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: Пивницкий Д.С.

Группа: М8о–206Б–19

Вариант: 11

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

· Управлении серверами сообщений (№6)

· Применение отложенных вычислений (№7)

· Интеграция программных систем друг с другом (№8)

**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.

**Описание программы**

Связь между вычислительными узлами будем поддерживать с помощью ZMQ\_PAIR. При инициализации установить время ожидания ZMQ\_SNDTIMEO и ZMQ\_RECVTIMEO, чтобы предусмотреть случай, когда дочерний процесс был убит. Для обмена информацией будем использовать специальную структуру node\_token\_t, в которой есть перечислимое поле actions. Вычислительные узлы обрабатывают каждое сообщение: если идентификатор сообщения не совпадает с идентификатором узла, то он отправляет сообщение дальше и ждёт ответа снизу. Каждый вычислительный узел имеет отдельный поток для вычислений и свою очередь вычислений. Чтобы получить результат вычислений обратно, нужно запросить их от вычислительного узла. Такой подход необходим, потому что неизвестно, сколько нужно ждать результат от узла. Для поиска подстроки в строке я использовал алгоритм Кнута-Морриса-Пратта с препроцессингом через Z-функцию строки.

**Набор тестов**

**Тест 1**

create 1 -1

create 2 1

create 3 2

create 4 3

create 5 4

ping 1

ping 2

ping 3

ping 4

ping 5

remove 5

remove 4

remove 3

remove 2

remove 1

**Тест 2**

create 1 -1

create 2 1

create 5 -1

create 3 2

create 6 5

create 7 5

create 4 3

create 8 5

create 9 8

create 0 1

remove 8

remove 4

remove 2

remove 6

remove 0

ping 1

ping 3

ping 5

ping 7

ping 9

remove 5

remove 7

remove 1

remove 9

remove 3

**Тест 3**

create 1 -1

create 10 1

create 5 1

create 7 5

create 3 1

create 6 5

create 8 7

create 2 1

create 4 3

create 9 8

ping 1

ping 2

ping 3

ping 4

ping 5

ping 6

ping 7

ping 8

ping 9

ping 10

remove 5

remove 2

remove 3

remove 8

remove 7

remove 1

remove 4

remove 6

remove 10

remove 9

**Тест 4**

create 1 -1

create 2 -1

create 3 -1

create 4 -1

create 5 -1

exec 4 aba ababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababa

exec 2 aba ababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababa

exec 5 aba ababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababa

exec 1 aba ababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababa

exec 3 aba ababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababababa

remove 4

remove 1

remove 3

remove 2

remove 5

**Тест 5**

create 1 -1

ping 1

exec 1 a b

remove 1

ping 1

**Основные файлы программы**

Для удобства функции инициализации сокета, получения и отправки сообщения вынесены в отдельный файл zmq\_std.hpp, топология в topology.hpp. В файле control.cpp расположен код для управляющего узла, а в calculation\_node.cpp для вычислительного узла. Функции для поиска подстроки находятся в файлах search.hpp и search.cpp

**zmq\_std.hpp:**

#ifndef ZMQ\_STD\_HPP

#define ZMQ\_STD\_HPP

#include <assert.h>

#include <errno.h>

#include <string.h>

#include <string>

#include <zmq.h>

const char\* NODE\_EXECUTABLE\_NAME = "calculation";

const char SENTINEL = '$';

const int PORT\_BASE = 8000;

const int WAIT\_TIME = 1000;

enum actions\_t {

fail = 0,

success = 1,

create = 2,

destroy = 3,

bind = 4,

ping = 5,

exec = 6

};

struct node\_token\_t {

actions\_t action;

long long parent\_id, id;

};

namespace zmq\_std {

void init\_pair\_socket(void\* & context, void\* & socket) {

int rc;

context = zmq\_ctx\_new();

socket = zmq\_socket(context, ZMQ\_PAIR);

rc = zmq\_setsockopt(socket, ZMQ\_RCVTIMEO, &WAIT\_TIME, sizeof(int));

assert(rc == 0);

rc = zmq\_setsockopt(socket, ZMQ\_SNDTIMEO, &WAIT\_TIME, sizeof(int));

assert(rc == 0);

