.data)

header = pickle.loads(header.data)

output\_file = 'binarized-jelly.bmp'

with open(output\_file, 'wb') as f:

    f.write(header)

    f.write(binarized\_data)

file\_name = 'xxxxx.bmp'

threshold = int(input('Ведите значене порога бинаризации:'))

bin\_data = server.binarize\_image(file\_name, threshold)

bin\_data = pickle.loads(bin\_data.data)

bin\_data.save('binarized-xx.bmp')

#поворот изображения

file\_name = 'Jellyfish.bmp'

rotated\_data = server.rotated\_image(file\_name)

rotated\_data = pickle.loads(rotated\_data.data)

with open('rotated\_image.bmp', 'wb') as file:

    file.write(rotated\_data)

def call\_statys(criteria1, criteria2, criteria3):

    try:

        print(server\_statys.data\_search(criteria1, criteria2, criteria3))

    except Exception as e:

        return 'Ошибка, сервре недоступен'

# criteria1 = input('Введите тип операции:')

# criteria2 = input('Введите дату г\_м\_д:')

# criteria3 = input('Введите длительность выполнения операции:')

criteria1 = 'binarized\_image'

criteria2 = '2024\_12\_05'

criteria3 = '5'

call\_statys(criteria1, criteria2, criteria3)

**Код сервера:**

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

import xmlrpc.client

import datetime

import pandas as pd

import pickle

from PIL import Image

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc\_paths = ('/RPC2',)

server = SimpleXMLRPCServer(("localhost", 8018))

print("Listening on port 8018...")

# Тест

def ping():

    return True

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

# Время сервера

def now():

    return datetime.datetime.now()

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

# Отображение строкового вида, типа и значений

def show\_type(arg):

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

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

# Сумма

def summa(a, b):

    return a + b

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

# Степень

def pow(a, b):

    return a \*\* b

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

# Проверка нахождения клиента в черном списке c использованием Pandas Data Frame

def black\_list\_check(list):

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

    exist = [(frame['Surname'] == list[0]) & (frame['Name'] == list[1]) & (frame['Patronym'] == list[2]) | (frame['Birth'] == list[3])]

    index = (frame.index[frame['Surname'] == list[0]]) == (frame.index[frame['Name'] == list[1]]) == (frame.index[frame['Patronym'] == list[2]]) == (frame.index[frame['Birth'] == list[3]])

    if index and exist:

        return (f"{' '.join(list[:3])}: bad\_boy")

    else:

        return(f"{' '.join(list[:3])}: good\_boy")

server.register\_function(black\_list\_check, 'black\_list\_check')

# Бинарная передача данных

# def send\_back\_binary(bin\_data):

#     data = bin\_data.data

#     return xmlrpc.client.Binary(data)

# server.register\_function(send\_back\_binary, 'send\_back\_binary')

# Инверсия цвета

# На вход изображение RGB размерности (M, N, 3) со значениями 0-255

def color\_inversion(input\_file):

    file = open(input\_file, 'rb')

    header = file.read(54)  # прочитать первые 54 байта (заголовок файла BMP)

    bits\_per\_pixel = int.from\_bytes(header[28:30], byteorder='little')

    if bits\_per\_pixel == 24:

        pixel\_data = file.read()

        inverted\_data = bytearray()

        for i in range(0, len(pixel\_data), 3):

            red = pixel\_data[i]

            green = pixel\_data[i + 1]

            blue = pixel\_data[i + 2]

            # Инвертируем цвета

            inverted\_data.extend((255 - red, 255 - green, 255 - blue))

        serialized\_data = pickle.dumps(inverted\_data)

    else:

        image\_data = bytearray(file.read())  # Чтение пиксельных данных

        # Инверсия цветов

        inverted\_data = bytearray()

        for byte in image\_data:

            inverted\_data.append(255 - byte)

        serialized\_data = pickle.dumps(inverted\_data)

    header = pickle.dumps(header)

    return xmlrpc.client.Binary(serialized\_data), header

server.register\_function(color\_inversion, 'color\_inversion')

