Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №5-7 по курсу

«Операционные системы»

Группа: М8О-215Б-23

Студент: Кобзев К. А.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 15.07.25

Москва, 2025

Постановка задачи

Вариант 36.

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:

Создание нового вычислительного узла

Формат команды: create id [parent]

- id – целочисленный идентификатор нового вычислительного узла
- parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода:

- «Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла
- «Error: Already exists» - вычислительный узел с таким идентификатором уже существует
- «Error: Parent not found» - нет такого родительского узла с таким идентификатором
- «Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться
- «Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> create 10 5

Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Исполнение команды на вычислительном узле

Формат команды:

- exec id [params]
- id – целочисленный идентификатор вычислительного узла, на который отправляется команда

Формат вывода:

- «Ok:id: [result]», где result – результат выполненной команды
- «Error:id: Not found» - вычислительный узел с таким идентификатором не найден
- «Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом
- «Error:id: [Custom error]» - любая другая обрабатываемая ошибка

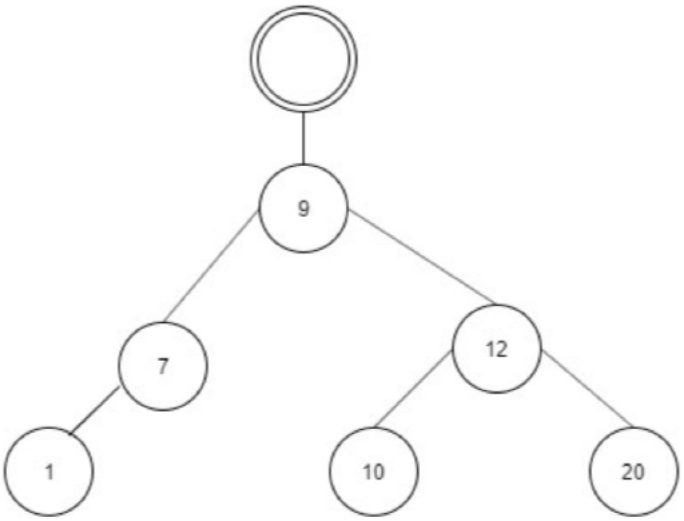
Пример:

Можно найти в описании конкретной команды, определенной вариантом задания.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

Вариант 16)

Типология 3



Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.

Тип команд 2

Набора команд 2 (локальный целочисленный словарь)

Формат команды сохранения значения: exec id name value

- id – целочисленный идентификатор вычислительного узла, на который отправляется команда
- name – ключ, по которому будет сохранено значение (строка формата [A-Za-z0-9]+)
- value – целочисленное значение

Формат команды загрузки значения: exec id name

Пример:

> exec 10 MyVar

Ok:10: 'MyVar' not found

> exec 10 MyVar 5

Ok:10

> exec 12 MyVar

Ok:12: 'MyVar' not found

> exec 10 MyVar

Ok:10: 5

> exec 10 MyVar 7

Ok:10

> exec 10 MyVar

Ok:10: 7

Примечания: можно использовать std:map.

Тип проверки доступности узлов 1.

Формат команды: pingall

Вывод всех недоступных узлов вывести разделенные через точку запятую.

Пример:

> pingall

Ok: -1 // Все узлы доступны

> pingall

Ok: 7;10;15 // узлы 7, 10, 15 — недоступны

Общий метод и алгоритм решения

Архитектура решения:

1. Управляющий узел (main.cpp):
   - Принимает команды от пользователя (create, exec, pingall).
   - Управляет дочерними процессами (вычислительными узлами), запуская их с помощью fork() и execl().
   - Хранит информацию о созданных узлах в std::map, что позволяет легко итерировать по ним для команды pingall.
   - Взаимодействует с вычислительными узлами, отправляя им команды и получая результаты через сокеты ZeroMQ. Для обнаружения "умерших" узлов используется таймаут на получение ответа.
2. Вычислительный узел (computing_node.cpp):
   - Запускается как отдельный процесс с уникальным ID, переданным в качестве аргумента.

