Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Операционные системы»

Лабораторная работа № 6-8

Тема: Управлении серверами сообщений, применение отложенных вычислений, интеграция программных систем друг с другом

Студент: Туманов Георгий
Группа: 80-201
Преподаватель: Соколов А.А.
Оценка:
Подпись:

Москва, 2019

1. 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.
Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы.

Вариант 29:
Топология: звезда
Набор команд: локальный словарь
Команда проверки: ping id

2. Описание программы

Calculator имеет 2 сокета: сокет чтения и сокет записи. По сокету чтения calculator читает команды сервера, а по сокету записи отправляет ответ.
Controller содержит сокет чтения для других серверов, множество id и сокетов чтения и записи для calculator'ов, множество портов и сокетов записи для других серверов. Сервер принимает по сокет чтения команду, выполняет её и возвращает ответ по порту, передаваемому в команде.
Ввод с клавиатуры реализован как ещё один псевдо-сервер. Он читает ввод с клавиатуры, преобразовывает его в команду сервера и посылает ему эту команду, после чего ждёт ответ от него и выводит на экран.

3. Набор testcases

№	Описание	Ввод
1	Тест работы одного сервера	8080> create 1 8080> ping 1 8080> exec 1 test 8080> exec 1 test 10 8080> exec 1 test $ kill *node1* 8080> ping 1

		8080> exec 1 test 8080> remove 1 8080> quit
2	Тест работы двух серверов	8080> create 1 8080> exec 1 A 10 8080> ping 1 8080> ping 2 8081> create 2 8081> exec 2 B 20 8081> ping 1 8081> ping 2 8080> union 8081 8080> ping 1 8080> ping 2 8080> exec 1 A 8080> exec 2 B 8081> ping 1 8081> ping 2 8081> exec 1 A 8081> exec 2 B 8081> quit 8080> ping 1 8080> ping 2 8080> exec 1 A 8080> exec 2 B 8080> quit

4. Результаты выполнения тестов.

test 1:

$ ./controller 8080
8080> create 1
OK: 2494
8080> ping 1
OK: 1
8080> exec 1 test
OK: 'test' not found
8080> exec 1 test 10
OK.
8080> exec 1 test
OK: 10
$ kill 2494
8080> ping 1

OK: 0
8080> exec 1 test
ERROR: node is unavailable
8080> remove 1
OK.
8080> quit

test 2:
$ ./controller 8080
8080> create 1
OK: 2576
8080> exec 1 A 10
OK.
8080> ping 1
OK: 1
8080> ping 2
ERROR: id not found

$ ./controller 8081
8081> create 2
OK: 2580
8081> exec 2 B 20
OK.
8081> ping 1
ERROR: id not found
8081> ping 2
OK: 1

8080> union 8081
OK.
8080> ping 1
OK: 1
8080> ping 2
OK: 1
8080> exec 1 A
OK: 10
8080> exec 2 B
OK: 20

8081> ping 1
OK: 1
8081> ping 2

OK: 1
8081> exec 1 A
OK: 10
8081> exec 2 B
OK: 20
8081> quit

8080> ping 1
OK: 1
8080> ping 2
ERROR: id not found
8080> exec 1 A
OK: 10
8080> exec 2 B
ERROR: id not found
8080> quit

5. Листинг программы

Makefile:

```
all: calculator controller

calculator: calculator.cpp
   g++ calculator.cpp -o calculator -lzmq

controller: controller.cpp ctrl.h
   g++ controller.cpp -o controller -lzmq -lpthread
```

