

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и
прикладной математики
Кафедра вычислительной математики и
программирования

Курсовой проект по курсу
"Операционные системы"

ПРОЕКТИРОВАНИЕ КОНСОЛЬНОЙ КЛИЕНТ-СЕРВЕРНОЙ ИГРЫ

Студент: Сеимов Максим Сергеевич

Группа: М8О-208Б-21

Преподаватель: Соколов Андрей Алексеевич

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

1 Постановка задачи

Цель работы

Целью курсового проекта является:

- Приобретение практических навыков в использовании знаний, полученных в течении курса
- Проведение исследования в выбранной предметной области

Задание

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
- Присоединиться к одной из существующих игр; игра сама выбирается сервером

Игра - «Быки и коровы» (угадывать необходимо числа). Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений. 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 представлена следующими исполняемыми файлами (компилируемыми в папке с помощью утилиты **make**):

- **server** - программа-сервер. Запускается первой и постоянно ждёт сообщений от клиентов; при получении команды о создании запускает игровой сервер и перенаправляет на него игроков (создавшего и присоединяющих, если в игре ещё есть места);
- **client** - программа-клиент. При запуске подключается к серверу и все полученные команды от пользователя отправляет серверу;
- **game** - сервер конкретной игры. Не предназначена для запуска, её запускает основной сервер при получении команды создания.

Исходные файлы:

- **interprocess.h** - заголовочный файл с объявлением функций для реализации меж-процессорного взаимодействия (отправка и получение сообщений и т.д.);
- **interprocess.c** - реализация этих функций;
- **client.c** - текст клиентской программы;
- **server.c** - текст программы-сервера;
- **game.c** - текст программы игрового сервера.

Основные файлы программы

interprocess.h:

```
1  #ifndef __INTERPROCESS_H__
2  #define __INTERPROCESS_H__
3
4  #include <assert.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <unistd.h>
8  #include <zmq.h>
9
10 #define TRY_READ(C) \
11     if (read(0, (C), 1) < 0) { \
12         perror("read"); \
13         exit(EXIT_FAILURE); \
14     }
15
16 #define SERVER_PATH "./game"
17
18 #define REQUEST_TIMEOUT 2000
19 #define ADDITIONAL_TIME 100
20
21 #define NUMBER_OF_GAMES 10
22
23 #define EMPTY_MSG ""
24
25 #define CLIENT_ADRESS_PREFIX "tcp://localhost:"
26 #define SERVER_ADRESS_PREFIX "tcp://*:"
27
28 #define BASE_PORT 5050
29 #define STR_LEN 64
30 #define STR_LEN_LONG 2048
31
32 typedef enum _command {
33     CREATE = 0,
34     JOIN,
35     EXIT,
36     KILL,
37     UNKNOWN
38 } command_t;
39
40 int is_available_recv(void *socket);
41 int is_available_send(void *socket);
42
43 const char *int_to_string(unsigned a);
44 const char *portname_client(unsigned short port);
45 const char *portname_server(unsigned short port);
46
47 void create_game(int port, int number);
48
49 const char *read_word();
50 command_t get_command();
51
52 void send_guess_res(void *socket, char *log, int guessed);
53
54 #endif
```

