Изучаемые вопросы   
1. Структура и расположение системной информации на томе FATxx.

2. Загрузочная запись BOOT: структура, расположение на томе.

3. Таблица расположения файлов FAT: структура, расположение на томе.

4. Корневой директорий ROOT: структура, расположение на томе.

5. Директорий/каталог: структура, расположение, структура записи.

6. Изучить алгоритм расположения файла на FAT томе:

а) определение номера начального кластера файла.

б) расчет номеров цепочки кластеров файла.

7. Исследовать изменение элементов таблиц FAT, родительского каталога при выполнении над файлом команд COPY, MOVE, DEL, RENAME.

8. Атрибуты файла: кодировка, управление.

9. Программно определить дату и время создания файла с именем NAME, подчиненного корневому каталогу ROOT.

**Создать программу, которая**:

а) выводит на экран дамп N-го сектора;

#include <windows.h>

#include <stdio.h>

int main(int argc, char \*\* argv){

int retCode = 0;

BYTE sector[512];

DWORD bytesRead;

HANDLE device = NULL;

int numSector = 5;

device = CreateFile(L"\\\\.\\C:", // Drive to open

GENERIC\_READ, // Access mode

FILE\_SHARE\_READ|FILE\_SHARE\_WRITE, // Share Mode

NULL, // Security Descriptor

OPEN\_EXISTING, // How to create

0, // File attributes

NULL); // Handle to template

if(device == INVALID\_HANDLE\_VALUE){

printf("CreateFile: %u\n", GetLastError());

return 1;}

SetFilePointer (device, numSector\*512, NULL, FILE\_BEGIN) ;

if (!ReadFile(device, sector, 512, &bytesRead, NULL)) printf("ReadFile: %u\n", GetLastError());

else printf("Success!\n");

return 0;}

б) выводит на экран расшифрованную таблицу BPB;

#include <stdio.h>

#include <stdlib.h>

#include <libintl.h>

#include <locale.h>

#include <entry.h>

#include <disk.h>

#include <fat32.h>

F32\_BPB bpb;

F32\_Info info;

unsigned long \*cacheFsec;

static unsigned short cacheFindex = 0;

int f32\_determineFATType(){

unsigned long rootDirSectors, totalSectors, DataSector;

rootDirSectors = ((bpb.BPB\_RootEntCnt \* 32) + (bpb.BPB\_BytesPerSec-1)) / (bpb.BPB\_BytesPerSec);

totalSectors = (bpb.BPB\_TotSec16) ? (unsigned long)bpb.BPB\_TotSec16 : bpb.BPB\_TotSec32;

info.FATstart = (unsigned long)bpb.BPB\_RsvdSecCnt;

info.BPSector = bpb.BPB\_BytesPerSec;

info.fSecClusters = info.BPSector / 4;

info.FATsize = (bpb.BPB\_FATSz16) ? (unsigned long)bpb.BPB\_FATSz16 : bpb.BPB\_FATSz32;

info.firstDataSector = bpb.BPB\_RsvdSecCnt + bpb.BPB\_NumFATs \* info.FATsize;

info.firstRootSector = info.firstDataSector + (bpb.BPB\_RootClus - 2) \* bpb.BPB\_SecPerClus;

info.clusterCount = (totalSectors - (info.FATstart + (info.FATsize \* bpb.BPB\_NumFATs) + rootDirSectors )) / bpb.BPB\_SecPerClus + 1;

if (info.clusterCount < 4085 && !strcmp(bpb.BS\_FilSysType,"FAT12 ")) return FAT16

else if (info.clusterCount < 65525L && !strcmp(bpb.BS\_FilSysType,"FAT16 "))

return FAT16;

else if (!memcmp(bpb.BS\_FilSysType,"FAT32 ",8)) return FAT32;

else error(0,\_("Can't determine FAT type (label: '%s')\n"), bpb.BS\_FilSysType);}

nt f32\_mount(int image\_descriptor){

int ftype;

d\_mount(image\_descriptor);

