Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Implementácia prístupu k súborovému systému exFAT

(diplomová práca)

Marek Ludha

Školiteľ: RNDr. Jaroslav Janáček

Bratislava, 2009

Čestne prehlasujem, že som túto prácu vypracoval samostatne s použitím citovaných zdrojov a s pomocou školiteľa.

. . . . . .

Abstrakt

exFAT je súborový systém, podporu ktorého na osobné počítače priniesol Service Pack 1 pre operačný systém Windows Vista. V čase písania tohoto dokumentu neboli o ňom dostupné žiadne technické informácie (až na stručné vymenovanie niektorých nových vlastností). Cieľom tejto práce je umožniť prístup k tomuto súborovému systému (pomocou) softvéru s otvoreným zdrojovým kódom. V prvých dvoch kapitolách prináša prehľad súborového systému FAT a niektorých vlastností NTFS, ktorých znalosť je potrebná na pochopenie fungovania exFAT. V tretej kapitole načrtávam spôsob, akým som dospel k prezentovaným informáciam o exFAT. Nasleduje technická dokumentácia, ktorá zhŕňa poznatky tretej kapitoly a dopĺňa ich informáciami získanými štúdiom [Hir]. Súčasťou tejto práce je patch pre Linux 2.6, ktorý umožňuje čítanie z tohoto súborového systému.

Použité konvencie

V tejto práci sa snažím používať slovenské ekvivalenty odborných výrazov všade, kde takýto ekvivalent existuje a je rozumne zaužívaný a známy. Napriek tomu nechcem obetovať zrozumiteľnosť a jednoznačnosť textu snahe o čistotu jazyka. Preto budem bez ostychu používať žargónové výrazy všade, kde nenájdem rozumnejšiu alternatívu. So slovami „aplikačný rámec“, „úzke hrdlo“ a „zostavenie“ pozdravujem jazykovedcov.

Všetky čísla, ktorých zápis začína prefixom 0x sú v šestnástkovej sústave, ostatné čísla sú v desiatkovej sústave. Všade, kde hovorím o binárnej reprezentácii čísel, predpokladám spôsob zápisu little endian. To znamená byty čísla sú uložené tak, že najmenej významný byte je na najnižšom offsete. Pri číslovaní bitov má najmenej významný bit vždy číslo 0. Rovnako sektory sú číslované od 0.

Pri určovaní množstva dát budem používať SI predpony v zaužívanom význame (ktorý je v rozpore so štandardizovaným významom). Predponám IEC (MiB, KiB) sa budem vyhýbať. To znamená, že napríklad 1 MB = 1024 kB = 1048576 B.

Úvod

exFAT je súborový systém, ktorý si kladie za cieľ adresovať niektoré problémy, o ktorých sa očakáva, že vzniknú v blízkej budúcnosti v oblasti prenosných médii (USB kľúče, pamäťové karty). Situácia tu bude podobná ako okolo roku 2000, keď veľkosti pevných diskov presahovali 10 GB a súborový systém FAT32 sa začal stávať neefektívnym pre narábanie s takýmito médiami. Dôvodom bol nárast počtu clusterov, v dôsledku čoho vzrástla veľkosť FAT tabuľky a predĺžila sa doba vykonávania operácii, ktoré intenzívne narábajú s FAT tabuľkou, ako hľadanie voľného miesta, hľadanie dát priradených k súboru. Zvýšenie veľkosti clustra zasa viedlo k neakceptovateľnému plytvaniu s voľným miestom [Koz]. Táto situácia bola riešená nasadením súborového systému NTFS, ktorý dosahuje na veľkých diskoch lepší výkon [Mik]. Ten ale nie je vhodný pre prenosné zariadenia, ktoré často narábajú práve s USB kľúčmi a pamäťovými kartami, pretože je relatívne zložitý a práca s ním vyžaduje veľké množsto operačnej pamäte.

Prínos tejto práce spočíva vo vytvorení dokumentácie a softvéru pre exFAT pod slobodnou licenciou, čo priamo zvýši použiteľnosť operačného systému GNU/Linux tým, že umožní jeho lepšiu interoperabilitu s operačnými systémami Microsoft Windows, ktoré sú v kategórii desktopových operačných systémov dominantné. Vytvorenie dokumentácie a vzorovej implementácie ďalej zjednoduší vytváranie podpory pre exFAT pre iné operačné systémy.

Obsah

Kapitola 1

FAT

„You can't be late until you show up."
(popular saying)

Informácie z tejto kapitoly pochádzajú z [Mic06] a čiastočne z vlastných skúseností.

Súborový systém FAT (ďalej len FAT) existuje v troch verziách, označovaných FAT12, FAT16 a FAT32. Väčšina informácii je spoločná pre všetky verzie, na prípadné odlišnosti upozorním jednotlivo.

Pri popise FAT budú všetky čísla sektorov relatívne k priestoru určenému pre tento súborový systém, pričom budem predpokladať, že ho je možné určiť bez znalosti samotného súborového systému. Pri pevných diskoch to vyplýva z delenia disku na partície, USB kľúče a diskety majú väčšinou vyhradenú celú kapacitu pre jediný súborový systém.

Základná štruktúra FAT na disku je nasledujúca:

1. oblasť boot sektora
2. FAT tabuľka (tabuľky)
3. koreňový adresár
4. dátová oblasť

Najprv popíšem použité dátové štruktúry, potom sa budem venovať výpočtom nevyhnutným pre narábanie s FAT.

1.1 Oblasť boot sektora

Prvý sektor tejto oblasti sa nazýva boot sektor, pretože pri štarte operačného systému z tohoto súborového systému je načítaný BIOSom do pamäte a spustený. Tento sektor prešiel historicky istým vývojom, zaujímať nás bude iba jeho konečný stav. Jeho štruktúra je zhrnutá v nasledujúcej tabuľke:

Tabuľka 1.1: Štruktúra boot sektoru

Názov	Offset	Dĺžka	Význam
JMPBoot	0	3	Skok na kód, ktorý načítava operačný systém. Preskakuje nasledujúce údaje, ktoré nepredstavujú spustiteľný kód.
OEMName	3	8	Podľa oficiálnej dokumentácie nemá žiadny význam, odporúča sa pri formátovaní nastaviť na „MSWIN4.1“ a pri čítaní ignorovať.
BytesPerSec	11	2	Hodnota počtu bytov na sektor, ktorá bola použitá pri nízkoúrovňovom formátovaní média. Odporúča sa používať len 512, ďalšie možné hodnoty sú 1024, 2048 a 4096.
SecPerClus	13	1	Počet sektorov na alokačnú jednotku (cluster). Oficiálne podporované hodnoty sú 1, 2, 4, 8, 16, 32, 64 a 128. Ďalšie obmedzenie je, že cluster nesmie obsahovať viac ako 32 kB.
RsvdSecCnt	14	2	Počet sektorov v oblasti boot sektora. Pre FAT12 a FAT16 sa používa 1, pre FAT32 typicky 32. Akákoľvek nenulová hodnota je prípustná.
NumFATs	16	1	Počet kópii FAT tabuľky. Väčšinou sa používajú dve kópie, aby dáta ostali čitateľné aj po zlyhaní sektoru v jednej z FAT tabuliek. FAT32 dovoľuje špecifikovať ktorá kópia sa má používať. Niektoré rozšírenia súborového systému FAT používajú jednu kópiu ako zálohu pri journalovaných operáciach.
RootEntCnt	17	2	Pre FAT12 a FAT16 počet záznamov v koreňovom adresáry zaokrúhlený nahor tak, ako keby bol posledný využitý sektor zaplnený. Pre FAT32 vždy 0.

Tabuľka 1.1: Štruktúra boot sektoru (pokr.)

Názov	Offset	Dĺžka	Význam
TotSec16	19	2	Počet sektorov vyhradených pre súborový systém. Ak je nastavený na 0, počet sektorov je v TotSec32.
MediaDesc	21	1	Identifikátor média. V minulosti mal každý typ média použiteľný s FAT priradený jednoznačný identifikátor, ktorý určoval jeho geometriu. V súčasnosti sa nepoužíva. Musí byť nastavený rovnako ako prvý záznam FAT tabuľky. Typicky sa používa 0xF8, ďalšie prípustné hodnoty sú 0xF0, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, a 0xFF.
FATSize16	22	2	Pre FAT12/FAT16 počet sektorov obsadených jednou kópiou FAT tabuľky, pre FAT32 musí byť 0 (používa sa 32b verzia tejto hodnoty).
SecPerTrk	24	2	Počet sektorov na stope. Relevantný iba pre média s geometriou. Používal sa pri čítaní pomocou starých prerušení BIOSu.
NumHeads	26	2	Počet hláv. Rovnaká relevantnosť ako pri predchádzajúcom zázname.
HiddSec	28	4	Počet sektorov na médiu pred začiatkom miesta vyhradenom pre tento súborový systém.
TotSec32	32	4	Počet sektorov na médiu. Ak je nastavený na 0, potom platí hodnota TotSec16 (a musí byť nenulová). Pre FAT32 nesmie byť nula.
Signature	510	2	Hodnota 0xAA55. BIOS podľa tejto hodnoty pri bootovaní určuje, či sú údaje v boot sektore platné.

Od tohoto bodu sa formát boot sektora líši pre FAT12/FAT16 a FAT32.

Tabuľka 1.2: Formát boot sektora špecifický pre FAT12 / FAT16

Názov	Offset	Dĺžka	Význam
DrvNum	36	1	0 pre diskety, 0x80 pre pevné disky. Umožnil skrátenie bootovacieho kódu MS-DOSu o pár bytov.
Reserved1	37	1	Windows NT si sem ukladá nedokumentované informácie. Pri formátovaní treba nastaviť na 0.
BootSig	38	1	Hodnota 0x29. Indikuje, že nasledujúce 3 záznamy sú prítomné (pre kompatibilitu s historickými formátmi).
VolID	39	4	Hodnota známa ako „volume serial number“. Pri formátovaní sa nastaví na pseudo-náhodnú hodnotu, ktorá býva odvodená od aktuálneho času. Využíva sa pri detekcii, či nedošlo k výmene média medzi prístupmi (a teda či údaje uložené v cache stále platia).
VolLab	43	11	Názov média/partície, ktorý nastavil používateľ. Musí byť presná kópia názvu z koreňového adresára.
FSType	54	8	Jeden z reťazcov „FAT12 “, „FAT16 “, alebo „FAT “ (pre FAT32 „FAT32 “). Všetky reťazce sú medzerami doplnené na 8 znakov. Odporúča sa na nastaviť ho podľa typu FAT, ale nemá žiadnu informačnú hodnotu (slúži na „oklamanie nepriateľa“).

Tabuľka 1.3: Formát boot sektora špecifický pre FAT32

Názov	Offset	Dĺžka	Význam
FATSize32	36	4	Počet sektorov obsadených jednou kópiou FAT tabuľky.
ExtFlags	40	2	Bity 0-3 udávajú číslo aktívnej FAT tabuľky (začínajúc od 0). Ak je bit 7 nastavený, aktívna je len jedna kópia. Nulový bit 7 znamená, že zmeny sa zapisujú do všetkých kópii FAT tabuľky.

Tabuľka 1.3: Formát boot sektora špecifický pre FAT32 (pokr.)

Názov	Offset	Dĺžka	Význam
FSVer	42	2	Pôvodný zámer bol indikovať tu verziu súborového systému, aby sa zabezpečilo jeho jednoduché rozširovanie do budúcnosti. V praxy ostalo vždy pri verzii 0.
RootClus	44	4	Prvý cluster koreňového adresára. Kvôli uľahčeniu obnovy poškodeného súborového systému sa odporúča použiť prvý cluster, ktorý neobsahuje chybné sektory (na disku bez chybných sektorov 2).
FSInfo	48	2	Číslo sektora, ktorý obsahuje štruktúru FSInfo. Väčšinou 1.
BkBootSec	50	2	Číslo sektora, od ktorého začína kópia oblasti boot sektora. Odporúča sa použiť 6. Prípadné zmeny tejto oblasti sa do kópie nezapisujú. Slúži na obnovu pre prípad zlyhania hlavnej kópie.
Reserved2	52	12	Hodnota 0.

Nasledujúce záznamy majú rovnaký význam ako pri FAT12/FAT16, ale iný offset:

Tabuľka 1.4: Offsety presunutých záznamov pre FAT32

Názov	Offset	Dĺžka
DrvNum	64	1
Reserved1	65	1
Bootsig	66	1
VolID	67	4
VolLab	71	11
FSType	82	8

Pri FAT32 je súčasťou oblasti boot sektora aj štruktúra FSInfo. Jej úlohou je zmierniť problémy vyplývajúce z toho, že FAT32 môže mať obrov-

ské množstvo záznamov vo FAT tabuľke a prechádzanie všetkých záznamov, ktoré bolo v minulosti súčasťou bežných operácii, už nie je žiadúce.

Tabuľka 1.5: Formát sektora so štruktúrou FSInfo

Názov	Offset	Dĺžka	Význam
LeadSig	0	4	Hodnota 0x41615252. Indikuje, že informácie v tomto sektore sa dajú interpretovať ako štruktúra FSInfo a nedostali sa sem len náhodou.
Reserved1	4	480	Byty s nulovou hodnotou.
StrucSig	484	4	Hodnota 0x61417272. Podobne ako LeadSig.
FreeClusCount	488	4	Posledná známa hodnota počtu voľných clusterov. Hodnota 0xFFFFFFFF znamená, že ich počet nie je známy a treba ich spočítať. Slúži na rýchle (a približné) určenie voľného miesta.
NextFree	492	4	Slúži ako rada ovládaču pre FAT, od ktorého clustera má význam hľadať neobsadené clustere. Hodnota 0xFFFFFFFF znamená, že táto skutočnosť nie je známa (a teda hľadať treba od clustera 2).
Reserved2	496	12	Byty s nulovou hodnotou.
TrailSig	508	4	Hodnota 0xAA550000. Po zapísaní budu posledné dva byty zodpovedať signatúre boot sektora.

1.2 FAT tabuľka

FAT tabuľka je dátová štruktúra, ktorá pre každý cluster obsadený súborom definuje nasledujúci cluster obsadený tým istým súborom. Pre ostatné clustre oznamuje ich stav. Tieto záznamy majú dĺžku 12b, 16b alebo 32b podľa typu FAT. Prípustné hodnoty pre FAT32 sú nasledujúce:

Hodnoty pre ostatné typy FAT sa dajú odvodiť použitím bitového AND s jednotkovou maskou príslušnej dĺžky (0xFFFF pre FAT16 a 0x0FFF pre FAT12).

