1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт прикладной математики и механики
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «**Принципы разработки операционных систем**»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 4851003/10002 Тоцкий В.

<*подпись*>

1. Преподаватель
2. ассистент Крундышев В.М.

<*подпись*>

1. Санкт-Петербург
2. 2023

цель работы

Цель работы – изучение основ разработки ОС, принципов низкоуровневого взаимодействия с аппаратным обеспечением, программирования системной функциональности и процесса загрузки системы.

Ход работы

1. Описание алгоритма работы загрузчика ОС, процесса загрузки ОС.

* Инициализация адресов сегментов памяти.
* Отключение прерываний.
* Загрузка размера и адреса таблицы дескрипторов.
* Включение адресной линии А20.
* Перевод процессора в защищенный режим.
* "Дальний" переход для загрузки корректной информации.
* Загрузка селекторов сегментов для стека и данных в регистры.
* Передача управления загруженному ядру.

Для реализации решения задания к загрузчику было создано 4-ёх байтное слово buffer, в которой хранятся последние четыре символа, введённых с клавиатуры пользователем. Реализована функция input, которая сначала принимает от пользователя символ с помощью функции «ожидания нажатия и считывания нажатой» 0x00 и вызова прерывания int 0x16. Полученный символ, который помещён в регистр «al», помещается в младший байт буфера, и далее идёт проверка на то, последние два символа составляют последовательно ‘bm’, если нет, то также идёт проверка на ‘std’. После, если какая-либо последовательность совпала, то по адресу 0x9004 записывается 0 или 1 (для ‘std’ или ‘bm’ соответственно) и управление передаётся на участок кода, ответственный за передачу управлению ядру.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Блок-схема работы загрзучика.

1. Описание метода решения задания к ядру ОС.

|  |  |
| --- | --- |
| void out\_str(int color, const char\* ptr, unsigned int strnum) | Функция вывода строки на экран |
| void out\_symb(int color, unsigned char symb, unsigned int strnum, unsigned int pos) | Функция вывода символа на экран |
| void clear() | Очистка текстового экрана |
| int strcmp\_(unsigned char \*s1, unsigned char \*s2) | Сравнение строк |
| void strcpy\_(unsigned char \*dst, const char \*src) | Копирование строки из src в dst |
| void strcat\_(unsigned char \*str1, const char \*str2) | Добавление второй строки к концу первой |
| int strlen\_(unsigned char \*str1) | Вычисление длины строки |
| void swap(char \*x, char \*y) | Функция для замены двух чисел |
| char\* reverse(char \*buffer, int i, int j) | Функция для обращения `buffer[i…j]` |
| char\* int\_to\_char(int value, char\* buffer, int base) | Перевод числа в строку |
| void precompute\_bad\_chars(char \*str, int size, int bad\_char\_table[]) | Функция заполнения таблицы смещений для Б-М |
| void action() | Исполнение команды |
| void on\_key(unsigned char scan\_code) | Обработка нажатой клавиши |
| void un\_key(unsigned char scan\_code) | Обработка отпущенных клавиш |
| void keyb\_process\_keys() | Считывание скан-кода нажатой или отпущенной клавиши |

Таблица 1 – Описание функций, необходимых для реализации задания.

1. Входные данные и результаты работы.

