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: Ветошкина София Владимировна
Группа: М8О-203Б-23
Вариант: 35
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/sofiavetoshkina/os_labs/tree/main

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.

Общие сведения о программе

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]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено

введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода:

«Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не

удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> create 10 5

Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным

узлом

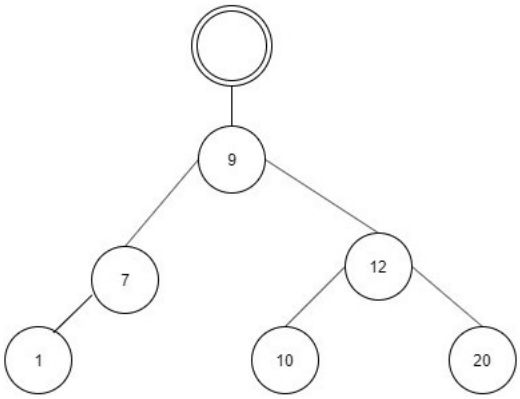
«Error:id: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

Можно найти в описании конкретной команды, определенной вариантом задания.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

Топология:



Все вычислительные узлы хранятся в идеально сбалансированном бинарном дереве поиска. [parent] — является необязательным параметром. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

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 2

Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

Пример:

> ping 10

Ok: 1 // узел 10 доступен

> ping 17

Ok: 0 // узел 17 недоступен

Метод и алгоритм решения задачи

Данная программа реализует распределенную систему для асинхронной обработки запросов с использованием технологии очередей сообщений (ZeroMQ). Программа поддерживает два типа узлов:

1. **Управляющий узел** — координирует работу системы, обрабатывает команды пользователя, управляет созданием и удалением вычислительных узлов.
2. **Вычислительные узлы** — выполняют указанные команды, обрабатывают запросы и передают результаты.

Программа построена в соответствии с требованием задания:

Узлы организованы в виде идеально сбалансированного бинарного дерева.

Используется асинхронная передача сообщений между узлами с помощью ZeroMQ.

Реализована обработка следующих основных команд:

1. **create**: создание нового вычислительного узла.
2. **exec**: выполнение команды на заданном узле (подсчет суммы чисел).
3. **ping**: проверка доступности узла.
4. **kill**: удаление узла и всех его потомков из дерева.

Программа устойчива к сбоям: при удалении узла (kill) его родительские узлы сохраняют свою работоспособность.

Архитектура программы

1. **Файлы и их функции**:
    - **ClientProgram.cpp**: реализация управляющего узла. Обеспечивает взаимодействие с пользователем, обрабатывает команды и отправляет их вычислительным узлам.
    - **ServerProgram.cpp**: реализация вычислительного узла. Обрабатывает команды, выполняет расчеты и передает результаты управляющему узлу.
    - **CalculationNode.h**: описание вычислительного узла, включая его взаимодействие с дочерними и родительскими узлами через ZeroMQ.
    - **BalancedTree.h**: реализация сбалансированного бинарного дерева, хранящего информацию о топологии узлов.
    - **ZMQFunctions.h**: вспомогательные функции для работы с ZeroMQ, включая отправку, получение сообщений, управление соединениями и портами.
2. **Технологии и библиотеки**:
    - **ZeroMQ**: для передачи сообщений между узлами. Используются типы сокетов REQ и REP.
    - **C++ стандартные библиотеки**: для работы с потоками, строками, ввода-вывода и базовой логики.

Основные возможности

1. **Создание нового узла (create)**:
    - Узел добавляется в сбалансированное бинарное дерево в первую доступную позицию.
    - Если узел с заданным идентификатором уже существует или его родитель недоступен, возвращается сообщение об ошибке.
2. **Асинхронное выполнение команд (exec)**:

- Команды выполняются параллельно на разных узлах. Результат возвращается, как только вычисление завершено.

3. **Проверка доступности узлов (ping)**:
   - Позволяет убедиться, что узел работает и готов к обработке запросов.
4. **Удаление узла (kill)**:
   - Узел и все его потомки удаляются из дерева. Родительские узлы продолжают функционировать.

Принципы работы

1. **Распределенность**:
   - Каждый вычислительный узел — это отдельный процесс, запускаемый через fork и управляемый управляющим узлом.
   - Управляющий узел и вычислительные узлы взаимодействуют только через очереди сообщений.
2. **Асинхронность**:
   - Управляющий узел не блокируется на выполнение одной команды; он может отправлять команды другим узлам, пока текущие выполняются.
3. **Сбалансированность дерева**:
   - Новые узлы добавляются таким образом, чтобы поддерживать минимальную высоту дерева. Узлы распределяются равномерно между уровнями.
4. **Отказоустойчивость**:
   - Если узел завершает работу (например, после команды kill), система остается работоспособной, а недоступные узлы помечаются как такие.

