Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5-7 по курсу**

**«Операционные системы»**

Студент: Рамалданов Рустамхан Ражудинович

Группа: М8О-208Б-22

Вариант: 14

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Демонстрация работы программы

7. Выводы

**Репозиторий**

<https://github.com/pepelulka/OS_labs>

**Постановка задачи**

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.

**Общие сведения о программе**

Программа написана для операционной системы Linux. Для реализации очереди сообщений была использована библиотека ZeroMQ. Были написаны две программы – для управляющего узла и для вычислительного узла.

**Общий метод и алгоритм решения**

У управляющего узла есть один собирающий PULL сокет, в который узлы скидывают результат своей работы. Для каждого дочернего вычислительного узла имеется по PUSH сокету. У каждого вычислительного узла есть 4 сокета – 2 PUSH и 2 PULL – по паре для общения с верхними вершинами и с нижними. Управляющий узел отправляет сообщения дочерним вершинам, а они в свою очередь пересылают сообщение дальше, если оно адресовано не им, либо выполняют необходимые действия, если сообщения были отправлены им.

**Исходный код**

**========================== base.hpp ==========================**

**#pragma once**

**#include <iostream>**

**#include <thread>**

**#include <mutex>**

**#include <condition\_variable>**

**#include <chrono>**

**#include <unistd.h>**

**namespace lab5 {**

**using namespace std::chrono\_literals;**

**using NodeId = size\_t;**

**using Port = size\_t;**

**const Port LOWER\_PORT = 10555;**

**const Port UPPER\_PORT = 10600;**

**const auto FIRST\_PING\_TIME = 2000ms;**

**const auto SECOND\_PING\_TIME = 500ms;**

**// For ping we use message "r" for descend and "p" for ascend**

**int CreateProcess(const std::string &path, NodeId id, Port pullPort, Port pushPort);**

**}**

**========================== computer\_base.hpp ==========================**

**#pragma once**

**#include <iostream>**

**#include <sstream>**

**#include "base.hpp"**

**#include "zmq.hpp"**

**namespace lab5 {**

**class TComputerNode {**

**private:**

**NodeId id;**

**Port pullPort = 0, pushPort = 0, pushPortBottom = 0;**

**zmq::context\_t context;**

**std::string path;**

**zmq::socket\_t pullSocket, pushSocket, pushSocketBottom;**

**bool haveChild = false;**

**public:**

**TComputerNode(const std::string& \_path, NodeId \_id, Port \_pullPort, Port \_pushPort);**

**void CreateChild(NodeId id1, Port bPullPort, Port bPushPort);**

**void Routine();**

**};**

**}**

**========================== manager\_base.hpp ==========================**

**#pragma once**

**#include <iostream>**

**#include <set>**

**#include <map>**

**#include <queue>**

**#include <sstream>**

**#include <stack>**

**#include "base.hpp"**

**#include "wqueue.hpp"**

**#include "zmq.hpp"**

**namespace lab5 {**

**class TMainManager;**

**enum TTaskType {**

**CREATE, // create node**

**EXEC, // exec**

**RESULT, // result from sink**

**PING // stop**

**};**

**struct TTask {**

**TTaskType type;**

**NodeId id;**

**// for create**

**int parent;**

**// for exec**

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

