VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

OPTIMALIZACE MAZÁNÍ SOUBORŮ V EXT4 PRO SNADNĚJŠÍ OBNOVU SOUBORŮ

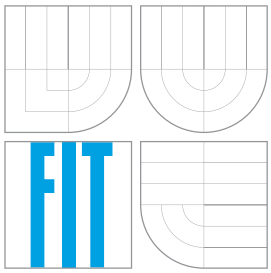
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUBOŠ UHLIARIK

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

OPTIMALIZACE MAZÁNÍ SOUBORŮ V EXT4 PRO SNADNĚJŠÍ OBNOVU SOUBORŮ

OPTIMIZE THE EXT4 UNLINK PROCESS TO MAKE RECOVERING DELETED FILES EASIER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUBOŠ UHLIARIK

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. TOMÁŠ VOJNAR, Ph.D.

BRNO 2014

Abstrakt

Práce se zabývá úpravou souborového systému ext4 za účelem optimalizace procesu mazání souborů pro jejich snadnější obnovu. Součástí práce je popis změn souborového systému ext4, včetně popisu nástroje, který se stará o samotnou obnovu smazaných souborů. Dále práce obsahuje popis, jak byly provedené změny v souborovém systému ext4 testovány. V této práci jsou rovněž popsány existující metody pro obnovu souborů, včetně všech jejich hlavních výhod a nevýhod.

Abstract

This thesis aims at changes of ext4 file system for optimizing unlink process and making undelete process easier. Part of the thesis descibes changes which were made in ext4 file system, including description of tool which was created as a part of this thesis for undeleting files. The thesis also includes description of tests which were used for testing modificated ext4 file system. In this thesis there are also described existing methods for undeleting deleted files, including all their advantages and disadvantages.

Klíčová slova

Linux, Kernel, ext4, VFS, i-uzel, extenty, extent strom, obnovení smazaného souboru

Keywords

Linux, Kernel, ext4, VFS, inode, extents, extent tree, undelete deleted file

Citace

Luboš Uhliarik: Optimalizace mazání souborů v ext4 pro snadnější obnovu souborů, bakalářská práce, Brno, FIT VUT v Brně, 2014

Optimalizace mazání souborů v ext4 pro snadnější obnovu souborů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Ing. Tomáše Vojnara, Ph.D. Další informace mi poskytl zástupce společnosti Red Hat, pan Ing. Lukáš Czerner. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Luboš Uhliarik
20. května 2014

Poděkování

Rád bych poděkoval vedoucímu práce prof. Tomáši Vojnarovi za vedení práce a poskytnuté rady. Dále bych chtěl poděkovat Ing. Lukáši Czernerovi z firmy Red Hat.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá analýzou a následnou úpravou procesu mazání souboru v souborovém systému ext4, který je dnes standardní součástí linuxového jádra. Důvod úpravy procesu mazání je vytvořit podmínky pro to, aby bylo při neúmyslném smazání souboru možné ho co nejsnadněji obnovit, a to v co možno největším počtu případů. Součástí této práce je také vytvoření nástroje, který bude schopen provést obnovu odstraněného souboru na základě provedených úprav v jádře Linux.

Obnovení smazaného souboru má význam například v případech, kdy je soubor smazán neúmyslně a vzápětí si uživatel uvědomí, že daný soubor odstranit nechtěl. Momentálně lze tuto situaci řešit různými nástroji na obnovu smazaných souborů, bohužel tyto nástroje nejsou funkční za všech okolností a každý tento nástroj má různé výhody a nevýhody.

Cílem této práce je tedy najít řešení, které by ulehčilo proces obnovy smazaného souboru a toto řešení realizovat v jádře operačního systému Linux pro souborový systém ext4.

Struktura práce je následující. Kapitola 2 se věnuje tomu, co to souborový systém je, popisu VFS (virtuálního souborového systému) a rovněž základnímu popisu souborového systému ext4. Kapitola 3 obsahuje detailnější popis toho, jak probíhá mazání souborů v ext4, včetně identifikace problému, který zabraňuje jednoduchému obnovení smazaného souboru. Následující kapitola 4 popisuje navržené změny v ext4, které usnadní rekonstrukci smazaného souboru. V kapitole 5 se práce věnuje popisu implementace navržených úprav a způsobu komunikace s hlavními vývojáři souborového systému ext4 o provedených změnách. Předposlední kapitola 6 se zabývá testováním provedených změn. V závěrečné kapitole 7 jsou zhodnoceny výsledky práce včetně uvedení různých návrhů na budoucí vylepšení prezentovaného řešení.

Zadání a řešení této práce bylo vytvořeno ve spolupráci s firmou Red Hat, kde jsem řešení a postupy konzultoval s vývojářem souborového systému ext4 Ing. Lukášem Czernerem.

Kapitola 2

Souborový systém ext4

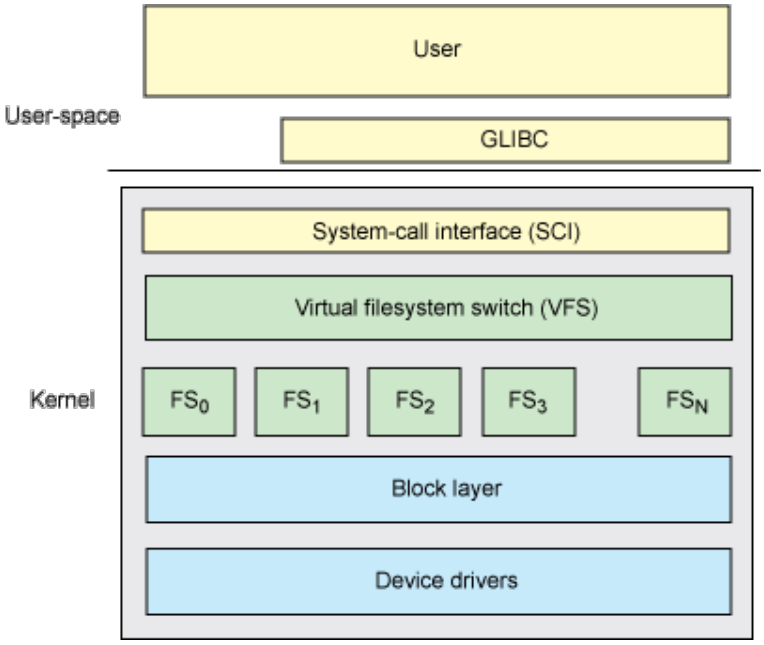
Tato kapitola popisuje základní principy a struktury souborového systému ext4, které je potřeba znát pro pochopení toho, jak mazání souborů v ext4 funguje. Ještě před tím je zde ale uveden stručný popis a definice VFS (virtuálního souborového systému), který je s ext4 blízce spojen a pochopení základů jeho fungování je proto pro pochopení zbytku textu nezbytné.

2.1 Popis VFS

Virtuální souborový systém (Virtual Filesystem, zkráceně VFS) je subsystém jádra Linuxu, který nabízí uživatelským programům jednotné přístupové rozhraní pro práci se soubory a souborovým systémem. VFS je tedy abstraktní vrstvou nad ostatními souborovými systémy, což dovoluje programům využívat jednotná systémová volaní k přístupu do různých souborových systémů [8].

Informace uvedené v předchozím odstavci budou důležité při zkoumání procesu mazání souboru, protože je nutné si uvědomit, že při jakékoliv operaci se souborem (například systémové volání `unlink` sloužící pro smazání souboru), se zavolá nejdříve obslužná funkce z VFS. Následně se v této funkci zavolá vlastní funkce pro souborový systém, na kterém soubor leží, a ta provede zbytek operace [1].

Na obrázku 2.1 jsou vyobrazeny jednotlivé úrovně výpočetního systému používajícího VFS. Nad vrstvou VFS leží standardní rozhraní jádra pro systémové volání zvané SCI (System-Call Interface). Toto rozhraní umožňuje volat z uživatelského prostoru (user space) funkce jádra, které pak v prostoru jádra (kernel space) volají další vnitřní funkce. Z toho vyplývá, že aplikace z uživatelského prostoru, která bude požadovat otevření souboru, zavolá systémové volání `sys_open`. Toto volání projde přes knihovnu glibc až do části jádra, kde se zpracovávají systémová volání. Jakmile bude toto volání zpracováno, zavolá se příslušná funkce z VFS. VFS poskytuje společný souborový model pro všechny souborové systémy, které musí mít implementované funkce požadované VFS. Pod vrstvou VFS leží samotný souborový systém, v tomto případě se bude jednat o ext4. Předposlední vrstvou jsou ovladače pro dané zařízení a poslední vrstvu pak tvoří záznamové medium (disk, síťové úložiště, část disku a další) [1].



Obrázek 2.1: Jednotlivé vrstvy architektury VFS [1]

2.2 Popis ext4 formátu

Souborový systém ext4 (dále už jen ext4) rozděluje paměť záznamového zařízení, na kterém se nachází, na mnoho malých bloků. Blok je skupina sektorů (nejmenší adresovatelná jednotka) na disku. Implicitně tyto bloky mají velikost 4KiB, nicméně tato velikost může být změněna při vytváření souborového systému. Tyto bloky jsou základní jednotkou v souborovém systému ext4 [2]. V celém dokumentu bude používán pojem `ext4 blok`, kterým je myšleno označení skupiny sektorů a základní nejmenší jednotka s kterou může ext4 pracovat.

Samotný prostor na disku s ext4 je rozdělen na řadu po sobě jdoucích skupin bloků (block group). Tato skupina bloků je sestavena právě z velkého množství ext4 bloků. Ve skupině bloků se nachází data blocks (datové bloky), které uchovávají samotná data na disku, je zde uložena tabulka s i-uzly (inodes), bitová mapa i-uzlů, která označuje, které i-uzly jsou použité a které jsou volné, a další položky, které není potřeba zde uvádět [2].

Všechny položky v ext4 strukturách jsou uloženy v bajtovém pořadí Little-endian. Bajtové pořadí Little-endian znamená, že LSB (nejméně významný bajt) je uložen na adrese s nejnižší hodnotou a za ním jsou uloženy bajty s adresou vyšší hodnoty až po MSB (nejvíce významný bajt). Ve stejném bajtovém pořadí pracuje většina procesorů, nicméně ne všechny, a proto musí být před zpracováním položky v procesoru provedena konverze z Little-endian na bajtové pořadí, v kterém pracuje procesor. Naopak položky v žurnálu (JBD2) jsou uloženy jako Big-Endian (pořadí bajtů je opačné, než je tomu u bajtového pořadí Little-endian) [2].

ext4 obsahuje mnoho struktur, avšak v tomto dokumentu budou uvedeny a popsány pouze ty, které jsou potřeba pro pochopení toho, jak v ext4 probíhá proces mazání souboru.

Skupina bloků

Skupina bloků má strukturu, jakou lze vidět na obrázku 2.2.

