Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Епифанов Евгений Валерьевич

Группа: М8О–212Б–22

Вариант: 8

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

# Постановка задачи

## Цель работы

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

## Задание

**Проектирование консольной клиент-серверной игры**

На основе любой из выбранных технологий:

* Pipes
* Sockets
* Сервера очередей
* И другие

Создать собственную игру более, чем для одного пользователя.

Игра может быть устроена по принципу: клиент-клиент, сервер-клиент.

**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:

* Создать игру, введя ее имя
* Присоединиться к одной из существующих игр по имени игры

**Вариант 8:**

«Быки и коровы» (угадывать необходимо числа).

Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. 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 двух основных программ: server.cpp (код сервера) и client.cpp (код клиента).

Также используется заголовочный файл:

funcs.h (заголовочный файл со специальными функциями для отправки/получения сообщений от сервера и клиента).

Makefile – для сборки программного проекта.

# Общий метод и алгоритм решения.

Правила игры: загадывается произвольное число из 5 цифр (0-9), игрок в качестве попытки пишет предполагаемое число. Если игрок угадал, то игра завершается, иначе он получает список коров и быков. Корова – угадана цифра, но не её позиция. Бык – угадана цифра и её позиция.

Устройство сервера:

Общение между сервером и клиентом происходит с помощью named pipes (fifos). При запуске сервера создается 2 канала связи: общий и для админа. После чего запускается цикл и сервер ждет получения новых команд.

Команда “login” – регистрация нового пользователя. Создается новый канал связи с конкретным юзером. Также он добавляется в вектор с активными на данный момент пользователями.

Команда “create [game name]” – создание новой игры. Формируется случайное число и создается новый поток для игры.

Команда “find” – поиск существующей игры. Из списка всех доступных игр случайным образом выбирается одна, пользователь получает на выходе название лобби с игрой.

Команда “quit” – выход. Клиент покидает сервер. Канал, связанный с данным пользователем, удаляется.

Команда “shut\_down” – отключение сервера. Может быть выполнена только пользователем с правами администратора.

Как проходит игра:

Для того, чтобы сервер мог отвечать и взаимодействовать с другими пользователями, для новой игры создается отдельный поток. В него передается название игры и загаданное число. Создается новый канал связи для данной игры. Поток ожидает команды:

Команда “connect [game name]” – подключение игрока к игре с определенным именем. При подключении игроку предлагается начать угадывать число. Если игры с таким именем нет, то будет выведено соответствующее предупреждение.

Команда “ans […]” – попытка пользователя угадать число. Запускается функция проверки данного слова. Если введенное число состоит не из 5 цифр, будет выведена ошибка. Если число полностью совпадает, то игрок победил, и игра завершается. Если в игре участвовало несколько игроков, то остальных игрокам пишется имя победителя. Если число угадано не полностью, выводится информация в соответствии с правилами игры.

Команда “leave” – выход игрока из игры.

Устройство клиента:

При запуске клиента появляется приветственное окно, где пользователь должен ввести login. (Если в качестве логина ввести ‘admin’, то это будет рассматриваться как вход с правами администратора. Разница между администратором и обычным пользователем в том, что у админа есть возможность отключить работу сервера. Поэтому для входа программа потребует ввести пароль, в отличии от обычного пользователя). После ввода логина на сервер отправляется специальная команда “login”. Когда соединение будет установлено, создается отдельный поток для получения сообщений от сервера. При отключении сервера отключается и клиент. У игрока при взаимодействии с сервером есть возможность: создать игру, найти игру для себя, подключиться к существующей игре и выйти.

