Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Курсовой проект по курсу

«Операционные системы»

Группа: М8О-209БВ-24

Студент: Котик М. Н.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 20.12.25

Москва, 2024

Постановка задачи

Вариант 11.

Консоль-серверная игра "Быки и коровы". Необходимо написать 2 программы: сервер и клиент. Сначала запускается сервер, а далее клиенты соединяются с сервером. Сервер координирует клиентов между собой. При запуске клиента игрок может выбрать одно из следующих действий:

1) Создать игру, введя ее имя
2) Присоединиться к одной из существующих игр по имени игры

"Быки и коровы" (угадывать необходимо слова). Общение между сервером и клиентом необходимо организовать при помощи memory map. При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Если кто-то из игроков вышел из игры, то игра должна быть продолжена.

Общий метод и алгоритм решения

Для реализации многопользовательской консольной игры "Быки и коровы" была разработана клиент-серверная архитектура с использованием POSIX Shared Memory для межпроцессного взаимодействия. Сервер управляет состоянием всех игр, а клиенты подключаются к нему для создания и участия в играх.

Использованные системные вызовы:

***int shm_open(const char name, int oflag, mode_t mode);** – создает или открывает объект shared memory.

****void mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset);** – маппирует shared memory в адресное пространство процесса.

***int munmap(void addr, size_t length);** – удаляет маппинг памяти.

***int shm_unlink(const char name);** – удаляет объект shared memory.

****int pthread_mutex_init(pthread_mutex_t mutex, const pthread_mutexattr_t attr);** – инициализирует мьютекс с атрибутом PTHREAD_PROCESS_SHARED для синхронизации между процессами.

***int pthread_mutex_lock(pthread_mutex_t mutex);** – блокирует мьютекс.

***int pthread_mutex_unlock(pthread_mutex_t mutex**); – разблокирует мьютекс.

****int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void (start_routine) (void), void arg);** – создает поток для периодической очистки неактивных игр.

***time_t time(time_t tloc);** – используется для отслеживания времени последней активности игроков.

Алгоритм работы:

1) Сервер инициализирует shared memory сегмент размером 2MB с именем "/bulls_cows_shm".
2) В shared memory хранится структура SharedMemoryData, содержащая массив игр и глобальный мьютекс.
3) Каждая игра содержит свой мьютекс для синхронизации доступа к данным игры.
4) Клиенты подключаются к shared memory и работают с общими структурами данных.
5) При создании игры указывается загаданное слово и максимальное количество игроков.
6) Игроки могут присоединяться к существующим играм, пока есть свободные места.
7) Каждый игрок делает предположения, сервер вычисляет количество быков и коров.
8) При выходе игрока игра продолжается с оставшимися участниками.
9) Игра завершается, когда кто-то угадывает слово или не остается активных игроков.

Код программы

client.h

```h
#ifndef CLIENT_H
#define CLIENT_H

#include "game_common.h"

int init_client_connection();
bool create_game_client(const char *game_name, const char *secret_word, int
max_players);
bool join_game_client(const char *game_name, const char *player_name);
bool make_guess_client(const char *game_name, int player_id, const char *guess);
bool leave_game_client(const char *game_name, int player_id);
void list_games_client();
void print_game_status(const char *game_name);

#endif
```

