Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Павлов Иван Дмитриевич
Группа: М8О-207Б-21
Вариант: 12
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Pavloffff/MAI_OS/tree/main/KP

Постановка задачи

Цель работы

Целью работы является:

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

Задание

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

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.

Общие сведения о программе

Программа состоит из 4 файлов: client.cpp — общается с пользователем, apidemon.cpp — управляет игровыми сессиями, server.cpp — управляет выбранной сессией, взаимодействует с client.cpp, kptools.h — класс игрока, сессии и различные константы.

Общий метод и алгоритм решения

Связь между процессами реализована с помощью разделяемой памяти и семафоров. В разделяемой памяти хранятся json-структуры. Демон создает новую сессию вызовом fork(), или подключается к уже существующей посредством главного семафора. Далее Сервер общается с игроками сессии уже с помощью отдельных семафоров. На демоне лежит std::map с сессиями для поиска командой find. Больше класс почти не используется, вся связь

происходит путем изменения json-структур внутри разделяемой памяти. Игровой процесс реализован максимально в соответствии с ТЗ.

Исходный код

kptools.h

```
#ifndef __KPTOOLS_H__
#define __KPTOOLS_H__

#include <string>
#include <iostream>
#include <vector>
#include <map>
#include <sys/stat.h>
#include <mutex>

const std::string mainFileName = "main.back";
const std::string mainSemName = "main.semaphore";
int accessPerm = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;

int semSetvalue(sem_t *semaphore, int state)
{
    std::mutex mx;
    int s = 0;
    sem_getvalue(semaphore, &s);
    mx.lock();
    while (s++ < state) {
        sem_post(semaphore);
    }
    while (s-- > state + 1) {
```

```
        sem_wait(semaphore);
    }
    mx.unlock();
    return s;
}

namespace gametools
{
    class Player
    {
    public:
        std::string name;
        int bulls;
        int cows;
        std::string ans;
        bool operator<(const Player& x);
        friend std::ostream& operator<<(std::ostream& cout, const Player &p) {
            cout << "name: " << p.name << "\n";
            cout << "bulls: " << p.bulls << "\n";
            cout << "cows: " << p.cows << "\n";
            cout << "ans: " << p.ans << "\n";
            return cout;
        }
        Player();
        ~Player();
    };

    class Session
    {
```

```
public:
    std::string sessionName;
    int _sz;
    unsigned int cntOfPlayers;
    int curPlayerIndex = 0;
    std::vector<Player> playerList;
    std::string hiddenNum;
    friend std::ostream& operator<<(std::ostream& cout, const Session &s) {
        cout << "Name of session: " << s.sessionName << "\n";
        cout << "Count of players: " << s.cntOfPlayers << "\n";
        cout << "Turn of player: " << s.curPlayerIndex << "\n";
        cout << "Players:\n";
        for (auto i : s.playerList) {
            cout << i << "\n";
        }
        cout << "hidden Number: " << s.hiddenNum << "\n";
        return cout;
    }
    Session();
    ~Session();
};

Session::Session()
{
}

Session::~Session()
{
}
```

```
bool Player::operator<(const Player& x)
{
    if (this->bulls > x.bulls) {
        return true;
    }
    return this->cows > x.cows;
}

Player::Player()
{
}

Player::~Player()
{
}

void pvPrint(std::vector<Player> &v)
{
    for (int i = 0; i < v.size(); i++) {
        std::cout << v[i];
    }
}

void smPrint(std::map<std::string, gametools::Session> &s)
{
    for (auto i: s) {
        std::cout << i.second << "\n";
    }
```

```
    }
}

#endif
```