Исходный код

CalculationNode.h

```
#pragma once

#include "ZMQFunctions.h"
#include "unistd.h"

class CalculationNode {
public:
    int id;
    int left_id = -2, right_id = -2, parent_id;
    int left_port, right_port, parent_port;
    zmq::context_t context;
    zmq::socket_t left, right, parent;

    CalculationNode(int id, int parent_port, int parent_id)
        : id(id), left_id(-2), right_id(-2), parent_id(parent_id),
          left_port(0), right_port(0), parent_port(parent_port),
          context(), left(context, ZMQ_REQ), right(context, ZMQ_REQ),
parent(context, ZMQ_REP) {
        if (id != -1) {
            connect(parent, parent_port);
        }
    }

    std::string create(int child_id) {
        int port;
        bool isLeft = false;
        if (left_id == -2) {
            left_port = bind(left, child_id);
            left_id = child_id;
            port = left_port;
            isLeft = true;
        } else if (right_id == -2) {
            right_port = bind(right, child_id);
            right_id = child_id;
            port = right_port;
        } else {
            return "Error: Can not create the calculation node";
        }

        int fork_id = fork();
        if (fork_id == 0) {
            const char* pathToServer =
getenv("PATH_TO_CALCULATION_NODE");
```

```
        if (execl(pathToServer, "server", std::to_string(child_id).c_str(),
                std::to_string(port).c_str(), std::to_string(id).c_str(),
                (char*)NULL) == -1) {
            std::cerr << "Error: Can not run the execl-command" << std::endl;
            exit(EXIT_FAILURE);
        }
    } else {
        try {
            zmq::socket_t& childSocket = isLeft ? left : right;
            childSocket.setsockopt(ZMQ_SNDTIMEO, 3000);
            send_message(childSocket, "pid");
            std::string child_pid = receive_message(childSocket);
            return "Ok: " + child_pid;
        } catch (const std::exception& e) {
            return "Error: Can not connect to the child";
        }
    }
    return "Error: Unexpected error in create function";
}

std::string ping(int id) {
    std::string answer = "Ok: 0";
    if (this->id == id) {
        answer = "Ok: 1";
        return answer;
    }
    if (left_id == id) {
        std::string message = "ping " + std::to_string(id);
        send_message(left, message);
        try {
            message = receive_message(left);
            if (message == "Ok: 1") {
                answer = message;
            }
        }
        catch(int){}
    } else if (right_id == id) {
        std::string message = "ping " + std::to_string(id);
        send_message(right, message);
        try {
            message = receive_message(right);
            if (message == "Ok: 1") {
                answer = message;
            }
        }
```

```
        catch(int){}
    }
    return answer;
}

std::string sendstring(std::string string, int id) {
    std::string answer = "Error: Parent not found";
    if (left_id == -2 && right_id == -2) return answer;
    if (left_id == id) {
        if (ping(left_id) == "Ok: 1") {
            send_message(left, string);
            try {
                answer = receive_message(left);
            }
            catch(int){}
        }
    } else if (right_id == id) {
        if (ping(right_id) == "Ok: 1") {
            send_message(right, string);
            try {
                answer = receive_message(right);
            }
            catch(int){}
        }
    } else {
        if (ping(left_id) == "Ok: 1") {
            std::string message = "send " + std::to_string(id) + " " + string;
            send_message(left, message);
            try {
                message = receive_message(left);
            }
            catch(int) {
                message = "Error: Parent not found";
            }
            if (message != "Error: Parent not found") answer = message;
        }
        if (ping(right_id) == "Ok: 1") {
            std::string message = "send " + std::to_string(id) + " " + string;
            send_message(right, message);
            try {
                message = receive_message(right);
            }
            catch(int) {
                message = "Error: Parent not found";
            }
```

```
            if (message != "Error: Parent not found") answer = message;
        }
    }
    return answer;
}

std::string exec(std::string string) {
    std::istringstream string_thread(string);
    int result = 0;
    int amount, number;
    string_thread >> amount;
    for (int i = 0; i < amount; ++i) {
        string_thread >> number;
        result += number;
    }
    std::string answer = "Ok: " + std::to_string(id) + ": " + std::to_string(result);
    return answer;
}

std::string treeclear(int child) {
    if (left_id == child) {
        left_id = -2;
        unbind(left, left_port);
    }
    else {
        right_id = -2;
        unbind(right, right_port);
    }
    return "Ok";
}

std::string kill() {
    if (left_id != -2){
        if (ping(left_id) == "Ok: 1") {
            std::string message = "kill";
            send_message(left, message);
            try {
                message = receive_message(left);
            }
            catch(int){}
            unbind(left, left_port);
            left.close();
        }
    }
    if (right_id != -2) {
```

```
            if (ping(right_id) == "Ok: 1") {
                std::string message = "kill";
                send_message(right, message);
                try {
                    message = receive_message(right);
                }
                catch (int){}
                unbind(right, right_port);
                right.close();
            }
        }
        return std::to_string(parent_id);
    }

    ~CalculationNode() {}
};
```

