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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**«ПРИНЦИПЫ РАЗРАБОТКИ ОПЕРАЦИОННЫХ СИСТЕМ»**

по дисциплине «Операционные системы»

1. Выполнил
2. студент гр. 4851003/10002 Галкин К. К.

1. Руководитель
2. К. н. т Крундышев В. М.

Санкт-Петербург

2023

1. **ЦЕЛЬ РАБОТЫ**

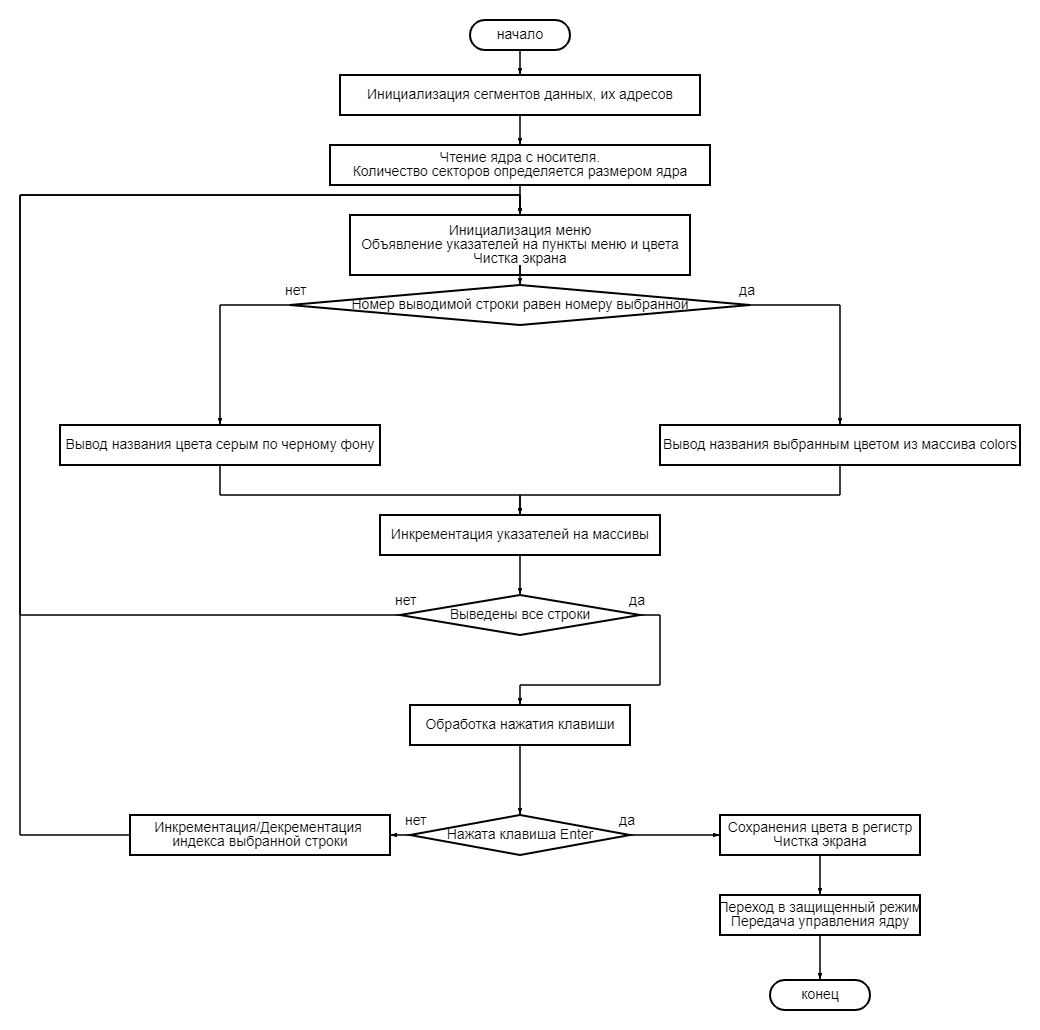
Цель работы — изучение основ разработки ОС, принципов низкоуровневого взаимодействия с аппаратным обеспечением, программирования системной функциональности и процесса загрузки системы.