**// for result**

**std::string result;**

**static TTask MakeCreate(NodeId id, int parent);**

**static TTask MakeExec(NodeId id, const std::vector<int>& data);**

**static TTask MakeResult(const std::string &result);**

**static TTask MakePing();**

**};**

**using TTaskQueue = TWQueue<TTask>;**

**class TSink {**

**private:**

**TTaskQueue &tq;**

**zmq::context\_t context;**

**zmq::socket\_t inputSocket;**

**//**

**bool pingingMode = false;**

**std::stack<NodeId> pingingResult; // Contains result of pinging, vector of nodeId's that is availible**

**std::mutex mtx; // pingingResult & pingingMode mutex**

**//**

**Port port;**

**public:**

**friend class TMainManager;**

**std::set<NodeId> GetPingingResultAndFinishPinging();**

**void StartPinging();**

**TSink(TTaskQueue& \_tq, Port \_port);**

**void Routine();**

**};**

**class TFrontend {**

**private:**

**std::queue<std::string> q;**

**std::mutex mtx;**

**TTaskQueue \* tq;**

**void Handle(const std::string &line);**

**public:**

**friend class TMainManager;**

**void PushResult(const std::string &msg);**

**void Routine();**

**};**

**class TPortPool {**

**private:**

**inline static Port lastPort = LOWER\_PORT - 1;**

**public:**

**TPortPool() = delete;**

**static Port get();**

**};**

**// Manage nodes (and can push into them)**

**class TNodeStructure {**

**private:**

**zmq::context\_t context;**

**std::string path;**

**std::set<NodeId> nodes;**

**std::map<NodeId, std::set<NodeId>> childSets;**

**std::map<NodeId, zmq::socket\_t> childSockets;**

**std::map<NodeId, Port> childPorts;**

**Port sinkPort;**

**TFrontend &front;**

**TSink &sink;**

**zmq::socket\_t sinkSocket;**

**public:**

**friend class TMainManager;**

**TNodeStructure() = delete;**

**TNodeStructure(const std::string &\_path,**

**Port \_sinkPort,**

**TFrontend &\_front,**

**TSink &\_sink) : context(1), path(\_path), sinkPort(\_sinkPort), front(\_front), sink(\_sink), sinkSocket(context, zmq::socket\_type::push) {**

**sinkSocket.connect("tcp://localhost:" + std::to\_string(sinkPort));**

**}**

**void AddNewChild(NodeId id);**

**void SendCreate(NodeId id, int parent);**

**void SendExec(NodeId id, const std::vector<int>& a);**

**void SendPing();**

**};**

**// TMainManager stuff:**

**class TMainManager {**

**private:**

**TTaskQueue wq;**

**Port sinkPort;**

**public:**

**TSink sink;**

**private:**

**TNodeStructure ns;**

**TFrontend &frontend;**

**public:**

**friend class TSink;**

**friend class TNodeStructure;**

**friend class TFrontend;**

**TMainManager(const std::string &path, TFrontend &front);**

**void Routine();**

**};**

**}**

**========================== wqueue.hpp ==========================**

**#pragma once**

**#include <iostream>**

**#include <thread>**

**#include <mutex>**

**#include <condition\_variable>**

**#include <queue>**

**namespace lab5 {**

**template<class T>**

**class TWQueue {**

**private:**

**std::queue<T> q;**

**std::mutex mtx;**

**std::condition\_variable cv;**

**public:**

**void push(const T& el) {**

**{**

**std::unique\_lock l(mtx);**

**q.push(el);**

**}**

**cv.notify\_one();**

**}**

**T pop() {**

**std::unique\_lock l(mtx);**

**if (q.empty()) {**

**cv.wait(l);**

**}**

**T result = q.front();**

**q.pop();**

**return result;**

**}**

**size\_t size() {**

**std::unique\_lock l(mtx);**

**return q.size();**

**}**

**};**

**}**

**========================== base.cpp ==========================**

**#include "base.hpp"**

**namespace lab5 {**

**int CreateProcess(const std::string &path, NodeId id, Port pullPort, Port pushPort) {**

**int pid = fork();**

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

**execl(path.c\_str(), path.c\_str(), std::to\_string(id).c\_str(), std::to\_string(pullPort).c\_str(), std::to\_string(pushPort).c\_str(), nullptr);**

**}**

**return pid;**

**}**

**}**

**========================== computer\_base.cpp ==========================**

**#include "computer\_base.hpp"**

**namespace lab5 {**

**TComputerNode::TComputerNode(const std::string& \_path, NodeId \_id, Port \_pullPort, Port \_pushPort) : id(\_id), pullPort(\_pullPort), pushPort(\_pushPort), context(1), path(\_path),**

