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»

Группа: М80-206Б-22

Студент: Волков А.Д.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 11.01.2024

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.

Вариант

Вариант 12.
«Быки и коровы» (угадывать необходимо числа). Общение между сервером и клиентом необходимо организовать при помощи memory map. При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Должна быть реализована функция поиска игры, то есть игрок пытается войти в игру не по имени, а просто просит сервер найти ему игру.

Общий алгоритм

Игра включает в себя 3 программы: код клиента, код сервера и код менеджера сессий. Клиент, сервер и менеджер взаимодействуют с помощью memory mapping, а синхронизируются сигналами. Практически все сообщения клиента, сервера и менеджера имеют json структуру. Клиент работает на трех потоках: поток работы с сервером, поток для поимки сигнала о победителе в сессии и поток для поимки сигнала о том, что сервер и менеджер сессий завершили работу. Для начала клиенту предлагается ввести имя, потом команды для подключения к сессии или создания сессии. Потом начнется игра. В течении игры к сессии, могут подключаться новые игроки. Если какой-то игрок в сессии выиграл, то сессия закрывается, а игроки удаляются. Сервер отвечает на игровые сообщения клиентов и делает логи в формате json. Менеджер в свою очередь управляет сессиями, а именно создает, подключает к ним игроков и удаляет их. Также сервер контролирует количество игроков в сессии, чтобы оно не превышало заданное количество при ее создании.

Метод решения

Была использована сторонняя библиотека nlohmann/json (https://github.com/nlohmann/json) для удобства распределения информации в отображаемой памяти. Для начала запускается менеджер, который сразу отображает все необходимую для общения с клиентом информацию в память. Потом менеджер ждет команд от клиентов. Клиенты для начала должны ввести имя, потом ввести команду для создания или подключения к сессии. Эта команда определенным образом парсится и отображается в память. Менеджер сессий получает сигнал от клиента что данные о команде записаны в память и начинает их обрабатывать. Если не сработало никакое исключение, то менеджер дает разрешение на игру, если же срабатывает какое то исключение, то менеджер говорит об этом клиенту, и клиент должен ввести команду заново. Сервер же включается только при создании сессии, и для каждой сессии включается свой сервер. Когда создана сессия, то клиенты могут завершить свою работу, если завершит работу менеджер сессий. Далее сервера коммуницируют с клиентами, и когда в сессии кто-то победил, то сервер отправляет всем клиентам в сессии сигнал о том что кто-то победил, и клиенты завершают свою работу. После этого менеджер удаляет сессию из словаря сессий и дает возможность переиспользовать сессию с таким же названием.

Листинг программы

game_configuration.cpp

```
#pragma once

#include <bits/stdc++.h>
#include <nlohmann/json.hpp>
#include <iostream>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <semaphore.h>
#include <cstring>

#define GAME_FILENAME "game"
#define SERVER_PID_IN_MEMORY "server_pid"
#define MANAGER_PID_IN_MEMORY "manger_pid"
#define MANAGER_CLIENT_COMMUNICATION "manager"
#define MANAGER_WINNER_THREAD_PID_IN_MEMORY "manager_winner_pid"
#define MANAGER_WINNER_THREAD_CLIENT_COMMUNICATION "manager_winner"
#define SIZE 4096

struct Player {
    std::string name;
    int bulls = 0;
    int cows = 0;
    int supposition = 0;
    bool win = false;
    pid_t pid;
    std::string sess_name;
    Player(std::string _name) {
        name = _name;
        pid = getpid();
    }
};

std::ostream& operator<<(std::ostream& os, const Player& player) {
    os << "bulls: " << player.bulls << "\n";
    os << "cows: " << player.cows << "\n";
    return os;
}

struct Session {
    std::string session_name;
    std::vector<std::string> players_list;
    int max_players_quantity;
};

bool operator==(const Session& session1, const Session& session2) {
    return session1.session_name == session2.session_name;
}

int random_number() {
    srand(time(NULL));
    return rand() % 900 + 100;
}
```

```
std::string toup(std::string str) {
    std::transform(str.begin(), str.end(), str.begin(), toupper);
    return str;
}

void game(nlohmann::json& player_stats, std::string answer) {
    std::string supposition = std::to_string(player_stats["supposition"].get<int>());
    int cows = 0; int bulls = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (answer[i] == supposition[j]) {
                if (i == j) {
                    bulls++;
                } else {
                    cows++;
                }
            }
        }
    }
    if (bulls == 3) {
        player_stats["win"] = true;
    }
    player_stats["bulls"] = bulls;
    player_stats["cows"] = cows;
}
```

