МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА (№75)

Лабораторная работа на тему:
Программная реализация проекта "Серверная игра: крестики-нолики"

Выполнил студент:
Тухватуллин Ислам Рамилевич
Группа: С23-722
Проверил:
Курасов Юрий Викторович
Оценка:________________
Дата:_________________
Подпись:_______________

Москва, 2025

Тухватуллин И. Р. С23-722

Содержание

1.Задача

Написать многопоточное клиент-серверное приложение, позволяющее играть в игру “Крестики-нолики”.

Тухватуллин И. Р. С23-722

2.Теоретическая часть

Алгоритм использования

1. Запускаем сервер, который работает на порте 2000.

2. Запускаем клиентское приложение и вводим никнейм.

3. Открывается меню со списком возможностей:

    1 - Показать пользователей онлайн

    invite <nick> - Пригласить игрока в игру

    check - Проверка на наличие приглашений

    accept - Принять приглашение

    reject - Отклонить приглашение

    q - Завершить работу клиента

    "WHILE IN A GAME => ONLY: move r c, checkdesk, or q\n"

4. Проверяем список игроков онлайн, выбрав соответствующее поле в меню.

5. Приглашаем свободного пользователя в игру.

6. Ждём ответа.

7. При положительном ответе начинаем игру.

9. Заканчиваем игру.

Тухватуллин И. Р. С23-722

Системные вызовы

- send() – Отправка сообщения в сокет
- strcmp() - Сравнивает две строки
- recv() – Получение сообщения из сокета
- close() – Закрывает файловый дескриптор
- accept() – Принятие соединения на сокете
- bind() – Привязка адреса к сокету
- listen() – Ожидание подключения
- socket() – Создание сокета сокета

Дополнительные функции

static int findFreePlayerSlot() // Поиск свободного слота

static bool isNameTaken(const char *name) // Проверка никнейма

static int addPlayer(const char *name) // Добавление игрока

static void removePlayer(int idx) // Удаление игрока

static int findPlayerByName(const char *name) // Поиск по имени

static void showOnlinePlayers(int clientSock) // Игроки онлайн

static int createGame(int pX, int pO) // Создание игры

static void showBoard(int clientSock, int gId) // Показать поле

static char checkGameOver(int gId) // Проверка на окончание игры

static void endGame(int gId, char winnerSymbol) // Окончание игры

static void makeMove(int clientSock, int myIndex, int row, int col) // Сделать ход

static void checkDesk(int clientSock, int myIndex) // Проверка доски

static void handleClientCommands(int clientSock, int myIndex) // Обработка команд клиента

3.Практическая часть

Клиентская сторона

Запуск клиента следует производить с вводом IP-адреса и порта. Программа создаст клиентский сокет и отправит запрос на соединение. После успешного соединения происходит регистрация пользователя. Далее пользователь получает доступ ко всему функционалу программы. Чтобы закрыть программы следует ввести команду “q”.

Серверная сторона

При запуске серверное приложение создаёт сокет и привязывает его к адресу 127.0.0.1:2000, затем ждёт запросов на соединения. Далее начнется обработка пользовательского ввода.