game_common.h

```h
#ifndef GAME_COMMON_H
#define GAME_COMMON_H

#include <stdbool.h>
#include <pthread.h>

#define MAX_GAMES 10
#define MAX_PLAYERS_PER_GAME 4
#define MAX_PLAYER_NAME 50
#define MAX_GAME_NAME 50
#define MAX_WORD_LENGTH 20
#define MAX_ATTEMPTS 100
#define SHM_NAME "/bulls_cows_shm"
#define SHM_SIZE (2 * 1024 * 1024)
```

```c
typedef struct
{
    char guess[MAX_WORD_LENGTH];
    int bulls;
    int cows;
    int attempt_number;
    bool is_correct;
} AttemptResult;

typedef struct
{
    char name[MAX_PLAYER_NAME];
    int player_id;
    int score;
    int attempts_count;
    AttemptResult attempts[MAX_ATTEMPTS];
    bool is_connected;
    bool is_active;
    time_t last_activity;
} Player;

typedef struct
{
    char game_name[MAX_GAME_NAME];
    char secret_word[MAX_WORD_LENGTH];
    int max_players;
    int current_players;
    bool is_active;
    bool is_game_over;
    int winner_id;
    time_t created_at;
    time_t last_move_at;

    Player players[MAX_PLAYERS_PER_GAME];

    pthread_mutex_t game_mutex;
    pthread_cond_t game_cond;

    int total_moves;
    int words_tried;
} Game;

typedef enum
{
    MSG_CREATE_GAME,
    MSG_JOIN_GAME,
    MSG_MAKE_GUESS,
    MSG_LEAVE_GAME,
    MSG_LIST_GAMES,
    MSG_GAME_UPDATE,
    MSG_GAME_OVER,
```

```
    MSG_ERROR
} MessageType;

typedef struct
{
    MessageType type;
    char game_name[MAX_GAME_NAME];
    char player_name[MAX_PLAYER_NAME];
    char data[MAX_WORD_LENGTH * 2];
    int player_id;
    int max_players;
    int result_code;
    time_t timestamp;
} GameMessage;

typedef struct
{
    Game games[MAX_GAMES];
    int active_games;
    pthread_mutex_t shm_mutex;
    char server_status[100];
} SharedMemoryData;

void init_game(Game *game, const char *name, const char *secret_word, int
max_players);
bool add_player_to_game(Game *game, const char *player_name, int *player_id);
bool remove_player_from_game(Game *game, int player_id);
void calculate_bulls_cows(const char *secret, const char *guess, int *bulls, int
*cows);
bool is_game_joinable(const Game *game);
void broadcast_game_update(Game *game, SharedMemoryData *shm_data);

#endif
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <time.h>
#include "client.h"
#include "game_common.h"

static SharedMemoryData *shm_data = NULL;
static int shm_fd = -1;
static int client_game_id = -1;
static int client_player_id = -1;
static char current_game_name[MAX_GAME_NAME] = "";
```

```
int init_client_connection()
{
    shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1)
    {
        perror("Не удалось открыть shared memory");
        return -1;
    }

    shm_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shm_data == MAP_FAILED)
    {
        perror("Не удалось маппировать shared memory");
        close(shm_fd);
        return -1;
    }

    printf("Подключение к серверу установлено\n");
    return 0;
}

bool create_game_client(const char *game_name, const char *secret_word, int
max_players)
{
    if (!shm_data)
    {
        printf("Нет подключения к серверу\n");
        return false;
    }

    pthread_mutex_lock(&shm_data->shm_mutex);

    for (int i = 0; i < shm_data->active_games; i++)
    {
        if (shm_data->games[i].is_active &&
            strcmp(shm_data->games[i].game_name, game_name) == 0)
        {
            pthread_mutex_unlock(&shm_data->shm_mutex);
            printf("Игра с именем '%s' уже существует\n", game_name);
            return false;
        }
    }

    int free_slot = -1;
    for (int i = 0; i < MAX_GAMES; i++)
    {
        if (!shm_data->games[i].is_active)
        {
            free_slot = i;
```

```
            break;
        }
    }

    if (free_slot == -1)
    {
        pthread_mutex_unlock(&shm_data->shm_mutex);
        printf("Достигнуто максимальное количество игр\n");
        return false;
    }

    Game *new_game = &shm_data->games[free_slot];

    pthread_mutex_lock(&new_game->game_mutex);
    pthread_mutex_unlock(&shm_data->shm_mutex);

    init_game(new_game, game_name, secret_word, max_players);

    char player_name[50];
    printf("Введите ваше имя: ");
    fgets(player_name, sizeof(player_name), stdin);
    player_name[strcspn(player_name, "\n")] = 0;

    int player_id;
    if (!add_player_to_game(new_game, player_name, &player_id))
    {
        pthread_mutex_unlock(&new_game->game_mutex);
        printf("Не удалось добавить игрока в игру\n");
        return false;
    }

    shm_data->active_games++;
    client_game_id = free_slot;
    client_player_id = player_id;
    strcpy(current_game_name, game_name);

    printf("Игра '%s' создана успешно. Ваш ID: %d\n", game_name, player_id);

    pthread_mutex_unlock(&new_game->game_mutex);
    return true;
}

bool join_game_client(const char *game_name, const char *player_name)
{
    if (!shm_data)
    {
        printf("Нет подключения к серверу\n");
        return false;
    }

    pthread_mutex_lock(&shm_data->shm_mutex);
```

```
    Game *target_game = NULL;
    int found_game_id = -1;

    for (int i = 0; i < MAX_GAMES; i++)
    {
        if (shm_data->games[i].is_active &&
            strcmp(shm_data->games[i].game_name, game_name) == 0)
        {
            target_game = &shm_data->games[i];
            found_game_id = i;
            break;
        }
    }

    if (!target_game)
    {
        pthread_mutex_unlock(&shm_data->shm_mutex);
        printf("Игра '%s' не найдена\n", game_name);
        return false;
    }

    pthread_mutex_lock(&target_game->game_mutex);
    pthread_mutex_unlock(&shm_data->shm_mutex);

    if (!is_game_joinable(target_game))
    {
        pthread_mutex_unlock(&target_game->game_mutex);
        printf("Невозможно присоединиться к игре '%s'\n", game_name);
        return false;
    }

    int player_id;
    if (!add_player_to_game(target_game, player_name, &player_id))
    {
        pthread_mutex_unlock(&target_game->game_mutex);
        printf("Не удалось присоединиться к игре\n");
        return false;
    }

    client_game_id = found_game_id;
    client_player_id = player_id;
    strcpy(current_game_name, game_name);

    printf("Вы присоединились к игре '%s' как игрок %d\n", game_name, player_id);

    printf("Текущие игроки в игре:\n");
    for (int i = 0; i < target_game->current_players; i++)
    {
        if (target_game->players[i].is_connected)
        {
```

```
            printf("- %s (ID: %d)\n", target_game->players[i].name, i);
        }
    }

    pthread_mutex_unlock(&target_game->game_mutex);
    return true;
}

bool make_guess_client(const char *game_name, int player_id, const char *guess)
{
    if (!shm_data)
    {
        printf("Нет подключения к серверу\n");
        return false;
    }

    pthread_mutex_lock(&shm_data->shm_mutex);

    Game *game = NULL;

    for (int i = 0; i < MAX_GAMES; i++)
    {
        if (shm_data->games[i].is_active &&
            strcmp(shm_data->games[i].game_name, game_name) == 0)
        {
            game = &shm_data->games[i];
            break;
        }
    }

    if (!game)
    {
        pthread_mutex_unlock(&shm_data->shm_mutex);
        printf("Игра '%s' не найдена\n", game_name);
        return false;
    }

    pthread_mutex_lock(&game->game_mutex);
    pthread_mutex_unlock(&shm_data->shm_mutex);

    if (player_id < 0 || player_id >= game->current_players ||
        !game->players[player_id].is_connected)
    {
        pthread_mutex_unlock(&game->game_mutex);
        printf("Игрок с ID %d не найден или не активен\n", player_id);
        return false;
    }

    int bulls, cows;
    calculate_bulls_cows(game->secret_word, guess, &bulls, &cows);
```

```
    printf("Результат: Быки: %d, Коровы: %d\n", bulls, cows);

    Player *player = &game->players[player_id];
    player->attempts_count++;
    player->last_activity = time(NULL);

    if ((size_t)bulls == strlen(game->secret_word))
    {
        game->is_game_over = true;
        game->winner_id = player_id;
        printf("\nПОЗДРАВЛЯЕМ! Вы угадали слово '%s'!\n", game->secret_word);
        printf("Игра завершена. Победитель: %s\n", player->name);
    }

    pthread_mutex_unlock(&game->game_mutex);
    return true;
}

bool leave_game_client(const char *game_name, int player_id)
