**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.

Теория

Загрузчик операционной системы — системное программное обеспечение, обеспечивающее загрузку операционной системы непосредственно после включения компьютера (процедуры POST) и начальной загрузки. [1]

Загрузчик операционной системы:

* обеспечивает необходимые средства для диалога с пользователем компьютера (например, загрузчик позволяет выбрать операционную систему для загрузки);
* приводит аппаратуру компьютера в состояние, необходимое для старта ядра операционной системы (например, на не-x86 архитектурах перед запуском ядра загрузчик должен правильно настроить виртуальную память);
* загружает ядро операционной системы (Ntoskrnl.exe в случае MS Windows) в ОЗУ. Загрузка ядра операционной системы не обязательно происходит с жесткого диска. Загрузчик может получать ядро по сети. Ядро может храниться в ПЗУ или загружаться через последовательные интерфейсы (это может пригодиться на ранней стадии отладки создаваемой компьютерной системы);
* формирует параметры, передаваемые ядру операционной системы (например, ядру Linux передаются параметры, указывающие способ подключения корневой файловой системы);
* передаёт управление ядру операционной системы.

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:\Users\Saboteur>diskpart  Microsoft DiskPart версии 6.1.7601  (С) Корпорация Майкрософт, 1999-2008.  На компьютере: SABOTEUR-ПК  DISKPART> list disk  Диск ### Состояние Размер Свободно Дин GPT  -------- ------------- ------- ------- --- ---  Диск 0 В сети 465 Gбайт 0 байт  Диск 1 В сети 3824 Mбайт 0 байт  DISKPART> exit |

|  |
| --- |
| D:\>cd Programs\VirtualBox  D:\Programs\VirtualBox>VBoxManage internalcommands createrawvmdk -filename D:\Pr  ograms\VirtualBox\VirtualBoxVMs\USB.vmdk -rawdisk \\.\PhysicalDrive1  RAW host disk access VMDK file D:\Programs\VirtualBox\VirtualBoxVMs\USB.vmdk cre  ated successfully. |

VBoxManage internalcommands createrawvmdk -filename D:\Programs\VirtualBox\VirtualBoxVMs\USB.vmdk -rawdisk \\.\PhysicalDrive1

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 |

Нулевой сектор логического диска должен содержать в байте со смещением 0x1FE (51010) код 0x55, а в следующем байте (смещение 0x1FF) – код 0xAA. Указанные два байта являются признаком загрузочного диска.[2]

Скомпилируем данную программу с помощью компилятора yasm:

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

Полученный файл загрузчика записываем в загрузочный сектор flash накопителя с помощью утилиты WinHex. Теперь можно загрузить созданную виртуальную машину.

Результат загрузки представлен на рисунке 1.

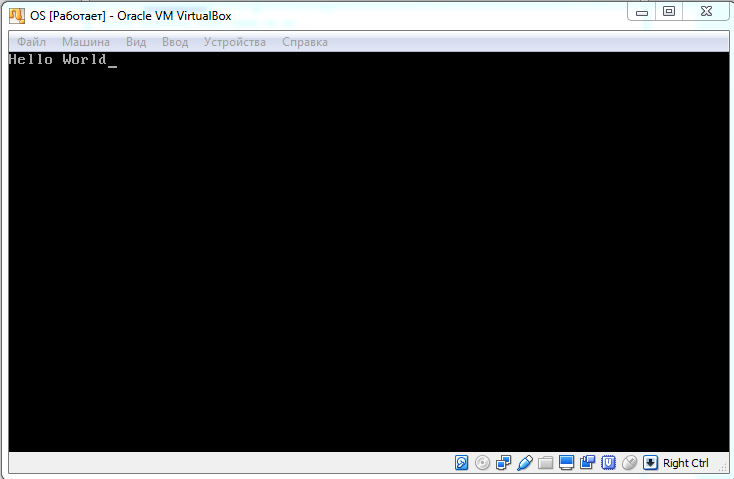


Рис.1. Загрузка виртуальной машины с flash накопителя

1. Вторичный загрузчик

Первичный загрузчик будет, как и ранее, записан в первичном секторе. Вторичный загрузчик запишем в первый раздел(Рис.2).

