Лабораторная работа № 6-8

по курсу Операционные системы:

Управлении серверами сообщений

Применение отложенных вычислений

Интеграция программных систем друг с другом

Выполнил студент группы М80-204Б МАИ Дюсекеев Алишер

Оценка_____

Лабораторные работы №6-8

Цель работы

Целью является приобретение практических навыков в:

- Управлении серверами сообщений (№6)
- Применение отложенных вычислений (№7)
- Интеграция программных систем друг с другом (№8)

Задание

Реализовать клиент-серверную систему по асинхронной обработке запросов. Необходимо составить программы сервера и клиента. При запуске сервер и клиент должны быть настраиваемы, то есть должна быть возможность поднятия на одной ЭВМ нескольких серверов по обработке данных и нескольких клиентов, которые к ним относятся. Все общение между процессами сервера и клиентов должно осуществляться через сервер сообщений.

Серверное приложение – банк. Клиентское приложение клиент банка. Клиент может отправить какую-то денежную сумму в банк на хранение. Клиент также может запросить из банка произвольную сумму. Клиенты могут посылать суммы на счета других клиентов. Запросить собственный счет. При снятии должна производиться проверка на то, что у клиента достаточно денег для снятия денежных средств. Идентификатор клиента задается во время запуска клиентского приложения, как и адрес банка. Считать, что идентификаторы при запуске клиентов будут уникальными.

Возможные сервера сообщений

1. ZeroMQ

Конфигурации для клиентов и серверов (Вариант №21)

Внутреннее хранилище сервера

2. Вектор

Тип ключа клиента

3. Double

Дополнительные возможности сервера

1. Сохранение данных о счетах клиентов при завершении работы сервера и возобновлении

Работа программы.

Программа состоит из клиентов и сервера.

Работа клиента:

```
typedef struct MD
{
	double clientId;
	int request;
	double who;
	int sum;
} MessageData;

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


	void* context = zmq_ctx_new();
	srand(time(0));
	double clientId;
	std::cin >> clientId;
	MessageData md;
	md.clientId = clientId;
	printf("Client %f Starting...\n", clientId);

	void* senderSocket = zmq_socket(context, ZMQ_REQ);
	zmq_connect(senderSocket, "tcp://localhost:4040");
	int request;
	std::cout << "{Menu}" << std::endl;
	std::cout << "1. Put on shore "<< std::endl;
	std::cout << "2. Take of shore" << std::endl;
	std::cout << "3. Check shore" << std::endl;
	std::cout << "4. Put on other user shore" << std::endl;
	std::cout << "0. Quit" << std::endl;
	printf("Input your request: ");
	std::cin >> request;
	std::cout << std::endl;

	while (request != 0)
	{
		md.request = request;
		if (request < 1 || request > 4) continue;
		if (request == 1) {
			printf("Input sum you want to put\n");
			std::cin >> md.sum;
		}
		if (request == 2) {
			printf("Input sum you want to take\n");
			std::cin >> md.sum;
		}
		if (request == 4) {
			printf("Input sum you want to give\n");
			std::cin >> md.sum;
			printf("Input userId\n");
			std::cin >> md.who;
		}

		zmq_msg_t zmqMessage;
		zmq_msg_init_size(&zmqMessage, sizeof(MessageData));
		memcpy(zmq_msg_data(&zmqMessage), &md, sizeof(MessageData));

		printf("Sending request\n");
		int send = zmq_msg_send(&zmqMessage, senderSocket, 0);
		zmq_msg_close(&zmqMessage);
```

```
                zmq_msg_t reply;
                zmq_msg_init(&reply);
                zmq_msg_recv(&reply, senderSocket, 0);
                size_t repSize = zmq_msg_size(&reply);
                printf("Received answer: - %s\n", zmq_msg_data(&reply));
                zmq_msg_close(&reply);

                printf("Input new request: ");
                std::cin >> request;
                std::cout << std::endl;
        }
        zmq_close(senderSocket);
        zmq_ctx_destroy(context);

        return 0;
}
```

Клиент подключается к сокету, созданному с помощью ZeroMQ. На сервер, во время его работы, клиент может отправить запросы на изменения в банковском счете. На выбор даются отправка на счет, снятие со счета, проверка счета и также можно положить деньги в другой аккаунт.

