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

Курсовая работа по курсу
«Операционные системы»

Распределённое хранилище

Студент: Артемьев Дмитрий Иванович
Группа: М8О-206Б-18
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2019

Цель курсового проекта

1. Приобретение практических навыков в использовании знаний, полученных в течении курса

2. Проведение исследования в выбранной предметной области

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант курсового проекта: Создание клиента и сервера для распределенного хранилища данных. Обеспечивать синхронизацию между хранилищами при десинхронизации.

Сведения о программе

Программа состоит из следующих файлов:

1. src/main.cpp - реализация класса клиента, пример его использования, однопоточный

2. src/router.cpp - реализация класса сервера, отвечающий за взаимодействие клиента и хранилищ, многопоточный

3. src/storage.cpp - реализация класса хранилища с возможностью подключаться к серверу и отключаться от него, многопоточная

4. src/functions.hpp - определение и реализация некоторых часто используемых обёрток сокетов zmq

Идеи программы

Так как я нигде не нашёл сколько-нибудь полезную информацию по данной задаче, в программе решил использовать наиболее простой и целесообразный в рамках задания подход для синхронизации хранилищ - копирование хранилища, содержащего последнюю версию в более старые хранилища.

Конечно, при этом получится просадка по времени в случае хранилищ больших размеров и небольшой разнице в версиях, однако другой подход, заключающийся в том, чтобы досылать только недостающие версии, требует хранения истории версий, которая увеличит размер хранилища в 2-3 раза.

Клиент:

Пользователи используют клиент, позволяющий работать с удалёнными хранилищами, использовать хранилища в целом также, как std::map<std::string, int>. Клиенту не

нужно знать устройство самого сервера, поскольку взаимодействие между ними происходит через использование обозначенных запросов. Мысленно можно представить, что клиент запускается на слабом компьютере с малым количеством памяти.

Клиент позволяет: записать пару ключ-значение, удалить пару по ключу, вывети значение по ключу, посмотреть текущую версию доступных хранилищ.

Сервер:

Сервер обеспечивает связь между клиентом и хранилищами (которые можно подключать и отключать от сервера). При присоединении нового хранилища, сервер подаёт ему команду на синхронизацию с самым свежим хранилищем. Данные с этого свежего хранилища отправляются на новое хранилище, таким образом, новое хранилище становится копией самого свежего. В случае потери связи со всеми хранилищами, на запросы клиента сервер будет отвечать ошибками. Если имеющиеся хранилища содержат не все данные, что должны (например, все хранилища были отсоединены, а потом добавилось новое), то на запросы с клиента также будут приходить сообщения об ошибках. С клиента можно отслеживать, содержит ли текущая версия хранилищ все данные, так как сервер хранит номер последней версии хранилищ.

Сервер позволяет: получать запросы с клиента, перенаправлять их на хранилища, отвечать клиенту в зависимости от полученной с хранилищ информации, синхронизировать между собой хранилища при совершении действий над ними (таким образом при бездействии процессор не будет нагружаться).

Хранилище:

Хранилище содержит std::map<std::string, int> и реализует взаимодействие с сервером, отвечая на его запросы к это структуре. Если обнаруживается, что хранилище содержит не все данные, то оно отправит на сервер запрос о синхронизации с самым свежим хранилищем. Хранилище никаким образом не зависит от клиента (общается только с сервером).

Для того, чтобы установить соединение с сервером, необходимо ввести порт сервера, а также идентификационный ключ (в простом случае - любой набор букв, цифр). Есть также возможность отключаться от сервера, с сохранением всех накопленных данных, а затем снова подключаться к нему.

Хранилище позволяет: отключаться/подключаться к серверу, сохранять данные по запросам с сервера в структуру std::map, синхронизироваться с другими хранилищами через сервер.

Устройство клиента:

В клиенте используется простой класс StorageAPI, осуществляющий взаимодействие с сервером. Для пользователя это удобный интерфейс, схожий с тем, который предоставляет стандартная библиотека C++ при использовании std::map.

