Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6 по курсу**

**«Операционные системы»**

Студент: Орусский Вячеслав Русланович

Группа: М8О-206Б-21

Вариант: 20

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

Содержание

[Содержание 2](#_Toc133018104)

[**Репозиторий** 3](#_Toc133018105)

[**Постановка задачи** 3](#_Toc133018106)

[**Цель работы** 3](#_Toc133018107)

[**Задание** 3](#_Toc133018108)

[**Общие сведения о программе** 3](#_Toc133018109)

[**Исходный код** 5](#_Toc133018110)

[Message.h 5](#_Toc133018111)

[Message.cpp 6](#_Toc133018112)

[ZMQTools.h 10](#_Toc133018113)

[ZMQTools.cpp 10](#_Toc133018114)

[Client.cpp 11](#_Toc133018115)

[Server.cpp 14](#_Toc133018116)

[**Демонстрация работы программы** 16](#_Toc133018117)

[**Выводы** 16](#_Toc133018118)

**Репозиторий**

[Ссылка](https://github.com/RoKivals/OperatingSystems/tree/master/lab%23678/src)

**Постановка задачи**

**Цель работы**

* Управление 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 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.

**Вариант 20.**

Топология: общее дерево

Тип проверки: ping id

Команда для вычисления: поиск подстроки в строке

**Общие сведения о программе**

Программы компилируются из файлов main.cpp, node.cpp с использованием заголовочных файлов topology.h и zmq\_f.h с помощью cmake и CMakeFile. Также используется заголовочные файлы: unistd.h, iostream, vector, zmq.hpp, sstream, string, map.

**Общий метод и алгоритм решения**

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

**Исходный код**

Message.h

#ifndef OPERATINGSYSTEMS\_LAB\_678\_SRC\_MESSAGE\_H\_  
#define OPERATINGSYSTEMS\_LAB\_678\_SRC\_MESSAGE\_H\_  
  
#include "zmq.hpp"  
#include <string>  
#include <vector>  
  
*// Тип сообщения  
enum* message\_type\_t {  
 MSG\_PING,  
 MSG\_CREATE,  
 MSG\_REMOVE,  
 MSG\_EXEC,  
 MSG\_EXEC\_ANSWER,  
 MSG\_PING\_ANSWER,  
 MSG\_CREATE\_ANSWER,  
};  
  
*// Направление сообщения (между хостом и сервером)  
enum* message\_direction\_t {  
 DIR\_TO\_CLIENT,  
 DIR\_TO\_SERVER,  
};  
  
*// Лист с информацией о сообщении  
struct* header\_t {  
 message\_direction\_t dir;  
 message\_type\_t type;  
 *int* to\_id\_node;  
};  
  
*// ------------------------------------------------------------//  
// различные структуры ответов на сообщения //  
struct* create\_body {  
 *int* child\_id;  
};  
  
*struct* exec\_body {  
 std::string text;  
 std::string pattern;  
};  
  
*struct* ping\_body\_answer {  
 *int* src\_id;  
};  
  
*struct* exec\_body\_answer {  
 std::vector<*int*> entries;  
};  
  
*struct* create\_body\_answer {  
 *int* pid;  
 std::string error;  
};  
  
*// -------------------------------------------------------//*zmq::message\_t fill\_message\_exec(*int* dest\_id, std::string &text, std::string &pattern);  
zmq::message\_t fill\_message\_exec\_answer(std::vector<*int*> &enrties);  
zmq::message\_t fill\_message\_create(*int* parent\_id, *int* child\_id);  
zmq::message\_t fill\_message\_create\_answer(*int* pid, std::string error);  
zmq::message\_t fill\_message\_remove(*int* dest\_id);  
zmq::message\_t fill\_message\_ping(*int* dest\_id);  
zmq::message\_t fill\_message\_ping\_answer(*int* src\_id);  
  
header\_t \*get\_message\_header(zmq::message\_t &data);  
create\_body get\_message\_create(zmq::message\_t &data);  
create\_body\_answer get\_message\_create\_answer(zmq::message\_t &data);  
exec\_body get\_message\_exec(zmq::message\_t &data);  
exec\_body\_answer get\_message\_exec\_answer(zmq::message\_t &data);  
ping\_body\_answer get\_message\_ping\_answer(zmq::message\_t &data);  
  
