МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

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»

Студент: Теребаев К. Д.
Группа: М8О-203Б-22
Преподаватель: Миронов Е. С.
Дата: _____
Оценка: _____
Подпись: _____

Москва, 2024

## Цель работы

- Приобретение практических навыков в использовании знаний, полученных в течении курса
- Проведение исследования в выбранной предметной области

## Задание

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

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

1 вариант.

Морской бой. Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. Каждый игрок должен при запуске ввести свой логин. Для каждого игрока должна вестись статистика игр (сколько побед/поражений). Игрок может посмотреть свою статистику

## Общие сведения о программе

1. FIFO pipe - именованный канал. Один из методов межпроцессного взаимодействия. Именованный канал позволяет различным процессам обмениваться данными, даже если программы, выполняющиеся в этих процессах, изначально не были написаны для взаимодействия с другими программами.
2. Inotify – механизм, позволяющий через файловый дескриптор наблюдать за директорией или файлом, отслеживая их события.

## Общий алгоритм решения

Создаем программу для сервера, после запуска которого запускаем созданные программы для клиентов. Через FIFO получаем логины, и далее через FIFO передаем все ходы. По окончании меняем статистику и выводим ее.

## Основные файлы программы

Board.hpp

```
#include <unistd.h>

#include <iostream>
#include <vector>
```

```
const int SIZE = 10;

typedef enum {
    empty = 0,
    ship,
    hit,
    miss,
} cell_t;

class Board {
private:
    std::vector<std::vector<cell_t>> main_board;
    std::vector<std::vector<cell_t>> hit_board;
    int points;

public:
    void throw_if_invalid_cell(int x, int y) const {
        if (x < 0 || x >= SIZE || y < 0 || y >= SIZE) {
            throw std::runtime_error("Cell must be on the board!");
        }
    }

    void throw_if_invalid_ship(int length, int x1, int y1, int x2, int y2) const
{
        if (x1 < 0 || x2 >= SIZE || y1 < 0 || y2 >= SIZE) {
            throw std::runtime_error("Ships must be on the board!");
        }
        if (!((x2 - x1 == length - 1 && y1 == y2) || (y2 - y1 == length - 1 && x1
== x2))) {
            throw std::runtime_error("Wrong length or orientation!");
        }
        int left = x1 > 0 ? x1 - 1 : x1, right = x2 < SIZE - 1 ? x2 + 1 : x2,
bottom = y1 > 0 ? y1 - 1 : y1, top = y2 < SIZE - 1 ? y2 + 1 : y2;
        for (int i{bottom}; i <= top; ++i) {
            for (int j{left}; j <= right; ++j) {
                if (main_board[i][j] != empty) {
                    throw std::runtime_error("The ships are too close to each
other!");
                }
            }
        }
    }

    void add_ship(int length) noexcept {
        int x1, x2, y1, y2;
        std::string pos, pos1, pos2;
        while (true) {
            std::cout << "Enter the coordinates of the " << length << "-decker
ship ";
            if (length == 1) {
```

```
                std::cout << "(format: A0): " << std::endl;
                std::cin >> pos;
                y1 = y2 = static_cast<int>(toupper(pos[0]) - 'A');
                x1 = x2 = static_cast<int>(pos[1] - '0');
            } else {
                std::cout << "(format: A0 A0): " << std::endl;
                std::cin >> pos1 >> pos2;
                y1 = static_cast<int>(toupper(pos1[0]) - 'A');
                x1 = static_cast<int>(pos1[1] - '0');
                y2 = static_cast<int>(toupper(pos2[0]) - 'A');
                x2 = static_cast<int>(pos2[1] - '0');
                if (x1 > x2) std::swap(x1, x2);
                if (y1 > y2) std::swap(y1, y2);
            }

            try {
                if ((length == 1 && pos.size() != 2) || (length != 1 &&
(pos1.size() != 2 || pos2.size() != 2))) {
                    throw std::runtime_error("Wrong answer!");
                }
                throw_if_invalid_ship(length, x1, y1, x2, y2);
            } catch (std::exception& e) {
                std::cout << e.what() << " Try again." << std::endl;
                continue;
            }

            for (int i{y1}; i <= y2; ++i) {
                for (int j{x1}; j <= x2; ++j) {
                    main_board[i][j] = ship;
                }
            }
            break;
        }
    }

    bool success(int x, int y) const noexcept {
        return (main_board[y][x] == ship);
