Writing and Deleting files on hard drives with NTFS

F. DARNOWSKI, A. CHOJNACKI

darnowski.fryderyk@gmail.com, andrzej.chojnacki@wat.edu.pl

Military University of Technology, Faculty of Cybernetics
Institute of Computer and Information Systems
W. Urbanowicza Str. 2, 00-908 Warsaw, Poland

The goal of this article was to present detailed information about writing and deleting process on the NTFS (New Technology File System) formatted drives. The most important are the algorithms used by computer to write data BFA (Best Fit Algorithm) and FFA (First-Free Algorithm) to update $MFT (Master File Table). The naming convention of the areas of the drive is presented. The proposed rules of writing and deleting algorithm were successfully validated with real NTFS volume.

Keywords: hard disk, NTFS, $MFT.

DOI: 10.5604/01.3001.0013.1457

1. Introduction

The algorithms used in Digital Forensics (DF) to search and carve files have been widely researched in the past two decades. The standard method of searching for file headers and footers was improved by realizing that most files have structured architecture, or the algorithm used to fill the data into files is known. This led to the file content-awareness search (Garfinkel 2007 [6], Wei et al. 2010 [19]). Some content aware algorithms were based on a statistical approach (Veenman 2007 [20]) paired with neural networks (Amirani et al. 2008 [1]). To counter the growing disk volume size, random sampling of disk clusters was proposed (Garfinkel 2010 [7]). This method proved valuable in areas where the information that a disk has "some" files of interest is more important than actually finding "all" possible files (for example: a quick pre-search of many drives to prioritize them for future work). Somehow, the opposite approach is to hash every file, and even hash parts of files, and then hash all disk areas to search for those file parts (Garfinkel et al. 2010 [8], Young et al. 2012 [21]). That method allowed the recovery of partially overwritten files. As simple hashing could only find identical content, in order to find similarities between files, fuzzy-hash algorithms were proposed (Roussev et al. 2010 [16], Roussev et al. 2013 [17]). Because the size of the hashes of known files became a storage problem for on-site analysis, the use of a Bloom filters hashing algorithm combined with cluster sampling was proposed (Penrose et al. 2015 [15]). Today, even parallel GPU computing, that had enormous success in other fields of DF, is finally being introduced in carving and data recovery (Škrbina, Stojanovski 2012 [18], Bayne et al. 2019 [2]).

All of the algorithms presented above have one thing in common. All of them rely heavily on analyzing the data. The structure of any individual file can be known, but the location of the file is considered to be purely random. This is a somehow strange approach, as the algorithm that is used to write files on an NTFS Volume was described in some detail by Carrier in 2005 [3] and by Microsoft itself [9].

2. New Technology File System NTFS in detail

The NTFS was created in order to replace the old FAT (File Allocation Table) file system. FAT has not entirely disappeared today, as it retains a strong position in the field of memory cards (especially mobile phones and digital cameras). In the hard drives of our home computers, NTFS dominates the market, as 85-90% of all personal computers have Windows installed, depending on which survey you look at [13].

The basic concept of NTFS is "everything is a file". Instead of strict boundaries between system area and data area, as in FAT, NTFS

stores its metadata or system data in system files. System files are hidden from the user and their names start with "$". The $MFT file is the most important of them. The Master File Table ($MFT) stores the information of every file that exists on the hard drive. It even stores information about itself. $MFT and the hard drive itself is populated with data according to a special algorithm.

For reasons of simplicity we can assume that the $MFT file is a table with records of 1024 B in size. Every file that exists on a hard drive has a corresponding $MFT record. Every record has its own unique index. A simple record consists of information about the file name, creation, modification or access times, status of the file (deleted or not) and many other attributes. It is worth mentioning that when a user deletes a file, nothing is changed except two bytes in the corresponding $MFT record. The file data and all other record data is intact. Another interesting attribute is the *file identifier* or as we describe it – the delete count. Every record has this variable, its value is increased by one every time a file attached to the record is deleted.

We can assume that this variable describes how many times this record was used in the past. By "use" we mean writing a completely new file. When a new file is created two things can happen. One: if every $MFT record has status set as active (not deleted) a new record is created at the end of the table. Two: if a record with status set as deleted exists, then the record with smallest index is erased and populated with the new metadata of the new file. Small files are stored directly in the $MFT record (resident files).With files larger than 500–700 B, the $MFT record stores only pointers to the area of the disk where the actual file data resides (non-resident files).

An example $MFT is presented in Table 1. The names of the files and their parameters are purely random. We assumed that the hard drive is 25 clusters in size. According to this particular $MFT there are five files on the hard drive (A, B, C, D, F) with two files being deleted (B, F).

The distribution of files from Table 1 is presented in Figure 1. Different areas on the hard drive are as follows: $\mathbf{O}_U$ – unallocated area, an area not addressed by any of the $MFT record; $\mathbf{O}_A$ – allocated area, an area occupied by undeleted files; $\mathbf{O}_D$ – deleted area; that is area occupied by deleted files, whose records still exist in $MFT.

Tab. 1. Example of data stored in $MFT:
a – file name, b – the starting points of the file fragments, c – the sizes of the file fragments, d – status (0 – active, 1 – deleted), e – delete count

$MFT					
id	**a**	**b**	**c**	**d**	**e**
1	A	0	3	0	0
2	B	4	4	1	3
3	C	9	4	0	0
4	D	17	1	0	0
5	F	18	7	1	1

There could be a situation where two or more files address the same area (one file overwrites another). The creation of such a situation is presented in chapters 7 and 8.

