Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Операционные системы»

Студентка:	А. Довженко
Преподаватель:	Е. С. Миронов
Группа:	08-207
Вариант:	14
Дата:	
Оценка:	
Подпись:	

Москва, 2017

Лабораторная работа №6

Реализовать клиент-среверную 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 хранения. 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.

Вариант 14.
Сервер сообщений: ZeroMQ.
Внутреннее хранилище сервера: вектор.
Тип ключа клиента: целочисленный 32-битный тип.
Дополнительные возможности сервера: возможность временной приостановки работы сервера без выключения. Сообщения серверу можно отправлять, но ответы сервер не отправляет до возобновления работы.

1 Описание

Для реализации связи клиент-сервер был выбран паттерн RequestResponse. Клиент отправляет запрос на сервер и ждет ответа. После того, как ответ пришел, клиент может продолжать работу. Клиент подключается к серверу, производит проверку на наличие клиента в базе, затем работает с сервером. Сервер обрабатывает запросы клиента, смотрит его наличие и состояние баланса, в случае ошибки или недостатке средств посылает соответствующий ответ клиенту, также логгирует свои действия в стандартный поток вывода. На клиентской части реализован аккаунт администратора, который может останавливать и возобновлять работу сервера. Если работа сервера остановлена, то в ответ на запросы клиента, он отправляет пустые ответы. Как только работа сервера возобновлена, он продолжает обрабатывать новые приходящие запросы.

Системные вызовы:

void exit(int status); – функция выходит из процесса с заданным статусом.
int zmq connect(void *socket, const char *endpoint); – подключает socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.
int zmq bind(void *socket, const char *endpoint); – присоединяет socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.
void *zmq socket(void *context, int type); – создает сокет типа type из котнекста context.
int zmq msg send(zmq msg t *msg, void *socket, int flags); – отправляет сообщение msg в socket с параметрами flags, возвращает количество отправленных байт, в случае ошибки возвращает -1.
int zmq msg init(zmq msg t *msg) – инициализирует сообщение msg как пустой объект.
int zmq msg recv(zmq msg t *msg, void *socket, int flags); – получает сообщение из socket в msg с параметрами flags, возвращает количество полученных байт, в случае ошибки возвращает -1.
int zmq msg close(zmq msg t *msg) – очищает содержимое msg, аналог free для сообщений zmq, возвращает 0 в случае успеха и -1 в случае неудачи.
int zmq close(void *socket); – закрывает сокет socket, возвращает 0 в случае успеха и -1 в случае неудачи.
int zmq ctx destroy(void *context); – разрушает контекст context, блокирует доступ всем операциям кроме zmq close, все сообщения в сокетах либо физически отправлены, либо "висят".

2 Исходный код

3 message.h

```
#ifndef _MESSAGE_H_
#define _MESSAGE_H_

#include <inttypes.h>

#define STR_SIZE 256

typedef int32_t ID;

typedef struct _msg {
    ID client;
    int sum;
    int action;
    ID receiverClient;
    void *requester;
    char message[STR_SIZE];
    char fileName[STR_SIZE];
} MsgData;

#endif
```

4 client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include <string.h>
#include <pthread.h>

#include "zmq.h"
#include "message.h"

void menuAdmin()
{
    printf("1) Stop server \n");
    printf("2) Run server \n");
    printf("3) Exit \n");
}

void menuUser()
{
    printf("1) Put money into account\n");
    printf("2) Get money from account\n");
```