    }

    void set_hit(int x, int y, cell_t cell) noexcept {
        if (hit_board[y][x] != hit && hit_board[y][x] != miss) {
            hit_board[y][x] = cell;
            if (cell == hit) {
                ++points;
            }
        }
    }

    void set_main(int x, int y, cell_t cell) noexcept {
        if (main_board[y][x] != hit && main_board[y][x] != miss) {
            main_board[y][x] = cell;
```

```
        }
    }

    Board()
        : main_board(std::vector<std::vector<cell_t>>(SIZE,
std::vector<cell_t>(SIZE, empty))),
          hit_board(std::vector<std::vector<cell_t>>(SIZE,
std::vector<cell_t>(SIZE, empty))),
          points(0) {}

    virtual ~Board() = default;

    bool check_win() const noexcept {
        return points == 20;
    }

    void set_ships() noexcept {
        print(std::cout);
        for (int i{0}; i < 4; ++i) {
            for (int j{0}; j <= i; ++j) {
                add_ship(4 - i);
                print(std::cout);
            }
        }
    }

    void clear() {
        points = 0;
        for (int i{0}; i < SIZE; ++i) {
            for (int j{0}; j < SIZE; ++j) {
                main_board[i][j] = empty;
                hit_board[i][j] = empty;
            }
        }
    }

    void print(std::ostream& os) const noexcept {
        os << "  ";
        for (int i{0}; i < SIZE; ++i) {
            os << i << " ";
        }

        os << "\t";

        os << "  ";
        for (int i{0}; i < SIZE; ++i) {
            os << i << " ";
        }

        os << std::endl;
        for (int i{0}; i < SIZE; ++i) {
```

```
            os << static_cast<char>('A' + i) << " ";
            for (int j{0}; j < SIZE; ++j) {
                switch (main_board[i][j]) {
                    case ship:
                        os << "# ";
                        break;
                    case hit:
                        os << "x ";
                        break;
                    case miss:
                        os << "* ";
                        break;
                    default:
                        os << ". ";
                        break;
                }
            }

            os << "\t";

            os << static_cast<char>('A' + i) << " ";
            for (int j{0}; j < SIZE; ++j) {
                switch (hit_board[i][j]) {
                    case ship:
                        os << "# ";
                        break;
                    case hit:
                        os << "x ";
                        break;
                    case miss:
                        os << "* ";
                        break;
                    default:
                        os << ". ";
                        break;
                }
            }

            os << std::endl;
        }
    }
};

std::ostream& operator<<(std::ostream& os, Board& board) {
    board.print(os);
    return os;
}
```

ServerSocket.hpp

```
#pragma once
```

```cpp
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>

#include <stdexcept>
#include <string>

class ServerSocket {
private:
    std::string login;
    int fd_req, fd_rep;

public:
    ServerSocket(const std::string& _login) : login(_login) {
        std::string req_path = "./tmp/" + login + "_req", rep_path = "./tmp/" +
login + "_rep";
        if (mkfifo(req_path.c_str(), O_RDWR) == -1) {
            throw std::runtime_error("Can't create FIFO");
        }
        if ((fd_req = open(req_path.c_str(), O_RDWR)) == -1) {
            throw std::runtime_error("Can't open FIFO");
        }
        if (mkfifo(rep_path.c_str(), O_RDWR) == -1) {
            throw std::runtime_error("Can't create FIFO");
        }
        if ((fd_rep = open(rep_path.c_str(), O_RDWR)) == -1) {
            throw std::runtime_error("Can't open FIFO");
        }
    }

    ~ServerSocket() {
        std::string req_path = "./tmp/" + login + "_req", rep_path = "./tmp/" +
login + "_rep";
        close(fd_req);
        close(fd_rep);
        unlink(req_path.c_str());
        unlink(rep_path.c_str());
    }

    std::string receive(size_t size) {
        char tmp[++size];
        if (read(fd_rep, tmp, size) == -1) {
            throw std::runtime_error("Can't read from FIFO");
        }
        return std::string{tmp};
    }

    void send(const std::string& message) {
        if (write(fd_req, message.c_str(), message.size() + 1) == -1) {
            throw std::runtime_error("Can't write to FIFO");
        }
```

```
    }
};
```

