**//main.c**

#include "MyPrint.h"

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

initscr();

start\_color();//цветовой режим

// Инициализация цветовых пар

init\_pair(GREEN\_COLOR, COLOR\_GREEN, COLOR\_BLACK);

init\_pair(RED\_COLOR, COLOR\_RED, COLOR\_BLACK);

init\_pair(BLUE\_COLOR, COLOR\_BLUE, COLOR\_BLACK);

cbreak(); // вводимые символы немедленно передаются программе, без ожидания нажатия на Enter

noecho(); // отключаем отображение вводимых символов

curs\_set(0); // скрываем курсор

attron(COLOR\_PAIR(GREEN\_COLOR));

if (argc <= 1){

menu\_sys\_info();}

else {

processArguments(argc, argv);}

attroff(COLOR\_PAIR(GREEN\_COLOR));

getch();

endwin();

return 0;

}

**//tools.h**

#ifndef TOOLS\_H\_

#define TOOLS\_H\_

#include <ncurses.h> // Заголовочный файл для работы с ncurses - библиотекой для управления текстовым окном в терминале

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h> // Заголовочный файл для работы с каталогами и файлами в них

#include <unistd.h> // Заголовочный файл для работы с POSIX API, такими как системные вызовы, форкинг процессов и т.д.

#include <pwd.h> // Заголовочный файл для работы с информацией о пользователях

#include <fcntl.h> // Заголовочный файл для работы с файловыми дескрипторами

#include <ifaddrs.h> // Заголовочный файл для получения списка сетевых интерфейсов

#include <netdb.h> // Заголовочный файл для работы с сетевыми функциями и структурами данных

#include <sys/socket.h> // Заголовочный файл для работы с сокетами

#include <net/if.h> // Заголовочный файл для работы с сетевыми интерфейсами

#include <sys/utsname.h> // Заголовочный файл для получения информации о системе

#include <sys/stat.h> // Заголовочный файл для работы со структурой stat, которая содержит информацию о файле

#include <sys/sysinfo.h> // Заголовочный файл для получения информации о системе

#include <sys/statvfs.h> // Заголовочный файл для получения информации о файловой системе

#include <sys/types.h> // Заголовочный файл для определения системных типов данных

#include <sys/ioctl.h> // Заголовочный файл для работы с вводом/выводом

#include <linux/rtc.h> // Заголовочный файл для работы с часами реального времени

#include <X11/Xlib.h> // Заголовочный файл для работы с X Window System

#define LENGTH\_ROW 155

#define CENTER 80

#define LEFT\_CENTER 70

#define RIGHT\_CENTER 100

#define START\_ROW 25

#define GREEN\_COLOR 1

#define RED\_COLOR 2

#define BLUE\_COLOR 3

int process\_count();

float covert\_bytes\_to\_gigabytes(float n);

void read\_sysfs\_file(const char \*path, char \*\*buffer);

void do\_ramka();

#endif

**//tools.c**

#include "tools.h"

int process\_count() {

int process\_count = 0; // Счетчик процессов

DIR\* proc\_dir = opendir("/proc");

if (proc\_dir != NULL) {

struct dirent\* entry;

while ((entry = readdir(proc\_dir)) != NULL) {

// Проверяем, является ли элемент каталогом

struct stat file\_stat;

char file\_path[256];

strncpy(file\_path, "/proc/", sizeof(file\_path));

strncat(file\_path, entry->d\_name, sizeof(file\_path) - strlen(file\_path) - 1);

if (stat(file\_path, &file\_stat) == 0 && S\_ISDIR(file\_stat.st\_mode)) {

// Проверяем, является ли имя каталога числом (идентификатором процесса)

int pid = atoi(entry->d\_name);

if (pid != 0) {

process\_count++; // Увеличиваем счетчик процессов

}

}

}

closedir(proc\_dir);

} else {

printf("Failed to open /proc directory.\n");

}

return process\_count;

}

float covert\_bytes\_to\_gigabytes(float n)

{

return (n / 1024.0 / 1024.0 / 1024.0);