calculator.cpp:

```
#include <zmq.h>
#include <iostream>
#include <map>

//recieves:
// 2
// 0 var
// 1 var value
//sends:
// 0 - when ready
// 0 - ping
// 0 - success writing
// -1 - error reading
// 1 value - success reading

int main(int argc, char *argv[])
{
        if (argc != 3)
        {
```

```
std::cout << "Usage: " << argv[0] << " portRead portSend" << std::endl;
return 0;
}
std::map<std::string,int> base; //base of vars and values

//connect to controller
void *context = zmq_ctx_new();
void *socketRead = zmq_socket(context, ZMQ_PULL); //for reading from controller
void *socketSend = zmq_socket(context, ZMQ_PUSH); //for sending to controller

//connect sockets
std::string addrR = "tcp://localhost:";
std::string addrS = "tcp://localhost:";
addrR += argv[1];
addrS += argv[2];
zmq_connect(socketRead, addrR.c_str());
zmq_connect(socketSend, addrS.c_str());

//setup sockets
int zero = 0;
zmq_setsockopt(socketSend, ZMQ_LINGER, (void*)&zero, sizeof(zero));
zmq_setsockopt(socketRead, ZMQ_LINGER, (void*)&zero, sizeof(zero));

//when ready, send 0
char ZERO = 0;
zmq_send(socketSend, (void*)&ZERO, sizeof(ZERO), 0);

while (true)
{
char type; //0 - read, 1 - write, 2 - ping
zmq_recv(socketRead, (void*)&type, sizeof(type), 0);

if (type == 2) //ping
{
//send zero as OK
char snd = 0;
zmq_send(socketSend, (void*)&snd, sizeof(snd), 0);
continue;
}

std::string name = ""; //name of variable (sent if 0 or 1)
char letter;

while (true)
{
zmq_recv(socketRead, (void*)&letter, sizeof(letter), 0);
if (letter == '\0') break;
name += letter;
}

if (type == 1) //write
```

```
		{
		int val; //read value
		zmq_recv(socketRead, (void*)&val, sizeof(val), 0);
		base[name] = val; //set value
		//send zero as OK
		char snd = 0;
		zmq_send(socketSend, (void*)&snd, sizeof(snd), 0);
		}
		else //read
		{
		//try to find in base
		auto iter = base.find(name);
		if (iter == base.end()) //fail
		{
		//send -1 as error
		char snd = -1;
		zmq_send(socketSend, (void*)&snd, sizeof(snd), 0);
		}
		else //found
		{
		//send zero as KEEP READ
		char snd = 1;
		zmq_send(socketSend, (void*)&snd, sizeof(snd), 0);
		//send answer
		int snd2 = iter->second;
		zmq_send(socketSend, (void*)&snd2, sizeof(snd2), 0);
		}
		}
		}

		zmq_close(socketRead);
		zmq_close(socketSend);
		zmq_ctx_destroy(context);
		return 0;
}
```