apidemon.cpp

```
#include <iostream>
#include <unistd.h>
#include <thread>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include "../include/kptools.h"
#include <vector>
#include <algorithm>
#include <map>
#include <nlohmann/json.hpp>

using namespace gametools;

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

int main(int argc, char const *argv[])
{
    sem_unlink(mainSemName.c_str());
    sem_t *mainSem = sem_open(mainSemName.c_str(), O_CREAT, accessPerm,
0);
```

```
    int state = 0;
    semSetvalue(mainSem, 1);
    sem_getvalue(mainSem, &state);
    while (1) {
        sem_getvalue(mainSem, &state);
        if (state == 0) {
            int mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
            struct stat statBuf;
            fstat(mainFd, &statBuf);
            int sz = statBuf.st_size;
            ftruncate(mainFd, sz);
            char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
            nlohmann::json createReply;
            std::string strToJson = mapped;
            munmap(mapped, sz);
            close(mainFd);
            createReply = nlohmann::json::parse(strToJson);
            nlohmann::json request;
            if (createReply.contains("type")) {
                std::string joinSemName;
                sem_t *joinSem;
                Player player;
                player.ans = createReply["ans"];
                player.bulls = createReply["bulls"];
                player.cows = createReply["cows"];
                player.name = createReply["name"];
                if (createReply["type"] == "create") {
                    joinSemName = createReply["sessionName"];
```

```
            joinSemName += ".semaphore";
            sem_unlink(joinSemName.c_str());
            joinSem = sem_open(joinSemName.c_str(), O_CREAT, accessPerm,
0);
            semSetvalue(joinSem, 0);
            if (sessions.find(createReply["sessionName"]) == sessions.cend()) {
                Session session;
                session.sessionName = createReply["sessionName"];
                session.cntOfPlayers = createReply["cntOfPlayers"];
                session.hiddenNum = createReply["hiddenNum"];
                session.playerList.push_back(player);
                sessions.insert({session.sessionName, session});
                for (auto i: sessions) {
                    std::cout << i.second << std::endl;
                }
                request["check"] = "ok";
                request["state"] = 0;
                pid_t serverPid = fork();
                if (serverPid == 0) {
                    sem_close(mainSem);
                    execl("./server", "./server", strToJson.c_str(), NULL);
                    return 0;
                }
            } else {
                request["check"] = "error";
            }
        } else if (createReply["type"] == "join") {
            joinSemName = createReply["sessionName"];
            joinSemName += ".semaphore";
```

```
            if (sessions.find(createReply["sessionName"]) != sessions.cend()) {
                if (sessions[createReply["sessionName"]].cntOfPlayers <=
sessions[createReply["sessionName"]].playerList.size()) {
                    request["check"] = "error";
                }
                player.ans = createReply["ans"];
                player.bulls = createReply["bulls"];
                player.cows = createReply["cows"];
                player.name = createReply["name"];

sessions[createReply["sessionName"]].playerList.push_back(player);


                std::string joinFdName = createReply["sessionName"];
                int joinFd = shm_open(joinFdName.c_str(), O_RDWR |
O_CREAT, accessPerm);
                std::string strFromJson = createReply.dump();
                char *buffer = (char *) strFromJson.c_str();
                int sz = strlen(buffer) + 1;
                ftruncate(joinFd, sz);
                char *mapped = (char *) mmap(NULL, sz, PROT_READ |
PROT_WRITE, MAP_SHARED, joinFd, 0);
                memset(mapped, '\0', sz);
                sprintf(mapped, "%s", buffer);
                munmap(mapped, sz);
                close(joinFd);
                sem_post(joinSem);
                request["check"] = "ok";
                request["state"] =
sessions[createReply["sessionName"]].playerList.size() - 1;
                request["cnt"] =
sessions[createReply["sessionName"]].cntOfPlayers;
```

```
            } else {
                request["check"] = "error";
            }
        } else if (createReply["type"] == "find") {
            for (auto i: sessions) {
                if (i.second.playerList.size() <= i.second.cntOfPlayers) {
                    request["sessionName"] = i.second.sessionName;
                    createReply["sessionName"] = i.second.sessionName;
                }
            }
            if (!(request.contains("sessionName"))) {
                request["check"] = "error";
            }
            if (!request.contains("check") || request["check"] != "error") {
                player.ans = createReply["ans"];
                player.bulls = createReply["bulls"];
                player.cows = createReply["cows"];
                player.name = createReply["name"];
                request["check"] = "ok";
                request["state"] =
sessions[createReply["sessionName"]].playerList.size() - 1;
                request["cnt"] =
sessions[createReply["sessionName"]].cntOfPlayers;
            }
        }
    } else {
        sem_post(mainSem);
        continue;
    }
    std::string strFromJson = request.dump();
```

```
            char *buffer = (char *) strFromJson.c_str();
            sz = strlen(buffer) + 1;
            mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
            ftruncate(mainFd, sz);
            mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
            memset(mapped, '\0', sz);
            sprintf(mapped, "%s", buffer);
            munmap(mapped, sz);
            close(mainFd);
            sem_post(mainSem);
        }
    }
    return 0;
}
```