- При запуске создает сокет REP и привязывается к уникальному порту, который зависит от его ID.
- В бесконечном цикле ожидает команды от управляющего узла.
- Реализует функционал локального целочисленного словаря с помощью std::map<std::string, int>.
- Корректно обрабатывает команды exec id name value (сохранение) и exec id name (получение).
- Отвечает на ping-запросы для проверки доступности.

3. Взаимодействие:
   - Для обмена сообщениями между узлами используется библиотека ZeroMQ, которая обеспечивает надежную и быструю доставку сообщений.
   - Применяется шаблон Request-Reply. Управляющий узел отправляет запрос (Request), а вычислительный узел отвечает (Reply).

Код программы

main.cpp

```
#include <iostream>

#include <string>

#include <vector>

#include <sstream>

#include <map>

#include <unistd.h>

#include <signal.h>

#include <zmq.hpp>


using namespace std;

using namespace zmq;


struct TreeNode

{

   int id;

   pid_t pid;

   TreeNode *left = 0;

   TreeNode *right = 0;
```

```
    TreeNode(int nodeId, pid_t processId) : id(nodeId), pid(processId) {}
};

context_t context(1);
socket_t main_socket(context, ZMQ_REP);
TreeNode *root = 0;
map<int, TreeNode *> nodes;

void delete_tree(TreeNode *node)
{
    if (node == 0)
        return;
    delete_tree(node->left);
    delete_tree(node->right);
    delete node;
}

vector<string> split(const string &s)
{
    stringstream ss(s);
    string item;
    vector<string> tokens;
    while (ss >> item)
        tokens.push_back(item);
    return tokens;
}

string send_receive(socket_t &socket, const string &message)
{
    socket.send(buffer(message), send_flags::none);
```

```
    message_t reply;
    socket.set(sockopt::rcvtimeo, 2000);
    if (socket.recv(reply, recv_flags::none))
        return reply.to_string();
    return "Error: Node is unavailable";
}

int main()
{
    int port = 4040;
    string adr = "tcp://127.0.0.1:" + to_string(port);
    main_socket.bind(adr);
    string line;
    while (cout << "> " && getline(cin, line))
    {
        if (line == "exit")
            break;
        auto args = split(line);
        if (args.empty())
            continue;
        string command = args[0];
        if (command == "create" && (args.size() == 2 || args.size() == 3))
        {
            int id = stoi(args[1]);
            if (nodes.count(id))
            {
                cout << "Error: Already exists" << endl;
                continue;
            }
```

```
        if (args.size() == 3 && root != 0)
        {
            int parentId = stoi(args[2]);
            if (!nodes.count(parentId))
            {
                cout << "Error: Parent not found" << endl;
                continue;
            }
            socket_t parent_socket(context, ZMQ_REQ);
            parent_socket.set(sockopt::rcvtimeo, 2000);
            parent_socket.connect("tcp://127.0.0.1:" + to_string(4040 + parentId));
            parent_socket.send(buffer("ping"), send_flags::none);
            message_t parent_reply;
            if (!parent_socket.recv(parent_reply, recv_flags::none))
            {
                cout << "Error: Parent is unavailable" << endl;
                continue;
            }
        }
        pid_t pid = fork();
        if (pid == -1)
        {
            perror("fork");
            cout << "Error: [Custom error] Failed to create process" << endl;
        }
        else if (pid == 0)
        {
            execl("./computing_node", "computing_node", to_string(id).c_str(), adr.c_str(), (char
*)NULL);
            perror("execl");
```

```
    exit(1);
}
else
{
    message_t request;
    if (!main_socket.recv(request, recv_flags::none)) {
        cerr << "Error" << endl;
    }
    main_socket.send(buffer("OK"), send_flags::none);
    TreeNode *newNode = new TreeNode(id, pid);
    nodes[id] = newNode;
    if (root == 0)
        root = newNode;
    else
    {
        TreeNode *current = root;
        TreeNode *parent = 0;
        while (current != 0)
        {
            parent = current;
            if (id < current->id)
                current = current->left;
            else
                current = current->right;
        }
        if (id < parent->id)
            parent->left = newNode;
        else
            parent->right = newNode;
    }
```

```
            cout << "Ok: " << pid << endl;
        }
    }
    else if (command == "exec" && args.size() >= 2)
    {
        int id = stoi(args[1]);
        string exec_params;
        if (args.size() >= 3)
        {
            for (size_t i = 2; i < args.size(); ++i)
            {
                exec_params += args[i] + " ";
            }
        }
        else
        {
            cout << "> ";
            getline(cin, exec_params);
        }
        // проверяем существование узла
        if (!nodes.count(id))
        {
            cout << "Error:" << id << ": Not found" << endl;
            continue;
        }
        socket_t socket(context, ZMQ_REQ);
        socket.connect("tcp://127.0.0.1:" + to_string(4040 + id));
        string result = send_receive(socket, exec_params);

        if (result == "Error: Node is unavailable")
```

```
    {
        cout << "Error:" << id << ": Node is unavailable" << endl;
    }
    else
    {
        cout << result << endl;
    }
}
else if (command == "pingall")
{
    string unavailable_nodes;
    for (auto const &[id, node] : nodes)
    {
        socket_t ping_socket(context, ZMQ_REQ);
        ping_socket.set(sockopt::rcvtimeo, 2000);
        ping_socket.connect("tcp://127.0.0.1:" + to_string(4040 + id));
        ping_socket.send(buffer("ping"), send_flags::none);
        message_t reply;
        if (!ping_socket.recv(reply, recv_flags::none))
        {
            unavailable_nodes += to_string(id) + ";";
        }
    }
    if (unavailable_nodes.empty())
        cout << "Ok: -1" << endl;
    else
    {
        if (!unavailable_nodes.empty())
            unavailable_nodes.pop_back();
        cout << "Ok: " << unavailable_nodes << endl;
```

```
            }
        }
        else
        {
            cout << "Error: Unknown command" << endl;
        }
    }
    for (auto const &[id, node] : nodes)
    {
        kill(node->pid, SIGTERM);
    }
    delete_tree(root);
    return 0;
}
```