client.cpp

```
#include "game_configuration.hpp"

static int flag = 1;
static int winner_flag = 1;
static int exit_flag = 1;

void signal_handler(int sugnum) {
    flag = 0;
}

void winner_signal_handler(int signum) {
    winner_flag = 0;
}

void exit_signal_handler(int signum) {
    exit_flag = 0;
}

void *exit_handler(void *arg) {
    signal(SIGINT, exit_signal_handler);
    while (exit_flag) {sleep(1);}
    std::cout << "\n";
    exit(0);
}

void *winner_handler(void *arg) {
    std::string *session = (std::string *) arg;
    signal(SIGUSR2, winner_signal_handler);
    while (winner_flag) {sleep(1);}
    int game_fd = shm_open((*session).c_str(), O_CREAT | O_RDWR, 0666);
```

```
    ftruncate(game_fd, SIZE);
    char *game_mmap = static_cast<char *>(mmap(NULL, SIZE, PROT_WRITE | PROT_READ,
MAP_SHARED, game_fd, 0));
    std::cout << "\n" << "[INFO] " << std::string(game_mmap) << "\n";
    sleep(1);
    munmap(game_mmap, SIZE);
    close(game_fd);
    shm_unlink((*session).c_str());
    int manth_fd = shm_open(MANAGER_WINNER_THREAD_PID_IN_MEMORY, O_CREAT | O_RDWR,
0666);
    ftruncate(manth_fd, SIZE);
    void *manth_mmap = mmap(NULL, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, manth_fd,
0);
    pid_t manth_pid = std::atoi((char *) manth_mmap);
    int man_fd = shm_open(MANAGER_WINNER_THREAD_CLIENT_COMMUNICATION, O_CREAT | O_RDWR,
0666);
    ftruncate(man_fd, SIZE);
    void *man_mmap = mmap(NULL, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, man_fd, 0);
    strcpy((char *) man_mmap, (const_cast<char *>((*session).c_str())));
    kill(manth_pid, SIGUSR2);
    exit(0);
}

void *client(void *arg) {
    sleep(1);
    Player player = *((Player *) arg);
    std::string pid_sess = std::string(SERVER_PID_IN_MEMORY) + player.sess_name;
    int pid_fd = shm_open(pid_sess.c_str(), O_CREAT | O_RDWR, 0666);
    ftruncate(pid_fd, SIZE);
    void *pid_mmap = mmap(NULL, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, pid_fd, 0);
    pid_t server_pid = std::atoi((char *) pid_mmap);
    int game_fd = shm_open(player.sess_name.c_str(), O_CREAT | O_RDWR, 0666);
    ftruncate(game_fd, SIZE);
    char *game_mmap = static_cast<char *>(mmap(NULL, SIZE, PROT_WRITE | PROT_READ,
MAP_SHARED, game_fd, 0));
    nlohmann::json stats {};
    stats["name"] = player.name;
    stats["session"] = player.sess_name;
    while (1) {
        int supposition;
        std::cout << "[CLIENT] Input your supposition: ";
        std::cin >> supposition;
        player.supposition = supposition;
        stats["bulls"] = player.bulls;
        stats["cows"] = player.cows;
        stats["supposition"] = player.supposition;
        stats["win"] = player.win;
        stats["pid"] = player.pid;
        std::cout << stats.dump() << "\n";
        strcpy(game_mmap, stats.dump().c_str());
        kill(server_pid, SIGUSR1);
        signal(SIGUSR1, signal_handler);
        while (flag) {sleep(1);}
        flag = 1;
        std::string message = std::string(game_mmap);
        stats = nlohmann::json::parse(message);
        player.bulls = stats["bulls"];
```

```
        player.cows = stats["cows"];
        player.win = stats["win"];
        std::cout << "\n[INFO] bulls: " << player.bulls << "\n";
        std::cout << "[INFO] cows: " << player.cows << "\n\n";
    }
    munmap(game_mmap, SIZE);
    close(game_fd);
    shm_unlink(player.sess_name.c_str());
}

int main() {
    int man_pid_fd = shm_open(MANAGER_PID_IN_MEMORY, O_CREAT | O_RDWR, 0666);
    ftruncate(man_pid_fd, SIZE);
    void *man_pid_mmap = mmap(NULL, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED,
man_pid_fd, 0);
    pid_t man_pid = std::atoi((char *) man_pid_mmap);