}

template<class T>

void recieve\_msg(T & reply\_data, void\* socket) {

int rc = 0;

zmq\_msg\_t reply;

zmq\_msg\_init(&reply);

rc = zmq\_msg\_recv(&reply, socket, 0);

assert(rc == sizeof(T));

reply\_data = \*(T\*)zmq\_msg\_data(&reply);

rc = zmq\_msg\_close(&reply);

assert(rc == 0);

}

template<class T>

void send\_msg(T\* token, void\* socket) {

int rc = 0;

zmq\_msg\_t message;

zmq\_msg\_init(&message);

rc = zmq\_msg\_init\_size(&message, sizeof(T));

assert(rc == 0);

rc = zmq\_msg\_init\_data(&message, token, sizeof(T), NULL, NULL);

assert(rc == 0);

rc = zmq\_msg\_send(&message, socket, 0);

assert(rc == sizeof(T));

}

template<class T>

bool send\_msg\_dontwait(T\* token, void\* socket) {

int rc;

zmq\_msg\_t message;

zmq\_msg\_init(&message);

rc = zmq\_msg\_init\_size(&message, sizeof(T));

assert(rc == 0);

rc = zmq\_msg\_init\_data(&message, token, sizeof(T), NULL, NULL);

assert(rc == 0);

rc = zmq\_msg\_send(&message, socket, ZMQ\_DONTWAIT);

if (rc == -1) {

zmq\_msg\_close(&message);

return false;

}

assert(rc == sizeof(T));

return true;

}

template<class T>

bool recieve\_msg\_wait(T & reply\_data, void\* socket) {

int rc = 0;

zmq\_msg\_t reply;

zmq\_msg\_init(&reply);

rc = zmq\_msg\_recv(&reply, socket, 0);

if (rc == -1) {

zmq\_msg\_close(&reply);

return false;

}

assert(rc == sizeof(T));

reply\_data = \*(T\*)zmq\_msg\_data(&reply);

rc = zmq\_msg\_close(&reply);

assert(rc == 0);

return true;

}

/\* Returns true if T was successfully queued on the socket \*/

template<class T>

bool send\_msg\_wait(T\* token, void\* socket) {

int rc;

zmq\_msg\_t message;

zmq\_msg\_init(&message);

rc = zmq\_msg\_init\_size(&message, sizeof(T));

assert(rc == 0);

rc = zmq\_msg\_init\_data(&message, token, sizeof(T), NULL, NULL);

assert(rc == 0);

rc = zmq\_msg\_send(&message, socket, 0);

if (rc == -1) {

zmq\_msg\_close(&message);

return false;

}

assert(rc == sizeof(T));

return true;

}

/\*

\* Returns true if socket successfully queued

\* message and recieved reply

\*/

template<class T>

bool send\_recieve\_wait(T\* token\_send, T & token\_reply, void\* socket) {

if (send\_msg\_wait(token\_send, socket)) {

if (recieve\_msg\_wait(token\_reply, socket)) {

return true;

} else {

return false;

}

} else {

return false;

}

}

}

#endif /\* ZMQ\_STD\_HPP \*/

**topology.hpp:**

#ifndef TOPOLOGY\_HPP

#define TOPOLOGY\_HPP

#include <iostream>

#include <list>

template<class T>

class topology\_t {

private:

using list\_type = std::list< std::list<T> >;

using iterator = typename std::list<T>::iterator;

using list\_iterator = typename list\_type::iterator;

list\_type container;

size\_t container\_size;

public:

explicit topology\_t() noexcept : container(), container\_size(0) {}

~topology\_t() {}

bool erase(const T & elem) {

for (list\_iterator it1 = container.begin(); it1 != container.end(); ++it1) {

for (iterator it2 = it1->begin(); it2 != it1->end(); ++it2) {

if (\*it2 == elem) {

if (it1->size() > 1) {

it1->erase(it2);

