1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Высшая школа кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «**Принципы разработки ОС**»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5131001/20001 Маронова К.Д.

<*подпись*>

1. Преподаватель
2. Огнёв Р.А.

<*подпись*>

1. Санкт-Петербург
2. 2024

**1. Цель работы**

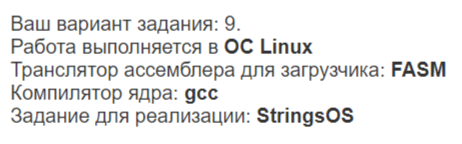
Изучение основ разработки ОС, принципов низкоуровневого взаимодействия с аппаратным обеспечением, программирования системной функциональности и процесса загрузки системы

**2. Постановка задачи**

Основная задача лабораторной работы состоит из нескольких пунктов:

1. Скомпилировать и запустить на эмуляторе пример загрузочного сектора start.asm, описанного в Теоретических сведениях, указанным в вашем варианте транслятором.
2. Пользуясь примерами из Теоретических сведений, разработать простой загрузчик для загрузки минимального ядра. Скомпилировать агрузчик и минимальное ядро указанными в вашем варианте задания транслятором и компилятором языка Си. Проверить работоспособность загрузчика и ядра на эмуляторе.
3. Разработать указанные в вашем варианте задания функции загрузчика.
4. Расширить реализацию минимального ядра, добавив в него функции, перечисленные в задании.

**3. Описание решения**



*Рисунок 1. «Вариант задания».*

**Описание алгоритма работы загрузчика ОС:**

1. Инициализация:

* Загрузчик сохраняет адреса сегментов кода, данных и стека.
* Загрузчик выводит на экран запрос "std/bm?". Для выбора алгоритма работы со строками.

2. Ожидание ответа пользователя:

Загрузчик ожидает нажатия клавиши.

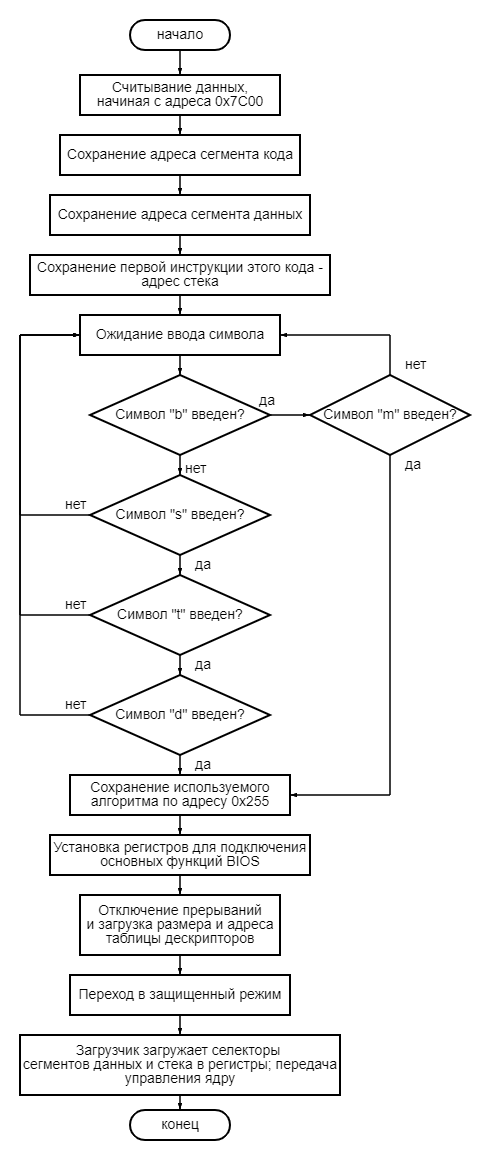
* Если нажата клавиша "s", загрузчик переходит к ожиданию буквы "t" и т.д. Иначе снова переходит к ожиданию буквы "s".
* Если нажата клавиша "b", загрузчик переходит к ожиданию буквы "m". Иначе снова переходит к ожиданию буквы "b".
* Выбранный алгоритм фиксируется по адресу 0х255 (для дальнейшей обработки в ядре).