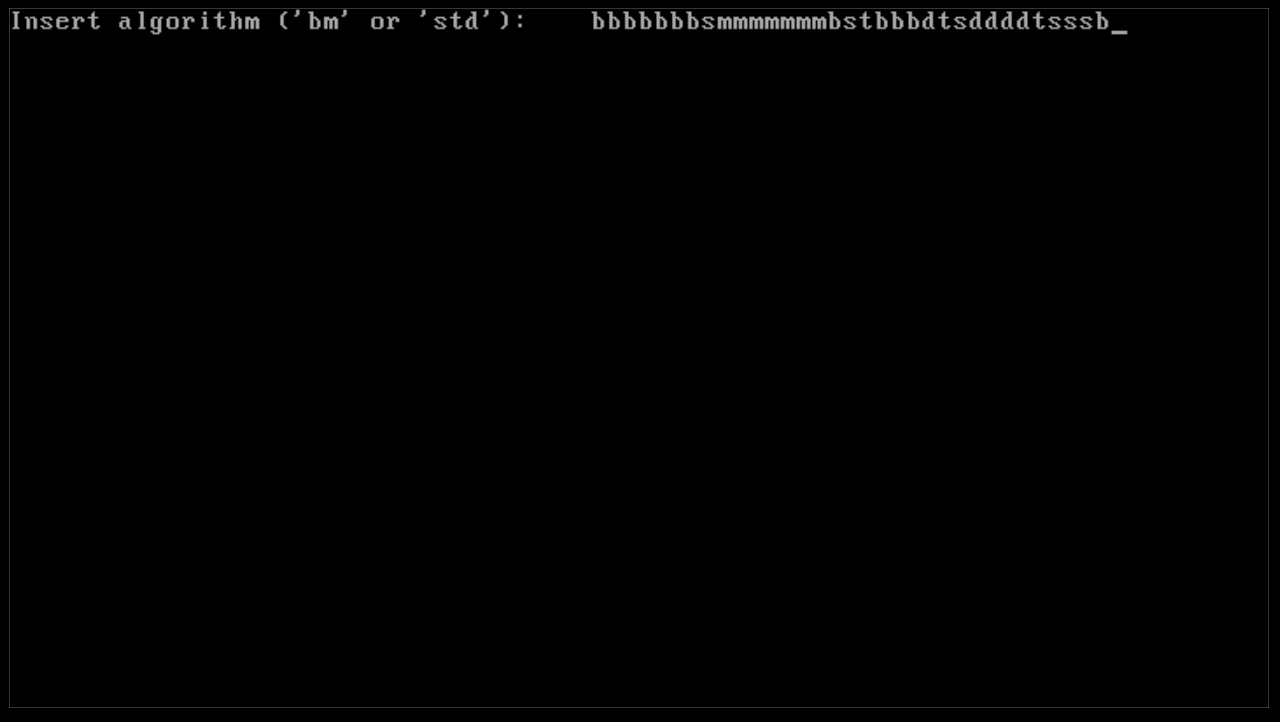


Рисунок 2 – Входные параметры загрузчика

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Тест первых двух команд (help и info) и отсутствие команды

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Тест остальных команд с параметром std.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Тест команд поиска подстроки в строке с параметром bm

При тестировании функции shutdown виртуальный компьютер выключился и окно эмулятора qemu закрылось.

Вывод

В ходе выполнения данной лабораторной работы были изучены основы разработки операционных систем, а именно написан код на языке Ассемблер для загрузчика системы и код на языке С++ для ядра ОС. Были изучены такие особенности загрузчика, как выделение памяти, переход в защищённый режим работы процессора, в частности познакомились с адресной линией А20. Также были приобретены навыки по написанию программ на языке ассемблера в синтаксисе Intel для транслятора GNU.

В процессе реализации ядра операционной системы были более подробно изучены механизмы прерываний, в частности необходимо было поработать с контроллером прерываний.

Приложение

1. **Код скрипта**

// компиляция ядра

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

// линковка ядра

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

// компиляция загрузчика

as --32 -o bootsect.o bootsect.asm

// линкова загрузчика

ld -Ttext 0x7c00 --oformat binary -m elf\_i386 -o bootsect.bin bootsect.o

// запуск эмуляции на qemu загрузчика и ядра в формате raw и floppy

qemu -drive format=raw,file=bootsect.bin,index=0,if=floppy -drive format=raw,file=kernel.bin,index=1,if=floppy

1. **Листинг загрузчика**

.code16

.intel\_syntax noprefix

.global \_start

\_start:

# clear

mov ax, 0x3 # Функция очистки консоли

int 0x10 # Вызов прерывания 0x10 - видео сервис

# print loading

mov bx, offset loading\_str # Загрузка в bx строки loading\_str

call puts # Вызов функции puts (вывод строки, лежащей в bx)