} else {

container.erase(it1);

}

--container\_size;

return true;

}

}

}

return false;

}

long long find(const T & elem) {

long long ind = 0;

for (list\_iterator it1 = container.begin(); it1 != container.end(); ++it1) {

for (iterator it2 = it1->begin(); it2 != it1->end(); ++it2) {

if (\*it2 == elem) {

return ind;

}

}

++ind;

}

return -1;

}

bool insert(const T & parent, const T & elem) {

for (list\_iterator it1 = container.begin(); it1 != container.end(); ++it1) {

for (iterator it2 = it1->begin(); it2 != it1->end(); ++it2) {

if (\*it2 == parent) {

it1->insert(++it2, elem);

++container\_size;

return true;

}

}

}

return false;

}

void insert(const T & elem) {

std::list<T> new\_list;

new\_list.push\_back(elem);

++container\_size;

container.push\_back(new\_list);

}

size\_t size() {

return container\_size;

}

template<class U>

friend std::ostream & operator << (std::ostream & of, const topology\_t<U> & top) {

for (auto it1 = top.container.begin(); it1 != top.container.end(); ++it1) {

of << "{";

for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {

of << \*it2 << " ";

}

of << "}" << std::endl;

}

return of;

}

};

#endif /\* TOPOLOGY\_HPP \*/

**control.cpp:**

#include <unistd.h>

#include <vector>

#include "topology.hpp"

#include "zmq\_std.hpp"

using node\_id\_type = long long;

const long long TEXT\_PORT = 100;

int main() {

int rc;

topology\_t<node\_id\_type> control\_node;

std::vector< std::pair<void\*, void\*> > childs;

std::string s;

node\_id\_type id;

while (std::cin >> s >> id) {

if (s == "create") {

node\_id\_type parent\_id;

std::cin >> parent\_id;

if (parent\_id == -1) {

void\* new\_context = NULL;

void\* new\_socket = NULL;

zmq\_std::init\_pair\_socket(new\_context, new\_socket);

rc = zmq\_bind(new\_socket, ("tcp://\*:" + std::to\_string(PORT\_BASE + id)).c\_str());

assert(rc == 0);

int fork\_id = fork();

if (fork\_id == 0) {

rc = execl(NODE\_EXECUTABLE\_NAME, NODE\_EXECUTABLE\_NAME, std::to\_string(id).c\_str(), NULL);

assert(rc != -1);

return 0;

} else {

node\_token\_t\* token = new node\_token\_t({ping, id, id});

node\_token\_t reply({fail, id, id});

if (zmq\_std::send\_recieve\_wait(token, reply, new\_socket) and reply.action == success) {

childs.push\_back(std::make\_pair(new\_context, new\_socket));

control\_node.insert(id);

} else {

rc = zmq\_close(new\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(new\_context);

assert(rc == 0);

}

}

} else if (control\_node.find(parent\_id) == -1) {

std::cout << "Error: Not found" << std::endl;

} else {

if (control\_node.find(id) != -1) {

std::cout << "Error: Already exists" << std::endl;

} else {

int ind = control\_node.find(parent\_id);

node\_token\_t\* token = new node\_token\_t({create, parent\_id, id});

node\_token\_t reply({fail, id, id});

if (zmq\_std::send\_recieve\_wait(token, reply, childs[ind].second) and reply.action == success) {

control\_node.insert(parent\_id, id);

} else {

std::cout << "Error: Parent is unavailable" << std::endl;

}

}

}

} else if (s == "remove") {

int ind = control\_node.find(id);

if (ind != -1) {

node\_token\_t\* token = new node\_token\_t({destroy, id, id});

node\_token\_t reply({fail, id, id});

bool ok = zmq\_std::send\_recieve\_wait(token, reply, childs[ind].second);

if (reply.action == destroy and reply.parent\_id == id) {

rc = zmq\_close(childs[ind].second);

assert(rc == 0);

rc = zmq\_ctx\_term(childs[ind].first);

assert(rc == 0);

std::vector< std::pair<void\*, void\*> >::iterator it = childs.begin();

while (ind--) {

++it;

