Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Jan Dolejš

HelenOS jako Xen hypervisor

Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: Mgr. Martin Děcký

Studijní program: Informatika
Studijní obor: Softwarové systémy

Praha 2012

Děkuji vedoucímu Mgr. Martinu Děckému, za odborné vedení práce, za rady a čas, který mi při vypracování této práce věnoval. Dále bych chtěl poděkovat Mgr. Tomáši Benhákovi za čas, který mi poskytl při vysvětlení principu fungování portu HelenOS pro Xen hypervisor a také všem kolegům a přátelům za podporu při jejím vypracování.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 30. 7. 2012 Jan Dolejš

Název práce: HelenOS jako Xen hypervisor
Autor: Jan Dolejš
Katedra / Ústav: Katedra distribuovaných a spolehlivých systémů
Vedoucí diplomové práce: Mgr. Martin Děcký

Abstrakt: Cílem práce je vytvoření prototypové implementace rozhraní hypervisoru Xen v operačním systému HelenOS pro platformu IA-32. Výstupem práce je port operačního systému HelenOS, na kterém lze provozovat vybranou paravirtualizovanou doménu. Práce obsahuje stručný úvod do způsobů virtualizace a uvádí hlavní rozdíly mezi nimi. Práce dále popisuje ty části architektury hypervisoru Xen a operačního systému HelenOS, které budou dále zahrnuty do prototypové implementace. Nedílnou součástí práce je výběr vhodné testovací domény a analýza změn nutných k jejímu provozování, stejně jako jejich popis.

Klíčová slova: HelenOS, Xen, virtualizace, paravirtualizace

Title: HelenOS as Xen hypervisor
Author: Jan Dolejš
Department: Department of Distributed and Dependable Systems
Supervisor: Mgr. Martin Děcký

Abstract: The aim of the master thesis is to create a prototype implementation of the interface of the Xen hypervisor within the HelenOS operating system. The target architecture of this prototype implementation is IA-32. The result of the thesis is a port of HelenOS which can be uses to run the selected paravirtualized domain. The thesis contains a brief introduction to the methods of virtualization and describes the main differences between them. Thesis also describes the parts of the architecture of the Xen hypervisor and the HelenOS operating system, which will be modified in the prototype implementation. The most important part of this thesis is to select of the t testing domain as well as analyze and describe all changes, which were required for the domain's operation.

Keywords: HelenOS, Xen, virtualization, paravirtualization

Obsah

1 Úvod 1
1.1 Cíle práce 1
1.2 Struktura práce 2
1.3 Obsah přiloženého CD 2
2 Stručný úvod do virtualizace běhu operačních systémů 3
2.1 Přínosy virtualizace 3
2.2 Úplná virtualizace 4
2.2.1 Dostačující podmínky úplné virtualizace. 5
2.2.2 Problém úplné virtualizace na IA-32 5
2.3 Virtualizace s přímou podporou procesoru 6
2.4 Paravirtualizace 6
3 Xen hypervisor 8
3.1 Historický vývoj hypervisoru Xen 8
3.2 Architektura 9
3.2.1 Změna úrovně oprávnění virtuálního stroje 9
3.2.2 Virtuální stroje 10
3.2.3 Info stránky 12
3.2.4 Pseudo-fyzický paměťový model 13
3.3 Pohled do zdrojových kódů hypervisoru Xen 14
3.3.1 Správa paměti 14
3.3.2 Kanály událostí 15
3.3.3 Plánování 16
4 Operační systém HelenOS 18
4.1 Struktura jádra 18
4.2 Správa paměti 19
4.2.1 Fyzická paměť 19
4.2.2 Virtuální paměť 19
4.3 Zpracování výjimek 21
4.4 Vlákna a úlohy 22
4.5 Plánování 22
4.6 Spuštění uživatelských úloh 23
5 Analýza 24
5.1 Analogie mezi HelenOS a Xenem 26
5.2 Předpoklady a změny úspěšného nabootování domény 26
5.2.1 Relokace jádra HelenOS 26

5.2.2 Stránkování HelenOS ... 28
5.2.3 Načtení zaváděné domény ... 28
5.2.4 Iniciální mapování ... 29
5.2.5 Přechod na úroveň oprávnění domény ... 32
5.3 Implementované funkce ... 33
5.3.1 Omezení prototypové implementace ... 33
5.3.2 Podpora segmentace ... 33
5.3.3 Výjimky ... 34
5.3.4 Kanály událostí ... 35
5.3.5 Grant tabulky ... 36
5.4 Změny suplující chybějící doménu nula ... 37
5.4.1 Backend ovladač konzole ... 37
5.5 Hypervolání ... 38
6 Implementace ... 39
6.1 Architektura ia32xenh ... 39
6.1.1 Zdrojové kódy architektury ia32xenh ... 39
6.1.2 Konfigurace a makefile ... 39
6.2 Prerekvizity ... 40
6.2.1 Relokace jádra HelenOS ... 40
6.2.2 Stránkování HelenOS – implementace metody PAE ... 40
6.2.3 Inicializace virtuálního adresového prostoru jádra ... 43
6.3 Načtení zaváděné domény ... 43
6.3.1 Funkce hinit ... 44
6.3.2 Načtení parametrů zaváděné domény ... 44
6.3.3 Vypočtení rozložení VAP domény ... 46
6.3.4 Vytvoření iniciálního mapování ... 49
6.3.5 Inicializace struktur start_info_t a shared_info_t ... 50
6.4 Zavedení inicializované domény ... 51
6.5 Implementované rozhraní ... 51
6.5.1 Správa stránkovacích tabulek ... 52
6.5.2 Správa segmentů ... 52
6.5.3 Výjimky ... 53
6.5.4 Kanály událostí ... 57
6.5.5 Grant tabulky ... 58
6.5.6 Ladicí konzole ... 58
6.5.7 Ostatní ... 58

6.6 Změny suplující chybějící dom0 59
6.6.1 Backend část ovladače konzole 59
6.7 Změny v testovací doméně 63
7 Ladění 64
7.1 Nástroje 64
7.2 Propojení Qemu a GDB 65
7.3 Často používané příkazy GDB 65
7.3.1 Načtení symbolů 65
7.3.2 Breakpointy 66
7.3.3 Prohlížení obsahu paměti 66
7.3.4 Obnovení pozastaveného emulátoru 67
8 Závěr 68
8.1 Možnosti budoucího rozšíření 68
9 Literatura 70

1 Úvod

Virtualizace běhu operačních systémů přináší další úroveň abstrakce. Umožňuje běh několika izolovaných operačních systémů současně. Hypervisor je nejvyšším arbitrem, který řídí přístup virtualizovaných operačních systémů k hardwaru počítače, řídí jejich běh a zaručuje jejich vzájemnou izolaci. Jedná se o analogii operačního systému a procesu v nevirtualizovaném prostředí.

HelenOS je multiplatformní, experimentální operační systém, který vznikl na půdě Matematicko-fyzikální fakulty Univerzity Karlovy v Praze a byl rozvíjen v rámci mnoha diplomových i semestrálních prací.

Xen vznikl v počítačových laboratořích na univerzitě Cambridge. První verze podporovala pouze paravirtualizaci platformy IA-32. V současné době[1] je podporována jak paravirtualizace platforem AMD64, IA-32, IA-64 a ARM, tak i plná virtualizace platforem IA-32 (Intel VT-x) a AMD64 (AMD-V).

Tato diplomová práce je doplňkem práce [1], v rámci které byla implementována podpora paravirtualizace platformy Xen v systému HelenOS.

1.1 Cíle práce

Hypervisor Xen byl do současné doby vyvíjen desítkami lidí po dobu několika let. Cílem této práce nemůže být implementace všech funkcí rozhraní hypervisoru Xenu. Z časových důvodů to není možné. Součástí analýzy je výběr vhodné testovací domény a takové podmnožiny hypervolání k implementaci, která bude dostatečná pro její úspěšné provozování. Cílovou architekturou je IA-32.

Práci lze rozdělit do dvou navazujících částí. První část lze charakterizovat jako teoretickou část. V ní jsou popsány virtualizace a ty funkční bloky operačního systému HelenOS a hypervisoru Xen, které budou součástí prototypové implementace. V druhé části jsou popsány kroky nutné k realizaci prototypové implementace rozhraní hypervisoru Xen. Do této části spadá hlavně kapitola věnovaná analýze změn a implementace.

[1] V době vypracování je aktuální verzí Xen 4.1.2

1.2 Struktura práce

Podrobnější komentář k obsahu jednotlivých kapitol:

- *Kapitola* 2 – Představení virtualizace, hlavní výhody virtualizace, virtualizační techniky a rozdíly mezi nimi.
- *Kapitola* 3 – Historický vývoj a architektura Xenu. Pohled na vybrané části zdrojových kódů Xenu.
- *Kapitola* 4 – Pohled na architekturu operačního systému HelenOS v oblasti správy paměti, zpracování přerušení, reprezentace úloh a vláken a další.
- *Kapitola* 5 – Výběr vhodné testovací domény prototypové implementace. Analýza změn v jádře HelenOS nutných k provozování vybrané domény.
- *Kapitola* 6 – Popis zvolené prototypové implementace rozhraní Xenu.
- *Kapitola* 7 – Možnosti pro ladění prototypové implementace. Popis často používaných příkazů debuggeru GDB.
- *Kapitola* 8 – Závěrečné shrnutí.

1.3 Obsah přiloženého CD

Obsahem přiloženého CD je:

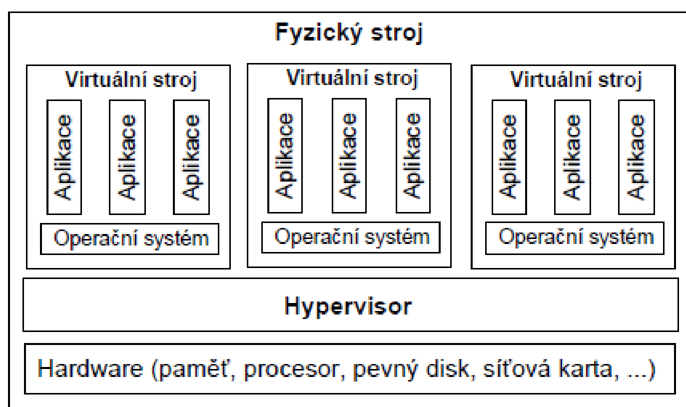
- *images/* – Adresář obsahující spustitelný obraz prototypové implementace rozhraní hypervisoru Xen. Součástí obrazu je i vybraná testovací doména.
- *src/* – Adresář obsahující zdrojové kódy prototypové implementace rozhraní hypervisoru Xen v systému HelenOS a zdrojové kódy testovací domény.
- *doc/* – Adresář obsahující text této práce v elektronické podobě.

2 Stručný úvod do virtualizace běhu operačních systémů

Cílem této kapitoly je seznámit čtenáře s problematikou virtualizace běhu operačních systémů. Její součástí je popis postupů a technik sloužících k vytvoření virtualizační vrstvy.

Obsah této kapitoly vychází především z prací [1; kapitola 2], [2; kapitola 3], [3; část 3.2] a knihy [4; kapitola 1]. Některé popisné pasáže textu vznikly jako volné parafráze prací [1] a [2].

Virtualizační vrstva, která se nachází mezi hardwarem a operačním systémem je nazývána hypervisor[2]. Hypervisor je správcem hardwaru, který virtuálním strojům vytváří iluzi skutečného fyzického stroje. Komunikace mezi virtuálním a fyzickým hardware zpravidla probíhá prostřednictvím hypervisoru. Schématický model virtualizace zobrazuje Obrázek 1.



Obrázek 1: Schéma virtualizace operačních systémů (zdroj: [1, Obr. 2-1])

2.1 Přínosy virtualizace

Motivace pro nasazení virtualizace operačních systémů vychází z téže motivace, jako zavedení multitaskingu. Výpočetní síla fyzických strojů roste rychleji, než nároky na ně kladené. Kvůli tomu se zvyšuje podíl nevyužitých prostředků fyzických strojů. Nevyužité prostředky navyšují náklady na provoz, u kterých je

[2] Hypervisor, simulátor, virtualizér, monitor virtuálního stroje a paravirtualizér jsou různá označení virtualizační vrstvy.

však nejvyšší tlak na jejich snižování. Logickým vyústěním nastalé situace je nasazení virtualizace.

Optimální využití dostupných zdrojů není jediným přínosem při použití virtualizace. Dalším přínosem je možnost jednoduchého klonování virtuálního stroje. Klonováním je usnadněno testování změn před nasazením do produkčních systémů. Snadnější testování přináší další snížení alokovaného objemu finančních prostředků nutných pro integraci změn.

Neopominutelnou výhodou použití virtualizace je možnost migrace mezi fyzickými stroji. Virtuální stroj může být migrován na jiný fyzický stroj jako reakce na narůstající množství chyb hardwaru nebo upgrade systému. Po provedení změn na fyzickém stroji lze migrované virtuální stroje umístit zpět na původní fyzický stroj.

V neposlední řadě je virtualizací běhu operačních systémů dosaženo většího stupně izolace, než v případě procesů v rámci operačního systému. Větší stupeň izolace přináší možnosti v podobě vytvoření množství jednoduchých virtuálních služeb s nízkými nároky na fyzický hardware.

Uvedený výčet přínosů při použití virtualizace není konečný. Další přínosy lze nalézt například v knize [4]. Autorovým záměrem bylo nastínit několik základních výhod při použití virtualizace operačních systémů.

2.2 Úplná virtualizace

Úplná virtualizace je technika, ve které je pro každý virtuální stroj vytvořen identický obraz fyzické architektury. Virtuální stroj má k dispozici stejné prostředky i instrukční sadu fyzického stroje a není schopen detekovat, že běží ve virtuálním prostředí.

2.2.1 Dostačující podmínky úplné virtualizace.

Dostačující podmínky, které musí být hypervisorem pro úplnou virtualizaci naplněny, poprvé definovali Popek a Goldberg ve svém článku [6]:

- *Správa zdrojů* – Virtuální stroje musí být vzájemně izolovány. Splnění této podmínky je možné pouze za předpokladu, že hypervisor spravuje a zprostředkovává přístup ke zdrojům fyzického stroje.
- *Ekvivalence* – Běh programu v rámci virtuálního stroje musí být ekvivalentní běhu programu na fyzickém stroji. Z podmínek ekvivalence se obvykle vynechává podmínka na zachování přesného časování, která nemůže být z důvodů sdílení času procesoru mezi virtuálními stroji naplněna.
- *Efektivita* – Všechny instrukce, které neohrožují vykonávání ostatních virtuálních strojů, musí být vykonávány nativně. Tato podmínka odlišuje virtualizaci od emulace.

Pro naplnění těchto podmínek je nutné, aby fyzický stroj naplňoval určité charakteristiky:

- Procesor podporuje alespoň dva režimy ochrany (privilegovaný, uživatelský).
- Je definováno asynchronní přerušení, pomocí něhož může být procesor upozorněn na vzniklé události.
- Správa paměti podporuje dynamickou relokaci a další bezpečnostní mechanismy, jako například stránkování.
- Množina instrukcí, které mění nastavení systémových zdrojů (například systémové registry) a množina instrukcí, jejichž chování (nebo výsledek) je ovlivněno nastavením systémových zdrojů, patří do množiny privilegovaných instrukcí.

2.2.2 Problém úplné virtualizace na IA-32

Procesor IA-32 obsahuje 17 instrukcí, které jsou závislé na nastavení systémových zdrojů, ale nejsou privilegované [7], čímž je narušena podmínka ekvivalence. I přes narušení podmínky ekvivalence je možné docílit úplné virtualizace.

Přepisování instrukcí

Přepisování instrukcí je technika úplné virtualizace, která je použita v populárním virtualizačním softwaru VMWare. Technika přepisování instrukcí funguje na principu prohledávání instrukčního proudu, identifikování privilegovaných instrukcí a jejich přepsání na volání emulačních funkcí.

Výkon virtuálního stroje, při použití techniky přepisování instrukcí, je dle knihy [4] na úrovni 80-97% výkonu fyzického stroje.

2.3 Virtualizace s přímou podporou procesoru

Virtualizace s přímou podporou procesoru byla poprvé představena firmou Intel pod označením Intel VT-x (z anglického „Intel Virtualization Technology") [8]. Zanedlouho poté představila firma AMD vlastní řešení AMD-V (z anglického „AMD Virtualization"). Ačkoliv se v principu jedná o dvě nezávislé technologie, jejich princip je stejný. Obě přidávají další úroveň oprávnění, která je vyhrazena pro běh hypervisoru a provádění dříve problematických instrukcí je mimo hypervisor zakázáno (respektive vyvolá výjimku, která je hypervisorem ošetřena).

2.4 Paravirtualizace

Paravirtualizace je virtualizační technika, ve které je pro každý virtuální stroj vytvořeno prostředí podobné, ne však stejné, fyzickému stroji. Paravirtualizace tedy porušuje podmínku ekvivalence, která je jednou z předpokladů úplné virtualizace. Ostatní Popek-Goldbergovy podmínky musí zůstat splněny.

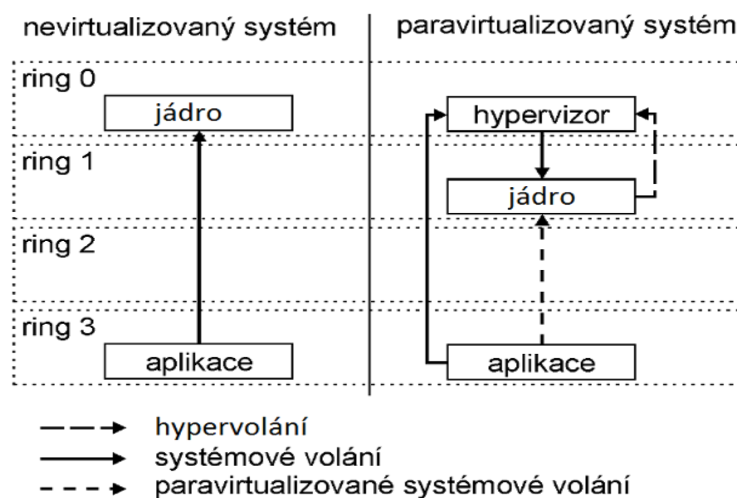
Důležitým problémem použití techniky paravirtualizace, který je potřeba vyřešit, je virtualizace procesoru. Většina procesorových architektur implementuje dvě, nebo čtyři úrovně ochrany, které jsou hierarchicky uspořádány od nejvíce privilegované (privilegovaný režim, který bývá obvykle označován jako „ring 0"[3]) po nejméně privilegované (uživatelský režim, který bývá obvykle označován jako „ring 3"). Kód vykonávaný v privilegovaném režimu není žádným způsobem omezován, smí provádět všechny instrukce, má přímý přístup k fyzickým zařízením i systémovým registrům. Naopak kód spuštěný v uživatelském režimu smí provádět jen instrukce z podmnožiny neprivilegovaných instrukcí, nemá přímý přístup k fyzickým zařízením ani k celé fyzické paměti. Vykonání privilegované operace probíhá skrze služby nabízené operačním systémem. Žádost o provedení služby je označena termínem systémové volání (z anglického „systém call").

