

PROPERTIES OF PARTITION TABLES

By unknown

WHY PARTITIONS?

The partition table of a disk cuts it into 'logical disks'. There are several reasons for wanting to do this. DOS does not support filesystems larger than 2 GB, so partitioning is required to break this '2 GB barrier'. Different partitions may carry different operating systems or different filesystems (FAT, HPFS, NTFS, ext2) to be used by one operating system. Sometimes small partitions are used for special purposes (OS/2 Boot Manager uses a small partition for itself, various laptops have a 'hibernation' partition where the state of the system is stored when it goes asleep). Some 'reliable' systems have backup partitions. For backup purposes, say to tape, it is often convenient to have partitions of a size such that the entire partition can be written to a single tape.

It is a good idea to keep your own things (say under /home) and privately installed packages (say under /usr/local) separate from the software installed from a distribution. In case these are on a different partition, it is easier to do a complete reinstall (or switch to a different distribution) without losing your own stuff.

For well-designed systems it is often possible to have all basic system software on a read-only partition, thus diminishing the probability of corruption and saving backup time. There is also a security aspect; for example on a Unix system one might mount all filesystems other than the root filesystem 'nosuid, nodev', and have /tmp, /home, /var not on the root filesystem, to minimize the possibility that some suid program is tricked into overwriting a vital system file via a hard link to it.

Finally there is the old BIOS problem that can make it impossible to boot a system that lives past cylinder 1024. This may mean that one has to have a partition that ends before the 1024 cylinder limit where the stuff needed at boot time is stored.

WHAT DOES A PARTITION TABLE LOOK LIKE?

One may have an arbitrary number of partitions on a disk. However, the Master Boot Record (MBR, sector 0 of the disk) only holds descriptors for 4 partitions, called the primary partitions. Usually the BIOS can boot only from a primary partition. (Of course it can boot a boot loader that itself is able to access nonprimary partitions or other disks.) The descriptors for the remaining partitions, called logical partitions, are scattered along the disk in a linked list of partition table sectors, starting with the MBR.

Each partition table sector contains 4 partition descriptors. A partition descriptor may be of type 05 (DOS extended partition), 0f (W95 extended partition), 85 (Linux extended partition), or c5 (DRDOS/ secured extended partition), in which case it points to another partition table sector. In this way, we obtain a quaternary tree of partitions. Linux accepts 85 as a synonym for 05 - this is useful if one wants to have extended partitions past the 1024 cylinder limit (to prevent DOS fdisk from crashing or hanging). Windows 95 uses 0f for LBA mapped extended partitions. Thus, an extended partition is not a partition containing data, but is a box containing other partitions. Nevertheless, the partition table sector that starts an extended partition has enough room left to contain a boot loader like LILO, so that it is possible to boot an extended partition.

Most operating systems severely restrict the accepted trees. Usually branching is not allowed, and one gets a linear chain of partition table sectors. Linux will accept several extended primary partitions.

PARTITION DESCRIPTORS

A partition table entry is 16 bytes long and contains 6 items (not listed in order). 1. A byte that is 0x80 or 0 denoting 'bootable' or not. The standard DOS MBR will not boot a partition unless it is the

PROPERTIES OF PARTITION TABLES

By unknown

unique bootable primary partition. For nonprimary partitions this byte is unused. 2. A byte that gives the type. 3. A 4-byte starting sector number. 4. A 4-byte length (in sectors). 5. A 3-byte starting sector given in C/H/ S (cylinder/head/sector) format. 6. A 3-byte final sector given in C/H/S format. Linux only uses items 2,4, and hence is not interested in the 'geometry' of the disk, and can use disks with up to 2^{33} sectors (4 TB). DOS uses 5-6 instead of 3-4, and this leads to the well-known problems with geometry, with the 1024 cylinder limit, the 500 MB limit, the 8 GB limit. For some details, see the large disk HOWTO .