```
21      printf("3) Send money to another account\n");
22      printf("4) Check balance\n");
23      printf("5) Leave the bank\n");
24  }
25
26  void *SendRecv(void *arg)
27  {
28      MsgData *md = (MsgData *) arg;
29      zmq_msg_t message;
30      zmq_msg_init_size(&message, sizeof(MsgData));
31      memcpy(zmq_msg_data(&message), md, sizeof(MsgData));
32      zmq_msg_send(&message, md->requester, 0);
33      zmq_msg_close(&message);
34
35      zmq_msg_init(&message);
36      zmq_msg_recv(&message, md->requester, 0);
37      md = (MsgData *) zmq_msg_data(&message);
38      printf("%s\n", md->message);
39      zmq_msg_close(&message);
40      pthread_exit(NULL);
41      return 0;
42  }
43
44  int main(int argc, char **argv)
45  {
46      void *context = zmq_ctx_new();
47      int admin = 0;
48
49      ID client, bank;
50      if (argc == 2 && !strcmp(argv[1], "admin")) {
51          admin = 1;
52      } else {
53          printf("Enter client's login: ");
54          scanf("%d", &client);
55      }
56
57      char adress[25];
58      printf("Enter bank's adress: ");
59      scanf("%d", &bank);
60
61      sprintf(adress, "%s%d", "tcp://localhost:", bank);
62
63      printf("tcp://localhost:%d \n", bank);
64
65      void *sendSocket = zmq_socket(context, ZMQ_REQ);
66      zmq_connect(sendSocket, adress);
67
68      if (admin) {
69          int act = 0;
```

```
70          menuAdmin();
71          do {
72              scanf("%d", &act);
73              MsgData md;
74              md.action = act + 10;
75              switch (act) {
76                  case 1: {
77                      pthread_t th;
78                      md.requester = sendSocket;
79                      pthread_create(&th, NULL, SendRecv, &md);
80                      pthread_detach(th);
81                      break;
82                  }
83
84                  case 2: {
85                      pthread_t th;
86                      md.requester = sendSocket;
87                      pthread_create(&th, NULL, SendRecv, &md);
88                      pthread_detach(th);
89                      break;
90                  }
91
92                  case 3:
93                      break;
94
95                  default: {
96                      printf("Inccorect command\n");
97                      break;
98                  }
99              }
100         } while (act != 3);
101
102
103     } else {
104         int act = 0, sum = 0;
105         menuUser();
106         do {
107             scanf("%d", &act);
108
109             MsgData md;
110             md.action = act;
111             md.client = client;
112
113             switch (act) {
114                 case 1: {
115                     printf("Enter the sum: ");
116                     scanf("%d", &sum);
117
118                     md.sum = sum;
```

```
119                     pthread_t th;
120                     md.requester = sendSocket;
121                     pthread_create(&th, NULL, SendRecv, &md);
122                     pthread_detach(th);
123                     break;
124                 }
125 
126                 case 2: {
127                     printf("Enter the sum: ");
128                     scanf("%d", &sum);
129 
130                     md.sum = sum;
131                     pthread_t th;
132                     md.requester = sendSocket;
133                     pthread_create(&th, NULL, SendRecv, &md);
134                     pthread_detach(th);
135 
136                     break;
137                 }
138 
139                 case 3: {
140                     int receiverClient;
141                     printf("Enter receiver id: ");
142                     scanf("%d", &receiverClient);
143 
144                     printf("Enter the sum: ");
145                     scanf("%d", &sum);
146 
147                     md.sum = sum;
148                     md.receiverClient = receiverClient;
149 
150                     pthread_t th;
151                     md.requester = sendSocket;
152                     pthread_create(&th, NULL, SendRecv, &md);
153                     pthread_detach(th);
154 
155                     break;
156                 }
157 
158                 case 4: {
159                     pthread_t th;
160                     md.requester = sendSocket;
161                     pthread_create(&th, NULL, SendRecv, &md);
162                     pthread_detach(th);
163 
164                     break;
165                 }
166 
167                 case 5:
```

```
168                     break;
169
170                 default: {
171                     printf("Inccorect command\n");
172                     break;
173                 }
174             }
175         } while (act != 5);
176     }
177
178     zmq_close(sendSocket);
179     zmq_ctx_destroy(context);
180
181     return 0;
182 }
```

5 bank.h

```
1  #ifndef _BANK_H_
2  #define _BANK_H_
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <stdint.h>
7  #include <inttypes.h>
8
9  #define SUCCESS 1
10 #define NOT_MEMORY -1
11 #define NOT_ENOUGH_MONEY -2
12 #define NOT_CLIENT -3
13 #define RECEIVER_NOT_CLIENT -4
14
15 typedef int32_t ID;
16
17 typedef struct _client {
18     ID client;
19     int sum;
20 } *Client;
21
22 typedef struct _clientDB {
23     Client clients;
24     uint32_t size;
25     uint32_t freespace;
26 } *ClientDB;
27
28 ClientDB ClientDBCreate(void);
29 void ClientAdd(ClientDB cDB, ID client);
30 void ClientDBPrint(ClientDB cDB);
31 Client ClientFind(ClientDB cDB, int clientint);
```

