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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «**Принципы разработки операционных систем**»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. №4851003/00002 Скрипко И.А.

<*подпись*>

1. Преподаватель
2. Крундышев В.М.

<*подпись*>

1. Санкт-Петербург
2. 2022
3. **Цель работы**

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

1. **Ход работы**
   1. **Подготовка к выполнению лабораторной работы**

Был получен вариант выполнения работы со следующими требованиями:

* Ваш вариант задания: 8
* Работа выполняется в ОС Linux
* Транслятор ассемблера для загрузчика: YASM, синтаксис: AT&T
* Компилятор ядра: gcc
* Задание для реализации: SolverOS

Для трансляции кода загрузчика в бинарный код использовались следующие команды:

yasm -p gas -f bin -o bootsect.tmp bootsect.asm

dd bs=31744 skip=1 if=bootsect.tmp of=bootsect.bin

Код ядра компилировался при помощи следующих команд:

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

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

Для проверки работоспособности на эмуляторе QEMU использовалась следующая команда:

qemu –fda bootsect.bin –fdb kernel.bin

* 1. **Алгоритм работы загрузчика**

На рисунке 1 представлена блок-схема алгоритма работы загрузчика.

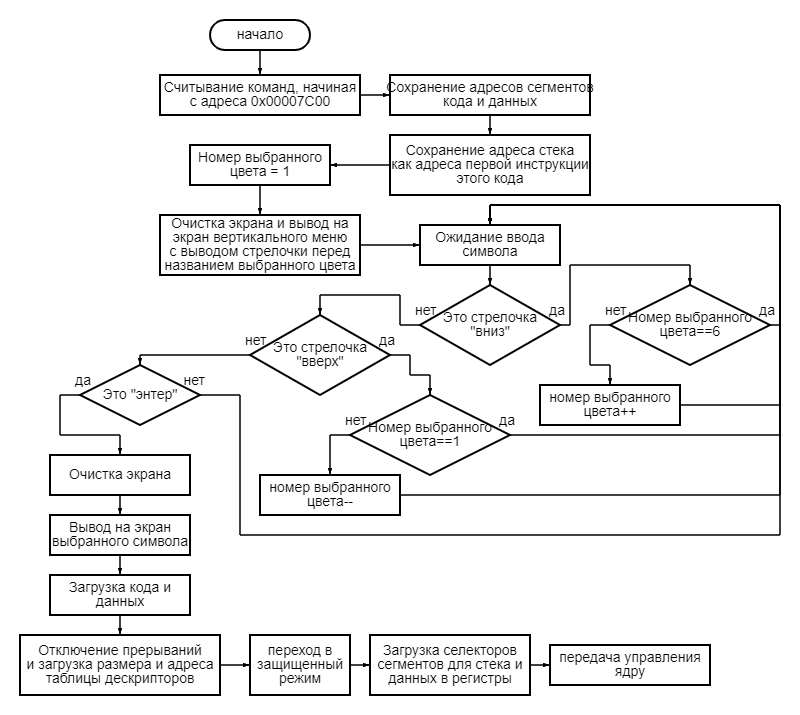


Рисунок 1 – Алгоритм работы загрузчика

В приложении А представлен комментированный листининг загрузчика на ассемблере.

По заданию к лабораторной работе в загрузчике должно выводиться вертикальное меню, и пользователь с помощью стрелочек может передвигаться по нему, а с помощью энтера производит выбор цвета, после чего выбранный цвет должен сохраниться по произвольно выбранному адресу.

Для вывода вертикального меню используется функция BIOS для печати символа на экран. С помощью этой функции печатается строка. Далее выводится 2 символа – перенос строки и откат каретки. Эти символы выводятся для того, чтобы название следующего цвета выводилось на следующей строке. Перед печатью названия строки производится сравнение на данный момент выбранного пользователем цвета (выбранный цвет хранится в виде числа), если планируется вывод выбранного цвета, то перед ним выводится знак «>», который будет указывать пользователю цвет, который он выбрал.

После вывода меню используется функция BIOS для ожидания ввода пользователем символа, после чего проверяется введенный им символ. Алгоритм работы программы в зависимости от введенного символа представлен в блок-схеме на рисунке 1.

