Санкт-Петербургский Государственный Политехнический Университет

Институт информационных технологий и управления

Кафедра Компьютерных Систем и Программных Технологий

Отчет по лабораторной работе № 5

По дисциплине «Проектирование ОС и компонентов»

По теме «Разработка первичных загрузчиков выбранными средствами. Мультизагрузка»

**Работу выполнила студентка группы № 13541/3**

Шаляпин Н.С. \_\_\_\_\_\_\_\_\_\_\_

**Работу принял преподаватель:**

Душутина Е.В. \_\_\_\_\_\_\_\_\_\_\_

.

Санкт-Петербург

2017

## 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 14.04.

|  |
| --- |
| osboxes@osboxes:~/files/1$ uname -a  Linux osboxes 4.4.0-31-generic #50~14.04.1-Ubuntu SMP Wed Jul 13 01:07:32 UTC 2016 x86\_64 x86\_64 x86\_64 GNU/Linux |

**Теория**

При подаче на устройства питания начинает выполняться BIOS (Basic Input / Output System). Он выполняет некоторые аппаратные тесты, далее начинается загрузка операционной системы с любого носителя. Это может быть либо жесткий диск, либо floppy диск, 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 (Master Boot Record, сокращенно MBR), размер 512 байт в начале диска. В начале MBR расположен код загрузчика. Следом за ним идет таблица разделов, описывающая схему разбиения логических дисков. В конце сектора находится сигнатура 55h AAh, подтверждающая, что это действительно MBR. Если хотя бы один из двух последних байтов отличается по значению, считается, что первый сектор не является MBR и не содержит осмысленной информации.

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. Кроме того, необходимо указать директиву USE16, потому что загрузчик выполняется в 16-разряном режиме.

Попробуем реализовать на ассемблере простейшую программу, которая выводит приветственное сообщение на экран, и поместим ее в загрузочный сектор.

**Подготовка рабочей станции**

Для выполнения экспериментов необходимо подготовить виртуальную машину с загрузкой из flash-накопителя c ФС FAT32. Виртуальная машина используется для того, чтобы не повредить систему на основной машине. Преимущество VirtualBox в том, что он позволяет эмулировать загрузку виртуальной машины с флеш-накопителя как загрузку с обычного жесткого диска. Для этого необходимо связать флеш-накопитель с VirtualBox с помощью утилиты VBoxManage:

|  |
| --- |
| C:\WINDOWS\system32>diskpart  Microsoft DiskPart, версия 10.0.14393.0  (С) Корпорация Майкрософт (Microsoft Corporation), 1999-2013.  На компьютере: LAPTOP-JGNT6U4K  DISKPART> list disk  Диск ### Состояние Размер Свободно Дин GPT  -------- ------------- ------- ------- --- ---  Диск 0 В сети 465 Gбайт 0 байт \*  Диск 1 В сети 1912 Mбайт 0 байт  DISKPART> exit  Завершение работы DiskPart...  C:\WINDOWS\system32>cd C:\Program Files\Oracle\VirtualBox  C:\Program Files\Oracle\VirtualBox>VBoxManage internalcommands createrawvmdk -filename C:\Users\Admin\USB.vmdk -rawdisk \\.\PhysicalDrive1  RAW host disk access VMDK file C:\Users\Admin\USB.vmdk created successfully. |

Сначала с помощью утилиты diskpart мы определяем порядковый номер флеш-накопителя, а затем создаем файл виртуального жесткого диска USB.vmdk. Далее необходимо создать виртуальную машину и в качестве ее жесткого диска указать созданный файл USB.vmdk.

1. **Пример реализации вывода сообщения из MBR.** 
   1. **Загрузка с образа внешнего носителя (floppy disk)**

Первым шагом данного этапа является создание пустого образа floppy диска. Для этого воспользуемся утилитой WinImage, создадим образ floppy.ima.

Далее в настройках виртуальной машины добавим привод гибкого диска и выберем образ ранее созданного гибкого диска floppy.ima (New -> Standard format: 1.44 MB -> OK, Image -> Change format -> Select Custom Image format -> FAT 12/16).

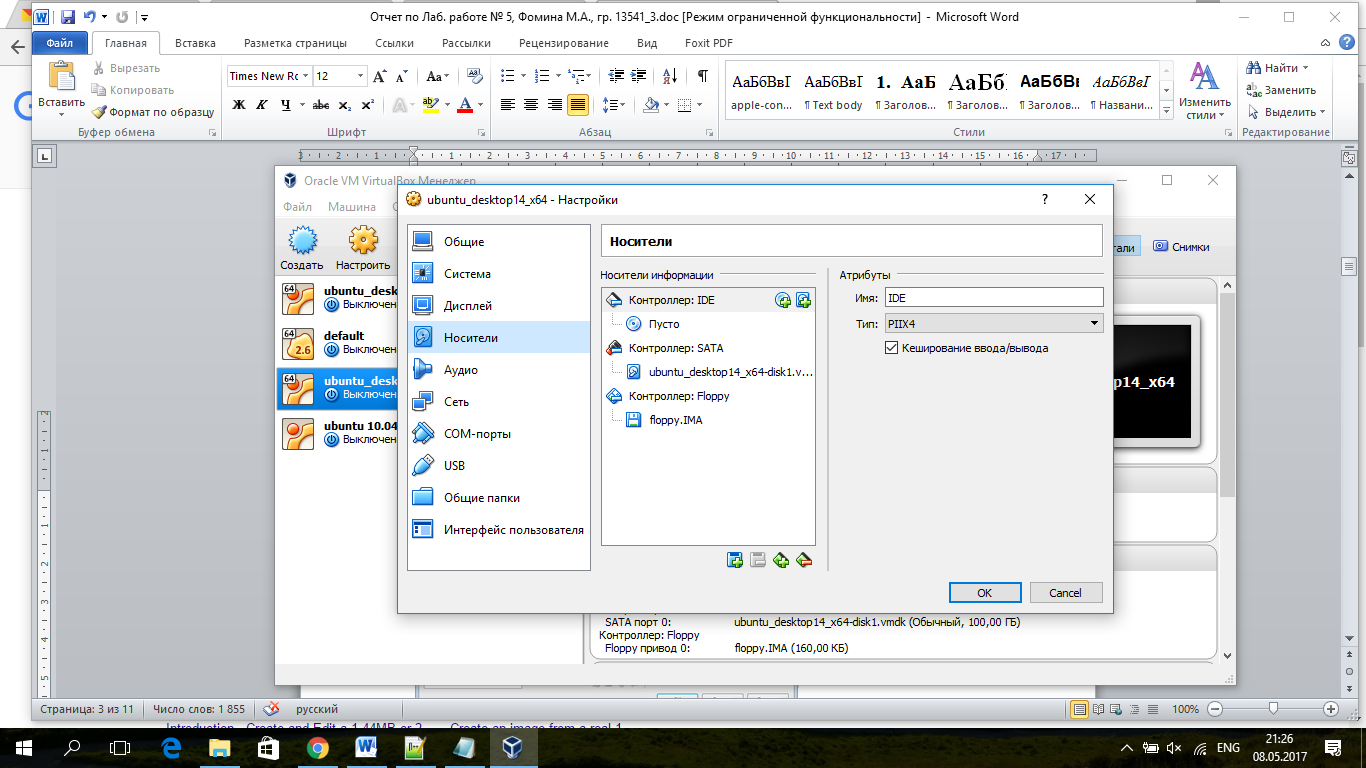


Рис. 1. Добавление гибкого диска floppy.ima

Также в настройках, обратим внимание на порядок загрузки системы. Видим, что гибкий диск имеет наибольший приоритет в процессе загрузки.

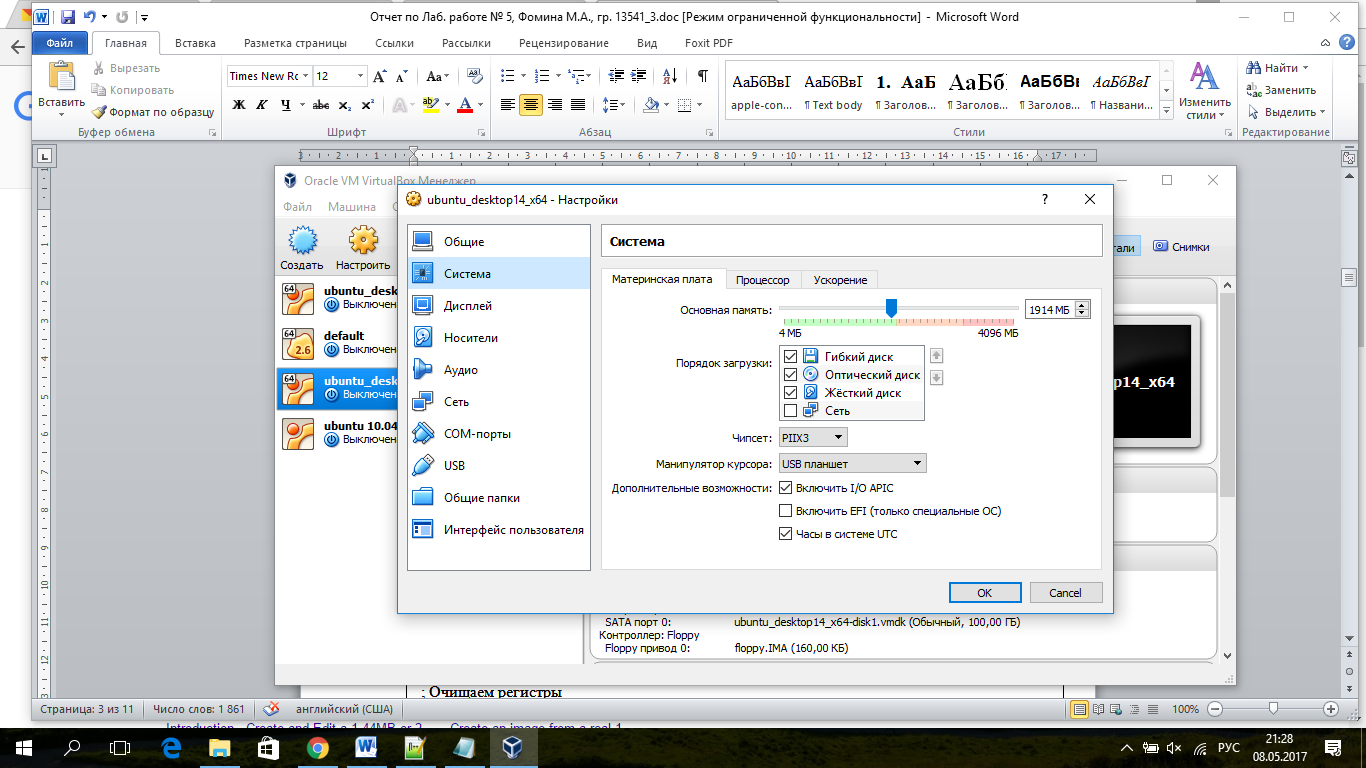


Рис. 2. Гибкий диск имеет наибольший приоритет

Текст программы (boot.asm):

|  |
| --- |
| use16 ; загрузчик выполняется в 16-разряном режиме  org 0x7C00 ;указатель на адрес, где будет находиться код, после загрузки системы  ; Установим видеорежим, очистка экрана  mov ax, 3  int 10h  ; Вывод приветственного сообщения  mov si, mHello ; сохранение указателя на строку в si  call print  ; Зависаем в бесконечном цикле  die: jmp short die  mHello db 'Hello, world - i was booted!',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  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 |

Обратим внимание, что для вывода строки на экран вызывается прерывание 0x10. Перед вызовом прерывания необходимо установить следующие значения в регистры

AH - 0x0E ; выводим один символ на экран

BH - 0x00 ;нет страниц

BL - 0x07 ;выбор цвета.