Skupina 0	1024 bajtů
Ext4 superblok	1 Ext4 blok
Skupina deskriptorů	mnoho Ext4 bloků
Rezervované GDT bloky	mnoho Ext4 bloků
Bitová mapa datových bloků	1 Ext4 blok
Bitová mapa i-uzlů	1 Ext4 blok
Tabulka i-uzlů	mnoho Ext4 bloků
Datové bloky	velmi mnoho Ext4 bloků

Obrázek 2.2: Layout ext4

U diskových pamětí bývá prvních 512 bajtů používáno pro hlavní spouštěcí záznam (MBR - Master Boot Record), ve kterém je uložena tabulka oddílů, kód zavaděče a další data. Proto v první skupině bloků je 1024 bajtů nevyužito (1024 zřejmě proto, že zde mohou být uloženy i jiné informace, které jsou větší než 512 bajtů). Každá jiná než první skupina bloků začíná superblokem, pro který je rezervován 1 ext4 blok, i když velikost superbloku je 1024 bajtů (z čehož nepřímo vyplývá, že minimální velikost ext4 bloku musí být nejméně 1024 bajtů). Za superblokem se nachází skupina deskriptorů skupiny bloků, které slouží k popisu této skupiny bloků. Rezervované GDT bloky slouží pro možnost zvětšování souborového systému za běhu. V bitové mapě datových bloků, která se nachází za rezervovanými GDT bloky, je vyznačeno, které ext4 datové bloky jsou používány a které ext4 bloky jsou prázdné. Ke stejnému účelu slouží bitová mapa i-uzlů, s tím rozdílem, že jsou zde označovaný využité a nevyužité i-uzly. V tabulce i-uzlů jsou za sebou uloženy struktury i-uzlů, přesněji struktury `ext4_inode` (viz Příklad 2.1). Jako poslední je v skupině bloků uloženo pole datových bloků. Tyto bloky mají stejnou velikost jako ext4 blok [2].

ext4 ovladač primárně pracuje se superblokem a skupinou deskriptorů skupiny bloků, které jsou uložené ve skupině bloků číslo 0. V dalších skupinách bloků jsou uloženy kopie superbloku a skupiny deskriptorů skupiny bloků, které mohou být použity v případě poškození souborového systému. Není však pravidlo, že v každé skupině bloků jsou uloženy tyto kopie. Pokud ve skupině bloků tyto záložní kopie chybí, skupina bloků bude začínat bitovou mapou datových bloků [2].

Superblok

Superblok obsahuje různorodé informace důležité pro správu celého souborového systému, jako je například počet ext4 bloků, počet i-uzlů a dalších. Strukturu superbloku zde nemá smysl podrobně popisovat, jelikož není tolik potřebná pro pochopení principu mazání souboru. Velikost superbloku je 1024 bajtů [2].

I-uzel

I-uzel je struktura, která uchovává všechny informace potřebné pro manipulaci se souborem nebo adresářem. V této struktuře lze najít metadata o souboru, jako například velikost souboru, počet pevných odkazů na soubor, datum poslední modifikace a mnoho dalších.

Pro každý soubor v souborovém systému tedy musí existovat I-uzel, který tento soubor popisuje. U souborového systému ext4 má záznam i-uzlu implicitně velikost 256 bajtů.

Ve starších verzích souborového systému Ext, i-uzel obsahoval přímé či nepřímé odkazy na ext4 bloky, které danému souboru náležely. V případě ext4 se dají také přímo či nepřímo mapovat ext4 bloky, implicitně se však používají extenty (rozsahy) a v i-uzlu je tedy uložen extent strom obsahující jednotlivé extenty [2].

Příklad 2.1 znázorňuje, jak vypadá struktura `ext4_inode` i-uzlu v souborovém systému ext4. Tato struktura obsahuje mnoho položek, popsány zde budou však pouze ty nejdůležitější, které jsou nutné pro pochopení procesu smazání souboru. Jednotlivé položky budu popisovat v pořadí, ve kterém se vyskytují v struktuře `ext4_inode`. Vlevo od vlastní struktury je uveden sloupec `Offset`, který není součásti struktury, ale je velmi vhodný pro přehlednost a rychlé vyhledání offsetu jednotlivých položek. Položka `i_mode` obsahuje oprávnění a typ souboru (soket, adresář, normální soubor a další). Další zajímavou položkou je `i_size_lo`, která uchovává dolních 32 bitů velikosti souboru v bajtech. Položky `i_atime`, `i_ctime`, `i_mtime` a `i_dtime` slouží k uložení času přístupu, vytvoření, úpravy a smazaní souboru. Položka `i_links_count` se používá k uložení počtu pevných odkazů na soubor. Položka `i_flags` obsahuje různé příznaky, jako jsou například `EXT4_EXTENTS_FL` (i-uzel používá extenty), `EXT4_HUGE_FILE_FL` (jedná se o velký soubor) a další. Jejich kompletní seznam lze najít v zdrojových kódech ext4. Důležitou položkou je `i_block`. Tato položka se používá pro ukládání různých dat, ale obecně platí, že u obyčejných souborů je zde uložena informace o mapování bloků (v ext4 implicitně extenty). Položka `i_block` má velikost 60 bajtů. Poslední zajímavou položkou je `i_size_high`, která slouží pro uchování horních 32 bitů z velikosti souboru a rozšiřuje tak položku `i_size_lo` [2].

Tabulka i-uzlů

Tabulka i-uzlů je souvislá část paměti (pole) po sobě jdoucích záznamů i-uzlů. Tato tabulka se nachází v každé skupině bloků, ve kterých má stejnou velikost [2].

Při vyhledávání i-uzlu daného čísla se nejdřív najde skupina bloků s intervalem i-uzlů, do kterého patří vyhledávaný i-uzel. Tato operace se dá popsat výrazem $(cislo\_iuzlu - 1)/pocet\_iuzlu\_ve\_skupine$. Následně se pokračuje ve vyhledávání v tabulce i-uzlů, kde se stanoví offset podle vzorce $(cislo\_iuzlu - 1)\%pocet\_iuzlu\_ve\_skupine$. Od čísla i-uzlu se odečítá hodnota 1, protože neexistuje i-uzel s číslem 0. Počet i-uzlů ve skupině bloků je uložen v superbloku a dá se pomocí různých nástrojů pro práci s ext4 přečíst [2].

Extent

Extent je posloupnost ext4 bloků, který je určen počáteční adresou ext4 bloku a počtem následujících bloků, které jsou umístěny sekvenčně za sebou. Důvod, proč se využívají extenty místo přímého mapování ext4 bloků je takový, že zlepšují výkonnost při práci se soubory s vyšším počtem bloků a také zmenšují fragmentaci souboru [2].

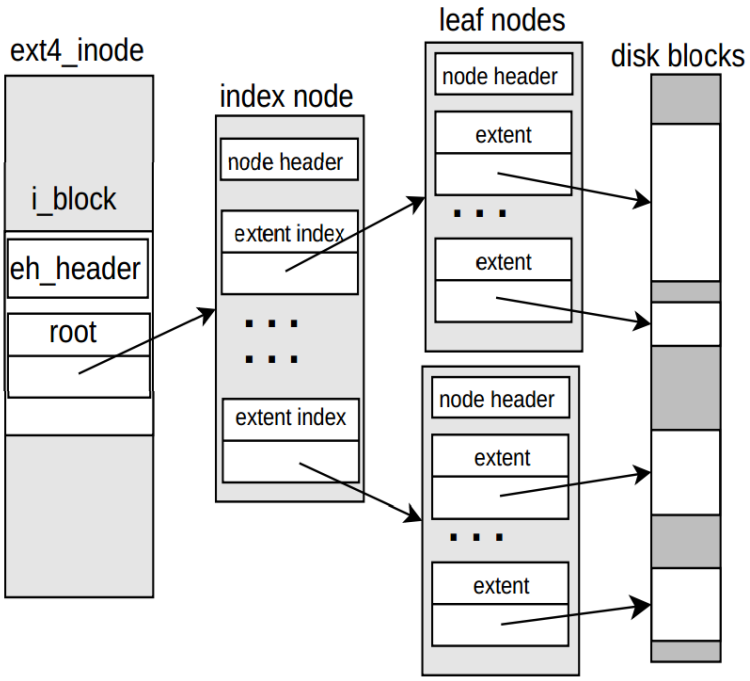
Jeden extent může mít velikost až 128 MiB (při velikosti ext4 bloku 4KiB) z toho důvodu, že položka uchovávající délku extentu ve struktuře `ext4_extent` má velikost 16 bitů (používá se však jen 15 bitů, tím pádem je maximální délka 32768 ext4 bloků) [2].

Extent strom

V ext4 bylo nahrazeno předešlé logické mapování bloků u souboru za extent strom (extent tree), což je v podstatě varianta B+ stromu (bez vyvažování, bez zřetězení, nanejvýš 5

Příklad 2.1: Struktura ext4 i-uzlu

```
Offset | struct ext4_inode {
0x0    |        __le16  i_mode;
0x2    |        __le16  i_uid;
0x4    |        __le32  i_size_lo;
0x8    |        __le32  i_atime;
0xC    |        __le32  i_ctime;
0x10   |        __le32  i_mtime;
0x14   |        __le32  i_dtime;
0x18   |        __le16  i_gid;
0x1A   |        __le16  i_links_count;
0x1C   |        __le32  i_blocks_lo;
0x20   |        __le32  i_flags;
0x24   |        union {
       |                struct {
       |                        __le32  l_i_version;
       |                } linux1;
       |                struct {
       |                        __u32  h_i_translator;
       |                } hurd1;
       |                struct {
       |                        __u32  m_i_reserved1;
       |                } masix1;
       |        } osd1;
0x28   |        __le32  i_block[EXT4_N_BLOCKS];
0x64   |        __le32  i_generation;
0x68   |        __le32  i_file_acl_lo;
0x6C   |        __le32  i_size_high;
0x70   |        __le32  i_obso_faddr;
0x74   |        union {
       |                struct {
       |                        __le16  l_i_blocks_high;
       |                        __le16  l_i_file_acl_high;
       |                        __le16  l_i_uid_high;
       |                        __le16  l_i_gid_high;
       |                        __u32   l_i_reserved2;
       |                } linux2;
       |                struct {
       |                        __le16  h_i_reserved1;
       |                        __u16   h_i_mode_high;
       |                        __u16   h_i_uid_high;
       |                        __u16   h_i_gid_high;
       |                        __u32   h_i_author;
       |                } hurd2;
       |                struct {
       |                        __le16  h_i_reserved1;
       |                        __le16  m_i_file_acl_high;
       |                        __u32   m_i_reserved2[2];
       |                } masix2;
       |        } osd2;
0x80   |        __le16  i_extra_isize;
0x82   |        __le16  i_pad1;
0x84   |        __le32  i_ctime_extra;
0x88   |        __le32  i_mtime_extra;
0x8C   |        __le32  i_atime_extra;
0x90   |        __le32  i_crtime;
0x94   |        __le32  i_crtime_extra;
0x98   |        __le32  i_version_hi;
       | };
```



Obrázek 2.3: Ukázka struktury extent stromu [9]

úrovní).

Cílová cesta u symbolických odkazů (symlinks), může být uložena místo struktury extent stromu, pokud je její velikost menší než 60 bajtů. Pokud je velikost cílové cesty k souboru větší, jsou k jejímu uložení použity extenty pro na-alokování ext4 bloků, které budou cílovou cestu obsahovat.

Jak takový extent strom vypadá, si lze prohlédnout na obrázku 2.3. První úroveň tohoto stromu je vždy uložena přímo v těle i-uzlu v položce `i_block`, kde se vejdou až 4 uzly tohoto stromu, a další úrovně stromu (pokud jsou potřeba) jsou pak uloženy v datových ext4 blocích [2].

Jeden uzel extent stromu je tvořen souvislou pamětí, která má u i-uzlu velikost 60 bajtů a u ext4 bloku je to právě velikost tohoto bloku. Tento uzel vždy začíná strukturou `ext4_header`. Pokud se jedná o listy stromu (leaf nodes – uzly na poslední úrovni stromu), jsou dále uloženy struktury `ext4_extent`, případně pokud se jedná o vnitřní uzel, jsou zde uloženy struktury `ext4_extent_idx`. Obě tyto struktury mají shodnou velikost 12 bajtů. U uzlu stromu, který se nenachází v i-uzlu, je jako poslední uložena struktura `ext4_extent_tail`, ve které je uložen kontrolní součet (checksum). V i-uzlu ukládání kontrolního součtu pro extent strom nemá smysl, protože i-uzel už obsahuje vlastní kontrolní součet. Velikost struktury `ext4_extent_tail` je 4 bajty [2].

V následujících příkladech budou uvedeny struktury, které se používají pro ukládání informací o extentech. Datové typy, které jsou v těchto strukturách použity, určují, v jakém bajtovém pořadí je položka struktury uložena a jakou má tato položka velikost. Například datový typ `__le16` značí, že se jedná o položku typu Little-Endian a má velikost 16 bitů. Pokud u datového typu není důležité pořadí bitů, je vloženo klasické bez-znaménkové číslo a jeho název začíná prefixem `__u` (od ang. slova unsigned – bez znaménka).