После выбора цвета его номер сохраняется в память, предназначенную для вывода информации на экран, оттуда его считает потом ядро и очистит экран перед началом своей работы.

* 1. **Алгоритм работы ядра**

В лабораторной работе следующие требования к ядру:

1. ОС должна считать параметры, сохраненные загрузчиком, и применить их.

2. Настроить прерывания и включить их.

3. Перейти в интерактивный режим: на экране должна быть строка приветствия. Пользователь вводит в нее команду, а ОС отвечает на нее. Команда состоит из ключевого слова и параметров (для некоторых команд) – указываются через пробел. Если команда не распознана — должны выводиться сведения об этом (ошибка).

4. ОС не позволяет вводить команды длиннее 40 символов. При попытке ввести команду длиннее лимита – ОС не будет реагировать на лишние нажатия. Должна поддерживаться клавиша Backspace.

5. Если все строки текущего экрана заполнены – экран очищается и взаимодействие с пользователем продолжается вновь, на чистом экране.

В соответствии с вариантом задания ядро должно поддерживать следующие функции:

* info – выводит информацию об авторе и средствах разработки ОС
* gcd arg1 arg2 – находит НОД чисел dig1 и dig2.
* solve equation – решает линейное уравнение вида ax+b=c. Если решение вещественное – выводится дробь. Дробь должна быть несокращаемая.
* shutdown – прекращает работу ОС.

В начале работы ОС считывает с экрана число, которое там сохранено загрузчиком – переменная color принимает значение считанного числа. Далее экран очищается и текст на экране инициализируется нужным цветом. Далее выводится приветственная строка, после чего инициализируются и настраиваются прерывания. Далее прерывания включаются, а ОС переходит в бесконечный цикл ожидания.

При нажатии пользователем на клавишу срабатывает прерывание, которое считывает значение с порта ввода и обрабатывает его – если нажат backspace, то очищает последний символ и переносит курсор, если нажат enter, то считывается введенная строка и обрабатывается, в противном случае обрабатывается как обычный символ – если он корректный и в строке введено меньше 40 символов, то выведется на экран, иначе проигнорируется.

Как было сказано выше, при нажатии на enter считывается введенная строка и обрабатывается. ОС поддерживает 4 функции указанные выше. Ядро сравнивает введенную пользователем строку и выбирает нужную команду, после чего обрабатывает аргументы. Если команда или аргументы некорректные, то выводит сообщение об ошибке, в противном случае обрабатывает и выводит корректный ответ на запрос пользователя.

*Функция info.*

Реализована с помощью вывода 3 строк на экран. При этом добавлена проверка на количество оставшегося места на экране. В программе цветом инициализируется 23 строки, поэтому при заполненных 21 и больше строк, сначала произойдет очистка экрана, после чего выведутся 3 строки.

*Функция gcd arg1 arg2.*

Обрабатываются 2 аргумента и сохраняются в 2 строки. Если первый символ одной из строк «-», то выводится ошибка некорректно введенных данных. Если две строки положительны, то вызывается функция для перевода строк в числа.

В этой функции выполняется проверка длины строк (в функции используются числа short, поэтому строки длиной больше 5 являются переполнением), в случае слишком большой длины выводится ошибка переполнения. Далее проверяются строки на содержание только символов-цифр – если строки содержат не только их, то выводится сообщение о некорректных аргументах. Далее, если длина равна 5, то происходит сравнение строки с максимально возможной строкой. В случае ошибки выводится сообщение о переполнении. Если все корректно, то строка переводится в число и возвращает результат в виде возвращаемого значения.

После перевода строк в числа вызывается функция для нахождения НОД для этих чисел. После нахождения НОД результат переводится в строку, с помощью написанной для этого функции, и выводится на экран.

*Функция solve equation.*

Обрабатываются аргументы уравнения и переводятся в строки, как и в функции gcd. Если числа отрицательные, то сохраняется минус, после чего в функцию перевода строки в число передается указатель на модуль числа. Далее результат домножится на -1. Если аргументы корректные, то далее производится действие c-b (перенос слагаемого b в правую часть уравнения), далее деление на a.

