Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №5-7 по курсу

«Операционные системы»

Студент: Юнусов Руслан Асифович
Группа: М8О-209Б-23
Вариант: 24
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/Rissochek/OSLabs/tree/main/lab5-7

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

- Управлении серверами сообщений (№5)
- Применение отложенных вычислений (№6)
- Интеграция программных систем друг с другом (№7)

Задание

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])

Исполнение команды на вычислительном узле (Формат команды: exec id [params])

Проверка доступности узла (Формат команды: pingall)

Вариант №24: топология — общее дерево, команда — работа с локальным словарем, проверка доступности — pingall.

Общие сведения о программе

Связь между вычислительными узлами будем поддерживать с помощью ZMQ_SUB и ZMQ_PUB. При этом каждый узел одновременно будет иметь в себе по 1 сокету каждого типа. ZMQ_PUB рассылает сообщение всем своим subscriber-ам(с помощью команды zstr_send) поэтому в случаях команд exec и pingall передаются дополнительно id узла для которого предназначено сообщение, чтобы не было зацикливания команд между узлами (т.к каждый узел подписан на всех своих детей и родителей, а также публицирует для всех своих детей и родителей)

Общий метод и алгоритм решения

Используемые методы системные вызовы:

zsock_t* zsock_new_pub(const char* endpoint)	Создает новый ZMQ_PUB сокет
zpoller_t* zpoller_new(void* reader)	Создает новый poller, проверяющий наличие сообщений от ZMQ_PUB, на который узел подписан
void* zpoller_wait(zpoller_t* self, timeout)	Проверяет наличие сообщений в неблокирующем режиме
char* zstr_recv (void *source);	Принимает сообщение от другого сокета
int zpoller_add (zpoller_t *self, void *reader);	Добавляет к poller-у сокет для отслеживания
int zsock_connect (zsock_t *self, const char *format, ...)	Соединяет сокет с другим сокетом по адрессу
int zstr_send (void *dest, const char *string)	Отправляет строку от нашего сокета