    int man_fd = shm_open(MANAGER_CLIENT_COMMUNICATION, O_CREAT | O_RDWR, 0666);
    ftruncate(man_fd, SIZE);
    void *man_mmap = mmap(NULL, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, man_fd, 0);

    std::string name;
    std::cout << "[CLIENT] Input your name: ";
    std::cin >> name;
    Player player(name);
    std::cout << "\n[INFO] create [session name] [max players quantity]\n";
    std::cout << "[INFO] join [session name]\n";
    std::cout << "[INFO] find\n\n";
    std::string sess_name;
    while (true) {
        std::cout << "[COMMAND] ";
        std::string command;
        std::cin >> command;
        if (command == "create") {
            nlohmann::json request {};
            request["action"] = "create";
            request["player"] = name;
            request["pid"] = player.pid;
            std::string session_name;
            std::cin >> session_name;
            sess_name = session_name;
            request["name"] = session_name;
            int maxc; std::cin >> maxc;
            request["max"] = maxc;
            strcpy((char *) man_mmap, request.dump().c_str());
            kill(man_pid, SIGUSR1);
            signal(SIGUSR1, signal_handler);
            while (flag) {sleep(1);}
            flag = 1;
            nlohmann::json reply = nlohmann::json::parse(std::string((char
*)man_mmap));
            std::cout << "[INFO] " << reply["desc"].get<std::string>() << "\n\n";
            if (!reply["ok"]) {
                continue;
            } else {
                break;
            }
```

```
        } else if (command == "join") {
            nlohmann::json request {};
            request["action"] = "join";
            request["player"] = name;
            request["pid"] = player.pid;
            std::string session_name;
            std::cin >> session_name;
            sess_name = session_name;
            request["name"] = session_name;
            strcpy((char *) man_mmap, request.dump().c_str());
            kill(man_pid, SIGUSR1);
            signal(SIGUSR1, signal_handler);
            while (flag) {sleep(1);}
            flag = 1;
            nlohmann::json reply = nlohmann::json::parse(std::string((char
*)man_mmap));
            std::cout << "[INFO] " << reply["desc"].get<std::string>() << "\n\n";
            if (!reply["ok"]) {
                continue;
            } else {
                break;
            }
        } else if (command == "find") {
            nlohmann::json request {};
            request["action"] = "find";
            request["player"] = name;
            request["pid"] = player.pid;
            strcpy((char *) man_mmap, request.dump().c_str());
            kill(man_pid, SIGUSR1);
            signal(SIGUSR1, signal_handler);
            while (flag) {sleep(1);}
            flag = 1;
            nlohmann::json reply = nlohmann::json::parse(std::string((char
*)man_mmap));
            std::cout << "[INFO] " << reply["desc"].get<std::string>() << "\n\n";
            if (!reply["ok"]) {
                continue;
            } else {
                sess_name = reply["session"];
                break;
            }
        } else {
            std::cout << "[INFO] Incorrect command!\n";
        }
    }
    player.sess_name = sess_name;

    pthread_t client_thread, signal_thread, exit_thread;
    pthread_create(&client_thread, NULL, client, &player);
    pthread_create(&signal_thread, NULL, winner_handler, &sess_name);
    pthread_create(&exit_thread, NULL, exit_handler, NULL);
    pthread_join(client_thread, NULL);
    pthread_join(signal_thread, NULL);
    pthread_join(exit_thread, NULL);
}
```

