**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные Системы»

**Курсовой проект по курсу**

**«Операционные системы»**

Консоль-серверная игра «Быки и коровы»

Студент: Семин Александр Витальевич

Группа: М8О-206Б-20

Преподаватель: Соколов Андрей Алексеевич

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

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 7:

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

1. **Набор тестов**

*Тест 1.*

Сначала запускается сервер:

progger@asus:~/Desktop/OS\_labs/kp\_v2$ ./server

Затем запустим двух клиентов и создадим в них одиночные игры:

progger@asus:~/Desktop/OS\_labs/kp\_v2$ ./client

Client started!

> create 1

game1# player

Player ID 0. Game ID: 1. Game: game1

game1# a

face

Bulls: 1, Cows: 0

game1# a

fork

Bulls: 4, Cows: 0

Congratulations!

You win!

game1 winner# exit

> list

No games running

> q

Leaving the client...

progger@asus:~/Desktop/OS\_labs/kp\_v2$ ./client

Client started!

Can't open pipe for writing: No such file or directory

progger@asus:~/Desktop/OS\_labs/kp\_v2$ ./client

Client started!

> create 1

game0# player

Player ID 1. Game ID: 0. Game: game0

game0# list

game0[1\1] game1[1\1]

game0# exit

> q

Leaving the client...

Таким станет вывод в сервере:

## Pipes created

CLIENT-PIPES:

/tmp/bulls\_and\_cows\_sw1

/tmp/bulls\_and\_cows\_sr1

Player was added successfully!

Player-ID: 0

## Thread started successfully!

## Thread: 0

## Pipes created

CLIENT-PIPES:

/tmp/bulls\_and\_cows\_sw2

/tmp/bulls\_and\_cows\_sr2

Player was added successfully!

Player-ID: 1

## Thread started successfully!

## Thread: 1

$REQUEST: create the game

-------------------------

$REQUEST: create the game

-------------------------

$REQUEST: print reply

---------------------

--------Reply-------

Total games: 2

game0[1\1] | 'neck' Active player's ID: 0

game1[1\1] | 'fork' Active player's ID: 0

=====End of reply====

$REQUEST: print reply

---------------------

--------Reply-------

Total games: 2

game0[1\1] | 'neck' Active player's ID: 0

game1[1\1] | 'fork' Active player's ID: 0

=====End of reply====

$REQUEST: check user's answer

-----------------------------

GAME = game1 [1\1].

WORD: ''face''

Hidden word: 'fork'

BULLS: 1

COWS: 1

Active player's ID 0

$REQUEST: check user's answer

-----------------------------

GAME = game1 [1\1].

WORD: ''fork''

Hidden word: 'fork'

BULLS: 4

COWS: 4

Active player's ID 0

$REQUEST: list of games

-----------------------

Games count 2

# Active games: game0[1\1] game1[1\1]

$REQUEST: leave the game

------------------------

0.Game-name: game0(game0) PC: 0

--------Reply-------

Total games: 2

game0[0\0] | 'neck' Active player's ID: 0

game1[1\1] | 'fork' | completed | Active player's ID: 0

=====End of reply====

--------Reply-------

Total games: 1

game1[1\1] | 'fork' | completed | Active player's ID: 0

=====End of reply====

$REQUEST: leave the game

------------------------

1.Game-name: game1(game1) PC: 0

--------Reply-------

Total games: 1

game1[0\0] | 'fork' | completed | Active player's ID: 0

=====End of reply====

--------Reply-------

Total games: 0

=====End of reply====

$REQUEST: list of games

-----------------------

Games count 0

# Active games: No games running

!GAME OVER!

!GAME OVER!

*Тест 2.*

Пример игры для нескольких людей:

progger@asus:~/Desktop/OS\_labs/kp\_v2$ ./server

progger@asus:~/Desktop/OS\_labs/kp\_v2$ ./client (далее клиент1)

Client started!

> create 2

Waiting for players…

progger@asus:~/Desktop/OS\_labs/kp\_v2$ ./client (далее клиент2)

Client started!

> join game0

Joining to the game0 2 2

game0 2 2

Waiting for your turn...

Клиент1:

game0# list

game0[2\2]

game0# a

leaf

Bulls: 1, Cows: 0

Waiting for your turn...

Клиент2:

game0# a

road

Bulls: 4, Cows: 0

Congratulations!

You win!