V příkladu 2.2 je uvedena struktura pro hlavičku, která se nachází na začátku každého

uzlu extent stromu. První položka `eh_magic` má vždy hexadecimální hodnotu 0xF30A, která označuje, že jsou zde uložena data ze struktury `ext4_extent_header`. Další položkou struktury je `eh_entries`, která udává, kolik leží za touto strukturou validních záznamů (struktura `ext4_extent_idx`, případně struktura `ext4_extent`). V položce `eh_max` je uložen nejvyšší možný počet záznamů v aktuálním uzlu. Položka `eh_depth` slouží pro uchování hodnoty úrovně aktuálního uzlu. Pokud je hodnota položky `eh_depth` rovna 0, jsou následně v celém uzlu uloženy struktury typu `ext4_extent`. V opačném případě, jsou v tomto uzlu uloženy struktury typu `ext4_extent_idx`. Poslední položka `eh_generation` se v ext4 nepoužívá [2].

Příklad 2.2: Struktura pro hlavičku uzlu v extent stromu

```
struct ext4_extent_header {
    __le16  eh_magic;
    __le16  eh_entries;
    __le16  eh_max;
    __le16  eh_depth;
    __le32  eh_generation;
};
```

Příklad 2.3 obsahuje strukturu pro popis vnitřních uzlů extent stromu, které jsou rovněž známé pod názvem indexační uzly (index nodes). Tato struktura má velikost 12 bajtů. První položka `ei_block` slouží k uložení čísla ext4 bloku, od kterého index tohoto uzlu pokrývá ext4 bloky souboru. Položka `ei_leaf_lo` uchovává dolních 32 bitů adresy ext4 bloku, který slouží jako uzel extent stromu. Odkazovaný uzel může opět obsahovat buď struktury `ext4_extent_idx`, nebo struktury `ext4_extent`, podle toho, jaká je hodnota položky `eh_depth` struktury `ext4_extent_header`. Položka `ei_leaf_hi` je horních 16 bitů z předchozí adresy. Poslední položka `ei_unused` není využívána[2].

Příklad 2.3: Struktura pro vnitřní uzel stromu

```
struct ext4_extent_idx {
    __le32  ei_block;
    __le32  ei_leaf_lo;
    __le16  ei_leaf_hi;
    __u16   ei_unused;
};
```

V příkladu 2.4 lze vidět strukturu, která popisuje samotný extent (listy extent stromu). Struktura `ext4_extent` má velikost 12 bajtů. První položka `ee_block` slouží k uložení čísla bloku souboru, od kterého extent začíná. Položka `ee_len` slouží k uložení délky extentu. Poslední dvě položky uchovávají adresu ext4 bloku, od kterého extent začíná, kde `ee_start_hi` je horních 16 bitů adresy a `ee_start_lo` je dolních 32 bitů adresy.

Příklad 2.4: Struktura pro listový uzel

```
struct ext4_extent {
    __le32  ee_block;
    __le16  ee_len;
    __le16  ee_start_hi;
    __le32  ee_start_lo;
};
```

Poslední strukturou v uzlu extent stromu je struktura z příkladu 2.5 `ext4_extent_tail` obsahující kontrolní součet. Tato struktura se vkládá pouze do uzlů extent stromu, které

jsou mimo i-uzel. To se děje, jak už bylo napsáno dříve, z důvodu, že samotný i-uzel obsahuje kontrolní součet, a proto ho není potřeba počítat.

Příklad 2.5: Struktura pro kontrolní součet

```
struct ext4_extent_tail {
    __le32  et_checksum;
};
```

Žurnál

Součástí souborového systému ext4 je žurnál (journal nebo také zkráceně JBD2), který chrání ext4 před poškozením dat, které mohou vzniknout při neočekávaném pádu systému. Žurnál je vlastně malá souvislá část paměti na disku uvnitř souborového systému (nástroj mkfs, který souborový systém vytváří se snaží žurnál umístit doprostřed), která slouží k co možná nejrychlejšímu ukládání informací o blocích, které mají být zapsány na disk [2].

Z důvodu výkonnosti ext4 implicitně žurnáluje pouze metadata, nikoliv data. Z toho vyplývá, že v případě pádu systému nebude garantována konzistence datových bloků souboru. Pomocí přidání parametrů při připojování diskového oddílu se dá zajistit i žurnálovaní dat, to však zapříčiní rapidní zhoršení výkonu, jelikož všechny zapsané datové bloky se budou muset žurnálovat [2].

Formát a struktura žurnálu zde uvedeny nebudou, protože je není potřeba znát k pochopení toho, jak funguje mazání souboru.

Kapitola 3

Mazání a rekonstrukce souboru v ext4

Tato kapitola se věnuje popisu procesu mazání, a rekonstrukce souboru s cílem identifikovat problém, který brání rekonstrukci souboru v aktuálním stavu Linuxového jádra. Většina získaných informací je výsledkem experimentů, které byly součástí této práce. Tyto experimenty byly vždy porovnány se zdrojovými kódy a bylo na nich ověřeno správné pochopení zdrojových kódů. Kapitola rovněž obsahuje popis nástrojů a postupů, které byly k experimentování použity. Předpokladem k pochopení této kapitoly jsou základní znalosti práce s operačním systémem GNU/Linux.

3.1 Analýza smazaného souboru

Prvním důležitým krokem je ujasnit si, co se vlastně stane s daty a metadaty, když je soubor smazán z disku. K experimentu si vytvořím prázdný diskový oddíl naformátovaný na souborový systém ext4. K této operaci je nejjednodušší použít standardní nástroj mkfs. Při vytváření oddílu v rámci tohoto experimentu není důležitá velikost ani další parametry, ale pouze souborový systém oddílu, kterým bude ext4. Dále se v textu bude vycházet z toho, že nový oddíl je vytvořen na zařízení `/dev/sda` a jméno oddílu je `/dev/sda2`.

Následným krokem je připojení souborového systému k libovolnému adresáři v systému. Po připojení zařízení a změně aktuálního adresáře za adresář, ke kterému je zařízení připojeno, se pomocí příkazu z příkladu 3.1 vytvoří soubor, který se bude používat v následujících příkladech. Důležitý je příkaz `sync`, který zajistí zapsání dat z rychlé vyrovnávací paměti pamětí (cache) na disk.

Příklad 3.1: vytvoření testovacího souboru

```
$ echo "Testovaci soubor" > pokus.txt
$ sync
```

Jakmile je soubor vytvořen, může se začít s analýzou organizace zapsaných dat na disku. Způsobů, jak tyto data a metadata z disku přečíst, je mnoho. Dále v textu však budou používány převážně nástroje z balíčku nástrojů Sleuthkit. Prvním krokem je získání čísla i-uzlu, který obsahuje metadata o dříve vytvořeném souboru `pokus.txt`. Číslo i-uzlu lze vypsat pomocí standardního programu `ls` přidáním parametru `-i`. Výstup pak bude vypadat podobně jako na příkladu 3.2.

Příklad 3.2: číslo i-uzlu souboru pokus.txt

```
$ ls -i pokus.txt
12 pokus.txt
```

Z výstupu programu `ls` lze vyčíst, že metadata o souboru pokus.txt jsou uložena v i-uzlu číslo 12. Je potřeba však zjistit, kde fyzicky na disku leží tento i-uzel. K tomu poslouží program fsstat, který slouží k vypsání informací, na kterých ext4 blocích se jednotlivé části souborového systému nachází. Jako parametr programu fsstat se předává název zařízení, o kterém se budou informace vypisovat (v tomto případě `/dev/sda2`). Na příkladu 3.3 je zobrazena vybraná část výstupu programu.

Příklad 3.3: část výstupu programu fsstat[1]

```
$ fsstat /dev/sda2

FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext3

...

BLOCK GROUP INFORMATION
--------------------------------------------
Number of Block Groups: 16
Inodes per group: 8192
Blocks per group: 32768

Group: 0:
  Inode Range: 1 - 8192
  Block Range: 0 - 32767
  Layout:
    Super Block: 0 - 0
    Group Descriptor Table: 1 - 1
    Data bitmap: 129 - 129
    Inode bitmap: 145 - 145
    Inode Table: 161 - 672
    Data Blocks: 673 - 32767
  Free Inodes: 8180 (99%)
  Free Blocks: 24409 (74%)
  Total Directories: 2

...
```

V tomto výstupu je nejdříve potřeba vyhledat, ve které skupině bloků se bude nacházet hledaný i-uzel číslo 12. Každá skupina bloků má ve výstupu programu pod sebou vypsaný interval, které i-uzly jsou v dané skupině bloků obsaženy. I-uzlu číslo 12 tedy odpovídá skupina bloků 0 (`Group 0`), ve které se nachází i-uzly s číslem v intervalu 1-8192 (`Inode Range: 1 - 8192`). Poté, co byla nalezena skupina do které i-uzel patří, je potřeba se podívat, kde přesně na zařízení se nachází tabulka i-uzlů. V tomto případě tabulka začíná na 161. ext4 bloku a pokračuje až po 672. ext4 blok (`Inode Table: 161 - 672`). Nyní je potřeba spočítat, ve kterém bloku se bude nacházet požadovaný i-uzel číslo 12. K tomu

[1]Nástroj fsstat, tak jako ostatní nástroje z balíčku Sleuthkit, je napsán s podporou pro souborový systém Ext3, ale vzhledem k tomu, že ext4 je s Ext3 zpětně kompatibilní (až na drobné výjimky), lze tento nástroj použít k získání čísla ext4 bloku, od kterého začíná tabulka i-uzlů.

je potřeba znát velikost ext4 bloku, která je implicitně 4 KiB. Velikost ext4 bloku je také vypsána po spuštění nástroje `mkfs` při formátování diskového oddílu. V případě, kdy je již naformátovaný diskový oddíl, lze tuto informaci zjistit pomocí příkazu z příkladu 3.4. Stejným způsobem lze zjistit také velikost i-uzlu (Příklad 3.5).

Příklad 3.4: zjištění velikosti ext4 bloku

```
$ tune2fs -l /dev/sda2 | grep -i 'block size'
Block size:               4096
```

Příklad 3.5: zjištění velikosti i-uzlu

```
$ tune2fs -l /dev/sda2 | grep -i 'inode size'
Inode size:               256
```

S předpokladem, že velikost ext4 bloku je 4096 bajtů a velikost i-uzlu je 256 bajtů se dá jednoduše spočítat, že do jednoho ext4 bloku se vejde přesně 16 i-uzlů. Jak již bylo řečeno, tabulka i-uzlů je souvislá část paměti (pole struktur i-uzlu). Proto když je potřeba zjistit, ve kterém prvku tohoto pole se hledaný i-uzel nachází, je potřeba celočíselně vydělit číslo i-uzlu (v tomto případě 12) číslem představující maximální možný počet i-uzlů v jednom ext4 bloku (v tomto případě 16). Výsledkem této operace je číslo, které představuje offset, který se přičítá k číslu ext4 bloku prvního prvku tabulky i-uzlů. V tomto případě se offset rovná 0, tudíž i-uzel se bude nacházet v prvním ext4 bloku z tabulky i-uzlů. Druhý offset, který je potřeba spočítat, určuje, kde přesně v tomto ext4 bloku se hledaný i-uzel nachází. Ten se spočítá tak, že se vynásobí velikost i-uzlu (256 bajtů) se zbytkem po celočíselném dělení čísla i-uzlu číslem představujícím maximální možný počet i-uzlů v jednom ext4 bloku zmenšeným o 1. Výsledek této operace je 2816 bajtů.

Číslo ext4 bloku (vypočítaného jako číslo prvního ext4 bloku tabulky i-uzlů zvětšeným o offset), ve kterém se nachází i-uzel je 161. Tento ext4 blok je možné pomocí programu `blkcat` zkopírovat a odeslat přes rouru na vstup programu hexdump, který ihned zobrazí data v hexadecimálním formátu. Výsledek této akce uvádí příklad 3.6.

