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»**

**Тема работы**

**“Брокеры сообщений”**

Студент: Ковриженков Дмитрий Олегович

Группа: М8О-203Б-23

Вариант: 43

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Постановка задачи**

**Задача:** 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

Топология: дерево общего вида.

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

Команда проверки 2 Формат команды: ping id Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

**Общие сведения**

В данной лабораторной работе реализована система распределенных вычислений на основе ZeroMQ. Программа состоит из клиента (client.cpp) и рабочих процессов (worker.cpp), взаимодействующих через сетевые сокеты. Структура системы Клиент (client.cpp): Отправляет команды для создания, удаления и управления рабочими процессами. Подключается к рабочим процессам через ZeroMQ. Поддерживает команды: create <id> – создать новый рабочий процесс. ping <id> – проверить доступность процесса. exec <id> <число\_элементов> <числа> – вычислить сумму чисел. remove <id> – удалить рабочий процесс. exit – завершить все процессы. Рабочие процессы (worker.cpp): Принимают команды от клиента через ZeroMQ. Обрабатывают команду exec, вычисляя сумму переданных чисел. Отвечают на команду ping, подтверждая свою активность. Завершают работу по команде remove. Модуль сетевого взаимодействия (net\_func.h): Реализует функции bind(), connect(), send\_message(), reseave(), обеспечивающие работу ZeroMQ. Структура узлов (node.h): Реализует класс Node для управления рабочими процессами. Поддерживает операции создания, удаления и взаимодействия с дочерними узлами. Сборка проекта (CMakeLists.txt): Использует ZeroMQ для сетевого взаимодействия. Поддерживает тестирование с GTest.

**Вывод**

В ходе выполнения лабораторной работы были изучены: Основы распределенных вычислений с использованием ZeroMQ. Межпроцессное взаимодействие через TCP-сокеты. Работа с асинхронными рабочими процессами. Обработка команд пользователя в клиенте. Динамическое управление рабочими процессами с помощью system() и pkill. Эта лабораторная работа демонстрирует принципы построения распределенных систем, обработки сетевых сообщений и управления процессами в многозадачной среде.

Приложение

src/client.cpp

#include "../include/net\_func.h"

#include <iostream>

#include <zmq.hpp>

#include <sstream>

#include <vector>

#include <unordered\_set>

#include <unistd.h>

std::unordered\_set<int> active\_nodes;

int main() {

zmq::context\_t context(1);

zmq::socket\_t socket(context, ZMQ\_REQ);

std::string command;

std::cout << "Client started. Enter commands (create, ping, exec, remove, exit):" << std::endl;

while (true) {

std::cout << "> ";

std::getline(std::cin, command);

std::istringstream iss(command);

std::string cmd;

iss >> cmd;

if (cmd == "create") {

int id;

iss >> id;

if (active\_nodes.find(id) != active\_nodes.end()) {

std::cout << "Error: Worker " << id << " already exists" << std::endl;

continue;

}

std::string worker\_command = "./worker " + std::to\_string(id) + " -1 &";

int status = system(worker\_command.c\_str());

if (status == -1) {

std::cout << "Error: Failed to start worker" << std::endl;

continue;

}

sleep(1);

my\_net::connect(&socket, 4040 + id);

my\_net::send\_message(&socket, "ping " + std::to\_string(id));

std::string response = my\_net::reseave(&socket);

if (response == "Ok: 1") {

active\_nodes.insert(id);

std::cout << "Worker " << id << " started successfully on port " << (4040 + id) << std::endl;

} else {

std::cout << "Error: Worker did not respond" << std::endl;

}

}

else if (cmd == "ping") {

int id;

iss >> id;

if (active\_nodes.find(id) == active\_nodes.end()) {

std::cout << "Error: Worker " << id << " not found" << std::endl;

continue;

}

my\_net::send\_message(&socket, "ping " + std::to\_string(id));

std::string response = my\_net::reseave(&socket);

std::cout << response << std::endl;