Клиент1:

You loosed! Good luck next time

game0 loser# exit

> q

Leaving the client...

Клиент2:

game0 winner# list

game0[2\1]

game0 winner# exit

> list

No games running

> q

Leaving the client...

Вывод сервера:

## Pipes created

CLIENT-PIPES:

/tmp/bulls\_and\_cows\_sw1

/tmp/bulls\_and\_cows\_sr1

Player was added successfully!

Player-ID: 0

## Thread started successfully!

## Thread: 0

$REQUEST: create the game

-------------------------

## Pipes created

CLIENT-PIPES:

/tmp/bulls\_and\_cows\_sw2

/tmp/bulls\_and\_cows\_sr2

Player was added successfully!

Player-ID: 1

## Thread started successfully!

## Thread: 1

$REQUEST: join to the game

----------------------

$REQUEST: list of games

-----------------------

Games count 1

# Active games: game0[2\2]

$REQUEST: check user's answer

-----------------------------

GAME = game0 [2\2].

WORD: ''leaf''

Hidden word: 'road'

BULLS: 1

COWS: 1

Active player's ID 1

$REQUEST: check user's answer

-----------------------------

GAME = game0 [2\2].

WORD: ''road''

Hidden word: 'road'

BULLS: 4

COWS: 4

Active player's ID 0

$REQUEST: leave the game

------------------------

!GAME OVER!

$REQUEST: list of games

-----------------------

Games count 1

# Active games: game0[2\1]

$REQUEST: leave the game

------------------------

0.Game-name: game0(game0) PC: 0

--------Reply-------

Total games: 1

game0[0\0] | 'road' | completed | Active player's ID: 1

=====End of reply====

--------Reply-------

Total games: 0

=====End of reply====

$REQUEST: list of games

-----------------------

Games count 0

# Active games: No games running

!GAME OVER!

!GAME OVER!

1. **Листинг программы**

**Message.h**

#ifndef \_\_MESSAGE\_H\_

#define \_\_MESSAGE\_H\_

#define MAX\_REPLY\_SIZE 1024

#define MAX\_REQUEST\_SIZE 1024

void read\_str(int fd, char\* str, int max\_size);

int write\_msg(int fd, char\* buf, int size);

#endif /\*\_\_MESSAGE\_H\_\*/

**Message.c**

#include "Message.h"

#include <unistd.h>

#include <stdio.h>

void read\_str(int fd, char\* str, int max\_size){

char symb;

int len;

int i = 0;

while((len = read(fd, &symb, 1)) >= 0 && i < (max\_size - 1)){

if(len == 0)

continue;

if(symb == '\n')

break;

str[i++] = symb;

}

str[i] = 0;

}

int write\_msg(int fd, char\* buf, int size){

int write\_rvl;

int written = 0;

do{

write\_rvl = write(fd, buf + written, size - written);

if(write\_rvl < 0){

perror("Write ERROR!");

return 0;

}

written += write\_rvl;

} while(written < size);

return 1;

}

**game.h**

#ifndef GAME\_H\_

#define GAME\_H\_

#include <time.h>

#include <stdbool.h>

#include <ctype.h>

#include <pthread.h>

#define GAME\_NAME 32

#define WIN\_BULLS 4

typedef struct{

int fd\_r;

int fd\_w;

int user\_id;

pthread\_t t\_id;

} pl\_st;

typedef struct{

int win\_id;

char name[GAME\_NAME];

char\* hidden\_word;

int max\_players;

int pl\_number;

int active\_pl\_id;

pl\_st \*players[1];

} game\_st;

game\_st\* new\_game(char \*name, int max\_players, pl\_st \*first\_player);

void bulls\_and\_cows(game\_st\* g, char\* user\_word, int \*bulls, int \*cows);

#endif

**game.c**

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#include "game.h"

static char\* get\_rand\_word() {

srand(time(NULL));

int i = rand()%21;

char\* str = malloc(sizeof(char)\*4);

if (i == 0) str = "bear";

if (i == 1) str = "vibe";

if (i == 2) str = "neck";

if (i == 3) str = "rose";

if (i == 4) str = "bike";

if (i == 5) str = "road";

if (i == 6) str = "year";

if (i == 7) str = "wine";

if (i == 8) str = "fork";

if (i == 9) str = "page";

if (i == 10) str = "sign";

if (i == 11) str = "leaf";

if (i == 12) str = "wind";

if (i == 13) str = "home";

if (i == 14) str = "head";

if (i == 15) str = "hole";

if (i == 16) str = "camp";

if (i == 17) str = "lamp";

if (i == 18) str = "plan";

if (i == 19) str = "face";

if (i == 20) str = "cave";

return str;