Tabuľka 1.6: Prípustné hodnoty pre FAT32

Rozsah hodnôt	Význam
0x00000000	voľný cluster
0x00000001 - 0x0FFFFFF6	obsadený cluster
0x0FFFFFF7	chybný cluster
0x0FFFFFF8 - 0x0FFFFFFF	posledný cluster patriaci danému súboru

FAT12 a FAT16 nemajú nikde uložený počet voľných clusterov, teda jediný spôsob ako zistiť množstvo voľného miesta je spočítať FAT záznamy s hodnotou 0. FAT32 môže mať uložený posledný známy stav tejto hodnoty v štruktúre FSInfo.

Dĺžka samotného záznamu pre FAT32 je v skutočnosti len 28 bitov. To znamená, že horné 4 bity sú rezervované, a teda pri zapisovaní ich treba uchovať a pri čítaní ignorovať. Pri vytváraní súborového systému sa odporúča nastaviť ich na 0. Ďalej z toho napríklad vyplýva, že hodnoty 0x10000000, 0xF0000000, aj 0x00000000 označujú prázdny cluster.

Číslovanie clusterov začína od 2, takže prvé dva záznamy vo FAT (s indexom 0 a 1) majú zvláštny význam. Prvý obsahuje kópiu média deskriptora z boot sektoru v najnižších 8 bitoch, zvyšné bity sú doplnené jednotkami. Druhý záznam obsahuje pri vytvorení súborového systému hodnotu indikujúcu posledný cluster. Pri FAT16 a FAT32 sa následne najvyšší bit používa na indikáciu korektného odpojenia súborového systému. Ak je nastavený na 0, znamená to, že súborový systém nebol korektne odpojený a teda niektoré údaje sa nemuseli zapísať z cache a odporúča sa skontrolovať konzistentnosť pred nasledujúcim pripojením. Druhý najvyšší bit indikuje, že pri poslednom pripojení boli nájdené nové chybné sektory, ak je nastavený na 0. To znamená, že sa odporúča testovať čitateľnosť všetkých sektorov s cieľom overiť, ktoré všetky sektory sú chybné.

Pri FAT32 sa teoreticky môže stať, že 0x0FFFFFF7 bude číslo clustera. V takomto prípade by sa jeho výskyt vo FAT tabuľke nedal odlíšiť od príznaku chybného clustera. Preto treba pri vytváraní FAT32 súborového systému dbať na to, aby najvyššie číslo clustera bolo nanajvýš 0x0FFFFFF6. Pri FAT12 a FAT16 je obmedzený maximálny počet clusterov, preto nejednoznačnosť tejto hodnoty nehrozí.

Keďže FAT12 má dĺžku záznamu 1.5 B a prípustné dĺžky sektora nie sú jej celočíselnými násobkami, jeden záznam môže v tomto prípade presahovať

hranice sektora. Tento prípad je pre nás nezaujímavý, preto sa ním nebudeme zaoberať.

Posledná poznámka sa týka dĺžky oblasti pre FAT tabuľku. Hodnoty FATSize16 a FATSize32 udávajú koľko sektorov je vyhradených pre FAT tabuľku. To neznamená, že všetky vyhradené sektory budú použité. Takisto je málo pravdepodobné, že posledný sektor bude použitý FAT tabuľkou do konca. O tomto nevyužitom mieste neplatia žiadne predpoklady.

1.3 Štruktúra pre záznam adresárov

Adresár je podobný súboru v tom, že má priradené miesto v dátovej oblasti a táto skutočnosť je zaznamenaná vo FAT tabuľke. Tu uložené dáta budem nazývať záznamom príslušného adresára. Líši sa jedine tým, že má nastavený atribút, ktorý indikuje, že ide o adresár. Koreňový adresár sa vo FAT12 a FAT16 odlišuje od ostatných v tom, že jeho dáta sú uložené v oblasti koreňového adresára (pri FAT32 je dĺžka tejto oblasti 0 a jeho dáta sú v dátovej oblasti). Z toho vyplýva, že pri FAT12 a FAT16 je počet záznamov v koreňovom adresáry obmedzený miestom, ktoré je pre túto oblasť vyhradené (jeho veľkosť sa ukladá do boot sektora). Naopak pri FAT32 je nutné priestor pre dáta koreňového adresára alokovať, teda môže dôjsť k jeho fragmentácii. Z historických dôvodov existuje limit 65535 na počet záznamov v ľubovoľnom adresáry.

V koreňovom adresáry sa môže nachádzať súbor s atribútom VOLUME_ID. Tento súbor musí mať nulovú dĺžku a jeho názov určuje názov partície. Táto hodnota musí byť rovnaká ako v boot sektore.

Schopnosť ukladať dlhé názvy súborov bola do FAT pridaná až s príchodom operačného systému Windows. Keďže pri pôvodnom návrhu sa s touto vlastnosťou nepočítalo, reprezentácia dlhých názvov je značne komplikovaná a neprehľadná. Naštastie sa pri exFAT používa nový spôsob ukladania názvov, takže táto vlastnosť FAT pre nás nebude podstatná.

Krátky názov súborov sa pred uložením skonvertuje na veľké písmená. Následne sa uloží do 11 znakov tak, že časť pred bodkou sa uloží do prvých 8 znakov a časť za bodkou do posledných 3 znakov. Z toho vyplýva, že samotná bodka sa nikde nezapisuje. Nevyužité znaky sa doplnia medzerou.

Prvý znak názvu môže mať špeciálny význam. Ak má hodnotu 0xE5, ide o záznam o zmazanom súbore, pričom tento záznam sa považuje za voľný. Ak má hodnotu 0x00, potom je tento záznam voľný a navyše za ním nenasle-

dujú žiadne ďalšie záznamy. Ak má hodnotu 0x05, potom prvý znak názvu príslušného súboru je 0xE5.

Každý adresár okrem koreňového musí na začiatku obsahovať záznamy o špeciálnych adresároch. Prvý záznam má meno „. “ (počet medzier na konci je 10) a prvý cluster rovnaký ako adresár, v ktorom je obsiahnutý. Druhý záznam má meno „.. “ (9 medzier na konci) a prvý cluster rovnaký ako nadradený adresár. Obidva majú všetky časy a dátumy nastavené podľa adresára, ktorý ich obsahuje.

Dáta adresára pozostávajú z lineárne zoradených 32 B štruktúr. Ak si odmyslíme dlhé názvy, každá z nich predstavuje záznam jedného súboru. Z kontextu bude vždy zrejmé, či záznamom adresára myslím túto dátovú štruktúru alebo dáta o obsahu adresára.

Tabuľka 1.7: Štruktúra adresárového záznamu

Názov	Offset	Dĺžka	Význam
Name	0	11	Krátke meno súboru.
Attr	11	1	Atribúty súboru. Prítomnosť atribútu sa indikuje nastavením príslušného bitu. Možné atribúty sú nasledujúce: 0x01 READ_ONLY 0x02 HIDDEN 0x04 SYSTEM 0x08 VOLUME_ID 0x10 DIRECTORY 0x20 ARCHIVE Atribút ARCHIVE je nastavený pri vytvorení súboru, dá sa použiť v zálohovacích utilitách na indikáciu, že príslušný súbor nebol zálohovaný. Význam atribútov READ_ONLY, HIDDEN a SYSTEM je intuitívne jasný, ich presné použitie závisí od operačného systému. Horné dva bity sú rezervované. Pri vytvorení súboru ich treba vynulovať a pri čítaní ignorovať.
NTReserved	12	1	Rezervované. Pri vytváraní treba nastaviť na 0, pri čítaní ignorovať.

Tabuľka 1.7: Štruktúra adresárového záznamu (pokr.)

Názov	Offset	Dĺžka	Význam
CreateTimeTenth	13	1	Oficiálna dokumentácia sa pri význame tejto hodnoty rozchádza v tvrdeniach (sama so sebou). Najprv hovorí o hodnote tisícin sekundy v čase vytvorenia súboru („Millisecond stamp at file creation time.“) Neskôr spomína, že ide o počet desatín sekundy („this field is a count of tenths of a second“). Z rozsahu platných hodnôt a kontextu usudzujem, že v skutočnosti je význam tohoto záznamu „počet stotín sekundy, ktorý treba pripočítať k sekundám z CreateTime, aby sme dostali presnejšiu časovú informáciu“ a ukladá sa sem počet stotín sekundy od poslednej párnej sekundy. Platný rozsah hodnôt je 0-199.
CreateTime	14	2	Čas vytvorenia súboru.
CreateDate	16	2	Dátum vytvorenia súboru.
LastAccessDate	18	2	Dátum posledného prístupu k súboru (čítanie alebo zápis). Čas posledného prístupu sa neukladá.
FirstClusHI	20	2	Horné dva byty z čísla prvého clustera. Pre FAT12 a FAT16 vždy 0.
WriteTime	22	2	Čas posledného zápisu do súboru.
WriteDate	24	2	Dátum posledného zápisu do súboru.
FirstClusLO	26	2	Spodné dva byty z čísla prvého clustera.
FileSize	28	4	Veľkosť súboru v bytoch.

1.4 Formáty času a dátumu

Tento formát sa používa pre čas vytvorenia a posledného zápisu a prístupu k súboru. Kóduje sa ako čas uplynutý od začiatku dňa 1.1.1980. Čas má granularitu 2 sekundy, z toho vyplýva že všetky časy v tomto formáte obsahujú párny počet sekúnd. Jediná výnimka je čas vytvorenia, pre ktorý existuje

hodnota CreateTimeTenth. Táto spresňuje čas vytvorenia súboru až na desatiny sekundy.

Tabuľka 1.8: Formát dátumu

Bity	Význam
0-4	Deň v mesiaci, platný rozsah hodnôt 1-31
5-8	Mesiac v roku, platný rozsah 1-12
9-15	Počet rokov od 1980, platný rozsah 0-127

Tabuľka 1.9: Formát času

Bity	Význam
0-4	Polovica počtu uplynutých sekúnd, platný rozsah hodnôt 0-29
5-10	Minúty, platný rozsah hodnôt 0-59
11-15	Hodiny, platný rozsah hodnôt 0-23

1.5 Postup čítania FAT

Prvým sektorom súborového systému FAT je boot sektor, takže dáta z neho spoločné pre všetky typy FAT sú ľahko lokalizovateľné hneď na začiatku. Priamo v ňom je uložený počet sektorov v oblasti boot sektora, za ktorou tesne nasleduje prvá kópia FAT tabuľky. Počet sektorov obsadených jednou kópiou FAT tabuľky ako aj počet FAT tabuliek je takisto uložený tu, čím sa dostávame na začiatok oblasti koreňového adresára. Pre FAT32 táto oblasť prakticky neexistuje, ale tento špeciálny prípad je ošetrený tým, že počet záznamov v ňom je nastavený na 0. Veľkosť miesta obsadeného koreňovým adresárom vypočítame vynásobením počtu záznamov ich dĺžkou (32B) a zaokrúhlením nahor:

$$RootDirSectors = (RootEntCnt * 32 + (BytesPerSec - 1))/BytesPerSec$$

Za koreňovým adresárom nasleduje dátová oblasť:

$$FirstDataSector = RsvdSecCnt + (NumFATs * FATSize) + RootDirSectors$$

(za hodnotu FATSize považujeme tú z hodnôt FATSize16 a FATSize32, ktorá je nenulová)

Následne určíme typ FAT. Je dôležité uvedomiť si, že FATSize udáva, koľko sektorov je alokovaných pre FAT a môže byť o dosť viac ako je v skutočnosti potrebných, preto sa na tento účel nehodí. Typ FAT sa určuje výhradne podľa počtu prítomných clusterov (čo je odlišné ako najvyššie číslo clustera, keďže clustre sa číslujú od 2). Počet sektorov v dátovej oblasti určíme ako rozdiel všetkých sektorov a sektorov v ostatných oblastiach:

$$DataSec = TotSec - (RsvdSecCnt + (NumFATs * FATSize) + RootDirSectors)$$

(za hodnotu TotSec považujeme tú z hodnôt TotSec16 a TotSec32, ktorá je nenulová)

Ďalej určíme, koľko clustrov je v týchto sektoroch obsiahnutých:

$$CountofClusters = DataSec/SecPerClus$$

O FAT12 ide vtedy, keď je počet clustrov nižší ako 4085. Ak nejde o FAT12 a počet clustrov je nižší ako 65525, ide o FAT16. Ak nenastal ani jeden z týchto prípadov, ide o FAT32.

Pri prístupe k dátam súboru bude potrebné určiť pre daný cluster N kde vo FAT tabuľke sa nachádza záznam o nasledujúcom clustri. Pri FAT16 a FAT32 je tento výpočet jednoduchý, číslo príslušného sektora dostaneme keď k začiatku FAT pripočítame počet sektorov, ktoré sa nachádzajú pred týmto záznamom:

$$ThisFATSecNum = RsvdSecCnt + (N * FATType/BytsPerSec)$$

(FATType je počet bytov na záznam FAT tabuľky, 2 pre FAT16 a 4 pre FAT32). Offset v tomto sektore dostaneme výpočtom jednoduchého modula:

$$ThisFATEntOffset = (N * FATType)\%BytsPerSec$$

Pri FAT12 je tento výpočet zložitejší, pretože záznam môže začínať uprostred bytu a môže presahovať hranicu sektora. Keďže exFAT používa dĺžku FAT záznamu výhradne 4B, nebudeme sa týmto prípadom zaoberať.

Posledný dôležitý výpočet je určiť číslo prvého sektora pre dané číslo clustera N. Z doteraz vypočítaných hodnôt to ide priamočiaro:

$$FirstSectorofCluster = FirstDataSector + (N - 2) * SecPerClus$$

Kapitola 2

NTFS

„Life would be much simpler and things would get done much faster if it weren't for other people."
Blore

Súborový systém exFAT si požičal niekoľko nápadov zo staršieho súborového systému NTFS, preto sa nimi budeme v krátkosti zaoberať. Táto kapitola čerpá z [pro].

2.1 Bitmapa

Bitmapa určuje ktoré clustre sú voľné (a ktoré obsadené). Pre každý cluster obsahuje jeden bit, pričom nastavený bit znamená alokovaný cluster. Veľkosť bitmapy musí byť zaokrúhlená na najbližší vyšší násobok 8 bytov, pričom záznamy pre clustre vzniknuté v dôsledku zaokrúhľovania sú nastavené na 1 (obsadené).

