Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Курсовой проект

«Быки и коровы на memory map»

Группа: М80-206Б-22

Студент: Голубев Т. Д.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 08.01.24

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.

Проектирование консольной клиент-серверной игры

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

1. Создать игру, введя ее имя,

2. Присоединиться к одной из существующих игр по имени игры. «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.

Общий метод и алгоритм решения

Сервер принимает запросы от клиентов. Среди запросов может быть создание игры. Это запускает новый процесс «Сервер игры». Клиенты подключаются к этому серверу, общаются с ним. Игра происходит через запросы к этому серверу.

Процессы общаются с помощью memory map. Существует структура «Сообщение», которая состоит из полей «Тип» и «Данные». Общение клиента с сервером осуществляется с помощью этих сообщений, которые размещаются в memory map.

Во время запуска игры случайно выбирается слово из базы данных. Собираются ответы пользователей и проверяется их правильность. Пользователям сообщается количество «быков» (угаданных букв с точностью до позиции) и «коров» (угаданных букв с неверной позицией). Игра завершается, когда хотя бы один из игроков угадал слово.

mmap.h

```
#pragma once

#include <unistd.h>
#include <iostream>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <exception>
#include <string>

namespace bc {
    enum ModeFlags {
        read = PROT_READ,
        write = PROT_WRITE,
        exec = PROT_EXEC,
        none = PROT_NONE
    };

    template <class T>
    class MemoryMap {
    public:
        MemoryMap() = delete;
        MemoryMap(const std::string& s, size_t size, int mode);
        void delete_shm_file();
        ~MemoryMap();
        MemoryMap(const MemoryMap<T>& other);
        MemoryMap(MemoryMap<T>&& other) noexcept;
        MemoryMap<T>& operator=(const MemoryMap<T>& other);
```

```
29          MemoryMap<T>& operator=(MemoryMap<T>&& other) noexcept;
30          T* data() const noexcept;
31          size_t size() const noexcept;
32          T& operator[](int idx);
33          const T& operator[](int idx) const;
34          const std::string& name() const;
35      private:
36          T* _data;
37          int _fd;
38          std::string _name;
39          size_t _size;
40          int _mode;
41      };
42
43      template <class T>
44      MemoryMap<T>::MemoryMap(const std::string& name, size_t size, int mode) :
45  _name{name}, _size{size}, _mode{mode} {
46          _fd = shm_open(name.c_str(), O_CREAT | O_RDWR, S_IREAD | S_IWRITE);
47          if (ftruncate(_fd, sizeof(T) * size) != 0) {
48              throw std::runtime_error("ftruncate error");
49          }
50          if (_fd == -1) {
51              throw std::runtime_error("shm_open error");
52          }
53          _data = (T*) mmap(NULL, size, mode, MAP_SHARED, _fd, 0);
54          if (_data == MAP_FAILED) {
55              throw std::runtime_error("mmap error");
56          }
57      }
58
59      template <class T>
60      void MemoryMap<T>::delete_shm_file() {
61          int error_code = shm_unlink(_name.c_str());
62          if (error_code == -1) {
63              throw std::runtime_error("shm_unlink");
64          }
65      }
66
67      template <class T>
68      MemoryMap<T>::~MemoryMap() {
69          munmap(_data, _size);
70      }
71
72      template <class T>
73      MemoryMap<T>::MemoryMap(const MemoryMap<T>& other) :
74      MemoryMap<T>(other._name, other._size, other._mode) {}
75
76      template <class T>
77      MemoryMap<T>::MemoryMap(MemoryMap<T>&& other) noexcept :
78      _data{other._data},
79      _fd{std::move(other._fd)},
80      _name{std::move(other._name)},
81      _size{std::move(other._size)},
82      _mode{std::move(other._mode)} {}
83
84      template <class T>
85      MemoryMap<T>& MemoryMap<T>::operator=(const MemoryMap<T>& other) {
86          _name = other._name;
87          _size = other._size;
88          _mode = other._mode;
89          _fd = shm_open(_name.c_str(), O_CREAT | O_RDWR, S_IREAD | S_IWRITE);
90          if (ftruncate(_fd, sizeof(T) * _size) != 0) {
```

```
91              throw std::runtime_error("ftruncate error");
92          }
93          if (_fd == -1) {
94              throw std::runtime_error("shm_open error");
95          }
96          _data = (T*) mmap(NULL, _size, other._mode, MAP_SHARED, _fd, 0);
97          if (_data == MAP_FAILED) {
98              throw std::runtime_error("mmap error");
99          }
100         return *this;
101     }
102
103     template <class T>
104     MemoryMap<T>& MemoryMap<T>::operator=(MemoryMap<T>&& other) noexcept {
105         _data = other._data;
106         _fd = std::move(other._fd);
107         _name = std::move(other._name);
108         _size = std::move(other._size);
109         _mode = std::move(other._mode);
110         return *this;
111     }
112
113     template <class T>
114     T* MemoryMap<T>::data() const noexcept {
115         return _data;
116     }
117
118     template <class T>
119     size_t MemoryMap<T>::size() const noexcept {
120         return _size;
121     }
122
123     template <class T>
124     T& MemoryMap<T>::operator[](int idx) {
125         if (idx > _size - 1) {
126             throw std::range_error("out of range");
127         }
128         return _data[idx];
129     }
130
131     template <class T>
132     const T& MemoryMap<T>::operator[](int idx) const {
133         if (idx > _size - 1) {
134             throw std::range_error("out of range");
135         }
136         return _data[idx];
137     }
138
139     template <class T>
140     const std::string& MemoryMap<T>::name() const {
141         return _name;
142     }
143
144     template <class T>
145     void str_to_mmap(const std::string& str, MemoryMap<T>& mmap, int start_idx) {
146         if ((mmap.size() - start_idx) < str.size()) throw std::logic_error("string
147 is too long");
148         int j = start_idx;
149         for (int i = 0; i < str.size(); ++i) {
150             mmap[j] = str[i];
151             ++j;
152         }
```

```
153         mmap[start_idx + str.size()] = '\0';
154     }
155 
156     template <class T>
157     std::string mmap_to_str(const MemoryMap<T>& mmap, int start_idx) {
158         std::string str(mmap.size() - start_idx, ' ');
159         int i = start_idx;
160         int j = 0;
161         while (mmap[i] != '\0') {
162             str[j] = mmap[i];
163             ++i;
164             ++j;
165         }
166         return str;
        }
    }   // bulls & cows
```