computing_node.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <zmq.hpp>
#include <sstream>

using namespace std;
using namespace zmq;

vector<string> split(const string &s)
{
    stringstream ss(s);
    string item;
    vector<string> tokens;
```

```
    while (ss >> item) tokens.push_back(item);
    return tokens;
}

int main(int argc, char *argv[])
{
    if (argc != 3) {
        cerr << "Usage: computing_node <id> <control_node_address>" << endl;
        return 1;
    }
    int id = stoi(argv[1]);
    string control_adr = argv[2];
    context_t context(1);

    socket_t control_socket(context, ZMQ_REQ);
    control_socket.connect(control_adr);
    control_socket.send(buffer("Ready " + to_string(id)));
    message_t ack;
    if (!control_socket.recv(ack)) {
        cerr << "Error: Failed to receive acknowledgment from control node. Exiting." << endl;
        return 1;
    }

    socket_t worker_socket(context, ZMQ_REP);
    worker_socket.bind("tcp://127.0.0.1:" + to_string(4040 + id));

    map<string, int> dictionary;

    while (true)
    {
```

```
    message_t request;
    if (worker_socket.recv(request, recv_flags::none))
    {
        string msg_str = request.to_string();
        string reply_str;

        if (msg_str == "ping") {
            reply_str = "Ok:id: pong";
        } else {
            auto args = split(msg_str);
            if (args.size() == 2) {
                string name = args[0];
                int value = stoi(args[1]);
                dictionary[name] = value;
                reply_str = "Ok:" + to_string(id);
            } else if (args.size() == 1) {
                string name = args[0];
                if (dictionary.count(name)) {
                    reply_str = "Ok:" + to_string(id) + ": " + to_string(dictionary[name]);
                } else {
                    reply_str = "Ok:" + to_string(id) + ": '" + name + "' not found";
                }
            } else {
                reply_str = "Error:id:" + to_string(id) + ": [Custom error] Invalid command format";
            }
        }
        worker_socket.send(buffer(reply_str), send_flags::none);
    }
}
```

```
    return 0;
}
```

CMakeCache.txt

```
cmake_minimum_required(VERSION 3.10)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(PkgConfig REQUIRED)
pkg_check_modules(ZMQ REQUIRED libzmq cppzmq)

function(configure_target target_name source_file)
    add_executable(${target_name} ${source_file})
    target_include_directories(${target_name} PRIVATE ${ZMQ_INCLUDE_DIRS})
    target_link_libraries(${target_name} PRIVATE ${ZMQ_LIBRARIES})
    target_compile_options(${target_name} PRIVATE ${ZMQ_CFLAGS_OTHER})
    target_link_directories(${target_name} PRIVATE ${ZMQ_LIBRARY_DIRS})
endfunction()

configure_target(control_node main.cpp)
configure_target(computing_node computing_node.cpp)
```

Протокол работы программы

Тестирование:

Сборка:

- mkdir build
- cd buildВведите
- cmake ..