Příklad 3.6: zobrazení i-uzlu

```
$ blkcat /dev/sda2 161 |hexdump -v -s 2816 -n 256
0000b00 81a4 0000 0011 0000 3c13 5349 3c13 5349
0000b10 3c13 5349 0000 0000 0000 0001 0008 0000
0000b20 0000 0008 0001 0000 f30a 0001 0004 0000
0000b30 0000 0000 0000 0000 0001 0000 9291 0000
0000b40 0000 0000 0000 0000 0000 0000 0000 0000
0000b50 0000 0000 0000 0000 0000 0000 0000 0000
0000b60 0000 0000 c06e de12 0000 0000 0000 0000
0000b70 0000 0000 0000 0000 0000 0000 0000 0000
0000b80 001c 0000 89fc 9394 89fc 9394 89fc 9394
0000b90 3c13 5349 89fc 9394 0000 0000 0000 0000
0000ba0 0000 0000 0000 0000 0000 0000 0000 0000
0000bb0 0000 0000 0000 0000 0000 0000 0000 0000
0000bc0 0000 0000 0000 0000 0000 0000 0000 0000
0000bd0 0000 0000 0000 0000 0000 0000 0000 0000
0000be0 0000 0000 0000 0000 0000 0000 0000 0000
0000bf0 0000 0000 0000 0000 0000 0000 0000 0000
0000c00
```

Po porovnání dat z hexaeditoru se strukturou i-uzlu z příkladu 2.1 lze vidět hodnoty jednotlivých položek. Je třeba si dát pozor na to, že data jsou uložena v bajtovém pořadí

Little-Endian. Program `hexdump` automaticky převádí data na bajtové pořadí Big-Endian, ale jen po dvojicích bajtů. Tím pádem je nutné ještě změnit pořadí jednotlivých dvojic bajtů, pokud má položka větší velikost než dva bajty.

Dále se soustředíme na data kořenového uzlu extent stromu (ležícího v i-uzlu), která začínají na 40. bajtu i-uzlu a mají velikost 60 bajtů (viz Příklad 2.1). Celý uzel extent stromu z i-uzlu pak vypadá jak ukazuje příklad 3.7.

Příklad 3.7: zobrazení uzlu extent stromu

```
$ blkcat /dev/sda2 161 |hexdump -v -s 2856 -n 60
0000b28 f30a 0001 0004 0000 0000 0001 0000 0000
0000b38 0001 0000 9291 0000 0000 0000 0000 0000
0000b48 0000 0000 0000 0000 0000 0000 0000 0000
0000b58 0000 0000 0000 0000 0000 0000
0000b64
```

Po porovnání výstupu se strukturami z příkladů 2.2, 2.3 a 2.4 lze vidět jednotlivé hodnoty jednotlivých položek struktur. Uzel extent stromu začíná klasicky magickou hodnotou 0xF30A, poté následuje počet záznamů v tomto uzlu (hodnota 1), maximální počet záznamů v tomto uzlu (hodnota 4), úroveň aktuálního uzlu extent stromu (hodnota 0) a poslední položkou struktury `ext4_extent_header` je položka `eh_generation`, která se v ext4 nevyužívá. Jelikož se jedná o úroveň číslo 0, následující záznamy v uzlu se budou interpretovat jako data struktury `ext4_extent`. Jak bylo napsáno výše, počet záznamů v tomto uzlu je roven hodnotě 1, tudíž se bude načítat pouze jedna struktura (soubor je složen pouze z jednoho extentu). Po načtení jedné struktury typu `ext4_extent` jsou k dispozici tyto informace:

- Extent začíná od 0. bloku souboru (položka `ee_block` má hodnotu 0)
- délka extentu je 1 ext4 blok (položka `ee_len` má hodnotu 0)
- data extentu začínají od 37521. ext4 bloku (kombinace položek `ee_start_hi` a `ee_start_lo`)

Nyní je známo, že data tohoto souboru jsou uložena v jednom ext4 bloku, jehož číslo je 37521. Příkazem `blkcat` si bude lehké tuto skutečnost ověřit. Zobrazení dat souboru ukazuje příklad 3.8.

Příklad 3.8: zobrazení dat souboru

```
$ blkcat /dev/sda2 37521 |hexdump
0000000 6554 7473 766f 6361 2069 6f73 6275 726f
0000010 000a 0000 0000 0000 0000 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
*
0001000
$ blkcat /dev/sda2 37521
Testovaci soubor
```

Proč jsou ve výstupu programu hexdump nulové bajty až do konce souboru? Protože data souboru jsou vždy zarovnána na celý ext4 blok. Jak Linux pozná, které data ještě patří do souboru, a které slouží jen jako zarovnání bloku a nejsou součástí souboru? Toto řeší položky `i_size_lo` a `i_size_high`, které jsou součástí i-uzlu, a které když se složí v jednu hodnotu, určují přesnou velikost souboru v bajtech. Právě pomocí těchto položek se přesně

ví, kde soubor končí. U příkladu 3.8 má hodnota vzniklá spojením těchto položek hodnotu 17 (hexadecimálně 0x11).

Nyní, když už bylo demonstrováno, kde a jak vyhledat metadata a data souboru, provedeme smazání souboru a opětovně vypíšeme ext4 bloky obsahující tyto data a metadata tak, jak to lze vidět na příkladu 3.9.

Příklad 3.9: smazání souboru a vypsání ext4 bloků

```
$ rm pokus.txt
$ sync
$ blkcat /dev/sda2 161 |hexdump -v -s 2816 -n 256
0000b00 81a4 0000 0000 0000 d154 534a d6f0 534a
0000b10 d6f0 534a d6f0 534a 0000 0000 0000 0000
0000b20 0000 0008 0001 0000 f30a 0000 0004 0000
0000b30 0000 0001 0000 0000 0001 0000 9292 0000
0000b40 0000 0000 0000 0000 0000 0000 0000 0000
0000b50 0000 0000 0000 0000 0000 0000 0000 0000
0000b60 0000 0000 c06f de12 0000 0000 0000 0000
0000b70 0000 0000 0000 0000 0000 0000 0000 0000
0000b80 001c 0000 80f4 468d 80f4 468d a238 a21c
0000b90 d154 534a a238 a21c 0000 0000 0000 0000
0000ba0 0000 0000 0000 0000 0000 0000 0000 0000
0000bb0 0000 0000 0000 0000 0000 0000 0000 0000
0000bc0 0000 0000 0000 0000 0000 0000 0000 0000
0000bd0 0000 0000 0000 0000 0000 0000 0000 0000
0000be0 0000 0000 0000 0000 0000 0000 0000 0000
0000bf0 0000 0000 0000 0000 0000 0000 0000 0000
0000c00

$ blkcat /dev/sda2 37521 |hexdump
0000000 6554 7473 766f 6361 2069 6f73 6275 726f
0000010 000a 0000 0000 0000 0000 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
*
0001000
```

Po smazání souboru je opět nutné provést příkaz `sync`, aby se změněné ext4 bloky uložily na disk. Při porovnání hexadecimální podoby i-uzlu z doby před smazáním (Příklad 3.6) a po smazání souboru (Příklad 3.9) lze konstatovat, že následující položky struktury i-uzlu z Příkladu 2.1 byly pozměněny procesem mazání:

- vynulování položek `i_size_lo` a `i_size_high` (velikost souboru),
- snížení hodnoty položky `i_links_count` (počet pevných odkazů na soubor) na hodnotu 0,
- vynulování položky `i_blocks_lo` (počet zabraných sektorů disku),
- `i_atime` (čas posledního přístupu k souboru), `i_ctime` (čas poslední úpravy informací v i-uzlu), `i_mtime` (čas poslední úpravy obsahu souboru), `i_dtime` (čas smazání souboru), `i_crtime` (čas vytvoření souboru),
- vynulování položek `eh_entries` a `eh_depth` struktury `ext4_extent_header` (2.2), která je součástí položky `i_block`,
- vynulování položek `ee_len`, `ee_start_hi` a `ee_start_lo` struktury `ext4_extent` (2.4), která je součástí položky `i_block`,

- `i_ctime_extra` (rozšíření přesnosti `i_ctime`), `i_mtime_extra` (rozšíření přesnosti `i_mtime`), `i_atime_extra` (rozšíření přesnosti `i_atime`), `i_crtime_extra` (rozšíření přesnosti `i_crtime`)

Položka `i_links_count` uchovává počet pevných odkazů na soubor. Informace, že počet pevných odkazů je nulový pouze potvrzuje, že soubor byl smazán, a proto není pro rekonstrukci souboru potřeba.

Taktéž hodnota položky `i_blocks_lo` není potřeba pro rekonstrukci souboru.

Položky, které uchovávají čas, nejsou pro obnovu souboru důležité. Jediný případ, kdy připadá v úvahu jejich využití, je při prohledávání i-uzlů a obnově souborů podle času smazání (`i_dtime`) nebo jakékoliv jiné filtrování podle ostatních časů uložených v i-uzlu.

Položky (`i_size_lo` a `i_size_high`) jsou pro obnovu souboru daleko důležitější. Absence těchto položek způsobí, že nebude známo, jak byl smazaný soubor velký. Velikost souboru by se dala nahradit sečtením velikostí všech extentů, ze kterých se soubor skládal. Tento způsob má však jeden problém, a to ten, že nelze přesně určit, kolik dat bylo obsaženo v posledním ext4 bloku posledního extentu souboru. Varianty, jak problém řešit, jsou následující:

a) Nechat obnovený soubor zarovnaný na velikost celého ext4 bloku, kde zarovnání budou tvořit nulové bajty.

b) Odstranění nulových bajtů z konce souboru. Toto však může způsobit, že budou odstraněny nulové bajty, které byly součástí mazaného souboru.

c) Uchovat zbytek po dělení velikosti souboru velikostí ext4 bloku. Výsledná hodnota je postačující k určení velikosti dat v posledním ext4 bloku.

Nejdůležitější informace, které se mění při smazání souboru, jsou pak položky `eh_entries`, `eh_depth`, `ee_len`, `ee_start_hi` a `ee_start_lo`. Tyto položky jsou nutné pro rekonstrukci souboru a nedají se žádným způsobem nahradit ani vypočítat.

Dobrou zprávou je, že data souboru jsou na disku zachována v původní podobě, a jediné, co se měnilo, byla metadata.

Výše byl uveden nejjednodušší, kdy soubor obsahuje data, která jsou obsažena v jednom extentu. Na situaci se nic nemění, dokud data souboru nejsou rozdělena do více jak 4 extentů. Pokud se tak stane, extent strom bude obsahovat více úrovní a jeho mazání bude probíhat trochu odlišným způsobem.

Pro zjištění, jaké změny při odstranění takového souboru nastanou, je potřeba nejprve takový fragmentovaný soubor vytvořit. Jednou z možností je najít takový soubor, který se skládá z mnoha extentů. Zpravidla se jedná o soubory se záznamy činností systému (v GNU/Linuxu typicky v adresáři `/log`). Problém je však v tom, že nalézt takový soubor může trvat delší dobu (pokud se to vůbec povede) a pro pozdější testování je vhodnější takový soubor umět vytvořit programově. Z tohoto důvodu jsem naprogramoval jednoduchý program v jazyce C pro vytváření fragmentovaných souborů jménem fragmented. Princip programu fragmented spočívá v tom, že vždy zapíše blok dat a následně pomocí funkce `lseek` se jeden blok dat přeskočí (v tomto bloku budou zapsány nulové bajty). Tím je vytvořen tzv. řídký soubor (sparse file), kde data, která jsou přeskočena, způsobí, že následující blok dat bude zapsán v samostatném extentu. Nápověda k programu fragmented, který budu používat v následujících příkladech, je zobrazena v příkladu 3.10 [10].

