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 A, B, C. Программа A принимает из стандартного ввода строки, а далее их отправляет программе B. Отправка строк должна производиться построчно. Программа B печатает в стандартный вывод, полученную строку от программы A. После получения программа B отправляет программе A сообщение о том что строка получена. До тех пор пока программа A не получит сообщение о получении строки от программы B, она не может отправлять следующую строку программе B. Программа C пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой B. Данную информацию программа C получает от программ A и B 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 “A” 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 “B” создадим сокет типа ZMQ_REP и подключим его к адресу сокета ZMQ_REQ. После того как мы получаем данные из программы “A”, мы считываем эти данные функцией RecieveMessage и печатаем их в стандартный вывод. После этого через этот сокет отправляем подтверждение программе “A”. 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 C пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой B. Данную информацию программа C получает от программ A и B соответственно.

В программе “C” создадим два сокета типа ZMQ_SUB и подключим их к адресам сокетов ZMQ_PUB из программ “A” и “B”. 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 A
void ReadA(Socket& socket) {
    while (true) {
        std::string received = socket.ReceiveMessage();
        std::cout << ("A sent " + received + " symbols\n");
    }
}
// Считывание данных из программы B
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.