Московский Авиационный Институт (Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования

## Курсовой проект по курсу

**«Операционные системы»**

Группа: М8О-215Б-23

Студент: Лизунов К.Р.

Преподаватель: Миронов Е.С. Оценка:

Дата:

Москва, 2024.

# Постановка задачи

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом.

Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Консоль-серверная игра. Необходимо написать консоль-серверную игру.

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 (возможно больше, если предусмотрено вариантом):

* Создать игру, введя ее имя
* Присоединиться к одной из существующих игр по имени игры

Морской бой. Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. Каждый игрок должен при запуске ввести свой логин. Должна быть предоставлена возможность отправить приглашение на игру другому игроку по логину

# Общие сведения о программе

Программа представляет собой консольную клиент-серверную игру "Морской бой", реализованную с использованием **именованных pipe'ов** (FIFO). Она состоит из двух основных файлов:

## server.c:

* + Это серверная часть программы, которая координирует взаимодействие между клиентами.
  + Сервер отвечает за создание игр, обработку ходов игроков, проверку попаданий и управление очередностью ходов.
  + Сервер также отслеживает состояние игр (например, количество оставшихся кораблей у каждого игрока) и завершает игру, когда один из игроков побеждает.

## client.c:

* + Это клиентская часть программы, которая взаимодействует с сервером.
  + Клиент позволяет игроку создать игру, присоединиться к существующей игре, разместить корабли на поле и делать ходы.
  + Клиент отправляет запросы серверу и получает от него ответы (например, результат хода: попадание или промах).

## Заголовочные файлы

Программа использует следующие стандартные заголовочные файлы:

## <stdio.h>:

* + Используется для ввода и вывода данных (например, printf, scanf).

## <stdlib.h>:

* + Предоставляет функции для работы с памятью (например, malloc, free) и другие утилиты (например, exit).

## <string.h>:

* + Содержит функции для работы со строками (например, strcpy, strcmp, strncmp).

## <unistd.h>:

* + Предоставляет системные вызовы для работы с файлами, процессами и pipe'ами (например, read, write, close, mkfifo).

## <fcntl.h>:

* + Содержит функции для управления файловыми дескрипторами (например, open, O\_RDONLY, O\_WRONLY).

## <sys/stat.h>:

* + Используется для работы с правами доступа к файлам (например, mkfifo).

## <sys/types.h>:

* + Определяет типы данных, используемые в системных вызовах (например, pid\_t).

## <signal.h>:

* + Предоставляет функции для обработки сигналов (например, signal, SIGINT).

## Системные вызовы

Программа активно использует системные вызовы для работы с pipe'ами и файлами:

## mkfifo:

* + Создает именованный pipe (FIFO). Используется для организации обмена данными между сервером и клиентами.

## open:

* + Открывает файл или pipe для чтения или записи. Например, сервер открывает pipe для чтения, а клиент — для записи.

## read:

* + Читает данные из файла или pipe. Сервер и клиенты используют этот вызов для получения сообщений.

## write:

* + Записывает данные в файл или pipe. Сервер и клиенты используют этот вызов для отправки сообщений.

## close:

* + Закрывает файловый дескриптор. Используется для освобождения ресурсов после завершения работы с pipe'ом.

## unlink:

* + Удаляет именованный pipe. Используется для очистки ресурсов после завершения работы программы.

## signal:

* + Устанавливает обработчик сигналов. В программе используется для обработки сигнала SIGINT (Ctrl+C) и корректного завершения работы сервера.

# Общий метод и алгоритм решения

## Сервер:

* + Сервер создает именованный pipe (server\_pipe) и ожидает подключения клиентов.
  + Когда клиент подключается, сервер создает отдельный pipe для этого клиента и обрабатывает его запросы (создание игры, присоединение к игре, ходы).
  + Сервер проверяет попадания и управляет очередностью ходов, отправляя клиентам результаты их действий.

## Клиент:

* + Клиент подключается к серверу через именованный pipe.
  + Игрок может создать игру, присоединиться к существующей игре, разместить корабли на поле и делать ходы.
  + Клиент отправляет запросы серверу и получает от него ответы (например, результат хода: попадание или промах).

# Код программы

Код программы приведён в приложении 1.