Деление на a может быть целочисленным и нецелочисленным. При целочисленном все хорошо и получается ответ – он переводится в строку, если ответ отрицательный, то в строке также ставится минус, после чего она выводится на экран. Если ответ нецелочисленный, то сохраняется знак ответа, после чего запускается цикл поиска НОД числителя и знаменателя и деление на него числителя и знаменателя до тех пор, пока НОД не будет равен 1. Далее вызывается специально написанная функция для перевода 2 чисел в строчный вид дроби. После этого производится вывод на экран ответа.

* 1. **Написанные функции для ОС**

В таблице 1 представлены написанные для ОС функции и их описания.

Таблица 1 – Функции ОС

|  |  |  |
| --- | --- | --- |
| Имя функции | Принимаемые аргументы | Описание |
| void write\_string(unsigned char\* text); | Строка | Выводит строку на экран. |
| void init\_text\_clr(); | - | Устанавливает цвет текста ОС. |
| void clear\_screen(); | - | Очищает содержимое экрана. |
| void info(); | - | Выводит информацию об авторе, выбранном цвете и средствах разработки. |
| void choose\_command(); | - | Проверяет на корректность введенную команду и аргументы, в случае обнаружения ошибки выводит об этом сообщение, если все корректно, то обрабатывает запрос и выводит ответ. |
| short str\_to\_number(const unsigned char\* str, const unsigned char size); | Строка, размер строки | Переводит строку в число. Число должно быть положительным. Если переданная строка некорректная, то вернет -1. Если переполнение, то -2. В противном случае число, в которое была переведена строка. |
| void number\_to\_str(short n, unsigned char\* str); | Число, строка | Переводит переданное число в строку. Число должно быть положительным. Результат сохраняется в str. |
| short nod(short n1, short n2); | Число1, число2 | Находит НОД чисел 1 и 2. |
| void make\_fraction(const unsigned char\* num, const unsigned char\* den, unsigned char\* out); | Строка1, строка2, строка3 | Переводит строки 1 и 2 в вид дроби и сохраняет в строку3. |

1. **Результаты тестирования**

На рисунках 2-4 представлены результаты работы разработанной ОС. На рисунке 3 также можно наблюдать, что надписи «Welcome to SolverOS!» нет, так как перед вызовом этой функции экран был заполнен, так что он очистился, прежде чем выводить результат работы функции.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Результат работы функции gcd

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Результаты работы функции info

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Результаты работы функции solve

1. **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 SolverOS, выполняющая функции поиска НОД и решения уравнения.

**Приложение А**

Листинг «bootsect.asm»

.org 0x7c00 #указание адреса загрузки кода

.code16 #переход в 16-битный режим

\_start:

#инициализация адресов сегментов

#синтаксис: инструкция приемник, источник

#перед регистрами ставится знак "%"

#перед константами ставится знак "$"

#каждая команда пишется с суффиксом, указывающим размер операндов команды, movw - суффикс w - размер 2 байта

movw %cs, %ax #сохранение адреса сегмента кода в ax

movw %ax, %ds #сохранение этого адреса как начало сегмента данных

movw %ax, %ss #и как начало сегмента стека

movw $\_start, %sp #сохранение адреса стека в качестве адреса первой инструкции кода, после чего стек будет расти вверх

movb $0x01, %ch

#нужная строка помещается в bx (строка зависит от выбранного цвета)

#далее вызывается функция puts для вывова цвета на экран

#так выводятся по очереди все цвета, у выбранного цвета дополнительно выводится стрелочка

