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

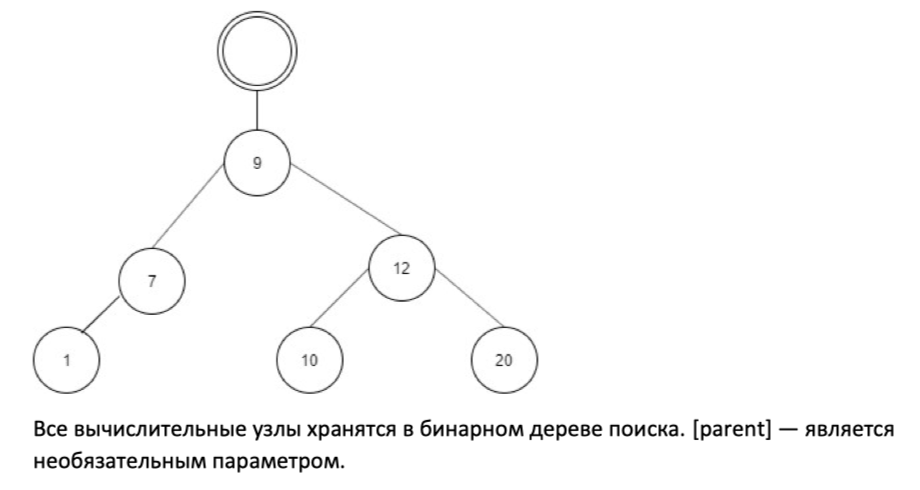
Пример:

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



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.

1. 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.

1. 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.