Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Зверев М. Е.

Группа: М8О–306Б–19

Вариант: на удовлетворительно

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

## Постановка задачи

## Цель работы

Приобретение практических навыков в:

* Приобретение практических навыков в использовании знаний, полученных в течении курса.
* Проведение исследования в выбранной предметной области

## Задание

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 отправялять 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

Программа состоит из трех файлов a.cpp, b.cpp и c.cpp. Межпроцессное взаимодействие осуществляется с помощью очереди сообщений zmq. Для работы необходимо запустить 3 программы в разных терминалах.

## Описание идеи решения

Взаимодействие между процессами реализовано с помощью сокетов. В работе используется 4 вида сокетов ZMQ\_REQ, ZMQ\_ PUSH, ZMQ\_ PULL, ZMQ\_REP.

Сокет типа ZMQ\_REQ используется для отправки запросов и получения ответов. Этот тип сокета допускает только чередующуюся последовательность вызовов zmq\_send(запрос) и последующих вызовов zmq\_recv(ответ).

Сокет типа ZMQ\_PUSH используется узлом конвейера для отправки сообщений нижестоящим узлам конвейера. Сообщения передаются по кругу всем подключенным нижестоящим узлам. Функция zmq\_recv() не реализована для этого типа сокета.

Сокет типа ZMQ\_PULL используется узлом конвейера для приема сообщений от вышестоящих узлов конвейера. Сообщения помещаются в справедливую очередь из всех подключенных восходящих узлов. Функция zmq\_send() не реализована для данного типа сокета.

Сокет типа ZMQ\_REP используется службой 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).

Программа А функцией getline() 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.

Программа “А” использует 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\_PUSH 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.

Программа “В” сиспользует сокет типа ZMQ\_PULL, 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\_PULL 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\_REP и ZMQ\_PUSH и подключим их к адресам сокетов ZMQ\_PULL из программ “А” и “В”. Далее будем считывать данные из описанных выше сокетов при помощи функции ReceiveMessage.

## Основные файлы программы

**a.cpp**

#include <iostream>

#include <vector>

#include <string>

#include <cstring>

#include "zmq.h"

using namespace std;

void SendMessage(void \*request, const string &message) {

zmq\_msg\_t req; // Массив

zmq\_msg\_init\_size(&req, message.size()); // Инициализация сообщения

memcpy(zmq\_msg\_data(&req), message.c\_str(), message.size()); // c\_str -> req

zmq\_msg\_send(&req, request, 0);

zmq\_msg\_close(&req);

}

string ReceiveMessage(void \*respond) {

string result = "";

zmq\_msg\_t reply;

zmq\_msg\_init(&reply); // Инициализация пустого сообщения перед отправкой

zmq\_msg\_recv(&reply, respond, 0);

char\* ptr = static\_cast<char\*>(zmq\_msg\_data(&reply));

result = string(ptr, zmq\_msg\_size(&reply));

zmq\_msg\_close(&reply);

return result;

}

int main() {

void \*context\_c = zmq\_ctx\_new();

void \*request\_c = zmq\_socket(context\_c, ZMQ\_REQ);

zmq\_connect(request\_c, "tcp://localhost:9090");

void \*context\_b = zmq\_ctx\_new();

void \*request\_b = zmq\_socket(context\_b, ZMQ\_PUSH);

zmq\_connect(request\_b, "tcp://localhost:8080");

cout << "Enter strings:" << '\n';

while(true) {

string str;

getline(cin, str);

SendMessage(request\_c, str);

SendMessage(request\_b, to\_string(str.size()));

string received\_message = ReceiveMessage(request\_c);

}

zmq\_close(request\_c);

zmq\_ctx\_destroy(context\_c);

zmq\_close(request\_b);

zmq\_ctx\_destroy(context\_b);

return 0;

}

**b.cpp**

#include <iostream>

#include <string>

#include <cstring>

#include "zmq.h"

using namespace std;

int ReceiveMessage(void \*socket) {

size\_t result = 0;

zmq\_msg\_t reply;

zmq\_msg\_init(&reply);

zmq\_msg\_recv(&reply, socket, 0);

char\* ptr = static\_cast<char\*>(zmq\_msg\_data(&reply));

result = atoi(ptr);

return result;