#после чего происходит прыжок в choose для выбора цвета

movb $0x01, %ch #счетчик для выбора цвета

menu:

movw $0x0003, %ax #очистить экран

int $0x10

#вывести меню

cmp $0x01, %ch #если выбран этот цвет

jne g #если выбран другой цвет

call this #если этот, то вывести стрелочку перед цветом

g:

movw $gray, %bx

call puts

call next\_str

cmp $0x02, %ch

jne wh

call this

wh:

movw $white, %bx

call puts

call next\_str

cmp $0x03, %ch

jne y

call this

y:

movw $yellow, %bx

call puts

call next\_str

cmp $0x04, %ch

jne c

call this

c:

movw $cian, %bx

call puts

call next\_str

cmp $0x05, %ch

jne m

call this

m:

movw $magenta, %bx

call puts

call next\_str

cmp $0x06, %ch

jne gr

call this

gr:

movw $green, %bx

call puts

call next\_str

jmp choose

#функция для выбора цвета

choose:

movb $0x00, %ah #ожидание нажатия и считывание нажатой пользователем клавиши

int $0x16 #работа с клавиатурой

cmp $0x48, %ah #up

je up

cmp $0x50, %ah #down

je down

cmp $0x1c, %ah #enter

je loader

jmp choose

#функция выводит стрелочку, указывая выбранный цвет

this:

movb $0x3e, %al #символ, который будет напечатан

movb $0x0e, %ah #печать символа на экран

int $0x10 #вызов прерывания для печати

ret

#вывод цвета

puts:

movb 0(%bx), %al #записываем в al нынешний символ

test %al, %al #логическое и (результат находится в флаге ZF)

jz end\_puts #если было 0 и 0

movb $0x0e, %ah #печать символа на экран

int $0x10 #вызов прерывания для печати

addw $1, %bx #переход к следующему символу

jmp puts #вывод следующего символа

end\_puts:

ret

#выбор нижнего цвета

down:

addb $0x01, %ch

cmp $7, %ch

je so\_many\_down

jmp menu

so\_many\_down:

movb $6, %ch

jmp menu

#выбор верхнего цвета

up:

subb $0x01, %ch

cmp $0, %ch

je so\_many\_up

jmp menu

so\_many\_up:

movb $1, %ch

jmp menu

#перенос строки и откат каретки для вывода следующего цвета на след строке

next\_str:

movb $0x0a, %al #символ переноса строки

movb $0x0e, %ah #печать символа на экран

int $0x10 #вызов прерывания для печати

movb $0x0d, %al #символ отката каретки

movb $0x0e, %ah #печать символа на экран

int $0x10 #вызов прерывания для печати

ret

#объявление строк, после их автоматически ставится нулевой байт

gray:

.asciz "Gray"

white:

.asciz "White"

yellow:

.asciz "Yellow"

cian:

.asciz "Cian"

magenta:

.asciz "Magenta"

green:

.asciz "Green"

loader:

movw $0x0003, %ax #очистить экран

int $0x10

movb %ch, %al #символ переноса строки

movb $0x0e, %ah #печать символа на экран

int $0x10 #вызов прерывания для печати

movb $1, %dl #номер диска

movb $0, %dh #номер головки

movb $0, %ch #номер цилиндра

movb $1, %cl #номер сегмента

movb $17, %al #сколько секторов считать

movw $0x1000, %bx

movw %bx, %es

movw $0, %bx

movb $0x02, %ah

int $0x13

cli #отключение прерываний

lgdt gdt\_info #загрузить таблицу

#Включение адресной линии A20

inb $0x92, %al

orb $2, %al

outb %al, $0x92

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

movl %cr0, %eax

orb $1, %al

movl %eax, %cr0

jmp $0x8, $protected\_mode

gdt:

#Нулевой дескриптор

.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

#Сегмент кода

.byte 0xff, 0xff, 0x00, 0x00, 0x00, 0x9A, 0xCF, 0x00

#Сегмент данных

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

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

.word gdt\_info - gdt #Размер таблицы

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

.code32

protected\_mode:

#Загрузка селекторов сегментов для стека и данных в регистры

movw $0x10, %ax

movw %ax, %es

movw %ax, %ds

movw %ax, %ss

#Передача управления ядру

call 0x10000

#Сектор будет считаться загрузочным, если содержит в конце своих 512 байтов два следующих байта: 0x55 и 0xAA

.zero (512 -(. - \_start) - 2) #заполнение 510 изи 512 байтов нулями

.byte 0x55, 0xAA #заполнение последних вух байт так, чтобы сектор считался загрузочным

**Приложение Б**

Листинг «kernel.cpp»

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

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

\_\_asm("jmp kmain");

#define ADDR\_VIDEO\_BUF 0xb8000 //адрес начала памяти для работы с виеоадаптером