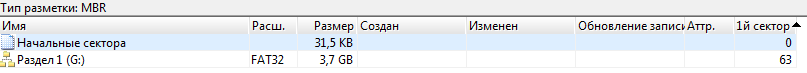


Рис.2 Разметка flash накопителя

В коде первичного загрузчика необходимо указать смещение первого раздела. Эту информацию можно извлечь из MDR записи начального сектора (Рис. 3).

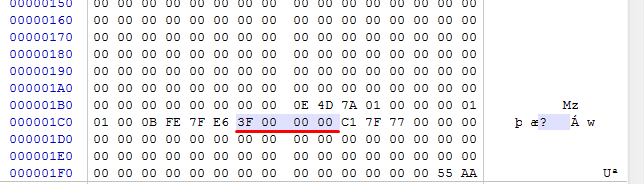


Рис.3 Адрес смещения первого раздела.

Адрес смещения записан младшими байтами вперед.

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”.

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 0x3F ;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 |

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 |

Таблица учета FAT32 формата. При компиляции программы, область, отводимая под FAT32 заполняется 0. При копировании программы на flash накопитель мы пропускаем эту область.

|  |
| --- |
| var\_name\_version:  dd 0  dd 0  var\_byte\_per\_sector:  dw 0  var\_sector\_by\_claster:  db 0  var\_reserved\_sector\_counter:  dw 0  var\_num\_fat\_copies:  db 0  var\_root\_links\_number:  dw 0  var\_total\_sector:  dw 0  var\_media\_descriptor:  db 0  var\_sector\_per\_fat:  dw 0  var\_sector\_per\_track:  dw 0  var\_number\_of\_heads\_per\_cylinder:  dw 0  var\_number\_of\_hidden\_sectors:  dd 0  var\_number\_of\_sector\_in\_partition:  dd 0  var\_number\_of\_sectors\_per\_fat:  dd 0  var\_flags:  dw 0  var\_fat\_drive\_version:  dw 0  var\_cluster\_number:  dd 0  var\_FS\_info:  dw 0  var\_section\_of\_backup\_root\_sector:  dw 0  var\_reserved:  dd 0  dd 0  dd 0  var\_logical\_drive\_number:  db 0  var\_reserved\_second:  db 0  var\_boot\_signature:  db 0  var\_serial\_ID:  dd 0  var\_volume\_name:  dd 0  dd 0  dw 0  db 0  var\_fat\_name:  dd 0  dd 0 |

Компиляция загрузчиков

|  |
| --- |
| D:\University\Master2\OS\labLoader>yasm-1.3.0-win64.exe -f bin secondLoader.asm  -o secondLoader.bin  D:\University\Master2\OS\labLoader>yasm-1.3.0-win64.exe -f bin firstLoader.asm -  o firstLoader.bin |

Шестнадцатеричное представление первичного и вторичного загрузчиков представлены на рисунках 4 и 5 соответственно.

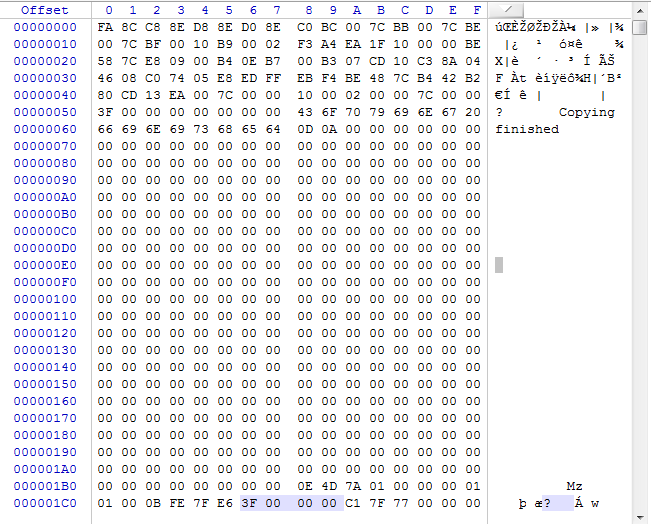
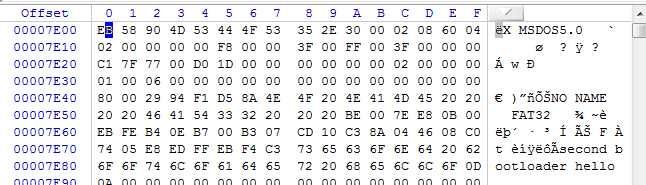