# Демонстрация работы программы

Листинг терминала с примерами работы программы приведён на рисунке 1:

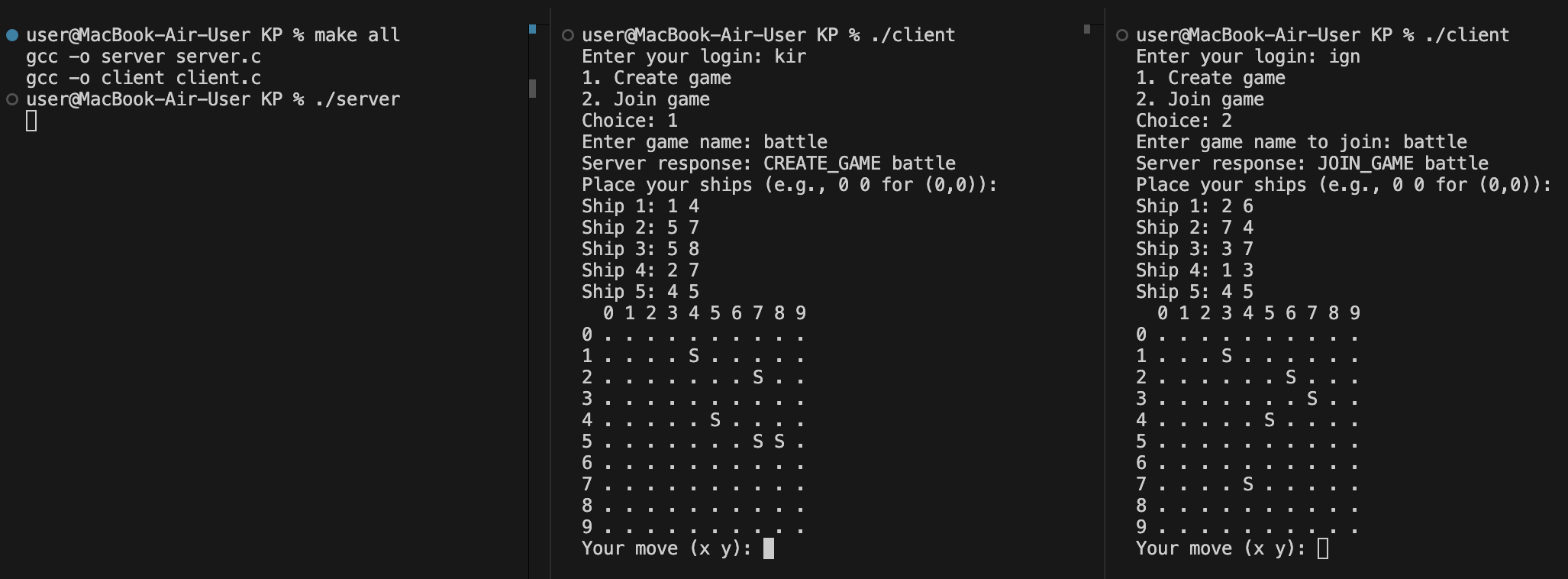


Рисунок 1 — Пример работы программы.

# Вывод

В ходе выполнения курсового проекта был разработан клиент-серверный прототип игры "Морской бой", реализованный с использованием **именованных pipe'ов** (FIFO). Программа успешно реализовала базовый функционал, включая создание игр, подключение к существующим играм, размещение кораблей на

игровом поле и поочередные ходы игроков. Для связи между клиентом и сервером

использовались именованные pipe'ы, что обеспечило простую и надежную коммуникацию.

# Приложение 1

// client.c

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <fcntl.h> #include <sys/stat.h> #include <sys/types.h>

#define BUFFER\_SIZE 100

#define PIPE\_NAME\_SIZE 25

#define BOARD\_SIZE 10

void print\_board(char board[BOARD\_SIZE][BOARD\_SIZE]) { printf(" ");

for (int i = 0; i < BOARD\_SIZE; i++) { printf("%d ", i);

}

printf("\n");

for (int i = 0; i < BOARD\_SIZE; i++) { printf("%d ", i);

for (int j = 0; j < BOARD\_SIZE; j++) { printf("%c ", board[i][j]);

}

printf("\n");