/\* Loading BPB \*/

if (d\_readSectors(0, (char\*)&bpb, 1, 512) != 1)

error(0,\_("Can't read BPB !"));

if (debug\_mode) {

int i;

fprintf(output\_stream, \_("(f32\_mount) Bytes per Sector: %d\n"), (short)bpb.BPB\_BytesPerSec);

fprintf(output\_stream, \_("(f32\_mount) Sectors per Cluster: %u\n"), bpb.BPB\_SecPerClus);

fprintf(output\_stream, \_("(f32\_mount) Reserved sectors count: %d\n"), bpb.BPB\_RsvdSecCnt);

fprintf(output\_stream, \_("(f32\_mount) Number of FATs: %d\n"), bpb.BPB\_NumFATs);

fprintf(output\_stream, \_("(f32\_mount) Root entries count: %d\n"), bpb.BPB\_RootEntCnt);

fprintf(output\_stream, \_("(f32\_mount) Media: %x\n"), bpb.BPB\_Media);

fprintf(output\_stream, \_("(f32\_mount) Sectors per track: %d\n"), bpb.BPB\_SecPerTrk);

fprintf(output\_stream, \_("(f32\_mount) Total sectors 32: %ld\n"), bpb.BPB\_TotSec32);

fprintf(output\_stream, \_("(f32\_mount) FAT size 32: %ld\n"), bpb.BPB\_FATSz32);

fprintf(output\_stream, \_("(f32\_mount) FS version major: %d\n"), bpb.BPB\_FSVerMajor);

fprintf(output\_stream, \_("(f32\_mount) FS version minor: %d\n"), bpb.BPB\_FSVerMinor);

fprintf(output\_stream, \_("(f32\_mount) Root clusters: %ld\n"), bpb.BPB\_RootClus);

fprintf(output\_stream, \_("(f32\_mount) Volume ID: %ld\n"), bpb.BS\_VolID);

fprintf(output\_stream, "(f32\_mount) Volume label: '");

for (i = 0; i < sizeof(bpb.BS\_VolLab); i++)

fprintf(output\_stream, "%c", bpb.BS\_VolLab[i]);

fprintf(output\_stream, "'\n");

fprintf(output\_stream, "(f32\_mount) File system type: '");

for (i = 0; i < sizeof(bpb.BS\_FilSysType); i++)

fprintf(output\_stream, "%c", bpb.BS\_FilSysType[i]);

fprintf(output\_stream, "'\n");

}

if ((ftype = f32\_determineFATType()) != FAT32)

error(0,\_("File system on image isn't FAT32, but FAT%d !"),ftype);

if (!(bpb.BPB\_ExtFlags & 0x80))

info.FATmirroring = 1;

else {

info.FATmirroring = 0;

info.FATstart += (bpb.BPB\_ExtFlags & 0x0F) \* info.FATsize;

}

if (debug\_mode) {

fprintf(output\_stream, "(f32\_mount) FAT mirroring: %s\n", (info.FATmirroring)?"yes":"no");}

if ((cacheFsec = (unsigned long \*)malloc(sizeof(unsigned long) \* info.fSecClusters)) == NULL)

error(0,\_("Out of memory !"));

return 0;}

int f32\_mounted(){

if (info.FATstart && d\_mounted()) return 1;

else return 0;}

int f32\_umount(){

info.FATstart = 0;

free(cacheFsec);

d\_umount();

return 0;}

int f32\_readFAT(unsigned long cluster, unsigned long \*value){

unsigned long logicalLBA;

unsigned short index;

unsigned long val;

if (!f32\_mounted()) return 1;

logicalLBA = info.FATstart + ((cluster \* 4) / info.BPSector);

index = (cluster % info.fSecClusters);

if (logicalLBA > (info.FATstart + info.FATsize))

error(0,\_("Trying to read cluster > max !"));

if (cacheFindex != logicalLBA) {

if (d\_readSectors(logicalLBA, cacheFsec, 1, info.BPSector) != 1) error(0,\_("Can't read from image (pos.:0x%lx)!"), logicalLBA);

else cacheFindex = logicalLBA;}

val = cacheFsec[index] & 0x0fffffff;

