**Проектирование ОС и компонентов**

Разработка первичных загрузчиков выбранными средствами. Мультизагрузка

**Теоретическая часть**

## Программа работы:

Разработка первичных загрузчиков выбранными средствами. Мультизагрузка.

1. В загрузочный сектор поместить программу вывода на экран произвольного сообщения. Программу написать на ассемблере (предпочтительно встраиваемом в стандартную конфигурацию) и на С (при необходимости допустимы ассемблерные вставки), убедиться в работоспособности обоих вариантов. См. пример кода ниже. В качестве носителя предпочтителен выбор флэш. Начать можно с экспериментов над виртуальной дискетой, как показано ниже.

2. Создать первичный загрузчик для виртуального, а затем реального носителя, (пример показан для виртуальной дискеты и ФС FAT32), который будет находить файл программы на носителе и загружать ее на выполнение. Сделать 2 варианта на разных языках (asm и С). Привести весь план экспериментов и результаты их проведения в виде «логфайлов» и screenshot–ов. ФС можно выбрать на свое усмотрение.

Лабораторная работа выполнялась в Virtual Box c ОС Ubuntu 16.04

|  |
| --- |
| anar@anar-Vittualbox:~/files/1$ uname -a  Linux anar-Vittualbox 4.8.0-49-generic #52~16.04.1-Ubuntu SMP Wed Jul 13 01:07:32 UTC 2016 x86\_64 x86\_64 x86\_64 GNU/Linux |

**Теоретическая часть**

BIOS выполняет некоторые аппаратные тесты, далее начинается загрузка операционной системы с любого носителя. Это может быть жесткий диск, либо floppy диск или flash диск. Загрузка системы начинается с того, что BIOS считывает первый сектор жесткого диска, размещает его в памяти по адресу 0000:7С00h и передает туда управление. Этот сектор называется главным загрузочным сектором (MBR), размер 512 байт в начале диска.

BIOS загружает MBR по адресу 7C00h, поэтому в начале ассемблерного кода должна стоять директива ORG 7C00h. Затем BIOS передаёт управление по физическому адресу 0x7C00 (то есть сектору MBR), предварительно записав в регистр DL номер диска, с которого этот сектор считан. MBR - программа, которая загружает основное ядро ​​операционной системы. Необходимо указать директиву USE16, потому что загрузчик выполняется в 16-разряном режиме.

**Загрузка с floppy disk диска**

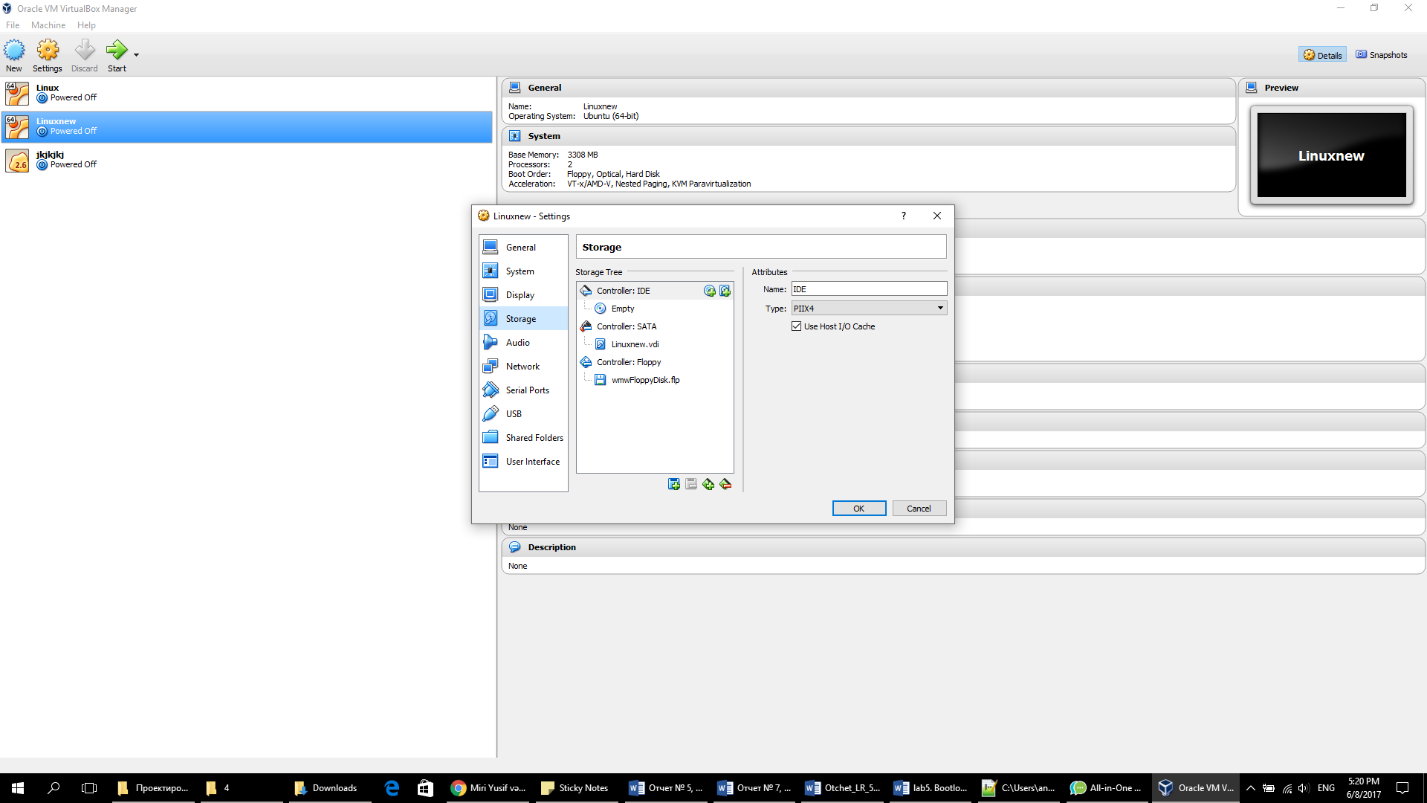
Надо создать пустой образ floppy диска. Для этого надо воспользоватся утилитой WinImage, создается образ floppy.ima/floppy.flp. В настройках виртуальной машины добавляется привод гибкого диска и выберается образ ранее созданного гибкого диска floppy.ima (New -> Standard format: 1.44 MB -> OK, Image -> Change format -> Select Custom Image format -> FAT 12/16).

