Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6-8 по курсу**

**«Операционные системы»**

Студент: Рылов Александр Дмитриевич

Группа: М8О-207Б-21

Вариант: 24

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/Brokiloene/os

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:  
 Управлении серверами сообщений (№2)  
 Применение отложенных вычислений (№4)  
 Интеграция программных систем друг с другом (№3)

**Задание**

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать два 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;

• Удаление существующего вычислительного узла;

• Исполнение команды на вычислительном узле;

• Проверка доступности вычислительного узла.

**Задание варианта**

Вариант 24.

Дерево общего вида

Исполнение команды — поиск подстроки в строке.

Команда проверки — проверка доступности всех узлов.

**Общие сведения о программе**

Программа распределительного узла компилируется из файла main.c, программа вычислительного узла компилируется из файла node.c. В программе используется библиотека для работы с сервером сообщений ZeroMQ. В программе используются следующие системные вызовы:

fork –– создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.

execl — используется для выполнения другой программы. Эта другая программа, называемая процессом-потомком (child process), загружается поверх программы, содержащей вызов exec. Имя файла, содержащего процесс-потомок, задано с помощью первого аргумента. Какие-либо аргументы, передаваемые процессу-потомку, задаются либо с помощью параметров от arg0 до argN, либо с помощью массива arg[].

Также были использованы следующие вызовы из библиотеки ZMQ:

zmq\_ctx\_new –– создает новый контекст ZMQ.

zmq\_connect — создает входящее соединение на сокет.

zmq\_disconnect — отсоединяет сокет от заданного endpoint’a.

zmq\_socket — создает ZMQ сокет.

zmq\_close — закрывает ZMQ сокет.

zmq\_ctx\_destroy –– уничтожает контекст ZMQ.

**Общий метод и алгоритм решения**

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с ZMQ.

2. Проработать принцип общения между клиентскими узлами и между первым клиентом и сервером и алгоритм выполнения команд клиентами.

3. Реализовать необходимые функции-обертки над вызовами функций библиотеки ZMQ.

4. Написать программу сервера и клиента

**Исходный код**

**manage\_node.c**

#include "list.h"

#include "lab6\_utils.h"

#include <zmq.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> //fork

#include <string.h> //strcmp

#include <sys/wait.h>

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

{

void \*context = zmq\_ctx\_new();

void \*requester = zmq\_socket(context, ZMQ\_REQ);

check\_null(requester, "requester error");

List \*childs = list\_create();

check\_null(childs, "list\_create error");

char cmd[10];

int c;

printf("> ");

while((c = getchar()) != EOF) {

ungetc(c, stdin);

//printf("[mn]: list: ");

//print\_list(childs);

printf("> ");

scanf("%s", cmd);

if (strcmp(cmd, "create") == 0) {

//print\_list(childs);

int id, parent\_id;

scanf("%d %d", &id, &parent\_id);

id += PORT;

if (parent\_id == -1) {

if (list\_find(childs, id) == 1) {

printf("bad child id\n");

continue;

}

int pid = fork();

check\_neg\_one(pid, "fork error");

if (pid == 0) {

zmq\_close(requester);

zmq\_ctx\_destroy(context);

list\_destroy(childs);

char id\_str[8];

memset(id\_str, 0, 8);

sprintf(id\_str, "%d", id);

char \*argvc[] = {"computing\_node", id\_str, NULL};

check\_neg\_one(execv("computing\_node", argvc), "execv computing\_node error");

}

//id -= PORT;

//printf("[mn59]: %s %d %d\n", cmd, id, parent\_id);

sleep(1); // если создать/удалить/снова создать процесс, ping не сработает (???)

