bDOS v0.5

Basic Disk Operating System version 0.5

Developer Documentation and user manual

Foreword

This operating system was written from scratch as a part of Operating Systems course' project. It took us 3 long months to bring it to this shape. We were completely new to NASM and still daily we find something that we haven't seen before. Its very primitive but very informative, it gave us a chance to learn alot more than we ever could have.
The aim was to write an OS to apply all the principles studied in the class. bDOS uses a floppy to boot, a FAT12 filesystem, some basic commands, its all very modular and students can feel free to add more to it as they wish. All the code is very well commented, to make sure the people understand it clearly. This project was difficult but we somehow due to our faculty Prof. Vijayrajan K and constant zeal made it and now we are proud to call ourselves the system developers.

Prateek Gupta
Abhishek Sharma
Abhinav Thakur

System Requirements

1. 1.44MiB Floppy Disk Drive
2. 16 MiB RAM
3. 8086 or Higher microprocessor
4. 20MiB Hard Disk Space
5. Display Unit

Source Code

The source code is comprised of 6 files.

1. boot.asm
Contains bootloader

2. Kernel.asm
Contains kernel

3. Routines.inc
Contains routines used by kernel

4. fat12.inc
Contains FAT12 driver used by Kernel

5. Data.inc
Contains Data used by Kernel

6. init.inc
Initializes all the registers before loading kernel

boot.asm

```
[BITS 16]
        ORG 0

                ;
                ;   code by Prateek Gupta, Abhishek Sharma, Abhinav Thakur
                ;       This Command Line OS was developed by us as a part of
OS project.
        jmp     START
        nop

    OEM_ID              db "VITOS 0.1"
    BytesPerSector      dw 0x0200
    SectorsPerCluster   db 0x01
    ReservedSectors     dw 0x0001
    TotalFATs           db 0x02
    MaxRootEntries      dw 0x00E0
    TotalSectorsSmall   dw 0x0B40
    MediaDescriptor     db 0xF0
    SectorsPerFAT       dw 0x0009
    SectorsPerTrack     dw 0x0012
    NumHeads            dw 0x0002
    HiddenSectors       dd 0x00000000
    TotalSectorsLarge   dd 0x00000000
    DriveNumber         db 0x00
    Flags               db 0x00
    Signature           db 0x29
    VolumeID            dd 0xFFFFFFFF
    VolumeLabel         db "VITOS   BOOT"
    SystemID            db "FAT12   "

    START:
    ; code located at 0000:7C00,marks start of boot
        cli
        mov     ax, 0x07C0
        mov     ds, ax
        mov     es, ax
        mov     fs, ax
        mov     gs, ax

        mov     ax, 0x0000
        mov     ss, ax
        mov     sp, 0xFFFF
        sti
```

```
    mov     si, msgLoading
    call    DisplayMessage
LOAD_ROOT:
; size of root
    xor     cx, cx
    xor     dx, dx
    mov     ax, 0x0020
    mul     WORD [MaxRootEntries]
    div     WORD [BytesPerSector]
    xchg    ax, cx
; location of root
    mov     al, BYTE [TotalFATs]
    mul     WORD [SectorsPerFAT]
    add     ax, WORD [ReservedSectors]
    mov     WORD [datasector], ax
    add     WORD [datasector], cx

    mov     bx, 0x0200
    call    ReadSectors

    mov     cx, WORD [MaxRootEntries]
    mov     di, 0x0200
.LOOP:
    push    cx
    mov     cx, 0x000B
    mov     si, ImageName
    push    di
rep  cmpsb
    pop     di
    je      LOAD_FAT
    pop     cx
    add     di, 0x0020
    loop    .LOOP
    jmp     FAILURE
LOAD_FAT:
; save starting cluster of boot image
    mov     si, msgCRLF
    call    DisplayMessage
    mov     dx, WORD [di + 0x001A]
    mov     WORD [cluster], dx              ; file's first cluster
; compute size of FAT and store in "cx"
