**Тема**: Загрузчики

**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**

Загрузчик операционной системы — системное программное обеспечение, обеспечивающее загрузку операционной системы непосредственно после включения и начальной загрузки. Первым делом после запуска компьютера запускается BIOS. BIOS проводит инициализацию компьютера, определяет загрузочное устройство и копирует первый сектор устройства (MBR 512 байт) в оперативную память по адресу 0x7C00.

Далее запускается первичный загрузчик. Из-за ограничения на его размеры (512 байт) и сложность процесса загрузки, основной задачей первичного загрузчика является запуск вторичного загрузчика. Первичный загрузчик находит логический раздел, копирует первый сектор логического раздела в память по адресу 0x7C00 и передает ему управление.

Вторичный загрузчик занимается загрузкой ОС. Он находит на носителе образ ядра, загружает его в память, инициализирует необходимые регистры и передает управление ядру ОС.

**Выполнение работы**

1. **Подготовка рабочей станции**

Для выполнения данной работы необходимо подготовить виртуальную машину с загрузкой из flash-накопителя. Виртуальная машина используется для того, чтобы не повредить систему на основной машине. Для создания виртуальных машин использовалась VirtualBox. Его преимущество 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 10.0.15063.0

(C) Корпорация Майкрософт (Microsoft Corporation).

На компьютере: KIVI

DISKPART> list disk

Диск ### Состояние Размер Свободно Дин GPT

-------- ------------- ------- ------- --- ---

Диск 0 В сети 119 Gбайт 0 байт \*

Диск 1 В сети 931 Gбайт 0 байт \*

Диск 2 В сети 14 Gбайт 0 байт

DISKPART> exit

Завершение работы DiskPart...

C:\WINDOWS\system32>cd /d D:\Programms\VirtualBox

D:\Programms\VirtualBox>VBoxManage internalcommands createrawvmdk -filename C:\USB.vmdk -rawdisk \\.\PhysicalDrive2

RAW host disk access VMDK file C:\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. **Загрузчик HelloWorld**

Для знакомства с созданием загрузчиков создадим первичный загрузчик, который выводит на экран приветственное сообщение. Текст программы-загрузчика:

[BITS 16]

[ORG 0x7C00]

mov si, HelloString ; 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

;Data

HelloString db 'Hello World', 0

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

dw 0xAA55 ;Добавляем сигнатуру в конец загрузчика

Скомпилируем данную программу с помощью компилятора yasm:

yasm-1.3.0-win64.exe -f bin helloworld.asm -o helloworld.bin

Полученный в результате компиляции файл helloworld.bin необходимо загрузить в первый сектор флеш-накопителя. Сделаем это с помощью программы WinHex (рис. 1). После этого при запуске виртуальной машины мы увидим наше сообщение (рис. 2).