//printf("ping= %d\n", ping(id));

if (ping(id) == 1) {

push\_back(childs, id);

printf("computing\_node № %d with pid [%d] has been created\n", id, pid);

}

}

else {

parent\_id += PORT;

// if (list\_find(childs, parent\_id) == 0 || list\_find(childs, id) == 1) {

// printf("bad parent/child id\n");

// continue;

// }

//printf("I am here\n");

if (list\_find(childs, parent\_id) == 1 && ping(parent\_id) == 1) {

void \*requester2 = zmq\_socket(context, ZMQ\_REQ);

Message m = {"create", parent\_id, id, ""};

//printf("childid=%d, parent\_id=%d\n", id, parent\_id);

char address[32];

client\_address\_gen(parent\_id, address);

//printf("address=%s", address);

check\_neg\_one(zmq\_connect(requester2, address), "zmq\_connect error");

send\_msg(&m, requester2);

recv\_msg(&m, requester2);

zmq\_close(requester2);

//push\_back(childs, m.id);

if (strcmp(m.cmd, "bad") == 0) {

printf("%s: error with creating [%d] by [%d]\n", m.cmd, m.id, m.num);

} else {

printf("%s: [%d] was created by [%d]\n", m.cmd, m.id, parent\_id);

}

}

else {

//printf("I am here\n");

// break;

int cur\_id = echo(childs, parent\_id, requester);

//printf("[mn]: cur\_id = %d\n", cur\_id);

if (cur\_id == -1 || ping(cur\_id) != 1) {

printf("bad id\n");

continue;

}

Message m = {"create", parent\_id, id, ""};

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

//printf("[mn110]: %s %d %d\n", m.cmd, m.id, m.num);

if (strcmp(m.cmd, "bad") == 0) {

printf("%s: error with creating [%d] by [%d]\n", m.cmd, m.id, m.num);

}

else if (strcmp(m.cmd, "ok") == 0) {

printf("%s: [%d] was created by [%d] via [%d]\n", m.cmd, m.id, parent\_id, cur\_id);

}

}

}

} else if (strcmp(cmd, "exec") == 0) {

int id;

scanf("%d", &id);

id += PORT;

getchar(); // scanf сам не считывает '\n'

char \*pattern = (char \*) malloc(sizeof(char) \* 1024), \*text = NULL;

size\_t len = 0;

int pattern\_size = getline(&pattern, &len, stdin);

int text\_size = getline(&text, &len, stdin);

pattern\_size -= 1; // getline выдает количество считанных символов

pattern[pattern\_size] = '#'; // " для char \*, ' для char

char \*all = strcat(pattern, text);

Message m = {"exec", pattern\_size, id, ""};

memset(m.str, 0, 1024);

memcpy(m.str, all, strlen(all));

//printf("[mn]: exec id=%d, lf=%d\n", id, list\_find(childs, id));

if (list\_find(childs, id) == 1 && ping(id) == 1) {

void \*requester2 = zmq\_socket(context, ZMQ\_REQ);

//printf("145\n");

char address[32];

client\_address\_gen(id, address);

printf("[mn]: address=%s\n", address);

//check\_neg\_one(zmq\_connect(requester2, address), "zmq\_connect error");

//printf("152\n");

send\_msg(&m, requester2);

//printf("154\n");

recv\_msg(&m, requester2);

//printf("156\n");

printf("%s:\n%s \nproduced by [%d]\n", m.cmd, m.str, m.id);

zmq\_close(requester2);

} else {

//printf("160\n");

void \*echo\_requester = zmq\_socket(context, ZMQ\_REQ);

int cur\_id = echo(childs, id, echo\_requester);

zmq\_close(echo\_requester);

//printf("[mn]: cur\_id = %d\n", cur\_id);

if (cur\_id == -1 || ping(cur\_id) == -1) {

printf("bad id\n");

free(pattern);

free(text);

continue;

}

char address[32];

client\_address\_gen(cur\_id, address);

//printf("[mn]: address=%s\n", address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

//printf("[mn110]: %s %d %d\n", m.cmd, m.id, m.num);

if (strcmp(m.cmd, "bad") == 0) {

printf("bad id\n");

//printf("180\n");

}

else if (strcmp(m.cmd, "ok") == 0) {

printf("%s:\n%s \nproduced by [%d]\n", m.cmd, m.str, m.id);