}

int main() {

void \*context\_a = zmq\_ctx\_new();

void \*socket\_a = zmq\_socket(context\_a, ZMQ\_PULL);

zmq\_bind(socket\_a, "tcp://\*:8080");

void \*context\_c = zmq\_ctx\_new();

void \*socket\_c = zmq\_socket(context\_c, ZMQ\_PULL);

zmq\_bind(socket\_c, "tcp://\*:7070");

bool to\_stop\_a = false, to\_stop\_c = false;

string received\_a = "Received from A: ";

string received\_c = "Received from C: ";

int a\_bytes = 0;

int c\_bytes = 0;

for (;;) {

if (a\_bytes != -1) {

a\_bytes = ReceiveMessage(socket\_a);

}

if (c\_bytes != -1) {

c\_bytes = ReceiveMessage(socket\_c);

}

if (!to\_stop\_a && !to\_stop\_c) {

cout << received\_a << a\_bytes << '\n';

cout << received\_c << c\_bytes << '\n';

}

else if (!to\_stop\_a && to\_stop\_c) {

cout << received\_a << a\_bytes << '\n';

}

else {

cout << received\_c << c\_bytes << '\n';

}

}

zmq\_close(socket\_a);

zmq\_ctx\_destroy(context\_a);

zmq\_close(socket\_c);

zmq\_ctx\_destroy(context\_c);

return 0;

}

**c.cpp**

#include <iostream>

#include <string>

#include <cstring>

#include "zmq.h"

using namespace std;

string ReceiveMessage(void \*respond) {

string result = "";

zmq\_msg\_t request;

zmq\_msg\_init(&request);

zmq\_msg\_recv(&request, respond, 0);

char\* ptr = static\_cast<char\*>(zmq\_msg\_data(&request));

result = string(ptr, zmq\_msg\_size(&request));

zmq\_msg\_close(&request);

return result;

}

void SendMessage(void \*respond, const string &message) {

zmq\_msg\_t reply;

zmq\_msg\_init\_size(&reply, message.size());

memcpy(zmq\_msg\_data(&reply), message.c\_str(), message.size());

zmq\_msg\_send(&reply, respond, 0);

zmq\_msg\_close(&reply);

}

int main() {

void \*context\_a = zmq\_ctx\_new();

void \*respond\_a = zmq\_socket(context\_a, ZMQ\_REP);

zmq\_bind(respond\_a, "tcp://\*:9090");

void \*context\_b = zmq\_ctx\_new();

void \*respond\_b = zmq\_socket(context\_b, ZMQ\_PUSH);

zmq\_connect(respond\_b, "tcp://localhost:7070");

string message = "Received message from A";

string empty\_message = "";

for (;;) {

string received\_message = ReceiveMessage(respond\_a);

cout << received\_message << '\n';

SendMessage(respond\_a, message);

SendMessage(respond\_b, to\_string(received\_message.size()));

}

zmq\_close(respond\_a);

zmq\_ctx\_destroy(context\_a);

zmq\_close(respond\_b);

zmq\_ctx\_destroy(context\_b);

return 0;

}

## Пример работы

mavrik@LAPTOP-CP135GLC:/mnt/c/Users/markz/Desktop/Labs/OS/KP$ ./a

Enter strings:

hello

privet

poka

goodwork

well

)

mavrik@LAPTOP-CP135GLC:/mnt/c/Users/markz/Desktop/Labs/OS/KP$ ./b

Received from A: 5

Received from C: 5

Received from A: 6

Received from C: 6

Received from A: 4

Received from C: 4

Received from A: 8

Received from C: 8

Received from A: 4

Received from C: 4

Received from A: 1

Received from C: 1

Received from A: 1

Received from C: 1

mavrik@LAPTOP-CP135GLC:/mnt/c/Users/markz/Desktop/Labs/OS/KP$ ./c

hello

privet

poka

goodwork

well

)

## Вывод

Сокеты являются одним из лучших способов построения взаимодействия процессов. Их преимуществом является быстрая передача данных и высокая производительность. Гибкость сокетов позволяют обмениваться информацией между клиентом и сервером, публицистом и подписчиками, то есть производить работу по разным шаблонам. Мы можем самостоятельно настроить операции передачи и считывания сообщений, таким образом можно удобно реализовать шифрование данных. В качестве недостатков можно отметить относительную сложность в программировании, заключающуюся в том, что нужно в определённом, специфичном формате передавать данные. Сегодня сокеты широко применимы в программировании. Благодаря им можно написать большинство мультипроцессорных программ.