Работа сервера:

```
typedef struct ClientAccount {
        double clientId;
        int shore;
} Client;

typedef struct MD
{
        double clientId;
        int request;
        double who;
        int sum;
} MessageData;

void WorkWithclient(std::vector<ClientAccount *> *cv, ClientAccount *client, MessageData *m,
void* serverSocket) {
        zmq_msg_t reply;
        //zmq_msg_init_size(&reply, strlen("ok") + 1);
        //memcpy(zmq_msg_data(&reply), "ok\0", 3);
        //zmq_msg_send(&reply, serverSocket, 0);


        if (m->request == 1) {
                client->shore += m->sum;
                zmq_msg_init_size(&reply, strlen("Done.") + 1);
                memcpy(zmq_msg_data(&reply), "Done.", 6);
                zmq_msg_send(&reply, serverSocket, 0);
                zmq_msg_close(&reply);
                printf("Message sent 1\n");
                return;
        }
        if (m->request == 2) {
                if (client->shore >= m->sum) {
                        client->shore -= m->sum;
                        zmq_msg_init_size(&reply, strlen("Done.") + 1);
                        memcpy(zmq_msg_data(&reply), "Done.", 6);
                        zmq_msg_send(&reply, serverSocket, 0);
                        zmq_msg_close(&reply);
                        printf("Message sent 2.suc\n");
                        return;
                }
                else {
                        zmq_msg_init_size(&reply, strlen("There is not enough money to do such
procedure") + 1);
```

```
                        memcpy(zmq_msg_data(&reply), "There is not enough money to do such
procedure", strlen("There is not enough money to do such procedure") + 1);
                        zmq_msg_send(&reply, serverSocket, 0);
                        zmq_msg_close(&reply);
                        printf("Message sent 2.fail\n");
                        return;
                }
        }
        if (m->request == 3) {
                char str[20];
                sprintf_s(str, "%d", client->shore);
                zmq_msg_init_size(&reply, strlen(str) + 1);
                memcpy(zmq_msg_data(&reply), str, strlen(str) + 1);
                zmq_msg_send(&reply, serverSocket, 0);
                zmq_msg_close(&reply);
                printf("Message sent 3\n");
                return;
        }

        if (m->request == 4) {
                int founduser = 0;
                for (int i = 0; i < cv->size(); i++) {
                        if ((*cv)[i]->clientId == m->who) {
                                founduser = 1;
                                if (client->shore >= m->sum) {
                                        client->shore -= m->sum;
                                        (*cv)[i]->shore += m->sum;
                                        zmq_msg_init_size(&reply, strlen("Done.") + 1);
                                        memcpy(zmq_msg_data(&reply), "Done.\0", 6);
                                        zmq_msg_send(&reply, serverSocket, 0);
                                        zmq_msg_close(&reply);
                                        printf("Message sent 4.suc\n");
                                        return;
                                }
                                else {
                                        zmq_msg_init_size(&reply, strlen("There is not enough
money to do such procedure") + 1);
                                        memcpy(zmq_msg_data(&reply), "There is not enough money to
do such procedure\0", strlen("There is not enough money to do such procedure") +1);
                                        zmq_msg_send(&reply, serverSocket, 0);
                                        zmq_msg_close(&reply);
                                        printf("Message sent 4.fail\n");
                                        return;
                                }
                        }
                }
                if (!founduser) {
                        zmq_msg_init_size(&reply, strlen("User not found") + 1);
                        memcpy(zmq_msg_data(&reply), "User not found\0", strlen("User not found")
+ 1);
                        zmq_msg_send(&reply, serverSocket, 0);
                        zmq_msg_close(&reply);
                        printf("Message sent 4.fail2\n");
                        return;
                }
        }
        return;
}

int main(int argc, char const *argv[])
{
        std::vector<ClientAccount *> cv;
        std::ifstream input_file("C://tests//test_data.dat", std::ios::binary);

        input_file.seekg(0, input_file.end);
        int length = input_file.tellg();
        input_file.seekg(0, input_file.beg);

        int size = (int)(length / sizeof(ClientAccount));
```

```
printf("Size: %d %d %d\n", length, size, sizeof(ClientAccount));
ClientAccount saved_accounts;

input_file.read((char*)&saved_accounts, sizeof(ClientAccount) * size);
for (int k = 0; k < size; k++) {
	ClientAccount *account = &saved_accounts + k;
	cv.push_back(account);

}
for (int i = 0; i < cv.size(); i++) {
	printf("Id: %f Sum: %d\n", cv[i]->clientId, cv[i]->shore);
}

void* context = zmq_ctx_new();
void* serverSocket = zmq_socket(context, ZMQ_REP);
zmq_bind(serverSocket, "tcp://*:4040");
printf("Starting da bank\n");

for (;;)
{
	printf("*");
	Sleep(1000);
	zmq_msg_t message;
	zmq_msg_init(&message);
	zmq_msg_recv(&message, serverSocket, 0);
	MessageData *m = (MessageData *)zmq_msg_data(&message);
	printf("Message from client: %f  request: %d\n", m->clientId, m->request);
	if (m->clientId == -0.1) {
		break;
	}
	if (m->request < 1 || m->request > 4) {
		continue;
	}
	printf("Message from client: %f  request: %d\n", m->clientId, m->request);
	int clientbool = 1;
	for (int i = 0; i < cv.size(); i++) {
		if (cv[i]->clientId == m->clientId) {
			printf("Client exists %f %d\n", cv[i]->clientId, cv[i]->shore);
			WorkWithclient(&cv, cv[i], m, serverSocket);
			printf("Client done %f %d\n", cv[i]->clientId, cv[i]->shore);
			clientbool = 0;
			break;
		}
		else {
			clientbool = 1;
		}
	}
	if (clientbool)
	{
		ClientAccount *newclient = (ClientAccount*)malloc(sizeof(ClientAccount));
		newclient->clientId = m->clientId;
		newclient->shore = 0;
		WorkWithclient(&cv, newclient, m, serverSocket);
		cv.push_back(newclient);
		printf("Client created %f\n", newclient->clientId);
	}
	zmq_msg_close(&message);
	printf("#");
}
zmq_close(serverSocket);
zmq_ctx_destroy(context);
std::ofstream output_file("C://tests//test_data.dat", std::ios::binary);
for (int i = 0; i < cv.size(); i++) {
	ClientAccount acc = *cv[i];
	output_file.write((char*)&acc, sizeof(ClientAccount));
}
output_file.close();

return 0;
```

```
}
```

