Multi-booting Solaris and other operating systems

Mariusz Zynel

Last revised: Dec 15, 2003

Contents

I. Introduction

I.1 Preface

Growing popularity of Unix operating systems, including free Solaris and Linux, on desktop market and relatively low prices of large IDE hard disk drives raised the problem of having multiple operating systems on a single x86 machine. There are many reasons for installing alternative operating system along with Solaris: need of support for specific hardware, testing portability of the developed software, entertainment, kids etc.

On summer ’99, approximately once a week one newbie question related to multi-booting Solaris x86 was asked on Solaris On Intel (http://groups.yahoo.com/group/solarisonintel) mailing list. Most people wanted to setup Solaris and Windows NT on their machines. I had a chance to went through this a couple of times, learned a lot about dual-booting, so I was doing my best answering questions and trying to help. That way I shared my experience with others. There were FAQs devoted to Solaris x86, but information on dual-booting was incomplete and sometimes misleading. At that moment, I was visiting Shinshu University in Nagano, Japan, hardly working on Jordan’s curve theorem formalization using Mizar proof-checker. I needed some off-time and thought that writing a guide is a good idea. That was the beginning.

Now, this is the second approach to write a "perfect multi-booting guide". Thanks to feedback from users around the world, many corrections have been made. Numerous inquires indicated problems that should be covered by the document.

Hope this guide really helps.

Acknowledgments

Many thanks are due to the staff of Kiso Laboratory in the Shinshu University, Nagano, Japan for their support during preparation of this documentation.

The author is greatly indebted to Bruce Riddle for initiative and encouragement.

The author would like to express his gratitude for all who contributed to this documentation. Thanks go to (in alphabetical order):

- Bruce Adler - for clarification of x86 Boot partition addressing,
- Shantanu Datta - for pointing out the problem of extended partitions of ID 0x0F,
- Bernd Finger - for clarifying addressing conventions used by pcfs driver,
- Marian Ion - for example of GRUB configuration file,
- Richard Ji - for pointing out the performance issue when setting up two systems on two hard disk drives,
- Stefan Kanthak - for the news on modern BIOS Int 13h extensions support and details on NTFS and NT 4.0,
- Allan Kerr - for exhaustive explanation and references regarding the size of NTFS boot partition,
- Charles M. Kozierok - for providing whole sections on mysterious hardware and BIOS aspects,
- Alan Pae - for establishing the first mirror site in US and for HTML debugging,
- Thomas Pornin - for information about Solaris fdisk behavior,
- Michael Riley - for clearing up Solaris disk naming conventions and a hint on how to use multiple Solaris partitions on a single hard disk,
- Frank Sexton - for input on booting NT from a floppy disk,
- David N. Stivers - for details on setup and booting Solaris, NT and Linux.

Special acknowledgments for authors referenced in this document for making their documentation freely available on the Internet. Special thanks also go to all whose inquires and feedback indicated issues worth explanation, confirmed or forced review of presented solutions.

Feedback

All suggestions, comments and corrections are very welcomed. If you find misspelled word, unclear phrase, broken link or your experiences related to the topics of the document are different please let me know. Please also do not hesitate to ask for help in case of problems. Just mail to mariusz@solaris-x86.org.

I.2 Legal notice

Disclaimer

Copyright

Trademarks

I.3 Applicability

What IS this document about?

The document deals with the problem of making Sun Solaris Intel Edition operating system coexisting with other operating systems on a single x86 machine. It provides, however, general information that might be used for setting up any two, or more, operating systems per a machine.

The following operating systems were tested by the author and contributors, and are proved to coexist together:

- Sun Solaris 2.5.1, 2.6, 7, 8, 9, 10 x86,
- Linux,
- FreeBSD,
- MS DOS 2.5 and above,
- MS Windows 3.x, 9x, Millenium, NT 4.0, 2000, XP.

This does not eliminate any other operating systems. The information provided may be unsatisfactory, or incomplete in such a case though.

Two variants of installation are discussed: multiple operating systems on a single hard disk drive, as well as on separate hard disk drives. Factors, which should be considered when making decision on installation are presented. Instructions on how to preserve an existing operating system are given.

Some general issues are also covered to understand the problem better. There are sections on the x86 boot process, BIOS issues, connecting and partitioning hard disk drives.

The document gives instructions on how to use Solaris, Linux and Windows NT boot managers. Some general information on third party boot managers is also provided.

There are sections devoted to Solaris x86 maintenance. Disk naming conventions are explained with examples. There are instructions on how to partition hard disks, create slices, create and mount file systems under Solaris x86.

Various example installations are presented. Their advantages and disadvantages are pointed out.

Note: The term *Windows NT* used in the document refers to Windows NT 4.0 as well as to Windows 2000 (NT 5.0) or XP (NT 5.1), unless it is stated otherwise.

What IS NOT this document about?

The document is not intended to be a guide or manual for setting up any specific operating system. It contains no details on how to install any particular operating system. Only general hints and suggestions on installation are given, with focus on Solaris x86.

The guide does not contain recommendations of hardware, and could not be considered a buyer’s guide.

I.4 History

Dec 15, 2003

Thanks to Bernd Finger for clarifying addressing conventions used by pcfs driver in Solaris 9.

May 27, 2003

Thanks to Alan Pae the mirror in the US is setup again. It is available at http://www.geocities.com/paedalbu/multiboot (http://www.geocities.com/paedalbu/multiboot). Also thanks to Alan reports broken links are fixed now.

Dec 26, 2002

The following additions has been made:

- FAQ section,
- a note on Solaris fdisk behavior,
- GRUB configuration example and clarification.

Nov 3, 2002

A new scenario has been added: "Two hard disks, Solaris and Windows". Due to the results of extensive tests of mounting FAT volumes under Solaris some corrections were applied to the text.

Dec 2, 2001

Problems with extended partitions of ID 0x0F indicated. Some minor corrections. Guide made available through the anonymous rsync (http://rsync.samba.org/rsync) server on the main site (rsync://math.uwb.edu.pl/multiboot).

Nov 11, 2001

Newer BIOS-es support Int 13h extensions. Therefore, hard disk capacity limitations known as 8 GB or 1024 cylinder barriers, are suppressed in those operating systems or boot managers which make use of those extensions. Related documents are now updated. Thanks go to Stefan Kanthak for pointing the problem out.

Sept 21, 2001

Details on how to fix NT boot sector added. Example of safe Solaris + NT installation corrected. Some technical cosmetic.

Aug 28, 2001

Third major revision. Section "Troubleshooting" added to simplify searching for a workaround. "Reconfiguring boot device" and "History" added. A lot of minor updates, including linking problems.

Aug 25, 2001

The first mirror site established in US: http://www.solarisresources.com/multiboot. Thanks go to Alan Pae.

May 15, 2001

Minor update. There is a simple but sure method to address x86 Boot partition. Credits to Bruce Adler.

Feb 2, 2001

Minor update. The size of NTFS boot partition corrected. Credits to Allan Kerr.

Aug 3, 2000

Second major revision. The guide gained its current shape. A lot of serious updates. More practical details have been added related to Solaris and maintenance of multi-boot environment.

Sept 27, 1999

First major revision, more examples added.

Jul 5, 1999

The very first, initial issue. Site established in Poland, Bialystok at
http://math.uwb.edu.pl/~mariusz/multiboot (http://math.uwb.edu.pl/~mariusz/multiboot), remotely from Japan.

II. Generalities

II.1 Geometry of a hard disk drive

Basics

The generic term used to refer to the way the disk structures its data into platters, tracks and sectors, is its *geometry*. In the early days this was a relatively simple concept: the disk had a certain number of heads, tracks per surface, and sectors per track. These were entered into the BIOS set up so the PC knew how to access the drive.

With newer drives the situation is more complicated. The simplistic limits placed in the older BIOS's have persisted to this day, but the disks themselves have moved on to more complicated ways of storing data, and much larger capacities. The result is that tricks must be employed to ensure compatibility between old BIOS standards and newer hard disks.

Note: These issues are related to IDE/ATA hard disks, not SCSI, which use a different addressing methodology.

Physical geometry

The *physical geometry* of a hard disk is the actual physical number of *head*, *cylinder*, and *sector* used by the disk. On older disks this is the only type of geometry that is ever used. The original setup parameters in the system BIOS are designed to support the geometries of these older drives, in particular the fact that every *track* has the same number of sectors.

All newer drives that use zoned bit recording must hide the internal physical geometry from the rest of the system, because the BIOS can only handle one number for sectors per track. These drives use logical geometry figures, with the physical geometry hidden behind routines inside the drive controller.

Logical geometry

Drive parameters obtained by its auto-detection in the system BIOS setup, additional software such as ATAID, or those printed in a hard disk's setup manual are the *logical geometry* parameters that the hard disk manufacturer has specified for the drive. Since newer drives use zoned bit recording and hence have ten or more values for sectors per track depending on which region of the disk is being examined, it is not possible to set up the disk in the BIOS using the physical geometry. Also, the BIOS has a limit of 63 sectors per track, and all newer hard disks average more than 100 sectors per track, so even without zoned bit recording, there would be a problem.

To get around this issue, the BIOS is given bogus parameters that give the approximate capacity of the disk, and the hard disk controller is given intelligence so that it can do automatic translation between the logical and physical geometry. Virtually all modern disks use a logical geometry with 16 heads and 63 sectors, since these are the largest values allowed by the BIOS. The actual physical geometry is totally different, but the BIOS (and the system) knows nothing about this. As far as the system is concerned, the disk has 16 heads and 63 sectors on every track, and the hard disk itself takes care of all the "dirty work". The physical geometry is totally hidden from view.

Here is an example showing the difference between the physical and logical geometry for a sample drive, a 3.8 GB Quantum Fireball TM (from [Quantum]):

Specification	**Physical Geometry**	**Logical Geometry**
Read/Write Heads	6	16
Cylinders (tracks per Surface)	6,810	7,480
Sectors Per track	122 to 232	63
Total Sectors	7,539,840	7,539,840

Table II.1.1: Physical and logical geometry of 3.8 GB Quantum Fireball TM.

The fact that both geometries equate to the same number of total sectors is not a coincidence. The purpose of the logical geometry is to enable access to the entire disk using terms that the BIOS can handle. The logical geometry could theoretically end up with a smaller number of sectors than the physical, but this would mean wasted space on the disk. It can never specify more sectors than physically exist, of course.

The translation between logical and physical geometry is the lowest level of translation that occurs when using a modern hard disk. It is different from BIOS geometry translation, which occurs at a higher level, to overcome hard disk capacity limitations (cf. [Kozierok]).

II.2 Hard disk capacity limitations

The Int 13h software interrupt

The software interrupt Int 13h, supports hard disk access commands that can be given to the BIOS, which then passes them on to the hard disk. Addressing of sectors is done by the use of 3-dimensional coordinate system. First two coordinates: a *cylinder* number and a *head* number, determine the *track* on the hard disk, and start from zero. The third one is the number of a *sector* on this track, and by convention, starts from one. The Int 13h interface allocates 24 bits for the specification of the sector coordinates, precisely:

- 10 bits for the cylinder number, or a total of 1,024 cylinders,
- 8 bits for the head number, or a total of 256 heads,
- 6 bits for the sector number, or a total of 63 sectors.

This means that the BIOS can support disks containing up to approximately 16.5 million sectors, which at 512 bytes per sector yields a maximum of 7.875 gigabytes. Disks larger than this require either an operating system that totally bypasses the BIOS without using Int 13h or a BIOS and operating system supporting Int 13h extensions.

The 504 MB barrier

The most (in)famous hard disk barrier is the 504 MB limitation for standard IDE/ATA hard disks. It is alternatively referred to as the 504 MB or the 528 MB barrier, depending on whether binary or decimal megabytes are considered. Due to this barrier, a hard disk with a size over 504 MB will normally appear only as having 504 MB under some circumstances. This problem is a result of combining the geometry specification limitations of the IDE/ATA standard and the BIOS Int 13h standard.

The problem is that due to poor planning, the standards are not the same; they each reserve different numbers of bits for the geometry. In order to use an IDE/ATA hard disk with the standard BIOS disk routines then, the limitations of both standards must be taken under account, which means that only the smaller of each geometry number can be used. Here is how the two standards allocate bits for the geometry:

Standard	**Bits For**			
	Cylinder Number	**Head Number**	**Sector Number**	**Total Geometry**
IDE/ATA	16	4	8	28
BIOS Int 13h	10	8	6	24
Combination	10	4	6	20

Table II.2.1: The number of bits used for the geometry specification of IDE/ATA and BIOS standards, and their combination.

The maximum number supported for any parameter is 2^N, where N is the number in the table above. So this means that under IDE/ATA, 2^16 or 65,536 cylinders are supported. We can then multiply all the figures together to get a total number of sectors supported, and then multiply that by 512 bytes (per sector) to get the maximum supported capacity:

Standard	**Maximum Cylinders**	**Maximum Heads**	**Maximum Sectors**	**Maximum Capacity**
IDE/ATA	65,536	16	256	128 GB
BIOS Int 13h	1,024	256	63	7.88 GB
Combination	1,024	16	63	504 MB

Table II.2.2: The maximal capacity supported by IDE/ATA and BIOS standards, and their combination.

Note: The BIOS Int 13h limit for sectors is 63, and not 64, because by convention sectors are numbered starting from 1, not 0.

The 504 MB figure is just 1,024 * 16 * 63 * 512; the problem is the combination of the limitations of the two standards. Due to the 16-head limitation of IDE/ATA, no IDE hard disk is ever specified with more than 16 logical heads; they always have a large number of cylinders instead. The problem is that when you put the disk in a machine with a standard, non-translating BIOS, it can't see more than 1,024 of the cylinders. There are several different ways that the system may react to a drive too large for it to handle.

The normal solution to the 504 MB problem is to use BIOS geometry translation. Software drive overlays will also avoid the problem, but at a cost.

The 7.88 GB barrier

This barrier, sometimes just called the "8 GB barrier", is based on a BIOS limitation. When the standard of Int 13h was planed capacity of hard disks did not exceed 10 MB, and nobody assumed use of disks grater than 8 GB. Today, it is a problem.

After reading previous two sections the source of that barrier is clear. The software interrupt Int 13h simply does not allow to pass more than 24 bit long arguments to the hard disk via the BIOS. This results in unbreakable barrier of 7.88 GB, which is the product of 1,024 cylinders, 256 heads, 63 sectors and 512 bytes per sector. Unbreakable, if standard interrupt Int 13h is used. This includes all IDE/ATA disks and SCSI disks. By obvious reason there is no use in applying BIOS geometry translation to get around this problem.

The only way to get around the 7.88 GB barrier is to make a break with the past and change the way hard disk accesses are done. One of the solution is so-called *Int 13h extensions* which allows much larger capacity drives to be used. Using this method requires consistent BIOS and operating system support for the extensions.

Another way to work around this barrier is *Direct Disk Access*, in other word *bypassing the BIOS*. More advanced operating systems take away responsibility for dealing with the hard disk from the BIOS and employ their own access routines for the hard disk, which are faster and more efficient than using the default BIOS code. This is mainly done for performance reasons (cf. [Kozierok]).

Note: Besides barrier described in this section, there are yet two another for IDE/ATA hard disk drives, called: 1.97 GB and 2 GB. More detailed discussion of this topic can be found in [Kozierok].