```
32 void ClientDBDestroy(ClientDB *cDB);
33
34 void ClientAccIncrease(Client cl, int sum);
35 void ClientAccDecrease(Client cl, int sum);
36 int ClientHasEnoughMoney(Client cl, int sum);
37 void MoneyPut(int Clientint, int sum, ClientDB cDB);
38 int MoneyGet(int Clientint, int sum, ClientDB cDB);
39 int MoneySend(ID clientSender, ID clientReceiver, int sum, ClientDB cDB);
40 int CheckAccount(ID client, ClientDB cDB);
41
42 #endif
```

6 bank.c

```
1  #include "bank.h"
2
3  ClientDB ClientDBCreate(void)
4  {
5      ClientDB cDB = (ClientDB) malloc(sizeof(*cDB));
6      if (!cDB) {
7          fprintf(stderr, "ERROR: no memory\n");
8          exit(NOT_MEMORY);
9      }
10     cDB->clients = (Client) malloc(sizeof(*(cDB->clients)));
11     cDB->size = 0;
12     cDB->freespace = 1;
13     return cDB;
14 }
15
16 void DBResize(ClientDB cDB)
17 {
18     cDB->clients = realloc(cDB->clients, 2 * cDB->size * sizeof(*(cDB->clients)));
19     if (!cDB->clients) {
20         fprintf(stderr, "ERROR: no memory\n");
21         exit(NOT_MEMORY);
22     }
23     cDB->freespace = cDB->size;
24 }
25
26 void ClientAdd(ClientDB cDB, ID client)
27 {
28     if (!cDB->freespace) {
29         DBResize(cDB);
30     }
31
32     cDB->clients[cDB->size].client = client;
33     cDB->clients[cDB->size].sum = 0;
34     cDB->size++;
35     cDB->freespace--;
```

```
36 }
37
38 void ClientDBPrint(ClientDB cDB)
39 {
40     if (cDB) {
41         for (uint32_t i = 0; i < cDB->size; ++i) {
42             printf("ID: %d\t", cDB->clients[i].client);
43             printf("SUM: %d\n", cDB->clients[i].sum);
44         }
45     }
46 }
47
48 Client ClientFind(ClientDB cDB, int clientint)
49 {
50     if (cDB) {
51         for (uint32_t i = 0; i < cDB->size; ++i) {
52             if (cDB->clients[i].client == clientint) {
53                 return &(cDB->clients[i]);
54             }
55         }
56     }
57     return NULL;
58 }
59
60 void ClientDBDestroy(ClientDB *cDB)
61 {
62     free((*cDB)->clients);
63     (*cDB)->clients = NULL;
64     free(*cDB);
65     *cDB = NULL;
66 }
67
68 void ClientAccIncrease(Client cl, int sum)
69 {
70     cl->sum += sum;
71 }
72
73 void ClientAccDecrease(Client cl, int sum)
74 {
75     cl->sum -= sum;
76 }
77
78 int ClientHasEnoughMoney(Client cl, int sum)
79 {
80     return cl->sum >= sum;
81 }
82
83 void MoneyPut(int clientint, int sum, ClientDB cDB)
84 {
```

