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

Лабораторная работа №6 по курсу
«Операционные системы»

Студент: Шумилова Александра
Группа: М8О-207Б-21
Вариант: 6
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/tonsoleils/OS

Постановка задачи

Цель работы

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.

Вариант 6: 1, 2, 3.

Общие сведения о программе

Программы компилируются из файлов main.cpp, node.cpp с использованием заголовочных файлов topology.h и zmq_f.h с помощью cmake и CMakeFile. Также используется заголовочные файлы: unistd.h, iostream, vector, zmq.hpp, sstream, string, map.

Общий метод и алгоритм решения

В файле node.cpp находится реализация ноды. На вход подаётся current_id и child_id. Далее нода создается и обрабатывает входящие сообщения, содержащие команды, согласно условию задания: heartbit, create, remove, exec. После обработки команды и выполнения необходимых действий, она отсылает сообщение с результатом работы.

В файле zmq_f.h находится реализация необходимого функционала для работы с ZMQ: отправка сообщений, получение сообщений, коннект и дисконнект, бинд и анбинд.

В файле topology.h находится реализация топологии согласно варианту: 1 управляющий узел, вычислительные узлы находятся в списках. Класс Topology содержит контейнер нод,

представляющий из себя список списков, а так же необходимые методы для работы: вставка ноды, поиск ноды по id, удаление ноды по id, получение id первой ноды в списке.

В файле main.cpp реализована основная работа по распределению комманд, подающихся на вход. В начале происходит необходимая подготовка (создание основного сокета, подготовка вектора сокетов и т.д.). Далее происходит обработка команд, перечисленных выше, необходимые подготовительные действия перед отправкой сообщения, сама отправка сообщение, получение сообщения с результатом выполнения команды и вывод в стандартный поток вывода.

Исходный код

topology.h

```
#ifndef LAB6_TOPOLOGY_H
#define LAB6_TOPOLOGY_H

#include <list>
#include <stdexcept>

class Topology { // Топология
private:
    std::list<std::list<int>> container; // список списков, наши ноды

public:
    void Insert(int id, int parent_id) { // вставить новую ноду. id - ид ноды, parent_id - ид родителя
        if (parent_id == -1) { // если родитель - управляющий узел
            std::list<int> new_list; // подготавливаем новый список
            new_list.push_back(id); // вставляем туда ноду
            container.push_back(new_list); // и отправляем в хранилище нод
        } else {
            int list_id = Find(parent_id); // ищем родителя по айди
            if (list_id == -1) {
                throw std::runtime_error("Wrong parent id"); // если не нашли - ошибка
            }
            auto it1 = container.begin();
```

```
        std::advance(it1, list_id);
        for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
            if (*it2 == parent_id) { // иттерируемся по хранилищу и ищем родителя.
                it1->insert(++it2, id); // если нашли родителя - вставляем.
                return;
            }
        }
    }
}

int Find(int id) { // ищем ноду по айди
    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) { // иттерируемся по хранилищу и ищем нужную ноду по айди
                return cur_list_id; // если нашли - возвращаем её
            }
        }
        ++cur_list_id;
    }
    return -1; // если не нашли - возвращаем -1
}

void Erase(int id) { // удалить ноду по айди
    int list_id = Find(id); // ищем ноду по айди
    if (list_id == -1) {
        throw std::runtime_error("Wrong id"); // если не нашли - пробрасываем ошибку.
    }
    auto it1 = container.begin();
    std::advance(it1, list_id);
    for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
        if (*it2 == id) { // иттерируемся по списку и ищем ноду
```

```
            it1->erase(it2, it1->end()); // если нашли - удаляем её
            if (it1->empty()) { // если список после удаления оказался пустым
                container.erase(it1); // удаляем список
            }
            return;
        }
    }
}

int GetFirstId(int list_id) { // получить первый id в списке
    auto it1 = container.begin(); // итератор
    std::advance(it1, list_id); // смещаем итератор
    if (it1->begin() == it1->end()) { // если итератор попал на конец коллекции
        return -1; // возвращаем -1
    }
    return *(it1->begin()); // возвращаем первый элемент
}
};

#endif // LAB6_TOPOLOGY_H
```