Note: Some older SCSI host adapters did have problems that made them unable to access hard disks over 1 GB in size, or some other arbitrary limits. Present host adapters should not have these limitations.

Note: FAT-16 file system is restricted to 2 GB per partition. It is a file system issue, not a hardware or BIOS one.

Note: Solaris 7 and earlier can access up to first 1024 cylinders (ca. 8 GB) on IDE/ATA hard disks.

II.3 BIOS geometry translation modes

BIOS geometry translation

While the use of logical hard disk geometry gets around the problem that physical hard disk geometries cannot be properly expressed using standard BIOS settings, they don’t go far enough. In most cases, higher levels of translation are needed as well because other problems relating to old design decisions make it impossible for even the logical geometry to be used with modern large hard disks.

In order to get around hard disk capacity barriers, another layer of translation is often applied on top of the geometry translation that occurs inside the hard disk. this translation is performed by the BIOS. There are three BIOS translation modes:

1. Normal / Standard CHS Mode,
2. Extended CHS (ECHS) / Large Mode,
3. Logical Block Addressing (LBA).

Note: None of this has any relevance when using only SCSI hard disks. They are not subject to any of the BIOS limitations other than the 7.88 GB barrier, which is not overcome through translation.

Note: SCSI by its nature uses logical block addressing (LBA) to address devices.

Normal / Standard CHS Mode

In this mode, there is no translation done at the BIOS level, and the logical geometry presented by the disk is used by the BIOS directly. "CHS" stands for "*cylinder*, *head*, *sector*", the three parameters - coordinates- used in hard disk geometry specifications. This mode can be used with drives not exceeding 504MB in capacity otherwise, only the 504 MB part of the disk will be accessible when no BIOS bypassing is used.

Extended CHS (ECHS) / Large Mode

Extended CHS, also called ECHS or large mode in some BIOS's, uses BIOS translation to get around the 504 MB size barrier inherent in standard CHS mode.

BIOS translation works by taking advantage of the fact that BIOS Int 13h allows more heads than the IDE/ATA standard but fewer cylinders. The BIOS takes the logical geometry that the hard disk specifies according to the IDE/ATA standard, and *translates* it into an equivalent geometry that will "fit" into the maximums allowed by the BIOS Int 13h standard. This is done by dividing the number of logical cylinders by an integer, and then multiplying the number of logical heads by the same number. The translation made by BIOS is simply an exchange of coordinate systems.

Consider a 3.1 GB Western Digital Caviar hard drive, AC33100. This drive actually has a capacity of 2.95 binary GB, and logical geometry of 6,136 cylinders, 16 heads and 63 sectors. This is well within the bounds of the IDE/ATA limitations, but exceeds the BIOS limit of 1,024 cylinders. The BIOS picks a translation factor such that dividing the logical number of cylinders by this number will produce a number of cylinders below 1,024. Usually one of 2, 4, 8, or 16 are selected; in this case the optimal number is 8. The BIOS then divides the number of cylinders by 8 and multiplies the number of heads by 8. This results in a translated geometry of 767 cylinders, 128 heads and 63 sectors. Evidently, the capacity stays unchanged, and the new geometry fits quite nicely into the BIOS limits:

Standard	**Cylinders**	**Heads**	**Sectors**	**Capacity**
IDE/ATA Limits	65,536	16	256	128 GB
Hard Disk Logical Geometry	6,136	16	63	2.95 GB
BIOS translation Factor	divide by 8	multiply by 8	--	--
BIOS translated Geometry	767	128	63	2.95 GB
BIOS Int 13h Limits	1,024	256	63	7.88 GB

Table II.3.1: ECSH geometry translation applied to 3.1 GB Western Digital Caviar TM.

The BIOS presents the translated geometry to the operating system and the hard disk is seen as it has 767 cylinders, 128 heads and 63 sectors. Whenever the operating system or an application wants to use BIOS Int 13h calls, they use this geometry. The BIOS, when it executes its disk access routines, translates back to the real logical geometry used by the hard disk before sending its request to the disk.

Extended CHS or large mode are, in practice, not that frequently used. Instead, LBA mode is more popular; it is similar in concept but does the translation differently.

Logical Block Addressing (LBA)

As we mentioned, regular addressing of IDE/ATA drives is done by specifying a cylinder, head and sector address where the data that is required resides. Extended CHS addressing adds a translation step that changes the way the geometry appears in order to break the 504 MB barrier, but the addressing is still done in terms of cylinder, head and sector numbers, however, they are just translated one or more times before they get to the actual disk itself.

In contrast, *logical block addressing* or *LBA* involves a new way of addressing sectors. Instead of referring to a cylinder, head and sector number, each sector is instead assigned a unique "sector number". 3-dimensional coordinate system is just linearized. In essence, the sectors are numbered 0, 1, 2, etc. up to (N-1), where N is the number of sectors on the disk. An analogy would be as follows. The mailing address is composed of a street number, street name and city name. Here we have three coordinates as in conventional CHS addressing. Instead however, let's say that every house in the country were given a unique identifying number. That is how LBA works.

In order for LBA to work, it must be supported by the BIOS, as well, as by the hard disk. Most newer hard disks do in fact support LBA, and when auto-detected by a BIOS supporting LBA, will be set up to use that mode.

A drive using LBA is not subject to the 504 MB disk size barrier, however there has been a great deal of confusion regarding LBA and what it does. In particular, a lot of people think that it is the LBA addressing that "gets around the 504 MB barrier". Strictly speaking, this is inaccurate. It isn’t the LBA that is getting around the barrier, because LBA is just a different way of addressing same geometry; if you were still limited to 1,024 cylinders, 16 heads and 63 sectors, you would still have logical sectors numbered 0, 1, 2, etc. up to 1,032,191, and you would still be stuck with 504 MB.

The reason that setting a drive’s mode to LBA will get around the 504 MB barrier is that LBA mode automatically enables geometry translation as well. This translation is still required because the software calling the BIOS Int 13h routines knows nothing about LBA. It is the translation that is what really gets around the barrier, but of course all of this happens transparently to the user.

When LBA is turned on, the BIOS will enable geometry translation; this is done the same way that it is done in Extended CHS or large mode. The translated geometry is still what is presented to the operating system for use in Int 13h calls. The difference is that when using ECHS the BIOS translates the parameters used by these calls from the translated geometry to the drive’s logical geometry. With LBA, it translates from the translated geometry directly into a logical block (sector) number (cf. [Kozierok]).

II.4 Logical structure of a hard disk drive, boot process

Primary and extended partitions, file systems

Each hard disk has to be *partitioned* before any data can be stored on it. A partition is a contiguous chunk of the hard disk space and contains either, a *file system*, or *logical volumes*, as it is *primary* or *extended*. A file system is an arrangement of directories and files strictly connected with an operating systems which implements it. Internally, a logical volume resembles a primary partition. It also contains a file system, and from such point of view, primary partitions and logical volumes are indistinguishable, thus we call them all simply *volumes*. Logical volumes may be considered primary partitions, lined up in a chain within an extended partition, so that the first one holds the address of the second one and so on. Sometimes, logical volumes are called *logical drives* or *logical partitions*.

Note: We make strict distinction between *partition*s and *slice*s. Partitions are an element of the x86 architecture, while slices are related to data storage on Unix systems.

Only four partitions per a hard disk drive are allowed. Extended partitions and logical volumes were developed to allow the use of larger hard disks with DOS, while the compatibility with previous standards is preserved. Logical volumes help also to keep data on a hard disk well organized, and to avoid the slack which appears when large clusters are used on large partitions. See [Kozierok] for further details on this topic.

Within DOS and Windows 3.x/95/98 only the first primary and the first extended partition of the hard disk are accessible. Solaris, Linux, and Windows NT can access all volumes in any combination, i.e. their fdisk programs may create and delete multiple primary and extended partitions. In DOS and Windows for all volumes, that are accessible, consecutive letters - *drive letters* - are assigned, starting with C. To determine the way letters are assigned following rules apply:

1. IDE/ATA disks takes precedence before SCSI ones,
2. a hard disk marked as a boot disk takes precedence before others,
3. physical connection of IDE/ATA disks determine the order of letters: master, slave for older, one channel, IDE controller and primary master, primary slave, secondary master, secondary slave for newer, two channel, EIDE controller,
4. SCSI host adapter’s BIOS settings and the order of devices in the SCSI chain determine the order of SCSI disks,

5. primary partitions takes precedence before logical volumes,
6. order of logical volumes on one extended partition stays unchanged.

This schema of assignment has nothing to do with lettering in the *BIOS boot sequence*. When two channel EIDE controller is used then primary master drive has assigned letter C, primary slave D, secondary master E and secondary slave F.

In Windows NT drive letters to all volumes, except the boot partition, may be assigned manually. In W2K additionally volumes (file systems) may be mounted in directories.

The main difference between primary and extended partition is that only the first one is boot-able by BIOS. There are boot managers e.g. Lilo or Ranish PM that are able to boot from logical volume. Obviously, operating system files may be stored on primary as well on extended partition. Usually, only a few files which are used in the early phase of the boot process must reside on the boot primary partition.

There are many file systems, but the most commonly used are: *FAT-16*, *FAT-32* implemented in Service Release 2 of Windows 95, *NTFS* implemented by Windows NT 4.0, *UFS* used by Solaris, and *HPFS* used by OS/2. FAT-16 is recognized and fully supported by most operating systems available on PC platform. FAT-16 volume is a good solution for data exchange between operating systems running on the same machine. FAT-16 and FAT-32 are not compatible. NT 4.0 does not recognize FAT-32 without additional software while W2K fully supports it.

Note: NTFS version 3 used by NT 4.0 is not compatible with NTFS version 5 introduced in Service Pack 4 for NT 4.0 and used by W2K. All NTFS volumes version 3 are automatically converted to version 5 when used under W2K. The operation can not be reversed, which makes them unusable in NT 4.0.

Solaris UFS file system has been improved in Solaris 7 by acquisition of journal file system methods, which can save headaches with data restoration after the system crash or unattended power failure. The new feature is called "logging" and can be turned on when a file system is mounted. On the command line it is done by supplying `-o logging` file system specific option to `mount` command (see `mount`, `mount_ufs` man pages). To mount default file systems during boot process `logging` should be added to mount option field in `/etc/vfstab` configuration file (see vfstab man page).

Master Boot Record (MBR)

The first sector of a hard disk drive is called its *Master Boot Record* or briefly MBR. Its coordinates are Cylinder 0, Head 0, Sector 1. It stores two sorts of information: *Master Partition Table* and *Master Boot Code*. Master Partition Table, also known as *fdisk table*, contains the descriptions of the partitions on the hard disk. Since MBR must fit into 512 bytes there is only room for four partitions. Therefore, a hard disk can have up to four partitions.

Offset	Size	Description
+ 0x000	0x1BE	Master Boot Code
+ 0x1BE	0x010	Partition #1
+ 0x1CE	0x010	Partition #2
+ 0x1DE	0x010	Partition #3
+ 0x1EE	0x010	Partition #4
+ 0x1FE	0x002	End of Master Boot Table marker - 0x55 0xAA

Table II.4.1: Master Boot Record specification.

Note: Windows NT and W2K store the information on drive letter assignment in *NT Disk Signature*, which is unique for every hard disk drive. This signature is simply 4 bytes of MBR starting at byte 0x1B8 - right before the definition of partition number 1. Most of boot managers and partition managers do not preserve this bit of data, which may lead to troubles in case of non-standard drive lettering.

Each partition entry in the Master Partition Table has its Partition ID, Activity Flag, Start Cylinder/Head/Sector, and End Cylinder/Head/Sector specified.

Offset	Size	Description
+ 0x00	0x01	Activity Flag
+ 0x01	0x01	Start Head
+ 0x02	0x02	0x0A bits for Start Cylinder, 0x06 bits for Start Sector
+ 0x04	0x01	Partition ID
+ 0x05	0x01	End Head
+ 0x06	0x02	0x0A bits for End Cylinder, 0x06 bits for End Sector
+ 0x08	0x04	The number of the first sector of the partition, relative to the entire disk
+ 0x0C	0x04	The size of the partition measured in sectors

Table II.4.2: MBR partition entry specification.

The Partition ID specifies the type of the partition (or to what OS it belongs), Activity Flag determines which primary partition should be booted, and the Start/End specify location of the partition [Richmond]. The following Partition IDs are mostly used:

Partition ID	Description
0x01	DOS12 (12-bit FAT)
0x04	DOS16 (16-bit FAT)
0x05	Extended
0x06	BIGDOS FAT-16
0x07	Windows NT (NTFS), OS/2 (HPFS)
0x0B	FAT-32
0x0C	FAT-32 LBA
0x0E	FAT 16 LBA
0x0F	Extended LBA
0x63	Unix SysV/386
0x81	Linux/Minix
0x82	Solaris x86 UFS, Linux swap
0x83	Linux ext2fs
0x85	Linux Extended
0xA5	FreeBSD, NetBSD, BSD/386
0xBE	x86 Boot
0xEB	BeOS

Table II.4.3: Example partition identifiers.

If there is an extended partition on the disk, Master Partition Table holds a link to an *Extended Partition Table* that describes the first logical volume for the partition. That table contains information about that first logical volume, and a link to the next Extended Partition Table which describes the second logical volume on that partition, and so on. This allows for infinite number of logical volumes, theoretically.

To load an operating system, BIOS as a starting point of the boot process, needs to know where that operating system resides. Instead, BIOS loads Master Boot Code - small initial boot program in the MBR. This program eventually transfers control to the partition boot program stored at the first sector of the active partition.

Volume Boot Sector (VBS)

Each volume - primary partition or logical volume - has its own *Volume Boot Sector*. This is distinct from the MBR that controls the entire disk, but is similar in concept. Each VBS contains *Disk Parameter Block* and *Volume Boot Code*. Disk Parameter Block, sometimes called the media parameter block, is a data table that contains specific information about the volume, such as its size, number of sectors it contains, label name etc. Volume Boot Code is a code that is specific to an operating system that is using this volume and is used to start the load of the operating system.

The Volume Boot Sector is created during a high-level format of a hard disk partition. On floppy disks, the first physical sector is VBS.

Solaris Volume Boot Sector is called *Partition Boot Record*, or shortly PBR.

The boot process

The *boot process*, also known as *bootstrapping*, on PC begins when the computer is turned on (*cold start*), or after the reset (*warm start*). First, the system firmware in the BIOS ROM executes a power-on self test (POST), a kind of hardware check, and runs BIOS extensions in peripheral board ROMs. After devices are configured, BIOS invokes software interrupt Int 19h. The Int 19h handler typically loads the first physical sector of the boot disk into memory locations 0x0000:0x7C00 through 0x0000:0x7DFF and then passes control to the boot code placed in that sector. Which disk is the boot one depends on the BIOS boot sequence setting. When SCSI is selected, then suitable setting in the SCSI host adapter's BIOS ROM determines the boot drive. Newer BIOSes allow to boot from CD-ROM, LAN or some removable media like ZIP or LS-120.

