Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе**

**Дисциплина**: Проектирование ОС и компонентов

**Тема**: Загрузчик ОС

Выполнил студент гр. 13541/4 Степанов Д.С.

(подпись)

Руководитель Душутина Е.В.

(подпись)

“ ” 2017 г.

Санкт - Петербург

2017

**Цель работы**

Изучение и разработка первичного и вторичного загрузчика, с последующей загрузкой выбранного пользователем исполняемого файла. Создание загрузчика ОС. Создание загрузчика программы. Загрузка должна производиться с flash-накопителя.

**Теоретическая часть**

Загрузка операционной системы подразумевает под собой перезапись операционной системы с диска (жесткого, гибкого или flash-накопителя) в оперативную память компьютера.

Первым шагом при загрузке операционной системы является запуск BIOS (Basic Input Output System), который производит тестирование и инициализацию системы. После этого BIOS находит устройство, с которого необходимо произвести загрузку операционной системы и копирует первый сектор (512 байт) MBR с выбранного устройства в оперативную память, обычно по адресу 0x7C00 и передает управление на этот адрес.

После копирования MBR запускается первичный загрузчик операционной системы. В силу сложности современных ОС и жесткого ограничения на размер первичного загрузчика (512 байт), основная цель первичного загрузчика - передача управления вторичному загрузчику, который уже в свою очередь производит копирование и запуск операционной системы. В ходе работы, первичный загрузчик находит активный логический раздел, копирует первый сектор загрузочного раздела в оперативную память по адресу 0x07C00 и передает управление на этот адрес. После данных манипуляций, управление переходит к вторичному загрузчику операционной системы.

Вторичный загрузчик является финальным загрузчиком операционной системы. На этапе работы вторичного загрузчика производится копирование ядра операционной системы с носителя в ОЗУ, производится инициализация регистров и, после этого, управление передается ядру операционной системы. Алгоритм загрузки операционной системы представлен на рисунке 1.

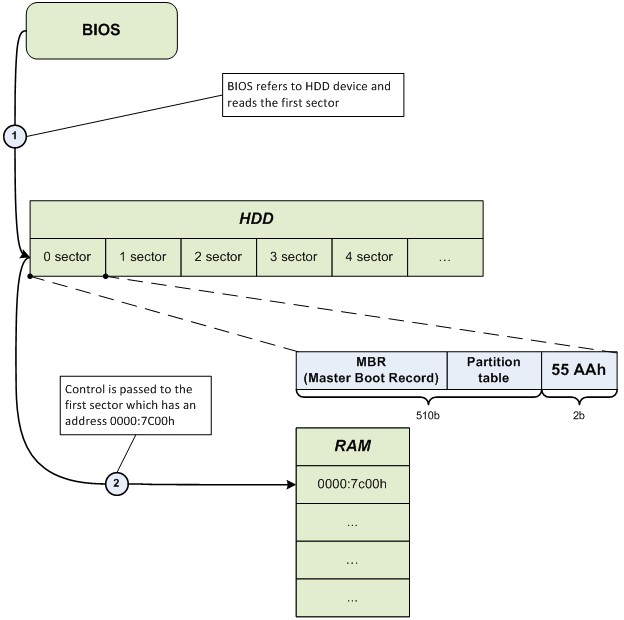


Рис. 1: Алгоритм загрузки операционной системы.

Первичный загрузчик операционной системы, как уже было сказано ранее, размещается в загрузочном (начальном) секторе накопителя. Первичный загрузчик имеет размер равный 512 байт. Данный размер первичного загрузчика обусловлен его универсальностью. Во-первых раз- личные накопители могут иметь различный размер сектора (от половины килобайта до восьми или даже больше). Кроме того, если считать один, первый, сектор диска можно всегда одним и тем же способом, то команды чтения нескольких секторов на разных устройствах могут вы- глядеть по-разному. Поэтому первичный загрузчик занимает обычно не более одного сектора в самом начале диска, в его загрузочном секторе. Задача первичного загрузчика - определить, где на диске находится вторичный загрузчик, загрузить и запустить его. Вторичный загрузчик мож- но положить в заранее определенное место диска, или положить в заранее определенное место карту размещения, описывающую, где именно искать его части (размер вторичного загрузчика ограничен, поэтому построить такую карту возможно).

