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**

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

**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:

Изображение выглядит как круг, зарисовка, диаграмма, рисунок

Автоматически созданное описание

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: Сan 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**

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описание

**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.