}

}

free(pattern);

free(text);

}

else if (strcmp(cmd, "ping") == 0) {

int id;

scanf("%d", &id);

id += PORT;

printf("[%d]: %d\n", id, ping(id));

}

else if (strcmp(cmd, "remove") == 0) {

int id;

scanf("%d", &id);

id += PORT;

Message m = {"remove", 0, id, ""};

//printf("[mn]: %s %d %d\n", m.cmd, m.id, m.num);

if (list\_find(childs, id) == 1 && ping(id) == 1) {

void \*requester2 = zmq\_socket(context, ZMQ\_REQ);

char address[32];

client\_address\_gen(id, address);

check\_neg\_one(zmq\_connect(requester2, address), "zmq\_connect error");

send\_msg(&m, requester2);

recv\_msg(&m, requester2);

list\_delete(childs, id);

zmq\_close(requester2);

printf("%s: [%d] with pid %d has been removed\n", m.cmd, m.id, m.num);

wait(NULL); // а то будет зомби

} else {

int cur\_id = echo(childs, id, requester);

//printf("[mn]: cur\_id = %d\n", cur\_id);

if (cur\_id == -1 || ping(cur\_id) == -1) {

printf("bad id\n");

continue;

}

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

if (strcmp(m.cmd, "bad") == 0) {

printf("%s: cannot remove %d", m.cmd, m.id);

}

else if (strcmp(m.cmd, "ok") == 0) {

printf("%s: [%d] with pid %d has been removed via [%d]\n", m.cmd, m.id, m.num, cur\_id);

}

}

}

else if (strcmp(cmd, "heartbit") == 0) {

int delay;

scanf("%d", &delay);

for (int i = 0, sz = size(childs); i < sz; ++i) {

int cur\_id = list\_get(childs, i);

check\_neg\_one(cur\_id, "get error");

//printf("[mn]: delay=%d cur\_id=%d\n", delay, cur\_id);

int answers = 0;

for (int j = 0; j < 4; ++j) {

if (ping(cur\_id) == 1) {

answers += 1;

}

usleep(delay \* 1000);

}

if (answers == 4) {

printf("[%d]: ready\n", cur\_id);

Message m = {"heartbit", delay, 0, ""};

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

zmq\_disconnect(requester, address);

} else {

printf("[%d]: bad\n", cur\_id);

}

}

}

}

for (int i = 0, sz = size(childs); i < sz; ++i) {

void \*requester2 = zmq\_socket(context, ZMQ\_REQ);

int cur\_id = list\_get(childs, i);

check\_neg\_one(cur\_id, "get error");

if (ping(cur\_id) == 1) {

Message m2 = {"remove", 0, cur\_id, ""};

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(requester2, address), "zmq\_connect error");

send\_msg(&m2, requester2);

recv\_msg(&m2, requester2);

zmq\_close(requester2);

}

}

zmq\_close(requester);

zmq\_ctx\_destroy(context);

list\_destroy(childs);

return 0;

}

**computing\_node.c**

#include "list.h"

#include "lab6\_utils.h"

#include <zmq.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> //fork

#include <string.h> //strcmp

#include <sys/wait.h>

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

{

if (argc < 2) {

perror("argc < 2");

exit(EXIT\_FAILURE);

}

int my\_id = atoi(argv[1]);

//printf("%d\n", my\_id);

List \*childs = list\_create();

check\_null(childs, "list\_create error");

void \*context = zmq\_ctx\_new();

void \*responder = zmq\_socket(context, ZMQ\_REP);

char address[32];

memset(address, 0, 32);

memcpy(address, SERVER\_PATTERN, sizeof(SERVER\_PATTERN));

strcat(address, argv[1]);

check\_neg\_one(zmq\_bind(responder, address), "zmq bind error");

//printf("[%d]: bound to %s\n", my\_id, address);

while(1) {

Message m;

recv\_msg(&m, responder);

