Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Юнусов Руслан Асифович
Группа: М8О-209Б-23
Вариант: 24
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2025

Содержание

Репозиторий

https://github.com/Rissochek/KP_OS

Постановка задачи

Цель работы

Целью работы является:

Приобретение практических навыков в использовании знаний, полученных в течении курса

Проведение исследования в выбранной предметной области

Задание

Разработать клиент-серверную 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.

Вариант №24: Необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи memory map

Общие сведения о программе

Связь между клиентом и сервером осуществляется при помощи shared memory. Каждому клиенту выделяется 2 именованных области памяти, одна из которых работает для передачи сообщений от клиента к серверу/другим клиентам, а другая работает для передачи сообщений от сервера/клиентов данному пользователю. Также есть общий для всех сегмент памяти, предназначенный для регистрации пользователей.

Общий метод и алгоритм решения

В ходе выполнения программы используются такие системные вызовы:

mmap(для отображения содержимого файла в адресное пространство процесса), shm_open (для создания или открытия именованного объекта разделяемой памяти), ftruncate (для установки размера считываемого файла), а также shm_unlink и mummap для освобождения выделенной памяти.

Исходный код

server.c

```
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

#define SHM_REG_NAME "/registration"
#define SHM_REG_SIZE 50
#define SHM_CHAT_SIZE sizeof(message_t)
#define USER_STORE_SIZE 20
#define GROUP_NAME_PREFIX '#'
#define COMMAND_PREFIX '/'

typedef struct{
   char destination[100];
   char source[100];
   char message[256];
} message_t;

typedef struct{
   char shm_send_name[100];
   char shm_recv_name[100];
} chat_thread_args;

typedef struct {
   char name[100];
```

```
    char members[USER_STORE_SIZE][100];
    size_t member_count;
} group_t;

typedef struct {
    group_t groups[USER_STORE_SIZE];
    size_t group_count;
    pthread_mutex_t mutex;
} server_data_t;
server_data_t server_data = {.group_count = 0, .mutex = PTHREAD_MUTEX_INITIALIZER};

void* handle_memptr(const char* shm_name, size_t shm_size);
void* handle_chat_thread(void* arg);
void create_group(const char* group_name);
void join_group(const char* group_name, const char* username);

int main(){
    pthread_t threads[USER_STORE_SIZE];
    size_t threads_counter = 0;
    char* reg_memptr = (char*)handle_memptr(SHM_REG_NAME, SHM_REG_SIZE);
    char** user_store = malloc(USER_STORE_SIZE * sizeof(char*));
    chat_thread_args *args;
    size_t user_count = 0;

    printf("Сервер запущен и ожидает регистрации пользователей...\n");
    while (1){
        char username[50];
        strncpy(username, reg_memptr, sizeof(username) - 1);
        username[sizeof(username) - 1] = '\0';
        if (username[0] != '\0' && user_count < USER_STORE_SIZE){
            memset(reg_memptr, '\0', SHM_REG_SIZE);
```

```
            printf("Новый пользователь: %s\n", username);
            user_store[user_count++] = strdup(username);

            args = malloc(sizeof(chat_thread_args));
            snprintf(args->shm_send_name, sizeof(args->shm_send_name), "/%s_send", username);
            snprintf(args->shm_recv_name, sizeof(args->shm_recv_name), "/%s_recv", username);