Рисунок 1

ПО умолчанию загрузка с гибких носителей имеет больший приоритет. В целом проверить это возможно следующим способом: в меню запуска выбрать пункт «Включить питание из BIOS» и после загрузки BIOS выбрать раздел Boot, где уже можно задать приоритеты.

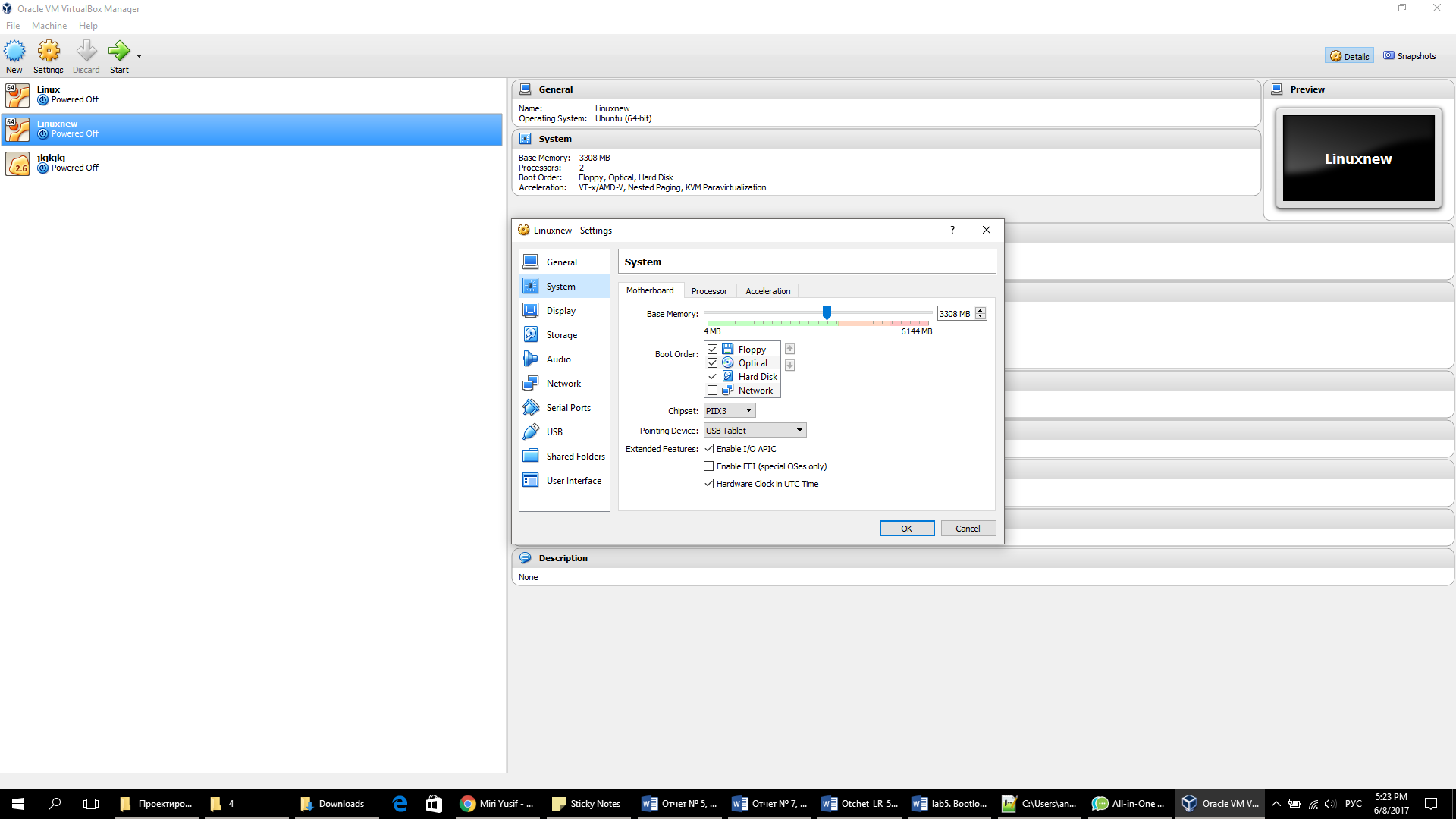


Рисунок 2

Код программы (boot.asm):

|  |
| --- |
| use16 ; загрузчик выполняется в 16-разряном режиме  org 0x7C00 ;указатель на адрес, где будет находиться код, после загрузки системы  ; Установится видеорежим, очистка экрана  mov ax, 3  int 10h  ; Выводится приветственного сообщения  mov si, mHello ; сохранение указателя на строку в si  call print  ; Зависается в бесконечном цикле  die: jmp short die  mHello db 'Hello, world - i was booted!',10,13,0  ; Подпрограмма вывода  print:  cld pusha  .PrintChar:  lodsb ; Считает байт по адресу SI в AL, SI увеличиться на 1  test al, al ; проверка, что в регистре AL - 0 (конец строки)  jz short .Exit ; завершение процедуры  mov ah, 0x0E ; сообщаем BIOS, что выводится один символ на экран  mov bh, 0x00 ;нет страниц  mov bl, 0x07 ; выбор цвета  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходится к печати следующего символа  .Exit:  popa  ret ; выход из процедуры  TIMES 510 - ($ - $$) db 0 ;заполняется оставшееся место на секторе 0  DW 0xAA55 ;добавление специальной сигнатуры в конце загрузчика |

Перед вызовом прерывания необходимо установить следующие значения в регистры

AH - 0x0E ; выводим один символ на экран  
BH - 0x00 ;нет страниц  
BL - 0x07 ;выбор цвета.

Компилируется программа:

|  |
| --- |
| anar@anar-Vittualbox:~/files/1$ nasm -f bin boot.asm -o boot |

Компьютер перезагружается и можно увидить следующий результат:

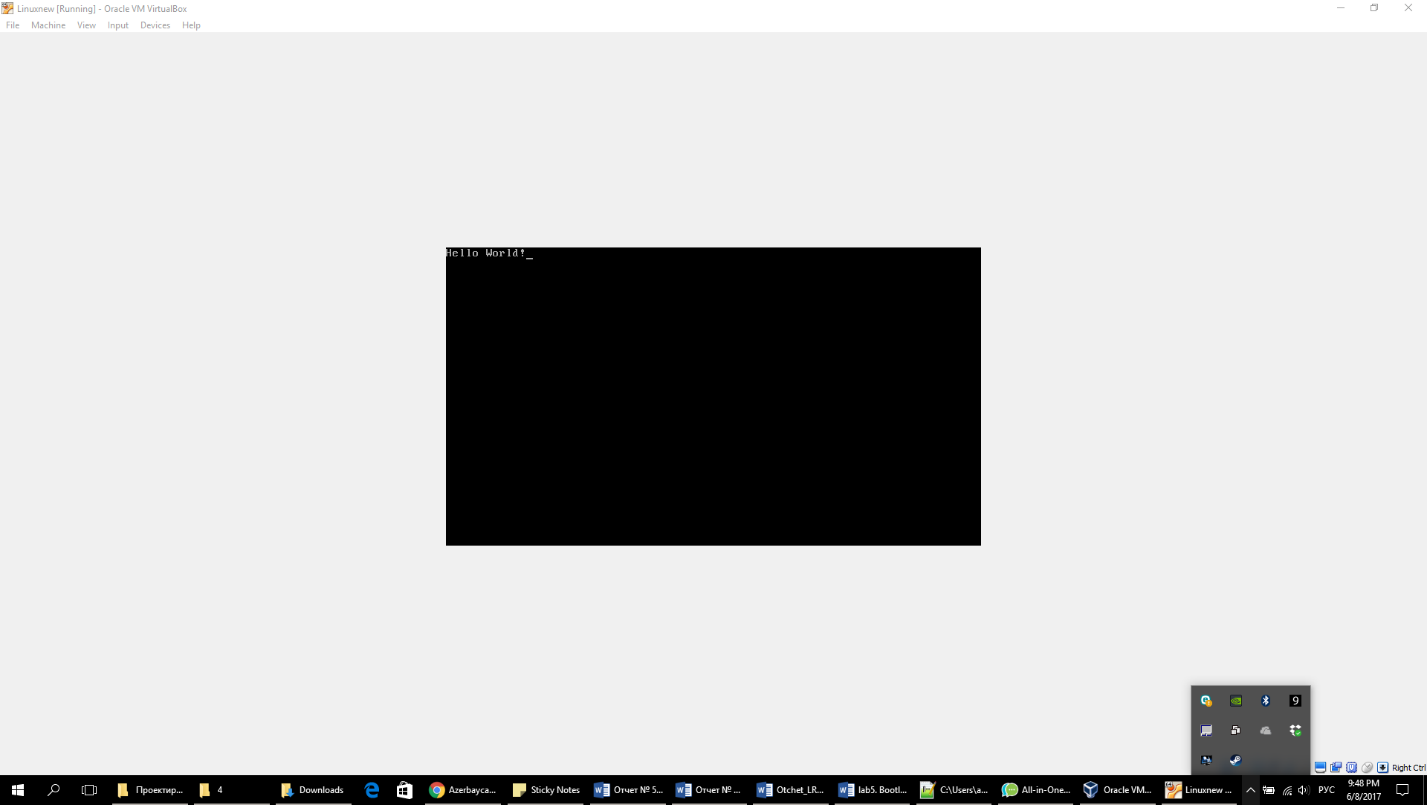


Рисунок 3

Как видно BIOS удалось обнаружить на floppy диске созданный загрузчик, строка успешно выведена на экран.

# Модификация загрузчика, первичный загрузчик выводит сообщение

# Проверка на эмулируемом жёстком диске

Внесётся модификация в первичный загрузчик, чтобы он печатал сообщение, о том, что чтение закончено.

Код boot.asm:

|  |
| --- |
| [BITS 16]  [ORG 0x7C00]  BS\_jmpBoot:  jmp start  nop  %include "fatTable.asm"  start:  cli ; запрещение аппаратных прерываний  MOV DH, 0x0 ;head (0=base)  MOV CH, 0x0 ;track/cylinder  MOV CL, 0x02 ;номер сектора (нумерация начинается с 1)  MOV BX, 0x1000 ;адрес RAM для загрузки ядра  MOV ES, BX ;старший адрес RAM 0x1000  MOV BX, 0x0 ;младший адрес RAM 0x0  ReadSector:  MOV AH, 0x02 ;чтение сектора диска  MOV AL, 0x01 ;количество секторов для чтения  INT 0x13  ; Вывод сообщения  mov si, message ; сохранение указателя на строку в si  call print  ;указатель на адрес RAM (0x1000)  MOV AX, 0x1000  MOV DS, AX  MOV ES, AX  MOV FS, AX  MOV GS, AX  MOV SS, AX  JMP 0x1000:0x0  message db 'Read of second loader finished!',10,13,0  ; Подпрограмма вывода  print:  cld  pusha  .PrintChar:  lodsb ; Считать байт по адресу SI в AL, SI увеличиться на 1  test al, al ; проверка, что в регистре AL - 0 (конец строки)  jz short .Exit ; завершение процедуры  mov ah, 0x0E ; сообщаем BIOS, что выводим один символ на экран  mov bh, 0x00 ;нет страниц  mov bl, 0x07 ; выбор цвета  mov bx, 0x00 ;  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходим к печати следующего символа  .Exit:  popa  ret ; выход из процедуры  TIMES 510 - ($ - $$) db 0 ;заполняет оставшееся место на секторе 0  DW 0xAA55 ;добавление специальной сигнатуры в конце загрузчика |

Код kernel.asm можно сказать не изменился, были убраны только первые две строчки, чтобы не очищался экран.

Первичный загрузчик, расположенный в первом секторе реального флэш-накопителя, выводит сообщение и передаёт управление вторичному загрузчику, расположенному во втором секторе, в результате на экране появляется сообщения от первичного и вторичного загрузчика.