```
-- The CXX compiler identification is AppleClang 17.0.0.17000013
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found PkgConfig: /opt/homebrew/bin/pkg-config (found version "2.5.1")
-- Checking for modules 'libzmq;cppzmq'
```

-- Found libzmq, version 4.3.5
-- Found cppzmq, version 4.11.0
-- Configuring done (0.3s)
-- Generating done (0.0s)
-- Build files have been written to: path/lab5-7/src/build

- make
  [25%] Building CXX object CMakeFiles/control_node.dir/main.cpp.o
  [50%] Linking CXX executable control_node
  [50%] Built target control_node
  [75%] Building CXX object CMakeFiles/computing_node.dir/computing_node.cpp.o
  [100%] Linking CXX executable computing_node
  [100%] Built target computing_node

Запуск:

- ./control_node
- create 10
  Ok: 34128
- create 5
  Ok: 34129
- create 12
  Ok: 34130
- create 2
  Ok: 34131
- exec 10 myVar 123
  Ok:10
- exec 10 myVar
  Ok:10: 123
- exec 5 anotherVar
  Ok:5: 'anotherVar' not found
- exec 12 myVar 99
  Ok:12
- exec 10 myVar
  Ok:10: 123
- pingall
  Ok: -1
- kill -9 34129
- pingall
  Ok: 5

strace

386 execve("./control_node", ["./control_node"], 0xffffd017f518 /* 12 vars */) = 0

386 brk(NULL) = 0x18a2f000

386 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff8709b000

Настройка сокета для связи

386 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP) = 9

386 setsockopt(9, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0

Управляющий узел создает сокет. AF_INET указывает на использование семейства адресов IPv4.

SOCK_STREAM означает, что это будет TCP-сокет, обеспечивающий надежную потоковую передачу данных. ZeroMQ будет использовать этот сокет для своего REQ-REP паттерна.

- 386 bind(9, {sa_family=AF_INET, sin_port=htons(4040), sin_addr=inet_addr("127.0.0.1")}, 16) = 0

Процесс привязывает созданный сокет (файловый дескриптор 9) к конкретному адресу: 127.0.0.1 (localhost) и порту 4040.

Теперь он может принимать входящие подключения на этот адрес.

- 386 listen(9, 100) = 0

386 getsockname(9, {sa_family=AF_INET, sin_port=htons(4040), sin_addr=inet_addr("127.0.0.1")}, [128 => 16]) = 0

386 getsockname(9, {sa_family=AF_INET, sin_port=htons(4040), sin_addr=inet_addr("127.0.0.1")}, [128 => 16]) = 0

386 getpid() = 386

386 write(6, "\1\0\0\0\0\0\0\0", 8) = 8

- 388 <... epoll_pwait resumed>[{events=EPOLLIN, data={u32=413428128, u64=413428128}}], 256, -1, NULL, 8) = 1

386 getpid(<unfinished ...>

388 ppoll([{fd=6, events=POLLIN}], 1, {tv_sec=0, tv_nsec=0}, NULL, 0) = 0 (Timeout)

388 epoll_pwait(7, <unfinished ...>

Кманда для создания узла.

- 386 <... read resumed>"create 10 5\n", 1024) = 12

386 futex(0xffff86e637f8, FUTEX_WAKE_PRIVATE, 2147483647) = 0

Управляющий узел (PID 386) прочитал команду create 10 5 из стандартного ввода

- 386 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xffff870890f0) = 389

389 set_robust_list(0xffff87089100, 24 <unfinished ...>

386 getpid(<unfinished ...>

389 <... set_robust_list resumed>) = 0

386 <... getpid resumed>) = 386

386 ppoll([{fd=8, events=POLLIN}], 1, NULL, NULL, 0 <unfinished ...>

Системный вызов