ClientSocket.hpp

```
#pragma once
#include <fcntl.h>
#include <unistd.h>

#include <stdexcept>
#include <string>

class ClientSocket {
private:
    std::string login;
    int fd_req, fd_rep;

public:
    ClientSocket(const std::string& _login) : login(_login) {
        std::string req_path = "./tmp/" + login + "_req", rep_path = "./tmp/" +
login + "_rep";
        if ((fd_req = open(req_path.c_str(), O_RDWR)) == -1) {
            throw std::runtime_error("Can't open FIFO");
        }
        if ((fd_rep = open(rep_path.c_str(), O_RDWR)) == -1) {
            throw std::runtime_error("Can't open FIFO");
        }
    }

    ~ClientSocket() {
        close(fd_req);
        close(fd_rep);
    }

    std::string receive(size_t size) const {
        char tmp[++size];
        if (read(fd_req, tmp, size) == -1) {
            throw std::runtime_error("Can't read from FIFO");
        }
        return std::string{tmp};
    }

    void send(const std::string& message) const {
        if (write(fd_rep, message.c_str(), message.size() + 1) == -1) {
            throw std::runtime_error("Can't write to FIFO");
        }
    }

    std::string get_login() const noexcept {
        return login;
    }
```

```
};
```

server.cpp

```cpp
#include <pthread.h>
#include <sys/inotify.h>

#include <chrono>
#include <vector>

#include "ClientSocket.hpp"

const size_t MAX_LEN_LOGIN = 11;
const size_t MAX_LEN_FILE_NAME = MAX_LEN_LOGIN + 5;
const size_t EVENT_SIZE = sizeof(struct inotify_event);
const size_t BUF_LEN = EVENT_SIZE + MAX_LEN_FILE_NAME;

void send_statistics(const ClientSocket& socket) {
    std::string path = "./data/" + socket.get_login();
    if (access(path.c_str(), F_OK) == 0) {
        FILE* file = fopen(path.c_str(), "rb");
        int win, lose;
        fread(&win, sizeof(win), 1, file);
        fread(&lose, sizeof(lose), 1, file);
        socket.send(std::to_string(win) + " " + std::to_string(lose));
        fclose(file);
    } else {
        FILE* file = fopen(path.c_str(), "wb");
        int zero{0};
        fwrite(&zero, sizeof(int), 1, file);
        fwrite(&zero, sizeof(int), 1, file);
        socket.send("0 0");
        fclose(file);
    }
}

void inc_win(const std::string& login) {
    std::string path = "./data/" + login;
    FILE* file = fopen(path.c_str(), "rb+");
    int win, lose;
    fread(&win, sizeof(win), 1, file);
    fread(&lose, sizeof(lose), 1, file);
    fseek(file, 0, 0);
    ++win;
    fwrite(&win, sizeof(int), 1, file);
    fwrite(&lose, sizeof(int), 1, file);
    fclose(file);
}

void inc_lose(const std::string& login) {
    std::string path = "./data/" + login;
```

```
    FILE* file = fopen(path.c_str(), "rb+");
    int win, lose;
    fread(&win, sizeof(win), 1, file);
    fread(&lose, sizeof(lose), 1, file);
    fseek(file, 0, 0);
    ++lose;
    fwrite(&win, sizeof(int), 1, file);
    fwrite(&lose, sizeof(int), 1, file);
    fclose(file);
}

void* game_thread(void* args) {
    auto players{static_cast<std::pair<std::string, std::string>*>(args)};
    ClientSocket first_socket{players->first}, second_socket{players->second};
    first_socket.send(second_socket.get_login());
    second_socket.send(first_socket.get_login());
    send_statistics(first_socket);
    send_statistics(second_socket);

    // game
    first_socket.receive(sizeof(char) * 2);
    second_socket.receive(sizeof(char) * 2);
    first_socket.send("1");
    second_socket.send("2");
    bool end = false;
    while (!end) {
        while (true) {
            std::string cell = first_socket.receive(sizeof(char) * 2);
            second_socket.send(cell);
            std::string res = second_socket.receive(sizeof(char));
            first_socket.send(res);
            if (res == "t") {
                std::string end_msg = first_socket.receive(sizeof(char));
                second_socket.send(end_msg);
                if (end_msg == "t") {
                    second_socket.send(first_socket.get_login());
                    end = true;
                    inc_win(first_socket.get_login());
                    inc_lose(second_socket.get_login());
                    break;
                }
            } else {
                break;
            }
        }
        if (end) break;
        while (true) {
            std::string cell = second_socket.receive(sizeof(char) * 2);
            first_socket.send(cell);
            std::string res = first_socket.receive(sizeof(char));
            second_socket.send(res);
```

```
            if (res == "t") {
                std::string end_msg = second_socket.receive(sizeof(char));
                first_socket.send(end_msg);
                if (end_msg == "t") {
                    first_socket.send(second_socket.get_login());
                    inc_win(second_socket.get_login());
                    inc_lose(first_socket.get_login());
                    end = true;
                    break;
                }
            } else {
                break;
            }
        }
    }

    send_statistics(first_socket);
    send_statistics(second_socket);
    return nullptr;
}

int main() {
    int fd, wd, length;
    char buffer[BUF_LEN];
    const std::string path = "./tmp";
    fd = inotify_init();
    if (fd < 0) {
        throw std::runtime_error("Couldn't initialize inotify");
    }
    wd = inotify_add_watch(fd, path.c_str(), IN_CREATE);
    if (wd == -1) {
        throw std::runtime_error("Couldn't add watch to " + path);
    }
    auto last_update = std::chrono::system_clock::now();
    std::string first_p{}, second_p{};
    while
(std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now(
) - last_update) < std::chrono::hours{1}) {
        length = read(fd, buffer, BUF_LEN);
        if (length < 0) {
            throw std::runtime_error("Read error");
        }
        struct inotify_event* event = (struct inotify_event*)buffer;
        if (event->len && event->mask & IN_CREATE) {
            std::string file_name{event->name};
            if (file_name.ends_with("_rep")) {
                last_update = std::chrono::system_clock::now();
                pthread_t thread;
                std::string login{file_name.substr(0, file_name.size() - 4)};
                if (first_p == "") {
                    first_p = login;
```

```
                } else {
                    second_p = login;
                    std::pair<std::string, std::string> players{first_p,
second_p};
                    pthread_create(&thread, nullptr, game_thread,
(void*)&players);
                    pthread_detach(thread);
                    first_p = second_p = "";
                }
            }
        }
    }

    inotify_rm_watch(fd, wd);
    close(fd);
}
```

client.cpp

```
#include <iostream>
#include <sstream>

#include "Board.hpp"
#include "ServerSocket.hpp"

const size_t MAX_LEN_LOGIN = 11;

std::string set_login() {
    std::string login, req_path, rep_path;
    while (true) {
        std::cout << "Enter your login (max length " << MAX_LEN_LOGIN << "): ";
        std::cin >> login;
        req_path = "./tmp/" + login + "_req";
        rep_path = "./tmp/" + login + "_rep";
        if (login.size() > MAX_LEN_LOGIN || access(req_path.c_str(), F_OK) == 0
|| access(rep_path.c_str(), F_OK) == 0) {
            std::cout << "Login is unavailable! Try again." << std::endl;
        } else {
            break;
        }
    }
    return login;
}

bool attack(Board& board, ServerSocket& socket) {
    while (true) {
        std::cout << "Enter the coordinates of cell to attack (format: A0):" <<
std::endl;
        std::string pos;
        std::cin >> pos;
```

```
        if (pos.size() != 2) {
            std::cout << "Wrong answer! Try again." << std::endl;
            continue;
        }
        int y = static_cast<int>(toupper(pos[0]) - 'A'), x =
static_cast<int>(pos[1] - '0');
        try {
            board.throw_if_invalid_cell(x, y);
            socket.send(pos);

            std::string res = socket.receive(sizeof(char));
            if (res == "t") {
                board.set_hit(x, y, hit);
                std::cout << static_cast<char>(y + 'A') << x << ": hit!" <<
std::endl;
                std::cout << board;
                return true;
            } else {
                board.set_hit(x, y, miss);
                std::cout << static_cast<char>(y + 'A') << x << ": miss" <<
std::endl;
                std::cout << board;
                return false;
            }

        } catch (std::exception& e) {
            std::cout << e.what() << " Try again." << std::endl;
        }
    }
}

bool defense(Board& board, ServerSocket& socket) {
    std::cout << "The enemy is attacking..." << std::endl;
    std::string pos = socket.receive(sizeof(char) * 2);
    int y = static_cast<int>(toupper(pos[0]) - 'A'), x = static_cast<int>(pos[1]
- '0');
    if (board.success(x, y)) {
        board.set_main(x, y, hit);
        std::cout << static_cast<char>(y + 'A') << x << ": hit!" << std::endl;
        socket.send("t");
        std::cout << board;
        return true;
    } else {
        board.set_main(x, y, miss);
        std::cout << static_cast<char>(y + 'A') << x << ": miss" << std::endl;
        socket.send("f");
        std::cout << board;
        return false;
    }
}
```

```
void play(ServerSocket& socket) {
    std::cout << "Start..." << std::endl;
    Board board;
    std::cout << "Set your ships:" << std::endl;
    board.set_ships();
    socket.send("OK");
    std::string turn = socket.receive(sizeof(char));
    bool end = false;
    while (!end) {
        if (turn == "1") {
            while (attack(board, socket)) {
                if (board.check_win()) {
                    socket.send("t");
                    std::cout << "You won!" << std::endl;
                    end = true;
                    break;
                } else {
                    socket.send("f");
                }
            }

            if (end) break;

            while (defense(board, socket)) {
                if (socket.receive(sizeof(char)) == "t") {
                    std::cout << socket.receive(MAX_LEN_LOGIN) << " won" <<
std::endl;
                    end = true;
                    break;
                }
            }
        }

        else {
            while (defense(board, socket)) {
                if (socket.receive(sizeof(char)) == "t") {
                    std::cout << socket.receive(MAX_LEN_LOGIN) << " won" <<
std::endl;
                    end = true;
                    break;
                }
            }

            if (end) break;

            while (attack(board, socket)) {
                if (board.check_win()) {
                    socket.send("t");
                    std::cout << "You won!" << std::endl;
                    end = true;
                    break;
```

```
                } else {
                    socket.send("f");
                }
            }
        }
    }
}

void get_statistics(std::string& login, ServerSocket& socket) {
    std::string stats = socket.receive(21);
    std::istringstream is{stats};
    int win, lose;
    is >> win >> lose;
    std::cout << "Player: " << login << std::endl << "win: " << win << std::endl
<< "lose: " << lose << std::endl;
}

int main() {
    std::string login = set_login();
    ServerSocket socket{login};
    std::string opponent = socket.receive(MAX_LEN_LOGIN * sizeof(char));
    std::cout << "Your game is found. Opponent: " << opponent << std::endl;
    std::cout << "Your statistics" << std::endl;
    get_statistics(login, socket);
    play(socket);
    get_statistics(login, socket);
}
```

**Вывод**

В ходе выполнения курсового проекта я познакомился с именованными pipe’ами FIFO, через которые данные можно передавать между независимыми программами, в отличие от простых pipe’ов, а также с механизмом inotify, позволяющим следить за изменениями в директориях и файлах. Знания, полученные в течение всего курса, определенно помогли мне справиться с этим заданием, так как я имел представление о способах передачи данных.