Implementace souborového systému v operačním systému HelenOS

Jakub Jermář

Úvod

Operační systém HelenOS je softwarový projekt, který z iniciativy studentů již několik let vzniká na půdě Matematicko-fyzikální fakulty Univerzity Karlovy v Praze. Cílem projektu není nahradit mainstreamové operační systémy, ale vytvořit pokud možno co nejlepší nosič pro zkoušení různých nápadů v oblasti implementace a výzkumu operačních systémů. Důkazem toho je například pět diplomových prací a jedna disertační práce, které jsou v současné době na fakultě řešeny a které s projektem HelenOS tematicky úzce souvisí. V minulosti byl HelenOS několikrát využit v kurzu operačních systémů jako projekt pro semestrální práce studentů. Smyslem všech těchto aktivit je postupně doplnit chybějící důležité vlastnosti pomocí více či méně obvyklých postupů. Charakteristickým rysem dosavadního vývoje je také úsilí o co možná nejobecnější vnitřní rozhraní a celkovou přenositelnost, což se projevuje tím, že HelenOS je možno spustit na celkem sedmi procesorových architekturách, počínaje procesory ARM a konče 64-bitovými procesory UltraSPARC.

Z různých reakcí, které se nám od zveřejnění projektu v roce 2006 dostaly, bylo patrné, že HelenOS je potenciálně velmi zajímavý projekt pro různé skupiny lidí – jako embedded operační systém, jako systém pro různá fyzikální měření, jako výukový systém nebo prostě jako zajímavá hračka. Uvědomili jsme si ale, že pro vznikající komunitu je složité se do jeho vývoje zapojit díky příliš velké adopční bariéře. Tato adopční bariéra byla z části tvořena skutečností, že pro vývoj HelenOSu je zapotřebí instalace simulátorů různých architektur a pro spolehlivý překlad ze zdrojových kódů také speciálních cross-kompilerů a cross-linkerů. Největší bariérou byla bez pochyby absence souborového systému, díky níž bylo obtížné uvědomit si potenciál, který HelenOS skrývá, a která značně omezovala použitelnost systému. V polovině roku 2007 tedy začaly snahy situaci napravit a navrhnout jednoduchý a zároveň rozšiřitelný rámec pro podporu souborových systémů v operačním systému HelenOS. Ve zbytku tohoto článku se budeme věnovat popisu toho, čeho bylo zatím dosaženo, s jakými implementačními problémy jsme se setkali a co na nás teprve čeká.

Pohled pod pokličku

V této části trochu nakoukneme HelenOSu pod pokličku. Zvláštní pozornost budeme věnovat návrhu IPC, protože ten má zásadní vliv na návrh samotného subsystému systému souborů.

Struktura systému

HelenOS obsahuje poměrně pokročilé mikrojádro, které vytváří příznivé prostředí pro běh uživatelských úloh (*task*) a umožňuje těmto úlohám navzájem komunikovat. Protože jsou adresové prostory jednotlivých úloh navzájem izolovány, jediným způsobem komunikace je buď prostřednictvím

zasílání IPC zpráv a nebo explicitním sdílením paměti. Některé uživatelské úlohy jsou v jistém smyslu speciální, protože zastávají úkoly, které by mělo v tradičním operačním systému na starosti jádro. Jedná se především o ovladače zařízení a o systémové služby jako je například virtuální konzole nebo právě souborový systém. Jádro těmto speciálním úlohám propůjčuje vyšší privilegia a umožňuje jim tak například přímý přístup k určitému hardware.

Vlákna a vlákénka

Uživatelské úlohy mohou být v HelenOSu vícevláknové, což umožňuje jádru využívat případného paralelizmu víceprocesorového systému tím, že naplánuje dvě vlákna (*thread*) jedné úlohy na dva různé procesory. HelenOS tedy podporuje a plánuje jaderná vlákna. Ta mohou mít, a v drtivé většině případů mají, i uživatelský kontext. Z důvodů popsaných níže v části o IPC se v uživatelském prostoru za podpory standardní knihovny každé jaderné vlákno rozpadá ještě na jedno až několik vlákének (*fibril*). Tato vlákénka běží v kontextu svého vlákna a jádro o nich nic neví a nedokáže je přímo plánovat. To má na starosti právě již zmíněná standardní knihovna. Na rozdíl od jaderných vláken, která mohou být v důsledku preempce pozastavena a odebrána z procesoru prakticky v libovolný okamžik, vlákénko běží, dokud se samo nevzdá řízení nebo dokud nezavolá IPC operaci, která nemůže být okamžitě provedena.