}

childs.erase(it);

} else if (reply.action == bind and reply.parent\_id == id) {

rc = zmq\_close(childs[ind].second);

assert(rc == 0);

rc = zmq\_ctx\_term(childs[ind].first);

assert(rc == 0);

zmq\_std::init\_pair\_socket(childs[ind].first, childs[ind].second);

rc = zmq\_bind(childs[ind].second, ("tcp://\*:" + std::to\_string(PORT\_BASE + reply.id)).c\_str());

assert(rc == 0);

}

if (ok) {

control\_node.erase(id);

std::cout << "OK" << std::endl;

} else {

std::cout << "Error: Node is unavailable" << std::endl;

}

} else {

std::cout << "Error: Not found" << std::endl;

}

} else if (s == "ping") {

int ind = control\_node.find(id);

if (ind != -1) {

node\_token\_t\* token = new node\_token\_t({ping, id, id});

node\_token\_t reply({fail, id, id});

if (zmq\_std::send\_recieve\_wait(token, reply, childs[ind].second) and reply.action == success) {

std::cout << "OK: 1" << std::endl;

} else {

std::cout << "OK: 0" << std::endl;

}

} else {

std::cout << "Error: Not found" << std::endl;

}

} else if (s == "exec") {

std::string pattern, text;

std::cin >> pattern >> text;

int ind = control\_node.find(id);

if (ind != -1) {

bool ok = true;

std::string text\_pattern = pattern + SENTINEL + text + SENTINEL;

for (size\_t i = 0; i < text\_pattern.size(); ++i) {

node\_token\_t\* token = new node\_token\_t({exec, text\_pattern[i], id});

node\_token\_t reply({fail, id, id});

if (!zmq\_std::send\_recieve\_wait(token, reply, childs[ind].second) or reply.action != success) {

ok = false;

break;

}

}

if (ok) {

std::cout << "OK" << std::endl;

} else {

std::cout << "Error: Node is unavailable" << std::endl;

}

} else {

std::cout << "Error: Not found" << std::endl;

}

}

}

for (size\_t i = 0; i < childs.size(); ++i) {

rc = zmq\_close(childs[i].second);

assert(rc == 0);

rc = zmq\_ctx\_term(childs[i].first);

assert(rc == 0);

}

}

**calculation\_node.cpp:**

#include <iostream>

#include <pthread.h>

#include <queue>

#include <tuple>

#include <unistd.h>

#include "search.hpp"

#include "zmq\_std.hpp"

const std::string SENTINEL\_STR = "$";

long long node\_id;

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

std::queue< std::pair<std::string, std::string> > calc\_queue;

void\* thread\_func(void\*) {

while (1) {

pthread\_mutex\_lock(&mutex);

while (calc\_queue.empty()) {

pthread\_cond\_wait(&cond, &mutex);

}

std::pair<std::string, std::string> cur = calc\_queue.front();

calc\_queue.pop();

pthread\_mutex\_unlock(&mutex);

if (cur.first == SENTINEL\_STR and cur.second == SENTINEL\_STR) {

break;

} else {

std::vector<unsigned int> res = KMPStrong(cur.first, cur.second);

std::cout << "OK: " << node\_id << " : ";

if (res.empty()) {

std::cout << "No matches" << std::endl;

} else {

for (size\_t i = 0; i < res.size() - 1; ++i) {

std::cout << res[i] << ", ";

}

std::cout << res.back() << std::endl;

}

}

}

return NULL;