On a hard disk, the first physical sector is its MBR, and the boot code is the Master Boot Code. First, MBR moves itself to 0x0000:0x0600, then loads the VBS of the first active primary partition on the boot disk to it's original location 0x0000:0x7C00, and jumps to its first byte, unless a non-standard Master Boot Code is in use, such as Lilo or Ranish PM. This completes the standard PC hard disk sequence.

For a floppy disk, the first physical sector is the VBS. The BIOS does not care whether it is loading an MBR or a VBS. When booting from a floppy disk, which has no partitioning information and therefore no MBR, only the VBS is effectively loaded (cf. [Locke]).

After the BIOS passed control to either, MBR in case of booting from a hard disk, or VBS when booting from a floppy, the boot program specific to the operating system is started. Its behavior depends strictly on the operating system. Generally, the rest of the boot process, as well as its initial part, is divided into more and more advanced boot stages. Some operating systems like Solaris, Linux, or NT allow for user interaction during the system load. All these systems offer their own boot managers, i.e. programs which give user some control over the boot process, and thus make coexistence of many operating systems on one machine easier. A boot manager need not to be the part of an operating system. There are many third party boot managers, from tiny OS-BS, to very powerful, though complex, System Commander.

DOS and Windows 3.x/9x boot process is simplified. There is no user interaction. Volume Boot Code simply loads `io.sys` file, which then loads device drivers and starts user shell.

Modern operating systems are not limited to boot off the local hard disk drive only. Solaris and NT are capable to boot across the network. This makes possible to have fully functional workstations with no hard disk drives, which reduces costs of management and administration.

III. Boot managers

III.1 Solaris boot manager

Primary boot

Solaris boot process consists of two conceptually distinct phases, primary boot and secondary boot. The primary boot is implemented in the BIOS ROM on the system board and in BIOS extensions in ROMs on peripheral boards. The code is x86 real-mode code. This part of the boot process is common for all PC-compatible systems.

A Solaris boot partition is a primary partition, and must be active in order to boot Solaris. It begins with one-cylinder boot *slice*, which contains the partition boot program `pboot` in the first sector, the standard Solaris disk label and the *VTOC* in the second and third sectors, and the `bootblk` program in the fourth and subsequent sectors (VTOC is discussed later).

When a system is booted from a hard disk drive, MBR of that disk is read into memory. If Solaris was the last operating system installed to this disk, its MBR contains `mboot` - master boot program. The program, though specific for Solaris, works as usual Master Boot Code, that is, it reads the first sector of the first active primary partition on the boot disk, and jumps to its first byte. If the active partition is Solaris boot partition, its VBS is read and `pboot` - partition boot program contained there is invoked. Then `pboot` in turn loads and runs `bootblk` program.

In case Solaris is boot from floppy or CD-ROM, the first physical sector of the media read by BIOS, is a VBS not MBR. The `pboot` program contained there is responsible for loading the image of the boot manager `strap.com`.

Note: From version 8 up, Solaris x86 installation CDs are boot-able.

Note: Some releases of Solaris 8 are affected by a bug which causes that Solaris Master Boot Code always loads Solaris VBS (PBR) from the first physical hard disk drive in the system. If Solaris is on second drive the error message "Bad PBR" is reported. See .

Boot manager

Programs `bootblk` and `strap.com`, loaded in result of primary boot, are very similar. Both contain Solaris boot manager code, which reads current disk partition information, and in case there is more than one primary partition, displays the table with all available partitions. It gives an opportunity to reboot another partition, in other words, another operating system. The default one is Solaris. The default timeout to make selection is 30 seconds. The difference between `bootblk` and `strap.com` is that, the first one is placed on UFS file systems and can not be configured, while the second is placed on FAT file systems (floppy disk or x86 Boot partition) and the timeout may be set in `/solaris/strap.rc` file.

Secondary boot

If Solaris partition is selected to boot off, the secondary boot - `boot.bin` - is loaded by the boot manager. The secondary boot is capable of reading and booting from a UFS file system on a hard disk or CD, or by way of LAN using NFS protocol. It switches the processor to 32-bit, paged, protected mode and performs some limited machine initialization. Then, it runs Device Configuration Assistant, which either, boots the system automatically, or let the user select boot device, depending on the state of `auto-boot` eeprom variable (see `eeprom` man page). After that, root file system specified in `bootpath` eeprom variable is mounted.

Secondary boot is programmable, which makes it very flexible. If the root file system is successfully mounted, `boot.bin` invokes a command interpreter, that interprets `/etc/bootrc` script. The `/etc/bootrc` script lets the user to specify additional options for booting kernel, e.g. -r to reconfigure devices automatically.

Device Configuration Assistant

Device Configuration Assistant, or shortly DCA, is used to add, remove and configure device drivers necessary to boot a machine up. Note that keyboard, display and mouse are configured with `kdmconfig` (see `kdmconfig` man page). DCA is run at the beginning of Solaris setup to determine what device drivers need to be load, and to select the device from which the installation software will be loaded. DCA is shipped with Solaris media kit on a floppy disk. Solaris 8 installation CD (*Installation* as well as *Software 1* CD) is boot-able and contains DCA.

DCA is invoked by the secondary boot every time Solaris is boot up. In this situation DCA behavior is determined by the state of two eeprom variables: `auto-boot` and `auto-boot-timeout`. The first one, if true, causes DCA to boot Solaris kernel off the device and partition specified in `bootpath` variable, otherwise presents a list of all devices capable of doing that and waits for a user selection. The list usually contains all hard disk drives, CD-ROMs, and LAN adapters. The second variable `auto-boot-timeout` determines the amount of time DCA waits for pressing Esc (escape) key to enter into DCA utility.

By default `auto-boot` is set to true, and `auto-boot-timeout` to 5 seconds. With these settings DCA displays information that it waits for pressing Esc and delays boot process for 5 seconds. Both variables may be modified using either, DCA itself, or `eeprom` utility. To modify variables with DCA one needs to enter DCA utility pressing Esc key during Solaris boot time. First, DCA probes hardware, then presents the list of possible boot devices. New values may be assigned to variables under *Boot Tasks* option. If Solaris is already running, type as root:

```
eeprom auto-boot?=true
```

to set `auto-boot` to true, or

```
eeprom auto-boot-timeout=0
```

to make DCA not to wait for Esc key.

Restoring Solaris boot manager

Solaris Master Boot Code of the MBR might be overwritten during installation of other operating system or a third party boot manager after Solaris was installed. Solaris Master Boot Code is stored in the file `/usr/lib/fs/ufs/mboot`. It can be restored using Solaris `fdisk` utility (see `fdisk` man page) as follows:

```
fdisk -b /usr/lib/fs/ufs/mboot -n /dev/rdsk/c0d0p0
```

where `/dev/rdsk/c0d0p0` stands for the raw device associated with the first (usually primary master) IDE/ATA hard disk drive on the system.

Sometimes it is necessary to recover VBS of a Solaris boot partition or Solaris boot manager program. `installboot` utility is capable of doing that (see `installboot` man page). To refresh VBS of Solaris partition located on the first IDE/ATA drive type as root:

```
installboot /usr/platform/i86pc/lib/fs/ufs/pboot \
/usr/platform/i86pc/lib/fs/ufs/bootblk /dev/rdsk/c0d0s2
```

Since `installboot` can not verify the partition specified as its argument, it may write Solaris boot manager into any partition. Therefore care must be taken not to destroy other partitions. The same operation may be also done with `fmthard` utility discussed later.

The following files contain critical executable code for boot process:

- `/usr/lib/fs/ufs/mboot` - Master Boot Code,
- `/usr/platform/i86pc/lib/fs/ufs/pboot` - Volume Boot Code,
- `/usr/platform/i86pc/lib/fs/ufs/bootblk` - Solaris boot manager.

x86 Boot partition

x86 Boot partition was introduced in Solaris 8. Normally, it is a small primary partition, with FAT file system and DCA installed. It is necessary when Webstart installation is used in order to boot Solaris kernel from miniroot (temporary) file system. It provides also a user the way to load Solaris kernel from non-boot hard disk drive, select one of multiple Solaris installations, or boot Solaris across a network - all that DCA booted from floppy can do.

It is created by default during Solaris 8 installation. Practice shows, however, that x86 Boot partition is not necessary to run Solaris 8. DCA is still below `/boot` directory and can be run during boot time.

x86 Boot partition can be created with Solaris `fdisk` program. At this stage it is simply a reserved portion of a hard disk with partition ID set to 0xBE. To make it functional FAT file system must be constructed, and Solaris boot manager installed. If x86 Boot partition is created and set active, on `c0d0` (the first IDE drive), then command:

```
   mkfs -F pcfs -o b=BOOT,B=/boot/mdbootbp,s,i=/boot/strap.com
/dev/rdsk/c0d0p0:boot
```

will do the job. To install DCA mount the partition and copy necessary files:

```
   mount -F pcfs /dev/dsk/c0d0p0:boot /mnt

cp -r /boot/* /mnt

echo "/boot/ / p"> /mnt/solaris.map
```

`/dev/dsk/c0d0p0:boot` is always the right address for x86 Boot partition on `c0d0` drive. `solaris.map` file is required when DCA is installed on a FAT file system to remap default UFS paths to FAT ones. It is present on DCA floppy disk, but redundant on UFS variant of DCA.

Reconfiguring boot device

Solaris requires details on physical connection of all devices to function properly. It saves these data during installation. When disk controller or cables are changed the data gets outdated. The following needs to be updated to boot Solaris:

- `bootpath` in `/boot/solaris/bootenv.rc`,
- /etc/path_to_inst.save,
- device nodes in `/devices`,
- either, `/etc/vfstab`, or device links under `/dev/dsk` and `/dev/rdsk`.

The first and the last must be done manually while the rest can be fixed with `devfsadm` utility (Solaris 8 or later). This is how to achieve that:
Assume that the old boot path was:
`/pci@0,0/pci9004,7881@13/sd@0,0:a`,
where:

- `pci@0,0` is the architecture of the disk controller and its address,
- `pci9004,7881` is a device PNP identification, consists of a vendor and part number,
- `13` is an IRQ level,
- `sd` means SCSI driver (you may have ata, cmdk or eisa here),
- `0,0:a` is the address of the Solaris root file system (slice) on the Solaris fdisk partition.

The boot path represents a subdirectory in `/devices`, Particular parts of the boot path change depending on what has changed in physical connection. For instance, when the controller is moved from one slot to another on the same motherboard the IRQ may change only.
The first question that arises here is *what is the current boot path*? It can be found with DCA. When the boot device is selected, the suitable boot path is displayed. Yet better method is to mount the root slice and check the full path associated to the disk device node used (the one under `/dev/dsk`).
The step by step procedure would look as follows:

1. Boot the system from the Solaris installation disk (in case of Solaris 8 use "Software 1 CD") to single user mode, that is, at the prompt to select between Interactive and Jump Start installation enter `b -s`. A shell will follow after this.
2. Set terminal type to AT386 to avoid problems with `vi` editor. Type in:

```
   TERM=AT386; export TERM
```

3. Mount root `/` file system:

   ```
   mount /dev/dsk/c0t0d0s0 /a
   ```

   The command is for the first disk on the first SCSI controller. It varies from system to system. Consult Disks, partitions and slices under Solaris for details.
4. Determine the slice number of `/usr` file system:

   ```
   grep /usr /a/etc/vfstab
   ```

   You will see a line:

   ```
   /dev/dsk/c0t0d0s4   /dev/rdsk/c0t0d0s4    /usr   ufs   1   no   logging
   ```

   which means that the number of the `/usr` slice is 4.
5. Mount `/usr` file system:

   ```
   mount /dev/dsk/c0t0d0s4 /a/usr
   ```

6. Rebuild `/a/devices` and `/a/dev`:

   ```
   /usr/sbin/chroot /a /usr/sbin/devfsadm -C -v
   ```

   ```
   /usr/sbin/chroot /a /usr/sbin/devfsadm -v
   ```

   The first command cleans up invalid entries, the second creates new ones. `-v` stands for verbose mode. If you prefer to see first what is going to be changed without actually changing anything add `-s` option.
7. Either, modify `/a/etc/vfstab`, or recreate soft links under `/a/dev/dsk` and `/a/dev/rdsk`, so that `vfstab` entries match appropriate slices on appropriate disks on the system. The second is more risky, but all customizations that involve disk device names are retained. Make sure that `/a/dev/dsk` links match `/a/dev/rdsk` links. Note that all those soft links can be removed and recreate with `devfsadm` as above safely.
8. Unmount the hard disk

   ```
   sync
   ```

   ```
   umount /a/usr
   ```

   ```
   umount /a
   ```

9. Try to reboot from hard disk.

   ```
   init 6
   ```

Note: Backup all of your data prior any modification to the system hardware which have direct relation with the hard disk, i.e. motherboard, disk controller including IDE or SCSI adapter.

III.2 NT Loader and `boot.ini` file

NT boot process

Windows NT system may be booted only from a primary partition or a floppy disk, however NT operating system files may be stored on either, a primary partition, or a logical volume. It does not modify Master Boot Code stored in the MBR of the boot disk, and uses standard BIOS method for loading Volume Boot Sector. Itself, Windows NT, does not require its primary partition to be active, however, when default Master Boot Code is used, that partition must be active to boot. This property of NT may be used only with other boot managers, such as Solaris boot manager, Linux Loader or OS-BS FreeBSD boot manager.

Allowed file systems on the NT primary partition are FAT-16 and NTFS for NT 4.0, additionally FAT-32 for W2K. The following files are used in the early phase of Windows NT boot process:

1. `ntldr`,
2. `ntdetect.com`,
3. `boot.ini`,
4. (optionally) `ntbootdd.sys`.

All those files are placed on the root of the boot partition, not necessarily the same partition, where NT operating system files reside. In particular they may be placed on a boot floppy disk. The last file on the list: `ntbootdd.sys` is optional and absent in most cases. File `ntldr` contains the main boot program. In opposite to `msdos.sys` - MS DOS boot program - `ntldr` need not to reside at the first sectors of the boot partition, and may be copied without the use of special methods. This applies also to all previously listed files.

NT Loader allows to load not only Windows NT, but almost all operating systems for IBM PC compatibles. To load other than Windows NT operating system, a file containing the VBS or Master Boot Record specific for that operating system is required. When it is the VBS or MBR depends on the behavior of the operating system and is explained in the latter section. User chooses operating systems from a text menu displayed by NT Loader. The NT boot process, including the menu for operating systems, may be configured within NT Loader's configuration file: `boot.ini`.