    xor     ax, ax
    mov     al, BYTE [TotalFATs]            ; number of FATs
    mul     WORD [SectorsPerFAT]            ; sectors used by FATs
    mov     cx, ax
; compute location of FAT and store in "ax"
    mov     ax, WORD [ReservedSectors]      ; adjust for bootsector
; read FAT into memory (7C00:0200)
```

```
    mov     bx, 0x0200                  ; copy FAT above bootcode
    call    ReadSectors
; read image file into memory (0050:0000)
    mov     si, msgCRLF
    call    DisplayMessage
    mov     ax, 0x0050
    mov     es, ax                      ; destination for image
    mov     bx, 0x0000                  ; destination for image
    push    bx
LOAD_IMAGE:
    mov     ax, WORD [cluster]              ; cluster to read
    pop     bx                          ; buffer to read into
    call    ClusterLBA                  ; convert cluster to LBA
    xor     cx, cx
    mov     cl, BYTE [SectorsPerCluster]    ; sectors to read
    call    ReadSectors
    push    bx
; compute next cluster
    mov     ax, WORD [cluster]              ; identify current cluster
    mov     cx, ax                      ; copy current cluster
    mov     dx, ax                      ; copy current cluster
    shr     dx, 0x0001                  ; divide by two
    add     cx, dx                      ; sum for (3/2)
    mov     bx, 0x0200                  ; location of FAT in memory
    add     bx, cx                      ; index into FAT
    mov     dx, WORD [bx]               ; read two bytes from FAT
    test    ax, 0x0001
    jnz     .ODD_CLUSTER
.EVEN_CLUSTER:
    and     dx, 0000111111111111b
   jmp     .DONE
.ODD_CLUSTER:
    shr     dx, 0x0004
.DONE:
    mov     WORD [cluster], dx
    cmp     dx, 0x0FF0
    jb      LOAD_IMAGE
DONE:
    mov     si, msgCRLF
    call    DisplayMessage
    push    WORD 0x0050
    push    WORD 0x0000
    retf
FAILURE:
    mov     si, msgFailure
    call    DisplayMessage
    mov     ah, 0x00
    int     0x16
```

```
    int     0x19


DisplayMessage:
    lodsb
    or      al, al
    jz      .DONE
    mov     ah, 0x0E
    mov     bh, 0x00
    mov     bl, 0x07
    int     0x10
    jmp     DisplayMessage
.DONE:
    ret


ReadSectors:
.MAIN:
    mov     di, 0x0005
.SECTORLOOP:
    push    ax
    push    bx
    push    cx
    call    LBACHS
    mov     ah, 0x02
    mov     al, 0x01
    mov     ch, BYTE [absoluteTrack]
    mov     cl, BYTE [absoluteSector]
    mov     dh, BYTE [absoluteHead]
    mov     dl, BYTE [DriveNumber]
    int     0x13
    jnc     .SUCCESS
    xor     ax, ax
    int     0x13
    dec     di
    pop     cx
    pop     bx
    pop     ax
    jnz     .SECTORLOOP
    int     0x18
.SUCCESS:
    mov     si, msgProgress
    call    DisplayMessage
    pop     cx
    pop     bx
    pop     ax
    add     bx, WORD [BytesPerSector]
    inc     ax
```

```
    loop    .MAIN
    ret


ClusterLBA:
    sub     ax, 0x0002
    xor     cx, cx
    mov     cl, BYTE [SectorsPerCluster]
    mul     cx
    add     ax, WORD [datasector]
    ret


LBACHS:
    xor     dx, dx
    div     WORD [SectorsPerTrack]
    inc     dl
    mov     BYTE [absoluteSector], dl
    xor     dx, dx
    div     WORD [NumHeads]
    mov     BYTE [absoluteHead], dl
    mov     BYTE [absoluteTrack], al
    ret

absoluteSector db 0x00
absoluteHead   db 0x00
absoluteTrack  db 0x00

datasector  dw 0x0000
cluster     dw 0x0000
ImageName   db "KERNEL  BIN"
msgLoading  db 0x0D, 0x0A, "Loading BDOS ", 0x0D, 0x0A, 0x00
msgCRLF     db 0x0D, 0x0A, 0x00
msgProgress db ".", 0x00
msgFailure  db 0x0D, 0x0A, "ERROR : Press Any Key to Reboot", 0x00

    TIMES 510-($-$$) DB 0
    DW 0xAA55
```