---

[3] Českým ekvivalentem je slovní spojení „úroveň oprávnění nula".

V případě paravirtualizace je privilegovaný režim vyhrazen hypervisoru; virtuálnímu stroji je přidělena nižší úroveň oprávnění (která vychází z použité procesorové architektury). Tento proces je nazýván `ring deprivileging`.

Kvůli přesunu virtuálního stroje na nižší úroveň oprávnění si virtuální stroj musí být vědom faktu, že běží ve virtualizovaném prostředí. Privilegované operace jsou prováděny pomocí hypervisorem explicitně definovaného rozhraní. Komunikace virtuálního stroje a hypervisoru je realizována pomocí takzvaných hypervolání (z anglického „hypercall“), která jsou obdobou systémových volání mezi uživatelskými úlohami a jádrem (architekturu hypervolání architektury IA-32 zobrazuje Obrázek 2).



Obrázek 2: Architektura hypervolání v IA-32

Pro docílení spolupráce mezi virtuálním strojem a hypervisorem je nutné modifikovat operační systém virtualizovaného stroje tak, aby místo privilegovaných instrukcí používal odpovídající hypervolání hypervisoru.

3 Xen hypervisor

Cílem této kapitoly je podat srozumitelný popis architektury virtualizéru Xen. Kromě popisu architektury virtualizerů Xen jsou zde popsány i vybrané části z volně dostupných zdrojových kódů hypervisoru Xen, které mohou sloužit jako inspirace při budoucím rozšiřování této práce.

3.1 Historický vývoj hypervisoru Xen

Xen hypervisor je virtualizační nástroj, jehož vývoj započal jako výzkumný projekt na univerzitě v Cambridge v druhé polovině devadesátých let 20. století. První veřejně přístupná verze Xenu byla uvolněna v roce 2003; Tato verze podporovala jediný 32-bitový procesor.

Mnoho změn přinesla třetí verze Xenu, která byla vydána o dva roky později. V této verzi již byly podporovány i architektury AMD64 a IA-64. Navíc přibyla i podpora technologie Intel VT-x a později i AMD-V[4]. Díky tomu bylo možné, na určitých procesorech, provozovat úplnou virtualizaci, která funguje nad neupraveným operačním systémem. Třetí verze hypervizoru Xen pracuje s metodou stránkování PAE (z anglického „Physical Address Extension"), díky které může být 32-bitovým virtuálním strojům přiděleno až 64 GB fyzické paměti. O rozšíření hypervisoru Xen se zasloužilo i jeho začlenění do virtualizačního řešení firem Red Hat, Novell a Sun.

V roce 2008 byl společností Samsung vytvořen projekt Xen ARM, jehož cílem bylo rozšíření portfolia podporovaných platforem o platformu ARM. Podpora platformy ARM bude začleněna do hlavní vývojové linie v průběhu tohoto roku.

V roce 2010 byla vydána zatím poslední hlavní verze s pořadovým číslem čtyři. Součástí této verze je i možnost použít linuxové jádro jako privilegovanou doménu nula za pomoci PVOps. Linux 2.6.37 je první verze linuxového jádra, kterou lze použít jako privilegovanou doménu bez dalšího. Linux od své třetí verze

---

[4] Podpora technologie Intel VT-x byla součástí verze 3.0.0; podpora technologie AMD-V verze 3.0.2

obsahuje plnou podporu privilegované i neprivilegované domény hypervisoru Xen. Hlavní distributoři Linuxu oznámili znovuzavedení Xenu do jejich distribucí.

Nejnovější veřejně dostupnou verzí hypervisoru Xen je verze 4.1.2.

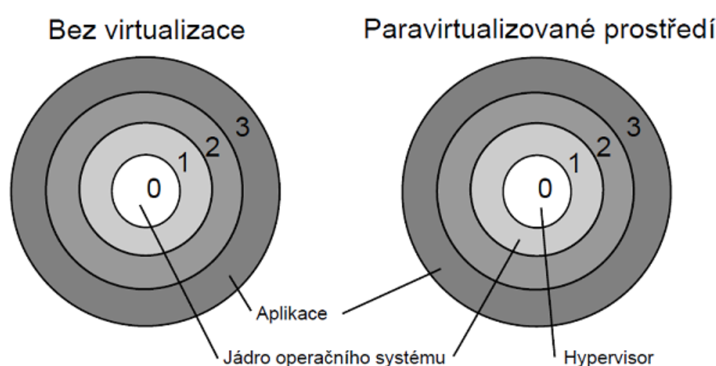
3.2 Architektura

V této části jsou popsány ty prvky architektury hypervisoru Xen, které jsou nutné k pochopení principů fungování hypervisoru Xen. Popis ostatních funkčních bloků, stejně jako popis rozhraní jednotlivých hypervolání je součástí práce [1].

Obsah této části vychází především z práce [1; kapitola 3] a knihy [4]. Některé popisné pasáže textu vznikly jako volné parafráze práce [1].

3.2.1 Změna úrovně oprávnění virtuálního stroje

Jednou z největších změn pro operační systém běžící pod Xenem je přesun z privilegované úrovně oprávnění procesoru do některé z neprivilegovaných úrovní oprávnění procesoru. Na jakou úroveň oprávnění je systém přesunut se odvíjí od architektury procesoru. V případě architektury IA-32[5] je operační systém přesunut z úrovně oprávnění nula na úroveň oprávnění jedna tak, jak zobrazuje Obrázek 3.



Obrázek 3: Prostředí nativní vs paravirtualizované (převzato z [1, Obrázek 3.1])

---

[5] Pro zajímavost uveďme, že architektura x86-64 poskytuje pouze dvě úrovně oprávnění, což znamená, že paravirtualizovaný operační systém běží na stejné úrovni jako jeho aplikace.

Z pohledu paravirtualizovaného jádra není změna úrovně oprávnění jedinou změnou, se kterou se musí vypořádat.

První změnou, se kterou se paravirtualizované jádro operačního systému musí vypořádat je režim procesoru v průběhu jeho zavádění. Všechny procesory architektury IA-32 již od dob procesoru 8086 startují v reálném režimu (z anglického „real mode“). Reálný režim je 16-bitový režim s přístupem k 20-bitovému adresového prostoru. Jedna z prvních úloh moderních operačních systémů je přepnout režim procesoru do privilegovaného režimu (z anglického „priviledged mode“), který poskytuje prostředky pro izolaci procesů a umožňuje vykonávání 32-bitových instrukcí. Vzhledem k faktu, že odpovědnost za start fyzického stroje přebírá hypervisor Xen, přebírá i odpovědnost za přechod do privilegovaného režimu. Pro paravirtualizovaný operační systém z toho vyplývá skutečnost, že je zaváděn do odlišného prostředí procesoru.

Změna úrovně oprávnění, o které již bylo hovořeno, implikuje nemožnost vykonávat privilegované instrukce uvnitř paravirtualizovaného jádra. Všechny tyto instrukce musí být nahrazeny odpovídajícími hypervoláními, které jsou poskytovány hypervisorem.

Další významná změna, mezi neparavirtualizovaným a paravirtualizovaným operačním systémem vyplývá ze skutečnosti, že paravirtualizovaný operační systém nespotřebovává všechen dostupný čas procesoru, ale pouze ten, který mu byl hypervisorem přidělen. V praxi to znamená, že jeho systémové hodiny musí být soustavně synchronizovány pomocí prostředků poskytovaných hypervisorem Xen.

3.2.2 Virtuální stroje

Virtuální stroje jsou v terminologii Xenu nazývány doménami. Existuje několik druhů domén, které budou popsány dále v textu.

Doména nula

Při zavádění hypervisoru Xen je současně zavedena doména, která byla Xenu předána ve formě bootloader modulu. Tato doména se nazývá doména nula[6] (dále v textu bude používáno označení dom0), nebo také privilegovaná doména.

Dom0 má v architektuře Xenu výsadní postavení. Součástí Xenu nejsou žádné ovladače zařízení ani uživatelské rozhraní. Ty jsou poskytovány nástroji běžícími v rámci dom0.

Důležitou úlohou této domény je obsluha fyzických zařízení, ke kterým má jako jediná přístup. Přístup k fyzickému zařízení je v neprivilegované doméně zprostředkován virtuálním ovladačem, který je rozdělen[7] na dvě části. Jedna část ovladače se nachází v domU (tato část je označována anglickým slovem „frontend") a druhá (společná) část se nachází v dom0 (tato část je označována anglickým slovem „backend"). Backend část ovladače komunikuje s ovladačem fyzického zařízení, jenž je součástí dom0. Tímto modelem je realizováno řízení přístupu k fyzickému zařízení (respektive ovladači fyzického zařízení), které je nezbytně nutné právě proto, že většina fyzických zařízení nepodporuje přístup z několika operačních systémů současně.

Dom0 je jediná doména, která je vytvářena přímo Xenem. Ostatní domény jsou vytvořeny pomocí nástrojů běžících v rámci dom0.

Doména U

Neprivilegovaná doména (dále v textu označována jako domU; toto označení vychází z anglického slova „unprivileged") jsou domény, které byly spuštěny dom0. Jejich standardní úroveň oprávnění jim neumožňuje úspěšně volat ta hypervolání, která pracují přímo s hardwarem, ačkoliv jim tato oprávnění mohou být přidělena.

---

[6] Technicky se jedná o první zaváděnou doménu

[7] Převzato z anglické terminologie Xenu, ve které jsou nazývány „split drivers".

Pro komunikaci domU a hardwaru je nutné, aby doména implementovala frontend část rozděleného ovladače. Pro každou kategorii zařízení (konzole, blokové zařízení, síťové zařízení, atd.) stačí implementovat jeden frontend ovladač. Díky tomu existuje velké množství portů operačních systémů, které neobsahují implementaci mnoha fyzických zařízení, na domU.

HVM domény

HVM domény (z anglického „Hardware Virtual Machine"), jsou nemodifikované domény virtualizované s podporou procesoru. Jejich provoz je možný pouze na fyzických strojích s procesorem podporující technologii Intel VT-x, nebo AMD-V. Hypervisorem Xen jsou tyto virtualizační technologie podporovány až od verze 3 (respektive 3.0.2 pro AMD-V).

Detailní popis principu fungování HVM domén je mimo rozsah této práce.

3.2.3 Info stránky

Info stránky obsahují základní informace o systému. Detailní popis všech položek struktur reprezentující info stránky lze nalézt v práci [1].

Start info

Start info stránkou jsou doméně prezentovány informace o systému, které se za jejího běhu nemění. Struktura `start_info_t`, kterou je tato stránka reprezentována, je doméně namapována do virtuálního adresového prostoru před jejím spuštěním. Adresa, na kterou byla struktura namapována, je doméně předána prostřednictvím registru `ESI`.

Start info stránka obsahuje informace o konzoli, iniciálních stránkovacích tabulkách a další.

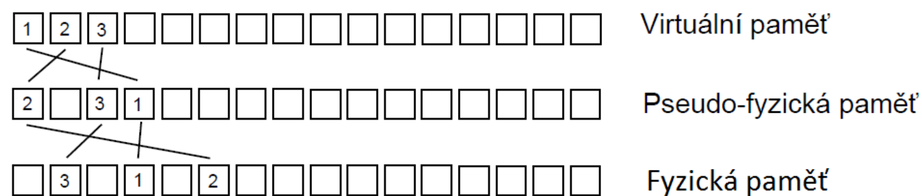
Shared info

Shared info stránkou jsou doméně prezentovány informace o systému, které se za jejího běhu mění. Struktura `shared_info_t`, kterou je tato stránka reprezentována, není namapována do virtuálního adresového prostoru domény. Fyzická adresa této struktury je uložena do jedné z položek start info stránky.

Shared info stránka obsahuje informace o virtuálních procesorech přidělených doméně, aktuálním času a kanálech událostí.

3.2.4 Pseudo-fyzický paměťový model

V operačních systémech podporujících mechanismy ochrany paměti má každý proces svůj adresový prostor. Z jeho pohledu má přístup k veškeré fyzické paměti. Hypervisor potřebuje vytvořit takovou představu i pro virtualizované operační systémy. Ta je vytvořena pomocí nové abstraktní vrstvy paměti, která je označována jako pseudo-fyzická. To znamená, že překlad mezi virtuálními a fyzickými adresami, je dvojúrovňový (viz Obrázek 4).



Obrázek 4: Druhy paměti Xenu (převzato z [1, obrázek 3.2])

Použitý třístupňový model byl zvolen z několika důvodů. Prvním důvodem je skutečnost, že mnoho existujících operačních systémů vychází z předpokladu, že fyzická paměť je souvislá (a číslovaná od fyzické adresy nula). Z tohoto důvodu by požadavek podpory pro tzv. „řídké“ adresové prostory ze strany domén znamenal buď pokles výkonu domény, nebo nutnost zásadních změn v jádře. Druhý důvod pro použití třístupňového modelu vychází z podpory migrace domén na jiné fyzické stroje. Těžko lze předpokládat, že by cílový fyzický stroj mohl migrující doméně poskytnout stejné fyzické rámce, které byly poskytnuty výchozím fyzickým strojem.

Pro převody mezi fyzickými a pseudo-fyzickými rámci slouží převodní tabulky *P2M* a *M2P*, které jsou součástí adresového prostoru každé domény. Adresa tabulky *P2M* je doméně předána prostřednictvím položky *mfn_list* struktury *start_info_t*. Tabulka *M2P* je součástí namapované oblasti hypervisoru Xen, díky čemuž je její virtuální adresa vždy stejná.

3.3 Pohled do zdrojových kódů hypervisoru Xen

Zdrojové kódy hypervisoru Xen vychází ze zdrojových kódů linuxového jádra. Od linuxového jádra k aktuální verzi hypervisoru Xen bylo provedeno mnoho změn; ne pro všechny změny je zpracována dokumentace[8]. Součástí práce na prototypové implementaci rozhraní hypervisoru Xen bylo, vzhledem k nedostatku relevantních zdrojů, i studium zdrojových kódů hypervisoru Xen. V této části je popsáno několik implementačních detailů vybraných funkčních bloků hypervisoru Xen.

3.3.1 Správa paměti

Pro potřeby validace paměti jsou ke každé stránce přiřazeny informace o typu a počtu referencí. Typem stránky je v každém okamžiku právě jeden ze vzájemně vyloučených typů:

- Adresář stránek (z anglického „page directory“)
- Stránkovací tabulka (z anglického „page table“)
- Tabulka lokálních deskriptorů (z anglického „local descriptor table“)
- Tabulka globálních deskriptorů (z anglického „global descriptor table“)
- Zapisovatelná stránka (z anglického „writeable“)
- Sdílená stránka

Typ stránky předurčuje množinu povolených operací stránky. Informace o typu stránky jsou uloženy prostřednictvím struktury `page_info`, která je součástí souboru `xen/include/asm-x86/mm.h`.

Tabulka struktur `page_info` je umístěna ve virtuálním adresovém prostoru hypervisoru Xen, který leží v horních 168 MB virtuální paměti. Jeho obsah je definován v hlavičkovém souboru `xen/asm-x86/config.h`.

V hypervisoru Xen je definováno množství převodních funkcí mezi různými druhy paměti a souvisejícími pomocnými strukturami. Například číslo fyzického

[8] Autorův názor na stav dokumentace zdrojových kódů hypervisoru Xen je, že je tristní.

rámce je zároveň index do tabulky struktur *page_info*. Všechny převody jsou umístěny v souboru *xen/asm-x86/page.h*.

Podrobné informace o typu stránky jsou nutné například pro implementaci přímo zapisovatelných stránkovacích tabulek nebo pro zaručení izolace mezi doménami.

3.3.2 Kanály událostí

Kanál událostí je abstraktní konstrukce, která je definovaná svými koncovými body. Zaslání zprávy skrze kanál od jednoho koncového bodu vyvolá doručení zprávy druhému koncovému bodu. Doména (i hypervisor) pracuje pouze s koncovými body.

Koncový bod kanálu událostí je reprezentován strukturou *evtchn*, která se nachází v souboru *xen/include/xen/shed.h* (tento soubor obsahuje i struktury reprezentující doménu a virtuální procesory):

```
struct evtchn
{
#define ECS_FREE         0 /* Channel is available for use.*/
#define ECS_RESERVED     1 /* Channel is reserved.*/
#define ECS_UNBOUND      2 /* Channel is waiting to bind to a
remote domain. */
#define ECS_INTERDOMAIN  3 /* Channel is bound to another doma-
in. */
#define ECS_PIRQ         4 /* Channel is bound to a physical
IRQ line. */
#define ECS_VIRQ         5 /* Channel is bound to a virtual IRQ
line. */
#define ECS_IPI          6 /* Channel is bound to a virtual IPI
line. */
    u8  state;             /* ECS_* */
    u8  consumer_is_xen;   /* Consumed by Xen or by guest? */
    u16 notify_vcpu_id;    /* VCPU for local delivery notifica-
tion */
    union {
        struct {
            domid_t remote_domid;
        } unbound;     /* state == ECS_UNBOUND */
        struct {
            u16            remote_port;
            struct domain *remote_dom;
        } interdomain; /* state == ECS_INTERDOMAIN */
```

```
        struct {
            u16             irq;
            u16             next_port;
            u16             prev_port;
        } pirq;         /* state == ECS_PIRQ */
        u16 virq;       /* state == ECS_VIRQ */
    } u;
#ifdef FLASK_ENABLE
    void *ssid;
#endif
};
```

Každý koncový bod obsahuje informaci o svém stavu (položka `state`), jejíž hodnotou je určeno, která část variantního záznamu je platná.

Struktura `domain` reprezentující doménu obsahuje pole reprezentující stránky, jejichž obsahem je struktura koncového bodu kanálu událostí. Maximální počet alokovaných stránek se odvíjí od velikosti proměnné typu long. Koncové body jsou spravovány hypervisorem; vždy je přidělen první volný, čímž jsou minimalizovány paměťové nároky na jejich správu. Při inicializaci domény je jeden koncový bod nastaven jako rezervovaný. Tím je zajištěna dostupnost minimálně jedné stránky struktur koncových bodů (na jednu stránku se jich vejde 128).

Doméně zůstává struktura koncového bodu utajena, komunikuje s ním prostřednictvím tzv. `portu`[9]. Port slouží jako index do dvoudimenzionálního pole. První dimenzí je pořadové číslo stránky; druhou je umístění fyzické struktury koncového bodu v rámci stránky. Postupným přidělováním koncových bodů je zajištěna nejen minimální možná paměťová náročnost, ale i jednoduchá převoditelnost mezi indexy a strukturami reprezentující koncové body kanálu.

3.3.3 Plánování

Jednou z klíčových vlastností podporovaných Xenem je multitasking. Hypervisor je zodpovědný za to, že každé aktivní doméně bude přidělen procesorový čas.