mutex.h

```
1  #pragma once
2  
3  #include <pthread.h>
4  #include <string>
5  #include "mmap.h"
6  
7  /* This is my process-shared mutex */
8  
9  namespace bc {
10     enum MutexFlag {
11         create,
12         connect
13     };
14 
15     class Mutex {
16     private:
17         std::string _name;
18         MemoryMap<pthread_mutex_t> _mtx;
19     public:
20         Mutex(const std::string& name, MutexFlag flag);
21         Mutex(const Mutex& other);
22         Mutex(Mutex&& other) noexcept;
23         ~Mutex() noexcept;
24         Mutex& operator=(const Mutex& other);
25         Mutex& operator=(Mutex&& other) noexcept;
26         void lock();
27         void unlock();
28         const std::string& name() const;
29         void delete_for_all();   // it delete shmfile, should be called by server
30     };
31 }   // bulls & cows
```

mutex.cpp

```
1  #include "mutex.h"
2  
3  using namespace bc;
4  
```

```
Mutex::Mutex(const std::string& name, MutexFlag flag) : _name{name}, _mtx(name, 1,
ModeFlags::write | ModeFlags::read) {
    if (flag == MutexFlag::create) {
        pthread_mutexattr_t attr;
        pthread_mutexattr_init(&attr);
        pthread_mutexattr_setpshared(&attr, PTHREAD_PROCESS_SHARED);
        pthread_mutex_init(&_mtx[0], &attr);
    }
}

Mutex::Mutex(const Mutex& other) : _name{other._name}, _mtx{other._mtx} {}

Mutex::Mutex(Mutex&& other) noexcept : _name{std::move(other._name)},
_mtx{std::move(other._mtx)} {}

Mutex::~Mutex() noexcept {
    // there is no memory leak because of destructor of bc::MemoryMap
}

Mutex& Mutex::operator=(const Mutex& other) {
    _name = other._name;
    _mtx = other._mtx;
    return *this;
}

Mutex& Mutex::operator=(Mutex&& other) noexcept {
    _name = std::move(other._name);
    _mtx = std::move(other._mtx);
    return *this;
}

void Mutex::lock() {
    int res = pthread_mutex_lock(&_mtx[0]);
    if (res != 0) throw std::runtime_error(std::to_string(res));
}

void Mutex::unlock() {
    int res = pthread_mutex_unlock(&_mtx[0]);
    if (res != 0) throw std::runtime_error(std::to_string(res));
}

const std::string& Mutex::name() const {
    return _name;
}

void Mutex::delete_for_all() {
    _mtx.delete_shm_file();
}
```

