Лабораторная работа № 6 по курсу : Операционные системы

Выполнил студент группы М8О-206Б-17 МАИ *Новиков Павел Сергеевич.*

Цель работы

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.

Задание

Вариант 13.

- Сервер сообщений: ZeroMQ.
- Внутреннее хранилище сервера: линейный список
- Тип ключа клиента: вещественный.
- Дополнительные возможности сервера: возможность временной приостановки работы сервера без выключения. Сообщения серверу можно отправлять, но ответы сервер не отправляет до возобновления работы.

Информация

Для реализации связи клиент-сервер был выбран паттерн Request-Response. Клиент отправляет запрос на сервер и ждёт 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.
Как только работа сервера возобновлена, он продолжает обрабатывать новые приходящие запросы.

Системные вызовы:

1. **void** ∗zmq_ctx_new (); – создаёт новый контекст, возвращает его адрес, в случае неудачи вернётся $NULL$

2. **void** ∗zmq_socket(**void** ∗context, **int** type); – создает сокет типа type из котнекста context.

3. **int** zmq_connect(**void** ∗socket, **const char** ∗endpoint); – подключает socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.

4. **int** zmq_bind(**void** ∗socket, **const char** ∗endpoint); – присоединяет socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.

5. **int** zmq_msg_init(zmq_msg_t ∗msg); – инициализирует сообщение msg как пустой объект.

6. **int** zmq_msg_init_size (zmq_msg_t ∗msg, size_t size); – инициализирует сообщение msg для отправки, устанавливает размер сообщения в size байт. При успехе возвращет 0, при неудаче -1

7. **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.

8. **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.

9. **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.

10. **int** zmq_close(**void** ∗socket); – закрывает сокет socket, возвращает 0 в случае успеха и -1 в случае неудачи.

11. **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".

Описание программы

Для простоты обмена сообщениями данные передавались только с помощью одной структуры:

```
typedef struct MD
{
        Action act;
        double clientId;
        size_t amount;
        double recievId;

} MessageData;
```

Код сервера:

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <zmq.h>

#include "TList.h"
#include "bank.h"

volatile sig_atomic_t flag = 0;

void block_func(int sig) {
    if (!flag) {
        flag = 1;
    } else {
        exit(0);
    }
}

void unblock_func(int sig) { flag = 0; }

int main(int argc, char const* argv[]) {
    const char* NoSuch = "No such account\0";
    const char* Done = "Done!\0";
    const char* Insuf = "Insufficient Funds\0";

    char repl_balance[50];
    El *client, *reciever;
    int amount;
    size_t size;

    char connect[15];
    size_t bankID;
    printf("Enter Bank ID: ");
    while (1) {
        scanf("%lu", &bankID);
        if (bankID > 49151 || bankID < 1024)
            printf("Invalid ID\n");
        else
            break;
    }
    sprintf(connect, "tcp://*:%lu", bankID);

    void* context = zmq_ctx_new();
    void* serverSocket = zmq_socket(context, ZMQ_REP);
```

```
zmq_bind(serverSocket, connect);
printf("Starting...\n");
TList* list = l_create();

signal(SIGINT, block_func);
signal(SIGTSTP, unblock_func);

