



Ext4: Nové triky starého psa

Jan Kára <jack@suse.cz>
SUSE Labs



Obsah

Trocha historie a základní struktura filesystému

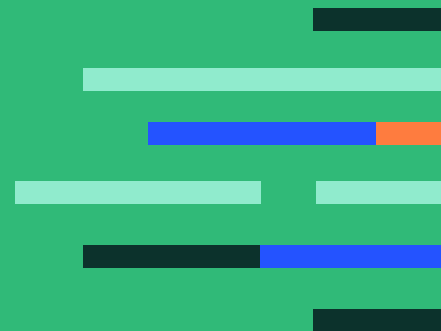
Zajímavé funkce:

- Velké alokační bloky
- Velké filesystémové bloky
- Podpora atomických zápisů
- Mód bez žurnálu
- Zrychlení fsync(2) pomocí “fastcommit” transakcí
- Podpora velkých rozšířených atributů (xattr)

Co bude dál?



Trocha historie





Trocha historie (ext2 → ext3 → ext4)

Ext2 vytvořen v roce 1993

- Inspirace filesystemem Minix z OS Minix a FFS z BSD
- Dobře navržená rozšiřitelnost diskového formátu - sady kompatibilních, nekompatibilních a read-only kompatibilních vlastností

Ext3 (2001)

- Rostoucí velikost filesystemů a tím čas fsck
- Ext2 filesystem s žurnálem (jako kompatibilní vlastností)

Ext4 (2006)

- Nový formát inodů (b-strom s rozsahy bloků (extenty))

Základní struktura

Filesystem rozdělen na alokační grupy shodné velikosti

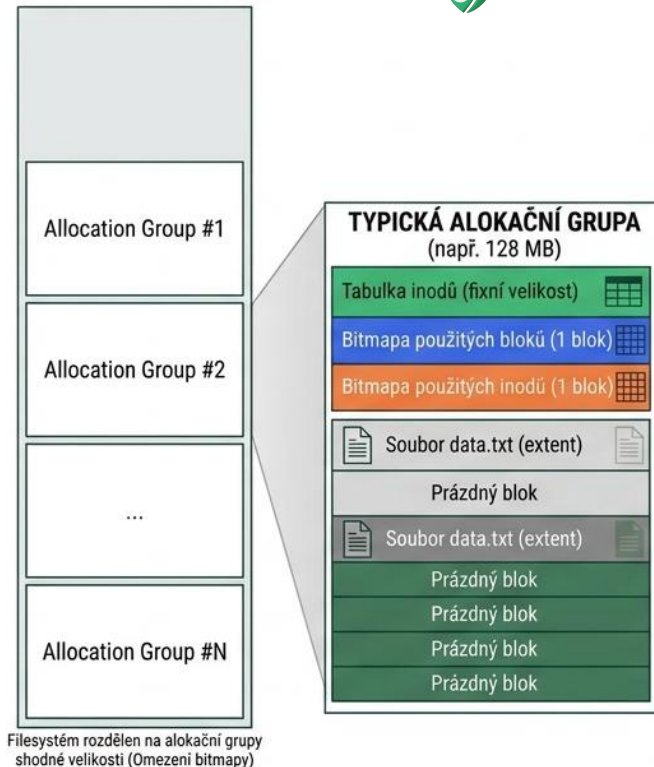
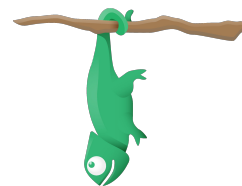
Každá grupa má:

- Tabulku inodů pevné velikosti
- Bitmapu použitých bloků (1 blok)
- Bitmapu použitých inodů (1 blok)

Tabulka inodů a bitmapy nejsou nutně uvnitř grupy

Velikost blokové bitmapy omezuje velikost grupy - typicky 128 MB při velikosti bloku 4 KB.

VÁŠ FILESYSTEM EXT4
(celková kapacita, např. 1 TB)



