Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

КУРСОВОЙ ПРОЕКТ

по дисциплине “Отказоустойчивы вычислительные системы”

на тему:

Разработка сетевого приложения

Выполнила:

Студентка гр. ИС-641

Бердникова А.М.

Проверил:

к.т.н, доцент

Майданов Ю.С.

Новосибирск 2020

**Оглавление**

Постановка задачи3

Общие сведения4

Описание работы6

Заключение11

Листинг12

ПОСТАНОВКА ЗАДАЧИ

Реализовать сетевую игру «Крестики – нолики» (3x3), основанную на клиент-серверном взаимодействии.

Клиент – приложение использующие графический интерфейс для взаимодействия с пользователем. Сервер – приложение, реализующее многопользовательскую серверную логику игры для некоторого количества игроков с использованием технологий отказоустойчивости.

ОБЩИЕ СВЕДЕНИЯ

1.Модель Клиент-Сервер

Модели клиент-сервер - это технология взаимодействия в информационной сети. Сервер обладает правом управления тем или иным ресурсом, а клиент – пользования им. Каждый конкретный сервер определяется видом того ресурса, которым он владеет. Например, назначением сервера баз данных является обслуживание запросов клиентов, связанных с обработкой данных; файловый сервер, или файл-сервер, распоряжается файловой системой.

Этот принцип распространяется и на взаимодействие программ. Программа, выполняющая предоставление соответствующего набора услуг, рассматривается в качестве сервера, а программы, пользующиеся этими услугами, принято называть клиентами. Программы имеют распределенный характер, т.е. одна часть функций прикладной программы реализуется в программе-клиенте, а другая - в программе-сервере, а для их взаимодействия определяется некоторый протокол.

Преимущества:

* Отсутствие дублирования кода программы-сервера программами-клиентами;
* Снижение требований к производительности компьютера на котором установлен клиент, за счет переноса нагрузки на сервер;
* Повышение безопасности хранения данных, так как хранятся на серверах.

Недостатки:

* Высокая стоимость оборудования;
* Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть. (Нивелируется внедрение технологий отказоустойчивости).

ОПИСАНИЕ РАБОТЫ

В процессе выполнения работы сервером ведется лог выполненных действий, что полезно при анализе отказоустойчивости.

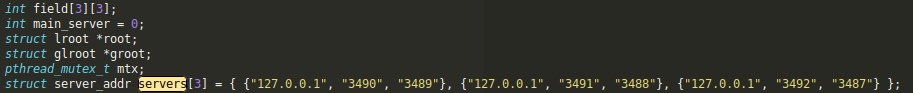


Рис.1. Сервера

В коде определены три сервера (рис. 1), где каждый сервер описывается структурой (рис. 2):



Рис. 2 Структура адреса сервера

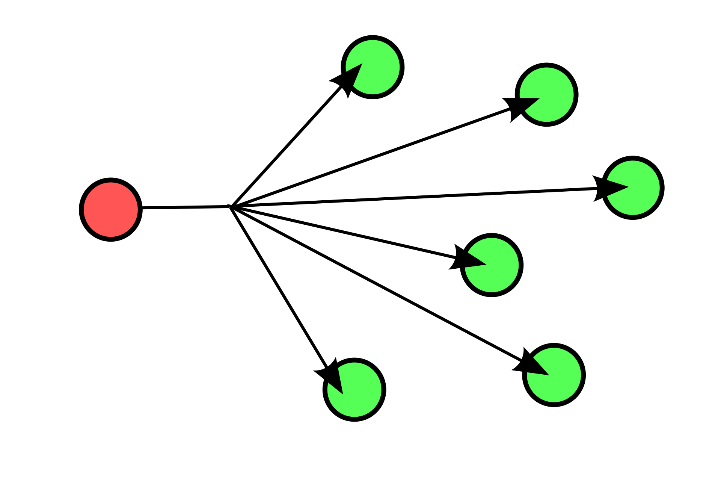


Рис. 3 Схема работы метода передачи данных.

Между активными серверами идет обмен информацией, в том числе и состоянием игры, используя алгоритм трансляции(broadcasting), рисунок 3. Проверка доступности серверов и игроков происходит в бесконечном цикле, пока идет игра на каждой итерации цикла проверяется доступность серверов и игроков.

С использование библиотеки pthread.h, сервер выделяет для каждой игры отдельный поток, который управляет игрой. Для избегания гонки данных, используется примитив синхронизации mutex, который позволяет реализовать доступ к общим данным.

