Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Кузьмичев Александр Николаевич
Группа: М80 – 306Б-18
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020

Содержание

1. **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 C. Отправка строк должна производится построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе A сообщение о том, что строка получена. До тех пор, пока программа A не примет «сообщение о получение строки» от программы C, она не может отправлять следующую строку программе C. Программа B пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно.

2. **Общий метод и алгоритм решения**

Организация межпроцессорного взаимодействия производится с помощью средств библиотеки ZeroMQ. Все три программы попарно используют связь типа сервер-клиент (REP-REQ). Для общения между программами используется структура Message. Определены функции отправки и приема сообщений send_message и receive_message соответственно. Так же отдельно определена функция отправки и приема строки send_text и receive_text. Это нужно для корректной отправки string-строки. Чтобы закончить ввод текста в программе A, требуется ввести слово END на отдельной строке.

3. **Файлы программы**
   ABC.h

```
#pragma once
#include <string.h>
#include <stdio.h>
#include <iostream>
```

```
#include "zmq.h"
using namespace std;

typedef struct {
	int sym;
	int task;
	int status;
} Message;

void print_menu() {
	cout << "1 - enter text" << endl
	<< "2 - send text" << endl
	<< "3 - clear buffer" << endl
	<< "0 - exit" << endl;
}

void send_message(void* socket, Message* out) {
	zmq_msg_t msg;
	zmq_msg_init_size(&msg, sizeof(Message));
	memcpy(zmq_msg_data(&msg), out, sizeof(Message));
	zmq_msg_send(&msg, socket, 0);
	zmq_msg_close(&msg);
}

void send_text(void* socket, string* text) {
	zmq_msg_t msg;
	zmq_msg_init_size(&msg, text->size());
	memcpy(zmq_msg_data(&msg), text->c_str(), text->size());
	zmq_msg_send(&msg, socket, 0);
	zmq_msg_close(&msg);
}

string receive_text(void* socket, zmq_msg_t* msg) {
	string text;
	zmq_msg_init(msg);
	zmq_msg_recv(msg, socket, 0);
	char* ptr = static_cast<char*>(zmq_msg_data(msg));
	return string(ptr, zmq_msg_size(msg));
}

Message* receive_message(void* socket, zmq_msg_t* msg) {
	Message *in;
```

```
        zmq_msg_init(msg);
        zmq_msg_recv(msg, socket, 0);
        in = static_cast<Message*>(zmq_msg_data(msg));
        return in;
    }

    Message init_message(){
        Message mes;
        mes.sym = -1;
        mes.task = -1;
        mes.status = -1;
        return mes;
    }
```

A.cpp

```
#include
<vector>
    #include "ABC.h"

    int main() {
        Message AtoB, *BtoA, AtoC, *CtoA;
        zmq_msg_t msg, out;
        vector<string> text;
        string s;
        int comand;
        AtoB = init_message();
        AtoC = init_message();

        void* context = zmq_ctx_new();
        void* psocket = zmq_socket(context, ZMQ_REQ);
        void* mysocket = zmq_socket(context, ZMQ_REP);
        zmq_bind(mysocket, "tcp://*:5001");
        zmq_connect(psocket, "tcp://localhost:5002");

        AtoB.status = 1;
        send_message(psocket, &AtoB);

        CtoA = receive_message(mysocket, &msg);
        if (CtoA->status == 1) cout << "A: C connected\n";
```

```
	else {
		cout << "Error: unsuccessful connection C to A";
		return -1;
	}
	zmq_msg_close(&msg);

	BtoA = receive_message(psocket, &msg);
	zmq_msg_close(&msg);

	print_menu();

	cout << "Enter comand:\n";

	bool loop = true;
	while(loop) {
		cout << "=> ";
		cin >> comand;
		switch (comand) {
			case 1:
				// text.clear();
				printf("Enter text, type END to finish:\n");
				cin.ignore(1);
				do{
					cout << "> ";
					getline(cin, s, '\n');
					if (s != "END") {
						text.push_back(s);
					}
				} while (s != "END");
				break;
			case 2:
				if(!text.empty()){
					for (const auto& item : text) {
						AtoC.task = 1;
						send_message(mysocket, &AtoC);
						CtoA = receive_message(mysocket,
&msg);
						zmq_msg_close(&msg);
						s = item;
						send_text(mysocket, &s);

						AtoB.sym = item.length();
						AtoB.task = 1;
```

```
                                        send_message(psocket, &AtoB);

                                        BtoA = receive_message(psocket,
&msg);
                                        if (BtoA->status == 2) {
                                                zmq_msg_close(&msg);
                                                CtoA =
receive_message(mysocket, &msg);
                                                if (CtoA->status != 2) {
                                                        zmq_msg_close(&msg);
                                                        cout << "Error while
sending string" << endl;
                                                        return -1;
                                                }
                                        } else {
                                                cout << "Error: missing
message from B" << endl;
                                                return -1;
                                        }
                                }
                        } else cout << "---No text to send---" << endl;
                        break;
                case 3:
                        text.clear();
                        cout << "---Buffer cleared---" << endl;
                        break;
                case 0:
                        AtoC.task = 2;
                        AtoB.task = 2;
                        send_message(mysocket, &AtoC);
                        send_message(psocket, &AtoB);
                        loop = false;
                        break;
                }
        }
        zmq_close(psocket);
        zmq_close(mysocket);
        zmq_ctx_destroy(context);
        return 0;
}
```