zmq_f.h

```
#ifndef LAB6_ZMQ_F_H
#define LAB6_ZMQ_F_H

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

const int MAIN_PORT = 4040;
```

```
void send_message(zmq::socket_t& socket, const std::string& msg) { // отправить сообщение
    zmq::message_t message(msg.size()); // подготавливаем переменную
    memcpy(message.data(), msg.c_str(), msg.size()); // копирует size байтов из msg.c_str() в message.data()
    socket.send(message); // возврат результата, отсылает очередь сообщений, созданных в message
}

std::string receive_message(zmq::socket_t& socket) { // получить сообщение
    zmq::message_t message; // подготавливаем переменную
    int chars_read; // переменная для количества считанных символов
    try {
        chars_read = (int)socket.recv(&message);  // получение сообщений
    }
    catch (...) {
        chars_read = 0; // отлавливаем исключение
    }
    if (chars_read == 0) {
        return "Error"; // если поймали исключение или считали 0 символов - ошибка
    }
    std::string received_msg(static_cast<char*>(message.data()), message.size());
    return received_msg;
}

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

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

```
}

void bind(zmq::socket_t& socket, int id) {
    std::string adress = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
    socket.bind(adress); // Асинхронно привязывается к конечной точке (endpoint).
}

void unbind(zmq::socket_t& socket, int id) {
    std::string adress = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
    socket.unbind(adress); // Отменяет привязку к ранее привязанной конечной точке.
}

#endif // LAB6_ZMQ_F_H
```

node.cpp

```
#include <unistd.h>
#include <iostream>
#include <string>
#include <vector>
#include <map>

#include "zmq_f.h"

int main(int argc, char* argv[])
{
    if (argc != 2 && argc != 3) { // если нам не передали аргументы - ошибка
        std::cout << "Wrong arguments. Not enough parameters!\n";
        exit(1);
    }
    int current_id = std::atoi(argv[1]); // считываем первый аргумент - айди ноды
    int child_id = -1; // айди ноды-потомка
```

```
    if (argc == 3) { // если нам передали два аргумента, значит передали айди потомка
        child_id = std::atoi(argv[2]); // считываем его
    }
    std::string adr = argv[1];


    zmq::context_t context; // контекст
    zmq::socket_t parent_socket(context, ZMQ_REP); // создаем сокет
    connect(parent_socket, current_id); // коннектимся к сокету


    zmq::socket_t child_socket(context, ZMQ_REQ); // создаем сокет потомка
    child_socket.setsockopt(ZMQ_SNDTIMEO, 5000); // Maximum time before a send operation returns with EAGAIN
    parent_socket.setsockopt(ZMQ_SNDTIMEO, 5000); // Maximum time before a send operation returns with EAGAIN


    std::string message;


    std::map<std::string, int> storage; // локальный словарь ноды


    while (1) {
        zmq::message_t message_main;
        message = receive_message(parent_socket); // получаем сообщение
        std::string recieved_message(static_cast<char*>(message_main.data()), message_main.size());
        std::istringstream request(message); // отправляем его в строковый поток
        int dest_id;


        request >> dest_id; // считываем айди ноды


        std::string command;
        request >> command; // считываем комманду


        if (command == "heartbit") {
```

```
            std::string ans = std::to_string(current_id) + ":Ok; "; // подготавливаем сообщение - ответ
            if (child_id != -1) {
                int timeout;
                request >> timeout; // считываем таймаут
                int fl = 0;
                for (int i = 0; i < 4; i++) { // 4 раза пингуем ноду
                    send_message(child_socket, message);
                    std::string repl = receive_message(child_socket);
                    if (repl != "Error") { // если не получили ошибку
                        ans += repl;
                        fl = 1;
                        break; // прекращаем
                    }
                    sleep(timeout / 1000); // ждём timeout / 1000 секунд
                }
                if (fl == 0) {
                    ans += "Node " + std::to_string(child_id) + " is not avail"; // если нода возвращает Error
                }
            }
            send_message(parent_socket, ans); // отправляем ответ
        } else if (dest_id == current_id) {
            if (command == "pid") { // если команда - pid
                send_message(parent_socket, "OK: " + std::to_string(getpid()));
            } else if (command == "create") { // если команда - create
                int new_child_id;
                request >> new_child_id; // считываем id новой ноды
                if (child_id != -1) {
                    unbind(child_socket, child_id); // анбиндим сокет
                }
                bind(child_socket, new_child_id); // биндим сокет на новый id
```