For an extended partition, only the first sector is important - it contains the descriptors for its logical partitions. There are various conventions about how the descriptor of an extended partition (different from the outer one) should look like. There is the paradigm of 'nested boxes', where each extended partition covers a disk area containing all the logical partitions inside. There is also the paradigm of 'chained boxes', where each extended partition (except possibly the outer one) just contains the next logical partition. I don't know which systems follow which paradigms. (David A. Burton <dburton@burtonsys.com> reports that System Commander uses the nested style.) However, for the outer (primary) extended partition it is common to contain all logical partitions inside (i.e., have a start and length field that describes a piece of the disk that contains all logical partitions). Of course the 'chained boxes' paradigm is more flexible since it allows logical partitions with a primary partition in between.

PARTITION HIDING

The OS/2 Boot Manager does not want you to have more than one primary DOS partition (MS-DOS itself does not mind), and will change the type from 01, 04, 06, 07 to 11, 14, 16, 17.

Also other programs or systems use this 'partition hiding'. For example, System Commander will OR the type with 0x10, changing the Linux 83 into the Amoeba 93.

CHS vs LBA

Some partition IDs imply a particular method of disk access. In particular, IDs 0c, 0e, 0f (the LBA versions of 0b, 06, 05) go with partition table entries that have C/H/S = 1023/255/63 and expect access via the extended INT-13 functions (AH=4x) of the BIOS.

LOGICALLY SECTORED FAT

Some systems use a filesystem that is fully compatible with a standard FAT12 or FAT16 partition, except for using a sector size larger than the usual 512 bytes, up to 8192 bytes. This is what is meant by "logically sectored FAT" in the above.

Logically sectored FATs have been a way to circumvent the dreaded 32 MB partition size limit before the introduction of DOS 3.31. Since the count of sectors was restricted to 16-bit on FAT16 (type 04h) the only way to grow the partition above the 32 MB limit in a reasonably compatible fashion was to increase the sector size instead. Physical sectors at ROM BIOS INT 13h level are always 512 bytes in size, but other devices may require support for other sector sizes in the operating system. Hence, when DOS logs in drives during bootstrap it will record the sector size values indicated in each partition it finds and if it is larger than the previously recorded value, it will slide up the maximum supported sector size to the found value. Very old DOS versions seem to have started with an initial value of 128 (showing some CP/M heritage here), but recent DOS versions use an initial value of 512 bytes. Once DOS has logged in all drives (including those not represented on INT 13h level, for example, SCSI disk, RAM disk or such), it will set up its internal buffering logic to use the maximum sector size found. This mechanism is present in all DOS versions (although it was partially broken in DOS 5.0 - 6.22).

PROPERTIES OF PARTITION TABLES

By unknown

WHAT DOES FDISK /MBR DO?

People often recommend the undocumented DOS command FDISK /MBR to solve problems with the MBR. This command however does not rewrite the entire MBR - it just rewrites the boot code, the first 446 bytes of the MBR, but leaves the 64-byte partition information alone. Thus, it won't help when the partition table has problems. Moreover, it can be dangerous to restore the boot code to its original state: if the cause of the problems was a boot sector virus, then vital information may have been stored elsewhere by the virus, and killing the virus may mean killing access to this information. (For example, the stoned.empire.monkey virus encrypts the original MBR to sector 0/0/3.) However, people who want to uninstall LILO, and do not know that LILO has a -u option, can use FDISK /MBR for this purpose.

In a Linux environment, one can wipe all of the MBR with a command like "dd if=/dev/zero of=/dev/hda count=1 bs=512". If only the boot code must be removed, but not the partition table, then "dd if=/dev/zero of=/dev/hda count=1 bs=446" will do. Be very careful with such commands. Usually one regrets them later.

STRUCTURE OF THE MBR - OS ADDITIONS

As we saw, the structure of the MBR (Master Boot Record, sector 0) is as follows: First 446 bytes boot loader code, then 64 bytes partition table (starting at offset 0x1be = 446), finally 2 bytes signature 0xaa55.

Just before the partition table some operating systems save some interesting stuff. For example, DRDOS stores a password starting at offset 0x1b6.

