INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

COMPUTER SYSTEMS LABORATORY

Fall 2011

Stage 0

Synopsis

During this stage you will construct a virtual hard disk image with the following arrangement:

- Two primary partitions
  - Partition #1, formatted as a FAT16 file system, is a bootable FreeDOS system (from Stage -1)
  - Partition #2, formatted as a MINIX version 2 file system, holds a single file (`lsc.sys`) in its root directory

We will keep the master boot record (MBR) from FreeDOS, as it will load a volume boot record (VBR) from any partition regardless of its type.

In addition, you will write the code for the volume boot record (VBR) of partition #2, such that it loads and runs the contents of `/lsc.sys`.

At the end of this stage, you should know:

- how can distinct operating systems coexist in the same hard disk;
- what is a partition table and how it is organized and used;
- the difference between master and volume boot record and what their roles are in the bootstrap of an operating system;
- the typical organization of a UNIX file system and how to navigate programmatically in its internal structures;
- how a PC gets to the point where the initial code of some particular operating system is loaded and run.

In addition, you will know the tools needed to:

- create, setup, manipulate, and inspect a virtual hard disk image;
- mount, format, and access the individual partitions of a virtual hard disk image from your system.

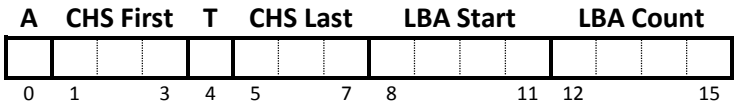
Accessing partitions on the virtual hard disk image

In Stage -1, you have built a virtual hard disk image with two partitions. These were created with `FDISK.EXE`, a utility program in FreeDOS, running inside a Bochs emulation session.

In this section, you will access the same virtual hard disk image from the Ubuntu/Linux development environment. This is fundamental, as it allows you to inspect and manipulate the contents of the virtual disk.

Observing the partition table in the virtual hard disk

The partition table is located at the end of the master boot record, which occupies the first physical sector of the hard disk. The table starts at offset 446 and has a length of 4 x 16 bytes. Each of these 16 bytes describes one disk partition, in the following way:



A	Active partition: 0x80; else: 0x00 [Note: at most one partition active at any time.]
CHS First	CHS absolute address of first sector in partition
T	Partition type [FAT 12, FAT 16, FAT 32, MINIX, Linux, NTFS, …]
CHS Last	CHS absolute address of last sector in partition
LBA Start	Linear absolute address of first sector in partition
LBA Count	Total number of sectors in partition

Figure 1 – Partition table entry format

A partitioned bootable disk should have exactly one partition table entry marked as active. Partition limits are indicated twice, using CHS and LBA addressing. LBA fields contain plain little-endian values, while CHS fields have a dour layout:

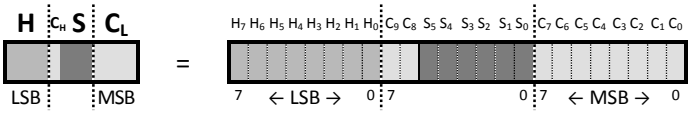


Figure 2 – CHS field format

Partition type indicates the partition's purpose, usually identifying the kind of file system it holds. See a list of partition types in: *http://www.win.tue.nl/~aeb/partitions/partition_types-1.html*

Use **`hd`** to inspect the current partition table on the virtual hard disk MBR. See the usage guide of **`hd`** in the manual page (type **`man hd`** at the shell prompt). Options **`-s`** and **`-n`** are particularly useful to select the exact 64 bytes that constitute the partition table.

Partition	Active?	Type	CHS First			CHS Last			LBA Start	LBA Count
			C	H	S	C	H	S		
#1										
#2										
#3										
#4										

Table 1 – Partition table after installing FreeDOS

Accessing the virtual hard disk from Linux via a loop device

Several UNIX variants allow accessing a regular file as if it were a physical disk. In Linux, the entities that make this possible are called *loop devices*. These appear under `/dev`, which is the standard directory for file system entries that represent devices.

1. Use the following command to find the first unused loop device:
   ```
   sudo losetup -f
   ```
2. Assuming that the previous command returned `/dev/loop0`, attach the virtual hard disk image to this loop device:
   ```
   sudo losetup /dev/loop0 ~/lsc/disks/lsc-hd63-flat.img
   ```
3. Finally, use Linux's `fdisk` to check the partition table via the loop device:
   ```
   sudo fdisk -lu /dev/loop0
   ```

The information printed by `fdisk` should be consistent with that of Table 1.

If everything is ok, run `fdisk` (this time without `-lu`) on `/dev/loop0` and change the partition type of the second partition to 0x81, indicating a MINIX file system.

Accessing the individual partitions in the virtual hard disk from Linux

Besides being able to use the virtual hard disk as a whole, it's highly convenient to access its individual partitions. We will be using two more loop devices for that purpose, one for each partition.

The first step is to calculate the start offset and length of the two partitions, in bytes. That should be a trivial task, given Table 1 and the fact that sectors have a fixed size of 512 bytes.