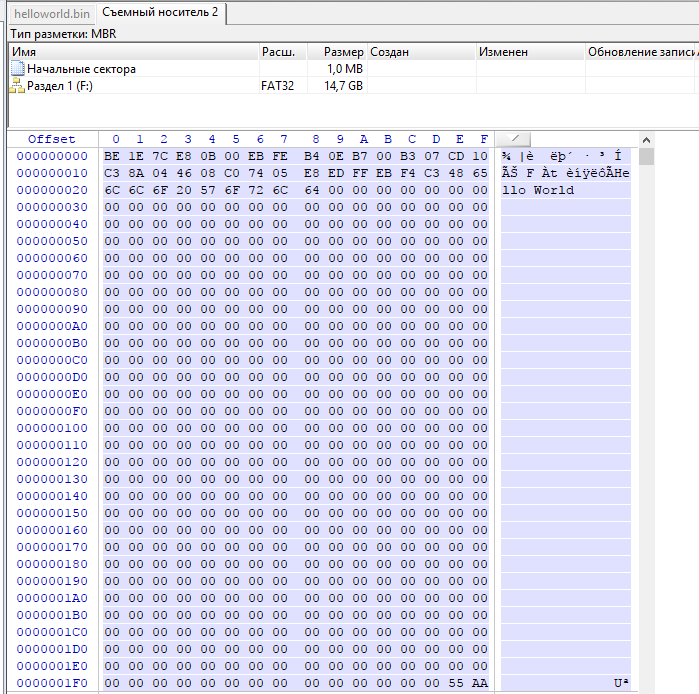


Рис. 1. Запись первичного загрузчика на флеш-носитель

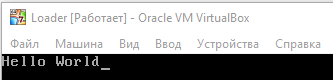


Рис. 2. Вывод сообщения из загрузчика на экран

Мы видим, что первичный загрузчик работает как предполагалось.

1. **Вторичный загрузчик**

Вторичный загрузчик располагается на разделе флеш-накопителя. Первичный загрузчик будет загружать вторичный загрузчик с флеш-накопителя. Поэтому, необходимо выяснить адрес первого раздела накопителя чтобы понять куда загружать загрузчик.

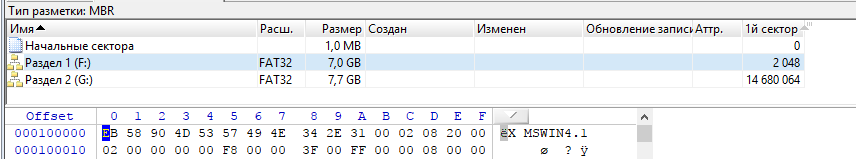


Рис. 3. Смещение первого раздела накопителя

Из рисунка видно, что первый раздел накопителя расположен по смещению 0x100000. Однако, в самом начале сектора располагается FAT-таблица, которую нельзя повредить. Поэтому, вторичный загрузчик располагается по смещению 0x5A относительно начала раздела.

Работа первичного загрузчика будет производиться по следующему алгоритму:

1. Инициализация регистров, стека;
2. Копирование загрузочного сектора по новому адресу (0x1000) 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.

Вторичный загрузчик просто выводит на экран сообщение “second bootloader hello”.

Исходный код первичного загрузчика:

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

Исходный код вторичного загрузчика:

org 0x7C00

bits 16

BS\_jmpBoot:

jmp start

nop

%include "fatTable.asm"

start:

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

HelloString: db "second bootloader hello", 13, 10, 0

end:

times 510 - (end - BS\_jmpBoot) db 0 ; добиваем нулями до 512 байт

db 0xAA,0x55

start\_second:

HelloString2: db "second bootloader hello", 13, 10, 0

end\_second:

times 510 - (end\_second - start\_second) db 0 ; добиваем нулями до 512 байт

db 0xAA,0x55

Скомпилируем оба загрузчика:

D:\Study\10 семестр\Проектирование ОС\lab6\src>yasm-1.3.0-win64.exe -f bin firstLoader.asm -o firstLoader.bin

D:\Study\10 семестр\Проектирование ОС\lab6\src>yasm-1.3.0-win64.exe -f bin secondHelloWorld.asm -o secondHelloWorld.bin

D:\Study\10 семестр\Проектирование ОС\lab6\src>

Загрузим бинарные файлы по указанным смещениям в флеш-носитель.