```
            pid_t pid = fork(); // создаем новый процесс
            if (pid < 0) { // если ошибка
                perror("Can't create new process!\n");
                exit(1);
            }
            if (pid == 0) {
                // запускаем новую ноду
                execl("node", "node", std::to_string(new_child_id).c_str(),
std::to_string(child_id).c_str(), NULL);
                perror("Can't create new process!\n");
                exit(1);
            }
            send_message(child_socket, std::to_string(new_child_id) + "pid"); // отправляем
сообщение с новым id
            child_id = new_child_id;
            send_message(parent_socket, receive_message(child_socket)); // отправляем
сообщение родителю
        } else if (command == "remove") { // если команда - удалить ноду
            send_message(parent_socket, "OK"); // отправляем сообщение
            disconnect(parent_socket, current_id); // и дисконнектимся от сокета
            break;
        } else if (command == "exec") { // если команда - exec
            std::string msg = "OK:" + std::to_string(dest_id); // подготавливаем сообщение
//              request >> mapKey;
//              request >> mapValue;
            size_t count = 0; // количество параметров (1 или 2)
            std::string mapKey; // ключ
            int mapValue; // значение

            request >> count; // считываем количество параметров
            request >> mapKey; // считываем ключ (он по-любому придёт)
            if (count == 2) { // если пришло 2 параметра
                request >> mapValue; // считываем значение
```

```
            }


            if (count == 1) { // если пришёл только ключ
                if (storage.count(mapKey) > 0) { // если ключ присутствует в словаре
                    msg += ": " + std::to_string(storage.at(mapKey)); // добавляем в сообщение
значение
                } else {
                    msg += ": '" + mapKey + "' not found"; // иначе возвращаем ошибку
                }
            } else if (count == 2) { // если пришло 2 параметра
                storage[mapKey] = mapValue; // добавляем в словарь по ключу значение
            }
            send_message(parent_socket, msg); // отправляем сообщение
        }
    } else if (child_id != -1) {
        send_message(child_socket, message);
        send_message(parent_socket, receive_message(child_socket));
        if (child_id == dest_id && command == "remove") {
            child_id = -1;
        }
    } else {
        send_message(parent_socket, "Error: node is unavailable!\n"); // ошибка, что нода не
доступна
    }
  }
}
```

main.cpp

```
#include <unistd.h>
#include <iostream>
#include <vector>
#include <zmq.hpp>
```

```
#include <sstream>


#include "topology.h"
#include "zmq_f.h"


int main() {
    Topology network; // наша топология
    std::vector<zmq::socket_t> branches; // вектор сокетов
    zmq::context_t context; // контекст. Класс context_t инкапсулирует функции, связанные с инициализацией и завершением контекста
    std::string command; // переменная для комманды
    zmq::socket_t main_socket(context, ZMQ_REP); // главный сокет
    std::string message;
    while (std::cin >> command) { // пока мы получаем на вход комманды
        if (command == "create") { // если команда - создать ноду
            int node_id, parent_id;
            std::cin >> node_id >> parent_id; // считываем айди ноды и айди ноды-родителя


            if (network.Find(node_id) != -1) {  // Поиск id ноды среди существующих
                std::cout << "Error: already exists!\n"; // если существует - ошибка
            } else if (parent_id == -1) { // если родитель - управляющий узел
                pid_t pid = fork();  // Создание дочернего узла
                if (pid < 0) {
                    perror("Can't create new process!\n");
                    exit(EXIT_FAILURE);
                } if (pid == 0) {
                    execl("node", "node", std::to_string(node_id).c_str(), NULL); // запускаем ноду
                    perror("Can't execute new process!\n");
                    exit(EXIT_FAILURE);
                }
                branches.emplace_back(context, ZMQ_REQ); // добавляем в вектор
```

```
            branches[branches.size() - 1].setsockopt(ZMQ_SNDTIMEO, 5000); // добавляем опцию таймаута
            bind(branches[branches.size() - 1], node_id); // биндим сокет
            send_message(branches[branches.size() - 1], std::to_string(node_id) + "pid"); // отправляем сообщение

            std::string reply = receive_message(branches[branches.size() - 1]); // и получаем ответ - pid
            std::cout << reply << "\n"; // выводим его
            network.Insert(node_id, parent_id); // и вставляем в топологию новую ноду
        } else if (network.Find(parent_id) == -1) { // если не нашли родителя
            std::cout << "Error: parent not found!\n";
        } else { // если родитель - не вычислительыний узел
            int branch = network.Find(parent_id); // ищем ноду
            // и отправляем сообщение о создании ноды
            send_message(branches[branch], std::to_string(parent_id) + "create " + std::to_string(node_id));

            std::string reply = receive_message(branches[branch]); // получаем в ответ pid
            std::cout << reply << "\n"; // выводим

            network.Insert(node_id, parent_id); // вставляем в топологию ноду
        }
    } else if (command == "remove") { // если команда - удалить ноду
        int id;
        std::cin >> id; // считываем id
        int branch = network.Find(id); // ищем ноду по айди
        if (branch == -1) {
            std::cout << "Error: incorrect node id!\n";
        } else {
            bool is_first = (network.GetFirstId(branch) == id); // проверяем, первая ли нода
            send_message(branches[branch], std::to_string(id) + " remove"); // оправляем сообщение о удалении
```

