Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторные работы №5-7 по курсу**

**«Операционные системы»**

**УПРАВЛЕНИЕ СЕРВЕРАМИ СООБЩЕНИЙ**

**ПРИМЕНЕНИЕ ОТЛОЖЕННЫХ ВЫЧИСЛЕНИЙ**

**ИНТЕГРАЦИЯ ПРОГРАММНЫХ СИСТЕМ ДРУГ С ДРУГОМ**

Студент: Жаворонков Михаил Николаевич

Группа: М8О–212Б–22

Вариант: 12

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Постановка задачи**

## Цель работы

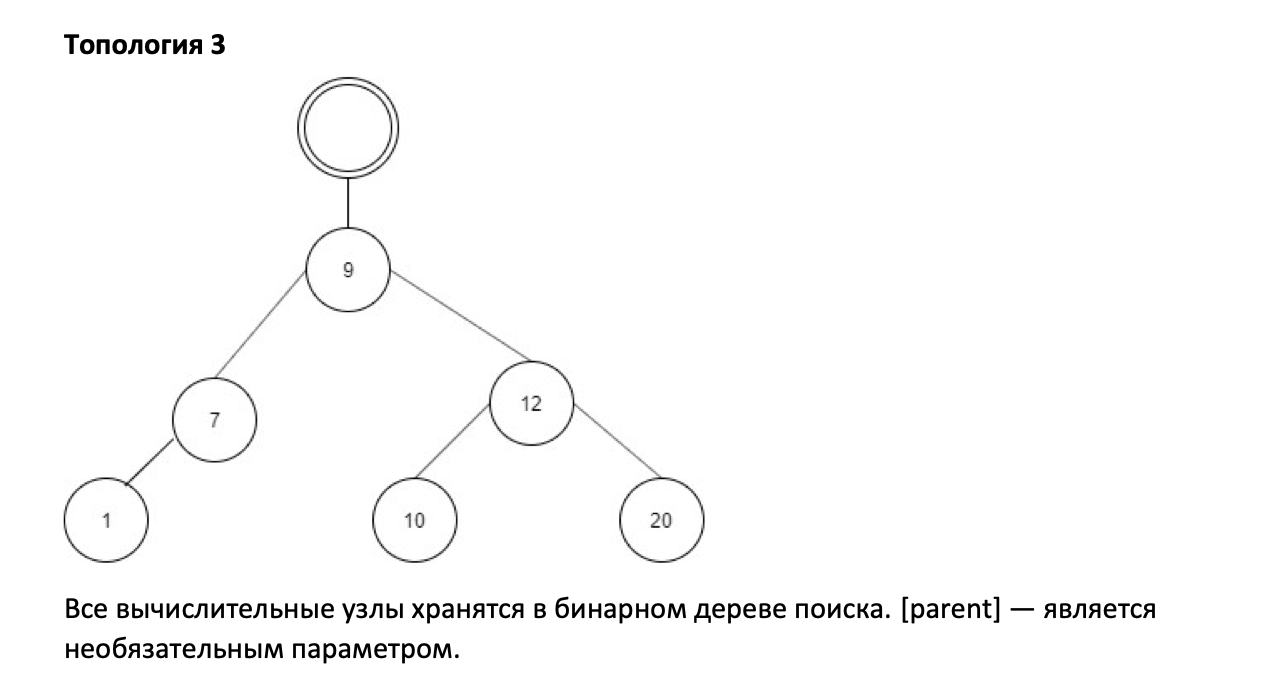
Целью является приобретение практических навыков в:

* Управлении серверами сообщений (№5)
* Применение отложенных вычислений (№6)
* Интеграция программных систем друг с другом (№7)

## Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность. Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы.

Задания моего варианта:



**Набор команд 2 (локальный целочисленный словарь)**

Формат команды сохранения значения: exec id name value

**Команда проверки 1 (pingall)**

**Общие сведения о программе**

Tree.h, tree.cpp – реализация дерева для хранения узлов;

Client.cpp – реализация вычислительного узла;

Main.cpp – реализация управляющего узла;

CMakeLists.txt – cmake файл для сборки программы.

