Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Операционные системы»

Студентка: А. Довженко
Преподаватель: Е. С. Миронов
Группа: 08-207
Дата:
Оценка:
Подпись:

Москва, 2017

Курсовой проект

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант: Проектирование консольной клиент-серверной игры Охота на лис. На основе сервера сообщений ZeroMQ создать собственную игру более чем для одного пользователя. Игра устроена по принципу клиент-сервер.

1 Описание

Для реализации связи клиент-сервер был выбран паттерн RequestResponse. Клиент отправляет запрос на сервер и ждет ответа. После того, как ответ пришел, клиент может продолжать работу. Сервер запускается с ключом количества игроков (от 2 до 4). Клиент подключается к серверу. Клиент может выбрать один из двух режимов для игры: в первом режиме нужно найти лис за время меньшее, чем время остальных игроков, во втором режиме надо найти всех лис за наименьшее число ходов. Клиенты выбирают режимы независимо друг от друга. То есть если играют 2 игрока, один из них выбрал режим времени, а другой режим ходов, и первый выиграл быстрее второго, но за большее число ходов, то победное очко будет засчитано обоим игрокам.

Когда все пользователи подключились, начинается игра. После начала игры каждый клиент играет с копией одного главного поля на своей стороне. Когда он найдет все 8 лис, эта информация отправится на сервер. Далее игрок может либо дождаться остальных игроков и увидеть результат, либо посмотреть свою статистику за все время, либо посмотреть топ-5 игроков за все время, либо выйти из игры. Как только на сервер придут все результаты, то сервер отсортирует их по количеству ходов и составит рейтинг этой игры, который смогут получить игроки.

На стороне сервера реализована база данных игроков. Она нужна для подсчета статистики. Каждый пользователь может узнать свою статистику и рейтинг 5 лучших игроков. В статистике есть количество побед, количество поражений и минимальное число ходов, за которое удалось выиграть. Статистика на оба режима игры одна.

Системные вызовы:

void exit(int status); – функция выходит из процесса с заданным статусом.
int zmq connect(void *socket, const char *endpoint); – подключает socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.
int zmq bind(void *socket, const char *endpoint); – присоединяет socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.
void *zmq socket(void *context, int type); – создает сокет типа type из котнекста context.
int zmq msg send(zmq msg t *msg, void *socket, int flags); – отправляет сообщение msg в socket с параметрами flags, возвращает количество отправленных байт, в случае ошибки возвращает -1.
int zmq msg init(zmq msg t *msg) – инициализирует сообщение msg как пустой объект.
int zmq msg recv(zmq msg t *msg, void *socket, int flags); – получает сообщение из socket в msg с параметрами flags, возвращает количество полученных байт, в случае ошибки возвращает -1.

int zmq msg close(zmq msg t *msg) – очищает содержимое msg, аналог free для сообщений zmq, возвращает 0 в случае успеха и -1 в случае неудачи.
int zmq close(void *socket); – закрывает сокет socket, возвращает 0 в случае успеха и -1 в случае неудачи.
int zmq ctx destroy(void *context); – разрушает контекст context, блокирует доступ всем операциям кроме zmq close, все сообщения в сокетах либо физически отправлены, либо "висят".

2 Исходный код

3 argument.h

```c
#ifndef ARG_H
#define ARG_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>

#include "zmq.h"

#define SERVER_USAGE "usage: ./server -p <number of players (from 2 to 4)>\n"
#define CLIENT_USAGE "usage: ./client -l <login>\n"

#define SIZE_LOG 100
#define MSG_SIZE 2048

#define error(msg) \
    do { perror(msg); exit(EXIT_FAILURE); } while (0)

typedef struct _cell {
    int numberOfFoxes;
    int cntFoxes;
    int fox;
    int print;
    int check;
} Cell;

typedef struct _result {
    char name[SIZE_LOG];
    size_t steps;
    int yep;
    int mode;
} Result;

typedef struct _args{
    Cell matrix[10][10];
    char log[SIZE_LOG];
    int id1;
    size_t players;
    void *requester;
    int status;
    int mode;
    size_t result;
```

```
} Args;

#endif
```

4 stats.h

```
#ifndef _STATS_H_
#define _STATS_H_

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <inttypes.h>
#include <string.h>
#include "argument.h"

#define SUCCESS 1
#define NOT_MEMORY -1
#define NOT_PLAYER -2

#define SIZE_LOG 100
#define DB_NAME "playersDB"

/* TODO: use tree for db, not vector */

typedef struct _player {
    char log[SIZE_LOG];
    int wins;
    int loses;
    int min_steps;
} *Player;

typedef struct _tablePlayers {
    Player players;
    uint32_t size;
    uint32_t freespace;
} *TablePlrs;

TablePlrs tableCreate(void);
void tableAdd(TablePlrs pl, char *log);
void tablePrint(TablePlrs pl);
Player tableFind(TablePlrs pl, char *log);
void tableDestroy(TablePlrs *pl);
void tableSort(TablePlrs pl);

int tableSave(TablePlrs pl, FILE *file);
int tableLoad(TablePlrs *pl, FILE *file);

void tableFill(TablePlrs pl, char *log, int wins, int loses, int min_steps);
```

```
44 void tableUpdate(TablePlrs pl, char *log, int res, int steps);
45
46 #endif
```