V praxy sa bitmapa používa na rýchle nájdenie voľného miesta, pretože na uloženie záznamu o jednom clustry potrebuje 1 bit, kým FAT32 potrebuje 4 byty, čo znamená 32-násobné zmenšenie potrebného miesta. Táto úspora sa priamo premietne do úspory času, keďže najpomalšia časť hľadania voľného miesta je práve čítanie FAT tabuľky/bitmapy.

2.2 Tabuľka veľkých písmen

Ide o tabuľku s 2 bytovými záznamami veľkosti 128kB, ktorá obsahuje 65536 záznamov. Pre každý code point z Basic Multilingual Plane obsahuje veľkú verziu daného písmena, ktorá bude použítá pri konverzii písmen z názvov súborov na veľké pri ich zoraďovaní a porovnávaní na zhodu. Súborový systém FAT tento problém neriešil, oficiálna dokumentácia iba zmieňuje, že konverzia na veľké písmená sa má riešiť podľa použitej kódovej stránky.

Kapitola 3

Analýza exFAT

„With a contrived example, you can prove anything.“
Joel Spolsky

V tejto kapitole popisujem postup, ktorým sa dá dospieť k rovnakým záverom o štruktúre súborového systému exFAT na disku, ako som dospel ja. Všetky informácie boli získané študovaním binárnych dát na disku vytvorených oficiálnymi ovládačmi a metódou pokus-omyl (nie reverzným inžinierstvom oficiálnych ovládačov). Použité verzie ovládača (súboru exfat.sys) sú 6.0.6001.16633 (kompilované 9. 9. 2007) a 5.1.2600.3453 (kompilované 29. 9. 2008). Na miestach, kde to dáva význam, sa budem snažiť zdôvodňovať závery ku ktorým som dospel.

3.1 Oficiálne informácie

V čase písania tejto práce sa mi podarilo nájsť jediný oficiálny zdroj technických informácii, [Gho06]. Za povšimnutie stoja nasledujúce skutočnosti:

- Za cieľ súborového systému exFAT sa považuje čo najväčšie podobnosť s FAT32 a súčasne vyriešenie niektorých jeho nedostatkov. Preto budem na začiatku predpokladať, že základné mechanizmy fungujú rovnako ako pri FAT32, až kým sa nepresvedčím o opaku.
- Prítomnosť bitmapy pre rýchlejšie hľadanie voľného miesta a mazanie. Budem predpokladať, že bitmapa bude podobná bitmape z NTFS.

- Na záznam veľkosti súboru je vyhradených 8 bytov, čo znamená maximálnu veľkosť súboru $2^{64}B$. Pre zápis veľkosti clustera je limit 2^{255}, z čoho sa dá očakávať, že na jeho zápis bude vyhradený jeden byte a zapisovať sa bude len exponent (veľkosť clustera je aj pri FAT obmedzená len na niektoré mocniny dvojky).

- Záznam adresára obsahoval pri FAT32 značné množstvo hackov kvôli ukladaniu dlhých názvov, preto predpokladám, že bude kompletne prerobený (čomu nasvedčuje aj informácia, že exFAT neukladá krátke názvy). Hovorí sa o jeho vysokej flexibilite, jednoduchosti pridávania nových vlastností a dobrej spätnej kompatibilite. Budem teda očakávať taký zápis informácii, pri ktorom sa dajú neznáme hodnoty ľahko ignorovať.

- K názvom súborov sa ukladá 2 bytový checksum.

3.2 Základné rozloženie exFAT

Používať budem USB kľúč s kapacitou 261 881 856 B, čo znamená 511 488 sektorov. Na začiatok ho naformátujem, známe parametre formátovania sú nasledujúce: volume label je usb_kluc, veľkosť clustera 4096 bytov, sériové číslo 7888-6DE6. Použité bolo kompletné (nie rýchle) formátovanie, ktoré vynuluje obsah všetkých sektorov, čím uľahčí odlišovanie skutočných dát od pozostatkov z predchádzajúceho použitia. Následne na neho skopírujem niekoľko súborov, v mojom prípade (v tomto poradí): exfat.sys, fmifs.dll, ifsutil.dll.

Prvý zaujímavý sektor je ten s poradovým číslom 0:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00   .v_EXFAT   .....
00000010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000040   00 00 00 00 00 00 00 00  00 CE 07 00 00 00 00 00   ................
00000050   80 00 00 00 F8 01 00 00  80 02 00 00 70 F9 00 00   _...o..._...pu..
00000060   06 00 00 00 E6 6D 88 78  00 01 00 00 09 03 01 80   ....am^x........_
00000070   01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
  ...               ...                    ...                      ...
000001E0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000001F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA   ..............Ua
```

Začiatok (skok 0xEB 0x76 0x90 a reťazec „EXFAT “) a signatúra (0xAA55) pripomínajú boot sektor súborového systému FAT, ale dáta začínajúce od offsetu 0x48 nie je možné týmto spôsobom interpretovať. K tomuto sektoru sa vrátim neskôr, keď budem o štruktúre súborového systému vedieť viac (budem vedieť aké hodnoty tu mám hľadať). Podľa periodicity, s akou sa opakuje obsah sektora to vyzerá tak, že oblasť boot sektora má v tomto prípade 12 sektorov a za ňou nasleduje jedna kópia. Tá sa líši len bytom s offsetom 0x70 v boot sektora, ktorý je teraz nulový. Z toho usudzujem, že jednotkový byte znamená hlavnú kópiu, nulový byte záložnú.

Ďalší zaujímavý sektor má poradové číslo 128:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   F8 FF FF FF FF FF FF FF  03 00 00 00 FF FF FF FF   oyyyyyyy....yyyy
00000010   05 00 00 00 FF FF FF FF  FF FF FF FF 00 00 00 00   ....yyyyyyyy....
00000020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

Začiatok indikuje, že sa jedná o začiatok FAT tabuľky s dĺžkou záznamu 32 bitov a média deskriptorom 0xF8. Trochu nečakané je, že táto tabuľka obsahuje niekoľko záznamov (použitý tvar je „cluster, ktorého sa záznam týka“ → „cluster za ním nasledujúci“): $2 \rightarrow 3$, $3 \rightarrow EOF$, $4 \rightarrow 5$, $5 \rightarrow EOF$, $6 \rightarrow EOF$. Veľkosť clustera je 4kB a dva z nahraných súborov presahujú veľkosťou 100kB, teda tieto záznamy sa musia týkať niečoho iného (a záznamy nahraných súborov chýbajú). Hľadanie deskriptora, ktorý je prítomný na začiatku FAT tabuľky, dalo iba jeden výsledok. Preto sa domnievam, že druhá kópia FAT nebola vytvorená.

V tomto okamihu je užitočné všimnúť si, že všetky EOF záznamy majú horné 4 bity (pri FAT32 rezervované) nastavené. To môže byť náhoda (keďže oficiálne o rezervovaných bitoch neplatia žiadne predpoklady), ale môže to aj znamenať, že záznamy vo FAT tabuľke využívajú plných 32 bitov. Aby som to overil, budem potrebovať médium, na ktoré sa zmestí rádovo 0x0FFFFFFF clusterov, čo pri najmenšej možnej veľkosti clustera 512 bytov znamená minimálne 128GB. Kvôli rezerve použijem 512GB disk, na ktorý nahrám súbor tak, aby zaberal aspoň dva clustre, ktorých číslo sa líši len najvyššími 4 bitmi a aby mal záznam vo FAT tabuľke. Zaujímavá časť FAT tabuľky potom vyzerá nasledovne:

```
Offset       0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0400FFFF0   FD FF FF 0F FE FF FF 0F  FF FF FF 0F 00 00 00 10   ý"".„"".""".....
040100000   01 00 00 10 02 00 00 10  03 00 00 10 04 00 00 10   ................
  ...              ...                    ...                     ...
0800FFFF0   FD FF FF 1F FE FF FF 1F  FF FF FF 1F 00 00 00 20   ý"".„""."""....
080100000   01 00 00 20 02 00 00 20  03 00 00 20 04 00 00 20   ... ... ... ...
```

Z toho vyplýva, že existujú clustre, ktorých čísla sa líšia iba hornými 4 bitmi, teda číslo clustra vo FAT tabuľke má plných 32 bitov. Navyše v opačnom prípade by hodnoty na offsetoch 0x0400FFFF0, 0x0400FFFF4 a 0x0400FFFF8 znamenali koniec súbora, čo by znamenalo problém s konzistenciou tabuľky.

Nasledujúci sektor, ktorý obsahuje dáta, je sektor 640:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   FF FF FF FF FF FF FF FF  FF 0F 00 00 00 00 00 00   yyyyyyyyy.......
00000010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

Tento sektor obsahuje 76 po sebe idúcich nastavených bitov. Ak by išlo o bitmapu, zodpovedalo by to 311 296 B využitého miesta. Nahrané súbory zaberajú spolu 282 624 B, po zaokrúhlení dĺžky každého súbora na celý cluster nahor 290 816 B. Po pripočítaní 5 clusterov, ktoré označila FAT tabuľka ako použité dostávame presne 311 296, čo podporuje teóriu, že ide o bitmapu. Ďalej to vyzerá tak, že 5 neočakávaných hodnôt vo FAT tabuľke skutočne znamenajú clustre obsadené niečím iným, ako mnou nahrané súbory. Keďže na kľúči je 511 488 sektorov a dĺžka clustra je 8 sektorov, očakávaná dĺka bitmapy je rádovo 16 sektorov (511488 / 8 / 8 / 512).

Obsah sektora za očakávaným koncom bitmapy (656):

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   00 00 01 00 02 00 03 00  04 00 05 00 06 00 07 00   ................
00000010   08 00 09 00 0A 00 0B 00  0C 00 0D 00 0E 00 0F 00   ................
00000020   10 00 11 00 12 00 13 00  14 00 15 00 16 00 17 00   ................
00000030   18 00 19 00 1A 00 1B 00  1C 00 1D 00 1E 00 1F 00   ................
00000040   20 00 21 00 22 00 23 00  24 00 25 00 26 00 27 00    .!."."#.$.%.&.'.
00000050   28 00 29 00 2A 00 2B 00  2C 00 2D 00 2E 00 2F 00   (.).*.+.,.-.../.
00000060   30 00 31 00 32 00 33 00  34 00 35 00 36 00 37 00   0.1.2.3.4.5.6.7.
00000070   38 00 39 00 3A 00 3B 00  3C 00 3D 00 3E 00 3F 00   8.9.:.;.<.=.>.?.
00000080   40 00 41 00 42 00 43 00  44 00 45 00 46 00 47 00   @.A.B.C.D.E.F.G.
00000090   48 00 49 00 4A 00 4B 00  4C 00 4D 00 4E 00 4F 00   H.I.J.K.L.M.N.O.
000000A0   50 00 51 00 52 00 53 00  54 00 55 00 56 00 57 00   P.Q.R.S.T.U.V.W.
000000B0   58 00 59 00 5A 00 5B 00  5C 00 5D 00 5E 00 5F 00   X.Y.Z.[.\.].^._.
000000C0   60 00 41 00 42 00 43 00  44 00 45 00 46 00 47 00   `.A.B.C.D.E.F.G.
000000D0   48 00 49 00 4A 00 4B 00  4C 00 4D 00 4E 00 4F 00   H.I.J.K.L.M.N.O.
000000E0   50 00 51 00 52 00 53 00  54 00 55 00 56 00 57 00   P.Q.R.S.T.U.V.W.
000000F0   58 00 59 00 5A 00 7B 00  7C 00 7D 00 7E 00 7F 00   X.Y.Z.{.|.}.~...
00000100   80 00 81 00 82 00 83 00  84 00 85 00 86 00 87 00   _._.'.f."...Ĺ.Ĺ.
00000110   88 00 89 00 8A 00 8B 00  8C 00 8D 00 8E 00 8F 00   ^.%...<.O.?...?.
00000120   90 00 91 00 92 00 93 00  94 00 95 00 96 00 97 00   _.'.'."."...-.-.
00000130   98 00 99 00 9A 00 9B 00  9C 00 9D 00 9E 00 9F 00   ~.t...>.o.?...Y.
00000140   A0 00 A1 00 A2 00 A3 00  A4 00 A5 00 A6 00 A7 00   ".!.c.L.Ď.Y.|...
00000150   A8 00 A9 00 AA 00 AB 00  AC 00 AD 00 AE 00 AF 00   ů.c.a.......R.-.
00000160   B0 00 B1 00 B2 00 B3 00  B4 00 B5 00 B6 00 B7 00   ř.+.2.3.ď.u.....
00000170   B8 00 B9 00 BA 00 BB 00  BC 00 BD 00 BE 00 BF 00   ..1.o...1.1.3...
00000180   C0 00 C1 00 C2 00 C3 00  C4 00 C5 00 C6 00 C7 00   A.-.¸.A.Ž.A.A...
00000190   C8 00 C9 00 CA 00 CB 00  CC 00 CD 00 CE 00 CF 00   E...E.Ő.I.Ö...I.
000001A0   D0 00 D1 00 D2 00 D3 00  D4 00 D5 00 D6 00 D7 00   ?.N.O.ŕ...O...ž.
000001B0   D8 00 D9 00 DA 00 DB 00  DC 00 DD 00 DE 00 DF 00   O.U.é.U.š.í.?.á.
000001C0   C0 00 C1 00 C2 00 C3 00  C4 00 C5 00 C6 00 C7 00   A.-.¸.A.Ž.A.A...
000001D0   C8 00 C9 00 CA 00 CB 00  CC 00 CD 00 CE 00 CF 00   E...E.Ő.I.Ö...I.
000001E0   D0 00 D1 00 D2 00 D3 00  D4 00 D5 00 D6 00 F7 00   ?.N.O.ŕ...O...ö.
000001F0   D8 00 D9 00 DA 00 DB 00  DC 00 DD 00 DE 00 78 01   O.U.é.U.š.í.?.x.
```

Obsah tohoto sektora nápadne pripomína tabuľku veľkých písmen z NTFS. Problém ale je, že určite nesiaha ďalej ako po sektor 672, čo znamená maximálne 8kB dát, pričom kompletná tabuľka veľkých písmen z NTFS má

128kB. Fungovaním tabuľky veľkých písmen sa ďalej zaoberám v kapitole 4.2.