{
    if (!shm_data)
    {
        printf("Нет подключения к серверу\n");
        return false;
    }

    pthread_mutex_lock(&shm_data->shm_mutex);

    Game *game = NULL;

    for (int i = 0; i < MAX_GAMES; i++)
    {
        if (shm_data->games[i].is_active &&
            strcmp(shm_data->games[i].game_name, game_name) == 0)
        {
            game = &shm_data->games[i];
            break;
        }
    }

    if (!game)
    {
        pthread_mutex_unlock(&shm_data->shm_mutex);
        printf("Игра '%s' не найдена\n", game_name);
        return false;
    }

    pthread_mutex_lock(&game->game_mutex);
    pthread_mutex_unlock(&shm_data->shm_mutex);

    if (!remove_player_from_game(game, player_id))
```

```
    {
        pthread_mutex_unlock(&game->game_mutex);
        printf("Не удалось выйти из игры\n");
        return false;
    }

    printf("Вы вышли из игры '%s'\n", game_name);

    int active_players = 0;
    for (int i = 0; i < game->current_players; i++)
    {
        if (game->players[i].is_connected)
        {
            active_players++;
        }
    }

    if (active_players == 0 && !game->is_game_over)
    {
        game->is_active = false;
        shm_data->active_games--;
        printf("Игра '%s' завершена (нет активных игроков)\n", game_name);
    }

    client_game_id = -1;
    client_player_id = -1;
    current_game_name[0] = '\0';

    pthread_mutex_unlock(&game->game_mutex);
    return true;
}

void list_games_client()
{
    if (!shm_data)
    {
        printf("Нет подключения к серверу\n");
        return;
    }

    pthread_mutex_lock(&shm_data->shm_mutex);

    printf("\n=== Список активных игр ===\n");

    int found_games = 0;
    for (int i = 0; i < MAX_GAMES; i++)
    {
        if (shm_data->games[i].is_active)
        {
            Game *game = &shm_data->games[i];
```

```
            pthread_mutex_lock(&game->game_mutex);

            int active_players = 0;
            for (int j = 0; j < game->current_players; j++)
            {
                if (game->players[j].is_connected)
                {
                    active_players++;
                }
            }

            printf("Игра: %s\n", game->game_name);
            printf("  Игроков: %d/%d\n", active_players, game->max_players);
            printf("  Статус: %s\n", game->is_game_over ? "Завершена" :
"Активна");

            if (game->is_game_over && game->winner_id != -1)
            {
                printf("  Победитель: %s\n", game->players[game-
>winner_id].name);
            }

            pthread_mutex_unlock(&game->game_mutex);
            printf("------------------------\n");
            found_games++;
        }
    }

    if (found_games == 0)
    {
        printf("Нет активных игр\n");
    }

    pthread_mutex_unlock(&shm_data->shm_mutex);
}

void print_game_status(const char *game_name)
{
    if (!shm_data)
    {
        printf("Нет подключения к серверу\n");
        return;
    }

    pthread_mutex_lock(&shm_data->shm_mutex);

    Game *game = NULL;

    for (int i = 0; i < MAX_GAMES; i++)
    {
        if (shm_data->games[i].is_active &&
```

```
            strcmp(shm_data->games[i].game_name, game_name) == 0)
        {
            game = &shm_data->games[i];
            break;
        }
    }

    if (!game)
    {
        pthread_mutex_unlock(&shm_data->shm_mutex);
        printf("Игра '%s' не найдена\n", game_name);
        return;
    }

    pthread_mutex_lock(&game->game_mutex);
    pthread_mutex_unlock(&shm_data->shm_mutex);

    printf("\n=== Статус игры '%s' ===\n", game_name);
    printf("Загаданное слово: [скрыто]\n");
    printf("Максимум игроков: %d\n", game->max_players);
    printf("Текущих игроков: %d\n", game->current_players);
    printf("Статус: %s\n", game->is_game_over ? "Завершена" : "Активна");

    if (game->is_game_over)
    {
        printf("Победитель: %s\n", game->players[game->winner_id].name);
    }

    printf("\nИгроки:\n");
    for (int i = 0; i < game->current_players; i++)
    {
        Player *player = &game->players[i];
        printf("- %s (ID: %d) [%s]\n",
               player->name, i,
               player->is_connected ? "в сети" : "не в сети");

        if (player->attempts_count > 0)
        {
            printf("  Попыток: %d, Последняя: ", player->attempts_count);
            if (player->attempts_count > 0)
            {
                AttemptResult *last = &player->attempts[player->attempts_count -
1];
                printf("'%s' - Быки: %d, Коровы: %d\n",
                       last->guess, last->bulls, last->cows);
            }
        }
    }

    pthread_mutex_unlock(&game->game_mutex);
}
```

```c
void client_main_menu()
{
    printf("\n=== Клиент 'Быки и Коровы' ===\n");

    if (init_client_connection() != 0)
    {
        printf("Не удалось подключиться к серверу. Убедитесь, что сервер
запущен.\n");
        return;
    }

    while (1)
    {
        printf("\nГлавное меню:\n");
        printf("1. Создать новую игру\n");
        printf("2. Присоединиться к существующей игре\n");
        printf("3. Сделать предположение (если в игре)\n");
        printf("4. Показать список игр\n");
        printf("5. Показать статус текущей игры\n");
        printf("6. Выйти из игры\n");
        printf("7. Выход из программы\n");
        printf("Выберите действие: ");

        int choice;
        scanf("%d", &choice);
        getchar();

        switch (choice)
        {
        case 1:
        {
            char game_name[MAX_GAME_NAME];
            char secret_word[MAX_WORD_LENGTH];
            int max_players;

            printf("Введите название игры: ");
            fgets(game_name, sizeof(game_name), stdin);
            game_name[strcspn(game_name, "\n")] = 0;

            printf("Введите загаданное слово: ");
            fgets(secret_word, sizeof(secret_word), stdin);
            secret_word[strcspn(secret_word, "\n")] = 0;

            printf("Введите количество игроков (макс %d): ",
MAX_PLAYERS_PER_GAME);
            scanf("%d", &max_players);
            getchar();

            create_game_client(game_name, secret_word, max_players);
            break;
```

```
        }

        case 2:
        {
            char game_name[MAX_GAME_NAME];
            char player_name[MAX_PLAYER_NAME];

            list_games_client();

            printf("Введите название игры для присоединения: ");
            fgets(game_name, sizeof(game_name), stdin);
            game_name[strcspn(game_name, "\n")] = 0;

            printf("Введите ваше имя: ");
            fgets(player_name, sizeof(player_name), stdin);
            player_name[strcspn(player_name, "\n")] = 0;

            join_game_client(game_name, player_name);
            break;
        }

        case 3:
        {
            if (client_game_id == -1)
            {
                printf("Вы не в игре. Присоединитесь к игре сначала.\n");
                break;
            }

            char guess[MAX_WORD_LENGTH];
            printf("Введите ваше предположение: ");
            fgets(guess, sizeof(guess), stdin);
            guess[strcspn(guess, "\n")] = 0;

            make_guess_client(current_game_name, client_player_id, guess);
            break;
        }

        case 4:
            list_games_client();
            break;

        case 5:
            if (client_game_id == -1)
            {
                printf("Вы не в игре.\n");
            }
            else
            {
                print_game_status(current_game_name);
            }
```

```
            break;

        case 6:
            if (client_game_id == -1)
            {
                printf("Вы не в игре.\n");
            }
            else
            {
                leave_game_client(current_game_name, client_player_id);
            }
            break;

        case 7:
            printf("Выход...\n");
            if (shm_data)
            {
                munmap(shm_data, SHM_SIZE);
            }
            if (shm_fd != -1)
            {
                close(shm_fd);
            }
            return;

        default:
            printf("Неверный выбор. Попробуйте снова.\n");
        }
    }
}

int main()
{
    client_main_menu();
    return 0;
}
```