IPC a asynchronní framework

HelenOS navzdory moderním trendům využívá převážně asynchronní komunikaci, avšak podporuje i synchronní. Názvosloví užité v celém IPC subsystému je odvozeno z přirozené abstrakce telefonního spojení mezi člověkem na jednom konci a záznamníkem na konci druhém. Asynchronnost je zde dána právě tím, že na straně volaného je záznamník a nikoliv další člověk, který by byl s to okamžitě odpovídat.

Běžná IPC komunikace probíhá následujícím způsobem. Uživatelské vlákénko použije jednu ze svých telefonních linek (*phone*), kterou má propojenu se záznamníkem (*answerbox*) cílové úlohy, a uskuteční po ní krátký hovor (*call*). Odesílající vlákénko může buď pokračovat ve své práci a nebo počkat na odpověď. Cílová úloha má nyní ve svém záznamníku zmeškaný hovor. Dříve či později dojde k jeho vyzvednutí nějakým vlákénkem patřícím do cílové úlohy, které zprávu zpracuje, odpoví na ni a nebo ji přepošle nějaké třetí straně. V každém případě nakonec někdo na zprávu odpoví. Odpověď je analogicky uložena do záznamníku, v tomto případě záznamníku úlohy, která uskutečnila původní hovor.

Životní cyklus jednoho asynchronního IPC požadavku v sobě skrývá nejedno úskalí. V případě, že úloha obsahuje více než jedno vlákénko, není úplně zřejmé, pro které z nich je určen hovor uložený na záznamníku úlohy. Problém je tak závažný, že nám nabourává koncept spojení, jak ho například známe z BSD socketů. Krom toho obsluha asynchronní komunikace většinou vede k nepřehledné změti různých callbackových funkcí a kódu, který si pomocí stavových proměnných snaží udržet přehled o tom, kdo zrovna čeká na jakou zprávu či odpověď.

HelenOS tyto problémy řeší nasazením specifického asynchronního frameworku a intenzivním využitím manažerských vlákének. Speciální manažerská vlákénka jsou jako jediná oprávněna vyzvedávat hovory přímo ze záznamníku a dále je předávat uživatelem definovaným běžným

vlákénkům, která obsluhují dílčí spojení. Pokud se běžné vlákénko rozhodne čekat na odpověď, která ještě není k dispozici, asynchronní framework sám zaregistruje potřebný callback pro zachycení odpovědi a přepne se na jiné vlákénko, které může pokračovat v běhu. Na původní vlákénko se opět přepne až tehdy, když už je odpověď na jeho IPC požadavek připravena. Z pohledu jednotlivých vlákének vypadá komunikace prostřednictvím asynchronního frameworku velmi přímočaře a jednoduše.

IPC a sdílení paměti

Jak již bylo řečeno, uživatelské úlohy a v nich obsažená vlákénka mohou komunikovat pomocí ukládání krátkých hovorů na záznamník volané úlohy. To, že zde hovory označujeme jako krátké má svůj doslovný význam, protože tato metoda komunikace dokáže přenést pouze velmi malé množství dat. Původně se jednalo o čtyřnásobek toho, co daná architektura uměla uložit do všeobecného registru. Pro zasílání delších a velmi dlouhých zpráv nebyl takovýto způsob komunikace přijatelný. Již od začátku jsme plánovali, že naše jádro bude podporovat sdílení paměti mezi jednotlivými úlohami. Na konec jsme sdílení paměti dovedli takřka k dokonalosti tím, že jsme jej plně zaintegrovali do IPC. Ukázalo se totiž, že pomocí krátkých IPC zpráv je možno velmi elegantně dohodnout podmínky sdílení mezi jednotlivými účastníky. Pro sdílení paměti použije úloha speciální IPC metodu, *IPC_M_SHARE_IN* nebo *IPC_M_SHARE_OUT,* resp., a v argumentech zprávy uvede údaje o zdrojové nebo cílové, resp., virtuální adrese, velikosti sdílení a případně ještě informace o typu sdílení. Jádro si všimne, že jedna úloha iniciuje sdílení paměti s jinou úlohou a začne sledovat vývoj událostí. Druhá úloha obdrží zprávu a na základě obdržených informací se rozhodne, zda na sdílení přistoupí či nikoliv. Pokud přistoupí, odpoví první úloze kladným návratovým kódem. Jádro kladný návratový kód zaznamená a provede kroky vedoucí k vytvoření již dohodnutého sdílení. Pokud k dohodě nedojde, jádro zůstane v této záležitosti pasívní a iniciátor obdrží chybovou návratovou hodnotu.