//printf("[%d]: %s %d %d\n", my\_id, m.cmd, m.id, m.num);

if (strcmp(m.cmd, "create") == 0) {

if (m.num == my\_id) {

if (list\_find(childs, m.id) == 1 || my\_id == m.id) {

//printf("I am here\n");

//printf("[%d]: %s %d %d\n", my\_id, m.cmd, m.id, m.num);

memcpy(&m.cmd, "bad", sizeof(char) \* 10);

send\_msg(&m, responder);

}

else {

int pid = fork();

check\_neg\_one(pid, "fork error");

if (pid == 0) {

zmq\_close(responder);

zmq\_ctx\_destroy(context);

list\_destroy(childs);

char id\_str[8];

memset(id\_str, 0, 8);

sprintf(id\_str, "%d", m.id);

char \*argvc[] = {"computing\_node", id\_str, NULL};

check\_neg\_one(execv("computing\_node", argvc), "execv computing\_node error");

}

sleep(1);

memcpy(&m.cmd, "ok", sizeof(char) \* 10);

m.num = pid;

send\_msg(&m, responder);

push\_back(childs, m.id);

}

} else {

//printf("[%d]: %s %d %d\n", my\_id, m.cmd, m.id, m.num);

if (list\_find(childs, m.num) == 1 && ping(m.num) == 1) {

Message m2 = {"create", m.num, m.id, ""};

//printf("%s %d %d\n", m2.cmd, m2.id, m2.num);

void \*requester = zmq\_socket(context, ZMQ\_REQ);

char address[32];

client\_address\_gen(m2.num, address);

// printf("address=%s", address);

// printf("I am here\n");

// break;

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m2, requester);

recv\_msg(&m2, requester);

zmq\_close(requester);

//memcpy(&m2.cmd, "ok", sizeof(char) \* 10);

send\_msg(&m2, responder);

//push\_back(childs, m.id);

//printf("%s: [%d] was created by [%d]\n", m.cmd, m.id, parent\_id);

}

else {

void \*requester = zmq\_socket(context, ZMQ\_REQ);

int cur\_id = echo(childs, m.num, requester);

if (cur\_id == -1 || ping(cur\_id) == -1) {

printf("bad\_id\n");

memcpy(&m.cmd, "bad", sizeof(char) \* 10);

send\_msg(&m, responder);

continue;

}

//printf("[%d]: list:\n", my\_id);

//print\_list(childs);

//printf("[%d]113: cur\_id = %d\n", my\_id, cur\_id);

Message m2 = {"create", m.num, m.id, ""};

//printf("[%d]: %s %d %d\n", my\_id, m2.cmd, m2.id, m2.num);

char address[32];

client\_address\_gen(cur\_id, address);

//printf("[%d]: address=%s", my\_id, address);

//sleep(30);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m2, requester);

recv\_msg(&m2, requester);

zmq\_close(requester);

send\_msg(&m2, responder);

//push\_back(childs, m.id);

//printf("%s: [%d] was created by [%d] via [%d]\n", m.cmd, m.id, parent\_id, cur\_id);

}

}

}

else if (strcmp(m.cmd, "exec") == 0) {

//printf("[%d]: %s %d %d %s\n", my\_id, m.cmd, m.id, m.num, m.str);

//print\_list(childs);

if (m.id == my\_id) {

memcpy(&m.cmd, "ok", sizeof(char) \* 10);

find\_substrings(&m);

send\_msg(&m, responder);

} else if (list\_find(childs, m.id) == 1 && ping(m.id) == 1) {

void \*requester = zmq\_socket(context, ZMQ\_REQ);

char address[32];

client\_address\_gen(m.id, address);

//printf("[%d]150: address=%s", my\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

zmq\_close(requester);

send\_msg(&m, responder);

}

else {

void \*requester = zmq\_socket(context, ZMQ\_REQ);

int cur\_id = echo(childs, m.id, requester);

if (cur\_id == -1 || ping(cur\_id) == -1) {

printf("bad\_id\n");

memcpy(&m.cmd, "bad", sizeof(char) \* 10);

send\_msg(&m, responder);

continue;