# Основные файлы программы

## server.cpp

#include <iostream>

#include <vector>

#include <thread>

#include <string>

#include <fcntl.h>

#include <sys/stat.h>

#include <string.h>

#include "funcs.h"

#define CLIENT\_ID(name) in(logins,name)

#define PLAYER\_ID(name) in(curr\_playrs\_name, name)

int create\_game\_pipe(std::string game\_name)

{

int curr\_playrs;

if (mkfifo(("game\_%" + game\_name).c\_str(), 0777) == -1) // создали FIFO файл для взаимодействия процессов

{

if(errno != EEXIST){

std::cout << "ERROR: game " << ("game\_%" + game\_name).c\_str() << std::endl;

exit(1);

}

}

int game\_input\_fd = open(("game\_%" + game\_name).c\_str(), O\_RDWR);

if (game\_input\_fd == -1)

{

std::cout << "ERROR: game input fofi wasn`t opened!" << std::endl;

exit(1);

}

return game\_input\_fd;

}

// номер логина в списке

int in(std::vector<std::string> logins, std::string str)

{

for (int i = 0; i < logins.size(); ++i)

{

if (logins[i] == str)

return i;

}

return -1;

}

int create\_main\_pipe()

{

if (mkfifo("main\_input", 0777) == -1) // Создание именного канала

{

if(errno != EEXIST)

{

std::cout << "ERROR: mkfifo main\_input!" << std::endl;

exit(1);

}

}

int fd\_recv = open("main\_input", O\_RDWR);

if (fd\_recv == -1)

{

std::cout << "ERROR: main input fifo wasn`t opened!";

exit(1);

}

return fd\_recv;

}

int create\_admin\_pipe()

{

if (mkfifo("admin", 0777) == -1){

if(errno != EEXIST)

{

std::cout << "ERROR: mkfifo admin!" << std::endl;

exit(1);

}

}

int admin\_fd = open("admin", O\_RDWR);

if (admin\_fd == -1)

{

std::cout << "ERROR: admin input fifo wasn`t opened!" << std::endl;

exit(1);

}

return admin\_fd;

}

int create\_client\_pipe(std::string rcvd\_name)

{

if (mkfifo(rcvd\_name.c\_str(), 0777) == -1)

{

if(errno != EEXIST){

std::cout << "ERROR: mkfifo client!" << std::endl;

exit(1);

}

}

int fd = open(rcvd\_name.c\_str(), O\_RDWR);

if (fd == -1)

{

std::cout << "ERROR: client input fifo wasn`t opened!" << std::endl;

exit(1);

}

return fd;

}

int hit\_check(std::string game\_word, std::string try\_word, int \*cows, int \*bulls,std::vector<int>& bulls\_cows\_index)

{

std::cout << game\_word << "!" << try\_word << std::endl;

if (try\_word.size() != game\_word.size()) return -1;

if (try\_word == game\_word) return -2;

\*cows = 0;

\*bulls = 0;

for(int i = 0; i < try\_word.size(); ++i)

{

if(try\_word[i] != game\_word[i])

{

if(game\_word.find(try\_word[i]) != std::string::npos)

{

\*cows = \*cows + 1;

bulls\_cows\_index[i] = 1;

}

}

else

{

\*bulls = \*bulls + 1;

bulls\_cows\_index[i] = 2;

}

}

return 0;

}

void game\_process(std::string game\_name, std::string game\_word)

{

std::vector<std::string> curr\_playrs\_name;

std::vector<int> curr\_playrs\_fd;

auto iter\_fd = curr\_playrs\_fd.cbegin();

auto iter\_log = curr\_playrs\_name.cbegin();

int game\_input\_fd = create\_game\_pipe(game\_name);

int cows,bulls;

std::string game\_respond;

int game\_status;

std::cout << "START GAME: " << game\_name << std::endl;

std::cout.flush();

std::string rcvd\_name, rcvd\_command, rcvd\_data;

std::vector<int> bulls\_cows\_index;

std::vector<int> cows\_index;

while (1)

{

recieve\_message\_server(game\_input\_fd, &rcvd\_name, &rcvd\_command, &rcvd\_data);

if (rcvd\_command == "connect")

