**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе**

**по дисциплине «Архитектура распределенных вычислительных систем»**

**Тема: ЗНАКОМСТВО С РАСПРЕДЕЛЕННЫМИ ВЫЧИСЛИТЕЛЬНЫМИ СИСТЕМАМИ.**

Студентка гр. 6383 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Василенко Т. В.

Студент гр. 6383 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Базаров И. А.

Студент гр. 6383 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глушков Н. Д.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Борисенко К. А.

Санкт-Петербург

2018.

**Цель работы:**

Знакомство с распределенными вычислительными системами.

**Оригинальный вариант задания:**

Развернуть клиент-серверное приложение, используя виртуальные

машины Ub1, Ub3. Ub1 — сервер, Ub3 — клиент. На Ub1 создать файл,

содержащий числа (integer). При вводе данных с клавиатуры на Ub3,

необходимо, чтобы клиент запрашивал новое число из файла, находящегося на Ub1. После получения данных, необходимо сложить (перемножить, разделить и др.) оба числа. Повторить данное действие в цикле 10 раз.

**Модифицированный вариант задания:**

Развернуть клиент-серверное приложение, используя виртуальные

машины ubuntu1, ubuntu3. ubuntu1 — сервер, ubuntu3 — клиент. На ubuntu1

создать файл, содрежащий числа (integer). На клиенте описать выполнение

операций (+, -, \*, /, ^). Клиент должен проводить вычисления, согласно

последовательностям чисел, хранящихся в файле на ubuntu1, и операций, делая запросы к серверу. При подключении клиента к ubuntu1, сервер отдает

следующее число из файла.

**Спецификация задания:**

Сервер ubuntu1 отдает числа и ждет соединений на порту TCP 15000. Как-

только подключается клиент, сервер отдает следующее число из файла. Число — натуральное — кодируется 8 байтами. После того, как числа в файле закончились, передается 0. Для последующих запросов файл читается сначала. Выполнение операций осуществляет клиент. Реализация задания выполнена на языке Python.

Модель Системы

C01 C21

Client

Ubuntu2

Server

Ubuntu1

C10 C12

Ip: 10.0.2.5 ip:10.0.2.4

События процессов:

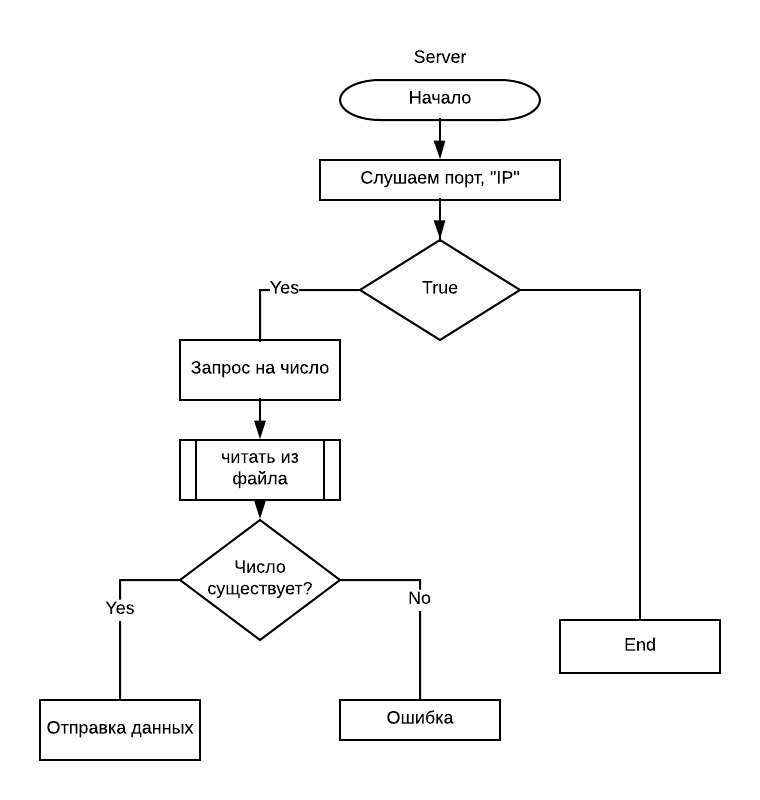
1. Запускаем сервер и клиент на виртуальных машинах. В сервере создаем(уже создан) файл с числами num.txt
2. Клиент отправляет запрос на сервер для получение чисел.
3. Сервер получает запрос(номер строки), считывает из файла и отправляет клиенту полученное число из файла
4. Клиент запрашивает операцию (+-\*/^)
5. Клиент опять же отправляет запрос на сервер для получение чисел.
6. Сервер получает запрос(номер строки), обрабатывает и отправляет клиенту полученное число из файла
7. Клиент получает число, выполняет операцию с числами и выдает результат.
8. События выполняются до тех пор, пока числа из файла не закончатся

