Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6 по курсу**

**«Операционные системы»**

Студент: Соколов Арсений Игоревич

Группа: М8О-207Б-21

Вариант: 16

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Демонстрация работы программы
6. Выводы

**Репозиторий**

https://github.com/valerasaray/os

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Управлении серверами сообщений (№6)
* Применение отложенных вычислений (№7)
* Интеграция программных систем друг с другом (№8)

**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. Список основных поддерживаемых команд:

* **Создание нового вычислительного узла** (Формат команды: create id [parent])
* **Исполнение команды на вычислительном узле** (Формат команды: exec id [params])
* **Проверка доступности узла** (Формат команды: ping id)
* **Удаление узла** (Формат команды remove id)

**Вариант №16:** топология - дерево общего вида, команда - локальный целочисленный словарь, проверка доступности - ping id.

**Общие сведения о программе**

Программа состоит из 3 файлов: client.cpp — интерфейс для общения с пользователем, server.cpp — исполнение команд (вычислительный узел), labtools.h — класс топологии, управление сообщениями и разные константы.

**Исходный код**

**labtools.h**

**#ifndef \_\_LABTOOLS\_H\_\_**

**#define \_\_LABTOOLS\_H\_\_**

**#include <iostream>**

**#include <map>**

**#include <vector>**

**#include <set>**

**#include <zmq.hpp>**

**#include <nlohmann/json.hpp>**

***// проверка на ошибку***

**#define CHECK\_ERROR(*expr*, *err*, *message*) \**

**do { \**

**auto \_\_result = (expr); \**

**if (\_\_result == err) { \**

**fprintf(stderr, "Error: %s\n", message); \**

**fprintf(stderr, "errno = %s, file %s, line %d\n", strerror(errno), \**

**\_\_FILE\_\_, \_\_LINE\_\_); \**

**exit(-1); \**

**} \**

**} while (0)**

**const int WAIT\_TIME = 1000; *// время ожидания при сетевом взаимодействии***

**const int PORT = 7000; *// номер порта для сетевого взаимодействия***

**class NTree**

**{**

**public:**

***// вложенный класс NTree (отображение ключа на множество дочерних узлов)***

**using Node = std::map<int, std::set<int>>;**

**Node node;**

**NTree();**

**void print(); *// текущее состояние дерева***

**int dfs(int *child*, int *curChild*); *// обход дерева в глубину***

**int findCheck(int *parent*, int *child*); *// проверка на существование связи между двумя узлами***

**int find(int *parent*, int *child*);**

**std::pair<int, int> findNode(int *node*); *// возврат пары родительский и дочерний узео***

**std::vector<int> findChilds(int *node*); *// вывод дочерних узлов в виде вектора***

**int insert(int *parent*, int *child*); *// создание связи между узлами***

**int erase(int *parent*, int *child*); *// удаление связи***

**void destroyUndertree(int *node*); *// удаление из дерева поддерева***

**~NTree();**

**};**

***// конструктор класса, который инициализирует дерево пустым множеством корневых узлов***

**NTree::NTree()**

**{**

**std::set<int> root;**

**this->node.insert({-1, root});**

**}**

***// деструктор класса***

**NTree::~NTree()**

**{**

**}**

***// вывод дерева на консоль***

**void NTree::print()**

**{**

**for (auto it = this->node.begin(); it != this->node.end(); it++) {**

**std::cout << it->first << ": ";**

**for (auto it1 = it->second.begin(); it1 != it->second.end(); it1++) {**

**std::cout << \*it1 << " ";**

**}**

**std::cout << "\n";**

**}**

**}**

***// обход дерева в глубину***

**int NTree::dfs(int *child*, int *curChild*) {**

**for (auto i: this->node[*curChild*]) {**

***// std::cout << i << "\n";***

**if (i == *child*) {**

**return 1;**

**}**

**return NTree::dfs(*child*, i);**

**}**

**return -1;**

**}**

***// поиск***

**int NTree::find(int *parent*, int *child*)**

**{**

**int ans = 0;**

**for (auto curChild: this->node[*parent*]) {**

**if (curChild == *child*) {**

**return ans;**

**} else if (dfs(*child*, curChild) != -1) {**

**return ans;**

**}**

**ans++;**

**}**

**return -1;**

**}**

***// поиск узлов***

**std::pair<int, int> NTree::findNode(int *node*)**

**{**

**for (auto i: this->node) {**

**if (i.second.find(*node*) != i.second.cend()) {**

**return std::make\_pair(i.first, 0);**

**}**

**}**

**return std::make\_pair(-1, -1);**

**}**

***// поиск дочерних узлов***

**std::vector<int> NTree::findChilds(int *node*)**

**{**

**std::vector<int> res;**

**for (auto i: this->node[*node*]) {**

**res.push\_back(i);**

**}**

**return res;**

**}**

**int NTree::findCheck(int *parent*, int *child*)**

**{**

**for (auto curChild: this->node[*parent*]) {**

**if (curChild == *child*) {**

**return 1;**

**}**

**}**

**return -1;**

**}**

**int NTree::insert(int *parent*, int *child*)**

**{**

**auto curParent = this->node.find(*parent*);**

**if (curParent != this->node.cend()) {**

**auto curChild = curParent->second.find(*child*);**

**if (curChild != curParent->second.cend()) {**

**return 0;**

**}**

**curParent->second.insert(*child*);**

**std::set<int> childVec;**

**this->node.insert({*child*, childVec});**

**return 1;**

**}**

**return 0;**

**}**

**int NTree::erase(int *parent*, int *child*)**

**{**

**auto curParent = this->node.find(*parent*);**

**if (curParent != this->node.cend()) {**

**auto curChild = curParent->second.find(*child*);**

**if (curChild != curParent->second.cend()) {**

**curParent->second.erase(*child*);**

**auto p = this->node.find(*child*);**

**this->node.erase(p);**

**return 1;**

**}**

**}**

**return 0;**

**}**

**void NTree::destroyUndertree(int *node*)**

**{**

**auto curNode = this->node.find(*node*);**

**if (curNode != this->node.cend()) {**

**for (auto it: curNode->second) {**

**this->destroyUndertree(it);**

**}**

**}**

**curNode->second.clear();**

**int parent = this->findNode(*node*).first;**

**auto parentN = this->node.find(parent);**

**parentN->second.erase(*node*);**

**this->node.erase(*node*);**

**}**

**namespace advancedZMQ**

**{**

**nlohmann::json sendAndRecv(nlohmann::json &*request*, zmq::socket\_t &*socket*, int *debug*)**

**{**

**std::string strFromJson = *request*.dump();**

**if (*debug*) {**

**std::cout << strFromJson << std::endl;**

**}**

**zmq::message\_t msg(strFromJson.size());**

**memcpy(msg.data(), strFromJson.c\_str(), strFromJson.size());**

***socket*.send(msg);**

**nlohmann::json reply;**

**zmq::message\_t msg2;**

***socket*.recv(msg2);**

**std::string strToJson(static\_cast<char \*> (msg2.data()), msg2.size());**

**if (!strToJson.empty()) {**

**reply = nlohmann::json::parse(strToJson);**

**} else {**

**if (*debug*) {**

**std::cout << "bad socket" << std::endl;**

**}**

**reply["ans"] = "error";**

**}**

**if (*debug*) {**

**std::cout << reply << "\n";**

**}**

**return reply;**

**}**

**void Send(nlohmann::json &*request*, zmq::socket\_t &*socket*)**

**{**

**std::string strFromJson = *request*.dump();**

**zmq::message\_t msg(strFromJson.size());**

**memcpy(msg.data(), strFromJson.c\_str(), strFromJson.size());**

***socket*.send(msg);**

**}**

**nlohmann::json Recv(zmq::socket\_t &*socket*)**

**{**

**nlohmann::json reply;**

**zmq::message\_t msg;**

***socket*.recv(msg);**

**std::string strToJson(static\_cast<char \*> (msg.data()), msg.size());**

**reply = nlohmann::json::parse(strToJson);**

**return reply;**

**}**

**}**

**#endif**

**client.cpp**

**#include <iostream>**

**#include <map>**

**#include <vector>**

**#include <string>**

**#include <string.h>**

**#include <unistd.h>**

**#include <csignal>**

**#include <thread>**

**#include <zmq.hpp>**

**#include <nlohmann/json.hpp>**

**#include "../include/labtools.h"**

***// использование пространства имён advancedZMQ***

**using namespace advancedZMQ;**

***// объявление объекта nodes класса NTree***

**NTree nodes;**

**int main(int *argc*, char const \**argv*[])**

**{**

***// создание нового сокета ZeroMQ для связи между процессами***

**zmq::context\_t context(1);**

**zmq::socket\_t socket(context, zmq::socket\_type::pair);**

**socket.setsockopt(ZMQ\_RCVTIMEO, WAIT\_TIME);**

**socket.setsockopt(ZMQ\_SNDTIMEO, WAIT\_TIME);**

***// привязка сокета к адресу tcp://127.0.0.1:PORT***

**socket.bind(("tcp://127.0.0.1:" + std::to\_string(PORT)));**

