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) {
```

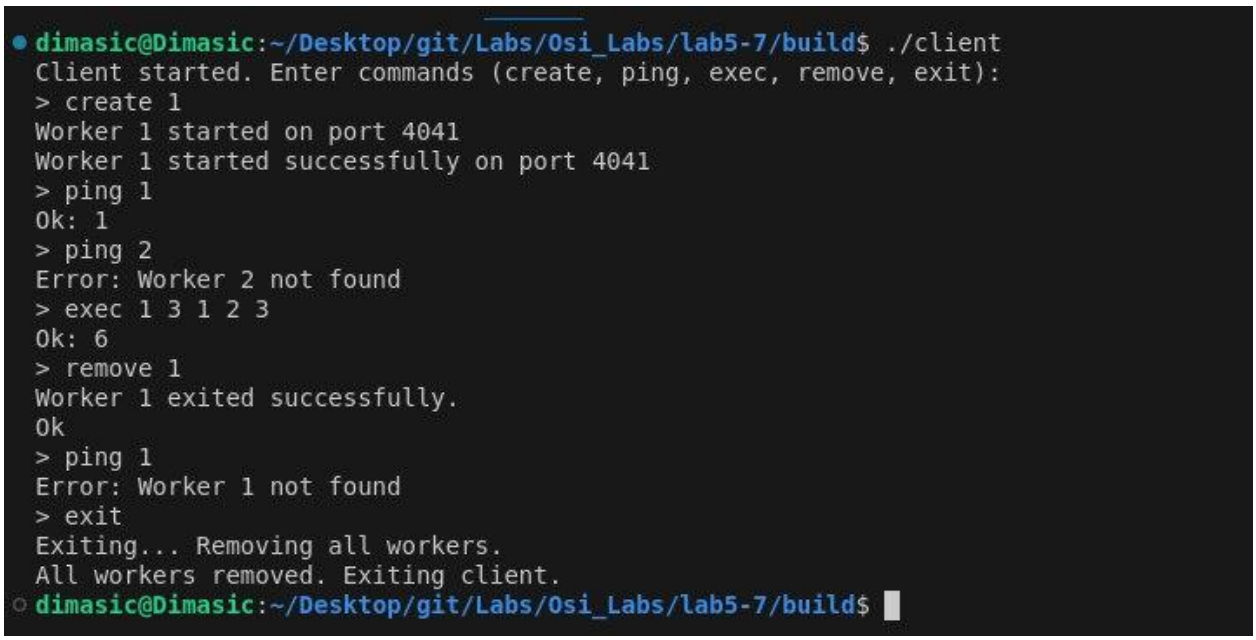
```
        if (children.find(child_id) == children.end()) {
            return false;
        }
        delete children[child_id];
        children.erase(child_id);
        return true;
    }

    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:


dimasic@Dimasic:~/Desktop/git/Labs/Osi_Labs/lab5-7/build$ ./client
Client started. Enter commands (create, ping, exec, remove, exit):
> create 1
Worker 1 started on port 4041
Worker 1 started successfully on port 4041
> ping 1
Ok: 1
> ping 2
Error: Worker 2 not found
> exec 1 3 1 2 3
Ok: 6
> remove 1
Worker 1 exited successfully.
Ok
> ping 1
Error: Worker 1 not found
> exit
Exiting... Removing all workers.
All workers removed. Exiting client.
dimasic@Dimasic:~/Desktop/git/Labs/Osi_Labs/lab5-7/build$