Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Ткаченко Егор Юрьевич

Группа: М8О-207Б-21

Вариант: 24

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

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

**Репозиторий**

[https://github.com/Tnirpps/OS\_lab](https://vk.com/away.php?to=https%3A%2F%2Fgithub.com%2FTnirpps%2FOS_lab&cc_key=)

**Постановка задачи**

**Цель работы**

1. Приобретение практических навыков в использовании знаний, полученных в течении курса.
2. Проведение исследования в выбранной предметной области.

**Задание**

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

Клиент-серверная система для передачи мгновенных сообщений. 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; необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи memory map.

**Общие сведения о программе**

Программа распределительного узла компилируется из файла main.c, программа клиентского узла компилируется из файла user.c. В программе используется библиотека для работы с shared memory и mutex. В программе используются следующие системные вызовы:

1. shm\_open() – создаёт/открывает объект общей памяти.
2. shm\_unlink() – обратная к shm\_open().
3. ftruncate() – устанавливает файлу заданную длину в байтах.
4. close() – закрывает файловый декриптор.
5. pthread\_mutex\_lock() – блокирует мьютекс.
6. pthread\_mutex\_unlock() – разблокирует мьютекс.
7. pthread\_cond\_wait() – открывает мьютекс и ждёт изменения состояния переменной условия.
8. pthread\_cond\_signal() – посылает сигнал на разблокировку.

**Общий метод и алгоритм решения**

Для реализации поставленной задачи необходимо:

1. Реализовать способ общения между процессами

2. Наладить блокировку процессов, чтобы избежать «race condition»

3. Добавить возможность создания групповых чатов.

**Исходный код**

================================== main.c ==================================

#include <fcntl.h>

#include <pthread.h>

#include <stdio.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#include <assert.h>

#include <malloc.h>

#define COLOR\_WHITE\_BOLD\_UNDERLINED "\e[1;04m"

#define COLOR\_GREEN "\033[1;32m"

#define COLOR\_ORANGE "\033[1;33m"

#define COLOR\_MAGENTA "\033[1;35m"

#define COLOR\_OFF "\e[m"

const char\* COLORS\_PRINT[6] = {COLOR\_OFF, "", COLOR\_WHITE\_BOLD\_UNDERLINED, COLOR\_ORANGE, COLOR\_MAGENTA, COLOR\_GREEN};

const char\* SHARED\_SERVER\_NAME = "shared\_SeRvEr";

const char\* SHARED\_MUTEX\_NAME = "shared\_mutex";

const char\* SHARED\_COND\_NAME = "shared\_cond";

const int MAX\_U\_COUNT = 100;

#define check\_ok(VALUE, OK\_VAL, MSG) if (VALUE != OK\_VAL) { printf("%s", MSG); return 1; }

#define check\_wrong(VALUE, WRONG\_VAL, MSG) if (VALUE == WRONG\_VAL) { printf("%s", MSG); return 1; }

// TODO: create structure for msg. Now it looks terrible

const int MSG\_LEN\_LIMIT = 128;

const int NICK\_MAX\_LEN = 8;

const char SERVER = 1;

const char EXE = 2;

const int SHARED\_MEMORY\_SIZE = 1 + 2 \* NICK\_MAX\_LEN + 1 + MSG\_LEN\_LIMIT;

#define LEN 8

#define COUNT 100

typedef struct {

char name[LEN];

int connected [COUNT];

} chat;

void print(const char\* msg, unsigned color) {

if (color >= 6) {

printf("%s", msg);

} else {

printf("%s", COLORS\_PRINT[color]);

printf("%s", msg);

printf("%s", COLORS\_PRINT[0]);

}

}

int authorization(const char\* string, char (\*users)[NICK\_MAX\_LEN], int count) {

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

int good = 1;

int j = 0;

while (users[i][j] != '\0' && string[j] != '\0') {

if (users[i][j] != string[j]) {

good = 0;

break;

}

/// deb

if (j >= NICK\_MAX\_LEN) {

printf("ERRRRORO\n");

return 0;

}

j++;

}

if (good == 1 && users[i][j] == string[j]) {

return i + 1;

}

}

return 0;

}

int is\_for\_server(const char\* shared) {

return (shared[0] == SERVER);