interprocess.c:

```
1  #include "interprocess.h"
2
3  const char *int_to_string(unsigned a) {
4      int x = a, i = 0;
5      if (a == 0)
6          return "0";
7      while (x > 0) {
8          x /= 10;
9          i++;
10     }
11     char *result = (char *)calloc(sizeof(char), i + 1);
12     while (i >= 1) {
13         result[--i] = a % 10 + '0';
14         a /= 10;
15     }
16
17     return result;
18 }
19
20 char *get_reply(void *socket) {
21     zmq_msg_t reply;
22     zmq_msg_init(&reply);
23     zmq_msg_rcv(&reply, socket, 0);
24     size_t result_size = zmq_msg_size(&reply);
25
26     char *result = (char *)calloc(sizeof(char), result_size + 1);
27     memcpy(result, zmq_msg_data(&reply), result_size);
28     zmq_msg_close(&reply);
29
30     return result;
31 }
32
33 void send_guess_res(void *socket, char *log, int guessed) {
34     zmq_msg_t command_msg;
35     zmq_msg_init_size(&command_msg, sizeof(guessed));
36     memcpy(zmq_msg_data(&command_msg), &guessed, sizeof(guessed));
37     zmq_msg_send(&command_msg, socket, ZMQ_SNDMORE);
38     zmq_msg_close(&command_msg);
39
40     zmq_msg_t l_msg;
41     zmq_msg_init_size(&l_msg, strlen(log));
42     memcpy(zmq_msg_data(&l_msg), log, strlen(log) + 1);
43     zmq_msg_send(&l_msg, socket, 0);
44     zmq_msg_close(&l_msg);
45 }
46
47 const char *portname_client(unsigned short port) {
48     const char *port_string = int_to_string(port);
49     char *name = (char *)calloc(sizeof(char), strlen(CLIENT_ADRESS_PREFIX) + strlen(
50         port_string) + 1);
51     strcpy(name, CLIENT_ADRESS_PREFIX);
52     strcpy(name + strlen(CLIENT_ADRESS_PREFIX) * sizeof(char), port_string);
53     return name;
54 }
55
56 const char *portname_server(unsigned short port) {
57     const char *port_string = int_to_string(port);
58     char *name = (char *)calloc(sizeof(char), strlen(SERVER_ADRESS_PREFIX) + strlen(
59         port_string) + 1);
```

```

58     strcpy(name, SERVER_ADRESS_PREFIX);
59     strcpy(name + strlen(SERVER_ADRESS_PREFIX) * sizeof(char), port_string);
60     return name;
61 }
62
63 void create_game(int port, int number) {
64     const char *arg0 = SERVER_PATH;
65     const char *arg1 = int_to_string(port);
66     const char *arg2 = int_to_string(number);
67     execl(SERVER_PATH, arg0, arg1, arg2, NULL);
68 }
69
70 const char *read_word() {
71     char *result = (char *)calloc(sizeof(char), STR_LEN);
72     char current;
73     int i = 0;
74     TRY_READ(&current);
75     while (current != ' ') {
76         if (current == '\n' || current == '\0')
77             break;
78         result[i++] = current;
79         TRY_READ(&current);
80     }
81     result = (char *)realloc(result, sizeof(char) * (strlen(result) + 1));
82     return result;
83 }
84
85 command_t get_command() {
86     const char *input = read_word();
87
88     if (strcmp("create", input) == 0)
89         return CREATE;
90
91     if (strcmp("join", input) == 0)
92         return JOIN;
93
94     if (strcmp("exit", input) == 0)
95         return EXIT;
96
97     if (strcmp("kill", input) == 0)
98         return KILL;
99
100     return UNKNOWN;
101 }

```

server.c:

```

1  #include "interprocess.h"
2
3  int main() {
4
5      void *context = zmq_ctx_new();
6      if (context == NULL) {
7          perror("context");
8          exit(EXIT_FAILURE);
9      }
10     void *responder = zmq_socket(context, ZMQ_REP);
11     int rc = zmq_bind(responder, portname_server(BASE_PORT));
12     assert(rc == 0);
13
14     int game_number = 0;