Příklad 3.10: nápověda k programu fragmented

```
$ ./fragmented -h
```

```
Pouziti: ./fragmented file [MOZNOSTI]

Moznosti:
        -h: vypise tuto napovedu
        -b block_size: specifikuje velikost bloku, vychozi
                hodnota je 4096
        -c extents_count: specifikuje pocet vytvorenych
             extentu, vychozi hodnota je 5
        -d data: specifikuje ktery bajt bude pouzit na
                vyplneni vsech bloku. Hodnota muze byt z
                intervalu 0-255. Vychozi hodnota je 5
```

Pro analýzu toho, co se stane se souborem s extent stromem složeného z více úrovní, je potřeba nejprve takový soubor vytvořit. Jak se ale zjistí, z kolika extentů musí být soubor poskládán, aby měl extent strom 3 úrovně? Toto se dá jednoduše spočítat. Nultá úroveň stromu je vždy uložena v i-uzlu a vejdou se do ní 4 indexační uzly (nebo 4 extenty, pokud má strom pouze jednu úroveň). Každý z indexačních uzlů odkazuje na ext4 blok, který má velikost 4096 bajtů. Struktura `ext4_extent_header` zabere 12 bajtů, další 4 bajty struktura `ext4_extent_tail` a do zbytku se uloží struktury `ext4_extent_idx` případně `ext4_extent`, kde každá má velikost 12 bajtů. Po výpočtu tedy vychází, že do jednoho bloku se vejde 340 extentů nebo indexačních uzlů. Z toho vyplývá, že pro vytvoření extent stromu o třech úrovních bude potřeba $4*340+1$ extentů. Jak bude takový extent strom vypadat prezentuje obrázek 3.1.

Vytvoření tohoto souboru s požadovanou fragmentací se provede, jak je uvedeno v příkladu 3.11.

Příklad 3.11: vytvoření fragmentovaného souboru

```
$ ./fragmented pokus.dat -c 1361 -d 65
$ sync
```

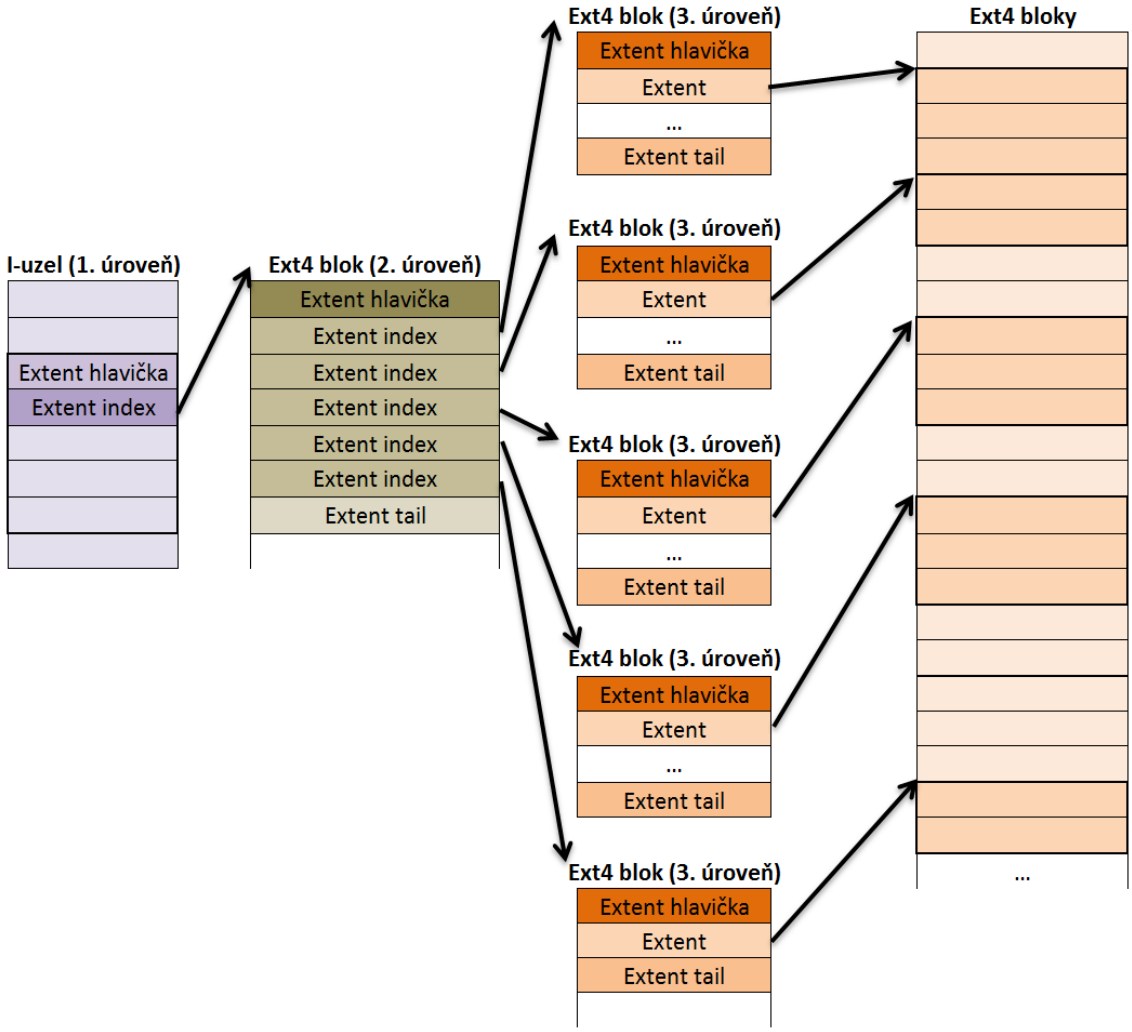
Tímto byl vytvořen soubor, který bude složen z 1361 extentů a data souboru budou znaky A (mimo přeskočené bloky, které budou vyplněny nulovými bajty). Vyhledání čísla i-uzlu, které náleží tomuto souboru, proběhne stejně jako u souboru s jedním extentem. Následně po smazání souboru a analýze metadat je patrné, že indexační uzly zůstaly beze změn.

Tím jsem došel k závěru, že při procesu mazání je změněna pouze malá část metadat souboru při jakékoliv jeho velikosti. Nejdůležitější změny, které znemožňují rekonstrukci souboru, jsou provedeny v extent stromu ve strukturách `ext4_ext_header` a `ext4_extent`. Nyní bude potřeba najít, která část zdrojového kódu Linuxu je zodpovědná za tyto změny.

3.2 Proces mazání souboru

Zdrojové kódy k souborovému systému ext4 jsou k dispozici v adresáři se zdrojovými kódy celého Linuxu (nejnovější verze ke stažení na https://www.kernel.org), přesněji v adresáři `/fs/ext4` a zdrojové kódy VFS v adresáři `/fs`. Najít přímo funkci, která je odpovědná za změnu dat v i-uzlu obyčejným prohledáváním a čtením zdrojových kódů, by bylo časově velmi náročné, tento proces je ale možno zjednodušit použitím nástroje `Ftrace`.

Ftrace je k dispozici od verze Linuxového jádra 2.6.27 a pro jeho používání musí být v jádře povolena jeho podpora (v sekci Kernel Hacking ->Tracers v menuconfig). Ftrace je anglická zkratka slov function tracer (trasovač funkcí), nicméně kromě trasování funkcí



Obrázek 3.1: Extent strom o třech úrovních

nabízí také mnoho dalších funkcí. Pomocí trasovače lze například pozorovat, které jednotlivé funkce Linuxu se volají po spuštění libovolné aplikace. Toto je velmi výhodné pro zjištění, které funkce se volají při odstraňování souboru. Rozhraní pro nástroj Ftrace se nachází v souborovém systému debugfs v adresáři `tracing`. Souborový systém debugfs je obvykle připojen v adresáři `/sys/kernel/debug`. Pokud připojen není, lze jej do tohoto adresáře připojit příkazem uvedeným v příkladu 3.12 [5].

Příklad 3.12: připojení debugfs

```
$ mount -t debugfs nodev /sys/kernel/debug
```

Následně je potřeba ověřit, zda je mezi dostupnými typy trasovačů trasovač pro trasování funkcí. Toto se provede vypsáním obsahu souboru `available_tracers`, ve kterém jsou mezerami odděleny jednotlivé typy trasovačů (příklad 3.13). Pro trasování funkcí a zaznamenání zásobníku volání (call stack) je potřeba trasovač `function_graph` [5].

Příklad 3.13: dostupné typy trasovačů

```
$ cat /sys/kernel/debug/tracing/available_tracers
blk function_graph wakeup_dl wakeup_rt wakeup function nop
```

Pro vybrání trasovače `function_graph` je potřeba do souboru current_tracer zapsat jeho jméno (příklad 3.14). V jeden okamžik lze vybrat pouze jeden konkrétní typ trasovače [5].

Příklad 3.14: vybrání trasovače

```
$ echo "function_graph" > current_tracer
```

Jakmile je vybrán typ trasovače, samotné trasování se zapne zapsáním hodnoty 1 do souboru tracing_on (příklad 3.15). Vypnutí trasovače naopak zapsáním hodnoty 0. Výstup trasování je pak zapsán v souboru trace [5].

Příklad 3.15: zapnutí trasování

```
$ echo "1" > tracing_on
```

Výstup trasovače (obsah souboru trace) však nyní obsahuje volání všech funkcí od všech procesů, které v době zapnutí trasovače byly aktivní. Pro prozkoumání procesu mazání souboru je vhodné, aby trasovač trasoval pouze jeden proces (program rm). K tomuto účelu slouží soubor set_ftrace_pid. Zapsáním PID (identifikátoru procesu) trasovaného procesu do souboru set_ftrace_pid bude trasovač trasovat pouze tento proces [5].

Pro zjednodušení trasování procesu byl vytvořen Bash skript trace.sh (Příklad 3.16), který provede nastavení trasovače. Tento skript předpokládá, že existuje trasovač `function_graph`, že je připojen souborový systém debugfs, a že bude skript spuštěn s právy superuživatele.

Příklad 3.16: skript trace.sh pro trasování procesu

```
$ cat trace.sh
#!/bin/bash
TRACE_PATH="/sys/kernel/debug/tracing"

# nastaveni typu trasovace
echo "function_graph" > "$TRACE_PATH/current_tracer"

# zapsani PID tohoto procesu
echo "$$" > "$TRACE_PATH/set_ftrace_pid"

# zapnuti trasovani
echo "1" > "$TRACE_PATH/tracing_on"

# parametr tohoto skriptu bude
# spusten jako trasovany program
exec $*
```

S vytvořeným skriptem už je jednodušší prozkoumat proces mazání souboru. Následující příklad 3.17 popisuje vytvoření souboru na souborovém systému ext4 a následně jeho smazání pomocí trasovaného příkazu rm. Výsledek trasování bude uložen v souboru /sys/-kernel/debug/tracing/trace, kde v hlavičce souboru je popsáno, co jednotlivé sloupečky znamenají.

Příklad 3.17: trasování mazání souboru

```
$ echo "Ahoj svete." > pokus.txt
$ sync
$ ./trace.sh rm pokus.txt
$ cat /sys/kernel/debug/tracing/trace | head -n 20
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 0)   0.043 us    |           } /* _raw_spin_lock */
 0)   0.479 us    |           flush_all_zero_pkmaps();
 0)   0.039 us    |           _raw_spin_unlock();
...
 0)               |     vfs_unlink() {
 0)               |       may_delete() {
 0)               |         inode_permission() {
 0)               |           __inode_permission() {
 0)   0.050 us    |             generic_permission();
 0)               |             security_inode_permission() {
 0)   0.049 us    |               cap_inode_permission();
 0)   0.403 us    |             }
 0)   1.141 us    |           }
 0)   1.503 us    |         }
 0)   1.953 us    |       }
...
 0)               |       ext4_unlink [ext4]() {
 0)               |         dquot_initialize() {
 0)               |           __dquot_initialize() {
 0)   0.116 us    |             dquot_active.isra.5();
 0)   0.858 us    |           }
 0)   1.285 us    |         }
...
```

Po ukončení skriptu `trace.sh`, se bude v souboru `/sys/kernel/debug/tracing/trace` nacházet zásobník volání funkcí jádra Linuxu, které byly volány v rámci sledovaného procesu (v tomto případě proces `rm`). Tento soubor je velice obsáhlý a pro rychlejší orientaci je nejvhodnější hledat funkce s prefixem `vfs_` či funkce s prefixem `ext4_`, které jak už jejich název napovídá, budou mít souvislost se souborovým systémem ext4, případně VFS. Při podrobnějším zkoumání zásobníku volání ze souboru `/sys/kernel/debug/tracing/trace`, lze objevit zajímavou funkci `vfs_unlink`. Jak už název funkce napovídá, jedná se o obecnou funkci VFS, která slouží k odstranění objektu (souboru). V této funkci je dále volána specifická funkce pro ext4 `ext4_unlink`. Ve funkci `ext4_unlink` se provádí změny přístupových a dalších časů v i-uzlu, snížení hodnoty počtu pevných odkazů o 1 a další. Změny extent stromu, které mají nejdůležitější vliv na obnovení souboru, se však v této funkci neprovádí [4].