ctrl.h:

```
#pragma once

#include <zmq.h>
#include <unistd.h>
#include <signal.h>

#include <algorithm>
#include <iostream>
#include <list>
#include <map>

class Controller
{
public:
		const static int TIMEOUT = 1000, PINGTIME = 10; //maximum waiting time
```

```
Controller(char *port)
{
//init my reading socket
m_context = zmq_ctx_new();
m_socket = zmq_socket(m_context, ZMQ_PULL);

int zero = 0;
zmq_setsockopt(m_socket, ZMQ_RCVTIMEO, (void*)&TIMEOUT, sizeof(TIMEOUT));
zmq_setsockopt(m_socket, ZMQ_LINGER, (void*)&zero, sizeof(zero));

std::string addr = "tcp://*:";
addr += port;
//try to bind
int err = zmq_bind(m_socket, addr.c_str()); //bind reading socket
if (err) throw std::logic_error("Port is not free: " + addr + " err: " + std::to_string(zmq_errno())); //fail
//set port
m_port = atoi(port);
//thread vars
m_threadWork = true;
m_threadDone = false;
}
~Controller()
{
m_threadWork = false;
while (!m_threadDone); //wait for thread to finish

//kill all calculators
for (auto &a : m_calculators)
{
kill(a.second.pid, SIGTERM);
kill(a.second.pid, SIGKILL);
zmq_close(a.second.socketRead);
zmq_close(a.second.socketSend);
}
//close all sockets
bool first = true;
for (auto &oc : m_otherControllers)
{
if (first) first = false; //skip 1st
else
{
char forget = 7; //disconnect
zmq_send(oc.socketSend, (void*)&forget, sizeof(forget), 0);
zmq_send(oc.socketSend, (void*)&m_port, sizeof(m_port), 0);
}
//close socket
zmq_close(oc.socketSend);
}
zmq_close(m_socket);
zmq_ctx_destroy(m_context);
```

```
        }

        int Add(int id)
        {
        if (m_calculators.find(id) != m_calculators.end())
        return -1; //this ID already exists

        //create sockets
        void *socketRead = zmq_socket(m_context, ZMQ_PULL);
        void *socketSend = zmq_socket(m_context, ZMQ_PUSH);
        //bind sockets
        zmq_bind(socketRead, "tcp://*:*");
        zmq_bind(socketSend, "tcp://*:*");
        //setup sockets
        int zero = 0;
        zmq_setsockopt(socketRead, ZMQ_RCVTIMEO, (void*)&PINGTIME, sizeof(PINGTIME));
        zmq_setsockopt(socketRead, ZMQ_LINGER, (void*)&zero, sizeof(zero));
        zmq_setsockopt(socketSend, ZMQ_SNDTIMEO, (void*)&PINGTIME, sizeof(PINGTIME));
        zmq_setsockopt(socketSend, ZMQ_LINGER, (void*)&zero, sizeof(zero));
        //get ports
        char portRead[1024], portSend[1024], *pR, *pS;
        size_t len = 1024;
        zmq_getsockopt(socketRead, ZMQ_LAST_ENDPOINT, (void*)&portRead, &len);
        zmq_getsockopt(socketSend, ZMQ_LAST_ENDPOINT, (void*)&portSend, &len);
        //get ports themselfs
        int dvoetochie, hp;
        for (dvoetochie = 0, hp = 1; dvoetochie < 1024; dvoetochie++) if (portRead[dvoetochie] == ':' && hp--
== 0) break;
        pR = portRead + dvoetochie + 1;
        for (dvoetochie = 0, hp = 1; dvoetochie < 1024; dvoetochie++) if (portSend[dvoetochie] == ':' && hp--
== 0) break;
        pS = portSend + dvoetochie + 1;

        //FORK!!!
        pid_t pid = fork();
        if (pid == 0) //child
        {
        execl("./calculator", "./calculator", pS, pR, NULL); //read from where we send and send to where whe
read from
        exit(-1);
        }
        else //parent
        {
        //wait answer
        char ans;
        int err = zmq_recv(socketRead, (void*)&ans, sizeof(ans), 0);
        if (err == -1 || ans != 0)
        {
        kill(pid, SIGTERM);
        kill(pid, SIGKILL);
        zmq_close(socketRead);
```

```
zmq_close(socketSend);
return -2; //cannot create node
}

//fill data
Calculator calc;
calc.pid = pid;
calc.socketRead = socketRead;
calc.socketSend = socketSend;
m_calculators[id] = calc;

return pid;
}
}
char Remove(int id)
{
auto iter = m_calculators.find(id); //check ID exists
if (iter == m_calculators.end()) return -1; //this ID doesn't exists

//kill him
kill(iter->second.pid, SIGTERM);
kill(iter->second.pid, SIGKILL);
zmq_close(iter->second.socketRead);
zmq_close(iter->second.socketSend);
//remove from map
m_calculators.erase(id);
return 0;
}
char Union(int port) //join with OC
{
//check port
if (port == m_port) return -1; //cannot union with myself
for (auto &oc : m_otherControllers)
if (oc.port == port) return -2;//cannot union twice

void *socketSend = zmq_socket(m_context, ZMQ_PUSH);
std::string addr = "tcp://localhost:" + std::to_string(port); //addres to OC
zmq_connect(socketSend, addr.c_str()); //connect to OC
//setup socket
int zero = 0;
zmq_setsockopt(socketSend, ZMQ_SNDTIMEO, (void*)&TIMEOUT, sizeof(TIMEOUT));
zmq_setsockopt(socketSend, ZMQ_LINGER, (void*)&zero, sizeof(zero));

//send OC data about connection
int err;
char three = 3, ans;
zmq_send(socketSend, (void*)&three, sizeof(three), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&m_port, sizeof(m_port), 0);//port
