NTFS file system

By Dmitrey Mikhailov

The Microsoft operating systems of the Windows NT set cannot be imagined without NTFS file system - one of most complex and successful of existing at present file systems. The given article will tell you what features and disadvantages this system has, on what principles based the organisation of the information and how to keep the system in the stable condition, what possibilities NTFS offers and how they can be used by the common user.

Part 1. NTFS physical structure

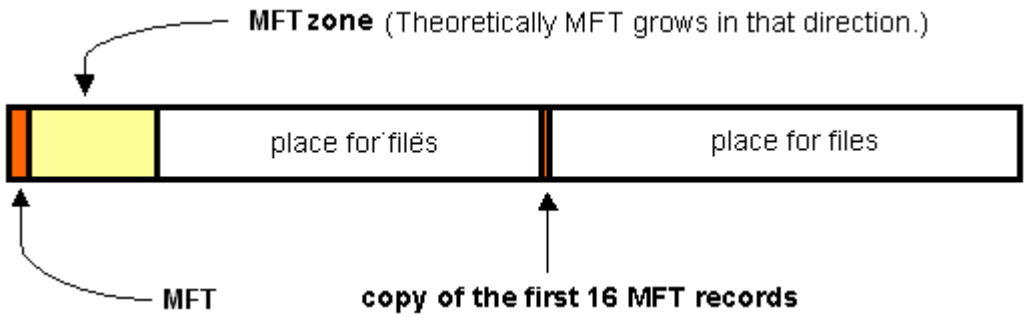
Let's begin from the common facts. The NTFS partition theoretically can be almost of any size. The limit certainly exists but I shall not point at it as it will be more than enough for the next hundreds of years of computer technology development at any growth rates. What about practice? Almost the same way. The maximum size of the partition NTFS at the moment is limited only by the hard disks sizes. NT4 probably will have some problems at the attempt of installation on the partition if any of its parts steps back more than on 8 GBytes from the disk physical beginning but this problem concerns only the load partition.

The way of NT4.0 installation on the empty disk is rather original and can lead to incorrect thoughts about NTFS possibilities. If you point the installation program that you wish to format disk in NTFS, maximum size which it will offer you will be only 4 GBytes. Why it is so little if NTFS partition size actually is unlimited? The answer is that installation section simply does not know this file system. :) The installation program formats this disk in usual FAT which maximum size in NT is 4 GByte (with usage of not absolutely standard huge cluster 64 KByte) NT is installed on this FAT. And during the first operating system load (in the installation phase) the fast partition conversion to NTFS is effected so that user notice nothing except the strange "limiting" on the NTFS size at the installation time. :)

Overview of the partition structure.

As well as any other system NTFS divides all useful place into clusters - data blocks used at a time. NTFS supports almost all sizes of clusters - from 512 bytes up to 64 KBytes. The 4 KBytes cluster is considered to be some standard. NTFS doesn't have any anomalies of cluster structure and I have nothing to say about it.

NTFS disk is symbolically divided into two parts. The first 12% of the disk are assigned to so-called MFT area - the space which MFT metafile grows into. Any data recording into this area is impossible. The MFT-area is always kept empty not to let the most important service file (MFT) be fragmented at growth. The rest 88% of the disks represent usual space for files storage.



Disk free space however includes all physically free space - free pieces of MFT-area are included there too. The mechanism of MFT-area usage is like this: when files already cannot be recorded in usual space, MFT-area is simply reduced (in operating systems current versions -twice) clearing the space for recording files. At clearing the usual area, MFT can be extended again. Thus it is possible for usual files to remain in this area and it is normal. The system tried to keep it free but failed. Life is going on... The metafile MFT all the same can be fragmented though it would be undesirable.

MFT and its structure

NTFS file system is a distinguished achievement of structuring: **each** system component is a file - even system information. The most important file on NTFS is named MFT or Master File Table - the common table of files. It is situated in MFT area and is the centralised directory of all remaining disk files and itself. MFT is divided into records of the fixed size (usually 1 KBytes), and each record corresponds to some file. The first 16 files are housekeeping and they are inaccessible to the operating system. They are named metafiles and the very first metafile is MTF itself. These first 16 elements MFT are the only part of the disk having the fixed position. It is interesting that the second copy of the first 3 records, for reliability (they are very important) is stored exactly in the middle of the disk. The remaining MFT-file can be stored as well as any other file at any places of the disk. It is possible to re-establish its position with its own help using the basis - the first MFT element.