Сокеты:

1. push_socket - сокет для отправки запросов на сервер (router)
2. pull_socket - сокет для получение ответов с сервера (router)

Потоки:
Нет необходимости использовать несколько потоков для реализации клиента.

Устройство сервера (router):

Сокеты:

1. connector - сокет для обработки соединений с хранилищами
2. frontend_push - сокет для отправки ответов на клиент
3. frontend_pull - сокет для получения запросов с клиента
4. up_update - сокет для получения запросов связанных с синхронизацией от хранилищ
5. down_update - мапа сокетов для отправки запросов связанных с синхронизацикй на конкретные хранилища
6. backend_push - мапа сокетов для отправки запросов конкретным хранилищам
7. backend_pull - мапа сокетов для получения ответов с конкретных хранилищ

Потоки:

1. synchronizer - поток, отвечающий за синхронизацию хранилищ между собой, использует сокеты up_update и down_update
2. connector - поток, отслуживающий соединения/отсоедининия хранилищ от сервера, использует сокет connector
3. handler - поток, обрабатывающий запросы, приходящие от клиента, посылающий запросы на хранилища и принимающий ответы с них, использует сокеты frontend_push, frontend_pull, backend_push, backend_pull.

Устройство хранилища (storage):

Сокеты:

1. connector - сокет для установки соединений с сервером, отключения от сервера
2. store_pull - сокет для получения запросов с сервера
3. store_push - сокет для отправки ответов на сервер
4. update_pull - сокет для получения ответов/запросов с сервера, связанных с синхронизацией хранилищ

5. update_push - сокет для отправки запросов/ответов на сервер, связанных с синхронизацией хранилищ

Потоки:

1. synchronizer - поток, отвечающий за синхронизацию хранилищ между собой, использует сокеты update_pull, update_push
2. user - поток, совершающий за взаимодействие с пользователем на подключение к серверу / отключение от сервера, использует сокет connector
3. handler - поток, обрабатывающий запросы, приходящие с сервера, посылающий ответ на сервер, использует сокеты store_pull, store_push.

Исходный код

main.cpp

```
#include <iostream>
#include <string>

#include <zmq.hpp>
#include "functions.hpp"

struct StorageAPI {
    StorageAPI(int port) :
        context(1),
        push_socket(context, ZMQ_PUSH),
        pull_socket(context, ZMQ_PULL)
    {
        push_socket.connect(get_port_name(port));
        pull_socket.connect(get_port_name(port+1));
    }

    void add(std::pair<std::string, int> pr) {
        std::string msg = "add " + pr.first + " " + std::to_string(pr.second);
        if (!send_message(push_socket, msg)) {
            std::cout << "Error sending for adding" << std::endl;
            return;
        }
        std::string ans = recieve_message(pull_socket);
        std::cout << ans << std::endl;
    }
```

```
    void remove(std::pair<std::string, int> pr) {
        std::string msg = "remove " + pr.first;
        if (!send_message(push_socket, msg)) {
            std::cout << "Error sending for removing" << std::endl;
            return;
        }
        std::string ans = recieve_message(pull_socket);
        std::cout << ans << std::endl;
    }

    void get(std::pair<std::string, int> pr) {
        std::string msg = "get " + pr.first;
        if (!send_message(push_socket, msg)) {
            std::cout << "Error sending for getting" << std::endl;
            return;
        }
        std::string ans = recieve_message(pull_socket);
        std::cout << ans << std::endl;
    }

    void version() {
        std::string msg = "version";
        if (!send_message(push_socket, msg)) {
            std::cout << "Error sending for version" << std::endl;
            return;
        }
        std::cout << "recieving" << std::endl;
        std::string ans = recieve_message(pull_socket);
        std::cout << "RECIEVED " ;
        std::cout << ans << std::endl;
    }

private:
    zmq::context_t context;
    zmq::socket_t push_socket;
    zmq::socket_t pull_socket;
};

int main() {
    int port;
    std::cout << "Enter port to connect: ";
    std::cin >> port;
```

```
    StorageAPI store(port);

    while (true) {
        std::cout << "-> ";
        std::string command;
        std::cin >> command;
        if (command == "add") {
            std::pair<std::string, int> pr;
            std::cin >> pr.first >> pr.second;
            store.add(pr);
        }
        else if (command == "remove") {
            std::pair<std::string, int> pr;
            std::cin >> pr.first;
            store.remove(pr);
        }
        else if (command == "get") {
            std::pair<std::string, int> pr;
            std::cin >> pr.first;
            store.get(pr);
        }
        else if (command == "version") {
            store.version();
        }
        else {
            std::cout << "Wrong command" << std::endl;
        }
    }
    return 0;
}
```