game_logic.c

```
#include "game_common.h"
#include <string.h>
#include <time.h>
#include <stdio.h>

void init_game(Game *game, const char *name, const char *secret_word, int
max_players)
{
    memset(game, 0, sizeof(Game));

    strncpy(game->game_name, name, MAX_GAME_NAME - 1);
    strncpy(game->secret_word, secret_word, MAX_WORD_LENGTH - 1);
```

```
    game->max_players = (max_players > MAX_PLAYERS_PER_GAME) ?
MAX_PLAYERS_PER_GAME : max_players;
    game->current_players = 0;
    game->is_active = true;
    game->is_game_over = false;
    game->winner_id = -1;
    game->total_moves = 0;
    game->words_tried = 0;

    game->created_at = time(NULL);
    game->last_move_at = time(NULL);

    for (int i = 0; i < MAX_PLAYERS_PER_GAME; i++)
    {
        game->players[i].player_id = i;
        game->players[i].is_connected = false;
        game->players[i].is_active = false;
        game->players[i].score = 0;
        game->players[i].attempts_count = 0;
        game->players[i].last_activity = 0;
    }
}

bool add_player_to_game(Game *game, const char *player_name, int *player_id)
{
    if (!game->is_active || game->is_game_over)
    {
        return false;
    }

    if (game->current_players >= game->max_players)
    {
        return false;
    }

    for (int i = 0; i < game->current_players; i++)
    {
        if (strcmp(game->players[i].name, player_name) == 0)
        {
            if (game->players[i].is_connected)
            {
                return false;
            }
            else
            {
                game->players[i].is_connected = true;
                game->players[i].is_active = true;
                game->players[i].last_activity = time(NULL);
                *player_id = i;
                return true;
            }
```

```
        }
    }

    int new_id = game->current_players;
    Player *new_player = &game->players[new_id];

    strncpy(new_player->name, player_name, MAX_PLAYER_NAME - 1);
    new_player->player_id = new_id;
    new_player->score = 0;
    new_player->attempts_count = 0;
    new_player->is_connected = true;
    new_player->is_active = true;
    new_player->last_activity = time(NULL);

    game->current_players++;
    *player_id = new_id;

    return true;
}

bool remove_player_from_game(Game *game, int player_id)
{
    if (player_id < 0 || player_id >= game->current_players)
    {
        return false;
    }

    if (!game->players[player_id].is_connected)
    {
        return false;
    }

    game->players[player_id].is_connected = false;
    game->players[player_id].is_active = false;

    return true;
}

void calculate_bulls_cows(const char *secret, const char *guess, int *bulls, int
*cows)
{
    *bulls = 0;
    *cows = 0;

    int secret_len = strlen(secret);
    int guess_len = strlen(guess);

    if (secret_len != guess_len)
    {
        return;
    }
```

```
    int secret_count[26] = {0};
    int guess_count[26] = {0};

    for (int i = 0; i < secret_len; i++)
    {
        if (secret[i] == guess[i])
        {
            (*bulls)++;
        }
        else
        {
            if (secret[i] >= 'a' && secret[i] <= 'z')
            {
                secret_count[secret[i] - 'a']++;
            }
            if (guess[i] >= 'a' && guess[i] <= 'z')
            {
                guess_count[guess[i] - 'a']++;
            }
        }
    }

    for (int i = 0; i < 26; i++)
    {
        *cows += (secret_count[i] < guess_count[i]) ? secret_count[i] :
guess_count[i];
    }
}

bool is_game_joinable(const Game *game)
{
    return (game->is_active &&
            !game->is_game_over &&
            game->current_players < game->max_players);
}

void broadcast_game_update(Game *game, SharedMemoryData *shm_data)
{
    time_t now = time(NULL);

    snprintf(shm_data->server_status, sizeof(shm_data->server_status),
             "Game '%s': %d/%d players, last update: %ld",
             game->game_name, game->current_players,
             game->max_players, now);
}
```