Рис.4. Первичный загрузчик

Рис.5 Вторичный загрузчик

Результат запуска загрузчиков представлен на рисунке 6.

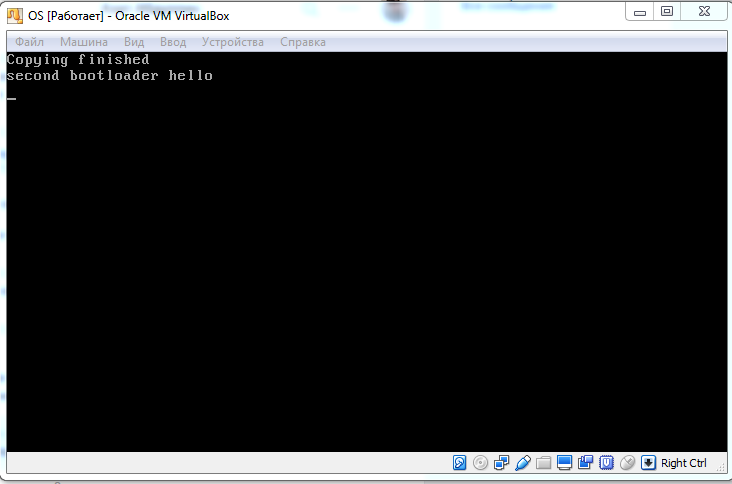


Рис.6. Загрузка первичного и вторичного загрузчиков.

Как видно из рисунка 6, вторичный загрузчик вывел желаемую строку.

1. Вторичный загрузчик с поиском загружаемого файла

Код первичного загрузчика не изменяется. В код вторичного загрузчика добавляется функция поиска фала загрузки.

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    mov bx, si ; сохраняем указатель на стек в bx  pusha    mov si, ProgramName1 ; 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 ; Преобразовать Word в DWord в регистре eax  mov ebp, esi ; Записываем в ebp номер кластера  xchg eax, esi ; Меняем значения в eax и esi  cdq ;Преобразовать DWord в QWord  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 $  ProgramName1 db "BOOT1 BIN" ; Имя исполняемой первой 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 |

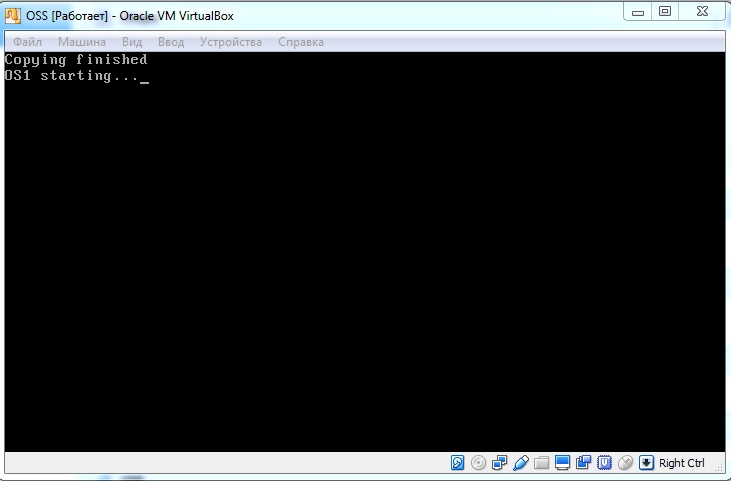


Рис.7. Работа вторичного загрузчика

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, маскируем 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.

a. Записываем имя файла для поиска в стек и запускаем функцию поиска получения адреса кластера ReadCluster.

b. 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).