#endif *//OPERATINGSYSTEMS\_LAB\_678\_SRC\_MESSAGE\_H\_*

Message.cpp

#include "msg.h"  
#include <cstring>  
  
zmq::message\_t fill\_message\_exec(*int* dest\_id, std::string &text, std::string &pattern) {  
 header\_t header{  
 DIR\_TO\_CLIENT,  
 MSG\_EXEC,  
 dest\_id,  
 };  
  
 *int* size\_of\_message = *sizeof*(header\_t) + text.size() + pattern.size() + 2 \* *sizeof*(*int*);  
 zmq::message\_t data(size\_of\_message);  
 *char* \*dest\_ptr = (*char* \*) data.data();  
  
 *// Копируем header* std::memcpy(dest\_ptr, &header, *sizeof*(header));  
 dest\_ptr += *sizeof*(header);  
  
 *// Копируем размер текста  
 int* text\_size = text.size();  
 std::memcpy(dest\_ptr, &text\_size, *sizeof*(text\_size));  
 dest\_ptr += *sizeof*(text\_size);  
  
 *// копируем сам текст* std::copy(text.begin(), text.end(), dest\_ptr);  
 dest\_ptr += text.size();  
  
 *// Копируем размер паттерна  
 int* pattern\_size = pattern.size();  
 std::memcpy(dest\_ptr, &pattern\_size, *sizeof*(pattern\_size));  
 dest\_ptr += *sizeof*(pattern\_size);  
  
 *// копируем сам паттерн* std::copy(pattern.begin(), pattern.end(), dest\_ptr);  
 dest\_ptr += pattern.size();  
  
 *return* data;  
}  
  
zmq::message\_t fill\_message\_exec\_answer(std::vector<*int*> &enrties) {  
 header\_t header{  
 DIR\_TO\_SERVER,  
 MSG\_EXEC\_ANSWER,  
 -1,  
 };  
  
 *int* size\_of\_message = *sizeof*(header\_t) + *sizeof*(*int*) \* enrties.size() + *sizeof*(*int*);  
 zmq::message\_t data(size\_of\_message);  
 *char* \*dest\_ptr = (*char* \*) data.data();  
  
 *// Копируем header* std::memcpy(dest\_ptr, &header, *sizeof*(header));  
 dest\_ptr += *sizeof*(header);  
  
 *// Копируем размер массива обнаруженных вхождений  
 int* entries\_size = enrties.size();  
 std::memcpy(dest\_ptr, &entries\_size, *sizeof*(entries\_size));  
 dest\_ptr += *sizeof*(entries\_size);  
  
 *// копируем сам массив* std::copy(enrties.begin(), enrties.end(), (*int* \*) dest\_ptr);  
 dest\_ptr += *sizeof*(*int*) \* enrties.size();  
  
 *return* data;  
}  
  
zmq::message\_t fill\_message\_create(*int* parent\_id, *int* child\_id) {  
 header\_t header{  
 DIR\_TO\_CLIENT,  
 MSG\_CREATE,  
 parent\_id  
 };  
  
 create\_body body{  
 child\_id  
 };  
  
 *int* size\_of\_message = *sizeof*(header\_t) + *sizeof*(body);  
 zmq::message\_t data(size\_of\_message);  
 *char* \*dest\_ptr = (*char* \*) data.data();  
  
 *// Копируем header* std::memcpy(dest\_ptr, &header, *sizeof*(header));  
 dest\_ptr += *sizeof*(header);  
  
 *// Копируем body* std::memcpy(dest\_ptr, &body, *sizeof*(body));  
 dest\_ptr += *sizeof*(body);  
  
 *return* data;  
}  
  
zmq::message\_t fill\_message\_create\_answer(*int* pid, std::string error) {  
 header\_t header{  
 DIR\_TO\_SERVER,  
 MSG\_CREATE\_ANSWER,  
 -1  
 };  
  
 *int* size\_of\_message = *sizeof*(header\_t) + *sizeof*(pid) + *sizeof*(*int*) + error.size() \* *sizeof*(error);  
 zmq::message\_t data(size\_of\_message);  
 *char* \*dest\_ptr = (*char* \*) data.data();  
  
 *// Копируем header* std::memcpy(dest\_ptr, &header, *sizeof*(header));  
 dest\_ptr += *sizeof*(header);  
  
 *// Копируем pid* std::memcpy(dest\_ptr, &pid, *sizeof*(pid));  
 dest\_ptr += *sizeof*(pid);  
  
 *// Копируем error.size  
 int* error\_size = error.size();  
 std::memcpy(dest\_ptr, &error\_size, *sizeof*(error\_size));  
 dest\_ptr += *sizeof*(error\_size);  
  