Windows NT stores a 4-byte "disk signature" or "volume ID" starting at offset 0x1b8. It is used to map drive letters to disks: in the HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices registry item the drive letter is coupled with this disk signature. It is used as a disk label to map disk info to disks in the HKEY_LOCAL_MACHINE\SYSTEM\DISK registry item. This signature is generated by the Disk Administrator when it initializes the disk, unless there already was a nonzero value there.

Grub had a 4-byte stage2 start address at 0x1b8, and a 2-byte version at 0x1bc, but recent versions preserve 0x1b8-0x1bd. Also LILO v20 and later preserves this area.

Some operating systems are reported to have 8 instead of 4 partition descriptors in the MBR. Cf. AST DOS under 14 and NEC DOS under 24 above.

THE ADVANCED ACTIVE PARTITION OF PTS

As mentioned above, the DOS MBR boot code will boot the (unique) primary partition that has been marked active. Usually, in a multi-boot situation, the boot manager toggles the active bit of the partition that is to be booted. PTS chose a different solution.

Matthias Paul writes: "So far the only DOS being able to boot out of a logical drive in an extended partition is PTS-DOS by use of so called "Advanced Active Partition" entries in the MBR. In order to remain as compatible as possible with existing DOS standards, this works a little bit different and requires a special 5th partition entry in front of the other four entries in the MBR and corresponding AAP-aware MBR bootstrap code. If the MBR contains a special AAP signature and this special entry exists and is flagged bootable, the MBR will use this instead of one of the other four entries. The entry may either point to the bootsector of a logical drive or to a 512 bytes long file (with system-attribute, so it won't be moved around during disk defragmentation) somewhere inside the

PROPERTIES OF PARTITION TABLES

By unknown

filesystem, which makes up a boot sector (same "IBM" signature, same load address, same register interface). In contrast to the usual MBR code, this MBR code interprets the boot flag byte as physical drive unit (80h..FEh), instead of using it only as a active flag (80h or 00h in older DOS issues or bit 7 set or cleared in newer DOS issues). This way, the AAP MBR could even load a boot sector from other than the first harddisk."

NAMING

DOS uses drive letters A: and B: for floppy disk drives, and assigns drive letters C: ... Z: in the order: first all primary DOS partitions on the first disk, then all primary DOS partitions on the second disk, ..., then all logical DOS partitions on first disk, etc. DOS will stop investigating logical partitions in a given extended partition as soon as a non-DOS partition is encountered. (DOS recognizes partition types 1, 4, 6 and 5 for extended.)

Systems like Windows 95, Windows 98, Windows NT, Windows 2000 and OS/2 follow a similar convention, but recognize different partition types. Thus, a drive can have a different drive letter for each of these operating systems. For details, see the Microsoft KnowledgeBase, e.g. Drive letters in Windows NT , Drive letters in Windows 2000 for unsupported partition types .

LIMITS

The partition table describes the location of partitions both in 1-dimensional ('LBA') and in 3dimensional (CHS) form. The former is easy enough, but for the latter one needs to know the disk geometry. Note that these days this geometry is entirely fake, and different systems use different faked geometries for the same disk, giving lots of problems. (For example, a modern disk may have 2 or 4 heads, but will probably report 15 or 16 heads to the BIOS, which in turn may report 255 heads to DOS or Windows.)

ATA Specification (for IDE disks) - the 137 GB limit

At most 65536 cylinders (numbered 0-65535), 16 heads (numbered 0-15), 255 sectors/track (numbered 1-255), for a maximum total capacity of 267386880 sectors (of 512 bytes each), that is, 136902082560 bytes (137 GB).

BIOS Int 13 - the 8.4 GB limit

At most 1024 cylinders (numbered 0-1023), 256 heads (numbered 0-255), 63 sectors/track (numbered 1-63) for a maximum total capacity of 8455716864 bytes (8.4 GB). This is a serious limitation today. It means that DOS cannot use present day large disks.