BalancedTree.h

```
#pragma once

#include <set>

class BalancedTree {
        class BalancedTreeNode {
            public:
                int id;
                BalancedTreeNode* left;
                BalancedTreeNode* right;
                int height;
                bool available;
                BalancedTreeNode(int id) {
                    this->id = id;
                    available = true;
                    left = nullptr;
                    right = nullptr;
                    height = 0;
                }
                void SetUnavailability(int id) {
                    if (this->id == id) {
                        available = false;
                    } else {
                        if (left != nullptr) left->SetUnavailability(id);
                        if (right != nullptr) right->SetUnavailability(id);
                    }
                }
                void Remove(int id, std::set<int> &ids) {
```

```
    if (left != nullptr && left->id == id) {
        left->RecursionRemove(ids);
        ids.erase(left->id);
        delete left;
        left = nullptr;
    } else if (right != nullptr && right->id == id) {
        right->RecursionRemove(ids);
        ids.erase(right->id);
        delete right;
        right = nullptr;
    } else {
        if (left != nullptr) left->Remove(id, ids);
        if (right != nullptr) right->Remove(id, ids);
    }
}
void RecursionRemove(std::set<int> &ids) {
    if (left != nullptr) {
        left->RecursionRemove(ids);
        ids.erase(left->id);
        delete left;
        left = nullptr;
    }
    if (right != nullptr) {
        right->RecursionRemove(ids);
        ids.erase(right->id);
        delete right;
        right = nullptr;
    }
}
void AddInNode(int id, int parent_id, std::set<int> &ids) {
    if (this->id == parent_id) {
        if (left == nullptr) {
            left = new BalancedTreeNode(id);
        } else {
            right = new BalancedTreeNode(id);
        }
        ids.insert(id);
    } else {
        if (left != nullptr) left->AddInNode(id, parent_id, ids);
        if (right != nullptr) right->AddInNode(id, parent_id, ids);
    }
}
int MinimalHeight() {
    if (left == nullptr || right == nullptr) return 0;
    int left_height = -1;
```

```
            int right_height = -1;
            if (left->available == true) left_height = left->MinimalHeight();
            if (right->available == true) right_height = right->MinimalHeight();
            if (right_height == -1 && left_height == -1) {
                available = false;
                return -1;
            }
            if (right_height  == -1) return left_height + 1;
            if (left_height == -1) return right_height + 1;
            return std::min(left_height, right_height) + 1;
        }
        int IDMinimalHeight(int height, int current_height) {
            if (height < current_height) return -2;
            if (height > current_height) {
                int current_id = -2;
                if (left != nullptr && left->available == true) {
                    current_id = left->IDMinimalHeight(height, (current_height +
1));
                }
                if (right != nullptr && right->available == true && current_id == -
2) {
                    current_id = right->IDMinimalHeight(height, (current_height +
1));
                }
                return current_id;
            }
            if (left == nullptr || right == nullptr) return id;
            return -2;
        }
        ~BalancedTreeNode() {}
    };
  private:
      BalancedTreeNode* root;
  public:
    std::set<int> ids;
    BalancedTree() {
        root = new BalancedTreeNode(-1);
    }
    bool Exist(int id) {
        return ids.find(id) != ids.end();
    }
    void SetUnavailabilityNode(int id) {
        root->SetUnavailability(id);
    }
    int FindID() {
```

```
            int h = root->MinimalHeight();
            return root->IDMinimalHeight(h, 0);
        }
        void AddInTree(int id, int parent) {
            root->AddInNode(id, parent, ids);
        }
        void RemoveFromRoot(int idElem) {
            root->Remove(idElem, ids);
        }
        ~BalancedTree() {
            root->RecursionRemove(ids);
            delete root;
        }
};
```