Kernel.asm

```
[BITS 16]
org 0x0

%include 'init.inc'         ;initialisation routine
%include 'routines.inc'             ;all routines used in kernel
%include 'fat12.inc'          ;fat12 filesystem driver
%include 'data.inc'                 ;all data used in kernel

mov si,welcome
call sprint               ;prints out the welcome string

main:
xor cx,cx
mov si,newlin
call sprint               ;prints a new line

mov si,drive0
call sprint               ;prints out 'A:\'

mov si,prompt
call sprint               ;prints the prompt '>'

mov di,bufferz
call inputs                ;waits for input from user

mov si,cmd
call flush                ;clears the cmd buffer

mov si,param1
call flush                ;clears the param1 buffer

mov si,ImageName
call flush

mov si,bufferz
call cmdparser             ;parses the user input into
command,parameter1,parameter2,parameter3

mov si,cmd
call ltou                 ;converts command in buffer into all caps

mov di,commands
.loop1:
mov si,cmd
call comps                ;compares the user command with the predefined
```

```
commands
inc cx                  ;cx is counting which command was compared by
order
cmp ah,0                ;if user input do not matches with the predefined
command,
je .loop1               ;then jump to make another loop
cmp cx,8                ;each command has it's own number and they are
'jumped' to according to this order
je type1
cmp cx,7
je cd2
cmp cx,6
je cd
cmp cx,5
je dir
cmp cx,4
je cls
cmp cx,3
je reboot
cmp cx,2
je ver
cmp cx,1
je help

mov si,bad              ;prints out a text if the entered command was not
found
call sprint
jmp main

;       HELP command        ;


help:                   ;prints out available commands
mov si,xhelp
call sprint
jmp main

;       VER command         ;


ver:
cmp word [param1],"/?"      ;if parameter1 buffer holds the '/?'
string,then
je helptext1            ;jump to specified label and display command
usage
mov si,ver1
call sprint             ;displays BDOS version
jmp main
```

```
helptext1:
mov si,verhelp
call sprint
jmp main

;         EXIT command         ;


reboot:
cmp word [param1],"/?"             ;if parameter1 buffer holds the '/?'
string,then
je helptext2                       ;jump to specified label and display command
usage
db 0x0ea,0x0f0,0x0ff,00h,0x0f0     ;code for reboot
helptext2:
mov si,exithelp
call sprint
jmp main

;         CLS command          ;


cls:
cmp word [param1],"/?"             ;if parameter1 buffer holds the '/?'
string,then
je helptext3                       ;jump to specified label and display command
usage
mov cx,25                          ;counter is set to 25 because DOS can display
25 line
loop:
mov si,clear
call sprint                       ;prints out 25 times CR and LF chars
dec cx                            ;cx is decremented in each run
cmp cx,0                           ;if cx = 0 then
je set_cursor                      ;set the cursor position
jmp loop
set_cursor:
xor bx,bx                          ;bx register is cleared,we want to print on first
page of screen
mov ah,0x02                        ;see Ralph Brown's interrupt list (int 0x10 ,
ah = 0x2)
xor dx,dx
int 0x10
jmp main
clear dw 0x0d0a,"$"                ;CR LF in hexadecimal
helptext3:
mov si,clshelp
call sprint
```

```
jmp main

;          DIR command          ;


dir:
cmp word [param1],"/?"              ;if parameter1 buffer holds the '/?'
string,then
je helptext0                        ;jump to specified label and display command
usage
pusha                               ;registers and flags are saved onto stack
lea si,[rdir]                       ;loads the starting memory address of the FAT
table into 'si'
add si,[fatsize]                    ;and adds the size of the FAT table to 'si' (now
si = root directory
sub si,32                           ;starting address - 32)

loop2:
add si,32                           ;starting address of root directory
push si                             ;'si' is saved onto stack
mov bx,si
push bx
cmp byte [ds:si],0xe5               ;if the first char in the root directory entry
equals to 0xe5 (deleted file)
je loop2                            ;then check the next entry
cmp byte [ds:si+2],0x00             ;if the 3rd char is equal to 0 ,then
je checkit                          ;jump to check the first char
mov cx,11                           ;filename is 11 chars long
mov di,filename.ext                 ;buffer for filenames

repeat1:
cmp cx,0                            ;loop until cx = 0 and then
je ext                              ;jump to check is it a directory
mov ax,[ds:si]                      ;copies one char to out buffer
mov [di],ax
inc di                              ;increments pointers
inc si
dec cx                              ;decrements the counter
jmp repeat1

ext:
cmp byte [bx+11],0x10               ;check byte 11 ,does it queals to 0x10
(directory) and then
je printit                          ;jumps to print directory out

printit:
mov byte [di],"$"                   ;put a string terminator '$' after the filename
mov si,filename.ext
```

```
call sprint                     ;display our filename
mov si,space2                       ;print out some spaces
cmp byte [bx+11],0x10               ;if the 11th byte in the entry equals to
0x10 then,
jne go                          ;jump over
mov si,dirsign
call sprint                     ;prints out '<DIR>' sign
mov si,space

go:
call sprint                     ;some spaces
mov di,filesize
pop bx                          ;bx is now the starting address of the root
directory entry
lea si,[bx+28]
call hex2ascii                    ;coverts file size (bits 28-31) from hex to ASCII
decimal
mov si,filesize
call sprint                     ;prints out file size