The DOS 528 MB limit

If the same values for c,h,s are used for the BIOS Int 13 call and for the IDE disk I/O, then both limitations combine, and one can use at most 1024 cylinders, 16 heads, 63 sectors/track, for a maximum total capacity of 528482304 bytes (528MB), the infamous 504 MB limit (if one takes $M=2^{20}$). This was already a problem many years ago, and all kinds of software, firmware and hardware solutions were invented. On the software side, there are Disk Managers, that circumvent the BIOS and go directly to the hardware. On the firmware side there are translating BIOSes, that use one geometry when talking to the disk, and another one when talking to the user program. (At best, this again allows access to 8.4 GB.) On the hardware side, there is LBA disk access, that no longer uses (c,h,s).

The 2.1 GB limit

PROPERTIES OF PARTITION TABLES

By unknown

Some older BIOSes only allocate 12 bits for the field in CMOS RAM that gives the number of cylinders. Consequently, this number can be at most 4095, and only $4095 \times 16 \times 63 \times 512 = 2113413120$ bytes are accessible.

The 3.2 GB limit

There was a bug in the Phoenix 4.03 and 4.04 BIOS firmware that would cause the system to lock up in the CMOS setup for drives with a capacity over 3277 MB.

The 4.2 GB limit

Simple BIOS translation (ECHS=Extended CHS, sometimes called 'Large disk support' or just 'Large') works by repeatedly doubling the number of heads and halving the number of cylinders shown to DOS, until the number of cylinders is at most 1024. Now DOS and Windows 95 cannot handle 256 heads or more, and in the common case that the disk reports 16 heads, this means that this simple mechanism only works up to $8192 \times 16 \times 63 \times 512 = 4227858432$ bytes (with a fake geometry with 1024 cylinders, 128 heads, 63 sectors/track). Note that ECHS does not change the number of sectors per track, so if that is not 63, the limit will be lower.

The 7.9 GB limit

Slightly smarter BIOSes avoid the previous problem by first adjusting the number of heads to 15 ('revised ECHS'), so that a fake geometry with 240 heads can be obtained, good for $1024 \times 240 \times 63 \times 512 = 7927234560$ bytes.

The 8.4 GB limit

Finally, if the BIOS does all it can to make this translation a success, and uses 255 heads and 63 sectors/track ('assisted LBA' or just 'LBA') it may reach $1024 \times 255 \times 63 \times 512 = 8422686720$ bytes, slightly less than the earlier 8.4 GB limit because the geometries with 256 heads must be avoided. (This translation will use for the number of heads the first value H in the sequence 16, 32, 64, 128, 255 for which the total disk capacity fits in $1024 \times H \times 63 \times 512$, and then computes the number of cylinders C as total capacity divided by $(H \times 63 \times 512)$.)

The 33.8 GB limit

Large disks report 16 heads, 63 sectors/track and 16383 cylinders. Many BIOSes compute an actual number of cylinders by dividing the total capacity by 16×63 . For disks larger than 33.8 GB this leads to a number of cylinders larger than 65535. Now the BIOS crashes or hangs. The solution is to upgrade the BIOS. If that is impossible, it sometimes helps to take the disk out of the BIOS, but that won't work if one has to boot from the disk, and may also fail because the BIOS already hangs during initial probing. Usually one can use a jumper to make the disk appear smaller. Also many operating systems have problems - only the most recent versions work with these disks.

The 137 GB limit

As already noted, the old ATA specification does not allow access to all of a disk that is larger than 137 GB. Indeed, it uses only 28 bits to specify a sector number. However, ATA-6 defines an extension with 48-bit sector number. The first disks needing the extension were Maxtor 160 GB disks, that came to market in Fall 2001.

For another discussion of this topic, see [Breaking the Barriers](#) , and, with more details, [IDE Hard Drive Capacity Barriers](#).

PROPERTIES OF PARTITION TABLES

By unknown

Hard drives over 8.4 GB are supposed to report their geometry as 16383/16/63. This in effect means that the 'geometry' is obsolete, and the total disk size can no longer be computed from the geometry.