Блок-схемы программы

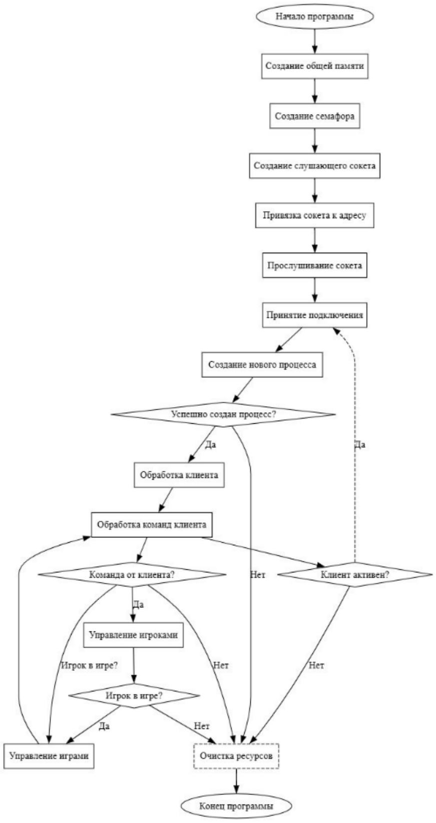


Рис 1. Блок-схема server.c

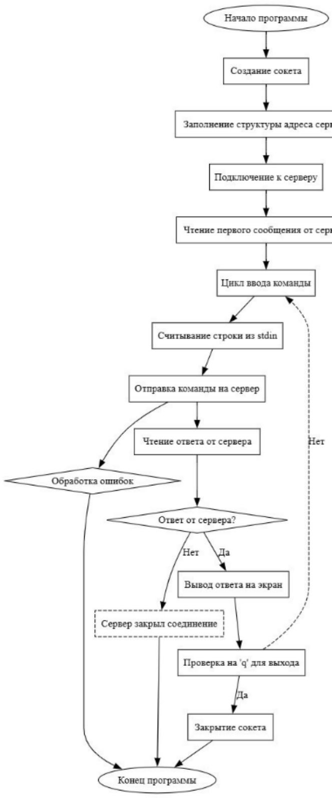


Рис 2. Блок-схема client.c

4.Приложения

Программный код

Исходный код сервера

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/stat.h>

#define SERVER_PORT 2000
#define MAX_PLAYERS 10
#define MAX_GAMES   10
#define BOARD_SIZE  3

#define HELLO_MSG "PLEASE ENTER YOUR LOGIN:\n"
#define MENU "\nAVAILABLE COMMANDS:\n" \
        "  1             - SHOW ONLINE PLAYERS\n" \
        "  invite <nick>  - INVITE PLAYER TO GAME\n" \
        "  check          - CHECK IF YOU HAVE INVITATIONS\n" \
        "  accept         - ACCEPT INVITATION\n" \
        "  reject         - REJECT INVITATION\n" \
        "  q             - QUIT\n\n" \
        "WHILE IN A GAME => ONLY: move r c, checkdesk, or q\n"
```

```
typedef struct {
    bool active;
    char name[64];

    int  gameId;
    char symbol;

    bool hasInvitation;
    int  invitedBy;
} Player;

typedef struct {
    bool used;
    int  playerX;
    int  playerO;
    int  turn;
    char board[BOARD_SIZE][BOARD_SIZE];

    int  lastMoveRow;
    int  lastMoveCol;
    int  lastMoveBy;
} Game;

typedef struct {
    Player players[MAX_PLAYERS];
    Game   games[MAX_GAMES];
} SharedData;

static int shm_id  = -1;
static int sem_id  = -1;
static SharedData *g_data = NULL;
static void sem_lock() {
    struct sembuf op[1];
    op[0].sem_num = 0;
    op[0].sem_op  = -1;
    op[0].sem_flg = 0;
    semop(sem_id, op, 1);
}
static void sem_unlock() {
    struct sembuf op[1];
    op[0].sem_num = 0;
    op[0].sem_op  = 1;
    op[0].sem_flg = 0;
    semop(sem_id, op, 1);
}

static int findFreePlayerSlot() {
    for (int i = 0; i < MAX_PLAYERS; i++) {
        if (!g_data->players[i].active) {
            return i;
        }
    }
    return -1;
}

static bool isNameTaken(const char *name) {
    for (int i = 0; i < MAX_PLAYERS; i++) {
        if (g_data->players[i].active &&
            strcmp(g_data->players[i].name, name) == 0) {
            return true;
        }
    }
    return false;
}
```

```
static int addPlayer(const char *name) {
    int idx = findFreePlayerSlot();
    if (idx >= 0) {
        g_data->players[idx].active = true;
        strncpy(g_data->players[idx].name, name,
                sizeof(g_data->players[idx].name) - 1);
        g_data->players[idx].name[sizeof(g_data->players[idx].name)-1] =
'\0';

        g_data->players[idx].gameId = -1;
        g_data->players[idx].symbol = '\0';
        g_data->players[idx].hasInvitation = false;
        g_data->players[idx].invitedBy = -1;
    }
    return idx;
}

static void removePlayer(int idx) {
    if (idx < 0 || idx >= MAX_PLAYERS) return;

    int gId = g_data->players[idx].gameId;
    if (gId >= 0 && gId < MAX_GAMES && g_data->games[gId].used) {
        g_data->games[gId].used = false;
    }

    g_data->players[idx].active = false;
    g_data->players[idx].name[0] = '\0';
    g_data->players[idx].gameId = -1;
    g_data->players[idx].symbol = '\0';
    g_data->players[idx].hasInvitation = false;
    g_data->players[idx].invitedBy = -1;
}
```

