Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

**“Морской бой на memory-mapped files”**

Студент: Салихов Тимур Русланович

Группа: М8О–212Б–22

Вариант: 5

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## 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.

**Проектирование консольной клиент-серверной игры**

На основе любой из выбранных технологий:

* Pipes
* Sockets
* Сервера очередей
* И другие

Создать собственную игру более, чем для одного пользователя. Игра может быть устроена по принципу: клиент-клиент, сервер-клиент.

Вариант 5: Морской бой. Общение между сервером и клиентом необходимо организовать при помощи memory map. 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**

Проект содержит 2 исполняемых файла: server и client, для запуска сервера и клиента соответственно. Проект состоит из следующих файлов:

1. server.cpp - программа для сервера
2. client.cpp - программа для клиента
3. mq.hpp - заголовочный файл для собственной очереди сообщений
4. mq.cpp - реализация функций для собственной очереди сообщений
5. player.hpp - заголовочный файл для класса игрок
6. player.cpp - реализация функций для класса игрок
7. db.hpp - заголовочный файл с базой данных
8. db.cpp - реализация функций базы данных

**Общий метод и алгоритм решения**.

Для обмена данными между сервером и клиентом был написан свой класс MQ, который реализует очередь сообщений через memory map. Объект класса хранит в себе мапы двух файлов: входного и выходного, а также указатели на первый элемент записи и первый элемент чтения. В конструктор класса передаются имена входного и выходного файла, после чего он открывает их при помощи shm\_open() и создает мапы для каждого из них. Функции receive и send читают и записывают сообщения соответственно, причем делают это закольцованным образом, что означает, что можно передавать большое количество данных, до тех пор, пока длинны сообщений не перейдут за 1024. Каждое сообщение оканчивается специальным символом ‘\0’, который служит маркером того, что сообщение завершилось.

Для хранения статистики игр пользователей был написан класс Db, который реализует простую базу данных. В качестве хранилища используется текстовый файл, в котором на каждой строке указано имя пользователя, количество побед и количество поражений. В конструктор передается имя файла с базой данных, после чего этот файл открывается и из его содержимого формируется структура std::map<std::string, std::pair<int, int>>, которая используется для хранения данных пользователей в памяти программы. Эту структуру используют функции findEntry и updateEntry для получения данных о пользователе и обновления данных о пользователе соответственно. Функция syncData вновь открывает файл и перезаписывает старые данные новыми из структуры map.

Класс player необходим для работы с игроками. В объекте хранится имя игрока, статистика побед и поражений, а также сколько непораженных частей кораблей осталось и непосредственно поле самого игрока. Конструктор принимает на вход имя игрока, победы, поражения и вектор с координатами расположения частей кораблей. Также существует конструктор без параметров, для хранения данных соперника. У класса есть 3 геттера: stats, name и remainingSquares, которые используются для получения соответствующих полей. Метод hit производит выстрел по указанным координатам и возвращает true, если произошло попадание и false, если попадания не было. Оператор вывода перегружен, чтобы выводить поле игрока. Метод anonymousPrint выводит поле, но маскирует положение кораблей. Метод extract\_field выводит поле в виде строки, для того чтобы его можно было передавать по mq. Метод update\_field принимает поле, переданное по mq, в строчном виде и обновляет данные объекта. Также в файле содержится функция coinflip, которая производит подброс монетки.

В начале программы и сервер и клиент запрашивают логин и расположение кораблей. После этого пользователи могут посмотреть свою статистику, проверить правильность заполнения поля и начать игру. После начала игры сервер и клиент начинают взаимодействовать между собой. Обмен данными осуществляется через mq, синхронизация обеспечивается двумя семафорами: для сервера и для клиента. После начала игры происходит обмен данными и подкидывается монетка для определения очередности. После этого игроки ходят по очереди по правилам морского боя до тех пор, пока один из них не выиграет. Сервер произаодит все основные манипуляции и отправляет различные запросы клиенту. Клиент мониторит запросы и отвечает на них.