}

int main(int argc, char\*\* argv) {

int rc;

assert(argc == 2);

node\_id = std::stoll(std::string(argv[1]));

void\* node\_parent\_context = zmq\_ctx\_new();

void\* node\_parent\_socket = zmq\_socket(node\_parent\_context, ZMQ\_PAIR);

rc = zmq\_connect(node\_parent\_socket, ("tcp://localhost:" + std::to\_string(PORT\_BASE + node\_id)).c\_str());

assert(rc == 0);

long long child\_id = -1;

void\* node\_context = NULL;

void\* node\_socket = NULL;

pthread\_t calculation\_thread;

rc = pthread\_mutex\_init(&mutex, NULL);

assert(rc == 0);

rc = pthread\_cond\_init(&cond, NULL);

assert(rc == 0);

rc = pthread\_create(&calculation\_thread, NULL, thread\_func, NULL);

assert(rc == 0);

std::string pattern, text;

bool flag\_sentinel = true;

std::cout << "OK: " << getpid() << std::endl;

bool has\_child = false;

bool awake = true;

while (awake) {

node\_token\_t token;

zmq\_std::recieve\_msg(token, node\_parent\_socket);

node\_token\_t\* reply = new node\_token\_t({fail, node\_id, node\_id});

if (token.action == bind and token.parent\_id == node\_id) {

/\*

\* Bind could be recieved when parent created node

\* and this node should bind to parent's child

\*/

zmq\_std::init\_pair\_socket(node\_context, node\_socket);

rc = zmq\_bind(node\_socket, ("tcp://\*:" + std::to\_string(PORT\_BASE + token.id)).c\_str());

assert(rc == 0);

has\_child = true;

child\_id = token.id;

node\_token\_t\* token\_ping = new node\_token\_t({ping, child\_id, child\_id});

node\_token\_t reply\_ping({fail, child\_id, child\_id});

if (zmq\_std::send\_recieve\_wait(token\_ping, reply\_ping, node\_socket) and reply\_ping.action == success) {

reply->action = success;

}

} else if (token.action == create) {

if (token.parent\_id == node\_id) {

if (has\_child) {

rc = zmq\_close(node\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(node\_context);

assert(rc == 0);

}

zmq\_std::init\_pair\_socket(node\_context, node\_socket);

rc = zmq\_bind(node\_socket, ("tcp://\*:" + std::to\_string(PORT\_BASE + token.id)).c\_str());

assert(rc == 0);

int fork\_id = fork();

if (fork\_id == 0) {

rc = execl(NODE\_EXECUTABLE\_NAME, NODE\_EXECUTABLE\_NAME, std::to\_string(token.id).c\_str(), NULL);

assert(rc != -1);

return 0;

} else {

bool ok = true;

if (has\_child) {

node\_token\_t\* token\_bind = new node\_token\_t({bind, token.id, child\_id});

node\_token\_t reply\_bind({fail, token.id, token.id});

ok = zmq\_std::send\_recieve\_wait(token\_bind, reply\_bind, node\_socket);

ok = ok and (reply\_bind.action == success);

}

if (ok) {

/\* We should check if child has connected to this node \*/

node\_token\_t\* token\_ping = new node\_token\_t({ping, token.id, token.id});

node\_token\_t reply\_ping({fail, token.id, token.id});

ok = zmq\_std::send\_recieve\_wait(token\_ping, reply\_ping, node\_socket);

ok = ok and (reply\_ping.action == success);

if (ok) {

reply->action = success;

child\_id = token.id;

has\_child = true;

} else {

rc = zmq\_close(node\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(node\_context);

assert(rc == 0);

}

}

}

} else if (has\_child) {

node\_token\_t\* token\_down = new node\_token\_t(token);

node\_token\_t reply\_down(token);

reply\_down.action = fail;

if (zmq\_std::send\_recieve\_wait(token\_down, reply\_down, node\_socket) and reply\_down.action == success) {

\*reply = reply\_down;

}

}

} else if (token.action == ping) {

if (token.id == node\_id) {

reply->action = success;

} else if (has\_child) {

node\_token\_t\* token\_down = new node\_token\_t(token);

node\_token\_t reply\_down(token);

reply\_down.action = fail;

if (zmq\_std::send\_recieve\_wait(token\_down, reply\_down, node\_socket) and reply\_down.action == success) {