ZMQFunctions.h

```
#pragma once

#include <iostream>
#include <string>
#include <zmq.hpp>
const int MAIN_PORT = 4040;

void send_message(zmq::socket_t &socket, const std::string &msg) {
    zmq::message_t message(msg.size());
    memcpy(message.data(), msg.c_str(), msg.size());
    socket.send(message);
}

std::string receive_message(zmq::socket_t &socket) {
    zmq::message_t message;
    int chars_read;
    try {
        chars_read = (int)socket.recv(&message);
    }
    catch (...) {
        chars_read = 0;
    }
    if (chars_read == 0) {
        throw -1;
    }
    std::string received_msg(static_cast<char*>(message.data()), message.size());
    return received_msg;
}

void connect(zmq::socket_t &socket, int port) {
```

```
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.connect(address);
}

void disconnect(zmq::socket_t &socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.disconnect(address);
}

int bind(zmq::socket_t &socket, int id) {
    int port = MAIN_PORT + id;
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    while(true) {
        try {
            socket.bind(address);
            break;
        } catch(...) {
            port++;
        }
    }
    return port;
}

void unbind(zmq::socket_t &socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.unbind(address);
}
```

ClientProgram.cpp

```
#include "CalculationNode.h"
#include "ZMQFunctions.h"
#include "BalancedTree.h"

int main() {
    std:: string command;
    CalculationNode node(-1, -1, -1);
    std:: string answer;
    std:: cout << "Hello there! Please check out the task commands: " << std:: endl;
    std:: cout << '\t' << "create id: for creating a new calculation node" << std:: endl;
    std:: cout << '\t' << "exec id n n1 n2... n: for calculating a sum" << std:: endl;
    std:: cout << '\t' << "ping id: for checking availabilty node" << std:: endl;
    std:: cout << '\t' << "kill id: for killing a calculation node" << std:: endl;
    BalancedTree tree;
    while ((std:: cout << "Please enter your command: ") && (std:: cin >>
command)) {
        if (command == "create") {
```

```
    int child;
    std:: cin >> child;
    if (tree.Exist(child)) {
        std:: cout << "Error: Already exists" << std:: endl;
    }
    else {
        while (true) {
            int idParent = tree.FindID();
            if (idParent == node.id) {
                answer = node.create(child);
                tree.AddInTree(child, idParent);
                break;
            }
            else {
                std:: string message = "create " + std:: to_string(child);
                answer = node.sendstring(message, idParent);
                if (answer == "Error: Parent not found") {
                    tree.SetUnavailabilityNode(idParent);
                }
                else {
                    tree.AddInTree(child, idParent);
                    break;
                }
            }
        }
        std:: cout << answer << std::endl;
    }
}
else if (command == "exec") {
    std:: string str;
    int child;
    std:: cin >> child;
    getline(std:: cin, str);
    if (!tree.Exist(child)) {
        std:: cout << "Error:" << child << ": Not found" << std:: endl;
    }
    else {
        std:: string message = "exec " + str;
        answer = node.sendstring(message, child);
        std:: cout << answer << std:: endl;
    }
}
else if (command == "ping") {
    int child;
    std:: cin >> child;
```

```
    if (!tree.Exist(child)) {
        std::cout << "Error: Not found" << std:: endl;
    }
    else if (node.left_id == child || node.right_id == child) {
        answer = node.ping(child);
        std:: cout << answer << std:: endl;
    }
    else {
        std:: string message = "ping " + std:: to_string(child);
        answer = node.sendstring(message, child);
        if (answer == "Error: Parent not found") {
            answer = "Ok: 0";
        }
        std:: cout << answer << std:: endl;
    }
}
else if (command == "kill") {
    int child;
    std:: cin >> child;
    std:: string message = "kill";
    if (!tree.Exist(child)) {
        std:: cout << "Error: Not found" << std:: endl;
    } else {
        answer = node.sendstring(message, child);
        if (answer != "Error: Parent not found") {
            tree.RemoveFromRoot(child);
            if (child == node.left_id){
                unbind(node.left, node.left_port);
                node.left_id = -2;
                answer = "Ok";
            } else if (child == node.right_id) {
                node.right_id = -2;
                unbind(node.right, node.right_port);
                answer = "Ok";
            } else {
                message = "clear " + std::to_string(child);
                answer = node.sendstring(message, std::stoi(answer));
            }
            std::cout << answer << std::endl;
        }
    }
}
else {
    std:: cout << "Please enter correct command!" << std:: endl;
}
```

```
    }
    node.kill();
    return 0;
}
```