Metafiles

The first 16 NTFS files (metafiles) are system files. Each of them is responsible for some aspect of system operation. The advantage of such modular approach is in amazing flexibility - for example on FAT the physical failure in the FAT area is fatal for all disk operation. As for NTFS it can displace and even fragment on the disk all system areas avoiding any damage of the surface except the first 16 MFT elements.

The metafiles are in the NTFS disk root directory, they start with a name character "$", though it is difficult to get any information about them by standard means. Curiously that even for these files the quite real size is reported, and it is possible to find out for example how many operating system spends on cataloguing of all your disk having looked at $MFT file size. In the following table the metafiles used at the moment and their function are indicated.

$MFT	Itself MFT

$MFTmirr	copy of the first 16 MFT records placed in the middle of the disk
$LogFile	journaling support file (see below)
$Volume	housekeeping information - volume label, file system version, etc.
$AttrDef	list of standard files attributes on the volume
$.	root directory
$Bitmap	volume free space bitmap
$Boot	boot sector (bootable partition)
$Quota	file where the users rights on disk space usage are recorded (began to work only in NT5)
$Upcase	File - the table of accordance between capital and small letters in files names on current volume. It is necessary because in NTFS file names are stored in Unicode that makes 65 thousand various characters and it is not easy to search for their large and small equivalents.

Files and streams

So the system has files and nothing except files. What does this concept on NTFS include?

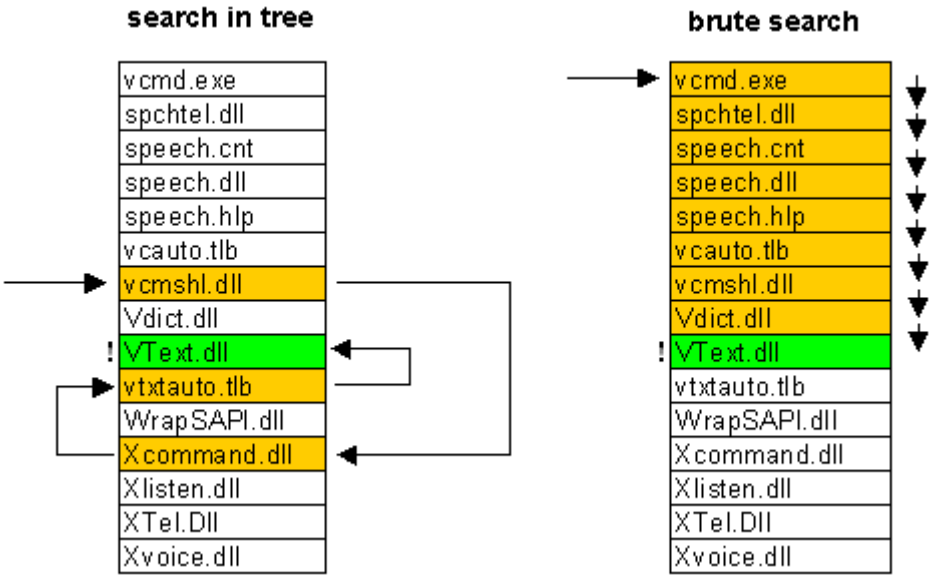
- First of all the compulsory element is the record in MFT. As it was said above all disk files are mentioned in MFT. All information about a file except data itself is stored in this place: a file name, its size, separate fragments position on the disk, etc. If one MFT record is not enough for information, then several records are used and not obligatory one after another.
- Optional element is file data streams. The definition "optional" seems to be a bit strange but nevertheless there is nothing strange here. Firstly a file may not have data and in this case disk free space isn't used on it. Secondly a file may have not very big size. Then a rather successful decision is applied: file data are stored just in MFT, in the place free from the master data in limits of one MFT record. The files with the size of hundreds byte usually don't have "physical" image in the fundamental file area. All such file data are stored in one place - in MFT.

There is an interesting case with file data. Each file on NTFS has a rather abstract constitution - it has no data, it has streams. One of the streams has the habitual for us sense - file data. But the majority of file attributes are also streams! Thus we have that the base file nature is only the number in MFT and the rest is optional. The given abstraction can be used for the creation of rather convenient things - for example it is possible to "stick" one more stream to a file, having recorded any data in it - for example information about the author and the file content as it was made in Windows 2000 (the most right bookmark in file properties which is accessible from the explorer). It is interesting that these additional streams are not visible by standard means: the observed file size is only the size of the main stream contains the traditional data. It is possible for example to have a file with a zero length and at its deleting 1 GByte of space is freed just because some program or technology has sticked an

additional stream (alternative data) of gigabyte size on it. But actually at the moment the streams are practically not used, so we might not be afraid of such situations though they are hypothetically possible. Just keep in mind that the file on NTFS is much deeper and more global concept than it is possible to imagine just observing the disk directories. Well and at last: the file name can consist of any characters including the full set of national alphabets as the data is represented in Unicode - 16-bit representation which gives 65535 different characters. The maximum file name length is 255 characters.