Sektor 672 má zjavne odlišnú štruktúru, preto pravdepodobne nebude súčasťou tabuľky veľkých písmen:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   83 08 75 00 73 00 62 00  5F 00 6B 00 6C 00 75 00   f.u.s.b._.k.l.u.
00000010   63 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   c...............
00000020   81 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   _...............
00000030   00 00 00 00 02 00 00 00  2E 1F 00 00 00 00 00 00   ................
00000040   82 00 00 00 0D D3 19 E6  00 00 00 00 00 00 00 00   ’....ŕ.a........
00000050   00 00 00 00 04 00 00 00  CC 16 00 00 00 00 00 00   ........I.......
00000060   85 02 3F 43 20 00 00 00  01 35 44 38 25 4F 29 37   ..?C ....5D8%O)7
00000070   02 35 44 38 81 00 00 00  00 00 00 00 00 00 00 00   .5D8_...........
00000080   C0 03 00 09 5A 49 00 00  00 14 02 00 00 00 00 00   A...ZI..........
00000090   00 00 00 00 07 00 00 00  00 14 02 00 00 00 00 00   ................
000000A0   C1 00 65 00 78 00 66 00  61 00 74 00 2E 00 73 00   -.e.x.f.a.t...s.
000000B0   79 00 73 00 00 00 00 00  00 00 00 00 00 00 00 00   y.s.............
000000C0   85 02 72 4D 20 00 00 00  09 35 44 38 BA 4E 29 37   ..rM ....5D8oN)7
000000D0   0A 35 44 38 B4 00 00 00  00 00 00 00 00 00 00 00   .5D8ď...........
000000E0   C0 03 00 09 77 31 00 00  00 5A 00 00 00 00 00 00   A...w1...Z......
000000F0   00 00 00 00 29 00 00 00  00 5A 00 00 00 00 00 00   ....)....Z......
00000100   C1 00 66 00 6D 00 69 00  66 00 73 00 2E 00 64 00   -.f.m.i.f.s...d.
00000110   6C 00 6C 00 00 00 00 00  00 00 00 00 00 00 00 00   l.l.............
00000120   85 02 30 B0 20 00 00 00  0C 35 44 38 19 4F 29 37   ..0ř ....5D8.O)7
00000130   0D 35 44 38 48 00 00 00  00 00 00 00 00 00 00 00   .5D8H...........
00000140   C0 03 00 0B D0 B6 00 00  00 E2 01 00 00 00 00 00   A...?...........
00000150   00 00 00 00 2F 00 00 00  00 E2 01 00 00 00 00 00   ..../...........
00000160   C1 00 69 00 66 00 73 00  75 00 74 00 69 00 6C 00   -.i.f.s.u.t.i.l.
00000170   2E 00 64 00 6C 00 6C 00  00 00 00 00 00 00 00 00   ..d.l.l.........
00000180   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

Keďže sa v ňom nachádzajú názvy všetkých súborov, ktoré sú na kľúči nahraté, pôjde zrejme o záznam koreňového adresára. Podľa očakávania má štruktúru úplne odlišnú od štruktúry použitej vo FAT.

Pozrime sa na obsah tohoto sektora od konca a poďme otestovať teóriu, že záznam adresára sa skladá z 32B záznamov. Posledné nenulové dáta predstavuje hodnota 0x00C1 nasledovaná niečím, čo by mohlo byť názov posledne

nahraného súbora, ak by sme to interpretovali ako postupnosť znakov kódovaných UTF-16 LE. Prvá hodnota 0x00C1 evidentne nie je súčasť názvu. Podľa oficiálnych informácii má byť štruktúra záznamu adresára ľahko rozšíriteľná, čo by mohlo napovedať, že ide o identifikátor typu záznamu (v tomto prípade názov súboru). V tomto zázname nie je miesto pre ostatné informácie o súbore (časy, veľkosť, prvý cluster) a keďže ide o posledný záznam, budeme ich hľadať bližšie k začiatku sektora. Pred týmto záznamom sú záznamy s identifikátormi 0x03C0 a 0x0285, ktoré by mohli tieto informácie obsahovať. Ďalej nasleduje rovnaká postupnosť pre zvyšné dva súbory (0x00C1 pre názov, potom 0x03C0 a 0x0285). Na začiatku sektora sú záznamy 0x0082, 0x0081 a 0x0883, ktoré sú prítomné len raz a to na začiatku, preto sa zrejme nebudú týkať jednotlivých súborov, ale samotného súborového systému alebo adresára.

Na základe týchto zistení budem predpokladať, že súborov sa týkajú minimálne záznamy 0x0285, 0x03C0 a 0x00C1. Ďalej 0x00C1 znamená názov súboru a tohoto súboru sa týkajú všetky záznamy až po jeho názov, ktoré ešte neboli priradené inému súboru (pri čítaní zhora nadol). Záznam 0x0883 zjavne obsahuje volume label, pričom 8 je počet znakov v názve (teda nie je súčasť identifikátora).