server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <time.h>
#include <pthread.h>
#include <signal.h>
#include "game_common.h"

static SharedMemoryData *shm_data = NULL;
static int shm_fd = -1;
static volatile bool server_running = true;
static pthread_t cleanup_thread;

SharedMemoryData *init_shared_memory()
{
    shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1)
    {
        perror("shm_open");
        return NULL;
    }

    if (ftruncate(shm_fd, SHM_SIZE) == -1)
    {
        perror("ftruncate");
        close(shm_fd);
        return NULL;
    }

    shm_data = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
                    MAP_SHARED, shm_fd, 0);
    if (shm_data == MAP_FAILED)
    {
        perror("mmap");
        close(shm_fd);
        return NULL;
    }

    pthread_mutexattr_t mutex_attr;
    pthread_mutexattr_init(&mutex_attr);
    pthread_mutexattr_setpshared(&mutex_attr, PTHREAD_PROCESS_SHARED);
    pthread_mutex_init(&shm_data->shm_mutex, &mutex_attr);

    for (int i = 0; i < MAX_GAMES; i++)
    {
        Game *game = &shm_data->games[i];
        pthread_mutexattr_t game_mutex_attr;
        pthread_mutexattr_init(&game_mutex_attr);
        pthread_mutexattr_setpshared(&game_mutex_attr, PTHREAD_PROCESS_SHARED);
        pthread_mutex_init(&game->game_mutex, &game_mutex_attr);
```

```
        pthread_condattr_t cond_attr;
        pthread_condattr_init(&cond_attr);
        pthread_condattr_setpshared(&cond_attr, PTHREAD_PROCESS_SHARED);
        pthread_cond_init(&game->game_cond, &cond_attr);
    }

    shm_data->active_games = 0;
    strcpy(shm_data->server_status, "Server running");

    printf("Shared memory initialized successfully\n");
    return shm_data;
}

void *cleanup_thread_func(void *arg)
{
    (void)arg;

    while (server_running)
    {
        sleep(30);

        pthread_mutex_lock(&shm_data->shm_mutex);

        time_t now = time(NULL);

        for (int i = 0; i < MAX_GAMES; i++)
        {
            Game *game = &shm_data->games[i];

            if (!game->is_active)
                continue;

            pthread_mutex_lock(&game->game_mutex);

            for (int j = 0; j < game->current_players; j++)
            {
                Player *player = &game->players[j];

                if (player->is_connected &&
                    (now - player->last_activity) > 300)
                {
                    printf("Player '%s' timed out from game '%s'\n",
                           player->name, game->game_name);

                    player->is_connected = false;
                    player->is_active = false;

                    int active_count = 0;
                    for (int k = 0; k < game->current_players; k++)
                    {
```

```
                        if (game->players[k].is_connected)
                        {
                            active_count++;
                        }
                    }

                    if (active_count == 0)
                    {
                        game->is_active = false;
                        shm_data->active_games--;
                    }
                }
            }

            if (game->is_game_over && (now - game->last_move_at) > 3600)
            {
                printf("Cleaning up finished game '%s'\n", game->game_name);
                game->is_active = false;
                shm_data->active_games--;
            }

            pthread_mutex_unlock(&game->game_mutex);
        }

        pthread_mutex_unlock(&shm_data->shm_mutex);
    }

    return NULL;
}

int main()
{
    printf("Starting Bulls and Cows Server...\n");

    if (!init_shared_memory())
    {
        fprintf(stderr, "Failed to initialize shared memory\n");
        return 1;
    }

    pthread_create(&cleanup_thread, NULL, cleanup_thread_func, NULL);

    printf("Server is running. Press Ctrl+C to stop.\n");
    printf("Active games: %d\n", shm_data->active_games);

    while (server_running)
    {
        sleep(1);

        static time_t last_stats = 0;
        time_t now = time(NULL);
```

```
        if (now - last_stats >= 10)
        {
            pthread_mutex_lock(&shm_data->shm_mutex);
            printf("\n=== Server Status ===\n");
            printf("Active games: %d\n", shm_data->active_games);

            for (int i = 0; i < MAX_GAMES; i++)
            {
                Game *game = &shm_data->games[i];
                if (game->is_active)
                {
                    printf("  Game '%s': %d/%d players, %s\n",
                           game->game_name, game->current_players,
                           game->max_players,
                           game->is_game_over ? "FINISHED" : "ACTIVE");
                }
            }
            printf("====================\n");
            pthread_mutex_unlock(&shm_data->shm_mutex);

            last_stats = now;
        }
    }

    printf("Shutting down server...\n");