mov si,newlin
call sprint                     ;newline
pop si
jmp loop2

checkit:
cmp byte [ds:si],0                ;if the first byte in the entry does not equals
to 0 then
jne loop2                        ;jump to loop again,else its end of the root
directory

finish:
popa                            ;restores registers and flags from stack
jmp main

helptext0:
mov si,dirhelp
call sprint
jmp finish
filename.ext db 0,0,0,0,0,0,0,0,0,0,0,0
filesize db 0,0,0,0,0,0,0,0
space db 0x20,"$"
space2 db "      $"

;         CD command          ;


cd:
cmp word [param1],"/?"              ;if parameter1 buffer holds the '/?'
```

```
string,then
je helptext5                    ;jump to specified label and display command
usage
pusha
cmp byte [param1],0x20          ;if the parameter1 buffer is empty
( filled with spaces) ,then
je show_path                    ;jump to show current path
mov si,param1
call ltou                       ;converts string in param1 buffer to uppercase
mov di,ImageName

dirname:
cmp byte [si],"$"               ;if the char in the ImageName bufer is a
string terminator ,then
je mod_path                     ;jump to modify the path
mov al,[si]                     ;copies a char from param1 to ImageName
mov byte [di],al
inc si                          ;pointers are incremented
inc di
jmp dirname

mod_path:
call rdirparse                  ;search for the filename in the directory
entries
mov si,drive0
call strlen                     ;determine the lenght of the current path string
lea bx,[drive0+di]              ;bx now points to the and of the current
path string
mov si,param1

cd_loop:
cmp byte [si],"$"               ;if the char is a string terminator ,then
je dirname1                     ;jump to put the slash on the end of the
string
mov al,[si]                     ;copies characters from param1 to drive0
mov byte [bx],al
inc bx                          ;pointers are incremented
inc si
jmp cd_loop

dirname1:
mov word [bx],"\$"              ;'\$' is appended to the end of the current
path string
lea bx,[rdir]                   ;starting address of the root directory is
calculated
add bx,[fatsize]
call LOAD_IMAGE                 ;loads the directory file from disk into
calculated location
```

```
cd_end:
jmp main

show_path:
call sprint
mov si,drive0
call sprint                     ;displays current path
pop si
call sprint
jmp cd_end

helptext5:
mov si,cdhelp
call sprint
jmp cd_end

;         CD.. command        ;


cd2:
lea bx,[rdir]                   ;calculates the starting address of the root
directory
add bx,[fatsize]
mov si,bx
add si,32
cmp word [si],".."              ;if the first 2 bytes in the second entry in
the root directory is not "..",then
jne cd2_end                     ;jump to end (no parent directory = root)
pusha
mov si,drive0
call strlen                     ;the lenght of the current path is determined
add di,si                       ;now di points to the end of the current path
string
sub di,2                        ;di - 2 because we have to "jump" over the last
slash

separator:
dec di
cmp byte [di],"\"               ;if the character doesn't equals to "\",then
jne separator                   ;try the next one,else
inc di                          ;increment the pointer and
mov byte [di],"$"               ;put a string terminator
popa                            ;restores registers and flags,among them di =
starting address of the root directory entry
mov di,[si+0x1a]                ;saves the starting cluster of the file
mov [cluster],di
cmp word [cluster],0            ;if cluster = 0 then
je rootload                     ;jump to load the root directory,else
```

```
call LOAD_IMAGE                 ;load the directory file specified by
filename
jmp cd2_end

rootload:
call LOAD_ROOT                  ;loads root directory

cd2_end:
jmp main

;       TYPE command        ;


type1:
cmp word [param1],"/?"              ;if parameter1 buffer holds the '/?'
string,then
je helptext6                    ;jump to specified label and display command
usage
pusha
mov si,param2
call ltou                       ;converts the string in parameter2 buffer into
uppercase
mov si,param1
call ltou                       ;converts the string in parameter2 buffer into
uppercase
mov di,ImageName
call fnconv                     ;converts the filename to match the directory
entry type (from param1 to ImageName)
cmp byte [di+0x9],0x20              ;if the file has no extension
(directory),then
je done2                        ;jump to display error message,else
call rdirparse                  ;search directory entries for the specified
filename
mov bx,0x4000                   ;location where the specified file is loaded
call LOAD_IMAGE                 ;load it
mov di,0x4000                   ;calculates the address of the end of the
file and
add word di,[si+28]
mov byte [di],"$"               ;adds a string terminator
cmp word [param2],"/P"              ;if parameter2 equals to "/p" then
je type1_page                   ;jump to type the file in pages,else
mov si,0x4000
call sprint                     ;display the content of the file
jmp main

type1_page:                     ;prints the content of the file in screenful
pages
mov si,0x4000
```

```
pusha
xor cx,cx
mov ah, 0x0E
repeat2:
        lodsb
        inc cx
    cmp al,"$"
        je done1
        int 0x10
        cmp cx,1080
        je next2
        jmp repeat2

next2:
call pause                      ;pauses after each pages of content
xor cx,cx
jmp  repeat2

done1:
popa
jmp main

done2:
mov si,errmsg1
call sprint
jmp done1
helptext6:
mov si,typehelp
call sprint
jmp done1

pause:
pusha
mov si,pausemsg
call sprint
xor ax,ax
int 0x16
mov si,newlin
call sprint
popa
ret
```