# input symbols

call input # Вызов функции input

# clear

mov ax, 03 # Функция очистки консоли

int 0x10 # Вызов прерывания 0x10 - видео сервис

mov dl, 0x01 # Номер диска (носителя)

mov dh, 0x00 # Номер головки

mov ch, 0x00 # Номер цилиндра (дорожки)

mov cl, 0x01 # 2 - номер цил. 6 - номер сектора

mov al, 0x30 # Количество сектров, взято 48 секторов

mov bx, 0x1000 # Адрес буфера, в который считываются данные (со смещением) 0x1000:0x0000

mov es, bx # В данном случае адрес со смещением будет равен 0x10000

xor bx, bx

mov ah, 0x02 # Функция - считывание заданного количества секторов с диска в память.

int 0x13 # Вызов прерывания 0x13 - дисковый ввод/вывод

# Переход в защищённый режим

cli # Обязательное отключение прерываний

lgdt gdt\_info # Загрузка размера и адреса таблицы дескрипторов

in al, 0x92 # Включение адресной линии А20

or al, 2 #

out 0x92, al #

mov eax, cr0 # Установка бита PE регистра CR0 - процессор перейдет в защищенный режим

or al, 1 #

mov cr0, eax #

jmp 0x8:protected\_mode # "Дальний" переход для загрузки корректной информации в cs

gdt:

# Нулевой дескриптор

.byte 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

.byte 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

.byte 0xff, 0xff, 0x00, 0x00, 0x00, 0x92, 0xCF, 0x00

gdt\_info:

.word gdt\_info - gdt # Размер таблицы (2 байта)

.word gdt, 0 # 32-битный физический адрес таблицы.

# Функция ввода до 'bm' или 'std' и помещение флага в память (0x9004)

input:

call shl\_ # Битовый сдвиг на 4 бита влево

mov ah, 0x00 # Функция ожидания нажатия и считывание нажатой клавиши

int 0x16 # Вызов прерывания 0x16 - работа с клавиатурой

add buffer[0], al # В младший байт 4 байтного слова "кладём" введённый символ

mov ah, 0x0e # Функция вывода на текстовый экран символа из al

int 0x10 # Вызов прерывания 0x10 - видео сервис

mov al, buffer[0] # Присваиваем регистру al значение введённого символа

cmp al, 'm' # Если в al НЕ символ 'm', то:

jne std\_check # Переходим в функцию std\_check для проверки последнего введённого символа и перед ним

mov al, buffer[1] #

cmp al, 'b' # Проверка прошлого введённого символа, равен ли он 'b'

jne std\_check #

mov bx, 0x9004 # В регистр bx помещается адрес 0x9004 (выбран произвольно)

mov cx, 1 # В регистр cx помещаем значение 1, если была введена 'bm'

mov [bx], cx # По адресу 0x9004 помещаем значение cx == 1

ret # Возвращаемся из функции

# Функция проверки на то, введена ли последовательность 'std'

std\_check:

mov al, buffer[0] # Присваиваем регистру al значение последнего введённого символа

cmp al, 'd' # Если в al НЕ символ 'd', то:

jne input # Переходим в функцию input для ввода нового символа

mov al, buffer[1] # Присваиваем регистру al значение предпоследнего введённого символа

cmp al, 't' # Если в al НЕ символ 't', то:

jne input # Переходим в функцию input для ввода нового символа

mov al, buffer[2] # Присваиваем регистру al значение предпредпоследнего введённого символа

cmp al, 's' # Если в al НЕ символ 's', то:

jne input # Переходим в функцию input для ввода нового символа

mov bx, 0x9004 # В регистр bx помещается адрес 0x9004 (выбран произвольно)

mov cx, 0 # В регистр cx помещаем значение 0, если была введена 'std'

mov [bx], cx # По адресу 0x9004 помещаем значение cx == 0

ret # Возвращаемся из функции

# Функция битового сдвига на 4 бита влево буфера, где хранится 4 введённых байта

shl\_:

mov al, buffer[2]

mov buffer[3], al

mov al, buffer[1]

mov buffer[2], al

mov al, buffer[0]

mov buffer[1], al

mov al, 0x00

mov buffer[0], al

ret

# Функция вывода на экран строки, адрес которой лежит по адресу в bx

puts:

mov al, [bx] # В регистр al помещается символ строки по адресу [bx]

test al, al # Проверка - конец ли строки ('\0')

jz end\_puts # Если да, то выходим из функции

mov ah, 0x0e # Функция вывода на текстовый экран символа из al

int 0x10 # Вызов прерывания 0x10 - видео сервис

add bx, 1 # Переход к следующему символу строки по адресу [bx]

jmp puts # Возврат к началу функции по циклу

end\_puts:

ret

loading\_str:

.asciz "Insert algorithm ('bm' or 'std'): "

# Буфер, в котором находятся 4 последних введённых символа с клавиатуры

buffer:

.long 0x00000000

# Передача управления от загрузчика ядру

.code32

protected\_mode:

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

inf\_loop:

jmp inf\_loop # Бесконечный цикл

.zero (512 - ($ - \_start) - 2) # Заполнение нулями до границы 512 - 2 текущей точки

.byte 0x55, 0xAA # 2 необходимых байта чтобы сектор считался загрузочным

1. **Листинг ядра**

\_\_asm("jmp kmain");

#define VIDEO\_BUF\_PTR (0xb8000)

#define IDT\_TYPE\_INTR (0x0E)

#define IDT\_TYPE\_TRAP (0x0F)

// Селектор секции кода, установленный загрузчиком ОС

#define GDT\_CS (0x8)

#define PIC1\_PORT (0x20)

// Базовый порт управления курсором текстового экрана. Подходит для большинства, но может отличаться в других BIOS и в общем случае адрес должен быть прочитан из BIOS data area.

#define CURSOR\_PORT (0x3D4)

#define VIDEO\_WIDTH (80) // Ширина текстового экрана

#define ALPHABET\_SIZE 256

#define max(a, b) ((a > b) ? a : b)

unsigned int strnum\_ = 2; // Текущая строка курсора

unsigned int pos\_ = 0; // Текущая позиция курсора в строке strnum\_

unsigned char tmpl[40] = {'\0'}; // Текущий template

int \*param = (int \*) 0x9004; // Переменная, которая хранит в себе параметр, введённый на этапе загрузчика

// Массив для перевода скан-кодов в ASCII таблицу

char kbd\_US [128] =

{

0, 27, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', '\b', '\t', /\* <-- Tab \*/

'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']', '\n',

0, /\* <-- control key \*/

'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ';', '\'', '`',

15, /\* Shift \*/

'\\', 'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.', '/', 0,

'\*',

0, /\* Alt \*/

' ', /\* Space bar \*/

0, /\* Caps lock \*/

0, /\* 59 - F1 key ... > \*/

0, 0, 0, 0, 0, 0, 0, 0,

0, /\* < ... F10 \*/

0, /\* 69 - Num lock\*/

0, /\* Scroll Lock \*/

0, /\* Home key \*/

0, /\* Up Arrow \*/

0, /\* Page Up \*/

'-',

0, /\* Left Arrow \*/

0,

0, /\* Right Arrow \*/

'+',

0, /\* 79 - End key\*/

0, /\* Down Arrow \*/

0, /\* Page Down \*/

0, /\* Insert Key \*/

0, /\* Delete Key \*/

0, 0, 0,

0, /\* F11 Key \*/

0, /\* F12 Key \*/

0, /\* All other keys are undefined \*/

};

unsigned char all\_comm[8][40] // Таблица комманд

{

"help",

"info",

"upcase",

"downcase",

"titlize",

"template",

"search",

"shutdown"

};

enum comm\_numb // Перечисление для команд в switch-case

{

HELP,

INFO,

UPCASE,

DOWNCASE,

TITLIZE,

TEMPLATE,

SEARCH,

SHUTDOWN

};

// Функция вывода строки на экран

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

{

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

video\_buf += strnum \* VIDEO\_WIDTH \* 2;

while (\*ptr)

{

if (\*ptr == '\n')

{

strnum++;

strnum\_++;

video\_buf = (unsigned char\*) VIDEO\_BUF\_PTR + strnum \* VIDEO\_WIDTH \* 2;