```
static int findPlayerByName(const char *name) {
    for (int i = 0; i < MAX_PLAYERS; i++) {
        if (g_data->players[i].active &&
            strcmp(g_data->players[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}

static void showOnlinePlayers(int clientSock) {
    char reply[2048];
    int offset = 0;

    offset += snprintf(reply + offset, sizeof(reply) - offset,
                       "===== ONLINE PLAYERS =====\n");

    for (int i = 0; i < MAX_PLAYERS; i++) {
        if (g_data->players[i].active) {
            offset += snprintf(reply + offset, sizeof(reply) - offset,
                               "> %s\n", g_data->players[i].name);
        }
    }
    offset += snprintf(reply + offset, sizeof(reply) - offset,
                       "==========================\n");

    send(clientSock, reply, offset, 0);
}

static int createGame(int pX, int pO) {
    for (int g = 0; g < MAX_GAMES; g++) {
        if (!g_data->games[g].used) {
            g_data->games[g].used    = true;
            g_data->games[g].playerX = pX;
            g_data->games[g].playerO = pO;
            g_data->games[g].turn    = 0;

            for (int r = 0; r < BOARD_SIZE; r++) {
                for (int c = 0; c < BOARD_SIZE; c++) {
                    g_data->games[g].board[r][c] = ' ';
                }
            }
            g_data->games[g].lastMoveRow = -1;
            g_data->games[g].lastMoveCol = -1;
            g_data->games[g].lastMoveBy  = -1;

            g_data->players[pX].gameId = g;
            g_data->players[pX].symbol = 'X';
            g_data->players[pO].gameId = g;
            g_data->players[pO].symbol = 'O';

            return g;
        }
    }
    return -1;
}

static char checkGameOver(int gId) {
    if (gId < 0 || gId >= MAX_GAMES) return ' ';
    if (!g_data->games[gId].used) return ' ';

    char (*board)[BOARD_SIZE] = g_data->games[gId].board;

    for (int r = 0; r < BOARD_SIZE; r++) {
        if (board[r][0] != ' ' &&
            board[r][0] == board[r][1] &&
            board[r][1] == board[r][2]) {
            return board[r][0];
        }
    }
    for (int c = 0; c < BOARD_SIZE; c++) {
        if (board[0][c] != ' ' &&
            board[0][c] == board[1][c] &&
            board[1][c] == board[2][c]) {
            return board[0][c];
        }
    }
    if (board[0][0] != ' ' &&
        board[0][0] == board[1][1] &&
        board[1][1] == board[2][2]) {
        return board[0][0];
    }
    if (board[0][2] != ' ' &&
        board[0][2] == board[1][1] &&
        board[1][1] == board[2][0]) {
        return board[0][2];
    }

    bool full = true;
    for (int r = 0; r < BOARD_SIZE; r++) {
        for (int c = 0; c < BOARD_SIZE; c++) {
            if (board[r][c] == ' ') {
                full = false;
                break;
            }
        }
        if (!full) break;
    }
    if (full) return 'D';

    return ' ';
}

static void endGame(int gId, char winnerSymbol) {
    if (gId < 0 || gId >= MAX_GAMES) return;
    if (!g_data->games[gId].used) return;

    g_data->games[gId].used = false;

    int px = g_data->games[gId].playerX;
    int po = g_data->games[gId].playerO;
    g_data->players[px].gameId = -1;
    g_data->players[px].symbol = '\0';
    g_data->players[po].gameId = -1;
    g_data->players[po].symbol = '\0';
}

static void forfeitGame(int quitter) {
    int gId = g_data->players[quitter].gameId;
    if (gId < 0 || gId >= MAX_GAMES) return;
    if (!g_data->games[gId].used) return;
    endGame(gId, 'F');
}

static void makeMove(int clientSock, int myIndex, int row, int col) {
    int gId = g_data->players[myIndex].gameId;
    if (gId < 0 || !g_data->games[gId].used) {
        const char *m = "YOU ARE NOT IN A GAME.\n";
        send(clientSock, m, strlen(m), 0);
        return;
    }
    Game *gm = &g_data->games[gId];
    char mySymbol = g_data->players[myIndex].symbol;
    bool myTurn = ((gm->turn == 0 && mySymbol == 'X') ||
                   (gm->turn == 1 && mySymbol == 'O'));

    if (!myTurn) {
        const char *m = "NOT YOUR TURN!\n";
        send(clientSock, m, strlen(m), 0);
        return;
    }

    if (row < 1 || row > BOARD_SIZE || col < 1 || col > BOARD_SIZE) {
        const char *m = "INVALID MOVE (row col must be 1..3)\n";
        send(clientSock, m, strlen(m), 0);
        return;
    }
    row--;
    col--;
    if (gm->board[row][col] != ' ') {
        const char *m = "CELL IS NOT EMPTY!\n";
        send(clientSock, m, strlen(m), 0);
        return;
    }