`boot.ini` file

`boot.ini` file contains configuration of the NT boot process. It is a regular Windows INI file, that is, it consists of sections which in turn contain variables we can assign values to. Its read-only flag is on by default, so you may need to switch it off before editing. Only part of options of that configuration file will be discussed in this document. For further details see [MS1] or [Russinovich]. An example of `boot.ini` file:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(1)partition(1)\WINNT

[operating systems]
multi(0)disk(0)rdisk(1)partition(1)\WINNT="MS Windows NT 4.0"
multi(0)disk(0)rdisk(1)partition(1)\WINNT="MS Windows NT 4.0 [VGA mode]" /basevideo /sos
C:\solaris.bin="Sun Solaris"
C:\dos.bin="MS DOS 6.22"
C:\floppy.bin="Boot floppy A:"
```

It is usually divided into two sections *boot loader* and *operating systems*. The meaning of variables of the first one is as follows:

default

Specifies the default operating system that will be loaded if a user does not make any choice in NT Loader menu.

timeout
Specifies time in seconds, to wait before *default* operating system is loaded.

Entries in the section *operating systems*, according to its name, configure NT Loader to boot operating systems and define NT Loader menu items. Each entry in this section is of the form:

location **=** *menu-item* { *option* }

where

location
Specifies the location of either, NT System Root, or the file containing the boot code of other then NT operating system. Accordingly, two different syntaxes are used, which we explain further.
menu-item
Text displayed in the NT Loader menu.
option
A switch that controls NT boot process (cf. [Russinovich]). It is possible to specify either none, one, or more switches in one entry. Commonly used are:
`basevideo`
Standard VGA display driver is used in GUI mode
`sos`
Print information about what drivers are being loaded as the system boots.
`win95`
Emulate the boot process of Windows 9x and load Windows 9x.
`win95dos`
Emulate the boot process of Windows 9x and load DOS.

Only 8 items may be displayed in the NT Loader menu, so only the first 8 entries will be available on this menu.

Booting NT

To boot Windows NT, an entry in the section *Operating Systems* of `boot.ini` file must contain *location* which specify the location of NT System Root. NT System Root is a directory where the files of Windows NT operating system reside. The NT environment variable `%SystemRoot%` points to that directory. NT operating system may be stored on a primary partition, as well as on a logical volume. When pointing to NT System Root the syntax for *location* is as follows:

(**multi(** *X* **)** | **scsi(** *X* **)**) **disk(** *Y* **) rdisk(** *Z* **) partition(** *W* **)** *win-nt-dir*

Parameter **multi** is valid for both IDE and SCSI hard disk drives, and indicates to Windows NT that it should rely on the computer's BIOS to boot the system. This means that the NT Loader will be using interrupt Int 13h calls to find and load the kernel of the system and any other files needed to boot Windows NT. In this case the *X*, *Y*, *Z*, and *W* parameters have the following meaning:

- *X* is the ordinal number of the adapter and should always be 0 (zero) since BIOS only identifies a single disk controller through Int 13h.
- *Y* is always 0 (zero) since BIOS specifies this parameter through Int 13h.
- *Z* is the ordinal for the disk on the adapter and is a number between 0 and 3, with limitations described below.
- *W* is the partition number. All partitions receive a number except for type 5 (Extended DOS) and type 0 (unused) partitions, with primary partitions being numbered first and then logical volumes. The first valid number for *W* is 1, as opposed to *X*, *Y*, and *Z* which start at 0 (zero).

When **multi** is used the following limitations apply:

1. In a pure IDE/ATA system, NT Loader may boot the system from one of up to four hard disk drives connected to the first dual-channel controller.
2. In a pure SCSI system, NT Loader may boot the system form the first two drives on the first SCSI controller, that is, the controller whose BIOS loads first.
3. In a mixed SCSI and IDE system, NT Loader may boot the system only from IDE/ATA drives on the first controller.

Parameter **scsi** indicates that NT Loader will load a boot device driver and use that driver to access the boot partition. The device driver used is in the file `ntbootdd.sys`, which is a copy of the device driver for the drive controller in use. In this case the *X*, *Y*, *Z*, and *W* parameters have the following meaning:

- *X* is the ordinal number of the adapter as identified by the `ntbootdd.sys` driver.
- *Y* is the SCSI ID of the boot disk.
- *Z* is the SCSI logical unit number (LUN) of the boot disk. This number is almost always 0 (zero).
- *W* is the partition number as above.

To determine the value of the *X* parameter, in case of multiple adapters that use different device drivers, only those adapters should be counted which are controlled by `ntbootdd.sys`. In that situation, to boot Windows NT from a hard disk on the adapter not controlled by `ntbootdd.sys`, the file `ntbootdd.sys` should be changed for the copy of appropriate device driver (cf. [MS1]).

Summarizing, the part of the *location* string to the backslash specifies the controller, the hard disk drive and the partition, where NT operating system files are kept. Hence, the rest of that string - *win-nt-dir* - is always the path to the NT System Root directory with the drive letter (and colon) truncated. Usually it is `WINNT`.

Booting other than NT operating system

To boot other than NT operating system, *location* of the entry in `boot.ini` file must specify the file containing the boot code of that operating system.

As we mentioned earlier, the boot code of other than NT operating system is its MBR or VBS. Generally, if the operating system uses its own specific code in the MBR, in other words if it overwrites during installation the Master Boot Code, not only the Master Boot Table, the boot code is its MBR. Otherwise, the boot code is the VBS of the boot partition of that operating system. The boot code for DOS, Windows 3.x and Windows 9x is in suitable VBS, for Solaris it may be either in Solaris MBR or VBS, for Linux it is there, where LILO was installed.

The boot code must be stored in a plain binary file. There is are quite many programs, either free or commercial, to manipulate MBRs and VBSes. The free program Boot reads MBR or VBS from a hard disk drive and stores its image in a 512 byte long file. The also performs reverse operation, that is, restores the MBR or VBS from a file. It is useful for backup purposes. Another freely available program [BootPart] creates special boot block files that may reboot any primary partition on any hard disk drive on a machine, even a floppy disk. It also automates editing `boot.ini` file, restores the VBS of the DOS, Windows 9x and NT boot partitions.

The boot code image files should be placed in the root of the boot partition, where `ntldr` and `boot.ini` are placed. Note that it need not to be a NT partition, but any primary FAT-16, FAT-32 in case of W2K, or NTFS partition. The syntax for *location* in this case is as follows:

c:\ | c:\ *file-name*

file-name is the file name of the boot code image file. It may be omitted when booting MS DOS, the boot code is stored in the standard file `bootsect.dos`, and there is no Windows 9x installed in the boot partition.

There is a serious limitation when using images of MBR or VBS. Only those operating systems may be booted this way, which boot partitions reside on the boot drive. It is imposed by the BIOS not NT Loader. There is no problem if we have a single hard disk drive. Problems arise when we want NT Loader to boot off the second drive. Say that the primary master IDE drive (C: in BIOS boot sequence) is a boot drive, and we want to boot with NT Loader an operating system which boot partition is placed on the secondary master drive (D: in BIOS lettering). VBS of that partition or MBR of the later drive is useless as it assumes the operating system is on the boot drive and reads from that one. The workaround is to use BootPart program. It does not simply make images of VBSes or MBRs but creates its own, special boot code which switches the boot drive on the fly if it is necessary. If we use BootPart in our example, the boot process could look as follows: NT Loader first loads a boot block created by BootPart, it then switches the boot drive to D:, loads appropriate VBS of boot partition and the boot process of the operating system begins.

Repair NT boot sector

When NT shares its boot partition with other operating system, e.g. Win9x, NT boot sector may be overwritten with fdisk or similar tool. This usually leads to problems with booting NT, at best, you can still boot Win9x. This is because NT boot sector contains its own specific code to load `ntldr` responsible of booting NT.

To repair NT boot sector, use either, BootPart, or NT installation software. In case of BootPart, boot to plain DOS and invoke:

```
bootpart
```

to see the list of all partitions in the system. Determine the drive letter of the partition which boot sector should be fixed. Note that BootPart shows `C:`, `D:` for disks, like BIOS does, not for volumes. For the first partition on the list, i.e. `C:`, type:

```
bootpart winnt boot:c:
```

Make sure the partition is active. The other method runs as follows:

1. boot NT installation floppies or CD,
2. choose repair damaged installation when prompted,
3. configure device drivers appropriately, and,
4. select "Inspect boot sector" only.

III.3 Linux Loader

Basics

Linux Loader, or shortly LILO, is a versatile Linux boot manager. It does not depend on a specific file system, can boot Linux kernel images from floppy and hard disks. On a machine with multiple operating systems installed, LILO can be used to boot other than Linux operating systems. LILO can be installed into VBS, or alternatively into MBR of any hard disk present on the system. When LILO is installed into MBR of the boot disk, it takes control over the boot process of the machine. In both cases, however, LILO allows to select the partition to boot off the operating system. It depends on the user's preferences which boot manager is a default one. If e.g. Solaris and Linux are configured adequately on a machine, endless cycling through Solaris and Linux boot managers is possible.

Linux device naming convention

Before we enter into the details of LILO configuration, some knowledge on how media devices are named in Linux is worth mentioning. All devices in Linux, as in Unix system, are accessed by (device) files. Floppy disks are addressed by `/dev/fd0` and `/dev/fd1`, which correspond to DOS drive A: nad B:, respectively. IDE/ATA hard disk drives are addressed as follows:

- `/dev/hda` -primary master,
- `/dev/hdb` - primary slave,
- `/dev/hdc` - secondary master,
- `/dev/hdd` - secondary slave.

Note that drive lettering is the same as in BIOS boot sequence setting. SCSI hard disk drives are addressed by `/dev/sda`, `/dev/sdb` and so on. Letter ’a’ means the first device in the SCSI chain, ’b’ the second one and so on.

Primary partitions on, both IDE/ATA and SCSI, hard disk drives are numbered from 1 to 4, so that e.g. the second primary partition on a secondary master IDE/ATA drive is addressed by `/dev/hdc2`. Extended partitions have no their addresses, as they are useless. Logical volumes on extended partitions are numbered from 5 up, so that e.g. the third logical volume on the extended partition on a primary slave IDE/ATA drive is addressed by `/dev/hdb7`.

Lilo configuration

LILO is fully customizable. Its configuration file is `/etc/lilo.conf`. LILO is initially preconfigured during the Linux installation process. With these, default settings LILO knows nothing about other operating systems installed on a machine, and can boot only Linux. In this situation it behaves simply as DOS or Windows 9x, which can’t boot anything but themselves.

The syntax of the `/etc/lilo.conf` file is simplified. It contains variables that can have values assigned to, and switches. Some variables implicitly determine sections. We discuss only those variables and switches which are necessary to configure LILO to boot Linux and other operating systems. For more details refer to LILO documentation distributed along Linux, [Veselosky] or [Skoric].

An example `/etc/lilo.conf` may look as follows:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
compact
prompt
timeout=50
image=/boot/vmlinuz-2.0.39
        label=linux
        root=/dev/hda3
        read-only
other=/dev/hda1
        label=nt
other=/dev/hda2
        label=solaris
```

This example file is taken from the system with one IDE/ATA hard disk drive with NT installed on the first primary partition, Solaris on the second one and Linux on the third one. LILO is installed into MBR and is used as the default boot manager. Used variables and switches have the following meaning.

boot
: Determines where LILO will be installed. Any device address associated with a fixed media or any partition address is a valid value. Usually it points to either MBR of the boot drive, or VBS of Linux boot partition. E.g. `/dev/hda` tells LILO to install into MBR of a primary master IDE/ATA hard disk drive. `/dev/hdb3` means third primary partition on a primary slave drive, and `/dev/sda5` means the first logical volume on the extended partition on the first SCSI drive.

map
: Variable used internally by LILO.

install

Specifies the file containing Volume Boot Code.

compact

Switch that makes LILO read the hard drive faster.

prompt

This is a switch that tells LILO to prompt a user at boot time to choose an operating system or enter parameters for the Linux kernel.

timeout

Specifies how long to wait at the prompt before booting the default operating system. Measured in tenths of a second. 50 means 5 seconds; 0 or not present variable at all causes LILO to wait endlessly or boot immediately as *prompt* switch is specified or not.

image

Specifies the name of a Linux kernel for LILO to boot, e.g. `/boot/vmlinuz-2.0.39`. This variable begins a section, and the lines that follow give specific parameters for this particular kernel. You may have up to sixteen image sections, but one should suffice. The first image listed in the file is the default, unless you specify otherwise.

label

The name that is used to identify this image at the LILO boot prompt. Typing this identifier will select this image. Usually set to `linux`.

root

Specifies the volume where the root (/) file system resides, in other words, the boot volume of Linux. Note that it may be either a primary partition or a logical volume. This file system will be mount at the boot time. If LILO is installed into VBS, this value should be equal to the value of *boot*.

read-only

This switch is usually present and causes the Linux kernel to initially mount the root file system as read-only and remount as read-write later.

other

This variable is optional and has similar meaning for other than Linux operating system as *image* has for Linux. It begins a section for such a system. Its value points to the boot partition of that operating system. LILO will simply load and execute the VBS of this partition at boot time.

label

Same as the label above. Specifies an identifier to refer to the operating system at LILO boot prompt.

Every time `/etc/lilo.conf` is modified, we have to apply changes to take effect. To do that run as root :

```
/sbin/lilo
```

Successful configuration of Linux and other operating systems specified in `/etc/lilo.conf` is reported by displaying "Added *label*". An asterisk indicates the default operating system to boot.

Using LILO

Though LILO realizes relatively advanced tasks, its usage is simplified. It is not menu driven, instead a prompt "LILO:" for user input is displayed. LILO then waits the amount of time specified in `/etc/lilo.conf` and if there is no user input, it loads the default operating system. Operating systems are selected by their labels specified in `/etc/lilo.conf`. To list all available identifiers TAB key may be pressed.

Additional parameters to Linux kernel may be given after the Linux identifier.

Recovering LILO

When Linux is installed prior other operating system, LILO that was written into MBR may no longer be available. This happens when e.g. DOS, Windows, including NT and W2K, or Solaris are installed after Linux. LILO might be wiped out not only during other operating system installation, but also by certain disk utilities or anti-virus software. In case we want LILO to manage the boot process from its very beginning, it needs to be restored. This is done the same way changes to LILO configuration are applied. So, Linux must be booted some other way, e.g. by Loadlin or Linux boot floppy, and `/sbin/lilo` must be run by root.

Keep in mind that this procedure restores Master Boot Code only, not Master Partition Table. That is in case of disaster, when entire MBR is lost, restoring LILO as described above will not fix the problem. In general this means that the data on the hard disk drive is lost. To avoid such problems MBR should be backed up after every modification.

Removing LILO

Discussing LILO and its configuration we should mention that it is possible to remove LILO from MBR safely. Before LILO is installed the original Master Boot Code is saved for future restoration. This feature may be used when one decides to uninstall Linux. It is done by running `/sbin/lilo` with option `-u`. If Linux can not be loaded we may also try DOS fdisk utility or Ranish PM to rebuild MBR.

III.4 Third party boot managers

OS-BS

OS-BS (ftp://ftp.freebsd.org/pub/FreeBSD/tools) is a FreeBSD boot manager. The main advantage of this boot manager is the size of its code, customizability and its price. It is a free software. It fits into MBR and does not require any additional partition to work. OS-BS can change active partitions on the fly. The choice of the desired operating system to boot is menu driven. Labels for operating systems and timeout are specified during OS-BS setup.

Ranish Partition Manager

Ranish PM (http://www.ranish.com/part/) offers not only a boot manager but also partition management for free. There are even two boot managers: a compact one which fits into MBR, and an advanced one which requires some free space for its own partition. With those boot managers it is possible to boot operating systems from multiple hard disk drives. Ranish PM can hide and activate partitions on the fly. It can also change logical volumes to primary partition, which makes them boot-able.