}

int get\_user\_index(char (\*users)[NICK\_MAX\_LEN], char\* user) {

for (int i = 0; i < MAX\_U\_COUNT; ++i) {

if (strcmp(users[i], user) == 0) {

return i;

}

}

return -1;

}

//[flag server;dest;poster; option;msg...msg]

void send\_msg(const char server, const char\* dest, const char\* poster, const char option, const char\* msg, char\* envelope) {

memset(envelope, server, sizeof(char));

memcpy(envelope + 1, dest, NICK\_MAX\_LEN);

memcpy(envelope + 1 + NICK\_MAX\_LEN, poster, NICK\_MAX\_LEN);

memset(envelope + 1 + 2 \* NICK\_MAX\_LEN, option, sizeof(char));

memcpy(envelope + 2 + 2 \* NICK\_MAX\_LEN, msg, MSG\_LEN\_LIMIT);

}

int server(char (\*users)[NICK\_MAX\_LEN], chat\* chats, int count, int c\_count) {

int fd;

int fdMutex;

int fdCond;

pthread\_mutex\_t\* mutex;

pthread\_cond\_t\* condition;

pthread\_mutexattr\_t mutex\_attribute;

check\_ok(pthread\_mutexattr\_init(&mutex\_attribute), 0, "Error initializing mutex attribute!\n")

check\_ok(pthread\_mutexattr\_setpshared(&mutex\_attribute, PTHREAD\_PROCESS\_SHARED), 0, "Error sharing mutex attribute!\n")

pthread\_condattr\_t condition\_attribute;

check\_ok(pthread\_condattr\_init(&condition\_attribute), 0, "Error initializing cond attribute!\n")

check\_ok(pthread\_condattr\_setpshared(&condition\_attribute, PTHREAD\_PROCESS\_SHARED), 0, "Error sharing cond attribute!\n")

/\* Shared file \*/

fd = shm\_open(SHARED\_SERVER\_NAME, O\_RDWR | O\_CREAT, S\_IRWXU);

check\_wrong(fd, -1, "Error creating shared file!\n")

check\_ok(ftruncate(fd, SHARED\_MEMORY\_SIZE), 0, "Error truncating shared file!\n")

/\* Shared mutex \*/

fdMutex = shm\_open(SHARED\_MUTEX\_NAME, O\_RDWR | O\_CREAT, S\_IRWXU);

check\_ok(ftruncate(fdMutex, sizeof(pthread\_mutex\_t)), 0, "Error creating shared mutex file!\n")

mutex = (pthread\_mutex\_t\*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fdMutex, 0);

check\_wrong(mutex, MAP\_FAILED, "Error mapping shared mutex!\n")

check\_ok(pthread\_mutex\_init(mutex, &mutex\_attribute), 0, "Error initializing mutex!\n")

/\* Shared cond \*/

fdCond = shm\_open(SHARED\_COND\_NAME, O\_RDWR | O\_CREAT, S\_IRWXU);

check\_ok(ftruncate(fdCond, sizeof(pthread\_cond\_t)), 0, "Error creating shared cond file!\n")

condition = (pthread\_cond\_t\*) mmap(NULL, sizeof(pthread\_cond\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fdCond, 0);

check\_ok(pthread\_cond\_init(condition, &condition\_attribute), 0, "Error initializing cond!\n")

check\_ok(pthread\_mutexattr\_destroy(&mutex\_attribute), 0, "Error destoying mutex attribute!\n")

check\_ok(pthread\_condattr\_destroy(&condition\_attribute), 0, "Error destoying cond attribute!\n")

char\* sharedFile;

sharedFile = mmap(NULL, SHARED\_MEMORY\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

check\_wrong(sharedFile, MAP\_FAILED, "Error creating shared file!")

sharedFile[0] = 0;

sharedFile[2 \* NICK\_MAX\_LEN + 1] = 99;

char c;

// options:

// 0 => reg;

// 1 => send msg

// 99 => empty

// flag: = SERVER or = 0 or EXE

// dest = кому отправлять сообщение

// poster = от кого оно пришло

// [flag server;dest;poster;option;msg...msg]

char dest[NICK\_MAX\_LEN];

char poster[NICK\_MAX\_LEN];

char\* msg = sharedFile + 2 + 2 \* NICK\_MAX\_LEN;

char command;