Исходный код

Control_node.c

```
#include <czmq.h>

#include <unistd.h>

#include <string.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>
```

```c
typedef struct Node Node;

struct Node{

    int id;

    int parent_id;

    size_t childs_count;

    size_t childs_container_size;

    Node* childs;

};


typedef struct{

    Node head;

} Tree;


size_t check_command_length(char* input);

Node* find_node_by_id(Node* head_node, int target_id);

int* compare_arrays(int *array1, size_t size1, int *array2, size_t size2);

void add_child_to_node(Node* node, Node* child);


int main(){

    // Публикуем по 5555 порту

    zsock_t* publisher = zsock_new_pub("tcp://*:5555");

    zsock_t* subscriber = NULL;

    // vars init

    Node head = {-1, 0, 0, 0, NULL};

    Tree tree = {head};

    int created_nodes[100] = {0};

    size_t counter = 0;

    int create_node_id; // id узла для создания

    int parent_id_for_new_node; // id отца нового узла
```

```
    int flag = 0; // был ли первый ребенок;
    zpoller_t* poller = zpoller_new(subscriber);

    printf("Starting main function\n");
    printf("Publisher socket created\n");

    // Логика обработки команд: create, exec, pingall, sub.
    while (1){
        if (flag == 1) {
            // Проверяем наличие сообщений в неблокирующем режиме
            zsock_t* ready_sock = zpoller_wait(poller, 2000);
            if (ready_sock) {
                char* input_string = zstr_recv(subscriber);
                if (!input_string) {
                    perror("Failed to receive message");
                    exit(EXIT_FAILURE);
                }
                printf("Received message: %s\n", input_string);

                char to_print[100] = {0}; // Инициализируем массив нулями
                // Обработка входящих сообщений от вычислительных узлов.
                char nodes_command[100];
                size_t counter = 0;
                sscanf(input_string, "%s", nodes_command);
                printf("Command: %s\n", nodes_command);

                // если это ответ на exec
                if (strcmp(nodes_command, "for") == 0) {
                    // проходим пока не найдем 0, потом кайфуем
                    while (input_string[counter] != '0') {
```

```
                    counter++;
                }
                // Копируем все, что идет далее (включая '0')
                strcpy(to_print, &input_string[counter]);
                printf("Response to exec command: %s\n", to_print);
            // } else if (strcmp(nodes_command, "pingall") == 0) {
            //     int alive_list[100] = {0};
            //     int alive_node_id;
            //     size_t local_counter = 0;
            //     size_t local_counter2 = 0;
            //     sscanf(input_string, "pingall %d", &alive_node_id);
            //     alive_list[local_counter] = alive_node_id;
            //     int* not_alive = compare_arrays(alive_list,
local_counter, created_nodes, counter);
            //     printf("OK: ");
            //     while (not_alive[local_counter2] != 0) {
            //         printf("%d; ", not_alive[local_counter2++]);
            //     }
            //     printf("\n");
            }
            free(input_string);
        }
    }

    // считываем команду
    char command[100];
    scanf("%s ", command);
    printf("Received command: %s\n", command);

    // create (вроде бы фулл правильно написан (надеюсь))
```

```
        if (strcmp(command, "create") == 0){
            scanf("%d %d", &create_node_id, &parent_id_for_new_node); // считываем данные команды id и parent_id
            printf("create_node_id: %d, parent_id_for_new_node: %d\n", create_node_id, parent_id_for_new_node);

            // поиск родителя
            Node* parent_node = find_node_by_id(&tree.head, parent_id_for_new_node);
            if (parent_node == NULL){
                printf("Error: Parent not found\n");
            }
            // если родитель есть
            else{
                if (find_node_by_id(&tree.head, create_node_id) != NULL){
                    printf("Error: Already exists\n");
                }else{
                    // создание новой ноды в дереве
                    created_nodes[counter++] = create_node_id;
                    Node new_node = {create_node_id, parent_id_for_new_node, 0, 0, NULL};
                    add_child_to_node(parent_node, &new_node);
                    printf("New node created and added to parent\n");

                    // создание нового узла
                    pid_t child_proccess_id = fork();
                    if (child_proccess_id == -1){
                        perror("Failed to fork(unlucky moment)");
                        exit(EXIT_FAILURE);
                    }
                    // создаем дочерний процесс
```

```
                    else if (child_proccess_id == 0){
                        char create_node_id_str[20];
                        char parent_id_for_new_node_str[20];
                        snprintf(create_node_id_str,
sizeof(create_node_id_str), "%d", create_node_id);
                        snprintf(parent_id_for_new_node_str,
sizeof(parent_id_for_new_node_str), "%d", parent_id_for_new_node);
                        char* new_argv[] = {"./counting",
create_node_id_str, parent_id_for_new_node_str, NULL};
                        if(execv(new_argv[0], new_argv) == -1){
                            perror("Failed to execv new process (unlucky
moment)");
                            exit(EXIT_FAILURE);
                        }
                    }
                    // логика в главном процессе (по идее суб на дочерний
процесс, если главный это его родитель)
                    else {
                        char address[100];
                        size_t port_to_sub;

                        if (flag == 0 && parent_id_for_new_node == -1){
                            // формируем адрес для подписки и
подписываемся по нему (создаем сокет)
                            port_to_sub = 5555 + create_node_id;