Код программы 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 |

Код программы загрузки одной из программ

|  |
| --- |
| bits 16  org 0x100  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 'OS1 starting...', 0 |

На рисунках 7 и 8 представлены варианты загрузки двух разных программ.

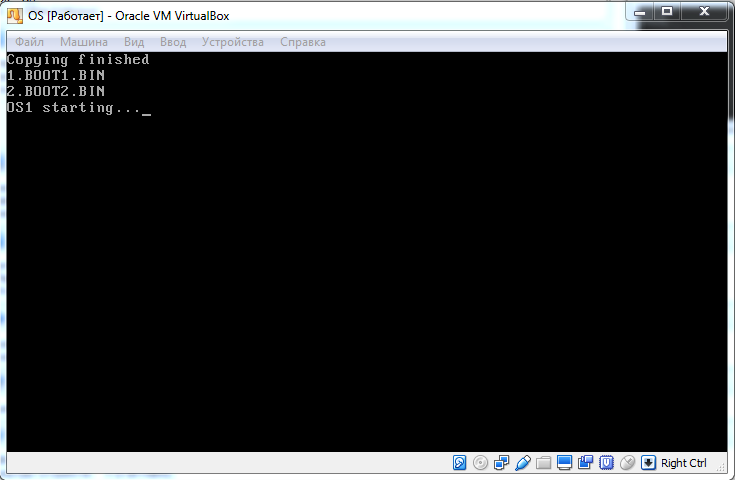


Рис. 8. Загрузка программы 1

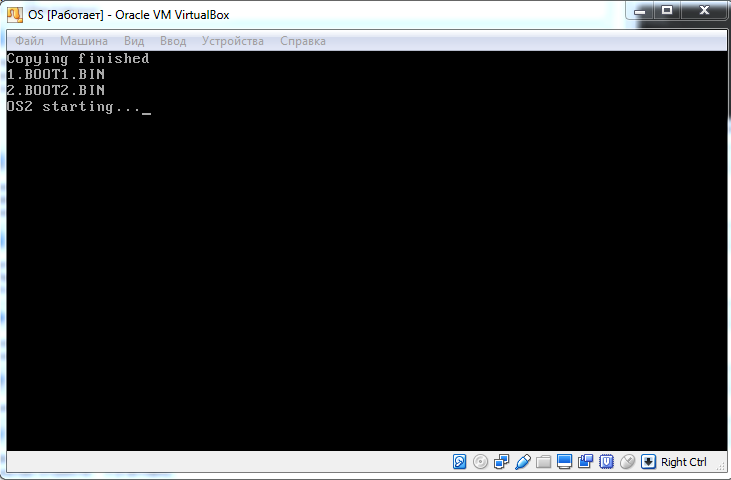


Рис.9. Загрузка программ 2

Как видно из рисунков, загрузка разных программ прошла успешна. Таки образом могут быть загружены и разные операционные системы.

Вывод

В данной лабораторной работе реализован и протестирован первичный загрузчик. Загрузчик записан на flash накопитель и его работоспособность установлена путем подключения к VirtualBox. Так же протестирована работа вторичного загрузчика. Далее на основе первичного и вторичного загрузчиков реализован мультизагрузчик.

Список используемых источников

1. Загрузчик операционной системы. URL: <https://ru.wikipedia.org/wiki/Загрузчик_операционной_системы>
2. Команды архитектуры x86 URL: <http://www.ccfit.nsu.ru/~kireev/lab2/lab2com.htm>
3. Регистры архитектуры x86 URL: <http://www.ccfit.nsu.ru/~kireev/lab2/lab2reg.htm>
4. Пишем свой загрузочный сектор URL: http://www.cyberguru.ru/programming/assembler/assembler-boot-sector.html