Routines.inc

```
;Print_string routine

sprint:
     pusha          ; Routine: output string in SI to screen
     mov ah, 0x0E             ; int 10h 'print char' function

repeat:
       lodsb               ; Get character from string
   cmp al,"$"
       je done             ; If char is zero, end of string
       int 0x10    ; Otherwise, print it
       jmp repeat

done:
popa
ret

;Input_string routine

inputs:
pusha
mov dx,di
input:
mov ah,0x0
int 0x16
cmp al,0x0d
je enter_pressed
cmp al,0x08
je backspace_pressed
mov ah,0x0e
int 0x10
stosb
jmp input

backspace_pressed:
cmp dx,di
je input
mov ah,0x0e
int 0x10
mov ah,0x0a
mov al,0x20
xor bx,bx
mov cx,2
int 0x10
dec di
```

```
mov byte [di],0
jmp input

enter_pressed:
mov byte [di],"$"
mov ah,0x0e
mov al,0x0d
int 0x10
mov al,0x0a
int 0x10
popa
ret

;Compare_string routine

comps:
push cx
xor ax,ax
cmp byte [di],"$"
je equ
cld
mov cx,[di]
xor ch,ch
inc di
repe cmpsb
jne notequ
cmp byte [si],"$"
je equ
cmp byte [si]," "
je equ
notequ:
add di,cx
pop cx
mov ah,0x0
jmp end
equ:
pop cx
mov ah,0x01
end:
ret

;Hexadecimal_to_ASCII routine


hex2ascii:
mov dword  eax,[si]
hextoasc:
                          ;si input address , di point result storage addres
```

```
        pusha
        mov ecx,00h
        mov ebx,0ah
        hexloop1:
                mov edx,0
                div ebx
                add dl,'0'
                push edx
                inc ecx
                cmp eax,0ah
                jge hexloop1
                add al,'0'
                mov [di],al
        hexloop2:
                pop eax
                inc di
                mov [di],al
                loop hexloop2
                inc di
                mov al,'$'
                mov [di],al
                popa
                ret

;memcpy routine

memcpy:
cld
rep movsb
ret

;string_lenght routine

strlen:
xor di,di
push si
calc_len:
cmp byte [si],"$"
je strlen_end
inc si
jmp calc_len
strlen_end:
mov di,si
pop si
sub di,si
ret

;flush_buffer routine
```

```
flush:
pusha
flush_start:
cmp byte [si],"«"
je flush_end
mov byte [si]," "
inc si
jmp flush_start
flush_end:
popa
ret

fnconv:
pusha
xor cx,cx
fnconv_loop:
cmp byte [si],"."
je fnconv_next
mov byte al,[si]
mov byte [di],al
inc si
inc di
inc cl
jmp fnconv_loop
fnconv_next:
mov ch,7
sub ch,cl
fnconv_loop1:
cmp ch,0
je fnconv_next1
mov byte [di],32
inc di
dec ch
jmp fnconv_loop1
fnconv_next1:
mov ch,3
fnconv_loop2:
cmp ch,0
je fnconv_end
inc si
inc di
dec ch
mov byte al,[si]
mov byte [di],al
jmp fnconv_loop2
fnconv_end:
popa
```

```
ret

;Lcase_to_Ucase routine

ltou:
pusha
ltou_start:
cmp byte [si],96
jng ltou_next
cmp byte [si],123
jnl ltou_next
sub word [si],32
inc si
jmp ltou_start
ltou_next:
cmp byte [si],"$"
je ltou_end
inc si
jmp ltou_start
ltou_end:
popa
ret

;Command_parser routine

cmdparser:
pusha
xor dx,dx
xor cx,cx
xor al,al
mov ah,1
mov di,cmd
cmdparser_start1:
cmp byte [si]," "
je cmdparser_next
cmp byte [si],"$"
je cmdparser_final
cmp word cx,12
je cmdparser_bad
inc cx                  ;counts the characters
mov ah,0
mov byte al,[si]
mov byte [di],al
inc si
inc di
jmp cmdparser_start1
cmdparser_loop:
mov byte [di],"$"
```

```
inc dx                  ;counts buffers
cmp word dx,3
je cmdparser_final
sub di,cx
add di,12
xor cx,cx
jmp cmdparser_start1
cmdparser_next:
inc ah                  ;counts the spaces
cmp byte ah,1
je cmdparser_loop
inc si
jmp cmdparser_start1
cmdparser_bad:
mov si,badmsg
call sprint
jmp cmdparser_end
cmdparser_final:
mov byte [di],"$"
cmdparser_end:
popa
ret
badmsg       db "Invalid string lenght!$"
```

