Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

по дисциплине «Операционные системы»

Выполнил
студент гр. 23508/4 Е.Г. Проценко

Проверил
профессор Е.Ю. Резединова

Санкт-Петербург
2016

1. Формулировка задания

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

2. Ход работы

2.1. Получение варианта

От преподавателя был получен вариант 18.

По заданию нужно написать InfoOS, компиляция в Windows, используя YASM(Intel), ms c compiler.

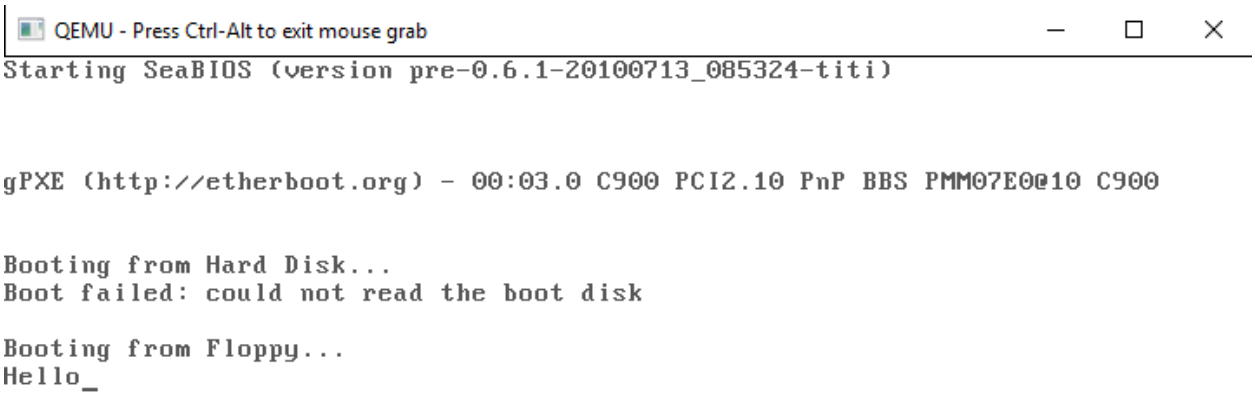
2.2. Компиляция и загрузка на эмуляторе пример загрузочного сектора

```
C:\Users\japro_000>cd C:\Users\japro_000\Desktop\Universe\4sem\OS\lab1

C:\Users\japro_000\Desktop\Universe\4sem\OS\lab1>yasm.exe start.asm

C:\Users\japro_000\Desktop\Universe\4sem\OS\lab1>cd qemu-0.13.0-windows

C:\Users\japro_000\Desktop\Universe\4sem\OS\lab1\qemu-0.13.0-windows>qemu.exe -f
da ..\start
```



2.3. Разработка загрузчика для загрузки минимального ядра и самого ядра

2.3.1. Загрузчик

Разработка загрузчика начинается с инициализации адресов сегментов: Сохранение адреса сегмента кода, сохранение этого адреса как начало сегмента данных и сегмента стека, сохранение адреса стека как адрес первой инструкции этого кода. Эти операции требуется не для любого BIOS, но их рекомендуется проводить.

Далее происходит очистка экрана и вывод надписи загрузки. Поскольку дальнейшие действия загрузчика и переход на ядро происходят очень быстро, то мы не успеваем этого заметить.

Далее, в память загружается сегмент кода и сегмент данных.

После, отключаются прерывания, обрабатывается таблица дескрипторов, переход в защищенный режим.

Теперь мы передаем управление коду, загруженному ранее.

Оставшиеся биты, кроме последних двух, обнуляются. Те два используются для записи для битов 0x55, 0xAA, чтобы сделать данный сектор, именно, загрузочным.

2.3.2. Ядро

Управление сразу передается в функцию kmain(). Здесь мы отключаем прерывания, инициализируем таблицу обработчиков прерываний (все обработчики будут пустыми). Инициализация прерывания 0x09, которое отвечает за работу с клавиатурой, на основе это прерывания мы будем получать код нажатой клавиши и будет описывать соответствующую реакцию ос на это.

После инициализации обработчиков прерываний, можно их запустить, путем включения прерываний.