5 stats.c

```
1  #include "stats.h"
2
3  TablePlrs tableCreate(void)
4  {
5      TablePlrs pl = (TablePlrs) malloc(sizeof(*pl));
6      if (!pl) {
7          fprintf(stderr, "ERROR: no memory\n");
8          exit(NOT_MEMORY);
9      }
10     pl->players = (Player) malloc(sizeof(*(pl->players)));
11     pl->size = 0;
12     pl->freespace = 1;
13     return pl;
14 }
15
16 void tableResize(TablePlrs pl)
17 {
18     pl->players = realloc(pl->players, 2 * pl->size * sizeof(*(pl->players)));
19     if (!pl->players) {
20         fprintf(stderr, "ERROR: no memory\n");
21         exit(NOT_MEMORY);
22     }
23     pl->freespace = pl->size;
24 }
25
26 void tableAdd(TablePlrs pl, char *log)
27 {
28     if (!pl->freespace) {
29         tableResize(pl);
30     }
31     strcpy(pl->players[pl->size].log, log);
32     pl->players[pl->size].wins = 0;
33     pl->players[pl->size].loses = 0;
34     pl->players[pl->size].min_steps = 100;
35     pl->size++;
36     pl->freespace--;
37 }
38
39 void tableFill(TablePlrs pl, char *log, int win, int lose, int min_step)
40 {
41     Player toFill = tableFind(pl, log);
42     if(toFill) {
43         toFill->wins = win;
```

```
44          toFill->loses = lose;
45          toFill->min_steps = min_step;
46      }
47  }
48
49  void tableUpdate(TablePlrs pl, char *log, int res, int steps)
50  {
51      Player player = tableFind(pl, log);
52      if (!player) {
53          tableAdd(pl, log);
54          player = tableFind(pl, log);
55      }
56      res == 1 ? player->wins++ : player->loses++;
57      if (steps < player->min_steps) {
58          player->min_steps = steps;
59      }
60  }
61
62  void tablePrint(TablePlrs pl)
63  {
64      if (pl) {
65          printf("PLAYER\tWINS\tLOSES\tMIN STEPS\n");
66          for (uint32_t i = 0; i < pl->size; ++i) {
67              printf("%s\t", pl->players[i].log);
68              printf("%d\t", pl->players[i].wins);
69              printf("%d\t", pl->players[i].loses);
70              printf("%d\n", pl->players[i].min_steps);
71          }
72      }
73  }
74
75  Player tableFind(TablePlrs pl, char *log)
76  {
77      if (pl) {
78          for (uint32_t i = 0; i < pl->size; ++i) {
79              if (!strcmp(pl->players[i].log, log)) {
80                  return &(pl->players[i]);
81              }
82          }
83      }
84      return NULL;
85  }
86
87  void tableDestroy(TablePlrs *pl)
88  {
89      free((*pl)->players);
90      (*pl)->players = NULL;
91      free(*pl);
92      *pl = NULL;
```

```
93  }
94
95  void tableSort(TablePlrs pl)
96  {
97      struct _player tmp;
98      if (pl) {
99          for(int i = 1; i < pl->size; ++i) {
100             for(int j = i; j > 0; --j) {
101                 if (pl->players[j - 1].wins < pl->players[j].wins) {
102                     tmp = pl->players[j - 1];
103                     pl->players[j - 1] = pl->players[j];
104                     pl->players[j] = tmp;
105                 }
106             }
107         }
108     }
109 }
110
111 /* TODO: binary read and write */
112 int tableSave(TablePlrs pl, FILE *file)
113 {
114     if (file < 0) {
115         printf("Cannot open file\n");
116         return 1;
117     }
118     for (int i = 0; i < pl->size; ++i) {
119         fprintf(file, "%s %d %d %d ", pl->players[i].log, pl->players[i].wins, pl->
                players[i].loses, pl->players[i].min_steps);
120     }
121     return 0;
122 }
123
124 int tableLoad(TablePlrs *pl, FILE *file)
125 {
126     Player tmp;
127     if (file < 0) {
128         printf("Cannot open file\n");
129         return 1;
130     }
131     while (fscanf(file, "%s %d %d %d", tmp->log, &tmp->wins, &tmp->loses, &tmp->
            min_steps) == 4) {
132         tableAdd((*pl), tmp->log);
133         tableFill((*pl), tmp->wins, tmp->loses, tmp->min_steps);
134     }
135     return 0;
136 }
```

