МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №5-7

По курсу «Операционные системы»

Студент: Кириллова Е.К.

Группа: М8О-203Б-23

Вариант: 41

Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Выводы

**Репозиторий**

https://github.com/ElenaKirillova05/osLabs/tree/main

**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.

Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.

Все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

**Набор команд 1 (подсчет суммы n чисел)**

Формат команды: exec id n k1 … kn

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

n – количество складываемых чисел (от 1 до 108)

k1 … kn – складываемые числа

Пример:

> exec 10 3 1 2 3

Ok:10: 6

**Команда проверки 3**

Тип проверки доступности узлов

Команда проверки 1

Формат команды: pingall

Вывод всех недоступных узлов вывести разделенные через точку запятую.

**Общие сведения о программе**

Общие сведения о программе программа реализует распределенную систему обмена сообщениями между узлами с использованием библиотеки ZeroMQ узлы связаны в бинарное дерево и могут создавать новые узлы выполнять вычисления пересылать сообщения и удалять узлы система состоит из серверной и клиентской частей сервер управляет узлами обрабатывает команды от пользователя и контролирует состояние узлов клиенты представляют собой узлы дерева каждый узел обменивается сообщениями с родительским узлом а также с левым и правым дочерними узлами если они существуют обмен сообщениями реализован асинхронно с использованием ZeroMQ сокетов типа pair для связи между узлами

**Общий метод и алгоритм решения**

Общий метод и алгоритм решения программа использует архитектуру клиент-сервер для управления узлами в бинарном дереве сервер инициализируется с набором доступных портов и ожидает команды от пользователя команды могут быть следующими создание узла выполнение вычислений в узле проверка доступности узлов ping all и удаление узлов сервер анализирует команду если это команда на создание нового узла он выделяет порт для связи с новым узлом порождает дочерний процесс и запускает клиентскую программу с передачей ей идентификатора узла и выделенного порта если команда касается уже существующего узла сервер пересылает команду соответствующему узлу через сокет клиенты обрабатывают команды в зависимости от их типа если это команда на создание узла клиент проверяет идентификатор нового узла и определяет будет ли новый узел левым или правым потомком если сокет связи с потомком уже существует клиент пересылает команду по этому сокету если нет то клиент выделяет порт для связи порождает дочерний процесс и запускает новый клиентский узел если команда на выполнение вычислений клиент проверяет соответствует ли идентификатор узла его собственному если да то производит вычисления и отправляет результат родительскому узлу если нет то пересылает команду дочерним узлам если команда на удаление узла клиент закрывает соответствующий сокет и удаляет информацию о дочернем узле сервер поддерживает состояние сети узлов и выполняет проверку их доступности с помощью команды ping all при получении этой команды каждый узел отправляет ответ своему родительскому узлу сервер собирает информацию о доступных узлах определяет недоступные узлы и инициирует их удаление