The directories

The directory on NTFS is a specific file storing the references to other files and directories establishing the hierarchical constitution of disk data. The directory file is divided into blocks, each of them contains a file name, base attributes and reference to the element MFT which already gives the complete information on an element of the directory. The inner structure of the directory is a binary tree. It means that to search the file with the given name in the linear directory such for example as for FAT, the operating system should look through all elements of the directory until it finds the necessary one. The binary tree allocates the names of files to make the file search faster - with the help of obtaining binary answers to the questions about the file position. The binary tree is capable to give the answer to the question in what group the required name is situated - above or below the given element. We begin with such question to the average element, and each answer narrows the area of search twice. The files are sorted according to the alphabet, and the answer to the question is carried out by the obvious way - matching of initial letters. The search area which has been narrowed twice starts to be researched the same way starting again from the average element.



It is necessary to realise that to search one file among 1000 files for example FAT should do about 500 matchings (most probably the file will be found in the middle of the search) and the system on the basis of a tree - at all about 10 (2^10 = 102. Saving of search time is in fact. Don't think that in traditional systems (FAT) everything is so uncared-for: firstly the maintenance of the binary tree files list is rather complex and secondly - even FAT in fulfilment of the modern system (Windows2000 or Windows98) uses similar search optimisation. This is just one more fact to be added to your knowledge. It would be desirable also to clear up the widespread mistake (which I absolutely shared recently) that to add a file

in the directory as a tree is more difficult than in the linear directory. These operations are comparable enough on time. To add a file in the directory it is necessary at first to be sure that the file with such name is not present yet there and then we shall have some problems in the linear system with the search of a file described above. These problems compensate with interest the ease of file addition in the directory.

What information can be got just having read a directory file? This is what is given by the command dir. To effect the elementary navigating on the disk it is not necessary to go in MFT for each file, it is only necessary to read the most common information about files from directories files. The main disk directory - root - differs from the usual directories by nothing except the special reference to it from the metafile MFT beginning.

Journalising

NTFS is a fail-safe system which can correct itself at practically any real failure. Any modern file system is based on such concept as **transaction** - the action made wholly and correct or not made at all. NTFS just doesn't have intermediate (erratic or incorrect) conditions - the data variation quantum cannot be divided on before failure or after it bringing breakups and muddle - it is either accomplished or cancelled.

Example 1. The data record on the disk is being carried out. Suddenly it is clarified that it is impossible to record in the place where we have just decided to record the next chunk of data because of the physical surface damage. The NTFS behaviour is rather logical in this case: the record transaction is rolled away wholly - the system realises that the record is not effected. The place is marked as failure and the data are recorded in another place and the new transaction starts.

Example 2. The case is more complex - the data record on the disk is being carried out. Suddenly the power is turned out and the system reboots. On what phase has the record stopped and where is the data? The transaction log comes to help. The system having realised the desire to write on the disk has flagged in the metafile $LogFile this condition. At reboot this file is studied to find out the uncompleted transactions which were interrupted by the crash and which result is unpredictable. All these transactions are cancelled: the place where the record was carried out is marked again as free, MFT indexes and elements are resulted in the condition they were before failure, and the system remains stable in whole. And what about the situation if the error has taken place at record in the journal? It is not terrible: the transaction either has not started (there is only an attempt to record the intention to make it) or was already completed - that is there is an attempt to record that the transaction is already fulfilled. In the last case at the following load the system itself will quite clear up that actually everything is recorded correctly and will not pay any attention to the "unfinished" transaction.

And nevertheless remember that journalising is not the absolute panacea but only a mean to reduce the number of errors and system failures. An ordinary NTFS user will hardly ever note the error of the system or will have to launch chkdsk. Experience shows that NTFS is restored in the completely correct condition even at failures in the moments very much loaded by disk activity. You can even defragment the disk and in the peak of this process push reset - the probability of data losses even in this case will be very low. It is important to realise however that the system of NTFS restoration guarantees the correctness of the whole **file system** only, not your data. If you effected disk writing and have got a crash - often the correct data cannot be restored. The miracles do not happen.