1. **ХОД РАБОТЫ**

Перед выполнением работы были поставлены следующие задачи:

* Написать загрузчик ОС в соответствии с вариантом
* Передавать через загрузчик управление ядру ОС
* Написать ядро ОС в соответствии с вариантом (SolverOS)
* Протестировать разработанное решение

Блок – схема и подробное описание принципа работы загрузчика представлено ниже:

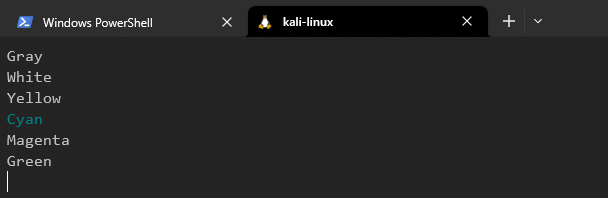


**Рисунок 1 Блок-схема работы загрузчика**

Несколько подробностей:

* Чистка экрана происходит благодаря прерыванию int 0x10 и ax = 3
* Указатели на пункты меню и массив цветов – регистры si, di кладутся адреса меток menu\_items (названия всех цветов) и colors (значения байта цвета текста и фона) соответственно.
* Вывод строк происходит посимвольно через функцию ah = 0x09 и прерывания int 0x10. До этого, в прошлых реализациях, была попытка использовать функцию ah = 0x0E с тем же прерыванием, но происходила проблема с использованием цвета. Он просто не показывался, даже в графическом режиме.
* При посимвольном выводе используется чтение позиции курсора через ah = 0x02 и его изменение через функцию ah = 0x03 и int 0x10. При этом, для сохранения данных, использовались команды pusha и popa, кладущие и забирающие данных регистров со стека
* Нажатия стрелки “вверх” и “вниз”, а так же клавиши Enter, происходит через значения регистра ah после прерывания 0x16. Важно отметить, что в регистр ah в результате прерывания кладется скан-код нажатой клавиши, а в al – само значение. Поэтому можно обрабатывать нажатия через любой из регистров, только значения будут другие.
* Переход в защищенный режим происходит по нажатию клавиши Enter. В этом случае происходит прыжок на метку load\_kernel, где запоминается значение цвета, чистится экран, загружается адрес и размер gdt – таблицы в GDTR регистр процессора. В таблице содержатся данные о двух сегментах: первый – сегмент кода размером 4 Гб, который начинается с адреса 0, флаг P = 1 показывает, что сегмент действителен с уровнем привилегий DPL = 0 и пользовательским режимом (S = 0), данный сегмент настроен на чтение и выполнение в 32-битном режиме. Второй сегмент – сегмент данных, инициализация аналогична, правда данных сегмент доступен только на запись.
* После включения GDT-сегмента происходит включение адресной линии А20 через порт 92h, где нужно контролировать только первый разряд, остальные изменять нельзя, и включение CR0 регистра, а конкретнее флага PE (1 бит регистра CR0) в значение 1. Затем идет длинный прыжок, где уже идет передача управления ядру и объявления текущего сегмента в памяти загрузочным.

Пример работы загрузчика можно наблюдать ниже:



**Рисунок 2 Выбор цвета при работе загрузчика**

Теперь, при переходе управления ядру, после инициализации прерываний и обработчика нажатия клавиш, значения цвета из регистра bl кладется в переменную color. По условию задания, SolverOS должна обладать возможностью решать линейные уравнения вида ax+b=c и находить НОД двух положительных чисел, не превосходящих максимальное значение типа int (было взято значение unsigned int, поскольку функция работает с положительными значениями). Рассмотрим таблицу, в которой находится описание основных функций ядра ОС, так же на ней будут выделены вспомогательные функции (серым цветом) для работы со строками и обработкой клавиш:

|  |  |
| --- | --- |
| Название функции | Описание |
| void on\_key(unsigned char scan\_code) | Обработка scan\_code нажатой клавиши. Проверка на то, что нажатая клавиша является клавишей shift и установка соответствующего флага. Вывод символа, который был нажат на экран. |
| void backspace\_handler() | Обработка нажатия клавиши Backspace. Двигает положение курсора, очищая данные за курсором, до того момента, пока не дойдет до символа ввода команды - # |
| void read\_command() | Чтение команды в строку input\_str из видеопамяти. |
| void print\_error(const char \*error) | Вывод ошибки error и нового символа ввода команды |
| void command\_parse() | Парсинг input\_str на команды ОС. Если команда принимает после себя аргументы, то функция записывает их и обрабатывает через вспомогательные функции. |
| strlen, strcmp, strtok, strrev, swap, \_atoi, \_itoa | Функции работы со строками из основных библиотек языка Си. |
| void clear() | Чистка экрана. Изначально была попытка отчистить экран через прерывание 0x10 и функцию ax =3, но возникли проблемы с отчищаемой областью. Поэтому было принято решение чистить посимвольно, выставляя значения байтов на экране в 0x00. |
| int gcd(int n1, int n2, char \*str) | Нахождение НОД чисел n1 и n2 через расширенный алгоритм Евклида. Для дальнейшей работы предусмотрен возврат результата, как целочисленного значения. Строка с результатом формируется прямо в этой функции и кладется в str. Проверка данных идет до вызова, в функции command\_parse. |
| void solve(int a, int b, int c) | Функция решения линейного уравнения вида ax+b=c. Число b переносится вправо с изменением знака. Сначала идет проверка на то, что right\_digit = b + c нацело делится на a. Если да, тогда результат – их целочисленное деление. Если нет, тогда происходит деление каждого из чисел (a, right\_digit) на их НОД (функция gcd), пока результат gcd не станет 1. После идет формирование результирующей строки. Проверка уравнения так же идет в функции command\_parse. |
| void shutdown() | Отправка команды для отключения питания на ACPI порт 0x604. Важно отметить, что такая реализация команды работает только на новых версиях эмулятора QEMU, для более старых версии нужно использовать другие порты и команды. Если ос запущена под VmWare, тогда функцию надо полностью менять |
| void info() | Вывод информации об авторе ОС, используемом компиляторе, синтаксисе и трансляторе. |

Тексты программ загрузчика и ядра указаны в приложении

1. **РЕЗУЛЬТАТЫ ТЕСТОВ**

Изображение выглядит как текст

Автоматически созданное описание

**Рисунок 3 Использование функции info**

Изображение выглядит как текст

Автоматически созданное описание

**Рисунок 4 Использование функции shutdown.**

Изображение выглядит как текст

Автоматически созданное описание

**Рисунок 5 Проверка функции gcd и ее аргументов**

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**Рисунок 6 Проверка функции solve**

1. **ВЫВОД**

В ходе выполнения работы были изучены принцип работы загрузчика в памяти, как читаются данные с носителей и как происходит переход в защищенный режим процессора, как происходит передача управления ядру. Были изучены основные прерывания, позволяющие взаимодействовать с компонентами машины (экран, клавиатура и др.), и их аргументы. В ходе разработки ядра поверхностно была изучена теория по ACPI и SMM режимам, которые позволяют управлять состоянием компьютера, посылая на порты различные команды. Разработанное решение было протестировано на не стандартный ввод, проведено 22 теста.