}

else if (cmd == "exec") {

int id, n;

iss >> id >> n;

if (active\_nodes.find(id) == active\_nodes.end()) {

std::cout << "Error: Worker " << id << " not found" << std::endl;

continue;

}

std::vector<int> numbers(n);

for (int i = 0; i < n; ++i) {

iss >> numbers[i];

}

std::ostringstream oss;

oss << "exec " << n;

for (int num : numbers) {

oss << " " << num;

}

my\_net::send\_message(&socket, oss.str());

std::string response = my\_net::reseave(&socket);

std::cout << response << std::endl;

}

else if (cmd == "remove") {

int id;

iss >> id;

if (active\_nodes.find(id) == active\_nodes.end()) {

std::cout << "Error: Worker " << id << " not found" << std::endl;

continue;

}

my\_net::send\_message(&socket, "remove " + std::to\_string(id));

std::string response = my\_net::reseave(&socket);

std::cout << response << std::endl;

std::string kill\_command = "pkill -SIGTERM -f \"./worker " + std::to\_string(id) + " -1\"";

system(kill\_command.c\_str());

usleep(500000);

std::string force\_kill\_command = "pkill -9 -f \"./worker " + std::to\_string(id) + " -1\"";

system(force\_kill\_command.c\_str());

active\_nodes.erase(id);

}

else if (cmd == "exit") {

std::cout << "Exiting... Removing all workers." << std::endl;

for (int id : active\_nodes) {

std::cout << "Removing worker " << id << std::endl;

my\_net::send\_message(&socket, "remove " + std::to\_string(id));

std::string response = my\_net::reseave(&socket);

std::cout << response << std::endl;

std::string kill\_command = "pkill -SIGTERM -f \"./worker " + std::to\_string(id) + " -1\"";

system(kill\_command.c\_str());

usleep(500000);

std::string force\_kill\_command = "pkill -9 -f \"./worker " + std::to\_string(id) + " -1\"";

system(force\_kill\_command.c\_str());

}

system("pkill -SIGTERM -f ./worker");

usleep(500000);

system("pkill -9 -f ./worker");

std::cout << "All workers removed. Exiting client." << std::endl;

active\_nodes.clear();

break;

}

else {

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

}

}

return 0;

}

src/worker.cpp

#include "../include/node.h"

#include "../include/net\_func.h"

#include <iostream>

#include <sstream>

#include <vector>

#include <zmq.hpp>

#include <csignal>

#include <unistd.h>

int my\_id = 0;

bool running = true;

void handle\_signal(int signum) {

std::cout << "Worker " << my\_id << " shutting down..." << std::endl;

running = false;

}

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

if (argc != 3) {

return -1;

}

my\_id = atoi(argv[1]);

Node me(my\_id, atoi(argv[2]));

zmq::context\_t context(1);

zmq::socket\_t socket(context, ZMQ\_REP);

int port = 4040 + my\_id;

std::string address = "tcp://127.0.0.1:" + std::to\_string(port);

try {

socket.bind(address);

} catch (const zmq::error\_t& e) {

std::cerr << "Error: Failed to bind socket on port " << port << " - " << e.what() << std::endl;

return -1;

}

std::cout << "Worker " << my\_id << " started on port " << port << std::endl;

signal(SIGTERM, handle\_signal);

signal(SIGINT, handle\_signal);

while (running) {

zmq::message\_t request;

try {

socket.recv(request, zmq::recv\_flags::none);

} catch (const zmq::error\_t& e) {

if (running) {

std::cerr << "Receive error: " << e.what() << std::endl;

}

break;

}

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

std::istringstream iss(message);

std::string cmd;

iss >> cmd;

if (cmd == "ping") {

my\_net::send\_message(&socket, "Ok: 1");