```
85      Client cl = ClientFind(cDB, clientint);
86
87      if (cl) {
88          ClientAccIncrease(cl, sum);
89      } else {
90          ClientAdd(cDB, clientint);
91          cl = ClientFind(cDB, clientint);
92          ClientAccIncrease(cl, sum);
93      }
94  }
95
96  int MoneyGet(int clientint, int sum, ClientDB cDB)
97  {
98      Client cl = ClientFind(cDB, clientint);
99      if (!cl) {
100         return NOT_CLIENT;
101     }
102     if (ClientHasEnoughMoney(cl, sum)) {
103         ClientAccDecrease(cl, sum);
104         return SUCCESS;
105     } else {
106         return NOT_ENOUGH_MONEY;
107     }
108 }
109
110 int MoneySend(ID clientSender, ID clientReceiver, int sum, ClientDB cDB)
111 {
112     Client clSender = ClientFind(cDB, clientSender);
113     if (!clSender) {
114         return NOT_CLIENT;
115     }
116     Client clReceiver = ClientFind(cDB, clientReceiver);
117     if (!clReceiver) {
118         return RECEIVER_NOT_CLIENT;
119     }
120
121     if (ClientHasEnoughMoney(clSender, sum)) {
122         ClientAccDecrease(clSender, sum);
123         ClientAccIncrease(clReceiver, sum);
124         return SUCCESS;
125     } else {
126         return NOT_ENOUGH_MONEY;
127     }
128 }
129
130 int CheckAccount(ID client, ClientDB cDB)
131 {
132     Client cl = ClientFind(cDB, client);
133     if (!cl) {
```

```
134         return NOT_CLIENT;
135     }
136     return cl->sum;
137 }
```

7 server.c

```
1  #include <string.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <time.h>
6  #include <unistd.h>
7
8  #include "zmq.h"
9  #include "bank.h"
10 #include "message.h"
11
12 int main(void)
13 {
14     int code = 0, blocking = 0;
15     ClientDB clientBase = ClientDBCreate();
16     void *context = zmq_ctx_new();
17     void *responsSocket = zmq_socket(context, ZMQ_REP);
18
19     char adress[25];
20     printf("Enter bank's adress: ");
21     ID bank;
22     scanf("%d", &bank);
23     sprintf(adress, "%s%d", "tcp://*:", bank);
24
25     zmq_bind(responsSocket, adress);
26
27     while (1) {
28         zmq_msg_t message;
29
30         zmq_msg_init(&message);
31         zmq_msg_recv(&message, responsSocket, 0);
32         MsgData *md = (MsgData *) zmq_msg_data(&message);
33         zmq_msg_close(&message);
34
35         char info[STR_SIZE];
36
37         switch (md->action) {
38             case 1: {
39                 if (blocking) {
40                     strcpy(info, "\0");
41                 } else {
42                     printf("Put money into the account id: %d\n", md->client);
```

```
43                  MoneyPut(md->client, md->sum, clientBase);
44                  ClientDBPrint(clientBase);
45                  strcpy(info, "Operation was completed successfully\0");
46              }
47
48              memcpy(md->message, info, strlen(info) + 1);
49              zmq_msg_init_size(&message, sizeof(MsgData));
50              memcpy(zmq_msg_data(&message), md, sizeof(MsgData));
51              zmq_msg_send(&message, responsSocket, 0);
52              zmq_msg_close(&message);
53
54              break;
55          }
56
57          case 2: {
58              if (blocking) {
59                  strcpy(info, "\0");
60              } else {
61                  printf("Get money from the account id: %d\n", md->client);
62
63                  code = MoneyGet(md->client, md->sum, clientBase);
64                  if (code == SUCCESS) {
65                      printf("Success\n");
66                      ClientDBPrint(clientBase);
67                      strcpy(info, "Operation was completed successfully\0");
68                  } else if (code == NOT_ENOUGH_MONEY) {
69                      printf("Not enough money\n");
70                      strcpy(info, "You not enough money\0");
71                  } else {
72                      printf("Not bank client\n");
73                      strcpy(info, "You aren't a client of bank\0");
74                  }
75              }
76
77              memcpy(md->message, info, strlen(info) + 1);
78              zmq_msg_init_size(&message, sizeof(MsgData));
79              memcpy(zmq_msg_data(&message), md, sizeof(MsgData));
80              zmq_msg_send(&message, responsSocket, 0);
81              zmq_msg_close(&message);
82
83              break;
84          }
85
86          case 3: {
87              if (blocking) {
88                  strcpy(info, "\0");
89              } else {
90                  printf("Send money from account id: %d to account id: %d\n", md->
                        client, md->receiverClient);
```