}

game\_st\* new\_game(char\* name, int max\_players, pl\_st\* first\_player){

static int game\_number = 0;

game\_st \*g = malloc(sizeof(game\_st) + sizeof(pl\_st \*) \* (max\_players - 1));

if (g == NULL) {

perror("Error: malloc\n");

return NULL;

}

if(name == NULL || strcmp(name, "") == 0)

sprintf(g->name, "game%d", game\_number++);

else

strcpy(g->name, name);

if (max\_players == 1)

g->active\_pl\_id = 0;

else

g->active\_pl\_id = -1;

g->max\_players = max\_players;

g->players[0] = first\_player;

g->pl\_number = 1;

g->win\_id = -1;

g->hidden\_word = get\_rand\_word();

return g;

}

void bulls\_and\_cows(game\_st\* g, char\* user\_word, int\* bulls, int\* cows) {

char bukva;

int bll = 0;

int cw = 0;

if (g == NULL) {

printf("\tHidden word: '%s'\n\tBULLS: %d\n\tCOWS: %d\n", g->hidden\_word, bll, cw);

if(bulls != NULL) \*bulls = bll;

if(cows != NULL) \*cows = cw - bll;

}

for (int i = 0; i < WIN\_BULLS; i++) {

bukva = user\_word[i];

if (bukva == g->hidden\_word[i]) bll++;

for (int j = 0; j < WIN\_BULLS; j++) {

if (g->hidden\_word[j] == bukva) {

cw++;

break;

}

}

}

printf("\tHidden word: '%s'\n\tBULLS: %d\n\tCOWS: %d\n", g->hidden\_word, bll, cw);

if(bulls != NULL) \*bulls = bll;

if(cows != NULL) \*cows = cw - bll;

}

**client.c**

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <string.h>

#include <stdbool.h>

#include "Message.h"

#include "game.h"

#define MAX\_CMD\_SIZE 64

#define GAME\_NAME 32

#define MAX\_PATH 128

char \*skip\_separator(char \*str){

if(str == NULL)

return NULL;

while(\*str != 0){

if(\*str == ' ' || \*str == '\t'){

str++;

}

else

break;

}

return str;

}

char \*search\_separator(char \*str){

if(str == NULL)

return NULL;

while(\*str != 0){

if(\*str != ' ' && \*str != '\t'){

str++;

}

else

break;

}

return str;

}

char wait\_request(pl\_st \*p, int \*n){

int num = -1;

char rep[32];

write\_msg(p->fd\_w, "w\n", 2);

read\_str(p->fd\_r, rep, 32);

switch(rep[0]){

case 'p':

case 't':

case 'r':

case 'w':

case 'l':

num = atoi(rep + 1);

break;

default:

rep[0] = 'r';

break;

}

if (rep[0] != 'l') {

num = atoi(rep+1);

}

if (n != NULL)

\*n = num;

return rep[0];

}

void erase(int n){

while(n-- > 0){

printf("\b \b");

}

fflush(stdout);

}

int wait\_for\_turn(game\_st \* game){

pl\_st \* p = game->players[0];

int num = -1;

int len = 0;

char w = ' ';

do{

sleep(1);

w = wait\_request(p, &num);

erase(len);

if(w == 'p')

len = printf("%s", "Waiting for players...");

else if(w == 't')

len = printf("%s", "Waiting for your turn...");

fflush(stdout);

} while(w != 'r' && w != 'l' && w != 'w');

erase(len);

if(w == 'l' || w == 'w'){

w == 'l' ? printf("You loosed! Good luck next time\n") : printf("You win!\n");

game->win\_id = w == 'w' ? 0 : 1;

}

return num;

}

static void put\_begin(game\_st \*g, pl\_st\* player){

bool pl\_exist = false;

if (g != NULL) {

for (int i = 0; i < g->max\_players; i++) {

if (g->players[i] == player) {

pl\_exist = true;

break;

}

}

}

if(!pl\_exist || g == NULL){

printf("> ");

fflush(stdout);

return;

}

if(g->win\_id < 0){

int n = wait\_for\_turn(g);

g->max\_players = n;

g->pl\_number = n;