**Основные файлы программы**

**server.cpp:**

#include <semaphore.h>

#include <iostream>

#include <sstream>

#include "db.hpp"

#include "mq.hpp"

#include "player.hpp"

auto main() -> int {

sem\_t\* client\_sem = sem\_open("client\_sem", O\_CREAT, 0777, 0);

sem\_t\* server\_sem = sem\_open("server\_sem", O\_CREAT, 0777, 0);

MQ mq = MQ("toserver.dat", "fromserver.dat");

Db db = Db("/Users/timursalihov/OperatingSystems/CW/database.dat");

std::string name;

std::cout << "Enter your nickname: ";

std::cin >> name;

std::pair<int, int> stats = db.findEntry(name);

std::vector<std::pair<int, int>> positions(20);

std::pair<int, int> coord;

std::cout << "Enter ship locations:\n";

for (size\_t i = 0; i != 20; ++i) {

std::cin >> positions[i].first >> positions[i].second;

}

Player me(name, stats.first, stats.second, positions);

Player client;

int option = 0;

while (option != 3) {

std::cout << "============================" << std::endl;

std::cout << "Choose option:" << std::endl;

std::cout << "1. Show stats" << std::endl;

std::cout << "2. Show field" << std::endl;

std::cout << "3. Start game" << std::endl;

std::cout << "============================" << std::endl;

std::cout << "Option: ";

std::cin >> option;

std::cout << "============================" << std::endl;

std::pair<int, int> stats = me.stats();

switch (option) {

case 1:

std::cout << "Name: " << me.name() << " Wins: " << stats.first << " Losses: " << stats.second << std::endl;

break;

case 2:

std::cout << me << std::endl;

break;

case 3:

std::cout << "Waiting for client response..." << std::endl;

break;

}

}

sem\_post(client\_sem);

sem\_wait(server\_sem);

std::cout << "Client connected successfully" << std::endl;

std::cout << "Requesting client data...";

std::string clientdata = mq.recieve();

std::stringstream clientdataStream;

clientdataStream << clientdata;

std::string op\_name;

std::pair<int, int> op\_stats;

clientdataStream >> op\_name >> op\_stats.first >> op\_stats.second;

std::string op\_field = mq.recieve();

client.update\_field(op\_field);

std::cout << "Received" << std::endl;

std::cout << "Sending server field to client...";

mq.send(me.extract\_field());

std::cout << "Sent" << std::endl;

sem\_post(client\_sem);

sem\_wait(server\_sem);

std::cout << "Flipping coin...";

bool myTurn = coinflip();

std::cout << "Flipped" << std::endl;

if (myTurn) {

mq.send("No");

std::cout << "You go first" << std::endl;

} else {

mq.send("Yes");

std::cout << "Opponent goes first" << std::endl;

}

sem\_post(client\_sem);

sem\_wait(server\_sem);

while (true) {

if (myTurn) {

std::pair<int, int> coord;

std::cout << "============================" << std::endl;

std::cout << "It's your turn" << std::endl;

std::cout << "Enter hit coordinates: ";

std::cin >> coord.first >> coord.second;

std::cout << "============================" << std::endl;

bool hit = client.hit(coord);

if (hit) {

std::cout << "You hit at " << coord.first << " " << coord.second << std::endl;

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent field:" << std::endl;

client.anonymousPrint();

std::cout << std::endl;

std::stringstream command;

command << coord.first << " " << coord.second;

mq.send("Op\_hit");

mq.send(command.str());

mq.send(client.extract\_field());

sem\_post(client\_sem);

sem\_wait(server\_sem);

if (client.remainingSquares() == 0) {

mq.send("Loss");

db.updateEntry(me.name(), {me.stats().first + 1, me.stats().second});

db.updateEntry(op\_name, {op\_stats.first, op\_stats.second + 1});

db.syncData();

sem\_post(client\_sem);

std::cout << "=================" << std::endl;

std::cout << "You won!" << std::endl;

std::cout << "=================" << std::endl;

break;