{

curr\_playrs\_name.push\_back(rcvd\_name);

curr\_playrs\_fd.push\_back(open(rcvd\_name.c\_str(), O\_RDWR));

std::cout << "CLIENT: " << rcvd\_name << " JOIN GAME: " << game\_name << std::endl;

std::cout.flush();

game\_respond = "|=|=| GAME " + game\_name + " STARTED ";

send\_message\_to\_client(curr\_playrs\_fd[PLAYER\_ID(rcvd\_name)], game\_respond.c\_str());

game\_respond = "|=|=| Print: |ans ...| or |leave| ";

send\_message\_to\_client(curr\_playrs\_fd[PLAYER\_ID(rcvd\_name)], game\_respond.c\_str());

}

else if (rcvd\_command == "ans")

{

bulls\_cows\_index = {0,0,0,0,0};

game\_status = hit\_check(game\_word, rcvd\_data, &cows, &bulls, bulls\_cows\_index);

if (game\_status == -1)

{

game\_respond = "Words size don't match!";

send\_message\_to\_client(curr\_playrs\_fd[PLAYER\_ID(rcvd\_name)], game\_respond.c\_str());

}

else if (game\_status == -2)

{

game\_respond = "You've Won!";

send\_message\_to\_client(curr\_playrs\_fd[PLAYER\_ID(rcvd\_name)], game\_respond.c\_str());

for (int i=0; i < curr\_playrs\_name.size(); i++)

{

game\_respond = "Winner is " + rcvd\_name + "\nAnswer is " + game\_word;

send\_message\_to\_client(curr\_playrs\_fd[i], game\_respond.c\_str());

do

{

game\_respond = "Print: |leave|";

send\_message\_to\_client(curr\_playrs\_fd[i], game\_respond.c\_str());

recieve\_message\_server(game\_input\_fd, &rcvd\_name, &rcvd\_command, &rcvd\_data);

}while(rcvd\_command != "leave");

}

close(game\_input\_fd);

std::cout << "FINISH GAME: " << game\_name << std::endl;

std::cout.flush();

int mainFD = open("main\_input", O\_RDWR);

game\_respond = "finish";

send\_message\_to\_server(mainFD, game\_name, game\_respond, "");

return;

}

else if (game\_status == 0)

{

std::string cows\_symbols = " symbols: ";

for(int elem = 0; elem < bulls\_cows\_index.size(); ++elem)

{

if(bulls\_cows\_index[elem] == 1) cows\_symbols = cows\_symbols + rcvd\_data[elem] + " ";

}

std::string bulls\_symbols = " symbols: ";

for(int elem = 0; elem < bulls\_cows\_index.size(); ++elem)

{

if(bulls\_cows\_index[elem] == 2) bulls\_symbols = bulls\_symbols + rcvd\_data[elem] + " ";

}

game\_respond = "Cows: " + std::to\_string(cows) + cows\_symbols + "\nBulls: " + std::to\_string(bulls) + bulls\_symbols;

std::cout << cows\_symbols << std::endl;

send\_message\_to\_client(curr\_playrs\_fd[PLAYER\_ID(rcvd\_name)], game\_respond.c\_str());

}

}

else if (rcvd\_command == "leave")

{

iter\_fd = curr\_playrs\_fd.cbegin();

curr\_playrs\_fd.erase(iter\_fd + PLAYER\_ID(rcvd\_name));

iter\_log = curr\_playrs\_name.cbegin();

curr\_playrs\_name.erase(iter\_log + PLAYER\_ID(rcvd\_name));

std::cout << "CLIENT: " << rcvd\_name << " LEFT GAME: " << game\_name << std::endl;

}

}

}

int main()