**int pid = fork(); *// создание дочернего процесса***

***// проверка: является ли процесс дочерним***

**if (pid == 0) {**

***// вызов исполняемого файла server***

***// execl() заменяет текущий процесс новым процессом, который запускает "server"***

**execl("./server", "./server", std::to\_string(-1).c\_str(), NULL);**

**return 0;**

**}**

***// вывод списка команд***

**std::cout << "create [id] [parentId] создать узел\n";**

**std::cout << "remove [id] [parentId] удалить узел\n";**

**std::cout << "exec [id] [start] начать измерение времени на узле\n";**

**std::cout << "exec [id] [stop] закончить измерение времени на узле\n";**

**std::cout << "exec [id] [time] посмотреть измеренное время\n";**

**std::cout << "ping [id] проверить наличие узла\n";**

**std::cout << "print показать дерево\n";**

**std::cout << "help чтобы показать команды снова\n\n";**

**std::string command;**

***// бесконечный цикл ввода команд пользователем***

**while (std::cin >> command) {**

**if (command == "create") {**

**int id; *// дочерний узел, который нужно создать***

**int parentId; *// родительский узел, к которому нужно присоединить новый узел***

**std::cin >> id >> parentId;**

***// проверка на существование родительского узла с заданным parentId***

**if (nodes.findNode(parentId).second == -1 && parentId != -1) {**

***// вывод ошибки, если он не существует***

**std::cout << "Ошибка: родительский узел номер " << parentId << " не существует" << std::endl;**

**continue;**

**}**

***// проверка на существование дочернего узла с заданным id***

**if (nodes.find(parentId, id) != -1) {**

**std::cout << "EОшибка: дочерний узел номер " << id << " уже существует" << std::endl;**

**continue;**

**}**

***// создание json запроса pingRequest для проверки доступности родительского узла***

**nlohmann::json pingRequest;**

***// содержание запроса:***

**pingRequest["type"] = "ping";**

**pingRequest["id"] = parentId;**

***// использование функции sendAndRecv для отправки pingRequest на сервер***

***// и получения ответа pingReply***

**nlohmann::json pingReply = sendAndRecv(pingRequest, socket, 0);**

***// проверка на содердание в ответе строки "ok" (родительский узел доступен)***

**if (pingReply["ans"] != "ok") {**

**socket.close(); *// закрытие сокета***

**context.close();**

**std::cout << "Ошибка: родительский узел номер " << parentId << " недоступен" << std::endl;**

**continue;**

**}**

***// создание json запроса request***

**nlohmann::json request;**

***// содержание запроса***

**request["type"] = "create";**

**request["id"] = id;**

**request["parentId"] = parentId;**

***// отправление на сервер запроса с помощью функции sendAndRecv,***

***// которая возвращает ответ сервера в виде запроса reply***

**nlohmann::json reply = sendAndRecv(request, socket, 0);**

***// проверка на содержания строки "ok" в поле "ans"***

***// (новый узел успешно добавлен к родительскому узлу)***

**if (reply["ans"] != "ok") {**

**socket.close(); *// закрытие сокета***

**context.close();**

**std::cout << "Ошибка: родительский узел номер " << parentId << " недоступен" << std::endl;**

**continue;**

**} else {**

***// id нового узла добавляется в дерево.***

**nodes.insert(parentId, id);**

**}**

**} else if (command == "remove") {**

**int id;**

**int parentId;**

**std::cin >> id;**

***// проверка - существует ли узел с таким идентификатором в дереве***

**if (nodes.findNode(id).second == -1) {**

**std::cout << "Ошибка узел номер " << id << " не найден" << std::endl;**

**continue;**

**}**

***// определение идентификатора родительского узла***

**parentId = nodes.findNode(id).first;**

***// создание json запроса pingRequest для проверки доступности родительского узла***

**nlohmann::json pingRequest;**

**pingRequest["type"] = "ping";**

**pingRequest["id"] = parentId;**

***// отправление на сервер запроса с помощью функции sendAndRecv,***

***// которая возвращает ответ сервера в виде запроса reply***

**nlohmann::json pingReply = sendAndRecv(pingRequest, socket, 0);**

***// проверка на содердание в ответе строки "ok" (родительский узел доступен)***

**if (pingReply["ans"] != "ok") {**

**std::cout << "Ошибка: родительский узел номер " << parentId << " недоступен" << std::endl;**

**continue;**

**}**

***// создание json запроса request***

**nlohmann::json request;**

***// формируется запрос на удаление***

**request["type"] = "remove";**

***// идентификатор элемента***

**request["id"] = id;**

**request["parentId"] = parentId;**

***// запрос на удаление***

**nlohmann::json reply = sendAndRecv(request, socket, 0);**

**if (reply["ans"] != "ok") {**

**std::cout << "Ошибка узел номер " << id << " недоступен" << std::endl;**

**continue;**

**} else {**

***// удаление***

**nodes.destroyUndertree(id);**

**std::cout << "ok" << std::endl;**

**}**

**} else if (command == "print") {**

***// вывод дерева***

**nodes.print();**

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

**int id, curInd;**

**std::string string;**

**std::cin >> id >> string;**

**if (nodes.findNode(id).second != -1) {**

**nlohmann::json request;**

**request["type"] = "exec";**

**request["id"] = id;**

**request["command"] = string;**

**nlohmann::json reply = sendAndRecv(request, socket, 0);**

**if (reply["ans"] != "ok") {**

**std::cout << "Ошибка узел номер " << id << " недоступен" << std::endl;**

**}**

**} else {**

**std::cout << "Ошибка узел номер " << id << " не найден" << std::endl;**

**}**

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

**int id = -1;**

**std::cin >> id;**

**nlohmann::json request;**

**request["type"] = "ping";**

**request["id"] = id;**

**nlohmann::json reply = sendAndRecv(request, socket, 0);**

**if (reply["ans"] == "ok") {**

**std::cout << "ok:" << id << std::endl;**

**} else {**

**std::cout << "Ошибка узел номер " << id << " недоступен" << std::endl;**

**}**

**} else if (command == "help") {**

**std::cout << "\ncreate [id] [parentId] создать узел\n";**

**std::cout << "\nremove [id] [parentId] удалить узел\n";**

**std::cout << "\nexec [id] [start] начать измерение времени на узле\n";**

**std::cout << "\nexec [id] [stop] закончить измерение времени на узле\\n";**

**std::cout << "\nexec [id] [time] посмотреть измеренное время\n";**

**std::cout << "\nping [id] проверить наличие узла\n";**

**std::cout << "\nprint показать дерево\n";**

**std::cout << "\nhelp чтобы показать команды снова\n\n";**

**}**

**}**

***// проверка, достигнут ли конец файла***

**if (std::cin.eof()) {**

***// создание объекта JSON для запроса на удаление узла***

**nlohmann::json destroyRequest;**

**destroyRequest["type"] = "remove";**

***// поиск всех дочерних узлов корня дерева и сохранение их вектора целых чисел***

**std::vector<int> nodesRoot = nodes.findChilds(-1);**

***// цикл, в котором отправляются запросы на удаление всех дочерних узлов корня.***

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

**destroyRequest["id"] = nodesRoot[i];**

**sendAndRecv(destroyRequest, socket, 0);**

**}**

**}**

**socket.close();**

**return 0;**

**}**

**server.cpp**

**#include <iostream>**

**#include <map>**

**#include <vector>**

**#include <string>**

**#include <string.h>**

**#include <unistd.h>**

**#include <csignal>**

**#include <thread>**

**#include <zmq.hpp>**

**#include <nlohmann/json.hpp>**

**#include "../include/labtools.h"**

**#include <chrono>**

***// Замер времени выполнения операций***

**class Timer {**

**public:**

***// время начала измерения времени выполнения***

**void start() {**

**m\_startTime = std::chrono::high\_resolution\_clock::now();**

**m\_running = true;**

**}**

***// время окончания измерения времени выполнения***

**void stop() {**

**m\_endTime = std::chrono::high\_resolution\_clock::now();**

**m\_running = false;**

**}**

***// время выполнения***

**double time() {**

**std::chrono::time\_point<std::chrono::high\_resolution\_clock> endTime;**

**if (m\_running) {**

**endTime = std::chrono::high\_resolution\_clock::now();**

**} else {**

**endTime = m\_endTime;**

**}**

**return std::chrono::duration\_cast<std::chrono::milliseconds>(endTime - m\_startTime).count();**

**}**