//wait him answering 0
err = zmq_recv(m_socket, (void*)&ans, sizeof(ans), 0);
if (err == -1 || ans != 17)
```

```
{
zmq_close(socketSend);
return -3;//Port is unjoinable
}
//Remove all duplicating calculators
for (auto &calc : m_calculators)
{
int id = calc.first;
char rem = 5;
zmq_send(socketSend, (void*)&rem, sizeof(rem), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&m_port, sizeof(m_port), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&id, sizeof(id), 0);
//wait answer
char ans;
err = zmq_recv(m_socket, (void*)&ans, sizeof(ans), 0); //we don't care, had he this node or not
}
//push it to the list
OtherController oc;
oc.port = port;
oc.socketSend = socketSend;
m_otherControllers.push_back(oc);
return 0;
}

char Exec(int id, int &ans, std::string var, std::list<int> visited) //read
{
auto iter = m_calculators.find(id); //check ID exists
if (iter == m_calculators.end())
{
visited.push_back(m_port);
//check OC
bool skip1st = true; //skip first OC
for (auto &oc : m_otherControllers)
{
if (skip1st)
{
        skip1st = false;
        continue;
}
if (std::find(visited.begin(), visited.end(), oc.port) != visited.end()) //already visited
        continue;
//send question
int sz = visited.size();
char q = 0;
zmq_send(oc.socketSend, (void*)&q, sizeof(q), ZMQ_SNDMORE);
zmq_send(oc.socketSend, (void*)&m_port, sizeof(m_port), ZMQ_SNDMORE);
zmq_send(oc.socketSend, (void*)&id, sizeof(id), ZMQ_SNDMORE);

for (int i = 0; i < var.size() + 1; i++) //send var name
{
        char c = var.c_str()[i];
```

```
            zmq_send(oc.socketSend, (void*)&c, sizeof(c), ZMQ_SNDMORE);
}

zmq_send(oc.socketSend, (void*)&sz, sizeof(sz), 0);
for (auto &port : visited)
            zmq_send(oc.socketSend, (void*)&port, sizeof(port), 0);

//wait answer
char status;
int err = zmq_recv(m_socket, (void*)&status, sizeof(status), 0);
if (err == -1 || status == -1) continue; //he has not this ID
if (status < 0) return status; //variable not found
zmq_recv(m_socket, (void*)&ans, sizeof(ans), 0); //get value
return 0;
}
return -1; //this ID doesn't exists
}

int err;
//send message
char type = 0; //read mode
//send type
err = zmq_send(iter->second.socketSend, (void*)&type, sizeof(type), 0);
if (err == -1) return -2; //node is unavailable
//send string
for (int i = 0; i < var.size() + 1; i++)
{
char c = var.c_str()[i];
err = zmq_send(iter->second.socketSend, (void*)&c, sizeof(c), 0);
if (err == -1) return -2; //node is unavailable
}

//recieve message
char good;
err = zmq_recv(iter->second.socketRead, (void*)&good, sizeof(good), 0);
if (err == -1) return -2; //node is unavailable
if (good != 1) return -3; //variable doesn't exists
//read answer
err = zmq_recv(iter->second.socketRead, (void*)&ans, sizeof(ans), 0);
if (err == -1) return -2; //node is unavailable
return 0;
}
char Exec(int id, std::string var, int value, std::list<int> visited) //write
{
auto iter = m_calculators.find(id); //check ID exists
if (iter == m_calculators.end())
{
visited.push_back(m_port);
//check OC
bool skip1st = true; //skip first OC
for (auto &oc : m_otherControllers)
```

```
{
if (skip1st)
{
	skip1st = false;
	continue;
}
if (std::find(visited.begin(), visited.end(), oc.port) != visited.end()) //already visited
	continue;
//send question
int sz = visited.size();
char q = 1;
zmq_send(oc.socketSend, (void*)&q, sizeof(q), ZMQ_SNDMORE);
zmq_send(oc.socketSend, (void*)&m_port, sizeof(m_port), ZMQ_SNDMORE);
zmq_send(oc.socketSend, (void*)&id, sizeof(id), ZMQ_SNDMORE);

for (int i = 0; i < var.size() + 1; i++) //send var name
{
	char c = var.c_str()[i];
	if (zmq_send(oc.socketSend, (void*)&c, sizeof(c), ZMQ_SNDMORE) == -1) break;
}
//send value
if (zmq_send(oc.socketSend, (void*)&value, sizeof(value), ZMQ_SNDMORE) == -1) continue;
//send visited list
if (zmq_send(oc.socketSend, (void*)&sz, sizeof(sz), 0) == -1) continue;
for (auto &port : visited)
	if (zmq_send(oc.socketSend, (void*)&port, sizeof(port), 0) == -1) break;

//wait answer
char status;
int err = zmq_recv(m_socket, (void*)&status, sizeof(status), 0);
if (err == -1 || status == -1) continue; //he has not this ID
return status; //return status
}
return -1; //this ID doesn't exists
}

int err;
//send message
char type = 1; //write mode
//send type
err = zmq_send(iter->second.socketSend, (void*)&type, sizeof(type), 0);
if (err == -1) return -2; //node is unavailable
//send string
for (int i = 0; i < var.size() + 1; i++)
{
char c = var.c_str()[i];
err = zmq_send(iter->second.socketSend, (void*)&c, sizeof(c), 0);
if (err == -1) return -2; //node is unavailable
}
//send value
err = zmq_send(iter->second.socketSend, (void*)&value, sizeof(value), 0);
```

```
if (err == -1) return -2; //node is unavailable