router.cpp

```cpp
#include <iostream>
#include <string>
#include <map>
#include <thread>
#include <zmq.hpp>

#include "functions.hpp"

class Router {
public:
    Router(int port);

    void Connector();
    void Handler();
    void Synchronizer();

    void add(const std::string& key, const int& value, std::string request);
    void remove(const std::string& key, std::string request);
    void get(const std::string& key, std::string request);
    std::pair<bool, std::string> version(const std::string& request);

private:
    void InitializeStorages(int count);

private:
    int my_port;
    zmq::context_t context;
    zmq::socket_t connector;

    zmq::socket_t frontend_push;
    zmq::socket_t frontend_pull;

    zmq::socket_t up_update;
    std::map<std::string, zmq::socket_t> down_update;
    std::map<std::string, zmq::socket_t> backend_push;
    std::map<std::string, zmq::socket_t> backend_pull;
    std::map<std::string, int> storages_versions;

    int my_version;
    std::vector<std::pair<zmq::socket_t, int>> storages;
};
```

```
Router::Router(int port) :
    my_port(port),
    context(1),
    connector(context, ZMQ_REP),
    frontend_push(context, ZMQ_PUSH),
    frontend_pull(context, ZMQ_PULL),
    up_update(context, ZMQ_PULL), // приходят сообщения
    my_version(0)
{
    connector.bind(get_port_name(my_port+2));
    frontend_pull.bind(get_port_name(my_port));
    frontend_push.bind(get_port_name(my_port+1));
    up_update.bind(get_port_name(my_port+3));
}

void Router::Connector() {
    while (true) {
        std::istringstream is(recieve_message(connector));
        std::string command;
        is >> command;
        if (command == "connect"){
            std::string ask;
            is >> ask;
            //std::cout << "WHO asked " << ask << std::endl;
            backend_push[ask] = zmq::socket_t(context, ZMQ_PUSH);
            backend_pull[ask] = zmq::socket_t(context, ZMQ_PULL);
            down_update[ask] = zmq::socket_t(context, ZMQ_PUSH);
            storages_versions[ask] = 0;
            int push_port = bind_socket(backend_push[ask]);
            int pull_port = bind_socket(backend_pull[ask]);
            int down_port = bind_socket(down_update[ask]);
            int up_port = my_port + 3;

            std::string init = ask + " init " +
                std::to_string(push_port) + " " +
                std::to_string(pull_port) + " " +
                std::to_string(down_port) + " " +
                std::to_string(up_port) + " ";
            send_message(connector, init);
            std::cout << "Permit connection" << std::endl;
            version("version");
```

```
        }
        else if (command == "disconnect") {
            std::string ask;
            is >> ask;
            send_message(connector, "disconnected");
            backend_push.erase(ask);
            backend_pull.erase(ask);
            down_update.erase(ask);
        }
    }
}

void Router::Handler() {
    while (true) {
        //send_message(backend_push, "FUCK YOU");
        //std::cout << recieve_message(backend_router) << std::endl;
        std::string request = recieve_message(frontend_pull);
        if (backend_push.empty()) {
            send_message(frontend_push, "There is no storages");
            continue;
        }

        std::istringstream is(request);
        std::string command;
        is >> command;
        if (command == "add") {
            std::pair<std::string, int> pr;
            is >> pr.first >> pr.second;
            add(pr.first, pr.second, request);
        }
        else if (command == "remove") {
            std::pair<std::string, int> pr;
            is >> pr.first;
            remove(pr.first, request);
        }
        else if (command == "get") {
            std::pair<std::string, int> pr;
            is >> pr.first;
            get(pr.first, request);
        }
        else if (command == "version") {
            std::pair<bool, std::string> vers = version(request);;
```

```
            if (vers.first)
                send_message(frontend_push, "Newest version, " + vers.second);
            else
                send_message(frontend_push, "Old version, " + vers.second);
        }
    }
}

void Router::Synchronizer() {
    while (true) {
        std::string request = recieve_message(up_update);
        //std::cout << "RECIEVED " << request << std::endl;
        std::istringstream is(request);
        std::string number;
        is >> number;
        std::string command;
        is >> command;
        if (command == "get_upd") {
            //std::cout << "get_upd" << std::endl;
            std::string key;
            int version = 0;
            for (auto& versions : storages_versions) {
                if (versions.second > version) {
                    key = versions.first;
                    version = versions.second;
                }