Nasleduje prirodzená otázka, čo sa stane ak nejaký názov presiahne 15 znakov, ktoré sa dajú uložiť do príslušného 32 B záznamu. Grafické aplikácie zvyknú detekovať, že ide o súborový systém podobný FAT a limitujú dĺžku názvu pri zadávaní na 11 znakov. Preto použijem API funkciu SetVolumeLabel, ktorá dokáže úspešne nastaviť 15-znakový názov:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000060   83 0F 30 00 31 00 32 00  33 00 34 00 35 00 36 00   f.0.1.2.3.4.5.6.
00000070   37 00 38 00 39 00 30 00  31 00 32 00 33 00 34 00   7.8.9.0.1.2.3.4.
```

Pri pokuse o nastavenie dlhšieho názvu dôjde k chybe č.154 s pomerne samovysvetľujúcim textom: „The volume label you entered exceeds the label character limit of the target file system.“

Predĺžim názov súborov:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000100   41 00 66 00 6D 00 69 00  66 00 73 00 2E 00 64 00   A.f.m.i.f.s...d.
00000110   6C 00 6C 00 00 00 00 00  00 00 00 00 00 00 00 00   l.l.............
00000120   85 02 30 B0 20 00 00 00  0C 35 44 38 19 4F 29 37   a.0- ....5D8.O)7
00000130   0D 35 44 38 48 00 00 00  00 00 00 00 00 00 00 00   .5D8H...........
00000140   C0 03 00 0B D0 B6 00 00  00 E2 01 00 00 00 00 00   A...??...?......
00000150   00 00 00 00 2F 00 00 00  00 E2 01 00 00 00 00 00   ..../....?......
00000160   C1 00 69 00 66 00 73 00  75 00 74 00 69 00 6C 00   +.i.f.s.u.t.i.l.
00000170   2E 00 64 00 6C 00 6C 00  00 00 00 00 00 00 00 00   ..d.l.l.........
00000180   85 05 14 08 20 00 00 00  01 35 44 38 25 4F 29 37   a... ....5D8%O)7
00000190   62 19 45 38 81 00 00 00  00 00 00 00 00 00 00 00   b.E8............
000001A0   C0 03 00 2F 14 CD 00 00  00 14 02 00 00 00 00 00   A../.=..........
000001B0   00 00 00 00 07 00 00 00  00 14 02 00 00 00 00 00   ................
000001C0   C1 00 65 00 78 00 66 00  61 00 74 00 2E 00 73 00   +.e.x.f.a.t...s.
000001D0   79 00 73 00 20 00 73 00  20 00 6E 00 61 00 7A 00   y.s. .s. .n.a.z.
000001E0   C1 00 76 00 6F 00 6D 00  20 00 70 00 72 00 65 00   +.v.o.m. .p.r.e.
000001F0   64 00 6C 00 7A 00 65 00  6E 00 79 00 6D 00 20 00   d.l.z.e.n.y.m. .
00000000   C1 00 6E 00 61 00 20 00  73 00 74 00 75 00 64 00   +.n.a. .s.t.u.d.
00000010   69 00 6A 00 6E 00 65 00  20 00 75 00 63 00 65 00   i.j.n.e. .u.c.e.
00000020   C1 00 6C 00 79 00 00 00  00 00 00 00 00 00 00 00   +.l.y...........
00000030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000040   85 05 DF 10 20 00 00 00  09 35 44 38 BA 4E 29 37   a.-. ....5D8oN)7
00000050   AA 19 45 38 B4 00 00 00  00 00 00 00 00 00 00 00   a.E8+...........
00000060   C0 03 00 2E CE 29 00 00  00 5A 00 00 00 00 00 00   A...+)...Z......
00000070   00 00 00 00 29 00 00 00  00 5A 00 00 00 00 00 00   ....)....Z......
00000080   C1 00 66 00 6D 00 69 00  66 00 73 00 2E 00 64 00   +.f.m.i.f.s...d.
00000090   6C 00 6C 00 20 00 70 00  72 00 65 00 6D 00 65 00   l.l. .p.r.e.m.e.
000000A0   C1 00 6E 00 6F 00 76 00  61 00 6E 00 79 00 20 00   +.n.o.v.a.n.y. .
000000B0   61 00 62 00 79 00 20 00  76 00 7A 00 6E 00 69 00   a.b.y. .v.z.n.i.
000000C0   C1 00 6B 00 6C 00 61 00  20 00 66 00 72 00 61 00   +.k.l.a. .f.r.a.
000000D0   67 00 6D 00 65 00 6E 00  74 00 61 00 63 00 69 00   g.m.e.n.t.a.c.i.
000000E0   C1 00 61 00 00 00 00 00  00 00 00 00 00 00 00 00   +.a.............
000000F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

Ak je za názvom súboru miesto v zázname adresára, sú pridané nové záznamy, v ktorých názov pokračuje. Ak miesto nie je, celý záznam (spolu so záznamami 0x03C0 a 0x0285) je zmazaný a vytvorený na konci.

3.3 Načítanie súborového systému

Pozíciu základných častí súborového systému som v predchádzajúcej kapitole odhadol „od oka“. Teraz sa pokúsim nájsť, kde sú informácie o ich pozícii zapísané. Skúsenosti s FAT naznačujú, že väčšinu z nich bude treba hľadať v boot sektore. Konkrétne hodnoty, ktoré potrebujem nájsť:

Tabuľka 3.1: Významné oblasti súborového systému

sektor	dĺžka	cluster	význam
0	12		boot sektor
12	12		kópia boot sektora
128	255752 záznamov, 500 sektorov		FAT tabuľka
640	63936 záznamov, 7992 bytov, 16 sektorov	2	bitmapa
656	neznáma	4	tabuľka veľkých písmen
672	premenlivá (momentálne 1 cluster, 2 sektory)	6	záznam koreňového adresára

Táto tabuľka ukazuje, že ak začneme číslovať clustre od 2, pričom cluster 2 nasleduje hneď za FAT tabuľkou (podobne ako pri FAT32), clustre 2-6 označené FAT tabuľkou za použité budú zodpovedať pozícii bitmapy, tabuľky veľkých písmen a záznamu koreňového adresára.

Vráťme sa teraz k boot sektoru. Prvý záznam (0xEB 0x76 0x90) zrejme zodpovedá prvému záznamu súborového systému FAT. Jeho význam je už menej zrejmý, pretože na mieste, ktoré označuje (offset 0x78) sa nenachádza žiadny vykonateľný kód, ale ani dáta. Nastáva teda paradoxná situácia, keď sa BIOS pokúsi z takéhoto média bootovať (keďže obsahuje platnú signatúru), ale používateľ nedostane žiadne upozornenie, že médium nie je bootovateľné. Túto situáciu riešia až ovládače verzie 5.1.2600.3453, ktoré na offset, kam smeruje tento skok vložia príslušný kód (viac v kapitole 3.7). Nasledujúci záznam „EXFAT “ má zrejme rovnaký význam ako pri FAT, teda žiadna iná hodnota sa neodporúča a slúži len na „oklamanie nepriateľa“. Nasledujúca zaujímava hodnota je na offsete 0x48. Pri interpretácii ako bezznamienková 32 bitová hodnota ide o číslo 511488, čo je počet sektorov vyhradených pre exFAT. Na offsete 0x50 je hodnota 128, čo budem zatiaľ považovať za prvý sektor FAT tabuľky. Nasleduje hodnota 504. Ak ju budeme interpretovať ako číslo sektora, dostaneme sa kúsok pred koniec FAT tabuľky. Mohlo by

ísť aj o počet sektorov vyhradených pre FAT (ktorý môže byť väčší ako je pre FAT tabuľku potrebné). Nasleduje hodnota 640, čo je prvý sektor bitmapy. Ak sa budem držať predpokladu, že clustre sa počítajú od sektora, ktorý nasleduje za koncom FAT, potom na tomto súborovom systéme bude (511488-640)/8 clustrov, čo zodpovedá nasledujúcej hodnote 63856. Všetky oblasti od tohoto bodu sa nachádzajú v časti súborového systému, ktorej zodpovedá platné číslo clustra, preto budem predpokladať, že čísla sektorov od tohoto momentu vystriedajú čísla clustrov. Nasledujúca hodnota je 6, čo je cluster na ktorom začína záznam koreňového adresára. Preskočený bol cluster s tabuľkou veľkých písmen, jeho číslo bude treba hľadať inde. Nasleduje hodnota 0x78886DE6, čo je zjavne sériové číslo.

3.4 Štruktúra záznamu adresára

Je očakávateľné, že v adresáry budú zapísané nasledujúce informácie: dátum a čas vytvorenia, posledného prístupu a zápisu, prvý cluster, veľkosť, atribúty a checksum názvu.

Ako dátum a čas vo formáte FAT sa dajú interpretovať časti záznamu 0x0585 s relatívnymi offsetmi (od začiatku záznamu) +0x08, +0x0C a +0x10. Prvý záznam zodpovedá vytvoreniu, druhý poslednej zmene a tretí poslednému prístupu. Čas vytvorenia je spresnený dodatočným bytom s offsetom +0x14, rovnako ako pri FAT. Na rozdiel od FAT sa ale ukladá čas posledného prístupu (Windows Explorer ho nezobrazuje, príkaz dir áno). Atribútom súboru zodpovedá byte s offsetom +0x04, významy jednotlivých bitov sú rovnaké ako pri FAT32. exFAT pravdepodobne nepodporuje vlastnosti NTFS, ktoré by mohli spôsobiť potrebu ďalších atribútov (komprimované súbory, sparse súbory, reparse pointy).

Pohľadajme teraz alokačné informácie súboru ifsutil.dll, ktorého dĺžka je 123392 B. Táto hodnota sa nachádza v zázname 0x03C0 na offsetoch +0x08 a +0x18. Či je medzi nimi nejaký významový rozdiel je momentálne neznáme. Podľa oficiálnych materiálov je pre dĺžku súbora vyhradených 8 bytov, čo vysvetľuje nulové byty nasledujúce za obidvomi offsetmi. Dĺžka názvu všetkých súborov nie je uložená spolu s názvom a pri všetkých súboroch jej zodpovedá dvojbytová hodnota s offsetom +0x02. Pohľadajme teraz dáta tohoto súboru. Prvá dlhšia postupnosť bytov, pomocou ktorej sa dajú všetky tri prítomné súbory odlíšiť je 0xF5 0x3E 0x35 0xDF 0xB1 0x5F 0x5B 0x8C a nachádza sa na offsete 0x80. Táto postupnosť sa nachádza na zodpovedajúcom relatív-

nom offsete v sektore 1000, na ktorom začína cluster $(1000 - 640)/8 + 2$, čo prináša zaujímavý výsledok 47. V celom zázname sa táto hodnota na offsete +0x14 a je tu miesto na 4 byty.

V tomto konkrétnom prípade je ľahké nájsť ostatné dáta tohoto súboru. Keďže celé voľné miesto na kľúči bolo pokope v čase vytvárania súboru, jeho dátam bol priradený prvý voľný cluster a toľko ďalších clusterov, koľko bolo potrebné. Na zakódovanie tejto informácie preto treba pri rozumnej reprezentácii potenciálne málo miesta. Stačil by jeden bit, ktorý bude indikovať, že súbor je súvislý. Alebo by sa dala použiť reprezentácia z NTFS, ktorá v princípe zapisuje postupnosť intervalov clusterov. Takéto malé množstvo informácie je ľahké prehliadnuť, preto kľúč nanovo naformátujem a vytvorím tam súbory takej veľkosti a v takom poradí, aby nutne muselo dôjsť k fragmentácii. Konkrétne vytvorím súbor 1 veľkosti 100MB, potom súbor 2 veľkosti 20MB, súbor 3 veľkosti 100MB. Potom súbor 2 vymažem a novým súborom 4 zaplním zvyšok voľného miesta. Najjednoduchšia alokačná stratégia, ktorá by zabránila fragmentácii, by musela umiestniť súbor 1 na začiatok, súbor 2 za nasledujúce voľné miesto a súbor 3 na koniec voľného miesta, aby po zmazaní 2 ostalo voľné miesto pokope. Takéto správanie je nepravdepodobné, ale ak by sa vyskytlo, dá sa mu ľahko prispôsobiť. Aby som predišiel nedorozumeniam s diskovou cache, po každom kopírovaní ju zosynchronizujem. Výsledok vyzerá nasledovne:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000100   C1 00 31 00 00 00 00 00  00 00 00 00 00 00 00 00   Á.1.............
00000110   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000120   05 02 8C 25 20 00 00 00  CF 25 48 38 85 25 48 38   ..Œ% ...Ï%H8…%H8
00000130   D0 25 48 38 09 00 00 00  00 00 00 00 00 00 00 00   Ð%H8............
00000140   40 03 00 01 19 00 00 00  00 00 40 01 00 00 00 00   @.........@.....
00000150   00 00 00 00 07 64 00 00  00 00 40 01 00 00 00 00   .....d....@.....
00000160   41 00 32 00 00 00 00 00  00 00 00 00 00 00 00 00   A.2.............
00000170   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000180   85 02 E7 66 20 00 00 00  D6 25 48 38 78 25 48 38   .ç f ...Ö%H8x%H8
00000190   DA 25 48 38 7B 00 00 00  00 00 00 00 00 00 00 00   Ú%H8{...........
000001A0   C0 03 00 01 19 80 00 00  00 00 40 06 00 00 00 00   À.........@.....
000001B0   00 00 00 00 07 78 00 00  00 00 40 06 00 00 00 00   .....x....@.....
000001C0   C1 00 33 00 00 00 00 00  00 00 00 00 00 00 00 00   Á.3.............
000001D0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000001E0   85 02 99 A6 20 00 00 00  6F 26 48 38 6B 26 48 38   …™¦ ...o&H8k&H8
000001F0   71 26 48 38 06 00 00 00  00 00 00 00 00 00 00 00   q&H8............

00000000   C0 01 00 01 1A 00 00 00  00 B0 16 03 00 00 00 00   À.............
00000010   00 00 00 00 07 DC 00 00  00 B0 16 03 00 00 00 00   .....Ü.....
00000020   C1 00 34 00 00 00 00 00  00 00 00 00 00 00 00 00   Á.4.............
00000030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

Najviac nás zaujíma alokačný záznam súboru 4. Jeho identifikátor sa zmenil z 0x03C0 na 0x01C0. Podobne sa druhý byte identifikátora používa na iné účely aj pri volume labely, čo ma privádza k presvedčeniu, že identifikátor má v skutočnosti len jeden byte. Všetky záznamy zmazaného súboru 2 sú stále prítomné, ale majú najvyšší bit identifikačného bytu zrušený. Takýmto spôsobom sa zrejme indikuje zmazaný súbor, teda na samotný identifikátor ostáva len 7 bitov. Ďalšia vec, ktorú som docielil zafragmentovaním súbora je, že FAT tabuľka teraz obsahuje záznam o dvoch reťaziach clusterov: 56328 - 63858 a 25607 - 30727, pričom za clusterom 63858 nasleduje 25607 a 30727 je posledný cluster alokovaný pre tento súbor. To ma privádza k záveru, že do FAT tabuľky sa zapisujú informácie len o zafragmentovaných súboroch a skutočnosť, že je súbor súvislý, sa indikuje nastavením príslušného bitu v zázname adresára. Takýto postup má veľký význam napríklad pri ukladaní fotografií na pamäťové karty, kde je očakávateľný nasledujúci postup pou-

žitia: Používateľ na začiatku fotí na prázdnu kartu, teda všetky súbory sú nefragmentované. Následne všetky fotografie skopíruje na PC, kde ich môže upravovať alebo archivovať a z karty ich zmaže. Všetko toto sa udeje bez prístupu do FAT tabuľky, čo ušetrí veľkú časť režijných nákladov.

Ďalej z toho vyplýva, že určovať veľkosť voľného miesta z FAT tabuľky by bolo nielen neefektívne, ale aj nekorektné. Na tento účel treba použiť bitmapu. Posledná veľmi zaujímavá skutočnosť spočíva v tom, že nadbytočné FAT záznamy nebudú nikomu prekážať, pretože sa na ne nebude nikto pozerať. Preto sa pri mazaní zafragmentovaného súboru neodstraňujú jeho záznamy z FAT tabuľky. Z toho vyplýva, že po zmazaní súboru ostanú o ňom dostupné všetky informácie ako keby nebol zmazaný, čo výrazne uľahčuje jeho obnovu. Pre porovnanie pri starších verziách FAT sa záznamy z FAT tabuľky odstraňovali. To znamená, že obnovovacia utilita mala dve možnosti. Buď sa mohla spoľahnúť, že súboru bolo alokované súvislé miesto, čo na častejšie používaných partíciach dosahovalo mizivú úspešnosť. Alebo bolo možné príslušné clustery hľadať ručne, čo je činnosť, ktorá sa dá veľmi dobre opísať slovami „hľadanie ihly v kope sena“.

3.5 Kontrolný súčet názvu

„So I have an announcement to make: if you are a programmer working in 2003 and you don't know the basics of characters, character sets, encodings, and Unicode, and I catch you, I'm going to punish you by making you peel onions for 6 months in a submarine. I swear I will.“

Joel Spolsky

Zo sľubovaných nových vlastností sa mi zatiaľ nepodarilo nájsť checksum názvu súboru. Vytvorím jeden prázdny súbor, budem ho premenovávať a sledovať, čo v jeho záznamoch sa zmení.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000040   85 02 14 9F 20 00 00 00  42 2D 48 38 41 2D 48 38   ..Y ...B-H8A-H8
00000050   42 2D 48 38 6A 00 00 00  00 00 00 00 00 00 00 00   B-H8j...........
00000060   C0 01 00 01 20 80 00 00  00 00 00 00 00 00 00 00   A... ...........
00000070   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000080   C1 00 61 00 00 00 00 00  00 00 00 00 00 00 00 00   Á.a.............
00000090   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000040   85 02 D2 7F 20 00 00 00  42 2D 48 38 41 2D 48 38   .Ò. ...B-H8A-H8
00000050   8D 2D 48 38 6A 00 00 00  00 00 00 00 00 00 00 00   .-H8j...........
00000060   C0 01 00 01 21 00 00 00  00 00 00 00 00 00 00 00   À...!...........
00000070   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000080   C1 00 62 00 00 00 00 00  00 00 00 00 00 00 00 00   Á.b.............
00000090   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

Ak si odmyslíme dátum poslednej zmeny, ostanú dvaja kandidáti na checksum: 2 byty na offsete +0x02 v zázname 0x85 alebo 2 byty s offsetom 0x04 v zázname 0xC0. Skúsim do súboru zapísať jeden byte:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000040   85 02 87 85 20 00 00 00  42 2D 48 38 E9 2D 48 38   . ...B-H8é-H8
00000050   E9 2D 48 38 6A 00 00 00  00 00 00 00 00 00 00 00   é-H8j...........
00000060   C0 03 00 01 21 00 00 00  01 00 00 00 00 00 00 00   À...!...........
00000070   00 00 00 00 07 00 00 00  01 00 00 00 00 00 00 00   ................
00000080   C1 00 62 00 00 00 00 00  00 00 00 00 00 00 00 00   Á.b.............
00000090   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

V tomto okamihu je užitočné postrehnúť, že prázdny súbor nie je považovaný za súvislý. Súvislým sa môže stať až v momente, keď sú do neho zapísané dáta. Prázdny súbor má ako prvý cluster uvedený 0, čo zodpovedá záznamu vo FAT tabuľke s deskriptorom média, ktorý po interpretácii ako záznam FAT tabuľky znamená koniec súboru.

Kandidát zo záznamu 0x85 sa zmenil aj bez zmeny názvu, preto checksum bude zrejme na offsete +0x04 v zázname 0xC0. Teraz budem postupne vytvárať súbory s rôznymi názvami a kontrolovať, ako to ovplyvňuje kontrolný súčet. Podľa toho budem aktualizovať svoju predstavu o algoritme na jeho výpočet.

Najprv sa pozrime na najjednoduchší prípad, keď má názov súboru jeden znak (keďže písmená v názve sa menia pred výpočtom kontrolného súčtu na veľké, budem sa zaoberať iba názvami, ktoré pozostávajú len z veľkých písmen).

Checksum v obidvoch prípadoch obsahuje tie isté bity ako názov posunuté o jednu pozíciu doprava. Z druhého prípadu vyplýva, že ide o rotáciu a nie posun. Teraz vyskúšam na viacpísmenovom názve čo sa deje v hashovacej funkcii medzi jednotlivými kolami (v každom kole sa spracuje jeden znak).

Tabuľka 3.2: Kontrolné súčty jednopísmenových názvov

	Hodnota	Binárny zápis
Názov	B	01000010
Checksum	0x0021	00000000 00100001
Názov	C	01000011
Checksum	0x8021	10000000 00100001

Názov si vyberiem tak, aby sa bity výsledku dali ľahko priradiť písmenu zo vstupu, z ktorého pochádzajú. Konkrétne za prvý znak zvolím 0 (nula), pretože v binárnom zápise obsahuje dve po sebe idúce nastavené bity a znak @, ktorý má nastavený len jeden bit.

Tabuľka 3.3: Kontrolný súčet dvojpísmenového názvu

	Hodnota	Binárny zápis
Názov	0@	00110000 01000000
Checksum	0x0026	00000000 00100110

Výsledok obsahuje tri bity so správnymi odstupmi, takže sa mi pravdepodobne podarilo zvoliť taký vstup, pri ktorom sa nastavené bity nedostali v žiadnom medzikroku na rovnakú pozíciu (teda operácia ♣, ktorá spája výsledky dvoch po sebe idúcich krokov operovala vždy na 0♣0, 0♣1 alebo 1♣0, nikdy nie na dvoch jednotkách). Binárny zápis druhého znaku je vo výsledku posunutý o 1 pozíciu doprava, zápis prvého znaku o 3 pozície. Z toho by mohol vyplývať nasledujúci algoritmus:

Algoritmus 1

```
uint checksum(char *name) {
  uint result = 0;
  for (int i=0; name[i]; i++)
    result = ror(result,2) ♣ ror(name[i],1);
  return result;
}
```

Pozrime sa teraz na operátor ♣. Zo známych binárnych operátorov by

mohlo ísť o binárne OR (ak by platilo 1♣1 = 1), alebo XOR (ak by 1♣1 = 0). Za názov súboru si tentokrát zvolím !H, pretože po troch rotáciach znaku ! a jednej rotácii H sa v menej významnom byte checksumu stretnú v druhom kole dva nastavené bity, pričom ďalší nastavený bit bude pomerne ďaleko:

Tabuľka 3.4: Operátor ♣

Znak	Menej významný byte
! po troch rotáciach	00000100
H po jednej rotácii	00100100
♣ predchádzajúcich hodnôt	00101000

Z toho vyplýva, že síce 1♣1 = 0, ale došlo k nastaveniu susedného bitu, teda ♣ neoperuje nezávisle na každej dvojici bitov a zatiaľ sa správa rovnako ako aritmetické sčitovanie. Pri sčitovaní môže dôjsť k pretečeniu, preto nasledujúca prirodzená otázka je, čo sa stane ak pretečenie nastane na najvýznamnejšom bite. Názov súboru teda zvolím tak, aby sa po príslušných rotáciach stretli v druhom kole dva nastavené najvýznamnejšie bity. Vhodný kandidát je napríklad DA:

Tabuľka 3.5: Kontrolný súčet pri pretečení najvýznamnejšieho bitu

Znak	Binárny zápis
D po troch rotáciach	10000000 00001000
A po jednej rotácii	10000000 00100000
súčet	00000000 00101001

Pretečený bit sa teda prejavil na najmenej významnej pozícii. Tradičná otázka teda je, akou operáciou sem bol pridaný?

Tabuľka 3.6: Operácia pre pretečený bit

Znak	Binárny zápis
D po troch rotáciach	10000000 00001000
C po jednej rotácii	10000000 00100001
súčet	00000000 00101010

Aj v tomto prípade je odpoveďou aritmetické sčítanie. Otázka pretečenia (cez najvýznamnejší bit) sa pri tomto sčítaní vyrieši rýchlo, keďže najväčší výsledok po sčítaní s pretečením je 0xFFFE, ku ktorému sa dá pripočítať jednotka. Dostávame sa teda k nasledujúcemu algoritmu:

Algoritmus 2