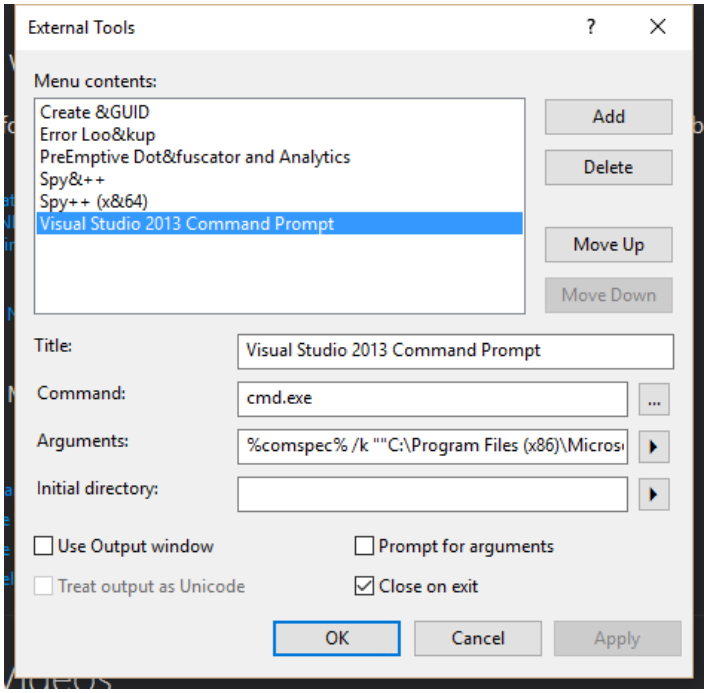
Теперь программа работает в бесконечном цикле с использованием инструкции hlt, для потребления меньшего количества ресурсов.

2.3.3. Дальнейшая работа

Теперь, после того, как обработчик прерываний настроен и сами прерывания включены, при нажатии клавиши клавиатуры, мы попадает в соответствующий обработчик. Здесь мы обрабатывает последовательность набранных ранее символов и вызываем соответствующую реакцию ядра.

3. Некоторые сложности при работе

3.1. Компиляция в консоли при помощи cl.exe просто так не проходила, т.к. говорилось о том, что *“cl.exe не является внутренней командой …”* Для решения этой проблемы пришлось покопаться в настройках Visual Studio: *Tools->External Tools.*



Здесь был создан выделенный на картинке элементом.
Arguments: `%comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\vcvars32.bat"" x86`
В другом случае может быть другой путь или другой bat файл, не vcvars32.bat
Теперь открывая командную строку через *Tools->Visual Studio 2013 Command Prompt* все будет работать.

3.2. Очень сложно найти рабочий QEMU. Вроде загрузчик работает и передает управление ядру и даже считывает клавиши, но, например, может не работать функция shutdown(). Поэтому приходилось искать нормальный QEMU и подбирать нужные параметры…

3.3. Как передать информацию с загрузчика на ядро.
На самом деле есть такой(ие) регистры, например, edi, которые указывают на дополнительную память, которая обычно не используется, так что можно спокойно записать туда загрузчиком и потом считать ядром.

```
;Загрузчик
mov bx , 8
mov [edi + 0xc], bx

//Ядро
short int tmp;
__asm
{
	mov bx, [edi + 0xc]
	mov tmp, bx
}
```