\*reply = reply\_down;

}

}

} else if (token.action == destroy) {

if (has\_child) {

if (token.id == child\_id) {

bool ok = true;

node\_token\_t\* token\_down = new node\_token\_t({destroy, node\_id, child\_id});

node\_token\_t reply\_down({fail, child\_id, child\_id});

ok = zmq\_std::send\_recieve\_wait(token\_down, reply\_down, node\_socket);

/\* We should get special reply from child \*/

if (reply\_down.action == destroy and reply\_down.parent\_id == child\_id) {

rc = zmq\_close(node\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(node\_context);

assert(rc == 0);

has\_child = false;

child\_id = -1;

} else if (reply\_down.action == bind and reply\_down.parent\_id == node\_id) {

rc = zmq\_close(node\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(node\_context);

assert(rc == 0);

zmq\_std::init\_pair\_socket(node\_context, node\_socket);

rc = zmq\_bind(node\_socket, ("tcp://\*:" + std::to\_string(PORT\_BASE + reply\_down.id)).c\_str());

assert(rc == 0);

child\_id = reply\_down.id;

node\_token\_t\* token\_ping = new node\_token\_t({ping, child\_id, child\_id});

node\_token\_t reply\_ping({fail, child\_id, child\_id});

if (zmq\_std::send\_recieve\_wait(token\_ping, reply\_ping, node\_socket) and reply\_ping.action == success) {

ok = true;

}

}

if (ok) {

reply->action = success;

}

} else if (token.id == node\_id) {

rc = zmq\_close(node\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(node\_context);

assert(rc == 0);

has\_child = false;

reply->action = bind;

reply->id = child\_id;

reply->parent\_id = token.parent\_id;

awake = false;

} else {

node\_token\_t\* token\_down = new node\_token\_t(token);

node\_token\_t reply\_down(token);

reply\_down.action = fail;

if (zmq\_std::send\_recieve\_wait(token\_down, reply\_down, node\_socket) and reply\_down.action == success) {

\*reply = reply\_down;

}

}

} else if (token.id == node\_id) {

/\* Special message to parent \*/

reply->action = destroy;

reply->parent\_id = node\_id;

reply->id = node\_id;

awake = false;

}

} else if (token.action == exec) {

if (token.id == node\_id) {

char c = token.parent\_id;

if (c == SENTINEL) {

if (flag\_sentinel) {

std::swap(text, pattern);

} else {

pthread\_mutex\_lock(&mutex);

if (calc\_queue.empty()) {

pthread\_cond\_signal(&cond);

}

calc\_queue.push({pattern, text});

pthread\_mutex\_unlock(&mutex);

text.clear();

pattern.clear();

}

flag\_sentinel = flag\_sentinel ^ 1;

} else {

text = text + c;

}

reply->action = success;

} else if (has\_child) {

node\_token\_t\* token\_down = new node\_token\_t(token);

node\_token\_t reply\_down(token);

reply\_down.action = fail;

if (zmq\_std::send\_recieve\_wait(token\_down, reply\_down, node\_socket) and reply\_down.action == success) {

\*reply = reply\_down;

}

}

}

zmq\_std::send\_msg\_dontwait(reply, node\_parent\_socket);

}

if (has\_child) {

rc = zmq\_close(node\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(node\_context);

assert(rc == 0);

}

rc = zmq\_close(node\_parent\_socket);

assert(rc == 0);

rc = zmq\_ctx\_term(node\_parent\_context);

assert(rc == 0);

pthread\_mutex\_lock(&mutex);

if (calc\_queue.empty()) {

pthread\_cond\_signal(&cond);

}

calc\_queue.push({SENTINEL\_STR, SENTINEL\_STR});

pthread\_mutex\_unlock(&mutex);

rc = pthread\_join(calculation\_thread, NULL);

assert(rc == 0);

rc = pthread\_cond\_destroy(&cond);

assert(rc == 0);

rc = pthread\_mutex\_destroy(&mutex);

assert(rc == 0);