MBR (Master Boot Record) - запись, которая загружается в память с накопителя и обеспечи-вает опознание логических разделов на нем, определяет активный раздел и загружает из него загрузочную запись (Boot Record - BR), которая продолжит запуск операционной системы (ОС). MBR появился на жестких дисках начиная с MS DOS версии 3.0, в более ранних версиях жёсткий диск форматировался как дискета, и в первом секторе располагался BR. Соответственно диск представлял из себя один раздел и не мог быть разбит на логические части - правда, при тех размерах дисков, которые тогда выпускались, это было неактуально.

В блоке MBR обязательно должна находится сигнатура - специальная, строго установленная, последовательность из 2 байт с шестнадцатеричными значениями 55h AAh, которая записывает- ся в последние 2 байта сектора и соответственно имеет смещение от начала сектора 1FEh. Если хотя бы один из двух последних байтов отличается по значению, считается, что первый сектор не является MBR и не содержит осмысленной информации. Если компьютер при старте, про- читав первый сектор, не обнаружит правильной сигнатуры, он не будет передавать управление располагающемуся там коду, даже если он правильный, а выдаст сообщение о том, что главная загрузочная запись не найдена. Или будет пробовать найти её на других устройствах - например, на дискете.

**Выполнение работы**

Для проведения экспериментов было решено использовать гипервизор, что бы не повредить данные на реальной машине. В качестве гипервизора был выбран VirtualBox, так как он бесплатен, и , во-вторых, потому что в VirtualBox можно эмулировать загрузку виртуальной машины с flash-накопителя, как с жесткого диска. Для этого необходимо создать файл, связующий VirtualBox и флэш-накопитель, при помощи утилиты VBoxManage. После чего появится файл, который можно подключить в VirtualBox, как жесткий диск. Для тестирования была создана пустая виртуальная машина, к которой в качестве виртуального жесткого диска был подключен flash-накопитель при помощи способа, описанного выше.

Для подготовки и проверки рабочей среды, а так же для понимания основных принципов загрузки, создадим первичный загрузчик, выводящий на экран приветственное сообщение:

org 0x7C00

bits 16

; Очищаем регистры

xor ax, ax

mov es, ax

mov ds, ax

mov ss, ax

mov sp, 0x1000

; Установим видеорежим, очистка экрана

mov ax, 3

int 10h

mov si, HelloString2 ;Записываем в стек указатель на строку

call PrintString ;Вызов процедуры печати строки

jmp $ ;Бесконечный цикл

PrintCharacter: ;Процедура печати символа

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

ret ;Возращение к вызванной процедуре

PrintString: ;Процедура печати строки

next\_character:

mov al, [si] ;Берем один байт из строки и записываем его в регистр AL

inc si ;Увеличиваем указатель SI

or al, al ;Проверка конца строки

jz exit\_function

call PrintCharacter ;Печатаем символ

jmp next\_character

exit\_function:

ret

Скомилированную программу с помощью HEX-редактора необходимо поместить в начальный сектор флеш-накопителя по адресу 0.

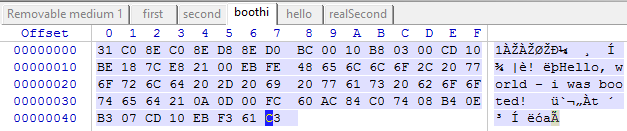


Рис. 1. Простейший первичный загрузчик в HEX-редакторе

После чего подсоединим флэш-накопитель к виртуальной машине и запустим её (рис. 2).

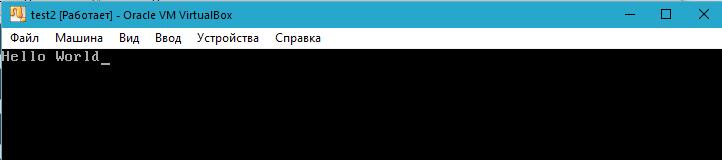


Рис.2. Работа простейшего первичного загрузчика

Ввиду того, что первичный загрузчик будет копировать вторичный из первого сектора, необходимо узнать смещение данного сектора из MBR раздела (рис. 3.)