    gm->board[row][col] = mySymbol;
    gm->lastMoveRow = row;
    gm->lastMoveCol = col;
    gm->lastMoveBy  = myIndex;

    char r = checkGameOver(gId);
    if (r == 'X' || r == 'O') {
        int px = gm->playerX;
        int po = gm->playerO;
        int winnerIdx = (r == 'X') ? px : po;

        char winnerName[64];
        strncpy(winnerName, g_data->players[winnerIdx].name,
sizeof(winnerName)-1);
        winnerName[sizeof(winnerName)-1] = '\0';

        char buf[128];
        snprintf(buf, sizeof(buf), "GAME OVER. WINNER: %s\n",
winnerName);
        send(clientSock, buf, strlen(buf), 0);

        endGame(gId, r);
    }
    else if (r == 'D') {
        const char *msg = "GAME OVER. DRAW!\n";
        send(clientSock, msg, strlen(msg), 0);
        endGame(gId, 'D');
    }
    else {
        gm->turn = 1 - gm->turn;
    }
}

static void checkDesk(int clientSock, int myIndex) {
    char bigBuffer[2048];
    int offset = 0;

    int gId = g_data->players[myIndex].gameId;
    if (gId < 0 || !g_data->games[gId].used) {
        offset += snprintf(bigBuffer + offset, sizeof(bigBuffer) - offset,
                           "YOU ARE NOT IN A GAME.\n");
        send(clientSock, bigBuffer, offset, 0);
        return;
    }

    Game *gm = &g_data->games[gId];
    if (gm->lastMoveBy == -1) {
        offset += snprintf(bigBuffer + offset, sizeof(bigBuffer) - offset,
                           "NO MOVES YET.\n");
    } else {
        int lr = gm->lastMoveRow;
        int lc = gm->lastMoveCol;
        int who = gm->lastMoveBy;
        offset += snprintf(bigBuffer + offset, sizeof(bigBuffer) - offset,
                           "LAST MOVE: (row=%d, col=%d) BY %s\n",
                           lr+1, lc+1, g_data->players[who].name);
    }

    offset += snprintf(bigBuffer + offset, sizeof(bigBuffer) - offset,
                       "CURRENT BOARD STATE:\n");

    char (*board)[BOARD_SIZE] = gm->board;
    offset += snprintf(bigBuffer + offset, sizeof(bigBuffer) - offset,
        "    1   2   3\n"
        "1   %c | %c | %c\n"
        "   ---+---+---\n"
        "2   %c | %c | %c\n"
        "   ---+---+---\n"
        "3   %c | %c | %c\n",
        board[0][0], board[0][1], board[0][2],
        board[1][0], board[1][1], board[1][2],
        board[2][0], board[2][1], board[2][2]
    );
    send(clientSock, bigBuffer, offset, 0);
}

static void handleClientCommands(int clientSock, int myIndex) {
    char buffer[256];

    while (true) {
        memset(buffer, 0, sizeof(buffer));
        ssize_t n = recv(clientSock, buffer, sizeof(buffer)-1, 0);
        if (n <= 0) {
            break;
        }
        if (buffer[n-1] == '\n' || buffer[n-1] == '\r') {
            buffer[n-1] = '\0';
        }

        sem_lock();
        int gId = g_data->players[myIndex].gameId;
        sem_unlock();

        if (gId != -1) {
            char reply[2048];
            int offset = 0;

            if (strcmp(buffer, "q") == 0) {
                forfeitGame(myIndex);
                removePlayer(myIndex);

                offset += snprintf(reply + offset, sizeof(reply) - offset,
                                   "YOU FORFEITED AND DISCONNECTED.\n");
                send(clientSock, reply, offset, 0);

                close(clientSock);
                return;
            }
            else if (strncmp(buffer, "move ", 5) == 0) {
                int r, c;
                if (sscanf(buffer+5, "%d %d", &r, &c) == 2) {
                    makeMove(clientSock, myIndex, r, c);
                } else {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "WRONG FORMAT. usage: move r c\n");
                    send(clientSock, reply, offset, 0);
                }
            }
            else if (strcmp(buffer, "checkdesk") == 0) {
                checkDesk(clientSock, myIndex);
            }
            else {
                offset += snprintf(reply + offset, sizeof(reply) - offset,
                                   "YOU ARE IN A GAME. AVAILABLE: [move r c |
checkdesk | q]\n");
                send(clientSock, reply, offset, 0);
            }
            {
                const char *ingameHint = "(INGAME) commands: move r c,
checkdesk, q\n";
                send(clientSock, ingameHint, strlen(ingameHint), 0);
            }
        }
        else {
            char reply[2048];
            int offset = 0;