Продемонстрируем как происходит многопоточное выполнение алгоритма серверов для этого узнаем PID запущенных серверов:

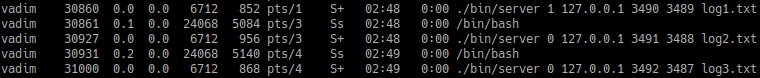


Рис. 4 PID-ы запущенных серверов

На рисунке 5 расположенным ниже изображен bash – скрипт для мониторинга многопоточности процесса:

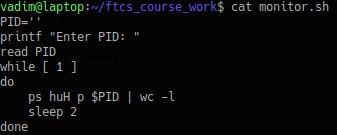


Рис. 5 Bash – скрипт

При создании игры подключении участников процесс сервер организует поток для игры:

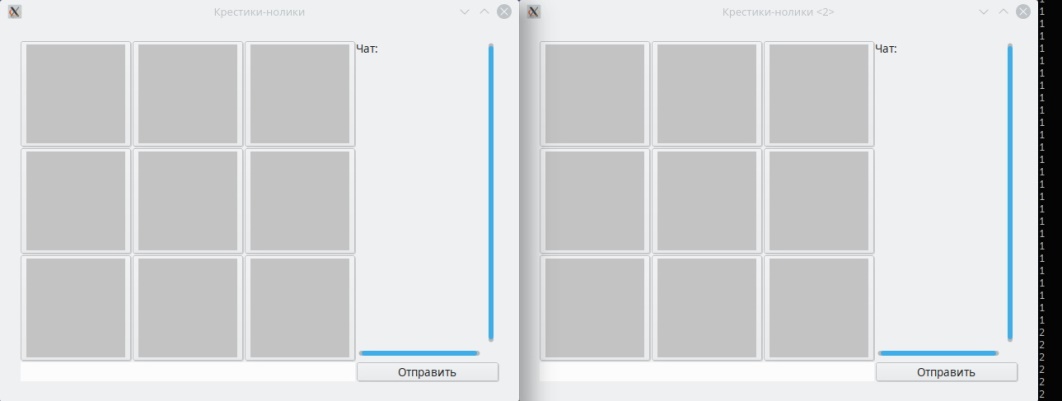


Рис. 6 Два окна клиента, справа работа скрипта.

Интерфейс игры состоит из игрового поля размера 3 на 3 ячейки, справа расположен игровой чат, игрок, который создает игру выбирает сторону за которую он будет играть, второй игрок играет за противоположную сторону:

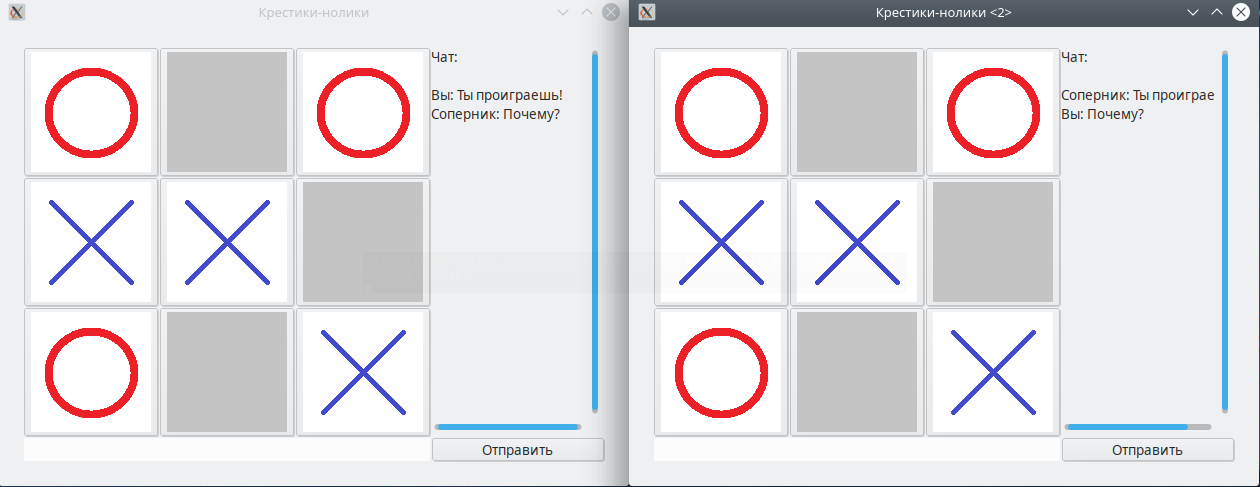


Рис. 7 Интерфейс игры

После окончания игроку выводится информация о победе, проигрыше или ничьей:

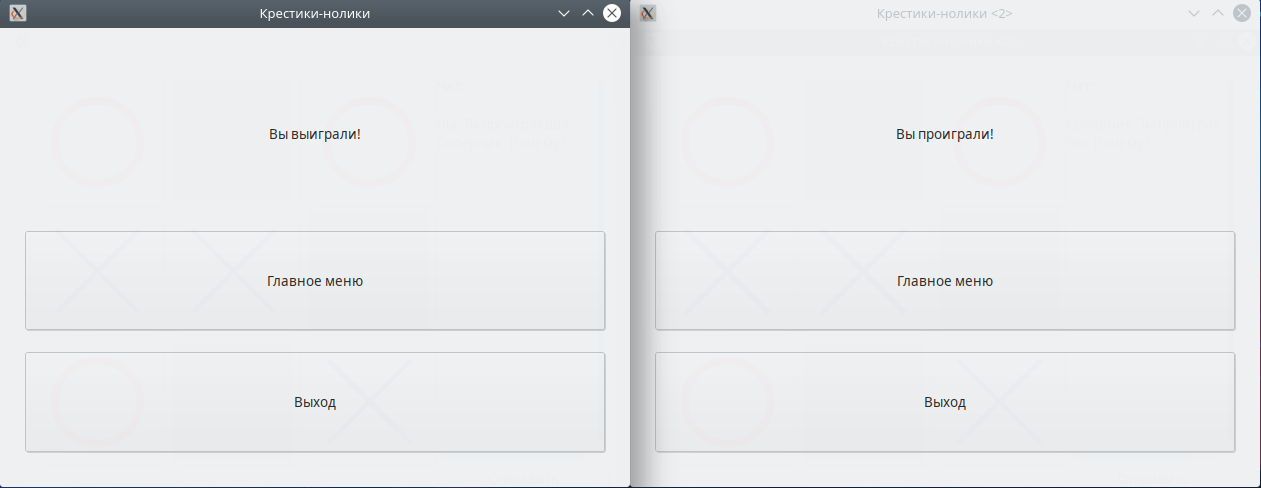


Рис. 8 Информация по окончанию игры

Продемонстрируем реализованную отказоустойчивость, для примера игра будет вестись на трех серверах, один из который является главным, а два дополнительных резервные:

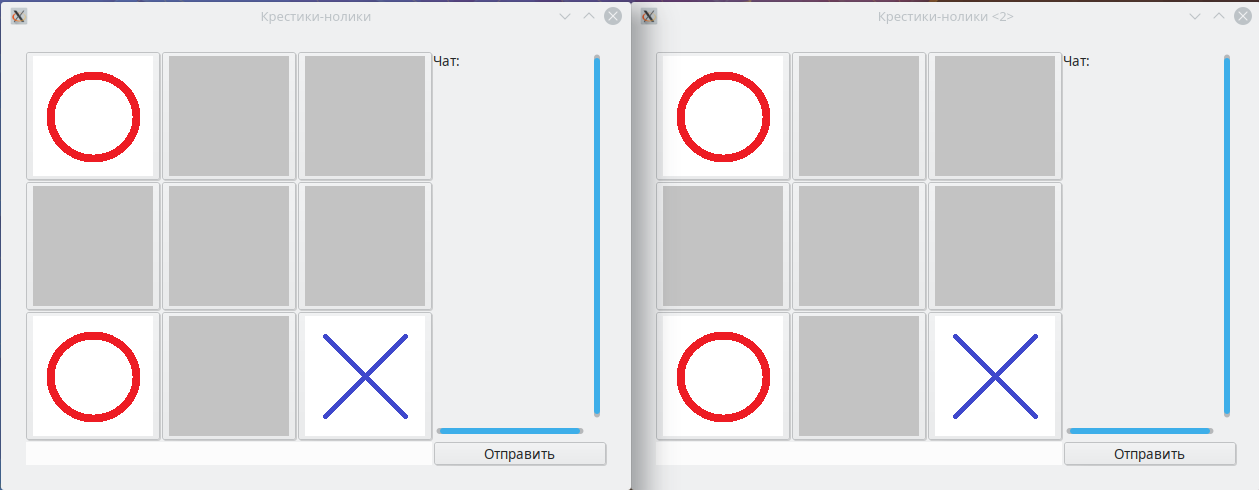


Рис.9 Начальное состояние полей игроков

Прервем выполнение главного сервера и прервем выполнение одного дополнительного сервера:

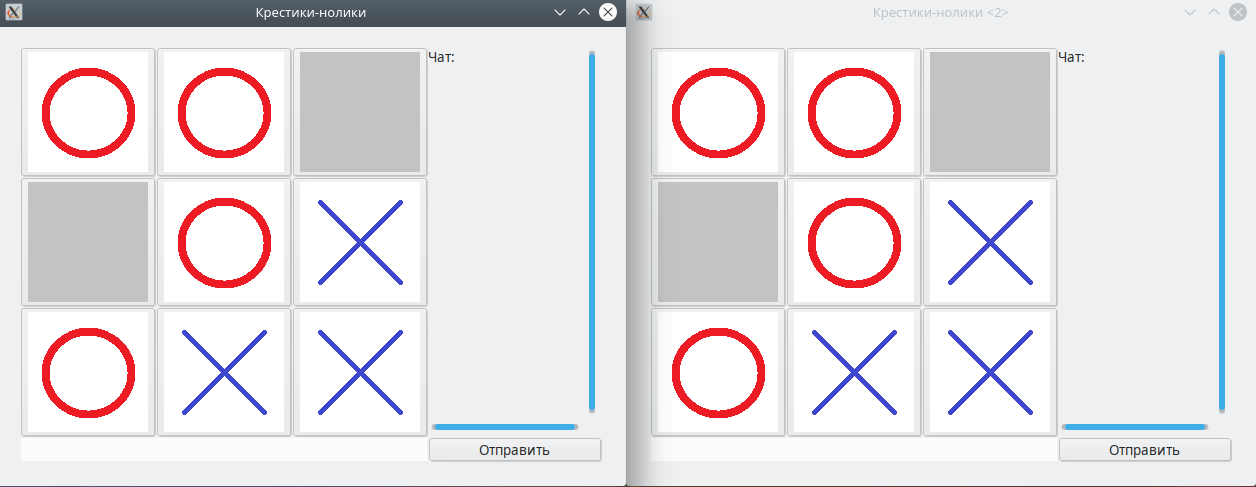


Рис. 10 Продолжение игры при одном доступном сервере

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была спроектирована и реализована схема отказоустойчивой работы функционирования клиент серверного взаимодействия.

Опытным путем была показана работоспособность данной схемы.

ПРИЛОЖЕНИЕ

Server.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/time.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <sys/wait.h>

#include <signal.h>

#include <pthread.h>

#include <time.h>

#include "../include/llist.h"

#include "../include/glist.h"

#define GAME\_PORT "3490"

#define CHECK\_PORT "3489"

#define BACKLOG 10

#define MAXDATASIZE 100

#define MSG\_LEN 64

#define SERV\_CNT 2

struct server\_addr {

char ip[16];

char port[6];

char chk\_port[6];

};

struct args {

int fd;

int chk\_fd;

int id;

char \*ip;

char \*port;

};

int field[3][3];

int main\_server = 0;

struct lroot \*root;

struct glroot \*groot;

pthread\_mutex\_t mtx;

struct server\_addr servers[3] = {{"127.0.0.1", "3490", "3489"},

{"127.0.0.1", "3491", "3488"},

{"127.0.0.1", "3492", "3487"}};

FILE \*logfd;

int check\_field() {

int res = 0, i, j;

if ((field[0][0] == field[1][0]) && (field[0][0] == field[2][0]) && (field[0][0] != 0))

res = 1;

if ((field[0][1] == field[1][1]) && (field[0][1] == field[2][1]) && (field[0][1] != 0))

res = 1;

if ((field[0][2] == field[1][2]) && (field[0][2] == field[2][2]) && (field[0][2] != 0))

res = 1;

if ((field[0][0] == field[0][1]) && (field[0][0] == field[0][2]) && (field[0][0] != 0))

res = 1;

if ((field[1][0] == field[1][1]) && (field[1][0] == field[1][2]) && (field[1][0] != 0))

res = 1;

if ((field[2][0] == field[2][1]) && (field[2][0] == field[2][2]) && (field[2][0] != 0))

res = 1;

if ((field[0][0] == field[1][1]) && (field[0][0] == field[2][2]) && (field[0][0] != 0))

res = 1;

if ((field[2][0] == field[1][1]) && (field[2][0] == field[0][2]) && (field[2][0] != 0))

res = 1;

if (res != 1) {

int fl = 0;

for (i = 0; i < 3; i++)

for (j = 0; j < 3; j++)

if (field[i][j] == 0)

fl = 1;

if (fl == 0) res = 2;

}

return res;

}

void sigchld\_handler(int s) {

while (waitpid(-1, NULL, WNOHANG) > 0);