1. **ПРИЛОЖЕНИЕ**

Исходный код загрузчика:

[BITS 16]

[ORG 0x7C00]

\_start:

; Инициализация сегментов данных и стека

mov ax, cs

mov ds, ax

mov ss, ax

mov sp, \_start ; Начало стека - адрес первой команды

;чтение ядра

mov ax, 0x1000

mov es, ax ;размер буфера

mov bx, 0x00 ;адрес буффера(0x0000: 0x1000)

mov ah, 0x02 ;режим чтения

mov dl, 1 ;номер диска

mov dh, 0x00 ;номер головки

mov cl, 0x01 ;номер сектора

mov ch, 0x00 ;номер цилиндра

mov al, 35 ;кол-во секторов. Считается как размер ядра / 512 с округлением в большую сторону

int 0x13

xor ch, ch ; Номер выбранного пункта меню (начинаем с первого)

call menu ; Прыгаем на меню

menu:

; Чистка всего экрана

mov ax, 3

int 0x10

xor dh, dh ; Счетчик цикла

mov di, colors ; Указатель на массив цветов

mov si, menu\_items ; Указатель на указатели строк

mov ah, 0x09 ; Вывод символов

mov bl, 0x07 ; Цвет символов (белый на черном фоне)

xor ah, ah

menu\_loop:

cmp ch, dh ; Сравниваем

je print\_selected ; Если сейчас будет печататься выбранный пункт меню, то прыгаем

jmp print\_char ; Иначе просто продолжаем печатать символы с обычным цветом

find\_color:

inc ah ; Инкрементируем индекс

inc di ; Инкрементируем указатель

jmp print\_selected

print\_selected:

cmp ah, ch ; Ищем индекс цвета в массиве colors в соответствии с выбранным пунктом

jne find\_color

mov bl, byte [di] ; Ставим на bl байт цвета текста

mov cl, bl

jmp print\_char

print\_char:

mov al, byte [si] ; Выбираем символ из массива

pusha ; Сохраняем на стек значения регистров

mov cx, 1 ; Печатаем символ один раз

mov ah, 0x09 ; Принт символа с цветом

int 0x10

popa ; Возврат значения регистров

; Блок перемещения курсора для корректной работы функции 0x09

pusha

mov bh, 0

mov ah, 0x03

int 0x10

inc dl

mov ah, 0x02

int 0x10

popa

inc si ; Бегаем по массиву строк

cmp byte [si], 0x00 ; Пока это не конец строки - печатаем ее

jne print\_char

inc si ; Убираем указатель с 0x00 байта

; Блок перемещения курсора для корректной работы функции 0x09

pusha

mov bh, 0

mov ah, 0x03

int 0x10

inc dh

mov dl, 0

mov ah, 0x02

int 0x10

popa

cmp dh, 5 ; Если напечатали последнюю строку - ждем взаимодействия пользователя

je wait\_for\_input ; Взаимодействие пользователя

inc dh ; Print string

mov bl, 0x07 ; Восстанавливаем цвет на всякий случай

jmp menu\_loop

wait\_for\_input:

; Ожидание ввода пользователя и обработка нажатий клавиш "вверх" и "вниз"

mov ah, 0x00 ; Функция чтения клавиши BIOS

int 0x16 ; Вызов прерывания BIOS для чтения клавиши

cmp ah, 0x48 ; Проверка на нажатие клавиши "вверх"

je move\_up ; Если это клавиша "вверх", то переходим к предыдущему пункту меню

cmp ah, 0x50 ; Проверка на нажатие клавиши "вниз"

je move\_down ; Если это клавиша "вниз", то переходим к следующему пункту меню

cmp ah, 0x1C

je load\_kernel

jmp wait\_for\_input ; Если это другая клавиша, то ожидаем новый ввод

move\_up:

cmp ch, 0 ; Проверка на начало списка пунктов меню

je wait\_for\_input ; Если это начало списка, то ожидаем новый ввод

dec ch ; Переходим к предыдущему пункту меню

jmp menu ; Выводим меню заново

move\_down:

cmp ch, 5 ; Проверка на конец списка пунктов меню

je wait\_for\_input ; Если это конец списка, то ожидаем новый ввод

inc ch ; Переходим к следующему пункту меню

jmp menu ; Выводим меню заново

colors:

db 0x07, 0x0F, 0x0E, 0x03, 0x05, 0x02

menu\_items:

db "Gray", 0

db "White", 0

db "Yellow", 0

db "Cyan", 0

db "Magenta", 0

db "Green", 0

load\_kernel:

xor bh, bh

mov bl, cl ; цвет текста

mov ax, 3

int 0x10

cli

lgdt [gdt\_info]

in al, 0x92

or al, 2

out 0x92, al

mov eax, cr0

or al, 1

mov cr0, eax

jmp 0x8:protected\_mode

gdt:

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

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

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

gdt\_info:

dw gdt\_info - gdt

dw gdt, 0

; Метка для перехода в защищенный режим

[BITS 32]

protected\_mode:

mov ax, 0x10

mov es, ax

mov ds, ax

mov ss, ax

call 0x10000

times (512 -($ - \_start)- 2) db 0

db 0x55, 0xAA

Исходный код 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)

#define NULL 0

/\*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 ACPI\_PORT 0x604

#define ACPI\_CMD 0x2000

#define UINT\_MAX 0xffffffff

#define HEIGHT 25

#define CURSOR\_COLOR 0x02

// 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

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

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

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

// https://wiki.osdev.org/PS/2\_Keyboard#Scan\_Code\_Set\_1

unsigned char no\_shift\_codes[58] = {

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

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

'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ';', '\'', '`', 0,

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

unsigned char shift\_on\_codes[58] = {

0, 0, '!', '@', '#', '$', '%', '^', '&', '\*', '(', ')', '\_', '+', 0, 0,

'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', '{', '}', 0, 0,

'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', ':', '\"', '~', 0,

'|', 'Z', 'X', 'C', 'V', 'B', 'N', 'M', '<', '>', '?', 0, 0, 0, ' '};

const char \*welcome = "Wellcome to SolverOS"; // Welcome message

const char \*default\_err = "Error: Command not found!"; // Default Command message

const char \*gcd\_incorrect = "Error: Command Incorrect"; // GCD function error if integers less than 0

const char \*gcd\_overflow = "Error: Integer overflow"; // GCD function error if integer above than UINT\_MAX

const char \*equation\_incorrect = "Error: Equation is incorrect";

int color; // Color value form bootsector

unsigned char shift = 0; // Is shift pressed?

unsigned int current\_string = 0; //

unsigned int cursor\_position = 0;

unsigned int string\_len = 0;

char input\_str[41];

// Interrupt functions

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 default\_intr\_handler();

// Keyboard functions

void keyb\_handler();

void keyb\_init();

static inline unsigned char inb(unsigned short port);

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

void keyb\_process\_keys();

void backspace\_handler();

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

// Cursor position function

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

// Command parsing functions

void command\_parse();

void read\_command();

// Convert functions

char \*\_itoa(int num, char \*str, int base);

unsigned long long int \_atoi(char \*str);

// String and print functions

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

void out\_chr(int color, unsigned char ptr);

void strrev(char str[], int length);

int strcmp(const char \*a, const char \*b);

int strlen(const char \*a);

char \*strtok(char \*s, const char \*delim);

char \*strtok\_r(char \*s, const char \*delim, char \*\*last);

void swap(char &a, char &b);

// OS functinos

void solve(int a, int b, int c);

int gcd(int n1, int n2, char \*str);

void shutdown();

void info();

void help();

void clear();

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");

}

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 default\_intr\_handler()

{

asm("pusha");

// ... (реализация обработки)

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

}

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));

}

void swap(char &a, char &b)

{

char temp = a;

a = b;

b = temp;

}

int strlen(const char \*str)

{

const char \*s;

for (s = str; \*s; ++s)

;

return (s - str);

}

int strcmp(const char \*s1, const char \*s2)

{

for (; \*s1 == \*s2; s1++, s2++)

if (\*s1 == '\0')

return 0;

return ((\*(unsigned char \*)s1 < \*(unsigned char \*)s2) ? -1 : +1);

}

char \*strtok(char \*s, const char \*delim)

{

static char \*last;

return strtok\_r(s, delim, &last);

}

char \*strtok\_r(char \*s, const char \*delim, char \*\*last)