Компилируем программу:

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

Мы получили исполняемый код (файл **boot**). Теперь можно записать его в MBR раздел floppy диска и, при загрузке с этого диска, мы увидим на экране приветственное сообщение.

После необходимо поместить созданный исполняемый файл на floppy диск. Выполним команду dd со следующими параметрами:

* if — чтение файла boot
* of — запись в файл /dev/fd0 (используемый floppy disk)
* bs — размер блока 512.

|  |
| --- |
| osboxes@osboxes:~/files/1$ sudo dd if=boot bs=512 of=/dev/fd0  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.0572043 s, 9.0 kB/s |

Перезагружаем компьютер и видим следующий результат:

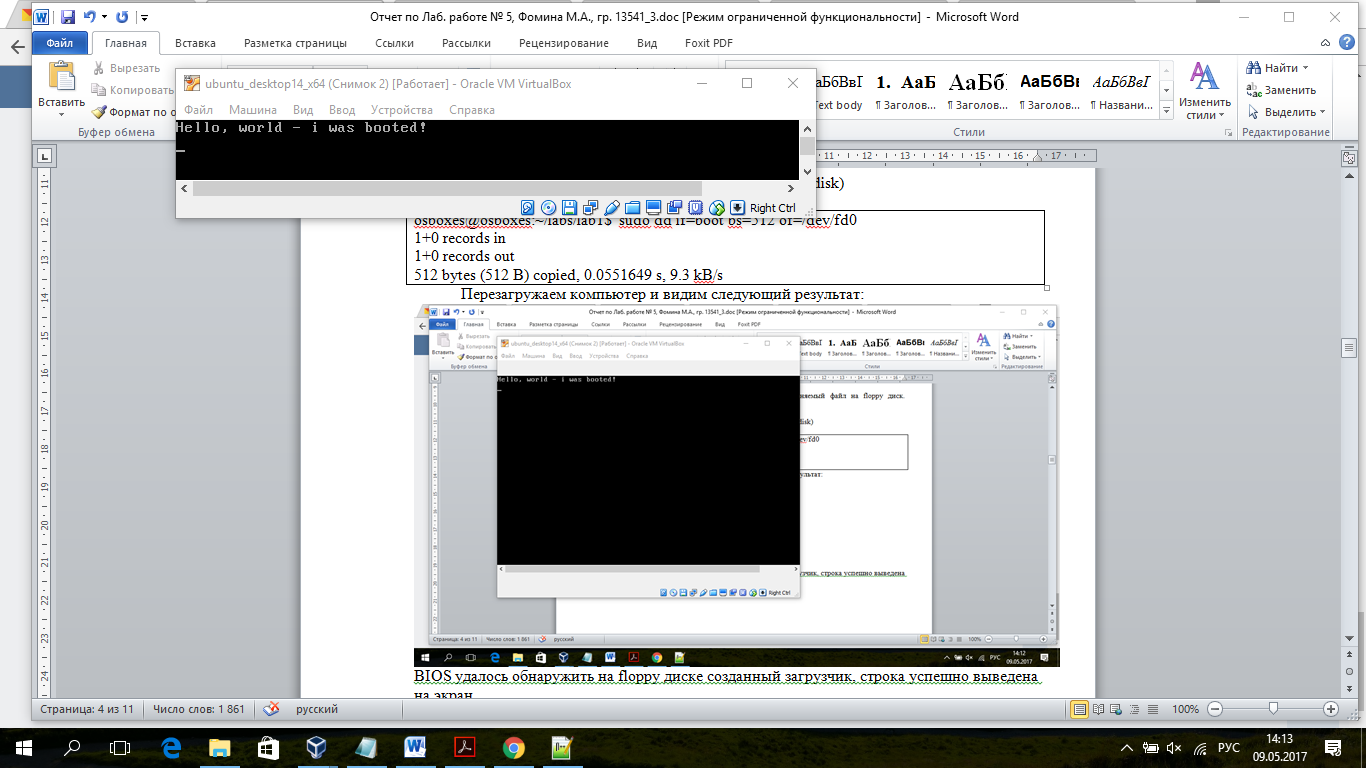


Рис. 3. Результат работы загрузчика

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

* 1. **Эмуляция загрузки с жесткого диска**

Для эмуляции жесткого диска воспользуемся программой QEMU. QEMU — свободная программа с открытым исходным кодом для эмуляции аппаратного обеспечения различных платформ. Включает в себя эмуляцию процессоров Intel x86 и устройств ввода-вывода. Была установлена программа QEMU.

|  |
| --- |
| osboxes@osboxes:~/files$ sudo apt-get update  osboxes@osboxes:~/files$ sudo apt-get install qemu |

Создается не только загружаемый образ qemu для текущей целевой архитектуры, но и образы для других архитектур, включая ARM, MIPS, PowerPC, 68k и SPARC.

QEMU предоставляет специальную команду для создания жесткого диска, которая называется qemu-img. Эта утилита создает образы различных форматов. Для quemu-img необходимо указать операцию (create для создания нового образа диска), формат (qcow для форматирования образа qemu), размер и имя образа диска. Был создан образ диска на 128 Мб.

|  |
| --- |
| osboxes@osboxes:~/files$ qemu-img create -f qcow qemu-disk.img 128M  Formatting 'qemu-disk.img', fmt=qcow size=134217728 encryption=off |

Командой dd поместим файл загрузчика на образ диска.

|  |
| --- |
| osboxes@osboxes:~/files$ sudo dd if=1/boot bs=512 of=qemu-disk.img  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.112614 s, 4.5 kB/s |

Запустим созданный образ диска, используя QEMU эмулятор.

|  |
| --- |
| osboxes@osboxes:~/files$ sudo qemu-system-i386 -hda qemu-disk.img |

Открывается новое окно, которое эмулирует работу созданной системы.

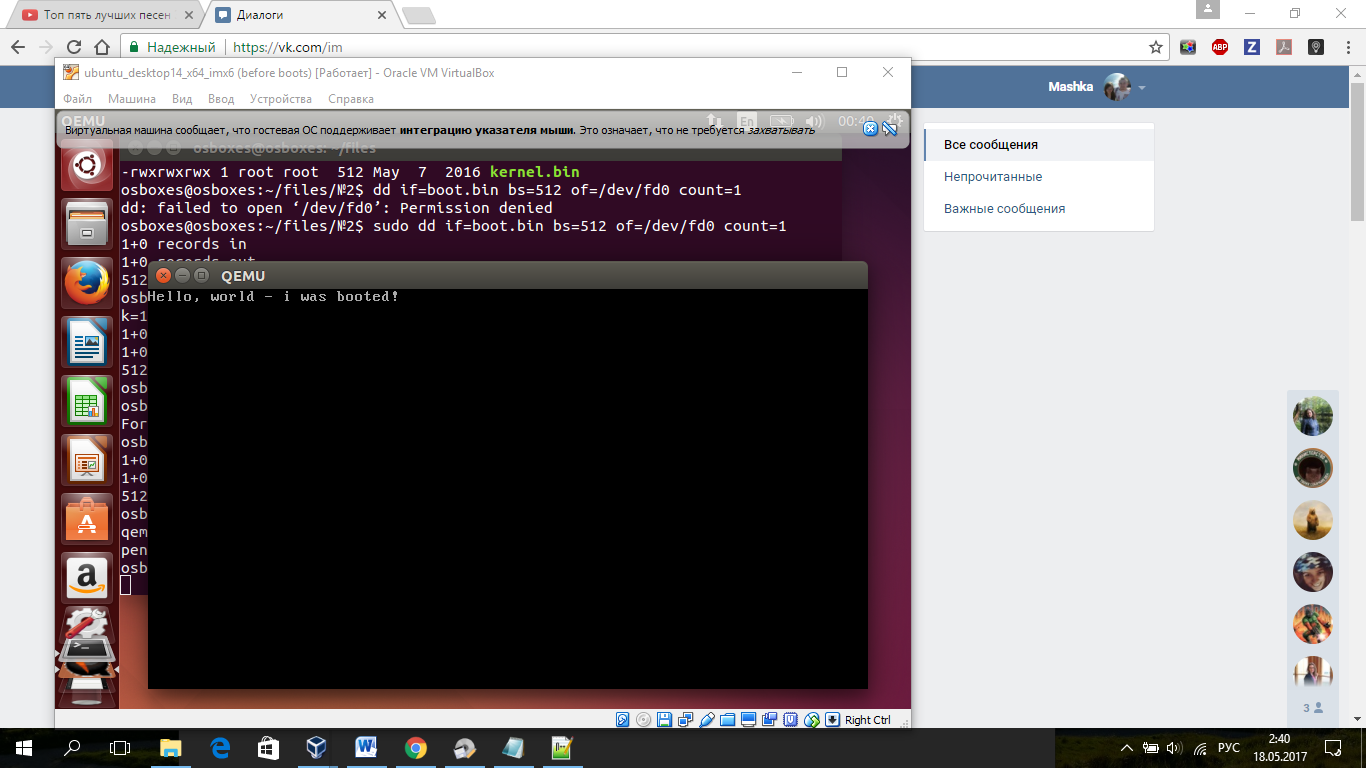


Рис. 4. Эмуляция загрузки с жёсткого диска

Видим, что загрузчик с жесткого диска успешно вывел на экран сообщение.

* 1. **Загрузка с flash-накопителя**

Полученный в результате компиляции файл boot необходимо загрузить в первый сектор флеш-накопителя. Сделаем это с помощью программы WinHex (рис. 5), предварительно сделаем резервную копию (Файл -> Создать образ диска). Открываем диск: Инструменты -> Открыть диск -> Физический носитель. Для сохранения секторов нажимаем Файл -> Сохранить сектора. После этого при запуске виртуальной машины на экране появляется наше сообщение (рис. 6).