\*value = val;

return 0;}

int f32\_writeFAT(unsigned long cluster, unsigned long value){

unsigned long logicalLBA;

unsigned short index;

if (!f32\_mounted()) return 1;

value &= 0x0fffffff;

logicalLBA = info.FATstart + ((cluster \* 4) / info.BPSector);

index = (cluster % info.fSecClusters); /\* index in FAT table sector \*/

if (logicalLBA > (info.FATstart + info.FATsize))

error(0,\_("Trying to write cluster > max !"));

if (cacheFindex != logicalLBA) {

if (d\_readSectors(logicalLBA, cacheFsec, 1, info.BPSector) != 1) error(0,\_("Can't read from image (pos.:0x%lx) !"), logicalLBA);

else cacheFindex = logicalLBA; }

cacheFsec[index] = cacheFsec[index] & 0xf0000000;

cacheFsec[index] = cacheFsec[index] | value;

if (d\_writeSectors(logicalLBA, cacheFsec, 1, info.BPSector) != 1)

error(0,\_("Can't write to image (pos.:0x%lx) !"), logicalLBA);

if (info.FATmirroring)

if (d\_writeSectors(logicalLBA + info.FATsize, cacheFsec, 1, info.BPSector) != 1)

return 1;

return 0;}

unsigned long f32\_getStartCluster(F32\_DirEntry entry){

return ((unsigned long)entry.startClusterL + ((unsigned long)entry.startClusterH << 16)); }

void f32\_setStartCluster(unsigned long cluster, F32\_DirEntry \*entry){

(\*entry).startClusterH = (unsigned short)((unsigned long)(cluster & 0xffff0000) >> 16);

(\*entry).startClusterL = (unsigned short)(cluster & 0xffff); }

unsigned long f32\_getNextCluster(unsigned long cluster){

unsigned long val;

if (f32\_readFAT(cluster, &val))

error(0,\_("Can't read from FAT !"));

return val; }

int f32\_readCluster(unsigned long cluster, void \*buffer){

unsigned long logicalLBA;

if (!f32\_mounted()) return 1;

if (cluster > info.clusterCount)

error(0,\_("Trying to read cluster > max !"));

logicalLBA = info.firstDataSector + (cluster - 2) \* bpb.BPB\_SecPerClus;

if (d\_readSectors(logicalLBA, buffer, bpb.BPB\_SecPerClus, info.BPSector) != bpb.BPB\_SecPerClus)

return 1;

else return 0;}

int f32\_writeCluster(unsigned long cluster, void \*buffer)

{

unsigned long logicalLBA;

if (!f32\_mounted()) return 1;

if (cluster > info.clusterCount)

error(0,\_("Trying to write cluster > max !"));

logicalLBA = info.firstDataSector + (cluster - 2) \* bpb.BPB\_SecPerClus;

if (d\_writeSectors(logicalLBA, buffer, bpb.BPB\_SecPerClus, info.BPSector) != bpb.BPB\_SecPerClus)

return 1;

else

return 0;

}

в) определяет положение и размер системных данных BOOT, FAT, ROOT – и выводит их значения на экран.

procedure FileSizeEx(const FileName: string; out Size, SizeOnDisk: UINT);

var

Drive: string;

FileHandle: THandle;

SectorsPerCluster,

BytesPerSector,

Dummy: DWORD;

ClusterSize: DWORD;

SizeHigh, SizeLow: DWORD;

begin

Assert(FileExists(FileName));

Drive := IncludeTrailingPathDelimiter(ExtractFileDrive(FileName));

if not GetDiskFreeSpace(PChar(Drive), SectorsPerCluster, BytesPerSector, Dummy, Dummy) then

RaiseLastOSError;

ClusterSize := SectorsPerCluster \* BytesPerSector;

FileHandle := CreateFile(PChar(FileName), 0, FILE\_SHARE\_READ or FILE\_SHARE\_WRITE or FILE\_SHARE\_DELETE,

nil, OPEN\_EXISTING, 0, 0);

if (FileHandle = INVALID\_HANDLE\_VALUE) then

RaiseLastOSError;

try

SizeLow := Windows.GetFileSize(FileHandle, @SizeHigh);

if (GetLastError <> NO\_ERROR) and (SizeLow = INVALID\_FILE\_SIZE) then

RaiseLastOSError;

Size := UINT(SizeHigh shl 32 or SizeLow);

finally

if (FileHandle <> INVALID\_HANDLE\_VALUE) then

CloseHandle(FileHandle);

end;

SizeLow := GetCompressedFileSize(PChar(FileName), @SizeHigh);

if (GetLastError <> NO\_ERROR) and (SizeLow = INVALID\_FILE\_SIZE) then

RaiseLastOSError;

SizeOnDisk := UINT(SizeHigh shl 32 or SizeLow);

if (SizeOnDisk mod ClusterSize) > 0 then

SizeOnDisk := SizeOnDisk + ClusterSize - (SizeOnDisk mod ClusterSize);

end;