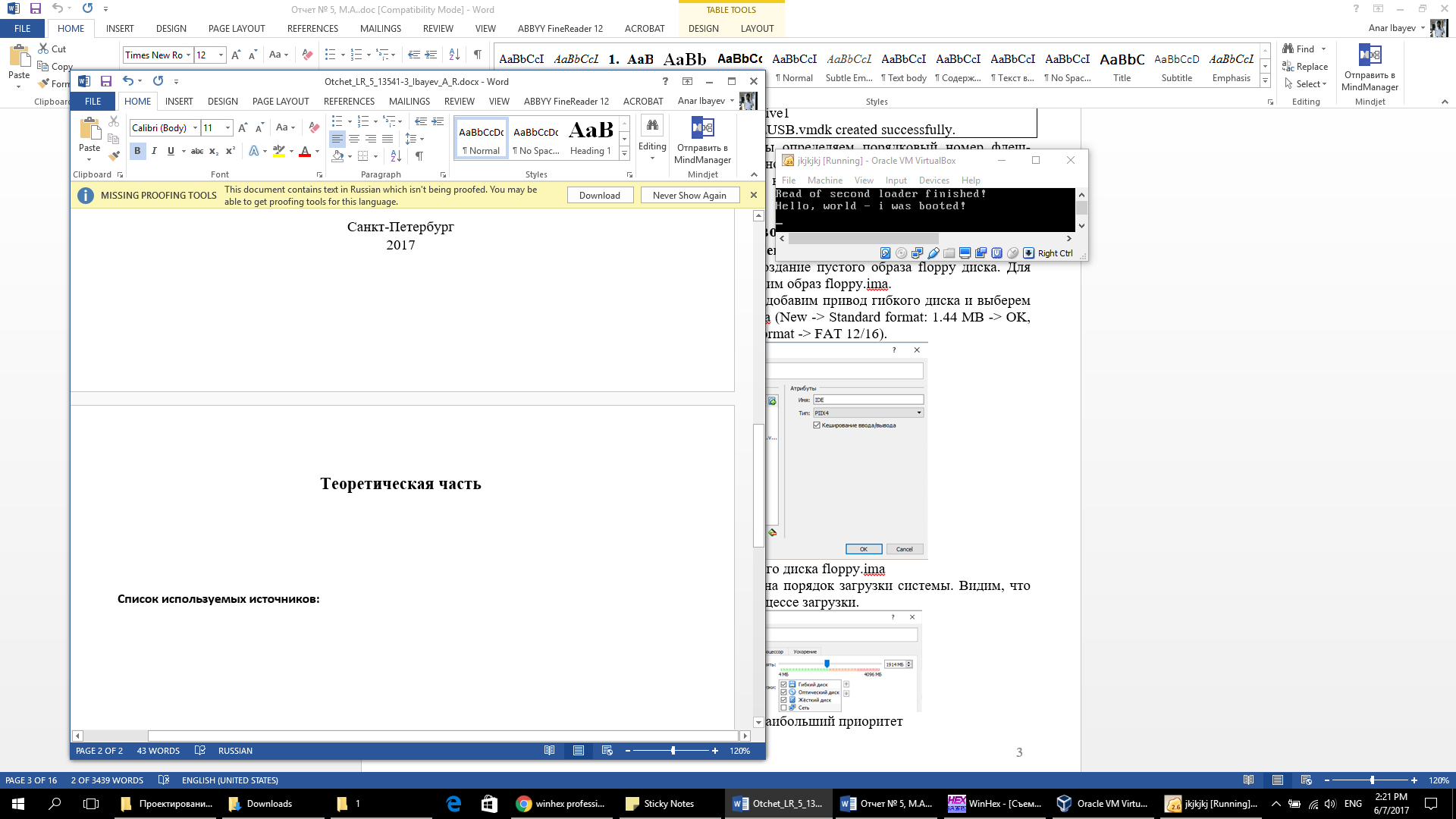


Рисунок 4

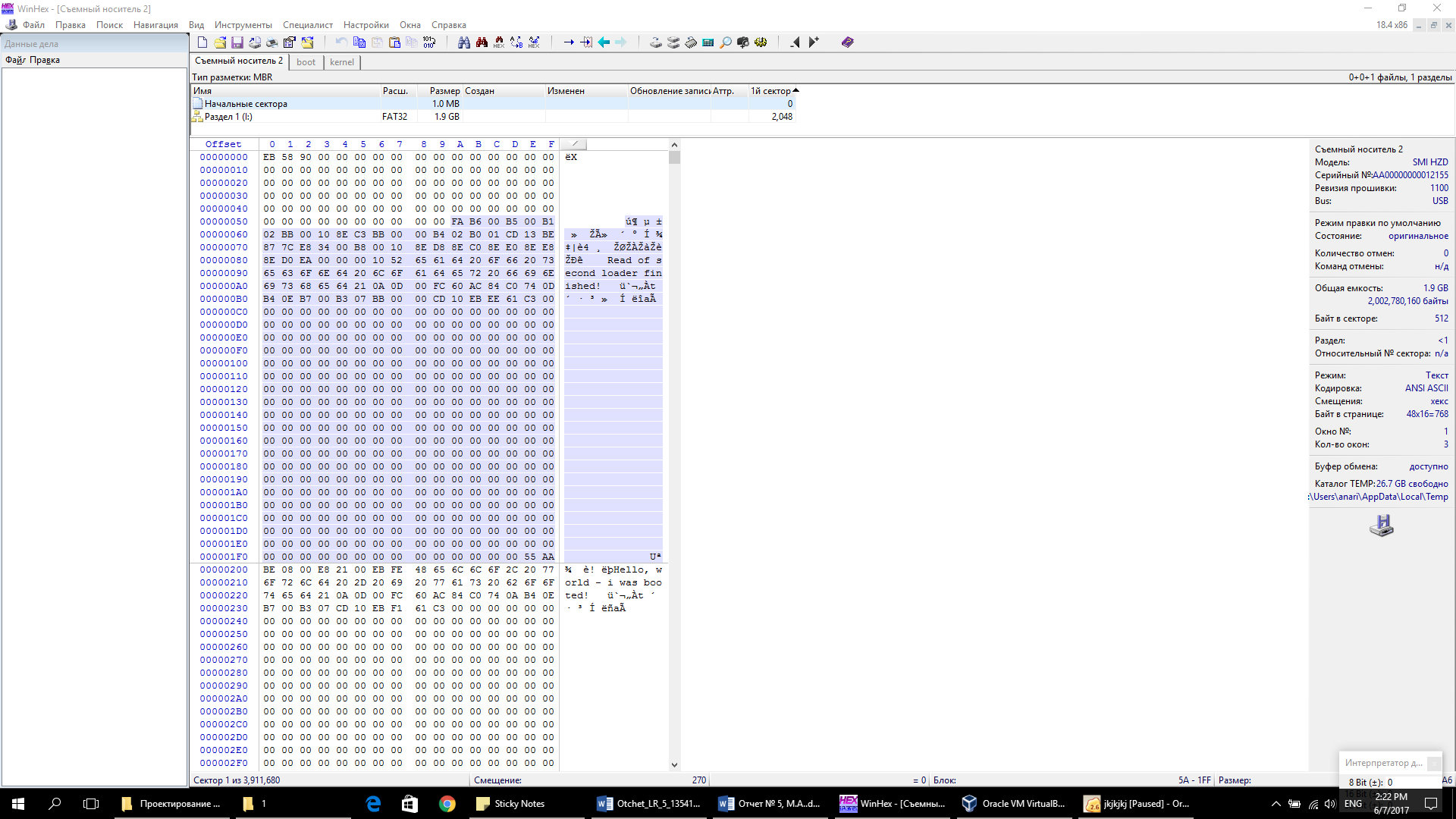
Полученные в результате компиляции файлы необходимо поместить в первые два раздела флеш-накопителя: файл boot, начиная со смещения 0x5A, - в первый сектор по смещению 0x5A, файл kernel – во второй сектор. Для этого была использована программа WinHex. После сохранения секторов при запуске виртуальной машины на экране появляются сообщения.

