Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа**

**по курсу**

**«Операционные системы»**

**Взаимодействие между процессами.  
Сокеты. Библиотека ZeroMQ**

Студент: Пивницкий Д.С.

Группа: М8о–206Б–19

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

* Управление процессами в ОС
* Изучение работы с библиотекой ZeroMQ
* Приобретение практических навыков в использовании знаний, полученных в течении курса
* Проведение исследования в выбранной предметной области

**Задание**

Требуется создать три программы А, В, С. Программа А принимает из стандартного ввода строки, а далее их отправляет программе В. Отправка строк должна производиться построчно. Программа В печатает в стандартный вывод, полученную строку от программы А. После получения программа В отправляет программе А сообщение о том что строка получена. До тех пор пока программа А не получит сообщение о получении строки от программы В, она не может отправлять следующую строку программе В. Программа С пишет в стандартный вывод количество отправленных символов программой А и количество принятых символов программой В. Данную информацию программа С получает от программ А и В соответственно

**Общие сведения о программе**

Взаимодействие между процессами реализовано с помощью сокетов.

Для реализации данной программы нам понадобится 4 типа сокетов:

**ZMQ\_REQ**: Сокет этого используется клиентом для отправки запросов на сервер и получения ответов от него. Этот тип сокета допускает только чередующуюся последовательность вызовов zmq\_send(запрос) и последующих вызовов zmq\_recv(ответ). Каждый отправленный запрос распределяется по всем службам, и каждый полученный ответ сопоставляется с последним выданным запросом. Если никакие службы недоступны, то любая операция отправки на сокете блокируется до тех пор, пока не станет доступна хотя бы одна служба.

**ZMQ\_REP**: Сокет этого типа используется сервером для приема запросов от клиента и отправки ответов ему. Этот тип сокета допускает только чередующуюся последовательность вызовов zmq\_recv(запрос) и последующих вызовов zmq\_send(ответ). Каждый полученный запрос помещается в очередь из числа всех клиентов, и каждый отправленный ответ направляется клиенту, который выдал последний запрос. Если первоначальный запрашивающий больше не существует, ответ уничтожается.

**ZMQ\_SUB**: Сокет этого типа используется подписчиком для подписки на данные, распространяемые издателем. Изначально сокет ZMQ\_SUB не подписан ни на какие сообщения, поэтому с помощью ZMQ\_SUBSCRIBE, находящейся в zmq\_setsockopt(3), мы указываем, на какие сообщения подписываться.

**ZMQ\_PUB**: Сокет этого типа используется издателем для распространения данных. Отправленные сообщения распространяются среди всех подключенных одноранговых узлов.

**Общий метод и алгоритм решения**.

*Программа А принимает из стандартного ввода строки, а далее их отправляет программе В. Отправка строк должна производиться построчно. До тех пор пока программа А не получит сообщение о получении строки от программы В, она не может отправлять следующую строку программе В.*

Создадим в программе “А” сокет типа ZMQ\_REQ и назначим ему адрес. Далее, вводим первую строку входных данных. Применяем к ней функцию SendMessage, таким образом, другой процесс, подключившись к этому сокету, получит данное сообщение. Далее считываем ответ сервера в том же цикле, то есть, пока мы не получим ответ сервера, мы не сможем начать новую итерацию. Также создадим сокет типа ZMQ\_PUB и назначим ему адрес. Он будет отправлять данные о текущих входных значениях, в нашем случае - о количестве символов.

*Программа В печатает в стандартный вывод, полученную строку от программы А. После получения программа В отправляет программе А сообщение о том что строка получена.*

В программе “В” создадим сокет типа ZMQ\_REP и подключим его к адресу сокета ZMQ\_REQ. После того как мы получаем данные из программы “А”, мы считываем эти данные функцией RecieveMessage и печатаем их в стандартный вывод. После этого через этот сокет отправляем подтверждение программе “А”. Также создадим сокет типа ZMQ\_PUB и назначим ему адрес. Он будет отправлять данные о текущих полученных значениях, в нашем случае - о количестве символов*.*

*Программа С пишет в стандартный вывод количество отправленных символов программой А и количество принятых символов программой В. Данную информацию программа С получает от программ А и В соответственно.*

В программе “С” создадим два сокета типа ZMQ\_SUB и подключим их к адресам сокетов ZMQ\_PUB из программ “А” и “В”. Далее создадим два потока thread, в которые будем считывать данные из описанных выше сокетов при помощи функций ReadA и ReadB.

Я не объединял программы с помощью методов fork, exec, и.т.д. для более наглядной демонстрации работы сокетов и разделенного вывода.