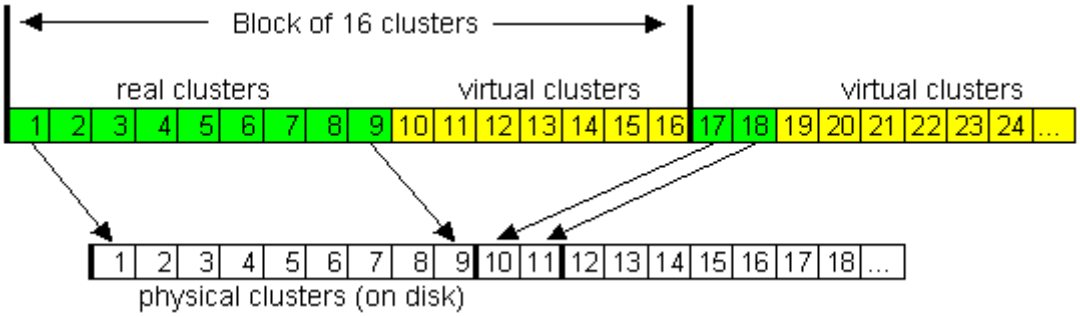
Compression

Files on the NTFS volume have one rather useful attribute - "compressed". NTFS has built - in support of disk compression. Earlier Stacker or DoubleSpace was used for this purpose. Any file or directory in the individual order can be stored on the disk in the compressed form and this process is completely clear for applications. The file compression has very much high speed and only one large negative property - huge virtual fragmentation of compressed files which however does not bother anybody. The compression is carried out by blocks of 16 clusters and uses so-called " virtual clusters ". This decision is extremely flexible and permit to achieve interesting effects - for example a half of file can be compressed and a half is not. It is achieved because the information storage about compression rate of the defined fragments is very similar to usual file fragmentation: for example the typical record of physical layout for real, not compressed file:

The file clusters from 1 to 43-rd are stored in the disk clusters from 400-th
The file clusters from 44 to 52-nd are stored in the disk clusters from 8530-th...

Physical layout of a typical compressed file:

The file clusters from 1 to 9-th are stored in the disk clusters from 400-th
The file clusters from 10 to 16 are not stored anywhere
The file clusters from 17 to 18 are stored in the disk clusters from 409-th
The file clusters from 19 to 36-th are not stored anywhere



It is visible that the compressed file has "virtual" clusters which don't have the real information. As soon as the system sees such virtual clusters, it realises at once that the data of the previous block multiple to16 should be decompressed and the data just will fill in virtual clusters and this is all algorithm.

Security

NTFS contains a lot of means for differentiation of the objects rights, it is supposed to be the most perfect file system from all nowadays existing. In theory it is undoubtedly so, but in current implementations unfortunately the rights system is far enough from the ideal and is a hard but not always logical set of the characteristics. The rights assigned to any object and unambiguously by the system itself. The large variations and additions of the rights were carried out already several times and at the creation of Windows 2000 they came to the rational enough set.

NTFS file system rights are close connected with the system itself, and that means they are not obligatory to be kept by another system if it is given physical access to the disk. For

preventing physical access in Windows2000 (NT5) the standard possibility was taken (about this see below). The rights system in its current condition is rather complex and I doubt that I can tell something interesting and useful to the readers. If you are interested in this topic, you can find a lot of books on the NT network architecture where it is described more than in detail.

The description of file system constitution can be completed, it is necessary to describe only some just practical or original things.

Hard Links

This thing is in NTFS for a rather long time but it was used very seldom - and nevertheless: Hard Link is when the same file has two names (some directory entries are pointing to the same MFT record). Let us admit that the same file has the names 1.txt and 2.txt: if a user deletes file 1, file 2 will remain. If he deletes file 2, file 1 will remain. That means both names are completely equal from the moment of creation. A file is physically deleted only when its last name is deleted.

Symbolic Links (NT5)

There is much more practical possibility permitting to make the virtual directories, very much as virtual disks - by the command subst in DOS. The applications are wide enough: first it is the simplification of the directories system. If you do not like the directory Documents and settings\Administrator\Documents, you can link it in the root directory and the system will go on communicating with the directory by the former way but you - with much shorter name completely equivalent to it. For creation of such links it is possible to use the program junction (junction.zip (15 KBytes), 36 KBytes) which was written by the known technician Mark Russinovich (http://www.sysinternals.com/). The program works only in NT5 (Windows 2000) as well as possibility itself.