Рисунок 5

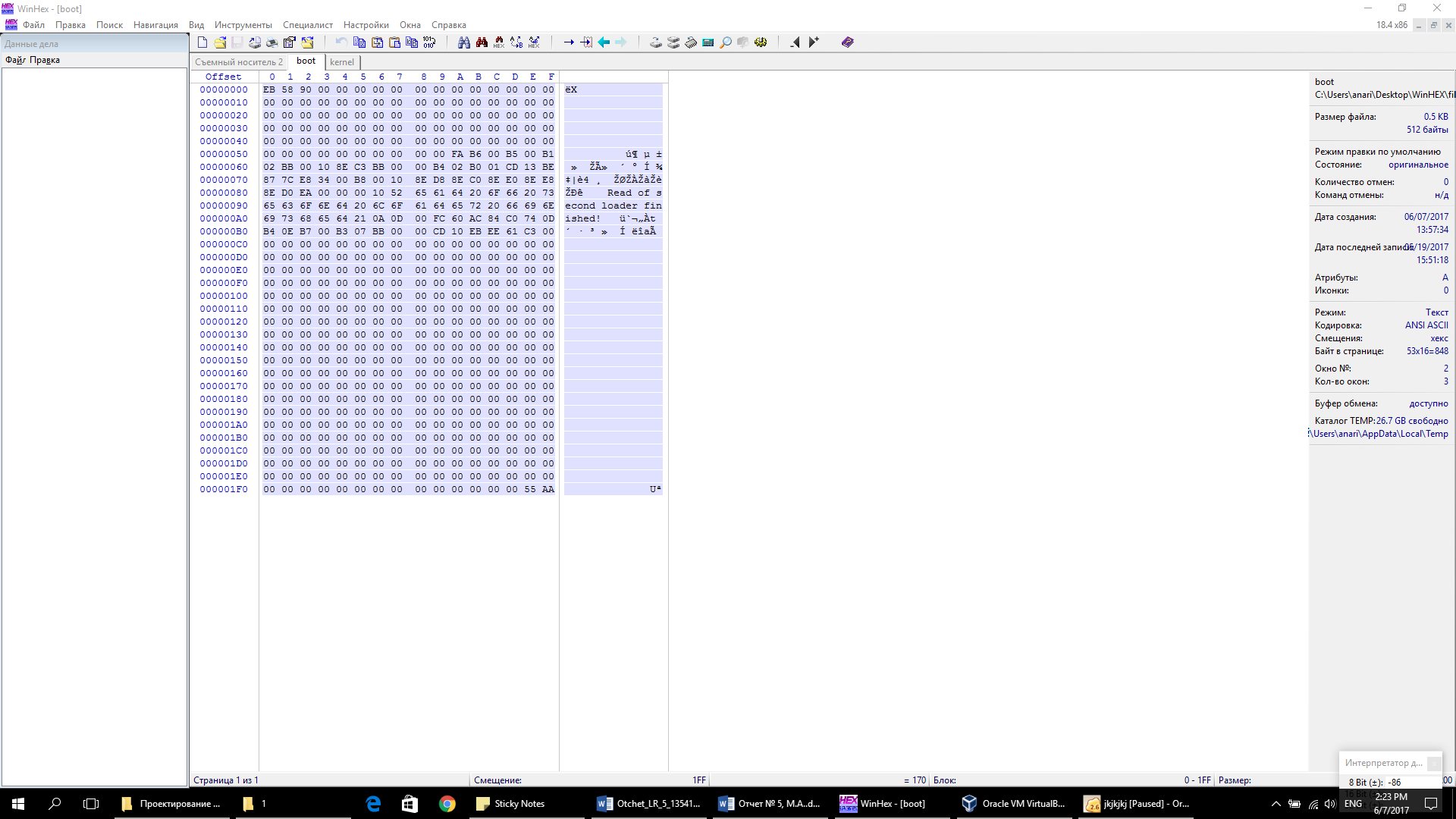


Рисунок 6

**Загрузка на исполнение файла из ФС FAT32, мультизагрузчик**

Первичный загрузчик будет использован из предыдущего пункта, он будет копировать 2-ой и 3-ий сектор вторичного загрузчика, а потом передавать ему управление. Пользователь выбирает одну из двух программ, нажимая клавишу 1 или 2. Если пользователь нажал другую клавишу, то выводится ошибка. Если выбрана программа, то начинается поиск файла в корневой директории. Если файл найден, то он загружается, иначе выводится ошибка.

Код boot.asm:

|  |
| --- |
| [BITS 16]  [ORG 0x7C00]  BS\_jmpBoot:  jmp start0  nop  %include "fatTable.asm"  start0:  cli ; запрещение аппаратных прерываний  MOV DH, 0x0 ;head (0=base)  MOV CH, 0x0 ;track/cylinder  MOV CL, 0x02 ;номер сектора (нумерация начинается с 1)  MOV BX, 0x1000 ;адрес RAM для загрузки ядра  MOV ES, BX ;старший адрес RAM 0x1000  MOV BX, 0x0 ;младший адрес RAM 0x0  ReadSector:  MOV AH, 0x02 ;чтение сектора диска  MOV AL, 0x02 ;количество секторов для чтения  INT 0x13  ; Вывод сообщения  mov si, message ; сохранение указателя на строку в si  call print  ;указатель на адрес RAM (0x1000)  MOV AX, 0x1000  MOV DS, AX  MOV ES, AX  MOV FS, AX  MOV GS, AX  MOV SS, AX  JMP 0x1000:0x0  message db 'Read of second loader finished!',10,13,0  ; Подпрограмма вывода  print:  cld  pusha  .PrintChar:  lodsb ; Считать байт по адресу SI в AL, SI увеличиться на 1  test al, al ; проверка, что в регистре AL - 0 (конец строки)  jz short .Exit ; завершение процедуры  mov ah, 0x0E ; сообщается BIOS, что выводится один символ на экран  mov bh, 0x00 ;нет страниц  mov bl, 0x07 ; выбор цвета  mov bx, 0x00 ;  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходит к печати следующего символа  .Exit:  popa  ret ; выход из процедуры  TIMES 510 - ($ - $$) db 0 ;заполняет оставшееся место на секторе 0  DW 0xAA55 ;добавление специальной сигнатуры в конце загрузчика |