```
91                  code = MoneySend(md->client, md->receiverClient, md->sum, clientBase
                        );
92                  if (code == SUCCESS) {
93                      printf("Success\n");
94                      ClientDBPrint(clientBase);
95                      strcpy(info, "Operation was completed successfully\0");
96                  } else if (code == NOT_ENOUGH_MONEY) {
97                      printf("Not enought money\n");
98                      strcpy(info, "You not enough money");
99                  } else if (code == RECEIVER_NOT_CLIENT) {
100                     printf("Receiver not bank client\n");
101                     strcpy(info, "Receiver is not a client of bank\0");
102                 }
103             }
104
105             memcpy(md->message, info, strlen(info) + 1);
106             zmq_msg_init_size(&message, sizeof(MsgData));
107             memcpy(zmq_msg_data(&message), md, sizeof(MsgData));
108             zmq_msg_send(&message, responsSocket, 0);
109             zmq_msg_close(&message);
110             break;
111         }
112
113         case 4: {
114             if (blocking) {
115                 strcpy(info, "\0");
116             } else {
117                 printf("Check account id: %d\n", md->client);
118                 code = CheckAccount(md->client, clientBase);
119                 if (code == NOT_CLIENT) {
120                     printf("Not bank client\n");
121                     strcpy(info, "You aren't client of bank\0");
122                 } else {
123                     printf("Client sum is %d\n", code);
124                     ClientDBPrint(clientBase);
125                     sprintf(info, "%s%d%c", "Your account is ", code, '\0');
126                 }
127             }
128
129             memcpy(md->message, info, strlen(info) + 1);
130             zmq_msg_init_size(&message, sizeof(MsgData));
131             memcpy(zmq_msg_data(&message), md, sizeof(MsgData));
132             zmq_msg_send(&message, responsSocket, 0);
133             zmq_msg_close(&message);
134
135             break;
136         }
137
138         case 11: {
```

```
139                 printf("Block server\n");
140                 blocking = 1;
141                 strcpy(info, "Server is blocked\0");
142                 md->requester = responsSocket;
143                 memcpy(md->message, info, strlen(info) + 1);
144                 zmq_msg_init_size(&message, sizeof(MsgData));
145                 memcpy(zmq_msg_data(&message), md, sizeof(MsgData));
146                 zmq_msg_send(&message, responsSocket, 0);
147                 zmq_msg_close(&message);
148                 break;
149             }
150 
151             case 12: {
152                 printf("Unblock server\n");
153                 blocking = 0;
154                 strcpy(info, "Server is unblocked\0");
155                 md->requester = responsSocket;
156                 memcpy(md->message, info, strlen(info) + 1);
157                 zmq_msg_init_size(&message, sizeof(MsgData));
158                 memcpy(zmq_msg_data(&message), md, sizeof(MsgData));
159                 zmq_msg_send(&message, responsSocket, 0);
160                 zmq_msg_close(&message);
161                 break;
162             }
163 
164         }
165         zmq_msg_close(&message);
166 
167     }
168     zmq_close(responsSocket);
169     zmq_ctx_destroy(context);
170 
171     ClientDBDestroy(&clientBase);
172 
173 
174 }
```

8 Тестирование

- Тестирование связи один ко многим.

Некорректные запросы

```
server1:
karma@karma:~/mai_study/OS/lab6$ ./server
Enter bank's adress: 4040

server2:
karma@karma:~/mai_study/OS/lab6$ ./server
Enter bank's adress: 4041


server3:
karma@karma:~/mai_study/OS/lab6$ ./server
Enter bank's adress: 4042

client:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 666
Enter bank's adress: 4040
tcp://localhost:4040
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank
2
Enter the sum: 100
You aren't a client of bank
3
Enter receiver id: 10
Enter the sum: 100
You aren't a client of bank
4
```

```
You aren't client of bank

server1:
Get money from the account id: 666
Not bank client
Send money from account id: 666 to account id: 10
Check account id: 666
Not bank client
```

Положить деньги на счет

```
client:
1
Enter the sum: 100
Operation was completed successfully

server1:
Put money into the account id: 666
ID: 666 SUM: 100
```

Снять деньги со счета