Ranish PM is capable of creating most of known partitions including Solaris x86 and Linux partitions. It can format, clear, resize empty partitions, move and copy FAT-16 and FAT-32 partitions. It is possible to manually adjust partitioning data in MBR. The utility is able to recover MS DOS, Windows 9x, NT, W2K, PC DOS and DR DOS Volume Boot Code (partition boot block). Also manual changes to VBS of FAT-12, FAT-16 and FAT-32 are possible.

GRUB - GRand Unified Bootloader

GRUB (http://www.gnu.org/software/grub/) is an attempt to produce a bootloader for IBM PC-compatible machines that has both the capability to be friendly to beginning or otherwise non-technically interested users and the flexibility to help experts in diverse environments.

The GRUB project introduces Multiboot Standard proposal, which specifies general rules of behavior of boot processes used in operating systems. It is an attempt to cleanup and standardize PC boot methods.

Complexity is the price of rich functionality and convenience of GRUB. However, this complexity is hidden behind a simple text configuration file. GRUB requires additional space for executables and configuration files so, it does not fit the MBR. It offers menu driven, as well as, flexible command line user interface. The original feature of GRUB is geometry translation done independently of the BIOS and transparently for the operating system.

GRUB is free, distributed under GNU GPL license.

An axample entry in `/grub/menu.lst` to boot Solaris may look as follows:

```
title Solaris
        rootnoverify (hd0,1)
        chainloader +1
        makeactive
        boot
```

Of course, `(hd0,1)` should be modified according to the actual partition scheme. In this example, `hd0` mean the first hard disk drive, and the number `1` means the second partition on that drive. Hard disks, as well as partitions are counted from `0` in GRUB convention. `(hd0,4)` specifies the first extended partition on the first hard disk. Command `makeactive` tells GRUB to make the Solaris partition active which is required to boot Solaris.

System Commander

System Commander (http://www.v-com.com/) is a very sophisticated boot manager, which can boot all operating systems for IBM PC compatibles. It is a commercial software.

System Commander installs into MBR and requires FAT-16 partition to store its code and configuration data. It can change active partitions and hide partitions on the fly. Its OS Wizard prepares system for installation of a new operating system. Newly installed operating systems are automatically detected and included to the main menu. It is possible to setup users with different privileges for booting operating systems, so that the access to the machine is controlled by passwords.

IV. Practical howto

IV.1 First things first

Planning partitions

The following factors should be considered when planning partitioning of a hard disk:

1. **Older BIOSes can access only the first 1024 cylinders** of the hard disk drive. Therefore, it is not able to boot partitions that start past 1024-th cylinder. The workaround is to make boot partitions for all installed operating systems small enough to fit them all within the first 1024 cylinders.
   - Solaris can be installed into a primary partition only. It is not possible to have Solaris UFS file system on a logical volume. Also, there is no use in having more than one Solaris primary partition on the same disk, especially, when we want to install an alternative operating systems. The conclusion is as follows: if you have an old BIOS, cannot split your Solaris installation into two hard disks, and want to install some additional operating system on the same hard disk, partition the hard disk so that Solaris partition is the last primary partition that begins (but not necessarily ends) within the first 1024 cylinders.
   - Linux Loader (LILO) requires Linux root `/` and `/boot` file systems, where the kernel image resides, to be within the first 1024 cylinders. Those file systems may be put into a single fdisk partition not exceeding 0.5 GB. Other file systems like `/var`, `/usr` and `/home` may be put into logical volumes. Entire Linux distribution may be installed into logical volumes.
   - Windows 9x, NT, W2K and XP need less than one megabyte for their boot files. System may be installed into a logical volume on an extended partition beyond 1024-th cylinder. This way we also save the limited number of primary partitions per a hard disk, and use one for multiple MS operating systems. Even if we decide to dedicate a primary partition to every installed operating system it needs not to exceed 1GB for Windows 9x, NT, and 2 GB for

W2K. User data and applications could be placed in logical volumes.

A **modern BIOS supporting INT 13h extensions** can access cylinders greater than 1024. If both Master Boot Code and Partition Boot Code utilize the INT 13h extensions the 8GB or 1024 cylinder limitation is overcome.

- Solaris 8 does not suffer 1024 cylinder limitation. Its partition may begin past 1024-th cylinder, and as long as Solaris Master Boot Code or other boot manager capable of loading *VBS* beyond 1024-th cylinder is used, there is no problem.
- LILO since 21.4 are able to boot from cylinder 1024 and beyond.

2. **Windows NT boot files should reside within the first 1024 cylinders**. This limitation is caused by NT boot sector which uses BIOS Int 13h calls to access boot files. Theoretically, boot files can be placed at the very beginning of the boot partition, which allows to make boot partitions that exceed 1024-th cylinder. It is, however, recommended not to make NTFS boot partitions that exceed 1024-th cylinder [MS2].
3. **Safety**. When installing two or more operating systems, it is most safe to install them into separate, dedicated hard disk drives. This will prevent data loss of one system when another accidentally crashes the hard disk. Also repartitioning of one system does not influence the other. In this case only one of the hard disks works at the time.
4. **Performance**. In contrast to the above, safe model, getting all hard disk drives working for one operating system may improve performance significantly. It can be achieved by partitioning hard disks so that every operating system uses at least one volume on each hard disk.

   The performance depends also on how long distance hard disk heads have to go to access requested data. So, frequently used files like swap should be stored near the middle of the hard disk, and volumes that belong to one operating system should lay side by side.
5. **Provide a file system for data exchange** between operating systems. It must be supported by all systems. Most broadly supported is FAT-16.
6. **Solaris 7 and earlier do not support IDE hard disk drives larger than 8 GB**, that is, they may access only the first 1024 cylinders of such a disk. Since the problem is on Solaris `fdisk` side, the workaround is quite straightforward:
   1. Using Linux `fdisk` program on Linux installation floppy, or Ranish PM create a Linux swap primary partition. It is also possible to create e.g. DOS partition and change its ID to 0x82 (Solaris ID) with a favorite hex-editor and `BOOT` program, or a disk editor.
   2. Boot Solaris from e.g. its installation CD.
   3. Run `format` utility and create slices on Solaris partition.
   4. Run `newfs` to construct file systems on all created slices.
7. **Solaris 2.6 and earlier do not support FAT-32**.
8. **Windows NT up to the Service Pack 4 does not support hard disk drives larger than 8 GB**. The only way to solve this is to install updated in SP4 ATAPI driver. Separate driver may be downloaded and installed, as well as entire SP4.

   Actually, the boot partition is limited to 4GB only when NT is installed into an empty hard disk. If an NTFS partition is created in advance, the boot partition might be as large as possible.
9. **MS DOS, Windows 3.x and Windows 95 up to OSR2 do not support hard disks larger than 8 GB**.
10. **FAT-16 partitions may not exceed 2 GB**.
11. **Windows fdisk creates Extended LBA partition of ID 0x0F** on disks larger than 32 GB, instead of extended partition of ID 0x05. This may cause problems if another operating system, that is to be installed on the same disk, does not support partitions 0x0F. There are problems during installation of Red Hat 7.1 (and probably older versions), though Linux in general supports partitions 0x0F.
12. **Windows 2000 can not create FAT-32 partitions larger than 32 GB.**
13. **Multiple extended partitions are not broadly supported**, though NT supports such.
14. **Solaris fdisk reorders entries in the fdisk table in MBR.** It may happen that some fdisk utility leaves entries in the fdisk table unsorted, or there are higher slots occupied while lower are left free. If that is the case, Solaris fdisk may reorder the partition entries, which may cause that previously installed operating systems will not boot properly. This affects Linux and BSD. Ranish

PM lets to monitor and change the order of partition entries in the fdisk table.

How to preserve an existing operating system?

If an operating system is already setup and running, but occupies entire hard disk, while we want to install another operating system and do not want to loose the existing one, there are a couple of possible ways to go:

1. **Get additional hard disk drive**. IDE drives are quite affordable nowadays.
2. **Resize the partition** with Ranish PM or some other partition manager if it supports the file system used. This operation is not supported for all file systems. FAT-16, FAT-32, NTFS and Linux ext2fs can be resized with newest Partition Magic. Solaris UFS cannot. Resizing is rather unsafe, the system and data should be backed up prior resizing. The partition should be defragmented prior resizing.
3. **Backup the user data, then reinstall the operating system and software** from scratch into a smaller partition, finally restore the user data. This solution is most safe and reliable, though most laborious. Every file system for every operating system may be "resized" this way.
4. **Backup the entire software and data, then restore them into a smaller partition**. This solution may fail for some operating systems, e.g. Solaris or Linux. Copying file by file is error prone and requires all file systems to be restored properly.

   For Solaris, in such a situation use `ufsdump`/`ufsrestore` utilities. They can be used to either shrink or enlarge all types of slices, including *root*. To use that method a storage should be provided that fits file of size equal to the size of slice contents.

   All MS operating systems may be backed up as a collection of files, and then restored into a smaller or larger partition. The change of file system is also possible, i.e. the system installed on FAT-16 may be restored on FAT-32 or NTFS and conversely. In case of Windows 9x, NT it is important to restore not only files but also the crucial drives letters, i.e. if applications were installed into D: they have to be restored into D:. For W2K crucial mount points must be restored additionally. User data and archive file systems usually need not to be restored to the same, logical location. To copy file by file of Windows 9x, NT or W2K, an extra operating system that can handle that must be used. That is, if we intend to copy NT on NTFS file by file, the system prohibits opening some files (e.g. registry files) and we have to boot additional copy of NT or W2K to do the job. This requires additional hard disk drive and plenty of storage room.

   Generally NT 4.0 is more robust than W2K. It is not so picky about drive letters and device IRQs assignment as W2K.

IV.2 Installation

Hardware connection

Before we present examples of a system setup we would like to discuss some basic, though general, hardware issues required to install software successfully.

All SCSI devices on the system must be connected so that they form a chain terminated on its both extreme ends. All devices that lay between the ends must have their termination disabled. Termination is done with a jumper or micro switch on internal devices, and by a terminating plug on external devices. Modern SCSI adapters terminate themselves automatically.

Each device attached to the SCSI host bus adapter, as well as the host adapter itself, must have a unique SCSI ID number (target number, usually 0 through 7 or 0 through 15). Those IDs are set manually with jumpers or micro switches, or automatically by the host adapter BIOS if SCAM (SCSI Configured AutoMatically) protocol is supported by all devices in the chain.

Almost every modern system board is equipped with two channel *EIDE* (Extended IDE) controller (older boards have only one channel controller). To every channel up to two devices may be connected, which makes four IDE devices possible.

In practice, IDE devices that share a channel, in contrast to SCSI devices on the same chain, always work with the speed of the slower device. E.g. a CD-ROM that supports only PIO mode 3 connected as a slave, slows down a modern master hard disk supporting PIO mode 4. To improve the performance connect hard disks to the primary channel and other devices like CD-ROMs, CD burners, DVD, ZIP drives to the secondary channel.

BIOS settings

Currently available hard disk drives on the market go far beyond 8 GB barrier. As for now there is no BIOS geometry translation that might overcome that limitation. Modern operating systems like: Solaris 8, Linux, NT 4.0 + SP4, W2K, or Windows 98 override BIOS disk routines and use Int 13h extensions. This allows to use such large hard disks. On the BIOS level, however, the problem still persists. BIOS cannot access cylinders past 1024-th one, and therefore is not able to boot partitions past 1024-th cylinder. 1024 cylinders correspond to 504 MB or 7.88 GB as LBA is disabled or enabled in BIOS. This is meaningless on systems where operating systems are installed on separate hard disk drives, and such a drive may be set to Normal, LBA or Auto. Large is not recommended. When multiple operating systems are installed on a single drive LBA should be enabled.

Note: Do not change BIOS geometry translation after an operating system has been installed, or that system will not boot any more.

Solaris keeps track on physical connection of devices. Sometimes, there are situation in which such a behavior might be considered a serious limitation, but actually, this makes Solaris stable and reliable. In case of DOS or successors, when a new hard disk drive is attached one always wonders how drive letters will be rearranged. NT is a bit smarter with its *NT Disk Signature* but suffers the same weakness.

During installation of a new operating system on a separate hard disk drive, it is safe to disconnect the unused one. Swapping hard disks is not recommended. This causes physical addresses to be different during installation and after it. When you usually boot off primary master and want that hard disk to be turned off during installation on a slave or secondary drive, change *BIOS boot sequence* setting instead of swapping drives.

Note: The general rule for temporary booting off other than usual drive is: do not swap cables, reconfigure your BIOS instead. Most modern BIOSes are capable of setting boot sequence (refer to your BIOS manual for more details).

Available values of the BIOS boot sequence entry, that is, boot devices and their order, vary from BIOS to BIOS. Letters C, D, E and F in this entry refer to IDE hard disk drives. To boot from SCSI drive, SCSI option should be selected in BIOS boot sequence. In some BIOSes there is no SCSI option available in this entry, but there is *HDD sequence SCSI/IDE* entry instead. It determines which controller SCSI or IDE should be searched first during boot up. The order of SCSI devices is setup in the SCSI controller BIOS. Older BIOSes may have no option to boot from SCSI at all.

Prepartitioning

Most operating systems are capable of creating and deleting partitions during their installation. This may save headaches with searching for an fdisk utility to cleanup disk or to create partition prior installation. While cleaning disk prior installation is a good habit, prepartitioning may lead into troubles when incompatible fdisk or format utility is used. Generally it is better however, to rely on the fdisk that comes with the operating system than a third party utility. On the other hand prepartitioning lets to plan disk layout better and assign necessary space for different volumes.

There are however exceptions to this rule. When Solaris will be installed along another operating system, it is better to create Solaris partition prior Solaris installation. This is due to specific CHS geometry translation used in Solaris, different than in other PC operating systems (see). A Solaris fdisk partition can be created easily with Ranish PM. The other method is to create any type of primary partition and change its partition ID to 0x82 in the fdisk table in MBR.

During installation of Solaris into an existing partition, you will be asked if it is a fresh installation or upgrade. This is because the contents of the Solaris partition is not verified.

For the sake of performance one should never let NT setup create an NTFS partition, because this will be created as FAT-16 partition and later converted to NTFS, with the worst cluster size of one sector, and the Master File Table (the file holding all the meta-data) created highly fragmented in all the free FAT-16 clusters found at the beginning of the partition.

Also boot partition of NT is limited to 4GB if NT setup program is used to create NT that partition.

It is recommended to format NTFS partitions with fdisk tool capable of doing so, eg. latest Partition Magic, or from a running NT installation by mounting the hard disk into another PC. The cluster size should be 8 sectors. With NT command line format utility this is done by typing:

```
format /A:4096 /FS:NTFS
```

IV.3 Disks, partitions and slices under Solaris

Disk addressing conventions

All devices in Unix, in particular in Solaris, are accessed through *special files*. In Solaris all actual special files are placed below `/devices` directory. For convenience `/dev` directory contains symbolic links to these device special files. *Disk entries*, i.e. symbolic links to disk device special files are placed in `/dev/dsk` and `/dev/rdsk`, depending on whether they refer to block devices or raw devices. In this context, by disks we mean hard disks, CD-ROMs, ZIP drives and the like (see `disks` man page for the exact technical definition). Disk entries have the following syntax:

c*C* [**t***T*] **d***D* (**s***S* | **p***P*)

where *C, T, D, S*, and *P* are non-negative decimal numbers. Actually, a disk entry is a *complete* address of a *slice* or a *partition*, and consists of two parts: logical device address formed by **c, t, d**, and slice **s** or partition **p** address on that device.

