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: Николаева Елизавета Сергеевна

Группа: М8О-201Б-20

Вариант: 38

Преподаватель: Миронов Евгений Сергеевич

Москва, 2021

**Содержание**

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

**Репозиторий**

https://github.com/esnikolaeva/OS\_6-8

**Постановка задачи**

Реализовать распределенную систему по асинхронной обработке запросов. В данной 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 убитогомогут стать недоступными, но родительские узлы должны сохранить свою работоспособность. Управляющий отвечает за ввод команд от пользователя и отправку этих команд на вычислительные.  
Вариант 38:  
create id  
exec id subcommand(start |stop|time)

heartbeat time

**Общие сведения о программе**

Для выполнения данной лабораторной работы я предварительно реализовала 4 файла.

**Общий метод и алгоритм решения**

topology.h – реализация 1 топологии (списка списков)

zmq\_functions.h - отдельный файл для функций zero-message queue, сделанный для удобства работы и во избежание загрязнения кода  
control.cpp – файл работы с клиентом  
count.cpp - реализация программы клиента

**Исходный код**

**control.cpp**

#include <iostream>

#include <unistd.h>

#include <sstream>

#include <set>

#include <zmq.hpp>

#include <chrono>

#include <vector>

#include <map>

#include "topology.h"

#include "zmq\_functions.h"

int main() {

Topology network;

std::vector<zmq::socket\_t> branches; //массив сокетов

zmq::context\_t context;

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

std::string comand;

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

std::string message;

while (std::cin >> comand) {

// std::cout <<comand<<std::endl;

if (comand == "create") {

int node\_id, parent\_id;

std::cin >> node\_id >> parent\_id;

if (network.Find(node\_id) != -1) { // Поиск id выч. узла среди существующих

std::cout << "Error: already exists!\n";

} else if (parent\_id == -1) {

pid\_t pid = fork(); // Создание дочернего узла

if (pid < 0) {

perror("Can't create new process!\n");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

execl("./count", "./count", std::to\_string(node\_id).c\_str(), NULL);

perror("Can't execute new process!\n");

exit(EXIT\_FAILURE);

}

//std::cout<<"branches.size before "<<branches.size()<<std::endl;

branches.emplace\_back(context, ZMQ\_REQ);

//std::cout<<"branches.size after "<<branches.size()<<std::endl;

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 << "\n";

network.Insert(node\_id, parent\_id);

} else if (network.Find(parent\_id) == -1) {

std::cout << "Error: parent not found!\n";

} 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 << "\n";

network.Insert(node\_id, parent\_id);

}

} else if (comand == "remove") {

int id;

std::cin >> id;

int branch = network.Find(id); // Проверка, существует ли узел

if (branch == -1) {

std::cout << "Error: incorrect node id!\n";

} else {

bool is\_first = (network.GetFirstId(branch) == id);

send\_message(branches[branch], std::to\_string(id) + " remove");

std::string reply = receive\_message(branches[branch]);

std::cout << reply << "\n";

network.Erase(id);

if (is\_first) {

unbind(branches[branch], id);

branches.erase(branches.begin() + branch);

}

}

} else if (comand == "exec") {

int dest\_id;

std::string subcomand;

std::cin >> dest\_id >> subcomand;

int branch = network.Find(dest\_id);

if (branch == -1) {

std::cout << "Error: incorrect node id!\n";

} else {

if (subcomand == "start") {

send\_message(branches[branch], std::to\_string(dest\_id) + "exec " + " start");

} else if (subcomand == "stop") {

send\_message(branches[branch], std::to\_string(dest\_id) + "exec " + " stop");

} else if (subcomand == "time") {

send\_message(branches[branch], std::to\_string(dest\_id) + "exec " + " time");

}

std::string reply = receive\_message(branches[branch]);

std::cout << reply << "\n";

}

} else if (comand == "ping") {

int dest\_id;

std::set<int> available\_nodes;

std::cin >> dest\_id;

if (network.Find(dest\_id) == -1) {

std::cout << "Error: Not found!\n";

} else {

int yes = 0;

//std::cout<<branches.size()<<std::endl;

for (int i = 0; i < branches.size(); ++i) {

int first\_node\_id = network.GetFirstId(i);

//std::cout<<"ping::first id : "<<first\_node\_id<<std::endl;

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) {

if (node == dest\_id) {

yes = 1;

}

}

}

std::cout << "OK: " << yes << "\n";

}

}

else if(comand == "heartbeat"){

int TIME;

std::cin>>TIME;

for (int i = 0; i < branches.size(); ++i) {

int first\_node\_id = network.GetFirstId(i);

send\_message(branches[i], std::to\_string(first\_node\_id) + " heartbeat " + std::to\_string(TIME));

std::cout << receive\_message(branches[i]) << "\n";

}

}

else if (comand == "exit") {

for (size\_t i = 0; i < branches.size(); ++i) {

int first\_node\_id = network.GetFirstId(i);

send\_message(branches[i], std::to\_string(first\_node\_id) + " remove");

std::string reply = receive\_message(branches[i]);

if (reply != "OK") {

std::cout << reply << "\n";

} else {

unbind(branches[i], first\_node\_id);

}

}

exit(0);

} else {

std::cout << "Incorrect comand >" << comand << "<!\n";

}

}

}

**count.cpp**

#include <iostream>

#include <unistd.h>

#include <sstream>

#include <set>

#include "zmq\_functions.h"

#include "topology.h"