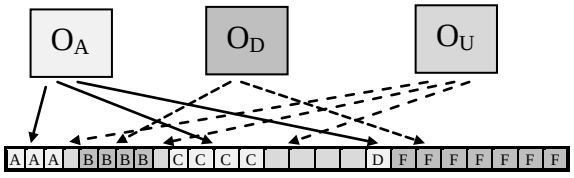


Fig. 1. Distribution of files from Table 1

In this article we will use the distribution table shown in Table 2 below. Every cluster can belong only to one area. Clusters belonging to an allocated area are stored in row one, clusters belonging to a deleted area are stored in row two and clusters belonging to an unallocated area are stored in row three.

Tab. 2. Distribution table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A							C	C	C	C					D							
O_D					B	B	B	B											F	F	F	F	F	F	F
O_U																									

3. Best Fit Algorithm

We assume that all files are big enough that their data is non-resident, which means they are stored outside the $MFT file. The data of every files is written using the Best Fit Algorithm (BFA) [3]. In theory, NTFS is always trying to put a file in the "best" free area. The "best" area can be described as the smallest unallocated area that is equal or bigger in size than the file that is going to be written. If there is more than one of these areas, the one closest to the beginning of the disk is chosen. In a case where all unallocated areas are smaller than the file size, fragmentation occurs. The biggest unallocated area is filled with data (if there are two or more unallocated areas with equal size the one closest to the beginning of drive is chosen).

The remaining part of the file is written in the "best" remaining area or fragmented again. This simple algorithm was successfully verified in our research. Some more details need to be explained before we can continue.

After writing the data, actualization of the $MFT file follows. If there are no deleted files, the new record will be created for the new file just after the last record (there cannot be "empty" records between active records). If deleted files exist, than the record with the smallest index is overwritten with new content for the new file. All previous content of a record is discarded. We will describe the writing algorithm in $MFT as the "First Free Algorithm" (FFA), where free means there is either an empty record or deleted file. In the following sections, we assumed the corresponding rules:

a) writing and deleting is sequential;
b) size of file is known at the time of writing;
c) there are only non-resident files with minimum size of 1 cluster;
d) data is written first;
e) data is written using BFA;
f) data is written into $\mathbf{O}_U$ even if fragmentation occurs;
g) if file size is bigger than $\mathbf{O}_U$ then the file is written into the area created by merging the $\mathbf{O}_U$ and $\mathbf{O}_D$ areas;
h) if file size is bigger than the merged area then writing does not occur;
i) when all data is written, the $MFT is updated with the new record using FFA;
j) deleting a file increases the delete count variable by one and changes the status flag from 0 to 1.

The scant literature on the subject provides information that, during writes BFA and FFA algorithms are used. The question of what data area is selected for an algorithm to work on, to the best of the authors' knowledge, hasn't really been explored yet. For example, if a new file overwrites the $MFT entry of the existing file X, is the data belonging to file X taken into consideration by the writing algorithm? In other words, is the new file written into the $\mathbf{O}_U$ or $\mathbf{O}_U$+$\mathbf{O}_D$ area? In the following chapters we will describe all the nuances of the proposed rules of writing.

4. Process of writing in examples

According to rule a) there can only be writing or deleting operations. These operations are sequential – one operation must end before starting another. In chapters 5–8 we describe a few variants of writing a file with the name N. The status of the $MFT is taken from Table 1 so the initial distribution of files is equal to Table 2. The process of writing is explained using three distribution tables: first – before writing starts, second – after writing data, and third – after updating the $MFT. The $MFT is explained using two tables: before and after update. According to rule i) a new file will always occupy the record with index 2.

5. File size smaller or equal to at least one unallocated area

We are writing file N of size 2.

The new file, according to rules e) and f), will be stored in the third unallocated area (Table 3).

Tab. 3. Distribution of files before writing data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$\mathbf{O}_A$	A	A	A							C	C	C	C					D							
$\mathbf{O}_D$					B	B	B	B											F	F	F	F	F	F	F
$\mathbf{O}_U$				1					2					3	3	3	3								

The data of the file is written from left to right and occupies clusters 13 and 14 (Table 4).

Tab. 4. Distribution of files after writing data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$\mathbf{O}_A$	A	A	A							C	C	C	C	N	N			D							
$\mathbf{O}_D$					B	B	B	B											F	F	F	F	F	F	F
$\mathbf{O}_U$				1					2							3	3								

After successfully writing the data, the NTFS then updates $MFT (Table 5).

Tab. 5. Changes in $MFT when writing file N of size 2 before update

$MFT					
id	**a**	**b**	**c**	**d**	**e**
1	A	0	3	0	0
2	B	4	4	1	3
3	C	9	4	0	0
4	D	17	1	0	0
5	F	18	7	1	1

Using the FFA record, 2 will be overwritten with information on file N, therefore we lose the information about file B (Table 6). At this point the process of writing is complete. Even the data of file B is still intact, as file B has merely stopped being recognized by the file system (the system has lost awareness of its existence). File B is now part of an unallocated area (Table 7).

Tab. 6. Changes in $MFT when writing file N of size 2 after update

$MFT					
id	**a**	**b**	**c**	**d**	**e**
1	A	0	3	0	0
2	N	13	2	0	3
3	C	9	4	0	0
4	D	17	1	0	0
5	F	18	7	1	1

Tab. 7. Distribution of files after updating $MFT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A							C	C	C	C	N	N			D							
O_D																			F	F	F	F	F	F	F
O_U				1	B	B	B	B	2							3	3								

6. File size is bigger than the biggest unallocated area, but smaller than O_U

We are writing file N of size 5.