- *C* is the logical controller number, i.e. the number of an IDE adapter (actually its channel), SCSI host bus adapter or the like. It is assigned by `disks` utility, or by `devfsadm` on newer systems. First time the utility is run during Solaris installation, and consecutive numbers are assigned for particular disk controllers, so that IDE controllers take precedence before SCSI ones. The utility may be invoked later during system reconfiguration or manually by the superuser to add new disks. Then, consecutive numbers are assigned to the new controllers found on the system. E.g. if Solaris is installed on a machine with only primary IDE channel being in use and one SCSI adapter they have numbers 0 and 1 respectively, when a new IDE device is attached to a free secondary channel it will get the number 2.
- *T* is the target (*logical unit number*) of the SCSI device, i.e. the identification number of the device in a SCSI chain. Obviously, IDE disks have no **t** specified in their addresses.
- *D* is the number of the disk attached to the controller. Note that in case of an IDE adapter, two IDE devices may be attached to a single channel, then 0 represents the master device, and 1 the slave one.
- *S* is the number of the slice within a Solaris partition. Each Solaris partition can have up to 16 slices numbered 0 through 15. There are 21 special files per a disk device, 0 through 15 correspond to slices within the first Solaris partition on a disk (see Slices and VTOC).
- *P* equal to 0 means entire disk, values 1, 2, 3, 4 represent consecutive partitions on a disk. These 5 entries correspond to disk special files 16 through 20.

First note that a pair **c, d** form an IDE device address, and a triple **c, t, d** form a SCSI device address. Note also that presented addressing convention does not allow to access logical volumes within extended partitions. Some utilities like `mount` or `mkfs` make use of *disk-entry*:*volume* addressing convention, where *disk-entry* specifies either the entire disk (...**p***0*) or a partition (...**p***P*);

volume is either a number 1 through 24 or equally a letter c through z.

Under Solaris 9 it is recommended to use the `/dev/dsk/c0d0p0:1`, `/dev/dsk/c0d0p0:2`, `/dev/dsk/c0d0p0:3`, and so forth notation, where the number after the colon indicates the first, second, third, and so forth FAT (FAT-16 or FAT-32) volume. The pcfs driver automatically numbers any FAT volume it sees on the disk. This notation also works for primary FAT fdisk partitions. This addressing schema corresponds to DOS addressing using drive letters.

To understand how it works, consider the following example.

1. Primary - FAT-16	p0:1	p1:1
2. Primary - FAT-16	p0:2	p2:1
3. Primary - NTFS	-	-
4. Extended		
4.1. FAT-16	p0:3	p3:1
4.2. FAT-16	p0:4	p3:2

Table IV.3.1: Example partitioning and Solaris addresses.

Note: The above example was prepared and tested only under Solaris 9 and may not apply to earlier versions.

Slices and VTOC

Every Solaris partition is divided into smaller chunks called *slices*. They resemble x86 fdisk partitions of hard disk drive. Up to 16 slices may be defined per a Solaris partition. Since there are only 21 device special files per a disk, this imposes certain limitation. Only slices within the first Solaris partition can be addresses by **s**0 through **s**15. Note however that there are still up to 4 Solaris partitions per a disk allowed.

Some slices are reserved or have special meaning. Slice 2 is a *backup* slice representing entire Solaris partition. Slice 8 is a *boot* slice representing the first cylinder of the Solaris partition, containing partition boot block `pboot`, Solaris disk label, VTOC and boot manager program `bootblk`. Slices 2 and 8 can are unmountable. Slice 9 is an *alternates* slice that uses 2 cylinders. Slices 2, 8 and 9 are reserved and should not be redefined. Slice 0 is usually a *root* slice containing the root file system. The following tags may be assigned to slices: *swap*, *usr*, *stand*, *var* and *home* (see `fmthard` man page).

There are 4 possible states of a slice. They are determined by two flags specified when the slice is defined. One determines if the slice is mountable, the other if it is read-only.

Slices can be defined using `fmthard` or interactive `format` utility. `format` always shows 0 through 9 slices, including undefined ones. To define a slice we need to specify its number, tag, flag, starting sector relative to Solaris partition and its size.

All the information about slices is kept in a *Volume Table Of Contents*, shortly VTOC. Together with Solaris disk label VTOC occupies the second and third sector of a boot slice - the first cylinder of a Solaris partition. It can be printed out with `prtvtoc` utility. It is recommended to print and save all VTOCs after the system is setup to have them handy in the case of emergency.

Creating and mounting additional UFS file systems

When we add some new hard disk partition to Solaris it is recommended to use the native Solaris UFS file system. Though we lose compatibility with other operating systems our machine is running, we benefit reliability and performance.

First thing is to create Solaris partition. If it would be the second Solaris partition on our drive, we will need to apply some trick described in details later. Assume that we have two IDE drives on the same primary channel: master `c0d0` where Solaris is installed, and slave `c0d1`. Run `fdisk` program to create Solaris partition:

```
fdisk /dev/rdsk/c0d1/p0
```

This task is relatively easy since the above command runs `fdisk` in the interactive mode. It allows to delete partitions that we no longer need. Next step is to create slices on the new Solaris partition. Run program `format`, which also offers the interactive mode. We will probably see something like this:

```
AVAILABLE DISK SELECTIONS:
       0. c0d0           /pci@0,0/pci-ide@4,1/ide@0/cmdk@0,0
       1. c0d1           /pci@0,0/pci-ide@4,1/ide@1/cmdk@0,0
Specify disk (enter its number):
```

Solaris automatically recognized its own partition on `c0d1` and you do not need to remember which one it was. After selecting 1 a menu with `format` options shows up. `partition` sub-menu is responsible for slice management. `format` shows only 0 through 9 slices though 16 are allowed. Keep in mind that slices 2, 8,9 are reserved, and the 0-th slice should begin at sector 3. Command `label` should be invoked before quiting `partition`.
When the new Solaris partition is already sliced, type:

```
newfs /dev/rdsk/c0d1s0
```

to construct UFS file system on the 0-th slice. You need to construct file system on every slice you defined. Last step to perform is mounting:

```
mount /dev/rdsk/cod1s0 /export/home1
```

This will mount 0-th slice on `/export/home1` (`/export/home1` must exist prior mounting). To have new slices mounted at boot time add them to `/etc/vfstab` (read vfstab man page).

Two Solaris partitions on the same drive

When we decide to remove one of the operating systems or just free one of the partitions and dedicate it for Solaris we can get into troubles if that free space is on the same hard disk drive where Solaris is installed. The problem is the second Solaris partition can not be addressed using usual Solaris addressing schema.
Solaris partition can not be resized, besides, the hole and the Solaris partition need not to be placed side by side. One workaround is to backup all the data, repartition and restore data. We can even copy file systems as a whole, but a spare hard disk drive is necessary.
The simple solution makes use of the fact that besides Solaris slices we can mount 3 partitions. Consider the following partitioning:

1. p1 - primary partition where Solaris is installed,
2. p2 - primary partition with other operating system,
3. empty space,

on the drive `c0t0d0`. We begin with creating a new Solaris partition within an empty space, which will become p3:

```
fdisk /dev/rdsk/c0t0d0p0
```

`fmthard` and `newfs` use slices as addresses, therefore to modify VTOC on p3 we need to hide p1 first. Reboot into DOS and run your favorite disk editor capable of modifying MBRs e.g. Ranish PM to change the identifier of p1, so that Solaris would not recognize it. For safety make a backup of the MBR before. Now, reboot Solaris from its installation CD, stop the installation after the Open Windows session is started, and open a terminal window. At the moment p3 is the first Solaris partition on `c0t0d0`, so run `format` and create exactly one slice 0 in p3, so that it starts in 0-th sector and fills entire p3, much like slice 2. New UFS file system is created with:

```
newfs /dev/rdsk/c0t0d0s0
```

New file system is ready to mount. Restore the identifier of p1 back to 0x82, reboot Solaris as usual, and mount p3:

```
mount /dev/dsk/c0t0d0p3
```

p3 acts like a single slice and that is the trick. The interesting question arises: where is the VTOC of p3 now?

Mounting FAT partitions

When along Solaris there is an alternative operating system, it is convenient to have FAT volume for data exchange. While FAT-16 is supported by almost all x86 platforms, FAT-32 is not even supported by all MS operating systems e.g. NT 4.0. There are two FAT-16 and two FAT-32 variants - one for smaller disks and the other for large disks of capacity greater than 32GB. In Solaris terminology all those file systems are called *pcfs* (FAT-32 is supported starting from Solaris 7). Consider the following example of partitioning:

1. Primary - NTFS - Windows NT
2. Primary - UFS - Solaris - *active*
3. Primary - Linux ext2fs - Linux
4. Extended
   4.1. FAT-16
   4.2. Linux swap
   4.3. Linux ext2fs
   4.4. NTFS
   4.5. NTFS

To access FAT-16 partition under Solaris it must be mounted. Type as root:

```
mount -F pcfs /dev/dsk/c0t0d0p0:1 /mnt
```

and FAT partition content will be available below `/mnt` directory. The above command is valid for SCSI hard disks. For IDE drives drop `t0`.

File system mounting, as in the example above, requires superuser intervention and the file system will be unmounted after system reboot. To mount FAT partition permanently in directory `/share`, create this directory first, and add the following line to `/etc/vfstab` file:

```
/dev/dsk/c0t0d0p0:1     -       /share  pcfs    -       yes     -
```

Remember to end this line with a line feed character (break the line typing Return). This will mount FAT partition automatically during boot up. The same commands apply to both FAT-16 and FAT-32. Read mount_pcfs and vfstab man pages for more details.

There were problems reported with mounting FAT file systems within extended partition under Solaris 8. Tests, with Solaris 8 02/02, revealed that only two FAT volumes within an extended partition can be mounted, provided that they both are the very first volumes on that extended partition.

Under Solaris 9 the problem is fixed. See addressing conventions.

IV.4 Troubleshooting

Solaris is freshly installed on the second hdd and fails to boot, "Bad PBR" error message is reported

This is caused by a bug in Solaris 8 boot procedure. Solaris always tries to load its *VBS* (PBR) from the first physical hard disk drive in the system. If Solaris is booted from other than the first drive the error is reported.

The most recent release of Solaris 8, that is 02/02, also known as Update 7, seems not to suffer this problem.

Workaround

Change *BIOS boot sequence* settings so that Solaris is on the boot drive. If this also fails change hard disk cables so that Solaris is on the first drive (usually primary master *IDE*). In the later case reconfiguration of boot device may be necessary (see Reconfiguring boot device).

After Solaris is installed, other partitions on my hdd get unusable

This happens due to specific CHS geometry translation used in Solaris. It is incompatible with other operating system, fdisk and partitioning utilities like Ranish PM or Partition Magic.

Workaround

The workaround is to reinstall Solaris into existing partition, created earlier with e.g. Ranish PM, which is capable of creating Solaris partitions. It is also possible to create a partition of any type e.g. FAT-32 and change its ID (0x0B or 0x0C for FAT-32) to 0x82 with any disk editor (see details on the MBR).

When I try to boot Solaris it enters DCA and hangs there

Probably `auto-boot` eeprom setting is set to `false` or `boot-path` is wrong. See DCA.

I have changed motherboard (or SCSI controller, or cable connection) and Solaris does not boot anymore

During installation Solaris saves details of physical devices connected to the mainboard of the machine, that is, IRQs and device specifications necessary to attach proper software drivers. Whenever, one of those devices changes, the saved data needs to be updated accordingly. Refer to Reconfiguring boot device to fix the problem.

IV.5 Frequently Answered Questions

1. How to create a Solaris fdisk partition?
2. Is x86 Boot partition necessary?
3. How to avoid x86 Boot partition?
4. How to mount FAT file system in Solaris?
5. How to mount NTFS file system in Solaris?
6. How to mount Linux ext2fs file system in Solaris?

How to create a Solaris fdisk partition?

A Solaris fdisk partition can be created easily with Ranish PM. The other method is to create any type of primary partition and change its partition ID to 0x82 in the fdisk table in MBR, using a favourite disk editor .

Is x86 Boot partition necessary?

See x86 Boot partition subsection for details on this partition. In most cases it is useless.

How to avoid x86 Boot partition?

To avoid it run Interactive installation from "Software 1 CD" instead of Webstart installation from "Installation CD", and use fdisk to simply drop x86 Boot partition. When Solaris fdisk partition is created prior installation x86 Boot partition will not be created.

If the system is already setup simply remove that partition, use `installboot` command (see Restoring Solaris boot manager), and make Solaris partition active.

How to mount FAT file system in Solaris?

Refer to Mounting FAT partitions.

How to mount NTFS file system in Solaris?

NTFS file system is not supported by Solaris.

How to mount Linux ext2fs file system in Solaris?

Refer to

- ,
- .

V. Example installations

V.1 Single hard disk, Solaris, Windows NT

Overview

In this example there is a single one hard disk on the system. It might be either IDE or SCSI. Two operating systems are installed: Solaris and Windows NT. FAT-16 volume is created to provide data exchange between them. After the installation is complete, at the boot time a user is presented the Solaris boot manager menu to select which operating system to load.

Partitioning

The hard disk will be partitioned as follows (proportions are meaningless):

1. Primary - NTFS - Windows NT
2. Primary - Solaris UFS - Solaris - *active*
3. Extended
    3.1. FAT-16
    3.2. NTFS
    3.3. NTFS

Recipe

1. Connect hardware and configure BIOS (see IV.1 for details).
2. Install Windows NT. Create one primary partition with NTFS as a file system (see notes on the size). This will be C: drive.
3. Create Solaris partition. Be careful not to remove existing NT partition.

4. Install Solaris. Solaris boot partition becomes the active one. Solaris boot manager becomes the default, and despite Solaris does not recognize NTFS file system NT may be booted from its menu.
5. With NT Disk Administrator create an extended partition and logical volumes. Create at least one FAT-16 file system to enable communication between Solaris and NT. In case of Solaris 7 or earlier, FAT-16 file system must end before 1024-th cylinder to be accessible by Solaris. Additional NTFS volumes help in organizing NT user data and applications. They become D:, E: and so on.
6. Under Solaris, create `/share` directory, and add the following line to `/etc/vfstab` file under Solaris:

   ```
   /dev/dsk/c0t0d0p3:c     -       /share  pcfs    -       yes     -
   ```

   FAT-16 partition will be mounted automatically at the boot time (see notes on mounting FAT partitions). If the hard disk is IDE drop `t0`.

Remarks: It is also possible to setup Solaris before Windows NT. If that is the case, to boot Solaris either, activate Solaris boot manager, or install some third party boot manager. To activate Solaris boot manager make the Solaris partition active. Use DOS fdisk program on DOS boot floppy, or NT Disk Administrator.

NT Loader as the default boot manager

To make NT Loader the default boot manager that is able to boot Solaris some trick must be applied. The following programs on a DOS boot floppy disk will be necessary:

1. `boot.exe`,
2. [OS-BS 1.35] or [OS-BS 2.0 beta] setup program, that is `os-bs.com` or `osbs20b8.com`, respectively.