Запустим WinHex, откроем диск [Инструменты-->Открыть диск] Он должен выглядеть приблизительно так:

Offset 0 1 2 3 4 5 6 7 8 9 A B C D E F

00000000 E8 12 01 B9 F0 01 BE 10 7C BF 10 06 57 F3 A4 C3 и..№р.ѕ.|ї..Wу¤Г

00000010 8B 4E 14 83 F9 0E 75 08 8D 5E 07 43 02 07 E2 FB ‹N.ѓщ.u.Ќ^.C..вы

00000020 8C 56 0C 8C 56 0E 75 69 8A 56 10 84 D2 79 62 E8 ЊV.ЊV.uiЉV.„Тybи

00000030 F6 00 BB AA 55 CD 13 72 6F 3B 5E 5C 75 6A D1 E9 ц.»ЄUН.ro;^\ujСй

00000040 73 66 B4 42 C6 46 02 01 EB 66 89 B6 F6 FE 8A 44 sfґBЖF..лf‰¶цюЉD

00000050 04 84 C0 74 0F 3C 05 74 0B 3C 0F 74 07 8A 14 80 .„Аt.<.t.<.t.Љ.Ђ

00000060 E2 80 75 CB 83 C6 10 06 C4 5C 08 89 5E 08 8C 46 вЂuЛѓЖ..Д\.‰^.ЊF

00000070 0A 07 FE 8E F9 FE 75 D2 B0 31 C6 46 D5 50 88 46 ..юЋщюuТ°1ЖFХP€F

00000080 D2 BE 68 07 AC 84 C0 74 08 B4 0E B3 07 CD 10 EB Тѕh.¬„Аt.ґ.і.Н.л

00000090 F3 E8 81 00 88 46 11 BE AE 07 3C 05 75 C6 CD 16 уиЃ.€F.ѕ®.<.uЖН.

000000A0 33 D2 89 56 08 89 56 0A E8 7D 00 72 1B B8 01 02 3Т‰V.‰V.и}.r.ё..

000000B0 BF 05 00 8B DC 56 50 50 32 E4 CD 13 58 8B F5 CD ї..‹ЬVPP2дН.X‹хН

000000C0 13 58 5E 73 03 4F 75 EB B0 32 72 B2 40 8A 66 11 .X^s.Ouл°2rІ@Љf.

000000D0 9E 7B 04 C6 47 02 0E 72 35 75 0C 88 57 40 C4 4E ћ{.ЖG..r5u.€W@ДN

000000E0 08 89 4F 1C 8C 47 1E 79 06 8A 4E 12 88 4F 25 80 .‰O.ЊG.y.ЉN.€O%Ђ

000000F0 C7 02 81 7F FE 55 AA 75 85 81 7F FA CD 19 75 09 З.ЃюUЄu…ЃъН.u.

00000100 C6 47 FA E9 C7 47 FB 94 88 E8 1C 00 FF E4 74 CE ЖGъйЗGы”€и..ядtО

00000110 88 57 24 EB C9 5D 33 C0 8E D8 8E C0 8E D0 BC 00 €W$лЙ]3АЋШЋАЋРј.

00000120 7C 55 BD A2 07 FC FB C3 B4 08 52 06 CD 13 07 33 |UЅў.ьыГґ.R.Н..3

00000130 DB 8A DE 8B 46 0A 33 D2 83 E1 3F F7 F1 91 97 8B ЫЉЮ‹F.3Тѓб?чс‘—‹

00000140 46 08 F7 F7 42 87 CA 3B DA 72 17 43 F7 F3 8A F2 F.ччB‡К;Ъr.CчуЉт

00000150 86 C5 D1 E8 D1 E8 0A C8 D0 CC D0 CC 0A F4 84 E4 †ЕСиСи.ИРМРМ.ф„д

00000160 74 02 B4 41 5B 8A D3 C3 0D 0A 4D 42 52 20 45 72 t.ґA[ЉУГ..MBR Er

00000170 72 6F 72 20 00 0D 0A 00 72 65 73 73 20 61 6E 79 ror ....ress any

00000180 20 6B 65 79 20 74 6F 20 62 6F 6F 74 20 66 72 6F key to boot fro

00000190 6D 20 66 6C 6F 70 70 79 2E 2E 2E 00 00 00 00 00 m floppy........