[9] [4] popisuje rozdíl mezi kanály a porty následujícím způsobem (volně přeloženo): „Z technického hlediska je kanál abstraktním propojením mezi dvěma koncovými body, zatímco port je identifikátor používaný k identifikaci koncového bodu, ke kterému je kanál připojen.“

Xen poskytuje abstraktní rozhraní pro plánovač, které je definováno strukturou *scheduler*, která obsahuje ukazatele na funkce obsluhující požadovanou funkcionalitu plánovače. Rozhraní je součástí souboru hlavičkového souboru *xen/include/xen/sched-if.h*.

Přidání nového plánovače hypervisoru Xen znamená vytvořit novou strukturu *scheduler*, implementovat požadované funkce a přidat ji do statického pole *schedulers*, které je součástí souboru *xen/common/schedule.c*.

Plánovač může být vybrán během zavádění domény použitím zaváděcího parametru *sched*. Hodnota přiřazená zaváděcímu parametru *sched* musí odpovídat hodnotě proměnné *opt_name* (součást struktury *scheduler*) některého z plánovačů statického pole *schedulers*.

4 Operační systém HelenOS

Cílem kapitoly je popis struktury operačního systému HelenOS architektury IA-32 a těch funkčních bloků systému, jejichž pochopení je nezbytně nutné pro porozumění změn popsaných dále v textu práce. Dokumentaci ostatních funkčních bloků lze nalézt na oficiálních stránkách projektu HelenOS [9].

Obsah této kapitoly vychází především z práce [1; kapitola 4] a oficiálních stránek projektu HelenOS [9]. Některé popisné pasáže textu vznikly jako volné parafráze práce [1].

Architektura operačního systému HelenOS vychází z mikrojaderné architektury. V mikrojaderné architektuře je objem kódu běžící v privilegovaném režimu minimalizován. Ostatní potřebné části systému (například ovladače zařízení, správa souborového systému, a další) jsou řešeny jako běžné uživatelské úlohy[10] (z anglického „task“). Uživatelským úlohám jsou jádrem poskytnuty prostředky pro vzájemnou komunikaci prostřednictvím meziprocesové komunikace.

V době tvorby této práce HelenOS obsahoval port na sedm architektur (AMD64/x86-64, ARM, IA-32, IA-64, 32-bit MIPS, 32-bit PowerPC a SPARC V). Osmým portem je práce [1], která pro tuto chvíli nebyla zahrnuta do hlavní vývojové linie.

4.1 Struktura jádra

V jádře systému HelenOS jsou odděleny architektonicky nezávislé části od architektonicky závislých částí. Zdrojové kódy mikrojádra systému HelenOS jsou rozděleny do tří hlavních adresářů:

- *arch/* – Architektonicky závislé části systému HelenOS. Pro každý port existuje vlastní podadresář (například `ia32xenh`).
- *genarch/* – Části systému, které jsou společné pro více architektur. Obsahuje například generickou implementace čtyřúrovňových stránkovacích tabulek (viz část kapitoly Stránkování v HelenOS).
- *generic/* – Architektonicky nezávislé části jádra.

[10] V systému HelenOS jsou procesy nazývány úlohami. Použitá terminologie vychází z terminologie běžně používané v mikrojaderných systémech.

Jednotlivé funkční bloky systému HelenOS (například správa paměti), jsou členěny do dalších podadresářů (zdrojové kódy pro správu paměti se nalézají v adresářích *mm*) v rámci výše popsané adresářové struktury.

4.2 Správa paměti

Správa paměti je jedním z funkčních bloků systému HelenOS, který je součástí privilegovaného jádra. Zdrojové kódy souborů správy paměti jsou umístěny ve složce *mm*.

4.2.1 Fyzická paměť

Fyzická paměť je rozdělena na bloky označované jako zóny. Součástí každé zóny jsou informace o její bázové adrese, velikosti, dostupných volných rámcích a příznaky určující její typ. Fyzické rámce zóny jsou spravovány tzv. „Buddy alokátorem“, který udržuje informace o přidělených a volných blocích fyzické paměti.

V architektuře IA-32 jsou zóny vytvořeny v architektonicky závislé části fyzického alokátoru[11], konkrétně ve funkci *init_e820_memory*. Jak už název funkce napovídá, počet a velikost jednotlivých zón vychází z mapy paměti získané prostřednictvím služby PC BIOSu nazývané *e820*. Maximální velikost dostupné fyzické paměti je omezena konstantou *PHYSMEM_LIMIT32*. K omezení dostupné fyzické paměti existují dva důvody:

- Způsob vytváření mapování v jádře (viz Mapování jádra).
- Uspokojení paměťových nároků pro zařízení mapovaná do paměti.

4.2.2 Virtuální paměť

Virtuální adresový prostor každé[12] úlohy je rozdělen do bloků, které jsou označovány jako oblasti. Oblast je souvislý blok virtuálního adresového prostoru, který má stejné vlastnosti i způsob mapování na fyzické paměti. Každé oblasti je přiřazen tzv. „*paměťový backend*“, kterým jsou obsluhovány například vý-

[11] arch/src/frame.c

[12] Virtuální adresový prostor pro jádro je označen jako AS_KERNEL.

padky stránek. V současné době jsou systémem HelenOS podporovány tyto *paměťové backendy*:

- *Elf* – Pro mapování obrazu souboru v ELF formátu.
- *Anon* – Při výpadku stránky je alokován nový rámec, na který je stránka následně namapována.
- *Phys* – Pro namapování souvislého bloku fyzické paměti.

Segmentace

Segmentace je neopomenutelnou součástí ochrany paměti architektury IA-32. Jádro používá tzv. „ploché" segmenty, definované proměnnou *gdt* reprezentující globální tabulku deskriptorů (z anglického „global descriptor table", dále v textu označovanou jako *GDT*). Tato proměnná, která je součástí architektonicky závislé části jádra, je definována v souboru *pm.c*.

Stránkování

Součástí systému HelenOS je rozhraní pro správu mapování (vložení, vyhledání, mazání), reprezentuje struktura *page_mapping_operations_t*. Toto rozhraní je implementováno instancí *page_mapping_operations*, kterou jsou spravovány generické čtyřúrovňové stránkovací tabulky.

Instance *page_mapping_operations* je použita pro správu mapování v architektuře IA-32. Parametrizací této instance vzniknou stránkovací tabulky, které mají dvě efektivní úrovně stránkování. Obě úrovně stránkovacích tabulek obsahují 1024 záznamů. Velikost jedné stránky je 4 KB.

Součástí implementace architektonicky nezávislé části správy paměti jsou abstraktní příznaky stránkovacích tabulek. Jejich hodnoty neodpovídají skutečným hodnotám příznaků stránkovacích tabulek konkrétní architektury. Generické funkce[13] pro získání nebo nastavení příznaků položky stránkovacích tabulek slouží jako ukazatel na architektonicky závislé protějšky. Jejich součástí je konverze z abstraktních hodnot příznaků na hodnoty platné pro danou architekturu.

---

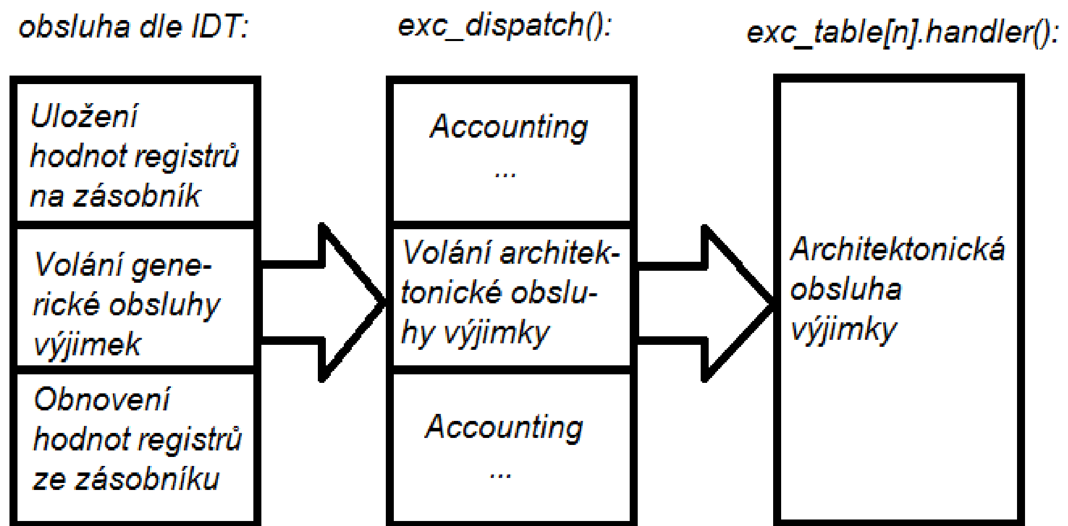
[13] Které jsou použity instancí page_mapping_operations

Mapování jádra

Virtuální adresový prostor jádra je vytvořen identickým mapováním dostupné fyzické paměti na virtuální adresy od 2 GB[14] výše. Mapování použité u výchozí architektury `ia32` omezuje[15] maximální velikost dostupné fyzické paměti na nejvýše 2 GB[16]. Převod mezi fyzickou a virtuální adresou jádra je realizován prostřednictvím maker `KA2PA` a `PA2KA` (přičtení, respektive odečtení 2 GB).

4.3 Zpracování výjimek

Mechanismus obsluhy výjimek zobrazuje Obrázek 5. Obsluhu lze z architektonického hlediska označit jako tříúrovňovou.



Obrázek 5: Architektura obsluhy výjimek HelenOS

První úroveň je architektonicky závislá a je implementována v jazyce assembler. Pro architekturu IA-32 jsou na zásobník (na nižší adresy, než jsou uloženy hodnoty registrů vložené procesorem) uloženy hodnoty tzv. „obecných“ registrů (z anglického „general purpose registers“) a hodnoty registrů `GS`, `FS`, `ES` a `DS` (tzv. „selektor“ registry, anglicky označované jako „selector registers“). Uložené hodnoty jsou reprezentovány pomocí struktury `istate_t`. Následují operace pro

[14] Od adresy 0x80000000
[15] Pro současnou hlavní vývojovou větev operačního systému HelenOS již toto omezení neplatí.
[16] 4.2.1: Velikost dostupné paměti je omezena konstantou PHYSMEM_LIMIT32

přípravu skoku do druhé úrovně, který je realizován instrukcí `call`. Po návratu z druhé úrovně jsou obnoveny hodnoty registrů hodnotami uloženými na zásobníku. Po obnovení všech uložených hodnot je vykonána instrukce `iret`.

Druhá úroveň je architektonicky nezávislá a je implementována v jazyce C. Zdrojové kódy související s implementací druhé úrovně se nachází v souboru `kernel/generic/src/interrupt/interrupt.c`. Druhá úroveň slouží k architektonickému odclonění zpracování výjimek. K tomuto účelu slouží struktura `exc_table_t`, která obsahuje odkaz na architektonickou obsluhu výjimky mající dva parametry – číslo výjimky a odkaz na strukturu `istate_t`. Architektonické obsluhy všech výjimek zahrnuje globální proměnná `exc_table` obsahující pole výše zmíněných struktur. Pro nastavení architektonických obsluh slouží funkce `exc_register`. Samotné zpracování druhé úrovně má na starost funkce `exc_dispatch`. V této funkci jsou nejprve provedeny výpočty o využití času procesoru pro aktuálně běžící vlákno (a úlohu). Následně je proveden skok do třetí úrovně. Poslední akcí, která následuje po provedení architektonické obsluhy výjimky, jsou provedeny výpočty doby spotřebované pro provedení obsluhy přerušení.

Třetí úroveň je architektonicky závislá a z druhé úrovně jsou jí předány dva parametry – číslo výjimky a odkaz na strukturu `istate_t`. Třetí úroveň je místem skutečné obsluhy vlastní výjimky.

4.4 Vlákna a úlohy

Zdrojové kódy souborů úloh a vláken jsou umístěny ve složkách `proc`. Úloha je reprezentována architektonicky nezávislou strukturou `task_t`, jejíž součástí je odkaz na architektonicky závislou strukturu `task_arch_t`. Stejný model je použit i pro vlákno (`thread_t` a `thread_arch_t`).

4.5 Plánování

Preemptivní plánovač je reprezentován funkcí `schedule`. Plánovací jednotkou plánovače v systému HelenOS jsou vlákna (není rozlišováno mezi uživatelskými a jadernými). Po doručení přerušení od časovače je zavolána funkce `schedule`,

jejíž součástí je i výběr vlákna, kterému bude přidělen následující procesorový čas

V případě, že vybrané vlákno nepatří do seznamu vláken právě běžícího procesu, je volána funkce *before_task_runs*. Před přepnutím kontextu je vždy volána funkce *before_thread_runs*. Součástí obou funkcí je volání architektonicky závislých variant funkcí se sufixem *_arch*.

4.6 Spuštění uživatelských úloh

Načtení a spuštění iniciálních uživatelských úloh probíhá v rámci jaderného vlákna *kinit*, které je vytvořeno a spuštěno po inicializaci všech jaderných subsystémů. Vlákno je generické, to znamená použité pro všechny dosavadní architektury operačního systému HelenOS.

Pro každý modul (vyjma posledního modulu, který je považován za RAM disk) je vytvořena uživatelská úloha s hlavním uživatelským vláknem, jejímž vstupním bodem je funkce *uinit*. Vlákna všech vytvořených úloh jsou následně přidána do fronty připravených vláken.

Funkce uinit

Funkcí *uinit* je obalena funkce „*main*" každého hlavního uživatelského vlákna. Tato funkce je generická a přijímá jeden parametr – odkaz na strukturu *uspace_arg_t*. Tato struktura je okopírována do adresového prostoru uživatelské úlohy a předána funkci *userspace*, která je definována pro každou architekturu. Úkolem této funkce je nastavit stupeň oprávnění vlákna a předat řízení uživatelské funkci „*main*".

5 Analýza

Cílem kapitoly je analyzovat změny, které bude třeba provést v systému HelenOS pro prototypovou implementaci paravirtualizačního rozhraní Xenu. V prvé řadě je třeba určit rozsah prototypové implementace. V části o architektuře Xenu (viz 3.2) jsou popsány rozdíly mezi jednotlivými druhy domén. Doména inicializovaná hypervisorem je označována jako dom0. Tato doména má výsadní postavení mezi doménami, které je dáno především tím, že jsou na ni delegovány určité funkce v rámci systému jako celku (ovladače zařízení, spouštění dalších domén, a další). Logicky se tedy nabízí myšlenka implementovat takový rozsah paravirtualizačního rozhraní, který je dostatečný pro provozování některé z existujících domén nula. Seznam známých dom0 shrnuje následující tabulka:

Distribuce	Verze	Balíček
Debian	4.0, 5.0, 6.0	xen-linux-system-2.6-xen-686 (IA-32) xen-linux-system-2.6-xen-amd64 (AMD64)
Fedora	16	
OpenSUSE	10.x, 11.x	*kernel-xen*
Redhat Enterprise Linux	5.x	kernel-xen
SUSE Linux Enterprise Server	10.x, 11.x	*kernel-xen*
Ubuntu	11.10	xen-hypervisor-4.1-i386 (IA-32) xen-hypervisor-4.1-amd64 (AMD64)
Xen Cloud Project (XCP)	Všechny	součástí platformy
XenServer	Všechny	součástí platformy
NetBSD	5.1	

Tabulka 1: Známé domény 0 (převzato z [5])

Každá z uvedených domén používá větší část funkcí poskytovaných hypervisor Xen. Celkem jich existuje několik desítek[17]. Úplná implementace jednoho hypervolání neznamená implementaci jediné konkrétní funkce, ale spíše implementaci funkcionality jednoho subsystému (správa paměti, události[18], sdílení paměti, …) Podpora jakékoli dom0 by znamenala implementaci stovky dílčích funkcí, které byly týmem okolo Xenu vyvíjeny několik let. To je nade vší pochybnost nad rámec jedné diplomové práce.

Jedinou zbývající možností je implementace takového rozhraní, které umožní provozování některé z již existujících domU. DomU, jenž je součástí zdrojových kódů hypervisoru, není vhodná kvůli příliš malému pokrytí využívaných hypervolání. Navíc je důležité zvolit takovou doménu, která je pokryta dokumentací a která je dobře odladitelná. Všechny tyto požadavky splňuje port HelenOS vytvořený v rámci práce [1].

Konečným cílem prototypové implementace rozhraní Xen je implementovat všechny varianty hypervolání, které jsou nezbytně nutné k provozování domény [1]. Nad rámec tohoto rozhraní je nutné implementovat některé funkce, které jsou standardně prováděny v dom0. Protože jsou další domény inicializovány a spouštěny dom0, je zřejmé[19], že součástí prototypové implementace nebude běh několika domén současně.

V další části kapitoly probereme změny nutné pro prototypovou implementaci paravirtualizačního rozhraní Xenu. Tyto změny lze rozčlenit do tří různých, ale ne zcela disjunktních kategorií. První kategorií změn jsou úpravy sloužící k úspěšnému nabootování domény (spuštění funkce `kinit` domény). Druhou kategorií změn jsou pak úpravy vedoucí k rozšíření služeb (hypervolání) nabízených prototypovou implementací rozhraní Xenu. Poslední kategorií změn jsou

---

[17] Aktuální počet poskytovaných hypervolání je 37.
[18] Například hypervolání HYPERCALL_event_channel_op poskytuje veškerou funkcionalitu (vytváření, propojení, rušení, atd.) související s událostmi (events).
[19] Podpora běhu několika domén by za tohoto stavu vyžadovala zásahy do architektury načítání iniciální domény. Místo jedné domény by bylo načteno více domén současně. Tato část implementace by zcela pozbyla smyslu v okamžiku rozšíření implementace o podporu některé z dom0.

úpravy suplující chybějící dom0. Cílová architektura prototypové implementace je IA-32 a Xen verze 4.1.

5.1 Analogie mezi HelenOS a Xenem

Virtualizované prostředí Xenu pracuje s entitami typu doména, kterým jsou podřízeny entity typu virtuální procesor. V rámci entity typu doména jsou spravovány zdroje typu stránkovací tabulky, sdílené stránky paměti, kanály událostí apod. Entity typu virtuální procesor jsou plánovačem hypervisoru plánovány pro běh na reálném procesoru.

Prostředí operačního systému HelenOS pracuje s entitami typu proces[20], kterým jsou podřízeny entity typu vlákno. V rámci entity typu proces jsou spravovány zdroje jako stránkovací tabulky, sdílené stránky paměti, apod. Entity typu vlákno jsou plánovačem HelenOS plánovány pro běh na reálném procesoru.

Z tohoto pohledu lze říci, že analogií domén jsou v prostředí operačního systému HelenOS procesy, a analogií virtuálních procesorů jsou vlákna. Procesy i vlákna jsou v operačním systému HelenOS rozděleny na společnou část a architektonicky závislou část. Díky tomu lze využít stávající infrastruktury procesů a vláken a rozšířit ji tak, aby v architektonické části pokrývala potřeby domén a virtuálních procesorů.