client.cpp

```
#include <iostream>
#include <unistd.h>
#include <thread>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <semaphore.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include "../include/kptools.h"
```

```
#include <vector>
#include <algorithm>
#include <map>
#include <nlohmann/json.hpp>
#include <set>
#include <iterator>
#include <random>
#include <sstream>

using namespace gametools;

std::string randomNumber()
{
    static std::vector<int> v = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    static std::random_device rd;
    static std::mt19937 generator(rd());
    std::shuffle(v.begin(), v.end(), generator);
    std::ostringstream oss;
    std::copy(v.begin(), v.begin() + 4, std::ostream_iterator<int>(oss, ""));
    return oss.str();
}

void client(std::string &playerName, std::string &sessionName, int state, int cnt)
{
    std::string apiSemName = sessionName + "api.semaphore";
    sem_t *apiSem = sem_open(apiSemName.c_str(), O_CREAT, accessPerm, 0);
    semSetvalue(apiSem, cnt);
    int apiState = 0;
    sem_getvalue(apiSem, &apiState);
```

```
    if (state > 1) {
        while (apiState != state) {
            sem_getvalue(apiSem, &apiState);
        }
    }
    sem_getvalue(apiSem, &apiState);
    sem_close(apiSem);
    std::string gameSemName = sessionName + "game.semaphore";
    sem_t *gameSem;
    if (state == 1) {
        sem_unlink(gameSemName.c_str());
        gameSem = sem_open(gameSemName.c_str(), O_CREAT, accessPerm, 0);
    } else {
        gameSem = sem_open(gameSemName.c_str(), O_CREAT, accessPerm, 0);
    }
    sem_getvalue(gameSem, &apiState);
    int firstIt = 1, cntOfBulls = 0, cntOfCows = 0, flag = 1;
    while (flag) {
        sem_getvalue(gameSem, &apiState);
        if (apiState % (cnt + 1) == state) {
            int gameFd = shm_open((sessionName + "game.back").c_str(), O_RDWR |
O_CREAT, accessPerm);
            struct stat statBuf;
            fstat(gameFd, &statBuf);
            int sz = statBuf.st_size;
            ftruncate(gameFd, sz);
            char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, gameFd, 0);
            std::string strToJson = mapped;
            nlohmann::json request = nlohmann::json::parse(strToJson);
```

```
if (request.contains("winner")) {
    std::cout << "Game over. Winner is " << request["winner"] << std::endl;
    sem_post(gameSem);
    flag = 0;
    break;
}
std::string ansField = playerName + "ans";
std::string bullsField = playerName + "bulls";
std::string cowsField = playerName + "cows";
std::cout << "statistic of player " << playerName << ":" << std::endl;
std::cout << "for answer: " << request[ansField] << std::endl;
std::cout << "count of bulls: " << request[bullsField] << std::endl;
std::cout << "count of cows: " << request[cowsField] << std::endl;
std::string answer;
std::cout << "Input number length of 4 with different digits: ";
std::cin >> answer;
if (answer.length() == 4) {
    std::set<char> s;
    for (int i = 0; i < 4; i++) {
        s.insert(answer[i]);
    }
    if (s.size() != 4) {
        std::cout << "\nWrong number. Try again" << std::endl;
        continue;
    }
} else {
    std::cout << "\nWrong number. Try again" << std::endl;
    continue;
}
```

```
            request[ansField] = answer;
            std::string strFromJson = request.dump();
            char *buffer = (char *) strFromJson.c_str();
            sz = strlen(buffer) + 1;
            ftruncate(gameFd, sz);
            mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, gameFd, 0);
            memset(mapped, '\0', sz);