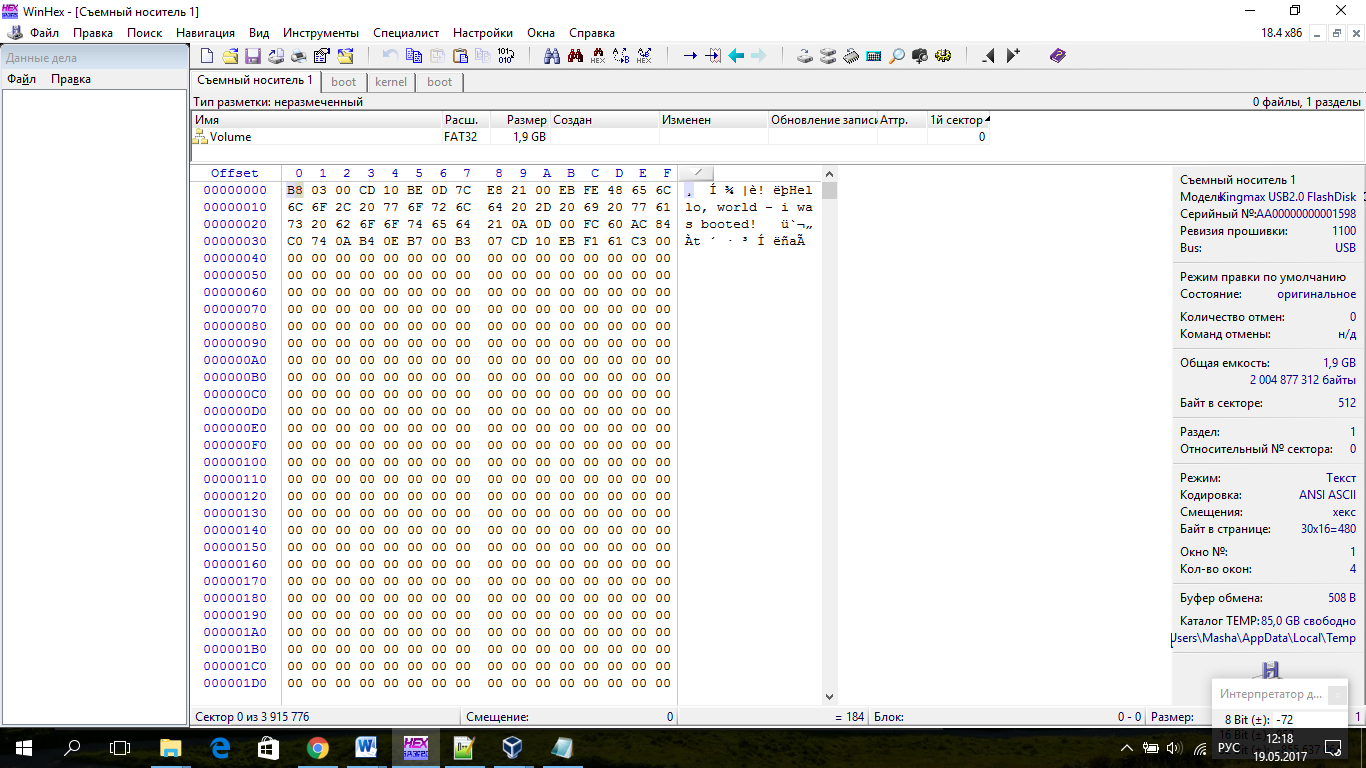


Рис. 5. Запись первичного загрузчика на флеш-носитель

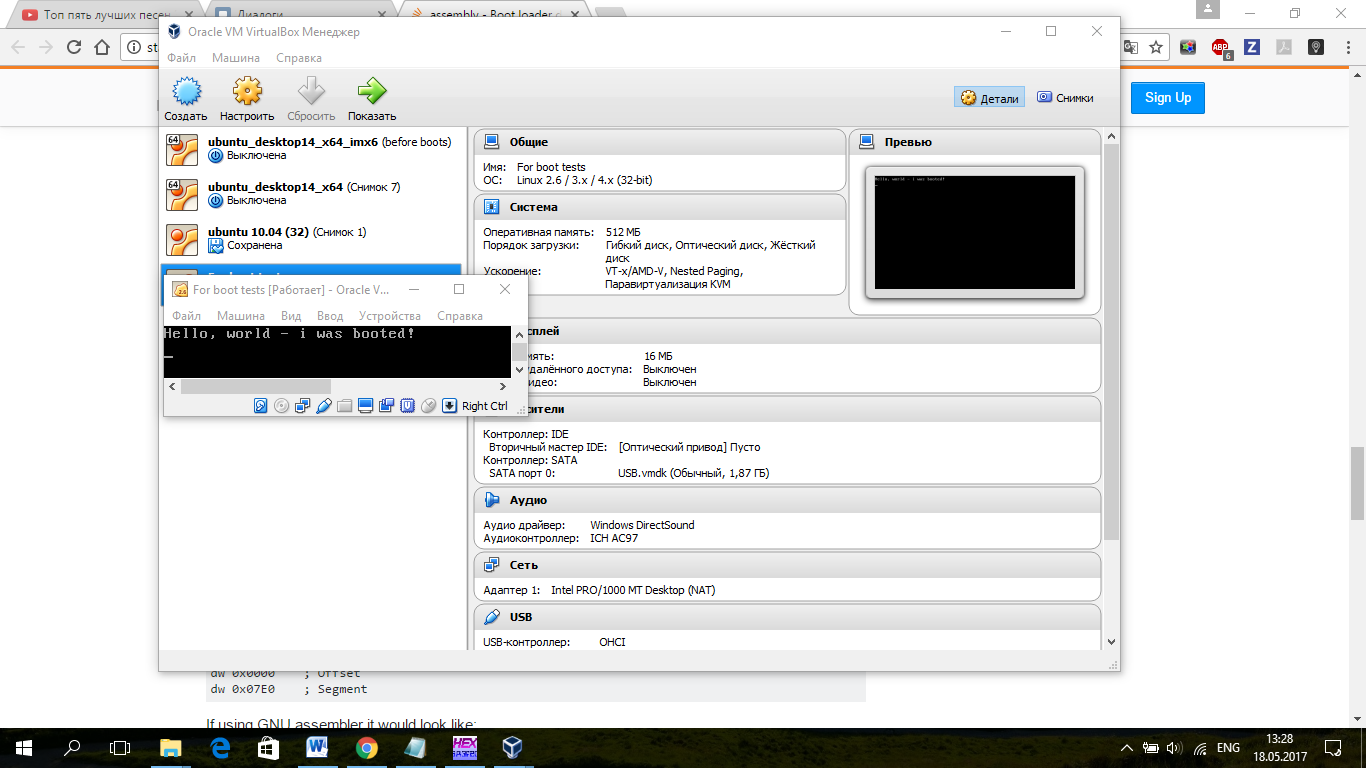


Рис. 6. Результат работы загрузчика

Первичный загрузчик, расположенный в первом секторе реального флэш-накопителя работает. Но была затёрта важная информация о FAT, содержимое секторов флэш-накопителя пришлось восстанавливать.

В самом начале сектора располагается FAT-таблица, которую нельзя повредить. Поэтому необходимо располагать загрузчик по смещению 0x5A относительно начала раздела. При копировании нужно следить, чтоб в конце сектора с загрузчиком оставалась сигнатура 0xAA55.

# Пример реализации загрузчика из 2 ступеней

# Проверка на эмулируемом жёстком диске и образе дискеты с ФС FAT

На floppy диск будет записано 2 сектора, каждый по 512 байт. Первый сектор – загрузчик, должен передать управление второму сектору. Второй сектор, назовем его kernel, просто выводит строку на экран так, как это было реализовано в предыдущем пункте.

Рассмотрим теперь реализацию первичного загрузчика для дискеты, который будет находить файл этой программы и загружать ее на выполнение.

Код boot.asm:

|  |
| --- |
| [BITS 16]  [ORG 0x7C00]  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  ReadFloppy:  MOV AH, 0x02 ;чтение сектора диска  MOV AL, 0x01 ;количество секторов для чтения  INT 0x13  ;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  TIMES 510 - ($ - $$) db 0 ;заполняет оставшееся место на секторе 0  DW 0xAA55 ;добавление специальной сигнатуры в конце загрузчика |

Для вызова следующей стадии загрузчика используется прерывание 0x13 с параметрами:

DH = 0x0 ;head (0=base)

CH = 0x0 ;track/cylinder

CL = 0x02 ; номер сектора (нумерация начинается с 1)

BX = 0x0; младший адрес RAM

ES = 0x1000 ; старший адрес RAM 0x1000

AH=0x02; чтения сектора с диска

AL=0x01; количество секторов для чтения.

Указали, что необходимо прочитать один сектор (под номером 2) из floppy диска и записать прочитанное в RAM 0x1000:0x0. После чего осуществить переход по адресу 0x1000:0x0.

Теперь необходимо написать вторую ступень загрузчика, осуществляющий вывод на экран строки. Код kernel.asm:

|  |
| --- |
| ; Установим видеорежим, очистка экрана  mov ax, 3  int 10h  ; Вывод приветственного сообщения  mov si, mHello ; сохранение указателя на строку в si  call print  ; Зависаем в бесконечном цикле  die: jmp short die  mHello db 'Hello, world - i was booted!',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  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходим к печати следующего символа  .Exit:  popa  ret ; выход из процедуры  TIMES 512 - ($ - $$) db 0 ;заполняет оставшееся место на секторе 0 |

Компилируем созданные файлы:

|  |
| --- |
| osboxes@osboxes:~/files/2$ nasm -f bin boot.asm -o boot  osboxes@osboxes:~/files/2$ nasm -f bin kernel.asm -o kernel |

Произведём эмуляцию загрузку с жёсткого диска.

|  |
| --- |
| osboxes@osboxes:~/files/2$ sudo dd if=boot bs=512 of=qemu-disk.img count=1  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.00318969 s, 161 kB/s  osboxes@osboxes:~/files/2$ sudo dd if=kernel bs=512 of=qemu-disk.img count=1 seek=1  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.0118845 s, 43.1 kB/s  osboxes@osboxes:~/files/2$ sudo qemu-system-i386 -hda qemu-disk.img |

В результате появилось окно с сообщением:

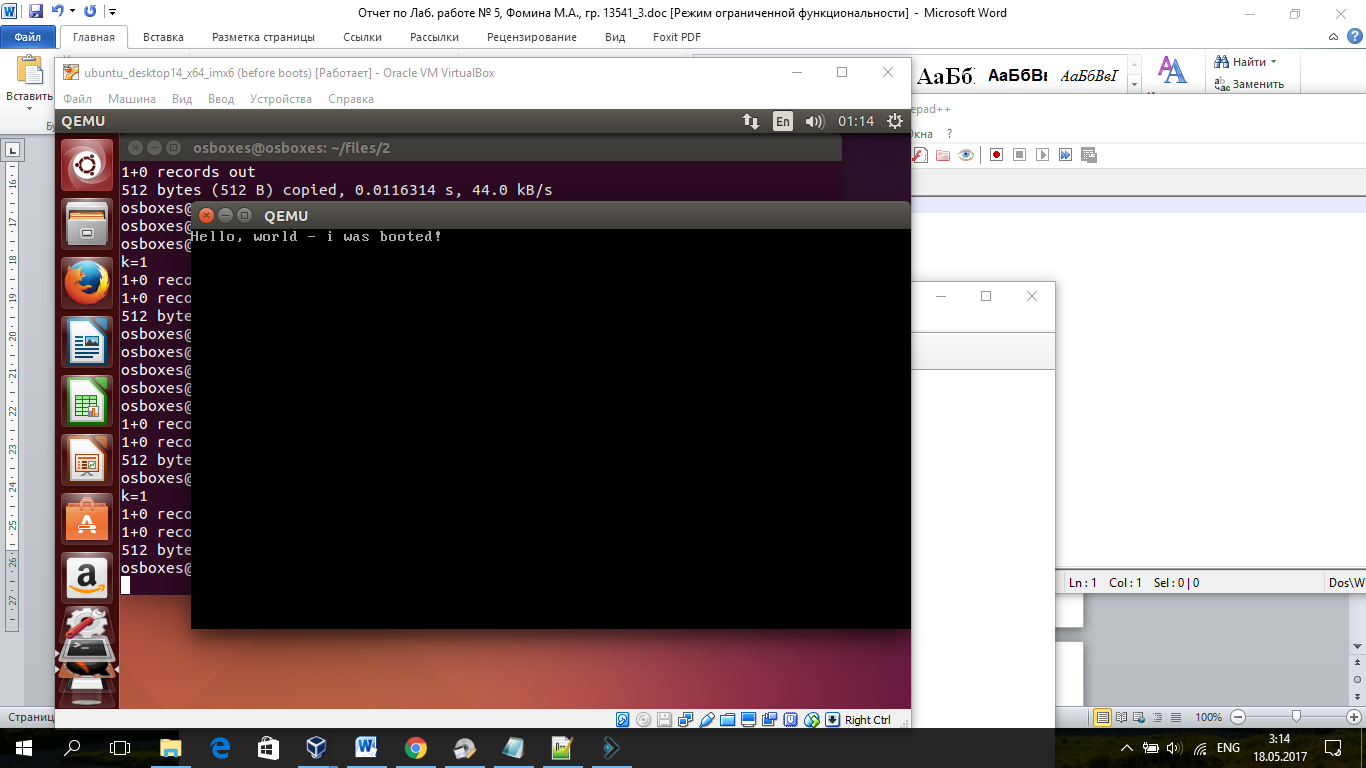


Рис. 7. Эмуляция загрузки с жёсткого диска

Поместим теперь файлы на floppy диск. Файл boot поместим в первый сегмент диска, kernel поместим во второй сегмент диска. Для этого выполним команду dd со следующими параметрами:

* if — чтение файла boot
* of — запись в файл /dev/fd0 (используемый floppy disk)
* bs — размер блока 512
* count ­­– 1, копируем 1 блок
* seek – 1, перед копированием пропускаем 1 блок от начала в выходном файле.

|  |
| --- |
| osboxes@osboxes:~/files/2$ sudo dd if=boot bs=512 of=/dev/fd0 count=1  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.0441966 s, 11.6 kB/s  osboxes@osboxes:~/files/2$ sudo dd if=kernel bs=512 of=/dev/fd0 count=1 seek=1  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.0232264 s, 22.0 kB/s |

Перезагружаем компьютер и видим следующий результат.

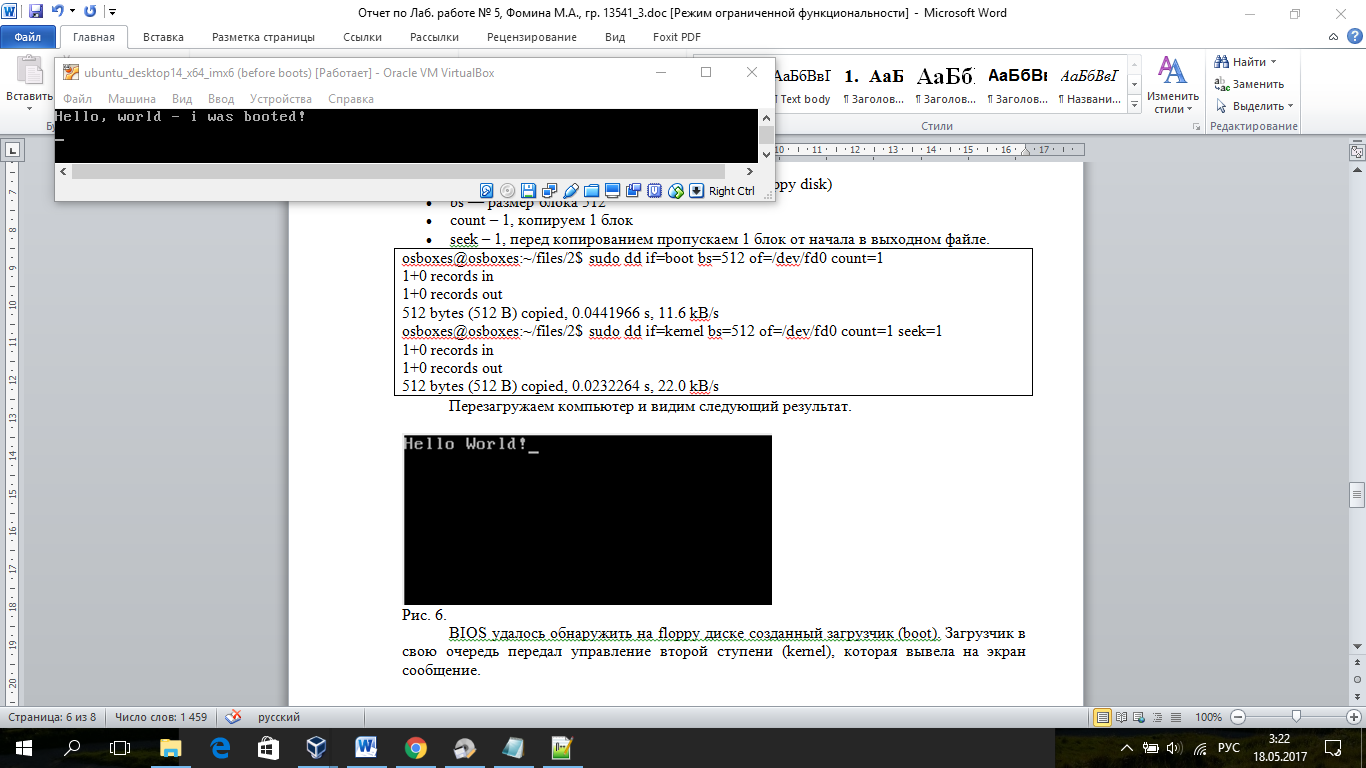


Рис. 8. Результат работы загрузчика

BIOS удалось обнаружить на floppy диске созданный загрузчик (boot). Загрузчик в свою очередь передал управление второй ступени (kernel), которая вывела на экран сообщение.

* 1. **Загрузка с flash-накопителя вторичного загрузчика**

Полученные в результате компиляции файлы необходимо поместить в первые два раздела флеш-накопителя: файл boot - в первый сектор по смещению 0x5A, файл kernel – во второй сектор. Для этого была использована программа WinHex (рис. 9, рис. 10). После сохранения секторов при запуске виртуальной машины на экране появляется сообщение (рис. 11).