}

}

void place\_ships(char board[BOARD\_SIZE][BOARD\_SIZE]) { printf("Place your ships (e.g., 0 0 for (0,0)): \n");

for (int i = 0; i < 5; i++) { // Размещаем 5 кораблей int x, y;

printf("Ship %d: ", i + 1);

scanf("%d %d", &x, &y);

board[x][y] = 'S';

}

}

void create\_game(int server\_fd, char \*game\_name) { char buffer[BUFFER\_SIZE];

sprintf(buffer, "CREATE\_GAME %s", game\_name); write(server\_fd, buffer, BUFFER\_SIZE); read(server\_fd, buffer, BUFFER\_SIZE); printf("Server response: %s\n", buffer);

}

void join\_game(int server\_fd, char \*game\_name) { char buffer[BUFFER\_SIZE];

sprintf(buffer, "JOIN\_GAME %s", game\_name); write(server\_fd, buffer, BUFFER\_SIZE); read(server\_fd, buffer, BUFFER\_SIZE);

if (strcmp(buffer, "GAME\_NOT\_FOUND") == 0) { printf("Game not found. Please try again.\n"); return;

}

printf("Server response: %s\n", buffer);

}

void make\_move(int server\_fd, char \*game\_name, char board[BOARD\_SIZE][BOARD\_SIZE]) { while (1) {

int x, y;

printf("Your move (x y): ");

scanf("%d %d", &x, &y); char buffer[BUFFER\_SIZE];

sprintf(buffer, "MOVE %s %d %d", game\_name, x, y); write(server\_fd, buffer, BUFFER\_SIZE); read(server\_fd, buffer, BUFFER\_SIZE);

if (strcmp(buffer, "HIT") == 0) { printf("Hit!\n");

board[x][y] = 'X';

} else if (strcmp(buffer, "MISS") == 0) { printf("Miss!\n");

board[x][y] = 'M';

} else if (strcmp(buffer, "WIN") == 0) { printf("You win!\n");

break;

} else if (strcmp(buffer, "LOSE") == 0) { printf("You lose!\n");

break;

} else if (strcmp(buffer, "GAME\_OVER") == 0) { printf("Game over!\n");

break;

}

print\_board(board);

}

}

int main() {

char login[20]; printf("Enter your login: "); scanf("%s", login);

char client\_pipe[PIPE\_NAME\_SIZE];

snprintf(client\_pipe, PIPE\_NAME\_SIZE, "%s\_pipe", login);

mkfifo(client\_pipe, 0666);

int server\_fd = open("server\_pipe", O\_WRONLY); write(server\_fd, client\_pipe, PIPE\_NAME\_SIZE);

int choice;

printf("1. Create game\n2. Join game\nChoice: "); scanf("%d", &choice);

char game\_name[20]; if (choice == 1) {

printf("Enter game name: "); scanf("%s", game\_name); create\_game(server\_fd, game\_name);

} else if (choice == 2) {

printf("Enter game name to join: "); scanf("%s", game\_name); join\_game(server\_fd, game\_name);

}

char board[BOARD\_SIZE][BOARD\_SIZE]; for (int i = 0; i < BOARD\_SIZE; i++) {

for (int j = 0; j < BOARD\_SIZE; j++) { board[i][j] = '.';

}

}

place\_ships(board); print\_board(board);

make\_move(server\_fd, game\_name, board);

close(server\_fd); unlink(client\_pipe); return 0;

}

// server.c

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <fcntl.h> #include <sys/stat.h> #include <sys/types.h> #include <signal.h>

#define MAX\_CLIENTS 2

#define PIPE\_NAME\_SIZE 25

#define BUFFER\_SIZE 100

#define BOARD\_SIZE 10 #define GAME\_FILE "game.txt"

typedef struct { char login[20]; int fd;

char board[BOARD\_SIZE][BOARD\_SIZE]; // Игровое поле int ships\_left; // Количество оставшихся кораблей

} Client;

typedef struct { char name[20]; Client players[2];

int turn; // Индекс игрока, который сейчас ходит int game\_over; // Флаг окончания игры

} Game;