fat12.inc

```
LOAD_ROOT:
    ; compute size of root directory and store in "cx"
        xor     cx, cx
        xor     dx, dx
        mov     ax, 0x0020                      ; 32 byte directory entry
        mul     WORD [MaxRootEntries]             ; total size of directory
        div     WORD [BytesPerSector]             ; sectors used by directory
        xchg    ax, cx
    ; compute location of root directory and store in "ax"
        mov     al, BYTE [TotalFATs]              ; number of FATs
        mul     WORD [SectorsPerFAT]              ; sectors used by FATs
        add     ax, WORD [ReservedSectors]          ; adjust for bootsector
        mov     WORD [datasector], ax             ; base of root directory
        add     WORD [datasector], cx
    ; read root directory into memory (7C00:0200)

        lea     bx,[rdir]
        add     bx,[fatsize]
        call    ReadSectors
        ret
LOAD_FAT:
    ; compute size of FAT and store in "cx"
        xor     ax, ax
        mov     al, BYTE [TotalFATs]              ; number of FATs
        mul     WORD [SectorsPerFAT]              ; sectors used by FATs
        mov     cx, ax
    ; compute location of FAT and store in "ax"
        mov     ax, WORD [ReservedSectors]          ; adjust for bootsector
    ; read FAT into memory (rdir address)
        mov     bx, rdir                        ; copy FAT on rdir address
        call    ReadSectors
    ret

;Root_directory_parse routine

rdirparse:
        pop ax
                mov word [pointer],ax
        mov     cx, WORD [MaxRootEntries]           ; load loop counter
        mov    di,rdir                          ; locate first root entry
                add    di,[fatsize]
    .LOOP:
        push    cx
        mov     cx, 0x000B                      ; eleven character name
        mov     si, ImageName                               ; image name to find
                push di
```

```
    rep  cmpsb                              ; test for entry match
        pop di
        je      next
        pop     cx
        add     di, 0x0020                  ; queue next directory entry
        loop    .LOOP
        jmp     FALIURE
       next:
           mov     dx, WORD [di + 0x001a]
        mov     WORD [cluster], dx              ; file's first cluster
        mov     si,di
        mov word ax,[pointer]
                jmp ax

;Load_Image routine

LOAD_IMAGE:
        pop ax
                mov word [pointer],ax
        mov ax,0x0050                       ; destination for image
                mov es,ax
        push    bx                          ;es:bx is loaded with destination
address before invoking
                                            ;this function
    LOAD_IT:
        mov     ax, WORD [cluster]              ; cluster to read
        pop     bx                          ; buffer to read into
        call    ClusterLBA                  ; convert cluster to LBA
        xor     cx, cx
        mov     cl, BYTE [SectorsPerCluster]    ; sectors to read
        call    ReadSectors
        push    bx
    ; compute next cluster
        mov     ax, WORD [cluster]              ; identify current cluster
        mov     cx, ax                      ; copy current cluster
        mov     dx, ax                      ; copy current cluster
        shr     dx, 0x0001                  ; divide by two
        add     cx, dx                      ; sum for (3/2)
        mov     bx,rdir                     ; location of FAT in memory
        add     bx, cx                      ; index into FAT
        mov     dx, WORD [bx]                   ; read two bytes from FAT
        test    ax, 0x0001
        jnz     .ODD_CLUSTER
    .EVEN_CLUSTER:
        and     dx, 0000111111111111b           ; take low twelve bits
       jmp     .DONE
    .ODD_CLUSTER:
        shr     dx, 0x0004                  ; take high twelve bits
```

```
.DONE:
    mov     WORD [cluster], dx              ; store new cluster
    cmp     dx, 0x0FF0                      ; test for end of file
    jb      LOAD_IT
            mov word ax,[pointer]      ;patch ;)
            jmp ax

; PROCEDURE ReadSectors

ReadSectors:
.MAIN:
    mov     di, 0x0005                      ; five retries for error
.SECTORLOOP:
    push    ax
    push    bx
    push    cx
    call    LBACHS
    mov     ah, 0x02                        ; BIOS read sector
    mov     al, 0x01                        ; read one sector
    mov     ch, BYTE [absoluteTrack]          ; track
    mov     cl, BYTE [absoluteSector]         ; sector
    mov     dh, BYTE [absoluteHead]           ; head
    mov     dl, BYTE [DriveNumber]            ; drive
    int     0x13                            ; invoke BIOS
    jnc     .SUCCESS                        ; test for read error
    xor     ax, ax                          ; BIOS reset disk
    int     0x13                            ; invoke BIOS
    dec     di                              ; decrement error counter
    pop     cx
    pop     bx
    pop     ax
    jnz     .SECTORLOOP                     ; attempt to read again
    int     0x18
.SUCCESS:
    pop     cx
    pop     bx
    pop     ax
    add     bx, WORD [BytesPerSector]         ; queue next buffer
    inc     ax                              ; queue next sector
    loop    .MAIN                           ; read next sector
    ret

; PROCEDURE LBACHS

LBACHS:
    xor     dx, dx                          ; prepare dx:ax for operation
    div     WORD [SectorsPerTrack]            ; calculate
    inc     dl                              ; adjust for sector 0
```

```
        mov     BYTE [absoluteSector], dl
        xor     dx, dx                          ; prepare dx:ax for operation
        div     WORD [NumHeads]                 ; calculate
        mov     BYTE [absoluteHead], dl
        mov     BYTE [absoluteTrack], al
        ret

    ; PROCEDURE ClusterLBA

    ClusterLBA:
        sub     ax, 0x0002                      ; zero base cluster number
        xor     cx, cx
        mov     cl, BYTE [SectorsPerCluster]    ; convert byte to word
        mul     cx
        add     ax, WORD [datasector]           ; base data sector
        ret
FALIURE:
mov si,fail
call sprint
jmp main
```