**Исходный код**

client.cpp:

```

#include "send\_recv.hpp"

void Init(std::uint32\_t node\_id, std::uint32\_t port\_id, ClientNode& client\_node) {

client\_node.id = node\_id;

client\_node.context\_client\_.emplace();

client\_node.socket\_parent\_.emplace(client\_node.context\_client\_.value(), zmq::socket\_type::pair);

client\_node.socket\_parent\_.value().connect(std::string("tcp://localhost:") + std::to\_string(port\_id));

}

void Append(std::optional<zmq::socket\_t>& node, zmq::context\_t& context, std::optional<std::uint32\_t>& setid, std::string message) {

std::stringstream ss(message);

std::string create;

ss >> create;

std::uint32\_t node\_id;

ss >> node\_id;

std::uint32\_t port\_id;

ss >> port\_id;

if (node.has\_value()) {

if (!Send(node.value(), message)) {

node.reset();

Append(node, context, setid, message);

}

} else {

node.emplace(context, zmq::socket\_type::pair);

node.value().bind("tcp://\*:" + std::to\_string(port\_id));

setid.emplace(node\_id);

pid\_t pid = fork();

if (pid == 0) {

std::string child\_id = std::to\_string(node\_id);

std::string port = std::to\_string(port\_id);

execl("./client", child\_id.c\_str(), port.c\_str(), nullptr);

}

}

}

void Create(ClientNode& client\_node, std::string message) {

std::stringstream ss(message);

std::string create;

ss >> create;

std::uint32\_t node\_id;

ss >> node\_id;

if (node\_id < client\_node.id) {

Append(client\_node.socket\_left\_, client\_node.context\_client\_.value(), client\_node.left\_id\_, std::move(message));

} else {

Append(client\_node.socket\_right\_, client\_node.context\_client\_.value(), client\_node.right\_id\_, std::move(message));

}

}

void Calculate(ClientNode& client\_node, std::string message) {

std::stringstream ss(message);

std::string exec;

ss >> exec;

std::uint32\_t node\_id;

ss >> node\_id;

if (node\_id < client\_node.id) {

Send(client\_node.socket\_left\_.value(), std::move(message));

} else if (node\_id > client\_node.id) {

Send(client\_node.socket\_right\_.value(), std::move(message));

} else {

std::int32\_t ans = 0;

std::int32\_t number = 0;

while (ss >> number) {

ans += number;

}

Send(client\_node.socket\_parent\_.value(), std::string("OK:") + std::to\_string(node\_id) + std::string(": ") + std::to\_string(ans));

}

}

void DestroyNode(ClientNode& client\_node, std::string message) {

std::stringstream ss(message);

std::string destroy;

ss >> destroy;

std::uint32\_t node\_id;

ss >> node\_id;

if (node\_id < client\_node.id) {

if (client\_node.left\_id\_.has\_value()) {

if (client\_node.left\_id\_.value() == node\_id) {

client\_node.left\_id\_.reset();

client\_node.socket\_left\_.reset();

} else {

if (!Send(client\_node.socket\_left\_.value(), std::move(message))) {

client\_node.left\_id\_.reset();

client\_node.socket\_left\_.reset();

}

}

} else {}

} else if (node\_id > client\_node.id) {

if (client\_node.right\_id\_.has\_value()) {

if (client\_node.right\_id\_.value() == node\_id) {

client\_node.right\_id\_.reset();

client\_node.socket\_right\_.reset();

} else {

if (!Send(client\_node.socket\_right\_.value(), std::move(message))) {

client\_node.right\_id\_.reset();

client\_node.socket\_right\_.reset();

}

}

} else {}

}

}

void ParseAndMakeAction(ClientNode& client\_node, std::string message) {

std::stringstream ss(message);

std::string command;

ss >> command;

if (command == "create") {

Create(client\_node, std::move(message));

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

Calculate(client\_node, std::move(message));

} else if (message == "ping all") {

Send(client\_node.socket\_parent\_.value(), std::string("node\_ping: ") + std::to\_string(client\_node.id));

if (client\_node.socket\_left\_.has\_value()) {

if (!Send(client\_node.socket\_left\_.value(), message)) {

client\_node.socket\_left\_.reset();

client\_node.socket\_right\_.reset();

}

}

if (client\_node.socket\_right\_.has\_value()) {

if (!Send(client\_node.socket\_right\_.value(), message)) {

client\_node.socket\_right\_.reset();

client\_node.right\_id\_.reset();

}

}

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

DestroyNode(client\_node, std::move(message));

} else {

Send(client\_node.socket\_parent\_.value(), std::move(message));

}

}

#ifndef TEST\_BUILD

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

ClientNode client\_node;

pid\_t pid = getpid();

std::uint32\_t node\_id = std::stoul(argv[0]);

std::uint32\_t port\_id = std::stoul(argv[1]);

Init(node\_id, port\_id, client\_node);

Send(client\_node.socket\_parent\_.value(), std::string("make node with id: ") + std::to\_string(node\_id) + std::string(", pid: ") + std::to\_string(pid));

while (true) {

while (true) {

std::optional<std::string> message = Recv(client\_node.socket\_parent\_);

if (message.has\_value()) {

ParseAndMakeAction(client\_node, std::move(message.value()));

} else {

break;

}

}

while (true) {

std::optional<std::string> message = Recv(client\_node.socket\_left\_);

if (message.has\_value()) {

ParseAndMakeAction(client\_node, std::move(message.value()));

} else {

break;

}

}

while (true) {

std::optional<std::string> message = Recv(client\_node.socket\_right\_);

if (message.has\_value()) {

ParseAndMakeAction(client\_node, std::move(message.value()));

} else {

break;

}

}

std::this\_thread::sleep\_for(0.1s);

}

}

#endif