}

void \*get\_in\_addr(struct sockaddr \*sa) {

if (sa->sa\_family == AF\_INET) {

return &(((struct sockaddr\_in \*) sa)->sin\_addr);

}

return &(((struct sockaddr\_in6 \*) sa)->sin6\_addr);

}

double wtime() {

struct timeval t;

gettimeofday(&t, NULL);

return (double) t.tv\_sec + (double) t.tv\_usec \* 1E-6;

}

void server\_log(char \*str1, int flg, const char \*str2) {

pthread\_mutex\_lock(&mtx);

time\_t t = time(NULL);

struct tm \*tm = localtime(&t);

char s[64];

strftime(s, sizeof(s), "%c", tm);

fprintf(logfd, "[%s] ", s);

if (flg == 1)

fprintf(logfd, "%s\n", str1);

else if (flg == 0)

fprintf(logfd, "%s\n", str1);

else

fprintf(logfd, str1, str2);

fflush(logfd);

pthread\_mutex\_unlock(&mtx);

}

void send\_to\_client(int \*sockfd, char \*message) {

if (send(\*sockfd, message, MSG\_LEN, 0) == -1)

server\_log("server thread: send", 1, NULL);

}

void send\_to\_reserve(int \*active\_srv, char \*message) {

int i;

for (i = 0; i < SERV\_CNT; i++) {

if (active\_srv[i] > 0) {

if (send(active\_srv[i], message, MSG\_LEN, 0) == -1)

server\_log("server thread: reserve send", 1, NULL);

}

}

}

void game\_process(int op1, int op1\_chk, int op2, int op2\_chk, int \*active\_srv) {

int fdmax, fdmax\_chk, command, amnt, i, j, ij, chk;

char message[MSG\_LEN];

fd\_set read\_fds, master, read\_fds\_chk, master\_chk;

struct timeval tv;

tv.tv\_sec = 0;

tv.tv\_usec = 0;

FD\_ZERO(&read\_fds);

FD\_ZERO(&master);

FD\_ZERO(&read\_fds\_chk);

FD\_ZERO(&master\_chk);

if (op1\_chk > op2\_chk) fdmax\_chk = op1\_chk;

else fdmax\_chk = op2\_chk;

FD\_SET(op1\_chk, &master\_chk);

FD\_SET(op2\_chk, &master\_chk);

if (op1 > op2) fdmax = op1;

else fdmax = op2;

FD\_SET(op1, &master);

FD\_SET(op2, &master);

while (1) {

read\_fds\_chk = master\_chk;

if (select(fdmax\_chk + 1, &read\_fds\_chk, NULL, NULL, &tv) == -1) {

server\_log("server thread: select", 1, NULL);

break;

}

if (FD\_ISSET(op1\_chk, &read\_fds\_chk)) {

memset(message, 0, MSG\_LEN);

if ((amnt = recv(op1\_chk, message, MSG\_LEN, 0)) <= 0) {

server\_log("server thread: recv", 1, NULL);

message[0] = 66;

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op2, message);

break;

} else {

command = message[0];

if (command == 100) {

send\_to\_client(&op1\_chk, message);

memset(message, 0, MSG\_LEN);

}

}

}

if (FD\_ISSET(op2\_chk, &read\_fds\_chk)) {

memset(message, 0, MSG\_LEN);

if ((amnt = recv(op2\_chk, message, MSG\_LEN, 0)) <= 0) {

server\_log("server thread: recv", 1, NULL);

message[0] = 66;

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op1, message);

break;

} else {

command = message[0];

if (command == 100) {

send\_to\_client(&op2\_chk, message);

memset(message, 0, MSG\_LEN);

}

}

}

read\_fds = master;

if (select(fdmax + 1, &read\_fds, NULL, NULL, &tv) == -1) {

server\_log("server thread: select", 1, NULL);

break;

}

if (FD\_ISSET(op1, &read\_fds)) {

memset(message, 0, MSG\_LEN);

if ((amnt = recv(op1, message, MSG\_LEN, 0)) <= 0) {

server\_log("server thread: recv", 1, NULL);

} else {

command = message[0];

if (command == 10) {

ij = (int) message[1];

i = ij / 10;

j = ij % 10;

field[i][j] = 1;

message[2] = 1;

send\_to\_client(&op2, message);

send\_to\_reserve(active\_srv, message);

memset(message, 0, MSG\_LEN);

chk = check\_field();

if (chk == 1) {

message[0] = 20;

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op1, message);

send\_to\_client(&op2, message);

break;

} else {

if (chk == 2) //ничья

{

message[0] = 40;