6 client.c

```c
#include "argument.h"

void help()
{
    printf("1) Result this game\n2) My stats\n3) Top-5 for all time\n4) Exit\n");
}

void printRes(Result *tmp, size_t n)
{
    for (int i = 0; i < n; ++i) {
        printf("%d) %s: %zu steps\n", i + 1, tmp[i].name, tmp[i].steps);
    }
    printf("\n");
}

void selectMode()
{
    printf("Select game mode:\n1) time\n2) minimum steps\n");
}

int main(int argc, char **argv)
{
    int i;
    int j;
    char answer[MSG_SIZE];
    size_t steps = 0;
    Result table[4];
    void *context = zmq_ctx_new();
    if (!context) error("zmq_ctx_new");
    void *requester = zmq_socket(context, ZMQ_REQ);

    /* PLAYER ATTRIBUTES */
    Args myGame;
    myGame.players = 0;
    myGame.status = -1;
    int caughtFoxes = 0;

    /* CHECK ARGUMENTS FROM INPUT */
    if (argc == 3 && !strcmp(argv[1],"-l")) {
        strcpy(myGame.log, argv[2]);
    } else {
        printf(CLIENT_USAGE);
        return 0;
    }

    /* CONNECTING */
    int cn = zmq_connect(requester, "tcp://localhost:5555");
    printf("Connect to tcp://localhost:5555\n");
    if (cn) error("connect");
```

```
50
51      int time = 1500;
52      zmq_setsockopt(requester, ZMQ_RCVTIMEO, &time, sizeof(time));
53
54      zmq_send(requester, &myGame, sizeof(Args), 0);
55      zmq_recv(requester, &myGame, sizeof(Args), 0);
56
57      if (myGame.players == 0) {
58          printf("There are no more places\n");
59          return 0;
60      }
61
62      /* 2 MODES: TIME AND STEPS */
63      selectMode();
64      while(1) {
65          scanf("%d", &myGame.mode);
66          if (myGame.mode != 1 && myGame.mode != 2) {
67              printf("Unknown mode. Pls, select 1 or 2.\n");
68          } else {
69              break;
70          }
71      }
72
73
74      /* WAITING ALL PLAYERS */
75      printf("Wait...\n");
76      while(1) {
77          zmq_send(requester, &myGame, sizeof(Args), 0);
78          zmq_recv(requester, &myGame, sizeof(Args), 0);
79          if(myGame.status == 1) {
80              break;
81          }
82      }
83
84      /* GAME */
85      while (caughtFoxes < 8) {
86          printf("Enter the coordinates (x & y), pls: (from 1 to 10)\n");
87          printf("x = ");
88          scanf("%d", &i);
89          if(i == -1) {
90              printf("Hey admin!!!\n");
91              break;
92          }
93          if (i < 1 || i > 10) {
94              printf("Pls, from 1 to 10.\n");
95              continue;
96          }
97          printf("y = ");
98          scanf("%d", &j);
```

```
99          if (j < 1 || j > 10) {
100             printf("Pls, from 1 to 10.\n");
101             continue;
102         }
103         --i;
104         --j;
105         if (myGame.matrix[i][j].check == 1) {
106             printf("You have checked this cell.\n");
107             continue;
108         } else {
109             myGame.matrix[i][j].check = 1;
110         }
111
112         if (myGame.matrix[i][j].fox == 1) {
113             printf("\nGrats, you caught %d fox(es).\n", myGame.matrix[i][j].cntFoxes);
114             caughtFoxes += myGame.matrix[i][j].cntFoxes;
115         } else {
116             printf("\nOh, fox is not here\n");
117         }
118
119         myGame.matrix[i][j].print = 1;
120         printf("\n");
121         for(int k = 0; k < 10; ++k) {
122             for(int l = 0; l < 10; ++l) {
123                 if(myGame.matrix[k][l].print == 0) {
124                     printf("x ");
125                 } else {
126                     if(myGame.matrix[k][l].fox != 1) {
127                         printf("%d ", myGame.matrix[k][l].numberOfFoxes);
128                     } else {
129                         printf("F ");
130                     }
131                 }
132             }
133             printf("\n");
134         }
135         printf("\n");
136         printf("Have to catch %d fox(es)\n", 8 - caughtFoxes);
137         ++steps;
138     }
139
140     printf("Congratulations, you catch all the foxes.\n");
141     printf(" |\\/| /\\ /\n .__.. \\_/ \\/ \n \\____/_ __/ \n /_/_/ \n\n");
142     myGame.result = steps;
143     myGame.status = 0;
144     zmq_send(requester, &myGame, sizeof(Args), 0);
145     zmq_recv(requester, &answer, sizeof(answer), 0);
146     printf("%s\n", answer);
147     while(1) {
```

```
148         int cmd;
149         help();
150         scanf("%d", &cmd);
151         switch(cmd) {
152             /* GET RESULT GAME */
153             case 1: {
154                 myGame.status = cmd;
155                 zmq_send(requester, &myGame, sizeof(Args), 0);
156                 zmq_recv(requester, &table, sizeof(Result) * 4, 0);
157                 if(table[0].yep) {
158                     printRes(table, myGame.players);
159                 } else {
160                     printf("Pls, wait other players.\n");
161                 }
162                 break;
163             }
164 
165             /* GET MY STATISTICS */
166             case 2: {
167                 myGame.status = cmd;
168                 zmq_send(requester, &myGame, sizeof(Args), 0);
169                 zmq_recv(requester, &answer, sizeof(answer), 0);
170                 printf("%s\n", answer);
171                 break;
172             }
173 
174             /* GET TOP-5 PLAYERS */
175             case 3: {
176                 myGame.status = cmd;
177                 zmq_send(requester, &myGame, sizeof(Args), 0);
178                 zmq_recv(requester, &answer, sizeof(answer), 0);
179                 printf("%s\n", answer);
180                 break;
181             }
182 
183             /* EXIT */
184             case 4: {
185                 myGame.status = cmd;
186                 zmq_send(requester, &myGame, sizeof(Args), 0);
187                 zmq_recv(requester, &answer, sizeof(answer), 0);
188                 printf("%s\n", answer);
189                 zmq_close(requester);
190                 zmq_ctx_term(context);
191                 return 0;
192                 break;
193             }
194 
195             default: {
196                 printf("Unknown command\n");
```

```
197                 break;
198             }
199         }
200     }
201     zmq_close(requester);
202     zmq_ctx_destroy(context);
203     return 0;
204 }
```