    server_running = false;
    pthread_join(cleanup_thread, NULL);

    if (shm_data)
    {
        munmap(shm_data, SHM_SIZE);
    }
    if (shm_fd != -1)
    {
        close(shm_fd);
        shm_unlink(SHM_NAME);
    }

    printf("Server stopped\n");
    return 0;
}
```

Протокол работы программы

Тестирование:

Терминал 1: Запуск сервера

$ ./server

Starting Bulls and Cows Server...

Shared memory initialized successfully

Server is running. Press Ctrl+C to stop.

Active games: 0

=== Server Status ===

Active games: 0

====================

Терминал 2: Клиент 1 создает игру

$ ./client

=== Клиент 'Быки и Коровы' ===

Подключение к серверу установлено

Главное меню:

1. Создать новую игру

2. Присоединиться к существующей игре

3. Сделать предположение (если в игре)

4. Показать список игр

5. Показать статус текущей игры

6. Выйти из игры

7. Выход из программы

Выберите действие: 1

Введите название игры: Game1

Введите загаданное слово: apple

Введите количество игроков (макс 4): 2

Введите ваше имя: Player1

Игра 'Game1' создана успешно. Ваш ID: 0

Терминал 3: Клиент 2 присоединяется к игре

$ ./client

=== Клиент 'Быки и Коровы' ===

Подключение к серверу установлено

Главное меню:

Выберите действие: 4

=== Список активных игр ===

Игра: Game1

  Игроков: 1/2

  Статус: Активна

Выберите действие: 2

Введите название игры для присоединения: Game1

Введите ваше имя: Player2

Вы присоединились к игре 'Game1' как игрок 1

Текущие игроки в игре:

- Player1 (ID: 0)

- Player2 (ID: 1)

Терминал 2: Игрок 1 делает предположение

Выберите действие: 3

Введите ваше предположение: apply

Результат: Быки: 3, Коровы: 1

Терминал 3: Игрок 2 делает предположение

Выберите действие: 3

Введите ваше предположение: angle

Результат: Быки: 2, Коровы: 1

Терминал 2: Игрок 1 угадывает слово

Выберите действие: 3

Введите ваше предположение: apple

Результат: Быки: 5, Коровы: 0

ПОЗДРАВЛЯЕМ! Вы угадали слово 'apple'!

Игра завершена. Победитель: Player1

Вывод strace:

client.c:

root@LAPTOP-TGRFAHQT:/mnt/c/Users/mkoti/os_laba1/kp# strace ./client

execve("./client", ["./client"], 0x7ffe5bb85c40 /* 29 vars */) = 0

brk(NULL) = 0x55880a789000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc04f2e950) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f498b6cb000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=20892, ...}, AT_EMPTY_PATH)
= 0

mmap(NULL, 20892, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f498b6c5000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC)
= 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) =
832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784,
64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848)
= 48

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32$\230\266\235"..., 68,
896) = 68

```
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...},
AT_EMPTY_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784,
64) = 784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f498b49c000

mprotect(0x7f498b4c4000, 2023424, PROT_NONE) = 0

mmap(0x7f498b4c4000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f498b4c4000

mmap(0x7f498b659000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f498b659000

mmap(0x7f498b6b2000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f498b6b2000

mmap(0x7f498b6b8000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f498b6b8000

close(3)                        = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f498b499000

arch_prctl(ARCH_SET_FS, 0x7f498b499740) = 0

set_tid_address(0x7f498b499a10)         = 2722

set_robust_list(0x7f498b499a20, 24)     = 0

rseq(0x7f498b49a0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f498b6b2000, 16384, PROT_READ) = 0

mprotect(0x5587cca06000, 4096, PROT_READ) = 0

mprotect(0x7f498b705000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f498b6c5000, 20892)           = 0

newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x2), ...},
AT_EMPTY_PATH) = 0

