Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №6-8 по курсу

«Операционные системы»

Студент: Рокотянский А.Е.
Группа: М8О-201Б-21
Вариант: 1-2-2
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Mikhail-cWc/OS_mai/tree/main/lab6

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

- Управлении серверами сообщений (№6)
- Применение отложенных вычислений (№7)
- Интеграция программных систем друг с другом (№8)

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. Список основных поддерживаемых команд: create, exec, kill, ping.

Общие сведения о программе

Программа распределительного узла компилируется из файла control.cpp, программа вычислительного узла компилируется из файла counting.cpp. В программе используется библиотека для работы с потоками и мьютексами, а также сторонняя библиотека для работы с сервером сообщений ZeroMQ.

Исходный код

control.cpp

```
#include <unistd.h>
#include <sstream>
#include <set>

#include "zmq_functions.h"
#include "topology.h"

int main() {
    topology network;
    std::vector<zmq::socket_t> branches;
    zmq::context_t context;

    std::string cmd;
    while (std::cin >> cmd) {

        if (cmd == "create") {
            int node_id, parent_id;
```

```
            std::cin >> node_id >> parent_id;

            if (network.find(node_id) != -1) {
                std::cout << "Error: already exists" << std::endl;
            }
             else if (parent_id == -1) {
                pid_t pid = fork();
                if (pid < 0) {
                    perror("Can't create new process");
                    return -1;
                }
                if (pid == 0) {
                    if (execl("./counting", "./counting", std::to_string(node_id).c_str(), NULL) < 0) {
                        perror("Can't execute new process");
                        return -2;
                    }
                }

                branches.emplace_back(context, ZMQ_REQ);
                branches[branches.size() - 1].setsockopt(ZMQ_SNDTIMEO, 5000);
                bind(branches[branches.size() - 1], node_id);
                send_message(branches[branches.size() - 1], std::to_string(node_id) + "pid");

                std::string reply = receive_message(branches[branches.size() - 1]);
                std::cout << reply << std::endl;
                network.insert(node_id, parent_id);
            }
            else if (network.find(parent_id) == -1) {
                std::cout << "Error: parent not found" << std::endl;
            }
            else {
                int branch = network.find(parent_id);
                send_message(branches[branch], std::to_string(parent_id) + "create " +
std::to_string(node_id));

                std::string reply = receive_message(branches[branch]);
                std::cout << reply << std::endl;
                network.insert(node_id, parent_id);
            }
        }
        else if (cmd == "exec") {
            int dest_id;
            std::string numbers;
            std::cin >> dest_id;
            std::getline(std::cin, numbers);
            int branch = network.find(dest_id);
            if (branch == -1) {
                std::cout << "ERROR: incorrect node id" << std::endl;
            }
            else {
                send_message(branches[branch], std::to_string(dest_id) + "exec " + numbers);
                std::string reply = receive_message(branches[branch]);
                std::cout << reply << std::endl;
            }
        }
        else if (cmd == "kill") {
            int id;
            std::cin >> id;
            int branch = network.find(id);
            if (branch == -1) {
                std::cout << "ERROR: incorrect node id" << std::endl;
            }
            else {
                bool is_first = (network.get_first_id(branch) == id);
                send_message(branches[branch], std::to_string(id) + " kill");

                std::string reply = receive_message(branches[branch]);
                std::cout << reply << std::endl;
```

```
                network.erase(id);
                if (is_first) {
                    unbind(branches[branch], id);
                    branches.erase(branches.begin() + branch);
                }
            }
        }
        else if (cmd == "ping") {
            std::set<int> available_nodes;
            for (size_t i = 0; i < branches.size(); ++i) {
                int first_node_id = network.get_first_id(i);
                send_message(branches[i], std::to_string(first_node_id) + " ping");

                std::string received_message = receive_message(branches[i]);
                std::istringstream reply(received_message);
                int node;
                while(reply >> node) {
                    available_nodes.insert(node);
                }
            }
            std::cout << "OK: ";
            if (available_nodes.empty()) {
                std::cout << "No available nodes" << std::endl;
            }
            else {
                for (auto v : available_nodes) {
                    std::cout << v << " ";
                }
                std::cout << std::endl;
            }
        }
        else if (cmd == "exit") {
            for (size_t i = 0; i < branches.size(); ++i) {
                int first_node_id = network.get_first_id(i);
                send_message(branches[i], std::to_string(first_node_id) + " kill");

                std::string reply = receive_message(branches[i]);
                if (reply != "OK") {
                    std::cout << reply << std::endl;
                }
                else {
                    unbind(branches[i], first_node_id);
                }
            }
            exit(0);
        }
        else {
            std::cout << "Incorrect cmd" << std::endl;
        }
    }
}
```