B.cpp

```
#include
"ABC.h"

            string adr = "tcp://localhost:";

            int main(){
                    Message BtoC, *CtoB, BtoA, *AtoB;
                    zmq_msg_t msg;
                    BtoA = init_message();
                    BtoC = init_message();

                    void* context = zmq_ctx_new();
                    void* psocket = zmq_socket(context, ZMQ_REQ);
                    void* mysocket = zmq_socket(context, ZMQ_REP);
                    zmq_bind(mysocket, "tcp://*:5002");
                    zmq_connect(psocket, "tcp://localhost:5003");

                    BtoC.status = 1;
                    send_message(psocket, &BtoC);

                    AtoB = receive_message(mysocket, &msg);
                    if (AtoB->status == 1) cout << "B: A connected\n";
                    else {
                            cout << "Error: unsuccessful connection A to B";
                            return -1;
                    }
                    zmq_msg_close(&msg);

                    BtoA.status = 1;
                    send_message(mysocket, &BtoA);

                    bool loop = true;
                    while(loop){
                            AtoB = receive_message(mysocket, &msg);
                            switch(AtoB->task){
                                    case 1:
                                            cout << "---------------" << endl;
                                            cout << "A sent " << AtoB->sym << " symbols" <<
            endl;
                                            zmq_msg_close(&msg);
```

```
                                BtoA.status = 2;
                                send_message(mysocket, &BtoA);
                                CtoB = receive_message(psocket, &msg);
                                if(CtoB->task == 1){
                                        cout << "C recieved " << CtoB->sym << "
        symbols" << endl;
                                        cout << "---------------" << endl << endl;
                                        zmq_msg_close(&msg);
                                        BtoC.status = 2;
                                        send_message(psocket, &BtoC);
                                } else {
                                        cout << "Error: missing message from C";
                                        return -1;
                                }
                                break;
                        case 2:
                                loop = false;
                                break;
                        }
                }
                zmq_close(psocket);
                zmq_close(mysocket);
                zmq_ctx_destroy(context);
                return 0;
        }
```

C.cpp

```
#include
"ABC.h"

        string adr = "tcp://localhost:";

        int main() {
                Message CtoA, *AtoC, CtoB, *BtoC;
                string text;
                zmq_msg_t msg;
                CtoA = init_message();
                CtoB = init_message();
```

```
	void* context = zmq_ctx_new();
	void* psocket = zmq_socket(context, ZMQ_REQ);
	void* mysocket = zmq_socket(context, ZMQ_REP);
	zmq_bind(mysocket, "tcp://*:5003");
	zmq_connect(psocket, "tcp://localhost:5001");

	CtoA.status = 1;
	send_message(psocket, &CtoA);

	BtoC = receive_message(mysocket, &msg);
	if (BtoC->status == 1) cout << "C: B connected\n";
	else {
		cout << "Error: unsuccessful connection B to C";
		return -1;
	}
	zmq_msg_close(&msg);

	bool loop = true;
	while(loop){
		AtoC = receive_message(psocket, &msg);
		switch(AtoC->task){
			case 1:
			{
				zmq_msg_close(&msg);
				send_message(psocket, &CtoA);
				string recvd = receive_text(psocket, &msg);
				CtoB.sym = recvd.length();
				CtoB.task = 1;
				zmq_msg_close(&msg);
				send_message(mysocket, &CtoB);

				BtoC = receive_message(mysocket, &msg);
				// cout << "Recieved string: " << recvd << endl;
				if(!recvd.empty()) cout << "Recieved string: " <<
recvd << endl;
				else cout << "Empty line" << endl;

				if(BtoC->status == 2){
					zmq_msg_close(&msg);
					CtoA.status = 2;
					send_message(psocket, &CtoA);
				}
				break;
```

```
                        }
                        case 2:
                                loop = false;
                                break;
                }
        }
        zmq_close(psocket);
        zmq_close(mysocket);
        zmq_ctx_destroy(context);
        return 0;
}
```

4. Демонстрация работы программы

A.out

alex@alex-lenovo:~$ ./A.out

A: C connected

1 - enter text

2 - send text

0 – exit

Enter comand:

=> 1

Enter text, type END to finish:

> first line 111111111

> second line 222

> 33333 enil driht

> END

=> 2

=>

B.out

alex@alex-lenovo:~$ ./B.out

B: A connected

A sent 20 symbols

C recieved 20 symbols

A sent 15 symbols

C recieved 15 symbols

A sent 16 symbols

C recieved 16 symbols

C.out

alex@alex-lenovo:~$ ./C.out

C: B connected

Recieved string: first line 111111111

Recieved string: second line 222

Recieved string: 33333 enil driht

5. Вывод

С помощью библиотеки ZeroMQ удобно организовывать межпроцессорные связи. ZMQ-сокеты позволяют легко производить обмен сообщениями. Единственное неудобство, хотя и не особо

большое, использованной мной связи REP-REQ – это необходимость чередования запроса и ответа как на стороне сервера, так и клиента. Это в некоторых случаях вынуждает делать лишние отправки, что влияет на быстродействие программы.