3. Переход в защищенный режим:

* Загрузчик отключает прерывания.
* Загрузчик загружает таблицу дескрипторов глобальной (GDT).
* Загрузчик включает адресную линию A20 (для того, чтобы использовать память свыше 1МБ.
* Загрузчик устанавливает бит PE регистра CR0.
* Загрузчик переходит в защищенный режим.

4. Запуск ядра ОС:

* Загрузчик передает управление ядру ОС.

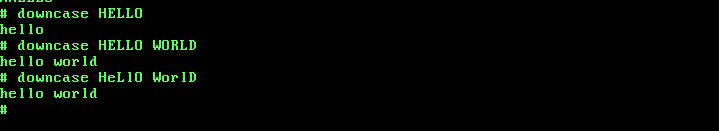


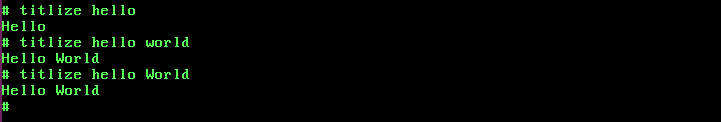
*Рисунок 2. «Блок-схема загрузчика».*

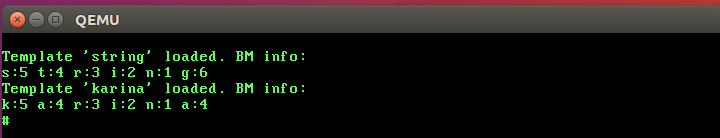
|  |  |  |  |
| --- | --- | --- | --- |
| Функция | Аргументы | Тип данных | Описание |
| outb\_symbol | color (int), symbol (unsigned char) | int, unsigned char | Выводит символ symbol с заданным цветом color на экран. |
| out\_word | color (int), ptr (const char\*) | int, const char\* | Выводит строку, указанную на ptr, с заданным цветом color на экран. |
| on\_key | scan\_code (unsigned char) | unsigned char | Обрабатывает нажатие клавиши с кодом scan\_code. |
| out\_num | num (int) | int | Выводит целое число num на экран. |
| comand\_func | scan\_code (unsigned char) | unsigned char | Выполняет команду, введенную пользователем, в зависимости от значения scan\_code. |
| bm | str (unsigned char\*) | unsigned char\* | Ищет подстроку str в тексте экрана с помощью алгоритма Бойера-Мурса. |
| screen\_clear | - | - | Очищает экран. |

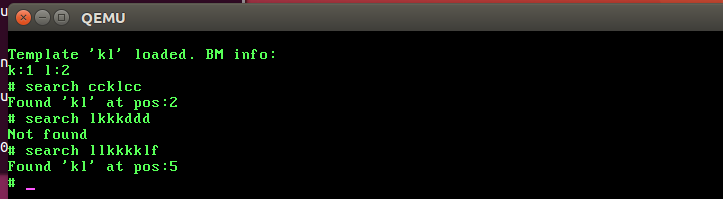
*Таблица 1. «Описание реализации».*

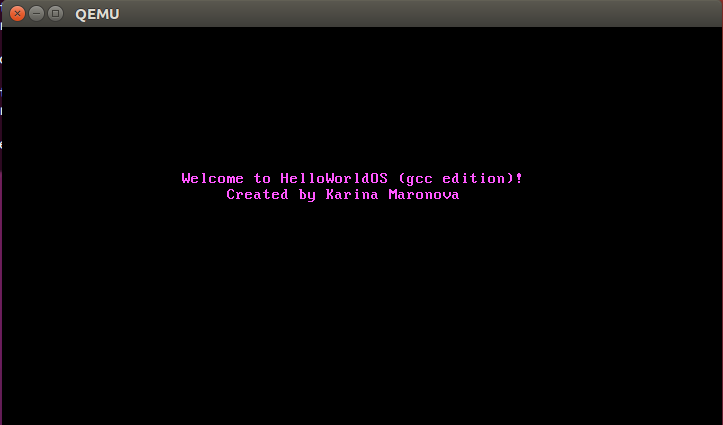
  
*Рисунок 3. «Тесты команды ‘upcase’».*

  
*Рисунок 4. «Тесты команды ‘downcase’».*

  
*Рисунок 5. «Тесты команды ‘titlize’».*

  
  