// while (1) {

while (1) {

check\_ok(pthread\_mutex\_lock(mutex), 0, "Error locking mutex on server!\n")

printf("wait for msg....\n");

while (is\_for\_server(sharedFile) != 1) {

check\_ok(pthread\_cond\_wait(condition, mutex), 0, "Error waiting cond on server!\n")

}

printf("smth for me\n");

// init of server

memcpy(dest, sharedFile + 1, NICK\_MAX\_LEN);

memcpy(poster, sharedFile + NICK\_MAX\_LEN + 1, NICK\_MAX\_LEN);

command = sharedFile[2 \* NICK\_MAX\_LEN + 1];

if (command == 0) {

int id = authorization(poster, users, count);

memcpy(sharedFile + 1, poster, NICK\_MAX\_LEN);

// memcpy(sharedFile + NICK\_MAX\_LEN + 1, ADMIN, NICK\_MAX\_LEN);

printf("%s\n", poster);

if (id > 0) {

// success

sharedFile[2 \* NICK\_MAX\_LEN + 1] = 1;

} else {

// fail

sharedFile[2 \* NICK\_MAX\_LEN + 1] = 0;

}

} else if (command == 1) {

int id = get\_user\_index(users, dest);

if (id == -1) {

int exist = 0;

for (int i = 0; i < c\_count; ++i) {

if (strcmp(chats[i].name, dest) == 0) {

int j = 0;

while (chats[i].connected[j] != -1) {

send\_msg(EXE, users[chats[i].connected[j]], poster, command, msg, sharedFile);

printf("[%s -> %s] %s", poster, users[chats[i].connected[j]], msg);

++j;

for (int t = 0; t < count; ++t) {

check\_ok(pthread\_cond\_signal(condition), 0, "Error sending signal on server!\n")

}

check\_ok(pthread\_mutex\_unlock(mutex), 0, "Error unlocking mutex on server!\n")

check\_ok(pthread\_mutex\_lock(mutex), 0, "Error locking mutex on server!\n")

printf("wait for empty....\n");

// fixed bug if smb send while this loop don't exec

while (sharedFile[2 \* NICK\_MAX\_LEN + 1] != 99) {

check\_ok(pthread\_cond\_wait(condition, mutex), 0, "Error waiting cond on server!\n")

}

}

printf("j = %d\n", j);

exist += j;

}

}

if (exist == 0) {

printf("No such chat\n");

// we don't send, just clear

sharedFile[2 \* NICK\_MAX\_LEN + 1] = 99;

}

} else {

// user [flag server;dest;poster;option;msg...msg]

send\_msg(0, dest, poster, 1, msg, sharedFile);

printf("типо отправил сообщение\n");

printf("[%s -> %s] %s", poster, dest, msg);

}

} else {

printf("command: %d", command);

}

sharedFile[0] = 0;

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

check\_ok(pthread\_cond\_signal(condition), 0, "Error sending signal on server!\n")

}

check\_ok(pthread\_mutex\_unlock(mutex), 0, "Error unlocking mutex on server!\n")

}

check\_wrong(munmap(sharedFile, SHARED\_MEMORY\_SIZE), -1, "Error unmapping fd1!")

check\_ok(munmap(mutex, sizeof(pthread\_mutex\_t)), 0, "Error unmapping mutex!\n")

check\_ok(munmap(condition, sizeof(pthread\_cond\_t)), 0, "Error unmapping cond!\n")

check\_wrong(shm\_unlink(SHARED\_SERVER\_NAME), -1, "Error unlinking shared file!\n")

check\_wrong(shm\_unlink(SHARED\_MUTEX\_NAME), -1, "Error unlinking shared mutex file!\n")

check\_wrong(shm\_unlink(SHARED\_COND\_NAME), -1, "Error unlinking shared cond file!\n")

return 0;

}

void read\_name(char\* s, const char\* obj) {

printf("Enter %s name (no longer than %d chars): ", obj, NICK\_MAX\_LEN - 1);

fgets(s, NICK\_MAX\_LEN, stdin);

// if line longer than NICK\_MAX\_LEN

printf("Have read: ");

printf("%s", s);

if (s[NICK\_MAX\_LEN - 2] != '\0') {

s[NICK\_MAX\_LEN - 2] = '\n';

while (getc(stdin) != '\n');