**private:**

**std::chrono::time\_point<std::chrono::high\_resolution\_clock> m\_startTime;**

**std::chrono::time\_point<std::chrono::high\_resolution\_clock> m\_endTime;**

**bool m\_running = false;**

**};**

**using namespace advancedZMQ;**

**std::vector<zmq::socket\_t> childSockets;**

**std::vector<int> childIds;**

**int main(int *argc*, char const \**argv*[])**

**{**

**int curId = std::stoi(std::string(argv[1])), flag = 1;**

***// Инициализация контекста ZeroMQ с одним потоком выполнения***

**zmq::context\_t parentContext(1);**

***// Создание ZeroMQ-сокета типа pair с использованием контекста parentContext***

**zmq::socket\_t parentSocket(parentContext, zmq::socket\_type::pair);**

***// Подключение созданного сокета parentSocket к адресу "tcp://127.0.0.1:(PORT + curId + 1)***

**parentSocket.connect(("tcp://127.0.0.1:" + std::to\_string(PORT + curId + 1)));**

***// Создание объекта Timer для измерения времени***

**Timer timer;**

**void \*arg = NULL;**

**while (flag) {**

***// Получение JSON-сообщения от parentSocket***

**nlohmann::json reply = Recv(parentSocket);**

***// Создание нового JSON-сообщения request***

**nlohmann::json request;**

**request["type"] = reply["type"];**

**request["id"] = curId;**

**request["pid"] = 0;**

**request["ans"] = "error";**

**if (reply["type"] == "ping") {**

***// проверка на совпадение id***

**if (reply["id"] == curId) {**

***// формирование ответного сообщения***

**request["ans"] = "ok";**

**request["id"] = curId;**

**request["pid"] = getpid();**

**} else {**

***// обход вектора дочерних сокетов и отправление на каждый новое сообщение типа "ping"***

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

**nlohmann::json pingRequest;**

**pingRequest["type"] = "ping";**

**pingRequest["id"] = reply["id"];**

**nlohmann::json pingReply = sendAndRecv(pingRequest, childSockets[i], 0);**

**if (pingReply["ans"] == "ok") {**

***// формирование ответного сообщения***

**request["ans"] = "ok";**

**request["id"] = curId;**

**request["pid"] = getpid();**

**break;**

**}**

**}**

**}**

**} else if (reply["type"] == "create") {**

**if (reply["parentId"] == curId) {**

**int i = reply["id"];**

***// Создается новый сокет для взаимодействия с созданным процессом***

**zmq::socket\_t childSocket(parentContext, zmq::socket\_type::pair);**

***// задаются опции сокета для установки времени ожидания приема и отправки сообщений***

**childSocket.setsockopt(ZMQ\_RCVTIMEO, WAIT\_TIME);**

**childSocket.setsockopt(ZMQ\_SNDTIMEO, WAIT\_TIME);**

***// Устанавливается соединение между сокетами родительского и созданного процессов.***

**childSocket.bind(("tcp://\*:" + std::to\_string(PORT + i + 1)).c\_str());**

***// создается новый процесс***

**int pid = fork();**

**if (pid == 0) {**

**int i = reply["id"];**

***// запуск исполняемого файла "server"***

**execl("./server", "./server", std::to\_string(i).c\_str(), NULL);**

***// завершение работы***

**return 0;**

**} else {**

***// Создание JSON-объекта pingRequest***

**nlohmann::json pingRequest;**

**pingRequest["type"] = "ping";**

**pingRequest["id"] = reply["id"];**

***// Отправка созданного запроса на сокет childSocket***

***// Ожидание ответа в течение 0 миллисекунд с помощью функции sendAndRecv***

**nlohmann::json pingReply = sendAndRecv(pingRequest, childSocket, 0);**

**if (pingReply["ans"] == "ok") {**

***// Вывод на экран значения поля "pid***

**std::cout << "OK: " << pingReply["pid"] << std::endl;**

**int i = reply["id"];**

***// Добавление сокета childSocket в вектор childSockets***

**childSockets.push\_back(std::move(childSocket));**

***// Добавление идентификатора reply["id"] в вектор childIds***

**childIds.push\_back(reply["id"]);**

***// Установка значения поля "ans" в "ok" в JSON-объекте request***

**request["ans"] = "ok";**

***// Установка значения поля "parentId" в идентификатор родительского процесса***

**request["parentId"] = reply["parentId"];**

**}**

**}**

**} else {**

***// Создание нового JSON-объекта newRequest, который получает те же поля, что и reply***

**nlohmann::json newRequest = reply;**

***// Запуск цикла по всем имеющимся дочерним сокетам в childSockets***

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