DETAILS FOR VARIOUS OPERATING SYSTEMS

Early MSDOS filled the partition table starting at the end. In particular, in the case of only one partition, the descriptor was stored in the fourth primary slot. These days DOS FDISK starts at the beginning, but other systems, like Unixware, still start at the end. Also lomega writes the single partition of a ZIP disk in the last entry (so that it has to be mounted as /dev/sda4 or /dev/hdc4 or so).

MSDOS 6.22 FDISK creates the four entries in the partition table sector that starts an extended partition as 1. a data partition (or empty), 2. the next extended partition, 3. and 4. empty. (But old versions of MSDOS start at the end, and first fill entry 4.) If the first logical partition (that is not the last one) is removed, only the link in position 2 remains. An extended partition table sector can describe only a single data partition (the first one encountered). When reading a table, FDISK accepts the entries in any order and position, but it will write the sector normalized as described.

DRDOS on the other hand expects zero to four entries in an extended partition table sector. Data partitions, possibly followed by the link to the next extended partition. Thus, this link is always the last significant entry, and will be the first entry if there is no data partition (because it has been deleted).

Many systems are willing to accept more than two nonempty parts in an extended partition, but will not create such themselves.

It is rumored that the outer extended partition should be the 4th in the MBR, but I don't know any systems that have this restriction. DRDOS FDISK always puts the extended partition in the fourth entry no matter how many other entries you may have.

MSDOS fdisk shows 4 primary partitions, and of the logical partitions only those that have a DOS type (1, 4 or 6). It will list the type of a logical partition as 'Unknown' if the partition is not formatted.

It is rumored that DRDOS ignores the high-order bit of the ID (and that is the reason for the additional Linux IDs 41, 42, 43), but I don't know whether that is true (and for which versions of DRDOS). It is also rumored that DRDOS will write 1 sector past the end of a partition - I have never seen this either. Confirmation? It is however true, that DRDOS fdisk only looks at the last 4 bits when printing a type, so that types 11, 21, etc are printed as DOS 2.0, but such types are not acceptable for DRDOS itself.

The OS/2 Warp fdisk is very instable, and hangs or crashes with general protection fault as soon as the partition table is somewhat unusual, cf. Cannot set an installable partition with FDISK .

The Windows NT Disk Administrator will corrupt your disk when it writes a signature on a disk with two or more logical partitions. See Disk Administrator Corrupts Partitions .

The use of Win95/Win98 FDISK in a mixed system is dangerous. It will delete a non-FAT logical partition when you had actually told it to delete a FAT partition somewhere farther down the chain of logical partitions. See Cannot View NTFS Logical Drive After Using FDISK.

PROPERTIES OF PARTITION TABLES

By unknown

The system partition in Windows NT 4 must be contained in the first 7.8 GB of the disk (or less, in case the BIOS geometry does not have 255 heads and 63 sectors/track; the actual restriction is that all of it must be accessible using BIOS Int 13). It must not be larger than 4 GB because Windows NT 4 first installs into a FAT16 partition and then converts it into NTFS during the second phase of the installation. It must start before the 4 GB mark (bug fixed in Service Pack 5). See Windows NT 4.0 Supports Maximum of 7.8-GB System Partition and Windows NT Partitioning Rules During Setup and Boot Partition Created During Setup Limited to 4 Gigabytes and Windows NT Does Not Boot to a Partition That Starts More Than 4 GB into Disk.

Windows NT and Windows 2000 use for SCSI disks whatever the BIOS says (usually C/H/S=C/255/63) for the boot drive, and C/64/32 for all other SCSI drives. See How Windows NT Handles Drive Translation.

Windows 2000 seems to require that the partition order agrees with the disk order.

The OS/2 fdisk writes some strange length in the descriptor of the last extended partition. This is probably a bug. OS/2 fdisk fails to update the length of the (outer) extended partition when a primary partition is created in the free space (space not used by a logical partition) at the end of this extended partition. This can lead to overlapping partitions.

OS/2 FDISK does not know about type f, but accepts DOS Extended Partitions extending beyond cylinder 1023. When some other partition handler, like Partition Magic 4.0, changes the type of a large extended partition from 05 to 0f, OS/2 loses access.