Návrh a implementace

Funkcionalita celého subsystému systému souborů je rozložena do tří samostatných celků:

- podpora ve standardní knihovně,
- podpora ve VFS serveru,
- implementace výstupního VFS protokolu v ovladači koncového souborovém systému a
- podpora v knihovně libfs.

Standardní knihovna

Standardní knihovna obsahuje kód, který převádí více méně POSIX volání dané uživatelské úlohy na protokol, kterému rozumí vstupní část VFS serveru a tímto způsobem mu předává klientské požadavky. Některá volání, jako například *opendir()*, *readdir()*, *rewinddir()* a *closedir()* implementuje standardní knihovna přímo pomocí jiných volání – v tomto případě tedy *open()*, *read()*, *lseek()* a *close()*. Protože VFS server umí pracovat pouze s absolutními souborovými cestami, zajišťuje standardní knihovna rovněž volání *getcwd()* a *chdir()* a také převod všech relativních cest na absolutní. Předávání absolutních cest nemusí být sice vždy nejefektivnější, zato přináší řadu výhod. Jednak velmi zjednodušuje návrh VFS serveru a knihovny libfs a také umožňuje lexikální zpracování všech výskytů

komponenty *tečka-tečka*. Kromě výše uvedeného, standardní knihovna neobsahuje žádné datové struktury a algoritmy pro podporu systému souborů. Každý požadavek, který nedokáže vyřešit sama obratem přeposílá formou jedné nebo několika IPC zpráv VFS serveru.

VFS server

VFS server je ústředním a nejsložitějším prvkem podpory souborových systémů v operačním systému HelenOS. Představme si pomyslné rozdělení tohoto serveru na část vstupní a výstupní.

Část vstupní přebírá požadavky od klientských uživatelských úloh. Parametry těchto požadavků jsou buď absolutní cesty k souborům nebo deskriptory již otevřených souborů. Pokud je parametrem cesta, provede VFS operaci *vfs_lookup_internal()*, jejímž výsledkem je tak zvaný VFS triplet. VFS triplet je uspořádaná trojice, která pomocí globálního čísla souborového systému, globálního čísla zařízení a čísla souboru na dané instanci souborového systému jednoznačně určuje nějaký soubor. Zahešováním podle VFS tripletu se VFS server pokusí najít odpovídající VFS uzel. VFS uzel, instance typu *vfs_node_t*, je abstrakce, která v adresovém prostoru VFS serveru představuje nějaký soubor, na který má VFS referenci. Všechny soubory, se kterými VFS pracuje jsou reprezentovány pomocí VFS uzlů. V případě, že parametrem operace je deskriptor souboru, je převod na VFS uzel daleko jednodušší, protože tabulka otevřených souborů, kterou VFS udržuje pro každou klientskou úlohu zvlášť, přímo obsahuje ukazatel na strukturu reprezentující odpovídající otevřený soubor (*vfs_file_t*) a ta zase přímo ukazuje na VFS uzel.

Jakmile VFS server zná VFS uzel, kterého se VFS operace týká, může ji předat ovladači příslušné instance souborového systému a nebo ji provést sám. V současné době realizuje VFS server například operaci *VFS_SEEK*, protože *VFS_SEEK* nevyžaduje další interakci s příslušným souborovým systémem. *VFS_SEEK*, která odpovídá POSIX volání *lseek()*, pouze posune ukazatel aktuální pozice pro čtení a zápis v příslušné struktuře *vfs_file_t*. K tomu může za určitých okolností potřebovat znát velikost daného souboru – tu ale VFS server udržuje přímo v odpovídajícím VFS uzlu.