The NTFS, according to rule f), is trying to write the new file in $\mathbf{O}_U$. As the file is bigger than every unallocated sub-area, fragmentation starts. The biggest unallocated area is clusters 13-16, as shown in Table 8. This area will be filled with the first part of the file.

Tab. 8. Distribution of files before writing data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A							C	C	C	C					D							
O_D					B	B	B	B											F	F	F	F	F	F	F
O_U				1					2					3	3	3	3								

The remaining part of the file (one cluster) will be written using BFA on the remaining unallocated area. After writing the first part of the file, we have two unallocated areas with size 1 (clusters 3 and 8). The system picks the one closest to the beginning of the hard disk, therefore the second fragment of the file is written in cluster 3 (Table 9).

Tab. 9. Distribution of files after writing data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A	N2						C	C	C	C	N1	N1	N1	N1	D							
O_D					B	B	B	B											F	F	F	F	F	F	F
O_U									2																

After successfully writing the data, the NTFS then updates the current $MFT (Table 5). Using FFA, record 2 will be overwritten with the information on file N, therefore we lose the information about file B (Table 10).

At this point the process of writing is complete. Even the data from file B is still intact, file B is no longer recognized by file system (the system has lost awareness of its existence). File B is now part of an unallocated area (Table 11).

Tab. 10. Changes in $MFT when writing file N of size 5 after update

$MFT					
id	**a**	**b**	**c**	**d**	**e**
1	A	0	3	0	0
2	N	13,3	4,1	0	3
3	C	9	4	0	0
4	D	17	1	0	0
5	F	18	7	1	1

Tab. 11. Distribution of files after updating $MFT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A	N2						C	C	C	C	N1	N1	N1	N1	D							
O_D																			F	F	F	F	F	F	F
O_U					B	B	B	B	2																

7. Files size is bigger than O_U

We are writing file N of size 7.

As there is not enough room for the file in the unallocated area, according to rule g), we merge $\mathbf{O}_U$ and $\mathbf{O}_D$. For future analysis we assume that unallocated area is increased by the deleted area (Table 12).

Tab. 12. Distribution of files before writing data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A							C	C	C	C					D							
O_D																									
O_U				1	B	B	B	B	2					3	3	3	3		F	F	F	F	F	F	F

The file system still knows where files B and F reside, but acts like this area was part of $\mathbf{O}_U$. Increasing $\mathbf{O}_U$ creates three unallocated areas with sizes counting from left to right: 6, 4 and 7 clusters. The new file is placed in the third unallocated area (clusters 18-24) as shown in Table 13.

Tab. 13. Distribution of files after writing data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A							C	C	C	C					D	N	N	N	N	N	N	N
O_D					B	B	B	B																	
O_U				1					2					3	3	3	3								

At this point an update of the $MFT occurs similar to the previous examples (Table 14) resulting in file B being lost by the file system (Table 15). In this particular example, file B lost its $MFT record with data intact, while file F lost all of its data but the $MFT record was intact. The file F is overwritten by file N. We can still recover the metadata of file F (name and time-stamps for example) but the actual data of file F is lost.

Tab. 14. Changes in $MFT when writing file N of size 7 after update

$MFT					
id	**a**	**b**	**c**	**d**	**e**
1	A	0	3	0	0
2	N	18	7	0	3
3	C	9	4	0	0
4	D	17	1	0	0
5	F	18	7	1	1

Tab. 15. Distribution of files after updating $MFT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A							C	C	C	C					D	N	N	N	N	N	N	N
O_B																									
O_U				1	B	B	B	B	2					3	3	3	3								

8. File size is bigger than the biggest area from merged O_U and O_D

We are writing file N of size 15

The merging of areas is similar to the example in chapter 7, creating three unallocated areas: clusters 3–8, clusters 13–16, and clusters 18–24 (Table 12). As the file data cannot fit into a single unallocated area fragmentation occurs. The first fragment is put into the third unallocated area (clusters 18–24), the remaining part of file is still bigger than the biggest remaining unallocated area. Another fragmentation starts. The second part of the file occupies the first unallocated area (clusters 3–8). The remaining part is written in the second unallocated area (clusters 13–16) as shown in Table 16.

Tab. 16. Distribution of files after writing data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A	N2	N2	N2	N2	N2	N2	C	C	C	C	N3	N3			D	N1	N1	N1	N1	N1	N1	N1
O_B																									
O_U																3	3								

At this point, an update of the $MFT occurs similarly to the previous examples (Table 17). As the data from file B has already been overwritten, the update of the $MFT does not change the O_U (Table 18).

Tab. 17. Changes in $MFT when writing file N of size 7 after update

$MFT					
id	**a**	**b**	**c**	**d**	**e**
1	A	0	3	0	0
2	N	18, 3, 13	7, 6, 2	0	3
3	C	9	4	0	0
4	D	17	1	0	0
5	F	18	7	1	1

In this particular example, file B loses its $MFT record and data, while file F loses all of its data, but the $MFT record remains intact. File N, as in chapter 7, overwrites file F.

Tab. 18. Distribution of files after updating $MFT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
O_A	A	A	A	N2	N2	N2	N2	N2	N2	C	C	C	C	N3	N3			D	N1	N1	N1	N1	N1	N1	N1
O_B																									
O_U																3	3								

9. Verification based on real NTFS partition observation

In order to verify the proposed model, we created an NTFS partition on a 4 GB thumb drive.