}

} else {

std::cout << "You miss at " << coord.first << " " << coord.second << std::endl;

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent field:" << std::endl;

client.anonymousPrint();

std::cout << std::endl;

myTurn = false;

std::stringstream command;

command << coord.first << " " << coord.second;

mq.send("Op\_miss");

mq.send(command.str());

mq.send(client.extract\_field());

sem\_post(client\_sem);

sem\_wait(server\_sem);

}

} else {

std::cout << "============================" << std::endl;

std::cout << "It's opponent's turn" << std::endl;

std::cout << "============================" << std::endl;

mq.send("Try");

sem\_post(client\_sem);

sem\_wait(server\_sem);

std::string response = mq.recieve();

std::stringstream res;

res << response;

std::pair<int, int> coord;

res >> coord.first >> coord.second;

bool hit = me.hit(coord);

if (hit) {

std::cout << "Opponent hit at " << coord.first << " " << coord.second << std::endl;

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent field:" << std::endl;

client.anonymousPrint();

std::cout << std::endl;

std::stringstream command;

command << coord.first << " " << coord.second;

mq.send("You\_hit");

mq.send(command.str());

mq.send(me.extract\_field());

sem\_post(client\_sem);

sem\_wait(server\_sem);

if (me.remainingSquares() == 0) {

mq.send("Win");

db.updateEntry(me.name(), {me.stats().first, me.stats().second + 1});

db.updateEntry(op\_name, {op\_stats.first + 1, op\_stats.second});

db.syncData();

sem\_post(client\_sem);

std::cout << "=================" << std::endl;

std::cout << "You lost!" << std::endl;

std::cout << "=================" << std::endl;

break;

}

} else {

std::cout << "Opponent miss at " << coord.first << " " << coord.second << std::endl;

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent field:" << std::endl;

client.anonymousPrint();

std::cout << std::endl;

myTurn = true;

std::stringstream command;

command << coord.first << " " << coord.second;

mq.send("You\_miss");

mq.send(command.str());

mq.send(me.extract\_field());

sem\_post(client\_sem);

sem\_wait(server\_sem);

}

}

sem\_unlink("server\_sem");

sem\_unlink("client\_sem");

}

}

**client.cpp:**

#include <semaphore.h>

#include <iostream>

#include <sstream>

#include "db.hpp"

#include "mq.hpp"

#include "player.hpp"

auto main() -> int {

sem\_t\* client\_sem = sem\_open("client\_sem", O\_CREAT, 0777, 0);

sem\_t\* server\_sem = sem\_open("server\_sem", O\_CREAT, 0777, 0);

MQ mq = MQ("fromserver.dat", "toserver.dat");

Db db = Db("/Users/timursalihov/OperatingSystems/CW/database.dat");

std::string name;

std::cout << "Enter your nickname: ";

std::cin >> name;

std::pair<int, int> stats = db.findEntry(name);

std::vector<std::pair<int, int>> positions(20);

std::pair<int, int> coord;

std::cout << "Enter ship locations:\n";

for (size\_t i = 0; i != 20; ++i) {

std::cin >> positions[i].first >> positions[i].second;

}

Player me(name, stats.first, stats.second, positions);

Player server;

int option = 0;

while (option != 3) {

std::cout << "============================" << std::endl;

std::cout << "Choose option:" << std::endl;

std::cout << "1. Show stats" << std::endl;

std::cout << "2. Show field" << std::endl;

std::cout << "3. Start game" << std::endl;

std::cout << "============================" << std::endl;

std::cout << "Option: ";

std::cin >> option;

std::cout << "============================" << std::endl;

std::pair<int, int> stats = me.stats();

switch (option) {

case 1:

std::cout << "Name: " << me.name() << " Wins: " << stats.first << " Losses: " << stats.second << std::endl;

break;

case 2:

std::cout << me << std::endl;

break;

case 3:

std::cout << "Waiting for server response..." << std::endl;

break;