```

send\_recv.hpp:

```

#pragma once

#include <zmq.hpp>

#include <iostream>

#include <vector>

#include <cstdint>

#include <optional>

#include <sstream>

#include <set>

#include <unistd.h>

#include <chrono>

#include <thread>

#include <mutex>

#include <algorithm>

using namespace std::chrono\_literals;

struct ClientNode {

std::uint32\_t id = 0;

std::optional<std::uint32\_t> left\_id\_;

std::optional<std::uint32\_t> right\_id\_;

std::optional<zmq::context\_t> context\_client\_;

std::optional<zmq::socket\_t> socket\_parent\_;

std::optional<zmq::socket\_t> socket\_left\_;

std::optional<zmq::socket\_t> socket\_right\_;

};

struct ServerNode {

bool check\_ping\_ = false;

std::uint32\_t id = 0;

std::uint32\_t child\_id = 0;

std::optional<zmq::socket\_t> socket\_root\_;

std::optional<zmq::context\_t> context\_root\_;

std::vector<std::pair<std::uint32\_t, bool>> ports\_;

std::set<std::uint32\_t> node\_id\_;

};

bool Send(zmq::socket\_t& receiver, std::string message) {

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

std::memcpy(request.data(), message.c\_str(), message.size());

if (receiver.send(request, zmq::send\_flags::dontwait).has\_value()) {

return true;

}

return false;

}

std::optional<std::string> Recv(std::optional<zmq::socket\_t>& sender) {

if (sender.has\_value()) {

zmq::message\_t message;

std::optional<size\_t> count\_bytes = sender.value().recv(message, zmq::recv\_flags::dontwait);

if (count\_bytes.has\_value()) {

return {std::string(static\_cast<char\*>(message.data()), message.size())};

}

}

return std::nullopt;

}