**nlohmann::json newReply = sendAndRecv(newRequest, childSockets[i], 0);**

***// Если ответ содержит поле "ans", равное "ok"***

**if (newReply["ans"] == "ok") {**

***// то устанавливается значение поля "ans" в исходном запросе request равным "ok"***

**request["ans"] = "ok";**

**break;**

**}**

**}**

**}**

**} else if (reply["type"] == "remove") {**

***// количество удаленных дочерних процессов***

**int c = 0;**

***// цикл по всем дочерним сокетам***

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

***// отправляем каждому из них сообщение типа "destroy" с идентификатором удаляемого процесса***

**int childId = childIds[i];**

**nlohmann::json destroyRequest;**

**destroyRequest["type"] = "destroy";**

***// Если идентификатор найден и удаление прошло успешно***

**if (childId == reply["id"]) {**

***// удаляем соответствующий дочерний сокет и идентификатор из векторов childSockets и childIds***

**Send(destroyRequest, childSockets[i]);**

**int k = reply["id"];**

**childSockets.erase(std::next(childSockets.begin() + i));**

**childIds.erase(childIds.begin() + i);**

**c++;**

**break;**

**}**

**}**

***// Если был удален хотя бы один дочерний процесс***

**if (c > 0) {**

***// устанавливаем ответ "ok" в сообщении.***

**request["ans"] = "ok";**

**c = 0;**

**}**

***// Отправка оставшимся дочерним процессам полученное сообщение для проверки корректности удаления.***

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

**nlohmann::json newReply = sendAndRecv(reply, childSockets[i], 0);**

**if (newReply["ans"] == "ok") {**

**request["ans"] = "ok";**

**}**

**}**

**} else if (reply["type"] == "destroy") {**

**nlohmann::json newRequest = reply;**

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

**int childId = childIds[i];**

**Send(newRequest, childSockets[i]);**

**}**

**flag = 0;**

**} else if (reply["type"] == "exec") {**

***// если идентификатор, указанный в запросе, совпадает с идентификатором текущего процесса***

**if (reply["id"] == curId) {**

**if (reply["command"] == "start") {**

***// запуск таймера***

**timer.start();**

**std::cout << "Ok:" << reply["id"] << std::endl;**

**request["ans"] = "ok";**

**} else if (reply["command"] == "time") {**

***// получаем текущее время таймера***

**std::cout << "Ok:" << reply["id"] << ":" << timer.time() << std::endl;**

**request["ans"] = "ok";**

**} else if (reply["command"] == "stop") {**

***// останавка таймера***

**timer.stop();**

**std::cout << "Ok:" << reply["id"] << std::endl;**

**request["ans"] = "ok";**

**} else {**

***// ошибка***

**request["ans"] = "error";**

**}**

**} else {**

**nlohmann::json newRequest = reply;**

***// цикл по всем дочерним сокетам***

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

***// Отправка запроса newRequest и ожидание ответа sendAndRecv(). Результат сохраняется в переменную newReply***

**nlohmann::json newReply = sendAndRecv(newRequest, childSockets[i], 0);**

**if (newReply["ans"] == "ok") {**

**request["ans"] = "ok";**

**break;**

**}**

**}**

**}**

**}**

***// отправляет сообщение request на сокет***

**Send(request, parentSocket);**

**}**

***// закрывает все сокеты, используемые для связи с дочерними процессами.***

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

**childSockets[i].close();**

**}**

**parentSocket.close();**

**return 0;**

**}**

**Демонстрация работы программы**

**create 1 -1**

**OK: 4075**

**create 2 1**

**OK: 4079**

**ping 2**

**ok:2**

**ping 0**

**Ошибка узел номер 0 недоступен**

**exec 2 start**

**Ok:2**

**print**

**-1: 1**

**1: 2**

**2:**

**create 3 2**

**OK: 4139**

**exec 2 stop**

**Ok:2**

**exec 2 time**

**Ok:2:29389**

**remove 2**

**ok**

**print**

**-1: 1**

**1:**

**Выводы**

Благодаря данной лабораторной работе я познакомился с сокетами ZeroMQ и библиотекой nlohmann/json.hpp, а также приобрел практические навыки в управлении серверами сообщений и применении отложенных вычислений