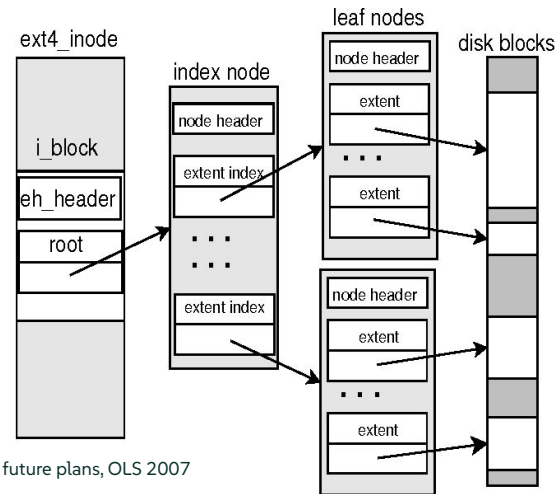
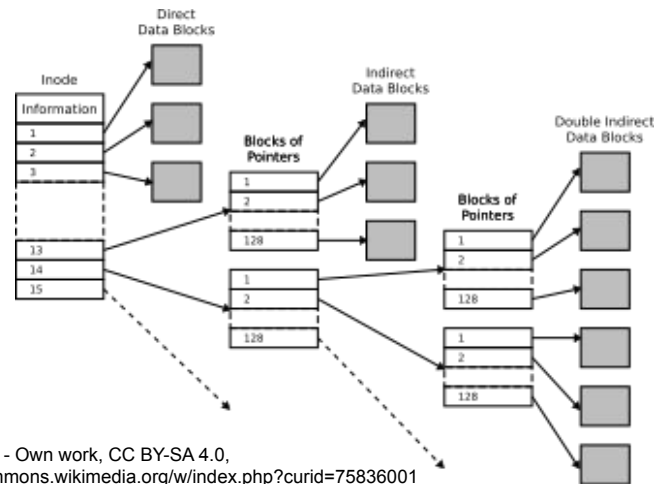
Inode

Popisuje nějaký filesystemový objekt - soubor, adresář, symlink, ...

Velikost nastavitelná, typicky 256 bytů.

Obsahuje velikost, práva, časy přístupu a modifikace, odkazy na datové bloky

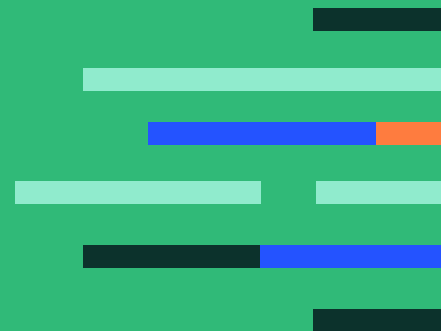
Ve zbylém místě rezervovaném pro inode se ukládají rozšířené atributy



Avantika Mathur et al.: The new ext4 filesystem: current status and future plans, OLS 2007



Zajímavé funkce



Velké alokační bloky (clustery)



Vlastnost filesystemu se jmenuje “bigalloc”

Implementováno v roce 2011

Místo na filesystemu se alokuje po kusech (clusterech) s velikostí několika (2^k) bloků

IO stále probíhá po jednotlivých blocích!

Menší počet grup (jsou větší), menší režie alokací, menší fragmentace

Více vyplývaného místa pro malé soubory, sparse soubory, metadata



Velké filesystémové bloky



Filesystém vždy podporoval velikost bloku až 64 KB

Dlouho nepodporovaný přístup k filesystému s velikostí bloku > velikost stránky

- 4 KB na x86

Nyní (od 2025) podporována libovolná velikost bloku bez ohledu na velikost stránky

Oproti velkým alokačním clusterům:

- Neplýtvá místem pro metadata
- Malé soubory zabírají více místa v page cache, větší režie malých čtení a zápisů



Podpora atomických zápisů (RWF_ATOMIC)



Moderní disky (NVME flash), cloud storage, podporují atomický zápis (vůči výpadku napájení) delší než 1 sektor

- 4 KB ale i třeba 1 MB

Umožňuje změny datových struktur bezpečné vůči výpadku napájení s menší režii

- Typicky pro databáze - třeba o 30% vyšší výkon MariaDB při zápisu

Podpora pro direct IO do velikosti jednoho clusteru (2025)



Žurnálování

Ext4 normálně používá žurnál, aby se zajistila konzistence filesystému i při nekorektním odpojení (výpadek napájení, pád systému)



Každá logická operace je složená z řady dílčích operací a chceme, aby se staly všechny nebo žádná

- Zápis do souboru - označení bloků jako použité v bitmapách, přidání bloků do b-stromu v inodu, změna velikosti souboru

Změnám pro jednu (případně několik) logických operací se říká *transakce*



Žurnálování (pokračování)



Ext4 dělá žurnálování filesystémových bloků - transakce v žurnálu obsahuje seznam změněných metadata bloků a jejich nový obsah

Na konci transakce se zapisuje “commit” blok - transakce je úplná

Po zápisu transakce do žurnálu se bloky líně přepisují na správné pozice ve filesystému (checkpoint).

Když jsou všechny bloky transakce přepsané, místo v žurnálu se může uvolnit

Při připojování filesystému se projde celý žurnál od nejstarší transakce po nejnovější (s platným commit blokem) a všechny bloky se přepíší na správné pozice.