Рис. 3. Смещение первого раздела флэш-накопителя

Работа первичного загрузчика будет производиться по следующему алгоритму:

1. Инициализация регистров, стека;

2. Копирование загрузочного сектора по новому адресу (0x100000) и передача ему управления;

3. Вывод сообщения (Copying finished), сигнализирующего об успешной загрузке по новому адресу;

4. Загрузка вторичного загрузчика по адресу 0x7C00 и передача управления по адресу 0x7C00.

Тестовый вторичный загрузчик необходим для тестирования работоспособности первичного загрузчика. В задачи тестового вторичного загрузчика входит загрузка по заданному адресу и вывод сообщения (second bootloader hello), сигнализирующего о том, что вторичный загрузчик загрузился.

Вторичный загрузчик будет располагаться в первом секторе накопителя. При загрузке, важно не повредить fat-таблицу.

Первичный загрузчик:

org 0x7C00

bits 16

;Начало работы первичного загрузчика

cli ; запрещение аппаратных прерываний

mov ax,cs ; сегмент кода

mov ds,ax ; сегмент данных

mov ss,ax ; сегмент стека

mov es,ax ; дополнительный сегмент данных

mov sp,7c00h ; Инициализация стека

mov bx,7c00h ; Адрес загрузки

;Копируем себя по адресу 0x1000

mov si,7c00h ; Копируем, начиная с адреса загрузки

mov di,1000h ; Адрес, куда производим копирование

mov cx,200h ; Сколько байт копируем (512 байт)

rep movsb

jmp 0000h:101fh ; Передаем управление по новому адресу

mov si, TestString ;Записываем в стек указатель на строку

call PrintString ;Вызов процедуры печати строки

PrintCharacter: ;Процедура печати символа

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

ret ;Возращение к вызванной процедуре

PrintString: ;Процедура печати строки

next\_character:

mov al, [si] ;Берем один байт из строки и записываем его в регистр AL

inc si ;Увеличиваем указатель SI

or al, al ;Проверка конца строки

jz exit\_function

call PrintCharacter ;Печатаем символ

jmp next\_character

exit\_function:

; загрузка загрузочной записи активного раздела

mov si, DAP

mov ah, 0x42 ; функция

mov dl, 0x80 ; номер диска

int 0x13

jmp 0000h:7c00h ; переходим ко вторичному загрузчику

DAP:

dw 0x10, 0x02, 0x7c00, 0x00 ; Считываем 2 сектора и помещаем в оперативную память по адресу 0x7c00

dd 0x0800 ;абсолютный адрес первого сектора

dd 0x0000

TestString db 'Copying finished', 13, 10, 0

times 510 - ($ - $$) db 0 ;Заполняем оставшиеся байты нулями

dw 0xAA55 ;Добавляем сигнатуру в конец загрузчика

На рисунке 4 представлен новый первичный загрузчик, загруженный в начальный сектор накопителя, а на рисунке 5 приведен результат работы.

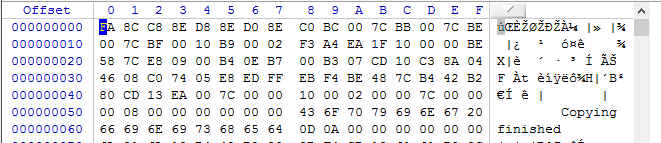


Рис. 4. Первичный загрузчик в HEX-редакторе

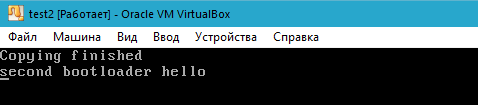


Рис. 5. Тестирование первичного загрузчика

Ввиду того, что тестовый вторичный загрузчик загрузился нормально, можно сделать вывод о том, что первичный загрузчик отрабатывает корректно. Следовательно, можно приступать к разработке полноценного вторичного загрузчика. Общий алгоритм работы вторичного загрузчика:

1. Отключаем прерывания (строка 10).
2. Выводим пользователю сообщение со списком возможных для загрузки исполняемых фай- лов (строки 17-19).
3. Сохраняем указатель на стек в регистре bx (строка 20) и анализируем введенный пользователем символ (строки 23-26). Если пользователь выбрал одно из ядер, то переходим к его загрузке (строки 35-41), иначе выводим сообщение об ошибке и завершаем работу (строки 30-32).
4. Копируем имя выбранной пользователем программы в область памяти, где будет располагаться имя исполняемого файла ,которую будем искать на накопителе (строки 44-52) и восстанавливаем указатель на стек (строка 53).
5. Сохраняем номер загрузочного диска BIOS, максируем значение номера кластера и записываем полученное значение в регистр esi (строки 57-59).
6. Начинаем цикл поиска файла в корневой директории (строки 62 - 107), внутри которого пытаемся найти заданный файл в корневой директории. Если файл найден, то переходим к его загрузке (строка 110), в противном случае - выводи сообщение об ошибке и завершаем работу (строка 107).
   1. Записываем имя файла для поиска в стек (строка 75) и запускаем функцию поиска получения адреса кластера ReadCluster (строка 67).
   2. Пытаемся найти запись о заданном файле в данном кластере (строки 97-99). Если файл найден, то записываем его адрес в si и переходим к загрузке (строки 111-114). В противном случае повторяем итерацию для следующего сектора (переход к шагу b) (строки 102-104).

Для тестирования работы вторичного загрузчика было создано два тестовых ядра программы: BOOT1.BIN и BOOT2.bin. Данные ядра после загрузки выводят сообщение о том, что они успешно загрузились и завершают работу.

Вторичный загрузчик:

org 0x7C00

bits 16

jmp short start ; Переход к исполняемому коду

nop

%include "fatTable.asm" ; Подключение таблицы с FAT-таблицей

start:

cld ; Отключение прерываний

; Печать пользователю сообщения с выбором ядра

mov si, SelectMessage ;Записываем в стек указатель на строку

call PrintString ;Вызов процедуры печати строки

; Считывание клавиши

mov ah, 10h

int 16h

mov ah, 0

mov bx, si ; сохраняем указатель на стек в bx

pusha

; Сравниваем считанный символ

cmp al,'1'

je select\_first

cmp al,'2'

je select\_second

; В противном случае выводим сообщение об ошибке

mov si, SelectErrorMessage ;Записываем в стек указатель на строку с сообщением об ошибке

call PrintString ; Выводим строку

int 18h ; Прерывание, сигнализирующее о неудачной загрузке

; Если выбран первый исполняемый файл

select\_first:

mov si, ProgramName1 ; переносим указатель стека на строку с именем первого файла

jmp write\_name ; переходим к копированию имени

; Если выбран второй исполняемый файл

select\_second:

mov si, ProgramName2 ; переносим указатель стека на строку с именем второго файла

write\_name:

mov di, ProgramName ; Записываем в di адрес, куда будет записано имя запускаемой программы

mov cx, 11 ; записываем в cx длинну имени

write\_char:

mov ax, [si] ; берем первый символ из имени программы

mov [di], ax ; записываем ее в область памяти для запускаемой программы

inc si ; увеличиваем si

inc di ; и di для перехода к следующему символу

dec cx ; уменьшаем счетчик

jnz write\_char ; если не ноль, то переходим к копированию следующего символа

mov si, bx ; восстанавливаем указатель на стек

popa

push cs

pop ds

mov [var\_logical\_drive\_number], dl ; сохранение номера загрузочного диска BIOS

and byte [var\_cluster\_number+3], 0Fh ; маскировка значения кластера

mov esi, [var\_cluster\_number] ; Записываем в esi номер кластера корневой директории

