Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Операционные системы»**

**ПРОЕКТИРОВАНИЕ КОНСОЛЬНОЙ КЛИЕНТ-СЕРВЕРНОЙ ИГРЫ**

Студент: Сектименко Ирина Владимировна

Группа: М8О–210Б–22

Вариант: 11

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы:

* Приобретение практических навыков в использовании знаний, полученных в течение курса
* Проведение исследования в выбранной предметной области

## Задание

Необходимо спроектировать и реализовать программный прототип в соответствие с выбранным вариантом (консоль-серверная игра).

Необходимо написать консоль-серверную игру. Она состоит из двух программ: 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

Игра «Быки и коровы» (угадывать необходимо слова). 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. Если кто-то из игроков вышел из игры, то игра должна быть продолжена.

## Общие сведения о программе

Для работы программы сначала запускается сервер (исполняемый файл, созданный из файла apidemon.cpp), а потом клиенты (один или несколько с параллельных терминалов) из исполняемых файлов, скомпилированный на основе client.cpp. Игра начнется, когда в комнате будет максимальное количество игроков. В файле apidemon.cpp обрабатывается первичное подключение клиентов, он же подключает сервер из скомпилированного файла serever.cpp командой execl.

В хедере kptools.h описан namespace gametools, в котором описаны классы Player и Session. Данные, которыми обмениваются клиент и сервер, хранятся в структуре json. Обмен данным происходит с помощью memory mapped file. Для синхронизации процессов разных клиентов используются семафоры.

## Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Создать вспомогательные классы Player и Session. Подключить необходимые для работы библиотеки.
2. Создание клиента:
   1. При запуске программы пользователь вводит имя (создается объект класса Player), команда создания или подключения к игре.
   2. Функция создания игры. В json записывается вся информация о создаваемой игре, генерируется число. Открывается файл с разделяемой между потоками памятью (memory mapped file), через него отправляется запрос на сервер и приходит ответ. Если игра создана, то игрок подключается к созданной игре.
   3. Функция подключения к игре. Создается json файл с информацией об игроке и игре, к которой необходимо подключиться. Информация на проверку отправляется на сервер. Если все хорошо, то игрок подключается к игре. Когда все игроки подключены, игра начинается.
   4. Функция client отвечает за игру пользователей. Пользователь вводит число, отправляет его на сервер, получает ответ и выводит результат.
   5. Для синхронизации всех пользователей используются семафоры.
3. Создание сервера:
   1. В файле apidemon происходит первичное взаимодействие клиента и сервера (обрабатывается запрос на создание игры или подключение к ней). Из него же и запускается непосредственно программа сервера.
   2. Сервер создает сессию (комнату для игроков), когда достигнуто максимальное количество игроков.
   3. В бесконечном цикле у него происходит обработка всех запросов от всех клиентов (их числа) и возвращение ответа клиентам.

## Основные файлы программы

**kptools.h:**

#pragma once

#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";

}

}

}

**apidemon.h:**

#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 "../json/single\_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 {

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;

}

**calculating\_node.h:**

#pragma once

#include <bits/stdc++.h>

#include "ZMQFunctions.h"

#include "unistd.h"

class CalculationNode {

private:

zmq::context\_t context;

public:

zmq::socket\_t left, right, parent;

int id, left\_id = -2, right\_id = -2, parent\_id;

int left\_port, right\_port, parent\_port;

CalculationNode(int id, int parent\_port, int parent\_id):

id(id),

parent\_port(parent\_port),

parent\_id(parent\_id),

left(context, ZMQ\_REQ),

right(context, ZMQ\_REQ),

parent(context, ZMQ\_REP)

{

if (id != -1) {

connect(parent, parent\_port);

}

}

std::string create(int child\_id) {

int port;

bool isleft = false;

if (left\_id == -2) {

left\_port = bind(left, child\_id);

left\_id = child\_id;

port = left\_port;

isleft = true;

}

else if (right\_id == -2) {

right\_port = bind(right, child\_id);

right\_id = child\_id;

port = right\_port;

}

else {

std::string fail = "Error: can not create the calculation node";

return fail;

}

int fork\_id = fork();

if (fork\_id == 0) {

if (execl("./server", "server", std::to\_string(child\_id).c\_str(), std::to\_string(port).c\_str(), std::to\_string(id).c\_str(), (char\*)NULL) == -1) {

std::cout << "Error: can not run the execl-command" << std::endl;

exit(EXIT\_FAILURE);

}

}

else {

std::string child\_pid;

try {

if (isleft) {

int time = 3000;

left.setsockopt(ZMQ\_SNDTIMEO, time);

send\_message(left, "pid");

child\_pid = receive\_message(left);

}

else {

int time = 3000;

right.setsockopt(ZMQ\_SNDTIMEO, time);

send\_message(right, "pid");

child\_pid = receive\_message(right);