}

// remove \n

for (int i = NICK\_MAX\_LEN - 1; i >= 0; --i) {

if (s[i] == '\n') {

s[i] = '\0';

break;

}

}

}

void set\_user\_to\_chat(int id, chat\* c) {

for (int i = 0; i < MAX\_U\_COUNT; ++i) {

if (c->connected[i] == -1 || c->connected[i] == id) {

c->connected[i] = id;

return;

}

}

}

int main() {

char users[MAX\_U\_COUNT][NICK\_MAX\_LEN];

chat chats[MAX\_U\_COUNT];

for (int i = 0; i < MAX\_U\_COUNT; ++i) {

memset(users[i], 0, NICK\_MAX\_LEN);

memset(chats[i].name, 0, NICK\_MAX\_LEN);

for (int j = 0; j < MAX\_U\_COUNT; ++j) {

chats[i].connected[j] = -1;

}

}

int u\_count = 0;

int c\_count = 0;

print("Hello, this is the best local messenger for Unix!\n", 0);

int chose = 0;

while (chose != -1) {

print("enter your query:\n", 0);

print("0: add User ", 3);

print("1: create Chat ", 3);

print("2: start server ", 5);

print("-1: exit\n", 4);

char s[NICK\_MAX\_LEN];

int users\_in\_chat;

memset(s, 0, NICK\_MAX\_LEN);

scanf("%d", &chose);

// get "\n"

getc(stdin);

switch (chose) {

case 0:

if (u\_count == MAX\_U\_COUNT) {

printf("cannot create more users\n");

continue;

}

read\_name(s, "User");

if (get\_user\_index(users, s) == -1) {

memcpy(users[u\_count], s, NICK\_MAX\_LEN);

++u\_count;

} else {

printf("%s have been created already\n", s);

}

break;

case 1:

if (c\_count == MAX\_U\_COUNT) {

printf("cannot create more chats\n");

continue;

}

read\_name(s, "Chat");

if (get\_user\_index(users, s) != -1) {

printf("invalid name, %s user exist", s);

continue;

}

/\*

\* it is possible to create 2 chats with the same name

\* they will work as one common

\*/

memcpy(chats[c\_count].name, s, NICK\_MAX\_LEN);

printf("Enter number of users to connect to chat '%s': ", chats[c\_count].name);

scanf("%d", &users\_in\_chat);

// get \n

getc(stdin);

if (users\_in\_chat > MAX\_U\_COUNT) {

printf("We cannot add so many users; the limit is: %d", MAX\_U\_COUNT);

memset(chats[c\_count].name, 0, NICK\_MAX\_LEN);

continue;

}

printf("Input %d existing users (one user in each line)\n", users\_in\_chat);

for (int j = 0; j < users\_in\_chat; ++j) {

read\_name(s, "User");

int id = get\_user\_index(users, s);

if (id == -1) {

printf("all users had to be created, we dont have %s; try another user\n", s);

j--;

continue;

}

set\_user\_to\_chat(id, &chats[c\_count]);

}

printf("\n");

++c\_count;

break;

case 2:

printf("\t\tServer has started.\t\t\n");

server(users, chats, u\_count, c\_count);

break;

case -1:

for (int i = 0; i < u\_count; ++i) {

printf("%s ,", users[i]);

}

print("clear ALL\n", 1);

break;

default:

print("no such chose \U0001F63F\n", 0);

}

}

printf("end\n");

return 0;

}

================================= user.c ===================================

#include <fcntl.h>

#include <pthread.h>

#include <stdio.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <string.h>

#define COLOR\_WHITE\_BOLD\_UNDERLINED "\e[1;04m"

#define COLOR\_GREEN "\033[1;32m"

#define COLOR\_ORANGE "\033[1;33m"

#define COLOR\_MAGENTA "\033[1;35m"

#define COLOR\_OFF "\e[m"

#define check\_ok(VALUE, OK\_VAL, MSG) if (VALUE != OK\_VAL) { printf("%s", MSG); return 1; }