The cluster size was set to default, which is 4096B or 8 sectors. Writing on flash memory is a very sophisticated process that is very different to writing on a hard drive [4]. These processes are done internally by the microcontroller of the thumb drive and are not visible to the Operating System. Therefore, from the NTFS perspective, writing on a thumb drive acts similarly to writing to a hard disk. The changes on the thumb drive were analyzed using WinHex. After formatting a volume to NTFS only the system files were present on the partition, with their location presented in Table 19. The $secure file is not on the partition after formatting, but is created before the first write process. Therefore, it always resides in cluster 35.

The three unallocated areas before $MFT have the sizes: 8, 248691 and 2509 clusters. For the tests, the following files were generated:

- 2.clust – size 2 clusters,
- 4.clust – size 4 clusters,
- 8.clust – size 8 clusters,
- File-548KB.pdf – size 137 clusters,
- File-980KB.pptx – size 246 clusters,
- File-8,25MB.odp – size 2112 clusters.

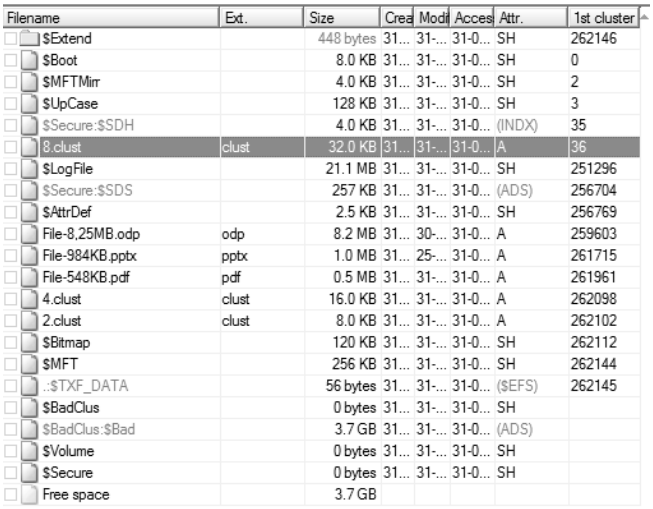
All of the above files together occupy 2509 clusters and fit ideally into the third unallocated area. All tests were done after fresh formatting of the volume.

Tab. 19. Files that are on the partition just after formatting

id	Name	1st cluster	Last cluster	size (clusters)	notes
1	$BOOT	0	1	2	
2	$MFTMirr	2	2	1	
3	$UpCase	3	34	32	
4	($secure)	35		1	Created before first write
5	#1 unallocated	36	43	8	
6	Reserved	44		1	
7	$TxLog2	45	2604	2560	
8	#2 unallocated	2605	251295	248691	approx. 970 MB
9	$LogFile	251296	256703	5408	
10	$Secure	256704	256768	628	
11	$AttrDef	256769	256769	1	
12	$Tops	256770	257025	256	
13	$TxfLog.blf	257026	257041	16	
14	$TxfLog1	257042	259601	2560	
15	$TxfLog	259602		1	Folder
16	#3 unallocated	259603	262111	2509	approx. 10 MB
17	$Bitmap	262112	262141	3584	
18	Reserved	262142	262143	2	
19	$MFT	262144	262207	64	
20	Reserved	262208	313375	51168	approx. 5% area reserved for $MFT
21	unallocated	313376			Ok. 2,6 GB

Test 1: *all of the above files were written from biggest to smallest.*

The situation after test 1 is presented in Figure 2.



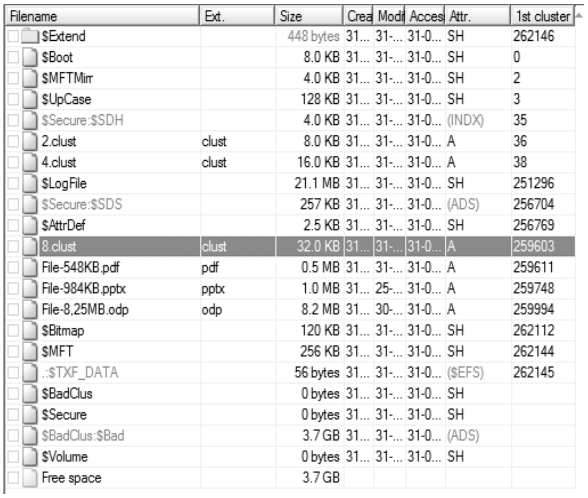
Filename	Ext.	Size	Crea	Modi	Acces	Attr.	1st cluster
$Extend		448 bytes	31...	31-...	31-0...	SH	262146
$Boot		8.0 KB	31...	31-...	31-0...	SH	0
$MFTMirr		4.0 KB	31...	31-...	31-0...	SH	2
$UpCase		128 KB	31...	31-...	31-0...	SH	3
$Secure:$SDH		4.0 KB	31...	31-...	31-0...	(INDX)	35
8.clust	clust	32.0 KB	31...	31-...	31-0...	A	36
$LogFile		21.1 MB	31...	31-...	31-0...	SH	251296
$Secure:$SDS		257 KB	31...	31-...	31-0...	(ADS)	256704
$AttrDef		2.5 KB	31...	31-...	31-0...	SH	256769
File-8,25MB.odp	odp	8.2 MB	31...	30-...	31-0...	A	259603
File-984KB.pptx	pptx	1.0 MB	31...	25-...	31-0...	A	261715
File-548KB.pdf	pdf	0.5 MB	31...	31-...	31-0...	A	261961
4.clust	clust	16.0 KB	31...	31-...	31-0...	A	262098
2.clust	clust	8.0 KB	31...	31-...	31-0...	A	262102
$Bitmap		120 KB	31...	31-...	31-0...	SH	262112
$MFT		256 KB	31...	31-...	31-0...	SH	262144
.:$TXF_DATA		56 bytes	31...	31-...	31-0...	($EFS)	262145
$BadClus		0 bytes	31...	31-...	31-0...	SH	
$BadClus:$Bad		3.7 GB	31...	31-...	31-0...	(ADS)	
$Volume		0 bytes	31...	31-...	31-0...	SH	
$Secure		0 bytes	31...	31-...	31-0...	SH	
Free space		3.7 GB					