{

srand(time(NULL));

std::vector<std::string> logins; // Логины пользователей

std::vector<int> client\_pipe\_fd; // Pipe для каждого клиента

std::vector<std::thread> games\_threads; // Поток для игры

std::vector<std::string> games\_name; // Имена игр

std::string game\_name\_table; // Имя игры

std::string game\_word; // Загаданное слово

int fd\_recv = create\_main\_pipe();

int admin\_fd = create\_admin\_pipe();

std::string login;

std::string rcvd\_name;

std::string rcvd\_command; // Полученные в будущем данные

std::string rcvd\_data;

auto iter\_fd = client\_pipe\_fd.cbegin(); // Смотреть можно, трогать нельзя

auto iter\_log = logins.cbegin();

auto iter\_game\_thread = games\_threads.cbegin();

auto iter\_game\_name = games\_name.cbegin();

while (1)

{

recieve\_message\_server(fd\_recv, &rcvd\_name, &rcvd\_command, &rcvd\_data);

if (rcvd\_command == "login" && rcvd\_name != "admin")

{

std::cout <<"New client: " << rcvd\_name << std::endl;

std::cout.flush();

// Создать pipe для клиента и добавить в вектор

client\_pipe\_fd.push\_back(create\_client\_pipe(rcvd\_name));

// Добавить имя пользователя в вектор логинов

logins.push\_back(rcvd\_name);

}

else if (rcvd\_command == "create")

{

// Получить имя игры и загаданное слово

extract\_game\_data(rcvd\_data, &game\_name\_table);

std::string number = std::to\_string(rand()%90000+10000);

game\_word = number;

games\_name.push\_back(game\_name\_table);

games\_threads.push\_back(std::thread(game\_process, game\_name\_table, game\_word));

}

else if (rcvd\_command == "find")

{

std::vector<int> curr\_playrs\_fd\_test;

std::vector<std::string> curr\_playrs\_name;

std::vector <std::string> game\_respond\_vector;

std::string game\_respond;

curr\_playrs\_name.push\_back(rcvd\_name);

curr\_playrs\_fd\_test.push\_back(open(rcvd\_name.c\_str(), O\_RDWR));

for (size\_t i = 0; i < games\_name.size(); ++i)

{

game\_respond\_vector.push\_back(games\_name[i]);

}

game\_respond = "|=|=| Random lobby: " + game\_respond\_vector[rand() % game\_respond\_vector.size()];

send\_message\_to\_client(curr\_playrs\_fd\_test[PLAYER\_ID(rcvd\_name)], game\_respond.c\_str());

}

else if (rcvd\_command == "finish")

{

std::remove(("game\_%" + rcvd\_name).c\_str());

games\_threads[in(games\_name, rcvd\_name)].detach();

iter\_game\_thread = games\_threads.cbegin();

games\_threads.erase(iter\_game\_thread + in(games\_name, rcvd\_name));

iter\_game\_name = games\_name.cbegin();

games\_name.erase(iter\_game\_name + in(games\_name, rcvd\_name));

}

else if (rcvd\_command == "quit")

{

close(client\_pipe\_fd[CLIENT\_ID(rcvd\_name)]);

std::remove(rcvd\_name.c\_str());

iter\_fd = client\_pipe\_fd.cbegin();

iter\_log = logins.cbegin();

client\_pipe\_fd.erase(iter\_fd + CLIENT\_ID(rcvd\_name));

logins.erase(iter\_log + CLIENT\_ID(rcvd\_name));

std::cout << "CLIENT: " << rcvd\_name << " LEFT\n";

}

else if (rcvd\_command == "shutdown" && rcvd\_name == "admin")

{

for(int i=0 ; i < logins.size(); i++)

{

send\_message\_to\_client(client\_pipe\_fd[i],"SERVER CLOSED");

close(client\_pipe\_fd[i]);

std::remove(logins[i].c\_str());

}

for(int i=0 ; i < games\_threads.size(); i++)

{

std::remove(games\_name[i].c\_str());

games\_threads[i].detach();

}

close(admin\_fd);

std::remove("admin");

std::remove("main\_input");

std::cout << "SERVER OFF\n";

return 0;