5.2 Předpoklady a změny úspěšného nabootování domény

5.2.1 Relokace jádra HelenOS

Hypervisor Xen je mapován do horních 168 MB[21] virtuálního adresového prostoru každé domény. Doména s tímto mapováním dále pracuje, proto je nezbytné jej implementovat i do prototypové implementace rozhraní hypervisoru.

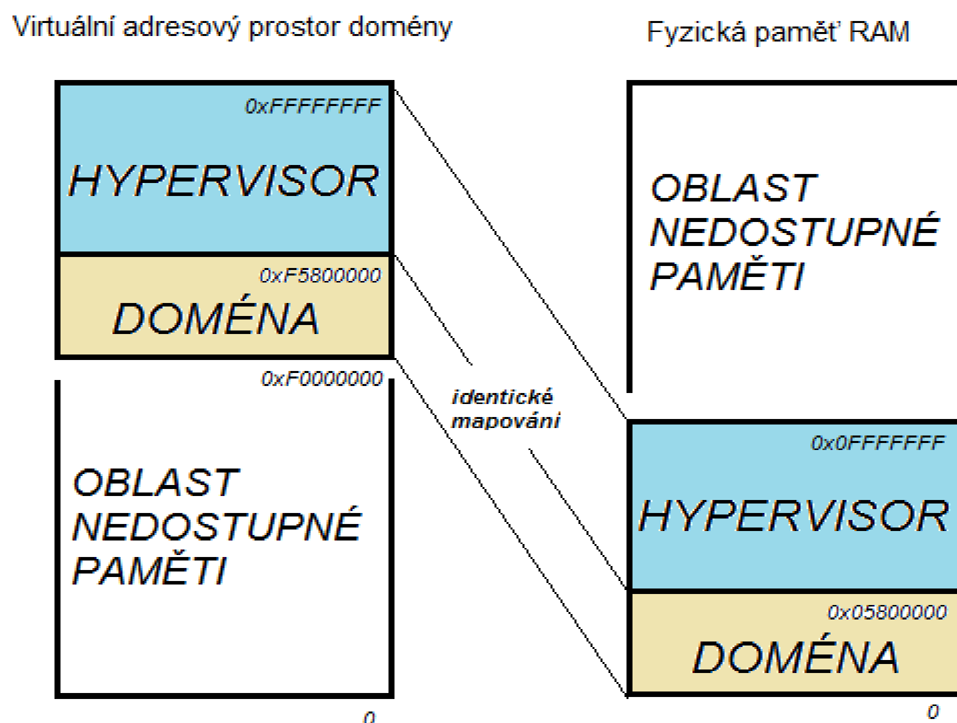
HelenOS v implementaci portu na architekturu IA-32 identicky mapuje dostupnou fyzickou paměť do horních 2 GB paměti (viz část věnovaná Mapování jádra HelenOS). Posun kódu jádra do horních 168 MB virtuálního adresového prosto-

[20] V mikrojaderné architektuře jsou procesy nazývány úlohami.

[21] Od virtuální adresy 0xF5800000

ru bez dalších změn, by kvůli způsobu převodu adres vyžadoval 2 GB dostupné fyzické paměti. Přitom samotné testovací doméně [1] stačí pouze několik desítek MB přidělené fyzické paměti. Při pouhé relokaci jádra, které samo o sobě potřebuje pouze několik MB fyzické paměti, by velká část fyzické paměti zůstala nevyužita.

Jako schůdné řešení se jeví posun začátku identického mapování hypervisoru směrem k vyšším adresám. Posun mapování znamená změnu (v tomto případě zmenšení, protože se jedná o posun směrem k vyšším adresám) dostupného bloku fyzické (i virtuální) paměti. Například posun hranice mapování na horní 1 GB paměti znamená požadavek na 1 GB fyzické paměti a dostupnost maximálně 1 GB fyzické paměti bez ohledu na její skutečné množství. Dle názoru autora je pro prototypovou implementaci více limitující minimální požadovaná velikost fyzické paměti, než její maximální dostupnost. Z tohoto důvodu byla hranice posunuta do horních 256 MB paměti (168 MB pro jádro, 88 MB pro doménu, viz Obrázek 6). Implementace posunu identického mapování je rozebrána v kapitole implementace (viz 6.2.1).



Obrázek 6: Virtuální a fyzická paměť z pohledu hypervisoru

5.2.2 Stránkování HelenOS

Hypervisor Xen pracuje pouze s doménami podporující metodu `PAE`. Současná implementace architektury IA-32 metodu `PAE` nepodporuje, protože existuje omezení maximální velikosti dostupné fyzické paměti. Snížením této hranice na 256 MB (viz 5.2) nedojde ke zpochybnění nutnosti implementace podpory metody `PAE` (která je detailně probrána v části 6.2.2).

5.2.3 Načtení zaváděné domény

Doména a její volitelný modul jsou hypervisoru předány ve formě modulů. Způsob načítání jednotlivých modulů v operačním systému HelenOS (popsaný v části 4.6) nelze použít pro zavedení domény hned z několika důvodů:

- *Způsob načítání jednotlivých uživatelských úloh* – Úlohy jsou načítány architektonicky nezávislými funkcemi umístěnými v `kernel/generic/-src/proc/program.c`. Adresový prostor úlohy obsahuje pouze mapování kernelu. Mapování pro samotnou úlohu není vytvořeno (mapování je vytvořeno až při přístupu k nenamapované části úlohy, tzv. „on demand“). Mapování domény je striktně dáno a musí být vytvořeno předtím, než je doména poprvé naplánována.
- *Omezení rolí a počtu modulů* – Hypervisoru jsou předány maximálně dva moduly. Prvním modulem je doména, druhý modul je volitelný a reprezentuje modul předaný doméně.

Kvůli výše uvedeným důvodům je nutné vytvořit mechanismus pro načítání modulů hypervisoru. Tento mechanismus popisuje část 6.3.

Parametry zaváděné domény

Obraz zaváděné domény musí obsahovat některé parametry nutné pro zavedení. Mezi tyto parametry patří například informace o požadované verzi rozhraní hypervolání a pseudo-fyzická adresa tzv. „`hypercall page`“. Tyto parametry mohou být součástí oblasti[22] „`_NOTE`“, nebo sekce `__xen_guest`.

[22] Sekce, nebo segment

V Doméně [1] je pro předání zaváděcích parametrů použita oblast „_NOTE“[23]. Prototypová implementace musí být schopná tyto parametry načíst, a proto musí obsahovat funkce pro parsování sekce „_NOTE“. Podpora parsování sekce ___xen_guest není důležitá, a proto není součástí prototypové implementace.

Obraz zaváděné domény

Hypervisor Xen podporuje komprimované domény. Komprimované domény jsou před zpracováním (načtení parametrů, vytvoření iniciálního mapování) vždy dekomprimovány. Algoritmus dekomprese nijak nesouvisí s podporou paravirtualizačního rozhraní Xen. Podporovány budou pouze nekomprimované obrazy domén.

5.2.4 Iniciální mapování

Každá doména vyžaduje předem dané identické iniciální mapování. Počáteční virtuální adresa tohoto mapování je nejnižší bázová adresa ze všech ELF sekcí obrazu domény. Tato adresa je poté zaokrouhlena na celé 4 MB směrem dolů. Iniciální mapování lze rozdělit na po sobě následující oblasti:

- Obraz jádra operačního systému, který je tvořen namapováním všech ELF sekcí.
- Zaváděný modul (volitelné).
- Tabulka P2M platná pro tuto doménu. Tabulka obsahuje převod pro všechny rámce, které má doména k dispozici.
- Struktura start_info_t.
- Stránkovací tabulky popisující iniciální mapování.
- Zaváděcí zásobník
- Minimálně 512 KB volné paměti.

Každá tato oblast je zarovnána na 4 KB. Konec oblasti volné paměti je navíc zarovnán na 4 MB. Dále je namapována oblast hypervisoru začínající na adrese 0xF5800000[24]. Přímo od virtuální adresy 0xF5800000 začíná sdílená tabulka M2P (mapovaná pouze pro čtení).

[23] Segment, jehož typ je PT_NOTE anebo sekce, jejíž typ je SHT_NOTE.
[24] Adresa počátku horních 168 MB paměti

Vytvoření tohoto mapování před samotným spuštěním domény je nutnou součástí prototypové implementace. Oblasti tabulky `P2M`, struktury `start_info_t` a iniciálních stránkovacích tabulek musí být vhodně inicializovány.

Inicializace tabulky P2M

Součástí prototypové implementace je pevně stanovená velikost přidělené fyzické paměti. Vyšší úroveň abstrakce umožňuje přidělit několik bloků fyzické paměti, ale jeden souvislý blok paměti není žádným důležitým omezením. Jeden souvislý blok umožní jednodušší vyplnění tabulek `P2M`, `M2P` (které jsou doménou použity k namapování zbývající dostupné fyzické paměti) a snadnější převod mezi virtuální adresou domény, virtuální adresou hypervisoru a skutečnou fyzickou adresou.

Struktura start_info_t

Struktura `start_info_t` poskytuje základní informace o systému, které se za běhu nemění. Položky této struktury je proto nutné inicializovat předtím, než je doména poprvé naplánována. Všechny položky této struktury budou inicializovány dle dokumentace. Výjimku tvoří pouze tyto položky:

- `store_mfn`, `store_evtchn` – Sdílená stránka a kanál událostí pro komunikaci s databází XenStore. Databáze XenStore je spravována démonem, který je součástí dom0.
- `cmd_line` – Obsah příkazové řádky z konfiguračního souboru není z pohledu prototypové implementace důležitý.
- `Console` – Informace o ovladači konzole. Vzhledem ke skutečnosti, že hypervisor musí suplovat backend část ovladače konzole za doménu nula, obsahuje informace pro domU.

Struktura shared_info_t

Struktura `shared_info_t` poskytuje dynamicky měněné informace o systému. Odkaz na strukturu `shared_info_t` je součástí struktury `start_info_t`. Obsahuje informace o událostech virtuálního procesoru a informace o časech. Z položek struktury `shared_info_t` je nutné inicializovat pouze položky související s kanály událostí. Inicializovaná doména také předpokládá to, že po svém spuštění bude příjem událostí z hypervisoru maskován.

Iniciální stránkovací tabulky

Stránkování je důležitou součástí ochrany paměti domény. Iniciální stránkovací tabulky musí reflektovat iniciální mapování domény popsané dříve v textu.

Stránkovací tabulky musí být fyzicky umístěny ve fyzických rámcích alokovaných pro doménu. Důvodem je nutnost přístupnosti stránkovacích tabulek pro doménu. Kdyby stránkovací tabulky ležely mimo fyzické rámce domény, tak by je nebylo možné doménou procházet, protože obsahem stránkovacích tabulek jsou fyzické adresy. Dalším omezením iniciálních stránkovacích tabulek je jejich umístění ve virtuálním adresovém prostoru domény. Toto umístění je dané iniciálním mapováním.

Pro splnění těchto požadavků je nezbytně nutné vytvořit mechanismus převodu mezi fyzickou adresou, virtuální adresou domény a virtuální adresou hypervisoru. Díky skutečnosti, že je doméně přidělen pouze jeden souvislý blok fyzické adresy (a virtuální adresový prostor hypervisoru je namapován identickým mapováním), je možné identicky převést virtuální adresu v adresovém prostoru domény na virtuální adresu virtuálního prostoru hypervisoru takto:

$$VA_{hypervisor} = PA2KA\left(PA_{počátek\ domény}\right) + (VA_{doména} - VA_{počátek\ domény})$$

Několika úpravami lze získat vztah pro převod virtuální adresy domény na fyzickou adresu:

$$PA_{doména} = PA_{počátek\ domény} + (VA_{doména} - VA_{počátek\ domény})$$

Pomocí těchto převodů lze iniciální stránkovací tabulky vytvořit přímým zápisem do paměti.

Změny ve stránkovacích tabulkách

Hypervisor Xen podporuje tři režimy úprav stránkovacích tabulek:

- *Plně paravirtualizovaná varianta* – Doména respektuje fakt, že jsou její stránkovací tabulky namapovány pouze pro čtení a pro každou změnu (nebo skupinu změn) použije příslušné hypervolání.
- *Přímo zapisovatelné stránkovací tabulky* – Doména zapisuje přímo do stránkovacích tabulek. Tento zápis je zachycen a odemulován.
- *Stínové stránkovací tabulky* – Doména používá stránkovací tabulky, které ovšem nejsou přímo používány procesorem. V případě změny stínových stránkovacích tabulek je tato změna Xenem propagována do skutečných stránkovacích tabulek a naopak. Jsou používány pro *HVM* domény.

Hlavní nevýhodou přímo zapisovatelných stránkovacích tabulek je nutnost udržovat seznam stránek, jejichž obsahem jsou stránkovací tabulky domény. Hypervisor Xen uchovává typ každého fyzického rámce, čímž je zajištěn vyšší stupeň ochrany paměti (více informací je popsáno v části 3.3.1). Pro prototypovou implementaci není nutné tuto ochranu implementovat, protože nelze předpokládat úmyslné narušení paměti doménou. Dalším důvodem, proč použít plně paravirtualizovanou variantu je fakt, že testovací doména [1] tuto variantu úpravy stránkovacích tabulek využívá. Implementace přímo zapisovatelných stránkovacích tabulek by tak nebyla, bez dodatečných úprav domény, otestována.

Nevýhody stínových stránkovacích tabulek se shodují s nevýhodami přímo zapisovatelných stránkovacích tabulek. Samotným hypervisorem Xen jsou použity pro *HVM* domény. Podpora plné virtualizace je nad rámec této práce.

5.2.5 Přechod na úroveň oprávnění domény

Architektura IA-32 implementuje čtyři úrovně ochrany, které jsou hierarchicky uspořádány od nejvíce privilegované (privilegovaný režim, který bývá obvykle označován jako „ring 0“[25]) po nejméně privilegované (uživatelský režim, který bývá obvykle označován jako „ring 3“). Lze si je představit jako soustředné kružnice s různě velkými poloměry. Čím nižší číslo úrovně oprávnění, tím nižší

[25] Českým ekvivalentem je slovní spojení „úroveň oprávnění nula“.

poloměr a tedy tím blíže ke středu kružnice. Čím blíže ke středu je úloha provozována, tím více privilegovaných operací může používat.

Hypervisor je provozován na úrovni nula a doména na úrovni jedna (viz Obrázek 3, který se nachází v části 3.2.1). Pro přechod na tuto úroveň lze použít mechanismus používaný pro přechod na úroveň určenou pro aplikace (v portu *ia32*). Kromě patřičné změny iniciálního kódového a datového segmentu domény je třeba uložit virtuální adresu (domény) struktury *start_info_t* do registru *ESI* těsně před změnou úrovně oprávnění.

5.3 Implementované funkce

5.3.1 Omezení prototypové implementace

Rozsah prototypové implementace je určen především potřebami testovací domény. Implementace doménou nevyužívaných hypervolání by přinesla rizika budoucího rozvoje v podobě neotestovaného kódu. Cílem práce je vytvořit stabilní prototypovou implementaci paravirtualizovaného rozhraní, na kterou bude možné navázat dalšími pracemi.

Z důvodů nutnosti implementovat některé služby nad rámec služeb poskytovaných hypervisorem standardně, je prototypová implementace virtualizačního rozhraní omezena na podporu jednoho virtuálního procesoru v rámci domény.

5.3.2 Podpora segmentace

Segmentace je neopomenutelnou součástí ochrany paměti architektury IA-32. Hypervisor Xen doménám poskytuje několik standardních globálních deskriptorů segmentů. Tyto deskriptory jsou v *GDT* nainstalovány od indexu 7168 (maximální počet segmentů je 8196). Další deskriptory (do indexu 7168) lze nainstalovat hypervoláním *HYPERVISOR_set_gdt*. Vytvoření standardních globálních deskriptorů a implementace hypervolání *HYPERVISOR_set_gdt* je součástí prototypové implementace.

Podpora doménou definovaných deskriptorů v *GDT* implikuje nutnost vkládání deskriptorů domény před jejím naplánováním a ukládání, respektive obnovení hodnot všech segmentových registrů při výměně kontextu vlákna. Vkládání re-

gistrů lze zajistit buď uchováváním celé *GDT* pro každou doménu a změnou registru *GDTR*, nebo přepisováním jediné systémové *GDT*. Vzhledem k paměťové náročnost jedné *GDT* (přesně 64 KB) a omezení maximální dostupné paměti, byla zvolena varianta s přepisováním jediné systémové *GDT*. Pro změny v uložení, respektive obnovení obsahu kontextu musí být provedeny změny v architektonicky závislých funkcích pracující s kontextem.

5.3.3 Výjimky

Mechanismus výjimek procesoru je v hypervisoru Xen použit pro hypervolání. Z tohoto důvodu je nezbytně nutné registrovat obsluhu přerušení (hypervolání) do tabulky vektorů přerušení. Tato obsluha bude sloužit pouze k vyvolání funkce implementující požadované hypervolání.

Samotná obsluha hypervolání k reálnému provozování domény nestačí. Doména musí mít možnost zpracovat přerušení, která vznikla v době jejího běhu. Hypervolání *HYPERVISOR_set_trap_table* slouží k nastavení obsluh (domény) pro jednotlivá přerušení.

Adresy funkcí obsluhujících přerušení jsou uloženy v tabulce vektorů přerušení (vychází z anglického označení „Interrupt descriptor table“, dále v textu bude označována zkratkou *IDT*), která je indexována číslem přerušení. Přímé přepisování *IDT* před každým naplánováním domény není možné z důvodu nemožnosti garance neměnnosti řídících registrů procesoru. Obsluha přerušení by mohla být přerušena[26] přeplánováním domény, čímž by mohlo dojít ke změnám hodnot v řídících registrech. To by, například při výpadku stránky (kde je adresa způsobující výpadek stránky uložena v registru *cr2*), mohlo vést k zacyklení domény.

Je tedy nezbytně nutné přerušení částečně zpracovat hypervisorem a teprve poté vyvolat obsluhu domény. Vyvolání obsluhy domény lze provést přímým

---

[26] Zakázání přerušení po dobu běhu obsluhy domény možné je, ale vedlo by ke zranitelnosti vůči útoku typu odepření služeb. Pro jeho realizaci by stačilo, kdyby útočná doména neopustila obsluhu přerušení.

skokem na adresu obsluhy domény v obsluze přerušení hypervisoru nebo přepsáním části zásobníku obsahujících návratové hodnoty.

Charakteristika obsluhy přerušení systému HelenOS (lze ji rozdělit na tři navazující části: inicializace, obsluha a úklid) neumožňuje jednoduché a funkční použití přímého skoku; zato umožňuje jednoduché přepisování zásobníku. Samotné přepsání zásobníku však nestačí. Přepsáním hodnot *CS* a *IP* se po provedení instrukce *iret* začne vykonávat nastavená obsluha přerušení domény, avšak i z obsluhy je nutný návrat na adresu původně uloženou na zásobníku. To lze zajistit okopírováním původních návratových hodnot na zásobník domény. Návrat z obsluhy přerušení domény pak znamená návrat na adresu, na které bylo přerušení původně vyvoláno[27]. Navržený mechanismus zpracování přerušení ilustruje Obrázek 14 (tento obrázek se nachází v části 6.5.3).