Fig. 2. Files located on volume after **test 1**

File-8,25MB.odp was, in line with the BFA rule, written into the third unallocated area starting from cluster 259603. The next two files, *File-980KB.pptx* and *File-548KB.pdf* were written just after. The third unallocated area shrank to 14 clusters after this operation. The file *8.clust* was written into the first unallocated area starting from cluster 36 and occupied it completely. The last two files *4.clust* and *2.clust* were written into the remaining third unallocated area.

Test 2: *all of the above files were written from smallest to biggest.*

The files *2.clust* and *4.clust* were written into the first unallocated area that shrinks to 2 clusters. Because file *8.clust* was too big for the remainder of the first unallocated area, it was written into the third one. Directly after all the other files, *File-548KB.pdf, File-980KB.pptx* and *File-8,25MB.odp*, were written (Figure 3).



Filename	Ext.	Size	Crea	Modi	Acces	Attr.	1st cluster
$Extend		448 bytes	31...	31-...	31-0...	SH	262146
$Boot		8.0 KB	31...	31-...	31-0...	SH	0
$MFTMirr		4.0 KB	31...	31-...	31-0...	SH	2
$UpCase		128 KB	31...	31-...	31-0...	SH	3
$Secure:$SDH		4.0 KB	31...	31-...	31-0...	(INDX)	35
2.clust	clust	8.0 KB	31...	31-...	31-0...	A	36
4.clust	clust	16.0 KB	31...	31-...	31-0...	A	38
$LogFile		21.1 MB	31...	31-...	31-0...	SH	251296
$Secure:$SDS		257 KB	31...	31-...	31-0...	(ADS)	256704
$AttrDef		2.5 KB	31...	31-...	31-0...	SH	256769
8.clust	clust	32.0 KB	31...	31-...	31-0...	A	259603
File-548KB.pdf	pdf	0.5 MB	31...	31-...	31-0...	A	259611
File-984KB.pptx	pptx	1.0 MB	31...	25-...	31-0...	A	259748
File-8,25MB.odp	odp	8.2 MB	31...	30-...	31-0...	A	259994
$Bitmap		120 KB	31...	31-...	31-0...	SH	262112
$MFT		256 KB	31...	31-...	31-0...	SH	262144
.:$TXF_DATA		56 bytes	31...	31-...	31-0...	($EFS)	262145
$BadClus		0 bytes	31...	31-...	31-0...	SH	
$Secure		0 bytes	31...	31-...	31-0...	SH	
$BadClus:$Bad		3.7 GB	31...	31-...	31-0...	(ADS)	
$Volume		0 bytes	31...	31-...	31-0...	SH	
Free space		3.7 GB					

Fig. 3. Files located on volume after **test 2**

Test 3: *filling first and third unallocated area.*

First, we copied the file *8.clust*. The file occupied the first unallocated area completely. Next, the files *2.clust, 4.clust, 8.clust, File-548KB.pdf, File-980KB.pptx* and *File-8,25MB.odp* were copied, which fully occupied the third unallocated. After this operation, only two unallocated areas were left on the volume, with 970 MB and 2.6 GB sizes respectively. The folder, containing 103 jpg files, 240MB size in total, was copied. The files were inserted into the second unallocated area and occupied clusters 2605-64156. The distribution of data on the volume is presented in Figure 4.

Occupied Area 0-2604	JPG files 2605- -64156	#1 unallocated area 64157- -251295	Occupied Area 251296- -313375	2# unallocated area 313376-

Fig. 4. Position of files after completing test 3

Based on this test we can assume that copying files that are similar in size will fill an unallocated area as follows: the files are copied one by one, and are located next to each other. The detailed view of the position of the last files is presented in Figure 5. The files are sorted by their first cluster.

Filename	Ext.	Size	Crea	Modi	Acces	Attr.	1st cluster
DSC_6312.JPG	JPG	2.1 MB	31...	09-...	31-0...	A	54026
DSC_6318.JPG	JPG	2.1 MB	31...	09-...	31-0...	A	54564
DSC_6319.JPG	JPG	2.0 MB	31...	06-...	31-0...	A	55109
DSC_6321.JPG	JPG	2.4 MB	31...	09-...	31-0...	A	55626
DSC_6322.JPG	JPG	2.2 MB	31...	09-...	31-0...	A	56241
DSC_6323.JPG	JPG	2.3 MB	31...	06-...	31-0...	A	56803
DSC_6331.JPG	JPG	2.4 MB	31...	07-...	31-0...	A	57402
DSC_6332.JPG	JPG	2.5 MB	31...	07-...	31-0...	A	58026
DSC_6333.JPG	JPG	2.4 MB	31...	07-...	31-0...	A	58669
DSC_6334.JPG	JPG	2.8 MB	31...	09-...	31-0...	A	59284
DSC_6335.JPG	JPG	2.6 MB	31...	09-...	31-0...	A	60002
DSC_6336.JPG	JPG	2.5 MB	31...	09-...	31-0...	A	60662
DSC_6337.JPG	JPG	2.4 MB	31...	09-...	31-0...	A	61303
DSC_6338.JPG	JPG	2.1 MB	31...	09-...	31-0...	A	61907
DSC_6339.JPG	JPG	2.2 MB	31...	09-...	31-0...	A	62445
DSC_6340.JPG	JPG	2.5 MB	31...	09-...	31-0...	A	63001
DSC_6341.JPG	JPG	2.1 MB	31...	09-...	31-0...	A	63631

Fig. 5. Detailed view of the position of the last copied files from **test 3** (WinHex)

Test 4: *Deleting and writing files using the outcome from* ***test 3****.*

After successfully completing test three the following operations were executed:

- deleting of files: DSC_6338.JPG, DSC_6339.JPG, DSC_6340. JPG,
- writing files: 1.TXT, File1,
- deleting of file: 1.TXT,
- writing of file: File2.