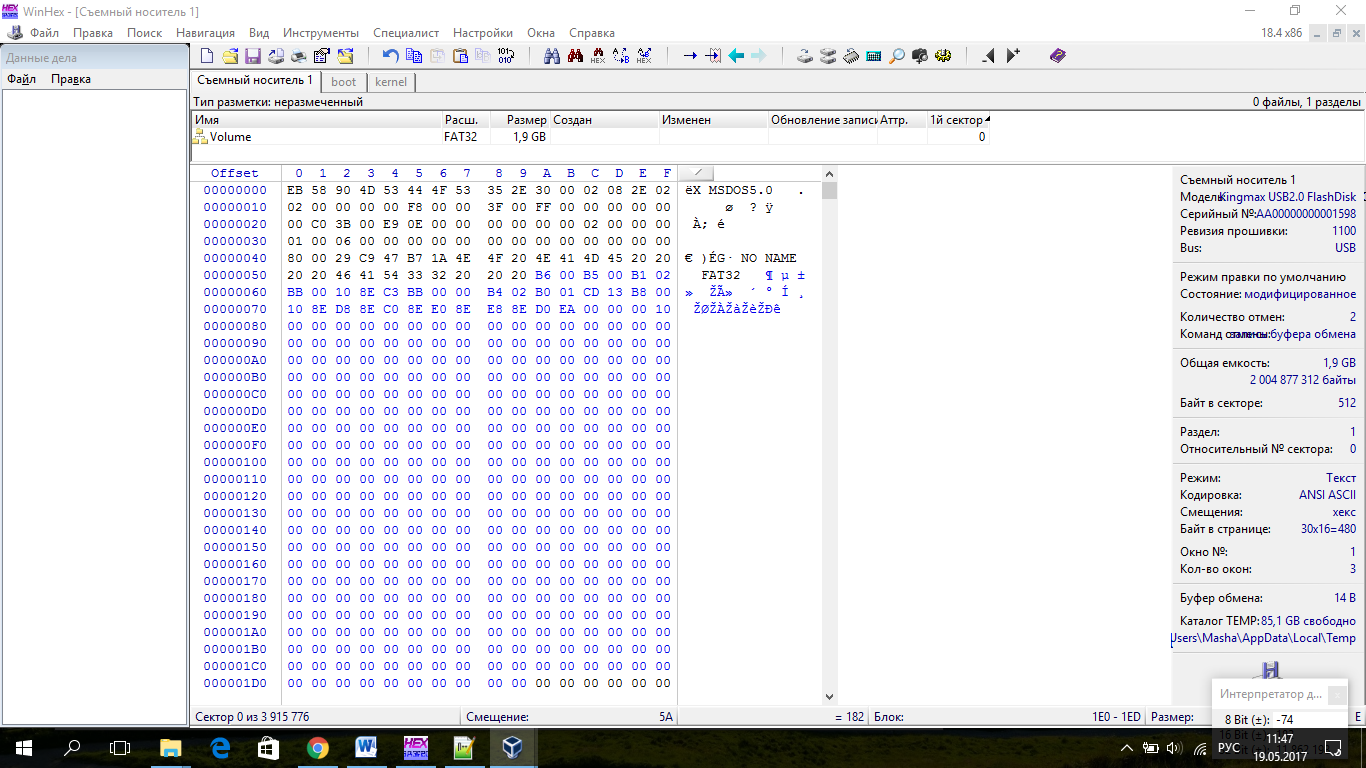


Рис. 9. Запись первичного загрузчика по смещению 0x5A на флеш-носитель

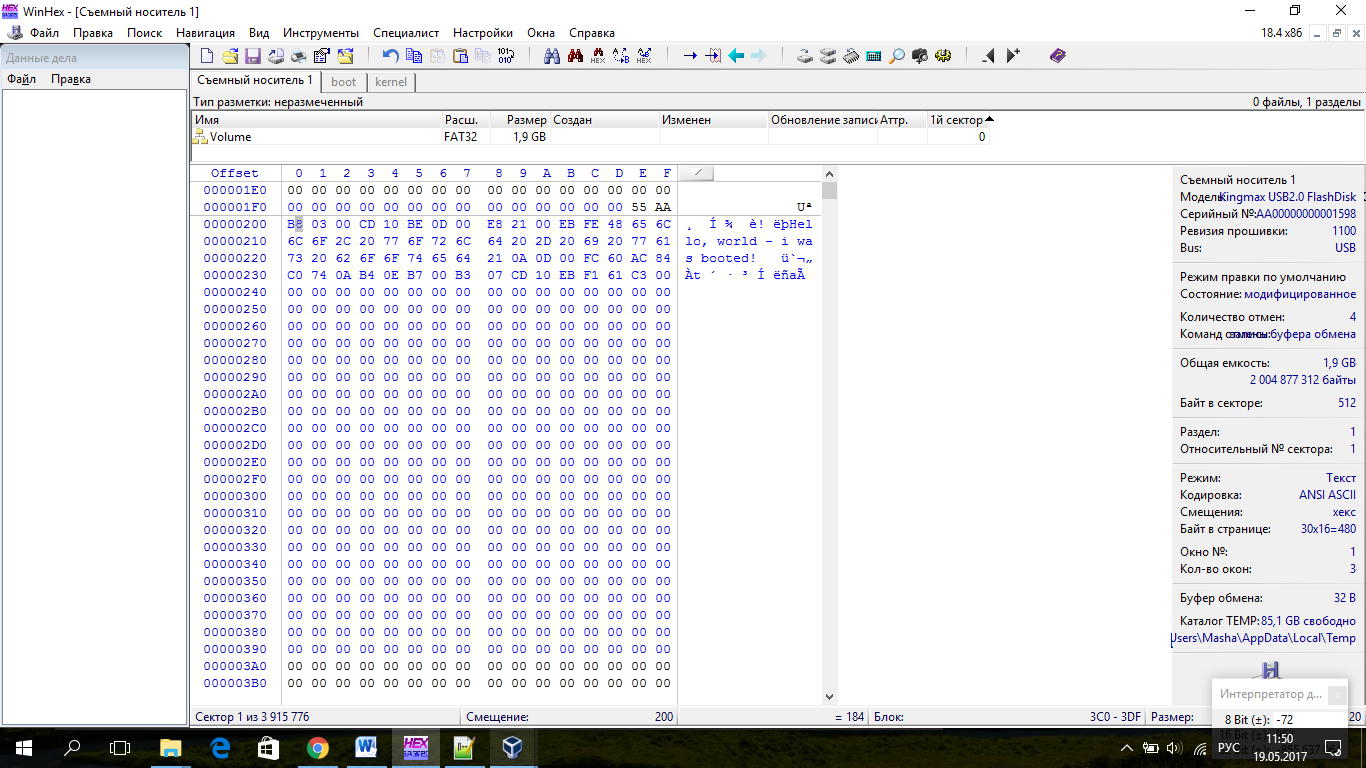


Рис. 10. Запись вторичного загрузчика во второй сектор на флеш-носитель

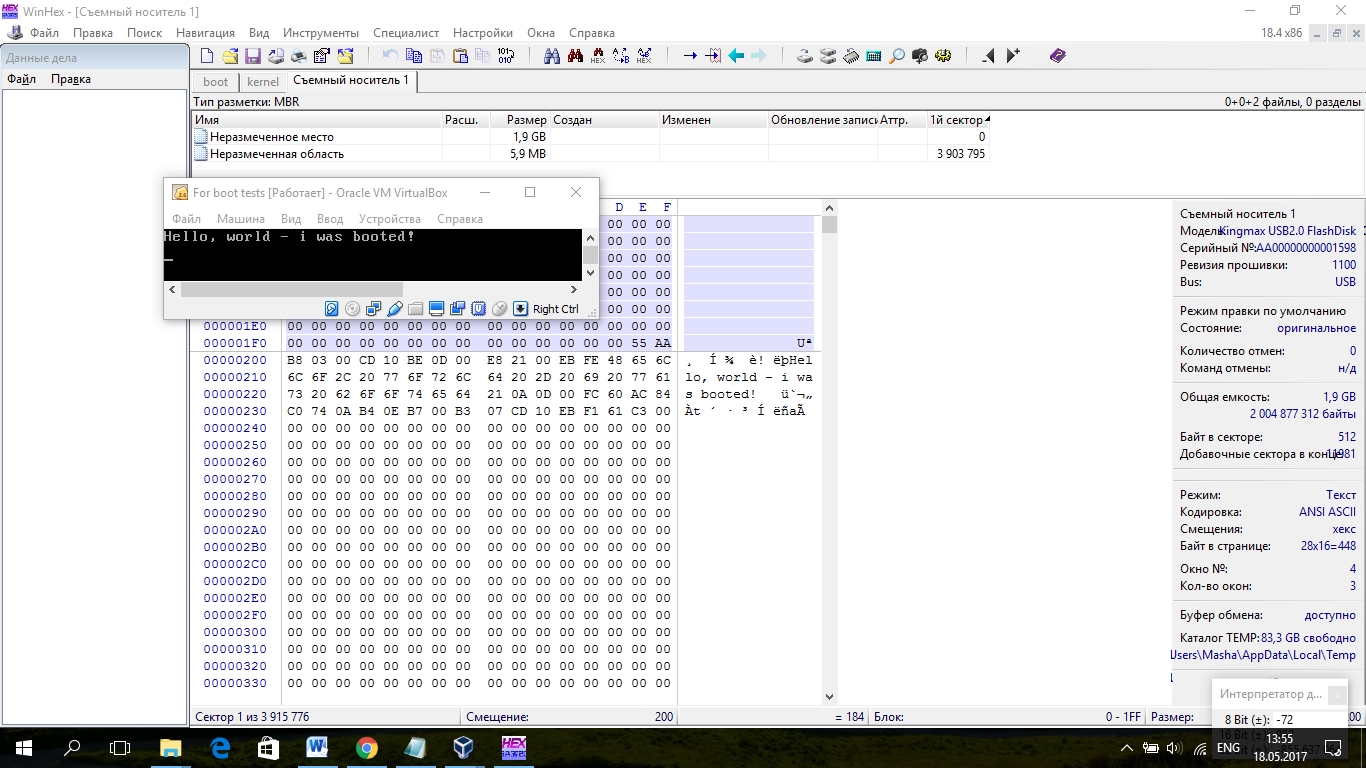


Рис. 11. Результат работы первичного и вторичного загрузчика

Первичный загрузчик, расположенный в первом секторе реального флэш-накопителя, передаёт управление вторичному загрузчику, расположенному во втором секторе, в результате на экране появляется приветственное сообщение.

# Модификация загрузчика из 2 ступеней, первичный загрузчик выводит сообщение

# Проверка на эмулируемом жёстком диске

Внесём модификацию в первичный загрузчик, чтобы он печатал сообщение, о том, что чтение закончено. Прошлый вариант работал не совсем корректно.

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 |

Указали, что необходимо прочитать один сектор (под номером 2) и записать прочитанное в RAM 0x1000:0x0. После чего осуществить переход по адресу 0x1000:0x0.

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

Компилируем файлы:

|  |
| --- |
| osboxes@osboxes:~/files/3$ nasm -f bin boot.asm -o boot  osboxes@osboxes:~/files/3$ nasm -f bin kernel.asm -o kernel |

Произведём эмуляцию загрузку с жёсткого диска.

|  |
| --- |
| osboxes@osboxes:~/files/3$ qemu-img create -f qcow qemu-disk.img 128M  Formatting 'qemu-disk.img', fmt=qcow size=134217728 encryption=off  osboxes@osboxes:~/files/3$ sudo dd if=boot bs=512 of=qemu-disk.img count=1  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.00433318 s, 118 kB/s  osboxes@osboxes:~/files/3$ sudo dd if=kernel bs=512 of=qemu-disk.img count=1 seek=1  1+0 records in  1+0 records out  512 bytes (512 B) copied, 0.0121594 s, 42.1 kB/s  osboxes@osboxes:~/files/3$ sudo qemu-system-i386 -hda qemu-disk.img |

В результате появилось окно с сообщением:

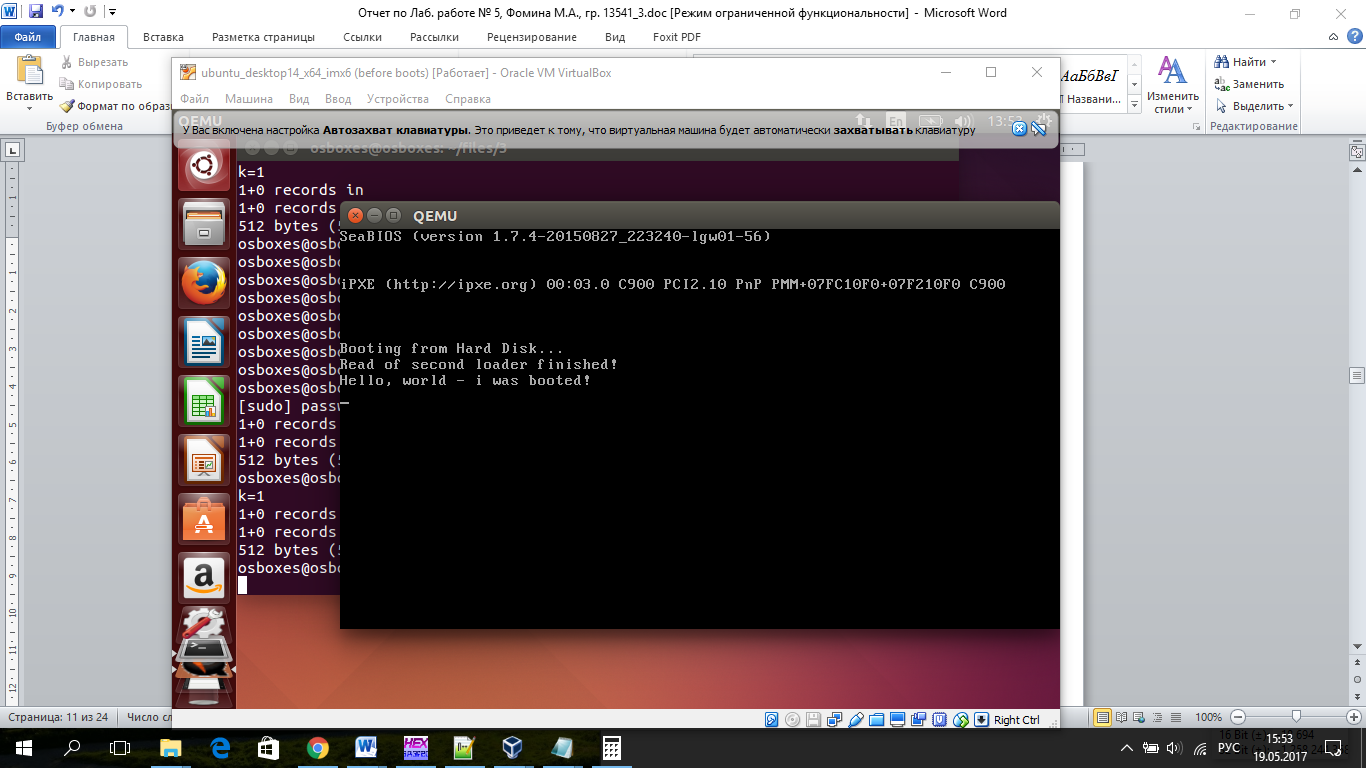


Рис. 12. Эмуляция загрузки с жёсткого диска

* 1. **Загрузка с flash-накопителя вторичного загрузчика**

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 (рис. 15).