counting.cpp

```
#include <unordered_map>
#include <unistd.h>
#include <sstream>
#include <unordered_map>

#include "zmq_functions.h"

int main(int argc, char* argv[]) {
    if (argc != 2 && argc != 3) {
        throw std::runtime_error("Wrong args for counting node");
```

```
    }
    int cur_id = std::atoi(argv[1]);
    int child_id = -1;
    if (argc == 3) {
        child_id = std::atoi(argv[2]);
    }

    std::unordered_map<std::string, int> dictionary;

    zmq::context_t context;
    zmq::socket_t parent_socket(context, ZMQ_REP);
    connect(parent_socket, cur_id);

    zmq::socket_t child_socket(context, ZMQ_REQ);
    child_socket.setsockopt(ZMQ_SNDTIMEO, 5000);
    if (child_id != -1) {
        bind(child_socket, child_id);
    }

    std::string message;
    while (true) {
        message = receive_message(parent_socket);
        std::istringstream request(message);
        int dest_id;
        request >> dest_id;

        std::string cmd;
        request >> cmd;

        if (dest_id == cur_id) {

            if (cmd == "pid") {
                send_message(parent_socket, "OK: " + std::to_string(getpid()));
            }

            else if (cmd == "create") {
                int new_child_id;
                request >> new_child_id;
                if (child_id != -1) {
                    unbind(child_socket, child_id);
                }
                bind(child_socket, new_child_id);
                pid_t pid = fork();
                if (pid < 0) {
                    perror("Can't create new process");
                    return -1;
                }
                if (pid == 0) {
                    execl("./counting", "./counting", std::to_string(new_child_id).c_str(),
std::to_string(child_id).c_str(), NULL);
                    perror("Can't execute new process");
                    return -2;
                }
                send_message(child_socket, std::to_string(new_child_id) + "pid");
                child_id = new_child_id;
                send_message(parent_socket, receive_message(child_socket));
            }
            else if (cmd == "exec") {
                int sum = 0;
                std::string number;
                while (request >> number) {
                    sum += std::stoi(number);
                }
                send_message(parent_socket, "OK: " + std::to_string(cur_id) + ": " + std::to_string(sum));
            }

            else if (cmd == "ping") {
                std::string reply;
```

```
                if (child_id != -1) {
                    send_message(child_socket, std::to_string(child_id) + " ping");
                    std::string msg = receive_message(child_socket);
                    reply += " " + msg;
                }
                send_message(parent_socket, std::to_string(cur_id) + reply);
            }
            else if (cmd == "kill") {
                if (child_id != -1) {
                    send_message(child_socket, std::to_string(child_id) + " kill");
                    std::string msg = receive_message(child_socket);
                    if (msg == "OK") {
                        send_message(parent_socket, "OK");
                    }
                    unbind(child_socket, child_id);
                    disconnect(parent_socket, cur_id);
                    break;
                }
                send_message(parent_socket, "OK");
                disconnect(parent_socket, cur_id);
                break;
            }
        }
        else if (child_id != -1) {
            send_message(child_socket, message);
            send_message(parent_socket, receive_message(child_socket));
            if (child_id == dest_id && cmd == "kill") {
                child_id = -1;
            }
        }
        else {
            send_message(parent_socket, "Error: node is unavailable");
        }
    }
}
```

topology.h

```
#include <list>
#include <stdexcept>

class topology {
private:
    std::list<std::list<int>> container;

public:
    void insert(int id, int parent_id) {
        if (parent_id == -1) {
            std::list<int> new_list;
            new_list.push_back(id);
            container.push_back(new_list);
        }
        else {
            int list_id = find(parent_id);
            if (list_id == -1) {
                throw std::runtime_error("Wrong parent id");
            }
            auto it1 = container.begin();
            std::advance(it1, list_id);
            for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
                if (*it2 == parent_id) {
                    it1->insert(++it2, id);
                    return;
                }
            }
        }
    }
```

```
    int find(int id) {
        int cur_list_id = 0;
        for (auto it1 = container.begin(); it1 != container.end(); ++it1) {
            for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
                if (*it2 == id) {
                    return cur_list_id;
                }
            }
            ++cur_list_id;
        }
        return -1;
    }

    void erase(int id) {
        int list_id = find(id);
        if (list_id == -1) {
            throw std::runtime_error("Wrong id");
        }
        auto it1 = container.begin();
        std::advance(it1, list_id);
        for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
            if (*it2 == id) {
                it1->erase(it2, it1->end());
                if (it1->empty()) {
                    container.erase(it1);
                }
                return;
            }
        }
    }

    int get_first_id(int list_id) {
        auto it1 = container.begin();
        std::advance(it1, list_id);
        if (it1->begin() == it1->end()) {
            return -1;
        }
        return *(it1->begin());
    }
};
```

Zmq_functions.h

```
#include <zmq.hpp>
#include <iostream>

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;
    bool chars_read;
    try {
        chars_read = socket.recv(&message);
    }
    catch (...) {
        chars_read = false;
    }
    if (chars_read == 0) {
        return "Error: node is unavailable [zmq_func]";
    }
    std::string received_msg(static_cast<char*>(message.data()), message.size());
    return received_msg;
}
```

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

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

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

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

Выводы

Составлена и отлажена программа на языке C++, осуществляющая отложенные вычисления на нескольких вычислительных узлах. Пользователь управляет программой через распределительный узел, который перенаправляет запросы в асинхронном режиме.