The representation of $MFT for the files in interest is shown on Table 20 and corresponds to the situation shown in Figure 5. The location of the files in interest is presented in Table 21.

Tab. 20. Changes in $MFT during **test 4**. Starting situation

$MFT					
id	**a**	**b**	**c**	**d**	**e**
140	DSC_6336	60662	641	0	0
141	DSC_6337	61303	604	0	0
142	DSC_6338	61907	538	0	0
143	DSC_6339	62445	556	0	0
144	DSC_6340	63001	630	0	0
145	DSC_6341	63631	526	0	0

Tab. 21. Position of files during **test 4**. Starting situation

O_A	DSC_6336	DSC_6337	DSC_6338	DSC_6339	DSC_6340	DSC_6341		
O_B								
O_U								

After completing the first deletion of 4 files, nothing changes on the drive except an update in $MFT: status flags are set to “delete” and delete counters are increased (Table 22). Files *DSC_6338.JPG DSC_6339.JPG*, *DSC_6340. JPG* cease being visible to the user and are transferred to $\mathbf{O}_D$ (Table 23).

Tab. 22. Changes in $MFT during **test 4** after deleting DSC_6338, DSC_6339, DSC_6340

$MFT					
id	**a**	**b**	**c**	**d**	**e**
140	DSC_6336	60662	641	0	0
141	DSC_6337	61303	604	0	0
142	DSC_6338	61907	538	1	1
143	DSC_6339	62445	556	1	1
144	DSC_6340	63001	630	1	1
145	DSC_6341	63631	526	0	0

Tab. 23. Position of files during t**est 4** after deleting DSC_6338, DSC_6339, DSC_6340

O_A	DSC_6336	DSC_6337						DSC_6341		
O_B			DSC_6338	DSC_6339	DSC_6340					
O_U										

Writing file *1.TXT* of 180 clusters in size will take place after file *DSC_6341.JPG* (rule d)). The update of $MFT happens and, because there are no other deleted files on the volume other than the three we just deleted, the file *1.TXT* will be given record 142 in $MFT (Table 24). The overwriting of record 142 means the system loses the information about file *DSC_6338.JPG* therefore its data is shifted to the $\mathbf{O}_U$ (Table 25).

Tab. 24. Changes in $MFT during **test 4** after writing 1.TXT

$MFT					
id	**a**	**b**	**c**	**d**	**e**
140	DSC_6336	60662	641	0	0
141	DSC_6337	61303	604	0	0
142	1.TXT	64157	180	0	1
143	DSC_6339	62445	556	1	1
144	DSC_6340	63001	630	1	1
145	DSC_6341	63631	526	0	0

Tab. 25. Position of files during **test 4** after writing 1.TXT

O_A	DSC_6336	DSC_6337								DSC_6341	1.TXT
O_B						DSC_6339	DSC_6340				
O_U			DSC_6338								

In the next step *File1* is written. *File1* is small enough (453 cluster) to fit into the unallocated area created from file *DSC_6338.JPG* (538 clusters). Writing *File1* overwrites record 143 during $MFT update (Table 26). After update, file *DSC_6339.JPG* is moved to $\mathbf{O}_U$ (Table 27).

Tab. 26. Changes in $MFT during **test 4** after writing File1

$MFT					
id	**a**	**b**	**c**	**d**	**e**
140	DSC_6336	60662	641	0	0
141	DSC_6337	61303	604	0	0
142	1.TXT	64157	180	0	1
143	File1	61907	453	0	1
144	DSC_6340	63001	630	1	1
145	DSC_6341	63631	526	0	0

Tab. 27. Position of files during **test 4** after writing File1

O_A	DSC_6336				DSC_6337				File1									DSC_6341	1.TXT
O_B														DSC_6340					
O_U												DSC_6339							

Deleting file *1.TXT* only updates $MFT (Table 28) and moves data to $\mathbf{O}_D$ (Table 29).

Tab. 28. Changes in $MFT during **test 4** after deleting 1.TXT

$MFT					
id	**a**	**b**	**c**	**d**	**e**
140	DSC_6336	60662	641	0	0
141	DSC_6337	61303	604	0	0
142	1.TXT	61907	538	1	2
143	File1	61907	453	0	1
144	DSC_6340	63001	630	1	1
145	DSC_6341	63631	526	0	0

Tab. 29. Position of files during **test 4** after deleting 1.TXT

O_A	DSC_6336				DSC_6337				File1									DSC_6341	
O_B														DSC_6340					1.TXT
O_U												DSC_6339							

The size of *File2* is 544 clusters and is smaller than the unallocated area from the remaining part of *DSC_6338.JPG* and *DSC_6339.JPG*. Once again the record 142 will be overwritten in $MFT (Table 30) and therefore *file 1.TXT* is lost by the system and moved to $\mathbf{O}_U$ (Table 31).