}

printf("%s%s%c ", g->name, g->win\_id < 0 ? "" : g->win\_id == 0 ? " winner" : " loser", g->active\_pl\_id >= 0 ? '#' : '>');

fflush(stdout);

}

static game\_st\* CreateGame(pl\_st \*player, char \*name, int max\_players){

int num;

char msg[MAX\_REQUEST\_SIZE];

char rep[MAX\_REPLY\_SIZE];

int len = snprintf(msg, MAX\_REQUEST\_SIZE, "c%d\*%s\n", max\_players, name == NULL ? "" : name);

write\_msg(player->fd\_w, msg, len);

read\_str(player->fd\_r, rep, MAX\_REPLY\_SIZE);

if(\*rep == '!' || \*rep == 0)

return NULL;

len = 0;

game\_st \*game = new\_game(rep, max\_players, player);

// printf("HW: %s\n", game->hidden\_word);

num = wait\_for\_turn(game);

game->active\_pl\_id = 0;

game->pl\_number = num;

game->max\_players = max\_players;

return game;

}

static game\_st\* JoinGame(pl\_st \*player, char \*name){

char server\_game\_name[GAME\_NAME];

char msg[MAX\_REQUEST\_SIZE];

char rep[MAX\_REPLY\_SIZE];

int server\_max\_players = -1;

int server\_active\_pl\_ids = -1;

int len = snprintf(msg, MAX\_REQUEST\_SIZE, "j%s\n", name == NULL ? "" : name);

write\_msg(player->fd\_w, msg, len);

read\_str(player->fd\_r, rep, MAX\_REPLY\_SIZE);

printf("Joining to the %s...\n", rep);

if(\*rep == '!')

return NULL;

sscanf(rep, "%s %d %d", server\_game\_name, &server\_max\_players, &server\_active\_pl\_ids);

if(\*server\_game\_name == 0)

return NULL;

// printf("%s %d %d\n", server\_game\_name, server\_max\_players, server\_active\_pl\_ids);

if(server\_game\_name[0] == 0 || server\_max\_players < 1 || server\_active\_pl\_ids < 1)

return NULL;

game\_st\* game = new\_game(server\_game\_name, server\_max\_players, player);

int num = wait\_for\_turn(game);

game->pl\_number = num;

game->max\_players = num;

return game;

}

void process\_cmd(int fd\_r, int fd\_w){

pl\_st player;

player.fd\_r = fd\_r;

player.fd\_w = fd\_w;

game\_st \*game = NULL;

char cmd[MAX\_CMD\_SIZE];

char rep[MAX\_REPLY\_SIZE];

char req[MAX\_REQUEST\_SIZE];

for(;;){

put\_begin(game, &player);

read\_str(0, cmd, MAX\_CMD\_SIZE);

if (strcmp(cmd,"exit") == 0 ) {

write\_msg(fd\_w, "e\n", 2);

read\_str(fd\_r, rep, MAX\_REPLY\_SIZE);

if(\*rep == '!') {

printf("Wrong command\n");

continue;

}

game = NULL;

continue;

} else

if(strcmp(cmd, "ping") == 0){

write\_msg(fd\_w, "ping\n", 5);

read\_str(fd\_r, rep, MAX\_REPLY\_SIZE);

printf("%s\n", rep);

continue;

} else

if(strncmp(cmd, "create ", 7) == 0){

char \*name = NULL;

int max\_players = 1;

if(game != NULL) {

printf("You can't create new game now!\n");

continue;

}

char \*p = cmd + 7;

p = skip\_separator(p);

if(\*p != 0){

char \*new\_p = search\_separator(p);

max\_players = atoi(p);

if(\*new\_p != 0){

p = skip\_separator(new\_p);

if(\*p != 0){

name = p;

p = search\_separator(p);

\*p = 0;

}

}

}

if(max\_players <= 0)

max\_players = 1;

if(name != NULL && \*name == 'g'){

printf("Wrong command\n");

continue;

}

game = CreateGame(&player, name, max\_players);

continue;

} else

if(strncmp(cmd, "join", 4) == 0){

if(game != NULL) {

printf("You can't join other game while you are in active game\n");

continue;

}

char \*name = NULL;

char \*p = cmd + 4;

p = skip\_separator(p);

if(\*p != 0){

char \*new\_p = search\_separator(p);

\*new\_p = 0;

name = p;