When the system is setup according to the description in the previous section, perform the following steps:

1. Boot system into DOS with DOS boot floppy.
2. Create an image file of the MBR. Issue the following command at the DOS prompt:

   ```
   boot /r /drive:0 mbr solaris.bin
   ```

3. Setup OS-BS boot manager. Run program `os-bs.com` (or `osbs20b8.com`), set NT as the only operating system, give it whatever label you like, set timeout to 1 second. Leave Solaris partition active and order OS-BS boot manager not to change it.
4. Restart the system and boot Windows NT.
5. Copy the file `solaris.bin` from the floppy disk to the root of the NT boot partition.
6. Edit `boot.ini` file so that it contains the following entries:

   ```
   [boot loader]
   timeout=30
   default=multi(0)disk(0)rdisk(0)partition(1)\WINNT

   [operating systems]
   multi(0)disk(0)rdisk(0)partition(1)\WINNT="MS Windows NT 4.0"
   multi(0)disk(0)rdisk(0)partition(1)\WINNT="MS Windows NT 4.0 [VGA mode]" /basevideo /sos
   C:\solaris.bin="Sun Solaris"
   ```

After the system is rebooted the NT Loader menu should appear. Now, it contains an entry to boot Solaris. When Solaris is selected its own boot manager appears.

V.2 Single hard disk, Solaris, Linux, Windows NT

Overview

There is only one hard disk drive on the system, IDE or SCSI. Three operating systems are installed: Solaris, Linux and Windows NT. We use FAT volume on the extended partition enables communication between the three operating systems. The default boot manager is Solaris one.

Partitioning

The hard disk will be partitioned as follows (proportions are meaningless):

1. Primary - NTFS - Windows NT
2. Primary - Linux ext2fs - Linux
3. Primary - Solaris UFS - Solaris - *active*
4. Extended
    4.1. FAT-32
    4.2. Linux swap
    4.3. Linux ext2fs
    4.4. NTFS
    4.5. NTFS

Recipe

1. Connect hardware and configure BIOS (see IV.1 for details).
2. Install Windows NT. Create one primary partition at the beginning of the hard disk and format it with NTFS (see notes on Planning partitions).
3. Install Linux. Remember that Linux needs a swap file system, which must be placed in a logical volume as we need one more primary partition for Solaris while there are only four partitions per a hard disk allowed. There is yet another advantage in putting Linux swap into extended partition. Since Solaris and Linux swap partitions share the same 0x82 ID, we avoid problems as Solaris will not look into the extended partition.

   With Linux `fdisk` create one primary partition of type ext2fs - 0x83 for at least `/` and `/boot` file systems, one primary partition for future Solaris installation - temporarily it may be FAT-32, and an extended partition. The first logical volume on the extended partition should be FAT-16 or FAT-32 to make it accessible by Solaris. After FAT volume create one logical volume for swap of type 0x82 and as many logical volumes of type ext2fs - 0x83 for other file systems, as you need. For desktop Linux you will probably put `/`, `/boot`, `/var`, `/tmp` and `/usr` file systems into the `fdisk` primary partition, swap and `/home` file systems into separate logical volumes. Before, read notes on the size and placement of Linux primary partition, and [Koehntopp], [Veselosky] for organizing Linux file systems.

   Install Lilo into `/dev/sda2` - Linux primary partition. At this point, Linux partition is marked active and no other system can be booted.
4. To avoid problems with Solaris disk geometry, use Ranish PM to change temporary FAT-32 primary partition to Solaris partition, i.e. type 0x82. You can also use your favorite disk editor to accomplish that task.
5. Install Solaris. The installer should detect created Solaris partition and do not create useless x86 Boot partition. Solaris partition becomes the active one. Solaris boot manager becomes the default. It lets you boot Solaris, Linux and NT.
6. Create FAT-16 volume on the extended partition e.g. with Linux `fdisk` or Ranish PM.
7. Under Linux, create `/share` directory, and add the following line to `/etc/fstab` file:

```
/dev/sda5       /share  vfat    user,rw,exec    0       0
```

This will make FAT-16 partition automatically mounted at the boot time.

8. Use NT Disk Administrator to create separate NTFS logical volumes for applications and user data. You may assign drive letters so that D: and E: will be NTFS volumes, and FAT-16 volume F: or e.g. S: (share).
9. Under Solaris, create `/share` directory, and add the following line to `/etc/vfstab` file under Solaris:

```
/dev/dsk/c0t0d0p4:c     -       /share  pcfs    -       yes     -
```

FAT-16 partition will be mounted automatically at the boot time (see notes on mounting FAT partitions).

LILO as the default boot manager

1. Boot Linux.
2. Edit `/etc/lilo.conf` as in the example:

```
boot=/dev/sda
map=/boot/map
install=/boot/boot.b
compact
prompt
timeout=50
image=/boot/vmlinuz-2.0.39
        label=linux
        root=/dev/sda2
        read-only
other=/dev/sda1
        label=nt
other=/dev/sda3
        label=solaris
```

3. Run as root `/sbin/lilo` to apply changes (see notes on LILO configuration).

Remarks:

- If the hard disk is IDE use `hda` instead of `sda` and drop `t0`.
- The same scenario can be used to install any MS Windows system not only NT. If that is the case NTFS should be replaced by FAT-32.

V.3 Single hard disk, Solaris, Linux, Windows NT, Windows 98

Overview

There is only one hard disk drive on the system. Four operating systems are installed: Solaris, Linux, Windows NT and Windows 98. FAT volume on the extended partition enables communication between all operating systems. The default boot manager is Solaris one. Windows 98 is selected to boot in the NT Loader menu.

Partitioning

The hard disk will be partitioned as follows (proportions are meaningless):

1. Primary - FAT-16 - Windows 98
2. Primary - Solaris UFS - Solaris - *active*
3. Primary - Linux ext2fs - Linux
4. Extended
   - 4.1. FAT-32
   - 4.2. Linux swap
   - 4.3. Linux ext2fs
   - 4.4. NTFS- Windows NT
   - 4.5. NTFS
   - 4.6. NTFS

Recipe

1. Connect hardware and configure BIOS (see IV.1 for details).
2. Install Windows 98. Create one primary FAT-16 partition for Windows 98 system only (see notes on the size). This could be FAT-32 in case of W2K.
3. Create Solaris primary partition with Ranish PM.
4. Install Solaris. Solaris boot manager is now the default one. It lets boot Solaris and Windows.
5. Install Linux. Remember that the type of Linux swap volume, is 0x82, the same as already existing Solaris partition. With Linux `fdisk` create one primary partition of type ext2fs 0x83 for `/`, `/boot` and other system file systems, an extended partition, one FAT logical volume at the beginning of the extended partition, one logical volume for swap of type 0x82 and one logical volume of type ext2fs 0x83 for `/home` file system (read notes on Linux installation in V.2).

   Install Lilo into `/dev/sda3` - Linux primary partition. At this point, Linux partition is marked active and only Linux may be booted.
6. Under Linux, with its `fdisk` utility, mark Solaris partition as active. This will make Solaris boot manager the default one, which allows to boot other systems.
7. Install Windows NT. NT setup program detects FAT-16 partition and Windows 98 installation. Windows 98 can be selected to boot in the NT Loader menu. Create a NTFS logical volume on the extended partition to install NT system into.
8. With NT Disk Administrator create NTFS volumes D:, E: for user data and applications.
9. Use NT Disk Administrator to set Solaris partition to active.
10. If necessary create one or more FAT-32 volumes for Windows 98. Keep in mind that only the first FAT volume within the extended partition can be mounted under Solaris.
11. Under Linux, create `/share` directory, and add the following line to `/etc/fstab` file:

    ```
    /dev/sda5       /share  vfat    user,rw,exec    0       0
    ```

    This will make (4.1) FAT-32 partition automatically mounted at the boot time.
12. Under Solaris, create `/share` directory, and add the following line to `/etc/vfstab` file:

    ```
    /dev/dsk/c0t0d0p4:c     -       /share  pcfs    -       yes     -
    ```

    (4.1) FAT-32 partition will be mounted automatically at the boot time (see notes on mounting FAT partitions).

Remarks: If the hard disk is IDE use `hda` instead of `sda` and drop `t0`.

LILO as the default boot manager

1. Boot Linux.
2. Edit `/etc/lilo.conf`, as in the example:

```
boot=/dev/sda
map=/boot/map
install=/boot/boot.b
compact
prompt
timeout=50
image=/boot/vmlinuz-2.0.39
        label=linux
        root=/dev/sda3
        read-only
other=/dev/sda1
        label=nt
other=/dev/sda2
        label=solaris
```

3. Run as root `/sbin/lilo` to apply changes (see notes on LILO configuration).

Next time you will boot the system LILO should show up at the start of boot up. Windows 98 can be loaded by selecting NT in LILO prompt and then Windows 98 in the NT Loader menu.

V.4 Two hard disks, Solaris, Windows NT - safety

Overview

There are two hard disk drives on the system, and two operating systems installed: Solaris and Windows NT. FAT-16 partition, is used to exchange data between systems. No third party boot manager is involved, Solaris boot manager boots both systems.

Solaris and NT are installed safely, that is, on separate hard disks. This allows to boot them directly by switching *BIOS boot sequence*.

Partitioning

Hard disks will be partitioned as follows (proportions are meaningless):

HDD-1	**HDD-2**
1. Primary - FAT-16 -	1. Primary - NTFS - Windows NT - *active*
2. Primary - Solaris UFS - Solaris - *active*	2. Extended
	2.1. NTFS
	2.2. NTFS

Recipe

1. Connect hardware and configure BIOS. Set HDD-2 as the boot drive (see IV.1 for details).
2. Install Windows NT. During setup, create a primary NTFS partition on HDD-2 for NT system. This will allow boot HDD-2 directly in case of problems (see notes on the size). After that you should be able to boot NT. The primary partition is C:.
3. Boot NT and use its Disk Administrator to create an extended partition and NTFS logical volumes on HDD-2, for user data, applications, archives etc. Assign drive letters D:, E:, etc. for all created

volumes and CD-ROM(s).

4. Use NT Disk Administrator to create a primary FAT-16 partition on HDD-1. This can be FAT-32 in case of W2K or latter NT version. Assign a drive letter to this volume. The partition will be used to exchange data between operating systems and to boot Windows NT. Size of FAT-16 should not exceed 1 GB to get reasonable cluster size.
5. From root of C: copy files
   1. `ntldr`,
   2. `ntdetect.com`,
   3. `boot.ini`,
   4. `ntbootdd.sys` if HDD-1 is SCSI.

   to the FAT partition on HDD-1.
6. Edit the copy of `boot.ini` file in FAT partition and change all rdisk(0) entries to rdisk(1). In case of mixed IDE/SCSI environments read notes on booting NT.
7. Reboot the machine to BIOS setup and change BIOS boot sequence to boot from HDD-1. Do not swap hard disk cables (see notes on hardware and BIOS for details).
8. The FAT partition on HDD-1 is not boot-able. To fix this follow the steps of Fixing NT boot sector section. Now, you should be able to boot NT from FAT partition on HDD-1.
9. On the remainder of HDD-1 create Solaris partition.
10. Install Solaris. Select HDD-1 for installation. This partition becomes active and Solaris boot manager will appear after reboot. To boot NT select 1, i.e. FAT partition.
11. Under Solaris, create `/share` directory, and add the following line:

    ```
    /dev/dsk/c0t0d0p1:c     -       /share  pcfs    -       yes     -
    ```

    to `/etc/vfstab` file, in order to have FAT partition mounted automatically at the boot time (see notes on mounting FAT partitions). If HDD-1 is an IDE drive drop `t0`. If HDD-1 is SCSI and HDD-2 IDE, then the address of HDD-1 is `c1t0d0`.

Remarks: The above procedure seems complicated. It could be simplified by setting HDD-1 as the boot drive at the very beginning, creating FAT-16, and then installing NT. This order, however, causes that FAT on HDD-1 becomes the drive C: and there could be problems with booting NT directly from HDD-2. The advantage of this simplified method is that DOS or Win9x can be installed on HDD-2, in contrary to the original procedure.

NT Loader as the default boot manager

1. Reboot to BIOS setup and change boot sequence to boot from HDD-2.
2. Boot NT.
3. Use Bootpart program to add Solaris entry to the NT Loader menu.

V.5 Two hard disks, Solaris, Windows NT - performance

Overview

There are two hard disk drives on the system running Solaris and Windows NT. FAT-16 partition, is used to exchange data between two systems. Solaris boot manager boots both systems.

Solaris and NT are installed considering performance over safety, that is, every system occupies the part of each hard disk. This way both hard disks work at the time for Solaris or NT.

Partitioning

Hard disks will be partitioned as follows (proportions are meaningless):

HDD-1	**HDD-2**
1. Primary - NTFS - Windows NT	1. Primary - Solaris UFS -
2. Primary - Solaris UFS - Solaris - *active*	2. Extended
	2.1. FAT-16
	2.2. NTFS
	2.3. NTFS

Recipe

1. Connect hardware and configure BIOS. Set HDD-1 as the boot drive (see IV.1 for details).
2. Install Windows NT. During setup, create a primary NTFS partition on HDD-1 for NT system (see notes on the size). After that you should be able to boot NT. The primary partition becomes C:.
3. Create Solaris partition.
4. Install Solaris. Select HDD-1 and HDD-2 for installation. Create primary Solaris partitions: one on the remainder of HDD-1 and one on HDD-2. Slice those partition according to your needs. You may put swap file system on HDD-1 and `/usr` on HDD-2. This can improve performance, but is unsafe since Solaris will not boot with HDD-2 disconnected.

   The Solaris partition on HDD-1 becomes active and Solaris boot manager will appear after reboot. To boot NT select 1, i.e. NTFS partition, in the manager.
5. Boot NT and use its Disk Administrator to create an extended partition on HDD-2. Create FAT-16 logical volume and assign it some later drive letter e.g. S: (like share). Its size should not exceed 1 GB to get reasonable cluster size. Create NTFS logical volumes, for user data, applications, archives etc. Assign drive letters D:, E: etc. for all created volumes and CD-ROM(s).
6. Under Solaris, create `/share` directory, and add the following line:

   ```
   /dev/dsk/c0t1d0p2:c     -       /share  pcfs    -       yes     -
   ```

   to `/etc/vfstab` file to mount FAT-16 volume automatically at the boot time (see notes on mounting FAT partitions). If HDD-2 is an IDE drive use `c0d1p2:c`, if it is slave on the primary channel, or `c1d0p2:c` if it is master on the secondary channel.

Remarks: Linux can be added easily as a third operating system. It is enough to leave some space on HDD-1 during Solaris setup, and on extended partition on HDD-2. Then install Linux according to instructions in section 2.

V.6 Two hard disks, Solaris, Linux, Windows NT, Windows 98

Overview

As usually, there are many possible ways to install the system which satisfies criteria given in the title of this section. Exceptionally, we present two of them instead of one.

There are two hard disk drives on the machine and the following operating systems are installed: Solaris, Linux, Windows NT, Windows 98. In the first - *Variant A* Solaris boot manager is used to boot all systems, in the second - *Variant B*, Linux Loader LILO.