Mód bez žurnálu

Žurnál má nezanedbatelný dopad na výkon pro operace intenzivně měnící metadata

- Třeba operace s adresáři, rozšířenými atributy, často volající fsync(2)
- Každý metadatový blok se zapisuje 2x (žurnál + místo ve filesystemu)

Konzistence není vždy nutná

- Fsck může být přijatelné
- Filesystem se prostě zahodí a vytvoří nový (cache)

Ext4 podporuje mód bez žurnálu (je to vlastnost filesystemu jako každá jiná)

Zrychlení fsync(2)

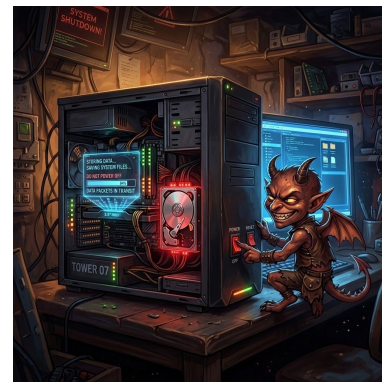


Cílem fsync(2) je, aby data v daném souboru byla přístupná i po případném následném výpadku

Nutné zapsat všechna data pro soubor, zapsat do žurnálu transakci s novým obsahem metadatových bloků

V ext4 je vždy aktivní jen jedna transakce, která sbírá všechny logické změny

fsync(2) tedy čeká na zápis i nesouvisejících metadat (a než doběhou právě běžící logické operace).



Zrychlení fsync(2) (pokračování)



Nová vlastnost žurnálu (fastcommit) implementována v roce 2020

Při volání fsync se nezapisuje standardní transakce, ale fastcommit transakce, která obsahuje seznam proběhlých logických operací od poslední normální transakce.

Třeba: Vytvoř soubor X v adresáři Y, přidej bloky P-Q do souboru Z na pozici R, ...

Výrazně snižuje množství metadat zapsaných v průběhu fsync(2)

Poskytuje třeba 30% zrychlení pro zatížení s častým voláním fsync(2)

- NFS server, mailový server, databázový server



Podpora velkých rozšířených atributů

Uživatel může u souboru ukládat rozšířené atributy (xattr)

- ACL
- SELinux kontext
- Cokoliv dalšího (viz `setfattr(1)`)

Standardně omezené na trochu místa v inode + jeden filesystémový blok (jména všech atributů + obsah)

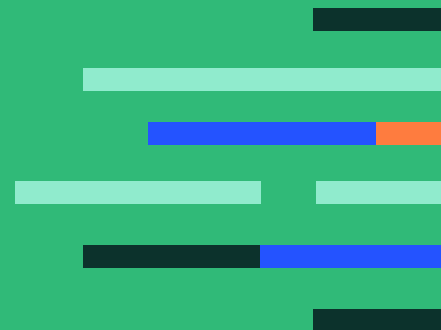
S vlastností `ea_inode` (naimplementováno 2017) se pro rozšířený atribut může naalokovat skrytý inode a data uložit v něm jako v souboru

Velikost atributu omezená na 64 KB (ve VFS)





A co dál?



Přechod na iomap infrastrukturu

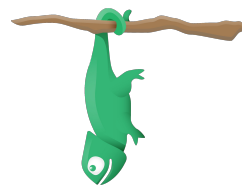


Přechod implementace na VFS infrastrukturu iomap

V ideálním případě jediný pozorovatelný efekt pro uživatele bude snížení nároků na CPU pro některé druhy zátěže

Některé operace už jsou převedené (direct IO, ...), nejvíce práce se standardním buferovým IO.

Podpora buferovaných atomických zápisů



Atomické zápisy momentálně fungují jen pro direct IO

Chystá se podpora pro atomické buferované zápisy

- Pravděpodobně se zápis rovnou pošle na disk (write-through), ale zároveň zůstane v cache
- Použití hlavně v PostgreSQL



Nová alokace bloků

Běžně alokace nových bloků při zápisu:

1. Označí bloky jako zabrané
2. Přidá bloky do inodu na správnou pozici a označí je jako dosud nezapsané
3. Zapiše data na disk
4. Označí bloky v inodu jako zapsané

Tento způsob vyžaduje dvě modifikace b-stromu extentů v inode.





Nová alokace bloků (pokračování)

Efektivnější způsob s jednou modifikací b-stromu:

1. Označit bloky jako zabrané **jen v paměti**
2. Zapsat do bloků data
3. Přidat bloky do inodu a označit je jako zabrané v bitmapě na disku



Shrnutí



Ačkoliv je filesystem starý přes 30 let, stále se rozvíjí a je široce používán

V průběhu let přidáno množství nových funkcí

Uvidíme, jak se mu bude dařit v budoucnosti





Děkuji za
pozornost