server.h

```
#pragma once

#include <string>
#include <unistd.h>
#include "mmap.h"
#include "mutex.h"
#include "message.h"

namespace bc {
    class Server {
    private:
```

```
12          int _id;
13          MemoryMap<char> _mmap;
14          Mutex _mtx;
15          static const int _mmap_size = 10000;
16      public:
17          Server();
18          ~Server();
19          void create_game(const std::string& game_name, int players);
20          int id() const;
21          void send_message(const Message& msg);
22          Message get_message();
23          void clear_mmap();
24      };
25  }   // bulls & cows
```

server.cpp

```
1  #include "server.h"
2
3  using namespace bc;
4
5  Server::Server() : _id{getpid()},
6  _mmap("bc_" + std::to_string(_id) + "_mmap", _mmap_size, ModeFlags::read |
7  ModeFlags::write),
8  _mtx("bc_" + std::to_string(_id) + "_mutex", MutexFlag::create) {
9      clear_mmap();
10 }
11
12 Server::~Server() {
13     _mtx.delete_for_all();
14     _mmap.delete_shm_file();
15 }
16
17 void Server::create_game(const std::string& game_name, int players) {
18     int pid = fork();
19     if (pid == -1) throw std::runtime_error("fork error");
20     if (pid == 0) {
21         int er = execl("../build/kp_game", "../build/kp_game", game_name.c_str(),
22 std::to_string(players).c_str(), NULL);
23         if (er == -1) {
24             std::cout << "create er: " << errno << std::endl;
25             exit(1);
26         }
27     }
28 }
29
30 int Server::id() const {
31     return _id;
32 }
33
34 void Server::send_message(const Message& msg) {
35     _mtx.lock();
36     _mmap[0] = msg.type;
37     str_to_mmap(msg.data, _mmap, 1);
38     _mtx.unlock();
39 }
40
41 Message Server::get_message() {
42     Message msg;
43     _mtx.lock();
44     msg.type = (MessageType) _mmap[0];
45     if (msg.type == MessageType::wait) {
```

```
46          _mtx.unlock();
47          return msg;
48      }
49      msg.data = mmap_to_str(_mmap, 1);
50      _mtx.unlock();
51      return msg;
52  }
53
54  void Server::clear_mmap() {
55      _mtx.lock();
56      _mmap[0] = MessageType::wait;
57      _mmap[1] = '\0';
        _mtx.unlock();
    }
```

game.h

```
 1  #pragma once
 2
 3  #include <set>
 4  #include <fstream>
 5  #include <random>
 6  #include "mmap.h"
 7  #include "mutex.h"
 8  #include "message.h"
 9
10  namespace bc {
11      class Game {
12      private:
13          MemoryMap<char> _mmap;
14          Mutex _mtx;
15          std::string _name;
16          int _players;
17          std::string _word;
18          int _words;  // count of
19          std::set<char> _letters;
20          std::ifstream _db;
21          int _winner_id;
22          static const int _mmap_size = 10000;
23      public:
24          Game(const std::string& name, int players, const std::string& db_name);
25          Game(const Game& other) = delete;
26          ~Game() noexcept;
27          void send_message(const Message& msg);
28          Message get_message();
29          void clear_mmap();
30          const std::string& word() const;
31          int players() const;
32          int& players();
33          const std::string& name() const;
34          MemoryMap<char>& mmap();
35          Mutex& mtx();
36          int winner_id() const;
37          std::string check_word(const std::string& word, int player_id);
38      private:
39          std::string _generate_word();
40      };
41  }   // bulls & cows
```