            }
            std::string ask_for_update = number + " upload";
            //std::cout << "will be sent to " << key << " " << ask_for_update << std::en
            send_message(down_update[key], ask_for_update);
        }
        else if (command == "updated") {
            int version;
            std::cin >> version;
            storages_versions[number] = version;
        }
        else if (command == "upload") {
            std::string reply = number + " download ";
            while (is) {
                std::string str;
                is >> str;
                reply += str + " ";
```

```
            }
            send_message(down_update[number], reply);
        }
    }
}


void Router::add(const std::string& key, const int& value, std::string request) {
    //std::cout << "add" << std::endl;
    std::string msg = std::to_string(my_version) + " " + request;

    for (auto& store : backend_push) {
        send_message(store.second, msg);
    }

    bool added = false;
    for (auto& store : backend_pull) {
        std::string ans = recieve_message(store.second);
        std::istringstream is(ans);
        char c = is.peek();
        if (c > '9' || c < '0')
            continue;
        int version;
        is >> version;
        storages_versions[store.first] = version;
        if (version > my_version) {
            added = true;
            my_version = version;
        }
    }

    if (added)
        send_message(frontend_push, "Ok.");
    else
        send_message(frontend_push, "Error: adding");
}

void Router::remove(const std::string& key, std::string request) {
    //std::cout << "remove" << std::endl;
    std::string msg = std::to_string(my_version) + " " + request;

    for (auto& store : backend_push) {
```

```
        send_message(store.second, msg);
    }

    bool removed = false;
    for (auto& store : backend_pull) {
        std::string ans = recieve_message(store.second);
        std::istringstream is(ans);
        char c = is.peek();
        if (c > '9' || c < '0')
            continue;
        int version;
        is >> version;
        storages_versions[store.first] = version;
        if (version > my_version) {
            removed = true;
            my_version = version;
        }
    }

    if (removed)
        send_message(frontend_push, "Ok.");
    else
        send_message(frontend_push, "Error: removing");
}

void Router::get(const std::string& key, std::string request) {
    //std::cout << "get" << std::endl;
    std::string msg = std::to_string(my_version) + " " + request;

    for (auto& store : backend_push) {
        //std::cout << store.first << std::endl;
        send_message(store.second, msg);
    }

    bool got = false;
    std::string answer;
    for (auto& store : backend_pull) {
        std::string ans = recieve_message(store.second);
        std::istringstream is(ans);
        char c = is.peek();
        if (c > '9' || c < '0')
            continue;
```

```
            int version;
            is >> version;
            storages_versions[store.first] = version;
            if (version == my_version) {
                got = true;
                is >> answer;
            }
        }

        if (got)
            send_message(frontend_push, answer);
        else
            send_message(frontend_push, "Error: getting");
}

std::pair<bool, std::string> Router::version(const std::string& request) {
        //std::cout << "version" << std::endl;
        std::string msg = std::to_string(my_version) + " " + request;

        for (auto& store : backend_push) {
            send_message(store.second, msg);
        }

        //std::cout << "all sent " << std::endl;
        bool newest = false;
        int avail_version = 0;
        for (auto& store : backend_pull) {
            std::string ans = recieve_message(store.second);
            std::istringstream is(ans);
            char c = is.peek();
            if (c > '9' || c < '0')
                continue;
            int version;
            is >> version;
            storages_versions[store.first] = version;
            if (version == my_version) {
                newest = true;
            }
            if (avail_version < version) {
                avail_version = version;
            }
        }
```

```
    return std::make_pair(newest, std::to_string(avail_version));
}


int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cout << "Usage: " << argv[0] << " <port>" << std::endl;
        return 1;
    }

    int port = std::stoi(argv[1]);

    Router router(port);

    std::thread handler(std::bind(&Router::Handler, &router));
    std::thread synchronizer(std::bind(&Router::Synchronizer, &router));
    std::thread connector(std::bind(&Router::Connector, &router));

    synchronizer.join();
    connector.join();
    handler.join();
    return 0;
}
```