ServerProgram.cpp

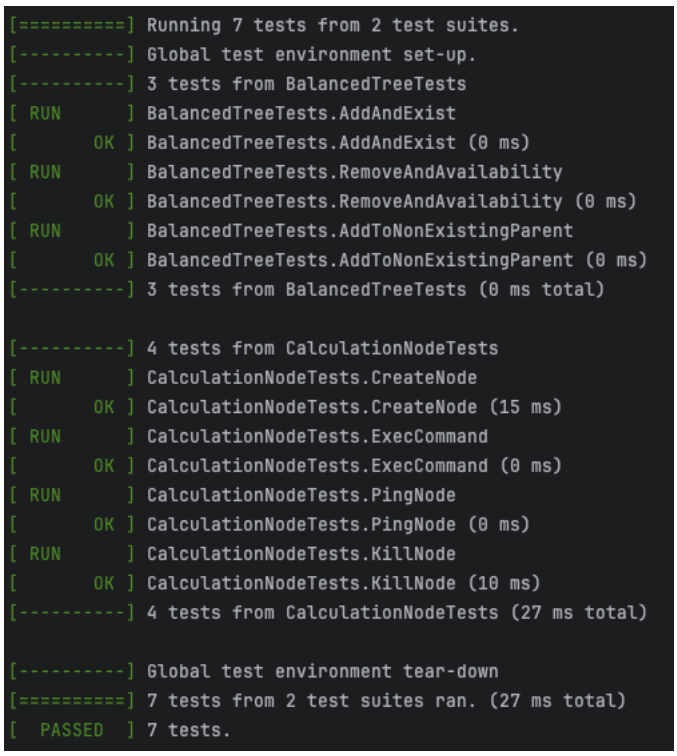
```
#include "CalculationNode.h"
#include "ZMQFunctions.h"
#include "BalancedTree.h"

int main(int argc, char *argv[]) {
    if (argc != 4) {
        std:: cout << "Usage: 1)./server, 2) child_id, 3) parent_port, 4) parent_id" <<
std:: endl;
        exit(EXIT_FAILURE);
    }
    CalculationNode node(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));
    while(true) {
        std::string message;
        std::string command;
        message = receive_message(node.parent);
        std::istringstream request(message);
        request >> command;
        if (command == "pid") {
            std::string answer = std::to_string(getpid());
            send_message(node.parent, answer);
        }
        else if (command == "ping") {
            int child;
            request >> child;
            std::string answer = node.ping(child);
            send_message(node.parent, answer);
        }
        else if (command == "create") {
            int child;
            request >> child;
            std::string answer = node.create(child);
            send_message(node.parent, answer);
        }
        else if (command == "send") {
            int child;
            std::string str;
            request >> child;
            getline(request, str);
            str.erase(0, 1);
            std::string answer = node.sendstring(str, child);
```

```
            send_message(node.parent, answer);
        }
        else if (command == "exec") {
            std::string str;
            getline(request, str);
            std::string answer = node.exec(str);
            send_message(node.parent, answer);
        }
        else if (command == "kill") {
            std::string answer = node.kill();
            send_message(node.parent, answer);
            disconnect(node.parent, node.parent_port);
            node.parent.close();
            break;
        }
        else if (command == "clear") {
            int child;
            request >> child;
            std::string answer = node.treeclear(child);
            send_message(node.parent, answer);
        }
    }
    return 0;
}
```

Демонстрация работы программы

```
Hello there! Please check out the task commands:
        create id: for creating a new calculation node
        exec id n n1 n2... n: for calculating a sum
        ping id: for checking availabilty node
        kill id: for killing a calculation node
Please enter your command: create 1
Ok: 6899
Please enter your command: create 2
Ok: 6902
Please enter your command: create 3
Ok: 6905
Please enter your command: create 4
Ok: 6908
Please enter your command: create 5
Ok: 6911
Please enter your command: kill 1
Ok
Please enter your command: ping 1
Error: Not found
Please enter your command: exec 1 2 1 2
Error:1: Not found
Please enter your command: ping 2
Ok: 1
Please enter your command:
```



Выводы

Программа реализует распределенную систему с использованием асинхронной обработки запросов на основе технологии ZeroMQ, обеспечивая взаимодействие между управляющим узлом и вычислительными узлами, организованными в идеально сбалансированное бинарное дерево. Она поддерживает команды для создания, удаления узлов, выполнения вычислений и проверки их доступности, сохраняя работоспособность даже при удалении части узлов. Архитектура системы демонстрирует высокую отказоустойчивость, масштабируемость и интуитивное управление, делая программу подходящей для распределенных вычислительных задач.