game.cpp

```
1  #include "game.h"
```

```cpp

using namespace bc;

Game::Game(const std::string& name, int players, const std::string& db_name) :
_name{name},
_mmap{"bcgame_" + name + "_mmap", _mmap_size, ModeFlags::read | ModeFlags::write},
_mtx{"bcgame_" + name + "_mutex", MutexFlag::create},
_db{db_name, std::ios::binary | std::ios::in},
_players{players},
_winner_id{0} {
    if (!_db.good()) throw std::runtime_error("Can't open database file");
    _db.read(reinterpret_cast<char*>(&_words), sizeof(int));
    _word = _generate_word();
    for (char letter : _word) {
        _letters.insert(letter);
    }
}

Game::~Game() noexcept {
    _db.close();
    _mmap.delete_shm_file();
    _mtx.delete_for_all();
}

void Game::send_message(const Message& msg) {
    _mtx.lock();
    _mmap[0] = msg.type;
    str_to_mmap(msg.data, _mmap, 1);
    _mtx.unlock();
}

Message Game::get_message() {
    Message msg;
    _mtx.lock();
    msg.type = (MessageType) _mmap[0];
    msg.data = mmap_to_str(_mmap, 1);
    _mtx.unlock();
    return msg;
}

std::string Game::_generate_word() {
    std::random_device dev;
    std::mt19937 rng(dev());
    std::uniform_int_distribution<std::mt19937::result_type> dist(0, _words - 1);
    _db.seekg(sizeof(int) + sizeof(char) * 5 * dist(rng), std::ios::beg);
    char raw_word[5];
    _db.read(raw_word, sizeof(char) * 5);
    std::string word(raw_word);
    return word;
}

const std::string& Game::word() const {
    return _word;
}

void Game::clear_mmap() {
    _mtx.lock();
    _mmap[0] = MessageType::wait;
    _mmap[1] = '\0';
    _mtx.unlock();
}

```

```cpp
int Game::players() const {
    return _players;
}

int& Game::players() {
    return _players;
}

const std::string& Game::name() const {
    return _name;
}

MemoryMap<char>& Game::mmap() {
    return _mmap;
}

Mutex& Game::mtx() {
    return _mtx;
}

int Game::winner_id() const {
    return _winner_id;
}

std::string Game::check_word(const std::string& word, int player_id) {
    int cows = 0;
    std::string str_cows = "";
    int bulls = 0;
    std::string str_bulls = "";
    for (int i = 0; i < word.size(); ++i) {
        if (word[i] == _word[i]) {
            ++bulls;
            str_bulls += word[i];
            str_bulls += ' ';
        } else if (_letters.find(word[i]) != _letters.end()) {
            ++cows;
            str_cows += word[i];
            str_cows += ' ';
        }
    }
    if (bulls == _word.size() && _winner_id == 0) {
        _winner_id = player_id;
    }
    std::string ans = std::to_string(cows) + ' ' + str_cows + ' ' +
std::to_string(bulls) + str_bulls;
    return ans;
}
```

client.h

```cpp
#pragma once

#include <unistd.h>
#include <string>
#include <thread>
#include <chrono>
#include "mmap.h"
#include "mutex.h"
#include "message.h"

namespace bc {
    class Client {
```

```
    private:
        int _id;
        MemoryMap<char> _mmap;
        Mutex _mtx;
        int _server_id;
        static const int _mmap_size = 10000;
    public:
        Client(int server_id);
        ~Client();
        int id() const;
        void create_game(const std::string& game_name, int players);
        void connect_to_game(const std::string& game_name);
        void connect_to_server();
        void send_message(const Message& msg);
        bool ping();
        void clear_mmap();
        Message get_message();
    };
}   // bulls & cows
```