Další funkcí, která je volána z funkce `vfs_unlink`, je pak funkce `d_delete`, kde následně přes mnoho dalších vnořených volání funkcí se dojde až k specifické funkci pro souborový systém ext4 `ext4_ext_remove_space`. Tato funkce slouží k procházení celého extent stromu a postupnému odstraňování extentů. Extent strom se prochází zprava doleva, od nejnižší úrovně stromu postupně nahoru. V každém uzlu extent stromu se extenty postupně odstraňují od posledního k prvnímu. O odstranění jednotlivých extentů se stará funkce `ext4_ext_rm_leaf`.

Kapitola 4

Návrh vylepšení možnosti rekonstrukce souboru v ext4

V této kapitole nejdříve budou popsány nástroje, kterými se lze v současnosti pokusit o obnovu smazaných souborů ze souborového systému ext4. Následně bude popsán návrh vylepšení ext4, který vede k snadnější obnově smazaných souborů.

4.1 Současné nástroje pro obnovu smazaného souboru

V současné době existující nástroje pro obnovu smazaných souborů jsou založeny na dvou hlavních technikách.

První technika se nazývá file carving. Tento způsob obnovy využívá toho, že některé typy souborů začínají (a někdy i končí) specifickou posloupností bajtů (například obrázkové soubory v JPEG formátu začínají hodnotou 0xffd8 a končí hodnotou 0xffd9). Díky tomu je možné na disku vyhledávat tyto hodnoty a následně stanovit, kde soubor na disku začíná a kde končí a následně tento interval diskového prostoru zkopírovat a vytvořit obnovený soubor. Tato metoda bude fungovat za předpokladu, že soubor je na diskové oblasti uložen spojitě, a že se v datech tohoto souboru nevyskytuje posloupnost bajtů, která svojí hodnotou odpovídá hodnotě, kterou standardně končí hledaný typ souboru. Problém této metody je, že ne všechny typy souborů začínají specifickou posloupností bajtů a ještě méně často jsou takto soubory ukončeny. Příkladem mohou být textové soubory, jejichž obsah může být jakýkoliv. Další nevýhodou této metody je, že pokud má soubor větší velikost (několik ext4 bloků), nemusí být nutně tyto bloky uloženy na disku za sebou a tudíž zkopírování paměti mezi začátkem a koncem souboru nebude fungovat. Programy, které využívají této metody pro obnovu souborů jsou například Photorec, Foremost nebo Scalpel [6].

Druhá technika využívá pro obnovu smazaných souborů žurnál. Každá změna metadat souboru je vždy nejprve zapsána do žurnálu a následně na disk. Díky uloženým informacím v žurnálu pak je možné se pokusit soubory obnovit. Nevýhodou tohoto řešení je, že ne vždy se mohou tyto smazaná metadata nacházet v žurnálu. Tuto techniku využívá program extundelete (http://extundelete.sourceforge.net/), který je momentálně zřejmě nejpoužívanější při obnově smazaných souborů ze souborového systému ext4.

4.2 Návrh vylepšení

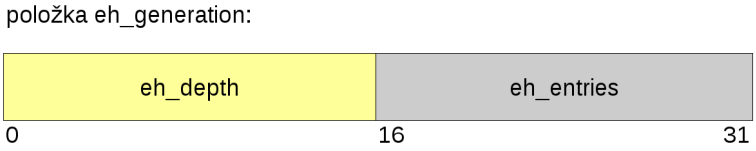
S cílem zlepšit možnosti obnovy smazaných souborů byla v této práci navržena úprava dříve zmiňovaných funkcí `ext4_ext_remove_space` a `ext4_ext_rm_leaf`. Aby byla rekonstrukce souboru proveditelná, budou se muset uchovat následující informace:

- obsah položek `eh_entries` a `eh_depth` struktury `ext4_extent_header` (Příklad 2.2) a
- obsah položek `ee_len`, `ee_start_hi` a `ee_start_lo` struktury `ext4_extent` (Příklad 2.4).

Velikost mazaného souboru (položky `i_size_lo` a `i_size_high` v i-uzlu) se uchovávat nebude z důvodu zachování stávajícího formátu ext4 a tudíž po obnově souboru bude obnovený soubor zarovnán na velikost ext4 bloku. Správné zarovnání velikosti posledního ext4 bloku se provede odstraněním posloupnosti nulových bajtů s rizikem smazání případných platných nulových bajtů z jeho konce. Odstranění nulových bajtů z konce souboru v programu pro obnovu odstraněných souborů je volitelné, a implicitně je tato funkce vypnutá.

Nulování položek `ee_len`, `ee_start_hi` a `ee_start_lo` struktury `ext4_extent` bude odstraněno.

Hodnota položky `eh_entries` struktury `ext4_header`, která je nulovaná v každém uzlu extent stromu a hodnota položky `eh_depth` struktury `ext4_header`, která je nulovaná pouze v první úrovni extent stromu (i-uzlu), budou po smazání souboru uloženy do položky `eh_generation` struktury `ext4_header`. Jak již bylo napsáno dřív, hodnota položky `eh_generation` se nevyužívá u souborového systému ext4 (pouze u souborového systému Lustre), ovšem vzhledem k tomu že, se do této položky budou ukládat data až po smazání celého uzlu extent stromu, nebudou tímto poškozena žádná platná data. Dolních 16 bitů položky `eh_generation` bude využito pro uložení hodnoty, kterou měla položka `eh_depth` před smazáním souboru. Položka `eh_depth` se ukládá pouze v první úrovni extent Stromu (i-uzlu), jelikož v dalších úrovních tato položka není nulována. Do horních 16 bitů položky `eh_generation` bude vždy uložena hodnota, kterou měla položka `eh_entries` před smazáním souboru. Hodnota položky `eh_entries` je nulována po smazání souboru v každé úrovni extent stromu, proto bude její hodnota, kterou měla před smazáním souboru, také ukládána v každé úrovni extent stromu do položky `eh_generation`. Po smazání souboru pak bude položka `eh_generation` naplněna tak, jak je to znázorněno na obrázku 4.2.



Obrázek 4.1: Data v položce eh_generation po smazání souboru

Kapitola 5

Implementace

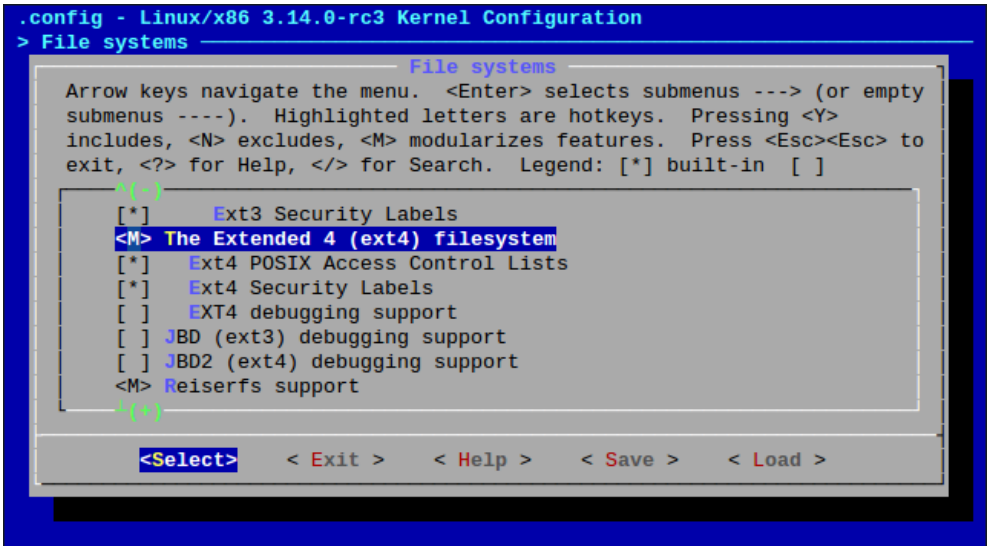
Navržené vylepšení jádra Linux bylo implementováno v programovacím jazyce C pro verzi jádra Linuxu 3.14.0-rc3 (staženo z git repositáře `git://git.kernel.org/pub/\-scm/linux/kernel/git/torvalds/linux.git`), která byla nejnovější verzí jádra v době začátku implementace. Program pro prezentaci výhod provedených změn byl rovněž naprogramován v jazyce C s využitím knihovny ext2fs, která se standardně používá při práci se souborovým systémem ext4.

5.1 Úprava jádra

Před úpravou a testováním provedených změn v jádře Linux si bylo důležité uvědomit, že pokud budou provedeny změny v ovladačích souborového systému ext4, které povedou k nefunkčnosti a pádu jádra Linuxu, nebude možné při příštím startu počítače načíst jádro Linux, pokud bude diskový oddíl, ve kterém se nachází kořenový adresář, ve formátu ext4. Proto byl pro diskový oddíl s kořenovým adresářem zvolen souborový systém Btrfs, na který nebudou mít změny souborového systému ext4 žádný vliv.

Další výhodou při následné kompilaci je nastavení kompilace ext4 jako modulu jádra. Tím bude zajištěno, že pokud budou provedeny změny pouze v rámci zdrojových kódů modulu ext4, nebude potřeba překompilovávat celé jádro Linuxu, ale pouze jaderný modul ext4. Toto nastavení se dá provést například v nástroji `make menuconfig` (viz Příklad 5.1), kde se pomocí mezerníku dá nastavit, zda bude souborový systém ext4 součástí jádra nebo zda bude zkompilován jako jaderný modul (v nástroji `make menuconfig` označeno písmenem M).