 *// Копируем error* std::copy(error.begin(), error.end(), dest\_ptr);  
 dest\_ptr += error.size();  
  
 *return* data;  
}  
  
zmq::message\_t fill\_message\_remove(*int* dest\_id) {  
 header\_t header{  
 DIR\_TO\_CLIENT,  
 MSG\_REMOVE,  
 dest\_id  
 };  
  
 *int* size\_of\_message = *sizeof*(header\_t);  
 zmq::message\_t data(size\_of\_message);  
 *char* \*dest\_ptr = (*char* \*) data.data();  
  
 std::memcpy(dest\_ptr, &header, *sizeof*(header));  
 dest\_ptr += *sizeof*(header);  
  
 *return* data;  
}  
  
zmq::message\_t fill\_message\_ping(*int* dest\_id) {  
 header\_t header{  
 DIR\_TO\_CLIENT,  
 MSG\_PING,  
 dest\_id  
 };  
  
 *int* size\_of\_message = *sizeof*(header\_t);  
 zmq::message\_t data(size\_of\_message);  
 *char* \*dest\_ptr = (*char* \*) data.data();  
  
 std::memcpy(dest\_ptr, &header, *sizeof*(header));  
 dest\_ptr += *sizeof*(header);  
  
 *return* data;  
}  
  
zmq::message\_t fill\_message\_ping\_answer(*int* src\_id) {  
 header\_t header{  
 DIR\_TO\_SERVER,  
 MSG\_PING\_ANSWER,  
 -1  
 };  
  
 ping\_body\_answer body{  
 src\_id  
 };  
  
 *int* size\_of\_message = *sizeof*(header\_t) + *sizeof*(body);  
 zmq::message\_t data(size\_of\_message);  
 *char* \*dest\_ptr = (*char* \*) data.data();  
  
 std::memcpy(dest\_ptr, &header, *sizeof*(header));  
 dest\_ptr += *sizeof*(header);  
  
 std::memcpy(dest\_ptr, &body, *sizeof*(body));  
 dest\_ptr += *sizeof*(body);  
  
 *return* data;  
}  
  
header\_t \*get\_message\_header(zmq::message\_t &data) {  
 *return* (header\_t \*) data.data();  
}  
  
create\_body get\_message\_create(zmq::message\_t &data) {  
 create\_body body;  
 std::memcpy(&body, (*char* \*) data.data() + *sizeof*(header\_t), *sizeof*(body));  
 *return* body;  
}  
  
create\_body\_answer get\_message\_create\_answer(zmq::message\_t &data) {  
 *char* \*src\_ptr = (*char* \*) data.data() + *sizeof*(header\_t);  
  
 *// Восстанавливаем pid  
 int* pid;  
 std::memcpy(&pid, src\_ptr, *sizeof*(*int*));  
 src\_ptr += *sizeof*(pid);  
  
 *// Восстанавливаем error size  
 int* error\_size;  
 std::memcpy(&error\_size, src\_ptr, *sizeof*(*int*));  
 src\_ptr += *sizeof*(error\_size);  
  
 *// Восстанавливаем error* std::string error(src\_ptr, error\_size);  
 src\_ptr += error\_size;  
  
 *return* {pid, error};  
}  
  
exec\_body get\_message\_exec(zmq::message\_t &data) {  
 *char* \*src\_ptr = (*char* \*) data.data() + *sizeof*(header\_t);  
  
 *int* text\_size;  
 std::memcpy(&text\_size, src\_ptr, *sizeof*(*int*));  
 src\_ptr += *sizeof*(*int*);  
  
 std::string text(src\_ptr, text\_size);  
 src\_ptr += text\_size;  
  
 *int* pattern\_size;  
 std::memcpy(&pattern\_size, src\_ptr, *sizeof*(*int*));  
 src\_ptr += *sizeof*(*int*);  
  
 std::string pattern(src\_ptr, pattern\_size);  
 src\_ptr += pattern\_size;  
  
 *return* {text, pattern};  
}  
  
exec\_body\_answer get\_message\_exec\_answer(zmq::message\_t &data) {  
 *char* \*src\_ptr = (*char* \*) data.data() + *sizeof*(header\_t);  
  
 *int* entries\_size;  
 std::memcpy(&entries\_size, src\_ptr, *sizeof*(*int*));  
 src\_ptr += *sizeof*(*int*);  
  
 std::vector<*int*> entries;  
 *int* \*src\_int\_ptr = (*int* \*) src\_ptr;  
 std::copy(src\_int\_ptr, src\_int\_ptr + entries\_size, std::back\_inserter(entries));  
  