}

else

{

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

video\_buf[1] = color;

video\_buf += 2;

}

ptr++;

}

}

// Функция вывода символа на экран

void out\_symb(int color, unsigned char symb, unsigned int strnum, unsigned int pos)

{

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

video\_buf += (strnum \* VIDEO\_WIDTH\*2) + pos \* 2;

video\_buf[0] = symb;

video\_buf[1] = color;

}

// 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

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

void default\_intr\_handler()

{

asm("pusha");

// ... (реализация обработки)

asm("popa; leave; iret");

}

// Регистрация определённого прерывания, задаваемого программистом

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) );

}

// Включить прерывания

void intr\_enable()

{

asm("sti");

}

// Выключить прерывания

void intr\_disable()

{

asm("cli");

}

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 в порт байта

{

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

}

static inline void outw (unsigned short port, unsigned short int data) // Запись в порт двух байтов

{

asm volatile ("outw %0, %1" : : "a" (data), "d" (port));

}

// 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)

{

unsigned short 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));

}

unsigned char command[40]; // Хранение текущей команды

int ind = 0;

int shift = 0; // Флаг нажатия кнопки SHIFT

// Очистка текстового экрана

void clear()

{

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

for (int i = 0; i < 2000; i++)

{

video\_buf[0] = ' ';

video\_buf[1] = 0x07;

video\_buf += 2;

}

}

// Сравнение строк

int strcmp\_(unsigned char \*s1, unsigned char \*s2)

{

int i = 0;

while (s1[i] != '\0' && s2[i] != '\0')

{

if (s1[i] != s2[i])

break;

i++;

}

return !(s1[i] == '\0' && s2[i] == '\0');

}

// Копирование строки из src в dst

void strcpy\_(unsigned char \*dst, const char \*src) {

int i = 0;

while (src[i] != '\0')

{

dst[i] = src[i];

i++;

}

dst[i] = '\0'; // добавляем нулевой символ

}

// Добавление второй строки к концу первой

void strcat\_(unsigned char \*str1, const char \*str2)

{

unsigned char\* begin = str1;

while (\*str1)

str1++;

while(\*str1++ = \*str2++);

\*str1 = '\0';

}

// Вычисление длины строки

int strlen\_(unsigned char \*str1)

{

int len = 0;

while (\*str1++) len++;

return len - 1;

}

// Функция для замены двух чисел

void swap(char \*x, char \*y) {

char t = \*x; \*x = \*y; \*y = t;

}

// Функция для обращения `buffer[i…j]`

char\* reverse(char \*buffer, int i, int j)

{

while (i < j) {

swap(&buffer[i++], &buffer[j--]);

}

return buffer;

}

// Итеративная функция для реализации функции `int\_to\_char()` в C

char\* int\_to\_char(int value, char\* buffer, int base)

{

// неправильный ввод

if (base < 2 || base > 32) {

return buffer;

}

// считаем абсолютное значение числа

int n = value;

int i = 0;

while (n)

{

int r = n % base;

if (r >= 10) {

buffer[i++] = 65 + (r - 10);

}

else {

buffer[i++] = 48 + r;

}

n = n / base;

}

// если число равно 0

if (i == 0) {

buffer[i++] = '0';

}

// Если основание равно 10 и значение отрицательное, результирующая строка

// предшествует знак минус (-)

// При любой другой базе значение всегда считается беззнаковым

if (value < 0 && base == 10) {

buffer[i++] = '-';

}

buffer[i] = '\0'; // нулевая завершающая строка

// переворачиваем строку и возвращаем ее

return reverse(buffer, 0, i - 1);

}

// Функция заполнения таблицы смещений для Б-М

void precompute\_bad\_chars(char \*str, int size, int bad\_char\_table[]) {

// Заполняем таблицу смещений для всех символов алфавита

for (int i = 0; i < ALPHABET\_SIZE; i++)

bad\_char\_table[i] = -1;

// Заполняем таблицу смещений только для символов из подстроки

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

bad\_char\_table[(int) str[i]] = i;

}

int bad\_char\_table[ALPHABET\_SIZE];