#define check\_wrong(VALUE, WRONG\_VAL, MSG) if (VALUE == WRONG\_VAL) { printf("%s", MSG); return 1; }

const char\* COLORS\_PRINT[6] = {COLOR\_OFF, "", COLOR\_WHITE\_BOLD\_UNDERLINED, COLOR\_ORANGE, COLOR\_MAGENTA, COLOR\_GREEN};

const char\* SHARED\_SERVER\_NAME = "shared\_SeRvEr";

const char\* SHARED\_MUTEX\_NAME = "shared\_mutex";

const char\* SHARED\_COND\_NAME = "shared\_cond";

const int MAX\_U\_COUNT = 100;

const int MSG\_LEN\_LIMIT = 128;

const int NICK\_MAX\_LEN = 8;

const char SERVER = 1;

const char EXE = 2;

const int SHARED\_MEMORY\_SIZE = 1 + 2 \* NICK\_MAX\_LEN + 1 + MSG\_LEN\_LIMIT;

typedef struct {

char\* me;

char\* shared;

pthread\_mutex\_t\* mutex;

pthread\_cond\_t\* cond;

} Token;

void print(const char\* msg, unsigned color) {

if (color >= 6) {

printf("%s", msg);

} else {

printf("%s", COLORS\_PRINT[color]);

printf("%s", msg);

printf("%s", COLORS\_PRINT[0]);

}

}

int is\_for\_me(const char\* shared, const char\* me) {

for (int i = 0; i < NICK\_MAX\_LEN; ++i) {

if (shared[i] != me[i]) {

return 0;

}

}

return 1;

}

void\* pooling(void\* arg) {

Token\* token = ((Token\*) arg);

printf("start pooling\n");

while (1) {

pthread\_mutex\_lock(token->mutex);

while (is\_for\_me(token->shared + 1, token->me) != 1) {

pthread\_cond\_wait(token->cond, token->mutex);

}

printf("[%s] %s", token->shared + NICK\_MAX\_LEN + 1, token->shared + 2 \* NICK\_MAX\_LEN + 2);

memset(token->shared, 0, SHARED\_MEMORY\_SIZE);

token->shared[2 \* NICK\_MAX\_LEN + 1] = 99; // make it empty

pthread\_cond\_signal(token->cond);

pthread\_mutex\_unlock(token->mutex);

}

return NULL;

}

void start\_pooling(const Token\* token) {

pthread\_t th;

if (pthread\_create(&th, NULL, &pooling, (void\*)token) != 0) {

printf("cannot create thread\n");

return;

}

}

int parse(char\* dest, char\* buff) {

if (buff[0] != '[') return -1;

int i = 1;

for (; buff[i] != ']'; ++i) {

if (i > NICK\_MAX\_LEN) return -1;

dest[i - 1] = buff[i];

}

for (int j = i; j <= NICK\_MAX\_LEN; ++j) {

dest[j - 1] = 0;

}

return (i + 2);

}

int main(int argc, char\*\* argv) {

/\* Shared file \*/

int fd = shm\_open(SHARED\_SERVER\_NAME, O\_RDWR, S\_IRWXU);

check\_wrong(fd, -1, "Error opening shared file in child process!\n")

char\* sharedFile = mmap(NULL, SHARED\_MEMORY\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

check\_wrong(sharedFile, MAP\_FAILED, "Error mapping shared file in child process!\n")

/\* Shared mutex \*/

int fdMutex = shm\_open(SHARED\_MUTEX\_NAME, O\_RDWR, S\_IRWXU);

check\_wrong(fdMutex, -1, "Error opening shared mutex file in child process!\n")

pthread\_mutex\_t\* mutex = mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fdMutex, 0);

check\_wrong(mutex, MAP\_FAILED, "Error mapping shared mutex file in child process!\n")

/\* Shared cond \*/

int fdCond = shm\_open(SHARED\_COND\_NAME, O\_RDWR, S\_IRWXU);

check\_wrong(fdCond, -1, "Error opening shared cond file in child process!\n")

pthread\_cond\_t\* condition = mmap(NULL, sizeof(pthread\_cond\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fdCond, 0);

check\_wrong(condition, MAP\_FAILED, "Error mapping shared cond file in child process!\n")

// have to auth

int auth = 0;

char me[NICK\_MAX\_LEN];

memset(me, 0, NICK\_MAX\_LEN);

while (auth == 0) {

printf("Enter User name (no longer than %d chars): ", NICK\_MAX\_LEN - 1);

fgets(me, NICK\_MAX\_LEN, stdin);