Game games[10]; int game\_count = 0;

void init\_board(char board[BOARD\_SIZE][BOARD\_SIZE]) { for (int i = 0; i < BOARD\_SIZE; i++) {

for (int j = 0; j < BOARD\_SIZE; j++) {

board[i][j] = '.'; // Инициализация пустого поля

}

}

}

void write\_game\_name(const char \*game\_name) { FILE \*file = fopen(GAME\_FILE, "w");

if (file) {

fprintf(file, "%s", game\_name); fclose(file);

printf("Game name '%s' written to file.\n", game\_name);

} else {

perror("Failed to open game file");

}

}

int check\_game\_name(const char \*game\_name) { FILE \*file = fopen(GAME\_FILE, "r");

if (!file) {

perror("Failed to open game file"); return 0; // Файл не существует

}

char existing\_game\_name[20];

if (fscanf(file, "%s", existing\_game\_name) == 1) { fclose(file);

printf("Checking game name '%s' against '%s'.\n", game\_name, existing\_game\_name); return strcmp(existing\_game\_name, game\_name) == 0;

}

fclose(file); return 0;

}

void handle\_client(int client\_fd, char \*client\_pipe) { char buffer[BUFFER\_SIZE];

read(client\_fd, buffer, BUFFER\_SIZE);

if (strncmp(buffer, "CREATE\_GAME", 11) == 0) {

// Создание игры char game\_name[20];

strcpy(game\_name, buffer + 12); write\_game\_name(game\_name); strcpy(games[game\_count].name, game\_name); games[game\_count].players[0].fd = client\_fd; strcpy(games[game\_count].players[0].login, client\_pipe); init\_board(games[game\_count].players[0].board);

games[game\_count].players[0].ships\_left = 5; // Устанавливаем 5 кораблей games[game\_count].turn = 0;

games[game\_count].game\_over = 0;

write(client\_fd, "GAME\_CREATED", 12); game\_count++;

} else if (strncmp(buffer, "JOIN\_GAME", 9) == 0) {

// Присоединение к игре char game\_name[20];

strcpy(game\_name, buffer + 10);

if (check\_game\_name(game\_name)) { for (int i = 0; i < game\_count; i++) {

if (strcmp(games[i].name, game\_name) == 0) { games[i].players[1].fd = client\_fd; strcpy(games[i].players[1].login, client\_pipe); init\_board(games[i].players[1].board); games[i].players[1].ships\_left = 5; // Устанавливаем 5 кораблей write(client\_fd, "GAME\_JOINED", 11);

break;

}

}

} else {

write(client\_fd, "GAME\_NOT\_FOUND", 14);

}

} else if (strncmp(buffer, "MOVE", 4) == 0) {

// Обработка хода char game\_name[20];

int x, y;

sscanf(buffer, "MOVE %s %d %d", game\_name, &x, &y); for (int i = 0; i < game\_count; i++) {

if (strcmp(games[i].name, game\_name) == 0) { if (games[i].game\_over) {

write(games[i].players[games[i].turn].fd, "GAME\_OVER", 9); break;

}

int target\_player = (games[i].turn + 1) % 2;

char result = games[i].players[target\_player].board[x][y]; if (result == 'S') {

games[i].players[target\_player].board[x][y] = 'X'; // Попадание games[i].players[target\_player].ships\_left--;

if (games[i].players[target\_player].ships\_left == 0) { write(games[i].players[games[i].turn].fd, "WIN", 3);

write(games[i].players[target\_player].fd, "LOSE", 4);

games[i].game\_over = 1;

} else {

write(games[i].players[games[i].turn].fd, "HIT", 3);

}

} else {

games[i].players[target\_player].board[x][y] = 'M'; // Промах write(games[i].players[games[i].turn].fd, "MISS", 4);

}

games[i].turn = target\_player; // Передача хода другому игроку break;

}

}

}

}

void cleanup() { unlink("server\_pipe"); unlink(GAME\_FILE);

}

int main() { atexit(cleanup); signal(SIGINT, cleanup);

mkfifo("server\_pipe", 0666);

int server\_fd = open("server\_pipe", O\_RDONLY);

while (1) {

char client\_pipe[PIPE\_NAME\_SIZE]; read(server\_fd, client\_pipe, PIPE\_NAME\_SIZE); int client\_fd = open(client\_pipe, O\_WRONLY);

handle\_client(client\_fd, client\_pipe); close(client\_fd);

}

close(server\_fd); return 0;

}