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.

2. **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!
```

3. 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;
}
```

```c
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

```c
#ifndef GAME_H_
#define GAME_H_

#include <time.h>
#include <stdbool.h>
#include <ctype.h>
#include <pthread.h>

#define GAME_NAME_SIZE 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_SIZE];
	char* hidden_word;
	int max_players;
	int pl_number;
	int active_pl_id;
	pl_st *players[1];
} game_st;

static inline bool active_game(game_st *g){
	return g->active_pl_id >= 0;
}

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 "game.h"
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdio.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_PATH_NAME_SIZE 128
#define MAX_CMD_SIZE       64
#define GAME_NAME_SIZE 32

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_SIZE];
	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_NAME_SIZE];
        char pl_w[MAX_PATH_NAME_SIZE];
        read_str(fd_r, pl_w, MAX_PATH_NAME_SIZE);
        read_str(fd_r, pl_r, MAX_PATH_NAME_SIZE);
        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 MAX_PATH_NAME_SIZE 128
#define GAME_NAME_SIZE 32
#define MAX_GAMES_COUNT    10
#define MAX_PLAYERS_COUNT  32

static sem_t phore;
static game_st *games[MAX_GAMES_COUNT];
static pl_st *players[MAX_PLAYERS_COUNT];
static int pl_number = 0;
static int games_count = 0;

void serv_sem_init(){
	sem_init(&phore, 1, 1);
}

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;
}

typedef struct{
	char path_sr[MAX_PATH_NAME_SIZE];
	char path_sw[MAX_PATH_NAME_SIZE];
} pipes_st;

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++ >= MAX_GAMES_COUNT)
			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("--------Reply-------\n");
      printf("Total games: %d\n", games_count);
      do{
            if(i++ >= MAX_GAMES_COUNT)
                  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);
//                printf("\n");
            g++;
            gc++;
      } while(true);

      printf("=====End of reply====\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 < MAX_GAMES_COUNT; i++){
            if(games[(i + ind) % MAX_GAMES_COUNT] == NULL){
                  rvl = (i + ind) % MAX_GAMES_COUNT;
                  games[rvl] = g;
                  games_count++;
                  ind = (i + ind + 1) % MAX_GAMES_COUNT;
                  break;
            }
      }
      unlock();
      return rvl;
}

void remove_game(int ind){
	if(ind >= MAX_GAMES_COUNT || 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 < MAX_PLAYERS_COUNT; i++){
		if(players[(i + ind) % MAX_PLAYERS_COUNT] == NULL){
			rvl = (i + ind) % MAX_PLAYERS_COUNT;
			players[rvl] = p;
			pl_number++;
			ind = (i + ind + 1) % MAX_PLAYERS_COUNT;
			break;
		}
	}
	printf("Player was added successfully!\n");
	unlock();
	p->user_id = rvl;
	return rvl;
}

void remove_player(int ind){
	if(ind >= MAX_PLAYERS_COUNT || 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_SIZE];
                  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("$REQUEST: check user's answer\n");
                  printf("-----------------------------\n");
                  int bulls = 0;
                  int cows = 0;
                  char* word = malloc(sizeof(char)*4);
                  for (int i = 0; i < 4; i++) {
                        word[i] = req[i+1];
                  }
                  printf("GAME = %s [%d\\%d].\nWORD: ''%s''\n", game->name,
game->max_players, game->pl_number, word);
                  if (game == NULL || !active_game(game)) {
                        printf("# NOT ACTIVE!\n");
                        write_msg(fd_w, "!\n", 2);
                        continue;
                  }
                  if (game->players[game->active_pl_id] != player) {
                        printf("NOT  ME  %s  %p  %p\n",  game->name,  game-
>players[game->active_pl_id], player);
                        write_msg(fd_w, "!\n", 2);
                        continue;
                  }
                  bulls_and_cows(game, word, &bulls, &cows);
                  int len = sprintf(rep, "%d%d\n", bulls, cows);
                  if (game->max_players > 1) {
                        do {
                                    game->active_pl_id = (game->active_pl_id
+ 1) % game->max_players;
                              } while  (game->players[game->active_pl_id]  ==
NULL);
                  }
                  printf("Active player's ID %d\n\n", game->active_pl_id);
                  write_msg(fd_w, rep, len);
                  if (bulls >= WIN_BULLS){
                        game->win_id = my_ind;
                  }
                  continue;
            }
            if (*req == 'l'){
                  printf("$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 < MAX_GAMES_COUNT; i++){
                        if (games[i] == NULL)
                              continue;
```

```
                    if (active_game(games[i]))
                          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 >= MAX_GAMES_COUNT)
                              printf("Game ID = %d\n", game_ind);
                        else if(games[game_ind] == NULL)
                              printf("Null ID\n");
                        else
                              printf("%d.Game-name:   %s(%s)   PC:   %d\n",
game_ind, games[game_ind]->name, game->name, game->pl_number);
                        unlock(); print_reply(rep); lock();
                        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 idx = add_player(player);
	printf("Player-ID: %d\n", idx);
	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(){
	serv_sem_init();
	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*
```

4. **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.