data.inc

```
;-------------------------------------
;Kernel_data
;-------------------------------------

prompt db ">$"
drive0 db "VITOS:\","$"
times 50 db 0
bufferz db "$"
times 50 db 0
xhelp dw
"HELP",0x09,"VER",0x09,"EXIT",0x09,"CLS",0x09,"DIR",0x09,"CD",0x0d0
a,0x0d0a,"$"
ver1 dw "VITOS 0.1",0x0d0a,"$"
bad dw 0x0d0a,"Bad command!",0x0d0a,"$"
welcome dw "VIT Disk Operating System version 0.1 ",0x0d0a,"$"
commands db
0x4,"HELP",0x3,"VER",0x4,"EXIT",0x3,"CLS",0x3,"DIR",0x2,"CD",0x4,"CD.
.",0x4,"TYPE","$"
newlin dw 0x0d0a,"$"
dirsign db " <DIR>$"
dirhelp dw 0x0d0a,"Lists local
directory",0x0d0a,0x0d0a,"DIR",0x0d0a,"$"
cdhelp dw 0x0d0a,"Displays the name or changes to local
directory.",0x0d0a,0x0d0a,"CD..",0x0d0a,"CD [directory]",0x0d0a,"$"
verhelp dw 0x0d0a,"Displays BDOS
version.",0x0d0a,0x0d0a,"VER",0x0d0a,"$"
clshelp dw 0x0d0a,"Clears the screen.",0x0d0a,0x0d0a,"CLS",0x0d0a,"$"
typehelp dw 0x0d0a,"Lists the contents of a text
file.",0x0d0a,0x0d0a,"TYPE filename.ext [/p]",0x0d0a,"$"
exithelp dw 0x0d0a,"Restarts the
system.",0x0d0a,0x0d0a,"EXIT",0x0d0a,"$"
fail dw 0x0d0a,"Operation failed!",0x0d0a,"$"
pausemsg dw 0x0d0a,"Press any key to continue..","$"
errmsg1 dw "Not a file or bad file!",0x0d0a,"$"
pointer dw 0

Bios_BPB:                ;Space for bios parameter block copied here from
0x7c00 address
BytesPerSector        dw 0
SectorsPerCluster     db 0
ReservedSectors       dw 0
TotalFATs             db 0
MaxRootEntries        dw 0
TotalSectorsSmall     dw 0
MediaDescriptor       db 0
SectorsPerFAT         dw 0
```

```
SectorsPerTrack       dw 0
NumHeads              dw 0
HiddenSectors         dd 0
TotalSectorsLarge     dd 0
DriveNumber           db 0

;data used by fat driver
fatsize        dw 0
rootdirsize    dw 0
absoluteSector db 0x00
absoluteHead   db 0x00
absoluteTrack  db 0x00
datasector     dw 0x0000
cluster        dw 0
ImageName      db 32,32,32,32,32,32,32,32,32,32,32,"$"
;command parser space
cmd            db 0,0,0,0,0,0,0,0,0,0,0
param1         db 0,0,0,0,0,0,0,0,0,0,0
param2         db 0,0,0,0,0,0,0,0,0,0,0
param3         db 0,0,0,0,0,0,0,0,0,0,0,"«"
rdir db 0                   ;Location of the FAT table and root dir in memory
```