#Бинаризация изображения по порогу

def binarize\_image(file\_name, threshold):

    file = open(file\_name, 'rb')

    header = file.read(54)

    data = file.read()

    binarized\_data = bytearray()

    bits\_per\_pixel = int.from\_bytes(header[28:30], byteorder='little')

    if bits\_per\_pixel == 24:

        for i in range(0, len(data), 3):

            pixel\_value = sum(data[i:i+3]) // 3

            if pixel\_value > threshold:

                binarized\_data.extend(bytes([255, 255, 255]))  # Белый цвет

            else:

                binarized\_data.extend(bytes([0, 0, 0]))  # Черный цвет

        serialized\_data = pickle.dumps(binarized\_data)

        header = pickle.dumps(header)

        return xmlrpc.client.Binary(serialized\_data), header

    else:

        file.close()

        image = Image.open(file\_name)

        image = image.convert('L')  # Преобразование изображения в черно-белое

        pixel\_data = image.load()

        for i in range(image.width):

            for j in range(image.height):

                if pixel\_data[i, j] < threshold:

                    pixel\_data[i, j] = 0

                else:

                    pixel\_data[i, j] = 255

        binarized\_image = image

        serialized\_data = pickle.dumps(binarized\_image)

        return serialized\_data

server.register\_function(binarize\_image,'binarize\_image')

# Поворот изображения

def rotated\_image(image\_path):

    with open(image\_path, 'rb') as file:

        data = file.read()

    header\_size = int.from\_bytes(data[10:14], byteorder='little')

    width = int.from\_bytes(data[18:22], byteorder='little')

    height = int.from\_bytes(data[22:26], byteorder='little')

    row\_padded = (width \* 3 + 3) & (~3)

    rotate\_data = bytearray(data)

    for i in range(height):

        for j in range(width):

            pixel\_start = header\_size + i \* row\_padded + (width - j - 1) \* 3

            rotated\_pixel\_start = header\_size + i \* row\_padded + j \* 3

            rotate\_data[rotated\_pixel\_start:rotated\_pixel\_start + 3] = data[pixel\_start:pixel\_start + 3]

    rotate\_data = pickle.dumps(rotate\_data)

    return xmlrpc.client.Binary(rotate\_data)

server.register\_function(rotated\_image,'rotated\_image')

server.serve\_forever()

**Код статус сервера:**

import sqlite3

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

# Создаем базу данных для хранения данных

conn = sqlite3.connect('log.db')

c = conn.cursor()

c.execute('CREATE TABLE IF NOT EXISTS transactions (operation TEXT, time TEXT, time\_speed TEXT)')

conn.commit()

# Это создает наш прокси-сервер

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc\_paths = ('/RPC2',)

server = SimpleXMLRPCServer(('localhost', 7000))

print("Listening on port 7000...")

def stats (cashe):

    c.execute("INSERT INTO transactions (operation,time,time\_speed ) VALUES (?,?,?)", (cashe[1],cashe[0],cashe[2]))

    conn.commit()

    return True

def data\_search(criteria1, criteria2, criteria3):

    # Подключение к базе данных

    conn = sqlite3.connect('log.db')

    cursor = conn.cursor()

    # Выполнение запроса с поиском по трем критериям

    query = f"SELECT \* FROM transactions WHERE operation = ? AND time = ? AND time\_speed < ?"

    cursor.execute(query, (criteria1, criteria2, criteria3))

    # Вывод результатов

    results = cursor.fetchall()

    return results

server.register\_function(data\_search, 'data\_search')

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

server.serve\_forever()

**Код прокси сервера**

#%Y\_%m\_%d\_%H\_%M\_%S

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

import xmlrpc.client

import datetime

import time

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc\_paths = ('/RPC2',)

proxy = xmlrpc.client.ServerProxy("http://localhost:8018")

stat\_server = xmlrpc.client.ServerProxy("http://localhost:7000")

server = SimpleXMLRPCServer(("localhost", 8000),

                            requestHandler=RequestHandler)

print("Listening on port 8000...")

def toFixed(numObj, digits=3):

    return f"{numObj:.{digits}f}"

def stats(cashe):

    try:

        stat\_server.stats(cashe)

    except Exception as e:

        return 'Ошибка, срервер вне зоны доступа'

# Тест

def ping():

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('ping')  # Отправляем тип операции перед началом

    cashe.append(type)

    ping\_result = proxy.ping()