}

//printf("[%d]: list:\n", my\_id);

//print\_list(childs);

//printf("[%d]157: cur\_id = %d\n", my\_id, cur\_id);

//Message m2 = {"create", m.num, m.id, ""};

//printf("[%d]: %s %d %d\n", my\_id, m2.cmd, m2.id, m2.num);

char address[32];

client\_address\_gen(cur\_id, address);

//printf("[%d]: address=%s", my\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

zmq\_close(requester);

send\_msg(&m, responder);

}

}

else if (strcmp(m.cmd, "remove") == 0) {

//printf("[%d]: %s %d %d\n", my\_id, m.cmd, m.id, m.num);

if (m.id == my\_id) {

for (int i = 0, sz = size(childs); i < sz; ++i) {

int cur\_id = list\_get(childs, i);

check\_neg\_one(cur\_id, "get error");

if (ping(cur\_id) == 1) {

void \*requester = zmq\_socket(context, ZMQ\_REQ);

Message m2 = {"remove", 0, cur\_id, ""};

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m2, requester);

recv\_msg(&m2, requester);

zmq\_close(requester);

}

}

//printf("[%d]: %s %d %d\n", my\_id, m.cmd, m.id, m.num);

memcpy(&m.cmd, "ok", sizeof(char) \* 10);

m.num = getpid();

send\_msg(&m, responder);

break;

} else if (list\_find(childs, m.id) == 1 && ping(m.id) == 1) {

Message m2 = {"remove", 0, m.id, ""};

//printf("%s %d %d\n", m2.cmd, m2.id, m2.num);

void \*requester = zmq\_socket(context, ZMQ\_REQ);

char address[32];

client\_address\_gen(m2.id, address);

//printf("[%d]: address=%s", my\_id, address);

// printf("address=%s", address);

// printf("I am here\n");

// break;

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m2, requester);

recv\_msg(&m2, requester);

zmq\_close(requester);

wait(NULL);

list\_delete(childs, m.id);

memcpy(&m2.cmd, "ok", sizeof(char) \* 10);

send\_msg(&m2, responder);

}

else {

void \*requester = zmq\_socket(context, ZMQ\_REQ);

int cur\_id = echo(childs, m.id, requester);

//printf("[%d]: cur\_id = %d\n", my\_id, cur\_id);

if (cur\_id == -1 || ping(cur\_id) == -1) {

memcpy(&m.cmd, "bad", sizeof(char) \* 10);

send\_msg(&m, responder);

continue;

}

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

zmq\_close(requester);

memcpy(&m.cmd, "ok", sizeof(char) \* 10);

send\_msg(&m, responder);

}

}

else if (strcmp(m.cmd, "echo") == 0) {

//printf("[%d-echo]: %s %d %d %s\n", my\_id, m.cmd, m.id, m.num, m.str);

if ((list\_find(childs, m.id) == 1 && ping(m.id) == 1) || my\_id == m.id) {

memcpy(&m.cmd, "ok", sizeof(char) \* 10);

send\_msg(&m, responder);

} else {

void \*requester = zmq\_socket(context, ZMQ\_REQ);

int cur\_id = echo(childs, m.id, requester);

zmq\_close(requester);

//printf("[%d]264: cur\_id = %d\n", my\_id, cur\_id);

if (cur\_id == -1 || ping(cur\_id) == -1) {

memcpy(&m.cmd, "bad", sizeof(char) \* 10);

send\_msg(&m, responder);

} else {

memcpy(&m.cmd, "ok", sizeof(char) \* 10);

send\_msg(&m, responder);

}

}

} else if (strcmp(m.cmd, "heartbit") == 0) {

int delay = m.num;

void \*requester = zmq\_socket(context, ZMQ\_REQ);

for (int i = 0, sz = size(childs); i < sz; ++i) {

int cur\_id = list\_get(childs, i);

check\_neg\_one(cur\_id, "get error");