}

}

sem\_wait(client\_sem);

std::cout << "Connection to server established" << std::endl;

std::cout << "Sending player field data...";

std::stringstream clientdata;

clientdata << me.name() << " " << me.stats().first << " " << me.stats().second;

mq.send(clientdata.str());

mq.send(me.extract\_field());

std::cout << "Sent" << std::endl;

std::cout << "Waiting for server field...";

sem\_post(server\_sem);

sem\_wait(client\_sem);

std::string op\_field = mq.recieve();

server.update\_field(op\_field);

std::cout << "Received" << std::endl;

sem\_post(server\_sem);

sem\_wait(client\_sem);

std::string turn = mq.recieve();

if (turn == "No") {

std::cout << "Coin flipped, opponnent goes first" << std::endl;

} else if (turn == "Yes") {

std::cout << "Coin flipped, you go first" << std::endl;

}

sem\_post(server\_sem);

sem\_wait(client\_sem);

std::string command = mq.recieve();

while (true) {

if (command == "Op\_hit") {

std::string coord\_str = mq.recieve();

std::stringstream coord\_stream;

coord\_stream << coord\_str;

std::pair<int, int> coord;

coord\_stream >> coord.first >> coord.second;

std::cout << "Opponent hit at " << coord.first << " " << coord.second << std::endl;

std::string upd = mq.recieve();

me.update\_field(upd);

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent's field: " << std::endl;

server.anonymousPrint();

std::cout << std::endl;

std::cout << "============================" << std::endl;

std::cout << "It's opponent's turn" << std::endl;

std::cout << "============================" << std::endl;

sem\_post(server\_sem);

sem\_wait(client\_sem);

command = mq.recieve();

} else if (command == "Op\_miss") {

std::string coord\_str = mq.recieve();

std::stringstream coord\_stream;

coord\_stream << coord\_str;

std::pair<int, int> coord;

coord\_stream >> coord.first >> coord.second;

std::cout << "Opponent missed at " << coord.first << " " << coord.second << std::endl;

std::string upd = mq.recieve();

me.update\_field(upd);

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent's field: " << std::endl;

server.anonymousPrint();

std::cout << std::endl;

sem\_post(server\_sem);

sem\_wait(client\_sem);

command = mq.recieve();

} else if (command == "Try") {

std::pair<int, int> coord;

std::cout << "============================" << std::endl;

std::cout << "It's your turn" << std::endl;

std::cout << "Enter hit coordinates: ";

std::cin >> coord.first >> coord.second;

std::cout << "============================" << std::endl;

std::stringstream coord\_stream;

coord\_stream << coord.first << " " << coord.second;

mq.send(coord\_stream.str());

sem\_post(server\_sem);

sem\_wait(client\_sem);

command = mq.recieve();

} else if (command == "You\_hit") {

std::string coord\_str = mq.recieve();

std::stringstream coord\_stream;

coord\_stream << coord\_str;

std::pair<int, int> coord;

coord\_stream >> coord.first >> coord.second;

std::cout << "You hit at " << coord.first << " " << coord.second << std::endl;

std::string upd = mq.recieve();

server.update\_field(upd);

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent's field: " << std::endl;

server.anonymousPrint();

std::cout << std::endl;

sem\_post(server\_sem);

sem\_wait(client\_sem);

command = mq.recieve();

} else if (command == "You\_miss") {

std::string coord\_str = mq.recieve();

std::stringstream coord\_stream;

coord\_stream << coord\_str;

std::pair<int, int> coord;

coord\_stream >> coord.first >> coord.second;

std::cout << "You miss at " << coord.first << " " << coord.second << std::endl;

std::string upd = mq.recieve();

server.update\_field(upd);

std::cout << "Your field: " << std::endl;

std::cout << me << std::endl;

std::cout << "Opponent's field: " << std::endl;

server.anonymousPrint();

std::cout << std::endl;

std::cout << "============================" << std::endl;

std::cout << "It's opponent's turn" << std::endl;

std::cout << "============================" << std::endl;

sem\_post(server\_sem);

sem\_wait(client\_sem);

command = mq.recieve();