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return ping\_result

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

# Время сервера

def now():

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('now')  # Отправляем тип операции перед началом

    cashe.append(type)

    result = proxy.now()

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return result

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

# Отображение строкового вида, типа и значений

def show\_type(arg):

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('show\_type')  # Отправляем тип операции перед началом

    cashe.append(type)

    View, type, value = proxy.type(arg)

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return View, type, value

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

# Сумма

def summa(a, b):

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('summa')  # Отправляем тип операции перед началом

    cashe.append(type)

    result = proxy.summa(a,b)

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return result

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

# Степень

def pow(a, b):

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('pow')  # Отправляем тип операции перед началом

    cashe.append(type)

    result = proxy.pow(a,b)

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return result

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

# Проверка нахождения клиента в черном списке c использованием Pandas Data Frame

def black\_list\_check(list):

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('black\_list\_check')  # Отправляем тип операции перед началом

    cashe.append(type)

    result = proxy.black\_list\_check(list)

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return result

server.register\_function(black\_list\_check, 'black\_list\_check')

# Бинарная передача данных

# def send\_back\_binary(bin\_data):

#     cashe = []

#     start\_time = time.time()

#     date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

#     cashe.append(date\_received)

#     type = ('send\_back\_binary')  # Отправляем тип операции перед началом

#     cashe.append(type)

#     result = proxy.send\_back\_binary(bin\_data)

#     end\_time = time.time()

#     res = end\_time - start\_time

#     cashe.append(toFixed(res))

#     stats(cashe)

#     return result

# server.register\_function(send\_back\_binary, 'send\_back\_binary')

# Инверсия цвета

# На вход изображение RGB размерности (M, N, 3) со значениями 0-255

def color\_inversion(input\_file):

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('color\_inversion')  # Отправляем тип операции перед началом

    cashe.append(type)

    result1, result2 = proxy.color\_inversion(input\_file)

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return result1, result2

server.register\_function(color\_inversion, 'color\_inversion')

# Бинаризация изображения по порогу

def binarize\_image(file\_name, threshold):

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('binarized\_image')  # Отправляем тип операции перед началом

    cashe.append(type)

    result = proxy.binarize\_image(file\_name, threshold)

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return result

server.register\_function(binarize\_image,'binarize\_image')

# Поворот изображения

def rotated\_image(image\_path):

    cashe = []

    start\_time = time.time()

    date\_received = datetime.datetime.now().strftime('%Y\_%m\_%d')

    cashe.append(date\_received)

    type = ('rotated\_image')  # Отправляем тип операции перед началом

    cashe.append(type)

    result = proxy.rotated\_image(image\_path)

    end\_time = time.time()

    res = end\_time - start\_time

    cashe.append(toFixed(res))

    stats(cashe)

    return result

server.register\_function(rotated\_image,'rotated\_image')

server.serve\_forever()

Были усовершенствованы навыки работы с серверной архитектурой, а также изучены правила организации сервис-ориентированной архитектуры вычислений с использованием прокси сервера. В данной работе была использована реляционная база данных для хранения типа операции, даты, и времени ее длительности. Заполнение базы данных осуществлялось при помощи sql запросов.

В ходе выполнения работы был создан удаленный репозиторий на сайте GitHub: https://github.com/Lalipop1/laboratornaya\_rabota

**Вывод:**

Целесообразность использования сервис-ориентированной архитектуры (SOA) заключается в следующем:

1. Повышение гибкости и масштабируемости системы. SOA позволяет легко добавлять или удалять сервисы, а также масштабировать их производительность в зависимости от нагрузки.
2. Улучшение модульности и повторного использования кода. Сервисы могут быть разработаны и использованы повторно, что сокращает время разработки и улучшает качество кода.
3. Упрощение интеграции с другими системами и сервисами. SOA обеспечивает стандартизированные интерфейсы для взаимодействия между различными компонентами системы, облегчая интеграцию.
4. Снижение рисков и повышение безопасности. Разделение системы на независимые сервисы уменьшает вероятность возникновения ошибок и сбоев, а также упрощает управление безопасностью.

В целом, использование SOA способствует повышению эффективности, гибкости и адаптивности информационных систем.