init.inc

```
;----------------------------------------
;BDOS_INIT
;----------------------------------------

cli
        mov     ax, 0x0050
        mov     es, ax
        mov     fs, ax
        mov     gs, ax
                                ; create stack
        mov     ax, 0x0000
        mov     ss, ax
        mov     sp, 0xFFFF
        sti

mov si,0xb
mov di,BytesPerSector
mov cx,25
call memcpy
cli
mov ax,0x0050
mov ds,ax
sti
call LOAD_FAT                           ;loads FAT table in the memory address
specified by 'rdir' label

xor     ax, ax                          ; compute size of FAT and store in fatsize
buffer
mov     al, BYTE [SectorsPerFAT]            ;
mul     WORD [BytesPerSector]               ;
mov [fatsize],ax
xor ax,ax
mov al,byte [MaxRootEntries]
mul word [BytesPerSector]
mul byte [SectorsPerCluster]
mov [rootdirsize],ax
call LOAD_ROOT                          ;loads root directory in the memory
address  'rdir+fatsize'
```

Instructions

To assemble this source code you will need NASM(Netwide Assembler) its available for free, either on Win32 or Linux environment use the following commands to assemble:

> nasm if=boot.asm of=boot.bin
> nasm if=kernel.asm of=kernel.bin

Now you will have two binary files named boot.bin and kernel.bin, use rawwrite.exe or any such program to write boot.bin to bootsector of the floppy disk, then place the kernel.bin to root of floppy disk. Then you would be able to boot your system from this floppy disk.

Conclusion

Building this project was really challenging, but we did it because it was interesting. Writing your own OS is the most fascinating task that any CS engineer would want to undertake. We learnt alot of new things, hacks and most importantly how everything works together so beautifully in a computer system.
We have studied all the concepts in different courses like Operating systems, Microprocessor and interfacing, Data Structures and Algorithms, Computer Architecture and Organisation. This project is the amalgam of all of them.

We have made it very modular so that its easy to add code later, we can add more functionality to it by adding more routines and features.

References

1. OSDev.org
2. Operating systems by William Stallings
3. Source code of MS DOS
4. NASM Handbook