Attention! Please keep in mind that these symbolic links can be deleted correctly by rm command. Be very carefully - deleting the link with explorer or any other file manager which do not know such concepts as symbolic linking will delete the information the link is pointing to!

Encryption (NT5)

There is a useful possibility for people who are troubled about their secrets - each file or directory can also be encrypted and thus cannot be read by another NT installation. In combination with standard and very much safe password on the system itself this possibility provides the safety of selected by you important data for the majority of applications.

Part 2. Features of NTFS defragmentatoin

Let's return to one interesting enough and important moment - NTFS fragmentation and defragmentation. The situation with these two concepts at the moment can not be called satisfactory in any way. At the very beginning it was said that NTFS is not subject to file fragmentation. It is not exactly so and the statement was changed - NTFS prevents fragmentation. It is not exactly so either. That is it certainly prevents but... It is already clear

that NTFS is a system which is predisposed to fragmentation inspite of official statements. But it doesn't suffer from it. All internal structures are constructed in such way that fragmentation does not hinder to find data fragments fast. But it doesn't save from the physical effect of fragmentation - waste disk heads motions.

To the source of the problem...

As it is known the system fragments files the best way when the free space is being ended, when it is necessary to use small-sized unused space remained from other files. The first NTFS property which directly promotes serious fragmentation appears here.

NTFS disk is divided into two areas. In beginning of the disk there is MFT area - the area where MFT grows (Master File Table). The area occupies minimum 12% of the disk, and the data record in this area is impossible. It is made not to let MFT be fragmented. But when all remaining disk is being filled in - the area is reduced twice:). And so on. Thus we have not single pass of the disk ending, but several. In result if NTFS works at the disk filled on about 90% - fragmentation grows greatly.

The incidental result - the disk filled more than on 88% is almost impossible to be defragmented - even defragmentation API cannot transfer the data in MFT area. It is possible that we will not have free space for a manoeuvre.

NTFS works and works and is fragmented - even in the case of free space is far from exhausting. This is promoted by the strange algorithm of finding free space for file storage - second serious omission. The action algorithm at any record is like this: some definite disk range is taken and filled in with a file. It is done by the very interesting algorithm: at first large unused space is filled in and then small one. I.e. the typical allocation of file fragments according to the size on fragmented NTFS looks so (sizes of fragments):

16 - 16 - 16 - 16 - 16 - [back] - 15 - 15 - 15 - [back] - 14 - 14 - 14.... 1 - 1 - 1 -1 - 1...

So the process goes up to most small-sized unused space in 1 cluster, in spite of the fact that on the disk there are also much larger pieces of free space.

Recall compressed files - at active overwriting of the large volumes compressed information on NTFS the huge quantity of "holes" are because of reallocation ñompressed cluster groups on the disk. If any file area began to be compressed better or worse, it is necessary either to take it from a continuous chain and to place in another place or to strap in size reserving unused space.

It is impossible to say that NTFS prevents file fragmentation. On the contrary it fragments them with pleasure. NTFS fragmentation can surprise any person familiar with file system operation in half a year of work. Therefore it is necessary to launch defragmentation. But here all our problems are not ended, they only start...

Means of the decision?

In NT there is standard API defragmentation. It has the interesting limit for files block relocating: it is possible to transfer not less than 16 clusters (!) at once, and these clusters

should start from the position16 clusters aligned in a file. In common, the operation is carried out extremely for 16 clusters. The results:

1. It is impossible to transfer anything in the unused space less than 16 clusters (except compressed files but it is not interesting at the moment).
2. A file being transferred in another place leaves (in the new place) "the temporarily occupied space" adding it on the size up to multiplicity to 16 clusters.
3. At attempt to transfer a file the wrong way ("not multiply to 16 ") the result is often unpredictable. Something is just not transferred, but the whole arena is gracefully littered wit "the temporary occupied space". These "the temporarily occupied space" serves for simplification of system restoration in the case of a machine failure and is usually freed half a minute later.

Nevertheless it would be logical to use this API if it is in stock. And it is used. Therefore the standard defragmentation process with the corrections on limitation API consists of the following phases (not necessarily in this order):

- Files extraction from MFT area. Not specially - just because it is impossible to put them back. This phase is harmless and even somehow useful.
- Files defragmentation. Unconditionally it is a useful process but a bit complicated by limitings of relocatings multiplicity - we have to move files more that it would be otherwise necessary.
- MFT, pagefile.sys and directories defragmentation. It is possible through API only in Windows2000, in the opposite case - at reboot, as a separate process like in old Diskeeper.
- The addition of files is closer to the beginning - so-called free space defragmentation. It is a terrible process...