}

game = JoinGame(&player, name);

if (game->pl\_number >= game->max\_players) game->active\_pl\_id = 0;

continue;

} else

if(cmd[0] == 'a') {

char\* w = malloc(sizeof(char)\*5);

read\_str(0, w, sizeof(char)\*5);

write\_msg(fd\_w, "a", 1);

write\_msg(fd\_w, w, strlen(w));

write\_msg(fd\_w, "\n", 1);

read\_str(fd\_r, rep, MAX\_REPLY\_SIZE);

if(\*rep == '!'){

printf("Wrong command\n");

continue;

}

printf("Bulls: %c, Cows: %c%s\n", rep[0], rep[1], rep[0] == (WIN\_BULLS + '0') ? "\nCongratulations!\n" : "");

free(w);

continue;

}else

if(strcmp(cmd, "list") == 0) {

write\_msg(fd\_w, "l\n", 2);

read\_str(fd\_r, rep, MAX\_REPLY\_SIZE);

printf("%s\n", rep);

continue;

} else

if(strcmp(cmd, "player") == 0) {

write\_msg(fd\_w, "p\n", 2);

read\_str(fd\_r, rep, MAX\_REPLY\_SIZE);

printf("%s\n", rep);

continue;

} else

if (cmd[0] == 'q') {

if(game != NULL){

printf("To leave your current game write the command <exit>\n");

continue;

}

write\_msg(fd\_w, "q\n", 5);

printf("Leaving the client...\n");

return;

} else {

printf("Wrong command\n");

}

}

}

int main () {

int fd\_r = -1;

int fd\_w = -1;

printf("Client started!\n");

if((fd\_r = open("/tmp/bulls\_and\_cows\_sw0", O\_RDONLY)) < 0){

perror("Can't open pipe for reading");

if(fd\_r >= 0)

close(fd\_r);

if(fd\_w >= 0)

close(fd\_w);

return 0;

}

char pl\_r[MAX\_PATH];

char pl\_w[MAX\_PATH];

read\_str(fd\_r, pl\_w, MAX\_PATH);

read\_str(fd\_r, pl\_r, MAX\_PATH);

close(fd\_r);

// printf("%s %s\n", pl\_r, pl\_w);

if((fd\_w = open(pl\_w, O\_WRONLY)) < 0){

perror("Can't open pipe for writing");

if(fd\_r >= 0)

close(fd\_r);

if(fd\_w >= 0)

close(fd\_w);

return 0;

}

if((fd\_r = open(pl\_r, O\_RDONLY)) < 0){

perror("Can't open pipe for reading");

if(fd\_r >= 0)

close(fd\_r);

if(fd\_w >= 0)

close(fd\_w);

return 0;

}

process\_cmd(fd\_r, fd\_w);

if(fd\_r >= 0)

close(fd\_r);

if(fd\_w >= 0)

close(fd\_w);

return 0;

}

**server.c**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h> //mknod

#include <semaphore.h>

#include <pthread.h>

#include "game.h"

#include "Message.h"

#define PLAYERS\_NUMBER 32

#define GAMES\_NUMBER 10

#define GAME\_NAME 32

#define MAX\_PATH 128

typedef struct{

char path\_sr[MAX\_PATH];

char path\_sw[MAX\_PATH];

} pipes\_st;

static sem\_t phore;

static game\_st \*games[GAMES\_NUMBER];

static pl\_st \*players[PLAYERS\_NUMBER];

static int pl\_number = 0;

static int games\_count = 0;

void lock(){

sem\_wait(&phore);

}

void unlock(){

sem\_post(&phore);

}

void add\_to\_str(char\* buf, int x){

char num[24];

int i = 0;

if (x < 0){

num[i++] = '-';

x = -x;

}

do{

num[i++] = x % 10 + '0';

x /= 10;

} while (x > 0);

while(--i >= 0)

\*buf++ = num[i];

\*buf = 0;

}

void pipes\_st\_init(pipes\_st \*pl, int num){

strcpy(pl->path\_sr, "/tmp/bulls\_and\_cows\_sr");

add\_to\_str(pl->path\_sr + 22, num);

strcpy(pl->path\_sw, "/tmp/bulls\_and\_cows\_sw");

add\_to\_str(pl->path\_sw + 22, num);

}

static int list\_reply(char\* rep){

int len = 0;

int i = 0;

game\_st \*\*g = games;

int l;

lock();

int gc = games\_count;

printf("Games count %d\n", gc);

if(gc == 0){

len = sprintf(rep, "No games running\n");

unlock();

return len;