```

server.cpp:

```

#include "send\_recv.hpp"

std::optional<std::uint32\_t> GetPort(std::vector<std::pair<std::uint32\_t, bool>>& ports) {

std::uint32\_t ans = 0;

for (auto& item : ports) {

if (item.second) {

item.second = false;

ans = item.first;

break;

}

}

if (ans == 0) {

return std::nullopt;

}

return {ans};

}

bool InServer(ServerNode& node, std::uint32\_t id) {

return node.node\_id\_.find(id) != node.node\_id\_.end();

}

void InitPorts(std::vector<std::pair<std::uint32\_t, bool>>& ports, std::uint32\_t start\_port) {

for (auto& item : ports) {

item.first = start\_port++;

item.second = true;

}

}

bool ExecParse(ServerNode& server\_node, std::string message) {

std::stringstream ss(message);

std::string exec;

ss >> exec;

std::uint32\_t node\_id = 0;

ss >> node\_id;

if (InServer(server\_node, node\_id)) {

Send(server\_node.socket\_root\_.value(), std::move(message));

return true;

}

return false;

}

void Init(ServerNode& server\_node) {

server\_node.context\_root\_.emplace();

server\_node.ports\_.resize(100, {0, true});

InitPorts(server\_node.ports\_, 1070);

server\_node.node\_id\_.emplace(0);

}

bool Create(ServerNode& server\_node, std::string message) {

std::stringstream ss(message);

std::string create;

ss >> create;

std::uint32\_t node\_id = 0;

ss >> node\_id;

if (InServer(server\_node, node\_id)) {

return false;

}

std::optional<std::uint32\_t> port\_id = GetPort(server\_node.ports\_);

if (!port\_id.has\_value()) {

return false;

}

server\_node.node\_id\_.emplace(node\_id);

if (server\_node.socket\_root\_.has\_value()) {

if (!Send(server\_node.socket\_root\_.value(), message + ' ' + std::to\_string(port\_id.value()))) {

server\_node.socket\_root\_.reset();

return Create(server\_node, std::move(message));

}

} else {

server\_node.socket\_root\_.emplace(server\_node.context\_root\_.value(), zmq::socket\_type::pair);

server\_node.socket\_root\_.value().bind("tcp://\*:" + std::to\_string(port\_id.value()));

server\_node.child\_id = node\_id;

pid\_t pid = fork();

if (pid == 0) {

std::string child\_id = std::to\_string(node\_id);

std::string port = std::to\_string(port\_id.value());

execl("./client", child\_id.c\_str(), port.c\_str(), nullptr);

}

}

return true;

}

void AnalizeSenders(ServerNode& server\_node, std::string message);

void ReceivQueue(ServerNode& server\_node, std::vector<std::uint32\_t>& availible\_nodes) {

while (true) {

std::optional<std::string> message = Recv(server\_node.socket\_root\_);

if (message.has\_value()) {

std::stringstream ss(message.value());

std::string command;

ss >> command;

if (command != "node\_ping:") {

AnalizeSenders(server\_node, std::move(message.value()));

} else {

std::uint32\_t node\_id;

ss >> node\_id;

availible\_nodes.push\_back(node\_id);

}

} else {

break;

}

}

}

std::vector<std::uint32\_t> GetUnavailibleNodes(ServerNode& server\_node, const std::vector<std::uint32\_t>& avail\_nodes) {

std::vector<std::uint32\_t> ans\_;

for (auto item : server\_node.node\_id\_) {

if (item == 0) continue;

if (std::find(avail\_nodes.begin(), avail\_nodes.end(), item) == avail\_nodes.end()) {

ans\_.push\_back(item);

}

}

return ans\_;

}

void Destroy(ServerNode& server\_node, const std::vector<std::uint32\_t>& destroy\_nodes) {

for (auto item : destroy\_nodes) {

server\_node.node\_id\_.erase(item);

}

}

void CallDestroyResponse(ServerNode& server\_node, const std::vector<std::uint32\_t>& destroy\_nodes) {

for (auto item : destroy\_nodes) {

std::cout << item << ' ';

if (!Send(server\_node.socket\_root\_.value(), std::string("destroy ") + std::to\_string(item))) {

server\_node.socket\_root\_.reset();

server\_node.child\_id = 0;

}

}

}

void AnalizeSenders(ServerNode& server\_node, std::string message) {

std::stringstream ss(message);

std::string command;

ss >> command;

if (command == std::string("node\_ping:")) {

if (server\_node.check\_ping\_ == true) {

std::uint32\_t node\_id;

ss >> node\_id;

std::vector<std::uint32\_t> availible\_nodes;

availible\_nodes.push\_back(node\_id);

ReceivQueue(server\_node, availible\_nodes);

std::vector<std::uint32\_t> unavailible\_nodes\_ = GetUnavailibleNodes(server\_node, availible\_nodes);

Destroy(server\_node, unavailible\_nodes\_);

CallDestroyResponse(server\_node, unavailible\_nodes\_);

server\_node.check\_ping\_ = false;

std::cout << "Availible nodes: ";

for (auto item : availible\_nodes) {

std::cout << item << ' ';

}

std::cout << std::endl;

} else {}

} else {

std::cout << message << std::endl;

}

}

void AnalizeUser(ServerNode& server\_node, std::string message) {

std::stringstream ss(message);

std::string command;

ss >> command;

if (command == std::string("create")) {

if (!Create(server\_node, std::move(message))) {

std::cout << "Max limit or node already in system" << std::endl;

}

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

if (!ExecParse(server\_node, std::move(message))) {

std::cout << "Node not in system!" << std::endl;

}

} else if (message == "ping all") {

server\_node.check\_ping\_ = true;

if (!server\_node.socket\_root\_.has\_value() || !Send(server\_node.socket\_root\_.value(), message)) {

server\_node.socket\_root\_.reset();

server\_node.child\_id = 0;

server\_node.check\_ping\_ = false;

server\_node.node\_id\_.clear();

server\_node.node\_id\_.emplace(0);

return;

}

std::cout << "Please wait. Nodes checking..." << std::endl;

std::this\_thread::sleep\_for(1s);

} else {

std::cout << "Incorrect command" << std::endl;

}

}

#ifndef TEST\_BUILD

int main() {

ServerNode server\_node;

Init(server\_node);

std::string user\_command;

while (true) {

while (true) {

std::optional<std::string> message = Recv(server\_node.socket\_root\_);

if (message.has\_value()) {

AnalizeSenders(server\_node, std::move(message.value()));

} else {

break;

}

}

std::getline(std::cin, user\_command);

AnalizeUser(server\_node, std::move(user\_command));

}

}

#endif