            if (pthread_create(&threads[threads_counter++], NULL, handle_chat_thread,
(void*)args) != 0){
                perror("Ошибка создания потока");
                exit(EXIT_FAILURE);
            }
            printf("Поток для %s создан.\n", username);
        }
        usleep(100000);
    }

    for(size_t i = 0; i < user_count; ++i){
        free(user_store[i]);
    }
    free(user_store);
    free(args);
    munmap(reg_memptr, SHM_REG_SIZE);
    shm_unlink(SHM_REG_NAME);
    return 0;
}

void* handle_chat_thread(void* arg){
    chat_thread_args* args = (chat_thread_args*)arg;
    printf("Поток прослушивания для %s создан.\n", args->shm_send_name);
```

```
    message_t* chat_send_memptr = (message_t*)handle_memptr(args->shm_send_name,
SHM_CHAT_SIZE);
    message_t message;


    while (1){
        if (chat_send_memptr->destination[0] != '\0'){
            memcpy(&message, chat_send_memptr, sizeof(message_t));
            memset(chat_send_memptr, '\0', SHM_CHAT_SIZE);


            printf("Получено сообщение от %s для %s: %s\n", message.source,
message.destination, message.message);


            if (message.destination[0] == COMMAND_PREFIX){
                if (strncmp(message.destination, "/create_group", strlen("/create_group")) == 0){
                    create_group(message.message);
                    printf("Группа %s создана.\n", message.message);
                }
                else if (strncmp(message.destination, "/join_group", strlen("/join_group")) == 0){
                    join_group(message.message, message.source);
                    printf("Пользователь %s присоединился к группе %s.\n", message.source,
message.message);
                }
            }
            else if (message.destination[0] == GROUP_NAME_PREFIX){
                char group_name[100];
                strncpy(group_name, message.destination + 1, sizeof(group_name) - 1);
                group_name[sizeof(group_name) - 1] = '\0';
                pthread_mutex_lock(&server_data.mutex);
                int group_found = 0;
                for(size_t i = 0; i < server_data.group_count; ++i){
                    if (strcmp(server_data.groups[i].name, group_name) == 0){
                        group_found = 1;
```

```
                for(size_t j = 0; j < server_data.groups[i].member_count; ++j){
                    if (strcmp(server_data.groups[i].members[j], message.source) != 0){
                        char shm_dest_recv[110];
                        snprintf(shm_dest_recv, sizeof(shm_dest_recv), "/%s_recv",
server_data.groups[i].members[j]);
                        printf("Отправка группового сообщения получателю: %s\n",
shm_dest_recv);
                        message_t* dest_memptr = (message_t*)handle_memptr(shm_dest_recv,
SHM_CHAT_SIZE);
                        memcpy(dest_memptr, &message, sizeof(message_t));
                        munmap(dest_memptr, SHM_CHAT_SIZE);
                        printf("Групповое сообщение отправлено получателю: %s\n",
server_data.groups[i].members[j]);
                    }
                }
                break;
            }
        }
        if (!group_found){
            printf("Группа %s не найдена.\n", group_name);
        }
        pthread_mutex_unlock(&server_data.mutex);
    }
    else{
        char shm_dest_recv[110];
        snprintf(shm_dest_recv, sizeof(shm_dest_recv), "/%s_recv", message.destination);