//printf("[%d]: delay=%d cur\_id=%d\n", my\_id, delay, cur\_id);

int answers = 0;

for (int j = 0; j < 4; ++j) {

if (ping(cur\_id) == 1) {

answers += 1;

}

usleep(delay \* 1000);

}

if (answers == 4) {

printf("[%d]: ready\n", cur\_id);

Message m = {"heartbit", delay, 0, ""};

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(requester, address), "zmq\_connect error");

send\_msg(&m, requester);

recv\_msg(&m, requester);

zmq\_disconnect(requester, address);

} else {

printf("[%d]: bad\n", cur\_id);

}

}

zmq\_close(requester);

memcpy(&m.cmd, "ok", sizeof(char) \* 10);

send\_msg(&m, responder);

}

}

zmq\_close(responder);

zmq\_ctx\_destroy(context);

list\_destroy(childs);

printf("[%d] dying...\n", my\_id);

return 0;

}

**lab6\_utils.c**

#include "lab6\_utils.h"

#include "list.h"

#include <zmq.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

void send\_msg(Message \*m, void \*socket)

{

zmq\_msg\_t msg;

zmq\_msg\_init(&msg);

zmq\_msg\_init\_size(&msg, sizeof(Message)); //check -1

memcpy(zmq\_msg\_data(&msg), m, sizeof(Message)); //check NULL

zmq\_msg\_send(&msg, socket, 0); //check -1

zmq\_msg\_close(&msg);

}

void recv\_msg(Message \*m, void \*socket)

{

zmq\_msg\_t msg;

zmq\_msg\_init(&msg);

zmq\_msg\_recv(&msg, socket, 0); //check -1

//m = (Message \*)zmq\_msg\_data(&msg); // check NULL

memcpy(m, zmq\_msg\_data(&msg), sizeof(Message));

zmq\_msg\_close(&msg);

}

int \*\_\_z\_function(char \*str, int size)

{

int \*z = calloc(size, sizeof(int));

//printf("%s", str);

for (int i = 1, l = 0, r = 0; i < size; ++i) {

if (i <= r) {

z[i] = min(r - i + 1, z[i - l]);

}

while(i + z[i] < size && str[z[i]] == str[i + z[i]]) {

++z[i];

// printf("%d size=%d %c %c\n", i + z[i], size, str[z[i]], str[i + z[i]]);

}

if (i + z[i] - 1 > r) {

l = i, r = i + z[i] - 1;

}

}

return z;

}

void print\_array(int size, int \*arr)

{

//size -= 1;

for (int i = 0; i < size; ++i) {

printf("%d ", arr[i]);

}

printf("\n");

}

void print\_part\_of\_str(char \*str, int from, int to, char \*res, int \*len)

{

for (int i = from; i <= to; ++i) {

\*len += sprintf(res + \*len, "%c", str[i]);

}

}

void find\_substrings(Message \*m)

{

int all\_size = strlen(m->str);

int \*z = \_\_z\_function(m->str, all\_size);

char \*res = (char \*) malloc(sizeof(char) \* 1024);

memset(res, 0, sizeof(char) \* 1024);

for (int i = m->num, len = 0; i < all\_size; ++i) {

if (z[i] == m->num) {

len += sprintf(res + len, "(%d): ", i - m->num);

print\_part\_of\_str(m->str, m->num + 1, i - 1, res, &len);

len += sprintf(res + len, "%s", GREEN);

print\_part\_of\_str(m->str, i, i + m->num - 1, res, &len);

len += sprintf(res + len, "%s", END\_COL);

print\_part\_of\_str(m->str, i + m->num, all\_size - 1, res, &len);

}

}

free(z);

memcpy(m->str, res, sizeof(char) \* 1024);

}

void client\_address\_gen(int id, char \*address)