4. Приложение

4.1. Загрузчик – bootsect.asm

```
[BITS 16]
[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.
                          ;mov ah, 0x0e ; В ah номер функции BIOS: 0x0e - вывод символа на активную видео страницу (эмуляция телетайпа)

call video_mode

mov bx, loading_str ; Для GNU assembler: mov bx, offset loading_str
call puts


load_kernel:
mov ax,0x1100 ; virtual adress
mov es,ax
mov bx,0x00
mov ah,0x02           ; function
mov dl,1              ; 'nomer diska'
mov dh,0              ; 'nomer golovki'
mov cl,3              ; (file pointer to raw data) / 200 + 1
mov ch,0              ; 'nomer dorojki'
mov al,6              ; (size of raw) / 200h
int 0x13

load_data:
mov ax,0x1200
mov es,ax
mov bx,0x00
mov ah,0x02
mov dl,1              ; 'nomer diska'
mov dh,0              ; 'nomer golovki'
mov cl,9              ; (file pointer to raw data) / 200 + 1
mov ch,0              ; 'nomer dorojki'
mov al,1              ; (size of raw) / 200h
int 0x13

;-------------------------------------------------------------
;-------------------------------------------------------------
; Отключение прерываний
cli

; Загрузка размера и адреса таблицы дескрипторов
lgdt [gdt_info] ; Для GNU assembler должно быть "lgdt gdt_info"

; Включение адресной линии A20
in al, 0x92
or al, 2
out 0x92, al

; Display cleaning
call video_mode

; 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 таблицы.
;-------------------------------------------------------------
;-------------------------------------------------------------

;-------------------------------------------------------------
;-------------------------------------------------------------
puts:
        mov al, [bx]  ; movb al, [bx]
        test al, al
        jz end_puts
        mov ah, 0x0e  ; movb ah, 0x0e
        int 0x10
        inc bx                ; add bx, 1           ; addw bx, 1
        jmp puts
end_puts:
ret

video_mode:
        mov ah,0x00
        mov al,0x03
        int 0x10
ret

;-------------------------------------------------------------
;-------------------------------------------------------------

;-------------------------------------------------------------
;-------------------------------------------------------------
loading_str:
db "I want chocolate cake!", 0
;-------------------------------------------------------------
;-------------------------------------------------------------

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 0x11000 ; Адрес равен адресу загрузки в случае если ядро скомпилировано в "плоский" код

; Внимание! Сектор будет считаться загрузочным, если содержит в конце своих 512 байтов два следующих байта: 0x55 и 0xAA
times (512 - ($ - start) - 2) db 0  ; Заполнение нулями до границы 512 - 2 текущей точки
db 0x55, 0xaa  ; 2 необходимых байта чтобы сектор считался загрузочным
```