storage.cpp

```
#include <iostream>
#include <string>
#include <map>
#include <thread>
#include <zmq.hpp>

#include "functions.hpp"

class Storage {
public:
    Storage() :
        context(1),
        connected(false),
        connector(context, ZMQ_REQ),
        store_pull(context, ZMQ_PULL),
        store_push(context, ZMQ_PUSH),
        update_push(context, ZMQ_PUSH),
        update_pull(context, ZMQ_PULL),
        my_version(0)
    { }

    void User() {
        std::cout << "connect/disconnect with router (connect <port> <keyword>)" << std:
        while (true) {
            std::string command;
            std::cin >> command;
            if (command == "connect") {
                if (connected) {
                    std::cout << "Already connected to " << host_port << std::endl;
                    continue;
                }
                int port;
                std::string key;
                std::cin >> port >> key;
                connected = Connect(port, key);
            }
            else if (command == "disconnect") {
                if (!connected) {
                    std::cout << "Not connected" << std::endl;
                    continue;
                }
```

```
            std::string mes = "disconnect " + my_key;
            send_message(connector, mes);
            std::string request = recieve_message(connector);
            std::cout << request << std::endl;
            connected = false;
        }

    }
}

bool Connect(int port, std::string key) {
    host_port = port;
    my_key = key;

    connector.connect(get_port_name(port+2));
    //std::cout << "MY KEY " << key << std::endl;
    std::string init_mes = "connect " + key;
    send_message(connector, init_mes);

    std::cout << "connecting..." << std::endl;
    std::string request = recieve_message(connector);
    //std::cout << "recieved " << request << std::endl;
    std::istringstream is(request);
    std::string number;
    is >> number;
    std::string command;
    is >> command;
    if (command == "init" && number == key) {
        int pull_port;
        int push_port;
        int up_port;
        int down_port;
        is >> pull_port >> push_port >> up_port >> down_port;
        store_pull.connect(get_port_name(pull_port));
        store_push.connect(get_port_name(push_port));
        update_pull.connect(get_port_name(up_port));
        update_push.connect(get_port_name(down_port));
        std::cout << "Succesfully connected" << std::endl;
        return true;
    }
    else {
        std::cout << "Error: missing key" << std::endl;
```

```
        return false;
    }
    return false;
}

void Handler() {
    while (true) {
        std::string request = recieve_message(store_pull);
        std::istringstream is(request);
        int version;
        is >> version;


        std::string command;
        is >> command;
        if (command == "add") {
            std::pair<std::string, int> pr;
            is >> pr.first >> pr.second;
            store[pr.first] = pr.second;
            my_version++;

            std::string msg = std::to_string(my_version) + " Ok.";
            send_message(store_push, msg);
        }
        else if (command == "remove") {
            std::pair<std::string, int> pr;
            is >> pr.first;
            store.erase(pr.first);
            my_version++;

            std::string msg = std::to_string(my_version) + " Ok.";
            send_message(store_push, msg);
        }
        else if (command == "get") {
            std::pair<std::string, int> pr;
            is >> pr.first;

            std::string msg = std::to_string(my_version) + " " +
                std::to_string(store[pr.first]) + " Ok.";
            send_message(store_push, msg);
            //std::cout << "sent get" << std::endl;
        }
```

```
        else if (command == "version") {
            std::string reply = std::to_string(version) + " Ok.";
            send_message(store_push, reply);
        }

        if (version > my_version) {
            //std::cout << "PLEASE UPDATE" << std::endl;
            std::string ask_for_update = my_key + " get_upd";
            send_message(update_push, ask_for_update);
        }
    }
}

void Synchronizer() {
    while (true) {
        std::string request = recieve_message(update_pull);
        std::istringstream is(request);
        std::string number;
        is >> number;
        std::string command;
        is >> command;
        if (command == "upload") {
            //std::cout << "UPLOADING" << std::endl;
            std::string reply = number +
                " upload " + std::to_string(my_version) + " ";
            for (auto& elem : store) {
                reply += elem.first + " " +
                    std::to_string(elem.second) + " ";
            }
            send_message(update_push, reply);
        }
        else if (command == "download") {
            //std::cout << "DOWNLOADING" << std::endl;
            int version;
            is >> version;
            my_version = version;
            store.clear();
            while (is) {
                std::string str;
                int num;
                is >> str >> num;
                store[str] = num;
```

```
                }
                std::string reply = number +
                    " updated " + std::to_string(my_version);
                send_message(update_push, reply);
            }
        }

    }

private:
    int host_port;
    std::string my_key;
    zmq::context_t context;
    bool connected;
    zmq::socket_t connector;

    zmq::socket_t store_pull;
    zmq::socket_t store_push;
    zmq::socket_t update_push;
    zmq::socket_t update_pull;
    std::map<std::string, int> store;
    int my_version;
};


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

    std::thread handler(std::bind(&Storage::Handler, &storage));
    std::thread synchronizer(std::bind(&Storage::Synchronizer, &storage));
    std::thread user(std::bind(&Storage::User, &storage));

    handler.join();
    synchronizer.join();
    user.join();

    return 0;
}
```