{

memset(address, 0, 32);

memcpy(address, ADDRESS\_PATTERN, sizeof(ADDRESS\_PATTERN));

char s[8];

memset(s, 0, 8);

sprintf(s, "%d", id);

strcat(address, s);

}

char \*server\_address\_gen(int id)

{

char s[8];

char \*c = malloc(sizeof(char) \* 32);

strcat(c, "tcp://\*:");

//memcpy(c, ADDRESS\_PATTERN, sizeof(ADDRESS\_PATTERN));

//memset(s, 0, 8);

sprintf(s, "%d", id);

strcat(c, s);

return c;

}

int ping(int id)

{

char id\_str[8];

memset(id\_str, 0, 8);

sprintf(id\_str, "%d", id);

char inproc\_address[32];

memset(inproc\_address, 0, 32);

memcpy(inproc\_address, PING\_PATTERN, sizeof(PING\_PATTERN));

strcat(inproc\_address, id\_str);

char address[32];

memset(address, 0, 32);

memcpy(address, ADDRESS\_PATTERN, sizeof(ADDRESS\_PATTERN));

strcat(address, id\_str);

//printf("[in-ping]: address=%s\n", address);

void \*context = zmq\_ctx\_new();

void \*requester = zmq\_socket(context, ZMQ\_REQ);

zmq\_connect(requester, address);

zmq\_socket\_monitor(requester, inproc\_address, ZMQ\_EVENT\_CONNECTED | ZMQ\_EVENT\_CONNECT\_RETRIED);

void \*pair\_socket = zmq\_socket(context, ZMQ\_PAIR);

zmq\_connect(pair\_socket, inproc\_address);

zmq\_msg\_t m;

zmq\_msg\_init(&m);

zmq\_msg\_recv(&m, pair\_socket, 0);

uint8\_t\* data = (uint8\_t\*)zmq\_msg\_data(&m);

uint16\_t event = \*(uint16\_t\*)(data);

zmq\_close(requester);

zmq\_close(pair\_socket);

zmq\_msg\_close(&m);

zmq\_ctx\_destroy(context);

if (event == ZMQ\_EVENT\_CONNECT\_RETRIED) {

return -1;

} else {

return 1;

}

}

int echo(List \*childs, int id, void \*socket) // вернет id узла, у которого в потомках есть искомый id

{

for (int i = 0, sz = size(childs); i < sz; ++i) {

int cur\_id = list\_get(childs, i);

check\_neg\_one(cur\_id, "echo error");

//printf("[in-echo] cur=%d id=%d\n", cur\_id, id);

if (cur\_id == id) {

//printf("[in-echo]: 158 returned\n");

return cur\_id;

}

//Message m = {"echo", parent\_id, id, ""};

Message m = {"echo", 0, id, ""}; //id

//printf("[echo]: %s %d %d\n", m.cmd, m.id, m.num);

// return -1;

char address[32];

client\_address\_gen(cur\_id, address);

check\_neg\_one(zmq\_connect(socket, address), "zmq\_connect error");

send\_msg(&m, socket);

recv\_msg(&m, socket);

zmq\_disconnect(socket, address);

//printf("[in-echo]: %s %d %d\n", m.cmd, m.id, m.num);

if (strcmp(m.cmd, "ok") == 0) {

//printf("[in-echo]: 179 returned cur\_id=%d\n", cur\_id);

return cur\_id;

}

}

return -1;

}

**lab6\_utils.h**

#ifndef \_LAB6\_UTILS\_H\_

#define \_LAB6\_UTILS\_H\_

#include "list.h"

#define PORT 9000

//#define ADDRESS\_PATTERN "tcp://localhost:"

#define ADDRESS\_PATTERN "tcp://127.0.0.1:"

#define SERVER\_PATTERN "tcp://\*:"

#define PING\_PATTERN "inproc://ping-"

#define MAXSIZE 1024

#define MIN\_INDEXES 8

#define END\_COL "\x1b[0m"

#define GREEN "\x1b[32m"

#define min(a, b) a > b ? b : a;