server.cpp

```
#include "game_configuration.hpp"

static int flag = 1;

void signal_handler(int signum) {
    flag = 0;
}

bool member(std::vector<pid_t> arr, pid_t pid) {
    for (int i = 0; i < arr.size(); i++) {
        if (pid == arr[i]) {
            return true;
        }
    }
    return false;
}

int main(int argc, char *argv[]) {
    int answer = random_number();
    std::vector<pid_t> clients_pid;
    std::cout << "[" << toup(std::string(argv[0])) << "] Answer in session " <<
std::string(argv[0]) << " is " << answer << "\n\n";
    std::string pid_sess = std::string(SERVER_PID_IN_MEMORY) + std::string(argv[0]);
    int pid_fd = shm_open(pid_sess.c_str(), O_CREAT | O_RDWR, 0666);
    ftruncate(pid_fd, SIZE);
    void *pid_mmap = mmap(NULL, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, pid_fd, 0);
    strcpy((char *)pid_mmap, std::to_string(getpid()).c_str());
    munmap(pid_mmap, SIZE);
    close(pid_fd);
    int game_fd = shm_open(argv[0], O_CREAT | O_RDWR, 0666);
    ftruncate(game_fd, SIZE);
    char *game_mmap = static_cast<char *>(mmap(NULL, SIZE, PROT_WRITE | PROT_READ,
MAP_SHARED, game_fd, 0));
    while (1) {
        signal(SIGUSR1, signal_handler);
        while (flag) {sleep(1);}
        std::string message = std::string(game_mmap);
        std::cout << "[RECEIVED] " << message << "\n";
        nlohmann::json stats = nlohmann::json::parse(message);
        if (!member(clients_pid, stats["pid"])) {
            clients_pid.push_back(stats["pid"]);
        }
        strcpy(game_mmap, "");
        game(stats, std::to_string(answer));
        std::cout << "[SENT] " << stats.dump() << "\n\n";
        if (stats["win"] == true) {
            std::string reply = "Game over! Player " + stats["name"].get<std::string>()
+ " won!";
            strcpy(game_mmap, reply.c_str());
            for (auto pid : clients_pid) {
                kill(pid, SIGUSR2);
                flag = 1;
            }
        } else {
            strcpy(game_mmap, stats.dump().c_str());
            kill(stats["pid"], SIGUSR1);
```

```
            flag = 1;
        }
    }
    munmap(game_mmap, SIZE);
    close(game_fd);
    shm_unlink(argv[0]);
}
```