```

```

15     int available = 0;
16     int *players = (int *)calloc(sizeof(int), NUMBER_OF_GAMES);
17     int *can_join = (int *)calloc(sizeof(int), NUMBER_OF_GAMES);
18
19     while (1) {
20         int arg = 0;
21         int number_of_players = 0;
22         command_t current = UNKNOWN;
23         while (1) {
24             zmq_msg_t msg;
25             int rec = zmq_msg_init(&msg);
26             assert(rec == 0);
27             rec = zmq_msg_recv(&msg, responder, 0);
28             assert(rec != -1);
29             switch (arg) {
30                 case 0:
31                     memcpy(&current, zmq_msg_data(&msg), sizeof(current));
32                     break;
33                 case 1:
34                     if (current == CREATE)
35                         memcpy(&number_of_players, zmq_msg_data(&msg), sizeof(number_of_players));
36                     break;
37             }
38             zmq_msg_close(&msg);
39             ++arg;
40             if (!zmq_msg_more(&msg))
41                 break;
42         }
43
44         int reply;
45         zmq_msg_t rep_msg;
46         switch (current) {
47             case CREATE::;
48                 if (game_number < NUMBER_OF_GAMES) {
49                     reply = BASE_PORT + 1;
50                     for (int i = 0; i < game_number; ++i) {
51                         reply += players[i];
52                     }
53                     if (number_of_players != 1) {
54                         available++;
55                     }
56                     players[game_number] = number_of_players;
57                     can_join[game_number++] = number_of_players - 1;
58                 } else {
59                     reply = 0;
60                 }
61
62                 zmq_msg_init_size(&rep_msg, sizeof(reply));
63                 memcpy(zmq_msg_data(&rep_msg), &reply, sizeof(reply));
64                 zmq_msg_send(&rep_msg, responder, 0);
65                 zmq_msg_close(&rep_msg);
66                 int fork_val = fork();
67                 if (fork_val == 0)
68                     create_game(reply, number_of_players);
69                 printf("%d\n", fork_val);
70                 break;
71             case JOIN::;
72                 int reply;
73                 if (available != 0) {
74                     reply = BASE_PORT + 1;

```

```

75         for (int i = 0; i < game_number; ++i) {
76             if (can_join[i] > 0) {
77                 reply += players[i] - can_join[i];
78                 if (--can_join[i] == 0)
79                     available--;
80                 break;
81             }
82             reply += players[i];
83         }
84     } else {
85         reply = 0;
86     }
87
88     zmq_msg_init_size(&rep_msg, sizeof(reply));
89     memcpy(zmq_msg_data(&rep_msg), &reply, sizeof(reply));
90     zmq_msg_send(&rep_msg, responder, 0);
91     zmq_msg_close(&rep_msg);
92     break;
93 case KILL:
94     zmq_msg_init_size(&rep_msg, 0);
95     zmq_msg_send(&rep_msg, responder, 0);
96     zmq_msg_close(&rep_msg);
97     zmq_close(responder);
98     zmq_ctx_destroy(context);
99     return 0;
100 default:
101     printf("Hmm\n");
102     zmq_close(responder);
103     zmq_ctx_destroy(context);
104     return 0;
105     break;
106 }
107 }
108 }

```

game.c:

```

1  #include <time.h>
2
3  #include "interprocess.h"
4
5  const char *player_name(int id) {
6      const char *id_str = int_to_string(id + 1);
7      char *name = (char *)calloc(sizeof(char), strlen("Player ") + strlen(id_str) + 1);
8      strcpy(name, "Player ");
9      strcpy(name + strlen("Player ") * sizeof(char), id_str);
10     return name;
11 }
12
13 const char *check_guess(long target, long guess) {
14     int b = 0, c = 0;
15     const char *target_str = int_to_string(target);
16     const char *guess_str = int_to_string(guess);
17     int bulls[5] = {0, 0, 0, 0, 0};
18     int cows[5] = {0, 0, 0, 0, 0};
19     for (int i = 0; i < 5; ++i) {
20         if (target_str[i] == guess_str[i]) {
21             b++;
22             bulls[i] = 1;
23         }
24     }