**Основные файлы программы**

**Library.cpp:**

#include <memory>

#include <stdexcept>

#include <iostream>

#include <exception>

#include <zmq.hpp>

// Общий класс сокет

class Socket {

public:

explicit Socket(int socket\_type);

virtual ~Socket() noexcept;

virtual std::string ReceiveMessage();

zmq::context\_t\* context\_ = nullptr;

zmq::socket\_t\* socket\_ = nullptr;

virtual void SendMessage(const std::string& msg, bool non\_block);

protected:

int socket\_type\_;

};

// Виды сокетов:

class Reply\_S : public Socket {

public:

Reply\_S() : Socket(ZMQ\_REP) {};

};

class Request\_S : public Socket {

public:

Request\_S() : Socket(ZMQ\_REQ) {};

};

class Publish\_S : public Socket {

public:

Publish\_S(): Socket(ZMQ\_PUB) {};

};

class Subscribe\_S : public Socket {

public:

Subscribe\_S(): Socket(ZMQ\_SUB) { socket\_->setsockopt(ZMQ\_SUBSCRIBE, "", 0);};

};

class Message {

// Операции для обработки строк (получить сообщение для сокета из строки string или получить строку string из сообщения сокета)

public:

static zmq::message\_t MakeMessage(const std::string& string\_m);

static std::string MakeString(const zmq::message\_t& msg);

};

zmq::message\_t Message::MakeMessage(const std::string& string\_m) {

zmq::message\_t resu(string\_m.size());

memcpy(resu.data(), string\_m.c\_str(), string\_m.size());

return resu;

}

std::string Message::MakeString(const zmq::message\_t& msg) {

return std::string((const char\*)msg.data(), msg.size());

}

// Конструктор

Socket::Socket(int socket\_type)

: context\_(new zmq::context\_t(1)), socket\_(new zmq::socket\_t(\*context\_, socket\_type)), socket\_type\_(socket\_type) {}

// Деконструктор

Socket::~Socket() noexcept {

delete socket\_;

delete context\_;

}

// Функция отправки сообщения

void Socket::SendMessage(const std::string& msg, bool non\_block) {

zmq::message\_t to\_send = Message::MakeMessage(msg);

zmq::send\_flags flags = non\_block ? zmq::send\_flags::dontwait : zmq::send\_flags::none;

zmq::send\_result\_t res = socket\_->send(to\_send, flags);

}

// Функция приема сообщения

std::string Socket::ReceiveMessage() {

zmq::message\_t received;

zmq::recv\_result\_t res = socket\_->recv(received);

return Message::MakeString(received);

}

// Считывание данных из программы А

void ReadA(Socket& socket) {

while (true) {

std::string received = socket.ReceiveMessage();

std::cout << ("A sent " + received + " symbols\n");

}

}

// Считывание данных из программы В

void ReadB(Socket& socket) {

while (true) {

std::string received = socket.ReceiveMessage();

std::cout << ("B recieved " + received + " symbols\n");

}

}

**A\_client.cpp:**

#include "lib.cpp"

int main() {

Request\_S ABSocket;

ABSocket.socket\_->bind("tcp://\*:1111");

Publish\_S CountA;

CountA.socket\_->bind("tcp://\*:2222");

std::string input;

while (std::getline(std::cin, input)) {

ABSocket.SendMessage(input, false);

CountA.SendMessage(std::to\_string(input.size()),true);

std::cout << ABSocket.ReceiveMessage() << "\n";

}

return 0;

}

**B\_server1.cpp:**

#include "lib.cpp"

int main() {

Reply\_S BASocket;

BASocket.socket\_->connect("tcp://localhost:1111");

Publish\_S CountB;

CountB.socket\_->bind("tcp://\*:3333");

while (true) {

std::string recieved\_message = BASocket.ReceiveMessage();

std::cout << recieved\_message << '\n';

CountB.SendMessage(std::to\_string(recieved\_message.size()),true);

BASocket.SendMessage("ACCEPTED", false);

}

return 0;

};

**C\_server2.cpp:**

#include "lib.cpp"

#include <thread>

int main() {

Subscribe\_S ResA;

ResA.socket\_->connect("tcp://localhost:2222");

Subscribe\_S ResB;

ResB.socket\_->connect("tcp://localhost:3333");

std::thread t1(ReadA, std::ref(ResA));

std::thread t2(ReadB, std::ref(ResB));

t1.join();

t2.join();

return 0;

}

**Пример работы**

daniel@daniel-Ideapad-Z570: ~ cd os/KP  
daniel@daniel-Ideapad-Z570: ~ ./A

ABC

ACCEPTED

a b c

ACCEPTED

ACCEPTED

\_

ACCEPTED

"text with symbols: @ ^ & # !!!!?/'

ACCEPTED

daniel@daniel-Ideapad-Z570: ~ ./B

ABC

abc

"text with symbols: @ ^ & # !!!!?/'

daniel@daniel-Ideapad-Z570: ~ ./C

A sent 3 symbols

B recieved 3 symbols

A sent 5 symbols

B recieved 5 symbols

A sent 0 symbols

B recieved 0 symbols

A sent 1 symbols

B recieved 1 symbols

A sent 35 symbols

B recieved 35 symbols

**Вывод**

Сокеты - удобный способ взаимодействия между процессами, имеющий как преимущества, так и недостатки. В процессе выполнения программы я столкнулся с некоторыми из них:

Преимущества:  
- быстрая передача данных и высокая производительность  
- подходит для обмена информацией в реальном времени между клиентом и сервером  
- можно самостоятельно настроить операции передачи и считывания сообщений, таким образом можно удобно реализовать шифрование данных и.т.д.

Недостатки:  
- передаваемые данные должны быть проанализированы и преобразованы к необходимому для передачи виду  
- относительная сложность в программировании

В целом - сокеты применяются в современном программировании, и изучение основных принципов работы с ними полезно. Библиотека ZeroMQ является актуальной и удобной библиотекой для создания собственных сокетов и работы с ними.