Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

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

Студент: Полонский Кирилл Андреевич

Группа: М8О-208Б-20

Вариант: \_\_

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/kirillpolonskii/OS/tree/master/os_cp>

**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. Способ организация межпроцессорного взаимодействия выбирает студент.

**Общие сведения о программе**

Файл zmq\_helper.hpp содержит вспомогательные данные, файлы prog\_a.cpp, и prog\_b.cpp и prog\_c.cpp содержат реализацию программ A, B, C, сборка осуществляется с помощью утилиты cmake.

**Общий метод и алгоритм решения**

Межпроцессорное взаимодействие осуществляется с помощью очереди сообщений, программы A и C используют паттерн REQ-REP, а программы A и B, C и B — PUSH-PULL.

**Исходный код**

zmq\_helper.hpp

#ifndef OS\_CP\_CLION\_ZMQ\_HELPER\_HPP

#define OS\_CP\_CLION\_ZMQ\_HELPER\_HPP

#include <zmq.hpp>

std::string addressAB = "tcp://127.0.0.1:5558";

std::string addressAC = "tcp://127.0.0.1:5559";

std::string addressBC = "tcp://127.0.0.1:5560";

#endif //OS\_CP\_CLION\_ZMQ\_HELPER\_HPP

prog\_a.cpp

#include <iostream>

#include "zmq\_helper.hpp"

zmq::context\_t context;

int main() {

zmq::socket\_t sockAB(context, zmq::socket\_type::push);

zmq::socket\_t sockAC(context, zmq::socket\_type::req);

sockAB.bind(addressAB);

sockAC.connect(addressAC);

std::string curStr;

while(std::cin >> curStr){

if(curStr == "exit"){

zmq::message\_t zMesToC(curStr);

sockAC.send(zMesToC, zmq::send\_flags::none);

zmq::message\_t zMesToB(curStr);

sockAB.send(zMesToB, zmq::send\_flags::none);

break;

}

zmq::message\_t zMesToC(curStr);

sockAC.send(zMesToC, zmq::send\_flags::none);

std::string mesToB = std::to\_string(curStr.size());

zmq::message\_t zMesToB(mesToB);

sockAB.send(zMesToB, zmq::send\_flags::none);

zmq::message\_t zMesFromC;

if(sockAC.recv(zMesFromC) == -1){

return 1;

}

if(zMesFromC.to\_string() == "String was received"){

continue;

}

}

sockAB.close();

sockAC.close();

return 0;

}

prog\_b.cpp

#include <iostream>

#include "zmq\_helper.hpp"

zmq::context\_t context;

int main() {

zmq::socket\_t sockAB(context, zmq::socket\_type::pull);

zmq::socket\_t sockBC(context, zmq::socket\_type::pull);

sockAB.connect(addressAB);

sockBC.connect(addressBC);

while(true){

std::cout << "in while**\n**";

zmq::message\_t zMesFromA;

if(sockAB.recv(zMesFromA) == -1){

return 1;

}

std::string mesFromA = zMesFromA.to\_string();

if(mesFromA == "exit"){

break;

}

else{

std::cout << mesFromA << std::endl;

}

//sleep(1);

zmq::message\_t zMesFromC;

if(sockBC.recv(zMesFromC) == -1){

return 1;

}

std::string mesFromC = zMesFromC.to\_string();

std::cout << mesFromC << std::endl;

}

sockAB.close();

sockBC.close();

return 0;

}

prog\_c.cpp

#include <iostream>

#include "zmq\_helper.hpp"

zmq::context\_t context;

int main() {

zmq::socket\_t sockAC(context, zmq::socket\_type::rep);

zmq::socket\_t sockBC(context, zmq::socket\_type::push);

sockAC.bind(addressAC);

sockBC.bind(addressBC);

std::string curStr;

while(true){

zmq::message\_t zMesFromA;

if(sockAC.recv(zMesFromA) == -1){

return 1;

}

std::string mesFromA = zMesFromA.to\_string();

if(mesFromA == "exit"){

break;

}

else{

std::cout << mesFromA << std::endl;

}

std::string mesToA("String was received");

zmq::message\_t zMesToA(mesToA);

sockAC.send(zMesToA, zmq::send\_flags::none);

std::string mesToB = std::to\_string(mesFromA.size());

zmq::message\_t zMesToB(mesToB);

sockBC.send(zMesToB, zmq::send\_flags::none);

}

sockAC.close();

sockBC.close();

return 0;

}

**Демонстрация работы программы**

| progA | progB | progC |
| --- | --- | --- |
| kirill@kirill-acpire:~/labsMAI/sem3/os\_cp\_clion/cmake-build-debug$ ./progC  hello  salut  bye | kirill@kirill-acpire:~/labsMAI/sem3/os\_cp\_clion/cmake-build-debug$ ./progB  5  5  5  5  3  3 | kirill@kirill-acpire:~/labsMAI/sem3/os\_cp\_clion/cmake-build-debug$ ./progC  hello  salut  bye |

**Выводы**

В ходе выполнения курсового проекта я закрепил навыки работы с очередью сообщений.