 *return* {entries};  
}  
  
ping\_body\_answer get\_message\_ping\_answer(zmq::message\_t &data) {  
 ping\_body\_answer body;  
 std::memcpy(&body, (*char* \*) data.data() + *sizeof*(header\_t), *sizeof*(body));  
 *return* body;  
}

ZMQTools.h

#ifndef OPERATINGSYSTEMS\_LAB\_678\_SRC\_ZMQTOOLS\_H\_  
#define OPERATINGSYSTEMS\_LAB\_678\_SRC\_ZMQTOOLS\_H\_  
#include "zmq.hpp"  
#include <string>  
  
*extern const int* BROADCAST\_ID;  
  
std::string create\_name\_of\_socket(*int* node\_id);  
std::string create\_name\_of\_socket\_to\_children(*int* node\_id);  
std::string create\_name\_of\_socket\_to\_parent(*int* node\_id);  
  
#endif *//OPERATINGSYSTEMS\_LAB\_678\_SRC\_ZMQTOOLS\_H\_*

ZMQTools.cpp

#include "ZMQTools.h"  
  
*const int* BROADCAST\_ID = -2;  
  
std::string create\_name\_of\_socket(*int* node\_id) {  
 std::stringstream sstream;  
 sstream << "ipc://sock\_" << node\_id;  
 *return* sstream.str();  
}  
  
std::string create\_name\_of\_socket\_to\_children(*int* node\_id) {  
 std::stringstream sstream;  
 sstream << create\_name\_of\_socket(node\_id) << "\_to\_children";  
 *return* sstream.str();  
}  
  
std::string create\_name\_of\_socket\_to\_parent(*int* node\_id) {  
 std::stringstream sstream;  
 sstream << create\_name\_of\_socket(node\_id) << "\_to\_parent";  
 *return* sstream.str();  
}

Client.cpp

*// Управляющий узел*#include <bits/stdc++.h>  
#include "zmq.hpp"  
#include <unistd.h>  
#include <csignal>  
#include "msg.h"  
#include "ZMQTools.h"  
  
*static* zmq::context\_t context;  
  
*static* zmq::socket\_t pub\_sock(context, ZMQ\_PUB);  
*static* zmq::socket\_t sub\_sock(context, ZMQ\_SUB);  
  
*static* std::string socket\_name;  
*static* std::vector<std::string> sockets\_of\_children;  
  
*void* server\_init() {  
 *try* {  
 socket\_name = create\_name\_of\_socket(0);  
 pub\_sock.bind(socket\_name);  
  
 *// setsockopt - изменить свойства сокета. Обязательным для SUB явлется сво-во ZMQ\_SUBSCRIBE  
 // Который определяет префикс, с которого должно начинаться сбщ,  
 // чтобы данный подписчик посчитал нужным его принять* sub\_sock.setsockopt(ZMQ\_SUBSCRIBE, 0, 0);  
 *int* timeout = 1000;  
 *// Максимальное время ожидания приёма - 1000 мс* sub\_sock.setsockopt(ZMQ\_RCVTIMEO, timeout);  
 }  
 *catch* (zmq::error\_t) {  
 std::cout << "Error:ZMQ: " << zmq\_strerror(errno) << std::endl;  
 exit(-1);  
 }  
  
}  
  
*bool* receive\_msg(zmq::message\_t &msg) {  
 zmq::recv\_result\_t res;  
 res = sub\_sock.recv(msg);  
  
 *return* res.has\_value();  
}  
  
*bool* is\_exists(*int* id) {  
 zmq::message\_t data = fill\_message\_ping(id);  
 pub\_sock.send(data);  
  
 zmq::message\_t receive\_data;  
 header\_t \*header;  
  
 *if* (!receive\_msg(receive\_data))  
 *return false*;  
  
 ping\_body\_answer ans = get\_message\_ping\_answer(receive\_data);  
 *if* (ans.src\_id != id)  
 *return false*;  
  
 *return true*;  
}  
  
*void* create() {  
 *int* new\_node\_pid = -1;  
 *int* parent\_id, child\_id;  
 std::cin >> child\_id >> parent\_id;  
  
 *if* (parent\_id != -1 && !is\_exists(parent\_id)) {  
 std::cout << "Error: Parent not found" << std::endl;  
 *return*;  
 }  
 *if* (is\_exists(child\_id)) {  
 std::cout << "Error: Already exists" << std::endl;  
 *return*;  
 }  
  