Partitioning - Variant A

Hard disks will be partitioned as follows (proportions are meaningless):

HDD-1	HDD-2
1. Primary - FAT-32 - Windows 98	1. Primary - Solaris UFS -
2. Primary - NTFS - Windows NT	2. Extended
3. Primary - Solaris UFS - Solaris - *active*	2.1. FAT-32
4. Primary - Linux ext2fs - Linux	2.2. Linux swap
	2.3. Linux ex2fs
	2.4. NTFS
	2.5. NTFS

Recipe - variant A

1. Connect hardware and configure BIOS, so that HDD-1 is the boot drive (see IV.1 for details).
2. Install Windows 98. Create one FAT-32 primary partition at the beginning of HDD-1. It should not exceed 1GB. It is enough to keep Windows 9x system files. Applications and user data may be stored on FAT-32 logical volumes within an extended partition (see notes on the size).
3. Install Windows NT. NT 4.0 will not recognize FAT-32 and Windows 98 installation contrary to W2K. In both cases create new primary partition and construct NTFS file system. It need not to be large since applications and user data can go to logical volumes on HDD-2.
4. Create one Solaris primary partition on HDD-1 and one on HDD-2. Leave some room for Linux and Windows.
5. Install Solaris.
6. Install Linux creating the fourth primary partition. This partition should start before 1024-th cylinder of the disk to make Linux possible to boot by the BIOS. This partition will be addressed by `/dev/sda4`. Take care not to destroy Solaris partition as Linux swap partition has the same ID - 0x82. Create an extended partition on HDD-2. Within that partition make FAT volume for data exchange, Linux swap partition and additional partition for `/home` file system. Linux swap will be `/dev/sdb6` and home `/dev/sdb7`. See notes on Linux installation in V.2.

   Install LILO into `/dev/sda4`.
7. Using Linux `fdisk` make Solaris partition active to instruct BIOS to load Solaris boot manager at the system boot up.
8. With NT Disk Administrator create NTFS volumes on HDD-2 for applications and user data. Assign drive letters according to your likes.
9. With Windows 98 `fdisk` and `format` utilities, or W2K Disk Administrator create FAT-32 volume on HDD-2 if necessary. Keep in mind there could be problems with mounting these FAT volumes under Solaris.
10. Under Solaris, create `/share` directory, and add the following line:

    ```
    /dev/dsk/c0t1d0p2:c     -       /share  pcfs    -       yes     -
    ```

    to `/etc/vfstab` file to mount FAT volume at boot up (see notes on mounting FAT partitions).
11. Under Linux, create `/share` directory, and add the following line:

    ```
    /dev/sdb5       /share  vfat    user,rw,exec    0       0
    ```

    to `/etc/fstab` file.

LILO as the default boot manager - variant A

1. Boot Linux.
2. Edit `/etc/lilo.conf` as in the example:

   ```
   boot=/dev/sda
   map=/boot/map
   install=/boot/boot.b
   compact
   prompt
   timeout=50
   image=/boot/vmlinuz-2.0.39
           label=linux
           root=/dev/sda3
           read-only
   other=/dev/sda1
           label=win98
   other=/dev/sda2
           label=nt
   other=/dev/sda3
           label=solaris
   ```

3. Run as root `/sbin/lilo` to apply changes (see notes on LILO configuration).

Partitioning - Variant B

HDD-1	HDD-2
1. Primary - FAT-32 - Windows 98	1. Primary - Solaris UFS - Solaris - *active*
2. Primary - NTFS - Windows NT - *active*	2. Primary - Linux ext2fs - Linux
3. Extended	3. Extended
3.1. FAT-32	3.1. Linux swap
3.2. NTFS	3.2. Linux ex2fs
3.3. NTFS	

Recipe - variant B

1. Connect hardware and configure BIOS so that HDD-1 is the boot drive (see IV.1 for details).
2. Install Windows 98 (see variant A step 2).
3. Install Windows NT (see variant A step 3).
4. With NT Disk Administrator create an extended partition on HDD-1. Within that partition create FAT-32 volume for data exchange, and NTFS volumes for applications and user data. Assign drive letters according to your likes.
5. Change BIOS boot sequence to boot from HDD-2 (do not swap cables).
6. Create Solaris primary partition on HDD-2 leaving some room for Linux.
7. Install Solaris. After selecting the disk for Solaris software installation, which in our case is HDD-2, you will be given an option to either: let Solaris to configure the boot device, or select it manually. Actually, both methods lead to the same result - the boot path for Solaris is not setup properly. It can be setup through the Boot Tasks in the DCA later.
8. Install Linux (see variant A step 5). Install LILO into `/dev/sdb` i.e. into MBR of HDD-2.
9. Configure LILO as follows (see notes on LILO configuration):

```
boot=/dev/sdb
map=/boot/map
install=/boot/boot.b
compact
prompt
timeout=50
image=/boot/vmlinuz-2.0.39
        label=linux
        root=/dev/sdb2
        read-only
other=/dev/sda1
        label=win98
other=/dev/sda2
        label=nt
other=/dev/sdb1
        label=solaris
```

10. Under Linux, create `/share` directory, and add the following line:

```
/dev/sda5       /share  vfat    user,rw,exec    0       0
```

   to `/etc/fstab` file.

11. Under Solaris, create `/share` directory, and add the following line:

```
/dev/dsk/c0t0d0p3:c     -       /share  pcfs    -       yes     -
```

   to `/etc/vfstab` file to mount FAT volume at boot up (see notes on mounting FAT partitions).

NT Loader as the default boot manager - variant B

1. Reboot to BIOS setup and change boot sequence to boot from HDD-1.
2. Boot NT.
3. Use Bootpart program to add Windows 98, Solaris and Linux entries to the NT Loader menu.

Remarks: For IDE drives use `hda` instead of `sda` and drop `t0`, `t1`. If HDD-1 is SCSI and HDD-2 IDE, then HDD-1 is addressed by `c1t0d0` and HDD-2 by `c0d0`.

V.7 Two hard disks, Solaris and Windows

Overview

This scenario is similar to the Variant B in the previous section. It is intended for those who have MS Windows or another operating system setup already, upgraded their system by adding a new hard disk drive and are going to install Solaris on that new disk.

So, there are two hard disk drives in the system. We will refer to them as to HDD-1 and HDD-2. HDD-1 is the current boot drive and it is totally occupied by some operating system. HDD-2 is clean.1

Partitioning

Hard disk HDD-1 will not be modified except it MBR. Hard disk HDD-2 will be partitioned as follows:

HDD-2

1. Primary - Solaris UFS - Solaris - *active*
2. Extended
    2.1. FAT-32

Recipe

1. HDD-2 can be SCSI or IDE. We assume that HDD-1 is hooked to its original controller and no cables were swapped. Some hard disks use different jumper settings for single versus master. That is the only required modification in case of IDE drives connected to the same IDE channel. BIOS is setup to boot HDD-1 and we will not change that.
2. Create Solaris primary partition on HDD-2. Leave some room for FAT volume used to exchange data between two operating systems, or let Solaris occupy the entire disk if you are going to mount some existing FAT volume on HDD-1.
3. Install Solaris. After selecting the disk for Solaris software installation, which in our case is HDD-2, you will be given an option to either: let Solaris to configure the boot device, or select it manually. Actually, both methods lead to the same result - the boot path for Solaris is not setup properly. It causes that every time Solaris is booted DCA is launched and manual selection of the Solaris boot disk is necessary. The boot path can be setup through the Boot Tasks in the DCA later, with `eeprom` command or simply by editing the file:

   ```
   /boot/solaris/bootenv.rc
   ```

   Recommended and the easiest is to use DCA.
4. Install Ranish PM compact boot manager into the MBR of HDD-1 and HDD-2. This will enable booting from both hard disk drives without the need to change BIOS settings or use a DCA floppy.
5. Under Solaris, create `/share` directory, and add the following line:

   ```
   /dev/dsk/c0t1d0p2:c     -       /share  pcfs    -       yes     -
   ```

   to `/etc/vfstab` file to mount FAT volume at boot up (see notes on mounting FAT partitions).

Remarks: If both HDD-1 and HDD-2 are IDE, then in most cases HDD-1 will be `c0d0` while HDD-2 will be `c0d1` if it is slave on the same channel as HDD-1, or `c1d0` if it is a master on the other IDE channel.

VI. Back matter

VI.1 Index

VI.2 Glossary

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

- A -

ATA

Advanced Technology Attachment - a disk drive interface standard based on the IBM PC ISA 16-bit bus but also used on other personal computers. The ATA specification deals with the power and data signal interfaces between the motherboard and the integrated disk controller and drive. The ATA "bus" only supports two devices - master and slave.

ATA-2

Advanced Technology Attachment Interface with Extensions - a standard which extends the Advanced Technology Attachment interface while maintaining compatibility with current IBM PC BIOS designs. ATA-2 provides for faster data rates, 32-bit transactions and (in some drives) DMA. Optional support for power saving modes and removable devices is also in the standard.

- B -

BIOS boot sequence

An entry in most moder BIOSes. Specifies where the BIOS should look first for an operating system. Enables to select devices in various order. Commonly available options are: ’A, C’, ’C only’, ’C, A’, ’D, A’, ’CD-ROM, C, A’, ’SCSI, C ,A’ or ’LAN, C, A’.

- C -

cylinder

The set of tracks on a multi-headed disk that may be accessed without head movement. That is, where multiple discs are mounted on the same axle, the collection of disk tracks which are at the same distance from the spindle about which the disks rotate. Each such group forms the shape of a cylinder. Placing data that are likely to be accessed together in cylinders reduces the access significantly as head movement (seeking) is slow compared to disk rotation and switching between heads.

- E -

EIDE

Extended Integrated Drive Electronics - ATA-2.

extended partition

A *partition* of a hard disk drive divided into smaller logical sections - *logical volumes*. Developed to break the limit of 4 partitions per a hard disk drive, related to small size of *MBR*.

- F -

FAT, FAT-16, FAT-32

File Allocation Table. File systems based on the idea of File Allocation Table. Older FAT-16 are used by all MS operating systems, newer FAT-32 is used since MS Windows 95 OSR2 was released. FAT-32 is recognized by Windows 98 and W2K, while NT 4.0 needs additional software to use FAT-32.

file system

1. A method of arrangement of directories and files, connected with an *operating system* which implements it.

2. A collection of directories and files.

- H -

head
An electronic device that reads/writes data from/to a disk. It is an interface between the magnetic physical media on which the data is stored and the electronic components that make up the rest of the hard disk. A head converts bits to magnetic pulses and stores them on a platter, and then reverses the process when the data is read back.

HPFS
File system used in OS/2.

- I -

IA
Intel Architecture - IBM PC compatible architecture or machine.

IDE
Integrated Drive Electronics - ATA.

- L -

LBA
Logical Block Addressing - A translation method used by BIOS of IBM PC compatibles to change the size limit of a hard disk drive from around 528MB to 8.4GB. The number of cylinders of a hard disk drive is decreased while the number of heads is proportionally increased, so that both parameters stay below their limits. The limits are 1024 for cylinders, 256 for heads and 64 for sector per track.

logical volume
A section of an extended partition that contains a file system.

- M -

MBR
Master Boot Record - The first sector of a hard disk drive. It stores information about partitions on that disk, the partition id, activity flag, starting cylinder/head/sector, and ending cylinder/head/sector. Four partitions are allowed.

- N -

NT Disk Signature
An entry in the *MBR* where Windows NT and W2K stores information on volume letter assignment. It occupies 4 bytes starting at 0x1B8 byte.

NTFS
Windows NT family, including Windows 2000, file system. It provides file permisions, long filenames (case is ignored).

- O -

operating system
(OS) A low-level software interface between hardware and applications. Sometimes comes with GUI - Graphic User Interface, and additional software.

- P -

partition
On IA machines, logical section of a hard disk drive. On UNIX systems sometimes called *slice*. Generally, two types of partitions are distinguished: *primary* and *extended*.

primary partition
Usually boot-able *partition* of a hard disk drive that contains an *operating system*.

- S -

SCSI
Small Computer System Interface - A processor-independent standard for system-level interfacing between a computer and intelligent devices including hard disks, floppy disks, CD-ROM, printers, scanners and many more.

sector
1. Physically, a continuous range of rotational angle of a disk, a segment of a track. Usually, physical sectors are staggered in order to give enough time to deal with one sector before the next is accessible.

2. Logically, a smallest portion of information that can be read/write at the time from/to disk, typically 512 bytes. On Macintosh and UNIX drives, sectors usually are grouped into blocks that function as the smallest data unit permitted. The terms block and sector are sometimes used interchangeably.

slice
On x86 machines, a logical section of a *partition*, common for Unix operating systems. Sometimes identified with partition on these systems. Usually contains a file system, e.g. /, /var, /usr or /export.

- T -

track
A concentric ring on a disk surface where data is stored.

- U -

UFS
Unix File System.

- V -

VBS
Volume Boot Sector (Partition Boot Sector aka PBR) - The first sector of a partition. If the partition is boot-able, it stores an executable program to load an operating system.

VTOC
Volume Table Of Contents - A table containing information on slices within a Solaris partition. The definition of a slice consists of 5 fields: slice number, tag, flag, starting sector and size. There is room for 16 slices per partition. VTOC occupies the second and third sector of the Solaris partition.

volume
A *primary* partition or a *logical volume*.

VI.3 References

Literature

[Howe]
Howe, Denis, *Free On-Line Dictionary of Computing* (http://foldoc.doc.ic.ac.uk/foldoc/index.html), 1999.

[Koehntopp]
Koehntopp, Kristan and Harris, Tony, *Linux Partition HOWTO* (http://www.linuxdoc.org/HOWTO/mini/Partition/index.html), June 12, 2000.

[Kozierok]
Kozierok, Charles M., *PC Guide* (http://www.pcguide.com/), April 25, 1999.
[Locke]
Locke, Jason C., *The NTLDR Hacking Guide* (http://www.dorsai.org/~dcl/publications/NTLDR_Hacking), September 15, 1996.
[MS1]
Microsoft, *Article ID: Q102873* (http://support.microsoft.com/support/kb/articles/q102/8/73.asp), January 14, 1999.
[MS2]
Microsoft, *Article ID: Q114841* (http://support.microsoft.com/support/kb/articles/q114/8/41.asp), October 25, 2000.
[Quantum]
Quantum Corporation, *Quantum Fireball TM Product Manual*, 1996.
[Richmond]
Richmond, Jay, *Installing and Using FreeBSD With Other Operating Systems* (http://www.freebsd.org/doc/en_US.ISO8859-1/articles/multi-os/index.html), August 6, 1996.
[Russinovich]
Russinovich, Mark, *BOOT.INI Option Reference* (http://www.sysinternals.com/ntw2k/info/bootini.shtml), March 12, 1999.
[Skoric]
Skoric, Miroslav, Lilo mini-Howto (http://www.linuxdoc.org/HOWTO/mini/LILO.html), 24 July, 2000.
[Sun]
Sun Microsystems, Inc., *Reference Manual Collection*, docs.sun.com.
[Veselosky]
Veselosky, Vince, *Control-Escape: Alternative Software* (http://www.control-escape.com/), 1999.

Software

- ATAID,
- Boot,
- BootPart, (Download),
- GRUB,
- OS-BS and other FreeBSD Tools, (Download),
- Ranish Partition Manager,
- System Commander.

Last revised: Dec 15, 2003.