Mechanismus vyvolání obsluhy přerušení je stejný, ať už bylo přerušení vyvoláno v doméně, či v její aplikaci. Rozdílný je způsob zjištění zásobníku domény. V případě, že je přerušení vyvoláno v doméně, je použit zásobník, jehož adresa je uložena na zásobníku. Naopak v případě, kdy je přerušení vyvoláno vykonáváním kódu aplikace, je nutné použít adresy (segment a adresa) nastavené hypervoláním *HYPERVISOR_stack_switch*.

5.3.4 Kanály událostí

Události jsou standardním mechanismem pro doručování informací od hypervisoru doméně, anebo mezi doménami. Koncepčně jsou velmi podobné UNIXovým signálům. Události lze rozdělit do tří kategorií:

- *Mezidoménová komunikace* – Události sloužící ke komunikaci mezi doménami.
- *Fyzické IRQ* – Události napojené na fyzické IRQ. Jsou využívány například doménou nula pro přístup k fyzickým zařízením.
- *Virtuální IRQ* – Události reprezentující IRQ virtuálních zařízení, jako například časovač.

[27] Nebo na adresu následující instrukce v závislosti na typu přerušení.

Ačkoliv cílem prototypové implementace je úspěšně provozovat jednu doménu, je nezbytně nutné implementovat podporu mezidoménové komunikace. Tato nutnost vyplývá z mechanismu fungování Xenovské konzole, která je spravována na základě komunikace mezi domU a dom0. Konkrétní implementované varianty hypervolání *HYPERVISOR_event_channel_op* jsou rozebrány v implementaci (viz 6.5.4).

Paravirtualizovaná doména [1] není dom0. Její ambicí není spravovat ani komunikovat s fyzickými zařízeními, proto podpora fyzických *IRQ* není součástí prototypové implementace.

Prototypová implementace musí obsahovat taková virtuální zařízení, která jsou nezbytně nutná k provozování testovací domény. Virtuální časovač je virtuální zařízení domény plně nahrazující časovač fyzický, který je použit pro plánování a synchronizaci. Plánování i synchronizace jsou důležitým aspektem infrastruktury domény, proto musí být generovány událostí virtuálního časovače. Takt virtuálního časovače bude pevně svázán s taktem časovače fyzického.

Za doručení a správu kanálu událostí zodpovídá hypervisor. Události jsou doručeny pomocí tzv. callbacků, jejichž adresa (segment a adresa) je registrována hypervoláním *HYPERVISOR_set_callbacks*.

Událost virtuálního časovače i událost typu mezidoménová komunikace nastává v okamžiku zpracování přerušení. Díky tomuto faktu lze pro vyvolání callbacků domény použít stejný mechanismus jako u obsluhy přerušení. Nevýhodou tohoto mechanismu je ztráta asynchronnosti doručování událostí.

5.3.5 Grant tabulky

Mechanismus tzv. „grant tabulek“ (vychází z anglického „grant tables“) je použit pro sdílení paměti mezi doménami. Vzhledem k faktu, že prototypová implementace nepodporuje více domén spuštěných současně, bude z tohoto mechanismu implementována pouze inicializace grant tabulek.

5.4 Změny suplující chybějící doménu nula

5.4.1 Backend ovladač konzole

Zařízení jsou doménám zpřístupněna pomocí tzv. rozdělených ovladačů (vychází z anglického označení „split drivers“). Rozdělený ovladač se skládá ze dvou částí – frontend a backend. Backend část, mající přístup k hardwaru (obvykle zprostředkovaný ovladačem reálného zařízení) se typicky nachází v dom0. Frontend část je součástí domU a její přístup k hardwaru je zprostředkován backend částí ovladače. Tyto části spolu komunikují prostřednictvím kanálu událostí a sdílené paměti.

Prototypová implementace nebude podporovat žádnou z existujících dom0, což implikuje nutnost zpracování požadavků frontend části ovladačů domU jiným způsobem. Jako nejjednodušší způsob se jeví zpracovat některé požadavky domény přímo hypervisorem.

K nejdůležitějším komunikacím rozdělených ovladačů mezi domU a dom0 patří komunikace se vstupně/výstupním zařízením. Z tohoto důvodu bude backend část ovladače konzole součástí prototypové implementace. Implementační detaily jsou rozebrány v části 6.6.1.

Další komunikace probíhá s databází XenStore, která slouží k uchování informací o dalších dostupných zařízeních. Vzhledem ke skutečnosti, že prototypová implementace nebude poskytovat žádná další zařízení, je implementace podpory databáze XenStore irelevantní.

5.5 Hypervolání

V předchozím textu byly shrnuty funkční bloky služeb, které budou součástí prototypové implementace. Tyto služby jsou realizovány pomocí konkrétních hypervolání:

1) Správa stránkovacích tabulek
   a) *HYPERVISOR_update_va_mapping*
   b) *HYPERVISOR_mmu_update*
      i) *MMU_NORMAL_PT_UPDATE*
      ii) *MMU_PT_UPDATE_PRESERVE_AD*
   c) *HYPERVISOR_mmuext_op*
2) Správa segmentů
   a) *HYPERVISOR_set_gdt*
   b) *HYPERVISOR_update_descriptor*
3) Obsluha přerušení
   a) *HYPERVISOR_set_trap_table*
   b) *HYPERVISOR_stack_switch*
   c) *HYPERVISOR_iret*
4) Kanály událostí
   a) *HYPERVISOR_set_callbacks*
   b) *HYPERVISOR_event_channel_op*
      i) *EVTCHNOP_bind_virq*
      ii) *EVTCHNOP_alloc_unbound*
      iii) *EVTCHNOP_send*
      iv) *EVTCHNOP_bind_ipi*
5) Grant tables
   a) *HYPERVISOR_grant_table_op*
      i) *GNTTABOP_setup_table*
6) Ladící konzole
   a) *HYPERVISOR_console_io*
      i) *CONSOLEIO_write*
7) Ostatní
   a) *HYPERVISOR_vcpu_op*
      i) *VCPUOP_is_up*

Ambicí prototypové implementace není úplná a stoprocentně zabezpečená varianta rozhraní hypervisoru Xen. Její ambicí je vytvořit takové řešení, které bude moci být dále rozvíjeno a zdokonalováno. V tomto duchu funguje i vypořádávání se s voláním nepodporovaného hypervolání (nebo nepodporované varianty hypervolání), které způsobí ukončení běhu hypervisoru. Díky tomu lze v budoucnu velmi snadno rozlišit chyby, které vznikly neimplementací hypervolání od chyb, které vznikly jeho vadnou implementací.

6 Implementace

Kapitola obsahuje podrobný popis změn, vedoucích k prototypové implementaci rozhraní hypervisoru Xen.

Hypervisorem je dále v textu míněno jádro upraveného operačního systému HelenOS.

6.1 Architektura ia32xenh

6.1.1 Zdrojové kódy architektury ia32xenh

Nový port jádra vychází z existujícího portu na architekturu IA-32. Tento port vznikl okopírováním architektonicky závislých zdrojových kódů portu IA-32 (*kernel/arch/ia32* na *kernel/arch/ia32xenh* a *boot/arch/ia32* na *boot/arch/ia32xenh*).

Obraz testovací domény a jejího modulu je uložen v adresáři *boot/arch/-ia32xenh*. Tyto soubory jsou použity jako moduly při překladu jádra pod architekturou *ia32xenh*.

6.1.2 Konfigurace a makefile

Přidání nové platformy implikuje nutnost změny v konfiguračním souboru *HelenOS.config*. Byla do něj přidána nová platforma *ia32xenh* a spolu s ní byla vytvořena standardní konfigurace pro platformu (nová konfigurace vychází[28] z konfigurace platformy *ia32*).

Nově byl vytvořen nový *Makefile* pro vytvoření parametru zavaděče diskového obrazu hypervisoru. *Makefile-ia32xenh.grub* vznikl okopírováním původního generického *Makefile.grub*. Další úpravy vyplynuly z potřeby předat testovací doménu a její modul jako modul hypervisoru. Žádné další moduly nejsou hypervisoru předávány.

---

[28] V principu šlo o nahrazení výskytu řetězce „PLATFORM=ia32“ za řetězec „PLATFORM=ia32|PLATFORM=ia32xenh“

Samotné vytvoření nového *Makefile* však nestačí. Dále bylo nutné nastavit hodnotu konstanty *BUILD* v souboru *boot/arch/ia32xenh/-Makefile.inc*[29] na *Makefile-ia32xenh.grub*.

6.2 Prerekvizity

6.2.1 Relokace jádra HelenOS

Za umístění objektů ve virtuálním adresovém prostoru je zodpovědný linker. Požadovaná změna v adresovém prostoru hypervisoru je realizována úpravou linker skriptu. Linker skript je rozdělen do několika sekcí (v případě architektury IA-32 na sekce „*maped*“ a „*unmapped*“). Každá sekce obsahuje informace o virtuálním i fyzickém umístění jednotlivých částí jádra.

Pro každou architekturu je vytvořen speciální linker skript, který se nachází v kořenovém adresáři konkrétní architektury (v případě architektury *ia32xenh* je umístěn v *kernel/arch/ia32xenh/_link.ld.in*). Pro změnu mapování byla změněna hodnota konstanty *BOOT_OFFSET*, která určuje adresu počátku mapování. Mapování vytvořené hypervisorem musí odpovídat mapování vytvořenému linkerem, a proto jsou všechna mapování tvořena pomocí stejného mechanismu, kterým je posun o pevně danou adresu. Posun adres je realizován pomocí maker *PA2KA* a *KA2PA*, jenž se nalézají v hlavičkovém souboru *kernel/arch/ia32xenh/mm/page.h*. Hodnota posunu byla upravena z původních 2 GB na novou hodnotu 256 MB.

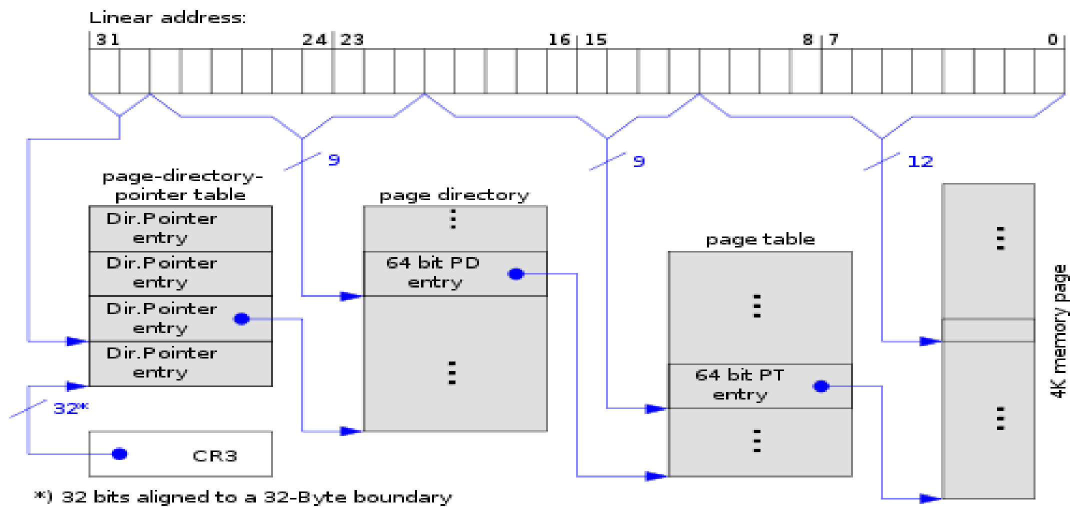
Úprava konstant *BOOT_OFFSET* a *PHYSMEM_LIMIT32*, změna posunu adres maker *PA2KA* a *KA2PA* a úprava linker skriptu jsou veškeré změny, které byly v souvislosti s relokací jádra provedeny.

6.2.2 Stránkování HelenOS – implementace metody PAE

Tato metoda umožňuje procesorům rodiny IA-32 adresovat více než 4 GB fyzické paměti. Poprvé byla podporována procesorem Intel Pentium Pro v roce 1995.

[29] Soubor Makefile.inc byl po provedení kroků v části 6.1 pouhým symbolickým odkazem na jiný soubor Makefile.inc. Z tohoto důvodu bylo nutné nejprve vytvořit vlastní soubor Makefile.inc a vložit do něj stejný obsah, jaký se nachází v odkazovaném souboru

Hlavní změnou, oproti stránkování bez *PAE*, je přidání další úrovně stránkovacích tabulek. Velikost jedné stránky je v závislosti na hodnotě bitu *PS* položky stránkovacího adresáře (z anglického „page directory“) buď 2 MB nebo 4 KB. Schéma stránkování s *PAE* s velikostí stránek 4 KB ilustruje Obrázek 8. Podrobný popis metody *PAE* je uveden v [10; svazek 3A, kapitola 4.4].



Obrázek 7: Stránkovací tabulky při zapnutém PAE (převzato z: http://en.wikipedia.org/wiki/-File:X86_Paging_PAE_4K.svg)

Prvním krokem bylo vytvoření iniciálního mapování hypervisoru a zapnutí metody *PAE*. Iniciální mapování mapuje horních a dolních 256 MB virtuálních adres na dolních 256 MB fyzických adres. Pro jednoduchost jsou použity 2 MB rámce, čímž je minimalizován počet vytvářených stránkovacích tabulek. Po jejich vytvoření jsou nastaveny související řídící registry *cr3*, *cr4* a *cr0*. Po nastavení řídících registrů je metoda *PAE* zapnuta. Inicializace stránkovacích tabulek a souvisejících řídících registrů je provedena ve funkci *map_kernel_pae*, která se nalézá v souboru */arch/src/boot/multiboot.S*.

Dalším krokem je implementace struktur stránkovacích tabulek a operací nad adresovým prostorem.

HelenOS obsahuje implementaci generických čtyřúrovňových stránkovacích tabulek a generické funkce pro práci s adresovým prostorem. Generické stránkovací tabulky lze snadno modifikovat pro metodu *PAE*, stačí změnit několik maker určující jednotlivé úrovně tabulek a práci s nimi a rozšířit definici zá-

znamu stránkovací tabulky (*pte_t*, obojí se nachází v */arch/include/mm/-page.h*).

Přímé použití generických funkcí pro práci s adresovým prostorem není možné. Překážkou je i explicitní průchod všech čtyř úrovní stránkovacích tabulek. Jednotlivé úrovně jsou procházeny pomocí maker (*KA2PA* a *PA2KA*) pro vzájemný převod virtuálních a fyzických adres. Pro úspěšný průchod tabulkami je nutné, aby odkaz na stránkovací tabulku nepoužité úrovně *n* byl stejný jako odkaz na stránkovací tabulku použité úrovně *n-1*. Kvůli nutnosti relokace jádra (viz 5.2.1) by splnění této podmínky vyžadovalo nestandardní[30] definici fyzické adresy nevyužité úrovně.

Konečně nemožnost použití generických funkcí vyplývá ze značného rozdílu ve struktuře záznamu stránkovacích tabulek mezi jednotlivými úrovněmi. Struktura první úrovně stránkovacích tabulek je odlišná od dalších úrovní. Funkce pro generické vložení převodu mezi virtuální a fyzickou adresou na všech úrovních nastavuje bity *U*, *W* a další, jejichž nastavení do první úrovně způsobuje výjimku typu výpadek stránky během překladu adres (strukturu stránkovacích tabulek se zapnutým *PAE* popisuje Obrázek 8).

	[63..36]	[35..32]	[31..21]	[20..12]	[11..09]	08	07	06	05	04	03	02	01	00
CR3	Not Applicable		Page Directory Base Pointer [31..05]							PCD	PWT	0		
Page Directory Pointer	**RSV**	Page Directory Address [35..12]			AVL	**RSV**	**RSV**	**RSV**	**RSV**	PCD	PWT	**RSV**	**RSV**	P
Page Directory Entry *2M*	**RSV**	Page Frame Address [35..21]		**RSV**	AVL	G	PS*	D	A	PCD	PWT	U	W	P
Page Directory Entry *4K*	**RSV**	Page Table Address [35..12]			AVL	0	PS†	0	A	PCD	PWT	U	W	P
Page Table Entry *4K*	**RSV**	Page Frame Address [35..12]			AVL	G	**RSV**	D	A	PCD	PWT	U	W	P

Obrázek 8: Stránkovací struktury pro PAE (zdroj: http://www.rcollins.org/ddj/Jul96/)

---

[30] Nejjednodušším způsob, jak splnit tuto podmínku, je definovat posun mezi adresami přesně jako polovinu dostupného virtuálního adresového prostoru (pro IA-32 posun o 2 GB, respektive o 0x80000000).

Z obou těchto důvodů byly implementovány (jako součást */genarch/src/-mm/page_pt.c*) generické funkce pro operace nad adresovým prostorem se zapnutou metodou *PAE*. Tyto funkce prochází tři úrovně stránkovacích tabulek (*L0*, *L1* a *L3*) a reflektují[31] rozdíly mezi úrovní nula a dalšími úrovněmi. Nově implementované funkce jsou sdruženy do struktury operací nad adresovým prostorem *page_mapping_operations_t*, čímž mohou být v budoucnu použity i v portu IA-32 bez hypervisoru.

6.2.3 Inicializace virtuálního adresového prostoru jádra

Vytvoření mapování adresového prostoru jádra je součástí funkce *page_arch_init*. Dostupné fyzické rámce jsou mapovány na virtuální adresy získané z makra *PA2KA*. Příznaky mapovaných stránek vychází z jejich umístění:

- VA v oblasti jádra jsou mapovány s příznakem *G*
- VA v oblasti sdílené tabulky *MA2PA* jsou mapovány s příznakem *R*
- VA v oblasti mimo sdílené tabulky *MA2PA* jsou mapovány s příznakem *W*

Dále je vytvořeno mapování oblasti privátní tabulky *MA2PA* na stejnou oblast fyzické paměti, na kterou byla mapována sdílená tabulka *MA2PA*. Příznaky pro tuto oblast jsou nastaveny na *PAGE_CACHEABLE* a *PAGE_WRITE*. Dvojité mapování na stejnou oblast fyzické paměti jde mimo logiku souvislého mapování, ale je nutné pro zajištění ochrany sdílené tabulky *MA2PA* před přepsáním doménou.

6.3 Načtení zaváděné domény

Doména a její volitelný modul jsou hypervisoru předány ve formě modulů. Způsob načítání jednotlivých modulů v operačním systému HelenOS se ukázal jako nevhodný. Pro účely načtení zaváděné domény byla vytvořena funkce *hinit*,

[31] Ve funkci ***pt_mapping_insert_pae*** je záznamu úrovně nula nastaven pouze bit P.

která je volána namísto generické funkce *kinit*, pokud je systém HelenOS přeložen pod architekturou *ia32xenh*.