client.cpp

```
#include "client.h"

using namespace bc;

Client::Client(int server_id) : _id{getpid()}, _server_id{server_id},
_mmap("bc_" + std::to_string(server_id) + "_mmap", _mmap_size, ModeFlags::read |
ModeFlags::write),
_mtx("bc_" + std::to_string(server_id) + "_mutex", MutexFlag::connect) {}

Client::~Client() {}

int Client::id() const {
    return _id;
}

void Client::create_game(const std::string& game_name, int players) {
    // 4 = players_num + ' ' + state + end of str
    if (game_name.size() > _mmap_size - 4) throw std::logic_error("Too long game
name");
    Message msg;
    msg.type = MessageType::server_create_game;
    msg.data = game_name + ' ' + std::to_string(players) + '\0';
    send_message(msg);
}

void Client::connect_to_game(const std::string& game_name) {
    MemoryMap<char> mmap("bcgame_" + game_name + "_mmap", _mmap_size,
ModeFlags::read | ModeFlags::write);
    _mmap = mmap;
    Mutex mtx("bcgame_" + game_name + "_mutex", MutexFlag::connect);
    _mtx = mtx;
}

void Client::connect_to_server() {
    MemoryMap<char> mmap("bc_" + std::to_string(_server_id) + "_mmap", _mmap_size,
ModeFlags::read | ModeFlags::write);
    _mmap = mmap;
    Mutex mtx("bc_" + std::to_string(_server_id) + "_mutex", MutexFlag::connect);
    _mtx = mtx;
}
```

```
41
42 void Client::send_message(const Message& msg) {
43     _mtx.lock();
44     _mmap[0] = msg.type;
45     str_to_mmap(msg.data, _mmap, 1);
46     _mtx.unlock();
47 }
48
49 Message Client::get_message() {
50     Message msg;
51     _mtx.lock();
52     msg.type = (MessageType) _mmap[0];
53     msg.data = mmap_to_str(_mmap, 1);
54     _mtx.unlock();
55     return msg;
56 }
57
58 bool Client::ping() {
59     Message msg;
60     msg.type = MessageType::ping;
61     msg.data = "ping";
62     send_message(msg);
63     msg = get_message();
64     int attempts = 0;
65     while (msg.type == MessageType::ping) {
66         ++attempts;
67         std::this_thread::sleep_for(std::chrono::milliseconds(100));
68         msg = get_message();
69         if (attempts == 20) break;
70     }
71     if (msg.type == MessageType::wait || msg.type == MessageType::start_round)
72 return true;
73     return false;
74 }
75
76 void Client::clear_mmap() {
      _mtx.lock();
      _mmap[0] = MessageType::wait;
      _mmap[1] = '\0';
      _mtx.unlock();
  }
```

message.h

```
1  #pragma once
2
3 #include <string>
4
5 namespace bc {
6     enum MessageType {
7         wait,
8         server_create_game,
9         ping,
10        start_round,
11        end_game,
12        guess,
13        ans_guess
14    };
15
16    struct Message {
17        MessageType type;
18        std::string data;
```

```
    };
}   // bulls & cows
```

server_main.cpp

```
#include <iostream>
#include <thread>
#include <chrono>
#include <sstream>
#include "server.h"

using namespace bc;

void routine(Server& ser) {
    Message msg;
    while (true) {
        msg = ser.get_message();
        switch (msg.type) {
            case MessageType::wait: {
                std::this_thread::sleep_for(std::chrono::milliseconds(1000));
                break;
            }
            case MessageType::server_create_game: {
                std::istringstream oss(msg.data);
                std::string game_name;
                int players;
                oss >> game_name >> players;
                try {
                    ser.create_game(game_name, players);
                } catch (std::exception& ex) {
                    std::cout << "Error: " << ex.what() << std::endl;
                }
                std::cout << "Game \"" << game_name << "\" has been created" <<
std::endl;
                ser.clear_mmap();
                break;
            }
            case MessageType::ping: {
                ser.clear_mmap();
                break;
            }
        }
    }
}

int main() {
    std::cout << "Starting server" << std::endl;
    Server ser;
    std::cout << "Server id: " << ser.id() << std::endl;
    routine(ser);

    return 0;
}
```

game_main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <thread>
#include <chrono>
#include <sstream>
```