If we want to put files on end in the beginning of the disk, we shall put one file. It leaves unused space from its end up to the block (16 clusters) boundary for alignment. Then we put the following, and after that we have the unused space of size less than 16 clusters. Then it is impossible to fill in it through API defragmentation! In result before optimisation the overview of free space looked like this: there were a lot of unused space of the same size. After optimisation - one "hole" at the end of the disk, and a lot of small < 16 clusters ones in the section filled by files. What places are first to be filled in? mall-sized "holes" < 16 clusters taking place closer to the beginning of the disk... Any file slowly built on optimised disk will consist of great number of fragments. Then the disk can be optimised again. And then once again...

Thus there are two about equivalent options. The first one is to optimise the disk often by such defragmentator paying no attention to such great fragmentation of the newly created files. The second variant is not to change anything and put up with regular but much weaker fragmentation of all disk files.

While there is only one defragmentator which ignores API defragmentation and works more directly - Norton Speeddisk 5.0 for NT. When it is compared to all remaining - Diskeeper,

O&O defrag, etc. - the main difference is not mentioned. It is just because this problem is carefully hidden. Speeddisk is the unique for today program which can optimise the disk completely not establishing small fragments of free space.

Unfortunately the defragmentator working through API which makes unused space <16 clusters was placed in Windows 2000.

All remaining defragmentators are just harmful at one-time application. If you launched it even one time, you would need to launch it then at least once a month to be saved from new files fragmentation. This is the problem of NTFS defragmentation by old means.

Part 3. What to select?

Any of the represented nowadays file systems is rather old and NTFS is a very old system! PC for a long time used only operating system DOS and FAT is owed it by its appearance. But some systems aimed to the future were developed and existed then. Two such systems obtained the wide recognition - NTFS created for the operating system Windows NT 3.1 and HPFS - a true friend of OS/2.

The implantation of new systems was difficult. In 1995 at appearance Windows95 nobody thought that something needed to be changed. FAT32 appeared in Windows 95 OSR2 didn't change the essence of the system which just does not give the possibility to organise effective operation with a plenty of data but widen the boarders.

HPFS (High Performance File System) actively used till nowadays by OS/2 users has shown itself as enough successful system, but also it had essential disadvantages - complete absence of automatic restorability means, excessive complexity of data structure and the low-level of flexibility.

NTFS could not win personal computers for a long time because of the fact that for organisation of effective operation with the data structures it demanded significant memory sizes. The systems with 4 or 8 MByte (standard of 1995-1996) were just unable to get though any plus from NTFS. Therefore it had incorrect reputation of a slow and bulky system. Actually it does not fit the reality - modern computer systems with memory more than 64 Mbytes get just huge increase of productivity from NTFS usage.

In the given table all essential pluses and minuses of the widespread presently systems such as FAT32, FAT and NTFS are shown together. It is hardly reasonable to discuss other systems, as now 97% of the users make choice between Windows98, Windows NT4.0 and Windows 2000 (NT5.0), and other variants are just not present.

	FAT	FAT32 (vFAT)	NTFS
Systems	DOS, Windows9x, all NTs	Windows98, NT5	NT4, NT5
Maximal volume size	2 GBytes	near by unlimited	near by unlimited
Maximal files	around 65000	near by unlimited	near by unlimited

count			
File name	Long names (up to 255 chars), system character set.	Long names (up to 255 chars), system character set.	Long names (up to 255 chars), unicode character set.
File attributes	Basic set	Basic set	All that programmers want
Security	no	no	yes (capability of physical encryption, starting from NT5.0)
Compression	no	no	yes
Fault tolerance	middle	low	fully (automatic)
Economy	minimal	improved (small clusters)	maximal
Performance	high for small files count	as for FAT, but also additional penalty for big volumes	very effective for all volumes

It would be desirable to tell that if your operating system is NT (Windows 2000), then to use any file system which differs from NTFS means to limit the convenience and flexibility of operating system operation. NT and especially Windows 2000 with NTFS are two parts of a unit. A lot of useful NT possibilities are directly connected with physical and logical structure of the file system, and you should use FAT or FAT32 there only for compatibility - if you have the task to read these disks from any other systems.