{

char \*spanp;

int c, sc;

char \*tok;

if (s == NULL && (s = \*last) == NULL)

return (NULL);

cont:

c = \*s++;

for (spanp = (char \*)delim; (sc = \*spanp++) != 0;)

{

if (c == sc)

goto cont;

}

if (c == 0)

{ /\* no non-delimiter characters \*/

\*last = NULL;

return (NULL);

}

tok = s - 1;

for (;;)

{

c = \*s++;

spanp = (char \*)delim;

do

{

if ((sc = \*spanp++) == c)

{

if (c == 0)

s = NULL;

else

s[-1] = 0;

\*last = s;

return (tok);

}

} while (sc != 0);

}

}

void strrev(char str[], int length)

{

int start = 0;

int end = length - 1;

while (start < end)

{

swap(\*(str + start), \*(str + end));

start++;

end--;

}

}

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

{

if (pos < 1 || pos > string\_len || pos > 79)

return;

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));

cursor\_position = pos;

}

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

{

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

video\_buf += 80 \* 2 \* current\_string;

while (\*ptr)

{

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

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

video\_buf += 2;

ptr++;

}

if (current\_string++ >= 25)

{ // Clear if we fill the window

clear();

out\_chr(color, (unsigned char)'#');

}

string\_len = 0;

cursor\_moveto(current\_string, 0);

}

void out\_chr(int color, unsigned char char\_code)

{

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

video\_buf += 80 \* 2 \* current\_string + cursor\_position \* 2;

video\_buf[0] = char\_code;

video\_buf[1] = color;

if (string\_len <= cursor\_position)

string\_len++;

cursor\_moveto(current\_string, cursor\_position + 1);

}

void on\_key(unsigned char scan\_code)

{

if (scan\_code <= 58)

{

switch (scan\_code)

{

case 42:

shift = 1;

break;

case 54:

shift = 1;

break;

default:

if (shift)

{

if (shift\_on\_codes[scan\_code])

out\_chr(color, shift\_on\_codes[scan\_code]);

}

else

{

if (no\_shift\_codes[scan\_code])

out\_chr(color, no\_shift\_codes[scan\_code]);

}

break;

}

}

}

/\* Обработка нажатия Backspace. Отчистка символов, которые стоят после курсора. \*/

void backspace\_handler()

{

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

video\_buf += 80 \* 2 \* current\_string + 2 \* cursor\_position;

if (cursor\_position == 1)

return;

video\_buf = video\_buf - 2;

video\_buf[0] = 0;

cursor\_moveto(current\_string, cursor\_position - 1);

string\_len--;

return;

}

/\* Чтение команды из видеопамяти в переменную. \*/

void read\_command()

{

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

video\_buf += 80 \* 2 \* current\_string + 2;

int tmp = 0;

while (tmp < string\_len)

{

input\_str[tmp] = video\_buf[0];

video\_buf += 2;

tmp++;

}

input\_str[tmp] = '\0';

}

/\* Вывод нужной ошибки \*/

void print\_error(const char \*error)

{

out\_str(color, error);

out\_chr(color, (unsigned char)'#');

}

/\* Парсинг введенной команды и ее аргументов, если таковые есть \*/

void command\_parse()

{

char delim[] = " ";

char \*command\_part = strtok(input\_str, delim);

if (strcmp(command\_part, "info") == 0)

{

info();

}

else if (strcmp(command\_part, "help") == 0)

{

help();

}

else if (strcmp(command\_part, "clear") == 0)

{

clear();

out\_chr(color, (unsigned char)'#');

}

else if (strcmp(command\_part, "shutdown") == 0)

{

shutdown();

}

else if (strcmp(command\_part, "gcd") == 0)

{

char operation\_number = 0;

int n1, n2;

int check\_status = 0;

while (command\_part != NULL)

{

switch (operation\_number)

{

case 0:

if (strcmp(command\_part, "gcd") != 0)

print\_error(gcd\_incorrect);

else

check\_status++;

break;

case 1:

n1 = \_atoi(command\_part);

if (n1 <= UINT\_MAX && strlen(command\_part) >= 10)