                            snprintf(address, sizeof(address),
"tcp://localhost:%zu", port_to_sub);
                            subscriber = zsock_new_sub(address, "");
                            zpoller_add(poller, subscriber);
                            flag = 1;
                            printf("Subscriber socket created and
connected to %s\n", address);
                        }
```

```
                        else if (flag != 0 && parent_id_for_new_node ==
-1){

                            // формируем адрес для подписки и
подписываемся по нему (просто подписываемся с готового сокета)

                            port_to_sub = 5555 + create_node_id;

                            snprintf(address, sizeof(address),
"tcp://localhost:%zu", port_to_sub);

                            zsock_connect(subscriber, address);

                            printf("Subscriber connected to %s\n",
address);

                        }

                        else if (parent_id_for_new_node != -1){

                            // логика отправки команды для подписки
родительского процесса на дочерний (для обратной связи)

                            char message_to_sub[100];

                            snprintf(message_to_sub,
sizeof(message_to_sub), "sub for %d on %d", parent_id_for_new_node,
create_node_id);

                            zstr_send(publisher, message_to_sub);

                            printf("Sent sub command: %s\n",
message_to_sub);

                        }

                    }

                }

            }

        }

        else if(strcmp(command, "exec") == 0){

            // считываем строку значений команды в формате node_id
var_name var_value

            int target_node_id;

            char var_name[30];

            size_t var_value;

            char input[100];
```

```
            char command_message[100];
            if (fgets(input, sizeof(input), stdin) == NULL) {
                perror("Ошибка чтения ввода\n");
                return 1;
            }
            // убираем \n в конце строки
            input[strcspn(input, "\n")] = 0;
            size_t words_count = check_command_length(input);
            printf("Input: %s, words_count: %zu\n", input, words_count);

            if (words_count == 2){
                sscanf(input, "%d %s", &target_node_id, var_name);
                snprintf(command_message, sizeof(command_message), "exec
for %d with %s %zu", target_node_id, var_name, words_count);
            }
            else if (words_count == 3){
                sscanf(input, "%d %s %zu", &target_node_id, var_name,
&var_value);
                snprintf(command_message, sizeof(command_message), "exec
for %d with %s %zu %zu", target_node_id, var_name, words_count,
var_value);
            }

            printf("Sending command message to publisher: %s\n",
command_message);
            zstr_send(publisher, command_message);
        }
        // pingall
        else if (strcmp(command, "pingall") == 0){
            char message[100];
            // отправляем родительский pingall
            snprintf(message, sizeof(message), "pingall %d", -1);
```

```
            zstr_send(publisher, message);
            printf("Sent pingall command: %s\n", message);
        }
        else if (strcmp(command, "kill") == 0){
            printf("Exiting program\n");
            exit(EXIT_SUCCESS);
        }
    }

    zpoller_destroy(&poller);
    return 0;
}

size_t check_command_length(char* input){
    char *str_copy = strdup(input);
    if (str_copy == NULL) {
        perror("Ошибка выделения памяти\n");
        return -1;
    }

    char *token;
    size_t word_count = 0;

    token = strtok(str_copy, " \t\n");
    while (token != NULL) {
        word_count++;
        token = strtok(NULL, " \t\n");
    }

    free(str_copy);
```

```
    return word_count;
}


Node* find_node_by_id(Node* head_node, int target_id){
    printf("Finding node by id: %d\n", target_id);
    if (head_node->id == target_id){
        printf("Found node: %p\n", (void*)head_node);
        return head_node;
    }else{
        for(size_t i = 0; i < head_node->childs_count; i++){
            printf("Checking child node: %d\n", head_node->childs[i].id);
            Node* result = find_node_by_id(&head_node->childs[i],
target_id);
            if (result != NULL){
                return result;
            }
        }
    }
    printf("Node not found\n");
    return NULL;
}


void add_child_to_node(Node* node, Node* child){
    printf("Adding child to node\n");
    if (node->childs == NULL){
        node->childs = malloc(sizeof(Node) * 3);
        if (node->childs == NULL) {
            perror("Failed to allocate memory for childs");
            exit(EXIT_FAILURE);
        }
```

```
        node->childs_container_size = 3;
    }
    if (node->childs != NULL){
        if (node->childs_count == node->childs_container_size){
            printf("Container is full\n");
            Node* buffer = realloc(node->childs,
node->childs_container_size * 2 * sizeof(Node));
            if (buffer == NULL){
                perror("Failed to reallocate memory for childs");
                exit(EXIT_FAILURE);
            }
            node->childs = buffer;
            node->childs_container_size = node->childs_container_size * 2;
        }
        node->childs[node->childs_count] = *child;
        printf("%d\n", node->childs[node->childs_count].id);
        node->childs_count++;
    }
    printf("Child added to node\n");
}


bool contains(int *array, size_t size, int element) {
    for (size_t i = 0; i < size; i++) {
        if (array[i] == element) {
            return true;
        }
    }
    return false;
}
```

```
int* compare_arrays(int *array1, size_t size1, int *array2, size_t size2)
{
    int arr[100] = {0};
    size_t counter = 0;
    for (size_t i = 0; i < size2; i++) {
        if (!contains(array1, size1, array2[i])) {
            arr[counter++] = array2[i];
        }
    }
    return arr;
}
```