            if (strcmp(buffer, "q") == 0) {
                removePlayer(myIndex);
                offset += snprintf(reply + offset, sizeof(reply) - offset,
                                   "YOU DISCONNECTED.\n");
                send(clientSock, reply, offset, 0);
                close(clientSock);
                return;
            }
            else if (strcmp(buffer, "1") == 0) {
                showOnlinePlayers(clientSock);
            }
            else if (strncmp(buffer, "invite ", 7) == 0) {
                char *targetName = buffer + 7;
                int other = findPlayerByName(targetName);
                if (other < 0) {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "PLAYER NOT FOUND\n");
                } else if (g_data->players[other].gameId != -1) {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "PLAYER IS IN A GAME!\n");
                } else if (g_data->players[other].hasInvitation) {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "PLAYER ALREADY HAS AN
INVITATION!\n");
                } else {
                    g_data->players[other].hasInvitation = true;
                    g_data->players[other].invitedBy = myIndex;
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "INVITATION SENT.\n");
                }
                send(clientSock, reply, offset, 0);
            }
            else if (strcmp(buffer, "check") == 0) {
                if (g_data->players[myIndex].hasInvitation) {
                    int invBy = g_data->players[myIndex].invitedBy;
```

```
                if (invBy >= 0 && invBy < MAX_PLAYERS && g_data-
>players[invBy].active) {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "YOU HAVE AN INVITATION FROM:
%s\nType 'accept' or 'reject'\n",
                                       g_data->players[invBy].name);
                } else {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "INVITATION INVALID.\n");
                    g_data->players[myIndex].hasInvitation = false;
                    g_data->players[myIndex].invitedBy = -1;
                }
            } else {
                offset += snprintf(reply + offset, sizeof(reply) - offset,
                                   "YOU HAVE NO INVITATIONS.\n");
            }
            send(clientSock, reply, offset, 0);
        }
        else if (strcmp(buffer, "accept") == 0) {
            if (!g_data->players[myIndex].hasInvitation) {
                offset += snprintf(reply + offset, sizeof(reply) - offset,
                                   "NO INVITATION.\n");
            } else {
                int invBy = g_data->players[myIndex].invitedBy;
                if (invBy < 0 || invBy >= MAX_PLAYERS || !g_data-
>players[invBy].active) {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "INVITATION INVALID.\n");
                } else if (g_data->players[invBy].gameId != -1 ||
                           g_data->players[myIndex].gameId != -1) {
                    offset += snprintf(reply + offset, sizeof(reply) - offset,
                                       "ONE OF YOU IS ALREADY IN A
GAME.\n");
```

```
          } else {
              int gameSlot = createGame(invBy, myIndex);
              if (gameSlot < 0) {
                  offset += snprintf(reply + offset, sizeof(reply) - offset,
                                  "NO FREE GAME SLOTS.\n");
              } else {
                  offset += snprintf(reply + offset, sizeof(reply) - offset,
                                  "GAME CREATED. YOU ARE 'O'.\n");
              }
          }
          g_data->players[myIndex].hasInvitation = false;
          g_data->players[myIndex].invitedBy = -1;
      }
      send(clientSock, reply, offset, 0);
  }
  else if (strcmp(buffer, "reject") == 0) {
      if (g_data->players[myIndex].hasInvitation) {
          g_data->players[myIndex].hasInvitation = false;
          g_data->players[myIndex].invitedBy = -1;
          offset += snprintf(reply + offset, sizeof(reply) - offset,
                          "INVITATION REJECTED.\n");
      } else {
          offset += snprintf(reply + offset, sizeof(reply) - offset,
                          "NO INVITATION.\n");
      }
      send(clientSock, reply, offset, 0);
  }
  else {
      offset += snprintf(reply + offset, sizeof(reply) - offset,
                      "UNKNOWN COMMAND.\n");
      send(clientSock, reply, offset, 0);
  }
  {
```