```

ТЕСТЫ

tests.cpp:

```

#define TEST\_BUILD

#include <gtest/gtest.h>

#include "send\_recv.hpp"

#include "server.cpp"

#include "client.cpp"

TEST(ClientNodeTest, InitTest) {

ClientNode client\_node;

Init(1, 1070, client\_node);

EXPECT\_EQ(client\_node.id, 1);

EXPECT\_TRUE(client\_node.context\_client\_.has\_value());

EXPECT\_TRUE(client\_node.socket\_parent\_.has\_value());

}

TEST(ClientNodeTest, AppendTest) {

zmq::context\_t context;

std::optional<zmq::socket\_t> socket;

std::optional<std::uint32\_t> node\_id;

std::string message = "create 2";

Append(socket, context, node\_id, message);

EXPECT\_TRUE(socket.has\_value());

EXPECT\_TRUE(node\_id.has\_value());

EXPECT\_EQ(node\_id.value(), 2);

}

TEST(ServerNodeTest, InitTest) {

ServerNode server\_node;

Init(server\_node);

EXPECT\_EQ(server\_node.node\_id\_.size(), 1);

EXPECT\_EQ(server\_node.ports\_.size(), 100);

}

TEST(ServerNodeTest, PingAllTest) {

ServerNode server\_node;

Init(server\_node);

std::string message = "ping all";

try {

EXPECT\_NO\_THROW({

if (server\_node.node\_id\_.empty()) {

FAIL() << "Node ID set is empty!";

}

AnalizeUser(server\_node, message);

});

} catch (const std::bad\_optional\_access& e) {

FAIL() << "Caught bad\_optional\_access exception: " << e.what();

} catch (const std::exception& e) {

FAIL() << "Caught unexpected exception: " << e.what();

}

}

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

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

```

**Вывод**

Разработанная программа реализует распределенную систему обмена сообщениями между узлами организованными в виде бинарного дерева с использованием библиотеки zeromq асинхронный обмен сообщениями и архитектура клиент-сервер обеспечивают гибкость и масштабируемость системы программа позволяет динамически создавать удалять узлы выполнять вычисления и проверять доступность узлов подход с использованием сокетов типа pair и механизмов zeromq обеспечивает надежную передачу сообщений и устойчивость системы к сбоям архитектура позволяет легко расширять функциональность и адаптировать систему под различные задачи распределенных вычислений