Counting_node.c

```
#include <czmq.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    char* var_name;
    size_t var_value;
} CustomVar;

typedef struct{
    CustomVar* vars_container;
    size_t vars_count;
    size_t vars_container_size;
} VarContainer;
```

```
void add_to_var_container(VarContainer* vars_container, CustomVar*
some_var);

bool check_var_is_existing(VarContainer* vars_container, CustomVar*
var_to_check);

CustomVar find_var(VarContainer* vars_container, CustomVar* var_to_check);


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

    int this_node_id = atoi(argv[1]);

    int parent_node_id = atoi(argv[2]);

    VarContainer var_container = {NULL, 0, 0};

    int target_node_id;

    char to_send[100];

    //формируем строку адреса слушания

    char address[100];

    size_t port_to_sub;

    size_t port_to_pub = 5555 + this_node_id;

    if (parent_node_id == -1){

        port_to_sub = 5555;

    }else{

        port_to_sub = 5555 + parent_node_id;

    }

    snprintf(address, sizeof(address), "tcp://localhost:%zu",
port_to_sub);

    zsock_t* subscriber = zsock_new_sub(address, ""); //слушаем все,
фильтровать будем потом

    snprintf(address, sizeof(address), "tcp://localhost:%zu",
port_to_pub);

    zsock_t* publisher = zsock_new_pub(address);


    printf("CHILD INFO\n");
```

```
    printf("Node ID: %d, Parent Node ID: %d\n", this_node_id,
parent_node_id);
    printf("Subscriber address: %s\n", address);
    printf("Publisher address: %s\n", address);


    //обработка команд exec, sub, pingall, OK
    while (1){
        //считываем сообщение от издателя


        char* input_string = zstr_recv(subscriber);
        if (!input_string) {
            perror("Failed to receive message");
            exit(EXIT_FAILURE);
        }
        printf("Received message: %s\n", input_string);


        char command[100];
        int command_node_id;
        //ВАЖНО после sscanf каретка не передвигается по строке!!!!
        sscanf(input_string, "%s", command);
        printf("Command: %s\n", command);


        //обработка команды sub
        if (strcmp(command, "sub") == 0){
            int node_id_to_sub;
            sscanf(input_string, "sub for %d on %d", &command_node_id,
&node_id_to_sub);
            printf("Command Node ID: %d, Node ID to Sub: %d\n",
command_node_id, node_id_to_sub);
            //если команда для нас, то подписываемся на нашего нового
ребенка
```

```
            if (command_node_id == this_node_id){
                port_to_sub = 5555 + node_id_to_sub;
                snprintf(address, sizeof(address), "tcp://localhost:%zu",
port_to_sub);
                zsock_connect(subscriber, address);
                printf("Subscribed to new child node at address: %s\n",
address);
            }
            //если команда не для нас, то отсылаем нашим детям
            else{
                zstr_send(publisher, input_string);
                printf("Forwarding sub command to children: %s\n",
input_string);
            }
        }
        else if (strcmp(command, "exec") == 0){
            //"exec for %zu with %s %zu", target_node_id, var_name,
words_count
            //or "exec for %zu with %s %zu %zu", target_node_id, var_name,
words_count, var_value
            sscanf(input_string, "exec for %d", &target_node_id);
            printf("Target Node ID: %d\n", target_node_id);
            //если мы не таргет нода, то отсылаем подписчикам!!!!
            if (target_node_id != this_node_id){
                zstr_send(publisher, input_string);
                printf("Forwarding exec command to children: %s\n",
input_string);
            }
            else{
                //логика обработки exec, если мы таргет нода
                char var_name[100];
                size_t words_count;
```

```
                size_t var_value;
                sscanf(input_string, "exec for %d with %s %zu",
&target_node_id, var_name, &words_count);
                CustomVar curr_var;
                bool is_exists;
                printf("Words count: %zu\n", words_count);
                if (words_count == 2){
                    curr_var = (CustomVar){strdup(var_name), 0};
                    if (check_var_is_existing(&var_container, &curr_var)){
                        curr_var = find_var(&var_container, &curr_var);
                        snprintf(to_send, sizeof(to_send), "for %d f1
OK:%d:%zu", parent_node_id, this_node_id, curr_var.var_value);
                        zstr_send(publisher, to_send);
                        printf("Sending response for exec command: %s\n",
to_send);
                    }
                    else{
                        snprintf(to_send, sizeof(to_send), "for %d f2
OK:%d: %s Not found", parent_node_id, this_node_id, curr_var.var_name);
                        zstr_send(publisher, to_send);
                        printf("Sending response for exec command: %s\n",
to_send);
                    }
                    free(curr_var.var_name);
                }
                else if(words_count == 3){
                    sscanf(input_string, "exec for %d with %s %zu %zu",
&target_node_id, var_name, &words_count, &var_value);
                    curr_var = (CustomVar){strdup(var_name), var_value};
                    add_to_var_container(&var_container, &curr_var);
                    snprintf(to_send, sizeof(to_send), "for %d f3 OK:%d",
parent_node_id, this_node_id);
```

```
                    zstr_send(publisher, to_send);
                    printf("Sending response for exec command: %s\n",
to_send);
                    free(curr_var.var_name);
                }
            }
        }