Výstupní část VFS pak komunikuje s ovladačem koncového souborového systému, jehož číslo a také číslo zařízení, na kterém je tento souborový systém připojen, zná díky informacím uloženým ve VFS uzlu. Soubor všech zpráv, které může VFS server poslat jednotlivým ovladačům koncových souborových systémů jednoznačně definuje výstupní protokol VFS. Všechny serverové úlohy, jejichž ambicí je ovládat nějaký konkrétní souborový systém, musí tomuto protokolu rozumět a musí jej implementovat. V podstatě se jedná o objektově orientovaný návrh, který má jen trochu přirozenější formu než VFS v monolitických kernelech, které bývají realizovány pomocí sady ukazatelů na funkce.

PLB a kanonické cesty

Významnou úlohou, na jejímž řešení se podílí VFS server spolu s připojenými souborovými systémy, je převod znakových řetězců reprezentující cesty k souborům na VFS triplety. VFS server postupuje zhruba tak, že se postupně ptá ovladačů souborových systémů připojených podél dotyčné cesty. Každý dotázaný ovladač vyřeší maximální část cesty od již vyřešené části až po případný další bod připojení *(mount point)* jiného souborového systému. Cestu nakonec vyřeší poslední dotázaný souborový systém a pošle VFS serveru požadovaný VFS triplet jako odpověď. Cílem návrhu podpory souborových systémů v operačním systému HelenOS bylo vyhnout se situaci, ve které by se cesta či její část neustále kopírovala mezi VFS serverem a dílčími ovladači koncových souborových systémů. Za tímto účelem

alokuje a udržuje VFS server cyklickou vyrovnávací paměť, do které umisťuje vyhledávané cesty. Každá implementace koncového souborového systému tuto vyrovnávací paměť, zkráceně označovanou jako PLB *(Pathname Lookup Buffer),* s VFS serverem sdílí v režimu pro čtení. Samotné umístění cesty do PLB a řízení distribuovaného vyhledání je implementováno v již zmíněné funkci *vfs_lookup_internal()*.

vfs_lookup_internal() musí garantovat, že do PLB uloží jen cesty v kanonickém tvaru. To znamená, že musí ověřit že cesta neobsahuje například dvě lomítka za sebou, že v cestě není žádný symbol *tečka* či *tečka-tečka*. Pokud není cesta v tomto smyslu v pořádku, použije se speciální restartovací automat, který ji postupným opracováváním převede na kanonický tvar.

VFS server tedy uloží již kanonickou cestu do PLB a zkontaktuje souborový systém, který je připojen ke kořeni adresářového stromu. Zašle mu zprávu typu *VFS_LOOKUP* a jako argumenty uvede indexy prvního a posledního znaku cesty v PLB. Poté, co kořenový souborový systém vyřeší svou část cesty, nedopoví nutně VFS serveru. Pokud je totiž ještě co řešit, přepošle zprávu *VFS_LOOKUP* tomu souborovému systému, na jehož bodě připojení sám skončil vyhledávání a upraví argumenty zprávy tak, aby index prvního znaku cesty v PLB ukazoval na první nevyřešený znak. Vyhledávání v tomto duchu pokračuje až se nakonec jednomu souborovému systému podaří cestu vyřešit celou. Teprve tento poslední souborový systém odpoví na původní zprávu *VFS_LOOKUP* a v odpovědi zahrne celý VFS triplet odpovídající nalezenému souboru. Díky právě popsanému návrhu ušetří HelenOS zhruba polovinu systémových volání, nemluvě o ušetřeném kopírování paměti.

Ovladač koncového souborového systému

Jak již bylo řečeno, každý ovladač souborového systému musí implementovat výstupní VFS protokol. Pro účely ověření funkčnosti celého konceptu jsme přistoupili k návrhu a realizaci velmi jednoduchého souborového systému, jehož datové struktury existují jen v paměti počítače a který nemá ani žádný diskový formát – TMPFS. Na druhou stranu, TMPFS má hierarchickou adresářovou strukturu a umožňuje plnohodnotně otestovat všechny souborové operace, které mohou uživatelské úlohy používat.