7 server.c

```
1  #include "argument.h"
2  #include "stats.h"
3
4  #define TIME_GAME 1
5  #define STEP_GAME 2
6
7      /* Create a field for game */
8  void fieldCreate(Cell field[][10])
9  {
10     /* initialization */
11     for (int i = 0; i < 10; ++i) {
12         for (int j = 0; j < 10; ++j) {
13             field[i][j].numberOfFoxes = 0;
14             field[i][j].cntFoxes = 0;
15             field[i][j].fox = 0;
16             field[i][j].print = 0;
17             field[i][j].check = 0;
18         }
19     }
20
21     /* fox placement */
22     srand(time(NULL));
23     for (int k = 0; k < 8; ++k) {
24         int i = rand() % 10;
25         int j = rand() % 10;
26         field[i][j].fox = 1;
27         field[i][j].cntFoxes++;
28     }
29 }
30
31     /* counting of foxes */
32 void countOfFoxs(Cell field[][10])
33 {
34     for (int i = 0; i < 10; ++i) {
35         for (int j = 0; j < 10; ++j) {
36             /* horizontal line */
37             for (int k = 0; k < 10; ++k) {
38                 if (field[i][k].fox == 1 && k != j) {
```

```
39                      field[i][j].numberOfFoxes += field[i][k].cntFoxes;
40                  }
41              }
42              /* vertical line */
43              for (int k = 0; k < 10; ++k) {
44                  if (field[k][j].fox == 1 && k != i) {
45                      field[i][j].numberOfFoxes += field[k][j].cntFoxes;
46                  }
47              }
48              /* diagonal line "\" */
49              for (int k = i - 1, l = j - 1; k >= 0 && l >= 0; --k, --l) {
50                  if (field[k][l].fox == 1) {
51                      field[i][j].numberOfFoxes += field[k][l].cntFoxes;
52                  }
53              }
54              for (int k = i + 1, l = j + 1; k < 10 && l < 10; ++k, ++l) {
55                  if (field[k][l].fox == 1) {
56                      field[i][j].numberOfFoxes += field[k][l].cntFoxes;
57                  }
58              }
59              /* diagonal line "/" */
60              for (int k = i + 1, l = j - 1; k < 10 && l >= 0; ++k, --l) {
61                  if (field[k][l].fox == 1) {
62                      field[i][j].numberOfFoxes += field[k][l].cntFoxes;
63                  }
64              }
65              for (int k = i - 1, l = j + 1; k >= 0 && l < 10; --k, ++l) {
66                  if (field[k][l].fox == 1) {
67                      field[i][j].numberOfFoxes += field[k][l].cntFoxes;
68                  }
69              }
70              if (field[i][j].fox == 1) {
71                  field[i][j].numberOfFoxes += field[i][j].cntFoxes;
72              }
73          }
74      }
75  }
76      /* a copy of struct "Argumets" */
77  void copy(Args *tmp, Args *tmp2)
78  {
79      for(int i = 0; i < 10; ++i) {
80          for(int j = 0; j < 10; ++j) {
81              tmp2->matrix[i][j] = tmp->matrix[i][j];
82          }
83      }
84  }
85
86      /* a sorting of results */
87  void sort(Result *table, size_t n)
```

```
88  {
89      Result tmp;
90      for(int i = 1; i < n; ++i) {
91          for(int j = i; j > 0; --j) {
92              if(table[j - 1].steps > table[j].steps) {
93                  tmp = table[j - 1];
94                  table[j - 1] = table[j];
95                  table[j] = tmp;
96              }
97          }
98      }
99  }
100
101     /* print of results */
102 void printRes(Result *tmp, size_t n)
103 {
104     for (int i = 0; i < n; ++i) {
105         printf("%d) %s: %zu steps\n", i + 1, tmp[i].name, tmp[i].steps);
106     }
107 }
108
109 void getTop(TablePlrs db, char *ans)
110 {
111     tableSort(db);
112     strcpy(ans, "USERNAME WINS\tLOSSES\tMIN STEPS");
113     int size = (db->size > 5) ? 5 : db->size;
114     for (int i = 0; i < size; ++i) {
115         sprintf(ans, "%s\n %-16s %d\t %d\t %d", ans, db->players[i].log, db->players[i
                ].wins, db->players[i].loses, db->players[i].min_steps);
116     }
117 }
118
119 int main(int argc, char **argv)
120 {
121     printf("Server start.\n");
122     size_t countOfPlayers;
123     int id[4] = { 0 };
124     char answer[MSG_SIZE];
125     Result table[4];
126     Args game;
127     Args tmp;
128
129     /* CONNECTING */
130     void *context = zmq_ctx_new();
131     if (!context) error("zmq_ctx_new");
132     void *responder = zmq_socket(context, ZMQ_REP);
133
134     int bind = zmq_bind(responder, "tcp://*:5555");
135     if (bind) error("bind");
```

```
136
137     /* CHECK ARGUMENTS FROM INPUT */
138     if (argc == 3 && !strcmp(argv[1],"-p")) {
139         countOfPlayers = atoi(argv[2]);
140     } else {
141         printf(SERVER_USAGE);
142         return 0;
143     }
144     if (!countOfPlayers || (countOfPlayers < 2 && countOfPlayers > 4)) {
145         printf(SERVER_USAGE);
146         return 0;
147     }
148
149     TablePlrs db = tableCreate();
150     FILE *fp = fopen(DB_NAME, "r+");
151     printf("Loading data base...\n");
152     if(tableLoad(&db, fp)) error("db load");
153     printf("Loaded db:\n");
154     tablePrint(db);
155     fseek(fp, 0, SEEK_SET);
156
157     /* INITIALIZATION AND FILL FOXES */
158     fieldCreate(game.matrix);
159     countOfFoxs(game.matrix);
160
161     for (size_t ready = 0; ready < countOfPlayers; ++ready) {
162         zmq_recv(responder, &tmp, sizeof(Args), 0);
163         if (!tmp.players) {
164             printf("User %s connected and start to play\n", tmp.log);
165             tmp.id1 = ready;
166             table[ready].yep = 0;