Код HELLO1.asm:

|  |
| --- |
| bits 16  [ORG 0x0100]  cli  ; Вывод приветственного сообщения  mov si, mHello ; сохранение указателя на строку в si  call print  ;mov al, "c"  ;mov ah, 0x0E ;Флаг того, что нужно вывести на экран один символ  ;mov bh, 0x00 ;Номер страницы  ;mov bl, 0x07 ;Флаг того, что выводится светлый текст на черном фоне  ;int 0x10 ;Вызов прерывания видео  ; Зависаем в бесконечном цикле  die: jmp short die  mHello db 'Hello world 1!',10,13,0  ; Подпрограмма вывода  print:  cld  pusha  .PrintChar:  lodsb ; Считать байт по адресу SI в AL, SI увеличиться на 1  test al, al ; проверка, что в регистре AL - 0 (конец строки)  jz short .Exit ; завершение процедуры  mov ah, 0x0E ; сообщается BIOS, что выводится один символ на экран  mov bh, 0x00 ;нет страниц  mov bl, 0x07 ; выбор цвета  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходит к печати следующего символа  .Exit:  popa  ret ; выход из процедуры  TIMES 512 - ($ - $$) db 0 ;заполняет оставшееся место на секторе 0 |

Код HELLO2.asm:

|  |
| --- |
| bits 16  [ORG 0x0100]  cli  ; Вывод приветственного сообщения  mov si, mHello ; сохранение указателя на строку в si  call print  ; Зависает в бесконечном цикле  die: jmp short die  mHello db 'Hello world 2!',10,13,0  ; Подпрограмма вывода  print:  cld  pusha  .PrintChar:  lodsb ; Считать байт по адресу SI в AL, SI увеличиться на 1  test al, al ; проверка, что в регистре AL - 0 (конец строки)  jz short .Exit ; завершение процедуры  mov ah, 0x0E ; сообщается BIOS, что выводится один символ на экран  mov bh, 0x00 ;нет страниц  mov bl, 0x07 ; выбор цвета  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходится к печати следующего символа  .Exit:  popa  ret ; выход из процедуры  TIMES 512 - ($ - $$) db 0 ;заполняет оставшееся место на секторе 0 |

Компиляция программ:

|  |
| --- |
| C:\Users\anari\Desktop\WinHEX\files\4 yasm-1.3.0-win64.exe -f bin HELLO1.asm -o HELLO1.bin C:\Users\anari\Desktop\WinHEX\files\4yasm-1.3.0-win64.exe -f bin HELLO2.asm -o HELLO2.bin C:\Users\anari\Desktop\WinHEX\files\4yasm-1.3.0-win64.exe -f bin multiboot.asm -o multiboot C:\Users\anari\Desktop\WinHEX\files\4yasm-1.3.0-win64.exe -f bin boot.asm -o boot |

Полученные в результате компиляции файлы необходимо поместить в первые три раздела флеш-накопителя: файл boot, начиная со смещения 0x5A, - в первый сектор по смещению 0x5A, файл multiboot – во второй и третий сектор, начиная со смещения 0x5A, причём в начало второго сектора нужно скопировать FAT-таблицу из первого сектора. Для этого была использована программа WinHex