**pullSocket(context, zmq::socket\_type::pull),**

**pushSocket(context, zmq::socket\_type::push),**

**pushSocketBottom(context, zmq::socket\_type::push) {**

**pushSocket.connect("tcp://localhost:" + std::to\_string(pushPort));**

**pullSocket.bind("tcp://localhost:" + std::to\_string(pullPort));**

**}**

**void TComputerNode::CreateChild(NodeId id1, Port bPullPort, Port bPushPort) {**

**if (haveChild) {**

**return ;**

**}**

**pushPortBottom = bPullPort;**

**pushSocketBottom.connect("tcp://localhost:" + std::to\_string(pushPortBottom));**

**haveChild = true;**

**CreateProcess(path, id1, bPullPort, bPushPort);**

**}**

**void TComputerNode::Routine() {**

**pushSocket.send(zmq::message\_t("OK: " + std::to\_string(getpid())), zmq::send\_flags::none);**

**while (true) {**

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

**auto ret = pullSocket.recv(msg, zmq::recv\_flags::none);**

**if (ret.has\_value() && (EAGAIN == ret.value())) {**

**throw std::logic\_error("Something bad...");**

**}**

**std::string content((char\*)msg.data(), (char\*)msg.data() + msg.size());**

**std::stringstream ss(content);**

**std::string command;**

**ss >> command;**

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

**size\_t id1, parent, port1;**

**ss >> id1 >> parent >> port1;**

**if (parent == id) {**

**CreateChild(id1, port1, pullPort);**

**} else {**

**pushSocketBottom.send(msg, zmq::send\_flags::none);**

**}**

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

**size\_t id1, n;**

**ss >> id1;**

**if (id != id1) {**

**pushSocketBottom.send(msg, zmq::send\_flags::none);**

**continue;**

**} else {**

**ss >> n;**

**int sum = 0;**

**for (size\_t i = 0; i < n;i++) {**

**int cur;**

**ss >> cur;**

**sum += cur;**

**}**

**zmq::message\_t msg("OK:" + std::to\_string(id) + ": " + std::to\_string(sum));**

**pushSocket.send(msg, zmq::send\_flags::none);**

**}**

**} else if (command[0] == 'r') {**

**if (pushPortBottom == 0) {**

**pushSocket.send(zmq::message\_t("p " + std::to\_string(id)), zmq::send\_flags::none);**

**} else {**

**pushSocketBottom.send(zmq::message\_t(std::string("r")), zmq::send\_flags::none);**

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

**std::vector<zmq::pollitem\_t> items = {**

**{ static\_cast<void\*>(pullSocket), 0, ZMQ\_POLLIN, 0 }**

**};**

**int pollRes = zmq::poll(items, SECOND\_PING\_TIME);**

**if (pollRes == 0) {**

**pushSocket.send(zmq::message\_t("p " + std::to\_string(id)), zmq::send\_flags::none);**

**} else {**

**zmq::message\_t msg1;**

**auto ret = pullSocket.recv(msg1, zmq::recv\_flags::none);**

**if (ret.has\_value() && (EAGAIN == ret.value())) {**

**throw std::logic\_error("Something bad...");**

**}**

**pushSocket.send(zmq::message\_t(msg1.to\_string() + " " + std::to\_string(id)), zmq::send\_flags::none);**

**}**

**}**

**} else if (command[0] == 'p') {**

**content += " " + std::to\_string(id);**

**pushSocket.send(zmq::message\_t(content), zmq::send\_flags::none);**

**} else {**

**pushSocket.send(msg, zmq::send\_flags::none);**

**}**

**}**

**}**

**}**

**========================== computer.cpp ==========================**

**#include <iostream>**

**#include <sstream>**

**#include "base.hpp"**

**#include "computer\_base.hpp"**

**#include "zmq.hpp"**

**using namespace lab5;**

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

**if (argc < 4) {**

**std::cerr << "Missing arguments!\n";**

**exit(EXIT\_FAILURE);**

**}**

**NodeId id = std::atoi(argv[1]);**

**Port pullPort = std::atoi(argv[2]), pushPort = std::atoi(argv[3]);**

**TComputerNode node(std::string(getenv("PATH\_TO\_COMP")), id, pullPort, pushPort);**

**node.Routine();**

**}**

**========================== manager\_base.cpp ==========================**

**#include "manager\_base.hpp"**

**namespace lab5 {**

**TTask TTask::MakeCreate(NodeId id, int parent) {**

**return TTask{TTaskType::CREATE, id, parent, {}, {}};**

**}**

**TTask TTask::MakeExec(NodeId id, const std::vector<int>& data) {**

**return