//recieve message
char good;
err = zmq_recv(iter->second.socketRead, (void*)&good, sizeof(good), 0);
if (err == -1) return -2; //node is unavailable
if (good != 0) return -2; //node is unavailable
return 0;
}
char Ping(int id, std::list<int> visited)
{
auto iter = m_calculators.find(id); //check ID exists
if (iter == m_calculators.end())
{
visited.push_back(m_port);
//check OC
bool skip1st = true; //skip first OC
for (auto &oc : m_otherControllers)
{
if (skip1st)
{
        skip1st = false;
        continue;
}
if (std::find(visited.begin(), visited.end(), oc.port) != visited.end()) //already visited
        continue;
//send question
int sz = visited.size();
char q = 2;
zmq_send(oc.socketSend, (void*)&q, sizeof(q), ZMQ_SNDMORE);
zmq_send(oc.socketSend, (void*)&m_port, sizeof(m_port), ZMQ_SNDMORE);
zmq_send(oc.socketSend, (void*)&id, sizeof(id), ZMQ_SNDMORE);
zmq_send(oc.socketSend, (void*)&sz, sizeof(sz), 0);
for (auto &port : visited)
        zmq_send(oc.socketSend, (void*)&port, sizeof(port), 0);

//wait answer
char ans;
int err = zmq_recv(m_socket, (void*)&ans, sizeof(ans), 0);
if (err == -1 || ans == -1) continue; //he has not this ID
return ans;
}
return -1; //this ID doesn't exists
}

//send message
char type = 2; //ping mode
//send type
zmq_send(iter->second.socketSend, (void*)&type, sizeof(type), 0);

//recieve message
```

```
char good;
int err = zmq_recv(iter->second.socketRead, (void*)&good, sizeof(good), 0);

if (err == -1 || good != 0) return 0;
return 1;
}

void ReadInputSocket()
{
int err;
char type;

while (m_threadWork)
{
err = zmq_recv(m_socket, (void*)&type, sizeof(type), 0);
if (err == -1) continue; //nothing to read

switch(type)
{
case 0: //read
{
        //get id for reading
        int asked_port, id, sz, temp_port;
        zmq_recv(m_socket, (void*)&asked_port, sizeof(asked_port), 0);
        zmq_recv(m_socket, (void*)&id, sizeof(id), 0);

        std::string name = "";
        char letter;
        while (1)
        {
        zmq_recv(m_socket, (void*)&letter, sizeof(letter), 0);
        if (letter == '\0') break;
        name += letter;
        }

        std::list<int> visited; //list of visited ports
        zmq_recv(m_socket, (void*)&sz, sizeof(sz), 0);//size of list

        while (sz-- > 0) //read elements
        {
        zmq_recv(m_socket, (void*)&temp_port, sizeof(temp_port), 0);
        visited.push_back(temp_port);//push to the list
        }

        //execute calculation
        int answer;
        char status = Exec(id, answer, name, visited);

        //send answer
        for (auto &oc : m_otherControllers)
        if (oc.port == asked_port) //get OC, who asked us
```

```
            {
            zmq_send(oc.socketSend, (void*)&status, sizeof(status), 0);
            if (status == 0) zmq_send(oc.socketSend, (void*)&answer, sizeof(answer), 0);
            break;
            }

            break;
}
case 1: //write
{
            //get id for writing
            int asked_port, id, sz, temp_port;
            zmq_recv(m_socket, (void*)&asked_port, sizeof(asked_port), 0);
            zmq_recv(m_socket, (void*)&id, sizeof(id), 0);

            std::string name = "";
            char letter;
            while (1)
            {
            zmq_recv(m_socket, (void*)&letter, sizeof(letter), 0);
            if (letter == '\0') break;
            name += letter;
            }
            int val;
            zmq_recv(m_socket, (void*)&val, sizeof(val), 0);

            std::list<int> visited; //list of visited ports
            zmq_recv(m_socket, (void*)&sz, sizeof(sz), 0);//size of list

            while (sz-- > 0) //read elements
            {
            zmq_recv(m_socket, (void*)&temp_port, sizeof(temp_port), 0);
            visited.push_back(temp_port);//push to the list
            }

            //execute calculation
            char status = Exec(id, name, val, visited);

            //send answer
            for (auto &oc : m_otherControllers)
            if (oc.port == asked_port) //get OC, who asked us
            {
            zmq_send(oc.socketSend, (void*)&status, sizeof(status), 0);
            break;
            }

            break;
}
case 2: //ping
{
            //get id for pinging
```

```
            int asked_port, id, sz, temp_port;
            zmq_recv(m_socket, (void*)&asked_port, sizeof(asked_port), 0);
            zmq_recv(m_socket, (void*)&id, sizeof(id), 0);

            std::list<int> visited; //list of visited ports
            zmq_recv(m_socket, (void*)&sz, sizeof(sz), 0);//size of list

            while (sz-- > 0) //read elements
            {
            zmq_recv(m_socket, (void*)&temp_port, sizeof(temp_port), 0);
            visited.push_back(temp_port);//push to the list
            }