#define PIC1\_PORT 0x20

//Базовый порт управления курсором текстового экрана

#define CURSOR\_PORT 0x3D4

#define VIDEO\_WIDTH 80//Ширина текстового экрана

#define VIDEO\_HEIGHT 48//высота текстового экрана

//коды символов

#define BACKSPACE 14

#define ENTER 28

#define SPACE 57

//для работы с курсором

unsigned char line = 0;

unsigned char column = 0;

//цвет шрифта

unsigned char color = 0;

//Селектор секции кода, установленный загрузчиком ОС

#define GDT\_CS 0x8

#define IDT\_TYPE\_INTR 0x0E

//#define IDT\_TYPE\_TRAP 0x0F

//массив для перевода символов в ascii

unsigned char scan\_codes[] =

{

0,

0, // ESC(this button starts with number "1")

'1','2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=',

0, // BACKSPACE(14)

0, // TAB(15)

'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', 0, 0,

' ',// ENTER(28)

0, // CTRL(29)

'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 0,0,'+',

0, // left SHIFT(42)

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

0, // right SHIFT(54)

'\*',// NUMPAD \*

0, // ALT

' ', // SPACE

0, //CAPSLOCK

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //F1 - F10

0, //NUMLOCK

0, //SCROLLOCK

0, //HOME

0,

0, //PAGE UP

'-', //NUMPAD

0, 0,

0, //(r)

'+', //NUMPAD

};

//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

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

// Пустой обработчик прерываний

void default\_intr\_handler()

{

asm("pusha");

asm("popa; leave; iret");

}

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

//прерывания

//void default\_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();

//работа с символами

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\_init();

void keyb\_handler();

void keyb\_process\_keys();

//мои функции

void write\_string(unsigned char\* text);//выводит строку на экран

void init\_text\_clr();//инициализирует цвет текста

void clear\_screen();//очищает весь экран

void set\_cursor();//устанавливает курсор

void info();

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

void on\_backspase();

void on\_enter();

void on\_symbol(const unsigned char symbol);

void choose\_command();

short strcmp(const unsigned char\* first, const unsigned char\* second);

unsigned short strlen(const unsigned char\* str);

char strchr(const unsigned char\* str, unsigned char letter);

void print\_error();

short str\_to\_number(const unsigned char\* str, const unsigned char size);

void number\_to\_str(short n, unsigned char\* str);

short nod(short n1, short n2);

void print\_error\_arg();

void print\_error\_overflow();

void make\_fraction(const unsigned char\* num, const unsigned char\* den, unsigned char\* out);

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

}

// Функция регистрации таблицы дескрипторов прерываний

void intr\_start()

{

int idt\_count = sizeof(g\_idt) / sizeof(g\_idt[0]);//количество элементов таблицы IDT

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

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

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

}

//разрешить прерывания

void intr\_enable()

{

asm("sti");

}

//запретить прерывания

void intr\_disable()

{

asm("cli");

}

//Чтение из порта

static inline unsigned char inb(unsigned short port)

{

unsigned char data;

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

return data;

}

//запись в порты вывода

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

{

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

}

//для shutdown

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

{

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

}

// регистрирует обработчик прерывания клавиатуры и разрешает контроллеру прерываний его вызывать в случае нажатия пользователем клавиши клавиатуры

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

}

// обработчик прерываний

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");

}

// считывает поступивший от пользователя символ

void keyb\_process\_keys()

{

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

if (inb(0x64) & 0x01)

{

unsigned char scan\_code;

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);

}

}

void on\_key(unsigned char scan\_code)

{

unsigned char symbol = scan\_codes[(unsigned int)scan\_code];

if (scan\_code == BACKSPACE)

{

on\_backspase();

return;

}

if (scan\_code == ENTER)

{

on\_enter();

return;

}

on\_symbol(symbol);

}

void on\_backspase()

{

if (!column)//если самая левая позиция в строке, то ничего не делается

return;

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF + 2 \* (VIDEO\_WIDTH \* line + column);

vb[0] = 0;//удаляем символ с экрана

column--;//вигаем позицию колонки на 1 влево

set\_cursor();//устанавливаем курсор левее

}

void on\_enter()