```
            std::string reply = receive_message(branches[branch]);
            std::cout << reply << std::endl; // получаем и выводим ответ
            network.Erase(id); // удаляем ноду из топологии
            if (is_first) { // если это первая нода
                unbind(branches[branch], id); // анбиндим сокет
                branches.erase(std::next(branches.begin(), branch)); // удаляем всё что после
            }
        }
    } else if (command == "exec") { // если команда - exec
        size_t count = 0; // количество параметров
        int destId;
        std::cin >> destId; // считываем id ноды, на которой запускаем

        std::string s;
        std::getline(std::cin,s); // считываем последующие параметры
        std::istringstream iss(s);

        // тут мы парсим команды для универсальности (чтобы можно было 1 или 2 параметра передавать)
        std::vector<std::string> params;
        std::string param;
        while (iss >> param) {
            count++;
            params.push_back(param);
        }

        int branch = network.Find(destId); // ищем ноду по айди
        if (branch == -1) {
            std::cout << "Error: incorrect node id!\n";
        } else {
            if (params.size() == 2) { // если нам передали 2 параметра
```

```
                send_message(branches[branch], std::to_string(destId) + "exec " +
std::to_string(count) + " " + params[0] + " " + params[1]);
            } else if (params.size() == 1) { // и если 1 параметр
                send_message(branches[branch], std::to_string(destId) + "exec " +
std::to_string(count) + " " + params[0]);
            }
            std::string reply = receive_message(branches[branch]);
            std::cout << reply << "\n"; // получаем и выводим ответ
        }
    } else if (command == "heartbit") { // если команда - heartbit
        int TIME;
        std::cin >> TIME; // считываем таймаут
        for (int i = 0; i < 10; i++) { // для примера - ограничим количество пингов до 10
            for (int i = 0; i < branches.size(); ++i) { // идем по всем сокетам
                // отправляем команду на пинг ноды
                send_message(branches[i], std::to_string(i) + " heartbit " + std::to_string(TIME));
                std::cout << receive_message(branches[i]) << "\n---\n"; // выводим ответ
            }
            sleep(TIME / 1000); // ждём TIME / 1000 секунд
        }
    } else if (command == "exit") { // если команда - выйти
        for (size_t i = 0; i < branches.size(); ++i) { // идём по всем сокетам
            int first_node_id = network.GetFirstId(i); // получаем id первой ноды
            send_message(branches[i], std::to_string(first_node_id) + " remove"); // удаляем её
            std::string reply = receive_message(branches[i]); // получаем ответ
            if (reply != "OK") {
                std::cout << reply << "\n"; // если не всё ок - выводим
            } else {
                unbind(branches[i], first_node_id); // иначе - анбиндим сокет
            }
        }
        exit(0);
```

```
        } else {
            std::cout << "Incorrect command: " << command << "<!\n"; // если ввели херню - вежливо скажем об этом
        }
    }
}
```

Демонстрация работы программы

tonsoleils@LAPTOP-31GE9NQM:/mnt/d/op_sys/os/lab5$ ./main

create 1 -1

OK: 166

create 2 -1

OK: 169

exec 1 myVal

OK:1: 'myVal' not found

exec 1 myVal 1

OK:1

exec 1 myVal

OK:1: 1

exec 2 myVal

OK:2: 'myVal' not found

heartbit 500

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

1:Ok;

2:Ok;

remove 2

OK

heartbit 1000

1:Ok;

1:Ok;

1:Ok;

1:Ok;

1:Ok;

1:Ok;

1:Ok;

1:Ok;

1:Ok;

1:Ok;

exit

Выводы

Проделав данную работу, я научилась принципам управлении серверами сообщений, применения отложенных вычислений, интеграции программных систем друг с другом. В частности, познакомилась с библиотекой ZeroMQ.