```

```

25     for (int i = 0; i < 5; ++i) {
26         for (int j = 0; j < 5; ++j) {
27             if (j != i) {
28                 if ((target_str[j] == guess_str[i]) && (bulls[j] == 0) && (cows[j] == 0)) {
29                     cows[j] = 1;
30                     c++;
31                     break;
32                 }
33             }
34         }
35     }
36     char *result = (char *)calloc(sizeof(char), 5);
37     sprintf(result, "%db%dc", b, c);
38     return result;
39 }
40
41 int main(int argc, char const *argv[]) {
42     int port = atoi(argv[1]);
43     int player_number = atoi(argv[2]);
44
45     void *context = zmq_ctx_new();
46     if (context == NULL) {
47         perror("context");
48         exit(EXIT_FAILURE);
49     }
50
51     void **sockets = calloc(sizeof(void *), player_number);
52     for (int i = 0; i < player_number; ++i) {
53         sockets[i] = zmq_socket(context, ZMQ_REQ);
54         int opt_val = 0;
55         int rc = zmq_setsockopt(sockets[i], ZMQ_LINGER, &opt_val, sizeof(opt_val));
56         assert(rc == 0);
57         if (sockets[i] == NULL) {
58             perror("socket");
59             exit(EXIT_FAILURE);
60         }
61
62         rc = zmq_connect(sockets[i], portname_client(port + i));
63         assert(rc == 0);
64
65         zmq_msg_t rep_msg;
66         zmq_msg_init_size(&rep_msg, sizeof(i));
67         memcpy(zmq_msg_data(&rep_msg), &i, sizeof(i));
68         zmq_msg_send(&rep_msg, sockets[i], 0);
69         zmq_msg_close(&rep_msg);
70
71         zmq_msg_t msg;
72         rc = zmq_msg_init(&msg);
73         assert(rc == 0);
74         rc = zmq_msg_recv(&msg, sockets[i], 0);
75         assert(rc != -1);
76         zmq_msg_close(&msg);
77     }
78     for (int i = 0; i < player_number; ++i) {
79         zmq_msg_t rep_msg;
80         const char *greetings = "The game begins!";
81         zmq_msg_init_size(&rep_msg, strlen(greetings));
82         memcpy(zmq_msg_data(&rep_msg), greetings, strlen(greetings) + 1);
83         zmq_msg_send(&rep_msg, sockets[i], 0);
84         zmq_msg_close(&rep_msg);

```



```

85
86     zmq_msg_t msg;
87     int rc = zmq_msg_init(&msg);
88     assert(rc == 0);
89     rc = zmq_msg_recv(&msg, sockets[i], 0);
90     assert(rc != -1);
91     zmq_msg_close(&msg);
92 }
93
94 srand((unsigned)time(NULL));
95 long number = rand() % 900000 + 10000;
96 char *guesses = (char *)calloc(sizeof(char), STR_LEN_LONG);
97 guesses = strcat(guesses, "All guesses:\n");
98 int guessed = 0;
99 int winner = 0;
100 while (!guessed) {
101     for (int i = 0; i < player_number; ++i) {
102         send_guess_res(sockets[i], guesses, guessed);
103         long guess;
104         zmq_msg_t msg;
105         int rc = zmq_msg_init(&msg);
106         assert(rc == 0);
107         rc = zmq_msg_recv(&msg, sockets[i], 0);
108         assert(rc != -1);
109         memcpy(&guess, zmq_msg_data(&msg), sizeof(guess));
110         zmq_msg_close(&msg);
111         if (guessed)
112             continue;
113
114         const char *check = check_guess(number, guess);
115
116         guesses = strcat(guesses, player_name(i));
117         guesses = strcat(guesses, " guessed ");
118         guesses = strcat(guesses, int_to_string(guess));
119         guesses = strcat(guesses, ". Result: ");
120         guesses = strcat(guesses, check);
121         guesses = strcat(guesses, "\n");
122
123         if (strcmp(check, "5b0c") == 0) {
124             guesses = strcat(guesses, player_name(i));
125             guesses = strcat(guesses, " is the winner!\n");
126             guessed = 1;
127             winner = i;
128         }
129     }
130 }
131 for (int i = 0; i <= winner; ++i) {
132     send_guess_res(sockets[i], guesses, guessed);
133     zmq_msg_t msg;
134     int rc = zmq_msg_init(&msg);
135     assert(rc == 0);
136     rc = zmq_msg_recv(&msg, sockets[i], 0);
137     assert(rc != -1);
138     zmq_msg_close(&msg);
139 }
140
141 for (int i = 0; i < player_number; ++i) {
142     zmq_close(sockets[i]);
143 }
144 zmq_ctx_destroy(context);