**Общий метод и алгоритм решения**.

1. Изучить библиотеку ZeroMQ для работы с очередями сообщений
2. Реализовать хранение уникальных значений в мапе
3. Реализовать бинарное дерево в соответствии с топологией задания
4. Реализовать главный управляющий узел, который принимает команды со стандартного входа и отправляет их на вычислительные узлы.
5. Реализовать вычислительный узел, который принимает команды от управляющего узла и выполняет их или перенаправляет команды на дочерние узлы.

**Основные файлы программы**

tree.h:

#pragma once

#include <vector>

struct Node {

int id;

Node\* left;

Node\* right;

bool found;

};

class Tree {

public:

void push(int);

void kill(int);

std::vector<int> get\_nodes();

~Tree();

private:

Node\* root = NULL;

Node\* push(Node\* t, int);

Node\* kill(Node\* t, int);

void get\_nodes(Node\*, std::vector<int>&);

void delete\_node(Node\*);

};

tree.cpp:

#include <iostream>

#include <vector>

#include <algorithm>

#include "../include/tree.h"

Tree::~Tree() {

delete\_node(root);

}

void Tree::push(int id) {

root = push(root, id);

}

void Tree::kill(int id) {

root = kill(root, id);

}

void Tree::delete\_node(Node\* node) {

if (node == NULL) {

return;

}

delete\_node(node->right);

delete\_node(node->left);

delete node;

}

std::vector<int> Tree::get\_nodes() {

std::vector<int> result;

get\_nodes(root, result);

return result;

}

void Tree::get\_nodes(Node\* node, std::vector<int>& v) {

if (node == NULL) {

return;

}

get\_nodes(node->left, v);

v.push\_back(node->id);

get\_nodes(node->right, v);

}

Node\* Tree::push(Node\* root, int val) {

if (root == NULL) {

root = new Node;

root->id = val;

root->left = NULL;

root->right = NULL;

root->found = false;

return root;

}

else if (val < root->id) {

root->left = push(root->left, val);

}

else if (val >= root->id) {

root->right = push(root->right, val);

}

return root;

}

Node\* Tree::kill(Node\* root\_node, int val) {

Node\* node;

if (root\_node == NULL) {

return NULL;

}

else if (val < root\_node->id) {

root\_node->left = kill(root\_node->left, val);

}

else if (val >root\_node->id) {

root\_node->right = kill(root\_node->right, val);

}

else {

node = root\_node;

if (root\_node->left == NULL) {

root\_node = root\_node->right;

}

else if (root\_node->right == NULL) {

root\_node = root\_node->left;

}

delete node;

}

if (root\_node == NULL) {

return root\_node;

}

return root\_node;

}

main.cpp

#include <iostream>

#include <unistd.h>

#include <string>

#include <vector>

#include <sstream>

#include <signal.h>

#include <cassert>

#include "../include/tree.h"

#include <zmq.hpp>

#include <chrono>

#include <thread>

using namespace std;

// const int TIMER = 500;

const int DEFAULT\_PORT = 5050;

int n = 2;

pthread\_mutex\_t mutex1;

zmq::context\_t context(1);

zmq::socket\_t main\_socket(context, ZMQ\_REQ);

// общая функция для отправки сообщения в дочерний процесс

bool send\_message(zmq::socket\_t &socket, const string &message\_string) {

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

return socket.send(message);

}

string receive\_message(zmq::socket\_t &socket) {

zmq::message\_t message;

bool ok = false;

try {

ok = socket.recv(&message);

}

catch (...) {

ok = false;

}

string recieved\_message(static\_cast<char\*>(message.data()), message.size());

if (recieved\_message.empty() || !ok) {

return "Root is dead";

}

return recieved\_message;

}

// меняем созданый fork процесс на дочерний, передавая туда нужные нам аргументы (клиент)

void create\_node(int id, int port) {

char\* arg0 = strdup("./client");

char\* arg1 = strdup((to\_string(id)).c\_str());

char\* arg2 = strdup((to\_string(port)).c\_str());

char\* args[] = {arg0, arg1, arg2, NULL};

execv("./client", args);