Pro potřeby složitějších souborových systémů, které mají diskový formát, jsme navrhli speciální server – DEVMAP, který dokáže propojit příslušný ovladač koncového souborového systému s ovladačem blokového zařízení podle globálního čísla zařízení. Potřeba komunikace mezi souborovým systémem a blokovým zařízením je jasná. Soubor zpráv, které bude ovladač souborového systému ovladači blokového zařízení zasílat jednoznačně určí protokol pro přístup k blokovým zařízením a, podobně jako v případě výstupního VFS protokolu, umožní používat libovolná bloková zařízení s ovladači podporujícími tento protokol.

Kromě implementace výstupních VFS operací musí ovladače koncových souborových systémů implementovat také rozhraní knihovny *libfs*, kterou popisujeme níže.

Knihovna libfs

Některé konstrukce, které logicky patří do ovladače koncového souborového systému, by se objevovaly v implementaci každého takového ovladače – někdy jen s minimální obměnou, někdy dokonce na vlas stejné jako v ostatních implementacích. Pro odstranění tohoto problému byla založena knihovna *libfs*, jejímž cílem je pomocí vhodných abstrakcí koncentrovat podobné duplicity do jednoho místa. V

současné době je součástí knihovny kód, kterým se jednotlivé souborové systémy registrují u VFS serveru a dávají mu vědět o své existenci.

Dalším a velmi podstatným příkladem *libfs* kódu je funkce *libfs_lookup()*. Tato funkce je vlastně generickou šablonou pro implementaci výstupní VFS operace *VFS_LOOKUP*. Tuto operaci musí podporovat každý koncový souborový systém, ale vzhledem k její složitosti a také provázanosti mezi místy připojení jednotlivých souborových systémů je strategicky výhodnější spravovat její kód centrálně. *libfs_lookup()* je specifická také tím, že neimplementuje pouze vyhledávání, ale rovněž vytváření a rušení jmen souborů v adresářovém stromě, vytváření prázdných souborů a adresářů a určování rodičovského VFS uzlu k uzlu zadaného cestou. Aby to fungovalo v rámci konkrétního souborového systému, musí tento implementovat několik operací, které *libfs* řeknou, jak přesně má prohledávat adresář, jak vytvořit či zrušit jméno souboru v adresářovém stromě nebo jak vytvořit či smazat soubor.

Změny vynucené implementací

Předchozí část se věnovala stručnému popisu toho, čeho bylo zatím na úrovni podpory souborových systému v operačním systému HelenOS dosaženo. V této části se podíváme na několik změn, které si tato podpora vynutila.

Vylepšení IPC

Některé operace VFS serveru by bylo hezké provést zasláním jedné krátké IPC zprávy, pokud by se ovšem vystačilo s dostupnými IPC argumenty. Příkladem takové operace je již zmiňovaná *vfs_lookup_internal()*. Tato interní funkce zasílá kořenovému souborovému systému zprávu *VFS_LOOKUP* o pěti argumentech, což je o dva více, než bylo možné v dřívější implementaci předcházející přidání podpory souborového systému. Tento případ jasně ukazuje, že na rozdíl od jednodušších protokolů, které HelenOS podporoval již dříve, protokol složitostí odpovídající potřebám protokolu VFS vyžaduje větší počet argumentů IPC zpráv. Omezení počtu IPC argumentů má své kořeny v počtu argumentů, které lze předat v registrech procesoru během systémového volání. Zvýšením tohoto limitu na šest jsme docílili toho, že při asynchronním typu IPC je nyní možno předat čtyři IPC argumenty přímo v registrech. Neoptimalizovaná verze IPC dokáže předat až pět IPC argumentů, s výhledem na další rozšíření do budoucna.

Pro komunikaci mezi standardní knihovnou a VFS serverem, během níž se předávají řetězcové argumenty, by bylo použití sdílené paměti příliš těžkopádné. Ze své povahy převážně jednorázové přesuny dat je mnohem lepší uskutečnit nikoliv sdílením, ale kopírováním. Kopírování vyznívá lépe i pro operace *read()* a *write()*, protože se opět jedná o jednorázový přesun dat mezi klientskou úlohou a ovladačem koncového souborového systému. Tato úvaha vedla k zavedení dvou nových systémových IPC metod, *IPC_M_DATA_READ* a *IPC_M_DATA_WRITE*, jejichž použití je analogické použití metod pro sdílení paměti. Rozdíl je samozřejmě v tom, že jádro při použití těchto metod nenastaví sdílení paměti, ale fyzicky zkopíruje blok dat z jednoho adresového prostoru do druhého.