Tab. 30. Changes in $MFT during **test 4** after writing File2

$MFT					
id	**a**	**b**	**c**	**d**	**e**
140	DSC_6336	60662	641	0	0
141	DSC_6337	61303	604	0	0
142	File2	62360	544	0	2
143	File1	61907	453	0	1
144	DSC_6340	63001	630	1	1
145	DSC_6341	63631	526	0	0

Tab. 31. Position of files during **test 4** after writing File2

O_A	DSC_6336				DSC_6337				File1				File2					DSC_6341	
O_B														DSC_6340					
O_U																			1.TXT

It is important to mention at this point that no data is moved or copied to the volume when we, say, move or transfer to $\mathbf{O}_U$ or $\mathbf{O}_D$. It only represents a change in the distribution table explained in chapters 2-8. The detailed view of the position of files for the last four operations is shown in Figure 6-9 (WinHex).

Fig. 6. Position of files during **test 4**, after writing 1.TXT

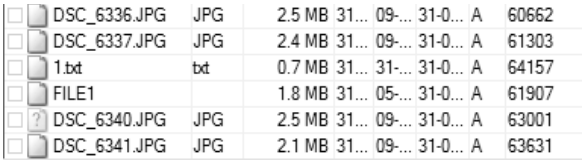


Fig. 7. Position of files during **test 4**, after writing File1

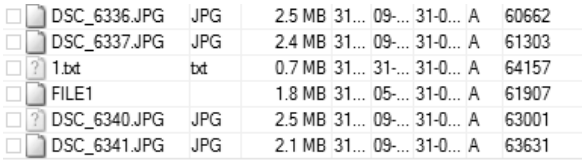


Fig. 8. Position of files during **test 4**, after deleting 1.TXT

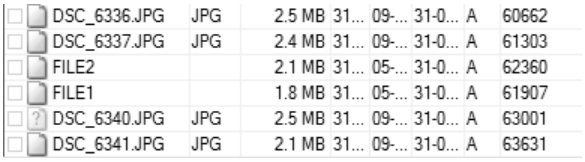


Fig. 9. Position of files during **test 4**, after writing File2

In the figures 9, the files are sorted by their $MFT index, so the view from WinHex corresponds to tables 24, 26, 28 and 30. This proofs that our concept of writes and deletes presented in chapter 3 is valid, as the changes in real NTFS Volume (data area and $MFT itself) are exactly the same as that predicted by our algorithm (tab. 20-31).

10. Summary

As described in chapter 1, all modern computer forensic software uses very sophisticated search algorithms on unallocated areas to find deleted files. All of these algorithms assume the pseudo-random position of files. With this assumption, it is not possible to narrow, with confidence, the search area. Algorithms have to search through all unallocated space.

This article proposes to thinking of the NTFS allocation algorithm as a Finite--State Machine (Gladyshev 2004 [10], Gladyshev 2005 [11]). With a Finite-State Machine (FSM) approach, we assumed that all of the operations start only when the previous ones finish. Therefore, knowing the current state of disk $\mathbf{S}_n$, we can predict the location of a new file to be written (record of $MFT file and data area), if we know the file's size. That means that, with a given file size, we know exactly what the state $\mathbf{S}_{n+1}$ will look like.

$$\mathbf{S}_n \rightarrow \text{write(x)} \rightarrow \mathbf{S}_{n+1}$$

What is more, having a sequence of known writes and deletes we can predict the location of every file in the sequence.

$$\mathbf{S}_n \rightarrow \text{write(x)} \rightarrow \mathbf{S}_{n+1} \rightarrow \text{write(y)} \rightarrow \mathbf{S}_{n+2}$$

As shown in this article, the above was proven valid. So now, the question arises: if we know the state of the disk "now", is it possible to reconstruct the possible state $\mathbf{S}_n$ x operations back?

$$\mathbf{S'_n} \leftarrow \text{write(x')} \leftarrow \mathbf{S'_{n+1}} \leftarrow \text{write(y')} \leftarrow \mathbf{S}_{n+2}$$

Having resolved possible sequences: write(x'), write(y'), delete(x') …. write(z') for the first time in digital forensics, we would know:

- if file y' exists, or has been partially or totally overwritten,
- the time of writing and deleting of the file y' (written after file x', deleted before writing file z'),
- if any recovered file belongs to this Volume or is left-over from the previous one (before formatting).

Olivier [14], states that computing complexity might be a challenge in FSM but as Gladyshev [11] demonstrated with a simple case, we think that now, with advances in other areas of Digital Forensics, this method is finally ready to be used as complement to the many other existing methods [5]. Some methods described in chapter 1 can be used with cooperation with FSM, scaling down the computer complexity:

- scanning unallocated areas: deciding which portions of a disk have content and which are empty,
- standard header-footer carving in content full area: finding files that are easily recoverable,
- statistical analysis and cluster sampling of the remaining area and grouping it by category.

The next step in our work will be to construct a mathematical model for the writing and deleting process. This model will be used to create a carving and analyzing tool which exploits the NTFS file writing algorithm.

11. Known limitations

The proposed model is true only when the system knows the size of file that is being copied. If the file size is unknown, the stream is created with some starting size while the data is copied. If that starting size is filled, then the next data part, bigger than the previous one, is created. The algorithm of allocating next data parts continues until maximum (selected by system) part size is reached. The allocation of those fragments is most likely BFA but further research is needed [9].

The FSM approach exploits the core of file systems architecture which can be traced way back to the 1970's.