}

// функция, собирающая полный адрес до дочернего процесса

string get\_port\_name(const int port) {

return "tcp://127.0.0.1:" + to\_string(port);

}

bool is\_number(const string& val) {

try {

int tmp = stoi(val);

return true;

}

catch(exception& e) {

cout << "Error: " << e.what() << "\n";

return false;

}

}

int main() {

Tree T;

std::vector<int> nodes;

string command;

int child\_pid = 0;

int child\_id = 0;

pthread\_mutex\_init(&mutex1, NULL);

cout << "Commands:\n";

cout << "1. create (id)\n";

cout << "2. exec (id) (name, value)\n";

cout << "3. kill (id)\n";

cout << "4. pingall \n";

cout << "6. exit\n" << endl;

while (true) {

cin >> command;

if (command == "create") {

n++;

size\_t node\_id = 0;

string str = "";

string result = "";

cin >> str;

if (!is\_number(str)) {

continue;

}

node\_id = stoi(str);

if (child\_pid == 0) { // если у сервера еще нет дочернего процесса

main\_socket.bind(get\_port\_name(DEFAULT\_PORT + node\_id));

// main\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER); // меняем максимальное время ожидание ответа

// main\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER); // меняем макс время отправки сообщения

child\_pid = fork(); // создаем дочерний процесс, вызванный фукнцией create

if (child\_pid == -1) {

cout << "Unable to create first worker node\n";

child\_pid = 0;

exit(1);

} else if (child\_pid == 0) { // внутри дочернего

create\_node(node\_id, DEFAULT\_PORT + node\_id); // создаем исполняющий узел с заданным user id

} else { // внутри серверного узла

child\_id = node\_id;

// main\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

// main\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

send\_message(main\_socket, "pid"); // отправляем в дочерний процесс запрос на pid

result = receive\_message(main\_socket); // получаем ответ с pid'ом

}

} else { // если у сервера есть исполнители

// main\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

// main\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

string msg\_s = "create " + to\_string(node\_id);

send\_message(main\_socket, msg\_s);

result = receive\_message(main\_socket);

}

if (result.substr(0, 2) == "Ok") { // если все создалось то добавляем в дерево айди кентика

T.push(node\_id);

nodes.push\_back(node\_id);

}

cout << result << "\n";

} else if (command == "kill") {

int node\_id = 0;

string str = "";

cin >> str;

if (!is\_number(str)) {

continue;

}

node\_id = stoi(str);

if (child\_pid == 0) { // если нет исполнителей

cout << "Error: Not found\n";

continue;

}

if (node\_id == child\_id) { // предполагаем, что если убить процесс посередке, дальше сигнал никак не пройдет

kill(child\_pid, SIGTERM);

kill(child\_pid, SIGKILL);

child\_id = 0;

child\_pid = 0;

T.kill(node\_id);

cout << "Ok\n";

continue;

}

string message\_string = "kill " + to\_string(node\_id);

send\_message(main\_socket, message\_string);

string recieved\_message;

recieved\_message = receive\_message(main\_socket);

if (recieved\_message.substr(0, min<int>(recieved\_message.size(), 2)) == "Ok") {

T.kill(node\_id);

}

cout << recieved\_message << "\n";

}

else if (command == "exec") {

string input\_string;

string id\_str = "";

string name = "";

string value = "0";

int id = 0;

getline(cin, input\_string);

istringstream iss(input\_string);

vector<std::string> words;

std::string word;

while (iss >> word) {

words.push\_back(word);

}

id\_str = words[0];

if (!is\_number(id\_str)) {

continue;

}

id = stoi(id\_str);

name = words[1];

if (words.size() == 2) {

string message\_string = "exec " + to\_string(id) + " " + name + " " + "NOVALUE";

send\_message(main\_socket, message\_string);

string recieved\_message = receive\_message(main\_socket);

cout << recieved\_message << "\n";

}

if (words.size() == 3) {

value = words[2];

string message\_string = "exec " + to\_string(id) + " " + name + " " + value;

send\_message(main\_socket, message\_string);

string recieved\_message = receive\_message(main\_socket);

cout << recieved\_message << "\n";