Во время запуска сервер проверяет, присутствует ли информация о клиенте в созданном файле и подгружает информацию в вектор, если таковая была найдена. Далее производится открытие сокета и ожидание подключения клиентов к нему. При получении запроса сервер начинает обрабатывать информацию о клиенте. Если это новый клиент, то он добавляется в вектор, а его счет становится 0. Если это клиент, который уже находится в векторе, то начинается обработка его запроса с выводом ответа обратно на сокет. При закрытии сервера элементы находящиеся в векторе записываются последовательно в файл.

Вывод работы программы.

Сервер:

```
Size: 80 5 16
Id: 1.000000 Sum: 75050
Id: 2.230000 Sum: 1
Id: 3.000000 Sum: 13883
Id: 4.230000 Sum: 320
Id: 1.110000 Sum: 25001
Starting da bank
*Message from client: 1.220000  request: 1
Message from client: 1.220000  request: 1
Message sent 1
Client created 1.220000
#*Message from client: 1.220000  request: 3
Message from client: 1.220000  request: 3
Client exists 1.220000 0
Message sent 3
Client done 1.220000 0
#*Message from client: 1.000000  request: 3
Message from client: 1.000000  request: 3
Client exists 1.000000 75050
Message sent 3
Client done 1.000000 75050
#*Message from client: 1.000000  request: 1
Message from client: 1.000000  request: 1
Client exists 1.000000 75050
Message sent 1
Client done 1.000000 80050
#*Message from client: 1.000000  request: 2
Message from client: 1.000000  request: 2
Client exists 1.000000 80050
Message sent 2.suc
Client done 1.000000 80000
#*Message from client: 1.000000  request: 3
Message from client: 1.000000  request: 3
Client exists 1.000000 80000
Message sent 3
Client done 1.000000 80000
#*Message from client: 1.000000  request: 4
Message from client: 1.000000  request: 4
Client exists 1.000000 80000
Message sent 4.suc
Client done 1.000000 65000
#*Message from client: 1.220000  request: 3
Message from client: 1.220000  request: 3
Client exists 1.220000 15000
Message sent 3
Client done 1.220000 15000
```

Клиент 1:

```
1
Client 1.000000 Starting...
{Menu}
1. Put on shore
2. Take of shore
3. Check shore
4. Put on other user shore
0. Quit
Input your request: 3

Sending request
Received answer: - 75050
Input new request: 1

Input sum you want to put
5000
Sending request
Received answer: - Done.
Input new request: 2

Input sum you want to take
50
Sending request
Received answer: - Done.
Input new request: 3

Sending request
Received answer: - 80000
Input new request: 4

Input sum you want to give
15000
Input userId
1.22
Sending request
Received answer: - Done.
```

Клиент 1.22:

```
1.22
Client 1.220000 Starting...
{Menu}
1. Put on shore
2. Take of shore
3. Check shore
4. Put on other user shore
0. Quit
Input your request: 1

Input sum you want to put
0
Sending request
Received answer: - Done.
Input new request: 3

Sending request
Received answer: - 0
Input new request: 3

Sending request
Received answer: - 15000
```

Вывод

В этой лабораторной работе я смог научиться работать с серверами сообщений и обмену информацией между двумя программами с помощью утилиты ZeroMQ. Передача данных через сокеты является очень удобным способом взаимодействия двух программ. Можно создавать поток информации, отправлять его на сокет и ждать пока другая программа обработает эту информацию. Также я вспомнил как нужно создавать файлы, чтобы можно было хранить информацию об объектах долговременно. Возможности передачи данных разными способами, такими как: FireAndForget, RquestReply, SubscribePublish делают систему крайне гибкой для создания программного обеспечения, способного работать между собой и с другими программами.