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op1, message);

send\_to\_client(&op2, message);

break;

} else { // продолжение игры

message[0] = 50;

send\_to\_client(&op1, message);

send\_to\_client(&op2, message);

}

}

} else if (command == 66) {

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op2, message);

break;

} else {

send\_to\_client(&op2, message);

memset(message, 0, MSG\_LEN);

}

}

}

if (FD\_ISSET(op2, &read\_fds)) {

memset(message, 0, MSG\_LEN);

if ((amnt = recv(op2, message, MSG\_LEN, 0)) <= 0) {

server\_log("server thread: recv", 1, NULL);

} else {

command = message[0];

if (command == 10) {

ij = (int) message[1];

i = ij / 10;

j = ij % 10;

field[i][j] = 2;

message[2] = 2;

send\_to\_client(&op1, message);

send\_to\_reserve(active\_srv, message);

memset(message, 0, MSG\_LEN);

chk = check\_field();

if (chk == 1) {

message[0] = 30;

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op1, message);

send\_to\_client(&op2, message);

break;

} else {

if (chk == 2) {

message[0] = 40;

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op1, message);

send\_to\_client(&op2, message);

break;

} else {

message[0] = 50;

send\_to\_client(&op1, message);

send\_to\_client(&op2, message);

}

}

} else if (command == 66) {

send\_to\_reserve(active\_srv, message);

send\_to\_client(&op1, message);

break;

} else {

send\_to\_client(&op1, message);

memset(message, 0, MSG\_LEN);

}

}

}

}

}

void \*game\_thread(void \*args) {

int new\_fd, chk\_new\_fd, amnt, op1, op1\_chk, op2, op2\_chk, command, side;

int i, j, ij, id, cnt = 0, tsock, rv;

int active\_srv[SERV\_CNT] = {-1};

char message[MSG\_LEN];

char \*ip, \*port;

fd\_set read\_fds, master;

struct addrinfo hints, \*servinfo = NULL, \*p;

struct list \*ptrlist;

struct glist \*gptrlist;

struct timeval tv;

tv.tv\_sec = 0;

tv.tv\_usec = 0;

FD\_ZERO(&read\_fds);

FD\_ZERO(&master);

struct args \*tmp = (struct args \*) args;

new\_fd = tmp->fd;

chk\_new\_fd = tmp->chk\_fd;

ip = tmp->ip;

port = tmp->port;

id = tmp->id;

memset(message, 0, MSG\_LEN);

message[0] = 100;

if ((amnt = recv(new\_fd, message, MSG\_LEN, 0)) <= 0) {

server\_log("server thread: recv", 0, NULL);

pthread\_exit(0);

}

command = (int) message[0];

if ((command == 1) || (command == 3)) {

message[amnt] = '\0';

if (command == 1) side = 1;

else side = 2;

pthread\_mutex\_lock(&mtx);

addelem(root, new\_fd, chk\_new\_fd, side, message + 4);

pthread\_mutex\_unlock(&mtx);

} else if (command == 5) {

server\_log("reserve server: adding game\n", 0, NULL);

gptrlist = glistfind(groot, message[1]);

if (gptrlist == NULL) {

pthread\_mutex\_lock(&mtx);

gptrlist = gaddelem(groot, message[1], -1);

pthread\_mutex\_unlock(&mtx);

}

FD\_SET(new\_fd, &master);

while (1) {

read\_fds = master;

if (select(new\_fd + 1, &read\_fds, NULL, NULL, &tv) == -1) {

server\_log("reserve server: select", 1, NULL);

}

if (FD\_ISSET(new\_fd, &read\_fds)) {

if ((amnt = recv(new\_fd, message, MSG\_LEN, 0)) <= 0) {

server\_log("reserve server: recv", 1, NULL);

break;

} else {

command = message[0];

if (command == 10) {

ij = (int) message[1];

i = ij / 10;

j = ij % 10;

gptrlist->field[i][j] = message[2];

} else if (command == 66 || command == 20 || command == 30 || command == 40) {

pthread\_mutex\_lock(&mtx);

gdeletelem(gptrlist, groot);

pthread\_mutex\_unlock(&mtx);

close(new\_fd);

break;

}

}

}

}

} else if (command == 7) {

pthread\_mutex\_lock(&mtx);

if (main\_server == 0) {

main\_server = 1;

}

pthread\_mutex\_unlock(&mtx);

gptrlist = glistfind(groot, message[1]);

pthread\_mutex\_lock(&mtx);

if (gptrlist->flg == 0) {

gptrlist->flg = 1;

gptrlist->chk\_fd = chk\_new\_fd;

gptrlist->fd = new\_fd;

pthread\_mutex\_unlock(&mtx);

