Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

**“Клиент-серверная система для передачи мгновенных сообщений”**

Студент: Чекменев Вячеслав Алексеевич

Группа: М8О-207Б-20

Вариант: 30

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Исходный код
5. Демонстрация работы программы
6. Выводы

**Постановка задачи**

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Базовый функционал должен быть следующим:

• Клиент может присоединиться к серверу, введя логин

• Клиент может отправить сообщение другому клиенту по его логину

• Клиент в реальном времени принимает сообщения от других клиентов

30. Необходимо предусмотреть возможность отправки отложенного сообщения или себе, или другому пользователю. При выходе с сервера отправка всё равно должна осуществиться. Связь между сервером и клиентом должна быть реализована при помощи memory map (заменено на ZeroMQ)

**Общие сведения о программе**

Программа состоит из трёх файлов – server.c, client.c, funcs.c, в которых расположены код сервера, код клиента, реализация вспомогательный функций и структур соответственно. Для удобства также был создан Makefile.

**Общий метод и алгоритм решения**

Общение между клиентом и сервером осуществляется по схеме:

- сервер принимает сообщения от клиентов в виде push-pull паттерна, потом направляет их в pub-sub

- клиент отсылает сообщение серверу в виде push-pull паттерна, а в отдельном потоке принимает сообщения от сервера в виде pub-sub паттерна

Для начала необходимо запустить сервер и «зарегистрировать» пользователей (ввести список допустимых логинов, которые сервер сохранит для дальнейшего использования в файл logins.txt).

После запускаются клиенты, там они отправляют и принимают сооющения

Отложенное сообщение работает по принципу:

- клиент вводит кол-во секунд «задержки»

- другой клиент в своем отдельном потоке принимает это кол-во секунд и делает sleep(<кол-во секунд>), то есть он может в это время параллельно отсыдать и принимать сообщения

**Исходный код**

**funcs.c**

#include <zmq.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/time.h>

#include <assert.h>

#define SIZE 256

typedef struct message\_

{

char sender\_login[SIZE];

char recipient\_login[SIZE];

char msg\_txt[SIZE];

int delay;

} MessageData\_t;

typedef struct logins\_

{

char logins[SIZE][SIZE];

short count;

} Logins\_t;

void receive\_struct(void \*socket, MessageData\_t \*md)

{

zmq\_recv(socket, md->sender\_login, sizeof(md->sender\_login), 0);

zmq\_recv(socket, md->recipient\_login, sizeof(md->recipient\_login), 0);

zmq\_recv(socket, md->msg\_txt, sizeof(md->msg\_txt), 0);

char delay\_str[SIZE];

zmq\_recv(socket, delay\_str, sizeof(delay\_str), 0);

sscanf(delay\_str, "%d", &md->delay);

}

void send\_struct(void \*socket, MessageData\_t md)

{

zmq\_send(socket, md.sender\_login, sizeof(md.sender\_login), 0);

zmq\_send(socket, md.recipient\_login, sizeof(md.recipient\_login), 0);

zmq\_send(socket, md.msg\_txt, sizeof(md.msg\_txt), 0);

char delay\_str[SIZE];

sprintf(delay\_str, "%d", md.delay);

zmq\_send(socket, delay\_str, sizeof(delay\_str), 0);

}

**server.c**

#include "funcs.h"

int main()

{

printf("Enter logins: \n");

char logins[SIZE][SIZE];

FILE \*fptr\_logins;

fptr\_logins = fopen("logins.txt", "w");

if (fptr\_logins == NULL) {

printf("Error!");

exit(1);

}

int i = 0;

while (1) {

scanf("%s", logins[i]);

if (!strcmp(logins[i], "end"))

break;

i++;

}

const short number\_of\_users = i;

printf("Users:\n");

for (int i = 0; i < number\_of\_users; ++i) {

fprintf(fptr\_logins, "%s\n", logins[i]);

}

fprintf(fptr\_logins, "%s\n", "end");

fclose(fptr\_logins);

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

void \*requester = zmq\_socket(context, ZMQ\_PULL);

char conn\_pull[] = "tcp://127.0.0.1:5558";

int rc = zmq\_bind(requester, conn\_pull);

assert (rc == 0);

void \*publisher = zmq\_socket(context, ZMQ\_PUB);

char conn\_pub[] = "tcp://127.0.0.1:5555";

rc = zmq\_bind(publisher, conn\_pub);

assert (rc == 0);

MessageData\_t buf;

printf("started...\n");

while(1)

{

receive\_struct(requester, &buf);

send\_struct(publisher, buf);

}

zmq\_close(requester);

zmq\_term(context);

remove("logins.txt");

return 0;

}

**client.c**

#include "funcs.h"

typedef struct args\_thrd\_

{

void \*socket;

char login[SIZE];