TTask{TTaskType::EXEC, id, 0, data, {}};**

**}**

**TTask TTask::MakePing() {**

**return TTask{TTaskType::PING, 0, 0, {}, {}};**

**}**

**TTask TTask::MakeResult(const std::string &result) {**

**return TTask{TTaskType::RESULT, 0, 0, {}, result};**

**}**

**std::set<NodeId> TSink::GetPingingResultAndFinishPinging() {**

**std::lock\_guard l(mtx);**

**if (!pingingMode) {**

**throw std::logic\_error("Something defenitely went wrong");**

**}**

**std::set<NodeId> result;**

**while (!pingingResult.empty()) {**

**result.insert(pingingResult.top());**

**pingingResult.pop();**

**}**

**pingingMode = false;**

**return result;**

**}**

**TSink::TSink(TTaskQueue& \_tq, Port \_port) : tq(\_tq), context(2), inputSocket(context, zmq::socket\_type::pull), port(\_port) {**

**inputSocket.bind("tcp://localhost:" + std::to\_string(port));**

**}**

**void TSink::Routine() {**

**while (true) {**

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

**auto ret = inputSocket.recv(msg, zmq::recv\_flags::none);**

**if (ret.has\_value() && (EAGAIN == ret.value())) {**

**throw std::logic\_error("Something bad...");**

**}**

**std::string content((char\*)msg.data(), (char\*)msg.data() + msg.size());**

**// Pinging**

**if (content.size() >= 1 && content[0] == 'p') {**

**std::lock\_guard l(mtx);**

**if (pingingMode) {**

**std::stringstream ss(content);**

**std::string dummy;**

**ss >> dummy;**

**NodeId id1;**

**while (ss >> id1) {**

**pingingResult.push(id1);**

**}**

**}**

**continue;**

**}**

**//**

**tq.push(TTask::MakeResult(content));**

**}**

**}**

**void TFrontend::Handle(const std::string &line) {**

**std::stringstream ss(line);**

**std::string command;**

**ss >> command;**

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

**size\_t id;**

**int parent;**

**ss >> id >> parent;**

**tq->push(TTask::MakeCreate(id, parent));**

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

**size\_t id, n;**

**ss >> id >> n;**

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

**for (size\_t i = 0; i < n; i++) {**

**int cur;**

**ss >> cur;**

**a.push\_back(cur);**

**}**

**tq->push(TTask::MakeExec(id, a));**

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

**tq->push(TTask::MakePing());**

**}**

**}**

**void TFrontend::PushResult(const std::string &msg) {**

**std::lock\_guard l(mtx);**

**q.push(msg);**

**}**

**void TFrontend::Routine() {**

**std::string line;**

**std::cout << "> ";**

**while (std::getline(std::cin, line)) {**

**{**

**std::lock\_guard l(mtx);**

**while (!q.empty()) {**

**std::cout << q.front() << std::endl;**

**q.pop();**

**}**

**}**

**Handle(line);**

**std::cout << "> ";**

**}**

**}**

**Port TPortPool::get() {**

**lastPort++;**

**if (lastPort == UPPER\_PORT) {**

**throw std::logic\_error("Don't have enough ports...");**

**}**

**return lastPort;**

**}**

**void TNodeStructure::AddNewChild(NodeId id) {**

**if (nodes.find(id) != nodes.end()) {**

**front.PushResult("Error: Already exists");**

**return ;**

**}**

**nodes.insert(id);**

**childSets[id].insert(id);**

**childPorts[id] = TPortPool::get();**

**childSockets[id] = zmq::socket\_t(context, zmq::socket\_type::push);**

**childSockets[id].connect("tcp://localhost:" + std::to\_string(childPorts[id]));**

**CreateProcess(path, id, childPorts[id], sinkPort);**

**}**

**void TNodeStructure::SendCreate(NodeId id, int parent) {**

**if (parent == -1) {**

**AddNewChild(id);**

**return ;**

**}**

**if (nodes.find(id) != nodes.end()) {**

**front.PushResult("Error: Already exists");**

**return ;**

**}**

**if (nodes.find(parent) == nodes.end()) {**

**front.PushResult("Error: Parent not found");**

**return ;**

**}**

**for (const auto& p : childSets) {**

**if (p.second.find(parent) != p.second.end()) {**

**childSets[p.first].insert(id);**

**nodes.insert(id);**

**Port pingPort = TPortPool::get();**

**zmq::message\_t msg("create " + std::to\_string(id) + " " + std::to\_string(parent) + " " + std::to\_string(TPortPool::get()) + " " + std::to\_string(pingPort));**