}

}

else if (command == "ping") {

string id\_str = "";

int id = 0;

cin >> id\_str;

if (!is\_number(id\_str)) {

continue;

}

id = stoi(id\_str);

string message\_string = "ping " + to\_string(id);

send\_message(main\_socket, message\_string);

string recieved\_message = receive\_message(main\_socket);

cout << recieved\_message << "\n";

}

else if (command == "pingall") {

string unreachable\_nodes;

for (int i = 0; i < nodes.size(); i++) {

int id = nodes[i];

string message\_string = "ping " + to\_string(id);

send\_message(main\_socket, message\_string);

string response = receive\_message(main\_socket);

if (response.substr(0, 2) != "Ok") {

unreachable\_nodes += to\_string(id) + ";";

}

}

if (unreachable\_nodes.empty()) {

cout << "Ok: -1\n"; // Все узлы доступны

} else {

unreachable\_nodes.pop\_back(); // Удаляем последнюю точку с запятой

cout << "Ok: " << unreachable\_nodes << "\n";

}

}

else if (command == "exit") {

int n = system("killall client");

break;

}

}

return 0;

}

client.cpp

#include <iostream>

#include <unistd.h>

#include <stdlib.h>

#include <string>

#include <sstream>

#include <exception>

#include <map>

#include <signal.h>

#include <zmq.hpp>

using namespace std;

const int TIMER = 500;

const int DEFAULT\_PORT = 5050;

int n = 2;

std::map<std::string, int> m;

bool send\_message(zmq::socket\_t &socket, const string &message\_string) {

zmq::message\_t message(message\_string.size());

memcpy(message.data(), message\_string.c\_str(), message\_string.size());

return socket.send(message);

}

string receive\_message(zmq::socket\_t &socket) {

zmq::message\_t message;

bool ok = false;

try {

ok = socket.recv(&message);

}

catch (...) {

ok = false;

}

string recieved\_message(static\_cast<char\*>(message.data()), message.size());

if (recieved\_message.empty() || !ok) {

return "";

}

return recieved\_message;

}

void create\_node(int id, int port) {

char\* arg0 = strdup("./client");

char\* arg1 = strdup((to\_string(id)).c\_str());

char\* arg2 = strdup((to\_string(port)).c\_str());

char\* args[] = {arg0, arg1, arg2, NULL};

execv("./client", args);

}

string get\_port\_name(const int port) {

return "tcp://127.0.0.1:" + to\_string(port);

}

void real\_create(zmq::socket\_t& parent\_socket, zmq::socket\_t& socket, int& create\_id, int& id, int& pid) {

cout << to\_string(id);

if (pid == -1) {

send\_message(parent\_socket, "Error: Cannot fork");

pid = 0;

}

else if (pid == 0) {

create\_node(create\_id, DEFAULT\_PORT + create\_id);

}

else { // отправляем pid в дочерний и в родительский узлы

id = create\_id;

send\_message(socket, "pid");

send\_message(parent\_socket, receive\_message(socket));

}

}

void real\_kill(zmq::socket\_t& parent\_socket, zmq::socket\_t& socket, int& delete\_id, int& id, int& pid, string& request\_string) {

if (id == 0) {

send\_message(parent\_socket, "Error: Not found");

}

else if (id == delete\_id) {

send\_message(socket, "kill\_children");

receive\_message(socket);

kill(pid, SIGTERM);

kill(pid, SIGKILL);

id = 0;

pid = 0;

send\_message(parent\_socket, "Ok");

}

else {

send\_message(socket, request\_string);

send\_message(parent\_socket, receive\_message(socket));

}

}

void real\_exec(zmq::socket\_t& parent\_socket, zmq::socket\_t& socket, int& id, int& pid, string& request\_string) {

if (pid == 0) {

string receive\_message = "Error:" + to\_string(id);

receive\_message += ": Not found";

send\_message(parent\_socket, receive\_message);

}

else {

send\_message(socket, request\_string);

string str = receive\_message(socket);

if (str == "") str = "Error: Node is unavailable";

send\_message(parent\_socket, str);