4.2. Ядро – kernel.c

```
// 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.
int kmain();
void out_str(int, const char*, unsigned int);

__declspec(naked) void startup()
{
	__asm
	{
		call 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)

// Структура описывает данные об обработчике прерывания
#pragma pack(push, 1) // Выравнивание членов структуры запрещено
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
};
// Структура, адрес которой передается как аргумент команды lidt
struct idt_ptr
{
	unsigned short limit;
	unsigned int base;
};
#pragma pack(pop)

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
__declspec(naked) void default_intr_handler()
{
	__asm
		{
			pusha
		}
	// ... (реализация обработки)
	__asm
	{
		popa
		iretd
	}
}

void intr_reg_handler(int num, unsigned short segm_sel, unsigned short flags, void* 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 g_idtp
	}
	//__lidt(&g_idtp);
}

void intr_enable()
{
	__asm sti;
}

void intr_disable()
{
	__asm cli;
}

//-------------------------------------
//-------------------------------------
//-------------------------------------

char scancodes[] = {
	0,
	0, // ESC
	'1','2','3','4','5','6','7','8','9','0', '-', '=',
	8, // BACKSPACE
	'\t', // TAB
	'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']',
	' ', // ENTER
	0, // CTRL
	'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', '<','>','+',
	0, // LEFT SHIFT
	'\\', 'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.', '/',
	0, // RIGHT SHIFT
	'*', // NUMPAD
	0, // ALT
	'_', // SPACE
	0, // CAPSLOCK
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // F1 - F10
	0, // NUMLOCK
	0, // SCROLLLOCK
	0, // HOME
	0,
	0, // PAGE UP
	'-', // NUMPAD
	0, 0,
	0,
	'+', // NUMPAD
	0, // END
	0,
```

```
	0, // PAGE DOWN
	0, // INS
	0, // DEL
	0, // SYS RQ
	0,
	0, 0, // F11 - F12
	0,
	0, 0, 0, // F13 - F15
	0, 0, 0, 0, 0, 0, 0, 0, 0, // F16 - F24
	0, 0, 0, 0, 0, 0, 0, 0
};
int line = 0;
int column = 0;
unsigned char empty_symbol;
unsigned char empty_color;
const char* sysmsg_entry = "# ";
const char* sysmsg_unknown = " Command not recognized";
const char* sysmsg_info1 = " InfoOS: v.01. Developer: Protsenko Evgeniy, 23508/4, SPbPU,
2016";
const char* sysmsg_info2 = " Compilers: bootloader: yasm, kernel: ms c compiler";
const char* sysmsg_help1 = " info";
const char* sysmsg_help2 = " help";
const char* sysmsg_help3 = " ticks";
const char* sysmsg_help4 = " mem";
const char* sysmsg_help5 = " disk";
const char* sysmsg_help6 = " time";
const char* sysmsg_help7 = " shutdown";
const char* sysmsg_emptystr = "";
#define ENTER 28
#define BACKSPACE 14
#define DEADCOLUMN 22
#define DEADLINE 5

struct time_info
{
	char year1;	// столетие
	char year2;	// год
	char month;
	char day;

	char hours;
	char minutes;
	char seconds;
};

void get_data_1(char datum, char * str2)
{
	if (datum < 0) datum *= -1;
	char str[5];
	int i = 0;
	if (datum == 0){
		str2[0] = '0';
		str2[1] = '\0';
	}
	else
	{
		while (datum != 0)
		{
			str[i++] = datum % 10 + '0';
			datum /= 10;
		}

		for (int j = 0; j < i; j++)
		{
			str2[j] = str[i - j - 1];
		}
		str2[i] = '\0';
	}
}
```

```
void get_data_2(short datum, char * str2)
{
	if (datum < 0) datum *= -1;
	char str[10];
	int i = 0;
	if (datum == 0){
		str2[0] = '0';
		str2[1] = '\0';
	}
	else
	{
		while (datum != 0)
		{
			str[i++] = datum % 10 + '0';
			datum /= 10;
		}

		for (int j = 0; j < i; j++)
		{
			str2[j] = str[i - j - 1];
		}
		str2[i] = '\0';
	}
}

void get_data_4(int datum, char * str2)
{
	if (datum < 0) datum *= -1;
	char str[15];
	int i = 0;
	if (datum == 0){
		str2[0] = '0';
		str2[1] = '\0';
	}
	else
	{
		while (datum != 0)
		{
			str[i++] = datum % 10 + '0';
			datum /= 10;
		}

		for (int j = 0; j < i; j++)
		{
			str2[j] = str[i - j - 1];
		}
		str2[i] = '\0';
	}
}

void shutdown()
{
	__asm
	{
		mov dx, 0x0604;
		mov ax, 0x2000;
		out dx, ax;
	}
}

void out_symbol(int color, char c, unsigned int strnum, unsigned int colnum)
{
	unsigned char* video_buf = (unsigned char*) VIDEO_BUF_PTR;
	video_buf += 80 * 2 * strnum + 2 * colnum;
	video_buf[0] = (unsigned char)c; // Символ (код)
	video_buf[1] = color; // Цвет символа и фона
}
```

```
char scan_ask(unsigned char a)
{
	char result;