; Чтение корневой директории

RootDirReadContinue:

push 60h ; 60h - смещение загрузочного образа BIOS

pop es ; Достаем значение из стека и помещаем его в es

xor bx, bx ; Обнуление bx

call ReadCluster ; Вызов функции чтения одного кластера из корневой директории

push esi ; сохранение номера следующего кластера в esi

pushf ; Записываем флаг в стек

; Поиск и запуск исполняемого файла

push 60h

pop es ; Достаем значение из стека и помещаем его в es

xor di, di ; Обнуляем di

mov si, ProgramName ; Записываем в стек имя программы для запуска

mov al, [var\_sector\_by\_claster] ; Записываем в al количество секторов в кластере

cbw ; Преобразуем содержимое al в знаковое слово (номер) в ax

mul word [var\_byte\_per\_sector]; Умножаем количество секторов в кластере на количество байт в секторе и получаем количество байт в кластере

shr ax, 5 ; Сдвиг вправо регистра ax на 5 бит и получаем количество записей каталога

mov dx, ax ; Записываем содержимое в регистр dx

; Поиск имени файла

; В стек помещено имя файла

; dx - количество точек входа

; в результате в esi будет записан номер кластера

FindName:

mov cx, 11 ; Записываем в cx длину имени (11 байт)

; Цикл поиска файла

FindNameCycle:

cmp byte [es:di], ch ; Сравниваем байты текущей записи с тем, что нужно найти

jne FindNameNotEnd ; Если не равны, то продолжаем искать

jmp ErrFind ; Если конец корневой директории (найдена NULL запись) то выводим сообщение об ошибке

; Продолжение поиска файла

FindNameNotEnd:

pusha ; сохраняем в стеке содержимое регистров AX, CX, DX, BX, SP, BP, SI, DI

repe cmpsb ; Сравниваем строки

popa ; Восстанавливаем содержимое регистров AX, CX, DX, BX, SP, BP, SI, DI

je FindNameFound ; Если равны, то файл найден и переходим по метке FindNameFound

add di, 32 ; Прибавляем к di 32 (переход к следующей записи)

dec dx ; Уменьшаем счетчик

jnz FindNameCycle ; Повторяем итерацию для новой записи

popf ; Восстанавливаем флаги

pop esi ; В esi записывается адрес следующего кластера корневой директории

jc RootDirReadContinue ; Если остались кластеры, то начинаем анализ следующего кластера

jmp ErrFind ; Файл не найден, выводим сообщение об ошибке

; Имя файла найдено

FindNameFound:

push word [es:di+14h]

push word [es:di+1Ah]

pop esi ; Записываем номер кластера в si

; Загрузка файла

push 60h

pop es

xor bx, bx

; Цикл поиска файла

FileReadContinue:

call ReadCluster ; чтение одного кластера в корневой директории

jc FileReadContinue ; Если не считали все, то продолжаем чтение

; Запуск исполняемой программы

push 60h

pop ds ; записываем в ds значение 60h

mov ax, ds ; В ax и ds номер сегмента, куда будет загружен исполняемый файл

sub ax, 10h

mov es, ax

mov ds, ax

mov ss, ax

xor sp, sp ; Очистка стека

push es

push word 100h ; Записываем в стек адрес, по которому будет загружен файл

mov dl, [cs:var\_logical\_drive\_number]

retf

; Функция чтения кластера

ReadCluster:

mov ax, [var\_byte\_per\_sector]

shr ax, 2 ; Помещаем в ax количество записей в секторе

cwde

mov ebp, esi ; Записываем в ebp номер кластера

xchg eax, esi ; Меняем значения в eax и esi

cdq

div esi ; В eax записан номер сектора

movzx edi, word [var\_reserved\_sector\_counter] ; Записываем в edi количество зарезервированных секторов

add edi, [var\_number\_of\_hidden\_sectors] ; Прибавляем к этому количество спрятанных секторов

add eax, edi ; Прибавляем номер сектора

push dx ; Записываем в стек dx (номер сектора)

mov cx, 1

call ReadSector ; Функция чтения одного сектора

pop si ; Выкидываем ненужную запись из si. Теперь на вершине стека номер сектора внутри кластера