In that era "a hard disk" was a big magnetic tape with one head available to do one operation at a time. This is not true of the modern NTFS as it has built-in mechanisms for performance improvement, system checks and many other system services (logging, indexing etc.). These services run in the background, disrupting the idea of one operation at a time. Thankfully, the system reserves more area for future work than it needs, so we can assume system area as static with fixed size (for at least some period of time) and remove it from consideration. This assumption is even truer for external drives

as writes, created there by operating system, are minimal.

We are approaching an SSD-only era, as more and more new computers are equipped only with SSD disks. The actual tests were not performed on solid-state drives but their internal algorithms of garbage collection and TRIM basically wipe the content of deleted files [4]. The allocation of clusters may still be BFA but recovering deleted files is mostly not possible due to TRIM wiping.

The Finite-state-machine approach is probably not possible to implement with new File Systems. For example Apple New Files System (APFS) [8] was designed from scratch to ignore all limitation of previous file systems as it was designed with SSD and parallel writes/reads in mind.

12. Bibliography

[1] Amirani M.C., Toorani M., Shirazi A.A.B., "A new approach to content-based file type detection", *Proceedings of the 13th IEEE Symposium on Computers and Communications* (*ISCC'08*), pp. 1103–1108, 2008.

[2] Bayne E., Ferguson R.I., Sampson A.T., "OpenForensics: A digital forensics GPU pattern matching approach for the 21st century", *Digital Investigation*, Vol. 24, Supplement, S29–S37 (2018).

[3] Carrier B., *File System Forensic Analysis*, Addison Wesley Professional, New York, 2005.

[4] Darnowski F., "Bezpieczne usuwanie danych z dysków SSD", in: *Przestępczość Teleinformatyczna*, pp. 109–115, Szczytno 2011.

[5] Darnowski F., Chojnacki A., "Selected Methods of File Carving and Analysis of Digital Storage Media in Computer Forensic", *Przegląd Teleinformatyczny*, T.3(21) Nr 1-2 (39) 2015.

[6] Garfinkel S.L., "Carving contiguous and fragmented files with fast object validation", *Digital Investigation*, Vol. 4(1), pp. 2–12, 2007.

[7] Garfinkel S.L., "Random sampling with sector identification", *Naval Postgraduate School presentation*, 2010.

[8] Garfinkel S.L., Nelson A., White D., Roussev V., "Using purpose-built functions and block hashes to enable small block and sub-file forensics", *Digital Investigation*, Vol. 7, 13–23 (2010).

[9] Ghotge V., Nema P., "Description of the Cluster Preallocation Algorithm in the NTFS File System", *Microsoft Product Support Services White Paper*, 2004.

[10] Gladyshev P., Patel A., "Finite State Machine Approach to Digital Event Reconstruction", *Digital Investigation*, Vol. 1, Issue 2, 130–149 (2004).

[11] Gladyshev P., Patel A., "Finite state machine analysis of a blackmail investigation", *International Journal of Digital Evidence*, Vol. 4, Issue 1, 1–13 (2005).

[12] Hansen K.H., Toolan F., "Decoding the APFS file system", *Digital Investigation*, Vol. 22, 107–132 (2017).

[13] http://www.netmarketshare.com/ (accessed 22.01.2019).

[14] Olivier M., "Scientific theory of digital forensic", in: *Advances in Digital Forensics XII*, 12th IFIP WG 11.9 International Conference, New Delhi, January 4–6, 2016, pp. 3–24.

[15] Penrose P., Buchanan W.J., Macfarlane R., "Fast contraband detection in large capacity disk drives", *Digital Investigation*, 12 (S1), S22-S29 (2015).

[16] Roussev V. Chow K.P., Shenoi S., "Data fingerprinting with similarity digests", in: *Advances in digital forensics VI*, Sixth IFIP WG 11.9 International Conference on Digital Forensics, pp. 207–226, Springer Berlin Heidelberg, 2010.

[17] Roussev V., Quates C., Martell R., "Real--time digital forensics and triage", *Digital Investigation*, Vol. 10, Issue 2, 158–167, (2013).

[18] Škrbina N., Stojanovski T., "Using parallel processing for file carving", *Proceedings of the Nineth Conference on Informatics and Information Technology* (*CIIT 2012*), pp. 175–179, 2012.

[19] Wei Y., Zheng N., Xu M., "An automatic Carving Method for RAR File Based on Content and Structure", *Proceedings of the 2010 Second International Conference on Information Technology and Computer Science*, pp. 68–72, IEEE, 2010.

[20] Veenman C., "Statistical Disk Cluster Classification for File Carving", in: *Proceedings of the Third International Symposium on Information Assurance and Security*, pp. 393–398, 2007.

[21] Young J., Foster K., Garfinkel S., Fairbanks K., "Distinct sector hashes for target file detection", *Computer*, Vol. 45, Issue 12, 28–35 (2012).

Zapis i kasowanie plików na twardych dyskach z systemem plików NTFS

F. DARNOWSKI, A. CHOJNACKI

Celem artykułu jest przedstawienie szczegółowych informacji na temat zapisu oraz kasowania plików na dyskach twardych wyposażonych w system plików NTFS (ang. *New Technology File System*). Najważniejsze przy tym są: algorytm wykorzystywany przez komputer do zapisu danych BFA (ang. *Best Fit Algorithm*) oraz algorytm FFT (ang. *First Free Algorithm*) aktualizacji pliku $MFT (ang. *Master File Table*). Zaprezentowano pewną konwencję opisu obszarów na dysku twardym. Przedstawione zasady zostały zweryfikowane na przykładach.

Słowa kluczowe: dysk twardy, NTFS, $MFT.