{

print\_error(gcd\_overflow);

return;

}

else if (n1 <= 0)

{

print\_error(gcd\_incorrect);

return;

}

else

check\_status++;

break;

case 2:

n2 = \_atoi(command\_part);

if (n2 == 0 && strlen(command\_part) >= 10)

{

print\_error(gcd\_overflow);

return;

}

else if (n2 <= 0)

{

print\_error(gcd\_incorrect);

return;

}

else

check\_status++;

break;

default:

break;

}

operation\_number++;

command\_part = strtok(NULL, delim);

}

if (check\_status == 3)

{

char result\_str[110] = "Result: ";

gcd(n1, n2, result\_str);

out\_str(color, result\_str);

out\_chr(color, (unsigned char)'#');

}

}

else if (strcmp(command\_part, "solve") == 0)

{

command\_part = strtok(NULL, delim);

int a, b, c;

char parameter[50];

char operation\_number = 0;

int i = 0, param\_index = 0;

if (strlen(command\_part) < 5)

{

print\_error(equation\_incorrect);

return;

}

if (command\_part[0] == 'x')

{

a = 1;

i++;

}

else

{

while (command\_part[i] != 'x')

{

parameter[param\_index] = command\_part[i];

param\_index++;

i++;

}

parameter[param\_index] = '\0';

if (param\_index == 1 && parameter[0] == '-')

a = -1;

else

a = \_atoi(parameter);

}

i++;

param\_index = 0;

while (command\_part[i] != '=')

{

parameter[param\_index] = command\_part[i];

param\_index++;

i++;

}

parameter[param\_index] = '\0';

b = \_atoi(parameter);

param\_index = 0;

i++;

while (i != strlen(command\_part))

{

parameter[param\_index] = command\_part[i];

param\_index++;

i++;

}

parameter[param\_index] = '\0';

c = \_atoi(parameter);

param\_index = 0;

if (a != 0)

solve(a, b, c);

else

print\_error(equation\_incorrect);

a = 0;

b = 0;

c = 0;

}

else

{

out\_str(color, default\_err);

out\_chr(color, (unsigned char)'#');

}

}

/\* Обработка всех нажатий, проверка на нажатие клавиши enter или backspace \*/

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)

{

// Обработка backspace

if (scan\_code == 14)

{

backspace\_handler();

}

if (scan\_code == 28)

{

if (string\_len == 1)

return;

read\_command();

current\_string++;

cursor\_position = 0;

string\_len = 0;

command\_parse();

}

on\_key(scan\_code);

}

else // Скан-коды выше 128 - это отпускание клавиши

{

if (scan\_code == 170 || scan\_code == 182) // Shift key up

shift = 0;

}

}

}

/\* Перевод числа в строку, учитывая знак числа \*/

char \*\_itoa(int num, char \*str, int base)

{

int i = 0;

int is\_negative = 0;

if (num == 0)

{

str[i++] = '0';

str[i] = '\0';

return str;

}

if (num < 0 && base == 10)

{

is\_negative = 1;

num = -num;

}

while (num != 0)

{

int rem = num % base;

str[i++] = (rem > 9) ? (rem - 10) + 'a' : rem + '0';

num = num / base;

}

if (is\_negative)

str[i++] = '-';

str[i] = '\0';

strrev(str, i);

return str;

}

/\* Перевод строки в число с учетом знака \*/

unsigned long long int \_atoi(char \*str)

{

long long int res = 0;

long long int sign = 1;

int i = 0;

if (str[0] == '+')

{

sign = 1;

i++;

}

else if (str[0] == '-')

{

sign = -1;

i++;

}

for (; str[i] != '\0'; ++i)

res = res \* 10 + str[i] - '0';

return sign \* res;

}

/\* Нахождение НОД двух положительных чисел \*/

int gcd(int n1, int n2, char \*str)

{

static int result;

int temp;

while (n2 != 0)