// Исполнение команды

void action()

{

unsigned char temp\_comm[40];

int i = 0;

while (command[i] != ' ' && i < 39)

{

temp\_comm[i] = command[i];

i++;

}

temp\_comm[i++] = '\0';

int ans;

for (ans = 0; ans < 8; ans++)

{

if(strcmp\_(temp\_comm, all\_comm[ans]) == 0)

break;

}

switch (ans)

{

case HELP:

{

const char \*text = "Command List:\n- help: displays a list of commands.\n- info: Displays information about the author and OS development tools.\n- upcase [string]: Prints the specified string in upper case.\n- donwcase [string]: Prints the specified string in lower case.\n- titlize [string]: Prints the string, the first letter in each word is big.\n- template [template]: Load a template for searching substrings into memory.\n- search [string]: First occurrence of the temp in the specified string.\n- shutdown: Shut down the computer";

out\_str(0x07, text, ++strnum\_);

break;

}

case INFO:

{

unsigned char text[256] = "String OS: v.01. Developer: Vyacheslav Totski, 4851003/10002, SpbPU, 2023\nCompilers: bootloader: GNU (Intel), kernel: gcc\nBootloader parameters: ";

if (\*param==0)

strcat\_(text, "std");

else

strcat\_(text, "bm");

out\_str(0x07, (const char\*)text, ++strnum\_);

break;

}

case UPCASE:

{

unsigned char symb;

strcpy\_(temp\_comm, (const char\*)(&command[i]));

i = 0;

while (temp\_comm[i] != '\0')

{

if (temp\_comm[i] >= 'a' && temp\_comm[i] <= 'z')

temp\_comm[i]-= 32;

i++;

}

out\_str(0x07, (const char\*)temp\_comm, ++strnum\_);

break;

}

case DOWNCASE:

{

unsigned char symb;

strcpy\_(temp\_comm, (const char\*)(&command[i]));

i = 0;

while (temp\_comm[i] != '\0')

{

if (temp\_comm[i] >= 'A' && temp\_comm[i] <= 'Z')

temp\_comm[i]+= 32;

i++;

}

out\_str(0x07, (const char\*)temp\_comm, ++strnum\_);

break;

}

case TITLIZE:

{

unsigned char symb;

strcpy\_(temp\_comm, (const char\*)(&command[i]));

i = 0;

if (temp\_comm[i] >= 'a' && temp\_comm[i] <= 'z' )

temp\_comm[i++] -= 32;

while (temp\_comm[i] != '\0')

{

if (temp\_comm[i] >= 'a' && temp\_comm[i] <= 'z' && temp\_comm[i-1] == ' ')

temp\_comm[i] -= 32;

i++;

}

out\_str(0x07, (const char\*)temp\_comm, ++strnum\_);

break;

}

case TEMPLATE:

{

strcpy\_(tmpl, (const char\*)(&command[i]));

i = 0;

unsigned char text[40] = "Template '\0";

strcat\_(text, (const char\*)tmpl);

strcat\_(text, "' loaded.");

if (\*param == 0)

{

out\_str(0x07, (const char\*)text, ++strnum\_);

break;

}

int len1 = strlen\_(tmpl);

char c[2];

precompute\_bad\_chars((char\*)tmpl, len1, bad\_char\_table);

strcat\_(text, " BM info:");

out\_str(0x07, (const char\*)text, ++strnum\_);

strnum\_++;

for (int index = 0; index <= len1; index++)

{

out\_symb(0x07, tmpl[index], strnum\_, pos\_++);

out\_symb(0x07, ':', strnum\_, pos\_++);

char b[2];

out\_symb(0x07, int\_to\_char(len1 + 1 - bad\_char\_table[(int) tmpl[index]], b, 10)[0], strnum\_, pos\_++);

out\_symb(0x07, ' ', strnum\_, pos\_++);

}

pos\_ = 0;

break;

}

case SEARCH:

{

int index = 0;

int answer = 0;

unsigned char text[40];

strcpy\_(text, (const char\*)(&command[i]));

int len1 = strlen\_(tmpl);

int len2 = strlen\_(text);

if (\*param == 0)