}

return "Ok: " + child\_pid;

}

catch (int) {

std::string fail = "Error: can not connect to the child";

return fail;

}

}

}

std::string ping(int id) {

std::string answer = "Ok: 0";

if (this->id == id) {

answer = "Ok: 1";

return answer;

}

else if (left\_id == id) {

std::string message = "ping " + std::to\_string(id);

send\_message(left, message);

try {

message = receive\_message(left);

if (message == "Ok: 1") {

answer = message;

}

}

catch(int){}

}

else if (right\_id == id) {

std::string message = "ping " + std::to\_string(id);

send\_message(right, message);

try {

message = receive\_message(right);

if (message == "Ok: 1") {

answer = message;

}

}

catch(int){}

}

return answer;

}

std::string sendstring (std::string string, int id) {

std::string answer = "Error: Parent not found";

if (left\_id == -2 && right\_id == -2) {

return answer;

}

else if (left\_id == id) {

if (ping(left\_id) == "Ok: 1") {

send\_message(left, string);

try{

answer = receive\_message(left);

}

catch(int){}

}

}

else if (right\_id == id) {

if (ping(right\_id) == "Ok: 1") {

send\_message(right, string);

try {

answer = receive\_message(right);

}

catch(int){}

}

}

else {

if (ping(left\_id) == "Ok: 1") {

std::string message = "send " + std::to\_string(id) + " " + string;

send\_message(left, message);

try {

message = receive\_message(left);

}

catch(int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

if (ping(right\_id) == "Ok: 1") {

std::string message = "send " + std::to\_string(id) + " " + string;

send\_message(right, message);

try {

message = receive\_message(right);

}

catch(int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

}

return answer;

}

std::string exec (std::string string, std::string pattern) {

std::string result = "";

int m = pattern.size(), n = string.size() - m;

for (int i = 0; i < n; ++i) {

bool flag = true;

for (int j = 0; j < m; ++j) {

if (string[i + j] != pattern[j]) {

flag = false;

break;

}

}

if (flag) {

result += (std::to\_string(i + 1) + " ");

}

}

std::string answer = "Ok: " + std::to\_string(id) + ": " + result;

return answer;

}

std::string treeclear (int child) {

if (left\_id == child) {

left\_id = -2;

unbind(left, left\_port);

}

else {

right\_id = -2;

unbind(right, right\_port);

}

return "Ok";

}

std::string kill () {

if (left\_id != -2){

if (ping(left\_id) == "Ok: 1") {

std::string message = "kill";

send\_message(left, message);

try {

message = receive\_message(left);

}

catch(int){}

unbind(left, left\_port);

left.close();

}

}

if (right\_id != -2) {

if (ping(right\_id) == "Ok: 1") {

std::string message = "kill";

send\_message(right, message);

try {

message = receive\_message(right);

}

catch (int){}

unbind(right, right\_port);

right.close();

}

}

return std::to\_string(parent\_id);

}

~CalculationNode() {}

};

**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 "../json/single\_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;

}

**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 "../json/single\_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 == "end") {

int first, second;

sem\_getvalue(apiSem, &first);

sem\_getvalue(apiSem, &second);

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);

}

}

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\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 {

std::cout << "Wrong command!\n";

continue;

}

}

return 0;

}

## Пример работы

**Пользователь 1**

irina@irina-VirtualBox:~/course\_2/os\_lab\_kr/build$ ./client

Input your name: r

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 1 2

Session 1 created

Session 1 joined

statistic of player r:

for answer: "0000"

count of bulls: 0

count of cows: 0

Input number length of 4 with different digits: 1234

statistic of player r:

for answer: "1234"

count of bulls: 0

count of cows: 0

Input number length of 4 with different digits: 6859

Game over. Winner is "r"

**Пользователь 2**

irina@irina-VirtualBox:~/course\_2/os\_lab\_kr/build$ ./client

Input your name: rr

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 1

Session 1 joined

statistic of player rr:

for answer: "0000"

count of bulls: 0

count of cows: 0

Input number length of 4 with different digits: 5678

statistic of player rr:

for answer: "5678"

count of bulls: 0

count of cows: 3

Input number length of 4 with different digits: 4567

Game over. Winner is "r"

## Вывод

В ходе курсовой работы я поняла, что memory mapped file – технология, которую можно использовать в достаточно больших проектах (а не только в лабораторных работах). Я научилась работать с json структурами на C++ (раньше уже работала на python), это намного упростило работу и мне не пришлось самостоятельно парсить строки.

Так же в процессе создания программы мне пришлось применить семафоры (к сожалению, в лабораторной работе у меня было задание, для написания которого они не требовались). Без них программа вела себя не предсказуемо.