}

void read\_sysfs\_file(const char \*path, char \*\*buffer) {

FILE \*file = fopen(path, "r");

if (file == NULL) {

printf("Error opening file: %s\n", path);

return;

}

\*buffer = (char \*)malloc(50 \* sizeof(char));

if (\*buffer == NULL) {

printf("Error allocating memory.\n");

fclose(file);

return;

}

fgets(\*buffer, 50, file);

fclose(file);

}

void do\_ramka(int start, int end, char\* title)

{

int i;

char choice = 'a';

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(start, i, "-");}

mvprintw(start, CENTER, "%s", title);

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(start + 1, i, "-");}

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(end, i, "-");}

for (i = start + 2; i < end; i++) {mvprintw(i, START\_ROW, "|");}

for (i = start + 2; i < end; i++) {mvprintw(i, LENGTH\_ROW - 1, "|");}

attroff(COLOR\_PAIR(GREEN\_COLOR));

attron(COLOR\_PAIR(BLUE\_COLOR));

mvprintw(end+1, LEFT\_CENTER, "Enter 'q' to exit modul");

attroff(COLOR\_PAIR(BLUE\_COLOR));

attron(COLOR\_PAIR(GREEN\_COLOR));

while(choice != 'q')

{choice = getch();} // Ввод символа с клавиатуры}

}

**//MyPrint.h**

#ifndef MYPRINT\_H\_

#define MYPRINT\_H\_

#include "tools.h"

void print\_IP\_info();

void print\_disk\_info();

void print\_memory\_info();

void print\_cpu\_info();

void print\_motherboardAndBios\_info();

void print\_realTimeClock\_info();

void print\_display\_info();

void print\_kernel\_info();

void print\_user\_info();

void print\_battery\_info();

void shapka();

void menu\_sys\_info();

void processArguments(int argc, char \*argv[]);

void switch\_info(char choice);

#endif

**//MyPrint.c**

#include "MyPrint.h"

void shapka()

{

clear(); // Очистка экрана

refresh(); // Обновление экрана

int i;

attroff(COLOR\_PAIR(GREEN\_COLOR));

attron(COLOR\_PAIR(BLUE\_COLOR));

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(1, i, "-");}

mvprintw(1, CENTER, "System Information");

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(2, i, "-");}

mvprintw(3, CENTER - 5, "[1] - Kernel Infomation ");

mvprintw(4, CENTER - 5, "[2] - User Infomation ");

mvprintw(5, CENTER - 5, "[3] - Memory && Processes Infomation ");

mvprintw(6, CENTER - 5, "[4] - Disk Infomation ");

mvprintw(7, CENTER - 5, "[5] - CPU Infomation ");

mvprintw(8, CENTER - 5, "[6] - Motherboard and Bios Infomation ");

mvprintw(9, CENTER - 5, "[7] - Real Time Clock Infomation ");

mvprintw(10, CENTER - 5, "[8] - Display Infomation ");

mvprintw(11, CENTER - 5, "[9] - IP Infomation ");

mvprintw(12, CENTER - 5, "[0] - Batery Infomation ");

mvprintw(13, CENTER - 5, "[q] - Exit programm ");

for (i = 3; i < 14; i++) {mvprintw(i, START\_ROW, "|");}

for (i = 3; i < 14; i++) {mvprintw(i, LENGTH\_ROW - 1, "|");}

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(14, i, "-");}

attroff(COLOR\_PAIR(BLUE\_COLOR));

attron(COLOR\_PAIR(GREEN\_COLOR));

}

void print\_battery\_info() {

clear(); // Очистка экрана

refresh(); // Обновление экрана

FILE \*file;

char buffer[128];

int energyNow, energyFull, voltage;

file = fopen("/sys/class/power\_supply/BAT0/energy\_now", "r");

if (file == NULL) {

printf("Failed to open file /sys/class/power\_supply/BAT0/energy\_now\n");

return;

}

fgets(buffer, 128, file);

energyNow = atoi(buffer);

fclose(file);

file = fopen("/sys/class/power\_supply/BAT0/energy\_full", "r");

if (file == NULL) {

printf("Failed to open file /sys/class/power\_supply/BAT0/energy\_full\n");

return;