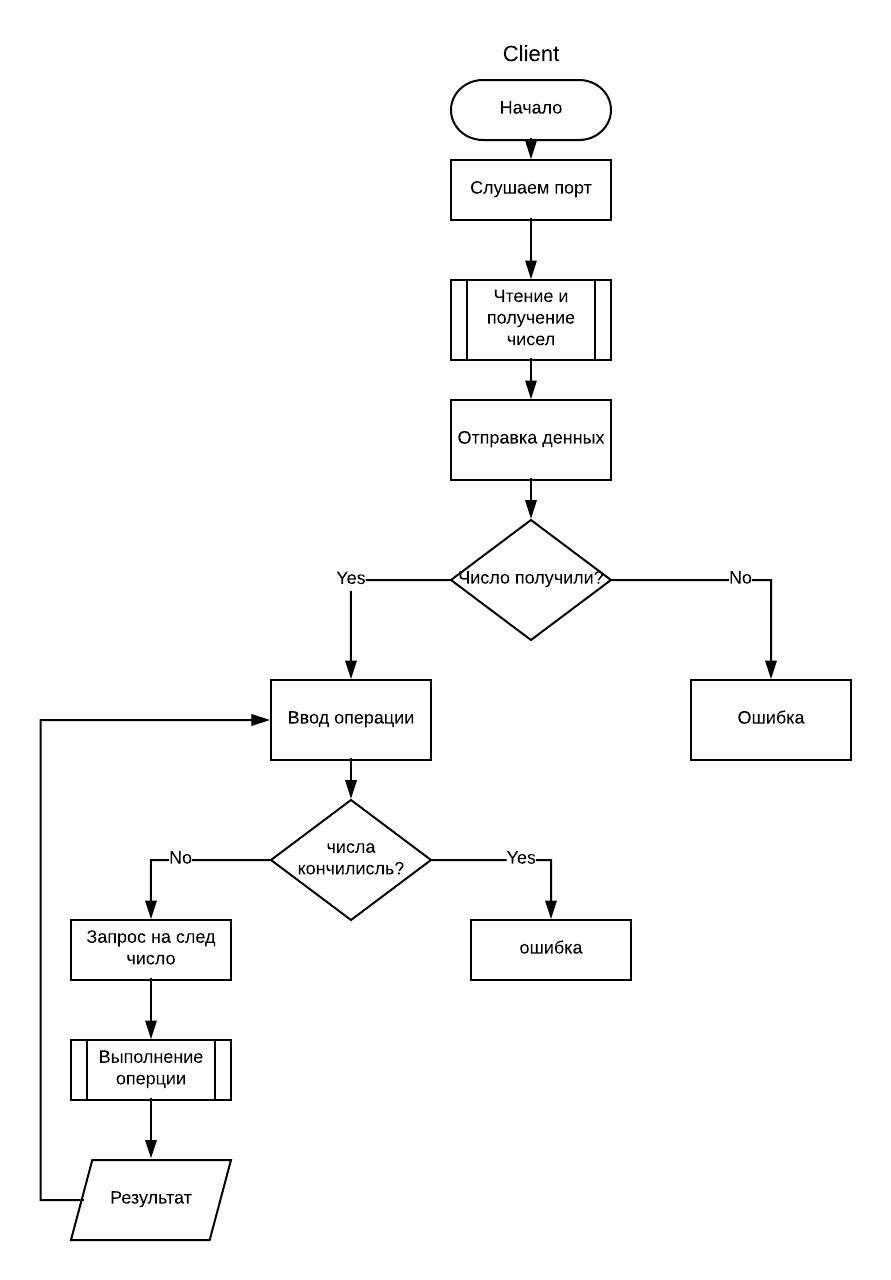
Описание событий:

1. e1 (client, P1, m1, C12)
2. e2(server, P2, m2, C12)
3. e3(server, P3, C10-C01, P4, m3, C21)
4. e4(client, P5, m4, P6)
5. e5(client, P7, m5)

|  |  |
| --- | --- |
| P1 – запрос на число |  |
| P2 – получение запроса | M2- received number |
| P3- читать из файла | M3- sent data |
| P4 – отправить данные | M4 – enter operation |
| P5 – запрос на операцию | M5 – result |
| P6 – ввод операций(+,-/,\*,^) |  |
| P7 – выполнение операций |  |

**Блок схема**



****

**Вывод:**

В ходе практической работы написано клиент-серверное приложение на языке Python, выполняющее базовые арифметические операции по запросу клиентов в соответствии с входными данными (числа), записанными в файле на сервере.

**Исходный код**

**server.py**

import socket

import struct

address = (``, 15000)

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

sock.bind(address)

sock.listen(1)

def serve():

conn, addr = sock.accept()

recv = conn.recv(8)

print("Received:", recv, "from", addr)

try:

line, = struct.unpack("q", recv)

except:

conn.close()

return serve()

err = False

num = 0

try:

with open("num.txt", "r") as nums\_file:

num\_s = nums\_file.readlines()[line - 1]

num = int(num\_s)

except:

err = True

data = struct.pack("q", -1 if err else num)

conn.send(data)

conn.close()

print("Sent", data)

serve()

serve()

**Client.py**

import socket

import struct

server\_add = ("10.0.2.4", 15000)

def get\_num(n):

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

data = struct.pack("q", n)

sock.connect(server\_add)

sock.sendall(data)

resp = sock.recv(8)

sock.close()

(result, ) = struct.unpack("q", resp)

return result

def read\_num:

n = input("Enter line number for number: ")

try:

n = int(n)

if n <= 0:

raise ValueError()

num = get\_num(n)

except (ValueError, struct.error):

print("Incorrect input")

return read\_num()

print("Received", num)

if num >= 0:

return num

else:

print("Server returned error. Try again.")

return read\_num()

def read\_op():

o = input(«Enter next operation»)

n = int(n)

if n< = 0:

raise ValueError()

num = get\_num(n)

except(ValueError, struct.error):

print(«Incorrect input»)

return read\_num()

print(«Recieved: », num)

if num >= 0:

return num

else:

print(«server returned error. Try again»)

return read\_num()

def read\_op:

o = input(«enter next operation»)

if o in «+-\*/^»:

return o

else:

print(«Incorrect input»)

return read\_op()

def work(value = 0, op = None):

num = read\_num()

if op == None:

value = num

elif op == "+":

print(value, "+", num, " =", value + num)

value += num

elif op == "-":

print(value, "-", num, "=", value - num)

value -= num

elif op == "\*":

print(value, "\*", num, "=", value \* num)

value \*= num

elif op == "/":

print(value, "/", num, "=", value // num)

value //= num

elif op == "^":

print(value, "^", num, "=", value \*\* num)

value \*\*= num

print("Current result:", value)

op = read\_op()

work(value, op)

work()