```cpp
#include "game.h"

using namespace bc;

const std::string DB_NAME = "../db/test_db.bin";

void routine(Game& game) {
    int players = 0;
    int attempts = 0;
    Message msg;
    bool is_game_ended = false;
    bool while_cond = true;
    while (while_cond) {
        msg = game.get_message();
        switch (msg.type) {
            case MessageType::wait: {
                std::this_thread::sleep_for(std::chrono::milliseconds(1000));
                ++attempts;
                if (attempts > 30) --game.players();
                if (players == game.players()) {
                    if (is_game_ended)  {
                        while_cond = false;
                        break;
                    }
                    Message call;
                    call.type = MessageType::start_round;
                    call.data = "";
                    game.send_message(call);
                    players = 0;
                    attempts = 0;
                }
                break;
            }
            case MessageType::ping: {
                attempts = 0;
                ++players;
                game.clear_mmap();
                break;
            }
            case MessageType::guess: {
                attempts = 0;
                ++players;
                std::istringstream iss(msg.data);
                int pid;
                std::string word;
                iss >> pid >> word;
                std::string ans = game.check_word(word, pid);
                if (game.winner_id() != 0) {
                    is_game_ended = true;
                }
                msg.type = MessageType::ans_guess;
                msg.data = std::to_string(pid) + " " + ans;
                game.send_message(msg);
                break;
            }
            default:
                attempts = 0;
                std::this_thread::sleep_for(std::chrono::milliseconds(1000));
                break;
        }
    }
    msg.type = MessageType::end_game;
```

```
    msg.data = std::to_string(game.winner_id());
    game.send_message(msg);
}

int main(int argc, char** argv) {
    if (argc != 3) {
        std::cerr << "Wrong number of args\n";
        return 1;
    }
    std::string filename(argv[1]);
    filename += "_log.txt";
    std::ofstream ofs(filename);
    ofs << "Game started.\n";
    int players = std::stoi(argv[2]);
    Game game(argv[1], players, DB_NAME);
    ofs << "Word: " << game.word() << '\n';
    routine(game);
    ofs << "Winner id: " << game.winner_id() << '\n';
    ofs << "Game ended.\n";
    ofs.close();

    return 0;
}
```

client_main.cpp

```
#include <iostream>
#include <sstream>
#include "client.h"

using namespace bc;

void help() {
    std::cout << "1. Create a game\n";
    std::cout << "2. Connect to the game\n";
    std::cout << "q. Exit" << std::endl;
}

void game(Client& cl) {
    std::this_thread::sleep_for(std::chrono::milliseconds(500));
    if (!cl.ping()) {
        std::cerr << "Can't connect to game\n";
        exit(1);
    }
    std::cout << "Waiting for start" << std::endl;
    Message msg;
    bool is_new_round_started = false;
    while (true) {
        msg = cl.get_message();
        if (msg.type == MessageType::start_round) {
            std::cout << "Round started" << std::endl;
            is_new_round_started = true;
        } else if (msg.type == MessageType::wait || msg.type ==
MessageType::guess) {
            std::this_thread::sleep_for(std::chrono::milliseconds(1000));
        } else if (msg.type == MessageType::ans_guess) {
            std::istringstream iss(msg.data);
            int pid;
            iss >> pid;
            if (pid != cl.id()) {
                std::this_thread::sleep_for(std::chrono::milliseconds(100));
                continue;
```

```
37              }
38              int cows, bulls;
39              iss >> cows;
40              std::cout << "cows " << cows << ": ";
41              for (int i = 0; i < cows; ++i) {
42                  char letter;
43                  iss >> letter;
44                  std::cout << letter << ' ';
45              }
46              std::cout << '\n';
47              iss >> bulls;
48              std::cout << "bulls " << bulls << ": ";
49              for (int i = 0; i < bulls; ++i) {
50                  char letter;
51                  iss >> letter;
52                  std::cout << letter << ' ';
53              }
54              std::cout << std::endl;
55              cl.clear_mmap();
56          } else if (msg.type == MessageType::end_game) {
57              std::istringstream iss(msg.data);
58              int player_id;
59              iss >> player_id;
60              if (player_id == cl.id()) {
61                  std::cout << "You win!" << std::endl;
62              } else {
63                  std::cout << "Player " << player_id << " win." << std::endl;
64              }
65              return;
66          }
67          if (is_new_round_started) {
68              is_new_round_started = false;
69              std::cout << "Enter a word: ";
70              std::string word;
71              std::cin >> word;
72              while (word.size() != 5) {
73                  std::cout << "Word lenght must be 5" << std::endl;
74                  std::cout << "Enter a word: ";
75                  std::cin >> word;
76              }
77              Message guess_msg;
78              guess_msg.type = MessageType::guess;
79              guess_msg.data = std::to_string(cl.id()) + word;
80              msg = cl.get_message();
81              while (msg.type != MessageType::start_round && msg.type !=
82 MessageType::wait) {
83                  std::this_thread::sleep_for(std::chrono::milliseconds(1000));
84                  msg = cl.get_message();
85              }
86              if (msg.type == MessageType::end_game) {
87                  std::istringstream iss(msg.data);
88                  int player_id;
89                  iss >> player_id;
90                  if (player_id == cl.id()) {
91                      std::cout << "You win!" << std::endl;
92                  } else {
93                      std::cout << "Player " << player_id << " win." << std::endl;
94                  }
95              }
96              cl.send_message(guess_msg);
97          }
98      }
```