} else if (command == "Win") {

std::cout << "=================" << std::endl;

std::cout << "You won!" << std::endl;

std::cout << "=================" << std::endl;

break;

} else if (command == "Loss") {

std::cout << "=================" << std::endl;

std::cout << "You lost!" << std::endl;

std::cout << "=================" << std::endl;

break;

}

}

sem\_unlink("server\_sem");

sem\_unlink("client\_sem");

}

**mq.hpp:**

#pragma once

#include <string>

class MQ {

private:

char\* \_inputQueue;

char\* \_outputQueue;

const char\* \_inpQueueName;

const char\* \_outpQueueName;

int \_queueSize;

int \_readIndex;

int \_writeIndex;

public:

MQ(std::string inputQueueName, std::string outputQueueName);

~MQ();

auto recieve() -> std::string;

auto send(std::string message) -> void;

};

**mq.cpp:**

#include "mq.hpp"

#include <sys/fcntl.h>

#include <sys/mman.h>

#include <unistd.h>

#include <string>

MQ::MQ(std::string inputQueueName, std::string outputQueueName) {

\_inpQueueName = inputQueueName.c\_str();

\_outpQueueName = outputQueueName.c\_str();

\_readIndex = 0;

\_writeIndex = 0;

int memoryd = shm\_open(inputQueueName.c\_str(), O\_RDWR | O\_CREAT, 0666);

ftruncate(memoryd, 1024);

\_inputQueue = (char\*)mmap(NULL, 1024, PROT\_READ | PROT\_WRITE, MAP\_SHARED, memoryd, 0);

memoryd = shm\_open(outputQueueName.c\_str(), O\_RDWR | O\_CREAT, 0666);

ftruncate(memoryd, 1024);

\_outputQueue = (char\*)mmap(NULL, 1024, PROT\_READ | PROT\_WRITE, MAP\_SHARED, memoryd, 0);

}

MQ::~MQ() {

munmap(\_inputQueue, 1024);

munmap(\_outputQueue, 1024);

shm\_unlink(\_inpQueueName);

shm\_unlink(\_outpQueueName);

}

auto MQ::recieve() -> std::string {

std::string answer = "";

char c = \_inputQueue[\_readIndex];

while (c != '\0') {

answer += c;

\_readIndex = (\_readIndex + 1) % 1024;

c = \_inputQueue[\_readIndex];

}

\_readIndex = (\_readIndex + 1) % 1024;

return answer;

}

auto MQ::send(std::string message) -> void {

for (size\_t i = 0; i != message.length(); ++i) {

char c = message[i];

\_outputQueue[\_writeIndex] = c;

\_writeIndex = (\_writeIndex + 1) % 1024;

}

\_outputQueue[\_writeIndex] = '\0';

\_writeIndex = (\_writeIndex + 1) % 1024;

}

**db.hpp:**

#pragma once

#include <map>

#include <string>

class Db {

private:

std::string \_fileName;

std::map<std::string, std::pair<int, int>> \_data;

public:

Db(std::string fileName);

~Db() = default;

auto findEntry(std::string name) -> std::pair<int, int>;

auto updateEntry(std::string name, std::pair<int, int> stats) -> void;

auto syncData() -> void;

};

**db.cpp:**

#include "db.hpp"

#include <fstream>

#include <map>

Db::Db(std::string fileName) {

\_fileName = fileName;

std::fstream file(\_fileName, std::ios::in);

std::string name;

std::pair<int, int> stats;

while (file.peek() != EOF) {

file >> name >> stats.first >> stats.second;

\_data[name] = stats;

}

file.close();

}

auto Db::findEntry(std::string name) -> std::pair<int, int> {

auto it = \_data.find(name);

std::pair<int, int> res;

if (it != \_data.end()) {

res = it->second;