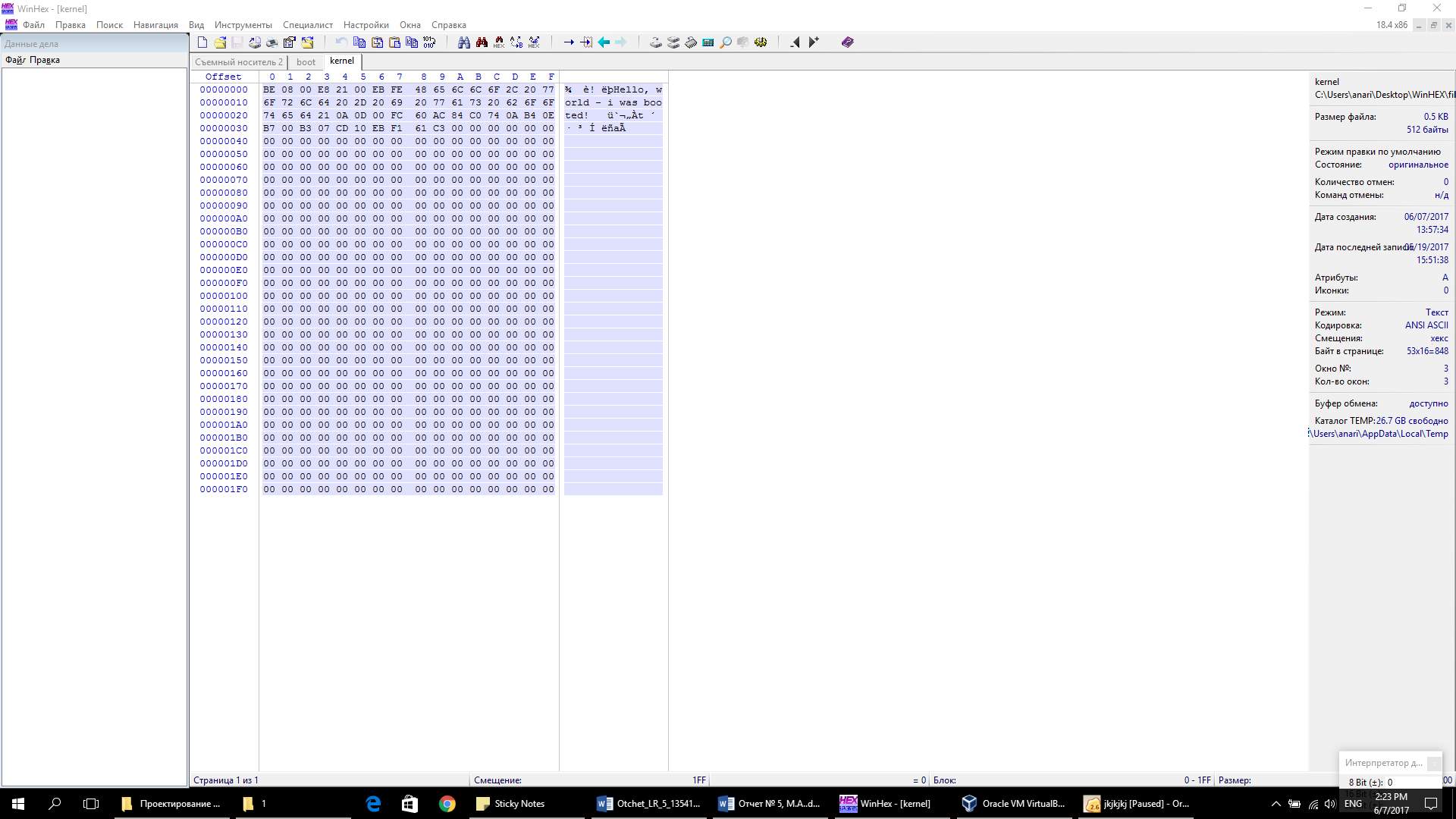
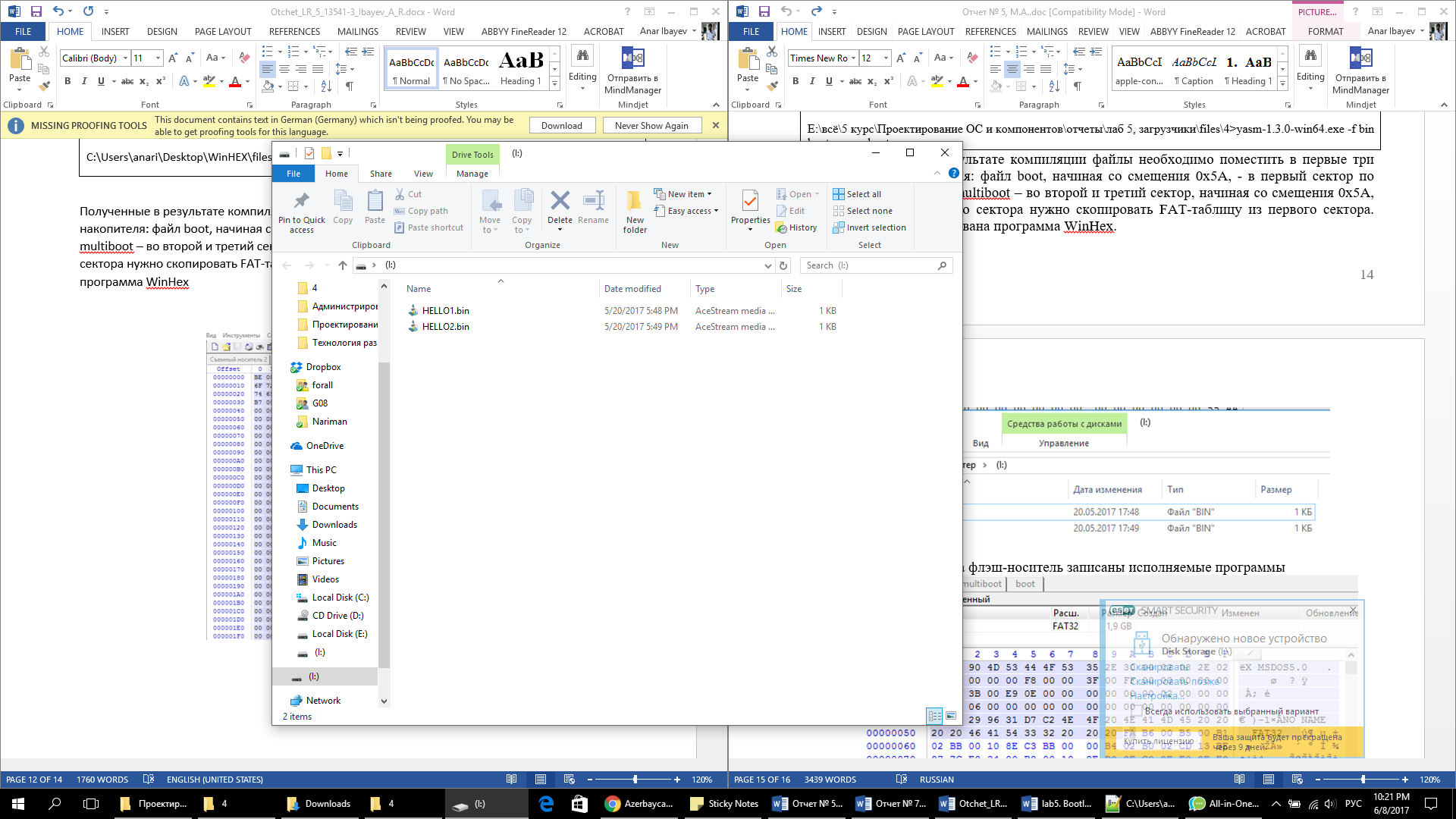


Рисунок 7

Рисунок 8

При нажатии клавиши отличной от 1 и 2 выводится ошибка. При нажатии клавиши 1 исполняется первая программа, при нажатии клавиши 2 исполняется вторая программа.