function.hpp

```cpp
#pragma once

#include <zmq.hpp>
#include <string>

#define TIMEOUT 200

bool send_message(zmq::socket_t& socket, const std::string& message_string) {
    zmq::message_t message(message_string.size());
    memcpy(message.data(), message_string.c_str(), message_string.size());
    bool ok = false;
    try {
        ok = socket.send(message);
    }catch(...) {ok = false;}
    return ok;
}

std::string msg_to_string(zmq::message_t& message) {
    return std::string(static_cast<char*>(message.data()), message.size());
}

std::string recieve_message(zmq::socket_t& socket) {
    zmq::message_t message;
    bool ok;
    try {
        ok = socket.recv(&message);
    } catch (...) {
        ok = false;
    }
    std::string recieved_message(static_cast<char*>(message.data()), message.size());
    if (recieved_message.empty() || !ok) {
        return "Error: Worker is not available";
    }
    return recieved_message;
}

std::string get_port_name(int port) {
    return "tcp://127.0.0.1:" + std::to_string(port);
}

int bind_socket(zmq::socket_t& socket) {
```

```
    int port = 30000;
    while (true) {
        try {
            socket.bind(get_port_name(port));
            break;
        } catch(...) {
            port++;
        }
    }
    int linger = 0;
    socket.setsockopt(ZMQ_RCVTIMEO, TIMEOUT);
    socket.setsockopt(ZMQ_SNDTIMEO, TIMEOUT);
    socket.setsockopt(ZMQ_LINGER, &linger, sizeof(linger));
    return port;
}
```

Выводы

Я проработал несколько моделей синхронизации централизованных распределённых хранилищ, научился создавать распределённые сетевые хранилища, используя библиотеку ZeroMQ, получил много практического опыта в написании многопоточных программ, улучшил свои навыки написания программ с межпроцессовым взаимодействием. А также просто получил удовольствие от проделанной работы.