000001A0 00 00 10 00 01 00 00 7C 00 00 C0 D5 A7 01 00 00 .......|..АХ§...

000001B0 00 00 80 41 00 F4 0E 00 60 C8 67 03 00 00 80 01 ..ЂA.ф..`Иg...Ђ.

000001C0 01 00 0B FE FF 7B 3F 00 00 00 3D A8 DA 00 00 00 ...юя{?...=ЁЪ...

000001D0 C1 7C 0F FE FF FF 7C A8 DA 00 86 FE 79 01 00 00 Б|.юяя|ЁЪ.†юy...

000001E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

000001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA ..............UЄ

00000200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

Первые 1BBh байт занимают код и данные загрузчика, среди которых отчетливо выделяются текстовые строки. По смещению 1BBh расположен 4-хбайтный ID-диска, назначаемый маздаем при запуске Disk Manager'а, поэтому байты 1BBh - 1BEh лучше не трогать:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Offset 0 1 2 3 4 5 6 7 8 9 A B C D E F

000001B0 00 00 80 41 00 F4 0E 00 60 C8 67 03 00 00 80 01 ..ЂA.ф..`Иg...Ђ.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Со смещения 1BEh начинается "Таблица Разделов", представляющая собой массив из 4-х записей типа "Рartition". Каждая partition имеет размер 10h байт и описывает свой логический диск, что позволяет нам создавать до четырёх разделов на каждом HDD. В общем, устройство "Главного Загрузочного Сектора" выглядит так:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Смещение | Размер | Назначение

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

000h Переменный Код загрузчика

1BBh 4h Идентификатор диска

1BEh 10h partition 1

1CEh 10h partition 2

1DEh 10h partition 3

1EEh 10h partition 4

1FEh 2h Cигнатура 55 AAh

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Как видно из дампа MBR, диск имеет 2 раздела, о чём свидетельствуют записи в "Таблице разделов":

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Offset 0 1 2 3 4 5 6 7 8 9 A B C D E F

; Рartition #1

000001B0 80 01 Ђ.

000001C0 01 00 0B FE FF 7B 3F 00 00 00 3D A8 DA 00 ...юя{?...=ЁЪ.

; Рartition #2

000001C0 00 00 ..

000001D0 C1 7C 0F FE FF FF 7C A8 DA 00 86 FE 79 01 Б|.юяя|ЁЪ.†юy.

; Рartition #3

000001D0 00 00 ..

000001E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............

; Рartition #4

000001E0 00 00 ..

000001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Каждая запись "Partition" состоит из:

\* адресов начала и конца раздела,

\* типа раздела (NTFS, FAT16, FAT32...),

\* кол-во секторов в разделе,

\* флага "загруженности" раздела.

Все адреса задаются в формате либо:

\* CHS (Cylinder-Head-Sector - Цилиндр-Головка-Сектор), ..либо

\* LBA (Logical Block Address - Логический Адрес Блока).

Конкретный формат определяется типом раздела (BootID), записанным в 4-ом байте от начала "Partition". У меня "BootID" для первого раздела равен "0Вh", а для второго "0Fh". Типов "BootID" - очень много, поэтому приведем самые/популярные:

Boot ID | Тип раздела

^^^^^^^^^|^^^^^^^^^^^^^^^^^^^^^^^^^^^^

00h Раздел свободен

01h FAT12, CHS

04h FAT16, CHS

05h Расширенный раздел (extended partition), CHS

06h BIGDOS FAT16 раздел (33 Мбайт - 4 Гбайт), CHS

07h NTFS-раздел, CHS

0Bh FAT32 раздел, CHS

0Ch FAT32 раздел с поддержкой расширенной BIOS INT 13h, LBA

0Eh FAT16 раздел с поддержкой расширенной BIOS INT 13h, LBA

0Fh Раздел с поддержкой расширенной BIOS int 13h, LBA

42h Динамический диск, LBA

Если взглянуть на "Таблицу разделов", то получиться приблизительно такая картина (через запятую идут смещения для 1-4 разделов, соответственно). Зелёным обозначены значения:

Смещения | Разм. | Назначение

^^^^^^^^^^^^^^^^|^^^^^^^|^^^^^^^^^^^^^^^

1BE,1CE,1DE,1EE 01h Флаг раздела: 80h - загрузочный, 00h - не загрузочный

1BF,1CF,1DF,1EF 01h Стартовая головка раздела

1C0,1D0,1E0,1F0 01h Стартовый сектор раздела

1C1,1D1,1E1,1F1 01h Младшие биты стартового цилиндра

1C2,1D2,1E2,1F2 01h Идентификатор системы (BootID)

1C3,1D3,1E3,1F3 01h Конечная головка раздела

1C4,1D4,1E4,1F4 01h Конечный сектор раздела

1C5,1D5,1E5,1F5 01h Младшие биты конечного цилиндра

1C6,1D6,1E6,1F6 04h Смещение раздела (сект) относительно начала таблицы разделов

1CA,1DA,1EA,1FA 04h Кол-во секторов в разделе

[\* Мои значения для раздела #1\*]

Offset 0 1 2 3 4 5 6 7 8 9 A B C D E F

000001B0 80 01 Ђ.

000001C0 01 00 0B FE FF 7B 3F 00 00 00 3D A8 DA 00 ...юя{?...=ЁЪ.

01BE = 80h = загрузочный раздел

01BF = 01h = раздел начинается с головки 1

01C0 = 01h = раздел начинается с сектора 1

01С1 = 00h = раздел начинается с цилиндра 0

01С2 = 0Bh = FAT32 раздел, CHS

01C3 = FEh = раздел кончается головкой 254

01С4 = FFh = раздел кончается сектором 255

01С5 = 7Bh = раздел кончается цилиндром 123

01С6 = 00 00 00 3Fh = начало с 63 сектора от таблицы разделов

01CA = 00 DA A8 3Dh = секторов в разделе 14.329.917 (..\* 512 = 6,8 Gb)

Этот первый/загрузочный сектор необходимо зарезервировать в WinHex'e с расширением \*.bin, ..так, про-запас. Если что пойдёт "не по-плану", то всегда сможем восстановить его, хотя есть и "fixmbr" с установочного диска.

Чтобы продолжить, необходимо ознакомимться с интерфейсом INT13, обслуживающим диски. Этот сервис предоставляет прямой доступ к жёсткому диску.

Подфункции: 00H сброс контроллера 08H дать парам диска 10H проверить готовность

01H дать статус 09H иниц табл парам 11H рекалибрация

02H читать секторы 0aH длинное чтение 14H диагностика

03H писать секторы 0bH длинная запись 15H дать тип диска

04H верификация 0cH искать цилиндр 16H изменить статус

05H форматир дорожку 0dH альтерн сброс 17H уст тип диска

По идее, из всех этих функций нам нужны только две - читать/писать сектор (02/03h соответственно), которые имеют следующий формат:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AH | Сервис

^^^^^|^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

02h Читать секторы

Вход: DL = номер диска (80h=HDD0; 81h=HDD1)

DH = номер головки чтения/записи

CH = номер дорожки (цилиндра)(0-n)

CL = номер сектора (1-n)

AL = число секторов (в сумме не больше чем один цилиндр)

ES:BX => адрес буфера вызывающей программы

Выход: флаг = 1 при ошибке, и код ошибки диска в AH.

ES:BX буфер содержит данные, прочитанные с диска

^^^^^^|^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

03h Писать секторы

Вход: (аналогично подфункции 02H)

ES:BX => данные, записываемые на диск.

Выход: флаг = 1 при ошибке, и код ошибки диска в AH.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Исходя из таблицы выше можно написать код загрузчика.

Номер функции заносится в регистр AH. В случае чтения он равен двум. Регистр AL отвечает за количество обрабатываемых секторов. Поскольку собираемся читать по одному сектору за раз, занесем сюда единицу. Регистр DH хранит номер головки, а DL - номер привода (80h - первый жесткий диск, 81h - второй, и т.д.).

5 младших битов регистра CL задают номер сектора, оставшиеся биты регистра CL и 8 битов регистра CH определяют номер цилиндра, который хотим прочитать. Регистровая пара ES:BX указывает на адрес буфера-приёмника. Вот, собственно говоря, и всё. После выполнения команды INT 13h считываемые данные окажутся в буфере, а если произойдет ошибка (например, головка споткнется о BAD-сектор) BIOS установит флаг переноса (carry flag) и мы будем вынуждены либо повторить попытку, либо вывести ошибку на экран.