}

do{

if(i++ >= GAMES\_NUMBER)

break;

if(\*g == NULL){

g++;

continue;

}

l = sprintf(rep, "%s[%d\\%d]\t", (\*g)->name, (\*g)->max\_players, (\*g)->pl\_number);

g++;

rep += l;

len += l;

} while(--gc > 0);

unlock();

\*(--rep) = '\n';

return len;

}

static int print\_reply(char\* rep){

game\_st\*\* g = games;

lock();

int gc = 0;

int i = 0;

printf("--------Games info-------\n");

printf("Total games: %d\n", games\_count);

do{

if(i++ >= GAMES\_NUMBER)

break;

if(\*g == NULL){

g++;

continue;

}

printf("%s[%d\\%d] | '%s' %s Active player's ID: %d\n", (\*g)->name, (\*g)->max\_players, (\*g)->pl\_number,

(\*g)->hidden\_word, (\*g)->win\_id < 0 ? "" : "| completed | ", (\*g)->active\_pl\_id);

g++;

gc++;

} while(true);

printf("========== End ==========\n\n");

unlock();

\*rep = 0;

return 0;

}

int new\_game\_serv(char \*name, int max\_players, pl\_st \*first\_player){

game\_st \*g = NULL;

static int ind = 0;

int rvl = -1;

int i;

g = new\_game(name, max\_players, first\_player);

// printf("HW: %s\n", g->hidden\_word);

if(g == NULL)

return -1;

lock();

for(i = 0; i < GAMES\_NUMBER; i++){

if(games[(i + ind) % GAMES\_NUMBER] == NULL){

rvl = (i + ind) % GAMES\_NUMBER;

games[rvl] = g;

games\_count++;

ind = (i + ind + 1) % GAMES\_NUMBER;

break;

}

}

unlock();

return rvl;

}

void remove\_game(int ind){

if(ind >= GAMES\_NUMBER || ind < 0)

return;

if(games[ind] != NULL){

games[ind] = NULL;

games\_count--;

}

}

int add\_player(pl\_st \*p) {

static int ind = 0;

int rvl = -1;

int i;

if(p == NULL)

return -1;

lock();

for(i = 0; i < PLAYERS\_NUMBER; i++) {

if(players[(i + ind) % PLAYERS\_NUMBER] == NULL) {

rvl = (i + ind) % PLAYERS\_NUMBER;

players[rvl] = p;

pl\_number++;

ind = (i + ind + 1) % PLAYERS\_NUMBER;

break;

}

}

printf("Player was added successfully!\n");

unlock();

p->user\_id = rvl;

return rvl;

}

void remove\_player(int ind){

if(ind >= PLAYERS\_NUMBER || ind < 0)

return;

lock();

if(players[ind] != NULL){

players[ind] = NULL;

pl\_number--;

}

unlock();

}

static void\* client\_thread(void \*arg){

int game\_ind = -1;

game\_st\* game = NULL;

pl\_st\* player = (pl\_st\*)arg;

int my\_ind = player->user\_id;

char req[MAX\_REQUEST\_SIZE];

char rep[MAX\_REPLY\_SIZE];

int fd\_r = player->fd\_r;

int fd\_w = player->fd\_w;

printf("## Thread: %d\n\n", my\_ind);

for (;;) {

read\_str(fd\_r, req, MAX\_REQUEST\_SIZE);

if (\*req == 'q') {

printf("Game over\n");

remove\_player(my\_ind);

free(player);

return NULL;

}

if (strcmp(req, "ping") == 0) {

printf("$REQUEST: ping the server\n");

printf("-------------------------\n");

write\_msg(fd\_w, "pong\n", 5);

continue;

}

if (\*req == 'c') {

printf("$REQUEST: create the game\n");

printf("-------------------------\n");

if (game != NULL) {

write\_msg(fd\_w, "!\n", 2);

continue;

}

char name[GAME\_NAME];

name[0] = 0;

int max\_players = -1;

sscanf(req + 1, "%d\*%s", &max\_players, name);

if(max\_players <= 0) {

write\_msg(fd\_w, "!\n", 2);

continue;

}

game\_ind = new\_game\_serv(name, max\_players, player);

if(game\_ind == -1) {

write\_msg(fd\_w, "!\n", 2);

continue;