*Рисунок 6. «Тесты команды ‘template’(std\bm)».*

  
*Рисунок 7. «Тесты команды ‘search».*

  
*Рисунок 8. «Приветственный экран ОС».*

**4. Выводы**

В ходе работы были изучены основы разработки ОС(изучены реальный и

защищённый режим, переход из одного в другой, изучена работа загрузчика и ядра), принципы низкоуровневого взаимодействия с аппаратным обеспечением: работа с сегментами, изучение таких базовых функций BIOS, как видео сервис(используется прерывание 0х10), дисковый ввод/вывод(0х13) и работа с клавиатурой(0х16).

Была разработана операционная система StringOS, выполняющая работу со строками.

Основные проблемы, встретившиеся в ходе работы заключались в следующем: 1) Постоянные трудности с тем, что объекты разных типов данных. Необходимо было приводить один тип к другому, особенно при работе с video\_buf. 2) Из-за ошибки в примере таблицы смещений из методического пособия, возникла трудность с алгоритмом Бойера-Мура. Однако она была отлажена, путем иного написания алгоритма. 3) Большое количество глобальных переменных порой приводило к неточной работе программы, но это так же было отлажено.

**Листинг программы:**

**Листинг загрузчика с текстом скрипта, вызывающего транслятор ассемблера, компилятор и линковщик, для сборки проекта на языке ассемблер:**

use16

org 0x7C00

start:

; Инициализация адресов сегментов. Эти операции требуется не для любого BIOS, но их рекомендуется проводить.

mov ax, cs ; Сохранение адреса сегмента кода в ax

mov ds, ax; Сохранение этого адреса как начало сегмента данных

mov ss, ax ; И сегмента стека

mov sp, start ; Сохранение адреса стека как адрес первой инструкции этого кода. Стек будет расти вверх и не перекроет код.

; STD or BM часть проверки

mov ax, cs ; Сохранение адреса сегмента кода в ax

mov ds, ax; Сохранение этого адреса как начало сегмента данных

mov ss, ax ; И сегмента стека

mov sp, start ; Сохранение адреса стека как адрес первой инструкции этого кода. Стек будет расти вверх и не перекроет код.

mov ah, 0x0e

mov al, ' '

int 0x10

int 0x10

mov al, 's'

int 0x10

mov al, 't'

int 0x10

mov al, 'd'

int 0x10

mov al, '/'

int 0x10

mov al, 'b'

int 0x10

mov al, 'm'

int 0x10

mov al, '?'

int 0x10

mov al, ' '

int 0x10

wait\_for\_letter\_s\_cycle:

mov ah, 0x00 ; регистровая пара ah и al. Команда мув записывает значения в al (поэтому ниже проверяем именно его)

int 0x16

cmp al, 's'

je wait\_for\_letter\_t\_cycle

cmp al, 'b'

je wait\_for\_letter\_m\_cycle

jmp wait\_for\_letter\_s\_cycle

wait\_for\_letter\_t\_cycle:

mov ah, 0x00

int 0x16

cmp al, 't'

je wait\_for\_letter\_d\_cycle

jne wait\_for\_letter\_s\_cycle

wait\_for\_letter\_d\_cycle:

mov ah, 0x00

mov cx, 'd'

mov [0x255], cx

int 0x16

cmp al, 'd'

je load\_drive

jne wait\_for\_letter\_s\_cycle

wait\_for\_letter\_m\_cycle:

mov ah, 0x00

mov cx, 'm'

mov [0x255], cx

int 0x16

cmp al, 'm'

je load\_drive

jmp wait\_for\_letter\_s\_cycle

;окончание проверки на std\bm

load\_drive:

mov bx, 0x1000

mov es, bx

xor bx, bx

mov al, 0x30 ; sum of sectors to read

mov dl, 0x01 ; number of drive

mov dh, 0x00 ; number of head

mov ch, 0x00 ; number of track(cylinder)

mov cl, 0x01 ; number of sector

mov ah, 0x02 ; function for int 0x13

int 0x13 ; read data from 0x1000

enable\_protected\_mode:

; Отключение прерываний

cli

; Загрузка размера и адреса таблицы дескрипторов

lgdt[gdt\_info]

; Включение адресной линии А20

in al, 0x92

or al, 2

out 0x92, al

; Установка бита PE регистра CR0 -процессор перейдет в защищенный режим

mov eax, cr0

or al, 1

mov cr0, eax

jmp 0x8:protected\_mode; "Дальний" переход для загрузки корректной информации в cs(архитектурные особенности не позволяют этого сделать напрямую).

gdt:

; Нулевой дескриптор

db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

; Сегмент кода: base=0, size=4Gb, P=1, DPL=0, S=1(user),

; Type=1(code), Access=00A, G=1, B=32bit

db 0xff, 0xff, 0x00, 0x00, 0x00, 0x9A, 0xCF, 0x00

; Сегмент данных: base=0, size=4Gb, P=1, DPL=0, S=1(user),

; Type=0(data), Access=0W0, G=1, B=32bit

db 0xff, 0xff, 0x00, 0x00, 0x00, 0x92, 0xCF, 0x00

gdt\_info: ; Данные о таблице GDT(размер, положение в памяти)

dw gdt\_info - gdt ; Размер таблицы (2 байта)

dw gdt, 0 ; 32-битный физический адрес таблиц

use32

protected\_mode:

; Загрузка селекторов сегментов для стека и данных в регистры

mov ax, 0x10 ; Используется дескриптор с номером 2 в GDT

mov es, ax

mov ds, ax

mov ss, ax

;Передача управления загруженному ядру

call 0x10000 ; Адрес равен адресу загрузки в случае если ядро скомпилировано в "плоский" код

inf\_loop:jmp inf\_loop ; Бесконечный цикл

times (512 -($ -start) -2) db 0

db 0x55, 0xAA

# Сборка загрузочного сектора на языке ассемблера с помощью FASM

fasm bootsect.asm

# Компиляция ядра на C++ с использованием компилятора g++

g++ -ffreestanding -m32 -o kernel.o -c kernel.cpp

# Линковка объектного файла ядра с помощью линковщика ld

ld --oformat binary -Ttext 0x10000 -o kernel.bin - -entry=kmain -m elf\_i386 kernel.o

# Запуск проекта в эмуляторе QEMU с указанием загрузочного и ядрового файла

qemu –fda bootsect.bin –fdb kernel.bin

**Листинг ядра на языке Си:**

// Эта инструкция обязательно должна быть первой, т.к. этот код компилируется в бинарный,

// и загрузчик передает управление по адресу первой инструкции бинарногообраза ядра ОС.

\_\_asm("jmp kmain");

#define VIDEO\_BUF\_PTR (0xb8000)

#define IDT\_TYPE\_INTR (0x0E)

#define IDT\_TYPE\_TRAP (0x0F)

#define GDT\_CS (0x8)

#define PIC1\_PORT (0x20)

#define CURSOR\_PORT (0x3D4)

#define VIDEO\_WIDTH (80)

unsigned short new\_pos=0;

//////////////В ОСНОВНОМ НОВЫЕ ФУНКЦИИ//////////////////////////

void keyb\_handler();

void keyb\_process\_keys();

char scan\_code\_table[] = {0,0,'1','2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=',0,'\t','q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']',' ',0,'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', '<','>','+',0,'\\', 'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.', '/',0,'\*',0,' ',0, 0,0,};

char letters[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z','A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

void screen\_clear();

void on\_key(unsigned char scan\_code);

void cursor\_moveto(unsigned int strnum, unsigned int pos);

void comand\_func(unsigned char scan\_code);

bool strcmp(unsigned char \*str1, const char \*str2);

int strlen(unsigned char \*str, bool flag);

char c[42]={0};

short str\_cnt = 0, symbol\_cnt=0;

bool flag=true, shift\_flag=false;

int skip\_table[255]={0};

void bm(unsigned char\* str);