Partition	Start offset	Length
#1		
#2		

Table 2 – Start offset and length of partitions, in bytes

Now set up two additional loop devices (possibly `/dev/loop1` and `/dev/loop2`) for the partitions on the virtual hard disk image. See the manual page of `losetup` to find how to indicate the start offset and the length of the zones inside the image file that are to be accessed via loop devices.

To have some indication that the loop device corresponding to the first partition is properly set up, issue the following command and inspect its result:

```
sudo fsck.msdos -v /dev/loop1
```

Create a MINIX (version 2) file system in the second partition with:

```
sudo mkfs.minix -v /dev/loop2
```

Now check the new file system and confirm that the number of blocks (of 1024 bytes each) is half the number of sectors (of 512 bytes each) in Table 1:

```
sudo fsck.minix -vfs /dev/loop2
```

Finally remove all loop devices, as we'll use them indirectly from now on:

```
sudo losetup -d /dev/loop2
sudo losetup -d /dev/loop1
sudo losetup -d /dev/loop0
```

Mounting the virtual hard disk partitions

Unlike Windows, UNIX systems don't have a separate file system hierarchy for each disk. In UNIX, there is a single root for the file hierarchy and secondary file hierarchies can be *mounted* and appear under selected directories of the base hierarchy.

In this section you will mount the two partitions from the virtual hard disk in your system, making them easily usable and inspectable.

First, create subdirectories to serve as mount points:

```
sudo mkdir /mnt/lsc
sudo chown isel.isel /mnt/lsc
mkdir /mnt/lsc/hd0p1
mkdir /mnt/lsc/hd0p2
```

Then mount the two partitions on the respective subdirectories:

```
sudo mount ~/lsc/disks/lsc-hd63-flat.img /mnt/lsc/hd0p1
       -o loop,offset=_ooo_,sizelimit=_sss_,
          nodev,noexec,rw,users,uid=1000,gid=1000

sudo mount ~/lsc/disks/lsc-hd63-flat.img /mnt/lsc/hd0p2
       -o loop,offset=_ooo_,sizelimit=_sss_,
          nodev,noexec,rw,users
```

It's probably more comfortable to change the owner of the root directory of the second partition (this only has to be done once):

```
sudo chown isel.isel /mnt/lsc/hd0p2
```

To check that the mount is working properly, create a new directory on the DOS partition and copy a file into there:

```
mkdir /mnt/lsc/hd0p1/extra
cp extra.zip /mnt/lsc/hd0p1/extra/
ls -l /mnt/lsc/hd0p1/extra/
```

Checkpoint:

The second partition now has a valid MINIX (version 2) file system with no files. Both partitions are accessible under `/mnt/lsc/hd0p1` (part. #1: FAT16) and `/mnt/lsc/hd0p2` (part. #2: MINIX v2). The MBR is still the one from FreeDOS.

You have used `hd` to inspect the contents of the virtual disk image and seen how to use loop devices to access the virtual hard disk as if it were a physical device.

This is the time to create scripts that ease the task of mounting and unmounting the partitions. It is also the moment to check that DOS is still booting: run Bochs, let it boot into DOS, change the current directory to `C:\extra`, unzip `extra.zip`, and execute the file with `.com` extension that appears.

Making a volume boot record

In a common scenario, the purpose of the master boot record (MBR) is to select the active partition and load its volume boot record (VBR), while the purpose of the VBR is to load bootstrap code specific of the operating system installed in its partition. For the second partition of the virtual hard disk, its VBR will have to find that bootstrap code in the MINIX file system.

MINIX file system

The MINIX file system is the simplest version of a UNIX-like file system available in Linux. We use it in its second version, as that is the most recent version fully supported by Linux kernel and tools. Unfortunately, the specification for this version of the MINIX file system is only available in the outdated 2nd edition of Andrew Tanenbaum's book about MINIX, *Operating Systems: Design and Implementation* (available in the school's central library).

A volume formatted with a MINIX 2 file system is logically organized in blocks, each containing a fixed number of physical disk sectors. The layout of a MINIX 2 file system has 6 distinct areas, each occupying an integral number of blocks.

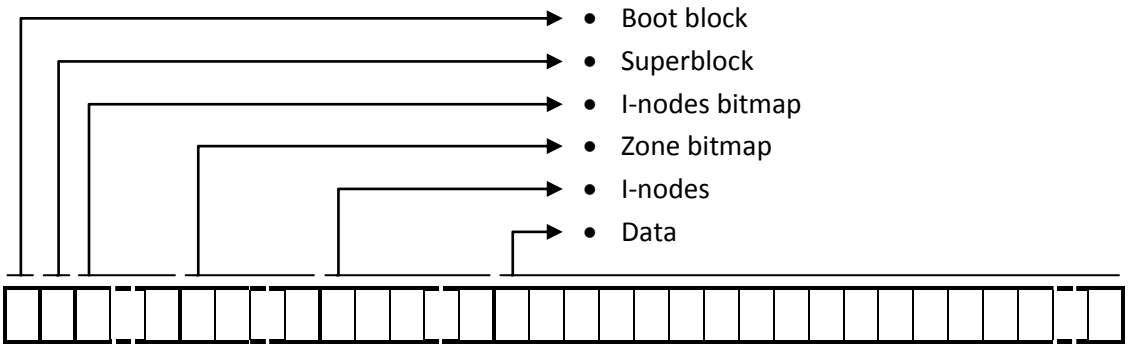


Figure 3 – Layout of a MINIX 2 formatted volume

The **boot block** takes exactly one block and contains the volume boot record. The **superblock** takes another block and contains information (see Table 4) about the layout of the file system.