}

game = games[game\_ind];

// printf("HW2: %s\n", game->hidden\_word);

write\_msg(fd\_w, game->name, strlen(game->name));

write\_msg(fd\_w, "\n", 1);

continue;

}

if (\*req == 'a') {

printf("$REQUEST: check user's answer\n");

printf("-----------------------------\n");

int bulls = 0;

int cows = 0;

char\* word = malloc(sizeof(char)\*4);

for (int i = 0; i < 4; i++) {

word[i] = req[i+1];

}

printf("GAME = %s [%d\\%d].\nWORD: ''%s''\n", game->name, game->max\_players, game->pl\_number, word);

if (game == NULL || game->active\_pl\_id < 0) {

printf("# NOT ACTIVE!\n");

write\_msg(fd\_w, "!\n", 2);

continue;

}

if (game->players[game->active\_pl\_id] != player) {

printf("NOT ME %s %p %p\n", game->name, game->players[game->active\_pl\_id], player);

write\_msg(fd\_w, "!\n", 2);

continue;

}

bulls\_and\_cows(game, word, &bulls, &cows);

int len = sprintf(rep, "%d%d\n", bulls, cows);

if (game->max\_players > 1) {

do {

game->active\_pl\_id = (game->active\_pl\_id + 1) % game->max\_players;

} while (game->players[game->active\_pl\_id] == NULL);

}

printf("Active player's ID %d\n\n", game->active\_pl\_id);

write\_msg(fd\_w, rep, len);

if (bulls >= WIN\_BULLS){

game->win\_id = my\_ind;

}

continue;

}

if (\*req == 'l'){

printf("$REQUEST: list of games\n");

printf("-----------------------\n");

int list\_len = list\_reply(rep);

printf("# Active games: %s\n", rep);

write\_msg(fd\_w, rep, list\_len);

continue;

}

if (\*req == 'p'){

printf("$REQUEST: print reply\n");

printf("---------------------\n\n");

int print\_len = print\_reply(rep);

print\_len += sprintf(rep + print\_len, "\rPlayer ID %d. Game ID: %d. Game: %s\n", my\_ind, game\_ind, game ? game->name : "NULL");

write\_msg(fd\_w, rep, print\_len);

continue;

}

if (\*req == 'j'){

printf("$REQUEST: join to the game\n");

printf("----------------------\n");

char \*p = req + 1;

if (game != NULL) {

write\_msg(fd\_w, "!\n", 2);

continue;

}

lock();

for (int i = 0; i < GAMES\_NUMBER; i++){

if (games[i] == NULL)

continue;

if (games[i]->active\_pl\_id >= 0)

continue;

if (\*p == 0 || strcmp(p, games[i]->name) == 0) {

games[i]->players[games[i]->pl\_number++] = player;

game = games[i];

game\_ind = i;

// printf("HW3: %s\n", game->hidden\_word);

if (game->pl\_number >= game->max\_players) game->active\_pl\_id = 0;

break;

}

}

unlock();

if(game == NULL) {

write\_msg(fd\_w, "!\n", 2);

continue;

}

int len = sprintf(rep, "%s %d %d\n", game->name, game->max\_players, game->pl\_number);

write\_msg(fd\_w, rep, len);

continue;

}

if(\*req == 'w'){

int len;

if(game->active\_pl\_id >= 0) {

if(game->win\_id >= 0){

if(game->win\_id == my\_ind)

len = sprintf(rep, "w%d\n", game->max\_players);

else

len = sprintf(rep, "l%d\n", game->max\_players);

}

else

if(game->players[game->active\_pl\_id] == player){

len = sprintf(rep, "r%d\n", game->max\_players);

}

else

len = sprintf(rep, "t%d\n", game->max\_players);

}

else

len = sprintf(rep, "p%d\n", game->pl\_number);

write\_msg(fd\_w, rep, len);

continue;

}

if(\*req == 'e'){

printf("$REQUEST: leave the game\n");

printf("------------------------\n");

if(game == NULL) {

write\_msg(fd\_w, "!\n", 2);

continue;

}

lock();

if (game->pl\_number == 1) {

for(int i = 0; i < game->max\_players; i++){

game->players[i] = NULL;

}

game->max\_players = 0;

game->pl\_number--;

} else {

for(int i = 0; i < game->max\_players; i++){

if(game->players[i] == player){

game->players[i] = NULL;

game->pl\_number--;