for (;;) {
    if (!flag) {
        zmq_msg_t message;
        zmq_msg_init(&message);
        if (-1 != zmq_msg_recv(&message, serverSocket, 0)) {
            MessageData* m = (MessageData*)zmq_msg_data(&message);
            Action act = m->act;
            zmq_msg_close(&message);
            zmq_msg_t reply;
            printf("Message from client: ");
            if (act == BALANCE) {
                printf("Check Balance\n");
                if ((client = l_search(list, m->clientId)) == NULL) {
                    size = strlen(NoSuch) + 1;
                    zmq_msg_init_size(&reply, size);
                    printf("%s %lu\n", NoSuch, size);
                    memcpy(zmq_msg_data(&reply), NoSuch, size);
                } else {
                    sprintf(repl_balance, "Balance: %d", balance(client));
                    size = strlen(repl_balance) + 1;
                    zmq_msg_init_size(&reply, size);
                    memcpy(zmq_msg_data(&reply), &repl_balance, size);
                    printf("%s\n", repl_balance);
                }
            } else if (act == PUT) {
                printf("Put money\n");
                if ((client = l_search(list, m->clientId)) == NULL) {
                    client = l_push(list, m->clientId);
                }
                client->acc.amount += m->amount;
                size = strlen(Done) + 1;
                zmq_msg_init_size(&reply, size);
                memcpy(zmq_msg_data(&reply), Done, size);
            } else if (act == WITHDRAW) {
                printf("Withdraw money\n");
                if ((client = l_search(list, m->clientId)) == NULL) {
                    size = strlen(NoSuch) + 1;
                    zmq_msg_init_size(&reply, size);
                    printf("%s %lu\n", NoSuch, size);
                    memcpy(zmq_msg_data(&reply), NoSuch, size);
                } else if (m->amount > client->acc.amount) {
                    size = strlen(Insuf) + 1;
                    zmq_msg_init_size(&reply, size);
                    printf("%s\n", Insuf);
                    memcpy(zmq_msg_data(&reply), Insuf, size);
                } else {
                    client->acc.amount -= m->amount;
                    size = strlen(Done) + 1;
                    zmq_msg_init_size(&reply, size);
                    memcpy(zmq_msg_data(&reply), Done, size);
                }
            } else if (act == SEND) {
                printf("Send money\n");
                if ((client = l_search(list, m->clientId)) == NULL ||
                    (reciever = l_search(list, m->recievId)) == NULL) {
                    size = strlen(NoSuch) + 1;
                    zmq_msg_init_size(&reply, size);
                    printf("%s %lu\n", NoSuch, size);
```

```
                    memcpy(zmq_msg_data(&reply), NoSuch, size);
                } else if (m->amount > client->acc.amount) {
                    size = strlen(Insuf) + 1;
                    zmq_msg_init_size(&reply, size);
                    printf("%s\n", Insuf);
                    memcpy(zmq_msg_data(&reply), Insuf, size);
                } else {
                    client->acc.amount -= m->amount;
                    reciever->acc.amount += m->amount;
                    size = strlen(Done) + 1;
                    zmq_msg_init_size(&reply, size);
                    memcpy(zmq_msg_data(&reply), Done, size);
                }
            }

            zmq_msg_send(&reply, serverSocket, 0);
            zmq_msg_close(&reply);
        }
    }
}

l_destroy(list);
zmq_close(serverSocket);
zmq_ctx_destroy(context);
return 0;
}
```

Вывод Strace

```
Enter Bank ID: 1228
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
Welcome!
Enter your ID:
5
Choose the action:
b - Check account balance
w - Withdraw money from account
p - Put money into account
s - Send money to another account
e - exit
b
poll([{fd=8, events=POLLIN}], 1, 0)      = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
BANK: No such account
p
Amount: 5
```

```
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
BANK: Done!
b
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)

Enter Bank ID: 1228
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
Starting...
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
Message from client: Check Balance
No such account 16
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
Message from client: Put money
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1)     = 1 ([{fd=8, revents=POLLIN}])
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
Message from client: Check Balance
Balance: 5
poll([{fd=8, events=POLLIN}], 1, 0)      = 0 (Timeout)
poll([{fd=8, events=POLLIN}], 1, -1
```

Выводы

Библиотеки передачи сообщений вроде ZMQ, могут быть полезными для более высокоуровневой работы с сокетами. В ней представлены различные типы сокетов для реализация разных паттернов передачи сообщений, а так же обработка таких вещей как сохранение поступающий запросов в очередь для последующей обработки С помощью подобных библиотек можно создавать простые клиент-серверные приложение, не уверен что такой подход подойдет для разработки высоконагруженых систем.