getrandom("\x18\x84\x7b\x92\xe2\x96\xd3\x21", 8, GRND_NONBLOCK) = 8

brk(NULL)                               = 0x55880a789000

brk(0x55880a7aa000)                     = 0x55880a7aa000

write(1, "\n", 1

)                       = 1
```

write(1, "=== \320\232\320\273\320\270\320\265\320\275\321\202 '\320\221\321\213\320\272\320\270 \320\270 \320\232"..., 48=== Клиент 'Быки и Коровы' ===

) = 48

openat(AT_FDCWD, "/dev/shm/bulls_cows_shm", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f498b299000

write(1, "\320\237\320\276\320\264\320\272\320\273\321\216\321\207\320\265\320\275\320\270\320\265 \320\272 \321\201\320\265\321\200"..., 64Подключение к серверу установлено

) = 64

write(1, "\n", 1

) = 1

write(1, "\320\223\320\273\320\260\320\262\320\275\320\276\320\265 \320\274\320\265\320\275\321\216:\n", 25Главное меню:

) = 25

write(1, "1. \320\241\320\276\320\267\320\264\320\260\321\202\321\214 \320\275\320\276\320\262\321\203\321\216 \320\270\320"..., 381. Создать новую игру

) = 38

write(1, "2. \320\237\321\200\320\270\321\201\320\276\320\265\320\264\320\270\320\275\320\270\321 \202\321\214\321\201\321\217 "..., 692. Присоединиться к существующей игре

) = 69

write(1, "3. \320\241\320\264\320\265\320\273\320\260\321\202\321\214 \320\277\321\200\320\265\320\264\320\277\320\276\320\273"..., 683. Сделать предположение (если в игре)

) = 68

write(1, "4. \320\237\320\276\320\272\320\260\320\267\320\260\321\202\321\214 \321\201\320\277\320\270\321\201\320\276\320\272"..., 404. Показать список игр

) = 40

write(1, "5. \320\237\320\276\320\272\320\260\320\267\320\260\321\202\321\214 \321\201\321\202\320\260\321\202\321\203\321\201"..., 575. Показать статус текущей игры

) = 57

write(1, "6. \320\222\321\213\320\271\321\202\320\270 \320\270\320\267 \320\270\320\263\321\200\321\213\n", 286. Выйти из игры

) = 28

write(1, "7. \320\222\321\213\321\205\320\276\320\264 \320\270\320\267 \320\277\321\200\320\276\320\263\321\200\320\260\320"..., 387. Выход из программы

) = 38

newfstatat(0, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x2), ...}, AT_EMPTY_PATH) = 0

write(1, "\320\222\321\213\320\261\320\265\321\200\320\270\321\202\320\265 \320\264\320\265\320\271\321\201\321\202\320\262\320\270\320"..., 35Выберите действие:) = 35

read(0, 1

"1\n", 1024)   = 2

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\275\320\260\320\267\320\262\320\260\320\275\320\270\320\265 "..., 42Введите название игры:) = 42

read(0, Мяу

"\320\234\321\217\321\203\n", 1024) = 7

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\267\320\260\320\263\320\260\320\264\320\260\320\275\320\275\320"..., 48Введите загаданное слово:) = 48

read(0, курсач

"\320\272\321\203\321\200\321\201\320\260\321\207\n", 1024) = 13

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\272\320\276\320\273\320\270\321\207\320\265\321\201\321\202\320"..., 65Введите количество игроков (макс 4):) = 65

read(0, 2

"2\n", 1024)   = 2

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\262\320\260\321\210\320\265 \320\270\320\274\321\217: ", 32Введите ваше имя:) = 32

read(0, маша

"\321\214\320\274\320\260\321\210\320\260\n", 1024) = 11

write(1, "\320\230\320\263\321\200\320\260 '\320\234\321\217\321\203' \321\201\320\276\320\267\320\264\320\260\320\275\320\260"..., 62Игра 'Мяу' создана успешно. Ваш ID: 0

) = 62

write(1, "\n", 1

) = 1

write(1, "\320\223\320\273\320\260\320\262\320\275\320\276\320\265 \320\274\320\265\320\275\321\216:\n", 25Главное меню:

) = 25

write(1, "1. \320\241\320\276\320\267\320\264\320\260\321\202\321\214 \320\275\320\276\320\262\321\203\321\216 \320\270\320"..., 381. Создать новую игру

) = 38

write(1, "2. \320\237\321\200\320\270\321\201\320\276\320\265\320\264\320\270\320\275\320\270\321 \202\321\214\321\201\321\217 "..., 692. Присоединиться к существующей игре

) = 69

write(1, "3. \320\241\320\264\320\265\320\273\320\260\321\202\321\214 \320\277\321\200\320\265\320\264\320\277\320\276\320\273"..., 683. Сделать предположение (если в игре)

) = 68

write(1, "4. \320\237\320\276\320\272\320\260\320\267\320\260\321\202\321\214 \321\201\320\277\320\270\321\201\320\276\320\272"..., 404. Показать список игр

) = 40

write(1, "5. \320\237\320\276\320\272\320\260\320\267\320\260\321\202\321\214 \321\201\321\202\320\260\321\202\321\203\321\201"..., 575. Показать статус текущей игры

) = 57

write(1, "6. \320\222\321\213\320\271\321\202\320\270 \320\270\320\267 \320\270\320\263\321\200\321\213\n", 286. Выйти из игры

) = 28

write(1, "7. \320\222\321\213\321\205\320\276\320\264 \320\270\320\267 \320\277\321\200\320\276\320\263\321\200\320\260\320"..., 387. Выход из программы

) = 38

write(1, "\320\222\321\213\320\261\320\265\321\200\320\270\321\202\320\265 \320\264\320\265\320\271\321\201\321\202\320\262\320\270\320"..., 35Выберите действие:) = 35

read(0, 3

"3\n", 1024) = 2

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\262\320\260\321\210\320\265 \320\277\321\200\320\265\320\264"..., 52Введите ваше предположение:) = 52

read(0, курсач

"\320\272\321\203\321\200\321\201\320\260\321\207\n", 1024) = 13

write(1,
"\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202:
\320\221\321\213\320\272\320\270: 12"..., 50Результат: Быки: 12, Коровы: 0

) = 50

write(1, "\n", 1

) = 1

write(1,
"\320\237\320\236\320\227\320\224\320\240\320\220\320\222\320\233\320\257\320\225\32
0\234! \320\222\321\213 \321\203\320"..., 71ПОЗДРАВЛЯЕМ! Вы угадали слово
'курсач'!

) = 71

write(1, "\320\230\320\263\321\200\320\260
\320\267\320\260\320\262\320\265\321\200\321\210\320\265\320\275\320\260.
\320\237\320"..., 62Игра завершена. Победитель: ьмаша

) = 62

write(1, "\n", 1

) = 1

write(1, "\320\223\320\273\320\260\320\262\320\275\320\276\320\265
\320\274\320\265\320\275\321\216:\n", 25Главное меню:

) = 25

write(1, "1. \320\241\320\276\320\267\320\264\320\260\321\202\321\214
\320\275\320\276\320\262\321\203\321\216 \320\270\320"..., 381. Создать новую игру

) = 38

write(1, "2.
\320\237\321\200\320\270\321\201\320\276\320\265\320\264\320\270\320\275\320\270\321
\202\321\214\321\201\321\217 "..., 692. Присоединиться к существующей игре

) = 69

write(1, "3. \320\241\320\264\320\265\320\273\320\260\321\202\321\214
\320\277\321\200\320\265\320\264\320\277\320\276\320\273"..., 683. Сделать
предположение (если в игре)

) = 68

write(1, "4. \320\237\320\276\320\272\320\260\320\267\320\260\321\202\321\214
\321\201\320\277\320\270\321\201\320\276\320\272"..., 404. Показать список игр

) = 40

write(1, "5. \320\237\320\276\320\272\320\260\320\267\320\260\321\202\321\214 \321\201\321\202\320\260\321\202\321\203\321\201"..., 575. Показать статус текущей игры

) = 57

write(1, "6. \320\222\321\213\320\271\321\202\320\270 \320\270\320\267 \320\270\320\263\321\200\321\213\n", 286. Выйти из игры

) = 28

write(1, "7. \320\222\321\213\321\205\320\276\320\264 \320\270\320\267 \320\277\321\200\320\276\320\263\321\200\320\260\320"..., 387. Выход из программы

) = 38

write(1, "\320\222\321\213\320\261\320\265\321\200\320\270\321\202\320\265 \320\264\320\265\320\271\321\201\321\202\320\262\320\270\320"..., 35Выберите действие:) = 35

read(0, 7

"7\n", 1024)   = 2

write(1, "\320\222\321\213\321\205\320\276\320\264...\n", 14Выход...

) = 14

munmap(0x7f498b299000, 2097152)   = 0

close(3)   = 0

exit_group(0)   = ?

+++ exited with 0 +++

root@LAPTOP-TGRFAHQT:/mnt/c/Users/mkoti/os_laba1/kp#

server.c

execve("./server", ["./server"], 0x7ffc5b22f6d8 /* 58 vars */) = 0

brk(NULL)   = 0x55b4b3774000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcd54e3ed0) = -1 EINVAL (Invalid argument)

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=113611, ...}) = 0

mmap(NULL, 113611, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f0c0d5a4000

close(3)   = 0