}

fgets(buffer, 128, file);

energyFull = atoi(buffer);

fclose(file);

file = fopen("/sys/class/power\_supply/BAT0/voltage\_now", "r");

if (file == NULL) {

printf("Failed to open file /sys/class/power\_supply/BAT0/voltage\_now\n");

return;

}

fgets(buffer, 128, file);

voltage = atoi(buffer);

fclose(file);

float chargeLevel = ((float)energyNow / energyFull) \* 100;

mvprintw(3, LEFT\_CENTER, "Current battery charge:");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%.2f%%\n", chargeLevel); // Вывод текущего заряда аккумулятора в процентах

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "Maximum battery charge: ");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%d mWh\n", energyFull); // Вывод максимального заряда аккумулятора в милливатт-часах

attroff(A\_BOLD);

mvprintw(5, LEFT\_CENTER, "Current battery voltage: ");

attron(A\_BOLD);

mvprintw(5, RIGHT\_CENTER, "%d mV\n", voltage); // Вывод текущего напряжения аккумулятора в милливольтах

attroff(A\_BOLD);

do\_ramka(1, 6, "Battery Information");

}

void print\_kernel\_info() {

clear(); // Очистка экрана

refresh(); // Обновление экрана

struct utsname system\_info;//структура, которая содержит информацию о системе

uname(&system\_info);//используется для получения информации о системе и заполняет переданную ей структуру

mvprintw(3, LEFT\_CENTER, "System: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%s", system\_info.sysname);

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "Node: ");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%s", system\_info.nodename);//дистрибутив

attroff(A\_BOLD);

mvprintw(5, LEFT\_CENTER, "Release: ");

attron(A\_BOLD);

mvprintw(5, RIGHT\_CENTER, "%s", system\_info.release);//версия ядра

attroff(A\_BOLD);

mvprintw(6, LEFT\_CENTER, "Version: ");

attron(A\_BOLD);

mvprintw(6, RIGHT\_CENTER, "%s", system\_info.version);//дополнительная информация

attroff(A\_BOLD);

mvprintw(7, LEFT\_CENTER, "Machine: ");

attron(A\_BOLD);

mvprintw(7, RIGHT\_CENTER, "%s", system\_info.machine);//архитектура

attroff(A\_BOLD);

do\_ramka(1, 8, "Kernel Information");

}

void print\_display\_info()

{

Display \*display = XOpenDisplay(NULL);

if (display == NULL) {

fprintf(stderr, "Error opening display\n");

return;

}

int screenCount = ScreenCount(display);

XCloseDisplay(display);

clear(); // Очистка экрана

refresh(); // Обновление экрана

mvprintw(3, LEFT\_CENTER, "Number of screens: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%d", screenCount);

attroff(A\_BOLD);

for (int screenNum = 0; screenNum < screenCount; ++screenNum) {

Screen \*screen = ScreenOfDisplay(display, screenNum);

if (screen == NULL) {

fprintf(stderr, "Error getting information for screen %d\n", screenNum);

continue;

}

int width = WidthOfScreen(screen);

int height = HeightOfScreen(screen);

mvprintw(screenNum + 4, LEFT\_CENTER, "Screen resolution %d:", screenNum);

attron(A\_BOLD);

mvprintw(screenNum + 4, RIGHT\_CENTER, "%d x %d\n", width, height);

attroff(A\_BOLD);

}

do\_ramka(1, screenCount + 5, "Display Information");

}

void print\_user\_info() {

clear(); // Очистка экрана

refresh(); // Обновление экрана

uid\_t uid = getuid(); // Получение идентификатора текущего пользователя

struct passwd \*user\_info = getpwuid(uid); // Получение информации о пользователе

if (user\_info == NULL)

{

perror("ERROR WITH ACCES SYSTEM");

return;

}

mvprintw(3, LEFT\_CENTER, "User Name: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%s", user\_info->pw\_name); // Вывод имени пользователя