{

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF + 2 \* (VIDEO\_WIDTH \* line + column);

vb[0] = column = 0;//устанавливаем конец строки и отматываем каретку

line++;//переносим строки

set\_cursor();//устанавливаем курсор

choose\_command();//так как пользователь выбрал команду, то обрабатываем и выполняем

}

void on\_symbol(const unsigned char symbol)

{

if (column == 40)//по условию пользователь не может ввести более 40 символов, поэтому будут игнорироваться

return;

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF + 2 \* (VIDEO\_WIDTH \* line + column);

vb[0] = symbol;

column++;

set\_cursor();

}

void choose\_command()

{

//надо обработать уже введенную строку, поэтому прыгаем на строку выше

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF + 2 \* VIDEO\_WIDTH \* (line - 1);

//41 символ - 40 введенных и 1 символ окончания строки

unsigned char command[] = "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0";

//проходим всю строку и считываем комманду, которую ввел пользователь

unsigned char j=0;

for (unsigned char i = 0; vb[i] != '\0'; i += 2)

command[j++] = vb[i];

command[j]=0;

if (\*vb == 'i')

{

write\_string(command);

unsigned char ch[]="info";

if (!strcmp(command, ch))//проверяем правильно ли написал пользователь команду

{

info();

return;

}

}

if (\*vb == 's')

{

unsigned char ch[]="shutdown";

if (!strcmp(command, ch))//проверяем правильно ли написал пользователь команду

{

outw(0x604, 0x2000);

return;

}

}

//если длина комманды корректна

if (strlen(command) >= 4)

{//если это корректная команда

//если это gcd

if (command[0] == 'g' && command[1] == 'c' && command[2] == 'd' && command[3] == ' ')

{

unsigned char\* ptr = command + 4;//ставим указатель на элемент с индексом 4 - начало первого числа

char space\_pos = strchr(ptr,(unsigned char)' ');//находим позицию второго числа

if (space\_pos == -1)

{

print\_error();//пользователь ввел только 1 аргумент

return;

}

//проверяем 2 аргумента

short num[2] = { 0,0 };

for (unsigned char i = 0; i < 2; i++)

{

if (\*ptr == '-')

{

print\_error\_arg();//пользователь ввел отрицательное число

return;

}

num[i] = str\_to\_number(ptr, (unsigned char)space\_pos);

if (num[i] == -1)

{

print\_error\_arg();//пользователь ввел некорректный аргумент

return;

}

if (num[i] == -2)

{

print\_error\_overflow();//пользователь ввел слишком длинное/большое число

return;

}

ptr += space\_pos+1;//переходим ко второму аргументу

space\_pos = (char)strlen(ptr);//находим длину второго аргумента

}

short res=nod(num[0], num[1]);

unsigned char out[6];

number\_to\_str(res, out);

if (line > 22)

clear\_screen();

write\_string(out);

return;

}

//если это solve

if (command[0] == 's' && command[1] == 'o' && command[2] == 'l' && command[3] == 'v'&&

command[4] == 'e'&& command[5] == ' ')

{

unsigned char\* ptr = command + 6;//ставим указатель на элемент с индексом 6 - начало уравнения

unsigned char negative = 0;

short num[3] = { 0,0,0 };

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

{

//ax+b=c

//работа с а/b/c

if (\*ptr == '-')

{

negative = 1;//показываем, что a/b отрицательное

ptr++;//ставим указатель на начало а/b

}

else if (\*ptr == '+')

ptr++;//ставим указатель на начало а/b

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

ptr++;

//ищем длину |ax| в первом заходе, b - во втором

char try\_end\_arg=-1, end\_arg=-1;

if (i == 0)

{

try\_end\_arg = strchr(ptr, (unsigned char)'+');

end\_arg = strchr(ptr, (unsigned char)'-');

}

if (i==1)

end\_arg = strchr(ptr, (unsigned char)'=');

if (i == 2)

end\_arg = strlen(ptr);

if ((try\_end\_arg != -1 && try\_end\_arg < end\_arg)||end\_arg==-1)

end\_arg = try\_end\_arg;

if (i == 0&&end\_arg!=-1) end\_arg--;//х рассматривать не надо

if (end\_arg == -1 || end\_arg > 5)