```
client:
2
Enter the sum: 20
Operation was completed successfully

server1:
Get money from the account id: 666
Success
ID: 666 SUM: 80
```

Проверка баланса
Снять деньги со счета

```
client:
4
Your account is 80
```

```
server1:
Check account id: 666
Client sum is 80
ID: 666 SUM: 80
```

Отправить деньги клиенту, которого нет

```
client:
3
Enter receiver id: 9
Enter the sum: 10
Receiver is not a client of bank

server1:
Send money from account id: 666 to account id: 9
Receiver not bank client
```

Выйти из банка и обратиться к другому

```
client:
5
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 666
Enter bank's adress: 4041
tcp://localhost:4041
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank
4
You aren't client of bank

server2:
Check account id: 666
Not bank client
```

- Тестирование связи многие к одному

```
server:
karma@karma:~/mai_study/OS/lab6$ ./server
Enter bank's adress: 4040

client1:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 1
Enter bank's adress: 4040
tcp://localhost:4040
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank

client2:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 2
Enter bank's adress: 4040
tcp://localhost:4040
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank
```

Первый и второй клиент кладут деньги

```
client1:
1
Enter the sum: 100
Operation was completed successfully

client2:
1
Enter the sum: 200
Operation was completed successfully

server:
```

```
Put money into the account id: 1
ID: 1   SUM: 100
Put money into the account id: 2
ID: 1   SUM: 100
ID: 2   SUM: 200
```

Отослать больше денег, чем есть на счете

```
client1:
3
Enter receiver id: 2
Enter the sum: 200
You not enough money

server:
Send money from account id: 1 to account id: 2
Not enought money
```

Отослать корректную сумму денег

```
client1:
4
Your account is 150

client2:
3
Enter receiver id: 1
Enter the sum: 50
Operation was completed successfully
4
Your account is 150

server:
Send money from account id: 2 to account id: 1
Success
ID: 1   SUM: 150
ID: 2   SUM: 150
Check account id: 2
Client sum is 150
```

```
ID: 1   SUM: 150
ID: 2   SUM: 150
Check account id: 1
Client sum is 150
ID: 1   SUM: 150
ID: 2   SUM: 150
```

Снять деньги у одного из клиентов

```
client1:
2
Enter the sum: 100
Operation was completed successfully

server:
Get money from the account id: 1
Success
ID: 1   SUM: 50
ID: 2   SUM: 150
```

- Тестирование связи многие ко многим

Создаем 2 сервера и 4 клиента

```
server1:
karma@karma:~/mai_study/OS/lab6$ ./server
Enter bank's adress: 4040

server2:
karma@karma:~/mai_study/OS/lab6$ ./server
Enter bank's adress: 4041

client1:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 1
Enter bank's adress: 4040
tcp://localhost:4040
1) Put money into account
2) Get money from account
```

```
3) Send money to another account
4) Check balance
5) Leave the bank

client2:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 2
Enter bank's adress: 4040
tcp://localhost:4040
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank

client3:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 11
Enter bank's adress: 4041
tcp://localhost:4041
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank

client4:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 12
Enter bank's adress: 4041
tcp://localhost:4041
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank
```

Добавим деньги на каждый счет

```
client1:
```

```
1
Enter the sum: 100
Operation was completed successfully

client2:
1
Enter the sum: 100
Operation was completed successfully

client3:
1
Enter the sum: 100
Operation was completed successfully

client4:
1
Enter the sum: 100
Operation was completed successfully

server1:
Put money into the account id: 1
ID: 1   SUM: 100
Put money into the account id: 2
ID: 1   SUM: 100
ID: 2   SUM: 100

server2:
Put money into the account id: 11
ID: 11  SUM: 100
Put money into the account id: 12
ID: 11  SUM: 100
ID: 12  SUM: 100
```

Пересылка между банками

```
client1:
3
Enter receiver id: 11
Enter the sum: 50
```

```
Receiver is not a client of bank

server1:
Send money from account id: 1 to account id: 11
Receiver not bank client
```

Пересылка в одном банке