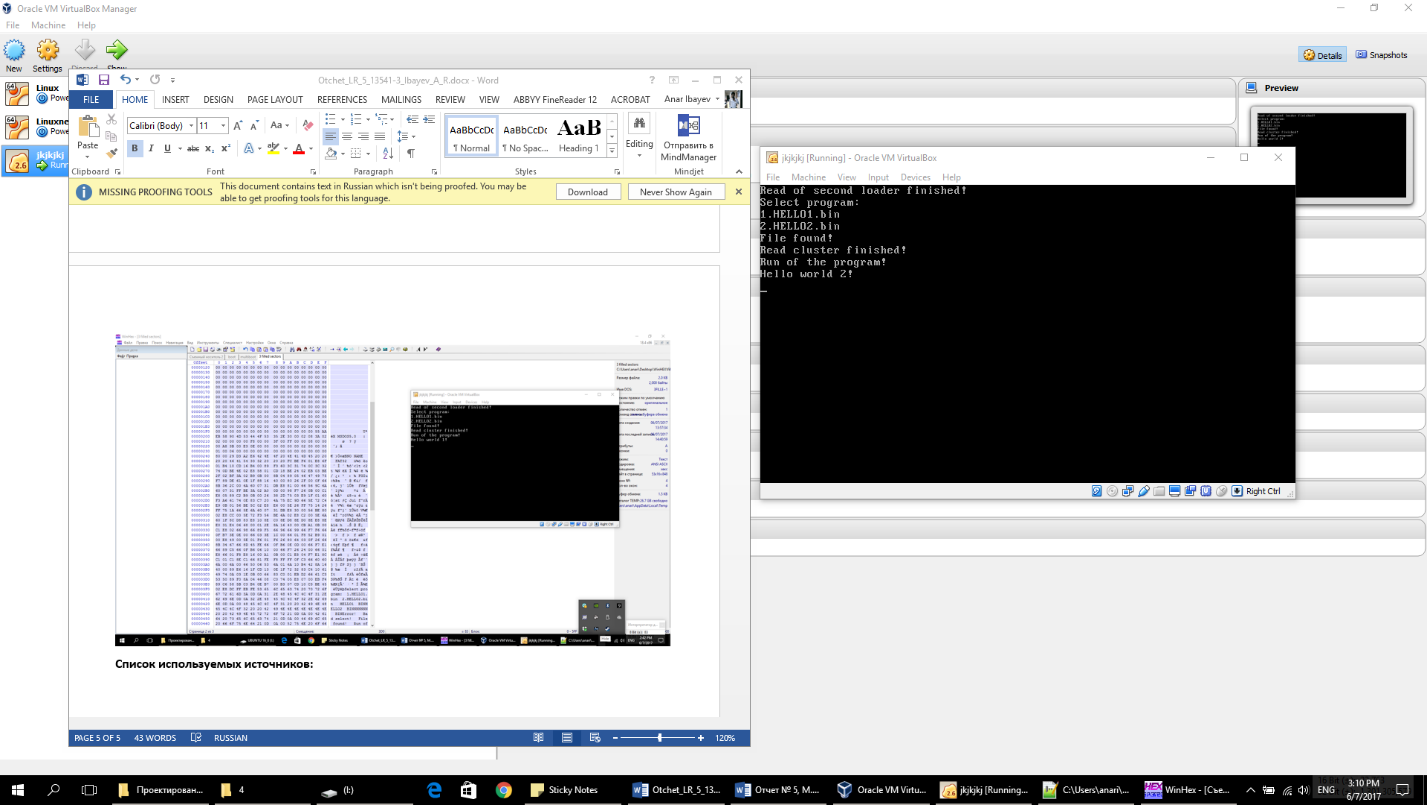


Рисунок 9 Нажата клавиша отличная от 1

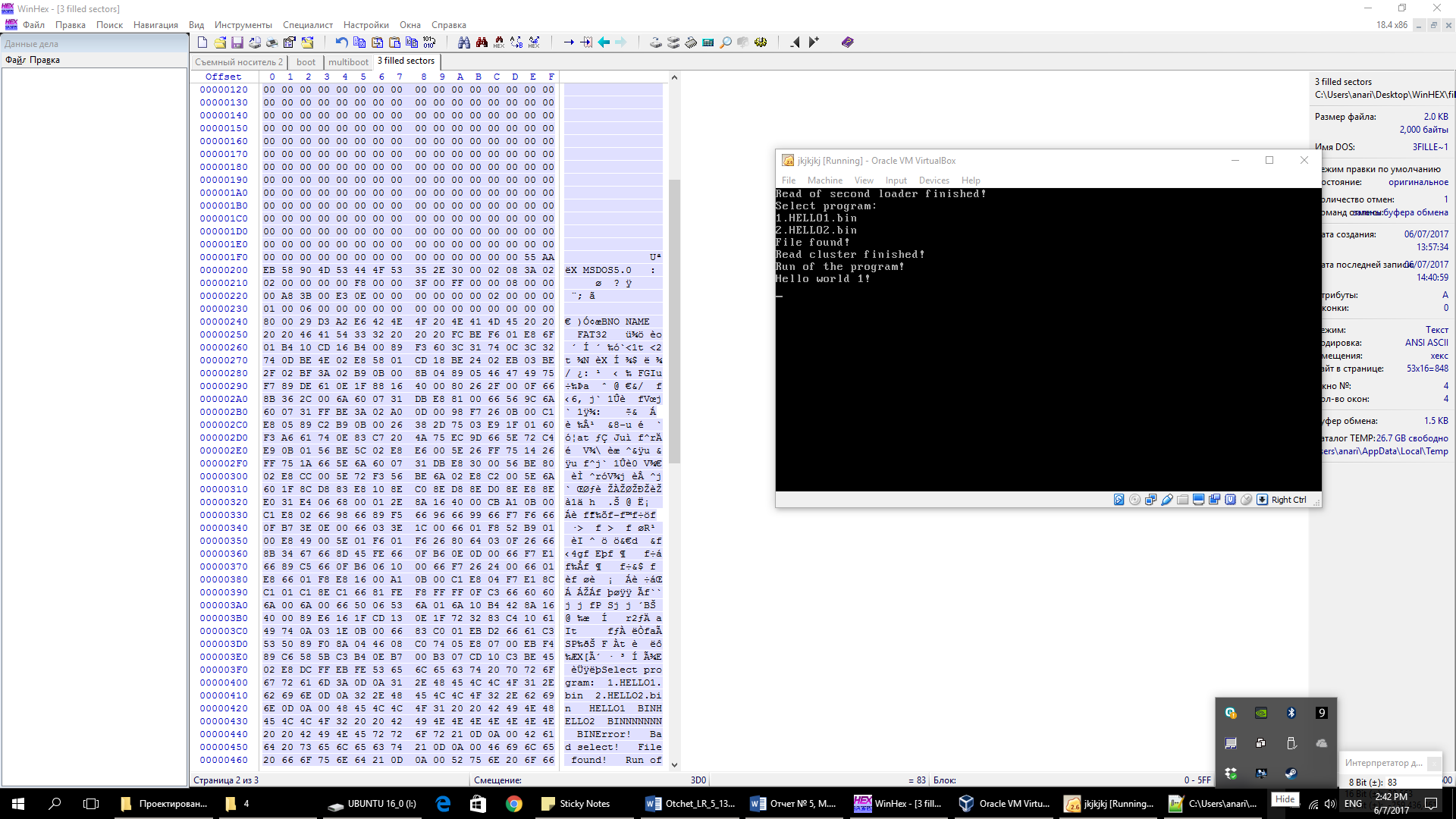


Рисунок 10 Нажата клавиша отличная от 2

**Вывод**

Мы рассмотрели, что такое загрузчик, как работает BIOS, и как компоненты системы взаимодействуют при загрузке системы. Проводились эксперименты с образом floppy диском и реальным флэш-накопителем. Для записи загрузчиков на флэш-накопитель использовалась программа WinHex, предварительно лучше сделать резервную копию (Файл -> Создать образ диска).

Был реализован загрузчик, выводящий приветственное сообщение. Был реализован первичный загрузчик, передающий управление вторичному загрузчику. Также был реализован загрузчик, позволяющий выбрать на исполнение одну из программ, которые загружаются на исполнение из FS FAT32.

**Список используемых источников:**

1. <http://wiki.osdev.org/ATA_in_x86_RealMode_(BIOS)>
2. <https://github.com/alexfru/BootProg/blob/master/boot32.asm>