}

}

}

if (game->max\_players >= 1) {

do {

game->active\_pl\_id = (game->active\_pl\_id + 1) % game->max\_players;

} while (game->players[game->active\_pl\_id] == NULL);

}

if(game->pl\_number <= 0){

if (game\_ind < 0 || game\_ind >= GAMES\_NUMBER)

printf("Game ID = %d\n", game\_ind);

else if(games[game\_ind] == NULL)

printf("Null ID\n");

else

printf("%d.Game: %s. Players: %d\n", game\_ind, game->name, game->pl\_number);

remove\_game(game\_ind);

unlock(); print\_reply(rep); lock();

free(game);

}

game\_ind = -1;

game = NULL;

unlock();

write\_msg(fd\_w, "ok\n", 3);

continue;

}

}

}

void pthr\_player\_begin(int fd\_r, int fd\_w){

pl\_st \*player = malloc(sizeof(pl\_st));

player->fd\_r = fd\_r;

player->fd\_w = fd\_w;

int ID = add\_player(player);

printf("Player-ID: %d\n", ID);

pthread\_create(&player->t\_id, NULL, &client\_thread, player);

printf("## Thread started successfully!\n");

}

void server\_thread\_start(pipes\_st pl){

int fd\_r = -1;

int fd\_w = -1;

if(mknod(pl.path\_sw, S\_IFIFO|S\_IWUSR|S\_IWOTH|S\_IRUSR|S\_IROTH, 0) < 0){

perror("Error: MKNOD path\_sw\n");

printf("PIPES: %s %s\n", pl.path\_sw, pl.path\_sr);

}

if(mknod(pl.path\_sr, S\_IFIFO|S\_IWUSR|S\_IWOTH|S\_IRUSR|S\_IROTH, 0) < 0){

perror("Error: MKNOD path\_sr\n");

printf("PIPES: %s %s\n", pl.path\_sw, pl.path\_sr);

}

printf("## Pipes created\n");

if((fd\_r = open(pl.path\_sr, O\_RDONLY)) < 0){

perror("Can't open pipe for reading!\n");

printf("PIPES: %s %s\n", pl.path\_sw, pl.path\_sr);

}

if((fd\_w = open(pl.path\_sw, O\_WRONLY)) < 0){

perror("Can't open pipe for writing!\n");

printf("PIPES: %s %s\n", pl.path\_sw, pl.path\_sr);

}

printf("CLIENT-PIPES:\n \t%s\n \t%s\n", pl.path\_sw, pl.path\_sr);

pthr\_player\_begin(fd\_r, fd\_w);

}

int main(){

sem\_init(&phore, 1, 1);

int pipe\_id = 1;

pipes\_st connection\_pl;

pipes\_st\_init(&connection\_pl, 0);

if(mknod(connection\_pl.path\_sw, S\_IFIFO|S\_IWUSR|S\_IWOTH|S\_IRUSR|S\_IROTH, 0) < 0)

perror("Error: MKNOD!\n");

for(;;) {

int fd\_w;

if((fd\_w = open(connection\_pl.path\_sw, O\_WRONLY)) < 0)

perror("Can't open pipe for writing\n");

pipes\_st client\_pl;

pipes\_st\_init(&client\_pl, pipe\_id);

write(fd\_w, client\_pl.path\_sr, strlen(client\_pl.path\_sr));

write(fd\_w, "\n", 1);

write(fd\_w, client\_pl.path\_sw, strlen(client\_pl.path\_sw));

write(fd\_w, "\n", 1);

server\_thread\_start(client\_pl);

sleep(1);

close(fd\_w);

pipe\_id++;

}

remove(connection\_pl.path\_sw);

return 0;

}

**makefile**:

all\_done: server client clean

server: server.c game.c Message.c

gcc -Wall server.c game.c Message.c -lzmq -o server>

client: client.c game.c Message.c

gcc -Wall client.c game.c Message.c -lzmq -o client

clean:

rm /tmp/bull\*

1. **Выводы**

В процессе реализации данного проекта мною частично были использованы знания из последней лабораторной работы, в которой был создан простой сервер и клиент. Здесь же возможно взаимодействие нескольких клиентов с одним сервером, что достигается засчет именованных пайпов, которые по сути своей являются отображенными в память файлами. Они очень удобны для общения между процессами, не являющимися родственными. Недостатком таких пайпов является необходимость в их ручном удалении.