```
uint ror(uint number, int count) {
  if (count>1)
    number = ror(number, count-1);
  return (number>>1) | ((number&1)<<15);
}

uint checksum(unsigned char *name) {
  uint result = 0;
  for (int i=0; name[i]; i++) {
    result = ror(result, 2) + ror(name[i], 1);
    if (result != (result & 0xFFFF))
      result = (result & 0xFFFF) + 1;
  }
  return result;
}
```

Ako kontrolu správnosti skúsim vypočítať kontrolné súčty niektorých súborov, s ktorými som doteraz robil:

Tabuľka 3.7: Kontrolné súčty dlhších názvov

Názov	Vypočítaný checksum
FMIFS.DLL PREMENOVANY ABY VZNIKLA FRAGMENTACIA	0x29CE
EXFAT.SYS S NAZVOM PREDLZENYM NA STUDIJNE UCELY	0xCD14
IFSUTIL.DLL	0xB6D0

Vypočítané hodnoty súhlasia s tým, čo možno vidieť na príslušnom výpise.

Doteraz sa mi úspešne darilo ignorovať jednu záležitosť, ktorá spočíva v tom, že názvy súborov sú v skutočnosti zložené zo znakov Basic Multilingual Plane a reprezentované dvomi bytmi. Predchádzajúci algoritmus je teda len

špeciálny prípad, ktorý funguje iba pre znaky, ktoré majú významnejší byte nulový. Poďme to teraz napraviť:

Tabuľka 3.8: Kontrolný súčet z názvu s rozšírenými znakmi

Názov	Checksum
ĽŠ	11100000 00111000
=‘	10100000 00110111

Názov v druhom riadku vznikol z názvu v prvom riadku vynulovaním významnejších bytov. Tieto byty mali pôvodne hodnotu 1. Na kontrolnom súčte sa to prejavilo takým spôsobom, že keby sme k nemu pripočítali jednotkové byty zrotované o 2 a 0, dostali by sme prvý checksum. To ma privádza k predpokladu, že v každom kole sa pripočítava menej významný byte zrotovaný o 1 a viac významný byte bez rotácie. Ak by sme v jednom kole spracovávali jeden byte namiesto jedného znaku, dal by sa tento algoritmus po miernych úpravách formulovať nasledovne:

Algoritmus 3

```
uint ror(uint number) {
  return (number>>1) | ((number&1)<<15);
}

uint checksum(unsigned char *name, int size) {
  uint result = 0;
  for (int i=0; i<size; i++) {
    result = ror(result) + name[i];
    if (result != (result & 0xFFFF))
      result = (result & 0xFFFF) + 1;
  }
  return result;
}
```

Spracovávanie jedného bytu na kolo (namiesto doterajšieho jedného znaku) ale spôsobí jeden nečakaný problém: zmení sa poradie rotácií a sčitovaní. Vidno to napríklad na checksume názvu DA. Algoritmus 2 v konečnom dôsledku sčíta 3-krát zrotované písmeno D a raz zrotované písmeno A. Algo-

ritmus 3 najprv 2-krát zrotuje D, potom k nemu pripočíta A a výsledok raz zrotuje. Kým pri predchádzajúcom algoritme došlo k pretečeniu na najvyššom bite, teraz dôjde k pretečeniu inde.

Tabuľka 3.9: Zmena poradia operácii spôsobí, že k pretečeniu nedôjde

Algoritmus 2		Algoritmus 3	
ror(D, 3)	10001000	ror(D, 2)	00010001
ror(A, 1)	10100000	A	01000001
súčet	100101000	súčet	01010010
pretečený bit	00101001	rotácia súčtu	00101001

Z toho vyplýva, že príklad DA neposkytuje dostatočnú odpoveď na otázku čo sa stane s bitom, ktorý pretečie počas sčitovania. Ako vhodný príklad sa ukazuje byť názov CCCBBB:

Tabuľka 3.10: Pretečenie pri spracúvaní vstupu po bytoch

	Pripočítaj pretečený bit	Zahoď pretečený bit
zrotovaná medzihodnota	11111111 11010101	11111111 11010101
spracovávaný znak (B)	00000000 01000010	00000000 01000010
súčet	1 00000000 00010111	1 00000000 00010111
spracovaný pretečený bit	00000000 00011000	00000000 00010111
zrotovaná medzihodnota	00000000 00001100	10000000 00001011
spracovávaný znak (\0)	00000000 00000000	00000000 00000000
súčet	00000000 00001100	10000000 00001011

Správna hodnota checksumu pre tento názov je 0x800B, čo dostanem zahodením pretečeného bitu. Výsledný algoritmus vyzerá nasledovne:

Algoritmus 4

```
uint ror(uint number) {
  return (number>>1) | ((number&1)<<15);
}

uint checksum(unsigned char *name, int size) {
  uint result = 0;
  for (int i=0; i<size; i++)
    result = (ror(result) + name[i]) & 0xFFFF;
  return result;
}
```

Jeho správnosť otestujem na zložitejších vstupoch (písmená v druhom prípade sú po poradí alfa, beta a gama):

Tabuľka 3.11: Kontrolné súčty zložitejších názvov

Názov	Vypočítaný checksum
STUDNICE ŽIAĽU, JAZVY KĽOVÚC BÔĽU;	0x825D
$\mathrm{A} + \mathrm{B} = \Gamma$	0x7A36

Vypočítané hodnoty sa zhodujú s prečítanými.

3.6 Informácia o časovom pásme

Operačné systémy Windows sú známe tým, že pri zobrazovaní časových pečiatok asociovaných so súbormi závisí výsledok od niektorých hodnôt, od ktorých by závisieť nemal, napríklad či aktuálny čas je letný alebo zimný [Don]. Keďže ide len o záležitosť zobrazovania a nie reprezentácie na disku, budem sa týmto problémom chcieť vyhnúť tým, že sa budem zaoberať len časovými pečiatkami, ktoré boli vytvorené v letnom čase (v čase písania tejto časti platil letný čas).

Táto vlastnosť pribudla až po vydaní ovládačov 6.0.6001.16633, teda jej umiestnenie na disku ľahko lokalizujem do atribútového slotu príslušného záznamu súboru (offsety 0x16 - 0x18) jednoduchým porovnaním zmien:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000060   85 02 1F 90 20 00 00 00  0D 59 8D 3A A9 92 B2 38   .... ....Y?:.‰28
00000070   0D 59 8D 3A AF 00 88 88  88 00 00 00 00 00 00 00   .Y?:..^^^.......
```

V tomto okamihu stojí za zmienku, že tento súbor bol vytvorený 13.4.2009 11:08:26 lokálneho času a presne tento čas je uložený. Teda oficiálny popis tejto vlastnosti, ktorý znie „Support for Universal Coordinated Time (UTC) time stamps“ [Cor] nie je korektný, pretože časové pečiatky sa stále ukladajú v lokálnom čase a na UTC čas je ich možné len prepočítať.

Skúsim teraz vytvárať súbory s rôznymi nastavenými časovými pásmami a sledovať hodnoty týchto troch bytov:

Tabuľka 3.12: Kódovanie časových zón

Rozdiel lokálneho času a UTC (hodiny)	Prečítaná hodnota
0	0x80
+1	0x84
+2	0x88
+13	0xB4
-1	0xFC
-12	0xD0

Vo všetkých prípadoch boli hodnoty týchto troch bytov rovnaké. To vedie k podozreniu, že po poradí zodpovedajú ukladaným pečiatkam. Skúsim teda predchádzajúci súbor modifikovať so zmenenou zónou:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000060   85 02 32 2E 20 00 00 00  0D 59 8D 3A 86 9D 92 3A   ..2. ....Y?:.?‰:
00000070   86 9D 92 3A AF 00 88 9C  9C 00 00 00 00 00 00 00   .?‰:..^oo.......
```

Spolu s poslednými dvomi pečiatkami sa zmenili aj príslušné zápisy zón, čo toto podozrenie potvrdzuje.

Predchádzajúca tabuľka naznačuje, že kladný posun oproti UTC sa vyjadruje ako štvornásobok počtu hodín pripočítaný k 0x80, záporný posun sa rovnako odpočítava od 0x0100. Násobenie štyrmi by sa dalo vysvetliť tým, že niektoré zóny sú posunuté od UTC o štvrťhodiny a zároveň každý posun sa dá vyjadriť ako nejaký násobok štvrťhodín. Overím:

Zápis týchto zón sedí s predchádzajúcim pozorovaním.

Tabuľka 3.13: Kódovanie časových zón posunutých o štvrťhodiny

Rozdiel lokálneho času a UTC (hodiny)	Prečítaná hodnota
+4:30	0x92
+5:30	0x96
+5:45	0x97

3.7 Bootovací kód

Ovládače verzie 5.1.2600.3453 pridávajú do boot sektora nasledujúce položky:

Tabuľka 3.14: Nové položky v boot sektore

Názov	Offset	Dĺžka	Význam
Bootstrap	0x78	42	Kód, ktorý sa vykoná pri bootovaní z tohoto média
StringTable	0x100	73	Tabuľka reťazcov, ktorý bootovací kód môže vypísať.
StringTableIndex	0x1FB	3	Indexy do tabuľky reťazcov, na ktorých začínajú jednotlivé reťazce.

Reťazce v StringTable obsahujú len 7-bitové hodnoty, oddelené sú hodnotou 0xFF a zakončené hodnotou 0x00. Indexy ukazujú na začiatok reťazcov relatívne od začiatku tabuľky. Tieto informácie vyplývajú z reverzného inžinierstva bootovacieho kódu:

```
Offset     Hex                 Opcode      Operandy
; nastav segmentové registre a zásobník
 00000078: 33C9                xor         cx,cx
 0000007A: 8ED1                mov         ss,cx
 0000007C: BCF07B              mov         sp,07BF0
 0000007F: 8ED9                mov         ds,cx
; ulož do ds:si offset prvého reťazca
 00000081: A0FB7D              mov         al,[07DFB]
 00000084: B47D                mov         ah,07D
 00000086: 8BF0                mov         si,ax
; začiatok cyklu na výpis aktuálneho reťazca
; načítaj nasledujúci znak
 00000088: AC                  lodsb
; ak som načítal 0xFF, prejdi na ďalší reťazec
 00000089: 98                  cbw
 0000008A: 40                  inc         ax
 0000008B: 740C                je          000000099
```

```
; ak som načítal 0x00, skonči výpis
 0000008D: 48                dec         ax
 0000008E: 740E              je          00000009E
; ináč vypíš aktuálny znak
 00000090: B40E              mov         ah,00E
 00000092: BB0700            mov         bx,00007
 00000095: CD10              int         010
; opakuj cyklus
 00000097: EBEF              jmps        000000088
; vyber tretí reťazec
 00000099: A0FD7D            mov         al,[07DFD]
 0000009C: EBE6              jmps        000000084
; počkaj na stlačenie klávesy
 0000009E: CD16              int         016
; pokračuj v bootovaní z iných zariadení
 000000A0: CD19              int         019
```

Kapitola 4

Dokumentácia exFAT

„Anyone can do any amount of work provided it isn't the work he is supposed to be doing at the moment."
Robert Benchley

Táto kapitola zhŕňa poznatky z predchádzajúcej kapitoly formou technickej dokumentácie. Dopĺňané sú informáciami získanými štúdiom zdrojového kódu [Hir], ktoré mi poskytol pán Ogawa Hirofumi pod licenciou GPL v2.

Súborový systém exFAT pozostáva z nasledujúcich oblastí:

1. oblasť boot sektora (2 kópie)
2. FAT tabuľka
3. bitmapa
4. tabuľka veľkých písmen
5. záznam koreňového adresára

Oblasti od bitmapy ďalej sa nachádzajú v dátovej oblasti a vždy začínajú na hranici clustera, takže sa na ne dá pozerať ako na špeciálne súbory, ktoré nemajú záznam v koreňovom adresáry.

4.1 Oblasť boot sektora

Začína vždy na začiatku priestoru, ktorý je vyhradený pre tento súborový systém a má dĺžku 12 sektorov. Za ňou nasleduje identická kópia, ktorá sa

líši iba príslušným príznakom. Prvý sektor (boot sektor) má nasledujúcu štruktúru:

Tabuľka 4.1: Štruktúra boot sektora

Názov	Offset	Dĺžka	Význam
Magic	0	3	Skok, ktorý sa vykoná pri bootovaní.
OEMName	3	8	Reťazec „EXFAT “ (počet medzier na konci je 3).
FirstSec	0x40	8	Počet sektorov na médiu pred začiatkom tohoto súborového systému.
TotalSec	0x48	8	Počet sektorov vyhradených pre tento súborový systém.
FATStart	0x50	4	Začiatok FAT tabuľky.
FATRecs	0x54	4	Kapacita FAT tabuľky (udáva počet záznamov).
ClusStart	0x58	4	Začiatok dátovej oblasti (sektor, na ktorom začína cluster s poradovým číslom 2).
ClusCnt	0x5C	4	Celkový počet clusterov.
RootStart	0x60	4	Cluster, na ktorom začína záznam koreňového adresára.
VolID	0x64	4	Volume serial number, rovnaký význam ako pri FAT.
Unknown1	0x68	2	Neznámy záznam, jediná pozorovaná hodnota je 0x0100.
State	0x6A	2	Vyjadruje predpokladanú konzistentnosť súborového systému. Tento záznam je vynechaný z počítania kontrolného súčtu. Známe hodnoty: • 0 - súborový systém bol čisto odpojený • 2 - súborový systém bol odpojený počas prevádzky
SecSizeBits	0x6C	1	Pozícia prvej jednotky v bitovom zápise veľkosti sektora. Skutočná hodnota je teda $2^{SecSizeBits}$.
ClusSize	0x6D	1	Veľkosť clustera v sektoroch znížená o 1. Veľkosť clustera v bytoch je $2^{SecSizeBits+ClusSize}$.
Unknown2	0x6E	2	Neznámy záznam, jediná pozorovaná hodnota je 0x8001.

Tabuľka 4.1: Štruktúra boot sektora (pokr.)

Názov	Offset	Dĺžka	Význam
Unknown3	0x70	1	Neznámy záznam, vynechaný z počítania kontrolného súčtu.
Signature	0x1FE	2	Hodnota 0xAA55.

Pomocou položky Signature je podpísaných prvých 8 sektorov. Sektor 11 obsahuje kontrolný súčet z dát, ktoré vzniknú spojením obsahu sektorov 0 až 10 a vynechaním offsetov 0x6A, 0x6B a 0x70. Tento kontrolný súčet je zapísany opakovane tak, aby vyplnil celý obsah sektora. Počítaný je 32-bitovou obdobou Algoritmu 4:

Algoritmus 5