 *if* (parent\_id == -1) { *// Управляющий узел создаёт дочерний процесс  
 int* fork\_pid = fork();  
 *if* (fork\_pid == -1) {  
 std::cout << "Error:fork: " << strerror(errno) << std::endl;  
 *return*;  
 }  
 *if* (fork\_pid == 0) {  
 std::stringstream sstream;  
 sstream << child\_id;  
 execl("server", "server", sstream.str().c\_str(), socket\_name.c\_str(), NULL);  
 }  
 *// Подписываемся на дочерний узел* std::string parent\_pub\_socket\_name = create\_name\_of\_socket\_to\_parent(child\_id);  
 sockets\_of\_children.push\_back(parent\_pub\_socket\_name);  
 sub\_sock.connect(parent\_pub\_socket\_name);  
  
 new\_node\_pid = fork\_pid;  
 } *else* {  
 *// иначе отсылкаем запрос всем детям(вниз по дереву)* zmq::message\_t data = fill\_message\_create(parent\_id, child\_id);  
 pub\_sock.send(data);  
  
 zmq::message\_t receive\_data;  
 receive\_msg(receive\_data);  
 create\_body\_answer ans = get\_message\_create\_answer(receive\_data);  
  
 *if* (ans.pid == -1) {  
 std::cout << "Error:remote\_create: " << ans.error << std::endl;  
 *return*;  
 }  
  
 new\_node\_pid = ans.pid;  
 }  
  
 std::cout << "OK: " << new\_node\_pid << std::endl;  
}  
  
*void* exec() {  
 */\* > exec id  
 > text\_string  
 > pattern\_string  
 [result] – номера позиций, где найден образец, разделенный точкой с запятой  
 \*/* std::string text, pattern;  
 *int* dest\_id;  
  
 std::cin >> dest\_id;  
 std::cin.get();  
 std::getline(std::cin, text);  
 std::getline(std::cin, pattern);  
  
 *if* (!is\_exists(dest\_id)) {  
 std::cout << "Error:" << dest\_id << ": Not found" << std::endl;  
 *return*;  
 }  
  
 zmq::message\_t data = fill\_message\_exec(dest\_id, text, pattern);  
 pub\_sock.send(data);  
  
 zmq::message\_t receive\_data;  
 header\_t \*header;  
 receive\_msg(receive\_data);  
 exec\_body\_answer ans = get\_message\_exec\_answer(receive\_data);  
  
 std::cout << "OK:" << dest\_id << ": [";  
 *if* (ans.entries.size() > 0)  
 std::cout << ans.entries[0];  
 *for* (*int* i = 1; i < ans.entries.size(); ++i) {  
 std::cout << "; " << ans.entries[i];  
 }  
 std::cout << "]" << std::endl;  
}  
  
*void* ping() {  
 *int* dest\_id;  
 std::cin >> dest\_id;  
 *if* (is\_exists(dest\_id))  
 std::cout << "OK: 1" << std::endl;  
 *else* std::cout << "Error: Not found" << std::endl;  
}  
  
*void* server\_run() {  
 std::string input\_cmd;  
 *while* (std::cin >> input\_cmd) {  
 *try* {  
 *if* (input\_cmd == "create")  
 create();  
 *else if* (input\_cmd == "exec")  
 exec();  
 *else if* (input\_cmd == "ping")  
 ping();  
 *else* {  
 std::cout << "Error: данной команды нет!" << std::endl;  
 }  
 }  
 *catch* (zmq::error\_t) {  
 std::cout << "Error:ZMQ: " << zmq\_strerror(errno) << std::endl;  
 }  
 }  
}  
  
*int* main() {  
 server\_init();  
 server\_run();  
}

Server.cpp

*// Вычислительный узел*#include <iostream>  
#include <string>  
#include "zmq.hpp"  
#include <unistd.h>  
#include <csignal>  
#include "msg.h"  
#include "ZMQTools.h"  
  
*static* zmq::context\_t context;  
*static* zmq::socket\_t pub\_sock\_to\_children(context, ZMQ\_PUB);  
*static* zmq::socket\_t pub\_sock\_to\_parent(context, ZMQ\_PUB);  
*static* zmq::socket\_t sub\_sock(context, ZMQ\_SUB);  
  
*static* std::string pub\_socket\_to\_children\_name;  
*static* std::string pub\_socket\_to\_server\_name;  
  
*static* std::string parent\_socket\_name;  
*static* std::vector<std::string> children\_sockets\_name;  
  