// if line longer than NICK\_MAX\_LEN

if (me[NICK\_MAX\_LEN - 2] != '\0') {

me[NICK\_MAX\_LEN - 2] = '\n';

while (getc(stdin) != '\n');

}

// remove \n

for (int i = NICK\_MAX\_LEN - 1; i >= 0; --i) {

if (me[i] == '\n') {

me[i] = '\0';

break;

}

}

// send query

check\_ok(pthread\_mutex\_lock(mutex), 0, "Error locking mutex in child!\n")

printf("wait for empty....\n");

while (sharedFile[2 \* NICK\_MAX\_LEN + 1] != 99 || sharedFile[0] == EXE) {

check\_ok(pthread\_cond\_wait(condition, mutex), 0, "Error waiting cond in child!\n")

}

printf("sent\n");

memset(sharedFile + 1, 0, NICK\_MAX\_LEN); // can be removed

memcpy(sharedFile + NICK\_MAX\_LEN + 1, me, NICK\_MAX\_LEN);

sharedFile[2 \* NICK\_MAX\_LEN + 1] = 0;

sharedFile[0] = SERVER;

for (int i = 0; i < 10; ++i) {

check\_ok(pthread\_cond\_signal(condition), 0, "Error sending signal in child!\n")

}

check\_ok(pthread\_mutex\_unlock(mutex), 0, "Error unlocking mutex in child!\n")

check\_ok(pthread\_mutex\_lock(mutex), 0, "Error locking mutex in child!\n")

printf("wait for anwer....\n");

while (is\_for\_me(sharedFile + 1, me) != 1) {

check\_ok(pthread\_cond\_wait(condition, mutex), 0, "Error waiting cond in child!\n")

}

auth = sharedFile[2 \* NICK\_MAX\_LEN + 1];

sharedFile[2 \* NICK\_MAX\_LEN + 1] = 99; // make it empty

memset(sharedFile, 0, 2 \* NICK\_MAX\_LEN + 1);

for (int i = 0; i < 10; ++i) {

check\_ok(pthread\_cond\_signal(condition), 0, "Error sending signal in child!\n")

}

check\_ok(pthread\_mutex\_unlock(mutex), 0, "Error unlocking mutex in child!\n")

if (auth == 1) {

print("SUCCESS!\n", 5);

break;

} else {

/\*

\* http://uc.org.ru/node/200

\*/

printf("\033[1;4;31m");

printf("FAIL=(\n");

printf("%s", COLOR\_OFF);

}

}

Token token = {me, sharedFile, mutex, condition};

start\_pooling(&token);

print("Now you can send msg like this:\n [Vasya01] Hello, how are you??\n or [!] to exit\n", 1);

// sending msg

char buff [NICK\_MAX\_LEN + 2 + MSG\_LEN\_LIMIT];

char dest[NICK\_MAX\_LEN];

char poster[NICK\_MAX\_LEN];

// buff = "[NICK] msg" so + 2

while (fgets(buff, NICK\_MAX\_LEN + 2 + MSG\_LEN\_LIMIT, stdin)) {

if (buff[1] == '!') {

break;

}

int start\_msg = parse(dest, buff);

if (start\_msg == -1) {

print("cannot parse your query\n", 3);

continue;

}

check\_ok(pthread\_mutex\_lock(mutex), 0, "Error locking mutex in child!\n")

printf("wait for empty....\n");

while (sharedFile[2 \* NICK\_MAX\_LEN + 1] != 99 || sharedFile[0] == EXE) {

check\_ok(pthread\_cond\_wait(condition, mutex), 0, "Error waiting cond in child!\n")

}

printf("sent\n");

memcpy(sharedFile + 1, dest, NICK\_MAX\_LEN);

memcpy(sharedFile + NICK\_MAX\_LEN + 1, me, NICK\_MAX\_LEN);

memcpy(sharedFile + 2 \* NICK\_MAX\_LEN + 2, buff + start\_msg, MSG\_LEN\_LIMIT);

sharedFile[2 \* NICK\_MAX\_LEN + 1] = 1;

sharedFile[0] = SERVER;

for (int i = 0; i < 10; ++i) {

check\_ok(pthread\_cond\_signal(condition), 0, "Error sending signal in child!\n")