session_manager.cpp

```
#include "game_configuration.hpp"

static int flag = 1;
static int exit_flag = 1;
static int winner_flag = 1;

std::map<std::string, pid_t> players_map;
std::map<std::string, Session> sessions;

void signal_handler(int signum) {
    flag = 0;
}

void exit_handler(int signum) {
    exit_flag = 0;
    for (auto i : players_map) {
        kill(i.second, SIGINT);
    }
    std::cout << "\n";
    exit(0);
}

void winner_handler(int signum) {
    winner_flag = 0;
}

pid_t create_process() {
    pid_t pid = fork();
    if (-1 == pid) {
        perror("fork");
        exit(-1);
    }
    return pid;
}

bool member(const Session& session, std::vector<Session> arr) {
    for (int i = 0; i < arr.size(); i++) {
        if (session == arr[i]) {
            return true;
        }
    }
    return false;
}

bool not_full_sess(std::map<std::string, Session> sessions) {

    for (auto i : sessions) {
        if (i.second.players_list.size() < i.second.max_players_quantity) {
            return false;
```

```
        }
    }
    return true;
}

void *winner(void *arg) {
    int manth_fd = shm_open(MANAGER_WINNER_THREAD_PID_IN_MEMORY, O_CREAT | O_RDWR,
0666);
    ftruncate(manth_fd, SIZE);
    void *manth_pid_mmap = mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
manth_fd, 0);
    strcpy((char *)manth_pid_mmap, std::to_string(getpid()).c_str());
    munmap(manth_pid_mmap, SIZE);
    close(manth_fd);
    int man_fd = shm_open(MANAGER_WINNER_THREAD_CLIENT_COMMUNICATION, O_CREAT | O_RDWR,
0666);
    ftruncate(man_fd, SIZE);
    char *session_mmap = static_cast<char *>(mmap(NULL, SIZE, PROT_WRITE | PROT_READ,
MAP_SHARED, man_fd, 0));
    while (1) {
        signal(SIGUSR2, winner_handler);
        while (winner_flag) {sleep(1);}
        winner_flag = 1;
        auto sess = sessions.find(std::string(session_mmap));
        for (int i = 0; i < sess->second.players_list.size(); i++) {
            players_map.erase(sess->second.players_list[i]);
        }
        sessions.erase(std::string(session_mmap));
        std::cout << "[" << toup(std::string(session_mmap)) << "] Session " <<
std::string(session_mmap) << " was closed!\n\n";
    }
}

void *manager(void *arg) {
    int pid_fd = shm_open(MANAGER_PID_IN_MEMORY, O_CREAT | O_RDWR, 0666);
    ftruncate(pid_fd, SIZE);
    void *pid_mmap = mmap(NULL, SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, pid_fd, 0);
    strcpy((char *)pid_mmap, std::to_string(getpid()).c_str());
    munmap(pid_mmap, SIZE);
    close(pid_fd);
    int fd = shm_open(MANAGER_CLIENT_COMMUNICATION, O_CREAT | O_RDWR, 0666);
    ftruncate(fd, SIZE);
    char *session_mmap = static_cast<char *>(mmap(NULL, SIZE, PROT_WRITE | PROT_READ,
MAP_SHARED, fd, 0));
    signal(SIGINT, exit_handler);
    while (exit_flag) {
        signal(SIGUSR1, signal_handler);
        while (flag) {sleep(1);}
        nlohmann::json session_action =
nlohmann::json::parse(std::string(session_mmap));
        if (players_map.find(session_action["player"]) == players_map.end()) {
            players_map[session_action["player"]] = session_action["pid"].get<pid_t>();
        }
        if (session_action["action"] == "create") {
            nlohmann::json reply {};
            if (sessions.find(session_action["name"].get<std::string>()) !=
sessions.end()) {
```

```
                reply["ok"] = false;
                reply["desc"] = "Session with the same name already exists!";
            } else {
                Session sess;
                sess.session_name = session_action["name"];
                sess.max_players_quantity = session_action["max"];

sess.players_list.push_back(session_action["player"].get<std::string>());
                sessions[sess.session_name] = sess;
                pid_t pid = create_process();
                reply["ok"] = true;
                reply["desc"] = "Session " + session_action["name"].get<std::string>()
+ " was created successfully!";
                if (pid == 0) {
                    char *argv[] = {const_cast<char*>(sess.session_name.c_str()), (char
*) NULL};
                    execv("server", argv);
                }
            }
            strcpy(session_mmap, reply.dump().c_str());
            kill(session_action["pid"], SIGUSR1);
            flag = 1;
        } else if (session_action["action"] == "join") {
            std::string session_name = session_action["name"];
            nlohmann::json reply {};
            if (sessions.find(session_name) == sessions.end()) {
                reply["ok"] = false;
                reply["desc"] = "Session not found!";
            } else {
                Session dest = sessions.find(session_name)->second;
                if (dest.players_list.size() == dest.max_players_quantity) {
                    reply["ok"] = false;
                    reply["desc"] = "Not more free places in session!";
                } else {
                    dest.players_list.push_back(session_action["player"]);
                    sessions.find(session_name)->second.players_list =
dest.players_list;
                    reply["ok"] = true;
                    reply["desc"] = "You successfully joined to session " +
session_action["name"].get<std::string>() + "!";
                }
            }
            strcpy(session_mmap, reply.dump().c_str());
            kill(session_action["pid"], SIGUSR1);
            flag = 1;
        } else if (session_action["action"] == "find") {
            nlohmann::json reply {};
            if (sessions.size() == 0 || not_full_sess(sessions)) {
                reply["ok"] = false;
                reply["desc"] = "Free sessions not found!";
            } else {
                for (auto& i : sessions) {
                    if (i.second.players_list.size() != i.second.max_players_quantity)
{
                        i.second.players_list.push_back(session_action["player"]);
                        reply["ok"] = true;
```

```
                        reply["desc"] = "You successfully joined to session " + i.first
+ "!";
                        reply["session"] = i.first;
                        break;
                    }
                }
            }
            strcpy(session_mmap, reply.dump().c_str());
            kill(session_action["pid"], SIGUSR1);
            flag = 1;
        }
    }
    munmap(session_mmap, SIZE);
    close(fd);
}

int main() {
    pthread_t manager_thread, win_thread;
    pthread_create(&manager_thread, NULL, manager, NULL);
    pthread_create(&win_thread, NULL, winner, NULL);
    pthread_join(manager_thread, NULL);
    pthread_join(win_thread, NULL);
}
```