	result = scancodes[a];
	return result;
}

int cmp_command(const char * str)
{
	int i;
	unsigned char* video_buf = (unsigned char*) VIDEO_BUF_PTR;
	video_buf += 80 * 2 * line + 4;
	for (i = 0; i < column - 2; i++)
	{
		if (str[i] != *(video_buf + i * 2))
			return 0;
	}
	if (str[i] != '\0') return 0;
	return 1;
}

void command_handler()
{
	if (cmp_command("info"))
	{
		out_str(0x07, sysmsg_info1, ++line);
		out_str(0x07, sysmsg_info2, ++line);
	}
	else if (cmp_command("help"))
	{
		out_str(0x07, sysmsg_help1, ++line);
		out_str(0x07, sysmsg_help2, ++line);
		out_str(0x07, sysmsg_help3, ++line);
		out_str(0x07, sysmsg_help4, ++line);
		out_str(0x07, sysmsg_help5, ++line);
		out_str(0x07, sysmsg_help6, ++line);
		out_str(0x07, sysmsg_help7, ++line);
	}
	else if (cmp_command("shutdown"))
	{
		shutdown();
	}
	else  out_str(0x07, sysmsg_unknown, ++line);
}

void on_key(unsigned int scan_code)
{

	if (scan_code == ENTER)
	{
		command_handler();

		out_str(0x07, sysmsg_emptystr, ++line);
		out_str(0x07, sysmsg_entry, ++line);
		column = 2;
		return;
	}
	if (scan_code == BACKSPACE)
	{
		if (column < 2) return;

		unsigned char* video_buf = (unsigned char*) VIDEO_BUF_PTR;
		video_buf += 80 * 2 * line + (column - 1) * 2;
		video_buf[1] = empty_color;
		video_buf[0] = empty_symbol;
		column -= 1;
		return;
	}
```

```
	if (column == DEADCOLUMN)
	{
		out_str(0x07, sysmsg_unknown, ++line);
		out_str(0x07, sysmsg_entry, ++line);
		column = 2;
		return;
	}

	out_symbol(0x07, scan_ask(scan_code), line, column++);

}

//-------------------------------------
//-------------------------------------
//-------------------------------------

#define PIC1_PORT (0x20)

//Чтение из порта
__inline unsigned char inb (unsigned short port)
{
	unsigned char data;
	__asm
	{
		push dx
		mov dx, port
		in al, dx
		mov data, al
		pop dx
	}
	return data;
}

//Запись
__inline void outb (unsigned short port, unsigned char data)
{
	__asm
	{
		push dx
		mov dx, port
		mov al, data
		out dx, al
		pop dx
	}
}

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);
	}
}

__declspec(naked) 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
```

```
		iretd
	}
}



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 прерываний, 0x02 - только IRQ1
(клавиатура)
}

//-------------------------------------
//-------------------------------------
//-------------------------------------

/*void lineup()
{
	int i;
	unsigned char* video_buf = (unsigned char*) VIDEO_BUF_PTR;
	while (video_buf != (unsigned char*)(VIDEO_BUF_PTR + 80 * (DEADLINE)))
	{
		video_buf[0] = video_buf[80];
		video_buf[1] = video_buf[81];
		video_buf++;
	}
	for(i = 0; i < 80; i++)
	{
		video_buf[i + 1] = empty_symbol;
		video_buf[i] = empty_color;
	}
	line--;
}*/

void out_str(int color, const char* ptr, unsigned int strnum)
{
	//if (line == DEADLINE) lineup();

	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++;
	}
}

void get_empty_symbol()
{
	unsigned char* video_buf = (unsigned char*) VIDEO_BUF_PTR;
	video_buf += 80*2*2;
	empty_symbol = video_buf[0];
	empty_color = video_buf[1];
}

int kmain()
{
	const char* hello = "Welcome to InfoOS (ms c edition)!";
	// Вывод строки
	out_str(0x07, hello, line++);

	get_empty_symbol();
```

```
        out_str(0x07, sysmsg_entry, line);
        column += 2;


        intr_disable();
        intr_init();
        keyb_init();
        //ticks_init();
        intr_start();
        intr_enable();

        while(1)
        {
                __asm
                {
                        hlt;
                }
        }
        return 0;
}
```

4.3. Используемые консольные команды

- Получение бинарника ядра

  cl.exe /GS- /c kernel.c

- Получение исполняемого файла (PE файла)

  link.exe /OUT:kernel.exe /BASE:0x10000 /NODEFAULTLIB /ENTRY:kmain /SUBSYSTEM:NATIVE kernel.obj

- Получение информации о кол-ве секторов кода и данных, необходимых для корректной загрузки в память загрузчиком и дальнейшей передачи работы ядру

  dumpbin /headers kernel.exe

- Получение бинарника загрузчика

  ..\yasm.exe –f bin –o ..\my\bootsect.bin ..\my\bootsect.asm

- Эмуляция в QEMU

  start qemu-system-i386w.exe –boot c –m 512 –L Bios –rtc clock=host,base=2010-12-03T05:06:07 –fda ..\my\bootsect.bin –fdb ..\my\kernel.exe