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

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

if (argc != 2 && argc != 3) {

std::cout << "Wrong arguments Not enough parameters!\n";

exit(1);

}

int current\_id = std::atoi(argv[1]);//конвертация строки в число

int child\_id = -1;

if (argc == 3) {

child\_id = std::atoi(argv[2]);

}

std::string adr = argv[1];

zmq::context\_t context; //инкапсуляция функц возможности, связанные с инициализацией

zmq::socket\_t parent\_socket(context, ZMQ\_REP);//создаёт сокет в контексте ZMQ\_REP-для отправки запросов и получения ответов

//main\_socket.connect(argv[1]);

connect(parent\_socket, current\_id);

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

child\_socket.setsockopt(ZMQ\_SNDTIMEO, 5000);

parent\_socket.setsockopt(ZMQ\_SNDTIMEO, 5000);

std::string message;

auto start\_clock = std::chrono::high\_resolution\_clock::now();

auto stop\_clock = std::chrono::high\_resolution\_clock::now();

auto time\_clock = 0;

bool flag\_clock = false;

while (1) {

zmq::message\_t message\_main; //создаёт структуру для создания, уничтожения и управления сообщениями

message = receive\_message(parent\_socket); //получение сообщения из сокета родителя

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

std::istringstream request(message); //реализация операции ввода на основе сроки

int dest\_id;

request >> dest\_id;

std::string comand;

request >> comand;

if(comand == "heartbeat") {

std::string ans = std::to\_string(current\_id) + ":Ok; ";

if (child\_id != -1) {

int timeout;

request >> timeout;

int fl = 0;

for (int att=0; att < 4; att++) {

send\_message(child\_socket, message);

std::string repl = receive\_message(child\_socket);

if (repl != "Error ......") {

ans += repl;

fl = 1;

break;

}

sleep(timeout/1000);

}

if (fl == 0) {ans += "Node " + std::to\_string(child\_id) + " is not avail";}

}

send\_message(parent\_socket, ans);

}

else if (dest\_id == current\_id) {

if (comand == "pid") {

send\_message(parent\_socket, "OK: " + std::to\_string(getpid()));

} else if (comand == "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!\n");

exit(1);

}

if (pid == 0) {

execl("./count", "./count", 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");

child\_id = new\_child\_id;

send\_message(parent\_socket, receive\_message(child\_socket));

} else if (comand == "remove") {

if (child\_id != -1) {

send\_message(child\_socket, std::to\_string(child\_id) + " remove");

std::string msg = receive\_message(child\_socket);

if (msg == "OK") {

send\_message(parent\_socket, "OK");

}

unbind(child\_socket, child\_id);

disconnect(parent\_socket, current\_id);

break;

}

send\_message(parent\_socket, "OK");

disconnect(parent\_socket, current\_id);

break;

}

else if (comand == "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(current\_id) + reply);

} else if (comand == "exec") {

std::string subcomand;

request >> subcomand;

std::string msg = "OK: " + std::to\_string(current\_id);

if (subcomand == "start") {

start\_clock = std::chrono::high\_resolution\_clock::now();

flag\_clock = true;

} else if (subcomand == "stop") {

if (flag\_clock) {

stop\_clock = std::chrono::high\_resolution\_clock::now();

time\_clock += std::chrono::duration\_cast<std::chrono::milliseconds>(stop\_clock - start\_clock).count();

flag\_clock = false;

}

} else if (subcomand == "time") {

if (flag\_clock == true) {

stop\_clock = std::chrono::high\_resolution\_clock::now();

time\_clock += std::chrono::duration\_cast<std::chrono::milliseconds>(stop\_clock - start\_clock).count();

start\_clock = stop\_clock;

}

//stop\_clock = std::chrono::high\_resolution\_clock::now();

//time\_clock += std::chrono::duration\_cast<std::chrono::milliseconds>(stop\_clock - start\_clock).count();

msg += ": " + std::to\_string(time\_clock);

}

send\_message(parent\_socket, msg);

}

} else if (child\_id != -1) {

send\_message(child\_socket, message);

send\_message(parent\_socket, receive\_message(child\_socket));

if (child\_id == dest\_id && comand == "remove") {

child\_id = -1;

}

} else {

send\_message(parent\_socket, "Error: node is unavailable!\n");

}

}

}

**topology.h**

#include <list>

#include <stdexcept>

// Тип топологии: 1

class Topology {

private:

std::list<std::list<int>> container;

public:

// Добавление нового узла

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;

}

}

}

}

// Поиск узла с заданным id в списке списков

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;

}

// Удаление узла с указанным id

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); // Изменяет переданный итератор

for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {

if (\*it2 == id) {

it1->erase(it2, it1->end());

if (it1->empty()) {

container.erase(it1);

}

return;

}

}

}

// Получение первого id узла в контейнере

int GetFirstId(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>

#include <string>

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());//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) {

zmq::message\_t message;

int chars\_read;

try {

chars\_read = (int)socket.recv(&message); //получение сообщений

}

catch (...) {

chars\_read = 0;

}

if (chars\_read == 0) {

return "Error ......";

}

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);

}

// Отключение клиента

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);

}

// Связка узла с сокетом

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);

}

// Освобождение узла от сокета

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);

}

**Выводы**

Данная лабораторная работа была очень и очень непростой. Во время ее выполнения я полностью осозналa концепцию очередей сообщений на основе zero message queue. На мой взгляд, это достойное завершение курса Операционных систем.