}

else if (cmd == "exec") {

int n;

iss >> n;

std::vector<int> numbers(n);

for (int i = 0; i < n; ++i) {

iss >> numbers[i];

}

if (numbers.size() != static\_cast<size\_t>(n)) {

my\_net::send\_message(&socket, "Error: Invalid number of arguments");

continue;

}

int result = me.Exec\_sum(numbers);

my\_net::send\_message(&socket, "Ok: " + std::to\_string(result));

}

else if (cmd == "remove") {

my\_net::send\_message(&socket, "Ok");

running = false;

}

else {

my\_net::send\_message(&socket, "Error: Unknown command");

}

}

std::cout << "Worker " << my\_id << " exited successfully." << std::endl;

return 0;

}

include/net\_func.h

#pragma once

#include <iostream>

#include <zmq.hpp>

#include <sstream>

#include <string>

namespace my\_net {

#define MY\_PORT 4040

#define MY\_IP "tcp://127.0.0.1:"

int bind(zmq::socket\_t \*socket, int id) {

int port = MY\_PORT + id;

while (true) {

std::string adress = MY\_IP + std::to\_string(port);

try {

socket->bind(adress);

break;

} catch (...) {

port++;

}

}

return port;

}

void connect(zmq::socket\_t \*socket, int port) {

std::string adress = MY\_IP + std::to\_string(port);

socket->connect(adress);

}

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());

try {

socket->send(message, zmq::send\_flags::none);

} catch (...) {}

}

std::string reseave(zmq::socket\_t \*socket) {

zmq::message\_t message;

try {

socket->recv(message, zmq::recv\_flags::none);

} catch (const zmq::error\_t& e) {

std::cerr << "Receive error: " << e.what() << std::endl;

return "Error: Receive failed";

}

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

}

}

include/node.h

#pragma once

#include <iostream>

#include <unordered\_map>

#include <vector>

class Node {

private:

int id;

int parent\_id;

std::unordered\_map<int, Node\*> children;

public:

explicit Node(int \_id, int \_parent\_id = -1) : id(\_id), parent\_id(\_parent\_id) {}

bool Create\_child(int child\_id) {

if (children.find(child\_id) != children.end()) {

return false;

}

children[child\_id] = new Node(child\_id, id);

return true;

}

bool Remove\_child(int child\_id) {

if (children.find(child\_id) == children.end()) {

return false;

}

delete children[child\_id];

children.erase(child\_id);

return true;

}

bool Ping\_child(int child\_id) {

return children.find(child\_id) != children.end();

}

int Exec\_sum(const std::vector<int>& numbers) {

int sum = 0;

for (int num : numbers) {

sum += num;

}

return sum;

}

std::string Send(std::string msg, int \_id) {

return "Ok: Simulated response for " + msg;

}

~Node() {

for (auto& pair : children) {

delete pair.second;

}

}

};

CMakeLists.txt

project(ZMQProject)

set(CMAKE\_CXX\_STANDARD 17)

set(CMAKE\_CXX\_STANDARD\_REQUIRED True)

include\_directories(/opt/homebrew/include)

include\_directories(${CMAKE\_SOURCE\_DIR}/include)

link\_directories(/opt/homebrew/lib)

add\_executable(client src/client.cpp)

add\_executable(worker src/worker.cpp)

target\_link\_libraries(client zmq)

target\_link\_libraries(worker zmq)

find\_package(GTest REQUIRED)

find\_package(Threads REQUIRED)

include\_directories(${GTEST\_INCLUDE\_DIRS})

add\_executable(tests

tests/test\_main.cpp

include/node.h

include/net\_func.h

)

target\_link\_libraries(tests PRIVATE ${GTEST\_LIBRARIES} zmq Threads::Threads)

enable\_testing()

add\_test(NAME ZMQTests COMMAND tests)

add\_custom\_target(clean COMMAND ${CMAKE\_COMMAND} -E remove -f client worker tests)

Пример вывода:

