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

#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

#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.