            sprintf(mapped, "%s", buffer);
            munmap(mapped, sz);
            close(gameFd);
            sem_post(gameSem);
        }
    }
}

int createSession(std::string &playerName, std::string &sessionName, int
cntOfPlayers)
{
    int state2 = 0;
    nlohmann::json createRequest;
    createRequest["type"] = "create";
    createRequest["name"] = playerName;
    createRequest["bulls"] = 0;
    createRequest["cows"] = 0;
    createRequest["ans"] = "0000";
    createRequest["sessionName"] = sessionName;
    createRequest["cntOfPlayers"] = cntOfPlayers;
    createRequest["hiddenNum"] = randomNumber();
```

```
    sem_t *mainSem = sem_open(mainSemName.c_str(), O_CREAT, accessPerm,
0);
    int state = 0;
    while (state != 1) {
        sem_getvalue(mainSem, &state);
    }
    int mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
    std::string strFromJson = createRequest.dump();
    char *buffer = (char *) strFromJson.c_str();
    int sz = strlen(buffer) + 1;
    ftruncate(mainFd, sz);
    char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
    memset(mapped, '\0', sz);
    sprintf(mapped, "%s", buffer);
    munmap(mapped, sz);
    close(mainFd);
    sem_wait(mainSem);
    sem_getvalue(mainSem, &state);
    while (state != 1) {
        sem_getvalue(mainSem, &state);
    }
    mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
    struct stat statBuf;
    fstat(mainFd, &statBuf);
    sz = statBuf.st_size;
    ftruncate(mainFd, sz);
    mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
```

```
    nlohmann::json reply;
    std::string strToJson = mapped;
    reply = nlohmann::json::parse(strToJson);
    if (reply["check"] == "ok") {
        std::cout << "Session " << sessionName << " created" << std::endl;
        state2 = reply["state"];
        sem_wait(mainSem);
        return 0;
    } else {
        std::cout << "Fail: name " << sessionName << " is already exists" <<
std::endl;
        sem_wait(mainSem);
        return 1;
    }
}

void joinSession(std::string &playerName, std::string &sessionName)
{
    nlohmann::json joinRequest;
    joinRequest["type"] = "join";
    joinRequest["name"] = playerName;
    joinRequest["bulls"] = 0;
    joinRequest["cows"] = 0;
    joinRequest["ans"] = "0000";
    joinRequest["sessionName"] = sessionName;
    sem_t *mainSem = sem_open(mainSemName.c_str(), O_CREAT, accessPerm,
0);
    int state = 0;
    while (state != 1) {
        sem_getvalue(mainSem, &state);
```

```
    }
    int mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
    std::string strFromJson = joinRequest.dump();
    char *buffer = (char *) strFromJson.c_str();
    int sz = strlen(buffer) + 1;
    ftruncate(mainFd, sz);
    char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
    memset(mapped, '\0', sz);
    sprintf(mapped, "%s", buffer);
    munmap(mapped, sz);
    close(mainFd);
    sem_wait(mainSem);
    sem_getvalue(mainSem, &state);
    while (state != 1) {
        sem_getvalue(mainSem, &state);
    }
    mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
    struct stat statBuf;
    fstat(mainFd, &statBuf);
    sz = statBuf.st_size;
    ftruncate(mainFd, sz);
    mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
    nlohmann::json reply;
    std::string strToJson = mapped;
    reply = nlohmann::json::parse(strToJson);
    int state2 = 0, cnt = 0;
```

```
    if (reply["check"] == "ok") {
        std::cout << "Session " << sessionName << " joined" << std::endl;
        state2 = reply["state"];
        cnt = reply["cnt"];
    } else {