```
 99 }
100
101 void interface(Client& cl) {
102     char cmd;
103     std::cout << "Type <h> for help page" << std::endl;
104     while (true) {
105         std::cout << "> ";
106         std::cin >> cmd;
107         switch (cmd) {
108             case '1': {
109                 std::string game_name;
110                 int players;
111                 std::cout << "Enter game name: ";
112                 std::cin >> game_name;
113                 std::cout << "Enter number of players: ";
114                 std::cin >> players;
115                 try {
116                     cl.create_game(game_name, players);
117                 } catch (std::exception& ex) {
118                     std::cout << "Error: " << ex.what() << std::endl;
119                     break;
120                 }
121                 cl.connect_to_game(game_name);
122                 game(cl);
123                 cl.connect_to_server();
124                 break;
125             }
126             case '2': {
127                 std::string game_name;
128                 std::cout << "Enter game name: ";
129                 std::cin >> game_name;
130                 cl.connect_to_game(game_name);
131                 game(cl);
132                 cl.connect_to_server();
133                 break;
134             }
135             case 'q': {
136                 return;
137             }
138             case 'h': {
139                 help();
140                 break;
141             }
142             default: {
143                 std::cout << "Wrong command" << std::endl;
144                 break;
145             }
146         }
147     }
148 }
149
150 int main() {
151     std::cout << "Starting client" << std::endl;
152     std::cout << "Enter server id: ";
153     int server_id;
154     std::cin >> server_id;
155     Client cl(server_id);
156     std::cout << "Your client id: " << cl.id() << std::endl;
157     if (!cl.ping()) {
158         std::cerr << "Can't connect to server\n";
159         return 1;
160     }
```

```
161     interface(cl);
162 
        return 0;
    }
```

Протокол работы программы

Тестирование:

```
cat_mood@nuclear-box:~/programming/mai-os-labs/kp/build$ ./kp_server
Starting server
Server id: 13996
Game "game" has been created
cat_mood@nuclear-box:~/programming/mai-os-labs/kp/build$ ./kp_client
Starting client
Enter server id: 13996
Your client id: 14035
Type <h> for help page
> h
1. Create a game
2. Connect to the game
q. Exit
> 2
Enter game name: game
Waiting for start
Round started
Enter a word: which
cows 0:
bulls 0:
Round started
Enter a word: about
cows 0:
bulls 5: a b o u t
You win!
cat_mood@nuclear-box:~/programming/mai-os-labs/kp/build$ ./kp_client
Starting client
```

```
Enter server id: 13996

Your client id: 14137

Type <h> for help page

> 1

Enter game name: game

Enter number of players: 2

Waiting for start

Round started

Enter a word: there

cows 1: t

bulls 0:

Round started

Enter a word: about

cows 0:

bulls 5: a b o u t

Player 14035 win.
```

Вывод

В ходе лабораторной работы я получил опыт работы с shared memory, shared mutex, разработки игр. Я столкнулся с проблемой синхронизации процессов, которую решил с помощью shared mutex и таймера прерывания процесса. Я разработал собственные обёртки и интерфейсы для shared mutex и memory map, сделал систему сообщений, с помощью которой процессы общаются между собой, отладил эту систему.