Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Операционные системы»**

**Тема работы**

**Использование знаний и навыков, полученных в течение курса**

Студент: Ханнанов Руслан Маратович

Группа: М8О-208Б-20

Вариант: 20

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/Naksen/OS

**Постановка задачи**

Необходимо написать 3-и программы. Далее будем обозначать эти программы A, B, C. Программа A принимает из стандартного потока ввода строки, а далее их отправляет программе С. Отправка строк должна производится построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе А сообщение о том, что строка получена. До тех пор пока программа А не примет «сообщение о получение строки» от программы С, она не может отправлять следующую строку программе С. Программа B пишет в стандартный вывод количество отправленных символов программой А и количество принятых символов программой С. Данную информацию программа B получает от программ A и C соответственно.

**Общие сведения о программе**

Предварительно реализованы три программы:

A.cpp

B.cpp

C.cpp

Между которыми будет происходить взаимодействие и обмен сообщениями в соответствии с заданием.

**Общий метод и алгоритм решения**

Makefile:

all:

    g++ A.cpp -lzmq -o A -Wall -pedantic

    g++ B.cpp -lzmq -o B -Wall -pedantic

    g++ C.cpp -lzmq -o C -Wall -pedantic

clean:

    rm -rf A B C

Общение между процессами я решил реализовать с помощью ZeroMQ. В программе используется тип соединения Request-Response. Программа А принимает сообщения из стандартного потока ввода и посылает их программе С. Программа С принимает сообщение, печатает его и пересылает в программу B, после чего присылает буфферное сообщение назад в программу А о том, что она получила новую строчку. После этого программа А отсылает программе B размер сообщения (количество символов), которое было отправлено С. Программа B выводит размер сообщения, полученного от С и размер сообщения отосланного от А. Если всё работает корректно, то эти размеры должны быть одинаковыми.

**Исходный код**A.cpp:

#include <unistd.h>

#include <sstream>

#include <set>

#include <string>

#include <iostream>

#include <zmq.hpp>

using namespace std;

int main(){

    const string endpoint = "tcp://localhost:5555";

    zmq::context\_t context;

    zmq::socket\_type type = zmq::socket\_type::req;

    zmq::socket\_t socket (context, type);

    cout << "Please, enter the string" << endl;

    socket.connect(endpoint);

    //TO B

    const string to\_b = "tcp://localhost:5554";

    zmq::socket\_type b\_type = zmq::socket\_type::req;

    zmq::socket\_t b\_socket (context, b\_type);

    b\_socket.connect(to\_b);

    pid\_t C = fork();

    if (C == -1) {

        perror("fork");

        return -1;

    }

    if (C == 0) {

        pid\_t B = fork();

        if (B == -1) {

            perror("fork");

            return -1;

        }

        if (B == 0){

            execl("./B", "./B",NULL);

        } else {

            execl("./C", "./C",NULL);

        }

    }

    string s;

    while(cin >> s) {

        // Sending to C

        zmq::message\_t reply;

        zmq::message\_t message(s.size());

        memcpy(message.data(), s.c\_str(), s.size());

        socket.send(message);

        if (s == "exit") {

            socket.disconnect(endpoint);

            b\_socket.disconnect(to\_b);

            return 0;

        }

        // Receive from C

        socket.recv(reply);

        std::string received\_msg(static\_cast<char\*>(reply.data()), reply.size());

        //Sending to B

        string c = "TO B";

        int cnt = s.size();

        string cnt\_s = to\_string(cnt);

        zmq::message\_t message\_to\_b(cnt\_s.size());

        memcpy(message\_to\_b.data(), cnt\_s.c\_str(), cnt\_s.size());

        b\_socket.send(message\_to\_b);

        //Get from B

        zmq::message\_t rep;

        b\_socket.recv(rep);

    }

}

B.cpp:

#include <unistd.h>

#include <sstream>

#include <string>

#include <zmq.hpp>

#include <iostream>

using namespace std;

int main(){

    const string to\_c = "tcp://\*:5554";

    zmq::context\_t context;

    zmq::socket\_type type = zmq::socket\_type::rep;

    zmq::socket\_t socket (context, type);

    socket.bind(to\_c);

    while (1) {

        // Receive the message from C

        zmq::message\_t message;

        socket.recv(message);

        std::string received\_msg(static\_cast<char\*>(message.data()), message.size());

        if (received\_msg == "exit") {

            socket.unbind(to\_c);

            exit(0);

        }

        cout << "B: size of message from C:" << received\_msg.size() << endl;

        // Send to C

        sleep(1);

        string answer = "b: Get new string";

        zmq::message\_t ans(answer.size());

        memcpy(ans.data(), answer.c\_str(), answer.size());

        socket.send(ans);

        // Get message from A

        zmq::message\_t message\_from\_a;

        socket.recv(message\_from\_a);

        std::string cnt(static\_cast<char\*>(message\_from\_a.data()), message\_from\_a.size());

        cout << "B: size of message from A:" << cnt << endl;

        // Send message to A

        string z = "get cnt from a";

        zmq::message\_t buf(z.size());

        memcpy(buf.data(), z.c\_str(), z.size());

        socket.send(buf);

    }

    return 0;

}

C.cpp:

#include <unistd.h>

#include <sstream>

#include <string>

#include <zmq.hpp>

#include <iostream>

using namespace std;

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

    const string endpoint = "tcp://\*:5555";

    // FROM A

    zmq::context\_t context;

    zmq::socket\_type type = zmq::socket\_type::rep;

    zmq::socket\_t socket (context, type);

    socket.bind(endpoint);

    // TO B

    const string to\_b = "tcp://localhost:5554";

    zmq::socket\_type b\_type = zmq::socket\_type::req;

    zmq::socket\_t b\_socket (context, b\_type);

    b\_socket.connect(to\_b);

    std::string message;

    while ( 1) {

        // Get message from A

        zmq::message\_t message;

        socket.recv(message);

        std::string received\_msg(static\_cast<char\*>(message.data()), message.size());

        if (received\_msg == "exit") {

            b\_socket.send(message);

            socket.unbind(endpoint);

            socket.disconnect(to\_b);

            exit(0);

        }

        cout << "C: " << received\_msg << endl;

        //Sending message to A

        string answer = "Get new string";

        zmq::message\_t ans(answer.size());

        memcpy(ans.data(), answer.c\_str(), answer.size());

        //cout << "C: Sending to A..." << endl;

        socket.send(ans);

        //Sending message to B

        b\_socket.send(message);

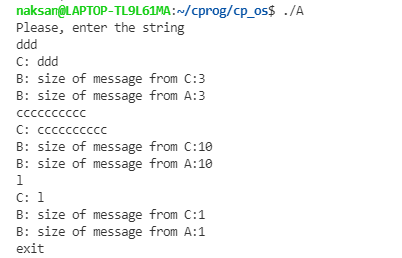
        zmq::message\_t ans\_from\_b;

        //Receiving message from B

        b\_socket.recv(ans\_from\_b);

    }

}

**Демонстрация работы программы**

**Выводы**Данная лабораторная работа понравилась мне тем, что передо мной стояла задача, а способ её выполнения можно было выбрать уже самому. В процессе работы я долго не мог решить, на чём остановиться, но в итоге выбрал ZeroMQ, так как появилось желание получше в нём разобраться и использовать что-то более сложное, чем pipe или memory mmap. В итоге я улучшил своё понимания работы с ZeroMQ и успешно реализовал поставленную задачу.