```

```

145
146     return 0;
147 }

```

client.c:

```

1  #include "interprocess.h"
2
3  void play_game(void *socket) {
4      int id;
5      zmq_msg_t msg;
6      int rc = zmq_msg_init(&msg);
7      assert(rc == 0);
8      rc = zmq_msg_recv(&msg, socket, 0);
9      assert(rc != -1);
10     memcpy(&id, zmq_msg_data(&msg), sizeof(id));
11     zmq_msg_close(&msg);
12
13     printf("You will be playing as Player %d\n", id + 1);
14     printf("Now let's wait for others\n");
15
16     rc = zmq_msg_init(&msg);
17     assert(rc == 0);
18     rc = zmq_msg_send(&msg, socket, 0);
19     assert(rc != -1);
20     zmq_msg_close(&msg);
21
22     rc = zmq_msg_init(&msg);
23     assert(rc == 0);
24     rc = zmq_msg_recv(&msg, socket, 0);
25     assert(rc != -1);
26     size_t result_size = zmq_msg_size(&msg);
27
28     char *text = (char *)calloc(sizeof(char), result_size + 1);
29     memcpy(text, zmq_msg_data(&msg), result_size);
30     zmq_msg_close(&msg);
31     printf("%s\n", text);
32
33     rc = zmq_msg_init(&msg);
34     assert(rc == 0);
35     rc = zmq_msg_send(&msg, socket, 0);
36     assert(rc != -1);
37     zmq_msg_close(&msg);
38
39     while (1) {
40         int arg = 0;
41         int type = -1;
42         char *guesses = (char *)calloc(sizeof(char), STR_LEN_LONG);
43         while (1) {
44             zmq_msg_t msg;
45             int rec = zmq_msg_init(&msg);
46             assert(rec == 0);
47             rec = zmq_msg_recv(&msg, socket, 0);
48             assert(rec != -1);
49             switch (arg) {
50                 case 0:
51                     memcpy(&type, zmq_msg_data(&msg), sizeof(type));
52                     break;
53                 case 1:
54                     memcpy(guesses, zmq_msg_data(&msg), zmq_msg_size(&msg));
55                     break;

```

```

56         }
57         zmq_msg_close(&msg);
58         ++arg;
59         if (!zmq_msg_more(&msg))
60             break;
61     }
62     printf("\n%s", guesses);
63     if (type == 0) {
64         printf("\nMake your move!\n");
65         long guess;
66         while (1) {
67             const char *guess_str = read_word();
68             guess = atoi(guess_str);
69             if (guess > 99999 || guess < 10000) {
70                 printf("Number should contain exactly 5 digits\n");
71                 continue;
72             }
73             break;
74         }
75         zmq_msg_t g_msg;
76         rc = zmq_msg_init(&g_msg);
77         assert(rc == 0);
78         zmq_msg_init_size(&g_msg, sizeof(guess));
79         memcpy(zmq_msg_data(&g_msg), &guess, sizeof(guess));
80         rc = zmq_msg_send(&g_msg, socket, 0);
81         printf("sent!\n");
82         assert(rc != -1);
83         zmq_msg_close(&g_msg);
84     } else {
85         zmq_msg_t g_msg;
86         rc = zmq_msg_init(&g_msg);
87         assert(rc == 0);
88         rc = zmq_msg_send(&g_msg, socket, 0);
89         assert(rc != -1);
90         zmq_msg_close(&g_msg);
91         break;
92     }
93 }
94 zmq_close(socket);
95 }
96
97 int main() {
98     command_t c;
99     void *context = zmq_ctx_new();
100     if (context == NULL) {
101         perror("context");
102         exit(EXIT_FAILURE);
103     }
104     void *client = zmq_socket(context, ZMQ_REQ);
105     if (client == NULL) {
106         perror("socket");
107         exit(EXIT_FAILURE);
108     }
109
110     int rc = zmq_connect(client, portname_client(BASE_PORT));
111     assert(rc == 0);
112     while (1) {
113         c = get_command();
114         zmq_msg_t req_msg;
115         zmq_msg_t reply;