*static int* node\_id;  
*static bool* run;  
  
*// node\_id, parent\_socket\_name  
void* client\_init(*int* argc, *char* \*argv[]) {  
 std::stringstream sstream;  
 sstream << argv[1];  
 sstream >> node\_id;  
  
 parent\_socket\_name = argv[2];  
  
 pub\_socket\_to\_children\_name = create\_name\_of\_socket\_to\_children(node\_id);  
 pub\_socket\_to\_server\_name = create\_name\_of\_socket\_to\_parent(node\_id);  
  
 pub\_sock\_to\_children.bind(pub\_socket\_to\_children\_name);  
 pub\_sock\_to\_parent.bind(pub\_socket\_to\_server\_name);  
  
 sub\_sock.connect(parent\_socket\_name);  
 sub\_sock.setsockopt(ZMQ\_SUBSCRIBE, 0, 0);  
  
 run = *true*;  
}  
  
*void* create(zmq::message\_t &data) {  
 create\_body body = get\_message\_create(data);  
 *int* child\_id = body.child\_id;  
 *int* fork\_pid = fork();  
 *if* (fork\_pid == -1) {  
 zmq::message\_t ans = fill\_message\_create\_answer(-1, strerror(errno));  
 pub\_sock\_to\_parent.send(ans);  
 *return*;  
 }  
 *if* (fork\_pid == 0) {  
 std::stringstream sstream;  
 sstream << child\_id;  
 execl("server", "server", sstream.str().c\_str(), pub\_socket\_to\_children\_name.c\_str(), NULL);  
 }  
  
 std::string parent\_pub\_socket\_name = create\_name\_of\_socket\_to\_parent(child\_id);  
 children\_sockets\_name.push\_back(parent\_pub\_socket\_name);  
 sub\_sock.connect(parent\_pub\_socket\_name);  
  
 zmq::message\_t ans = fill\_message\_create\_answer(fork\_pid, "");  
 pub\_sock\_to\_parent.send(ans);  
}  
  
*void* exec(zmq::message\_t &data) {  
 exec\_body body = get\_message\_exec(data);  
 std::vector<*int*> entries;  
  
 std::string::size\_type pos = 0;  
 *while* (std::string::npos != (pos = body.text.find(body.pattern, pos))) {  
 entries.push\_back(pos);  
 ++pos;  
 }  
  
 zmq::message\_t ans = fill\_message\_exec\_answer(entries);  
 pub\_sock\_to\_parent.send(ans);  
}  
  
*void* ping(zmq::message\_t &data) {  
 zmq::message\_t ans = fill\_message\_ping\_answer(node\_id);  
 pub\_sock\_to\_parent.send(ans);  
}  
  
*void* task(zmq::message\_t &data) {  
 header\_t \*header = get\_message\_header(data);  
 *if* (header->type == MSG\_CREATE)  
 create(data);  
 *else if* (header->type == MSG\_EXEC)  
 exec(data);  
 *else if* (header->type == MSG\_PING)  
 ping(data);  
}  
  
*void* client\_run() {  
 *while* (run) {  
 *try* {  
 zmq::message\_t data;  
 sub\_sock.recv(data);  
 header\_t \*header = get\_message\_header(data);  
 *// Если сообщение адресовано нам  
 if* (header->to\_id\_node == node\_id || header->to\_id\_node == BROADCAST\_ID)  
 task(data);  
 *// Иначе пропускаем его дальше  
 if* (header->to\_id\_node != node\_id) {  
 *if* (header->dir == DIR\_TO\_SERVER)  
 pub\_sock\_to\_parent.send(data);  
 *else* pub\_sock\_to\_children.send(data);  
 }  
 }  
 *catch* (zmq::error\_t) {  
 std::cout << zmq\_strerror(errno) << std::endl;  
 }  
  
 }  
}  
  
*int* main(*int* argc, *char* \*argv[]) {  
  
 client\_init(argc, argv);  
 client\_run();  
 *return* 0;  
}

**Демонстрация работы программы**

Для запуска нужно установить библиотеку ZeroMQ. Установка и запуск программы:

apt-get install libzmq3-dev

g++ server.cpp -lzmq -o server message.cpp ZMQTools.cpp

g++ client.cpp -lzmq -o client message.cpp ZMQTools.cpp

./client

**Выводы**

Проделав данную работу, я научился принципам управлении серверами сообщений, примененяя отложенные вычисления, интеграции программных систем друг с другом. В частности, познакомился с библиотекой ZeroMQ.