Тесты

Менеджер сессий и сервер

```
$ ./manager
[SESSION1] Answer in session session1 is 261

[RECEIVED]
{"bulls":0,"cows":0,"name":"player1","pid":185125,"session":"session1","supposition":12
3,"win":false}
[SENT]
{"bulls":0,"cows":2,"name":"player1","pid":185125,"session":"session1","supposition":12
3,"win":false}

[RECEIVED]
{"bulls":0,"cows":0,"name":"player2","pid":185157,"session":"session1","supposition":12
3,"win":false}
[SENT]
{"bulls":0,"cows":2,"name":"player2","pid":185157,"session":"session1","supposition":12
3,"win":false}

[RECEIVED]
{"bulls":0,"cows":0,"name":"player3","pid":185174,"session":"session1","supposition":32
1,"win":false}
[SENT]
{"bulls":1,"cows":1,"name":"player3","pid":185174,"session":"session1","supposition":32
1,"win":false}

[RECEIVED]
{"bulls":0,"cows":2,"name":"player1","pid":185125,"session":"session1","supposition":23
7,"win":false}
[SENT]
{"bulls":1,"cows":0,"name":"player1","pid":185125,"session":"session1","supposition":23
7,"win":false}
```

[RECEIVED]
{"bulls":0,"cows":2,"name":"player2","pid":185157,"session":"session1","supposition":98
9,"win":false}
[SENT]
{"bulls":0,"cows":0,"name":"player2","pid":185157,"session":"session1","supposition":98
9,"win":false}

[RECEIVED]
{"bulls":1,"cows":1,"name":"player3","pid":185174,"session":"session1","supposition":26
1,"win":false}
[SENT]
{"bulls":3,"cows":0,"name":"player3","pid":185174,"session":"session1","supposition":26
1,"win":true}

[SESSION1] Session session1 was closed!

[SESSION2] Answer in session session2 is 324

[RECEIVED]
{"bulls":0,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":12
3,"win":false}
[SENT]
{"bulls":1,"cows":1,"name":"player4","pid":185460,"session":"session2","supposition":12
3,"win":false}

[RECEIVED]
{"bulls":1,"cows":1,"name":"player4","pid":185460,"session":"session2","supposition":32
1,"win":false}
[SENT]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
1,"win":false}

[RECEIVED]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":34
1,"win":false}
[SENT]
{"bulls":1,"cows":1,"name":"player4","pid":185460,"session":"session2","supposition":34
1,"win":false}

[RECEIVED]
{"bulls":1,"cows":1,"name":"player4","pid":185460,"session":"session2","supposition":32
5,"win":false}
[SENT]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
5,"win":false}