- 389 execve("./computing_node", ["computing_node", "10", "tcp://127.0.0.1:4040"], 0xffffeb5f9f28 /* 12 vars */ <unfinished ...>

- 386 <... ppoll resumed>) = 1 ([{fd=8, revents=POLLIN}])

- 386 getpid() = 386

- 389 <... execve resumed>) = 0

386 read(8, <unfinished ...>

389 brk(NULL <unfinished ...>

389 <... mmap resumed>) = 0xffffba672000

386 ppoll([{fd=8, events=POLLIN}], 1, {tv_sec=0, tv_nsec=0}, NULL, 0 <unfinished ...>

389 faccessat(AT_FDCWD, "/etc/ld.so.preload", R_OK <unfinished ...>

огда выполняется pingall, управляющий узел в цикле пытается подключиться и отправить сообщение ping каждому известному ему вычислительному узлу.

- 386 <... ppoll resumed>) = 0 (Timeout)

- 389 <... faccessat resumed>) = -1 ENOENT (No such file or directory)

386 getpid(<unfinished ...>

<... ppoll resumed>) = 0 (Timeout)

391 <... mprotect resumed>) = 0

389 getpid(<unfinished ...>

- 391 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP <unfinished ...>

- 389 <... getpid resumed>) = 389

- 391 <... socket resumed>) = 9

389 ppoll([{fd=8, events=POLLIN}], 1, NULL, NULL, 0 <unfinished ...>

391 fcntl(9, F_GETFL) = 0x2 (flags O_RDWR)

391 fcntl(9, F_SETFL, O_RDWR|O_NONBLOCK) = 0

Новый узел должен сообщить, что он готов к работе.

- 391 connect(9, {sa_family=AF_INET, sin_port=htons(4040), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operation now in progress)

388 <... epoll_pwait resumed>[{events=EPOLLIN, data={u32=2147486576, u64=281472829229936}}], 256, -1, NULL, 8) = 1

391 epoll_ctl(7, EPOLL_CTL_ADD, 9, {events=0, data={u32=3019904048, u64=281473701647408}} <unfinished ...>

388 accept4(9, <unfinished ...>

391 <... epoll_ctl resumed>) = 0

Вычислительный узел (здесь его PID 391, в вашем логе он 389) пытается подключиться к управляющему узлу по адресу, который ему передали при запуске.

Это его сокет типа REQ.

- 388 <... accept4 resumed>{sa_family=AF_INET, sin_port=htons(34684), sin_addr=inet_addr("127.0.0.1")}, [128 => 16], SOCK_CLOEXEC) = 10

391 epoll_ctl(7, EPOLL_CTL_MOD, 9, {events=EPOLLOUT, data={u32=3019904048, u64=281473701647408}} <unfinished ...>

388 setsockopt(10, SOL_TCP, TCP_NODELAY, [1], 4 <unfinished ...>

391 <... write resumed>) = 8

389 <... ppoll resumed>) = 1 ([{fd=8, revents=POLLIN}])

Управляющий узел принял входящее соединение от вычислительного узла.

Теперь для общения с этим конкретным узлом создан новый файловый дескриптор.

- 391 sendto(9, "\1\0\0\10Ready 10", 12, 0, NULL, 0 <unfinished ...>

- 388 <... getpid resumed>) = 386

- 391 <... sendto resumed>) = 12

389 getpid(<unfinished ...>

391 epoll_pwait(7, <unfinished ...>

poll resumed>) = 0 (Timeout)

389 ppoll([{fd=8, events=POLLIN}], 1, NULL, NULL, 0 <unfinished ...>

388 epoll_pwait(7, [{events=EPOLLIN, data={u32=2147490256, u64=281472829233616}}], 256, -1, NULL, 8) = 1

- 388 recvfrom(10, "\1\0\0\10Ready 10", 8192, 0, NULL, NULL) = 12

388 getpid()   = 386

388 getpid()   = 386

388 write(8, "\1\0\0\0\0\0\0\0", 8) = 8

Управляющий узел получил сообщение.

Теперь он знает, что узел готов, и может отправить подтверждение.

- 388 sendto(10, "\1\0\0\3OK\0", 7, 0, NULL, 0 <unfinished ...>