        else if(strcmp(command, "pingall") == 0){
            //сначала отправляем сообщение детям и через себя передаем
наверх по дереву
            int parent_ping_id;
            sscanf(input_string, "pingall %d", &parent_ping_id);
            printf("Parent Ping ID: %d\n", parent_ping_id);
            if (parent_ping_id == parent_node_id){
                char message[100];
                //отправляем родительский pingall
                snprintf(message, sizeof(message), "pingall %d",
this_node_id);
                zstr_send(publisher, message);
                printf("Sending parent pingall: %s\n", message);
            }
            else if(parent_ping_id != this_node_id){
                char message[100];
                //отправляем дочерний pingall
                snprintf(message, sizeof(message), "pingall %d",
parent_ping_id);
                zstr_send(publisher, message);
                printf("Sending child pingall: %s\n", message);
            }
        }
        //логика обработки входящего сообщения обработки команды exec
```

```
        else if (strcmp(command, "for") == 0){
            char format[10];
            char message[100];
            int local_parent_node_id;
            int local_this_node_id;
            char var_name[40];
            int var_value;
            sscanf(input_string, "for %d %s", &target_node_id, format);
            printf("Format: %s\n", format);
            //если адресовано нам то обрабатываем, а если нет, то забиваем
(чтобы не было бесконечной отправки тудым сюдым)
            if (target_node_id == this_node_id){
                //format1 --- "for %zu f1 OK:%zu:%zu", parent_node_id,
this_node_id, curr_var.var_value
                if (strcmp(format, "f1") == 0){
                    sscanf(input_string, "for %d f1 OK:%d:%d",
&local_parent_node_id, &local_this_node_id, &var_value);
                    snprintf(message, sizeof(message), "for %d f1
OK:%d:%d", parent_node_id, local_this_node_id, var_value);
                }
                //format2 --- "for %zu f2 OK:%zu: %s Not found",
parent_node_id, this_node_id, curr_var.var_name
                else if (strcmp(format, "f2") == 0){
                    sscanf(input_string, "for %d f2 OK:%d: %s Not found",
&local_parent_node_id, &local_this_node_id, var_name);
                    snprintf(message, sizeof(message), "for %d f2 OK:%d:
%s Not found", parent_node_id, local_this_node_id, var_name);
                }
                //format3 --- "for %zu f3 OK:%zu", parent_node_id,
this_node_id
                else if (strcmp(format, "f3") == 0){
                    sscanf(input_string, "for %d f3 OK:%d",
&local_parent_node_id, &local_this_node_id);
```

```
                    snprintf(message, sizeof(message), "for %d f3 OK:%d",
parent_node_id, local_this_node_id);

                }

                zstr_send(publisher, message);

                printf("Sending response for exec command: %s\n",
message);

            }

        }

        free(input_string);

    }

}


void add_to_var_container(VarContainer* vars_container, CustomVar*
some_var){

    printf("Adding variable to container: %s, %zu\n", some_var->var_name,
some_var->var_value);

    if (vars_container->vars_count == 0 && vars_container->vars_container
== NULL){

        vars_container->vars_container = malloc(sizeof(CustomVar) * 3);

        vars_container->vars_container_size = 3;

    }

    if (vars_container->vars_count ==
vars_container->vars_container_size){

        CustomVar* buffer = realloc(vars_container->vars_container,
vars_container->vars_container_size * 2 * sizeof(CustomVar));

        if (buffer == NULL){

            perror("Ошибка выделения памяти\n");

            exit(EXIT_FAILURE);

        }

        vars_container->vars_container = buffer;

        vars_container->vars_container_size =
vars_container->vars_container_size * 2;
```

```
        vars_container->vars_container[vars_container->vars_count] =
*some_var;

        vars_container->vars_count++;

    }

    else if (vars_container->vars_count <
vars_container->vars_container_size){

        vars_container->vars_container[vars_container->vars_count] =
*some_var;

        printf("VarName: %s\n",
vars_container->vars_container[vars_container->vars_count].var_name);

        vars_container->vars_count++;

    }

    printf("Variable added to container: %s, %zu\n", some_var->var_name,
some_var->var_value);

}


bool check_var_is_existing(VarContainer* vars_container, CustomVar*
var_to_check){

    printf("Checking if variable exists: %s\n", var_to_check->var_name);

    CustomVar* vars_container_local = vars_container->vars_container;

    for (size_t i = 0; i < vars_container->vars_count; i++){

        if (strcmp(vars_container_local[i].var_name,
var_to_check->var_name) == 0){

            printf("Variable exists: %s\n", var_to_check->var_name);

            return true;

        }

    }

    printf("Variable does not exist: %s\n", var_to_check->var_name);

    return false;

}


CustomVar find_var(VarContainer* vars_container, CustomVar* var_to_check){
```

```
    printf("Finding variable: %s\n", var_to_check->var_name);

    CustomVar* vars_container_local = vars_container->vars_container;

    for (size_t i = 0; i < vars_container->vars_count; i++){

        if (strcmp(vars_container_local[i].var_name,
var_to_check->var_name) == 0){

            printf("Variable found: %s, %zu\n", var_to_check->var_name,
vars_container_local[i].var_value);

            return vars_container_local[i];

        }

    }

    printf("Variable not found: %s\n", var_to_check->var_name);

    return (CustomVar){NULL, 0};

}
```