[RECEIVED]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
6,"win":false}
[SENT]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
6,"win":false}

[RECEIVED]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
7,"win":false}

```
[SENT]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
7,"win":false}

[RECEIVED]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
8,"win":false}
[SENT]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
8,"win":false}

[RECEIVED]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
9,"win":false}
[SENT]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
9,"win":false}

[RECEIVED]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
0,"win":false}
[SENT]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
0,"win":false}

[RECEIVED]
{"bulls":2,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
4,"win":false}
[SENT]
{"bulls":3,"cows":0,"name":"player4","pid":185460,"session":"session2","supposition":32
4,"win":true}

[SESSION2] Session session2 was closed!
```

Игрок 1

```
$ ./client
[CLIENT] Input your name: player1

[INFO] create [session name] [max players quantity]
[INFO] join [session name]
[INFO] find

[COMMAND] create session1 3
[INFO] Session session1 was created successfully!

[CLIENT] Input your supposition: 123

[INFO] bulls: 0
[INFO] cows: 2

[CLIENT] Input your supposition: 237

[INFO] bulls: 1
[INFO] cows: 0

[CLIENT] Input your supposition:
[INFO] Game over! Player player3 won!
```

Игрок 2

```
$ ./client
[CLIENT] Input your name: player2

[INFO] create [session name] [max players quantity]
[INFO] join [session name]
[INFO] find

[COMMAND] join session1
[INFO] You successfully joined to session session1!

[CLIENT] Input your supposition: 123

[INFO] bulls: 0
[INFO] cows: 2

[CLIENT] Input your supposition: 989

[INFO] bulls: 0
[INFO] cows: 0

[CLIENT] Input your supposition:
[INFO] Game over! Player player3 won!
```

Игрок 3

```
$ ./client
[CLIENT] Input your name: player3

[INFO] create [session name] [max players quantity]
[INFO] join [session name]
[INFO] find

[COMMAND] find
[INFO] You successfully joined to session session1!

[CLIENT] Input your supposition: 321

[INFO] bulls: 1
[INFO] cows: 1

[CLIENT] Input your supposition: 261

[INFO] Game over! Player player3 won!
```

Игрок 4

```
$ ./client
[CLIENT] Input your name: player4

[INFO] create [session name] [max players quantity]
[INFO] join [session name]
```

```
[INFO] find

[COMMAND] find
[INFO] Free sessions not found!

[COMMAND] join session1
[INFO] Not more free places in session!

[COMMAND] create session2 1
[INFO] Session session2 was created successfully!

[CLIENT] Input your supposition: 123

[INFO] bulls: 1
[INFO] cows: 1

[CLIENT] Input your supposition: 321

[INFO] bulls: 2
[INFO] cows: 0

[CLIENT] Input your supposition: 341

[INFO] bulls: 1
[INFO] cows: 1

[CLIENT] Input your supposition: 325

[INFO] bulls: 2
[INFO] cows: 0

[CLIENT] Input your supposition: 326

[INFO] bulls: 2
[INFO] cows: 0

[CLIENT] Input your supposition: 327

[INFO] bulls: 2
[INFO] cows: 0

[CLIENT] Input your supposition: 328

[INFO] bulls: 2
[INFO] cows: 0

[CLIENT] Input your supposition: 329

[INFO] bulls: 2
```

```
[INFO] cows: 0

[CLIENT] Input your supposition: 320

[INFO] bulls: 2
[INFO] cows: 0

[CLIENT] Input your supposition: 324

[INFO] Game over! Player player4 won!
```

Вывод

В ходе выполнения курсового проекта я научился работать с сигналами, а также вспомнил основы работы с memory mapping. Также освежил знания по потокам и изучил клиент-серверную архитектуру. Главной проблемой во время выполнения данного курсового проекта было проектирование программы. На начальных этапах иногда приходилось полностью писать заново всю программу и менять полностью схему ее работы. Но это дало мне полное понимание того как работает моя программа и по итогу я смог добавить несколько фишек, которые делают использование этой программы удобнее.