        printf("Отправка сообщения получателю: %s\n", shm_dest_recv);
        message_t* dest_memptr = (message_t*)handle_memptr(shm_dest_recv,
SHM_CHAT_SIZE);
        memcpy(dest_memptr, &message, sizeof(message_t));
        munmap(dest_memptr, SHM_CHAT_SIZE);
        printf("Сообщение отправлено получателю: %s\n", message.destination);
```

```
            }
        }
        usleep(100000);
    }
    free(args);
    return NULL;
}

void create_group(const char* group_name){
    pthread_mutex_lock(&server_data.mutex);
    for(size_t i = 0; i < server_data.group_count; ++i){
        if (strcmp(server_data.groups[i].name, group_name) == 0){
            printf("Группа %s уже существует.\n", group_name);
            pthread_mutex_unlock(&server_data.mutex);
            return;
        }
    }
    if (server_data.group_count < USER_STORE_SIZE){
        strncpy(server_data.groups[server_data.group_count].name, group_name,
sizeof(server_data.groups[server_data.group_count].name) - 1);

server_data.groups[server_data.group_count].name[sizeof(server_data.groups[server_data.gro
up_count].name) - 1] = '\0';
        server_data.groups[server_data.group_count].member_count = 0;
        server_data.group_count++;
        printf("Группа %s успешно создана.\n", group_name);
    }
    pthread_mutex_unlock(&server_data.mutex);
}

void join_group(const char* group_name, const char* username){
    pthread_mutex_lock(&server_data.mutex);
```

```
    for(size_t i = 0; i < server_data.group_count; ++i){
        if (strcmp(server_data.groups[i].name, group_name) == 0){
            int already_member = 0;
            for(size_t j = 0; j < server_data.groups[i].member_count; ++j){
                if (strcmp(server_data.groups[i].members[j], username) == 0){
                    already_member = 1;
                    break;
                }
            }
            if (!already_member && server_data.groups[i].member_count < USER_STORE_SIZE){
                strncpy(server_data.groups[i].members[server_data.groups[i].member_count],
username, sizeof(server_data.groups[i].members[server_data.groups[i].member_count]) - 1);

server_data.groups[i].members[server_data.groups[i].member_count][sizeof(server_data.grou
ps[i].members[server_data.groups[i].member_count]) - 1] = '\0';
                server_data.groups[i].member_count++;
                printf("Пользователь %s успешно присоединился к группе %s.\n", username,
group_name);
            }
            else{
                printf("Пользователь %s уже состоит в группе %s или группа полна.\n",
username, group_name);
            }
            pthread_mutex_unlock(&server_data.mutex);
            return;
        }
    }
    pthread_mutex_unlock(&server_data.mutex);
    printf("Группа %s не найдена.\n", group_name);
}

void* handle_memptr(const char* shm_name, size_t shm_size) {
```

```
    int fd = shm_open(shm_name, O_RDWR | O_CREAT, 0666);
    if (fd < 0) {
        perror("SHM_OPEN");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(fd, shm_size) == -1){
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }

    void* memptr = mmap(NULL, shm_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (memptr == MAP_FAILED) {
        perror("MMAP");
        exit(EXIT_FAILURE);
    }
    close(fd);
    memset(memptr, '\0', shm_size);
    return memptr;
}
```