167             tmp.players = countOfPlayers;
168             copy(&game, &tmp);
169             printf("\t%zu players are ready\n", ready + 1);
170         } else {
171             --ready;
172         }
173         zmq_send(responder, &tmp, sizeof(Args), 0);
174     }
175
176     size_t num = countOfPlayers;
177     printf("Start the game...\n");
178
179     /* WAITING ALL PLAYERS */
180     while (num) {
181         zmq_recv(responder, &tmp, sizeof(Args), 0);
182         if(!id[tmp.id1]) {
183             --num;
184             tmp.status = 1;
```

```
185             id[tmp.id1] = 1;
186         }
187         zmq_send(responder, &tmp, sizeof(Args), 0);
188     }
189
190     printf("All players connected\n");
191
192     for (int i = 0; i < countOfPlayers; ++i) {
193         id[i] = 0;
194     }
195
196     num = countOfPlayers;
197     char timeWinner[SIZE_LOG];
198     int stepGame = 0;
199
200     while(1) {
201         if (!countOfPlayers) {
202             printf("Counting of scores...\n");
203             sort(table, num);
204             printRes(table, num);
205
206             /* STEP MODE -- DB UPDATE */
207             if (stepGame) {
208                 char userWithMinSteps[SIZE_LOG];
209                 strcpy(userWithMinSteps, table[0].name);
210                 int modeUserWithMinSteps = table[0].mode;
211                 int minSteps = table[0].steps;
212                 for (int i = 0; i < num; ++i) {
213                     if (table[i].steps < minSteps) {
214                         strcpy(userWithMinSteps, table[i].name);
215                         modeUserWithMinSteps = table[i].mode;
216                     }
217                 }
218
219                 if (modeUserWithMinSteps == STEP_GAME) {
220                     tableUpdate(db, userWithMinSteps, 1, minSteps);
221                 }
222                 for (int i = 0; i < num; ++i) {
223                     if (strcmp(userWithMinSteps, table[i].name) && table[i].mode ==
                            STEP_GAME) {
224                         tableUpdate(db, table[i].name, 0, table[i].steps);
225                     }
226                 }
227             }
228
229
230             table[0].yep = 1;
231             countOfPlayers = 1;
232             printf("Save data base.\n");
```

```
233             tablePrint(db);
234             tableSave(db, fp);
235             fclose(fp);
236         }
237         zmq_recv(responder, &tmp, sizeof(Args), 0);
238         switch (tmp.status) {
239 
240             /* FINISH GAME */
241             case 0: {
242                 if(!id[tmp.id1]) {
243                     strcpy(table[tmp.id1].name, tmp.log);
244                     table[tmp.id1].steps = tmp.result;
245                     id[tmp.id1] = 1;
246                     table[tmp.id1].mode = tmp.mode;
247                     printf("User %s finished the game\n", tmp.log);
248                     printf("\this result: %zu steps\n", tmp.result);
249                     strcpy(answer, "Your'ar finished. Pls, wait other players");
250                     zmq_send(responder, &answer, sizeof(answer), 0);
251                     --countOfPlayers;
252 
253                     /* TIME MODE -- DB UPDATE */
254                     if (tmp.mode == TIME_GAME) {
255                         if (countOfPlayers + 1 == num) {
256                             tableUpdate(db, tmp.log, 1, tmp.result);
257                             strcpy(timeWinner, tmp.log);
258                         } else {
259                             tableUpdate(db, tmp.log, 0, tmp.result);
260                         }
261                     } else if (tmp.mode == STEP_GAME) {
262                         stepGame = 1;
263                     }
264                 }
265                 break;
266             }
267 
268             /* RESULT GAME */
269             case 1: {
270                 if(countOfPlayers) {
271                     zmq_send(responder, &table, sizeof(table), 0);
272                 } else {
273                     table[0].yep = 1;
274                     zmq_send(responder, &table, sizeof(table), 0);
275                 }
276                 break;
277             }
278 
279             /* PLAYER STATISTICS */
280             case 2: {
281                 Player print;
```

```
282                 print = tableFind(db, tmp.log);
283                 sprintf(answer, "Username: %s\nWins: %d\nLosses: %d\nMinimum steps: %d\n
                        ", print->log, print->wins, print->loses, print->min_steps);
284                 zmq_send(responder, &answer, sizeof(answer), 0);
285                 break;
286             }
287 
288             /* TOP 5 PLAYERS */
289             case 3: {
290                 getTop(db, answer);
291                 zmq_send(responder, &answer, sizeof(answer), 0);
292                 break;
293             }
294 
295             /* EXIT */
296             case 4: {
297                 printf("Player %s disconnect\n", tmp.log);
298                 strcpy(answer, "GoodBuy");
299                 zmq_send(responder, &answer, sizeof(answer), 0);
300                 break;
301             }
302             default: {
303                 tmp.players = 0;
304                 zmq_send(responder, &tmp, sizeof(Args), 0);
305                 break;
306             }
307         }
308 
309     }
310 
311     zmq_close(responder);
312     zmq_ctx_destroy(context);
313     return 0;
314 }
```