```
            char menuBuf[2048];
            int off = 0;
            off += snprintf(menuBuf + off, sizeof(menuBuf)-off,
                        "%s", MENU);
            send(clientSock, menuBuf, off, 0);
         }
      }
   }
}

static void handleClient(int clientSock) {
   send(clientSock, HELLO_MSG, strlen(HELLO_MSG), 0);

   char buffer[128];
   memset(buffer, 0, sizeof(buffer));
   ssize_t n = recv(clientSock, buffer, sizeof(buffer)-1, 0);
   if (n <= 0) {
      close(clientSock);
      return;
   }
   if (buffer[n-1] == '\n' || buffer[n-1] == '\r') {
      buffer[n-1] = '\0';
   }

   sem_lock();
   bool taken = isNameTaken(buffer);
   if (taken) {
      sem_unlock();
      const char *m = "NICKNAME TAKEN.\n";
      send(clientSock, m, strlen(m), 0);
      close(clientSock);
      return;
   }
```

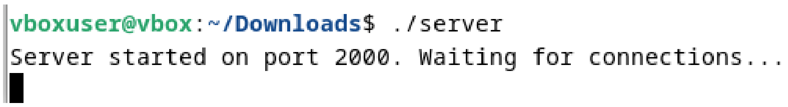
```
    int myIndex = addPlayer(buffer);
    if (myIndex < 0) {
        sem_unlock();
        const char *m = "SERVER IS FULL.\n";
        send(clientSock, m, strlen(m), 0);
        close(clientSock);
        return;
    }
    printf("Player connected: %s\n", buffer);
    sem_unlock();

    {
        char greetBuf[512];
        int offset = 0;
        offset += snprintf(greetBuf + offset, sizeof(greetBuf) - offset,
                           "HELLO, %s\n", buffer);
        offset += snprintf(greetBuf + offset, sizeof(greetBuf) - offset,
                           "%s", MENU);
        send(clientSock, greetBuf, offset, 0);
    }

    handleClientCommands(clientSock, myIndex);

    sem_lock();
    if (g_data->players[myIndex].active) {
        printf("Player %s disconnected.\n", g_data->players[myIndex].name);
        removePlayer(myIndex);
    }
    sem_unlock();

    close(clientSock);
}
```

```
int main() {
    key_t key_shm = ftok("/tmp", 0x66);
    if (key_shm == -1) {
        perror("ftok for shm");
        return 1;
    }
    shm_id = shmget(key_shm, sizeof(SharedData), IPC_CREAT | 0666);
    if (shm_id < 0) {
        perror("shmget");
        return 1;
    }
    g_data = (SharedData*) shmat(shm_id, NULL, 0);
    if (g_data == (void*)-1) {
        perror("shmat");
        return 1;
    }
    memset(g_data, 0, sizeof(*g_data));

    key_t key_sem = ftok("/tmp", 0x77);
    if (key_sem == -1) {
        perror("ftok for sem");
        return 1;
    }
    sem_id = semget(key_sem, 1, IPC_CREAT | 0666);
    if (sem_id < 0) {
        perror("semget");
        return 1;
    }
    if (semctl(sem_id, 0, SETVAL, 1) < 0) {
        perror("semctl SETVAL");
        return 1;
    }
    int listener = socket(AF_INET, SOCK_STREAM, 0);
    if (listener < 0) {
        perror("socket");
        return 1;
    }
    int opt = 1;
    setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, &opt,
sizeof(opt));

    struct sockaddr_in servaddr;
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_port        = htons(SERVER_PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    if (bind(listener, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind");
        close(listener);
        return 1;
    }
    if (listen(listener, 10) < 0) {
        perror("listen");
        close(listener);
        return 1;
    }

    printf("Server started on port %d. Waiting for connections...\n",
SERVER_PORT);

    while (true) {
        struct sockaddr_in cliaddr;
        socklen_t clilen = sizeof(cliaddr);
        int clientSock = accept(listener, (struct sockaddr*)&cliaddr, &clilen);
        if (clientSock < 0) {
            perror("accept");
            continue;
        }

        pid_t pid = fork();
        if (pid < 0) {
            perror("fork");
            close(clientSock);
            continue;
        }
        if (pid == 0) {
            close(listener);
            handleClient(clientSock);
            shmdt(g_data);
            _exit(0);
        } else {
            close(clientSock);
        }
    }

    close(listener);
    shmdt(g_data);
    shmctl(shm_id, IPC_RMID, NULL);
    semctl(sem_id, 0, IPC_RMID, 0);

    return 0;
}
```

Исходный код клиента

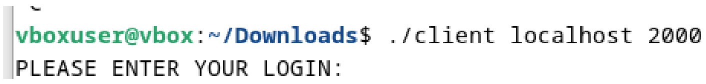
```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