```

```

116     int rep_val;
117     switch (c) {
118     case EXIT:
119         break;
120     case CREATE:;
121         const char *number_str = read_word();
122         int number = atoi(number_str);
123         if (number <= 0) {
124             printf("Players number should be a positive integer number\n");
125             continue;
126         }
127         zmq_msg_init_size(&req_msg, sizeof(c));
128         memcpy(zmq_msg_data(&req_msg), &c, sizeof(c));
129         zmq_msg_send(&req_msg, client, ZMQ_SNDMORE);
130         zmq_msg_close(&req_msg);
131
132         zmq_msg_init_size(&req_msg, sizeof(number));
133         memcpy(zmq_msg_data(&req_msg), &number, sizeof(number));
134         zmq_msg_send(&req_msg, client, 0);
135         zmq_msg_close(&req_msg);
136
137         zmq_msg_init(&reply);
138         zmq_msg_recv(&reply, client, 0);
139         memcpy(&rep_val, zmq_msg_data(&reply), sizeof(rep_val));
140         zmq_msg_close(&reply);
141
142         if (rep_val == 0) {
143             printf("Sorry, server is already full, try restarting it\n");
144             zmq_close(client);
145             zmq_ctx_destroy(context);
146             return 0;
147         } else {
148             zmq_close(client);
149             client = zmq_socket(context, ZMQ_REP);
150             rc = zmq_bind(client, portname_server(rep_val));
151             assert(rc == 0);
152             play_game(client);
153             zmq_ctx_destroy(context);
154             return 0;
155         }
156
157         break;
158     case JOIN:
159         zmq_msg_init_size(&req_msg, sizeof(c));
160         memcpy(zmq_msg_data(&req_msg), &c, sizeof(c));
161         zmq_msg_send(&req_msg, client, 0);
162         zmq_msg_close(&req_msg);
163
164         zmq_msg_t rep;
165         zmq_msg_init(&rep);
166         zmq_msg_recv(&rep, client, 0);
167
168         memcpy(&rep_val, zmq_msg_data(&rep), sizeof(rep_val));
169         zmq_msg_close(&rep);
170         if (rep_val == 0) {
171             printf("Sorry, no available games for you\n");
172             zmq_close(client);
173             zmq_ctx_destroy(context);
174             return 0;
175         } else {

```

```

176         zmq_close(client);
177         client = zmq_socket(context, ZMQ_REP);
178         rc = zmq_bind(client, portname_server(rep_val));
179         assert(rc == 0);
180         play_game(client);
181         zmq_ctx_destroy(context);
182         return 0;
183     }
184     break;
185 case KILL:
186     zmq_msg_init_size(&req_msg, sizeof(c));
187     memcpy(zmq_msg_data(&req_msg), &c, sizeof(c));
188     zmq_msg_send(&req_msg, client, 0);
189     zmq_msg_close(&req_msg);
190
191     zmq_msg_init(&reply);
192     zmq_msg_recv(&reply, client, 0);
193     c = EXIT;
194     break;
195
196 default:
197     printf("Invalid command\n");
198     continue;
199     break;
200 }
201 if (c == EXIT)
202     break;
203 }
204 zmq_close(client);
205 zmq_ctx_destroy(context);
206 }