}

**search.hpp**

#ifndef SEARCH\_HPP

#define SEARCH\_HPP

#include <string>

#include <vector>

std::vector<unsigned int> PrefixFunction(const std::string & s);

std::vector<unsigned int> KMPWeak(const std::string & pattern, const std::string & text);

std::vector<unsigned int> ZFunction(const std::string & s);

std::vector<unsigned int> StrongPrefixFunction(const std::string & s);

std::vector<unsigned int> KMPStrong(const std::string & pattern, const std::string & text);

#endif /\* SEARCH\_HPP \*/

**search.cpp**

#include "search.hpp"

std::vector<unsigned int> PrefixFunction(const std::string & s) {

unsigned int n = s.size();

std::vector<unsigned int> p(n);

for (unsigned int i = 1; i < n; ++i) {

p[i] = p[i - 1];

while (p[i] > 0 and s[i] != s[p[i]]) {

p[i] = p[p[i] - 1];

}

if (s[i] == s[p[i]]) {

++p[i];

}

}

return p;

}

std::vector<unsigned int> KMPWeak(const std::string & pattern, const std::string & text) {

std::vector<unsigned int> p = PrefixFunction(pattern);

unsigned int m = pattern.size();

unsigned int n = text.size();

unsigned int i = 0;

std::vector<unsigned int> ans;

if (m > n) {

return ans;

}

while (i < n - m + 1) {

unsigned int j = 0;

while (j < m and pattern[j] == text[i + j]) {

++j;

}

if (j == m) {

ans.push\_back(i);

} else {

if (j > 0 and j > p[j - 1]) {

i = i + j - p[j - 1] - 1;

}

}

++i;

}

return ans;

}

std::vector<unsigned int> ZFunction(const std::string & s) {

unsigned int n = s.size();

std::vector<unsigned int> z(n);

unsigned int l = 0, r = 0;

for (unsigned int i = 1; i < n; ++i) {

if (i <= r) {

z[i] = std::min(z[i - l], r - i);

}

while (i + z[i] < n and s[i + z[i]] == s[z[i]]) {

++z[i];

}

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

l = i;

r = i + z[i];

}

}

return z;

}

std::vector<unsigned int> StrongPrefixFunction(const std::string & s) {

std::vector<unsigned int> z = ZFunction(s);

unsigned int n = s.size();

std::vector<unsigned int> sp(n);

for (unsigned int i = n - 1; i > 0; --i) {

sp[i + z[i] - 1] = z[i];

}

return sp;

}

std::vector<unsigned int> KMPStrong(const std::string & pattern, const std::string & text) {

std::vector<unsigned int> p = StrongPrefixFunction(pattern);

unsigned int m = pattern.size();

unsigned int n = text.size();

unsigned int i = 0;

std::vector<unsigned int> ans;

if (m > n) {

return ans;

}

while (i < n - m + 1) {

unsigned int j = 0;

while (j < m and pattern[j] == text[i + j]) {

++j;

}

if (j == m) {

ans.push\_back(i);

} else {

if (j > 0 and j > p[j - 1]) {

i = i + j - p[j - 1] - 1;

}

}

++i;

}

return ans;}

**Пример работы**

Для работы программы и успешной компиляции необходимо установить библиотеку zmq.  
Для этого выполнить команду в терминале (для операционной системы linux mint)  
sudo apt-get install libzmq3-dev

daniel@daniel-Ideapad-Z570: ~ cd os/oslab6-8/lab6

daniel@daniel-Ideapad-Z570: ~ lab6 make

g++ -g -O2 -pedantic -pthread -std=c++17 -Wall -Werror -Wextra control.cpp -lzmq -o control

g++ -g -O2 -pedantic -pthread -std=c++17 -Wall -Werror -Wextra -c calculation\_node.cpp -lzmq

g++ -g -O2 -pedantic -pthread -std=c++17 -Wall -Werror -Wextra -c search.cpp

g++ -g -O2 -pedantic -pthread -std=c++17 -Wall -Werror -Wextra calculation\_node.o search.o -lzmq -o calculation