}

else{

std::cout << "Unknown command!" << std::endl;

}

}

}

## CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(lab2)

set(CMAKE\_CXX\_STANDARD 20)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

add\_executable(client client.cpp)

add\_executable(server server.cpp)

## funcs.h

#include <string>

#include <unistd.h>

//отправить сообщение серверу в удобной форме - логин$команда$сообщение

void send\_message\_to\_server(int fd, std::string curlogin, std::string command, std::string data)

{

std::string text = curlogin + "$" + command + "$" + data;

int k = text.size();

write(fd, &k, sizeof(k)); // размер отправляемого сообщения

char message[k];

for (int i = 0; i < k; ++i) message[i] = text[i];

write(fd, message, k); // куда что сколько

}

//отправить сообщение клиенту

void send\_message\_to\_client(int fd, std::string message)

{

std::string text = message;

int k = text.size();

write(fd, &k, sizeof(k));

char message\_c[k];

for (int i = 0; i < k; ++i) message\_c[i] = text[i];

write(fd, message\_c, k);

}

//получить сообщение в удобной для клиента форме

std::string recieve\_message\_client(int fd)

{

int size;

read(fd, &size, sizeof(size)); // откуда куда сколько

char message[size];

read(fd, message, size);

std::string recv;

for (int i = 0; i < size; ++i)

{

if (message[i] != '$')

recv.push\_back(message[i]);

else

recv = recv + ": ";

}

return recv;

}

//получить логин из сообщения для сервера

std::string extract\_login(std::string message)

{

std::string login;

int i = 0;

while (message[i] != '$')

{

login.push\_back(message[i]);

++i;

}

return login;

}

//получить команду из сообщения для сервера

std::string extract\_command(std::string message)

{

std::string command;

int i = 0;

while (message[i] != '$') ++i;

++i;

while (message[i] != '$')

{

command.push\_back(message[i]);

++i;

}

return command;

}

//получить информацию из сообщения для сервера

std::string extract\_data(std::string message)

{

std::string data;

int i = 0;

while (message[i] != '$') ++i;

++i;

while (message[i] != '$') ++i;

++i;

while (i < message.size())

{

data.push\_back(message[i]);

++i;

}

return data;

}

// получить информацию из сообщения для сервера

void recieve\_message\_server(int fd, std::string \*rcvd\_name, std::string \*rcvd\_command, std::string \*rcvd\_data)

{

int size;

read(fd, &size, sizeof(size));

char messagec[size];

read(fd, messagec, size);

std::string recv;

for (int i = 0; i < size; ++i) recv.push\_back(messagec[i]);

\*rcvd\_name = extract\_login(recv);

\*rcvd\_command = extract\_command(recv);

\*rcvd\_data = extract\_data(recv);

}

// Получить имя игры

void extract\_game\_data(std::string message, std::string \*game\_name)

{

int i = 0;

std::string recv1;

while (message[i] != '$')

{

recv1.push\_back(message[i]);

++i;

}

\*game\_name = recv1;

}

## client.cpp

#include <iostream>

#include <thread>

#include <vector>

#include <fcntl.h>

#include "funcs.h"

//функция приёма сообщений (для потока)

void func(int fd\_respond, std::string login)

{

while (1)

{

std::string respond = recieve\_message\_client(fd\_respond);

std::cout << std::endl << respond << std::endl;

std::cout.flush();

if (respond == "SERVER CLOSED")

exit(0);

std::cout << login << "> ";

std::cout.flush();

}

}

void write\_intro()

{

std::cout << "|=|=| The Game [Bulls & Cows] |=|=|" << std::endl;

std::cout << std::endl;

std::cout << "Please enter your login: ";

std::cout.flush();

}

void write\_menu(std::string login)