} else {

res = {0, 0};

\_data[name] = res;

}

return res;

}

auto Db::updateEntry(std::string name, std::pair<int, int> stats) -> void {

\_data[name] = stats;

}

auto Db::syncData() -> void {

std::fstream file(\_fileName, std::ios::out);

for (const auto& [key, value] : \_data) {

file << key << " " << value.first << " " << value.second << "\n";

}

file.close();

}

**player.hpp:**

#pragma once

#include <iostream>

#include <string>

#include <vector>

class Player {

private:

int \_wins, \_losses;

int \_remainingSquares;

std::string \_name;

std::vector<std::vector<char>> \_field;

public:

Player(std::string name, int wins, int losses, std::vector<std::pair<int, int>> positions);

Player();

~Player() = default;

auto stats() -> std::pair<int, int>;

auto hit(std::pair<int, int> position) -> bool;

auto name() -> std::string;

auto remainingSquares() -> int;

friend auto operator<<(std::ostream& os, const Player& p) -> std::ostream&;

auto anonymousPrint() -> void;

auto update\_field(std::string data) -> void;

auto extract\_field() -> std::string;

};

auto coinflip() -> bool;

**player.cpp:**

#include "player.hpp"

#include <iostream>

#include <sstream>

Player::Player(std::string name, int wins, int losses, std::vector<std::pair<int, int>> positions) {

\_name = name;

\_wins = wins;

\_losses = losses;

\_remainingSquares = 20;

\_field.resize(10);

for (size\_t i = 0; i != 10; ++i) {

for (size\_t j = 0; j != 10; ++j) {

\_field[i].push\_back('.');

}

}

for (std::pair<int, int> pos : positions) {

\_field[pos.first][pos.second] = 'O';

}

}

Player::Player() {

\_name = "";

\_wins = 0;

\_losses = 0;

\_remainingSquares = 20;

\_field.resize(10);

for (size\_t i = 0; i != 10; ++i) {

for (size\_t j = 0; j != 10; ++j) {

\_field[i].push\_back('.');

}

}

}

auto Player::stats() -> std::pair<int, int> {

std::pair<int, int> p{\_wins, \_losses};

return p;

}

auto Player::hit(std::pair<int, int> pos) -> bool {

if (\_field[pos.first][pos.second] == '.') {

\_field[pos.first][pos.second] = '#';

return false;

} else if (\_field[pos.first][pos.second] == 'O') {

\_remainingSquares--;

\_field[pos.first][pos.second] = 'X';

int left\_top\_row = pos.first - 1;

int left\_top\_column = pos.second - 1;

if (left\_top\_row >= 0 && left\_top\_column >= 0) {

\_field[left\_top\_row][left\_top\_column] = '#';

}

int right\_top\_row = pos.first - 1;

int right\_top\_column = pos.second + 1;

if (right\_top\_row >= 0 && right\_top\_column <= 9) {

\_field[right\_top\_row][right\_top\_column] = '#';

}

int right\_bottom\_row = pos.first + 1;

int right\_bottom\_column = pos.second + 1;

if (right\_bottom\_row <= 9 && right\_bottom\_column <= 9) {

\_field[right\_bottom\_row][right\_bottom\_column] = '#';

}

int left\_bottom\_row = pos.first + 1;

int left\_bottom\_column = pos.second - 1;

if (left\_bottom\_row <= 9 && right\_bottom\_column >= 0) {

\_field[left\_bottom\_row][left\_bottom\_column] = '#';

}

int left\_row = pos.first;

int left\_column = pos.second - 1;

bool left = false;

if (left\_column >= 0) {

left = true;

}

int up\_row = pos.first - 1;

int up\_column = pos.second;

bool up = false;

if (up\_row >= 0) {

up = true;

}

int right\_row = pos.first;

int right\_column = pos.second + 1;

bool right = false;

if (right\_column <= 9) {

right = true;