OS/2 Boot Manager keeps a private copy of the partition table data. This leads to problems when changing the partition table with 3rd party tools.

Windows 2000 tries to destroy OS/2 Boot Manager. Upon boot it ignores the 0a partition ID, and sees something resembling a FAT boot sector describing 2 FAT copies. When FASTFAT.SYS marks this partition as clean in the first reserved FAT entry, the mirror (2nd) FAT sector is also updated. However, there is no mirror FAT, and FASTFAT.SYS writes into the middle of the OS/2 Boot Manager code. This aggression was built into FASTFAT.SYS at a fairly late stage, and prerelease versions work without problems. See fastfat.html for a version that can coexist peacefully with OS/2 BM. Update both \WINNT\SYSTEM\DRIVERS\FASTFAT.SYS and \WINNT\SYSTEM\DLLCACHE\FASTFAT.SYS.

Then there is the problem of what to write in (c,h,s) if the numbers do not fit. The main strategies seem to be

1. Mark (c,h,s) as invalid by writing some fixed value.

- 1a. Write (1023,255,63) for any nonrepresentable CHS.

- 1b. Write (1022,254,63) for any nonrepresentable CHS.

- 1c. Write (1023,0,1) for the begin CHS of a partition that starts at or past cylinder 1024, and write (1023,255,63) for the end.

2. Leave h, s but do something to c. Of course, these fail if h or s does not fit.

PROPERTIES OF PARTITION TABLES

By unknown

2a. Truncate c to 1023, writing (1023, #heads-1, #sectors).

2b. Truncate c to 1022, writing (1022, #heads-1, #sectors).

2c. Reduce c mod 1024, writing only its last 10 bits.

Solaris 8 follows 1b or 2b. Andreas Jellinghaus reports that Partition Magic follows 1c and detects a problem if start CHS is set to (1023,255,63). Jeff Merkey reports that Novell Netware follows 2b. He writes: If you do not use their methods on NetWare partitions, NetWare will not recognize the partition entries correctly, and will attempt to reinitialize the entire partition table on a system if they are wrong (Ouch!). Some versions of Linux fdisk used 2a or 2c, and this confuses OS/2 fdisk - cf. Linux, OS/2 and >1024 Cylinder HDDs . David A. Burton <dburton@burtonsys.com> reports that System Commander Deluxe uses 1c. Mark (c,h,s) as invalid by writing c=1022. (Maybe this is really 2b?)

PARTITION MAGIC

A very convenient tool for manipulating partitions is Partition Magic, a commercial program from PowerQuest. Below a description of some of its error numbers. (The URL that gave this information no longer exists.) This is of interest also for those who do not have this program: it indicates what conditions the PowerQuest people think a partition table should satisfy.

(Not all of these conditions are complied with by DRDOS or OS/2 or Linux or Windows NT on Alpha, so a partition manipulator should accept a much wider range of partition tables, but such a program might try to follow these rules when creating partitions.)

100 - A forked extended partition

The MBR or some EPBR contains two extended partitions. (PowerQuest uses the acronym EPBR for a link in the chain of extended partition table sectors.) (Linux comment: there are three partition types indicating an extended partition, namely 0x5, 0xf, 0x85. DOS only recognizes the first. Recent Windows only recognizes the first two. Linux will accept two or more extended partitions in the MBR, and often it is useful to have a 0x5 chain for use by DOS (where this chain stays below the 1024 cylinder boundary) and a 0x85 chain for use by Linux. Nothing is wrong with having both 0x85 and one of 0x5, 0xf in the MBR. However, it is bad to have both 0x5 and 0xf. This is sometimes seen when people use some fdisk-type program that does not yet know about 0xf on a disk that already contains such an extended partition.)

104 - Partition contains no sectors

The LBA Number of sectors value in the partition table is 0.