} else {

pthread\_mutex\_unlock(&mtx);

if (message[2] == 1) {

op2 = gptrlist->fd;

op2\_chk = gptrlist->chk\_fd;

op1 = new\_fd;

op1\_chk = chk\_new\_fd;

} else {

op1 = gptrlist->fd;

op1\_chk = gptrlist->chk\_fd;

op2 = new\_fd;

op2\_chk = chk\_new\_fd;

}

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

field[i][j] = gptrlist->field[i][j];

}

}

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE;

for (i = 0; cnt < SERV\_CNT && i <= SERV\_CNT; i++) {

if ((strcmp(servers[i].ip, ip) != 0) || (strcmp(servers[i].port, port) != 0)) {

if ((rv = getaddrinfo(servers[i].ip, servers[i].port, &hints, &servinfo)) != 0) {

server\_log("getaddrinfo: %s\n", 2, gai\_strerror(rv));

}

for (p = servinfo; p != NULL; p = p->ai\_next) {

if ((active\_srv[cnt] = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol)) == -1) {

server\_log("server thread: sync socket", 1, NULL);

continue;

}

if (connect(active\_srv[cnt], p->ai\_addr, p->ai\_addrlen) == -1) {

close(active\_srv[cnt]);

server\_log("server thread: sync connect", 1, NULL);

continue;

}

if ((rv = getaddrinfo(servers[i].ip, servers[i].chk\_port, &hints, &servinfo)) != 0) {

server\_log("getaddrinfo: %s\n", 2, gai\_strerror(rv));

}

for (p = servinfo; p != NULL; p = p->ai\_next) {

if ((tsock = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol)) == -1) {

server\_log("server thread: sync check socket", 1, NULL);

continue;

}

if (connect(tsock, p->ai\_addr, p->ai\_addrlen) == -1) {

close(tsock);

server\_log("server thread: sync check connect", 1, NULL);

continue;

}

break;

}

close(tsock);

message[0] = 5;

message[1] = gptrlist->number;

if (send(active\_srv[cnt], message, MSG\_LEN, 0) == -1) {

server\_log("server thread: sync send", 1, NULL);

continue;

}

cnt++;

}

}

}

pthread\_mutex\_lock(&mtx);

gdeletelem(gptrlist, groot);

pthread\_mutex\_unlock(&mtx);

game\_process(op1, op1\_chk, op2, op2\_chk, active\_srv);

close(op1);

close(op2);

close(op1\_chk);

close(op2\_chk);

}

} else {

if (root->count != 0) {

ptrlist = root->first\_node;

message[0] = 2;

message[1] = root->count;

if (send(new\_fd, message, MSG\_LEN, 0) == -1) {

server\_log("server thread: send", 1, NULL);

}

for (i = 0; i < root->count; i++) {

if (send(new\_fd, ptrlist->name, strlen(ptrlist->name), 0) == -1) {

server\_log("server thread: send", 1, NULL);

pthread\_exit(0);

}

ptrlist = ptrlist->ptr;

if ((amnt = recv(new\_fd, message, MSG\_LEN, 0)) == -1) {

server\_log("server thread: recv", 1, NULL);

pthread\_exit(0);

}

}

memset(message, 0, MSG\_LEN);

if ((amnt = recv(new\_fd, message, MSG\_LEN, 0)) == -1) { //получение названия игры

server\_log("server thread: recv", 1, NULL);

pthread\_exit(0);

}

if (message[0] != 2) {

printf("server thread: connection refused\n");

pthread\_exit(0);

}

message[amnt] = '\0';

ptrlist = listfind(root, message + 4);

if (ptrlist == NULL) {

message[0] = 61;

if (send(new\_fd, message, MSG\_LEN, 0) == -1) {

server\_log("server thread: send", 1, NULL);

}

pthread\_exit(0);

}

memset(message, 0, MSG\_LEN);

if (ptrlist->sd == 1)

message[0] = 1;

else

message[0] = 3;

message[1] = id;

if (send(new\_fd, message, MSG\_LEN, 0) == -1) {

server\_log("server thread: send", 1, NULL);

}

memset(message, 0, MSG\_LEN);

message[0] = 100;

message[1] = id;

if (send(ptrlist->fd, message, MSG\_LEN, 0) == -1) { //отправка ждущему игроку

server\_log("server thread: send", 1, NULL);