///////////////////////////////////////

void default\_intr\_handler();

typedef void (\*intr\_handler)();

void intr\_reg\_handler(int num, unsigned short segm\_sel, unsigned short

flags, intr\_handler hndlr);

void intr\_init();

void intr\_start();

void intr\_enable();

void intr\_disable();

void out\_str(int color, const char\* ptr, unsigned int strnum);

static inline unsigned char inb (unsigned short port);

static inline void outb (unsigned short port, unsigned char data);

static inline void outw (unsigned int port, unsigned int data);

void keyb\_handler();

void keyb\_init();

void keyb\_process\_keys();

//////////////////////////////////////

// Структура описывает данные об обработчике прерывания

struct idt\_entry

{

unsigned short base\_lo; // Младшие биты адреса обработчика

unsigned short segm\_sel; // Селектор сегмента кода

unsigned char always0; // Этот байт всегда 0

unsigned char flags; // Флаги тип. Флаги: P, DPL, Типы - это константы - IDT\_TYPE...

unsigned short base\_hi; // Старшие биты адреса обработчика

} \_\_attribute\_\_((packed)); // Выравнивание запрещено

// Структура, адрес которой передается как аргумент команды lidt

struct idt\_ptr

{

unsigned short limit;

unsigned int base;

} \_\_attribute\_\_((packed)); // Выравнивание запрещено

struct idt\_entry g\_idt[256]; // Реальная таблица IDT

struct idt\_ptr g\_idtp; // Описатель таблицы для команды lidt

void outb\_symbol(int color, unsigned char symbol){

unsigned char\* video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

video\_buf += 2\*(str\_cnt\*VIDEO\_WIDTH + symbol\_cnt);

video\_buf[0] = symbol; // Символ (код)

video\_buf[1] = color; // Цвет символа и фона

cursor\_moveto(str\_cnt, symbol\_cnt);

//return;

}

void out\_word(int color, const char\* ptr)

{

unsigned char\* video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

video\_buf += 2\*(80 \* str\_cnt + symbol\_cnt);

while (\*ptr)

{

video\_buf[0] = (unsigned char) \*ptr; // Символ (код)

video\_buf[1] = color; // Цвет символа и фона

video\_buf += 2;

symbol\_cnt++;

ptr++;

}

}

void on\_key(unsigned char scan\_code){

if(flag){

flag=false;

screen\_clear();

str\_cnt=0;

symbol\_cnt=0;

out\_str(0x0A, "# ", 1);

symbol\_cnt=2;

cursor\_moveto(str\_cnt,symbol\_cnt);

}

if (str\_cnt>23){

screen\_clear();

out\_str(0x0A, "# ", 1);

str\_cnt=1;

symbol\_cnt=2;

}

else if (scan\_code == 28){ //enter

comand\_func(scan\_code);

out\_str(0x0A, "# ", ++str\_cnt);

symbol\_cnt=2;

str\_cnt--;

cursor\_moveto(str\_cnt,symbol\_cnt);

}

else if (scan\_code == 14){ //удаление буквы

if (symbol\_cnt >2){

unsigned char\* video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

video\_buf += 2\*(str\_cnt\*VIDEO\_WIDTH + (symbol\_cnt-1));

video\_buf[0] = '\0';

cursor\_moveto(str\_cnt,--symbol\_cnt);

}

}

else if (scan\_code == 57){ //пробел

if (symbol\_cnt<42){

unsigned char\* video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

video\_buf += 2\*(str\_cnt\*VIDEO\_WIDTH + (symbol\_cnt));

video\_buf++;

cursor\_moveto(str\_cnt,++symbol\_cnt);

}

}

else if (scan\_code == 42){ //проверка на шифт для заглавных букв

shift\_flag = true;

}

else if (symbol\_cnt<42){ //печать символа

if (shift\_flag == false){

char c = scan\_code\_table[(unsigned int)scan\_code];

outb\_symbol(0x0A, c);

symbol\_cnt++;