**childSockets[p.first].send(msg, zmq::send\_flags::none);**

**}**

**}**

**}**

**void TNodeStructure::SendExec(NodeId id, const std::vector<int>& a) {**

**if (nodes.find(id) == nodes.end()) {**

**front.PushResult("Error:" + std::to\_string(id) + ": Not found");**

**return ;**

**}**

**for (const auto& p : childSets) {**

**if (p.second.find(id) != p.second.end()) {**

**childSets[p.first].insert(id);**

**std::string content = "exec " + std::to\_string(id) + " " + std::to\_string(a.size()) + " ";**

**for (const auto& i : a) {**

**content += std::to\_string(i) + " ";**

**}**

**zmq::message\_t msg(content);**

**childSockets[p.first].send(msg, zmq::send\_flags::none);**

**}**

**}**

**}**

**void TSink::StartPinging() {**

**std::lock\_guard l(mtx);**

**pingingMode = true;**

**}**

**void TNodeStructure::SendPing() {**

**sink.StartPinging();**

**for (auto& p : childSockets) {**

**zmq::message\_t msg(std::string("r"));**

**p.second.send(msg, zmq::send\_flags::none);**

**}**

**std::this\_thread::sleep\_for(FIRST\_PING\_TIME);**

**// Harvest**

**{**

**std::set<NodeId> result = sink.GetPingingResultAndFinishPinging();**

**if (result.size() == nodes.size()) {**

**front.PushResult("Ok: -1");**

**// sinkSocket.send(zmq::message\_t(std::string("Ok: -1")), zmq::send\_flags::none);**

**} else {**

**std::string content = "Ok: ";**

**for (const auto& nId : nodes) {**

**if (result.find(nId) == result.end()) {**

**content += std::to\_string(nId) + " ";**

**}**

**}**

**front.PushResult(content);**

**// sinkSocket.send(zmq::message\_t(content), zmq::send\_flags::none);**

**}**

**}**

**}**

**TMainManager::TMainManager(const std::string &path, TFrontend &front) : sinkPort(TPortPool::get()), sink(wq, sinkPort), ns(path, sinkPort, front, sink), frontend(front) {**

**sinkPort = sink.port;**

**ns.sinkPort = sink.port;**

**frontend.tq = &wq;**

**}**

**void TMainManager::Routine() {**

**while (true) {**

**TTask task = wq.pop();**

**//**

**switch (task.type) {**

**case TTaskType::CREATE:**

**ns.SendCreate(task.id, task.parent);**

**break;**

**case TTaskType::EXEC:**

**ns.SendExec(task.id, task.data);**

**break;**

**case TTaskType::RESULT:**

**frontend.PushResult(task.result);**

**break;**

**case TTaskType::PING:**

**ns.SendPing();**

**break;**

**}**

**}**

**}**

**}**

**========================== manager.cpp ==========================**

**#include "manager\_base.hpp"**

**using namespace lab5;**

**int main() {**

**if (getenv("PATH\_TO\_COMP") == nullptr) {**

**std::cerr << "PATH\_TO\_COMP is not specified" << std::endl;**

**exit(EXIT\_FAILURE);**

**}**

**std::string path = std::string(getenv("PATH\_TO\_COMP"));**

**TFrontend front;**

**TMainManager man(path, front);**

**std::thread t1(&TFrontend::Routine, &front);**

**std::thread t2(&TMainManager::Routine, &man);**

**std::thread t3(&TSink::Routine, &man.sink);**

**t1.join();**

**t2.join();**

**t3.join();**

**}**

**Демонстрация работы программы**



**Выводы**

В ходе выполнения лабораторной работе я получил знания и навыки использования очередей сообщений. Я получил понимание концепции асинхронного программирования, узнал о сокетах и сетевом протоколе TCP.