client.c

```
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
```

```
#define SHM_REG_NAME "/registration"
#define SHM_REG_SIZE 50
#define SHM_CHAT_SIZE sizeof(message_t)
#define GROUP_NAME_PREFIX '#'
#define COMMAND_PREFIX '/'

typedef struct{
    char destination[100];
    char source[100];
    char message[256];
} message_t;

typedef struct{
    char shm_recv_name[100];
} chat_thread_args;

void* handle_memptr(const char* shm_name, size_t shm_size);
void* handle_listen_thread(void* args);

int main(){
    char login_username[50];
    pthread_t listen_thread;
    message_t msg = {};
    char* reg_memptr = (char*) handle_memptr(SHM_REG_NAME,
SHM_REG_SIZE);

    printf("Введите имя пользователя для регистрации: ");
    scanf("%49s", login_username);
    strncpy(reg_memptr, login_username, SHM_REG_SIZE - 1);
```

```
    reg_memptr[SHM_REG_SIZE - 1] = '\0';
    printf("Вы зарегистрированы как: %s\n", login_username);
    char shm_send[100];
    char shm_recv[100];
    snprintf(shm_send, sizeof(shm_send), "/%s_send", login_username);
    snprintf(shm_recv, sizeof(shm_recv), "/%s_recv", login_username);

    message_t* chat_send_memptr = (message_t*) handle_memptr(shm_send,
SHM_CHAT_SIZE);
    chat_thread_args *args = malloc(sizeof(chat_thread_args));
    strncpy(args->shm_recv_name, shm_recv, sizeof(args->shm_recv_name) - 1);
    args->shm_recv_name[sizeof(args->shm_recv_name) - 1] = '\0';
    if (pthread_create(&listen_thread, NULL, handle_listen_thread, (void*)args) !=
0){
        perror("Ошибка создания потока");
        exit(EXIT_FAILURE);
    }
    printf("Поток прослушивания запущен.\n");

    while (1){
        printf("Введите имя получателя ('exit' для выхода) или перейдите в меню
управления группой при помощи '/', для отправки сообщения в группу
необходимо вступить в нее и в качестве получателя указать #Имя_Группы: ");
        scanf("%s", msg.destination);
        if (strcmp(msg.destination, "exit") == 0){
            break;
        }

        if (strncmp(msg.destination, "/", 1) == 0){
```

```
            printf("Введите команду (например, /create_group Friends или
/join_group Friends): ");
            getchar();
            fgets(msg.message, sizeof(msg.message), stdin);
            msg.message[strcspn(msg.message, "\n")] = '\0';
        }
        else if (msg.destination[0] == GROUP_NAME_PREFIX){
            printf("Введите сообщение для группы %s: ", msg.destination);
            getchar();
            fgets(msg.message, sizeof(msg.message), stdin);
            msg.message[strcspn(msg.message, "\n")] = '\0';
        }
        else{
            printf("Введите сообщение для пользователя %s: ", msg.destination);
            getchar();
            fgets(msg.message, sizeof(msg.message), stdin);
            msg.message[strcspn(msg.message, "\n")] = '\0';
        }
        strncpy(chat_send_memptr->source, login_username,
sizeof(chat_send_memptr->source) - 1);
        chat_send_memptr->source[sizeof(chat_send_memptr->source) - 1] = '\0';
        strncpy(chat_send_memptr->destination, msg.destination,
sizeof(chat_send_memptr->destination) - 1);
        chat_send_memptr->destination[sizeof(chat_send_memptr->destination) - 1]
= '\0';
        strncpy(chat_send_memptr->message, msg.message,
sizeof(chat_send_memptr->message) - 1);
        chat_send_memptr->message[sizeof(chat_send_memptr->message) - 1] = '\0';


        printf("Сообщение отправлено.\n");
```

```
    }

    free(args);
    pthread_cancel(listen_thread);
    pthread_join(listen_thread, NULL);
    shm_unlink(shm_send);
    shm_unlink(shm_recv);
    munmap(chat_send_memptr, SHM_CHAT_SIZE);
    munmap(reg_memptr, SHM_REG_SIZE);
    printf("Клиент завершил работу.\n");
    return 0;
}
void* handle_memptr(const char* shm_name, size_t shm_size) {
    int fd = shm_open(shm_name, O_RDWR | O_CREAT, 0666);
    if (fd < 0) {
        perror(shm_name);
        exit(EXIT_FAILURE);
    }
    if (ftruncate(fd, shm_size) == -1){
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }
    void* memptr = mmap(NULL, shm_size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (memptr == MAP_FAILED) {
        perror("MMAP");
        exit(EXIT_FAILURE);
    }
    close(fd);
```

```
    memset(memptr, '\0', shm_size);
    return memptr;
}

void* handle_listen_thread(void* args){
    chat_thread_args* chat_args = (chat_thread_args*)args;
    message_t* chat_recv_memptr =
(message_t*)handle_memptr(chat_args->shm_recv_name, SHM_CHAT_SIZE);

    while (1){
        if (chat_recv_memptr->message[0] != '\0'){
            if (chat_recv_memptr->destination[0] == GROUP_NAME_PREFIX){
                printf("\n[Группа %s] %s: %s\n", chat_recv_memptr->destination,
chat_recv_memptr->source, chat_recv_memptr->message);
            }
            else{
                printf("\n%s: %s\n", chat_recv_memptr->source,
chat_recv_memptr->message);
            }
            memset(chat_recv_memptr, '\0', SHM_CHAT_SIZE);
        }
        usleep(100000);
    }

    munmap(chat_recv_memptr, SHM_CHAT_SIZE);
    return NULL;
}
```