6.3.1 Funkce hinit

Funkce *hinit* vychází z generické funkce *kinit*. Vychází z předpokladu, že první předaný modul je obraz zaváděné domény a druhý předávaný modul je jejím modulem. Více modulů není podporováno. Samotná doména je zavedena funkcí *program_create_from_image* v několika krocích:

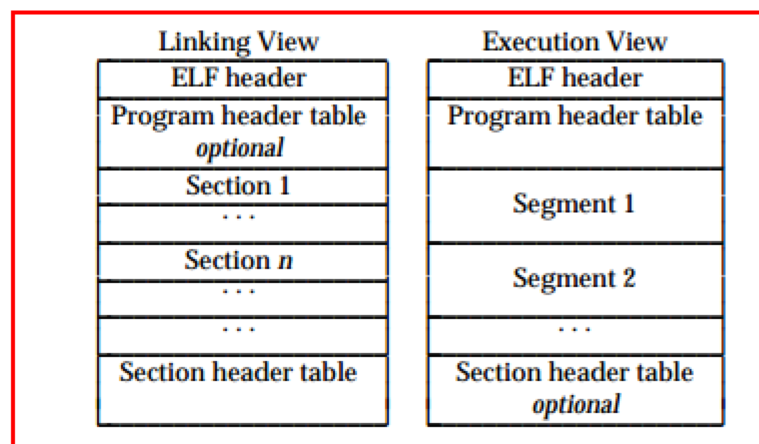
- Získání informací o doméně z oblasti „*_NOTE*“ obrazu domény (viz 6.3.2)
- Vypočtení rozložení virtuálního adresového prostoru domény (viz 6.3.3)
- Vytvoření iniciálního mapování (kterému se věnuje část 6.3.4)
- Inicializace struktur *start_info_t* a *shared_info_t* (viz 6.3.5)

6.3.2 Načtení parametrů zaváděné domény

Parametry zaváděné domény jsou uloženy v oblasti „*_NOTE*“, která je jednou z oblastí tvořící obraz domény. Předtím, než bude probrán způsob načítání parametrů zaváděné domény, je nutné uvést několik poznámek ke struktuře souboru ve formátu *ELF*.

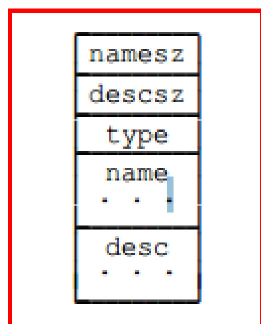
Struktura souboru ELF

Soubor formátu *ELF* je rozdělen do několika sekcí (viz Obrázek 9). Detailní popis jednotlivých částí struktury *ELF* je obsažen v dokumentaci [11]. Pro účely této práce budou popsány pouze ty bloky, které přímo souvisí s načítáním parametrů domény.



Obrázek 9: Struktura souboru ELF (převzato z [11, Figure 1-1])

Na počátku binárního souboru ve formátu *ELF* se nachází hlavička (z anglického „ELF header"). Hlavička obsahuje informace o struktuře celého zbytku souboru (například informace o počtu segmentů, vstupním bodu programu a další). Na základě informací obsažených v hlavičce lze nalézt požadovanou oblast „*_NOTE*".



Obrázek 10: Struktura sekce "_NOTE" (převzato z [7; Figure 2-3])

Struktura oblasti „*_NOTE*" je rozdělena na bloky obsahující informace typu *klíč=hodnota*. Velikost každého bloku (poznámky) závisí na velikosti informací uložených uvnitř bloku (velikosti klíče a hodnoty). Schéma struktury popisuje **Chyba! Nenalezen zdroj odkazů.**. Velikost každé položky bloku je zarovnána na 4 B. Položka *namesz* obsahuje nezarovnanou velikost položky *name* (klíč), položka *descsz* obsahuje nezarovnanou velikost položky *desc* (hodnota). Do architektonicky nezávislé části operačního systému HelenOS byla přidána struktura *elf_note_t*, která pracuje s jednotlivými bloky oblasti „*_NOTE*".

Načtení parametrů z ELFu

Pro načtení a uchování parametrů uložených v oblasti „*_NOTE*" byla vytvořena struktura *domain_parms* (*ia32xenh/include/elf.h*). Položky této struktury lze rozdělit do několika skupin:

1. Položky související se strukturou *ELF*u domény
2. Položky obsahující načtené parametry zaváděné domény
3. Položky obsahující vypočtené adresy jednotlivých částí iniciálního VAP domény
4. Položky obsahující informace o modulu domény
5. Položky obsahující informace o iniciálních stránkovacích tabulkách domény
6. Položka obsahující fyzickou adresu prvního rámce fyzického bloku paměti domény

7. Položky obsahující nejvyšší a nejnižší virtuální adresu segmentů k mapování (tzv. „loadable“).

Inicializace první, druhé a sedmé skupiny položek struktury *domain_parms* je součástí funkce *elf_parse_xen_notes*, jejímiž parametry jsou ukazatele na hlavičku *ELF* obrazu domény a na strukturu *domain_parms*. V této funkci je nejprve provedena kontrola kompatibility parametrů hlavičky. V případě problémů je inicializace domény ukončena a doména není vytvořena a spuštěna. Po kontrole parametrů hlavičky je přikročeno k vlastní inicializaci. Nejprve jsou procházeny všechny hlavičky segmentů. Segment, jehož typ je *PT_NOTE*, je předán k dalšímu zpracování funkcí *elf_xen_parse_notes*. Pokud žádný takový segment neexistuje, jsou procházeny i všechny hlavičky sekcí. Sekce, jejíž typ je *SHT_NOTE*, je předána ke zpracování funkci *elf_xen_parse_notes*. Pokud ani taková sekce neexistuje, je inicializace ukončena s chybovým kódem. V opačném případě inicializace pokračuje vypočtením rozložení virtuálního adresového prostoru domény (viz 6.3.3).

Inicializace parametrů zaváděné domény je součástí funkce *elf_xen_parse_notes*, respektive *elf_xen_parse_note* pro jednotlivé parametry. Typ obsahu jednotlivých poznámek je determinován hodnotou atributu *type* poznámky. Všechny podporované hodnoty atributu *type* jsou uloženy v hlavičkovém souboru *xen-public/elfnote.h*, který je součástí veřejného rozhraní hypervisoru Xen. Typ obsahu (položka *desc*) poznámky může být číslo nebo řetězec.

6.3.3 Vypočtení rozložení VAP domény

Testovací doméně je přidělen jeden souvislý blok 64 MB fyzické paměti, což usnadňuje převody mezi fyzickou adresou domény, virtuální adresou domény a virtuální adresou hypervisoru. Převodní vztah využívá skutečnosti, že počáteční virtuální adresa domény je doméně mapována na začátek alokované fyzické paměti. Pro převod slouží makra *DA2KA*, *DA2PA* a *PA2DA*.

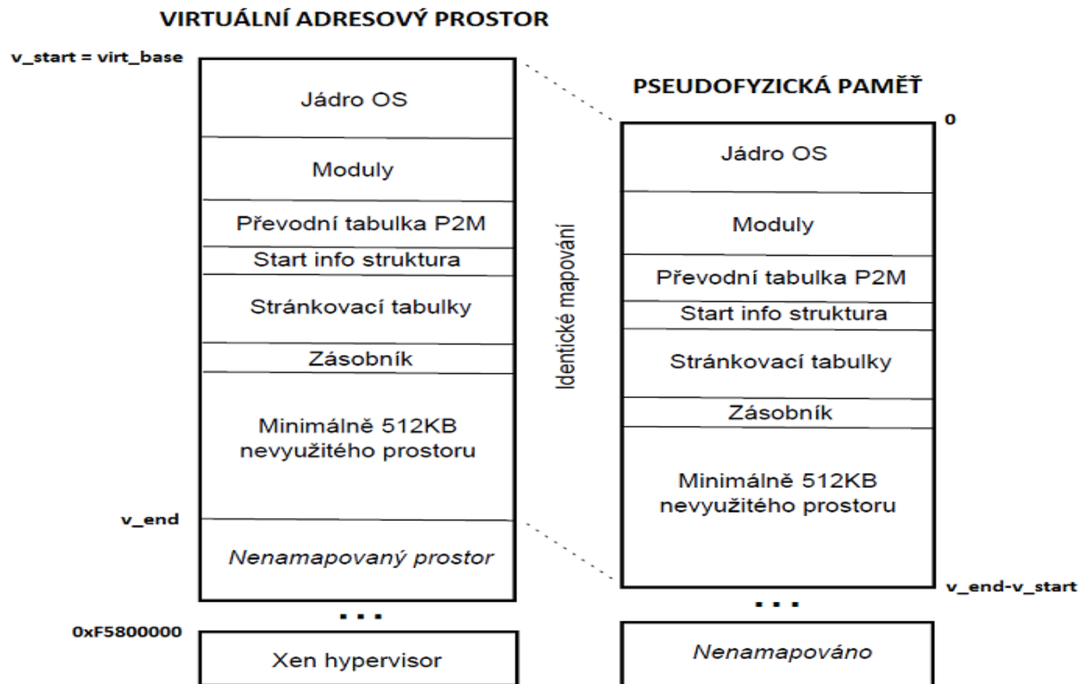
Iniciální virtuální adresový prostor domény je vytvořen ve dvou krocích. Nejprve jsou zjištěny (respektive vypočteny) počáteční a koncové adresy bloků pamě-

ti, které jsou uloženy do položek struktury `domain_parms`. Následně jsou tyto bloky inicializovány požadovaným obsahem (viz 5.2.4).

Prvním krokem je zjištění mezí bloků iniciálního mapování, které vychází z informací o doméně. Všechny bloky jsou zarovnány směrem nahoru na velikost stránky (velikost stránky je nastavena na 4 KB); počáteční a koncový blok je zarovnán na 4 MB. Z těchto informací je zjištěna počáteční virtuální adresa domény. Z `ELF` obrazu domény je vypočtena počáteční i konečná virtuální adresa jádra domény. Následuje blok s volitelným modulem domény. Dalším blokem je tabulka `P2M`. Velikost tohoto bloku je dána skutečně alokovanou fyzickou pamětí. Velikost bloku obsahující strukturu `start_info_t` je jedna stránka. Jediný blok, jehož velikost nelze určit prostým sečtením dvou čísel, je blok s iniciálním mapováním domény. Důvodem je skutečnost, že přidání další stránkovací tabulky domény může vyvolat potřebu další stránkovací tabulky pro pokrytí celého virtuálního adresového prostoru domény. Z tohoto důvodu je koncová adresa iniciálních stránkovacích tabulek domény počítána následujícím algoritmem:

1. Inicializace: Nastavení počtu stránkovacích tabulek na tři.
2. Přepočítání koncové adresy virtuálního adresového prostoru domény (`v_end`).
3. Zjištění požadovaného množství stránkovacích tabulek pro pokrytí celého VAP domény <`v_start=virt_base`; `v_end`>.
4. Porovnání požadovaného a aktuálního množství stránkovacích tabulek. V případě, že jsou obě čísla stejná, ukončení cyklu. V opačném případě zvětšení počtu aktuálních stránkovacích tabulek a návrat do bodu dva.

Posledním blokem je jedna stránka obsahující paměť pro iniciální zásobník domény. Koncová adresa prostoru domény je rovna koncové adrese zásobníku zarovnané na 4 MB směrem nahoru. Pokud je prostor mezi těmito adresami menší než 512 KB, je koncová adresa domény posunuta o další 4 MB směrem nahoru.



Obrázek 11: Start of a day memory layout (VAP, PFP)

Inicializace obsahu bloku kernelu je provedeno okopírováním `ELF` segmentů na adresu určenou parametrem `p_paddr` hlavičky segmentu. Jsou kopírovány pouze ty segmenty, jejichž typ je roven konstantě `PT_LOAD`. Po okopírování všech segmentů je okopírován celý modul. Tabulka `P2M` je reprezentována polem čísel fyzických rámců, které je indexováno čísly pseudo-fyzických rámců domény. Inicializace tabulky `P2M` probíhá současně s inicializací sdílené tabulky `M2P`. Díky skutečnosti, že přidělený blok fyzické paměti je souvislý, lze inicializaci obou tabulek provést jednoduchým cyklem. Pseudo-fyzické rámce jsou číslovány od nuly a nultý pseudo-fyzický rámec odpovídá prvnímu přidělenému fyzickému rámci (jak naznačuje Obrázek 11). Inicializace struktury `start_info_t` a iniciálních stránkovacích tabulek je rozebrána dále v textu. Blok obsahující iniciální zásobník není nijak inicializován.

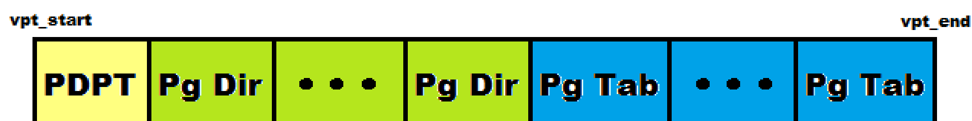
6.3.4 Vytvoření iniciálního mapování

Po získání informací o rozložení jednotlivých bloků ve VAP domény lze toto rozložení vytvořit pomocí iniciálních stránkovacích tabulek domény. Jejich umístění je v rámci iniciálního mapování pevně stanoveno. Pomocí převodních maker *DA2KA*, *PA2KA* a *KA2PA* lze vytvořit stránkovací tabulky s identickým mapováním přímým zápisem do adresového prostoru hypervisoru.

Stránkovací tabulky domény popisují dva souvislé bloky VAP – iniciální mapování domény a mapování horních 168 MB VAP hypervisoru.

Díky souvislosti virtuálního adresového prostoru domény lze stránkovací tabulky rozdělit tak, aby každé dvě po sobě navazující stránkovací tabulky jedné úrovně adresovaly souvislý blok virtuální paměti. Rozmístění jednotlivých stránkovacích tabulek popisuje následující algoritmus:

1. Oblast mezi *vpt_start* a *vpt_end* obsahuje postupně *PTL0*, *PTL1*s, *PTL3*s (viz Obrázek 12)
2. Každá úroveň stránkovacích tabulek popisuje převody od virtuální adresy *v_start* do *v_end* a to postupně od *v_start*. Například každá stránkovací tabulka třetí úrovně umožňuje adresovat celkem 2 MB paměti. První stránka třetí úrovně adresuje oblast <*v_start*; *v_start*+2 MB), další stránka <*v_start*+2 MB; *v_start*+4 MB) a tak dále. Poslední stránkou je adresována oblast <*v_end*-2 MB; *v_end*).



Obrázek 12: Schematické rozdělení iniciálních stránkovacích tabulek domény

Při vytváření mapování oblasti hypervisoru lze vycházet z předpokladu, že doména nebude měnit ani procházet tuto část stránkovacích tabulek. Díky tomu lze toto namapování vytvořit pouhým překopírováním a napojením té části stránky adresáře stránek (ze stránkovacích tabulek hypervisoru), která pokrývá mapování horních 168 MB. Související stránkovací tabulky není nutné kopírovat díky předpokladu nezasahování domény do této části stránkovacích tabulek.

Pro vytvoření bloku s iniciálním mapováním domény lze využít zvoleného rozdělení stránkovacích tabulek. Převodní tabulky (i index v rámci tabulky) každé stránky VAP domény lze totiž určit jednoznačně. Schematický algoritmus pro vytvoření iniciálního mapování domény je popsán následovně:

```
foreach (va = stránka ∈ iniciální VAP domény) {
  foreach (PTLx = použitá úroveň stránkovacích tabulek) {
    if(! PTLx[index dle va].PRESENT) {
/*získej virtuální adresu stránkovacích tabulek x+1 úrovně pou-
žité pro překlad va a vyplň ji nulami. */
/*nastav jejich fyzickou adresu do stránkovacích tabulek x-té
úrovně a nastav příznaky (P, …). */
}}}
```

6.3.5 Inicializace struktur start_info_t a shared_info_t

Inicializace struktury start_info_t

Inicializace položek struktury `start_info_t` vychází ze zdrojových kódů inicializace dom0 hypervisorem Xen. Položky struktury jsou z větší části inicializovány obsahem příslušných položek struktury `domain_parms`.

Údaje pro frontendovou část ovladače konzole jsou nastaveny v rámci inicializace struktury `shared_info_t`.

Inicializace struktury shared_info_t

Struktura `shared_info_t` poskytuje dynamicky měněné informace o systému. Struktura `start_info_t` obsahuje položku `shared_info`, ve které je uloženo číslo fyzického rámce, na kterém je alokována skutečná struktura `shared_info_t`. Prvním krokem při inicializaci je proto alokace jednoho fyzického rámce a vynulování jeho obsahu. Ukazatel na tento rámec je vložen do architektonicky závislé části definice úlohy.

Další akcí je zakázání doručování událostí pro všechny virtuální procesory (v našem případě pro první virtuální procesor, protože je podporován pouze jeden) a nastavení čísla fyzického rámce umístění této struktury ve struktuře `start_info_t`. Ostatní položky této struktury nejsou inicializovány.

Inicializace hypercall page

Jedním z povinných parametrů zaváděné domény je virtuální adresa na začátek stránky, na kterou mají být uloženy funkce pro vyvolání hypervolání. Stránka je v cyklu po blocích velikosti 32 B inicializována přímo hodnotami příkazů přeložených do assembleru. Jedinou výjimku tvoří oblast určenou hypervolání *iret*. Toto hypervolání na zásobníku očekává původní[32] hodnotu registru *EAX*.

6.4 Zavedení inicializované domény

Zavedení domény vychází ze zavedení modulu na architektuře *ia32*. Jediným rozdílem je nutnost předání virtuální adresy, mapované na rámec obsahující strukturu *start_info_t*, do registru *si*. Obsah registru *si* je nastaven v modifikované funkci *userspace* těsně před přechodem na nižší stupeň oprávnění. Číslo fyzického rámce bylo přidáno jako podmíněně překládaný parametr architektonicky nezávislé struktury *uspace_arg_t*. Toto řešení bylo upřednostněno před vytvořením architektonicky závislých částí této struktury podobně, jako ve strukturách vlákna a úlohy. To by znamenalo vytvoření architektonicky závislých částí pro každou doposud podporovanou architekturu. Vzhledem k faktu, že žádná další architektura nepotřebuje architektonicky závislé parametry, znamenalo by to vytvoření několika prázdných struktur a vytvoření struktury s jedním parametrem pro architekturu *ia32xenh*.

6.5 Implementované rozhraní

Kroky realizované v předchozí části inicializují VAP domény a předají řízení. Pro samotný běh domény je třeba poskytovat určité balíky služeb prostřednictvím hypervolání. Tato část popisuje implementaci jednotlivých skupin hypervolání. V popisu každé skupiny služeb jsou popsány změny v architektuře oproti výchozímu stavu (*ia32*) a následně změny, které přímo souvisí s implementací jednotlivých hypervolání.

---

[32] Do registru EAX je posléze vložena hodnota čísla hypervolání, které je doménou voláno

6.5.1 Správa stránkovacích tabulek

Změny v mechanismu stránkování vycházejí z implementace podpory metody *PAE* (viz 6.2.2).