{

temp = n1 % n2;

n1 = n2;

n2 = temp;

}

result = n1;

char solution[100];

\_itoa(n1, solution, 10);

int num\_index = 0;

int res\_index = strlen(str);

while (num\_index != strlen(solution))

{

str[res\_index] = solution[num\_index];

num\_index++;

res\_index++;

}

// result\_str[res\_index] = '\0';

return result;

}

/\* Решение уравнения вида ax+b=c \*/

void solve(int a, int b, int c)

{

char result\_str[110] = "Result: ";

char solution[100] = "";

char a\_str[50] = "";

char right\_str[50] = "";

int delim = 0;

int sign = -1;

int right\_digit = c + (b \* sign);

int correct\_result = 0;

int append\_str\_index = 0;

int result\_str\_index = strlen(result\_str);

if (right\_digit % a == 0)

{

correct\_result = right\_digit / a;

\_itoa(correct\_result, right\_str, 10);

while (append\_str\_index != strlen(right\_str))

{

result\_str[result\_str\_index] = right\_str[append\_str\_index];

append\_str\_index++;

result\_str\_index++;

}

}

else

{

delim = gcd(a, right\_digit, NULL);

while (delim != 1)

{

a /= delim;

right\_digit /= delim;

delim = gcd(a, right\_digit, NULL);

}

\_itoa(a, a\_str, 10);

\_itoa(right\_digit, right\_str, 10);

while (append\_str\_index != strlen(right\_str))

{

result\_str[result\_str\_index] = right\_str[append\_str\_index];

append\_str\_index++;

result\_str\_index++;

}

append\_str\_index = 0;

result\_str[result\_str\_index] = '/';

result\_str\_index++;

while (append\_str\_index != strlen(a\_str))

{

result\_str[result\_str\_index] = a\_str[append\_str\_index];

append\_str\_index++;

result\_str\_index++;

}

}

result\_str[result\_str\_index] = '\0';

out\_str(color, result\_str);

out\_chr(color, (unsigned char)'#');

}

/\* Выключение компьютера на эмуляции QEMU последней версии \*/

void shutdown()

{

asm volatile("outw %0, %1"

:

: "a"((unsigned short)ACPI\_CMD), "d"((unsigned short)ACPI\_PORT)); // отправляем команду ACPI на порт

}

/\* Вывод общей информации \*/

void info()

{

out\_str(color, "SolverOS. Author: Galkin Klim, 4851003/10002");

out\_str(color, "OS Linux, Translator YASM, Intel syntax, C compiler - GCC");

out\_chr(color, (unsigned char)'#');

}

/\* Вывод описания всех команд \*/

void help()

{

out\_str(color, "help - information about OS commands");

out\_str(color, "info - information about creator and used software");

out\_str(color, "solve equation - solve the equation ax+b=c");

out\_str(color, "gcd num1 num2- find the gcd of two nums");

out\_str(color, "shutdown - halt the OS");

out\_str(color, "clear - clean all strings");

out\_chr(color, (unsigned char)'#');

return;

}

/\* Чистка экрана \*/

void clear()

{

/\*asm volatile(

"mov $0x03, %ah\n" // загрузка значения 3 в регистр ah

"int $0x10\n" // вызов прерывания 0x10

);\*/

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

for (int i = 0; i < VIDEO\_WIDTH \* HEIGHT; i++)

{

\*(video\_buf + i \* 2) = '\0';

}

current\_string = 0;

cursor\_position = 0;

string\_len = 0;

return;

}

extern "C" int kmain()

{

intr\_init();

keyb\_init();

intr\_start();

intr\_enable();

asm("\t movl %%ebx, %0"

: "=r"(color));

// Вывод строки

clear();

out\_str(color, welcome);

help();

// Бесконечный цикл

while (1)

{

asm("hlt");

}

return 0;

}

Скрипт сборки загрузчика и ядра

cat configure.sh

yasm -f bin bootsec.asm -o bootsec.bin

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

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