}

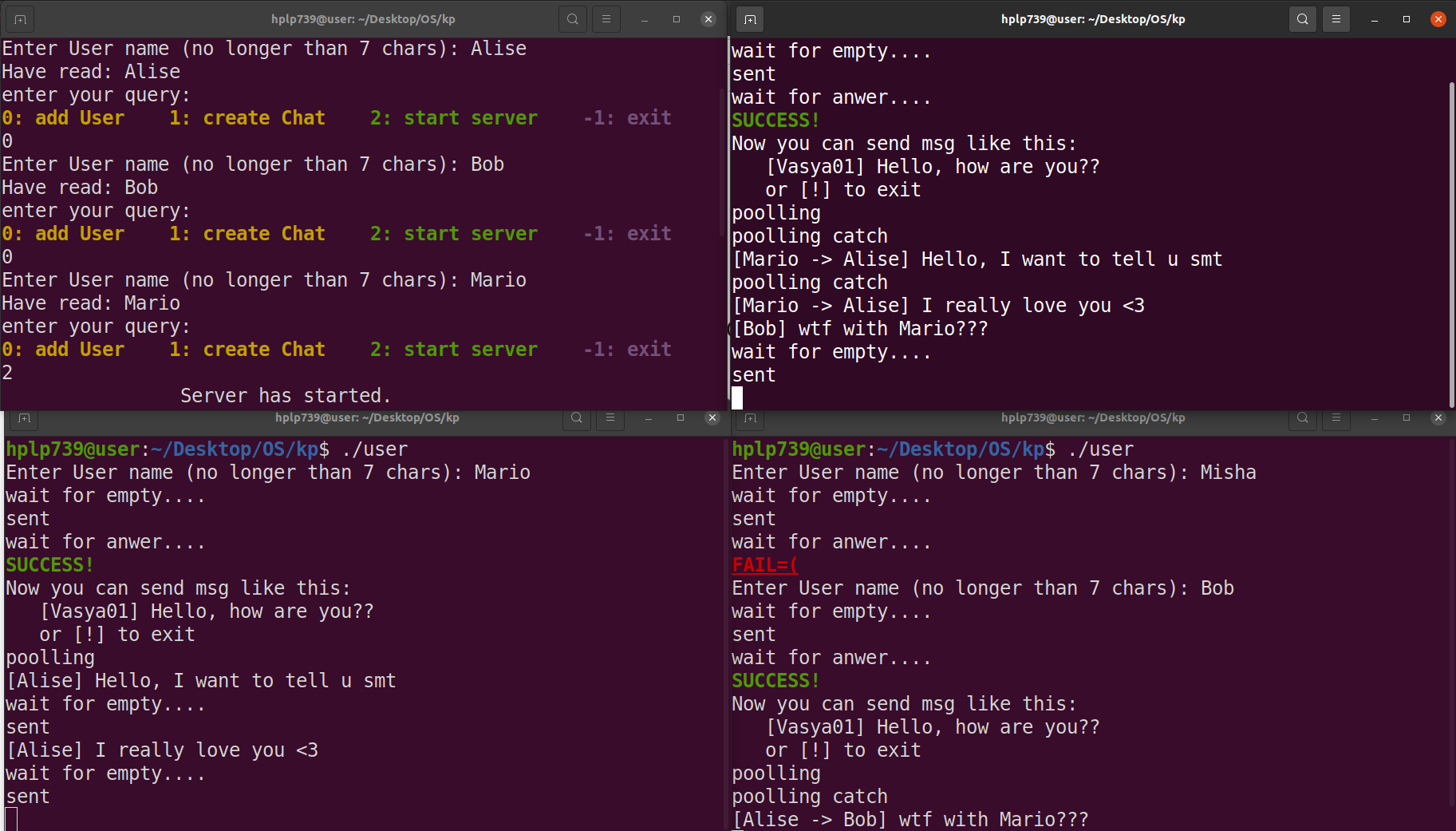
check\_ok(pthread\_mutex\_unlock(mutex), 0, "Error unlocking mutex in child!\n")

memset(buff, 0, NICK\_MAX\_LEN + 2 + MSG\_LEN\_LIMIT);

}

}

**Демонстрация работы программы**



**Выводы**

Составлена и отлажена программа на языке C, реализующая клиент-серверную систему «мгновенных сообщений». Общение между пользователем и сервером осуществляется при помощи memory map. В системе присутствует возможность создания групповых чатов.