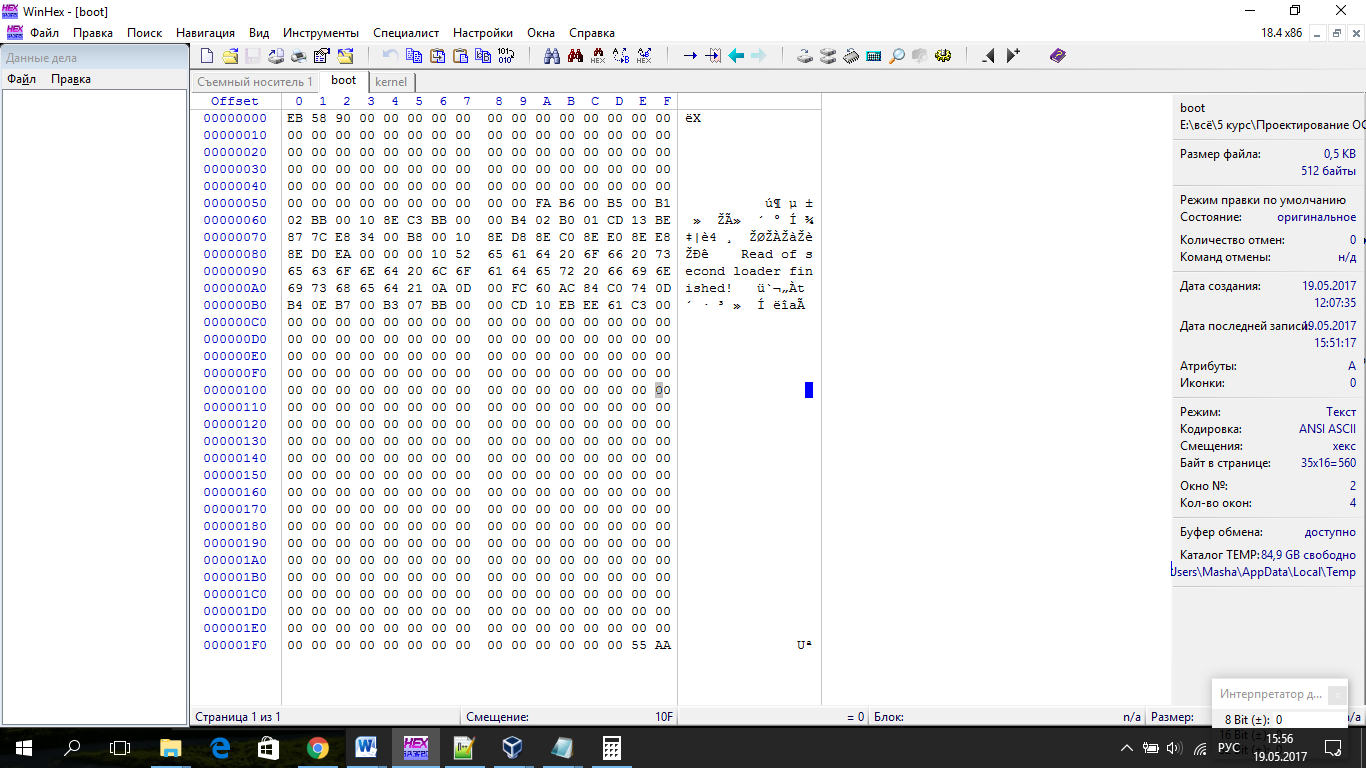


Рис. 13. Файл boot

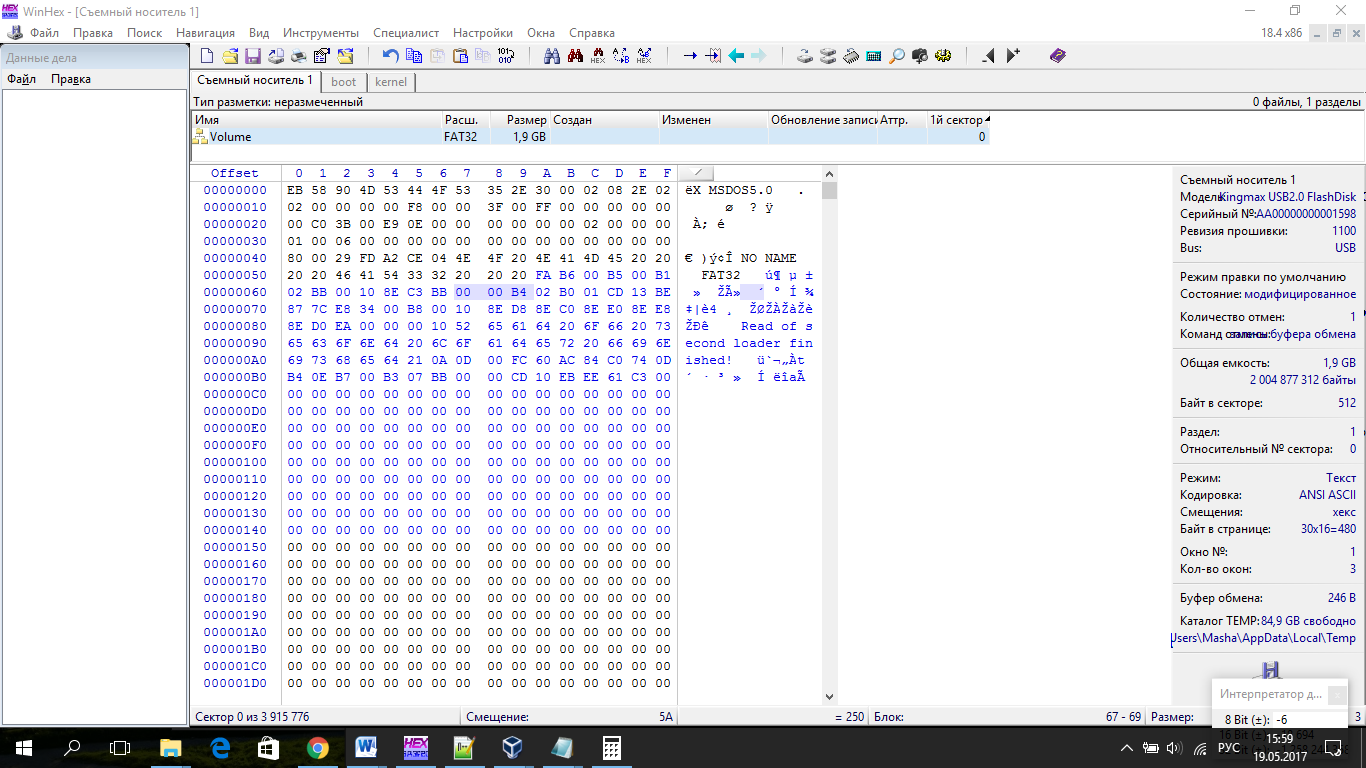


Рис. 14. Запись основной части первичного загрузчика по смещению 0x5A на флеш-носитель

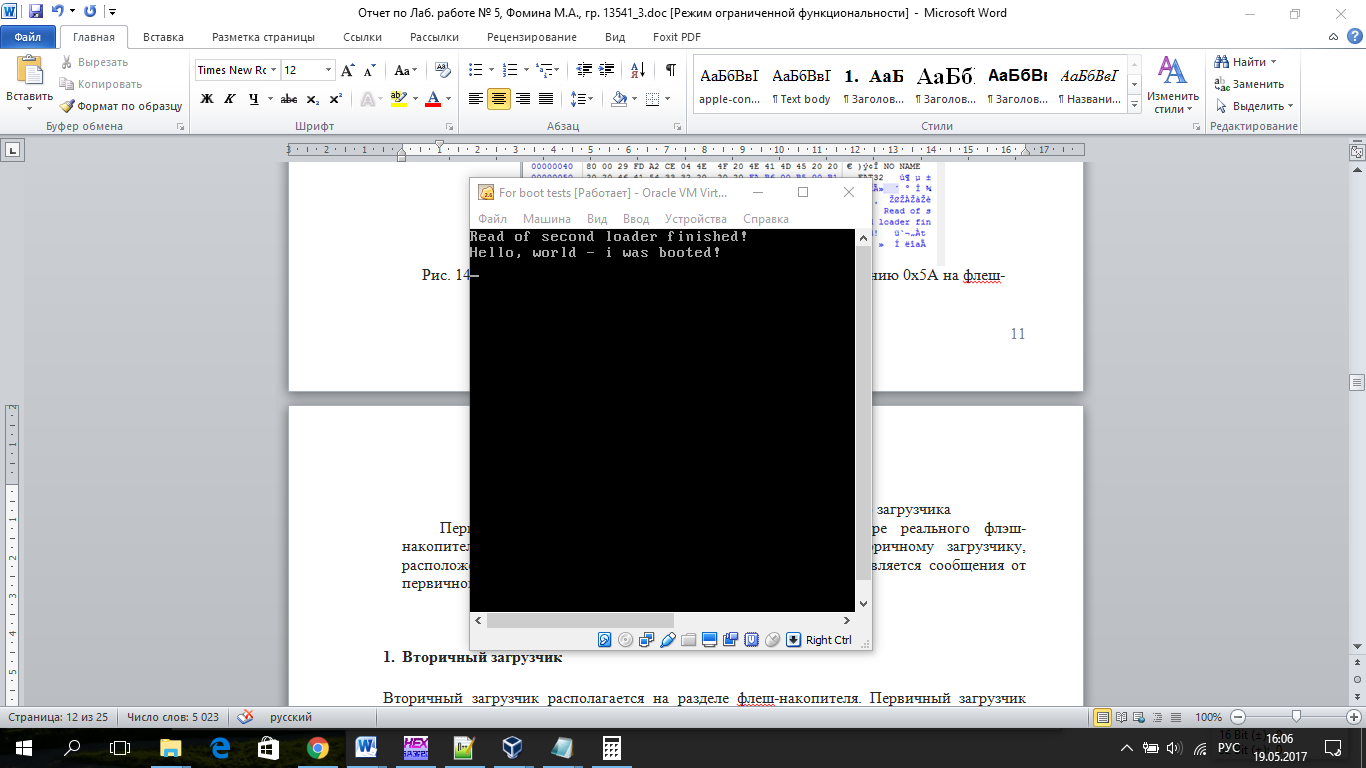


Рис. 15. Результат работы первичного и вторичного загрузчика

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.

1. **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. Программа вторичного загрузчика была построена на основе примеров ([4], [7]). 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 ; 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 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 нам 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 ; сообщаем BIOS, что выводим один символ на экран  mov bh, 0x00 ;нет страниц  mov bl, 0x07 ; выбор цвета  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходим 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  ; Зависаем 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 ; сообщаем BIOS, что выводим один символ на экран  mov bh, 0x00 ;нет страниц  mov bl, 0x07 ; выбор цвета  int 0x10 ; вызов прерывания 0x10  jmp short .PrintChar ; переходим к печати следующего символа  .Exit:  popa  ret ; выход из процедуры  TIMES 512 - ($ - $$) db 0 ;заполняет оставшееся место на секторе 0 |

Компиляция программ:

|  |
| --- |
| D:\!\13541\_3.2\ОС\Nikita\Lab5\files\4>yasm-1.3.0-win64.exe -f bin HELLO1.asm -o HELLO1.bin  D:\!\13541\_3.2\ОС\Nikita\Lab5\files\4>yasm-1.3.0-win64.exe -f bin HELLO2.asm -o HELLO2.bin  D:\!\13541\_3.2\ОС\Nikita\Lab5\files\4>yasm-1.3.0-win64.exe -f bin multiboot.asm -o multiboot  D:\!\13541\_3.2\ОС\Nikita\Lab5\files\4>yasm-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.

