Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Операционные системы»**

Студент: Базаргармаев Нима Дондокович

Группа: М8О –301Б-18

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Тестирование
6. Демонстрация работы программы
7. Вывод

**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 отправляет их 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**

Программа состоит из трех файлов a.cpp, b.cpp и c.cpp. Межпроцессное взаимодействие осуществляется с помощью очереди сообщений zmq.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Считываю данные в программе a.cpp и подключаюсь к программам b.cpp и c.cpp.
2. Аналогично подключаюсь в программах b.cpp и c.cpp
3. Принимаю данные и вывожу их в стандартный поток

**Листинг программы.**

**Файл a.cpp**

#include <iostream>

#include <utility>

#include <vector>

#include <string>

#include <cstring>

#include <unistd.h>

#include "zmq.h"

using namespace std;

vector<string> ReadData(size\_t size, istream& is = cin) {

vector<string> result;

result.reserve(size + 1);

for (size\_t i = 0; i < size; ++i) {

string value = "";

is >> value;

result.push\_back(move(value));

}

result.push\_back("END");

return result;

}

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

zmq\_msg\_t req;

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

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

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

zmq\_msg\_close(&req);

}

string ReceiveMessage(void\* respond) {

string result = "";

zmq\_msg\_t reply;

zmq\_msg\_init(&reply);

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

char\* ptr = reinterpret\_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\_REQ);

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

cout << "Enter the number of strings: ";

size\_t size = 0;

cin >> size;

vector<string> strings = ReadData(size, cin);

size\_t pos = 0;

bool send = false, received = true;

for(;;) {

if (pos == strings.size() - 1) {

SendMessage(request\_c, strings[pos]);

SendMessage(request\_b, "-1");

ReceiveMessage(request\_b);

break;

}

if (received) {

SendMessage(request\_c, strings[pos]);

SendMessage(request\_b, to\_string(strings[pos].size()));

ReceiveMessage(request\_b);

pos++;

send = true;

received = false;

}

if (send) {

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

cout << received\_message << endl;

received = true;

send = false;

}

}

zmq\_close(request\_c);

zmq\_ctx\_destroy(context\_c);

zmq\_close(request\_b);

zmq\_ctx\_destroy(context\_b);

return 0;

}

**Файл b.cpp**

#include <iostream>

#include <string>

#include <cstring>

#include <stdlib.h>

#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 = reinterpret\_cast<char\*>(zmq\_msg\_data(&reply));

result = atoi(ptr);

return result;

}

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

zmq\_msg\_t msg;

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

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

zmq\_msg\_send(&msg, socket, 0);

zmq\_msg\_close(&msg);

}

int main() {

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

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

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

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

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

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);

SendMessage(socket\_a, "");

}

to\_stop\_a = a\_bytes == -1 ? true: false;

if (c\_bytes != -1) {

c\_bytes = ReceiveMessage(socket\_c);

SendMessage(socket\_c, "");

}

to\_stop\_c = c\_bytes == -1 ? true : false;

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

break;

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

cout << received\_a << a\_bytes << endl;

cout << received\_c << c\_bytes << endl;

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

cout << received\_a << a\_bytes << endl;

} else {

cout << received\_c << c\_bytes << endl;

}

}

zmq\_close(socket\_a);

zmq\_ctx\_destroy(context\_a);

zmq\_close(socket\_c);

zmq\_ctx\_destroy(context\_c);

return 0;

}

**Файл c.cpp**

#include <iostream>

#include <string>

#include <cstring>

#include <unistd.h>

#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 = reinterpret\_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\_REQ);

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

string message = "Received message from A";

string empty\_message = "";

for (;;) {

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

if (received\_message == "END") {

SendMessage(respond\_b, "-1");

ReceiveMessage(respond\_b);

break;

} else if (received\_message == "") {

cout << "Empty" << endl;

SendMessage(respond\_a, empty\_message);

} else {

cout << received\_message << endl;

SendMessage(respond\_a, message);

}

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

ReceiveMessage(respond\_b);

}

zmq\_close(respond\_a);

zmq\_ctx\_destroy(context\_a);

zmq\_close(respond\_b);

zmq\_ctx\_destroy(context\_b);

return 0;

}

**Тестирование.**

См. следующий пункт.

**Демонстрация работы программы.**

user@DESKTOP-4RFVSK2:~/os/kp$ ./a

Enter the number of strings: 5

dsa

asdfdsg

dfhjhhiwe

mbkrltjbs

gnlerg

Received message from A

Received message from A

Received message from A

Received message from A

Received message from A

user@DESKTOP-4RFVSK2:~/os/kp$ ./b

Received from A: 3

Received from C: 3

Received from A: 7

Received from C: 7

Received from A: 9

Received from C: 9

Received from A: 9

Received from C: 9

Received from A: 6

Received from C: 6

user@DESKTOP-4RFVSK2:~/os/kp$ ./c

dsa

asdfdsg

dfhjhhiwe

mbkrltjbs

gnlerg

**Вывод.**

В ходе работы я научился организовывать межпроцессное взаимодействие программ с помощью библиотеки ZMQ. К выбору очереди сообщений я пришел не сразу, поэтому во время выполнения работы я так же познакомился и с другими способами организации межпроцессного взаимодействия в UNIX-подобных системах. Помимо этого, я научился работать с механизмами синхронизации процессов в операционных системах.