        std::cout << "Fail: name " << sessionName << " is not exists" << std::endl;
    }
    sem_wait(mainSem);
    if (reply["check"] == "ok") {
        client(playerName, sessionName, state2, cnt);
    }
}

std::string findSession(std::string &playerName)
{
    nlohmann::json findRequest;
    findRequest["type"] = "find";
    findRequest["name"] = playerName;
    findRequest["bulls"] = 0;
    findRequest["cows"] = 0;
    findRequest["ans"] = "0000";
    sem_t *mainSem = sem_open(mainSemName.c_str(), O_CREAT, accessPerm,
0);
    int state = 0;
    while (state != 1) {
        sem_getvalue(mainSem, &state);
    }
    int mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
    std::string strFromJson = findRequest.dump();
```

```
    char *buffer = (char *) strFromJson.c_str();
    int sz = strlen(buffer) + 1;
    ftruncate(mainFd, sz);
    char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
    memset(mapped, '\0', sz);
    sprintf(mapped, "%s", buffer);
    munmap(mapped, sz);
    close(mainFd);
    sem_wait(mainSem);
    sem_getvalue(mainSem, &state);
    while (state != 1) {
        sem_getvalue(mainSem, &state);
    }
    mainFd = shm_open(mainFileName.c_str(), O_RDWR | O_CREAT,
accessPerm);
    struct stat statBuf;
    fstat(mainFd, &statBuf);
    sz = statBuf.st_size;
    ftruncate(mainFd, sz);
    mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, mainFd, 0);
    nlohmann::json reply;
    std::string strToJson = mapped;
    reply = nlohmann::json::parse(strToJson);
    int state2 = 0, cnt = 0;
    std::string sessionName;
    if (reply["check"] == "ok") {
        std::string sessionName = reply["sessionName"];
        std::cout << "Session " << sessionName << " joined" << std::endl;
```

```
        state2 = reply["state"];
        cnt = reply["cnt"];
    } else {
        std::cout << "Fail: session not found" << std::endl;
    }
    sem_wait(mainSem);
    if (reply["check"] == "ok") {
        return reply["sessionName"];
    } else {
        return "";
    }
}

int main(int argc, char const *argv[])
{
    std::cout << "Input your name: ";
    std::string playerName;
    std::cin >> playerName;
    std::cout << std::endl;
    std::vector<int> v;
    Player player;
    std::string command;
    std::cout << "Write:\n command [arg1] ... [argn]\n";
    std::cout << "\ncreate [name] [cntOfPlayers] to create new game session by name and max count of players\n";
    std::cout << "\njoin [name] to join exists game session by name\n";
    std::cout << "\nfind to find game session\n\n";
    int flag = 1;
    while (flag) {
```

```
        std::cin >> command;
        if (command == "create") {
            std::string name;
            int cntOfPlayers;
            std::cin >> name >> cntOfPlayers;
            if (cntOfPlayers < 2) {
                std::cout << "Error: count of players must be greater then 1\n";
            }
            int c = createSession(playerName, name, cntOfPlayers);
            if (c == 0) {
                joinSession(playerName, name);
            }
            flag = 0;
        } else if (command == "join") {
            std::string name;
            std::cin >> name;
            joinSession(playerName, name);
            flag = 0;
        } else if (command == "find") {
            std::string sessionName = findSession(playerName);
            if (sessionName != "") {
                joinSession(playerName, sessionName);
            }
            flag = 0;
        } else {
            std::cout << "Wrong command!\n";
            continue;
        }
    }
```

```
    return 0;
}
```