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "User ID :");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%d", user\_info->pw\_uid); // Вывод идентификатора пользователя

attroff(A\_BOLD);

mvprintw(5, LEFT\_CENTER, "Home Directory: ");

attron(A\_BOLD);

mvprintw(5, RIGHT\_CENTER, "%s", user\_info->pw\_dir); // Вывод домашнего каталога пользователя

attroff(A\_BOLD);

mvprintw(6, LEFT\_CENTER, "Shell :");

attron(A\_BOLD);

mvprintw(6, RIGHT\_CENTER, "%s", user\_info->pw\_shell); // Вывод оболочки пользователя

attroff(A\_BOLD);

do\_ramka(1, 7, "User Information");

}

void print\_realTimeClock\_info() {

clear(); // Очистка экрана

refresh(); // Обновление экрана

int rtc\_fd = open("/dev/rtc", O\_RDONLY);

if (rtc\_fd == -1) {

perror("Error opening RTC");

return;

}

struct rtc\_time rtc;

if (ioctl(rtc\_fd, RTC\_RD\_TIME, &rtc) == -1) {

perror("Error reading RTC");

close(rtc\_fd);

return;

}

close(rtc\_fd);

mvprintw(3, LEFT\_CENTER, "Year: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%d", rtc.tm\_year + 1900);

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "Month: ");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%d", rtc.tm\_mon + 1);

attroff(A\_BOLD);

mvprintw(5, LEFT\_CENTER, "Day: ");

attron(A\_BOLD);

mvprintw(5, RIGHT\_CENTER, "%d", rtc.tm\_mday);

attroff(A\_BOLD);

mvprintw(6, LEFT\_CENTER, "Hour:");

attron(A\_BOLD);

mvprintw(6, RIGHT\_CENTER, "%d", rtc.tm\_hour);

attroff(A\_BOLD);

mvprintw(7, LEFT\_CENTER, "Minute:");

attron(A\_BOLD);

mvprintw(7, RIGHT\_CENTER, "%d", rtc.tm\_min);

attroff(A\_BOLD);

mvprintw(8, LEFT\_CENTER, "Second:");

attron(A\_BOLD);

mvprintw(8, RIGHT\_CENTER, "%d", rtc.tm\_sec);

attroff(A\_BOLD);

do\_ramka(1, 9, "RTC Information:");

}

void print\_memory\_info() {

clear(); // Очистка экрана

refresh(); // Обновление экрана

struct sysinfo mem\_info;

if (sysinfo(&mem\_info) == -1)

{

perror("ERROR WITH SYSTEM ACCES");

return;

}

mvprintw(3, LEFT\_CENTER, "Total RAM: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%.2f GB", covert\_bytes\_to\_gigabytes(mem\_info.totalram)); // Вывод общего объема оперативной памяти

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "Free RAM:");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%.2f GB", covert\_bytes\_to\_gigabytes(mem\_info.freeram)); // Вывод объема свободной памяти

attroff(A\_BOLD);

mvprintw(5, LEFT\_CENTER, "Used RAM:");

attron(A\_BOLD);

mvprintw(5, RIGHT\_CENTER, "%.2f GB", covert\_bytes\_to\_gigabytes(mem\_info.totalram - mem\_info.freeram)); // Вывод объема используемой памяти

attroff(A\_BOLD);

mvprintw(6, LEFT\_CENTER, "Num active process: ");

attron(A\_BOLD);

mvprintw(6, RIGHT\_CENTER, "%d", process\_count()); // Вывод объема используемой памяти

attroff(A\_BOLD);

do\_ramka(1, 7, "Memory Information");

}

void print\_disk\_info() {

clear(); // Очистка экрана

refresh(); // Обновление экрана

struct statvfs stat;

// Получение информации о файловой системе

if (statvfs("/", &stat) == -1) {

perror("ERROR WITH ACCES FILE SYSTEM");

return;

}

mvprintw(3, LEFT\_CENTER, "Total disk space: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%.2f GB", covert\_bytes\_to\_gigabytes(stat.f\_blocks \* stat.f\_frsize)); // Вывод общего объема памяти