            char ans = Ping(id, visited);
            //send answer
            for (auto &oc : m_otherControllers)
            if (oc.port == asked_port) //get OC, who asked us
            {
            zmq_send(oc.socketSend, (void*)&ans, sizeof(ans), 0);
            break;
            }

            break;
}
case 3: //connect
{
            //get port
            int port;
            err = zmq_recv(m_socket, (void*)&port, sizeof(port), 0);
            if (err == -1) break; //can't read port

            //connect to him
            void *socketSend = zmq_socket(m_context, ZMQ_PUSH);
            std::string addr = "tcp://localhost:" + std::to_string(port); //addres to OC
            zmq_connect(socketSend, addr.c_str()); //connect to OC
            //setup socket
            int zero = 0;
            zmq_setsockopt(socketSend, ZMQ_SNDTIMEO, (void*)&TIMEOUT, sizeof(TIMEOUT));
            zmq_setsockopt(socketSend, ZMQ_LINGER, (void*)&zero, sizeof(zero));

            //send 0 as ok
            char ok = 17;
            err = zmq_send(socketSend, (void*)&ok, sizeof(ok), 0);

            if (err == -1)
            {
            zmq_close(socketSend);
            break;
            }
            //push it to the list
            OtherController oc;
```

```
            oc.port = port;
            oc.socketSend = socketSend;
            m_otherControllers.push_back(oc);
            break;
}
case 4: //add
{
            //get port
            int asked_port;
            zmq_recv(m_socket, (void*)&asked_port, sizeof(asked_port), 0);
            //id
            int id;
            zmq_recv(m_socket, (void*)&id, sizeof(id), 0);
            //add
            int ans = Add(id);

            for (auto &oc : m_otherControllers)
            if (oc.port == asked_port) //get OC, who asked us
            {
            zmq_send(oc.socketSend, (void*)&ans, sizeof(ans), 0);
            break;
            }
            break;
}
case 5: //remove
{
            //get port
            int asked_port;
            zmq_recv(m_socket, (void*)&asked_port, sizeof(asked_port), 0);
            //id
            int id;
            zmq_recv(m_socket, (void*)&id, sizeof(id), 0);
            //add
            char ans = Remove(id);

            for (auto &oc : m_otherControllers)
            if (oc.port == asked_port) //get OC, who asked us
            {
            zmq_send(oc.socketSend, (void*)&ans, sizeof(ans), 0);
            break;
            }
            break;
}
case 6: //join
{
            //get port
            int asked_port;
            zmq_recv(m_socket, (void*)&asked_port, sizeof(asked_port), 0);
            //port to join with
            int port2;
            zmq_recv(m_socket, (void*)&port2, sizeof(port2), 0);
```

```
                //add
                char ans = Union(port2);

                for (auto &oc : m_otherControllers)
                if (oc.port == asked_port) //get OC, who asked us
                {
                zmq_send(oc.socketSend, (void*)&ans, sizeof(ans), 0);
                break;
                }
                break;
        }
        case 7: //unjoin
        {
                //get port
                int asked_port;
                zmq_recv(m_socket, (void*)&asked_port, sizeof(asked_port), 0);
                //port to join with

                for (auto iter = m_otherControllers.begin(); iter != m_otherControllers.end(); iter++)
                if (iter->port == asked_port)
                {
                zmq_close(iter->socketSend);
                m_otherControllers.erase(iter); //remove this one
                break;
                }
                break;
        }

        }
        }
        m_threadDone = true;
        }

        void ShowOC()
        {
        for (auto &oc : m_otherControllers)
        {
        std::cout << "OC: " <<  oc.port << std::endl;
        }
        }

private:
        //SOCKETS
        struct Calculator //calculator process
        {
        pid_t pid;
        void *socketRead; //socket for reading from calculator
        void *socketSend; //socket for sending to calculator
        };
        struct OtherController //pipeline with other controllers
        {
```

```
        int port;
        void *socketSend; //socket for sending messages to other controllers
        };
        void *m_context;
        void *m_socket; //my reading socket

        //FUNCTIONAL
        int m_port; //my port
        bool m_threadWork, m_threadDone; //variable for using thread

        std::map<int, Calculator> m_calculators; //map of ids and calculators
        std::list<OtherController> m_otherControllers; //list of other controllers
};
const int Controller::TIMEOUT, Controller::PINGTIME;
```