}

else{

char c = scan\_code\_table[(unsigned int)scan\_code];

int i=0;

for(i;i<26;i++){

if(c==letters[i]){

break;

}

}

outb\_symbol(0x0A, letters[i + 26]);

cursor\_moveto(str\_cnt,++symbol\_cnt);

shift\_flag=false;

}

}

}

void out\_num(int num)

{

int first;

if (!num) { outb\_symbol(0x0A, '0'); return; }

while(num)

{

if (num > 9){

first = num/10;

num%=10;

}

else { first = num; num/=10;}

switch(first)

{

case 1: { outb\_symbol(0x0A, '1'); break; }

case 2: { outb\_symbol(0x0A, '2'); break; }

case 3: { outb\_symbol(0x0A, '3'); break; }

case 4: { outb\_symbol(0x0A, '4'); break; }

case 5: { outb\_symbol(0x0A, '5'); break; }

case 6: { outb\_symbol(0x0A, '6'); break; }

case 7: { outb\_symbol(0x0A, '7'); break; }

case 8: { outb\_symbol(0x0A, '8'); break; }

case 9: { outb\_symbol(0x0A, '9'); break; }

}

}

}

void comand\_func(unsigned char scan\_code){

unsigned char\* video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

video\_buf += 2\*(str\_cnt\*VIDEO\_WIDTH + 2);

if (strcmp(video\_buf, "shutdown")){

outw(0x604, 0x2000);

}

else if (strcmp(video\_buf, "info")){

if (str\_cnt > 18) screen\_clear();

symbol\_cnt = 0;

out\_str(0x0A, " SYSTEM INFO", str\_cnt+4);

out\_str(0x0A, " Variant: 9", ++str\_cnt);

out\_str(0x0A, " OS: Linux", ++str\_cnt);

out\_str(0x0A, " Translator: FASM", ++str\_cnt);

out\_str(0x0A, " Compiler: gcc", ++str\_cnt);

out\_str(0x0A, " Author: Karina Maronova", ++str\_cnt);

if (\*(char\*)0x255=='d'){

out\_str(0x0A, "You have chosen STD algorythm", ++str\_cnt);

}

if (\*(char\*)0x255=='m'){

out\_str(0x0A, "You have chosen BM algorythm", ++str\_cnt);

}

}

else if (strcmp(video\_buf, "upcase")){

unsigned char \*str = video\_buf + 14;

int len = strlen(str,true);

symbol\_cnt = 0;

cursor\_moveto(++str\_cnt, symbol\_cnt);

while (len > 0) {

char c = \*str;

if (c >= 'a' && c <= 'z') {

c -= 'a' - 'A'; // преобразование в верхний регистр

}

outb\_symbol(0x0A, c);

cursor\_moveto(str\_cnt, ++symbol\_cnt);

str += 2;

len--;

}

}

else if (strcmp(video\_buf, "downcase")){

unsigned char \*str = video\_buf + 18;

int len = strlen(str,true);

symbol\_cnt = 0;

cursor\_moveto(++str\_cnt, 0);

while (len > 0) {

char c = \*str;

if (c >= 'A' && c <= 'Z') {

c += 'a' - 'A'; // преобразование в верхний регистр

}

outb\_symbol(0x0A, c);

cursor\_moveto(str\_cnt, ++symbol\_cnt);

str += 2;

len--;

}

}

else if (strcmp(video\_buf, "titlize")){

unsigned char \*str = video\_buf + 16;

int len = strlen(str,true);

symbol\_cnt = 0;

cursor\_moveto(++str\_cnt, 0);

while (len > 0)

{

char c = \*str;

if (\*(str - 2) == '\0') {//If last symbol was "space"

if (c >= 'a' && c <= 'z') {

c -= 'a' - 'A'; // преобразование в верхний регистр

}

}

outb\_symbol(0x0A, c);

cursor\_moveto(str\_cnt, ++symbol\_cnt);

str += 2;

len--;

}

}

else if (strcmp(video\_buf, "template")){

unsigned char \*str = video\_buf + 18;

int len = strlen(str,true);

for(int i = 0; i < len; i++){

c[i] = str[2\*i];