Each i-node in the **I-nodes** area occupies 64 bytes and is used to hold metadata (see Table 5) about one file system entry (file, directory, …). The number of blocks used for i-nodes is decided at format time and is stored in the superblock.

The **Data** area is usually the largest one and holds the contents of files and directories. Its blocks are grouped into ***zones***, with each zone representing 2^n blocks, for some $n >= 0$.

The **I-nodes bitmap** area has one bit per existing i-node and is used to find free i-nodes in the I-nodes area. Similarly, the **Zone bitmap** has one bit for each zone in the Data area and is used to find free zones in it.

Exercise

Given the Linux header file include/linux/minix_fs.h to complement the information from the previous section, use the `hd` utility to inspect the internal organization of the second partition in the virtual hard disk and fill in the following tables:

Block size (in bytes)	
Zone size (in bytes)	

Table 3 – Second partition block and zone sizes

Superblock		
s_ninodes		(number of i-nodes)
s_nzones		(number of zones, if MINIX 1)
s_imap_blocks		(number of i-node bitmap blocks)
s_zmap_blocks		(number of zone bitmap blocks)
s_firstdatazone		(first data zone)
s_log_zone_size		($\log_2$ (blocks/zone))
s_max_size		(maximum file size)
s_magic		(magic number)
s_state		(MINIX_VALID_FS or MINIX_ERROR_FS)
s_zones		(number of zones, if MINIX 2)

Table 4 – Second partition superblock

I-node of root directory		
i_mode		(file type and *rwx* permissions)
i_nlinks		(number of directory entries pointing at this file)
i_uid		(user id of the file owner)
i_gid		(group id of the file owner)
i_size		(number of bytes in the file)
i_atime		(access time)
i_mtime		(modification time)
i_ctime		(status change time)
i_zone[0]		(first data zone)
i_zone[1]		(second data zone)

Table 5 – Root directory i-node (incomplete)

Root directory entries		
#	**inode**	**name**
0		
1		
2		

Table 6 – Root directory entries

Find some text file with 1KiB to 2KiB and copy it to the root directory of the MINIX file system. Inspect the new entry in the root directory (register it in Table 6 too) and follow file system structures to find what disk sectors are being used to store the actual file content.

Checkpoint:

You have seen how to find the root directory of a MINIX 2 file system, how to inspect its content to find a particular file, and how to get to the actual disk sectors that store the content of that file. That is precisely what your volume boot record will have to do. So, you are now ready to start devising the structure of the source code for it.

Reading disk sectors

BIOS services for reading and writing disk sectors are available via software interrupt INT 13h. To select a particular service, register AH should be loaded with a service identifier before issuing the interrupt. A number of older services accept disk sector addresses in the outdated CHS format, while newer services (usually known as "INT 13h Extensions") use LBA addresses.

The service for reading disk sectors using LBA addresses is identified by code 42h, commonly known by a full codename of "INT 13h AH=42h".

To invoke this service, you will need an auxiliary data structure, called a *Disk Address Packet* (DAP), indicating which sectors should be loaded into what memory address:

offset	length	Description
0	1	length of DAP (allows future extensions): probably 16
1	1	-- unused (set it to 0) --
2	1	**number of sectors** to read, up to 127
3	1	-- unused (set it to 0) --
4 e 5	2	**destination offset** of memory buffer
6 e 7	2	**destination segment** of memory buffer
8 a 15	8	LBA address of the **first sector** to read

When invoking this BIOS service, use the following arguments:

- AH = 0x42
- DL = *drive id* (use the value present in DL when BIOS jumped into 0x07C00)
- DS = segment of DAP's memory location
- SI = offset of DAP's memory location

After a successful invocation:

- CF (*carry flag*): should be reset
- AH = 0
- DAP[2]: number of sectors successfully read

For additional details, check http://www.ctyme.com/intr/int.htm .

Exercise

Write a volume boot record that lists the names of the entries present in the first zone of the root directory.

Checkpoint:

You're able to read data from the MINIX file system in the (virtual) hard disk, which means you're now ready for the final task.

Final Exercise

Write a volume boot record (VBR) for the second partition of the virtual hard disk. This VBR loads `/lsc.sys` to memory address 0x10000 and then jumps into that address.

NOTE: you may consider only the first seven zones when reading the root directory and up to the first indirection when reading `/lsc.sys`