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: Румынина Екатерина Александровна

Группа: М8О-201Б-21

Вариант: 8

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Демонстрация работы программы
6. Выводы

**Репозиторий**

<https://github.com/KetRum0/mai_os_labs>

**Постановка задачи**

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.

Вариант 8:

* Топология 1 - все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1
* Тип команд 3 - локальный таймер. Формат команды сохранения значения: exec id subcommand subcommand – одна из трех команд: start, stop, time; start – запустить таймер; stop – остановить таймер; time – показать время локального таймера в миллисекундах
* Тип проверки доступности 2 - Формат команды: ping id Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

**Общие сведения о программе**

client.cpp – программа управляющего узла

server.cpp - программа вычислительного узла

topology.hpp – объявление функций для реализации топологии (список)

topology.cpp – программа для реализации топологии (список)

zmqf.hpp – объявление функций для работы с zmq

zmqf.cpp – программа для реализации функций для работы с zmq

**Исходный код**

**client.cpp**

#include <unistd.h>

#include <iostream>

#include <vector>

#include <zmq.hpp>

#include <sstream>

#include "topology.h"

#include "zmqf.h"

int main() {

Topology network;

std::vector<zmq::socket\_t> branches;

zmq::context\_t context;

std::string command;

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

std::string message;

std::cout << "> ";

while (std::cin >> command) {

if (command == "create") {

int node\_id, parent\_id;

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

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

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

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

pid\_t pid = fork();

if (pid < 0) {

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

exit(EXIT\_FAILURE);

}

if (pid == 0) {

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

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

exit(EXIT\_FAILURE);

}

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 << "\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 (command == "remove") {

int id;

std::cin >> id;

int branch = network.Find(id);

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 << std::endl;

network.Erase(id);

if (is\_first) {

unbind(branches[branch], id);

branches.erase(std::next(branches.begin(), branch));

}

}

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

size\_t count = 0;

int destId;

std::cin >> destId;

int branch = network.Find(destId);

if (branch == -1) {

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

} else {

std::string subcommand;

std::cin >> subcommand;

send\_message(branches[branch], std::to\_string(destId) + "exec " + subcommand);

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

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

}

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

int destId;

std::cin >> destId;

int branch = network.Find(destId);

if (branch == -1) {

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

} else {

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

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

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

}

} else if (command == "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 command: " << command << "<!\n";

}

std::cout << "> ";

}

}

**server.cpp**

#include <unistd.h>

#include <iostream>

#include <string>

#include <vector>

#include <chrono>

#include "zmqf.h"

using namespace std::chrono;

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

{

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

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

exit(1);

}

int current\_id = std::atoi(argv[1]);

int child\_id = -1;

if (argc == 3) {

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

}

zmq::context\_t context;

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

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;

int flag = 0;

uint64\_t start;

uint64\_t stop;

while (1) {

zmq::message\_t message\_main;

message = receive\_message(parent\_socket);

//std::cout << message;

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

//std::cout << recieved\_message;

std::istringstream request(message);

int dest\_id;

request >> dest\_id;

std::string command;

request >> command;

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

if (command == "ping") {

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

send\_message(parent\_socket, ans);

} else if (command == "pid") {

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

} else if (command == "create") {

int new\_child\_id;

request >> new\_child\_id;

//std::cout << 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("server", "server", 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 (command == "remove") {

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

disconnect(parent\_socket, current\_id);

break;

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

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

std::string subcommand;

request >> subcommand;

if (subcommand == "start"){

if(flag == 1){

msg += "timer is already started";

}

else{

flag = 1;

msg += "started";

start = duration\_cast<milliseconds>(system\_clock::now().time\_since\_epoch()).count();

}

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

if(flag == 0) {

msg += "timer is not started";

}

else {

flag = 0;

msg += "stopped";

}

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

if(flag == 0) {

msg += "0";

}

else {

stop = duration\_cast<milliseconds>(system\_clock::now().time\_since\_epoch()).count();

msg += std::to\_string(stop-start);

}

} else {

msg += "Error: incorrect subcommand!\n";

}

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 && command == "remove") {

child\_id = -1;

}

} else {

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

}

}

}

**topology.hpp**

#ifndef LAB6\_TOPOLOGY\_H

#define LAB6\_TOPOLOGY\_H

#include <list>

#include <stdexcept>

class Topology {

private:

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

public:

void Insert(int id, int parent\_id);

int Find(int id);

void Erase(int id);

int GetFirstId(int list\_id);

};

#endif // LAB6\_TOPOLOGY\_H

**topology.cpp**

#include <topology.h>

void Topology::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("Insert 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 Topology::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 Topology::Erase(int id) {

int list\_id = Find(id);

if (list\_id == -1) {

throw std::runtime\_error("Erase 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 Topology::GetFirstId(int list\_id) {

auto it1 = container.begin();

std::advance(it1, list\_id);

if (it1->begin() == it1->end()) {

return -1;

}

return \*(it1->begin());

}

**zmqf.hpp**

#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);

std::string receive\_message(zmq::socket\_t& socket);

void connect(zmq::socket\_t& socket, int id);

void disconnect(zmq::socket\_t& socket, int id);

void bind(zmq::socket\_t& socket, int id);

void unbind(zmq::socket\_t& socket, int id);

#endif

**zmqf.cpp**

#include "zmqf.h"

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;

int chars\_read;

try {

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

}

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);

}

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);

}

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);

}

**Демонстрация работы программы**

ket@ket-laptop:~/Desktop/mai\_os\_labs/lab678$ ./client

> create 1 -1

OK: 13687

> create 2 1

OK: 13690

> create 3 2

OK: 13693

> create 4 3

OK: 13696

> ping 1

1: Ok

> ping 4

4: Ok

> exec 1 start

OK: 1 started

> exec 2 start

OK: 2 started

> exec 1 time

OK: 1 12419

> exec 2 time

OK: 2 10666

> exec 2 stop

OK: 2 stopped

> exec 2 time

OK: 2 0

> remove 3

OK

> ping 3

Error: incorrect node id!

> ping 4

Error: incorrect node id!

> ping 2

2: Ok

> exit

**Выводы**

В результате выполнения данной лабораторной работы я приобрела практические навыки в управлении серверами сообщений, применении отложенных вычислений, интеграции программных систем друг с другом.