{

int j;

for (index = 0; index <= len2 - len1; index++)

{

for (j = 0; j < len1 + 1 && text[index + j] == tmpl[j]; j++);

if (j == len1 + 1)

{

answer = 1;

break;

}

}

}

else

{

while (index <= len2 - len1)

{

int j = len1;

while (j >= 0 && tmpl[j] == text[index + j])

j--;

if (j < 0)

{

answer = 1;

break;

}

index += max(1, j - bad\_char\_table[text[index + j]]);

}

}

if (answer)

{

strcpy\_(text, "Found '");

strcat\_(text, (const char\*)tmpl);

strcat\_(text, "' at pos: ");

char a[2];

int\_to\_char(index, a, 10);

strcat\_(text, (const char\*)a);

out\_str(0x07, (const char\*)text, ++strnum\_);

}

else

{

strcpy\_(text, "Not found '");

strcat\_(text, (const char\*)tmpl);

strcat\_(text, "'");

out\_str(0x07, (const char\*)text, ++strnum\_);

}

break;

}

case SHUTDOWN:

{

const char \*text = "Powering off...";

out\_str(0x07, text, ++strnum\_);

outw(0x604, 0x2000);

break;

}

default:

const char \*text = "Error: command not recognized";

out\_str(0x07, text, ++strnum\_);

}

}

// Обработка нажатой клавиши

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

unsigned char symb = kbd\_US[scan\_code];

if (strnum\_>=24)

{

strnum\_ = 0;

clear();

}

// BackSpace

if (symb == '\b')

{

if (pos\_ > 0)

{

pos\_--;

command[ind--] = '\0';

}

out\_symb(0x07, ' ', strnum\_, pos\_);

cursor\_moveto(strnum\_, pos\_);

}

// Enter

if (symb == '\n')

{

pos\_ = 0;

if (command[0] != '\0')

{

action();

command[0] = '\0';

ind = 0;

strnum\_++;

}

strnum\_++;

cursor\_moveto(strnum\_, pos\_);

}

// Ограничение ввода больше 40-ка символов

if (pos\_ >= 40)

{

return;

}

if (symb == 15)

{

shift = 1;

}

// a-z, 1-0

if (symb >= '!' && symb <= '~' || symb == ' ')

{

if (shift)

{

if (symb >= 'a' && symb <= 'z')

symb-=32;

}

command[ind++] = symb;

command[ind] = '\0';

out\_symb(0x07, symb, strnum\_, pos\_);

cursor\_moveto(strnum\_, ++pos\_);

}

}

// Обработка отпущенных клавиш

void un\_key(unsigned char scan\_code)

{

if (scan\_code == 170)

shift = 0;

}

// Считывание скан-кода нажатой или отпущенной клавиши

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); // Считывание символа с PS/2 клавиатуры

if (scan\_code < 128) // Скан-коды выше 128 - это отпускание клавиши

on\_key(scan\_code);

else

un\_key(scan\_code);

}

}

void keyb\_handler()

{

asm("pusha");

// Обработка поступивших данных

keyb\_process\_keys();

// Отправка контроллеру 8259 нотификации о том, что прерывание обработано

outb(PIC1\_PORT, 0x20);

asm("popa; leave; iret");

}

void keyb\_init()

{

// Регистрация обработчика прерывания

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

// segm\_sel=0x8, P=1, DPL=0, Type=Intr

// Разрешение только прерываний клавиатуры от контроллера 8259

outb(PIC1\_PORT + 1, 0xFF ^ 0x02); // 0xFF - все прерывания, 0x02 - бит IRQ1 (клавиатура).

// Разрешены будут только прерывания, чьи биты установлены в 0

}

extern "C" int kmain()

{

unsigned char hello[40] = "Welcome to StringOS!";

out\_str(0x07, (const char\*)hello, 0);

strcpy\_(hello, "To list commands, write help\0");

out\_str(0x07, (const char\*)hello, 1);

cursor\_moveto(strnum\_, pos\_);

intr\_enable();

keyb\_init();

intr\_start();

asm("hlt");

return 0;

}