Hypervolání

Implementace souvisejících hypervolání nezasahuje do struktury architektury, ani nepřidává žádné nové proměnné, ale přímočaře aplikuje požadované funkce. Pro její pochopení postačují komentáře u příslušných hypervolání.

6.5.2 Správa segmentů

Změny v *GDT* jsou vyvolány požadavkem na existenci standardních globálních deskriptorů. Jejich parametry vychází z implementace těchto deskriptorů v hypervisoru Xen.

Před naplánováním domény (ve funkci *arch_before_thread_runs*) je okopírován obsah doménově závislé části *GDT* do globální *GDT*.

Hypervolání

Seznam fyzických rámců obsahující tu část *GDT*, která náleží doméně, je reprezentován polem *gdt_frames*. Toto pole je jednou z položek struktury *vcpu_guest_context_t*, která je součástí veřejného rozhraní hypervisoru Xen. Ukazatel na tuto strukturu se stal součástí architektonicky závislé implementace vlákna.

Hypervoláním *HYPERVISOR_set_gdt* jsou nastavena čísla všech fyzických rámců *GDT* náležících doméně. Naproti tomu hypervoláním *HYPERVISOR_update_descriptor* je upraven obsah jednoho konkrétního deskriptoru, jehož fyzická adresa je součástí parametrů tohoto hypervolání. Součástí obou hypervolání je, stejně jako před každým naplánováním domény, okamžité promítnutí požadovaných změn i do globální *GDT*.

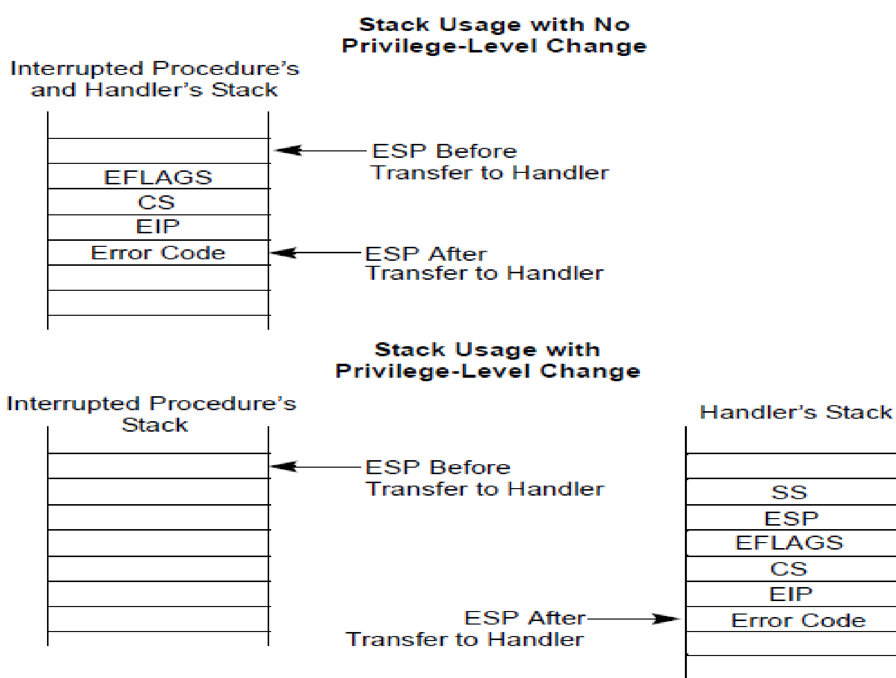
6.5.3 Výjimky

Před vlastním popisem úprav je vhodné uvést mechanismus výjimek architektury IA-32. Následující stručný popis čerpá z [10; svazek 3A; kapitola 6].

Výjimky na IA-32

Vyvoláním výjimky jsou na zásobník[33] uloženy hodnoty registrů `CS`, `EIP` a `EFLAGS`. Pro některé výjimky jsou na zásobník uloženy doplňující informace souhrnně označované jako `Error Code` (velikosti 4 B). Pokud je hodnota oprávnění v okamžiku vyvolání výjimky vyšší (oprávnění je nižší), než hodnota oprávnění obsluhy přerušení, jsou na zásobník uloženy i aktuální hodnoty registrů `SS` a `ESP`.

Obsah uložený na vrcholu zásobníku před a po vyvolání obsluhy výjimky ilustruje Obrázek 13.



Obrázek 13: Stav zásobníku před obsluhou přerušení (zdroj: [10; 3A; Figure 6-4])

---

[33] V případě, že obsluha výjimky běží na číselně nižším stupni oprávnění, než přerušený kód, je použit zásobník, jehož parametry jsou uloženy v TSS registru. V opačném případě je použit aktuální zásobník.

Po uložení hodnot registrů na zásobník je procesorem z *IDT* získána adresa obsluhy nastalé výjimky. Získaná adresa je vložena do registrů *CS* a *EIP* a podle typu přerušení je dále upraven registr *EFLAGS*. Poté je obnoveno vykonávání instrukcí (začátek obsluhy).

Každá obsluha přerušení je ukončena instrukcí *iret*, při které jsou obnoveny hodnoty registrů *CS*, *EIP*, *EFLAGS* a v případě potřeby i *SS* a *ESP*.

Změny v mechanismu obsluhy výjimek

Průběh obsluhy výjimek operačního systému HelenOS popisuje část 4.3. Důležitým faktem je uložení obsahu registrů na zásobník, kde jsou spolu s hodnotami registrů uložených procesorem dostupné skrze strukturu *istate_t* pro další části obsluhy výjimek.

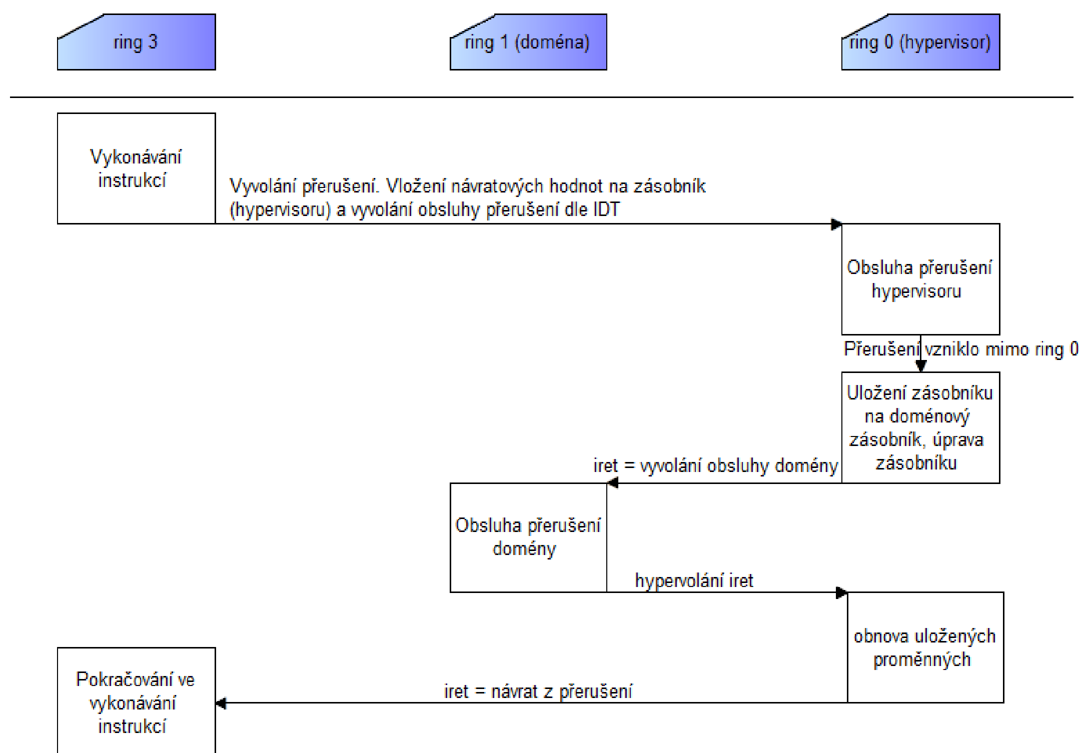
Třetí (architektonická) úroveň obsluhy výjimek vychází z obsluh pro architekturu IA-32. Rozdíl je v implementaci prvních 32 výjimek a výjimky pro systémové volání. Při zpracování těchto výjimek je kontrolována úroveň oprávnění instrukce, která danou výjimku způsobila. V případě, že se tato instrukce nalézá mimo hypervisor (úroveň oprávnění je vyšší než nula), je zpracování výjimky postoupeno aktuálně naplánované doméně.

Postoupení výjimky znamená přepsání hodnot registrů uložených na zásobníku a reprezentovaných strukturou *istate_t*. Přepsání je realizováno funkcí *call_domain_exception_handler* (*kernel/arch/ia32xenh/src/-hypervisor/callback.c*), ve které je realizována změna obsahu zásobníku hypervisoru následujícím způsobem:

1. Nejprve je získána adresa zásobníku domény. Na získaný zásobník budou ve čtvrtém kroku okopírovány vybrané hodnoty aktuálního zásobníku (reprezentované strukturou *istate_t*).
2. Dále je získána adresa do zásobníku hypervisoru, která je nastavena na adresu prvního prvku, jenž bude kopírován (*cs*, nebo *error_word*). Současně je získán počet bytů souvislého bloku zásobníku, jenž bude okopírován.

3. Do zásobníku domény jsou okopírována data získaná ze zásobníku hypervisoru dle druhého kroku.
4. Hodnoty registrů *CS* a *EIP*, které jsou uloženy na zásobníku hypervisoru jsou přepsány na hodnoty odkazující na obsluhu přerušení hypervisoru.
5. V případě, že výjimka nastala v uživatelském prostoru domény, jsou dále přepsány hodnoty registrů *SS* a *ESP*.
6. Povolení/zakázání doručování událostí doméně je nastaveno dle požadavků domény.

Po přepsání hodnot na zásobníku hypervisoru je instrukcí *iret* realizován skok přímo na požadovanou obsluhu domény. Posledním krokem obsluhy domény je hypervolání *iret*. Po přepsání hodnot na zásobníku hypervisoru je instrukcí *iret* realizován skok přímo na původní návratovou adresu výjimky. Celý proces obsluhy výjimky popisuje Obrázek 14.



Obrázek 14: Schéma obsluhy výjimky

Povolené výjimky

Každá položka *IDT* obsahuje dva bity sloužící pro určení minimální přípustné úrovně oprávnění k vyvolání výjimky. Vyvolání výjimky z nižší (číselně vyšší) úrovně oprávnění okamžitě způsobí tzv. „general protection fault" (dále jen #GPF) výjimku.

IDT je inicializována ve funkci *idt_init*. Prvním 32[34] výjimkám a výjimce reprezentující systémové volání (128) je nastavena úroveň oprávnění tři. To znamená, že mohou být vyvolány z libovolné úrovně oprávnění. Výjimce reprezentující hypervolání (130) je nastavena úroveň oprávnění jedna. Hypervolání tedy mohou být vyvolána pouze z hypervisoru, nebo domény.

Hypervolání

Seznam výjimek zpracovávaných doménou je hypervisoru předán skrze hypervolání *HYPERVISOR_set_trap_table*. Každá položka předaného pole obsahuje parametry odpovídající položkám struktury *trap_info*. Pole uvedených struktur je jednou z položek struktury *vcpu_guest_context_t*, která je součástí veřejného rozhraní hypervisoru Xen.

Během zpracování *HYPERVISOR_set_trap_table* jsou do pole *trap_ctxt* okopírovány hodnoty platné pro prvních 32 výjimek a pro výjimku reprezentující systémové volání. Parametry ostatních výjimek jsou ignorovány. Mechanismus předání výjimky ke zpracování doméně byl popsán výše v textu.

Pro úspěšné předání výjimky doméně je nutné získat adresu jejího zásobníku. Tato adresa je hypervisoru předána skrze hypervolání *HYPERVISOR_stack_switch*. Adresa doménového zásobníku je uložena do položek *kernel_ss* a *kernel_esp*. Obě položky jsou součástí struktury *vcpu_guest_context_t*.

---

[34] Tyto výjimky mohou být předány ke zpracování testovací doméně. V principu by všechny výjimky kromě výjimky reprezentující hypervolání, mohly být předávány doméně ke zpracování.

Implementace hypervolání *HYPERVISOR_iret* (funkce *do_iret*) se nachází v souboru *kernel/arch/ia32xenh/src/interrupt.c*. Hypervolání simuluje provedení instrukce *iret* – obnovuje hodnoty registrů na hodnoty uložené na zásobníku domény. Díky faktu, že hypervolání *iret* vyvolá výjimku lze přepsání registrů provést stejným způsobem jako u funkce *call_domain_exception_handler*.

6.5.4 Kanály událostí

Implementace reprezentace koncových bodů vychází z implementace hypervisoru Xen (viz 3.3.2). Do architektonicky závislé struktury reprezentující úlohu bylo přidáno pole stránek obsahující struktury *evtchn*. Princip jejich alokace a vzájemného převodu mezi koncovými body a porty vychází z implementace hypervisoru Xen (také 3.3.2).

Způsob předávání událostí doméně vychází z architektury předávání zpracování výjimek. To je možné díky tomu, že všechny podporované události nastávají v okamžiku vyvolání výjimky. Události jsou doručovány prostřednictvím tzv. „callback" funkce, jejíž adresa je uložena do položek *event_callback_cs* a *event_callback_eip*. Obě položky jsou součástí struktury *vcpu_guest_context_t*.

Samotná událost může být doručena pouze za předpokladu, že je splněno několik podmínek. Tyto podmínky vychází z implementace použité Xenem a popsané v knize [4]. Pouze událost virtuálního časovače je doméně doručována ihned a to pouze za předpokladu, že jsou všechny ostatní podmínky naplněny.

Hypervolání

Před doručením události doméně je třeba znát adresu callback funkce. Tato adresa je nastavena v hypervolání *HYPERVISOR_set_callback*.

Kanály událostí (respektive koncové body kanálů událostí) jsou doménou spravovány prostřednictvím hypervolání *HYPERVISOR_event_channel_op*. Prototypová implementace umožňuje vytvořit nový koncový bod, propojit volný

koncový bod s virtuálním hardwarem (časovač, klávesnice), či jiným koncovým bodem téže domény a posílat zprávy z jednoho koncového bodu na druhý.

Vytvoření a propojení koncových bodů znamená nastavení souvisejících položek struktury *evtchn*. Z implementačního hlediska se vždy jedná o nastavení souvisejících položek struktury *evtchn*.

Poslání zprávy od jednoho koncového bodu vyvolá nastavení příslušných položek struktury *shared_info_t* doméně vlastnící druhý koncový bod.

6.5.5 Grant tabulky

Inicializace grant tabulek předpokládá jejich existenci. Jejich definice byla převzata z implementace hypervisoru Xen a nalézá se v */hypervisor/grant_table.h*.

Hypervolání

Jedinou implementovanou částí hypervolání *HYPERVISOR_grant_table* je inicializace struktury grant tabulek. Inicializace grant tabulek znamená inicializaci struktury *grant_table_t*. Pro pochopení postačují komentáře, které se nacházejí v inicializační funkci *gnttab_setup_table*.

6.5.6 Ladicí konzole

Implementace ladicí konzole podporuje pouze zápis *ASCII* znaků na výstupní zařízení. Díky skutečnosti, že zpracování každého hypervolání probíhá ve VAP domény, která jej vyvolala, je zápis na výstupní konzoli implementován jednoduchým cyklem.

6.5.7 Ostatní

Každý virtuální procesor domény obsahuje informace o tom, z jakého důvodu je na něm přerušeno vykonávání instrukcí. Tyto důvody jsou reprezentovány bitovými konstantami s prefixem *VPF_* a jsou součástí hlavičkového souboru architektonicky závislé implementace vlákna (*ia32xenh/include/proc/thread.h*). Při pozastavení vykonávání instrukcí na virtuálním procesoru domény je proměnné *pause_flag* (která byla přidána jako položka architekto-

nicky závislé části jádra) tohoto procesoru nastaven bit odpovídající důvodu pozastavení. Prototypovou implementací je prozatím používán bit *VPF_DOWN*, kterým je indikováno vypnutí virtuálního procesoru.

Hypervolání

Hypervoláním *HYPERVISOR_vcu_op* je doména informována o stavu konkrétního virtuálního procesoru. Testovací doména vyžaduje pouze informaci o skutečnosti, zdali je virtuální procesor aktivní, či nikoliv. Tuto skutečnost lze ověřit získáním hodnoty bitu proměnné *pause_flag* na pozici *VPF_DOWN*.

6.6 Změny suplující chybějící dom0

6.6.1 Backend část ovladače konzole

Implementace backend části ovladače konzole je rozdělená na dvě samostatné části[35] - zpracování požadavků na obrazovku a zpracování stisknutých kláves. Obě tyto části pracují s globální proměnnou *domainU_console* struktury *xencons_interface*, která je inicializována v rámci inicializace domény. Tato struktura je součástí veřejného rozhraní Xenu a je definována následovně:

```
 struct xencons_interface {
    char in[1224];
    char out[5048];
    XENCONS_RING_IDX in_cons, in_prod;
    XENCONS_RING_IDX out_cons, out_prod;
};
```

Další důležitou globální proměnnou je proměnná *hypervisor_domainu_console_port*, která obsahuje číslo portu ovladače konzole. Toto číslo je doméně předáno jako proměnná *evtchn* struktury *start_info_t*. Obě globální proměnné jsou definovány v hlavičkovém souboru *ia32xenh/src/proc/domain.h*.

Inicializace frontendové části ovladače konzole

Při inicializaci struktury *start_info_t* je doméně vyplněna část určená pro domU. V této části jsou definovány proměnné *mfn* a *evtchn*.

[35] Obě části ovladače se nacházejí ve složce ia32xenh/dom0s

V proměnné *mfn* je uloženo číslo fyzického rámce paměti, jehož obsahem je struktura *xencons_interface*. Vzhledem k tomu, že s tímto rámcem pracuje i frontendová část ovladače fungující na úrovni domény, je nezbytně nutné, aby rámec samotný byl ve vlastnictví domény. Pro tyto účely byl vybrán rámec nacházející se před koncem iniciálního VAP domény. Virtuální adresa tohoto rámce (z pohledu hypervisoru) je uložena do globální proměnné *domainU_console*, ke které je přistupováno z backendové části ovladače konzole.

Proměnná *evtchn* obsahuje číslo kanálu, na který mají být posílány události pro zobrazení informací na výstupní zařízení a pro upozornění na stisknuté klávesy. Tento kanál je vytvořen funkcí pro vytvoření nového kanálu *evtchn_alloc_unbound*. Po vytvoření kanálu je nastaven jeho příznak *consumer_is_xen*. Číslo kanálu je uloženo do proměnné *evtchn*. Vytvořený kanál je použit jako jeden konec mezidoménového propojení mezi doménou a hypervisorem. Druhý konec tohoto propojení je uložen do globální proměnné *hypervisor_domainu_console_port*. Ta je dále používána pro detekci požadavku výpisu na výstupní zařízení.