Samotná úprava zdrojového kódu je provedena v souborech `/fs/ext4/ext4_extents.h` a `/fs/ext4/extents.c`. Soubor `/fs/ext4/ext4_extents.h` byl doplněn o makra `ext4_ext_store_entries` a `ext4_ext_store_depth` (viz Příklad 5.2). Makro `ext4_ext_store_entries` se stará o uložení hodnoty položky `eh_entries` do horních 16 bitů položky `eh_generation`. Položka `eh_generation` je uchována v bajtovém pořadí Little-endian, tudíž veškeré operace s touto položkou by měly byt prováděny s hodnotami, které jsou rovněž uloženy v bajtovém pořadí Little-endian. Pro převod na bajtové pořadí Little-endian bylo v kódu použito makro `cpu_to_le32`. Uložení hodnoty do položky `eh_generation` je implementováno pomocí bitových operací and a or. V první fázi je potřeba horních 16 bitů položky `eh_generation` vynulovat tak, že se provede logické vynásobení horních 16 bitů s hodnotou 0. Dolních 16 bitů položky `eh_generation` je potřeba zachovat, proto se musí provést logicky součin s hodnotou, která bude mít všechny bity nastavené v logické



Obrázek 5.1: Ukázka nástroje menuconfig

1. Jelikož velikost položky `eh_generation` je 32 bitů, bude ji potřeba logicky vynásobit pouze jedním číslem (0x0000ffff) o stejné velikosti, kde polovina bitů bude mít hodnotu logická 0 a druhá polovina logická 1. Po vynulování horních 16 bitů položky `eh_generation` se pomocí logického součtu s hodnotou proměnné `eh_entries` posunutou o 16 bitů doleva, převedenou na bitové pořadí Little-endian, nastaví horních 16 bitů položky `eh_generation`. Analogicky se provede nastavení dolních 16 bitů položky `eh_generation` hodnotou položky `eh_depth` v makru `ext4_ext_store_depth`.

```
+static inline void ext4_ext_store_entries(struct ext4_extent_header *eh,
+                                           __u16 eh_entries){
+       eh->eh_generation &= cpu_to_le32((unsigned long) 0x0000ffff);
+       eh->eh_generation |= cpu_to_le32(((unsigned long) (eh_entries) << 16)
+                                & 0xffff0000);
+}
+
+static inline void ext4_ext_store_depth(struct ext4_extent_header *eh,
+                                          __u16 eh_depth){
+       eh->eh_generation &= cpu_to_le32((unsigned long) 0xffff0000);
+       eh->eh_generation |= cpu_to_le32(((unsigned long) (eh_depth))
+                                & 0x0000ffff);
+}
```

Obrázek 5.2: Přidaná makra

V souboru `/fs/ext4/extents.c` byly provedeny změny ve funkcích `ext4_ext_rm_leaf` a `ext4_ext_remove_space`. Funkce `ext4_ext_rm_leaf` slouží k odstranění jednotlivých extentů z extent stromu. Jak již bylo popsáno v návrhu vylepšení, bude potřeba v každém extentu po jeho odstranění uchovat hodnoty položek `ee_len`, `ee_start_hi` a `ee_start_lo` a také v každé struktuře `eh_header` zachovat hodnotu položky `eh_entries` jejím uložením do položky `eh_generation` pomocí makra `ext4_ext_store_entries`. Položka `eh_depth` se

nuluje pouze v kořenovém uzlu extent stromu (i-uzlu). Hodnotu položky `eh_depth` bude potřeba uložit do položky `eh_generation`. K tomu je použito makro `ext4_ext_store_depth` uvedené na obrázku 5.2. Toto ukládání hodnoty položky `eh_depth` bude provedeno ve funkci `ext4_ext_remove_space`.

Úpravy ve funkci `ext4_ext_rm_leaf` jsou následující:

- Vytvoření nové proměnné `ex_ee_entries` typu `unsigned short` na začátku funkce a následné uložení hodnoty položky `eh_entries` z toho důvodu, že v průběhu funkce se tato hodnota položky dekrementuje s tím, jak se postupně odstraňují extenty.
- Odstranění části kódu, kde se volá funkce `ext4_ext_store_pblock`, která vede k nulování položek `ee_start_lo` a `ee_start_hi`.
- Přidání podmínky, kdy velikost extentu se zapíše do položky `ee_len` pouze v případě, kdy je velikost extentu nenulová. Přidání této podmínky je zde z toho důvodu, že v této funkci nemusí vždy docházet pouze k odstranění celého extentu, ale může zde docházet k jeho zkrácení. V původní verzi zdrojových kódů jádra Linux se velikost extentu zapisovala do položky `ee_len` vždy. Tudíž, když se velikost extentu změnila na velikost 0 (extent je smazán), vynulovala se hodnota položky `ee_len` a následně by při obnově souboru nebylo možné určit, jak byl původní extent dlouhý.
- Na konci funkce se nachází podmíněný příkaz, který je proveden v případě, že mazaný uzel extent stromu se nenachází v i-uzlu a z aktuálního uzlu extent stromu byly odstraněny všechny extenty. Tento podmíněný příkaz byl rozšířen o kód, který provede uložení hodnoty proměnné `ex_ee_entries` obsahující původní hodnotu položky `eh_entries` pomocí makra `ext4_ext_store_entries` do položky `eh_generation`. Jakmile je hodnota uložena v položce `eh_generation`, zavoláním funkce `ext4_ext_dirty` se provede označení daného ext4 bloku za neplatný, což povede k jeho pozdějšímu zapsání na disk.

Funkce `ext4_ext_remove_space` slouží k postupnému průchodu extent stromu a odstranění jednotlivých uzlů extent stromu. Hlavní úpravy v této funkci jsou následující:

- Vytvoření nové proměnné `ex_ee_entries` typu ukazatel na typ `unsigned short` na začátku funkce.
- Alokace vynulované paměti pomocí funkce `kzalloc` v případě, že počet odkazů na soubor je nulový (soubor bude smazán). Velikost alokované paměti je rovna počtu úrovní extent stromu vynásobeného velikostí datového typu `unsigned short`. Tím je získáno pole, které slouží k uchování hodnot položek `eh_entries` v jednotlivých úrovních extent stromu (indexem je označena úroveň stromu, ke které se vztahuje položka `eh_entries`). Následně se po odstranění celého uzlu obsahujícího extenty nebo indexační uzly vezme uložená hodnota z pole `ex_ee_entries` a uloží se pomocí makra `ext4_ext_store_entries` do položky `eh_generation` ve struktuře `extent_header`. Důvodem, proč se využívá toto pole je, že hodnota položky `eh_entries` je měněna funkcemi, které odstraňují extenty nebo indexační uzly. Po odstranění všech extentů z ext4 bloku případně i-uzlu, není nikde uložena původní hodnota položky `eh_entries`, proto se před mazáním extentů či indexačních uzlů ukládá do tohoto pole. Dalším důvodem, proč se ukládají data do položky `eh_generation` až po smazání uzlu extent stromu, je, že po smazání uzlu extent stromu mohou být v položce

eh_generation uložena jakákoliv data. Následně po úspěšné alokaci pole se uloží do prvního prvku hodnota položky eh_entries.

- V podmínce, která platí v případě, že úroveň (hodnota proměnné i) extent stromu nebyla zpracovávána, byla přidána podmínka, která platí v případě, když je proměnná entries_path různá od hodnoty NULL. Blok kódu po splnění podmínky obsahuje inicializaci prvku pole entries_path na indexu daném proměnnou i, položkou eh_entries.

- V části kódu, kde se zpracovávají příkazy v případě, že byly v aktuálním uzlu odstraněny všechny extenty případně indexační uzly, byla přidána podmínka a do ní kód, který provede uložení dříve uchované hodnoty v poli entries_path do položky eh_generation za pomocí makra ext4_ext_store_entries. Nakonec se zavoláním funkce ext4_ext_dirty provede označení ext4 bloku za neplatný, což povede k jeho pozdějšímu zapsání na disk.

- Uložení položek eh_entries a eh_depth do položky eh_generation pomocí maker ext4_ext_store_entries a ext4_ext_store_depth v i-uzlu. Ukládání opět proběhne za podmínky, že je hodnota proměnné entries_path různá od NULL.

- Uvolnění dříve alokované paměti pomocí funkce kfree.

Překlad jádra s úpravami

Během provádění změn v jádře Linuxu a jeho testování bylo potřeba jaderný modul několikrát zkompilovat a modul zavést do jádra Linuxu. Standardně je adresář, ve kterém se nachází adresáře s různými verzemi zdrojových kódů Linuxu umístěn v /usr/src. Po stažení adresáře se zdrojovými kódy Linuxu do adresáře /usr/src je potřeba vytvořit symbolický odkaz linux, který bude odkazovat na adresář zdrojovými kódy Linuxu, které se budou kompilovat. Po vytvoření odkazu a změně aktuálního adresáře do adresáře se zdrojovými kódy je vhodné si před první kompilací zkopírovat konfigurační soubory jádra, které je právě zavedené. Samozřejmě si lze tento konfigurační soubor nakonfigurovat manuálně, ale jeho zkopírováním se ušetří mnoho času. Konfigurační soubory dostupných jader se obvykle nachází v adresáři /boot a jejich jméno začíná slovem config. Konfigurační soubor v adresáři se zdrojovými kódy Linuxu má vždy jméno .config. Po překopírování starého konfiguračního souboru do souboru .config se příkazem make oldconfig spustí operace, která prohledá všechny volby v souboru .config a zeptá se uživatele na nastavení nových voleb, které se ve starém konfiguračním souboru nenacházely. Po nastavení nových voleb je možno spustit příkaz make menuconfig, kterým se nastaví zkompilování souborového systému ext4 jako jaderného modulu a případně změnilo další nastavení. Následně příkaz make provede kompilaci jádra Linux. Po zkompilování jádra je potřeba ještě zkompilovat jaderné moduly pomocí příkazu make modules. Zkompilované jádro a jaderné moduly se instalují příkazy make install a make modules_install. V našem případě příkaz make install nastaví cesty k novému jádru do zavaděče a následně stačí restartovat počítač a zavést nové jádro Linuxu [3] [7].

Po úpravách zdrojových kódů ext4 stačí spustit příkaz make modules, který upravené moduly zkompiluje a následně stačí zkompilovaný jaderný modul, který má příponu .ko, zkopírovat do adresáře aktuálního jádra s moduly (Příklad 5.1). Před samotnou kompilací jaderného modulu je potřeba tento modul pomocí příkazu rmmod odebrat a následně po úspěšné kompilaci příkazem modprobe nový jaderný modul zavést [11].

Příklad 5.1: kompilace a instalace upraveného modulu

```
$ rmmod ext4
$ cd /usr/src/linux
$ make modules
scripts/kconfig/conf --silentoldconfig Kconfig
make[1]: Nothing to be done for 'all'.
make[1]: Nothing to be done for 'relocs'.
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
  CALL    scripts/checksyscalls.sh
  CC [M]  fs/ext4/extents.o
  LD [M]  fs/ext4/ext4.o
  Building modules, stage 2.
  MODPOST 3958 modules
  CC      fs/ext4/ext4.mod.o
  LD [M]  fs/ext4/ext4.ko
$ cp fs/ext4/ext4.ko /lib/modules/3.14.0-rc3+/kernel/
fs/ext4/ext4.ko
$ modprobe ext4
```

Vytvoření PATCH souboru

Jakmile jsou všechny úpravy jádra hotové a otestované, lze pomocí nástroje git provést commit změn a následně vytvořit soubor PATCH, který bude obsahovat změny oproti původním zdrojovým kódům jádra Linuxu (Příklad 5.2). Po vytvoření PATCH souboru je nutné jej náležitě upravit, aby odpovídal formátu, který je popsaný v dokumentaci, která je umístěna v adresáři `Documentation` v kořenovém adresáři zdrojových souborů jádra Linux. Zkráceně lze říct, že první řádky by měly výstižně popisovat změny, které PATCH soubor obsahuje. Tento popis musí následovat na novém řádku podpis autora PATCHe, který má formát: `Signed-off-by: Jmeno autora <email autora>`. Za tímto podpisem následují tři pomlčky na samostatném řádku, za kterými je na dalším řádku umístěn obsah PATCH souboru. Po vytvoření a úpravě tohoto souboru lze soubor zkontrolovat pomocí skriptu napsaném v jazyce Perl, který je umístěn v adresáři `scripts` v kořenovém adresáři zdrojových kódů jádra Linux (Příklad 5.3).

Příklad 5.2: vytvoření PATCH souboru

```
$ git format-patch HEAD~
0001-Undelete-feature-for-ext4.patch
```

Příklad 5.3: kontrola vytvořeného PATCH souboru

```
$ cd /usr/src/linux
$ ./scripts/checkpatch.pl 0001-Undelete-feature-
for-ext4.patch
total: 0 errors, 0 warnings, 152 lines checked

0001-Undelete-feature-for-ext4.patch has no
obvious style problems and is ready for submission.
```

Jestliže kontrola vytvořeného PATCH souboru proběhla úspěšně, může se tento PATCH soubor vložit do emailu a odeslat pro konzultaci na mailing list

linux-ext4@vger.kernel.org. Email by měl v předmětu zprávy obsahovat v hranatých závorkách slovo PATCH následované subsystémem, kterého se PATCH týká (v tomto případě ext4). Za názvem subsystému je uvedena dvojtečka a následně výstižná fráze popisující typ změn. Obsah emailu obsahuje vložený PATCH soubor spolu s podpisem a popisem PATCH souboru jako prostý neformátovaný text.