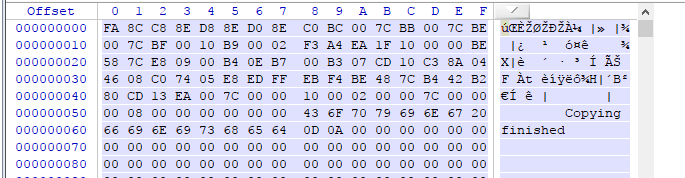


Рис. 4. Первичный загрузчик

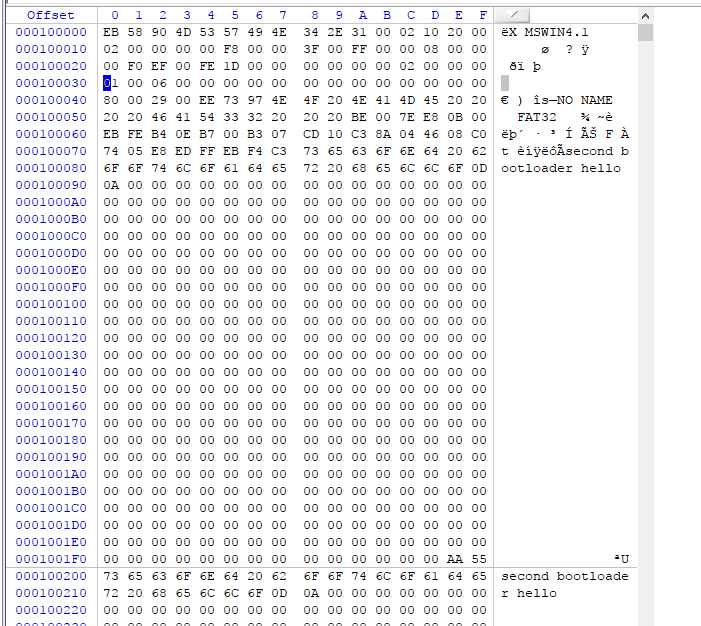


Рис. 5. Вторичный загрузчик

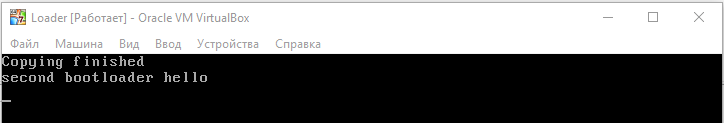


Рис. 6. Работа вторичного загрузчика.

На рисунке 6 видно, что созданный загрузчик работает правильно: первичному загрузчику удалось загрузить и передать управление вторичному загрузчику, который вывел на экран сообщение.

1. **Мультизагрузчик**

Мультизагрузчик позволяет при загрузке выбрать одну из нескольких загружаемых программ. Первичный загрузчик отработал правильно, поэтому он останется неизменный. Поменяется и усложнится только вторичный загрузчик. Общий алгоритм работы вторичного загрузчика:

1. Отключаем прерывания
2. Выводим пользователю сообщение со списком возможных для загрузки исполняемых файлов.
3. Сохраняем указатель на стек в регистре bx и анализируем введенный пользователем символ. Если пользователь выбрал одно из ядер, то переходим к его загрузке, иначе выводим сообщение об ошибке и завершаем работу.
4. Копируем имя выбранной пользователем программы в область памяти, где будет располагаться имя исполняемого файла, которую будем искать на накопителе и восстанавливаем указатель на стек.
5. Сохраняем номер загрузочного диска BIOS, максируем значение номера кластера и записываем полученное значение в регистр esi.
6. Начинаем цикл поиска файла в корневой директории, внутри которого пытаемся найти заданный файл в корневой директории. Если файл найден, то переходим к его загрузке, в противном случае - выводи сообщение об ошибке и завершаем работу.
   1. Записываем имя файла для поиска в стек и запускаем функцию поиска получения адреса кластера ReadCluster.
   2. Пытаемся найти запись о заданном файле в данном кластере. Если файл найден, то записываем его адрес в si и переходим к загрузке. В противном случае повторяем итерацию для следующего сектора (переход к шагу b).

Исходный код вторичного загрузчика:

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

Для того, чтобы проверить работу вторичного загрузчика были написаны две простые программы, которые выводят на экран разные сообщения. Скомпилируем все программы:

D:\Study\10 семестр\Проектирование ОС\lab6\src>yasm-1.3.0-win64.exe -f bin second.asm -o second.bin

D:\Study\10 семестр\Проектирование ОС\lab6\src>yasm-1.3.0-win64.exe -f bin BOOT1.asm -o BOOT1.bin

D:\Study\10 семестр\Проектирование ОС\lab6\src>yasm-1.3.0-win64.exe -f bin BOOT2.asm -o BOOT2.bin