} Args\_thrd\_t;

short message\_accepted = 0;

void \*accept\_message(Args\_thrd\_t \*args)

{

while (1) {

MessageData\_t buf;

receive\_struct(args->socket, &buf);

if (!strcmp(buf.recipient\_login, args->login)) {

sleep(buf.delay);

printf("\n<%s> message from %s: %s\n", args->login, buf.sender\_login, buf.msg\_txt);

fflush(stdout);

message\_accepted++;

printf("<%s> ", args->login);

fflush(stdout);

}

}

}

int in(char \*login, Logins\_t logins)

{

for (int i = 0; i < logins.count; ++i) {

if (!strcmp(login, logins.logins[i]))

return 1;

}

return 0;

}

int main()

{

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

void \*requester = zmq\_socket(context, ZMQ\_PUSH);

char conn\_push[] = "tcp://127.0.0.1:5558";

int rc = zmq\_connect(requester, conn\_push);

assert (rc == 0);

void \*subscriber = zmq\_socket (context, ZMQ\_SUB);

char conn\_sub[] = "tcp://127.0.0.1:5555";

rc = zmq\_connect (subscriber, conn\_sub);

assert (rc == 0);

rc = zmq\_setsockopt(subscriber, ZMQ\_SUBSCRIBE, "", 0);

assert (rc == 0);

FILE \*fptr\_logins;

fptr\_logins = fopen("logins.txt", "r");

if (fptr\_logins == NULL) {

printf("Error!");

exit(1);

}

printf("Hello, mate!\nPlease enter your login: ");

//fflush(stdout);

char login[SIZE];

scanf("%s", login);

short i = 0;

short counter = 1;

char check\_login[SIZE];

while (1) {

fscanf(fptr\_logins, "%s", check\_login);

if (!strcmp(check\_login, login)) {

printf("Hello, %s!\n", check\_login);

break;

}

if (!strcmp(check\_login, "end")) {

printf("Login %s not found, try again: \n", login);

if (counter <= 5) {

scanf("%s", login);

rewind(fptr\_logins);

counter++;

continue;

} else {

printf("number of tries has exceeded!\n");

return 0;

}

}

i++;

}

fclose(fptr\_logins);

MessageData\_t buf;

printf("started...\n");

pthread\_t thrd;

Args\_thrd\_t args;

strcpy(args.login, login);

args.socket = subscriber;

pthread\_create(&thrd, NULL, accept\_message, &args);

while(1)

{

int delay;

printf("<%s>\nrecipient: ", login);

fflush(stdout);

strcpy(buf.sender\_login, login);

scanf("%s", buf.recipient\_login);

printf("message: ");

scanf("%s", buf.msg\_txt);

printf("delay: ");

scanf("%d", &buf.delay);

send\_struct(requester, buf);

}

pthread\_detach(thrd);

zmq\_close(requester);

zmq\_close(subscriber);

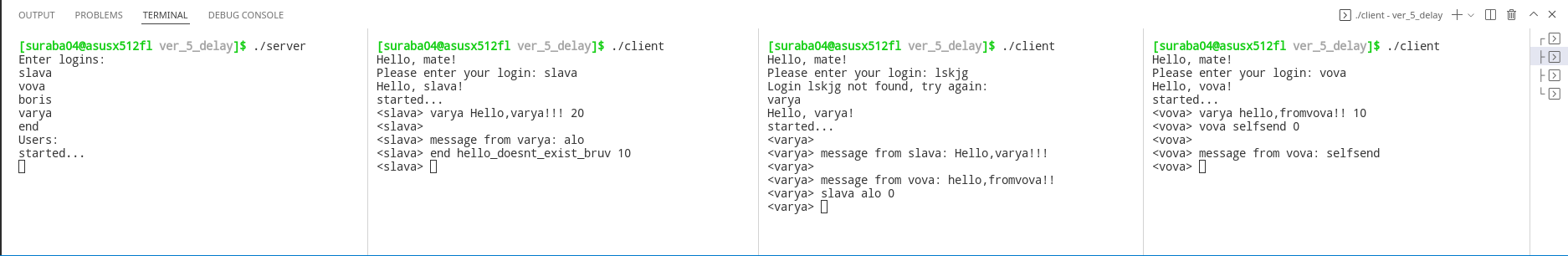
zmq\_term(context);

return 0;

}

**Демонстрация работы программы**

Регистрация пользователей четырех человек на сервере + отправка разных сообщений:



**Выводы**Данный курсовой проект оказался интересеным и полезным. Научился проектировать простые схемы, используя стандратные паттерны zmq. Понравилось строить схему общения между сервером и клиентами.

Местами программа работает откровенно небезопасно и неэффективно, то есть ее легко можно сломать, однако в силу того, что проект тренировочный, я не старался проработать все нюансы.