```
uint checksum(unsigned char *buffer, int size) {
  uint result = 0;
  for (int i=0; i<size; i++)
    result = ((result>>1) | (result<<31)) + buffer[i];
  return result;
}
```

4.2 FAT tabuľka, bitmapa a tabuľka veľkých písmen

Pozícia FAT tabuľky je udaná v boot sektore. Jej význam je podobný ako pri FAT32 až na dva rozdiely:

- Pri zmazaní súboru sa jeho záznam vo FAT tabuľke nemení.
- Súbory, ktoré sú v zázname adresára označené ako spojité, nemajú žiaden záznam vo FAT tabuľke. Ich pozícia na disku jednoznačne vyplýva z prvého clustera a dĺžky.

Bitmapa, tabuľka veľkých písmen a záznam koreňového adresára sa nachádzajú v dátovej oblasti, preto majú záznam vo FAT tabuľke (aj keď sú spojité).

Počiatočný cluster aj veľkosť bitmapy sú uložené v zázname koreňového adresára. Pre každý cluster dátovej oblasti (teda počínajúc clusterom 2) obsahuje jeden bit, ktorý je nastavený práve vtedy, keď je príslušný cluster alokovaný. Nepoužité bity na konci sú nulové.

Poloha a veľkosť tabuľky veľkých písmen sú určené príslušným záznamom v koreňovom adresáry. Ide o postupnosť 2-bytových hodnôt, ktoré predstavujú zaradom veľké verzie písmen z Basic Multilingual Plane. Špeciálna hodnota 0xFFFF znamená vynechaný rozsah. Nasledovaná je počtom vynechaných písmen. Za veľkú verziu každého z nich sa považuje ono samotné. Z toho vyplýva, že v tomto súbore je na pozícii $Offset$ zapísana veľká verzia písmena $Offset/2 - 2 * NumIntervals + SkippedLetters$ (ak táto pozícia nepopisuje interval), kde $NumIntervals$ je počet vynechaných intervalov po túto pozíciu a $SkippedLetters$ je počet vynechaných písmen. Konverzia názvov súborov na veľké písmená sa používa pri počítaní kontrolného súčtu názvu a pri porovnávaní dvoch názvov.

4.3 Štruktúra pre záznam adresára

Poradové číslo clustera, na ktorom začína záznam koreňového adresára, je zapísané v boot sektore. Jeho dĺžka je určená dĺžkou príslušnej reťaze z FAT tabuľky. Informácie o uložení záznamov ostatných adresárov sa nachádzajú v ich nadradených adresároch. Záznamy o špeciálnych adresároch s názvami . a .. sa nepoužívajú. Záznam každého adresára pozostáva z niekoľkých slotov veľkosti 32 bytov zapísaných za sebou. Najvýznamnejší bit prvého bytu indikuje platnosť slotu, zvyšné bity určujú typ. Podľa typu sa sloty delia na špeciálne a obyčajné. Špeciálne sloty sa vyskytujú každý práve raz v zázname koreňového adresára a obsahujú informácie o názve partície („volume label“), bitmape a tabuľke veľkých písmen.

Tabuľka 4.2: Štruktúra slotu pre volume label

Názov	Offset	Dĺžka	Význam
SlotID	0	1	Hodnota pre rozlíšenie typu slotu. V tomto prípade 0x83. Ak volume label nebol nastavený, tento slot je prítomný so zrušeným najvýznamnejším bitom.
NameLen	1	1	Dĺžka reťazca volume label v znakoch.
Name	2	30	Volume label.

Tabuľka 4.3: Štruktúra slotu pre bitmapu

Názov	Offset	Dĺžka	Význam
SlotID	0	1	Hodnota pre rozlíšenie typu slotu. V tomto prípade 0x81.
FirstClus	0x14	4	Začiatočný cluster bitmapy.
Size	0x18	8	Veľkosť bitmapy v bytoch.

Tabuľka 4.4: Štruktúra slotu pre tabuľku veľkých písmen

Názov	Offset	Dĺžka	Význam
SlotID	0	1	Hodnota pre rozlíšenie typu slotu. V tomto prípade 0x82.
Checksum32	4	4	Kontrolný súčet celej tabuľky počítaný Algoritmom 5.
FirstClus	0x14	4	Prvý cluster tabuľky.

Zvyšné typy slotov sa používajú na vytvorenie záznamov súborov. Ten vždy začína slotom pre atribúty súboru a patrí mu toľko bezprostredne nasledujúcich slotov, koľko špecifikuje hodnota SlotCount.

Tabuľka 4.5: Štruktúra slotu pre atribúty súboru

Názov	Offset	Dĺžka	Význam
SlotID	0	1	Hodnota pre rozlíšenie typu slotu. V tomto prípade 0x85.
SlotCount	1	1	Počet nasledujúcich slotov, ktoré tvoria záznam tohoto súboru.
Attr	4	1	Bitové pole atribútov súboru. Jeho význam je rovnaký ako pri FAT.

Tabuľka 4.5: Štruktúra slotu pre atribúty súboru (pokr.)

Názov	Offset	Dĺžka	Význam
CreateTime	8	4	Čas a dátum vytvorenia súboru. Formát zápisu času a dátumu je rovnaký ako pri FAT.
LastChangeTime	0x0C	4	Čas a dátum poslednej zmeny súboru.
LastAccessTime	0x10	4	Čas a dátum posledného prístupu k súboru.
CreateMillis	0x14	1	Počet milisekúnd o ktorý je čas vytvorenia súboru väčší oproti CreateTime.
LastChangeMillis	0x15	1	Počet milisekúnd o ktorý je čas poslednej zmeny súboru väčší oproti LastChangeTime.
CreateTZ	0x16	1	Časový posun času operačného systému oproti UTC v momente vytvorenia súboru.
LastChangeTZ	0x17	1	Časový posun času operačného systému oproti UTC v momente poslednej zmeny súboru.
LastAccessTZ	0x18	1	Časový posun času operačného systému oproti UTC v momente posledného prístupu k súboru.

Všetky časové pečiatky sa ukladajú ako kópia hodín operačného systému, teda s ohľadom na nastavenú časovú zónu. Tá sa ukladá do príslušnej položky so suffixom TZ s nasledovným kódovaním:

- TZ = 0 - informácia o časovej zóne nie je uložená
- 0x80 $\leq$ TZ $\leq$ 0xB4 - zaznamenaný čas je oproti UTC posunutý o TZ - 0x80 štvrťhodín dopredu
- 0xD0 $\leq$ TZ $\leq$ 0xFF - zaznamenaný čas je oproti UTC posunutý o 0x0100 - TZ štvrťhodín dozadu

Tabuľka 4.6: Štruktúra slotu pre informácie o umiestnení dát súboru

Názov	Offset	Dĺžka	Význam
SlotID	0	1	Hodnota pre rozlíšenie typu slotu. V tomto prípade 0xC0.

Tabuľka 4.6: Štruktúra slotu pre informácie o umiestnení dát súboru (pokr.)

Názov	Offset	Dĺžka	Význam
Cont	1	1	Hodnota 0x03 ak sú dáta tohoto súboru uložené v po sebe idúcich clusteroch a súbor nemá záznam vo FAT tabuľke. Hodnota 0x01 v opačnom prípade.
NameLength	3	1	Dĺžka názvu súboru v znakoch.
NameHash	4	2	Hash z názvu súbora skonvertovaného na veľké písmena vypočítaný Algoritmom 4.
FileSize	8	8	Veľkosť súboru v bytoch.
DataStart	0x14	4	Prvý cluster, ktorý je alokovaný pre dáta tohoto súboru. Ak nie sú alokované žiadne clustre, hodnota 0.
FileSize2	0x18	8	Veľkosť súboru v bytoch. Musí platiť FileSize = FileSize2.

Tabuľka 4.7: Štruktúra slotu pre názov súboru

Názov	Offset	Dĺžka	Význam
SlotID	0	1	Hodnota pre rozlíšenie typu slotu. V tomto prípade 0xC1.
Name	2	30	Názov súboru. Ak je dlhší ako 15 znakov, zvyšné časti názvu sú uložené po poradí v nasledujúcich slotoch.

Všetky názvy (súborov a partície) sa zapisujú ako postupnosť 16-bitových code pointov. Rozdiel oproti kódovaniu UTF-16LE spočíva v tom, že code pointy pozostávajúce z viacerých code units nie sú podporované a sú interpretované ako samostatné znaky.

Kapitola 5

Implementácia

„Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.“

Donald Knuth

Ovládač pre exFAT som implementoval ako modul do linuxového jadra verzie 2.6.25.13 (aktuálna verzia v čase začiatku implementácie). Pri jeho písaní som sa snažil volať existujúci spoločný kód pre všetky verzie FAT všade, kde som potreboval pracovať s dostatočne podobným konceptom. Keďže v minimálnej miere potrebujem modifikovať existujúce súbory, distribuujem ho obvyklým spôsobom, teda ako patch. Očakávam, že tento patch bude použiteľný bez zmeny aj s budúcimi verziami jadra až kým nedôjde k väčšej zmene modulu FAT alebo subsystému VFS.