5.2 Program pro obnovu smazaných souborů

Program pro obnovu smazaných souborů je napsán v jazyce C s využitím knihovny ext2fs, která zajišťuje veškeré operace specifické pro souborový systém ext4. Nápovědu, jak program spouštět, lze zobrazit spuštěním programu s parametrem `-h`.

Obnova souboru funguje na principu procházení extent stromu zleva doprava postupně od prvního extentu k poslednímu. K rekonstrukci souboru jsou použita uložená metadata v položce `eh_generation`.

Program obnovuje smazaný soubor zarovnaný na velikost ext4 bloku, kde k zarovnání jsou, jak už bylo napsáno, použity nulové bajty. V případě, že mají být nulové bajty z konce souboru odstraněny, spustí se program s parametrem `s`.

Celkovou ukázku, že obnovení programu funguje a soubor je obnoven s nepoškozenými daty, lze vidět na příkladu 5.4. Obsah smazaného souboru a souboru po obnovení je totožný. Obnovení souboru pomocí jeho čísla i-uzlu je spolehlivější, nicméně program obsahuje i možnost obnovy souboru podle jeho jména v době smazání za předpokladu, že adresář, ve kterém se soubor nacházel, stále existuje (Příklad 5.5).

Příklad 5.4: ukázka obnovení souboru

```
$ mount /dev/vol_group1/log_vol1 /mnt/data/
$ cd /mnt/data
$ echo "Ahoj svete. Tento soubor bude smazan" > pokus.txt
$ sync
$ ls -i pokus.txt
12 pokus.txt
$ cat pokus.txt
Ahoj svete. Tento soubor bude smazan
$ md5sum pokus.txt
a56de8072f801c4df68aa5ef81ef99eb  pokus.txt
$ rm pokus.txt
$ umount /mnt/data/
$ ./ext4-undelete /dev/vol_group1/log_vol1 -s
-i 12 -o obnova_pokus.txt
$ cat obnova_pokus.txt
Ahoj svete. Tento soubor bude smazan
$ md5sum obnova_pokus.txt
a56de8072f801c4df68aa5ef81ef99eb  obnova_pokus.txt
```

Příklad 5.5: obnovení souboru podle jména

```
$ ./ext4-undelete /dev/vol_group1/log_vol1 -n /pokus.txt
-s -o obnova_pokus2.txt
$ cat obnova_pokus2.txt
Ahoj svete. Tento soubor bude smazan
$ md5sum obnova_pokus2.txt
a56de8072f801c4df68aa5ef81ef99eb  obnova_pokus2.txt
```

Aby byla šance na obnovení smazaného souboru co největší, je důležité co nejrychleji po neúmyslném smazání souboru souborový oddíl, na kterém se nacházel tento soubor, odpojit, případně ho připojit s parametrem pouze pro čtení. Čím déle zůstává diskový oddíl se smazaným souborem připojen v režimu pro zápis, tím více stoupá pravděpodobnost, že budou data nebo metadata smazaného souboru na disku přepsána.

Kapitola 6

Testování

Po implementaci změn v jádře systému Linux byla funkce souborového systém ext4 otestována dvěma typy testů. Prvním typem byly regresní testy XFStests, které se standardně používají pro testování funkčnosti po změnách souborového systému ext4. Druhým typem testování byly power failure testy (testy v případě pádu systému v důsledku výpadku napájení), které byly z části nově vytvořeny a z části byly převzaty od vývojáře ext4 Mike Snitzera.

6.1 XFStests

Projekt XFStests zahrnuje testovací nástroj a sadu regresních testů pro různé souborové systémy, který byl primárně vytvořen pro testování souborového systému XFS. Později byl tento nástroj zdokonalen, aby podporoval více souborových systémů, ale název již zůstal nezměněn.

Na adrese http://xfs.org/index.php/Getting_the_latest_source_code lze najít aktuální URL adresu git repositáře pro XFStests. Pro kompilaci XFStests je potřeba z git repositáře stáhnout aktuální verzi XFStests, XFSprogs, fio a dalších nástrojů, na kterých XFStests závisí. Po úspěšné kompilaci XFStests je potřeba správně vytvořit konfigurační soubor v podadresáři `configs`, v kořenovém adresáři XFStests (adresář, kde byly staženy zdrojové kódy XFStests). Jméno konfiguračního souboru musí být shodné s názvem počítače (hostname), na kterém testy budou probíhat, až na doplněnou příponu config. V našem případě předpokládejme, že konfigurační soubor se jmenuje `lubos-VirtualBox.config`. Výčet a popis všech proměnných, které se dají v konfiguračním souboru nastavit, jsou uvedeny v souboru README v kořenovém adresáři projektu. Pro spuštění testů jsou potřeba 2 diskové oddíly. Jeden oddíl určený pro testování ext4 (v našem případě zařízení `/dev/sda2`) a jeden oddíl, na který budou během testování ukládána různá data (v našem případě zařízení `/dev/sda3`). Pro oba oddíly bude rovněž potřeba vytvořit adresáře, kam budou tyto oddíly připojeny. Jak bude celý konfigurační soubor určený pro testování ext4 vypadat, popisuje příklad 6.1.

Jakmile je konfigurační soubor nakonfigurován, lze spustit testování z kořenového adresáře XFStests příkazem `./check`.

Po ukončení skriptu `check` budou na standardní výstup vypsány výsledky testování. Upravené Linuxové jádro bylo otestováno a výsledky testů se nelišily od výsledků, které byly provedeny před úpravou zdrojových kódů.

Příklad 6.1: konfigurační soubor

```
$ cat lubos-VirtualBox.config
FSTYP=ext4

TEST_DEV=/dev/sda2
TEST_DIR=/mnt/data

SCRATCH_DEV=/dev/sda3
SCRATCH_MNT=/mnt/scratch
```

6.2 Power failure testy

Při úpravách funkcí jádra Linuxu, které pracují se souborovým systémem ext4, může nedopatřením vzniknout chyba, kterou nelze odhalit standardními regresními testy. Power failure testy testují případy, kdy dojde k neočekávanému výpadku systému během provádění vstupně/výstupních operací na zařízení. V této situaci je většinou souborový systém v nekonzistentním stavu, ale právě díky žurnálu, který souborový systém ext4 obsahuje, je možné ho do konzistentního stavu zpátky dostat. V této práci se jedná o případ, kdy v průběhu mazání souboru nastane pád systému. Kvůli navrženému vylepšení, které navrhuje ponechání původních hodnot v položkách `ee_len`, `ee_start_hi` a `ee_start_lo` ve struktuře `ext4_extent` (Příklad 2.4) po smazání extentu, vzniklo na mailing listu `linux-ext4@vger.kernel.org` podezření, že při pádu systému uprostřed procesu mazání souboru v situaci, kdy nebude dostatek paměti na rozšíření transakce během mazání jednotlivých extentů, se může stát, že po přehrání žurnálu budou dealokovány bloky, které byly v průběhu mazání souboru už přiděleny jiným extentům. Z tohoto důvodu je potřeba provedení power failure testů, které by měly odhalit případné chyby.

Princip power failure testů spočívá ve vytvoření snímku (snapshotu) aktuálního stavu dat na disku během provádění vstupně výstupních operací za pomocí nástroje `dmsetup`. Skript, který vytvoří snímek z LVM oddílu, byl převzat od vývojáře jádra Linuxu Mike Snitzera. Celé testování pak probíhalo tak, že se vytvářelo paralelně mnoho velmi fragmentovaných souborů, které se průběžně mazaly a během těchto operací se vytvořil snímek disku, který zachytil aktuální stav dat na tomto disku. Následně se pomocí programu `fsck` zkontroloval stav snímku a spustil se proces opravení souborového systému za pomocí přehrání žurnálu. Celý tento proces vytváření a mazání souborů spolu s pořizováním snímku souborového systému a jeho testováním probíhal opakovaně ve stovkách iterací. Během tohoto testování nebyly nalezeny žádné chyby, které by přehráním žurnálu nebylo možné opravit.

Kapitola 7

Závěr

Cílem práce bylo optimalizovat mazání souborů v ext4 pro jejich snadnější obnovu. Součástí práce bylo také vytvoření nástroje, který bude prezentovat výhody provedených změn. Nástroj pro obnovu souborů dokáže obnovit smazaný soubor, dokud metadata anebo data smazaného souboru nebyla přepsána. Tuto obnovu dokáže provést na základě čísla i-uzlu smazaného souboru, případně pomocí jména souboru. Provedené změny v jádře Linuxu byly zaslány pro diskuzi na mailing list souborového systému ext4, kde se diskutují provedené změny a jedná se o jejich zařazení do hlavní větve jádra Linuxu.

V budoucnu by bylo možné vylepšit zde prezentované výsledky např. implementací příkazu na obnovení souboru v rámci nástroje debugfs, který je součástí balíčku nástrojů e2fsprogs. Debugfs používá knihovnu ext2fs, která se rovněž používá v nástroji pro obnovu smazaných souborů napsaného pro tuto práci. Implementace v rámci nástroje debugfs by tedy už měla být jednoduchá. Dále by bylo možno vylepšit vytvořený nástroj pro obnovu smazaného souboru tak, aby zvládal rekurzivní obnovu smazaných souborů a adresářů. Nyní nástroj dokáže obnovit smazaný soubor podle jeho jména pouze v případě, že existuje adresář, ze kterého byl tento soubor smazán.

Literatura

[1] Anatomy of the Linux virtual file system switch. 2009-08-31, [online][cit. 2014-04-08]. URL <http://www.ibm.com/developerworks/library/l-virtual-filesystem-switch/>

[2] Ext4 Disk Layout. 2011-03-20, [online][cit. 2014-04-07]. URL <https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout>

[3] Linux Kernel Newbies. 2013-04-27, [online][cit. 2014-05-05]. URL <http://kernelnewbies.org/KernelBuild>

[4] vfs_unlink. 2014, [online][cit. 2014-05-05]. URL <https://www.kernel.org/doc/htmldocs/filesystems/API-vfs-unlink.html>

[5] Ftrace. 2014-01-27, [online][cit. 2014-05-05]. URL <http://elinux.org/Ftrace>

[6] Carrier, B.: *File system forensic analysis*. Upper Saddle River: Addison-Wesley, 2005, iSBN 0-32-126817-2.

[7] Jelínek, L.: *Jádro systému Linux*. Brno: Computer Press, vyd. 1. vydání, 2008, iSBN 978-80-251-2084-2.

[8] Love, R.: *Linux kernel development*. Addison-Wesley, třetí vydání, 2010, iSBN 978-0-672-32946-3.

[9] Mathur, A.; Cao, M.; Bhattacharya, S.; aj.: The new ext4 Filesystem: current status and future plans. http://www.cs.sunysb.edu/~prade/Teaching/Spring13/prez/L18/ols2007v2-pages-21-34.pdf, 2007, [online][cit. 2014-05-15].

[10] Pate, S. D.: *Unix filesystems*. Indianapolis: J. Wiley, c2003, iSBN 04-711-6483-6.

[11] Salzman, P. J.; Burian, M.; Pomerantz, O.: The Linux Kernel Module Programming Guide. http://www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf, 2007-05-18, [online][cit. 2014-05-05].

Příloha A

Obsah DVD

Na přiloženém DVD je adresářová struktura následující:

- Adresář `/BP` obsahuje tuto písemnou zprávu ve formátu PDF včetně zdrojového tvaru této zprávy
- Adresář `/KVM` obsahuje obraz virtuálního stroje ve formátu QCOW2, který je kompatibilní s KVM.
- Adresář `/VirtualBox` obsahuje obraz virtuálního stroje ve formátu OVF 2.0, který lze naimportovat do Oracle VM VirtualBox.
- Adresář `/ext4-undelete` obsahuje zdrojové kódy programu pro obnovu smazaných souborů.
- Adresář `/fragmented` obsahuje zdrojové kódy k programu, který vytváří velmi fragmentované soubory.
- Adresář `/kernel-patch` obsahuje PATCH soubor vytvořený v rámci této práce.
- Adresář `/skripty` obsahuje použité Bash skripty, které byly použity během experimentů v této práci.