add si, si ; Увеличиваем si в 4 раза

add si, si ; что бы получить маску следующего кластера

and byte [es:si+3], 0Fh ; маска значения кластера

mov esi, [es:si] ; В esi записываем номер следующего кластера

lea eax, [ebp-2]

movzx ecx, byte [var\_sector\_by\_claster] ; Записываем в ecx количество секторов в кластере

mul ecx ; Возводим в квадрат

mov ebp, eax ; Записываем в ebp остаток

movzx eax, byte [var\_num\_fat\_copies] ; Записываем в ax количество копий

mul dword [var\_number\_of\_sectors\_per\_fat] ; Умножаем на количество секторов в FAT-таблице

add eax, ebp ; прибавляем остаток от возведения в квадрат

add eax, edi ; Получаем адрес нужного сектора

call ReadSector

mov ax, [var\_byte\_per\_sector] ; Записываем в ax количество байт в секторе

shr ax, 4 ; Записываем ax количество слов в секторе (сдвиг враво на 4)

mul cx ; умножаем на количество секторов (записан в cx после функции ReadSector)

mov cx, es ; записываем в cx номер текущего кластера

add cx, ax ; получаем номер следующего кластера, прибавив к номеру текущег кластера размер текущего кластера

mov es, cx ; Запись в es:bx номер следующего кластера

cmp esi, 0FFFFFF8h ; Если последний кластер, то перенос будет равен 0, иначе 1

ret

; Чтение сектора

ReadSector:

pushad ; сохраняем регистры общего назначения в стек

; Чтение следующего сектора

ReadSectorNext:

pusha ; Сохраняем регистры в стеке

; Запись команд в стек

push byte 0 ; записываем в стек 0

push byte 0 ; записываем в стек 0

push eax ; записываем в стек 1 для регулирования LBA

push es ; записываем в стек es

push bx ; записываем в стек смещение

push byte 1 ; слово счетчика 1 сектор

push byte 16 ; размер пакета 16 байт, зарезервированныx 0 байт

mov ah, 42h

mov dl, [var\_logical\_drive\_number] ; Записываем в dl номер устройства

mov si, sp ; записываем в si указатель стека

push ss ; записываем в стек ss

pop ds ; записываем значение ss в регистр ds

int 13h ; прерывание дискового ввода - вывода

push cs ; запись значения в регистре cs

pop ds ; в регистр ds

jc short ErrRead

add sp, 16 ; две команды меняются местами чтобы не перезаписать флаг переноса

popa ; Восстанавливаем регистры

dec cx ; Уменьшаем счетчик

jz ReadSectorDone ; Последний сектор

add bx, [var\_byte\_per\_sector] ; регулирование смещения для следующего сектора

add eax, byte 1 ; регулирование LBA для следующего сектора

jmp short ReadSectorNext ; Читаем следующий сектор

ReadSectorDone:

popad ; сохраняем регистры в стеке

ret

PrintString: ;Процедура печати строки

mov ax, si ; сохраняем адрес стека

next\_character:

mov al, [si] ;Берем один байт из строки и записываем его в регистр AL

inc si ;Увеличиваем указатель SI

or al, al ;Проверка конца строки

jz exit\_function

call PrintCharacter ;Печатаем символ

jmp next\_character

exit\_function:

mov si, ax ; восстанавливаем стек

ret

PrintCharacter: ;Процедура печати символа

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

ret ;Возращение к вызванной процедуре

; Сообщение об ошибке

ErrRead:

ErrFind:

mov si, ErrorMessage ;Записываем в стек указатель на строку

call PrintString

exit\_err:

jmp short $

SelectMessage db "1.BOOT1.BIN", 13,10, "2.BOOT2.BIN", 13, 10, 0

ProgramName1 db "BOOT1 BIN" ; Имя исполняемой первой программы

ProgramName2 db "BOOT2 BIN" ; Имя исполняемой второй программы

ProgramName db "NNNNN BIN" ; Имя исполняемой программы

ErrorMessage db "ERROR", 13, 10, 0

SelectErrorMessage db "SELECT ERROR", 13, 10, 0

times (1024-2-($-$$)) db 0 ; Заполняем оставшееся пространство нулями

dw 0AA55h ; Сигнатура загрузчика

На рисунке 6 приведен вторичный загрузчик в HEX-редакторе.