}

symbol\_cnt = 0;

out\_word(0x0A, "Template '");

out\_word(0x0A, c);

out\_word(0x0A, "' loaded. ");

for (int i = 0; i < 255; i++){

skip\_table[i] = len;

}

for (int i = 0; i < len - 1; i++) {

skip\_table[(unsigned char)c[i]] = len - 1 - i;

}

//ВЫВОД СМЕЩЕНИЙ

if(\*(char\*)0x255=='m'){

out\_word(0x0A, "BM info:");

symbol\_cnt = 0;

cursor\_moveto(++str\_cnt, symbol\_cnt);

for (int i = 0; i < len; i++)

{

outb\_symbol(0x0A, c[i]);

cursor\_moveto(str\_cnt, ++symbol\_cnt);

outb\_symbol(0x0A, ':');

cursor\_moveto(str\_cnt, ++symbol\_cnt);

out\_num(skip\_table[c[i]]);

cursor\_moveto(str\_cnt, ++symbol\_cnt);

outb\_symbol(0x0A, ' ');

cursor\_moveto(str\_cnt, ++symbol\_cnt);

}

}

}

else if (strcmp(video\_buf, "search")){

bm((video\_buf + 14));

}

else{

out\_str(0x0C, "WRONG COMMAND! ", str\_cnt);

str\_cnt--;

}

}

void bm(unsigned char\* str){

char tmp\_str[126]={0};

int text\_len = strlen(str,true);

for(int i = 0; i < text\_len; i++){

tmp\_str[i] = str[2\*i];

}

int pattern\_len = strlen((unsigned char\*)c,false);

int i = pattern\_len - 1; // Текущая позиция в тексте

int j = pattern\_len - 1; // Текущая позиция в образце

while (i < text\_len) {

j = pattern\_len - 1;

while (j >= 0 && tmp\_str[i] == c[j]) {

i--;

j--;

}

if (j < 0) {

symbol\_cnt = 0;

cursor\_moveto(++str\_cnt, symbol\_cnt);

out\_word(0x0A, "Found '");

out\_word(0x0A, c);

out\_word(0x0A, "' at pos:");

out\_num(i+1);

return;

}

i += skip\_table[(unsigned char)tmp\_str[i]];

}

symbol\_cnt = 0;

cursor\_moveto(++str\_cnt, symbol\_cnt);

out\_word(0x0A, "Not found");

}

// Функция переводит курсор на строку strnum (0 –самая верхняя) в позицию pos на этой строке (0 –самое левое положение).

void cursor\_moveto(unsigned int strnum, unsigned int pos){

new\_pos = (strnum \* VIDEO\_WIDTH) + pos;

outb(CURSOR\_PORT, 0x0F);

outb(CURSOR\_PORT + 1, (unsigned char)(new\_pos & 0xFF));

outb(CURSOR\_PORT, 0x0E);

outb(CURSOR\_PORT + 1, (unsigned char)( (new\_pos >> 8) & 0xFF));

}

//////////////////////////////////// MAIN //////////////////////////////////////////////////////////

extern "C" int kmain()

{

const char\* hello = " Welcome to HelloWorldOS (gcc edition)!";

const char\* g\_test = " Created by Karina Maronova";

// Вывод строки

unsigned char \*video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

screen\_clear();

out\_str(0x0D, hello, 9);

out\_str(0x0D, g\_test, 10);

cursor\_moveto(80,25);

intr\_disable();

intr\_init();

keyb\_init();

intr\_start();

intr\_enable();

// Бесконечный цикл

while(1)

{

asm("hlt");

}

return 0;

}

/////////////////////////////////////////////////////////////////////////////////////////////////////

void screen\_clear(){

unsigned char \*video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

for (int i = 0; i < 80\*25; i++) //width and height

{

\*(video\_buf + i\*2) = '\0'; //Каждый элемент занимает 2 байта: аски код и атрибут - цвет

//start=false;

}

//out\_str(0x0A, "# ", 1);

//symbol\_cnt = 2;

cursor\_moveto(1, 3);

intr\_disable();

intr\_init();

keyb\_init();

intr\_start();

intr\_enable();

