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

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;
```

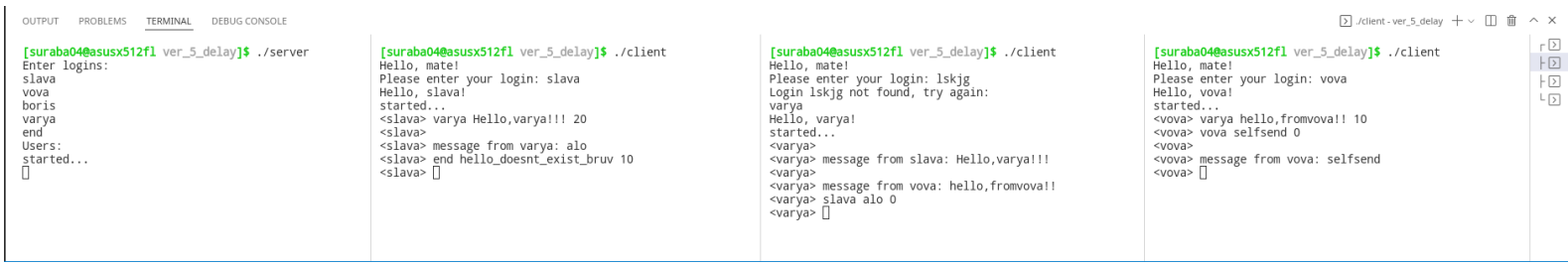
```
short message_accepted = 0;

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);
7
```

```
}

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.