8 Тестирование

- Некорректные входные данные

```
server:
karma@karma:~/mai_study/OS/kp$ ./server
Server start.
usage: ./server -p <number of players (from 2 to 4)>
karma@karma:~/mai_study/OS/kp$ ./server -p ve
Server start.
usage: ./server -p <number of players (from 2 to 4)>

player:
karma@karma:~/mai_study/OS/kp$ ./client
usage: ./client -l <login>
```

- Тестирование связи один ко многим.

- Попытка подключиться к серверу, когда к нему подключено уже максимальное количество игроков

```
server:
karma@karma:~/mai_study/OS/kp$ ./server -p 2
Server start.
Loading data base...
Loaded db:
PLAYER  WINS    LOSES   MIN STEPS
karma   16  9   0
nekarma 4   3   0
player1 7   7   0
player2 2   0   0
nastya  0   1   0
lol     2   1   0
kek     0   1   0
lol2    1   1   0
User karma connected and start to play
```

```
1 players are ready
User nekarma connected and start to play
2 players are ready
Start the game...

player1:
karma@karma:~/mai_study/OS/kp$ ./client -l karma
Connect to tcp://localhost:5555
Select game mode:
1) time
2) minimum steps
1
Wait...
Enter the coordinates (x & y),pls: (from 1 to 10)
x =

player2:
karma@karma:~/mai_study/OS/kp$ ./client -l nekarma
Connect to tcp://localhost:5555
Select game mode:
1) time
2) minimum steps
1
Wait...
Enter the coordinates (x & y),pls: (from 1 to 10)
x =

player3:
karma@karma:~/mai_study/OS/kp$ ./client -l lol
Connect to tcp://localhost:5555
There are no more places
```

- Некорректный ввод координат

```
player1:
Enter the coordinates (x & y),pls: (from 1 to 10)
x = лул
Incorrect coordinate
Enter the coordinates (x & y),pls: (from 1 to 10)
```

```
x = 3
y = -1
Pls,from 1 to 10.
```

- Два игрока в режиме времени. Ввод координаты -1 завершает ловлю лис и сразу отправляет к получению реузльтата. Это сделано для облегчения тестирования.

```
player1:
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 1
y = 2

Oh,fox is not here

x 4 x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x

Have to catch 8 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 5
y = 5

Grats,you caught 1 fox(es).

x 4 x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x F x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

```
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x

Have to catch 7 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = -1
Hey admin!!!
Congratulations,you catch all the foxes.
||  /  /
.__.. _/
____/_ __/
/_/_/

Your'ar finished. Pls,wait other players
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit


player2:
Enter the coordinates (x & y),pls: (from 1 to 10)
x = -1
Hey admin!!!
Congratulations,you catch all the foxes.
||  /  /
.__.. _/
____/_ __/
/_/_/

Your'ar finished. Pls,wait other players
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit

server:
User nekarma finished the game
his result: 0 steps
User karma finished the game
```

```
his result: 2 steps
Counting of scores...
1) nekarma: 0 steps
2) karma: 2 steps
Save data base.
PLAYER  WINS    LOSES   MIN STEPS
karma   16  10  0
nekarma 5   3   0
player1 7   7   0
player2 2   0   0
nastya  0   1   0
lol     2   1   0
kek     0   1   0
lol2    1   1   0
```

Видим, что число побед выгрывшего игрока увеличилось на 1, а число проигрышей проигравшего игрока увеличилось на 1.

- Получение результатов игры каждым из клиентов

```
player1:
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
1
1) nekarma: 0 steps
2) karma: 2 steps

player2:
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
1
1) nekarma: 0 steps
2) karma: 2 steps
```

- Получение личной статистики каждым из клиентов

```
player1:
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
2
Username: karma
Wins: 16
Losses: 10
Minimum steps: 0

player2:
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
2
Username: nekarma
Wins: 5
Losses: 3
Minimum steps: 0
```

- Получение топ-5

```
player1:
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
3
USERNAME        WINS    LOSSES  MIN STEPS
karma            16     10       0
player1          7       7       0
nekarma          5       3       0
player2          2       0       0
lol              2       1       0
```

- Выход обоих игроков

```
player1:
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
4
GoodBuy

player2:
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
4
GoodBuy

server:
Player karma disconnect
Player nekarma disconnect
```

- Тестирование режима минимального количества ходов
  Сценарий работы такой: подключаются два игрока с режимом количества ходов. Первый игрок делает 3 хода, второй 0 ходов. Смотрим, как изменились количество их побед и поражений в базе данных.