{

print\_error\_arg();//пользователь ввел некорректный аргумент

return;

}

if (end\_arg==0) num[i]=1;

else num[i] = str\_to\_number(ptr, (unsigned char)(end\_arg));

if (num[i] == -1)

{

print\_error\_arg();//пользователь ввел некорректный аргумент

return;

}

if (num[i] == -2)

{

print\_error\_overflow();//пользователь ввел слишком длинное/большое число

return;

}

if (negative)//если а/b/c отрицательное

num[i] \*= -1;

ptr += end\_arg;//переходим к b/c

if (i == 0) ptr++;//надо пропустить х

negative = 0;

}

//num[0] - a, num[1] - b, num[2] - c

short right = num[2] - num[1];//переносим b в правую сторону

short left = num[0];

short res;

char var = 1;

if (right % left == 0)//если ответ целочисленный

{

res = right / left;

}

else//ответ не целочисленный

{

short del;

//пока НОД не равен 1, находим его и делим на него

while ((del=nod(right, left)) != 1)

{

right /= del;

left /= del;

}

var++;//показываем, что ответ нецелочисленный

}

if (line > 22)

clear\_screen();

unsigned char out[13];

unsigned char\* ptr\_out = out;

if (var == 1)//если целочисленный ответ

{

if (res < 0)//если отрицательный результат

{

\*ptr\_out = '-';

res \*= -1;

ptr\_out++;

}

number\_to\_str(res, ptr\_out);

}

else//нецелочисленный ответ

{

unsigned char numerator[6];

unsigned char denominator[6];

if (right / left < 0)

{

\*ptr\_out = '-';

ptr\_out++;

}

if (right < 0) right \*= -1;

if (left < 0) left \*= -1;

number\_to\_str(right, numerator);

number\_to\_str(left, denominator);

make\_fraction(numerator, denominator, ptr\_out);

}

write\_string(out);

return;

}

}

print\_error();//если условия не прошли и функция не завершилась, то перед завершением выводим ошибку

}

void make\_fraction(const unsigned char\* num, const unsigned char\* den, unsigned char\* out)

{

while (\*num)

{

\*out = \*num;

num++;

out++;

}

\*out = '/';

out++;

while (\*den)

{

\*out = \*den;

den++;

out++;

}

\*out = 0;

}

//поиск наибольшего общего делителя

short nod(short n1, short n2)

{

while (n1 > 0 && n2 > 0)

if (n1 > n2)

n1 %= n2;

else

n2 %= n1;

short res = n1 + n2;

return res;

}

void number\_to\_str(short n, unsigned char\* str)

{

unsigned char size = 0;

short cpy = n;

//узнаем количество символов

while (cpy > 0)

{

size++;

cpy /= 10;

}

//переводим число в строку

str[size] = 0;

while (n > 0)

{

str[size - 1] = n % 10 + 48;

n /= 10;

size--;

}

}

//перевод строки в число

short str\_to\_number(const unsigned char\* str, unsigned char size)

{

//проверяем длину строки

if (size == 0 || size > 5) return -2;

//проверяем корректность строки

for (unsigned char i = 0; i < size; i++)

{

if (str[i] < 48 || str[i]>57)

return -1;

}

//проверяем переполнение: >32767

if (size == 5)

{

unsigned char max[]="32767";

if (strcmp(str, max) > 0)

return -2;

}

short n = 0;

for (unsigned char i=0;i<size;i++)

{

n\*=10;

n += (short)(str[i]) - 48;

}

return n;

}

short strcmp(const unsigned char\* first, const unsigned char\* second)

{

while (\*first)

{

if (!\*second) return 1;

if (\*second > \*first) return -1;

if (\*second < \*first) return 1;

second++;

first++;

}

if (\*second) return -1;

return 0;

}

unsigned short strlen(const unsigned char\* str)

{

unsigned short len = 0;

while (\*str != '\0')

{

str++;

len++;

}

return len;

}

//находит позицию пробела в строке, если символ не найден, то возвращает -1

char strchr(const unsigned char\* str, unsigned char letter)