```
client1:
3
Enter receiver id: 2
Enter the sum: 50
Operation was completed successfully


client3:
3
Enter receiver id: 12
Enter the sum: 50
Operation was completed successfully

server1:
Send money from account id: 1 to account id: 2
Success
ID: 1   SUM: 50
ID: 2   SUM: 150

server2:
Send money from account id: 11 to account id: 12
Success
ID: 11  SUM: 50
ID: 12  SUM: 150
```

- Тестирование остановки работы сервера

Запускаем сервер, клиент и клиент администратора

```
server:
karma@karma:~/mai_study/OS/lab6$ ./server
Enter bank's adress: 4040

client:
karma@karma:~/mai_study/OS/lab6$ ./client
Enter client's login: 1
Enter bank's adress: 4040
tcp://localhost:4040
1) Put money into account
2) Get money from account
3) Send money to another account
4) Check balance
5) Leave the bank

client admin:
karma@karma:~/mai_study/OS/lab6$ ./client admin
Enter bank's adress: 4040
tcp://localhost:4040
1) Stop server
2) Run server
3) Exit
```

Положим деньги клиенту

```
client:
1
Enter the sum: 600
Operation was completed successfully

server:
Put money into the account id: 1
ID: 1   SUM: 600
```

Заблокируем сервер

```
client admin:
1
Server is blocked
```

```
server:
Block server
```

Отправляем запросы от клиента

```
client:
2
Enter the sum: 100

1
Enter the sum: 300

4
```

Разблокируем сервер

```
client admin:
2
Server is unblocked

server:
Unblock server
```

Отправляем запросы от клиента

```
client:
2
Enter the sum: 100
Operation was completed successfully
4
Your account is 500

server:
Get money from the account id: 1
Success
ID: 1   SUM: 500
```

```
Check account id: 1
Client sum is 500
ID: 1   SUM: 500
```

9 Выводы

Это была интересная лабораторная, потому что у нас была относительная свобода в выборе архитектурных решений. Это всегда лучше, чем реализовывать какие-то системы по заранее заданному плану. Благодаря работе с сервером сообщений, я узнала о многих паттернах передачи сообщений, что несомненно поможет мне, если я захочу разработать приложение, в котором есть хоть какой-то обмен сообщениями.

Из недостатков моей работы можно отметить псведоасинхронность. Она достигается за счет средств библиотеки для работы с потоками языка Си. Каждая отправка сообщения и принятия ответа на клиенте выделяется в отдельный поток. Но т.к. клиент не может отсылать новые запросы, пока не получен ответ на старый, то полезность таких потоков стремится к нулю. Грамотнее было бы реализовывать асинхронность не на клиентской части, а в качестве "прослойки"между клиентами и серверами. Можно использовать паттерн, при котором между клиентами и серверами, работающими по паттерну RequestResponse, находился бы брокер, разделенный на фронтенд, принимающий запросы от клиентов и отправляющий соответствующие ответы , и бэкенд, обрабатывающй поток запросов, отсылающий их на сервера и получающий ответ с серверов. Для такого диалога можно было бы воспользоваться соектами DEALER и ROUTER, т.к. мы хотим отправлять множество ответов.

Моя так называемая база данных скорее всего упадет на больших данных, например на миллионе пользователей. Т.к. вариантом задана структура вектора с идентификаторами 32 битными целыми числами, можно было бы сразу создавать вектор, способный вместить максимальное количество пользователей. Это позволило бы осуществлять поиск в моей БД за константу. Если бы не существовало ограничений на хранилище, то я бы воспользовалась B-tree, что позволило бы мне сильно сэкономить по памяти и времени. Не очень хорошо хранить всю БД в оперативной памяти, лучше было бы подгружать ее только при соответствующих запросах и затем обновлять.

При построении серверов, которые сохраняют состояние клиентов, возникают классические проблемы. Если сервер хранит состояние каждого клиента, а клиентов подключается все больше и больше, в конце концов наступает момент исчерпания ресурсов. Даже если одни и те же клиенты сохраняют коннект и используется идентификация по умолчанию, то каждое соединение будет выглядить как новое.

Думаю, мою программу в том виде, в котором она существует сейчас, лучше нигде не использовать. Она требует серьезных улучшений. Но как первый опыт она вполне себе хороша.