daniel@daniel-Ideapad-Z570: ~ lab6 ls

calculation control search.cpp strace.txt zmq\_std.hpp

calculation\_node.cpp control.cpp search.hpp tests

calculation\_node.o makefile search.o topology.hpp

daniel@daniel-Ideapad-Z570: ~ lab6 ./control

create 1 -1

OK: 3174

ping 1

OK: 1

exec 1 a b

OK: 1 : No matches

exec 1 abra abracadabra

OK: 1 : 0, 7

ping 1

OK: 1

remove 1

OK

ping 1

Error: Not found

^C

(py37) ➜ lab6 ./control

create 1 -1

OK: 3242

create 2 1

OK: 3248

create 4 2

OK: 3256

create 10 7

Error: Not found

create 10 4

OK: 3262

ping 2

OK: 1

ping 10

OK: 1

ping 7

Error: Not found

exec 2 a b

OK: 2 : No matches

exec 2 aba anasdsdklcsadclabadsckl

OK: 2 : 15

remove 2

OK

ping 2

Error: Not found

^C

daniel@daniel-Ideapad-Z570: ~ lab6 ./control

create 1 -1

OK: 6278

create 2 1

OK: 6284

create 5 -1

OK: 6290

create 3 2

OK: 6296

create 6 5

OK: 6303

create 7 5

OK: 6311

create 4 3

OK: 6319

create 8 5

OK: 6325

create 9 8

OK: 6333

create 0 1

OK: 6341

remove 8

OK

remove 4

OK

remove 2

OK

remove 6

OK

remove 0

OK

ping 1

OK: 1

ping 3

OK: 1

ping 5

OK: 1

ping 7

OK: 1

ping 9

OK: 1

ping 0

Error: Not found

ping 2

Error: Not found

ping 4

Error: Not found

remove 5

OK

remove 7

OK

ping 5

Error: Not found

ping 7

Error: Not found

**Вывод**

В ходе выполнения лабораторной работы я изучил основы работы с очередями сообщений ZeroMQ и реализовал программу с использованием этой библиотеки. Для достижения отказоустойчивости я пробовал разные способы связи, больше всего подошёл ZMQ\_PAIR. Самым сложным в работе оказались удаление узла из сети и вставка узла между другими узлами. При таких операциях нужно было переподключать сокеты на вычислительных узлах.

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.

**Список литературы**

1. zmq\_socket(3) — 0MQ Api — ZeroMQ API  
URL: http://api.zeromq.org/2-1:zmq-socket (дата обращения 08.12.2020)

2. zmq\_bind(3) — 0MQ Api — ZeroMQ API   
URL: http://api.zeromq.org/2-1:zmq-bind (дата обращения 09.12.2020)

3. zmq\_connect(3) — 0MQ Api — ZeroMQ API  
URL: http://api.zeromq.org/2-1:zmq-connect (дата обращения 09.12.2020)

4. Sockets and Patterns | ØMQ — The Guide — ZeroMQ Guide   
URL: https://zguide.zeromq.org/docs/chapter2/ (дата обращения 08.12.2020)

5. Socket API — ZeroMQ   
URL: https://zeromq.org/socket-api/ (дата обращения 08.12.2020)

6. zmq\_setsockopt(3) — 0MQ Api — ZeroMQ API   
URL: http://api.zeromq.org/3-2:zmq-setsockopt (дата обращения 09.12.2020)

7. Messages — ZeroMQ   
URL: https://zeromq.org/messages/ (дата обращения 10.12.2020)

8. zmq\_msg\_send(3) — 0MQ Api — ZeroMQ API   
URL: http://api.zeromq.org/3-2:zmq-msg-send (дата обращения 10.12.2020)

9. zmq\_msg\_recv(3) — 0MQ Api — ZeroMQ API   
URL: http://api.zeromq.org/master:zmq-msg-recv (дата обращения 10.12.2020)