D:\Study\10 семестр\Проектирование ОС\lab6\src>

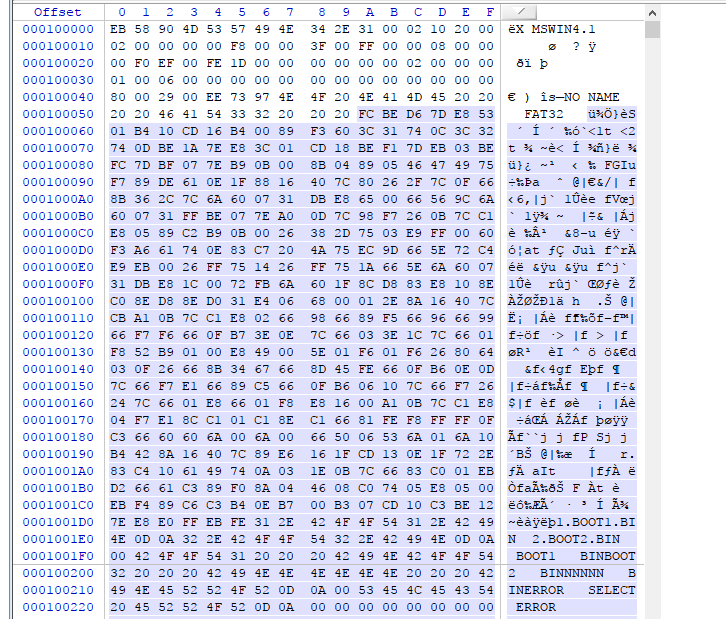


Рис. 5. Раздел 1 флеш-носителя со вторичным загрузчиком

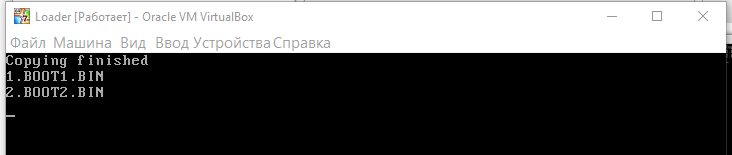


Рис. 6. Мультизагрузчик

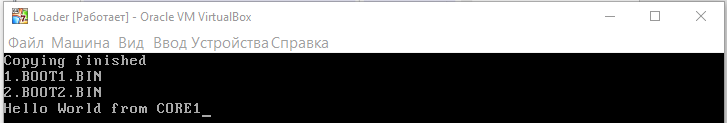


Рис. 8. Загрузка первой программы

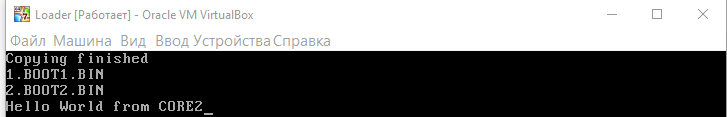


Рис. 9. Загрузка первой программы

Теоретически, используя созданный вторичный загрузчик можно загружать ОС. Для этого (на примере Linux) необходимо скопировать в корневой каталог флеш-накопителя скомпилированное ядро (файл с названием zImage или bzImage, получаемый в результате компиляции ядра). Скомпилированное ядро по сути это архив, при попытке его выполнить первым делом встроенный архиватор распакует само ядро и запустит его. Соответственно, чтобы загрузить ядро с помощью нашего загрузчика необходимо найти в корне накопителя файл с ядром, и скопировать его в память по адресу 0x7c00. Однако, при попытке произвести вышеописанные действия возникала неопределенная программа и виртуальная машина зависала при загрузке.

**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. Master Boot Record (MBR). — http://akina.hop.ru/mbr.php3.
2. [2] ATA in x86 RealMode (BIOS). —http://wiki.osdev.org/ATA\_in\_x86\_RealMode\_(BIOS).
3. https://github.com/sysproglab/os-projects/blob/master/loader/loader.md