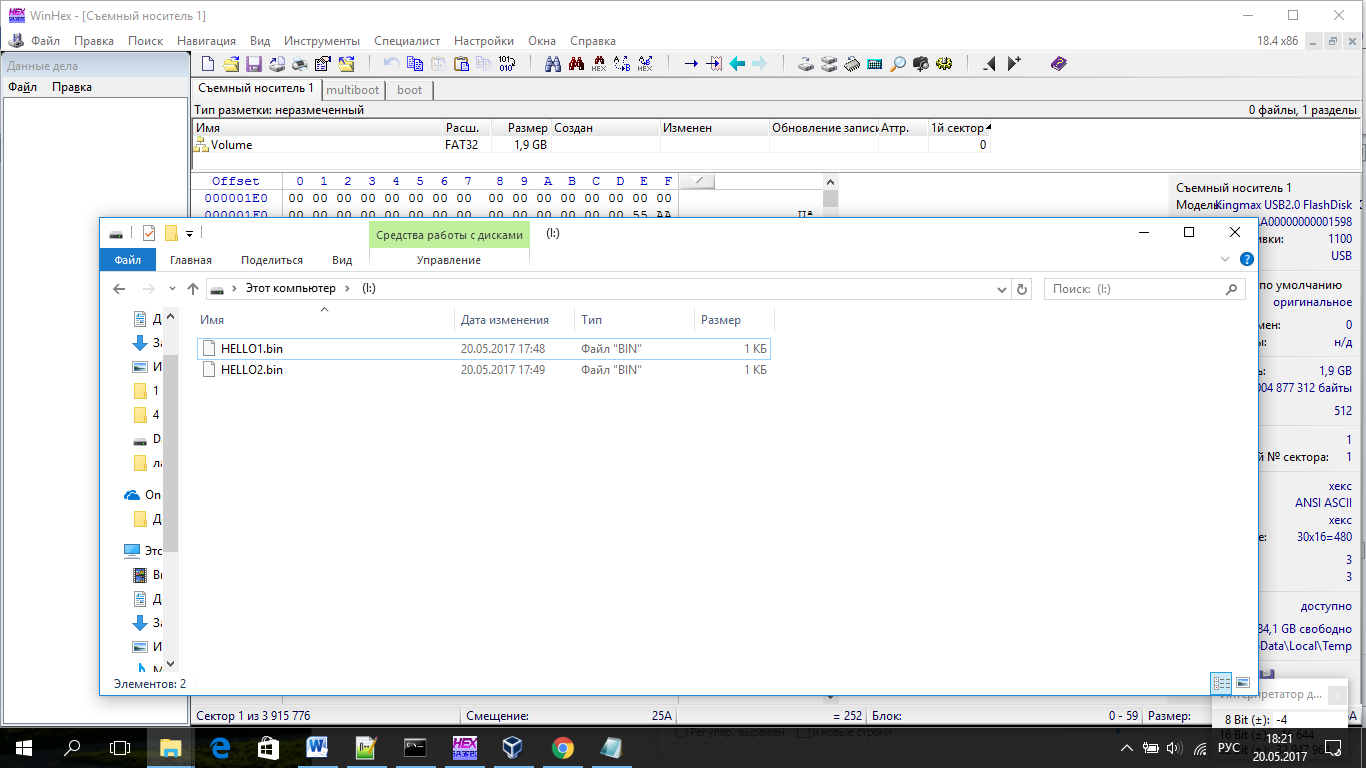


Рис. 16. На флэш-носитель записаны исполняемые программы

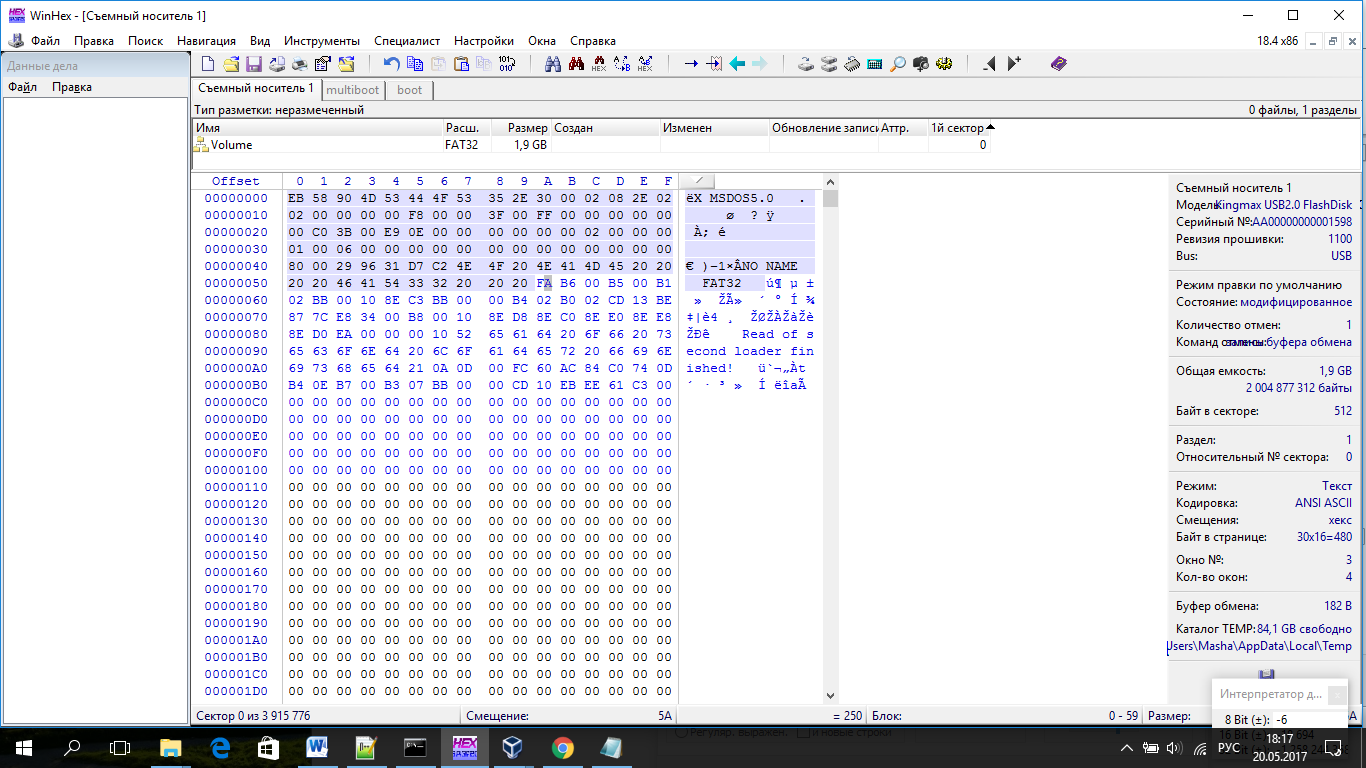


Рис. 17. Запись основной части первичного загрузчика (boot) по смещению 0x5A на флеш-носитель

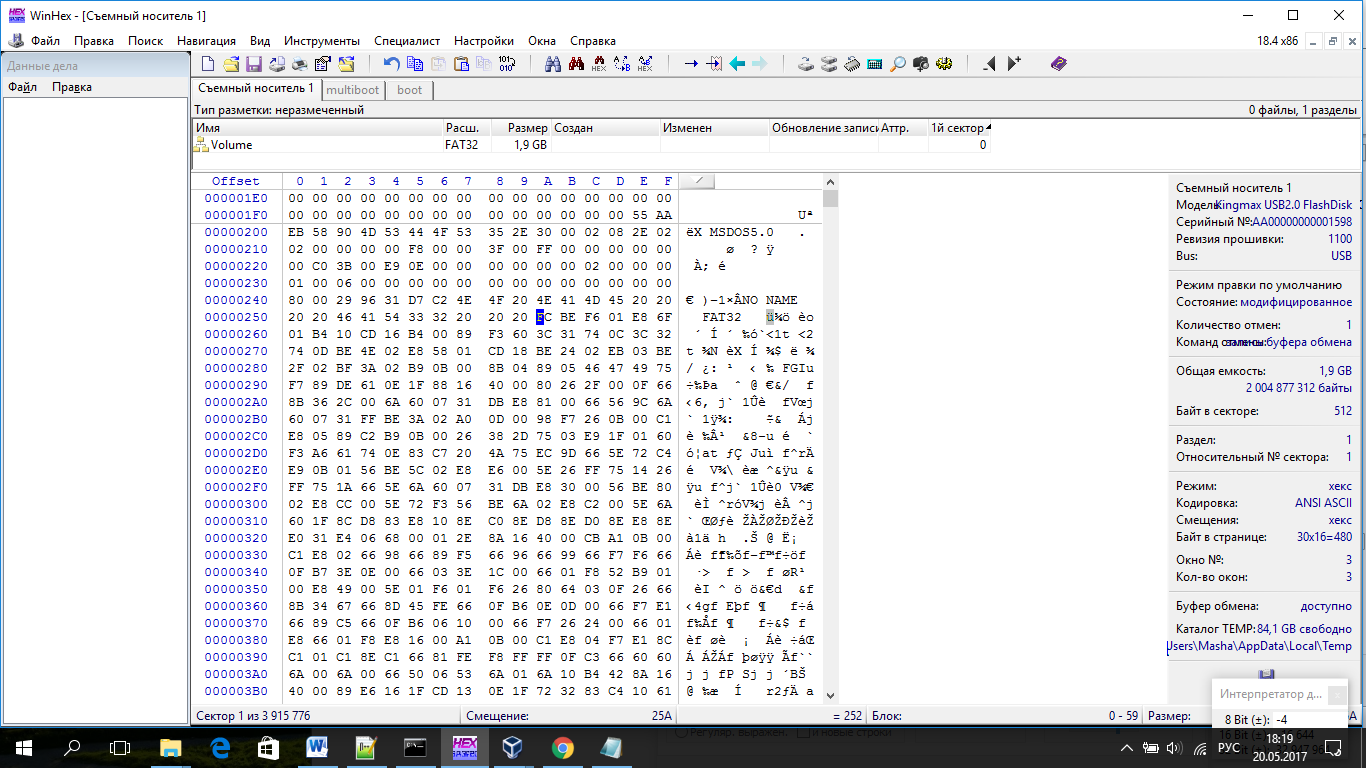


Рис. 18. Запись вторичного загрузчика (multiboot), начиная со второго сектора, по смещению 0x5A на флеш-носитель