controller.cpp:

```
#include <sstream>
#include <vector>
#include <thread>

#include "ctrl.h"

//questions
// 0 port id var sz ... -- ask for reading value
// 1 port id var value sz ... -- ask for writing value
// 2 port id sz ... -- ask for ping
// 3 port -- connect
// 4 port id -- add calculating node
// 5 port id -- remove calculating node
// 6 port port2 -- join with port2
// 7 port2 -- forget port2
//-where sz ... means list of visited controllers

void Potok(Controller *ctrl)
{
        ctrl->ReadInputSocket();
}

int main(int argc, char *argv[])
{
        if (argc != 2) //required argument
        {
        std::cout << "Usage: " << argv[0] << " port" << std::endl;
        return 1;
        }
        //create controller
        Controller ctrl(argv[1]);

        void *context = zmq_ctx_new();

        //create sending socket
        void *socketSend = zmq_socket(context, ZMQ_PUSH);
```

```
	int zero = 0;
	std::string addr = "tcp://localhost:";
	addr += argv[1];
	//connect sending to Controller
	zmq_connect(socketSend, addr.c_str());
	//zmq_setsockopt(socketSend, ZMQ_SNDTIMEO, (void*)&Controller::TIMEOUT,
sizeof(Controller::TIMEOUT));
	zmq_setsockopt(socketSend, ZMQ_LINGER, (void*)&zero, sizeof(zero));

	//create reading socket
	void *socketRead = zmq_socket(context, ZMQ_PULL);
	zmq_bind(socketRead, "tcp://*:*");
	//zmq_setsockopt(socketRead, ZMQ_RCVTIMEO, (void*)&Controller::TIMEOUT,
sizeof(Controller::TIMEOUT));
	zmq_setsockopt(socketRead, ZMQ_LINGER, (void*)&zero, sizeof(zero));
	//get port
	char portRead[1024], *pR;
	size_t len = 1024;
	zmq_getsockopt(socketRead, ZMQ_LAST_ENDPOINT, (void*)&portRead, &len);

	int dvoetochie, hp;
	for (dvoetochie = 0, hp = 1; dvoetochie < 1024; dvoetochie++) if (portRead[dvoetochie] == ':' && hp--
== 0) break;
	pR = portRead + dvoetochie + 1;

	int myPort = atoi(pR);
	//std::cout << myPort << std::endl;

	//start controller
	std::thread potok(Potok, &ctrl);
	potok.detach();

	//connect with Controller
	char msg = 3;
	zmq_send(socketSend, (void*)&msg, sizeof(msg), ZMQ_SNDMORE);
	zmq_send(socketSend, (void*)&myPort, sizeof(myPort), 0);

	//read 0 as ok
	char ans;
	zmq_recv(socketRead, (void*)&ans, sizeof(ans), 0);
	//std::cout << ('0' + ans) << std::endl;

	//MAIN CYCLE
	bool loop = true;
	while (loop)
	{
	std::cout << "> ";

	std::string input;
	std::cin >> input;
```

```
//switch command
if (input == "quit") loop = false; //exit
else if (input == "create") //add node
{
int id, parent; std::cin >> id;
std::getline(std::cin, input);
//send message
char add = 4;
zmq_send(socketSend, (void*)&add, sizeof(add), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&myPort, sizeof(myPort), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&id, sizeof(id), 0);

//get answer
int ans;
zmq_recv(socketRead, (void*)&ans, sizeof(ans), 0);
if (ans == -1) std::cout << "ERROR: id already exists" << std::endl;
else if (ans == -2) std::cout << "ERROR: cannot create node" << std::endl;
else std::cout << "OK: " << ans << std::endl;
}
else if (input == "remove") //remove node
{
int id; std::cin >> id;

//send message
char rem = 5;
zmq_send(socketSend, (void*)&rem, sizeof(rem), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&myPort, sizeof(myPort), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&id, sizeof(id), 0);

//get answer
char ans;
zmq_recv(socketRead, (void*)&ans, sizeof(ans), 0);
if (ans == -1) std::cout << "ERROR: id not found" << std::endl;
else std::cout << "OK." << std::endl;
}
else if (input == "exec") //execute calculation on node
{
int id, val;
std::string varName;
std::cin >> id >> varName;
std::getline(std::cin, input);