- 391 <... epoll_pwait resumed>[{events=EPOLLIN, data={u32=3019904048, u64=281473701647408}}], 256, -1, NULL, 8) = 1

- 388 <... sendto resumed>)   = 7

391 recvfrom(9, <unfinished ...>

Выполнение команды exec

- 386 <... read resumed>"exec 10 MyVar\n", 1024) = 14

386 eventfd2(0, EFD_CLOEXEC)   = 11

386 fcntl(11, F_GETFL)   = 0x2 (flags O_RDWR)

386 fcntl(11, F_SETFL, O_RDWR|O_NONBLOCK) = 0

388 <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

386 getpid(<unfinished ...>

- 388 socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_TCP <unfinished ...>

- 386 <... getpid resumed>)   = 386

- 388 <... socket resumed>)   = 12

386 read(11, <unfinished ...>

388 <... fcntl resumed>)   = 0

386 ppoll([{fd=11, events=POLLIN}], 1, {tv_sec=0, tv_nsec=0}, NULL, 0 <unfinished ...>

- 388 connect(12, {sa_family=AF_INET, sin_port=htons(4050), sin_addr=inet_addr("127.0.0.1")}, 16 <unfinished ...>

- 386 <... ppoll resumed>)   = 0 (Timeout)

- 386 getpid(<unfinished ...>

391 <... epoll_pwait resumed>[{events=EPOLLIN, data={u32=3019903840, u64=281473701647200}}], 256, -1, NULL, 8) = 1

386 <... getpid resumed>) = 386

391 getpid(<unfinished ...>

- 388 sendto(12, "\1\0\0\6MyVar ", 10, 0, NULL, 0 <unfinished ...>

- 386 read(11, <unfinished ...>

- 391 <... getpid resumed>) = 389

- 389 ppoll([{fd=10, events=POLLIN}], 1, {tv_sec=0, tv_nsec=0}, NULL, 0 <unfinished ...>

- 386 <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

- 391 read(6, <unfinished ...>

- 388 <... sendto resumed>) = 10

386 getpid(<unfinished ...>

391 <... read resumed>"\1\0\0\0\0\0\0\0", 8) = 8

391 <... ppoll resumed>) = 0 (Timeout)

- 388 <... recvfrom resumed>"\1\0\0\30Ok:10: 'MyVar' not found", 8192, 0, NULL, NULL) = 28

391 epoll_pwait(7, <unfinished ...>

389 getpid(<unfinished ...>

386 getpid() = 386

386 write(6, "\1\0\0\0\0\0\0\0", 8) = 8

poll_pwait (или ppoll). Это сердце асинхронного I/O, которое использует ZeroMQ

388 <... epoll_pwait resumed>[{events=EPOLLIN, data={u32=413428128, u64=413428128}}], 256, -1, NULL, 8) = 1

- 386 write(1, "Ok:10: 'MyVar' not found\n", 25 <unfinished ...>

388 getpid(<unfinished ...>

386 <... write resumed>) = 25

```
386   write(1, "Error:12: Not found\n", 20) = 20
- 386   write(1, "> ", 2)                  = 2
386   read(0, "pingall\n", 1024)       = 8


389   getpid()                         = 389
389   ppoll([{fd=10, events=POLLIN}], 1, {tv_sec=0, tv_nsec=0}, NULL, 0) = 0 (Timeout)
389   getpid()                         = 389
389   ppoll([{fd=10, events=POLLIN}], 1, NULL, NULL, 0 <unfinished ...>

# Завершение работы (exit)
- 386   <... read resumed>"exit\n", 1024) = 5
- 386   kill(389, SIGTERM)                = 0
389   <... ppoll resumed>)             = ? ERESTARTNOHAND (To be restarted if no handler)
386   getpid( <unfinished ...>
```

Вывод

В ходе выполнения этой лабораторной работы была разработана и реализована распределённая система, в которой процессы взаимодействуют между собой асинхронно с помощью очередей сообщений. Система включает один управляющий узел и несколько вычислительных узлов, которые создаются динамически по мере необходимости.