Příklad *read()* a *write()* naznačil, že IPC zprávy mohou přejít přes jeden či několik mezilehlých úloh – v tomto případě VFS server. Pro podporu přeposílání kopírovacích zpráv a zpráv pro sdílení paměti bylo potřeba toto chování povolit a zaručit, že úlohy, které takové zprávy přeposílají, nemohou do

obsahu zprávy nijak zasáhnout. Příjemným vedlejším efektem tohoto malého vylepšení je skutečnost, že samotná kopie dat nebo sdílení paměti bude provedeno pouze jednou, a to mezi prvním odesílatelem a posledním adresátem příslušné zprávy.

Tématu přeposílání zpráv se týká i další změna IPC. Protože jsou jednotlivá IPC spojení díky asynchronnímu frameworku obsluhována samostatnými vlákénky, způsobilo by pouhé přeposlání nějaké IPC zprávy dalšímu adresátovi vytvoření nového vlákénka v úloze tohoto adresáta. Nové vlákénko by mělo na starosti spojení mezi původním odesílatelem a tímto posledním adresátem, což ovšem nemusí být to, co potřebujeme. Příklad, který toto chování ilustruje je implementace operace *VFS_READ*, odpovídající POSIX volání *read()*, uvnitř VFS serveru. VFS server obdrží od standardní knihovny jedné ze svých klientských úloh zprávu *VFS_READ*, díky čemuž pozná, že jako další bude následovat zpráva typu *IPC_M_DATA_READ* obsahující pokyn pro nakopírování přečtených dat do klienta. Po jejím obdržení ji VFS server nemůže jenom tak přeposlat ovladači koncového souborového systému, protože by ji tento zařadil do jiného spojení. Místo toho ji VFS server při přeposlání označkuje, čímž dá dalšímu příjemci najevo, že zpráva prošla přes něj a že je ji tedy třeba chápat jako součást spojení mezi nimi. Označkování nemá žádný vliv na směrování odpovědi na zprávu – odpověď bude vždy doručena přímo původnímu odesílateli.

Závěr

Podpora souborových systémů obsažená v operačním systému HelenOS se vyznačuje přiměřenou jednoduchostí a přímočarostí. Zatím se jedná spíše o funkční prototyp než o plnohodnotný VFS, jaký lze nalézt v jiných, mnohem vyzrálejších operačních systémech, ale i přesto současná verze obsahuje několik zajímavých myšlenek. Bylo dosaženo částečné kompatibility s tou částí standardu POSIX, která se týká souborových systémů. Implementace zatím nepodporuje přístupová práva ani nezaznamenává čas posledního přístupu či modifikace souboru. Z našeho pohledu se však jedná o funkcionalitu sekundární, kterou bude možno přidat později. V některých případech se naše implementace mírně odklání od POSIXových chybových návratových kódů a některé okrajové situace, které v POSIXu znamenají chybu, zvládne ošetřit. Pokud jde o ovladače koncových souborových systémů, vytvořili jsme čistě paměťový souborový systém TMPFS, který ovšem dovoluje otestovat a vyzkoušet funkcionalitu celého VFS serveru. V současné době pracujeme na podpoře druhého koncového souborového systému – FAT16. Koexistence dvou souborových systémů nám umožní rozšířit VFS server a funkci *libfs_lookup()* tak, aby bylo možno připojit více souborových systémů současně. Pro diskové souborové systémy vyvstane potřeba vytvořit nějaké jednotné rozhraní pro vyrovnávací paměti diskových bloků. Postupným přidáváním dalších a dalších souborových systémů dojde k vyzrání a stabilizaci VFS protokolu.

Zda se nám tyto kroky podaří v budoucnu realizovat, záleží také do určité míry na schopnosti tvořící se HelenOS komunity přispívat do vývoje tohoto systému. Díky snahám nabídnout komunitě již použitelný základ systému by mělo být jednodušší stát se její součástí.