```

Пример работы

Запуск программы для одного и двух игроков (сервер был запущен перед каждым тестом):

```
idddqd@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_cp/src
(base) idddqd@IDDQD-IdeaPad-5:~/Projects/C/OS/os_cp/src$ ./client
create 1
You will be playing as Player 1
Now let's wait for others
The game begins!

All guesses:

Make your move!
12345
sent!

All guesses:
Player 1 guessed 12345. Result: 1b1c

Make your move!
95673
sent!

All guesses:
Player 1 guessed 12345. Result: 1b1c
Player 1 guessed 95673. Result: 0b2c

Make your move!
|
```

Рис. 1: Запрос на запуск игры для одного игрока и ход игры

```
idddqd@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_cp/src
(base) idddqd@IDDQD-IdeaPad-5:~/Projects/C/OS/os_cp/src$ ./client
create 2
You will be playing as Player 1
Now let's wait for others
The game begins!

All guesses:

Make your move!
12345
sent!

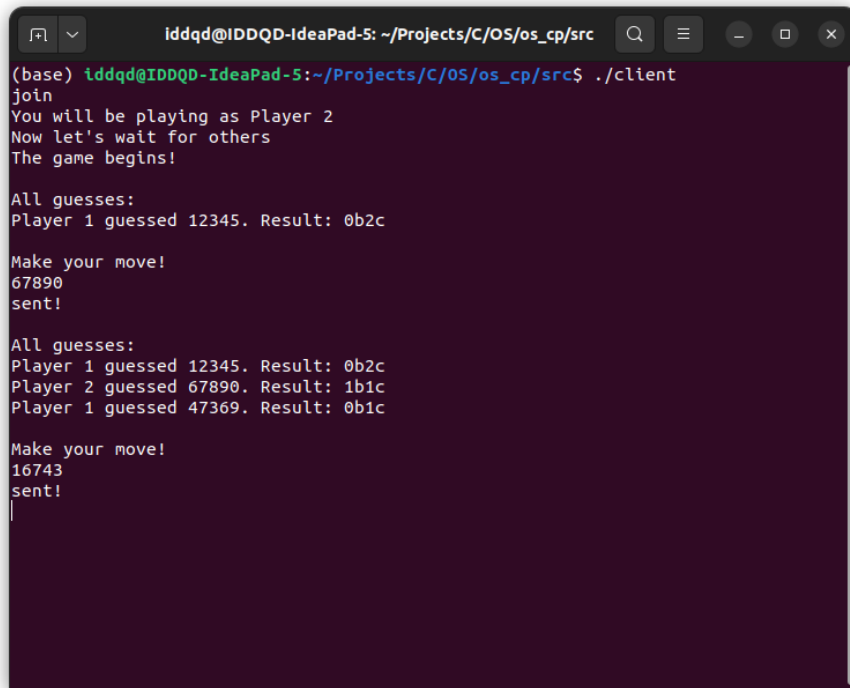
All guesses:
Player 1 guessed 12345. Result: 0b2c
Player 2 guessed 67890. Result: 1b1c

Make your move!
47369
sent!

All guesses:
Player 1 guessed 12345. Result: 0b2c
Player 2 guessed 67890. Result: 1b1c
Player 1 guessed 47369. Result: 0b1c
Player 2 guessed 16743. Result: 0b1c

Make your move!
```

Рис. 2: Запрос на запуск игры для двух игроков, ожидание подключения второго и ход игры



```
iddqd@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_cp/src
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_cp/src$ ./client
join
You will be playing as Player 2
Now let's wait for others
The game begins!

All guesses:
Player 1 guessed 12345. Result: 0b2c

Make your move!
67890
sent!

All guesses:
Player 1 guessed 12345. Result: 0b2c
Player 2 guessed 67890. Result: 1b1c
Player 1 guessed 47369. Result: 0b1c

Make your move!
16743
sent!
|
```

Рис. 3: Запрос на присоединение, поиск свободной игре и ход игры (та же, что и в предыдущем тесте)

Вывод

В ходе выполнения задания были изучены принципы работы очередей сообщений и написана игра типа клиент-сервер, одновременно принимающая запросы от множества пользователей и создающая множество параллельных независимых игр.