105 - Partition does not start on cylinder boundary

The Head value of CHS begin is not 0 or 1. PartitionMagic expects all FAT, HPFS and NTFS partitions to start and end on cylinder boundaries. (Comment: Windows NT on Alpha does not comply with this rule, and can create partitions starting on arbitrary sectors. There is no known operating system that requires this restriction. However, there exists software that tries to guess the disk geometry by looking at the CHS start and end values in a partition table. Note that with large disks CHS values are entirely meaningless.)

106 - Partition does not start with sector 1

The Sector value of CHS begin is not 1. (Same comment.)

PROPERTIES OF PARTITION TABLES

By unknown

107 - Partition begins beyond the end of the disk

The Cylinder value of CHS begin is larger than the number of cylinders that the BIOS reports. (Comment: Usually this means that programs or operating systems that use the BIOS cannot use this partition. It may help to change the BIOS translation. For Linux it does not matter, except that the /bootpartition containing LILO stuff should be accessible.)

108 - Partition does not end on cylinder boundary

The Head value of CHS end is not one less than the number of heads that the BIOS reports, or the Sector value of CHS end is not equal to the number of sectors per track that the BIOS reports. (See above under 105.)

109 - Partition ends after end of disk

The Cylinder value of CHS end is larger than the number of cylinders that the BIOS reports.

110 - Partition has different CHS and LBA lengths 111 - Logical partition starts outside extended

(Comment: the model here is that the extended partition is one big box, taking a consecutive piece of disk area, containing the logical partitions. Linux allows the logical partitions to be anywhere on the disk, also with primary partitions in between.)

112 - Logical partition ends outside extended

http://www.win.tue.nl/~aeb/partitions/partition_types-2.html (11 of 13)4/30/2004 9:57:40 AM

113 - Partitions overlap

A partition ends past the start of another. If the filesystems don't actually overlap, which they rarely do, then this can be fixed by truncating the overlapping partition. (Sometimes overlapping partitions are created by OS/2 fdisk: if there is still room in an extended partition it allows the creation of a primary partition that overlaps the end of the extended partition. Now if someone afterwards creates a logical partition inside the extended partition, data loss might occur.)

114 - Logical partition does not start one head away from EPBR

If the EPBR is found at sector N, and there are 63 sectors per track, then Partition Magic expects the logical partition to start at sector N+63.

115 - Logical partition does not end where Partition Magic expects

(Comment: Partition Magic expects the extended partition to be a big box containing a chain of pairwise disjoint boxes. Here each logical partition except for the first one has the same ending sector as the surrounding box. Another model one finds is a big box containing a smaller box, containing a smaller box ... In that model all EPBR extended partition entries will show the same end sector. In reality the end sector of an EPBR does not play a role anywhere.)

116 - Partition has different CHS and LBA begin

120 - Logical partitions not in ascending order

PowerQuest states: DOS, OS/2, Windows 95 and Windows NT require that logical partitions occur in the chain in the on-disk order. (Comment: Linux does not require this. However, reordering the links in the chain is trivial (for example with sfdisk). Note that disk names will be different after reordering.)

ACKNOWLEDGEMENTS

PROPERTIES OF PARTITION TABLES

By unknown

A lot of useful information was supplied by various people:

Thomas Wolfram (thomas@aeon.inberlin.de) - the author of os-bs,
Peter Gutmann (pgut01@cs.auckland.ac.nz) - the author of SFS,
Cody Batt (codyb@powerquest.com), Christian Carey (ccarey@CapAccess.ORG),
Dan Fandrich (dan@fch.wimsey.bc.ca), Kai Henningsen (kai@khms.westfalen.de),
Dan Hildebrand (danh@qnx.com), Todd Larason (jtl@molehill.org),
Mark Morgan Lloyd (markMLI.in@telemetry.co.uk).
Marek Michalkiewicz (marekm@i17linuxb.ists.pwr.wroc.pl),
David C. Niemi (niemidc@clark.net),
Matthias Paul (Matthias.Paul@post.rwth-aachen.de),
Loek Weerd (loekw@worldonline.nl),
S. Widlake (s.widlake@rl.ac.uk).