server.cpp

```
#include <iostream>
#include <unistd.h>
#include <thread>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <semaphore.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include "../include/kptools.h"
#include <vector>
#include <algorithm>
#include <map>
#include <nlohmann/json.hpp>

using namespace gametools;

Session session;
std::map<char, int> hidV;

std::pair<int, int> ggame(std::string ans)
{
    int cntOfBulls = 0, cntOfCows = 0, ind = 0;
```

```
    for (int i = 0; i < ans.size(); i++) {
        if (hidV.find(ans[i]) != hidV.cend()) {
            if (hidV[ans[i]] == i) {
                cntOfBulls++;
            } else {
                cntOfCows++;
            }
        }
    }
    return std::make_pair(cntOfBulls, cntOfCows);
}

void server(std::string &sessionName)
{
    std::string tmp = session.hiddenNum;
    for (int i = 0; i < tmp.size(); i++) {
        hidV.insert({tmp[i], i});
    }
    std::string gameSemName = sessionName + "game.semaphore";
    sem_t *gameSem = sem_open(gameSemName.c_str(), O_CREAT, accessPerm,
0);
    semSetvalue(gameSem, 0);
    int state = 0, firstIt = 1;
    while (1) {
        sem_getvalue(gameSem, &state);
        if (state % (session.cntOfPlayers + 1) == 0 && firstIt == 0) {
            int gameFd = shm_open((sessionName + "game.back").c_str(), O_RDWR |
O_CREAT, accessPerm);
            struct stat statBuf;
            fstat(gameFd, &statBuf);
```

```
        int sz = statBuf.st_size;
        ftruncate(gameFd, sz);
        char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, gameFd, 0);
        std::string strToJson = mapped;
        nlohmann::json request = nlohmann::json::parse(strToJson);
        for (auto i: session.playerList) {
            std::string ansField = i.name + "ans";
            std::string bullsField = i.name + "bulls";
            std::string cowsField = i.name + "cows";
            i.ans = request[ansField];
            std::pair<int, int> result = ggame(i.ans);
            i.bulls = result.first;
            i.cows = result.second;
            request[ansField] = i.ans;
            request[bullsField] = i.bulls;
            request[cowsField] = i.cows;
            if (i.bulls == 4) {
                request["winner"] = i.name;
            }
        }
        std::string strFromJson = request.dump();
        char *buffer = (char *) strFromJson.c_str();
        sz = strlen(buffer) + 1;
        ftruncate(gameFd, sz);
        mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, gameFd, 0);
        memset(mapped, '\0', sz);
        sprintf(mapped, "%s", buffer);
        munmap(mapped, sz);
```

```
            close(gameFd);
            std::cout << session << std::endl;
            sem_post(gameSem);
        } else if (state % (session.cntOfPlayers + 1) == 0 && firstIt == 1) {
            nlohmann::json request;
            for (auto i: session.playerList) {
                std::string ansField = i.name + "ans";
                std::string bullsField = i.name + "bulls";
                std::string cowsField = i.name + "cows";
                request[ansField] = i.ans;
                request[bullsField] = i.bulls;
                request[cowsField] = i.cows;
            }
            std::string strFromJson = request.dump();
            int gameFd = shm_open((sessionName + "game.back").c_str(), O_RDWR |
O_CREAT, accessPerm);
            char *buffer = (char *) strFromJson.c_str();
            int sz = strlen(buffer) + 1;
            ftruncate(gameFd, sz);
            char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, gameFd, 0);
            memset(mapped, '\0', sz);
            sprintf(mapped, "%s", buffer);
            munmap(mapped, sz);
            close(gameFd);
            firstIt = 0;
            std::cout << session << std::endl;
            sem_post(gameSem);
        }
    }
```

```
}

void waitAlllPlayers(std::string &sessionName)
{
   session.curPlayerIndex = 0;
   std::string joinSemName = (sessionName + ".semaphore");

   sem_t *joinSem = sem_open(joinSemName.c_str(), O_CREAT, accessPerm, 0);
   int state = 0;

   sem_getvalue(joinSem, &state);
   std::string joinFdName;
   while (state < session.cntOfPlayers) {
      sem_getvalue(joinSem, &state);
      if (state > session.curPlayerIndex) {
         joinFdName = sessionName.c_str();
         int joinFd = shm_open(joinFdName.c_str(), O_RDWR | O_CREAT,
accessPerm);
         struct stat statBuf;
         fstat(joinFd, &statBuf);
         int sz = statBuf.st_size;
         ftruncate(joinFd, sz);
         char *mapped = (char *) mmap(NULL, sz, PROT_READ | PROT_WRITE,
MAP_SHARED, joinFd, 0);
         std::string strToJson = mapped;
         nlohmann::json joinReply;
         munmap(mapped, sz);
         close(joinFd);
         joinReply = nlohmann::json::parse(strToJson);
         Player playerN;
```

```
            playerN.name = joinReply["name"];
            playerN.ans = joinReply["ans"];
            playerN.bulls = joinReply["bulls"];
            playerN.cows = joinReply["cows"];
            session.playerList.push_back(playerN);
            session.curPlayerIndex = state;
        }
    }
}

int main(int argc, char const *argv[])
{
    std::string strToJson = argv[1];
    nlohmann::json reply;
    reply = nlohmann::json::parse(strToJson);
    session.sessionName = reply["sessionName"];
    session.cntOfPlayers = reply["cntOfPlayers"];
    session.hiddenNum = reply["hiddenNum"];
    session.curPlayerIndex = 0;

    std::string apiSemName = session.sessionName + "api.semaphore";
    sem_unlink(apiSemName.c_str());
    sem_t *apiSem = sem_open(apiSemName.c_str(), O_CREAT, accessPerm, 0);
    semSetvalue(apiSem, session.cntOfPlayers);
    int f = 0;
    sem_getvalue(apiSem, &f);

    waitAlllPlayers(session.sessionName);
```

```
    while (f > 0) {
        sem_wait(apiSem);
        sem_getvalue(apiSem, &f);
        std::cout << f << std::endl;
        sleep(1);
    }
    sem_getvalue(apiSem, &f);
    server(session.sessionName);
    return 0;
}
```