}

int bottom\_row = pos.first + 1;

int bottom\_column = pos.second;

bool bottom = false;

if (bottom\_row <= 9) {

bottom = true;

}

std::vector<std::pair<int, int>> toFill;

while (left\_column >= 0 && \_field[left\_row][left\_column] == 'X') {

left\_column -= 1;

}

if (left\_column >= 0 && \_field[left\_row][left\_column] == 'O') {

return true;

} else if (left\_column >= 0 && \_field[left\_row][left\_column] == '.') {

toFill.push\_back({left\_row, left\_column});

}

while (right\_column <= 9 && \_field[right\_row][right\_column] == 'X') {

right\_column += 1;

}

if (right\_column <= 9 && \_field[right\_row][right\_column] == 'O') {

return true;

} else if (right\_column <= 9 && \_field[right\_row][right\_column] == '.') {

toFill.push\_back({right\_row, right\_column});

}

while (up\_row >= 0 && \_field[up\_row][up\_column] == 'X') {

up\_row -= 1;

}

if (up\_row >= 0 && \_field[up\_row][up\_column] == 'O') {

return true;

} else if (up\_row >= 0 && \_field[up\_row][up\_column] == '.') {

toFill.push\_back({up\_row, up\_column});

}

while (bottom\_row <= 9 && \_field[bottom\_row][bottom\_column] == 'X') {

bottom\_row += 1;

}

if (bottom\_row <= 9 && \_field[bottom\_row][bottom\_column] == 'O') {

return true;

} else if (bottom\_row <= 9 && \_field[bottom\_row][bottom\_column] == '.') {

toFill.push\_back({bottom\_row, bottom\_column});

}

for (std::pair<int, int> point : toFill) {

\_field[point.first][point.second] = '#';

}

return true;

}

return false;

}

auto Player::name() -> std::string {

return \_name;

}

auto Player::remainingSquares() -> int {

return \_remainingSquares;

}

auto operator<<(std::ostream& os, const Player& p) -> std::ostream& {

for (size\_t i = 0; i != 10; ++i) {

for (size\_t j = 0; j != 10; ++j) {

os << p.\_field[i][j] << " ";

}

os << "\n";

}

return os;

}

auto Player::anonymousPrint() -> void {

for (size\_t i = 0; i != 10; ++i) {

for (size\_t j = 0; j != 10; ++j) {

if (\_field[i][j] == 'O') {

std::cout << "."

<< " ";

} else {

std::cout << \_field[i][j] << " ";

}

}

std::cout << "\n";

}

}

auto Player::update\_field(std::string data) -> void {

std::stringstream ss;

ss << data;

for (size\_t i = 0; i != 100; ++i) {

int row, column;

char symbol;

ss >> row >> column >> symbol;

\_field[row][column] = symbol;

}

}

auto Player::extract\_field() -> std::string {

std::stringstream ss;

for (size\_t i = 0; i != 10; ++i) {

for (size\_t j = 0; j != 10; ++j) {

ss << i << " " << j << " " << \_field[i][j] << " ";

}

}

return ss.str();

}

auto coinflip() -> bool {

srand(time(0));

return rand() % 2;

}

**Пример работы**

****

****

****

**Вывод**

В процессе выполнения данной работы мне пришлось применить многие знания из курса “Операционные системы”, а также знания из курса “Объектно-ориентированное программирование”, поэтому данный проект помог мне в закреплении пройденного материала. Для осуществления передачи данных между неродственными процессами мне пришлось изучить системный вызов shm\_open(), который позволяет открыть shared memory map не только между родственными процессами, но и между неродственными. Также предстояло продумать логику взаимодействие сервера и клиента, чтобы правильно расставить семафоры для синхронной работы. Помимо этого пришлось продумать логику, связанную с попаданиями и правильным изменением полей в морском бое. Ну и наконец необходимо было придумать, как хранить, использовать и обновлять статистику игроков. Курсовой проект считаю отличным завершением курса “Операционные системы”.