```
player1:
karma@karma:~/mai_study/OS/kp$ ./client -l karma
Connect to tcp://localhost:5555
Select game mode:
1) time
2) minimum steps
2
Wait...
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 1
y = 1

Oh,fox is not here
```

```
3 x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

```
Have to catch 8 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 5
y = 5

Oh,fox is not here
```

```
3 x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x 3 x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

```
Have to catch 8 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 10
y = 10

Oh,fox is not here
```

```
3 x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x 3 x x x x x
```

```
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x 3

Have to catch 8 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = -1
Hey admin!!!
Congratulations,you catch all the foxes.
||  /  /
.__.. _/
____/_ __/
/_/_/

Your'ar finished. Pls,wait other players
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
2
Username: karma
Wins: 16
Losses: 11
Minimum steps: 0

1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
3
USERNAME        WINS    LOSSES  MIN STEPS
karma            16     11  0
player1          7  7   0
nekarma          6  3   0
player2          2  0   0
lol              2  1   0
1) Result this game
2) My stats
3) Top-5 for all time
```

```
4) Exit
4
GoodBuy


player2:
karma@karma:~/mai_study/OS/kp$ ./client -l nekarma
Connect to tcp://localhost:5555
Select game mode:
1) time
2) minimum steps
2
Wait...
Enter the coordinates (x & y),pls: (from 1 to 10)
x = -1
Hey admin!!!
Congratulations,you catch all the foxes.
||  /  /
.__.. _/
____/_ __/
/_/_/

Your'ar finished. Pls,wait other players
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
2
Username: nekarma
Wins: 6
Losses: 3
Minimum steps: 0

1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
3
USERNAME        WINS    LOSSES  MIN STEPS
karma            16     11  0
player1          7  7   0
```

```
nekarma          6  3   0
player2          2  0   0
lol              2  1   0
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
4
GoodBuy

server:
karma@karma:~/mai_study/OS/kp$ ./server -p 2
Server start.
Loading data base...
Loaded db:
PLAYER  WINS    LOSES   MIN STEPS
karma   16  10  0
nekarma 5   3   0
player1 7   7   0
player2 2   0   0
nastya  0   1   0
lol     2   1   0
kek     0   1   0
lol2    1   1   0
User karma connected and start to play
1 players are ready
User nekarma connected and start to play
2 players are ready
Start the game...
All players connected
User karma finished the game
his result: 3 steps
User nekarma finished the game
his result: 0 steps
Counting of scores...
1) nekarma: 0 steps
2) karma: 3 steps
Save data base.
PLAYER  WINS    LOSES   MIN STEPS
karma   16  11  0
nekarma 6   3   0
```

```
player1 7   7   0
player2 2   0   0
nastya  0   1   0
lol     2   1   0
kek     0   1   0
lol2    1   1   0
Player karma disconnect
Player nekarma disconnect
```

- Тестирование, когда один из игроков играет в режиме времени, а другой – в режиме ходов.
  Сценарий работы такой: подключаются два игрока, один с режимом времени, другой с режимом ходов. Первый игрок делает 3 хода и заканчивает игру первым, второй делает 0 ходов. Смотрим, как изменились количество их побед и поражений в базе данных.

```
player1:
karma@karma:~/mai_study/OS/kp$ ./client -l karma
Connect to tcp://localhost:5555
Select game mode:
1) time
2) minimum steps
1
Wait...
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 1
y = 2

Oh,fox is not here

x 2 x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

```
x x x x x x x x x x

Have to catch 8 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 2
y = 3

Oh,fox is not here

x 2 x x x x x x x x
x x 2 x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x

Have to catch 8 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = 3
y = 4

Oh,fox is not here

x 2 x x x x x x x x
x x 2 x x x x x x x
x x x 2 x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x

Have to catch 8 fox(es)
Enter the coordinates (x & y),pls: (from 1 to 10)
x = -1
```

```
Hey admin!!!
Congratulations,you catch all the foxes.
||  /  /
.__.. _/
____/_ __/
/_/_/

Your'ar finished. Pls,wait other players
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
3
USERNAME        WINS    LOSSES  MIN STEPS
karma            17     11  0
nekarma          7  3   0
player1          7  7   0
player2          2  0   0
lol              2  1   0
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
4
GoodBuy


player2:
karma@karma:~/mai_study/OS/kp$ ./client -l nekarma
Connect to tcp://localhost:5555
Select game mode:
1) time
2) minimum steps
2
Wait...
Enter the coordinates (x & y),pls: (from 1 to 10)
x = -1
Hey admin!!!
Congratulations,you catch all the foxes.
||  /  /
.__.. _/
```

```
____/_ __/
/_/_/

Your'ar finished. Pls,wait other players
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
3
USERNAME        WINS    LOSSES  MIN STEPS
karma            17     11  0
nekarma          7  3   0
player1          7  7   0
player2          2  0   0
lol              2  1   0
1) Result this game
2) My stats
3) Top-5 for all time
4) Exit
4
GoodBuy


server:
karma@karma:~/mai_study/OS/kp$ ./server -p 2
Server start.
Loading data base...
Loaded db:
PLAYER  WINS    LOSES   MIN STEPS
karma   16  11  0
nekarma 6   3   0
player1 7   7   0
player2 2   0   0
nastya  0   1   0
lol     2   1   0
kek     0   1   0
lol2    1   1   0
User karma connected and start to play
1 players are ready
User nekarma connected and start to play
2 players are ready
```

```
Start the game...
All players connected
User karma finished the game
his result: 3 steps
User nekarma finished the game
his result: 0 steps
Counting of scores...
1) nekarma: 0 steps
2) karma: 3 steps
Save data base.
PLAYER  WINS    LOSES   MIN STEPS
karma   17  11  0
nekarma 7   3   0
player1 7   7   0
player2 2   0   0
nastya  0   1   0
lol     2   1   0
kek     0   1   0
lol2    1   1   0
Player karma disconnect
Player nekarma disconnect
```