}

}

void real\_ping(zmq::socket\_t& parent\_socket, zmq::socket\_t& socket, int& id, int& pid, string& request\_string) {

if (pid == 0) {

string receive\_message = "Error:" + to\_string(id);

receive\_message += ": Not found";

send\_message(parent\_socket, receive\_message);

}

else {

send\_message(socket, request\_string);

string str = receive\_message(socket);

if (str == "") str = "Ok: 0";

send\_message(parent\_socket, str);

}

}

void exec(istringstream& command\_stream, zmq::socket\_t& parent\_socket, zmq::socket\_t& left\_socket,

zmq::socket\_t& right\_socket, int& left\_pid, int& right\_pid, int& id, string& request\_string) {

string name, value;

int exec\_id;

command\_stream >> exec\_id;

if (exec\_id == id) {

command\_stream >> name;

command\_stream >> value;

string receive\_message = "";

string answer = "";

if (value == "NOVALUE") {

receive\_message = "Ok:" + to\_string(id) + ":";

if (m.contains(name)) {

receive\_message += to\_string(m[name]);

} else {

receive\_message += " '" + name + "' not found";

}

} else {

m[name] = stoi(value);

receive\_message = "Ok:" + to\_string(id);

}

send\_message(parent\_socket, receive\_message);

} else if (exec\_id < id) {

real\_exec(parent\_socket, left\_socket, exec\_id, left\_pid, request\_string);

} else {

real\_exec(parent\_socket, right\_socket, exec\_id, right\_pid, request\_string);

}

}

void ping(istringstream& command\_stream, zmq::socket\_t& parent\_socket, zmq::socket\_t& left\_socket,

zmq::socket\_t& right\_socket, int& left\_pid, int& right\_pid, int& id, string& request\_string) {

int ping\_id;

string receive\_message;

command\_stream >> ping\_id;

if (ping\_id == id) {

receive\_message = "Ok: 1";

send\_message(parent\_socket, receive\_message);

} else if (ping\_id < id) {

real\_ping(parent\_socket, left\_socket, ping\_id, left\_pid, request\_string);

}

else {

real\_ping(parent\_socket, right\_socket, ping\_id, right\_pid, request\_string);

}

}

void kill\_children(zmq::socket\_t& parent\_socket, zmq::socket\_t& left\_socket,

zmq::socket\_t& right\_socket, int& left\_pid, int& right\_pid) {

if (left\_pid == 0 && right\_pid == 0) {

send\_message(parent\_socket, "Ok");

} else {

if (left\_pid != 0) {

send\_message(left\_socket, "kill\_children");

receive\_message(left\_socket);

kill(left\_pid, SIGTERM);

kill(left\_pid, SIGKILL);

}

if (right\_pid != 0) {

send\_message(right\_socket, "kill\_children");

receive\_message(right\_socket);

kill(right\_pid, SIGTERM);

kill(right\_pid, SIGKILL);

}

send\_message(parent\_socket, "Ok");

}

}

int main(int argc, char\*\* argv) {

// анализируем входные данные

int id = stoi(argv[1]);

int parent\_port = stoi(argv[2]);

zmq::context\_t context(3);

zmq::socket\_t parent\_socket(context, ZMQ\_REP);

parent\_socket.connect(get\_port\_name(parent\_port));

parent\_socket.set(zmq::sockopt::rcvtimeo, TIMER);

parent\_socket.set(zmq::sockopt::sndtimeo, TIMER);

int left\_pid = 0;

int right\_pid = 0;

int left\_id = 0;

int right\_id = 0;

zmq::socket\_t left\_socket(context, ZMQ\_REQ);

zmq::socket\_t right\_socket(context, ZMQ\_REQ);

while(true) {

string request\_string = receive\_message(parent\_socket); // получаем запрос от родительского процесса

istringstream command\_stream(request\_string);

string command;

command\_stream >> command;

if (command == "id") {

string parent\_string = "Ok:" + to\_string(id);

send\_message(parent\_socket, parent\_string);

} else if (command == "pid") {

string parent\_string = "Ok:" + to\_string(getpid());

send\_message(parent\_socket, parent\_string);

