1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 1**

1. по дисциплине «Операционные системы»
2. Выполнил
3. студент гр. 23508/4 Е.Г. Проценко
4. Проверил
5. профессор Е.Ю. Резединова
6. Санкт-Петербург
7. 2016
8. **Формулировка задания**

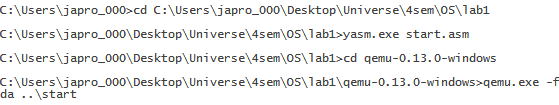
Цель работы — изучение основ разработки ОС, принципов низкоуровневого взаимодействия с аппаратным обеспечением, программирования системной функциональности и процесса загрузки системы.

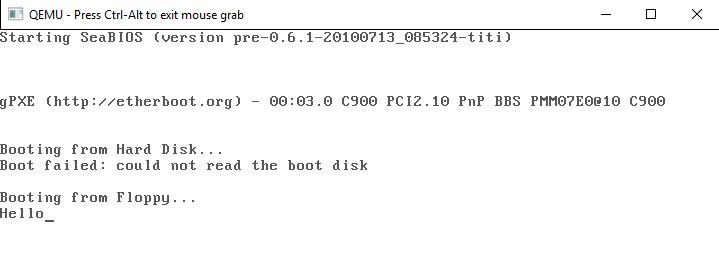
1. **Ход работы**
   1. **Получение варианта**

От преподавателя был получен вариант 18.

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

* 1. **Компиляция и загрузка на эмуляторе пример загрузочного сектора**





* 1. **Разработка загрузчика для загрузки минимального ядра и самого ядра**
     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.

* + 1. 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.

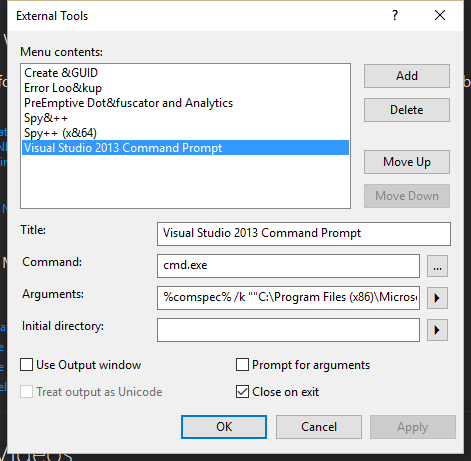
Теперь программа работает в бесконечном цикле с использованием инструкции hlt, для потребления меньшего количества ресурсов.

* + 1. 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.

1. **Некоторые сложности при работе**
   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.

* 1. 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…
  2. 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**

**}**

1. **Приложение**
   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 А20**

**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**

* 1. **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;**

**}**

* 1. **Используемые консольные команды**
* Получение бинарника ядра

**cl.exe /GS- /c kernel.c**

* Получение исполняемного 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**