Štruktúra tohoto modulu je z väčšej miery pevne diktovaná nutnosťou jeho komunikácie s VFS [GE]. Preto túto problematiku spomeniem iba letmo a venovať sa budem iba problémom, ktorých riešenie nie je priamočiare.

Pri písaní kódu sa držím tézy obľúbenej medzi vývojármi pre linuxový kernel, podľa ktorej sa dobre napísaný kód dá čítať ľahšie ako komentáre a namiesto písania komentárov je lepšie premyslieť si, ako by sa dal príslušný kód prepísať tak, aby z neho bolo zrejmé to, čo som chcel pôvodne napísať do komentáru [sty]. Preto píšem komentáre iba k rozhraniam funkcií a na miesta, kde by nebolo dobré vyhnúť sa nejakému implementačnému triku.

Pri načítaní modulu do jadra sa volá funkcia špecifikovaná makrom *module_init*, ktorá zaregistruje nový súborový systém volaním funkcie *regis-*

ter_filesystem(), ktorej ako parameter predá *struct file_system_type*. Pri požiadavke na pripojenie súborového systému sa z nej zavolá členská funkcia *get_sb*, ktorá v mojom prípade ukazuje na funkciu *exfat_get_sb()*. V tejto funkcii najprv skontrolujem kontrolný súčet oblasti boot sektora. Potom vypočítam veľkosť clustera a pozície FAT tabuľky, začiatku dátovej oblasti a koreňového adresára. Ďalej prechádzam záznam koreňového adresára a hľadám záznam o bitmape a tabuľke veľkých písmen. Nasleduje načítanie tabuľky veľkých písmen. Aby som maximalizoval rýchlosť konverzie názvov na veľké písmená, rozbalím celú tabuľku do poľa veľkosti 128kB. Iná možnosť by bola samostatne si pamätať písmená, ktoré majú veľkú verziu a samostatne začiatky a konce vynechaných intervalov. Binárnym vyhľadávaním na tomto druhom poli by som potom vedel zistiť, či konvertované písmeno patrí do nejakého intervalu (v takom prípade ho vrátim ako výsledok) alebo mám jeho veľkú verziu uloženú (v takom prípade viem z prefixových súčtov dĺžok intervalov určiť index do poľa veľkých písmen, kde sa nachádza). Túto verziu som zavrhol pre zložitejšiu implementáciu, ktorá je náchylnejšia na chyby a pomalšiu konverziu.

Ak všetko prebehlo bez chyby, vytvorím a vrátim struct inode, ktorá reprezentuje koreňový adresár. Následná komunikácia o tomto súborovom systéme prebieha volaním funkcií, ktoré sú asociované s touto inode pomocou *struct inode_operations* a *struct file_operations*.

Inode pre súbor sa od inode pre adresár líši iba implementovanými *inode_operations* a *file_operations*. Pre adresáre ide o operácie ako vymenovanie obsahu adresára, hľadanie konkrétneho súbora, pridanie záznamu pre nový súbor, zmazanie záznamu, vytvorenie prázdneho adresára, presun záznamu súboru. Pre súbory to sú hlavne operácie čítania a zapisovania dát, presun aktuálnej pozície, skrátenie a predĺženie alokovaného priestoru. Spoločné operácie súvisia s čítaním a menením atribútov.

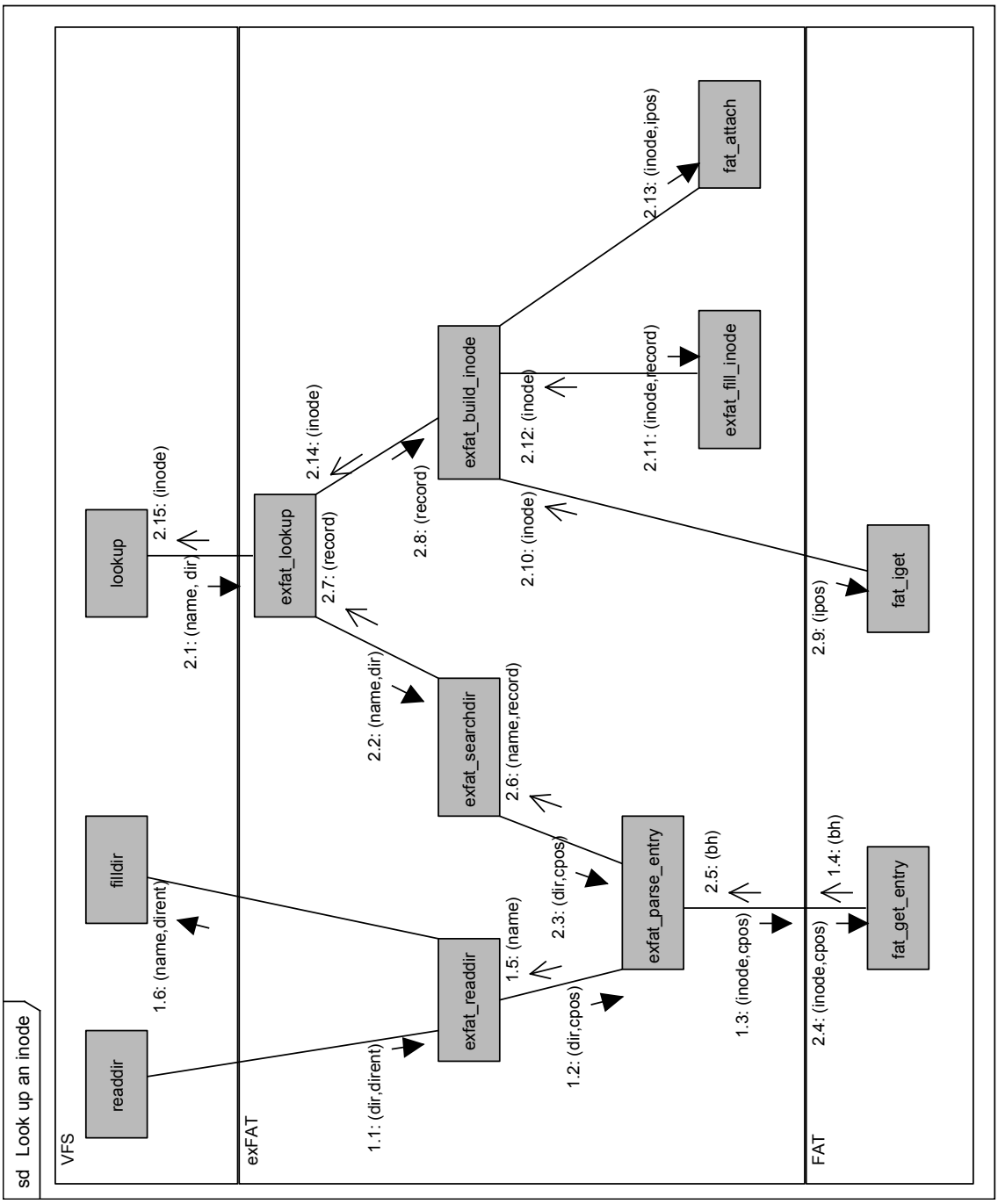
Operácie, ktoré potrebujú prechádzať po záznamoch súborov v jednom adresáry som implementoval pomocou funkcie *exfat_parse_entry()*. Tá v parametroch dostane pozíciu v zázname nejakého adresára a vráti meno a sloty patriace nasledujúcemu súboru (ak existuje). Volá pri tom funkciu *fat_get_entry()*, ktorá pozíciu v adresáry posunie vždy na nasledujúcu 32 bytovú štruktúru (veľkosť slotu je rovnaká pri FAT aj exFAT), pričom podľa potreby prečíta nasledujúci sektor alebo prednačíta do vyrovnávacej pamäte nasledujúce sektory z rovnakého clustera.

Operácie, ktoré čítajú obsah súboru (načítanie tabuľky veľkých písmen, spočítanie voľného miesta z bitmapy) implementujem pomocou funkcie *exfat_*

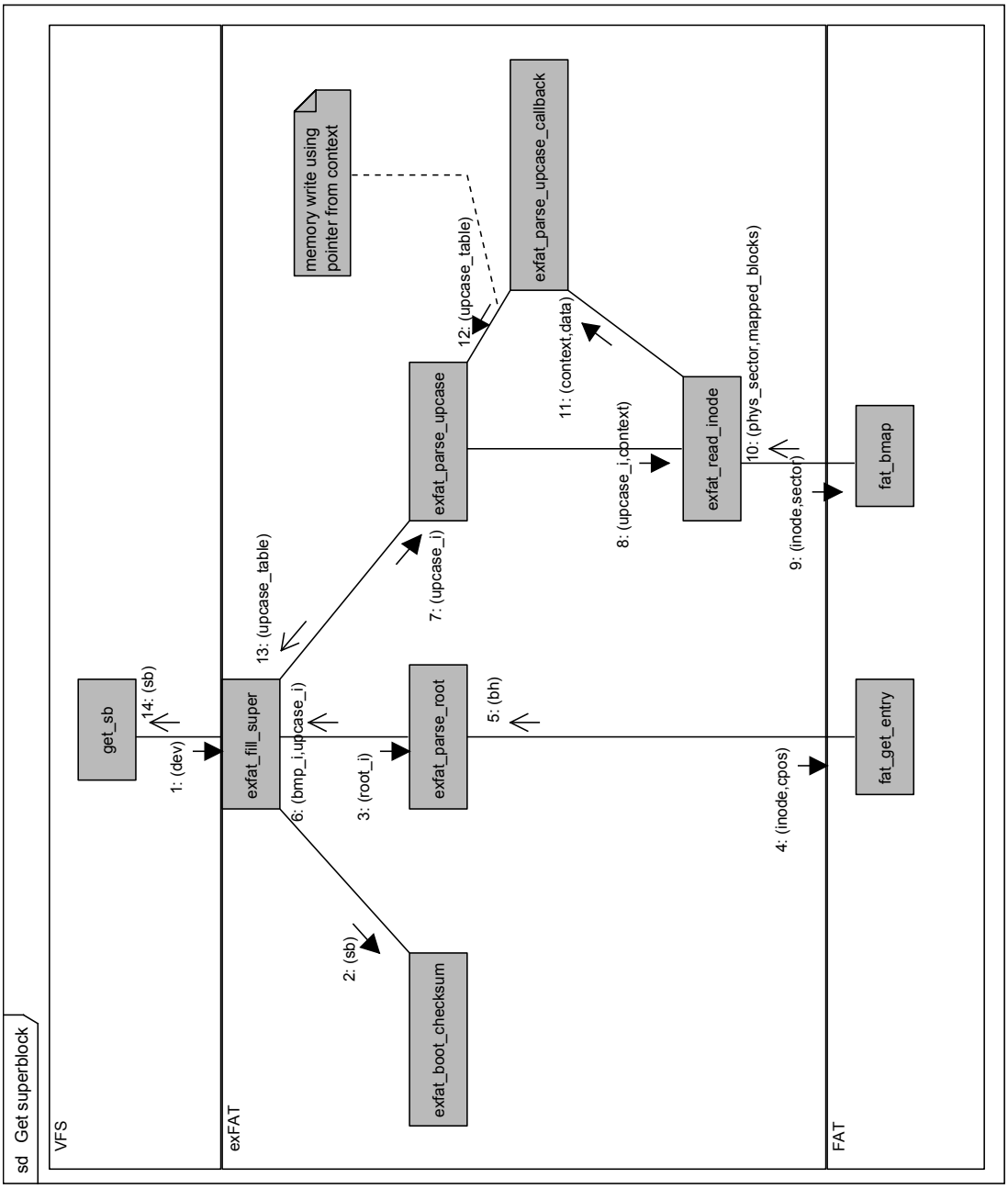
`read_inode()`. Ako parameter dostane inode z ktorej bude čítať, callback funkciu ktorú zavolá pre každý načítaný blok a ukazovateľ ktorý predá callback funkcii. Ten slúži ako kontext, teda pamäť, kde si callback funkcia uchováva obsah lokálnych premenných medzi volaniami. Vytvára ho funkcia, ktorá volá `exfat_read_inode()`, na svojom zásobníku.

Používam dva spôsoby ako ukladať pozíciu v zázname adresára, rozlišujem ich podľa názvu. Hodnota cpos udáva offset v dátovom priestore príslušného adresára, teda význam má len s adresárom, ktorého sa týka. Používam ju ako parameter funkcie `fat_bmap()`. Hodnota ipos v najnižších 4 bitoch udáva poradové číslo záznamu v rámci sektora a vo zvyšných bitoch číslo sektora, v ktorom sa tento záznam nachádza. Táto hodnota jednoznačne identifikuje každý súbor, preto sa používa pri cachovaní na posudzovanie, či sú dve inode rovnaké.

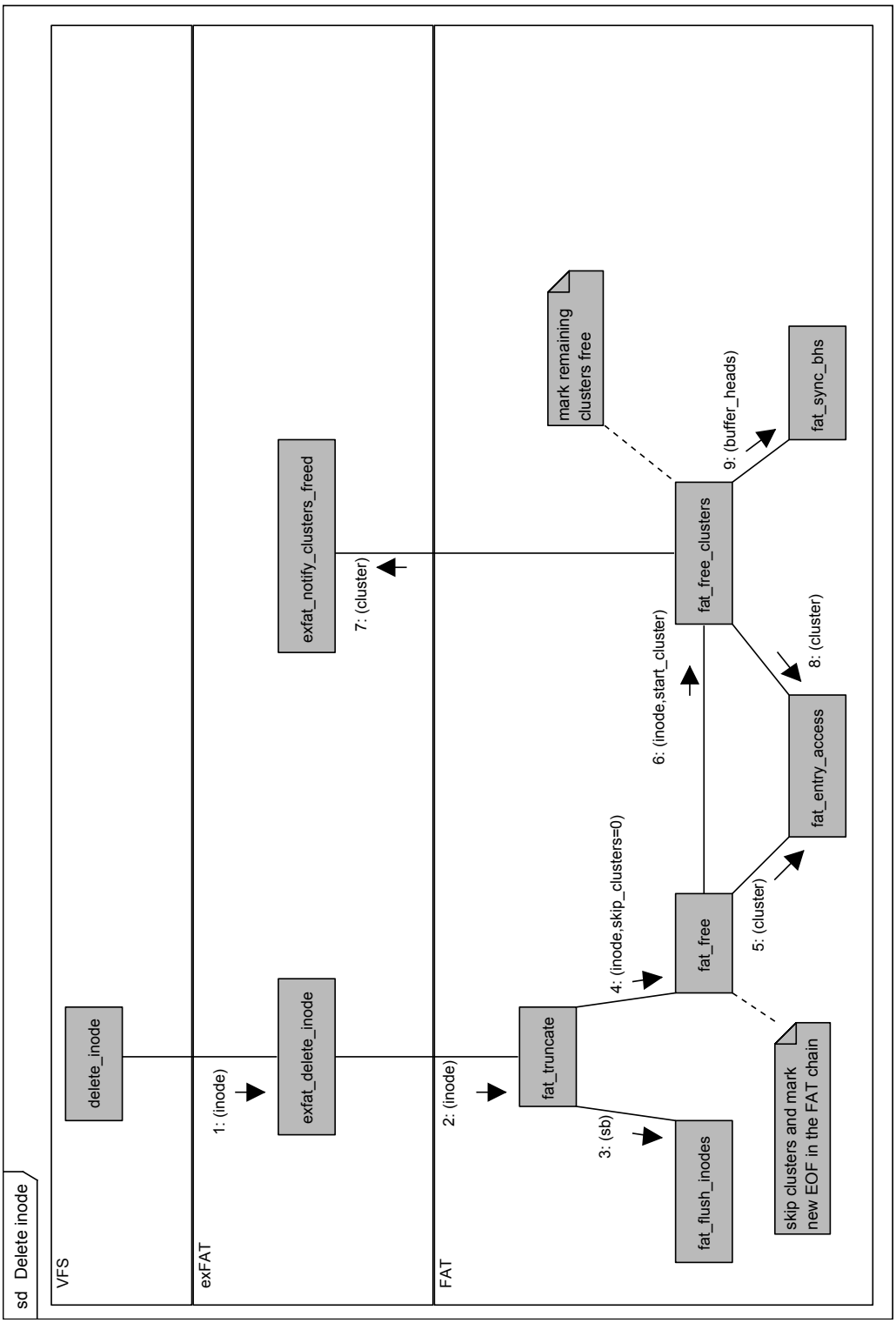
Štruktúru volaní funkcií znázorňujú nasledujúce diagramy. Komunikujúce entity sú teda funkcie, správy medzi nimi znamenajú volania alebo návraty z volaní. Pre lepšiu názornosť som si dovolil porušiť UML syntax a v názve správ uvádzam iba pomenovanie prenášaných dát, nie identifikátor správy ani skutočné parametre.



Obr. 5.1: Diagram volaní pri čítaní obsahu adresára a metadát súborov



Obr. 5.2: Diagram volaní pri načítavaní súborového systému



Obr. 5.3: Diagram volaní pri uvoľňovaní miesta pri mazaní súboru

Kapitola 6

Záver

V súlade so stanovenými cieľmi som vytvoril dokumentáciu súborového systému exFAT, ktorá je dostatočná pre implementáciu ovládača pre čítanie aj zápis. Ďalej som takýto ovládač implementoval ako modul pre jadro operačného systému GNU/LINUX. Možnosti rozšírenia tejto práce spočívajú v rozsiahlom testovaní napísaného softvéru, prípadne v implementácii utilít pre vytvorenie, kontrolu a opravu chýb na tomto súborovom systéme.

Literatúra

[Cor] Microsoft Corporation. Description of the exfat file system driver update package. http://support.microsoft.com/kb/955704.

[Don] Peter A. Donis. Windows vs. linux file timestamps. http://www.peterdonis.net/computers/computersarticle3.html.

[GE] Richard Gooch and Pekka Enberg. Overview of the linux virtual file system. File ./Documentation/filesystems/vfs.txt in kernel source tree. http://www.mjmwired.net/kernel/Documentation/filesystems/vfs.txt.

[Gho06] Vishal Ghotge. exfat. Technical report, Microsoft Corporation, 2006. slides 9-18.

[Hir] Ogawa Hirofumi. unpublished. Linux 2.6 patch.

[Koz] Charles M. Kozierok. Fat32 performance tradeoff: Fat32 cluster sizes and fat sizes. http://www.pcguide.com/ref/hdd/file/partFAT32-c.html.

[Mic06] Microsoft Corporation. *FAT: General Overview of On-Disk Format*, 1.03 edition, December 2006. http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx.

[Mik] Dmitry Mikhailov. Fat and ntfs performance. http://www.digit-life.com/articles/ntfs/index3.html.

[pro] Linux-NTFS project. Ntfs documentation. http://www.linux-ntfs.org/doku.php?id=downloads.

[sty] Linux kernel coding style. File ./Documentation/CodingStyle in kernel source tree. `http://www.mjmwired.net/kernel/Documentation/CodingStyle`.