#include <netdb.h>


int main(int argc, char *argv[]) {

    int sock;

    struct sockaddr_in server;

    struct hostent* hp;


    if (argc < 3) {

        printf("Usage: %s hostname port\n", argv[0]);

        exit(1);

    }


    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {

        perror("Opening stream socket");

        exit(1);

    }


    hp = gethostbyname(argv[1]);
    if (hp == NULL) {
        fprintf(stderr, "%s: unknown host\n", argv[1]);
        exit(2);
    }

    server.sin_family = AF_INET;
    memcpy((char*)&server.sin_addr, (char*)hp->h_addr, hp->h_length);
    server.sin_port = htons(atoi(argv[2]));

    if (connect(sock, (struct sockaddr*)&server, sizeof(server)) == -1) {
        perror("Connecting stream socket");
        exit(1);
    } else {
        printf("Connection successful\n\n");
    }
    {
        char recvbuf[2048];
        memset(recvbuf, 0, sizeof(recvbuf));
        ssize_t r = recv(sock, recvbuf, sizeof(recvbuf) - 1, 0);
        if (r > 0) {
            recvbuf[r] = '\0';
            printf("%s", recvbuf);
        } else if (r == 0) {
            printf("Server closed connection immediately.\n");
            close(sock);
            return 0;
        } else {
            perror("recv (initial)");
            close(sock);
            return 1;
        }
    }

    while (1) {
        char line[1024];
        if (!fgets(line, sizeof(line), stdin)) {
            break;
        }

        ssize_t len = strlen(line);
        if (send(sock, line, len, 0) < 0) {
            perror("send");
            break;
        }
        {
            char recvbuf[2048];
            memset(recvbuf, 0, sizeof(recvbuf));
            ssize_t r = recv(sock, recvbuf, sizeof(recvbuf) - 1, 0);
            if (r > 0) {
                recvbuf[r] = '\0';
                printf("%s", recvbuf);
            } else if (r == 0) {
                printf("Server closed connection.\n");
                break;
            } else {
                perror("recv");
                break;
            }
        }

        if (line[0] == 'q' && (line[1] == '\n' || line[1] == '\0')) {
            printf("Exiting client.\n");
            break;
        }
    }

    close(sock);
    return 0;
}
```

Результат тестирования



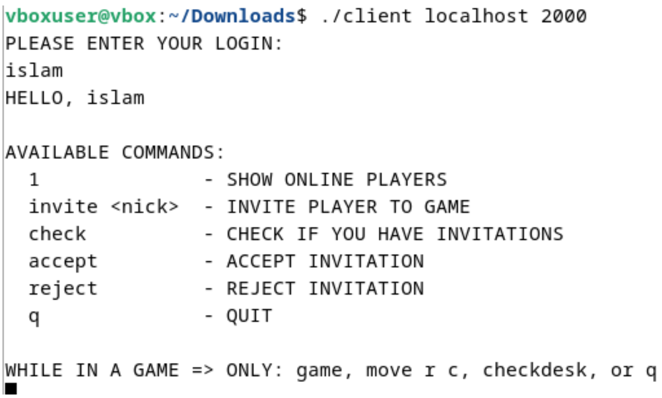
```
vboxuser@vbox:~/Downloads$ ./server
Server started on port 2000. Waiting for connections...
```

Рис. 3 Удачный запуск сервера



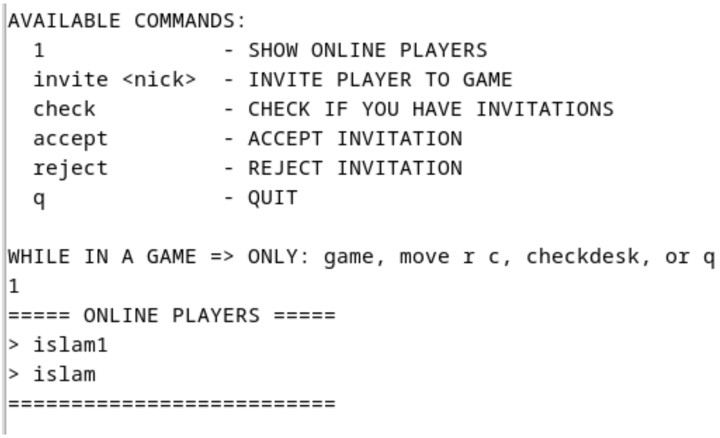
```
vboxuser@vbox:~/Downloads$ ./client localhost 2000
PLEASE ENTER YOUR LOGIN:
```

Рис. 4 Удачное подключение к серверу



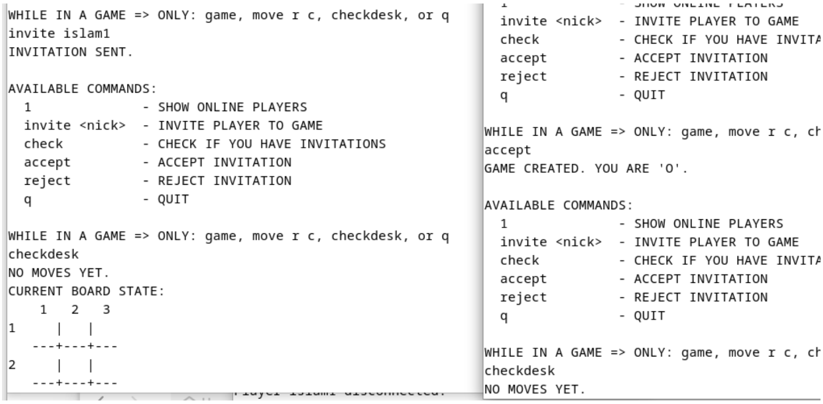
```
vboxuser@vbox:~/Downloads$ ./client localhost 2000
PLEASE ENTER YOUR LOGIN:
islam
HELLO, islam