} else if (command == "create") {

int create\_id;

command\_stream >> create\_id;

if (create\_id == id) { // если айди занят

string message\_string = "Error: Already exists";

send\_message(parent\_socket, message\_string);

} else if (create\_id < id) { // если поддерево левое

if (left\_pid == 0) { // если в левом дереве еще нет исполнителей

left\_socket.bind(get\_port\_name(DEFAULT\_PORT + create\_id));

left\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

left\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

left\_pid = fork();

real\_create(parent\_socket, left\_socket, create\_id, left\_id, left\_pid);

} else { // если в левом дереве уже есть исполнитель

send\_message(left\_socket, request\_string);

string str = receive\_message(left\_socket);

if (str == "") {

left\_socket.bind(get\_port\_name(DEFAULT\_PORT + create\_id));

left\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

left\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

left\_pid = fork();

real\_create(parent\_socket, left\_socket, create\_id, left\_id, left\_pid);

} else {

send\_message(parent\_socket, str); // отправляем результат об успешном создании в родителя

n++;

left\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

left\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

}

}

} else { // если поддерево правое

if (right\_pid == 0) {

right\_socket.bind(get\_port\_name(DEFAULT\_PORT + create\_id));

right\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

right\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

right\_pid = fork();

real\_create(parent\_socket, right\_socket, create\_id, right\_id, right\_pid);

} else {

send\_message(right\_socket, request\_string);

string str = receive\_message(right\_socket);

if (str == "") {

right\_socket.bind(get\_port\_name(DEFAULT\_PORT + create\_id));

right\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

right\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

right\_pid = fork();

real\_create(parent\_socket, right\_socket, create\_id, right\_id, right\_pid);

} else {

send\_message(parent\_socket, str);

n++;

right\_socket.set(zmq::sockopt::rcvtimeo, n \* TIMER);

right\_socket.set(zmq::sockopt::sndtimeo, n \* TIMER);

}

}

}

} else if (command == "kill") {

int delete\_id;

command\_stream >> delete\_id;

if (delete\_id < id) {

real\_kill(parent\_socket, left\_socket, delete\_id, left\_id, left\_pid, request\_string);

} else {

real\_kill(parent\_socket, right\_socket, delete\_id, right\_id, right\_pid, request\_string);

}

} else if (command == "exec") {

exec(command\_stream, parent\_socket, left\_socket, right\_socket, left\_pid, right\_pid, id, request\_string);

} else if (command == "ping") {

ping(command\_stream, parent\_socket, left\_socket, right\_socket, left\_pid, right\_pid, id, request\_string);

} else if (command == "kill\_children") {

kill\_children(parent\_socket, left\_socket, right\_socket, left\_pid, right\_pid);

}

if (parent\_port == 0) {

break;

}

}

return 0;

}

**Пример работы**

zhavkk@zhavkk:~/OSlabs/lab5\_7/mylab$ ./server

Commands:

1. create (id)

2. exec (id) (name, value)

3. kill (id)

4. pingall

6. exit

create 1

Ok:224578

create 2

Ok:224583

create 100

Ok:224603

create 50

Ok:224609

create 75

Ok:224629

create 70

Ok:224635

kill 70

Ok

exec 10 myvar

Error:10: Not found

exec 100 myvar

Ok:100: 'myvar' not found

exec 100 myvar 100

Ok:100

exec 100 myvar

Ok:100:100

exec 100 myvar 300

Ok:100

exec 100 myvar

Ok:100:300

exec 100 myvar2 301

Ok:100

exec 100 myvar

Ok:100:300

exec 100 myvar2

Ok:100:301

exec 50 myvar 299

Ok:50

exec 50 myvar

Ok:50:299

ping 50

Ok: 1

ping 834

Error:834: Not found

ping 100

Ok: 1

pingall

Ok: 70

exit

**Вывод**

В результате выполнения лабораторной работы мной были приобретены практические навыки в управлении серверами сообщений, применении отложенных вычислений и интеграции программных систем друг с другом.

Я познакомился с технологией очередей сообщений и изучил библиотеку ZeroMQ.

В результате лабораторной работы мной была реализована распределенная система по асинхронной обработке запросов в соответствии с заданием.