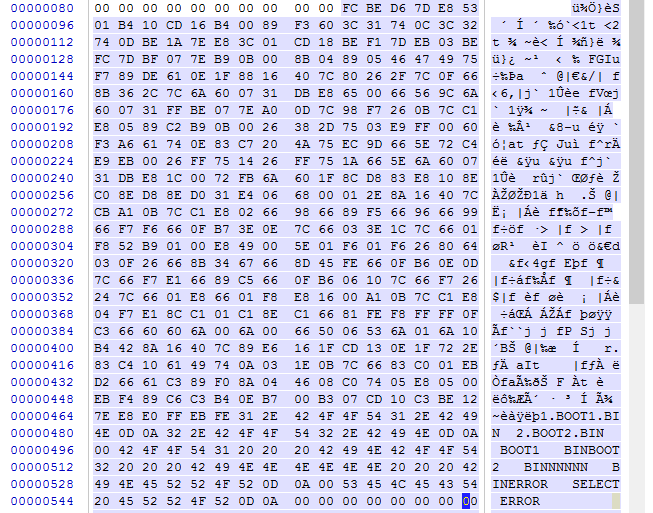


Рис. 6. Вторичный загрузчик из HEX-редактора

Запустим тестовую виртуальную машину. Результат запуска представлен на рисунке 7.

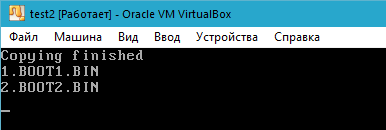


Рис. 7: Меню выбора файла для загрузки

На рисунке мы видим сообщение об успешном копировании первичного загрузчика на новый адрес, а так же меню выбора файла для загрузке. При нажатии на клавишу 1 загрузится первый файл - BOOT1.BIN (рисунок 8), а при нажатии на клавишу 2 - файл BOOT2.BIN При нажатии любой другой клавиши появится сообщение о неправильно нажатой клавише и загрузка прекратится (рисунок 9). Если выбранный файл отсутствует на носителе, то загрузка так же не произойдет (рисунок 10).

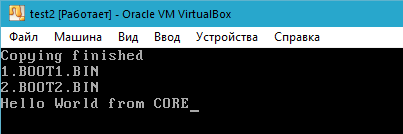


Рис. 8. Загрузка файла BOOT1.BIN

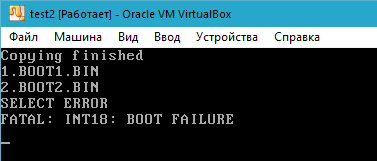


Рис.9. Нажатие на неправильную клавишу

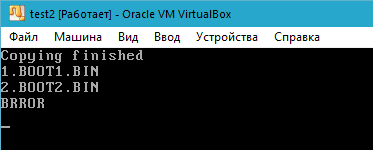


Рис. 10. Попытка загрузки несуществующего файла

**Вывод**

В данной работе были реализованы и протестированы первичный и вторичный загрузчик для файловой системы FAT32. Проверка работоспособности загрузчиков проводилась с помощью их загрузки на флеш-накопитель и подключением накопителя к виртуальной машине. Так же был реализован мультизагрузчик, который позволяет выбрать одну из друх программ для загрузки, производит их поиск в корневой директории файловой системы и запускает программу.

**Список использованных источников**

1. Интернет-источник. Master Boot Record (MBR). — http://akina.hop.ru/mbr.php3. Дата обращения – 20.04.2017
2. Интернет-источник. ATA in x86 RealMode (BIOS). —http://wiki.osdev.org/ATA\_in\_x86\_RealMode\_(BIOS). Дата обращения – 20.04.2017
3. Интернет-источник. <https://github.com/sysproglab/os-projects/blob/master/loader/loader.md> Дата обращения – 20.04.2017