{

std::cout << std::endl;

std::cout << "|=|=|=|=|=| GAME MENU |=|=|=|=|=|" << std::endl;

std::cout << "| Select an action: |" << std::endl;

std::cout << "| 1. create [name of the game] |" << std::endl;

std::cout << "| 2. connect [name of the game] |" << std::endl;

std::cout << "| 3. find |" << std::endl;

std::cout << "| 4. leave |" << std::endl;

if (login != "admin") std::cout << "| 5. quit |" << std::endl;

if (login == "admin") std::cout << "| 5. shutdown [!] |" << std::endl;

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

std::cout << std::endl;

std::cout.flush();

}

int server\_main\_input\_fd() // Для общения через server pipe

{

int fd = open("main\_input", O\_RDWR);

if (fd == -1)

{

std::cout << "ERROR: main fifo wasn`t opened!" << std::endl;

exit(1);

}

return fd;

}

int main()

{

int client\_main\_out\_fd = server\_main\_input\_fd();

write\_intro();

std::string login;

std::cin >> login;

std::string password;

if(login == "admin"){

std::cout << "Please enter the password: ";

std::cin >> password;

std::cout << std::endl;

if(password != "admin"){

std::cout << "ERROR: wrong password!" << std::endl;

exit(1);

}

}

send\_message\_to\_server(client\_main\_out\_fd, login, "login", "");

std::cout << "Connection..." << std::endl;

sleep(1);

int fd\_respond = open(login.c\_str(), O\_RDWR);

if (fd\_respond == -1)

{

std::cout << "ERROR: respond fifo wasn`t opened!" << std::endl;

exit(1);

}

std::cout << "Connected succesfully!" << std::endl;

write\_menu(login);

std::thread thr\_respond(func, fd\_respond, login);

std::string command, data;

std::string game\_word, game\_name;

int game\_fd;

while (1)

{

std::cout << login << "> ";

std::cin >> command;

if (command == "create")

{

std::cin >> game\_name;

data = game\_name + "$";

send\_message\_to\_server(client\_main\_out\_fd,login,command,data);

std::cout << "|=|=| Print: |connect " << game\_name << "| to open the room." << std::endl;

}

else if (command == "find")

{

data = "";

send\_message\_to\_server(client\_main\_out\_fd,login,command,data);

}

else if (command == "connect")

{

std::cin >> game\_name;

game\_fd = open(("game\_%" + game\_name).c\_str(), O\_RDWR);

if (game\_fd == -1)

{

std::cout << "ERROR: game not found!" << std::endl;

std::cout.flush();

}

else

{

data = "";

send\_message\_to\_server(game\_fd,login,command,data);

std::cout << login << "> ";

std::cout.flush();

while (1)

{

std::cin >> command;

if (command == "ans")

{

std::cin >> data;

send\_message\_to\_server(game\_fd,login,command,data);

}

else if (command == "leave")

{

data = "";

send\_message\_to\_server(game\_fd,login,command,data);

write\_menu(login);

break;

}

else

{

std::cout << login << "> ";

std::cout.flush();

}

}

}

}

else if (command == "quit" && login != "admin")

{

data = "";

send\_message\_to\_server(client\_main\_out\_fd,login,command,data);

thr\_respond.detach();

return 0;

}

else if (command == "shutdown" && login == "admin")

{

data = "";

send\_message\_to\_server(client\_main\_out\_fd,login,command,data);

thr\_respond.detach(); // Отсоединение связанного потока

return 0;

}

}

return 0;

}

# Пример работы

1. Игрок test1 играет в игру сам с собой. Игрок создает и находит игру.

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

1. 2 игрока test1 и test2 играют в одном лобби.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

1. Админ выключил сервер.





# Вывод

В результате курсового проекта я написал собственную игру “Быки и коровы” для одиночной и многопользовательской игры. Для этого потребовалось написать 2 программы: сервер и клиент, реализующие функционал в соответствии с заданным вариантом. Для того, чтобы организовать общение между клиентом и сервером, я изучил один из методов межпроцессорного взаимодействия – named pipe.