Zpracování požadavků pro výpis na obrazovku

Výstupním zařízením domény je terminál *VT100*, který s ovladačem komunikuje pomocí znaků *ASCII* abecedy. Některé znaky tvoří tzv. řídící sekvence, jež mají speciální význam. Znaky (nebo řídící sekvence), které jsou určeny pro výstupní zařízení, jsou frontendovou částí ovladače domény uloženy do položky *out* struktury *xencons_interface*. Za každý takto zapsaný znak je zvýšena hodnota čítače *out_prod*. Žádost o vypsání zatím nezpracované části pole *out* je realizována hypervoláním *HYPERVISOR_event_chanel_op*. Parametry hypervolání jsou konstanta *EVTCHNOP_send* a odkaz na strukturu *evtchn_send_t*, která obsahuje číslo portu ovladače konzole.

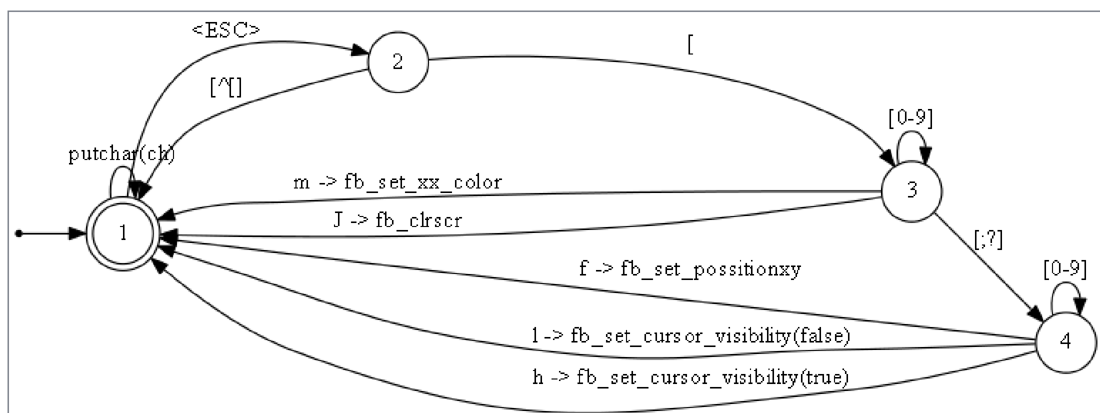
Zpracování hypervolání *HYPERVISOR_event_chanel_op* s parametrem *EVTCHNOP_send* je definováno ve funkci *evtchn_send*. V případě, že předané číslo portu odpovídá číslu portu uloženému v proměnné *hypervisor_domainu_console_port* a kanál odpovídající předanému portu je me-

zidoménový a má nastaven příznak consumer_is_xen, je vyvolána funkce write_to_real_console. Funkce reprezentuje backend část ovladače pro zpracování požadavků na obrazovku.

Funkce write_to_real_console zpracovává všechny dosud nezpracované znaky položky out. Protože pole out neobsahuje pouze ASCII znaky, ale i řídící sekvence terminálu VT100, bylo nutné nejprve určit, které řídící sekvence budou podporovány. Při definici sekvencí autor vycházel z webové dokumentace [12]. Konečný seznam všech sekvencí podporovaných prototypovou implementací je následující:

- *<ESC>[{attr}m* – Nastavení vlastností displeje (barva textu a pozadí)
- *<ESC>[5J* – Vymazání obrazovky
- *<ESC>[?l* – Schování kurzoru
- *<ESC>[?h* – Zobrazení kurzoru na aktuální pozici
- *<ESC>[{ROW};{COLUMN}f* – Nastavení pozice kurzoru

Pro zpracování jednotlivých znaků položky out je definován konečný automat se čtyřmi stavy, který popisuje Obrázek 15. Aktuální stav tohoto automatu je uložen v architektonicky závislé části proměnné THREAD jako položka console_state. Výchozí hodnota stavu je 1. Při každém volání funkce write_to_real_console jsou automatem zpracovány všechny zatím nezpracované znaky. Po zpracování znaku je navýšena hodnota čítače out_cons.



Obrázek 15: Konečný automat pro zpracování výstupních znaků

Interpretace jednotlivých řídících sekvencí je realizována pomocí nově vytvořených veřejných funkcí pracujících s výstupním framebufferem. Nově vytvořenými veřejnými funkcemi jsou:

- `fb_set_possitionxy` – Nastavení pozice kurzoru
- `fb_set_cursor_visibility` – Zobrazení/Schování kurzoru
- `fb_clrscr` – Vymazání obrazovky
- `fb_set_bgcolor` – Nastavení barvy pozadí (vlastnosti displeje)
- `fb_set_fgcolor` – Nastavení barvy textu (vlastnosti displeje)

Všechny tyto funkce pracují se strukturou `outdev_t` definující položku data, která obsahuje odkaz na strukturu `fb_instance_t`. Součástí této struktury jsou i informace o aktuální pozici kurzoru a informace o konkrétní buňce framebufferu (například barva pozadí, nebo písma). Podrobný popis implementace framebufferu je mimo rozsah této diplomové práce.

Zpracování stisknutých kláves

Jediná změna, provedená v oblasti zpracování stisknutých kláves souvisí s místem uložení stisknutého znaku. Z tohoto důvodu nebyl vytvořen téměř duplicitní soubor pro zpracování stisknutých kláves, ale stávající funkce `key_pressed`, která slouží k uložení stisknutého znaku, byla rozšířena pomocí podmíněných příkazů preprocesoru `#if` `#else` a `#endif`. V praxi to znamená, že pro všechny ostatní platformy bude přeložena stejně jako doposud.

Při překladu pro platformu `ia32xenh` je pro uložení stisknutého znaku volána funkce `xkbrd_push_character`. V této funkci je stisknutý znak uložen do položky in struktury `xencons_interface`. Po uložení znaku je zvýšena hodnota čítače `in_prod` a nastavena událost na portu konzole. Událost není doručena okamžitě, protože v okamžiku zpracování vláknem `kbrd` není obsluhována výjimka. Událost je proto doméně doručena společně s událostí virtuálního časovače.

6.7 Změny v testovací doméně

Součástí prototypové implementace jsou i drobné úpravy testovací domény [1], které jsou popsány dále v textu.

Problém s voláním privilegovaných instrukcí

Privilegované instrukce, které jsou součástí jádra domény, způsobí #GPF výjimku. V hypervisoru Xen je tato výjimka odchytávána a instrukce simulovány. Mechanismus odchytávání a simulace privilegovaných instrukcí není součástí prototypové implementace, a proto musí být jejich vyvolání eliminováno. V testovací doméně [1] byly odhaleny a eliminovány dva případy volání privilegovaných instrukcí:

1. Ve funkci `before_thread_runs` je prostřednictvím funkcí `fpu_enable` a `fpu_disable` proveden zápis do registru `cr0`.
2. Ve funkci `cpu_arch_init` je při nastaveném příznaku `sse` proveden zápis do registru `cr4`.

Problém s umístěním sdílené paměti ovladače konzole

Stránka obsahující strukturu `xencons_interface` slouží pro komunikaci mezi frontend a backend částí ovladače konzole. Struktura proto musí být alokována na některém z fyzických rámců spravovaných doménou. Pro tento účel byl vybrán jeden z rámců obsahující iniciální oblast <`v_start`; `v_end`> (viz 5.2.4), protože existoval předpoklad, že tyto rámce nebudou dále přidělovány. Tento předpoklad se ukázal jako mylný, a proto byl tento fyzický rámec explicitně označen[36] jako rezervovaný.

---

[36] Explicitní označení rámce je součástí funkce xconsole_init, kterou je inicializována frontendová část ovladače konzole.

7 Ladění

Způsoby hledání chyb v hypervisoru (de facto operačním systému) jsou velice odlišné od způsobů hledání chyb v klasických aplikacích. Zatímco nástroje pro vývoj moderních aplikací jsou čím dál tím více integrované, tak nástroje pro vývoj operačních systémů jsou stále stejné. Debugger, který je součástí integrovaného vývojové prostředí (označovaného zkratkou *IDE*) poskytuje mnohem větší úroveň komfortu při nastavování breakpointů, procházení paměti i ovládání toku instrukcí než řádkový debugger použitý pro ladění operačních systémů.

Řádkový debugger, virtualizační nástroj a další nástroje používané pro ladění operačních systému lze propojit s některými existujícími *IDE* (a tím vytvořit stejnou „in the box“ iluzi, jako při ladění aplikací), ale režie vytvoření tohoto propojení je tak velká, že se ve většině případů toto propojení nevyplatí vytvářet.

Cílem této kapitoly je představit nástroje a způsoby ladění, které byly použity během vývoje prototypové implementace rozhraní hypervisoru Xen.

7.1 Nástroje

Každou novou funkcionalitu implementovanou hypervisoru bylo nutné důkladně otestovat. Restartovat počítač a zavádět nejnovější verzi hypervisoru by bylo jak časově náročné, tak i nepohodlné. Naštěstí však existuje i jiné řešení, které spočívá v použití procesorového emulátoru.

Qemu je open-source emulátor, který je součástí mnoha linuxových distribucí. Důležitou vlastností tohoto emulátoru je to, že umožňuje spouštět bootovací obrazy disku. Přesně takový obraz disku je i výstupem při překladu hypervisoru.

Pro ladění je nutné propojit Qemu s některým z debuggerů. Pro tento účel je vhodný řádkový debugger *GDB*, se kterým Qemu umí spolupracovat.

7.2 Propojení Qemu a GDB

Posloupnost příkazů nutných k propojení *GDB* s emulátorem Qemu je popsána v článku [13]:

```
1. záložka konzole
qemu -cdrom „ceska k boot obrazu hypervisoru“ -m 256 -s -S
(standardně image.iso)
2. záložka konzole
gdb
target remote :1234
```

Přepínačem *S* je pozastaveno vykonání instrukcí do té doby, než je prostřednictvím *GDB* obnoveno příkazem *c*. Díky pozastavení vykonávání instrukcí lze připravit prostředí (symboly, breakpointy) před začátkem simulace.

7.3 Často používané příkazy GDB

Ambicí této práce není podat vyčerpávající výčet všech možných příkazů *GDB*, ale popsat alespoň ty příkazy, které se autorovi zdají jako nejdůležitější. Popis čerpá z oficiálního webového manuálu [14].

7.3.1 Načtení symbolů

Příkaz, kterým je usnadněna orientace v průběhu ladění, je příkaz pro načtení symbolů. Po načtení symbolů lze všude, kde jsou použity virtuální adresy, používat jména načtených symbolů. V případě, že je hypervisor přeložen s volbou řádkového ladění, je při každém pozastavení v *GDB* zobrazen zdrojový kód (spolu s jeho umístění na disku) reprezentovaný právě vykonávanou instrukcí. Načtení symbolů do debuggeru je provedeno příkazem:

```
symbol-file „cesta k ELF obrazu hypervisoru“ (standardně ker-
nel/kernel.raw)
```

Obdobným způsobem lze načíst symboly přeložené domény nebo některé z jejích služeb.

7.3.2 Breakpointy

Breakpointy jsou používány pro nastavení virtuální adres, při jejichž dosažení je pozastaveno vykonávání instrukcí v emulátoru. Breapoint lze nastavit, při nahraných symbolech, jménem funkce:

```
b <jméno funkce>
```

Čímž je vykonávání instrukcí pozastaveno před provedením první instrukce ve funkci. Breakpoint lze nastavit také na libovolnou virtuální adresu:

```
b *<virtuální adresa>
```

V obou případech je úspěšně přidanému breakpointu přiděleno číslo. Přidané breakpointy lze deaktivovat, reaktivovat, či úplně vymazat.

Breakpointům je možné přidat logickou podmínku, která musí být, pro pozastavení vykonávání instrukcí, naplněna. Tuto podmínku lze přidat příkazem:

```
cond <číslo breakpointu> <podmínka>
```

V případě, že jsou nahrány symboly, lze v podmínce použít všechny proměnné viditelné z místa breakpointu. Sekvence příkazů vedoucí k úspěšnému nastavení podmíněného breakpointu pak může odpovídat například:

```
symbol-file kernel/kernel.raw
b write_to_real_console
cond 1 source->out_cons>=3754
```

7.3.3 Prohlížení obsahu paměti

Další důležitou skupinou operací je prohlížení obsahu paměti. Obsah paměti od určité adresy lze vypsat pomocí příkazu:

```
x/x?[i|w|g|…] #počet jednotek <virtuální adresa>
```

Přítomnost jednotlivých přepínačů značí:

- `x` – Obsah paměti bude zobrazen v hexadecimálním tvaru
- `i` – Obsah paměti bude interpretován jako instrukce (vhodné pro zobrazení instrukcí v okolí instrukce způsobující problémy)
- `w` – Obsah paměti bude zobrazen v blocích po 4 B (vhodné pro zobrazení obsahu pole `M2P`, nebo `P2M`)
- `g` – Obsah paměti bude zobrazen v blocích po 8 B (vhodné pro zobrazení obsahu stránkovacích tabulek)

Obsah paměti lze také vypisovat s ohledem na její obsah:

```
print(/x)? <symbol>
```

Přítomnost přepínače `/x` vyvolá zobrazení obsahu v hexadecimálním tvaru. Příkaz je vhodný pro zobrazení obsahu proměnných (zejména u zobrazení obsahu struktur se ukázal být neocenitelným).

Místo symbolu lze použít i přetypovanou virtuální paměť, například příkaz:

```
print/x *((xencons_interface)0xFFFF0000)
```

je ekvivalentní příkazu

```
print/x *console_struct
```

za předpokladu, že symbol `console_struct` je ukazatelem na strukturu xencons_interface nalézající se na adrese `0xFFFF0000`.

7.3.4 Obnovení pozastaveného emulátoru

Při pozastavení vykonávání instrukcí (například při dosažení breakpointu) lze obnovit jejich vykonávání následujícími příkazy:

- `c` – Pokračování až do dalšího pozastavení.
- `n` – Pokračování provádění instrukcí až do dosažení odlišného řádku ve zdrojovém kódu v rámci aktuální funkce (nezanořující, vyžaduje načtené symboly aktuálního kontextu).
- `s` – Pokračování provedení instrukcí až do dosažení odlišného řádku ve zdrojovém kódu (zanořující, vyžaduje načtené symboly aktuálního kontextu).

Všechny tyto příkazy mají jeden číselný volitelný parametr, kterým lze nastavit počet opakování příkazu. Například příkaz „`n 9`“ je ekvivalentní devíti po sobě následujícím příkazům „`n`“.

8 Závěr

Předložená diplomová práce splňuje oba cíle, jež byly vytyčeny v jejím zadání. Prvním cílem byl výběr vhodné testovací domény a podmnožiny rozhraní hypervisoru Xen, která bude prototypovou implementací podporována. Tento cíl je naplněn v kapitole 5. Druhým cílem byla implementace vybrané podmnožiny rozhraní hypervisoru Xen v operačním systému HelenOS a popis provedených změn. Práce obsahuje popis relevantních částí operačního systému HelenOS (kapitola 4) i popis provedených změn (kapitola 6).

Jejím přínosem je popsání dosud neprobádané oblasti paravirtualizace v prostředí virtualizéru Xen. Ostatní dostupné práce jsou zaměřeny na portování operačního systému na platformu Xen. Práce navíc obsahuje i popis vybraných implementačních detailů Xenu, který je hodnotným doplňkem k prakticky neexistující dokumentaci zdrojových kódů Xenu.

8.1 Možnosti budoucího rozšíření

Představená prototypová implementace je prvním mezikrokem na cestě k podpoře plnohodnotné privilegované dom0. Dosažením tohoto stupně podpory rozhraní hypervisoru Xen bude umožněno provozování hypervisoru HelenOS všude tam, kde je nyní provozován Xenovský hypervisor.

Cesta k podpoře některé z plnohodnotných dom0 bude obtížná[37]. Dle autorova názoru bude práce na této podpoře rozdělena do několika dalších mezikroků.

Prvním mezikrokem by mohla být podpora více virtuálních procesorů a více vzájemně komunikujících dom0. To by vyžadovalo rozšíření rozhraní pro kanály událostí a grant tabulky. Dalším vhodným rozšířením by bylo rozlišování jednotlivých typů rámců fyzické paměti přidělených doménám například tak, jak je popsáno v části 3.3.1, spolu s implementací podpory přímo zapisovatelných stránkovacích tabulek.

[37] Po jeho dosažení se stále bude jednat o podporu rozhraní hypervisoru Xen pro architekturu IA-32. Podpora dalších architektur bude vyžadovat další změny v operačním systému HelenOS

Další potřebné mezikroky budou vycházet z vývoje v oblasti dostupnosti privilegovaných dom0. Existují projekty na vytvoření minimalistické verze dom0, které však pro tuto chvíli nejsou dotaženy do úspěšného konce. Podpora některé z budoucích minimalistických verzí dom0 by byla dalším vhodným mezikrokem na cestě k plné podpoře rozhraní hypervisoru Xen.

9 Literatura

[1] Tomáš Benhák, *Port HelenOS pro hypervisor Xen*, diplomová práce, Univerzita Karlova v Praze, 2011

[2] Lukáš Patka, *Virtualizační technologie,* diplomová práce, Masarykova univerzita v Brně, 2009

[3] Martin Děcký, *Mechanismy virtualizace běhu operačních systémů*, Univerzita Karlova v Praze, 2006

[4] David Chisnall, *The Definitive Guide to the Xen Hypervisor*, Prentice Hall, 2007

[5] *Dom0 Kernels for Xen*, dostupné on-line http://wiki.xen.org/wiki/-XenDom0Kernels

[6] Gerald J. Popek, Robert J. Goldberg, *Formal requirements for virtualizable third generation architectures*, magazine, Communications of the ACM, 1974

[7] John Scott Robin and Cynthia E. Irvine, *Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor*, 2000, dostupné on-line http://static.usenix.org/events/sec2000/robin.html

[8] Neiger, G., Santoni, A., Leung, F., Rodgers, D., Uhlig, R, *Intel® Virtualization Technology: hardware support for efficient processor virtualization*, Intel Technology Journal, 10(3):167–177, 2006.

[9] *HelenOS Design Documentation*, dostupné on-line http://www.helenos.org/-doc/design/html.chunked/

[10] *Intel® 64 and IA-32 Architectures Software Developer's Manuals*, dostupné on-line http://download.intel.com/products/processor/manual/325462.pdf

[11] *Executable and Linkable Format (ELF)*, dostupné on-line http://www.skyfree.org/linux/references/ELF_Format.pdf

[12] *ANSI/VT100 Terminal control escape sequences*, dostupné on-line http://www.termsys.demon.co.uk/vtansi.htm

[13] *Debugging HelenOS with QEMU and GDB,* dostupné on-line http://jakubsuniversalblog.blogspot.cz/2008/11/debugging-helenos-with-Qemu-and-GDB.html

[14] *Debugging with GDB*, dostupné on-line http://sourceware.org/GDB/-onlinedocs/GDB/