#define check\_null(foo, msg) do { \

if (foo == NULL) { perror(msg); exit(EXIT\_FAILURE); } \

} while(0);

#define check\_neg\_one(foo, msg) do { int \_\_res = foo; \

if (\_\_res == -1) { perror(msg); exit(EXIT\_FAILURE); } \

} while(0);

typedef struct

{

char cmd[10];

int num;

int id;

char str[1024];

} Message;

void send\_msg(Message \*m, void \*socket);

void recv\_msg(Message \*m, void \*socket);

int \*\_\_z\_function(char \*str, int size);

void print\_array(int size, int \*arr);

void print\_part\_of\_str(char \*str, int from, int to, char \*res, int \*len);

void find\_substrings(Message \*m);

void client\_address\_gen(int id, char \*address);

char \*server\_address\_gen(int id);

int ping(int id);

int echo(List \*childs, int id, void \*socket);

#endif

**list.c**

#include "list.h"

#include <stdio.h>

#include <stdlib.h>

List \*list\_create()

{

List \*l = (List \*) malloc(sizeof(List));

l->values = (int \*) malloc(sizeof(int));

l->size = 0;

return l;

}

int size(List \*l)

{

return l->size;

}

void \_\_resize(List \*l)

{

l->values = realloc(l->values, sizeof(int) \* (l->size));

}

void push\_back(List \*l, int num)

{

l->size++;

\_\_resize(l);

l->values[l->size - 1] = num;

}

void print\_list(List \*l)

{

int sz = size(l);

// if (sz == 0) {

// return;

// }

for (int i = 0; i < sz; ++i) {

printf("%d ", l->values[i]);

}

printf("\n");

}

void list\_delete(List \*l, int num)

{

int sz = size(l);

for (int i = 0; i < sz; ++i) {

if (l->values[i] == num) {

for (int j = i; j < sz - 1; ++j) {

l->values[j] = l->values[j + 1];

}

l->size--;

break;

}

}

if (l->size != 0) {

\_\_resize(l);

}

}

int list\_find(List \*l, int num)

{

int sz = size(l);

for (int i = 0; i < sz; ++i) {

if (l->values[i] == num) {

return 1;

}

}

return -1;

}

void list\_destroy(List \*l)

{

free(l->values);

free(l);

}

int list\_get(List \*l, int ind)

{

int sz = size(l);

if (ind < sz) {

return l->values[ind];

} else {

return -1;

}

}

**list.h**

#ifndef \_LIST\_H\_

#define \_LIST\_H\_

typedef struct

{

int size;

int \*values;

} List;

List \*list\_create();

int size(List \*l);

void \_resize(List \*l);

void push\_back(List \*l, int num);

void print\_list(List \*l);

void list\_delete(List \*l, int num);

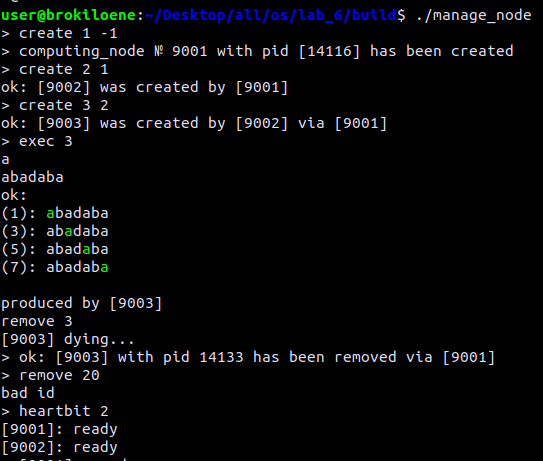
int list\_find(List \*l, int num);

void list\_destroy(List \*l);

int list\_get(List \*l, int ind);

#endif

**Демонстрация работы программы**



**Выводы**

В ходе выполнения лабораторной работы изучил основы работы с очередями сообщений ZMQ и реализовал программу с использованием этой библиотеки. В процессе открыл для себя много нового. Почти вся информация бралась из официальной документации.