9 Выводы

Мой курсовой проект вобрал в себя все те навыки, которым я научилась в течение курса. Разработка таких программ требует внимательного отношения к деталям еще на этапе проектирования. Неразумный выбор паттерна передачи сообщений между клиентами и сервером, структуры данных для хранения базы игроков или неправильная логика самой игры могут привести к тому, что при столкновении со сложностями при дальнейшей разработке программы придется переписывать всю программу чуть ли не с нуля. К счастью, знания и навыки, полученные в ходе курса, помогли мне всего этого избежать.

Все в этом мире несовершенно. И моя игра не исключение. Какие недочеты есть в моей программе?
1. Несколько игроков могут зайти под одним и тем же логином. Это плохо, потому что статистика изменится несколько раз.
2. Режим времени, который может выбрать игрок, был реализован в связи с дальнейшеми планами перейти от игры на одной машине к игре с нескольких машин. Планы так и остались планами, но концепция уже заложена. Надо доработать.
3. Искусственное ограничение на количество игроков, с одной стороны, освобождает от необходимости ускорять работу моей программы, а с другой, лишает возможности реально многопользовательской игры. Здесь разработчик чем-то схож с первобытным человеком, который при счете использовал такие понятия как один, два, много. Но в условии КП сказано, разработать программу для более чем одного клиента, и оно выполнено.

Но не все так плохо. Программа отлично работает для небольшого количества клиентов на одной машине. Не падает при некорректных входных данных. Наличие статистики придает дополнительный интерес. Думаю, немного доработав эту программу, мы с моими коллегами сможем играть в Охоту на лис в редких перерывах между выполнением лабораторных работ других курсов.

База данных реализована на векторе. Вставка осуществляется за константу, а вот поиск за линию. С одной стороны, при подключении каждого нового игрока мы выигрываем, а с другой, если каждый игрок будет запрашивать свою персональную статистику, мы неоправданно долго будем его искать. Сортировка базы данных и таблицы результатов выполняется за квадрат в худшем случае. Ужасающая сложность. Высоконагруженные системы такого не прощают. Но так как я искусственно создала максимальное ограничение на количество игроков (равное 4), то сложность приемлемая. Сортировка 4 элементов практически никак нам не помешает. При каждом выводе топ-5 рейтинга игроков записи в БД сортируются. Если предположить,

что хотя бы один из игроков захочет посмотреть рейтинг, то БД будет сортироваться 1 раз в игру. Количество побед, по которым происходит сортировка, меняется за игру максимально на 1 для игроков, выигравших в своих режимах. То есть, отсортировав БД в первый раз за все время в худшем случае за квадрат, при следующих сортировках мы затратим на это приближенно линейное время. В масштабах курсового проекта это очень даже неплохо. Из существенных минусов БД могу отметить небинарную запись в файл. Это несколько замедляет запись и чтение. Но замедление записи некритично, потому что происходит после того, как все игроки покинут игру.

Подводя итог всему курсу, хочу отметить, что мне было достаточно интересно выполнять лабораторные. Огромное количество тесткейсов в каждой лабораторной научило меня продумывать каждое свое действие еще на этапе написания кода, что существенно сократило время отладки. Лекции, на удивление, тоже были хороши, что для меня большая редкость. Мне не хватило лабораторных работ, похожих на 4, 6-8 и курсовой проект, где мы могли бы проектировать прототипы относительно больших систем. Также мне кажется, операционные системы это именно тот курс, где можно было бы попробовать себя в чем-то более низкоуровневом, чем все, что было до него. Возможно сделать какие-нибудь ассемблерные вставки в наш сишный код или пописать шелл-коды. На мой взгляд, лабораторные 1 и 5 совершенно бессмыслены. С системными вызовами мы в любом случае знакомимся при выполнении каждой лабораторной этого курса, а использование средств диагностики можно было бы перенести в любую другую л/р. 5 лабораторная почти дублирует 25-26 лабораторную 1 курса. Жаль, что этот курс семестровый, думаю системные программисты, на которых мы вроде бы учимся, могли бы поизучать ОС подольше.