}

// Пустой обработчик прерываний. Другие обработчики могут быть реализованы по этому шаблону

void default\_intr\_handler()

{

asm("pusha");

// ... (реализация обработки)

asm("popa; leave; iret");

}

typedef void (\*intr\_handler)();

void intr\_reg\_handler(int num, unsigned short segm\_sel, unsigned short

flags, intr\_handler hndlr)

{

unsigned int hndlr\_addr = (unsigned int) hndlr;

g\_idt[num].base\_lo = (unsigned short) (hndlr\_addr & 0xFFFF);

g\_idt[num].segm\_sel = segm\_sel;

g\_idt[num].always0 = 0;

g\_idt[num].flags = flags;

g\_idt[num].base\_hi = (unsigned short) (hndlr\_addr >> 16);

}

// Функция инициализации системы прерываний: заполнение массива с адресами обработчиков

void intr\_init()

{

int i;

int idt\_count = sizeof(g\_idt) / sizeof(g\_idt[0]);

for(i = 0; i < idt\_count; i++)

intr\_reg\_handler(i, GDT\_CS, 0x80 | IDT\_TYPE\_INTR,

default\_intr\_handler); // segm\_sel=0x8, P=1, DPL=0, Type=Intr

}

void intr\_start()

{

int idt\_count = sizeof(g\_idt) / sizeof(g\_idt[0]);

g\_idtp.base = (unsigned int) (&g\_idt[0]);

g\_idtp.limit = (sizeof (struct idt\_entry) \* idt\_count) - 1;

asm("lidt %0" : : "m" (g\_idtp) );

}

void intr\_enable()

{

asm("sti");

}

void intr\_disable()

{

asm("cli");

}

// функция для вывода строки

void out\_str(int color, const char\* ptr, unsigned int strnum)

{

unsigned char\* video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR;

video\_buf += 80\*2 \* strnum;

while (\*ptr)

{

video\_buf[0] = (unsigned char) \*ptr; // Символ (код)

video\_buf[1] = color; // Цвет символа и фона

video\_buf += 2;

ptr++;

}

str\_cnt++;

}

static inline unsigned char inb (unsigned short port) // Чтение из порта

{

unsigned char data;

asm volatile ("inb %w1, %b0" : "=a" (data) : "Nd" (port));

return data;

}

static inline void outb (unsigned short port, unsigned char data) // Запись

{

asm volatile ("outb %b0, %w1" : : "a" (data), "Nd" (port));

}

static inline void outw (unsigned int port, unsigned int data)

{

asm volatile ("outw %w0, %w1" : : "a" (data), "Nd" (port));

}

void keyb\_init(){

intr\_reg\_handler(0x09, GDT\_CS, 0x80 | IDT\_TYPE\_INTR, keyb\_handler);

outb(PIC1\_PORT+ 1, 0xFF^ 0x02);

}

void keyb\_handler(){

asm("pusha");// Обработка поступивших данных

keyb\_process\_keys();

outb(PIC1\_PORT, 0x20);

asm("popa; leave; iret");

}

void keyb\_process\_keys(){

// Проверка что буфер PS/2 клавиатуры не пуст (младший бит присутствует)

if (inb(0x64) & 0x01) {

unsigned char scan\_code;

unsigned char state;

scan\_code= inb(0x60);

if(scan\_code< 128) on\_key(scan\_code);

}

}

//////////////////////// РАБОТА СО СТРОКАМИ ///////////////////////////////////////////

bool strcmp( unsigned char \*str1, const char \*str2){

while (\*str1 != '\0' && \*str1 != ' ' && \*str2 != '\0' && \*str1 == \*str2)

{

str1+=2;

str2++;

}

if (\*str1 == \*str2) return true;

return false;

}

int strlen(unsigned char \*str, bool flag){

int i = 0;

if(flag){

while(\*str != '\0'||\*(str+2)!='\0') { str+=2; i++;}

}

else{

while(\*str != '\0') { str++; i++;}

}

return i;

}

////////////////////////////////////////////////////////////