Демонстрация работы программы

rissochek@admin:~/OS_KP$ ./server
Сервер запущен и ожидает регистрации пользователей...
Новый пользователь: bro1
Поток для bro1 создан.
Поток прослушивания для /bro1_send создан.
Новый пользователь: bro2
Поток для bro2 создан.
Поток прослушивания для /bro2_send создан.
Получено сообщение от bro1 для /create_group: F
Группа F успешно создана.
Группа F создана.
Получено сообщение от bro1 для /join_group: F
Пользователь bro1 успешно присоединился к группе F.
Пользователь bro1 присоединился к группе F.
Получено сообщение от bro2 для /join_group: F
Пользователь bro2 успешно присоединился к группе F.
Пользователь bro2 присоединился к группе F.
Получено сообщение от bro1 для #F: hii
Отправка группового сообщения получателю: /bro2_recv
Групповое сообщение отправлено получателю: bro2
Получено сообщение от bro2 для #F: ooo
Отправка группового сообщения получателю: /bro1_recv
Групповое сообщение отправлено получателю: bro1

rissochek@admin:~/OS_KP$ ./client
Введите имя пользователя для регистрации: bro1
Вы зарегистрированы как: bro1
Поток прослушивания запущен.
Введите имя получателя ('exit' для выхода) или перейдите в меню управления группой при помощи '/', для отправки сообщения в группу необходимо вступить в нее и в качестве получателя указать #Имя_Группы: /create_group F
Введите команду (например, /create_group Friends или /join_group Friends):
Сообщение отправлено.
Введите имя получателя ('exit' для выхода) или перейдите в меню управления группой при помощи '/', для отправки сообщения в группу необходимо вступить в нее и в качестве получателя указать #Имя_Группы: /join_group F
Введите команду (например, /create_group Friends или /join_group Friends):
Сообщение отправлено.

Введите имя получателя ('exit' для выхода) или перейдите в меню управления группой при помощи '/', для отправки сообщения в группу необходимо вступить в нее и в качестве получателя указать #Имя_Группы: #F hii
Введите сообщение для группы #F: Сообщение отправлено.
Введите имя получателя ('exit' для выхода) или перейдите в меню управления группой при помощи '/', для отправки сообщения в группу необходимо вступить в нее и в качестве получателя указать #Имя_Группы:
[Группа #F] bro2: ooo

rissochek@admin:~/OS_KP$ ./client
Введите имя пользователя для регистрации: bro2
Вы зарегистрированы как: bro2
Поток прослушивания запущен.
Введите имя получателя ('exit' для выхода) или перейдите в меню управления группой при помощи '/', для отправки сообщения в группу необходимо вступить в нее и в качестве получателя указать #Имя_Группы: /join_group F
Введите команду (например, /create_group Friends или /join_group Friends):
Сообщение отправлено.
Введите имя получателя ('exit' для выхода) или перейдите в меню управления группой при помощи '/', для отправки сообщения в группу необходимо вступить в нее и в качестве получателя указать #Имя_Группы:
[Группа #F] bro1: hii
#F ooo
Введите сообщение для группы #F: Сообщение отправлено.
Введите имя получателя ('exit' для выхода) или перейдите в меню управления группой при помощи '/', для отправки сообщения в группу необходимо вступить в нее и в качестве получателя указать #Имя_Группы:

Выводы

Разделяемая память позволяет процессам обмениваться данными напрямую через общую область памяти, что делает операции чтения и записи очень быстрыми. Процессы могут обращаться к данным в памяти так же, как если бы они работали с локальными переменными. Ее удобно использовать для работы с большими структурами данных, такими как массивы или структуры, так как они могут быть просто отображены в память и доступны для всех процессов. Она позволяет нескольким процессам одновременно читать и записывать данные, что делает её более гибкой для сложных сценариев взаимодействия. Все это было применено в итоговой программе. В общем у shared memory есть множество преимуществ в сравнении с pipes.