{

char pos = 0;

while (\*str)

if (\*str == letter)

return pos;

else

{

pos++;

str++;

}

return -1;

}

//вывод ошибки на экран

void print\_error()

{

if (line > 20)

clear\_screen();

unsigned char error[] = "Incorrect command";

write\_string(error);

unsigned char commands[] = "Correct commands: info, gcd dig1 dig2, solve equation, shutdown";

write\_string(commands);

}

void print\_error\_arg()

{

if (line > 21)

clear\_screen();

unsigned char error[] = "Incorrect argument";

write\_string(error);

}

void print\_error\_overflow()

{

if (line > 21)

clear\_screen();

unsigned char error[] = "Short overflow";

write\_string(error);

}

//установка цвета, который выбрал пользователь, тексту

void init\_text\_clr()

{

//заполняем байт с атрибутами цвета, второй байт пропускаем, там будет храниться символ

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF;

for (unsigned short i = 1; i < VIDEO\_HEIGHT \* VIDEO\_WIDTH; i += 2)

vb[i] = color;

}

//очистка экрана и установка курсора в начало

void clear\_screen()

{

//заполняем каждый первый байт с кодом символа, второй с атрибутами не трогаем

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF;

for (unsigned short i = 0; i < VIDEO\_HEIGHT \* VIDEO\_WIDTH; i += 2)

vb[i] = 0;

line = column = 0;

set\_cursor();

}

//установка курсора в новое место на экране

void set\_cursor()

{

//домножение на 2, так как по 2 байта на символ

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF + 2 \* (VIDEO\_WIDTH \* line + column);

vb[0] = ' ';

unsigned short new\_pos = (line \* VIDEO\_WIDTH) + column;

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));

}

//вывод строки на экран

void write\_string(unsigned char\* text)

{

unsigned char\* vb = (unsigned char\*)ADDR\_VIDEO\_BUF + 2 \* VIDEO\_WIDTH \* line;

while (\*text)

{

vb[0] = \*text;

vb += 2;

text++;

}

column = 0;

line++;

set\_cursor();

}

//вывод информации

void info()

{

if (line > 20)

clear\_screen();

unsigned char info\_1[] = "Author: Skripko Ivan; Group: 4851003/00002";

write\_string(info\_1);

unsigned char info\_2[] = "Compiler: gcc; Assembly Language Translator: YASM";

write\_string(info\_2);

if (color == 0x0F)

{

unsigned char info\_3[] = "Text color in console: white";

write\_string(info\_3);

return;

}

if (color == 0x0E)

{

unsigned char info\_3[] = "Text color in console: yellow";

write\_string(info\_3);

return;

}

if (color == 0x03)

{

unsigned char info\_3[] = "Text color in console: cian";

write\_string(info\_3);

return;

}

if (color == 0x0B)

{

unsigned char info\_3[] = "Text color in console: magenta";

write\_string(info\_3);

return;

}

if (color == 0x02)

{

unsigned char info\_3[] = "Text color in console: green";

write\_string(info\_3);

return;

}

else

{

unsigned char info\_3[] = "Text color in console: gray";

write\_string(info\_3);

return;

}

}

extern "C" int kmain()

{

//init text color

if (\*(unsigned char\*)ADDR\_VIDEO\_BUF == 1)

color = 0x08;

if (\*(unsigned char\*)ADDR\_VIDEO\_BUF == 2)

color = 0x0F;

if (\*(unsigned char\*)ADDR\_VIDEO\_BUF == 3)

color = 0x0E;

if (\*(unsigned char\*)ADDR\_VIDEO\_BUF == 4)

color = 0x03;

if (\*(unsigned char\*)ADDR\_VIDEO\_BUF == 5)

color = 0x0B;

if (\*(unsigned char\*)ADDR\_VIDEO\_BUF == 6)

color = 0x02;

clear\_screen();

init\_text\_clr();

unsigned char welcome[] = "Welcome to SolverOS!";

write\_string(welcome);

intr\_init();

keyb\_init();

intr\_start();

intr\_enable();

while (1)

{

asm("hlt");

}

return 0;

}