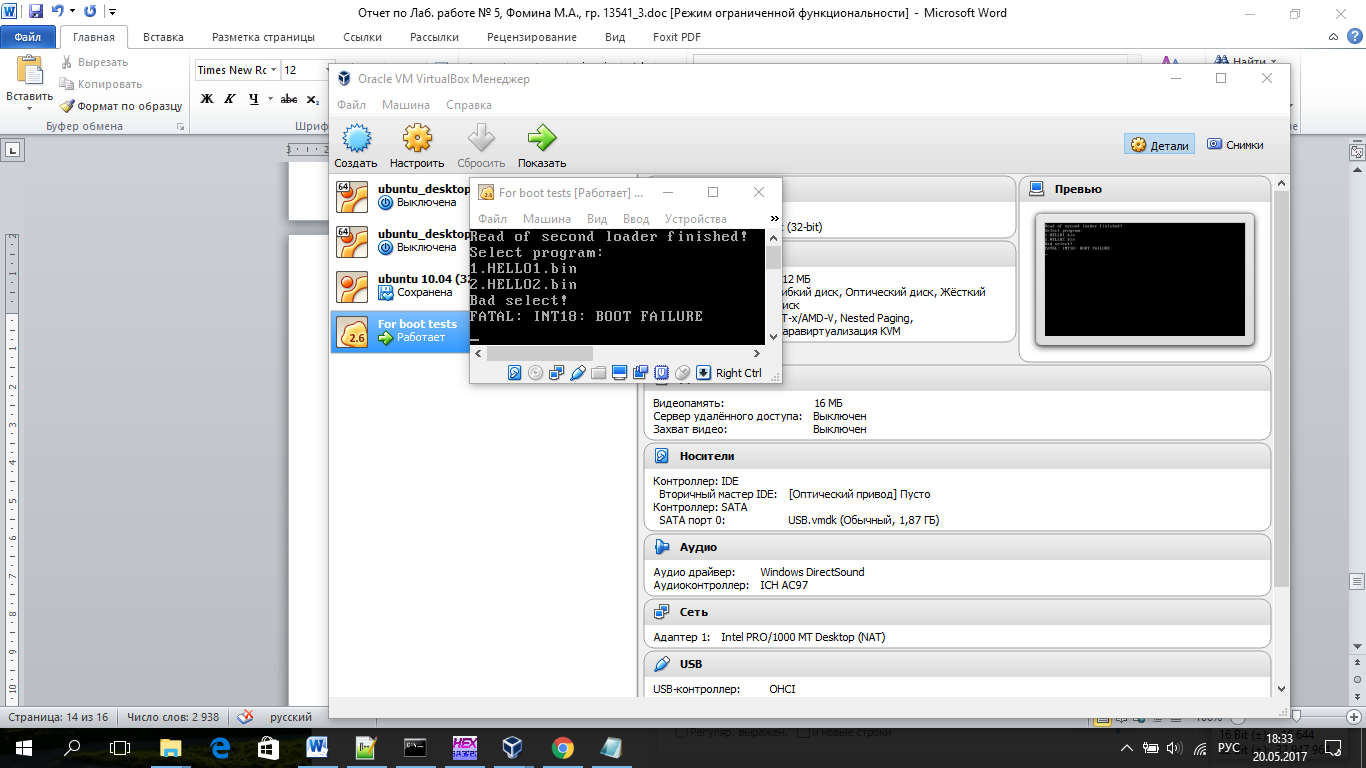


Рис. 19. Нажата клавиша отличная от 1 и 2

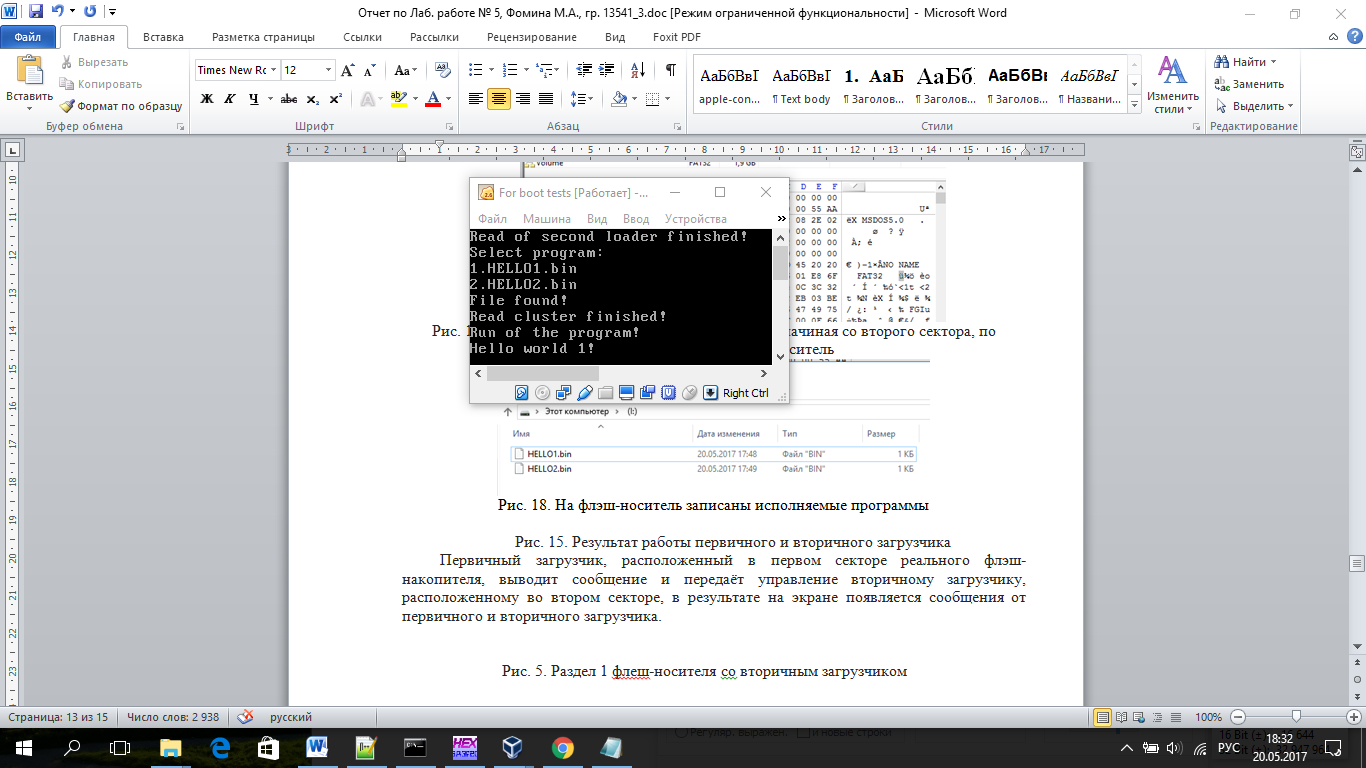


Рис. 20. При нажатии клавиши 1 исполняется первая программа

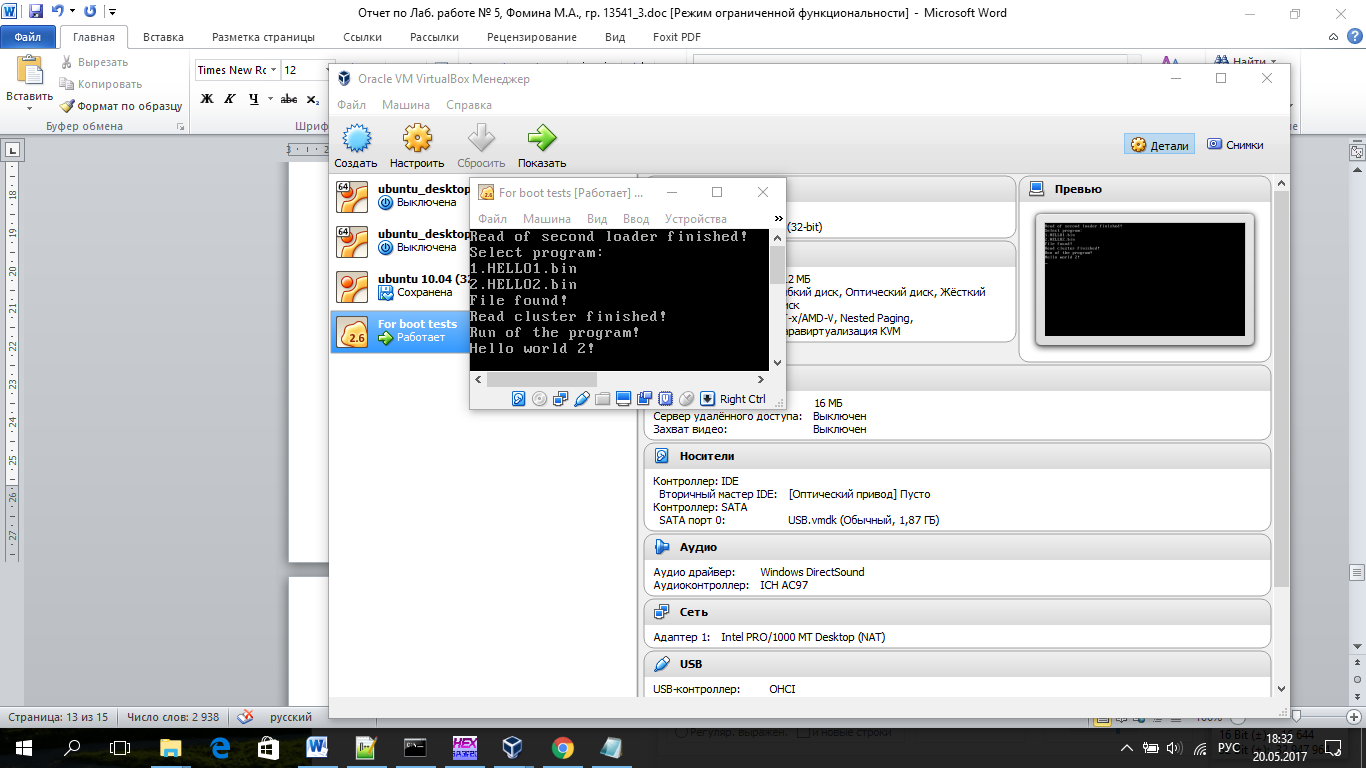


Рис. 21. При нажатии клавиши 2 исполняется вторая программа

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

**Вывод**

Проводились эксперименты с образом floppy диска, эмулируемым жёстким диском и реальным флэш-накопителем. Для эмуляции жесткого диска использовалась программа QEMU. QEMU — свободная программа с открытым исходным кодом для эмуляции аппаратного обеспечения различных платформ. 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 ФС FAT32.

**Список использованных источников**

1. <http://stackoverflow.com/questions/38502195/bootloader-kernel-only-booting-in-a-virtual-machine>
2. <http://www.qemu.org/>
3. <http://stackoverflow.com/questions/32701854/boot-loader-doesnt-jump-to-kernel-code/32705076#32705076>
4. <https://github.com/alexfru/BootProg/blob/master/boot32.asm>
5. <http://akina.hop.ru/mbr.php3>
6. <http://wiki.osdev.org/ATA_in_x86_RealMode_(BIOS)>
7. <https://github.com/sysproglab/os-projects/blob/master/loader/loader.md>