Демонстрация работы программы

rissochek@admin:/mnt/c/Users/rusla/coding/OSLabs/lab5-7$ ./control
Starting main function
Publisher socket created
create 10 -1
Received command: create
create_node_id: 10, parent_id_for_new_node: -1
Finding node by id: -1
Found node: 0x7ffed23d3070
Finding node by id: 10
Node not found
Adding child to node
10
Child added to node
New node created and added to parent
Subscriber socket created and connected to tcp://localhost:5565
CHILD INFO

Node ID: 10, Parent Node ID: -1
Subscriber address: tcp://localhost:5565
Publisher address: tcp://localhost:5565
exec 10 my 1
Received command: exec
Input: 10 my 1, words_count: 3
Sending command message to publisher: exec for 10 with my 3 1
Received message: exec for 10 with my 3 1
Command: exec
Target Node ID: 10
Words count: 3
Adding variable to container: my, 1
VarName: my
Variable added to container: my, 1
Sending response for exec command: for -1 f3 OK:10
Received message: for -1 f3 OK:10
Command: for
Response to exec command: OK:10
exec 10 my
Received command: exec
Input: 10 my, words_count: 2
Sending command message to publisher: exec for 10 with my 2
Received message: exec for 10 with my 2
Command: exec
Target Node ID: 10
Words count: 2
Checking if variable exists: my
Variable exists: my
Finding variable: my
Variable found: my, 1
Sending response for exec command: for -1 f1 OK:10:1
Received message: for -1 f1 OK:10:1
Command: for
Response to exec command: OK:10:1
create 20 10
Received command: create
create_node_id: 20, parent_id_for_new_node: 10
Finding node by id: 10
Checking child node: 10
Finding node by id: 10
Found node: 0x562639470eb0
Finding node by id: 20

Checking child node: 10
Finding node by id: 20
Node not found
Node not found
Adding child to node
20
Child added to node
New node created and added to parent
Sent sub command: sub for 10 on 20
Received message: sub for 10 on 20
Command: sub
Command Node ID: 10, Node ID to Sub: 20
Subscribed to new child node at address: tcp://localhost:5575
CHILD INFO
Node ID: 20, Parent Node ID: 10
Subscriber address: tcp://localhost:5575
Publisher address: tcp://localhost:5575
exec 20 my 1
Received command: exec
Input: 20 my 1, words_count: 3
Sending command message to publisher: exec for 20 with my 3 1
Received message: exec for 20 with my 3 1
Command: exec
Target Node ID: 20
Forwarding exec command to children: exec for 20 with my 3 1
Received message: exec for 20 with my 3 1
Received message: exec for 20 with my 3 1
Command: exec
Command: exec
Target Node ID: 20
Words count: 3
Adding variable to container: my, 1
VarName: my
Variable added to container: my, 1
Sending response for exec command: for 10 f3 OK:20
Received message: for 10 f3 OK:20
Command: for
Format: f3
Sending response for exec command: for -1 f3 OK:20
Received message: for -1 f3 OK:20
Command: for
Format: f3

Выводы

В ходе выполнения лабораторной работы я изучил основы работы с очередями сообщений ZeroMQ и реализовал программу с использованием этой библиотеки. Самым удобным вариантом коммуникации между узлами мне показался паттерн SUB/PUB из-за массовой рассылки, однако из-за этого же преимущества пришлось долго настраивать содержимое команд.

Когда параллельных вычислений становится мало, на помощь приходят распределённые вычисления (распределение вычислений осуществляется уже не между потоками процессора, а между отдельными ЭВМ). Очереди сообщений используются для взаимодействия нескольких машин в одной большой сети. Опыт работы с ZeroMQ пригодится мне при настройке собственной системы распределённых вычислений.