Сборка программы

```
ggame@ggame:~/OS/ready/KP/build$ cmake ..
-- The CXX compiler identification is GNU 11.3.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ggame/OS/ready/KP/build
ggame@ggame:~/OS/ready/KP/build$ make
[ 16%] Building CXX object CMakeFiles/client.dir/src/client.cpp.o
[ 33%] Linking CXX executable client
[ 33%] Built target client
[ 50%] Building CXX object CMakeFiles/apidemon.dir/src/apidemon.cpp.o
[ 66%] Linking CXX executable apidemon
[ 66%] Built target apidemon
[ 83%] Building CXX object CMakeFiles/server.dir/src/server.cpp.o
[100%] Linking CXX executable server
[100%] Built target server
```

Демонстрация работы программы

server:

```
ggame@ggame:~/OS/ready/KP/build$ ./apidemon
Name of session: session
Count of players: 2
Turn of player: 0
Players:
name: player1
bulls: 0
cows: 0
ans: 0000

hidden Number: 4521

1
1
0
Name of session: session
Count of players: 2
Turn of player: 2
Players:
name: player1
bulls: 0
cows: 0
ans: 0000

name: player2
bulls: 0
cows: 0
ans: 0000

hidden Number: 4521

Name of session: session
Count of players: 2
Turn of player: 2
Players:
name: player1
bulls: 0
cows: 0
ans: 0000
```

name: player2
bulls: 0
cows: 0
ans: 0000

hidden Number: 4521

Name of session: session
Count of players: 2
Turn of player: 2
Players:
name: player1
bulls: 0
cows: 0
ans: 0000

name: player2
bulls: 0
cows: 0
ans: 0000

hidden Number: 4521

Name of session: session
Count of players: 2
Turn of player: 2
Players:
name: player1
bulls: 0
cows: 0
ans: 0000

name: player2
bulls: 0
cows: 0
ans: 0000

hidden Number: 4521

Name of session: session
Count of players: 2
Turn of player: 2

Players:
name: player1
bulls: 0
cows: 0
ans: 0000

name: player2
bulls: 0
cows: 0
ans: 0000

hidden Number: 4521

player1:
Input your name: player1

Write:
command [arg1] ... [argn]

create [name] [cntOfPlayers] to create new game session by name and max count of players

join [name] to join exists game session by name

find to find game session

create session 2
Session session created
Session session joined
statistic of player player1:
for answer: "0000"
count of bulls: 0
count of cows: 0
Input number length of 4 with different digits: 1024
statistic of player player1:
for answer: "1024"
count of bulls: 1
count of cows: 2
Input number length of 4 with different digits: 4125
statistic of player player1:
for answer: "4125"
count of bulls: 2

count of cows: 2
Input number length of 4 with different digits: 4521
Game over. Winner is "player1"

player2:
ggame@ggame:~/OS/ready/KP/build$ ./client
Input your name: player2

Write:
 command [arg1] ... [argn]

create [name] [cntOfPlayers] to create new game session by name and max count of players

join [name] to join exists game session by name

find to find game session

join session
Session session joined
statistic of player player2:
for answer: "0000"
count of bulls: 0
count of cows: 0
Input number length of 4 with different digits: 7895
statistic of player player2:
for answer: "7895"
count of bulls: 0
count of cows: 1
Input number length of 4 with different digits: 4236
statistic of player player2:
for answer: "4236"
count of bulls: 1
count of cows: 1
Input number length of 4 with different digits: 4321
Game over. Winner is "player1"

Выводы

Разделяемая память хорошо справляются со своей задачей коммуникации между процессами. Был получен опыт разработки

консольной клиент-серверной игры. Благодаря этому я понимаю, как происходит процесс общения между клиентом и сервером.