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "Free space:");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%.2f GB", covert\_bytes\_to\_gigabytes(stat.f\_bfree \* stat.f\_frsize)); // Вывод объема свободной памяти

attroff(A\_BOLD);

mvprintw(5, LEFT\_CENTER, "Used space:");

attron(A\_BOLD);

mvprintw(5, RIGHT\_CENTER, "%.2f GB", covert\_bytes\_to\_gigabytes(stat.f\_blocks \* stat.f\_frsize - stat.f\_bfree \* stat.f\_frsize));//Вывод объема используемой памяти

attroff(A\_BOLD);

do\_ramka(1, 6, "Disk Information");

}

void print\_motherboardAndBios\_info()

{

char \*board\_vendor = NULL;

char \*board\_name = NULL;

char \*bios\_vendor = NULL;

char \*bios\_version = NULL;

read\_sysfs\_file("/sys/devices/virtual/dmi/id/board\_vendor", &board\_vendor);

read\_sysfs\_file("/sys/devices/virtual/dmi/id/board\_name", &board\_name);

read\_sysfs\_file("/sys/devices/virtual/dmi/id/bios\_vendor", &bios\_vendor);

read\_sysfs\_file("/sys/devices/virtual/dmi/id/bios\_version", &bios\_version);

clear(); // Очистка экрана

refresh(); // Обновление экрана

int i;

mvprintw(3, LEFT\_CENTER, "Vendor: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%s", board\_vendor);

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "Name:");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%s", board\_name);

attroff(A\_BOLD);

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(5, i, "-");}

mvprintw(6, CENTER, "BIOS Information");

for (i = START\_ROW; i < LENGTH\_ROW; i++) {mvprintw(7, i, "-");}

mvprintw(8, LEFT\_CENTER, "Vendor: ");

attron(A\_BOLD);

mvprintw(8, RIGHT\_CENTER, "%s", bios\_vendor);

attroff(A\_BOLD);

mvprintw(9, LEFT\_CENTER, "Name:");

attron(A\_BOLD);

mvprintw(9, RIGHT\_CENTER, "%s", bios\_version);

attroff(A\_BOLD);

do\_ramka(1, 10, "Motherboard Information");

free(board\_vendor);

free(board\_name);

free(bios\_vendor);

free(bios\_version);

}

void print\_cpu\_info() {

FILE\* cpuinfo = fopen("/proc/cpuinfo", "r");

if (cpuinfo == NULL) {

printf("Failed to open /proc/cpuinfo\n");

return;

}

char line[256];

char model[256];

char frequency[256];

int cores = 0;

while (fgets(line, sizeof(line), cpuinfo)) {

if (strncmp(line, "model name", 10) == 0) {

char\* model\_ptr = strchr(line, ':');

if (model\_ptr != NULL) {

strncpy(model, model\_ptr + 2, sizeof(model) - 1);

model[sizeof(model) - 1] = '\0';

}

} else if (strncmp(line, "cpu MHz", 7) == 0) {

char\* frequency\_ptr = strchr(line, ':');

if (frequency\_ptr != NULL) {

strncpy(frequency, frequency\_ptr + 2, sizeof(frequency) - 1);

frequency[sizeof(frequency) - 1] = '\0';

}

} else if (strncmp(line, "cpu cores", 9) == 0) {

char\* cores\_ptr = strchr(line, ':');

if (cores\_ptr != NULL) {

cores = atoi(cores\_ptr + 2);

}

}

}

fclose(cpuinfo);

clear(); // Очистка экрана

refresh(); // Обновление экрана

mvprintw(3, LEFT\_CENTER, "CPU Model: ");

attron(A\_BOLD);

mvprintw(3, RIGHT\_CENTER, "%s", model); // Вывод модели ЦПУ

attroff(A\_BOLD);

mvprintw(4, LEFT\_CENTER, "CPU Frequency(MHz):");

attron(A\_BOLD);

mvprintw(4, RIGHT\_CENTER, "%s", frequency); // Вывод частоты ЦПУ в мегагерцах

attroff(A\_BOLD);

mvprintw(5, LEFT\_CENTER, "CPU Cores::");

attron(A\_BOLD);

mvprintw(5, RIGHT\_CENTER, "%d", cores); // Вывод количества ядер ЦПУ используемой памяти

attroff(A\_BOLD);

do\_ramka(1, 6, "CPU Information");