```
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0",
O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 l\0\0\0\0\0\0"..., 832) = 832

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\v\311\337\300l\337\350\206\276\350\t\233\360\370\364%"
..., 68, 792) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f0c0d5a2000

mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f0c0d57c000

mmap(0x7f0c0d583000, 69632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7f0c0d583000

mmap(0x7f0c0d594000, 20480, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18000) = 0x7f0c0d594000

mmap(0x7f0c0d599000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c000) = 0x7f0c0d599000

mmap(0x7f0c0d59b000, 13400, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f0c0d59b000

close(3)                                = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC)
= 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) =
832

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\367\262\364\t\227z\347\211<\336\326\232n\264N"..., 68,
792) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2147480, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f0c0d57a000

mmap(NULL, 2173080, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f0c0d367000

mprotect(0x7f0c0d392000, 2093056, PROT_NONE) = 0

mmap(0x7f0c0d392000, 1667072, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2b000) = 0x7f0c0d392000

mmap(0x7f0c0d522000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bb000) = 0x7f0c0d522000
```

mmap(0x7f0c0d57c000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x214000) = 0x7f0c0d57c000

mmap(0x7f0c0d582000, 5288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f0c0d582000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f0c0d364000

arch_prctl(ARCH_SET_FS, 0x7f0c0d364740) = 0

mprotect(0x7f0c0d57c000, 16384, PROT_READ) = 0

mprotect(0x55b4b1752000, 4096, PROT_READ) = 0

mprotect(0x7f0c0d5c9000, 4096, PROT_READ) = 0

munmap(0x7f0c0d5a4000, 113611) = 0

shm_open("/bulls_cows_shm", O_RDWR|O_CREAT, 0666) = 3

ftruncate(3, 2097152) = 0

mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f0c0b364000

close(3) = 0

pthread_mutexattr_init({0x7ffcd54e1d80}) = 0

pthread_mutexattr_setpshared({0x7ffcd54e1d80}, 1) = 0

pthread_mutex_init({0x7f0c0b364000}, {0x7ffcd54e1d80}) = 0

pthread_mutexattr_destroy({0x7ffcd54e1d80}) = 0

write(1, "Starting Bulls and Cows Server.."..., 36) = 36

write(1, "Shared memory initialized succes"..., 36) = 36

write(1, "Server is running. Press Ctrl+C "..., 44) = 44

write(1, "Active games: 0\n", 16) = 16

pthread_create({0x7f0c0b365040}, NULL, 0x55b4b168e1a0, NULL) = 0

fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0

brk(NULL) = 0x55b4b3774000

brk(0x55b4b3795000) = 0x55b4b3795000

read(0,

…

КЛЮЧЕВЫЕ СИСТЕМНЫЕ ВЫЗОВЫ

openat() – открытие/создание shared memory объекта

mmap() – маппинг shared memory в адресное пространство процесса

munmap() – удаление маппинга памяти

close() – закрытие файлового дескриптора shared memory

read() – чтение данных из стандартного ввода

write() – запись данных в стандартный вывод

exit_group() – завершение процесса

brk() – управление размером кучи процесса

```
0 +++
```

Вывод

В ходе выполнения курсового проекта была успешно реализована многопользовательская консольная игра "Быки и коровы" с использованием shared memory для межпроцессного взаимодействия. Проект демонстрирует практическое применение технологий операционных систем: POSIX Shared Memory для обмена данными между процессами, мьютексов с атрибутом PTHREAD_PROCESS_SHARED для синхронизации доступа к общим ресурсам, и потоков для фоновых задач. Все требования варианта выполнены полностью, включая поддержку нескольких игроков в одной игре, указание количества игроков при создании, продолжение игры при выходе участников и общение через memory map. Основной сложностью была правильная организация синхронизации доступа к shared memory из нескольких клиентов, что было решено использованием иерархии мьютексов.