AVAILABLE COMMANDS:
  1               - SHOW ONLINE PLAYERS
  invite <nick>   - INVITE PLAYER TO GAME
  check           - CHECK IF YOU HAVE INVITATIONS
  accept          - ACCEPT INVITATION
  reject          - REJECT INVITATION
  q               - QUIT

WHILE IN A GAME => ONLY: game, move r c, checkdesk, or q
```

Рис. 5 Интерфейс приложения



```
AVAILABLE COMMANDS:
  1               - SHOW ONLINE PLAYERS
  invite <nick>   - INVITE PLAYER TO GAME
  check           - CHECK IF YOU HAVE INVITATIONS
  accept          - ACCEPT INVITATION
  reject          - REJECT INVITATION
  q               - QUIT

WHILE IN A GAME => ONLY: game, move r c, checkdesk, or q
1
===== ONLINE PLAYERS =====
> islam1
> islam
==========================
```

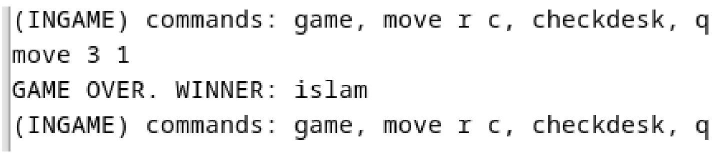
Рис. 6 Вывод списка пользователей онлайн



```
WHILE IN A GAME => ONLY: game, move r c, checkdesk, or q
invite islam1
INVITATION SENT.

AVAILABLE COMMANDS:
  1              - SHOW ONLINE PLAYERS
  invite <nick>  - INVITE PLAYER TO GAME
  check          - CHECK IF YOU HAVE INVITATIONS
  accept         - ACCEPT INVITATION
  reject         - REJECT INVITATION
  q              - QUIT

WHILE IN A GAME => ONLY: game, move r c, checkdesk, or q
checkdesk
NO MOVES YET.
CURRENT BOARD STATE:
    1   2   3
1     |   |
   ---+---+---
2     |   |
   ---+---+---
```

```
  invite <nick>  - INVITE PLAYER TO GAME
  check          - CHECK IF YOU HAVE INVITA
  accept         - ACCEPT INVITATION
  reject         - REJECT INVITATION
  q              - QUIT

WHILE IN A GAME => ONLY: game, move r c, ch
accept
GAME CREATED. YOU ARE 'O'.

AVAILABLE COMMANDS:
  1              - SHOW ONLINE PLAYERS
  invite <nick>  - INVITE PLAYER TO GAME
  check          - CHECK IF YOU HAVE INVITA
  accept         - ACCEPT INVITATION
  reject         - REJECT INVITATION
  q              - QUIT

WHILE IN A GAME => ONLY: game, move r c, ch
checkdesk
NO MOVES YET.
```

Рис. 7 Начало игры



```
(INGAME) commands: game, move r c, checkdesk, q
move 3 1
GAME OVER. WINNER: islam
(INGAME) commands: game, move r c, checkdesk, q
```

Рис. 8 Победный ход