}

void print\_IP\_info() {

struct ifaddrs \*ifaddr, \*ifa;

char host[NI\_MAXHOST];

if (getifaddrs(&ifaddr) == -1) {

perror("getifaddrs");

return;

}

clear(); // Очистка экрана

refresh(); // Обновление экрана

struct utsname system\_info;//структура, которая содержит информацию о системе

uname(&system\_info);//используется для получения информации о системе и заполняет переданную ей структуру

int i = 0;

for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa\_next) {

if (ifa->ifa\_addr == NULL)

continue;

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

if (getnameinfo(ifa->ifa\_addr, sizeof(struct sockaddr\_in), host, NI\_MAXHOST, NULL, 0, NI\_NUMERICHOST) == 0) {

mvprintw(i+3, LEFT\_CENTER, "Interface: %s\tAddress: %s\n", ifa->ifa\_name, host);

i++;

}

}

}

do\_ramka(1, i + 4, "IP Information");

freeifaddrs(ifaddr);

}

void switch\_info(char choice){

switch(choice)

{

case '1':

print\_kernel\_info();

break;

case '2':

print\_user\_info();

break;

case '3':

print\_memory\_info();

break;

case '4':

print\_disk\_info();

break;

case '5':

print\_cpu\_info();

break;

case '6':

print\_motherboardAndBios\_info();

break;

case '7':

print\_realTimeClock\_info();

break;

case '8':

print\_display\_info();

break;

case '9':

print\_IP\_info();

break;

case '0':

print\_battery\_info();

break;

case 'q':

attroff(COLOR\_PAIR(GREEN\_COLOR));

attron(COLOR\_PAIR(RED\_COLOR));

mvprintw(15, CENTER - 5, "Exiting program... ");

attroff(COLOR\_PAIR(RED\_COLOR));

attron(COLOR\_PAIR(GREEN\_COLOR));

break;

default:

attroff(COLOR\_PAIR(GREEN\_COLOR));

attron(COLOR\_PAIR(RED\_COLOR));

mvprintw(15, CENTER - 5, "Invalid Input ");

refresh(); // Обновление экрана перед выводом сообщения

attroff(COLOR\_PAIR(RED\_COLOR));

attron(COLOR\_PAIR(GREEN\_COLOR));

sleep(1); // Задержка выполнения на 1 секунду

break ;

}

}

void menu\_sys\_info() {

char choice = 'a';

while(choice != 'q')

{

shapka();

choice = getch(); // Ввод символа с клавиатуры

switch\_info(choice);

}

}

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

for (int i = 1; i < argc; i++) {

if (strcmp(argv[i], "-kernel") == 0) {

switch\_info('1');

} else if (strcmp(argv[i], "-user") == 0) {

switch\_info('2');

} else if (strcmp(argv[i], "m&p") == 0) {

switch\_info('3');

} else if (strcmp(argv[i], "-disk") == 0) {

switch\_info('4');

} else if (strcmp(argv[i], "-gpu") == 0) {

switch\_info('5');

} else if (strcmp(argv[i], "-m&b") == 0) {

switch\_info('6');

} else if (strcmp(argv[i], "-rtc") == 0) {

switch\_info('7');

} else if (strcmp(argv[i], "-display") == 0) {

switch\_info('8');

} else if (strcmp(argv[i], "-ip") == 0) {

switch\_info('9');

} else if (strcmp(argv[i], "-batery") == 0) {

switch\_info('0');

} else {

clear(); // Очистка экрана

refresh(); // Обновление экрана

attroff(COLOR\_PAIR(GREEN\_COLOR));

attron(COLOR\_PAIR(RED\_COLOR));

mvprintw(15, CENTER - 5, "Invalid Argument ");

attroff(COLOR\_PAIR(RED\_COLOR));

attron(COLOR\_PAIR(BLUE\_COLOR));

mvprintw(16, LEFT\_CENTER, "Enter 'q' to exit modul");

attroff(COLOR\_PAIR(BLUE\_COLOR));

attron(COLOR\_PAIR(GREEN\_COLOR));

char choice = 'a';

while(choice != 'q')

{choice = getch();} // Ввод символа с клавиатуры}

}

}

}