if (!input.empty()) //write
{
val = atoi(input.c_str());
char wrt = 1;
int zero = 0;
zmq_send(socketSend, (void*)&wrt, sizeof(wrt), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&myPort, sizeof(myPort), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&id, sizeof(id), ZMQ_SNDMORE);
```

```
for (int i = 0; i < varName.size() + 1; i++) //send var name
{
	char c = varName.c_str()[i];
	zmq_send(socketSend, (void*)&c, sizeof(c), ZMQ_SNDMORE);
}
zmq_send(socketSend, (void*)&val, sizeof(val), ZMQ_SNDMORE); //send value
zmq_send(socketSend, (void*)&zero, sizeof(zero), 0); //empty list
//get answer
char status;
zmq_recv(socketRead, (void*)&status, sizeof(status), 0);

if (status == 0) std::cout << "OK." << std::endl; //good
else if (status == -1) std::cout << "ERROR: id not found" << std::endl;
else if (status == -2) std::cout << "ERROR: node is unavailable" << std::endl;
}
else //read
{
char rid = 0;
int zero = 0;
zmq_send(socketSend, (void*)&rid, sizeof(rid), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&myPort, sizeof(myPort), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&id, sizeof(id), ZMQ_SNDMORE);

for (int i = 0; i < varName.size() + 1; i++) //send var name
{
	char c = varName.c_str()[i];
	zmq_send(socketSend, (void*)&c, sizeof(c), ZMQ_SNDMORE);
}
zmq_send(socketSend, (void*)&zero, sizeof(zero), 0);
//get answer
char status; int answer;
zmq_recv(socketRead, (void*)&status, sizeof(status), 0);
if (status == 0) //good
{
	zmq_recv(socketRead, (void*)&answer, sizeof(answer), 0);
	std::cout << "OK: " << answer << std::endl;
}
else if (status == -1) std::cout << "ERROR: id not found" << std::endl;
else if (status == -2) std::cout << "ERROR: node is unavailable" << std::endl;
else if (status == -3) std::cout << "OK: '" << varName << "' not found" << std::endl;
}
}
else if (input == "ping") //ping node
{
int id, zero = 0; std::cin >> id;

//send message
char add = 2;
zmq_send(socketSend, (void*)&add, sizeof(add), ZMQ_SNDMORE);
zmq_send(socketSend, (void*)&myPort, sizeof(myPort), ZMQ_SNDMORE);
```

```
        zmq_send(socketSend, (void*)&id, sizeof(id), ZMQ_SNDMORE);
        zmq_send(socketSend, (void*)&zero, sizeof(zero), 0);

        //get answer
        char ans;
        zmq_recv(socketRead, (void*)&ans, sizeof(ans), 0);
        if (ans == -1) std::cout << "ERROR: id not found" << std::endl;
        else std::cout << "OK: " << char('0' + ans) << std::endl;
        }
        else if (input == "union")
        {
        int port2; std::cin >> port2;

        //send message
        char join = 6;
        zmq_send(socketSend, (void*)&join, sizeof(join), ZMQ_SNDMORE);
        zmq_send(socketSend, (void*)&myPort, sizeof(myPort), ZMQ_SNDMORE);
        zmq_send(socketSend, (void*)&port2, sizeof(port2), 0);

        //get answer
        char ans;
        zmq_recv(socketRead, (void*)&ans, sizeof(ans), 0);
        if (ans == -1) std::cout << "ERROR: cannot union self" << std::endl;
        else if (ans == -2) std::cout << "ERROR: cannot union twice" << std::endl;
        else if (ans == -3) std::cout << "ERROR: cannot union with this port" << std::endl;
        else std::cout << "OK."<< std::endl;
        }
        else if (input == "oc")
        {
        ctrl.ShowOC();
        }
        else
        {
        std::cout << "ERROR: unknown command" << std::endl;
        }
        }

        zmq_close(socketSend);
        zmq_close(socketRead);
        zmq_ctx_destroy(context);
        return 0;
}
```

6. Выводы:

Научился реализовывать сервера сообщений для связи серверов и вычислительных узлов и для связи между серверами с помощью библиотеки ZeroMQ.