}

memset(message, 0, MSG\_LEN);

if ((amnt = recv(ptrlist->fd, message, MSG\_LEN, 0)) <= 0) {

server\_log("server thread: recv", 1, NULL);

message[0] = 66;

if (send(new\_fd, message, MSG\_LEN, 0) == -1)

server\_log("server thread: send", 1, NULL);

pthread\_mutex\_lock(&mtx);

deletelem(ptrlist, root);

pthread\_mutex\_unlock(&mtx);

pthread\_exit(0);

}

op1 = ptrlist->fd;

op1\_chk = ptrlist->chk\_fd;

op2 = new\_fd;

op2\_chk = chk\_new\_fd;

pthread\_mutex\_lock(&mtx);

deletelem(ptrlist, root);

pthread\_mutex\_unlock(&mtx);

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

field[i][j] = 0;

}

}

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE;

for (i = 0; cnt < SERV\_CNT && i <= SERV\_CNT; i++) {

if ((strcmp(servers[i].ip, ip) != 0) || (strcmp(servers[i].port, port) != 0)) {

if ((rv = getaddrinfo(servers[i].ip, servers[i].port, &hints, &servinfo)) != 0) {

server\_log("getaddrinfo: %s\n", 2, gai\_strerror(rv));

}

for (p = servinfo; p != NULL; p = p->ai\_next) {

if ((active\_srv[cnt] = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol)) == -1) {

server\_log("server: sync socket", 1, NULL);

continue;

}

if (connect(active\_srv[cnt], p->ai\_addr, p->ai\_addrlen) == -1) {

close(active\_srv[cnt]);

server\_log("server: sync connect", 1, NULL);

continue;

}

if ((rv = getaddrinfo(servers[i].ip, servers[i].chk\_port, &hints, &servinfo)) != 0) {

server\_log("getaddrinfo: %s\n", 2, gai\_strerror(rv));

}

for (p = servinfo; p != NULL; p = p->ai\_next) {

if ((tsock = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol)) == -1) {

server\_log("server: sync check socket", 1, NULL);

continue;

}

if (connect(tsock, p->ai\_addr, p->ai\_addrlen) == -1) {

close(tsock);

server\_log("server: sync check connect", 1, NULL);

continue;

}

break;

}

close(tsock);

message[0] = 5;

message[1] = id;

if (send(active\_srv[cnt], message, MSG\_LEN, 0) == -1) {

server\_log("server thread: sync send", 1, NULL);

continue;

}

cnt++;

}

}

}

game\_process(op1, op1\_chk, op2, op2\_chk, active\_srv);

close(op1);

close(op2);

close(op1\_chk);

close(op2\_chk);

} else {

memset(message, 0, MSG\_LEN);

message[1] = 60;

send\_to\_client(&new\_fd, message);

}

}

return NULL;

}

char \*concat(const char \*s1, const char \*s2) {

const size\_t len1 = strlen(s1);

const size\_t len2 = strlen(s2);

char \*result = malloc(len1 + len2 + 1); // +1 for the null-terminator

// in real code you would check for errors in malloc here

memcpy(result, s1, len1);

memcpy(result + len1, s2, len2 + 1); // +1 to copy the null-terminator

return result;

}

int main(int argc, char \*argv[]) {

int status, sockfd, new\_fd, yes = 1, rv, cnt = -1;

struct addrinfo hints, \*servinfo, \*p;

struct sockaddr\_storage their\_addr;

socklen\_t sin\_size;

struct sigaction sa;

char s[INET6\_ADDRSTRLEN];

char msg[MSG\_LEN];

pthread\_t tid;

int chk\_sockfd, chk\_new\_fd, yep = 1, inf;

struct addrinfo hnts, \*srvinfo, \*ptr;

struct sockaddr\_storage thr\_addr;

socklen\_t sn\_size;

char sip[INET6\_ADDRSTRLEN];

struct args th\_args;

if (argc != 6) {

fprintf(stderr, "usage: server <1/0> <IP> <PORT> <CHECK\_PORT> <LOG\_NAME>\n");

return 1;

}

if (strcmp(argv[1], "1") == 0) {

main\_server = 1;

printf("Main server:\n");

} else {

printf("Reserve server:\n");

groot = ginit();

}

pthread\_mutex\_init(&mtx, NULL);

char \*real\_log\_path = concat("./logs/", argv[5]);

logfd = fopen(real\_log\_path, "a");

if (logfd == NULL) {

printf("server: cannot create/open file file\n");

}

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE;

root = init();

if ((rv = getaddrinfo(argv[2], argv[3], &hints, &servinfo)) != 0)